

Systemy Sztucznej Inteligencji

dokumentacja projektu Auto Encoder

Maciej Stekla, grupa 2F

5 czerwca 2020

Część I

Opis programu

Program ma za zadanie nauczyć się kompresować obrazki o wymiarach 24x24 z czarnym prostokątem na białym tle do szesnastowymiarowego wektora, celem analizy czy współrzędne wektora będą oddziaływać na obraz zdekompresowany w intuicyjny sposób.

Instrukcja obsługi

Uruchomiony program wyświetla proste menu pozwalające wczytać sieć neuronową z pliku, przetestować jej dokładność, albo uczyć ją.

Uczenie polega na wykorzystaniu algorytmu Wstecznej Propagacji Błędów na zbiorze treningowym.

Dodatkowe informacje

Aby stworzyć zbiór treningowy, napisany został także generator. Tworzy on seryjnie pliki graficzne o zdefiniowanym kształcie. Użyty do tego projektu generator rysuje jeden losowy czarny prostokąt na białym tle na każdej próbie.

Część II

Opis działania

Sieć neuronowa Auto-Encodera ma budowę klepsydry. Pierwsza i ostatnia warstwa zawierają taką samą liczbę neuronów, a jedna ze środkowych warstw (nazwijmy ją X) zawiera neuronów znacząco mniej. Podczas treningu podajemy identyczną wartość wejściową i wartość oczekiwaną. Ponieważ warstwa X jest znacząco węższa niż pierwsza i ostatnia, zapis jej wartości zajmuje mniej miejsca.

Auto-Encoder znajduje swoje zastosowanie w kompresji stratnej wyspecjalizowanych na konkretnych zbiorach danych, np. twarzy lub muzyki.

Z jego pomocą można też wyizolować najbardziej znaczące aspekty charakteryzujące dany zbiór, oraz tworzyć zupełnie nowe próbki.

Algorytm

Działanie sieci neuronowej - algorytm Wstecznej Propagacji Błędu

Każda warstwa sieci neuronowej posiada tablicę neuronów. Neuron jest obiektem, który przechowuje wartość. Dodatkowo każda warstwa z wyjątkiem ostatniej posiada jeden neuron, którego wartość zawsze wynosi 1, tzw. bias. Neurony z dwóch sąsiednich warstw są ze sobą połączone synapsami. Z każdej warstwy z wyjątkiem ostatniej, każdy neuron jest połączony z każdym neuronem prócz biasu z warstwy następnej synapsą. Synapsy przechowują wartość zwaną wagą, i tylko te wagi zmieniają się w wyniku uczenia.

Aby dokonać klasyfikacji wektora x , przepisujemy jego współrzędne do neuronów pierwszej warstwy sieci, po czym propagujemy wartości w przód:

$$v_i^k := f\left(\sum_{j=1}^{N_k} v_j^{k-1} \cdot s_{ji}^k\right) : k \in (1, K)$$

Na ostatniej warstwie otrzymujemy wektor v^K , który porównujemy z wektorem oczekiwanym y i propagujemy błąd od ostatniej do pierwszej warstwy.

$$\delta_i^K = f'(v_i^K) \cdot (v_i^K - y_i) \sigma_{ji}^K + = v_i^{k-1} \cdot \delta_j^k \delta_i^k = f'(v_i^k) \cdot \sum_{j=0}^N s_{ij}^k \cdot v_j^{k+1} \sigma_{ji}^k + = v_i^{k-1} \cdot \delta_j^k \quad (1)$$

Otrzymane wartości σ dzielimy liczbę próbek treningowych i przemnożone przez wartość kroku λ aplikujemy do wag synaps:

$$s_{ij}^k + = \sigma_{ij}^k / T \cdot \lambda \quad (2)$$

Działanie Auto Encodera

Celem wytrenowania sieci do kompresowania prostokątów, każdą próbkę treningową P zamieniamy z obrazu na wektor.

