UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
IM. J. I J. ŚNIADECKICH W BYDGOSZCZY

WYDZIAŁ TELEKOMUNIKACJI, INFORMATYKI I ELEKTROTECHNIKI
ZAKŁAD TECHNIKI CYFROWEJ

SZTUCZNA INTELIGENCJA

DOKUMENTACJA PROJEKTU SZTUCZNEJ SIECI NEURONOWEJ ART1

AUTOR:
KRZYSZTOF KLEBA
SŁAWOMIR STROKOWSKI

KIERUNEK:
INFORMATYKA STOSOWANA

GRUPA NR 4
SEMESTR IV
ROK AKADEMICKI 2016/2017

SPIS TREŚCI

WSTĘP

Przedstawiony program jest realizacja projektu sztucznej sieci neuronowej ART1. Wybrana sieć polega na grupowaniu i klasyfikacji obrazów binarnych i jest zdolna do adaptacji. Sieć ART1 uczy się grupować obrazy wejściowe bez nauczyciela, co oznacza, że potrafi wykrywać grupy bez informacji o ich liczbie oraz utrzymywać stabilną reprezentację grup przez przepisane im neurony warstwy wyjściowej. Powyższy sposób działania jest zaimplementowany w opisanym niżej programie, a część teoretyczna wykorzystanych algorytmów oparta jest na opracowaniu sztucznych sieci neuronowych autorów J. Żurady, M. Barski oraz W. Jędruch.

CHARAKTERYSTYKA PROJEKTU

Aplikacja jest oparta na frameworku Swinga z wykorzystaniem Window Buildera, w języku programowania Jawa 8. Do tworzenia sieci i przechowywania neuronów zostały użyte elementy klasy kolekcji. Do programu zostały załadowane zestawy testowych obrazów wygenerowane i zniekształcone w programie do tworzenia grafiki rastrowej w rozdzielczości 16x16 pikseli.

OPIS TWORZENIA SIECI

Opis ogólny

Działanie sieci rozpoczyna się od wczytania pierwszego elementu - obrazu binarnego. Każdy następny obraz zaliczany jest do istniejącej grupy, jeżeli wykazuje podobieństwo do jej wzorców. W przeciwnym przypadku zostaje tworzona nowa klasa reprezentująca nowy neuron. Stworzony neuron pełni rolę reprezentanta nowej grupy.

Algorytm działania ART1

Algorytm składa się z pięciu głównych kroków:

1. Pierwszy krok jest odpowiedzialny za stworzenie pierwszego neuronu i nadania mu wartości początkowych dla macierzy **W** i **V** wynoszące odpowiednio :

$$W_{ij} = \frac{1}{1+n}, \qquad V_{ij} = 1, \qquad \forall i,j.$$

2. W następnym kroku należy podać obraz binarny w postaci wektora elementów obrazu binarnego, który przyjmuje wartości 0 lub 1. Następnie należy obliczyć wszystkie miary dopasowania pikseli obrazu do wektora **W:**

$$y_j = \sum_{i=1}^{n} w_{ij} x_{ij}$$

3. Następnie trzeba wybrać najlepiej dopasowaną kategorię, która obliczana jest metodą wyszukującą największy współczynnik dopasowania:

$$y_m = \max\ (y_j), \qquad j = 1,2,\dots,M$$

4. Dla zwycięskiego dopasowania neuronu przeprowadzamy test podobieństwa, sprawdzający czy dane prawdopodobieństwo jest większe od ustawionego progu:

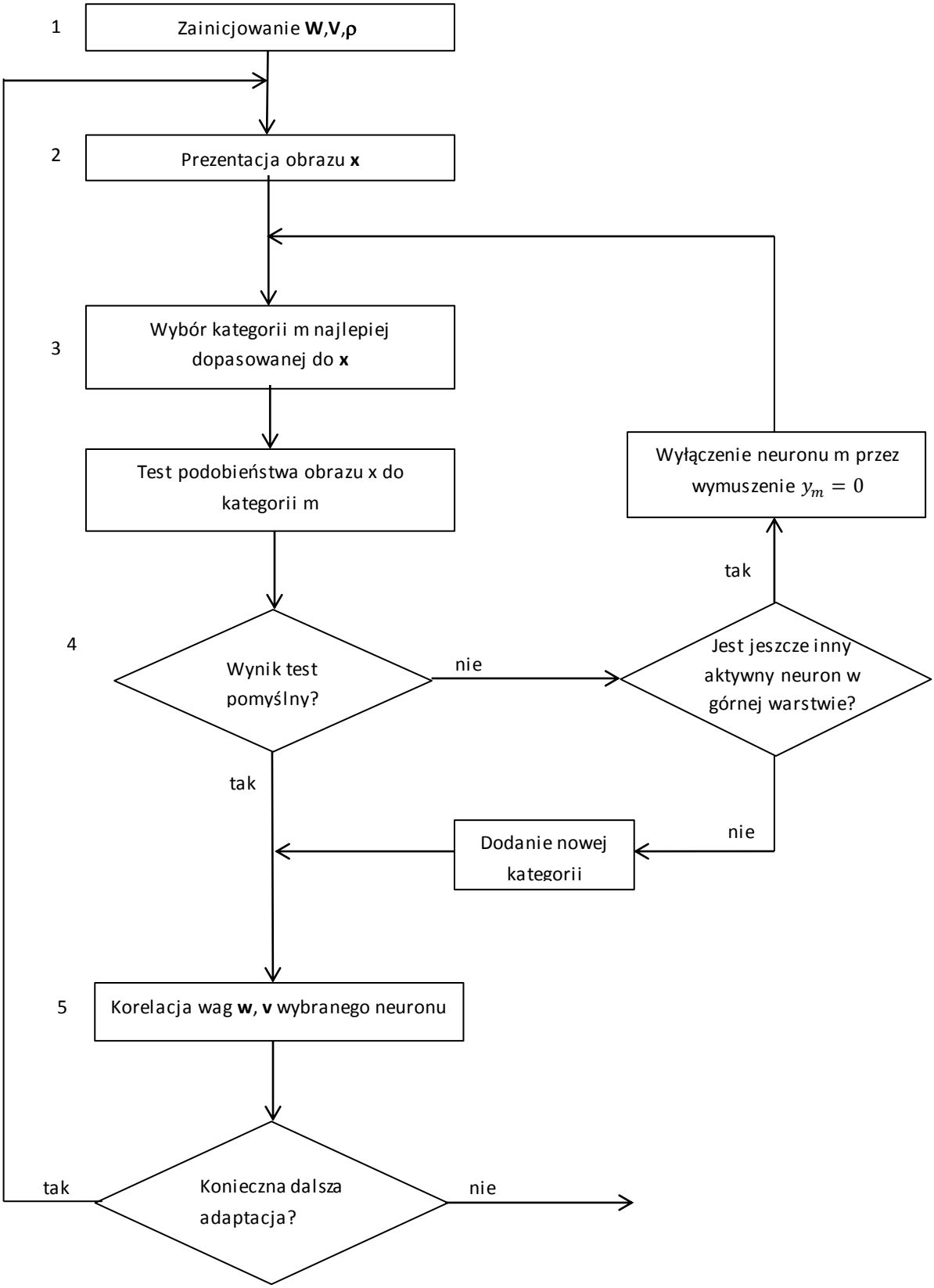
$$p_m \overset{\text{def}}{=} \frac{\sum_{i=1}^{n} v_{im} x_i}{\sum_{i=1}^{n} x_i} > \text{p},$$

Jeżeli wynik działania wyjdzie negatywny czyli dopasowanie jest poniżej progu, ale jeszcze są aktywne neurony w górnej warstwie, to wyzeruje wyjście testowe neuronu, czyniąc go w ten sposób nieaktywnym, następnie wraca do punktu 3 dopasowując obraz testowy do kolejnej kategorii. Jeśli w górnej warstwie nie ma już aktywnych neuronów, tworzy kolejny neuron z nowym numerem.

