Dokumentacja projektu
Przetwarzanie sygnałów

Jessica Chomin, Karolina Wieczorek, Piotr Olender, grupa 1

22 stycznia 2018

1 Część I

1.1 Opis programu

Program służy do rozpoznawania odręcznie pisanych liter. Po włączeniu go użytkownik wybiera, czy chce, aby program wczytał obrazki i stworzył na ich podstawie wzór, czy wczytać już gotowy znajdujący się w pliku tekstowym. Następnie może wybrać opcje klasyfikowania literki.

1.2 Instrukcja obsługi

Jeśli użytkownik wybierze '1', to program wykonuje wzór liter z alfabetu domyślnego.
Jeśli użytkownik wybierze '2', to program wczyta gotowy wzór.
Jeśli użytkownik wybierze '3', to musi wpisać ścieżkę obrazka, po czym program sklasyfikuje.

2 Część II

2.1 Opis działania

Program binaryzuje, przycina, skaluje do rozmiaru 256x256, binaryzuje ponownie, tworzy liste czarnych pikseli po czym interpoluje liste. W zależności od zamiaru owa lista interpolacyjna może trafić do wzorca danej litery lub zostać porównana z inna listą. Program klasyfikuje litery metodą k-najbiższych sąsiadów a i nterpoluje Interpolacją Czebyszewa.

2.2 Algorytm

W tej sekcji pokazujemy najważniesze algrotym dla działania programu

Dodaj jeden do ilości liter użytych do tego wzorca
n = ilość liter użytych do wzorca
for nie doszedłem do konca listy **do**
  Punkt(ze wzorca) = Punkt $\cdot(n/n+1+$ punkt z listy $\cdot(1/n+1))$
end for

Algorithm 1: Tworzenie wzorca

for cała szerokość obrazka **do**
  for cała wysokość obrazka **do**
    if piksel jest czarny **then**
      zsumuj go z innymi czarnymi pikselami w kolumnie
    end if
  end for
  zapisz punkt do listy
end for

Algorithm 2: Liczenie wysokości/szerokości obrazka

```
for nie doszedłem do końca pikseli po wysokości obrazka do
  for nie doszedłem do końca pikseli po szerokości obrazka do
    if znajdziesz pierwszy czarny piksel then
      zapisz jego pozycję jako y1
    end if
  end for
end for
for nie doszedłem do pierwszego piksela po wysokości obrazka idąc od końca do
  for nie doszedłem do końca pikseli po szerokości obrazka do
    if znajdziesz pierwszy czarny piksel then
      zapisz jego pozycję jako y
    end if
  end for
end for
wysokość = y1 - y + 1
for nie doszedłem do końca pikseli po szerokości obrazka do
  for nie doszedłem do końca pikseli po wysokości obrazka do
    if znajdziesz pierwszy czarny piksel then
      zapisz jego pozycję jako x
    end if
  end for
end for
for nie doszedłem do pierwszego piksela po szerokości obrazka idąc od końca do
  for nie doszedłem do pierwszego piksela po wysokości obrazka idąc od końca do
    if znajdziesz pierwszy czarny piksel then
      zapisz jego pozycję jako x1
    end if
  end for
end for
szerokość = x1 - x + 1
Stwórz prostokąt(x,y,szerokość, wysokość)
Zwróć przycięty obrazek
```

Algorithm 3: Przycinanie

Utwórz listę sąsiadów i wyzeruj ją
Utwórz listę odległości i ustaw każdą wartość na 1000
for każdego elementu w klasyfikowanej liście **do**
  for każdego elementu we wzorcu **do**
    znajdź odległość elementu w liście od elementu we wzorcu i zapisz go w dystansach
  end for
  znajdź minimalną odległość
  for każdego elementu w dystansach **do**
    if odległość[j] jest równa min **then**
      sasiad[j]++
    end if
  end for
end for
if istnieją co najmniej dwie maksymalne wartości w liście sąsiadów **then**
  Zapisz listę z maksymalnymi sąsiadami
  Wyzeruj sąsiadów
  Ustaw wszystkie wartości w dystansach na –1
  for dla każdego maksymalnego sąsiada **do**
    Wyznacz odległość od wzorca
  end for
end if
zwróć maksymalną wartość w sąsiadach