Data: obraz P

Result: wektor v

```
; // Przepisz wartości jasności pikseli rzędami
v := wektor ( $P.w \cdot P.h$ )-wymiarowy;
j := 0;
while j < P.h do
    i := 0;
    while i < P.w do
        |  $v_{j \cdot P.w + i} := P_{i,j}.b$ ;
    end
end
end
```

Algorithm 1: Zamiana obrazu na wektor

Dla każdego otrzymanego wektora, wprowadź go do algorytmu Wstecznej Propagacji jako x i y .

Implementacja

Program został zrealizowany w języku C# za pomocą środowiska Visual Studio.


Auto Encoder składa się z następujących plików:

- Network.cs - zawiera model sieci neuronowej. Zaprogramowana klasa jest elastyczna pod względem liczby i rozmiaru warstw, a także funkcji aktywacji. Zajmuje się również zapisywaniem i wczytywaniem sieci do pliku.
- Dataset.cs - zawiera kod związany z obranym typem danych, w tym przypadku obrazków prostokątów. Zajmuje się konwertowaniem plików do postaci wektora i odwrotnie, oraz obliczaniu dokładności wyniku.
Plik zawiera też ogólną klasę sample, której instancja przechowuje wektor wejściowy i wektor oczekiwany.
- Program.cs - zawiera funkcję główną. Zarządza on obiektami opisanymi w pozostałych plikach. Przechowuje także zbiory treningowy i testowy, oraz domyślne ścieżki zapisu.

Funkcje Network:

- SaveToFile(string path) - zapisuje sieć do pliku
- Wczytaj()
- Classify(input) - realizuje algorytm
- BackPropagation(rate, expected) - realizuje algorytm 1
- ApplyLearning() - realizuje algorytm 2

Testy



```
0. Wyjdź
1. Wczytaj sieć
2. Zapisz sieć
3. Testuj sieć
4. Trenuj sieć
4
Podaj wartość kroku: 0.01
Podaj liczbę epok: 21
Co ile epok wyświetlać postęp: 3
Average accuracy: 0,441898148148148 best 0,524305555555556. Loss = 1045,07180165946
Average accuracy: 0,470023148148148 best 0,498263888888889. Loss = 860,692784335184
Average accuracy: 0,483101851851852 best 0,534722222222222. Loss = 704,728241084979
Average accuracy: 0,546064814814815 best 0,581597222222222. Loss = 584,10771135819
Average accuracy: 0,677430555555556 best 0,755208333333333. Loss = 247,972451674964
Average accuracy: 0,821527777777778 best 0,960069444444444. Loss = 87,6919421335412
Average accuracy: 0,840972222222222 best 0,989583333333333. Loss = 80,6467053954961
---- Training ended ----
```

Sieć była trenowana na zbiorze 5000 losowych próbek, z wartością kroku równą 0.01. Można zauważyć, że pierwotna dokładność była na poziomie 50% i rosła w miarę trenowania, a błąd spadał. Niestety jeżeli przyjrzeć się wynikom, okazuje się, że sieć dla każdej próbki testowej zwracała niemal taki sam obraz, a same wartości zbliżały się ogółem do szarości.



Najbardziej obiecujący wynik zawiera całkiem wyraźne kształty prostokątów. Wygląda to jednak, jakby program nałożył na siebie kilka próbek. Ponadto, każdemu elementowi ze zbioru testowego był przypisany taki sam wynik.