5. Adaptacja nowych wag dla wektorów **v i w**. Dla wytypowanego neurony następuje obliczenia dla i =1,2,…n:

$$v_{im}(t+1) = v_{im}(t) x_i,$$
$$w_{im}(t+1) = \frac{v_{im}(t+1)}{0{,}5 + \sum_{i=1}^{n} v_{im}(t) x_i}$$

Po czym wszystkim wróć do kroku 2 wczytania kolejnego obrazu.


1
Zainicjowanie **W**,**V**,ρ
2
Prezentacja obrazu **x**
3
Wybór kategorii m najlepiej dopasowanej do **x**
Test podobieństwa obrazu x do kategorii m
4
Wynik test pomyślny?
nie
Jest jeszcze inny aktywny neuron w górnej warstwie?
tak
Wyłączenie neuronu m przez wymuszenie $y_m = 0$
tak
nie
Dodanie nowej kategorii
5
Korelacja wag **w**, **v** wybranego neuronu
Konieczna dalsza adaptacja?
tak
nie

IMPLEMENTACJA ALGORYTMU W PROGRAMIE

Część kodu odpowiadająca za implementację sztucznej inteligencji podzielona jest na 2 klasy główne. Pierwsza klasa czyli *neuron* odpowiada za tworzenie, przechowywanie informacji o neuronie oraz najważniejsze metody wykorzystujące w sieci. Druga klasa *network* odpowiada za tworzenie sieci oraz wywoływanie metod neuronu.

1. Pierwsza część algorytmu odpowiedzialna za stworzenie pierwszego neuronu oraz nadająca początkowe wartości wektorom **w** i **v** przedstawiono poniżej:

```
1    public Neuron(float pixelcount){
2        for(int i=0; i < pixelcount;i++){
3            this.w.add(1.0f/(1.0f+pixelcount));
4            this.v.add(1f);
5        }
6    }
     public void addToNeuron_list(int[] image) {
2        this.neuron_list.add(new Neuron(image.length));
3    }
```

Pierwsza część kodu jest konstruktorem klasy *neuron,* która przyjmuje na wejściu liczbę pikseli oraz implementuje równania z algorytmu. Inicjalizacja pierwszego neuronu zostaje wykonana w metodzie *addToNeuron_list* zawartej w klasie Network. Metoda ta przyjmuje na wejściu tablicę typu integer, zawierającą obraz binarny. Następnie nowy neuron dodawany jest do klasy kolekcji typu ArrayList za przechowującej obiekty klasy neuron.

2. Kolejnym krokiem algorytmu jest obliczenie wszystkich miar dopasowani dla wczytanego obrazu w klasie *network*. Odpowiadają za to dwie części programu.

```
1    public void calc_match(int[] current_image){
2        this.match =0f;
3        for (int i=0; i < current_image.length ;i++) {
4            this.match += current_image[i] * this.w.get(i);
5        }
6
7        System.out.println("match:" + this.match);
8    }
```

```
1    public Neuron returnWinnerNeuron(int[] current_image){
```

```
        for (Neuron neuron_list1 : neuron_list) {
            neuron_list1.calc_match(current_image);
        }
```

Pierwsza metoda zawarta jest w klasie *neuron*. Przyjmuje ona na wejściu wektor binarny aktualnie analizowanego obrazu. Metoda ta implementuje równanie zawarte dla kroku drugiego naszego algorytmu. Drugi fragment kodu przedstawia metodę *returnWinnerNeuron* zawartą w klasie *network.* Jest to główna metoda dla całej sieci. Kolejne kroki generowania neuronów będą zawarte w tej metodzie. Powyższy element metody inicjalizuje funkcję *calc_match* z klasy *neuron,* która oblicza miarę dopasowania dla każdego obiektu znajdującego się w liście neuronów.