Algorithm 4: Klasyfikacja

Stwórz obiekt klasy ImageSupport z danej scieżki.
Zbinaryzuj obiekt.
Przytnij obiekt, aby pozbyć się białych pól z krawędzi.
Przeskaluj obiekt do rozmiatu 256 na 256. Uśrednij piksele i zrób liste punktów.
Zinterpoluje liste interpolacją Czybyszewa.
Dodaj liste interpolacyjną obiektu do wzoru danej litery.

Algorithm 5: Dodawanie obrazka do wzorca

2.3 Kod programu

Klasa Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;

namespace Projekt_studia
{
    class Program
    {
        public static void addThisImageToPattern(classifier cls,
string path, int letter)
        {
            imageSupport pattern = new imageSupport(@path);
            pattern.binarization();
            pattern.btm = pattern.cut();
            pattern.ResizeImage();
            pattern.binarization();
            pattern.makePointsList();
            pattern.makeInterpolationList();
            cls.makePattern(pattern.interpolationList, letter);
        }

        public static string tellMeWhatIsThisLetter
(classifier cls, imageSupport letter)
        {
            string s = cls.classify(letter.interpolationList);
            return s;
        }
        static void Main(string[] args)
        {
            classifier cls = new classifier();
            Console.WriteLine("Program do sprawdzania literek");
            Console.WriteLine("1- Zrób wzór z alfabetu
 domyslnego");
            Console.WriteLine("2- Wczytaj gotowy wzor");
            Console.WriteLine("3- Klasyfikuj litere");
            ConsoleKeyInfo cki = Console.ReadKey();

            if (cki.KeyChar==’1’)
            {
                //tworzenie alfabetu
```

```
                int sterujaca = -1;
                for (int i = 0; i < 3*26; i++)
                {
                    Console.Write(i + " ");
                    if (i % 3 == 0)
                    {
                        Console.Write("Laduje" + i + "   ");
                        sterujaca++;
                    }
                    string temp = Convert.ToString(i);
                    string str = "C:\\Users\\Piotr\\Documents
\\projekty vs2017\\alfabet\\" + temp + ".jpg";
 // ścieżka do plików ze wzormai alfabetu
                    addThisImageToPattern(cls, str,sterujaca);
                }
                //koniec tworzenia alfabetu
                Console.Clear();
                Console.WriteLine("Załadowano obrazki z folderu
 alfabet");
                cls.savePatterns();
                Console.WriteLine("zapisano");
            }
            if(cki.KeyChar == '2')
            {
                cls.loadPatterns(); // ścieżka do wzorca jest
w klasyfikatorze
                Console.Clear();
                Console.WriteLine("wczytano ze wzoru.txt");
            }
            Console.WriteLine("rozpoczynam testy");
            if (cki.KeyChar == '3')
            {
                for (int i = 0; i < 3*26; i+=3)
                {
                    Console.WriteLine("Wpisz sciezke do
 obrazka");
                    string str = Console.ReadLine();
                    imageSupport test = new imageSupport(@str);
                      test.binarization();
                      test.btm = test.cut();
                      test.ResizeImage();
                      test.binarization();
                      test.makePointsList();
                      test.makeInterpolationList();
                      Console.WriteLine("Zakwalifikowałem
```