Pełen kod aplikacji

```
1 \begin{verbatim}
2
3 //Network.cs
4
5 abstract class Activation
6 {
7     public abstract double Function(double x);
8     public abstract double Derivative(double x);
9 }
10 class Sigmoid : Activation
11 {
12     double a;
13     public Sigmoid(double a)
14     {
15         this.a = a;
16     }
17     public override double Function(double x)
18     {
19         return 2 / (1 + Math.Exp(-a * x)) - 1;
20     }
21     public override double Derivative(double x)
22     {
23         double eax = Math.Exp(-a * x);
24         return 2 * a * eax / ((1 + eax) * (1 + eax));
25     }
26 }
27 class Neuron
28 {
29     public double value;
30     public double delta;
31     public Synapse[] left;
32     public Synapse[] right;
33     public Neuron()
34     {
35         value = 0;
36         this.left = null;
37         this.right = null;
38     }
39     public virtual void Calculate(Activation function)
40     {
41         value = 0;
42         for (int i = left.Length - 1; i >= 0; i--)
43             value += left[i].strength * left[i].left.value;
44         value = function.Function(value);
45     }
46 }
47 class Bias : Neuron
48 {
49     and has no connections from the left
50     public Bias()
51     {
52         value = 1;
```

```

53         this.left = new Synapse[0];
54         this.right = null;
55     }
56     public override void Calculate(Activation function)
57     {
58         value = 1;
59     }
60
61 }
62 class Layer
63 {
64     public Neuron[] neurons;
65     public Layer(int size, int Lsize = -1, int Rsize = -1)
66     {
67         if (Rsize >= 0) {
68             neurons = new Neuron[size + 1];
69             for (int i = size - 1; i >= 0; i--) {
70                 neurons[i] = new Neuron();
71                 if (Lsize >= 0)
72                     neurons[i].left = new Synapse[Lsize + 1];
73                 neurons[i].right = new Synapse[Rsize];
74             }
75             Bias b = new Bias();
76             b.right = new Synapse[Rsize];
77             neurons[size] = b;
78         } else {
79             neurons = new Neuron[size];
80             for (int i = size - 1; i >= 0; i--) {
81                 neurons[i] = new Neuron();
82                 if (Lsize >= 0)
83                     neurons[i].left = new Synapse[Lsize + 1];
84             }
85         }
86     }
87     public void Connect(Layer next)
88     {
89         for (int i = this.neurons.Length - 1; i >= 0; i--)
90             for (int j = this.neurons[i].right.Length - 1; j >= 0; j--)
91             {
92                 Synapse S = new Synapse();
93                 this.neurons[i].right[j] = S;
94                 next.neurons[j].left[i] = S;
95                 S.left = this.neurons[i];
96                 S.right = next.neurons[j];
97             }
98     }
99     class Synapse
100     {
101         public double strength;
102         public double fix;
103         public Neuron left;
104         public Neuron right;
105         public Synapse()
106         {

```

```

107         strength = 0;
108         fix = 0;
109     }
110 }
111 class Network
112 {
113     Layer[] layers;
114     Layer input_layer;
115     Layer output_layer;
116     Activation function;
117     public int InitializationSeed;
118     public Network(int[] layersizes, Activation function)
119     {
120         this.function = function;
121         InitializationSeed = new Random().Next();
122
123         int k;
124         layers = new Layer[layersizes.Length];
125         input_layer = new Layer(layersizes[0], -1, layersizes[1]);
126         layers[0] = input_layer;
127         for (k = 1; k < layersizes.Length - 1; k++) {
128             layers[k] = new Layer(layersizes[k], layersizes[k - 1],
129                                   layersizes[k + 1]);
130             layers[k - 1].Connect(layers[k]);
131         }
132         output_layer = new Layer(layersizes[k], layersizes[k - 1], -1);
133         layers[k] = output_layer;
134         layers[k - 1].Connect(layers[k]);
135         InitiateRandom();
136     }
137     public Network(string path, Activation function)
138     {
139         if (!File.Exists(path))
140             throw (new Exception("File not found"));
141         string[] RawData = File.ReadAllLines(path);
142         uint a = 0;
143         InitializationSeed = Int32.Parse(RawData[a++]);
144         Random RNG = new Random(InitializationSeed);
145         this.function = function;
146
147         string[] rawlines = RawData[a++].Split(' ');
148         int[] layersizes;
149         layersizes = new int[rawlines.Length];
150         for (int j = 0; j < rawlines.Length; j++)
151             layersizes[j] = Int32.Parse(rawlines[j]);
152
153         int k;
154         layers = new Layer[layersizes.Length];
155         input_layer = new Layer(layersizes[0], -1, layersizes[1]);
156         layers[0] = input_layer;
157         for (k = 1; k < layersizes.Length - 1; k++) {
158             layers[k] = new Layer(layersizes[k], layersizes[k - 1],
159                                   layersizes[k + 1]);
160             layers[k - 1].Connect(layers[k]);
161         }
162         output_layer = new Layer(layersizes[k], layersizes[k - 1], -1);
163         layers[k] = output_layer;
164         layers[k - 1].Connect(layers[k]);
165         InitiateRandom();
166     }
167 }

```



```

160         for (int j = layers[k].neurons[i].left.Length - 1; j >= 0; j
            --)
161             if (a < RawData.Length)
162                 layers[k].neurons[i].left[j].strength = Double.Parse(
                    RawData[a++]);
163             else
164                 layers[k].neurons[i].left[j].strength = RNG.NextDouble()
                    *2-1;
165     }
166     output_layer = new Layer(layersizes[k], layersizes[k - 1], -1);
167     layers[k] = output_layer;
168     layers[k - 1].Connect(layers[k]);
169     for (int i = layers[k].neurons.Length - 1; i >= 0; i--)
170         for (int j = layers[k].neurons[i].left.Length - 1; j >= 0; j
            --)
171             if(a<RawData.Length)
172                 layers[k].neurons[i].left[j].strength = Double.Parse(
                    RawData[a++]);
173             else
174                 layers[k].neurons[i].left[j].strength = RNG.NextDouble()
                    *2-1;
175 }
176 public void SaveToFile(string path)
177 {
178     List<string> Data = new List<string>();
179     Data.Add(InitializationSeed.ToString());
180     string layersizes = (output_layer.neurons.Length).ToString();
181     for (int i = layers.Length - 2; i >= 0; i--)
182         layersizes = (layers[i].neurons.Length - 1).ToString() + " " +
            layersizes;
183     Data.Add(layersizes);
184     for (int k = 1; k < layers.Length; k++)
185         for (int i = layers[k].neurons.Length - 1; i >= 0; i--)
186             for (int j = layers[k].neurons[i].left.Length - 1; j >= 0; j
                --)
187                 Data.Add(layers[k].neurons[i].left[j].strength.ToString())
                    ;
188     File.WriteAllLines(path, Data);
189 }
190 public void InitiateRandom()
191 {
192     Random RNG = new Random(InitializationSeed);
193     foreach (Layer k in layers.Skip(1))
194         foreach (Neuron n in k.neurons)
195             foreach (Synapse s in n.left)
196                 s.strength = RNG.NextDouble() * 2 - 1;
197 }
198 public override string ToString()
199 {
200     StringBuilder sb = new StringBuilder();
201     for (int k = 0; k < layers.Length; k++)
202     {
203         for (int i = 0; i < layers[k].neurons.Length; i++)
204             { sb.Append(layers[k].neurons[i].value); sb.Append(" "); }
205         sb.Append("\n");