```
litere " + i + " jako: " +tellMeWhatIsThisLetter(cls,test) );
                }
            }
            Console.ReadKey();
        }
    }
}
```

Klasa imageSuupport

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;


namespace Projekt_studia
{

    class imageSupport
    {
        public Bitmap btm;
        public List<Int64> pointsList= new List<Int64>();
        public List<Int64> interpolationList = new List<Int64>();
        public imageSupport(string path)
        {
            this.btm = new Bitmap(@path);// path jest ścieżką
 do obrazka

        }
        public void binarization()
        {

            for (int i = 0; i < btm.Width; i++)
            {
                for (int j = 0; j < btm.Height; j++)
                {
                    int r = btm.GetPixel(i, j).R;
                    int b = btm.GetPixel(i, j).B;
                    int g = btm.GetPixel(i, j).G;

                    if (r+g+b<350)
                    {
                        Color kolor = Color.FromArgb(0, 0, 0);
```

```
                    btm.SetPixel(i, j, kolor);
                }
                else
                {
                    Color kolor =
Color.FromArgb(255, 255, 255);
                    btm.SetPixel(i, j, kolor);
                }
            }
        }

    }
    public void makePointsList()
    {


        for (int i = 0; i < btm.Width; i++)
        {
            int counter = 0;
            int sum = 0;
            for (int j = 0; j < btm.Height; j++)
            {
                int r = btm.GetPixel(i, j).R;
                if (r==255)
                {
                    counter++;
                    sum += j;
                }
            }
            if (counter == 0)
                counter++;

            this.pointsList.Add(Convert.ToInt64(sum/counter));
        }
    }
    public void makeInterpolationList()
    {

        interpolationList.Add(1);
        interpolationList.Add(this.pointsList[1]);
        for(int i=2;i<pointsList.Count;i++)
        {
            double point = (2 * this.pointsList[i] *
 interpolationList[i - 1] - interpolationList[i - 2])/4;
            Int64 temp = (Int64)point;
```

```
                interpolationList.Add(temp);
            }
        }
        public void save(string path)
        {
            btm.Save(@path);
        }
        public void ResizeImage()
        {
            Bitmap resized = new Bitmap(this.btm,
new Size(256, 256));
            this.btm = resized;
        }
        public Bitmap cut()
        {
            Bitmap btm = this.btm;
            int x = 0;
            int x1 = 0;
            int y = 0;
            int y1 = 0;
            int width = 0;
            int height = 0;

            for (int i = 0; i < btm.Height; i++)
            {
                for (int j = 0; j < btm.Width; j++)
                {
                    if (btm.GetPixel(j, i).R < 200)
                    {
                        y = i;
                        j = btm.Width;
                        i = btm.Height;
                    }

                }
            }
            for (int i = btm.Height-1; i > 0; i--)
            {
                for (int j = 0 ; j <btm.Width; j++)
                {
                    if (btm.GetPixel(j, i).R < 200)
                    {
                        y1 = i;
                        j = btm.Width;
                        i = 0;
```

```
                }

            }
        }
        height = y1 - y + 1;

        for (int i = 0; i < btm.Width; i++)
        {
            for (int j = 0; j < btm.Height; j++)
            {
                if (btm.GetPixel(i, j).R <200)
                {
                    x = i;
                    j = btm.Height - 1;
                    i = btm.Width - 1;
                }
            }
        }
        for (int i = btm.Width - 1; i > 0; i--)
        {
            for (int j = btm.Height - 1; j > 0; j--)
            {
                if (btm.GetPixel(i, j).R <200)
                {
                    x1 = i;
                    i = 0;
                    j = 0;
                }
            }
        }
            width = x1 - x + 1;
        Rectangle part = new Rectangle(x, y, width, height);
        Bitmap bmpImage = new Bitmap(this.btm);
        return bmpImage.Clone(part, bmpImage.PixelFormat);


    }
  }
}
```