```

```

206     }
207
208     return sb.ToString();
209 }
210 public double[] Classify(double[] input)
211 {
212     int i;
213     for (i = input_layer.neurons.Length - 2; i >= 0; i--)
214         input_layer.neurons[i].value = input[i];
215
216     foreach (Layer L in layers.Skip(1))
217         foreach (Neuron N in L.neurons)
218             N.Calculate(function);
219
220     double[] output = new double[output_layer.neurons.Length];
221     for (i = output_layer.neurons.Length - 1; i >= 0; i--)
222         output[i] = output_layer.neurons[i].value;
223
224     return output;
225 }
226 public void BackPropagation(double rate, double[] expected)
227 {
228     for (int i = output_layer.neurons.Length - 1; i >= 0; i--) {
229         output_layer.neurons[i].delta =
230             (expected[i] - output_layer.neurons[i].value) *
231             function.Derivative(output_layer.neurons[i].value);
232         foreach (Synapse S in output_layer.neurons[i].left)
233             S.fix += S.left.value * S.right.delta * rate;
234     }
235     for (int k = layers.Length - 2; k > 0; k--) {
236         foreach (Neuron N in layers[k].neurons) {
237             N.delta = 0;
238             foreach (Synapse S in N.right)
239                 N.delta += S.strength * S.right.delta;
240             N.delta *= function.Derivative(N.value);
241         }
242         foreach (Neuron N in layers[k].neurons)
243             foreach (Synapse S in N.left)
244                 S.fix += S.left.value * S.right.delta * rate;
245     }
246 }
247 public void ApplyLearning(double step)
248 {
249     foreach (Layer L in layers.Skip(1))
250         foreach (Neuron N in L.neurons)
251             foreach (Synapse S in N.left) {
252                 S.strength = S.strength * 0.95 + S.fix * step;
253                 S.fix = 0;
254             }
255 }
256 }
257
258 //DataSet.cs
259
260 struct ImageNeuron

```

```

261 {
262
263     public static double[] LoadImage(string path)
264     {
265         Bitmap IMG = new Bitmap(path);
266         int W = IMG.Width, H = IMG.Height;
267         int x, y, i = W * H;
268         double[] pixels = new double[i];
269         y = H; while (y > 0) {
270             --y;
271             x = W; while (x > 0) {
272                 --x;
273                 Color X = IMG.GetPixel(x, y);
274                 pixels[--i] = (double)X.GetBrightness();
275             }
276         }
277         return pixels;
278     }
279     public static Bitmap SideToSide(double[] expected, double[] actual,
280                                     int w, int h)
281     {
282         int x, y, b, d = 4;
283         int wd = w + d;
284         Bitmap BMP = new Bitmap(w + d + w, h);
285         y = h; while (y > 0) {
286             --y;
287             x = w; while (x > 0) {
288                 --x;
289                 b = (int)(expected[y * w + x] * 255);
290                 if (b < 0) b = 0; if (b > 255) b = 255;
291                 BMP.SetPixel(x, y, Color.FromArgb(b, b, b));
292                 b = (int)(actual[y * w + x] * 255);
293                 if (b < 0) b = 0; if (b > 255) b = 255;
294                 BMP.SetPixel(x + wd, y, Color.FromArgb(b, b, b));
295             }
296             for (x = w; x < wd; ++x)
297                 BMP.SetPixel(x, y, Color.FromArgb(0, 64, 192));
298         }
299         return BMP;
300     }
301     public static double Accuracy(double[] expected, double[] actual)
302     {
303         double passed = 0;
304         for(int i = expected.Length; i-- > 0;) {
305             double pixelaccuracy = (expected[i] - 0.5) * (actual[i] - 0.5)
306             ;
307             if (pixelaccuracy >= 0) passed += 1;
308         }
309         return passed / expected.Length;
310     }
311 }
312 struct sample
313 {
314     public double[] input;

```

```

314     public double[] output;
315     public sample(int insize, int outsize)
316     {
317         input = new double[insize];
318         output = new double[outsize];
319     }
320     public sample(double[] input, double[] output)
321     {
322         this.input = input;
323         this.output = output;
324     }
325     public sample(double[] line, int insize, int outsize)
326     {
327         if (line.Length != insize + outsize)
328             throw (new Exception("Wrong sizes"));
329         this.input = new double[insize];
330         this.output = new double[outsize];
331         for (int j = 0; j < insize; j++) input[j] = line[j];
332         for (int j = 0; j < outsize; j++) output[j] = line[insize + j];
333     }
334 }
335
336 //Program.cs
337
338 class Program
339 {
340     const string TRAININGDATA = @"..\..\Samples";
341     const string DISPLAY = @"..\..\Display";
342     const string SAMPLES = @"..\..\AutoEncoder.txt";
343     static Network AutoEncoder;
344     static List<sample> TrainingSet;
345     static List<sample> TestSet;
346     static double Squared(double x) { return x * x; }
347     static void TestujSie[U+FFFD]()
348     {
349         double bestAcc=0, sumAcc=0; double SumLoss = 0;
350         int it = 0;
351         foreach (sample current in TestSet) {
352             ++it;
353             double[] NetGuess = AutoEncoder.Classify(current.input);
354             ImageNeuron.SideToSide(current.output, NetGuess, 24, 24).Save(
355                 String.Format(@"{0}\test{1}.png",DISPLAY,it));
356             int choice = 0; int correct = 0; double choiceVal = NetGuess
357                 [0];
358             double loss = Squared(NetGuess[0] - current.output[0]);
359             for (int i = current.output.Length - 1; i > 0; i--) {
360                 loss += Squared(NetGuess[i] - current.output[i]);
361                 if (NetGuess[i] > choiceVal) { choiceVal = NetGuess[i];
362                     choice = i; }
363                 if (current.output[i] == 1) correct = i;
364             }
365             double accuracy = ImageNeuron.Accuracy(current.output,
366                 NetGuess);
367             SumLoss += loss;
368             if (accuracy > bestAcc) bestAcc = accuracy;

```

```

366         sumAcc += accuracy;
367     }
368     Console.WriteLine("Average accuracy: {0} best {1}. Loss = {2}",
369         sumAcc/TestSet.Count, bestAcc, SumLoss / TestSet.Count);
370 }
371 static void TrenujSie[U+FFFD](double learning_coeff, int epochlength, int
    peekevery=0)
372 {
373     for (int epoch = 0; epoch < epochlength; epoch++) {
374         foreach (sample current in TrainingSet) //Train the network {
375             double[] NetGuess = AutoEncoder.Classify(current.input);
376             AutoEncoder.BackPropagation(learning_coeff, current.output);
377         }
378         AutoEncoder.ApplyLearning(1.0 / TrainingSet.Count);
379         if (peekevery > 0 && epoch % peekevery == 0) {
380             TestujSie[U+FFFD]();
381         }
382     }
383     Console.WriteLine("---- Training ended -----");
384     Console.ReadLine();
385 }
386 static void Wczytaj(string path = null)
387 {
388     if (path == null) {
389         Console.Write("Podaj sciezke: ");
390         path = Console.ReadLine();
391         if (path == "")
392             path = SAMPLES;
393     }
394     if (File.Exists(path))
395         AutoEncoder = new Network(path, new Sigmoid(1));
396     else
397         AutoEncoder = new Network(new int[] { 576, 576, 16, 576, 576
398                                     },
                                    new Sigmoid
                                    (1));
399     TrainingSet = new List<sample>();
400     TestSet = new List<sample>();
401
402     for (int i = 1; File.Exists(String.Format(@"{0}\sample{1}.png",
403         TRAININGDATA, i)); ++i) {
404         var vector = ImageNeuron.LoadImage(String.Format(@"{0}\sample
405             {1}.png",
406                 TRAININGDATA, i));
407         sample nowa = new sample(vector, vector);
408         TrainingSet.Add(nowa);
409     }
410     for (int i = 1; File.Exists(String.Format(@"{0}\test{1}.png",
411         TRAININGDATA, i)); ++i) {
412         var vector = ImageNeuron.LoadImage(String.Format(@"{0}\test
413             {1}.png",
414                 TRAININGDATA, i));
415         sample nowa = new sample(vector, vector);
416         TestSet.Add(nowa);
417     }