Klasa classifier

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.IO;

namespace Projekt_studia
{
    class classifier
    {
        public List<List<Int64>> Patterns =
new List<List<Int64>>();
 //0-a,1-o,2-u na koncu listy mam ilosc elementow
 uzytych do wzoru

        public classifier()
        {
            for(int i =0;i<26;i++)
            {
                List<Int64> p = new List<Int64>();
                p.Add(i);
                for (int j = 1; j < 257; j++)
                    p.Add(0);
                Patterns.Add(p);
            }
        }
        public void savePatterns()
        {
            using (StreamWriter sw = new StreamWriter(@"C:\\
Users\\Piotr\\Documents\\projekty vs2017
\\Projektstudia\\Projekt studia\wz.txt"))
 // tu jest sciezka
            {
                foreach(var item in Patterns)
                {
                    foreach(var item2 in item)
                    {
                        sw.Write(item2+",");
                    }
                    sw.WriteLine();
                }

            }


        }
        public void loadPatterns()
        {
```

```
            string[] tab = File.ReadAllLines(@"C:\\
Users\\Piotr\\Documents\\projekty vs2017
\\Projektstudia\\Projekt studia\wz.txt");
            foreach(var item in tab)
            {
                string[]tab2=item.Split(',');
                List<Int64> p = new List<Int64>();
                for(int i =1; i<tab2.Length-2;i++)
                {
                    p.Add(Convert.ToInt64(tab2[i]));
                }
                Patterns.Add(p);
            }


        }
        public void makePattern(List<Int64> interpolationLists, int letter)
        {
            Patterns[letter][256]++;
            Int64 n = Patterns[letter].Count - 1;
            for (int i = 1; i < interpolationLists.Count-1; i++)
            {
                Patterns[letter][i-1] = Patterns[letter][i] *
(n / n + 1) + interpolationLists[i-1] * (1 / n + 1);
            }

        }
        public string getLetter(Int64 index)
        {
            byte[] bytes = BitConverter.GetBytes(97 + index);
            string letter = Encoding.Unicode.GetString(bytes);

            return letter;
        }
        public string classify(List<Int64> letter)
        {
            Console.WriteLine("Klasyfikuje");
            List<Int64> neighbours = new List<Int64>();
            List<double> distance = new List<double>();
            for (int i = 0; i < 26; i++)
            {
                neighbours.Add(0);
                distance.Add(1000);
            }
            for (int i = 0; i < letter.Count; i++)
```

```
            {

                for (int j = 0; j < distance.Count; j++)
                {
                    distance[j] = Math.Abs(letter[i]
- Patterns[j][i]);
                }
                double min = distance.Min();
                for (int j = 0; j < distance.Count; j++)
                {
                    if (distance[j] == min)
                        neighbours[j]++;
                }
            }
            // elminacja podobienstwa do dwoch i wiecej liter
            List<int> bestNeighboursId = new List<int>();
            Int64 max = neighbours.Max();
            int count = -1;
            for(int i=0;i<neighbours.Count;i++)
            {
                if (neighbours[i] == max)
                {
                    bestNeighboursId.Add(i);

                    count++;
                }
            }
            if(count>0)
            {
                for(int i=0;i<neighbours.Count;i++)
                {
                    neighbours[i] = 0;
                    distance[i] = -1;
                }
                for (int i = 0; i < letter.Count; i++)
                {
                    foreach (int var in bestNeighboursId)
                    {

                        for (int j = 0; j < distance.Count; j++)
                        {
                            distance[var] = Math.Abs(letter[i] - Patterns[j][i]);
                        }
                        double min =distance[var];
                        for(int j=0;j<distance.Count;j++)
```

```
                        {
                            if (distance[j] < min && distance[j] >= 0)
                                distance[j] = min;
                        }
                        for (int j = 0; j < distance.Count; j++)
                        {
                            if (distance[j] == min)
                                neighbours[j]++;
                        }
                    }
                }
                max = neighbours.Max();
            }
            //policz ile podobienstw
            Int64 maxId=0;
            for (int i=0;i<neighbours.Count;i++)
            {
                Console.Write(neighbours[i]+" ");
                if (neighbours[i] == max)
                {
                    maxId = i;
                }
            }

                        return this.getLetter(maxId);



        }

    }
}
```