```

```

416 }
417
418 static void Main()
419 {
420     Wczytaj(SAMPLES);
421
422     while (true)
423     {
424         string path;
425         Console.Clear();
426         Console.WriteLine("
427             0. Wyjdź\n
428             1. Wczytaj siec\n
429             2. Zapisz siec\n
430             3. Testuj siec\n
431             4. Trenuj siec
432         ");
433         switch (Console.ReadLine())
434         {
435             case "0":
436                 return;
437
438             case "1":
439                 Wczytaj();
440                 break;
441
442             case "2":
443                 Console.Write("Podaj sciezke: ");
444                 path = Console.ReadLine();
445                 if (path == "")
446                     AutoEncoder.SaveToFile(SAMPLES);
447                 else
448                     AutoEncoder.SaveToFile(path);
449                 break;
450
451             case "3":
452                 TestujSie[U+FFFD]();
453                 Console.ReadLine();
454                 break;
455
456             case "4":
457                 int epochlength, peekevery; double learning_coeff;
458                 Console.Write("Podaj wartosc kroku: ");
459                 try { learning_coeff = Double.Parse(
460                     Console.ReadLine().Replace('.', ',')); }
461                 catch { break; }
462                 Console.Write("Podaj liczbe epok: ");
463                 try { epochlength = Int32.Parse(Console.ReadLine()); }
464                 catch { break; }
465                 Console.Write("Co ile epok wyswietlac postep: ");
466                 try { peekevery = Int32.Parse(Console.ReadLine()); }
467                 catch { peekevery=0; }
468
469                 if(peekevery>0)
470                     TrenujSie[U+FFFD](learning_coeff, epochlength, peekevery);

```

```

471         else
472             TmujSie[U+FFFD](learning_coeff, epochlength);
473
474         break;
475     }
476 }
477
478 }
479 }
480
481
482 //Generator
483
484 abstract class Image
485 {
486     protected int W;
487     protected int H;
488     public abstract Bitmap Draw();
489 }
490 class Rectangles : Image
491 {
492     private int N;
493     private static Random RNG;
494     public Rectangles(int width, int height, int count)
495     {
496         W = width; H = height;
497         N = count;
498         RNG = new Random();
499     }
500     public override Bitmap Draw()
501     {
502         Bitmap image = new Bitmap(W, H);
503         Color white = Color.FromArgb(255, 255, 255);
504         Color black = Color.FromArgb(0, 0, 0);
505         int x, y;
506
507         y = H; while (y > 0) {--y;
508             x = W; while (x > 0) {--x;
509                 image.SetPixel(x, y, white);
510             }
511         }
512         for (int i = 0; i < N; i++) {
513             int A1x = RNG.Next() % W; int A1y = RNG.Next() % H;
514             int A2x = A1x + RNG.Next() % (W-A1x)+1; int A2y = A1y + RNG.
515                 Next() % (H-A1y)+1;
516
517             for (y = A1y; y < A2y; ++y)
518                 for (x = A1x; x < A2x; ++x)
519                     image.SetPixel(x, y, black);
520         }
521         return image;
522     }
523 }
524 class Generator

```

```

525 {
526     private int W;
527     private int H;
528     Image shape;
529     public Generator(int width, int height, Image kind)
530     {
531         W = width; H = height;
532         this.shape = kind;
533     }
534     public void Generate(string path, int count)
535     {
536         if (Directory.Exists(path))
537             for (int i = count; i > 0; i--) {
538                 shape.Draw().Save(String.Format(@"{0}\sample{1}.png", path,
539                     i));
540             }
541     }
542
543     class Program
544     {
545         static void Main(string[] args)
546         {
547             int w, h, n;
548             Console.Write("Podaj dlugosc: ");
549             w = Int32.Parse(Console.ReadLine());
550             Console.Write("Podaj szerokosc: ");
551             h = Int32.Parse(Console.ReadLine());
552             Console.Write("Ile probek wygenerowac: ");
553             n = Int32.Parse(Console.ReadLine());
554             Generator G = new Generator(w, h, new Rectangles(w, h, 1));
555             G.Generate(@"..\..\..\AutoEncoder\Samples",n);
556         }
557     }
558
559
560 \end{verbatim}

```