3. Kolejny fragment kodu ma za zadanie wyszukać najlepszego dopasowania aktualnie wczytanego obrazu:

```
            Neuron neuro_n = this.neuron_list.get(0);
            for (int j = 0; j < neuron_list.size(); j++) {
                if(neuron_list.get(j).getMatch() > neuro_n.getMatch()){
                    neuro_n = neuron_list.get(j);
                }
```

Pierwsza linia tworzy zmienną typu neuron do której przypisany jest pierwszy obiekt z listy neuronów. Następnie w pętli for wyszukiwane jest najlepsze dopasowanie. Jeżeli aktualnie porównywany element jest większy od elementu zawartego zmienne *neuron_n*, do zmiennej zostaje przypisane aktualny element. Jest to odpowiednik metody max w powyższym algorytmie.

4. Test podobieństwa obrazu x do kategorii m zaimplementowany w następującej metodzie:

```
    public void calc_similarity(int[] current_image){
        this.similarity =0f;
        float similarity_bottom = 0;
        for (int i=0; i < current_image.length ;i++) {
            this.similarity += current_image[i] * this.v.get(i);
            similarity_bottom += current_image[i];
        }
        this.similarity /= similarity_bottom;
    }
```

Metoda jako parametr pobiera tablice aktualnie analizowanego obrazka. Pierwszą czynnością w tej metodzie jest wyzerowanie zmiennej *similarity* przechowującej wartość podobieństwa. Zmienna *similarity_bottom* reprezentuje mianownik wzoru w punkcie 4 tym algorytmu, co oznacza, ze jest to suma wszystkich elementów tablicy obrazu binarnego. Następnie pętla for realizuje jednocześnie obie sumy we wzorze i przydziela im odpowiednio wyliczone wartości. Działanie metody zakończone jest podzieleniem licznika i mianownika, przekazując wartość do pola *similarity* klasy Neuron.

Krok czwarty obejmuje kolejną czynność jaką jest sprawdzenie czy wartość *similarity* jest większa, niż ustalony próg podobieństwa. W przypadku powodzenia testu, następuje adaptacja wag neuronu, jednak w przypadku niepowodzenia wykonywany jest dalszy szereg czynności.

W pierwszej kolejności sprawdzany jest czy dopasowanie jest równe zero. Jeżeli jest równe zero to oznacza, że obraz nie pasuje do żadnego neuronu. Dla takiego przypadku tworzona jest nowa kategoria. W innym przypadku sprawdzany jest czy *similarity* jest powyżej progu ustanowionego na wejściu i w dalszym etapie następuje adaptacja wag dla neuronu z najwyższą wartością *match*.

```
for (int i = 0; i < neuron_list.size(); i++) {

•••

if (neuro_n.getMatch() == 0f){
                Neuron new_neuron= new Neuron(current_image.length);
                new_neuron.adaptat(current_image);
                this.neuron_list.add(new_neuron);
                 neuronID = neuron_list.indexOf(new_neuron);
                return new_neuron;
            }

            neuro_n.calc_similarity(current_image);
             if(neuro_n.getSimilarity() > this.p){
                neuro_n.adaptat(current_image);
                neuronID = neuron_list.indexOf(neuro_n);
                return neuro_n;
            }else{
                neuro_n.setMatch(0f);
            }
    }
```

Jeżeli obiekt nie pasuje do powyższych warunków neuron zostaje wyłączony poprzez wymuszenie *match* = 0, a pętla przechodzi do kolejnego elementu w górnej warstwie.

```
        Neuron new_neuron= new Neuron(current_image.length);
            new_neuron.adaptat(current_image);
            this.neuron_list.add(new_neuron);
            neuronID = neuron_list.indexOf(new_neuron);
            return new_neuron;
```

Jeśli wszystkie obiekty z górnej warstwy nie zostały dopasowane do powyższych warunków zostanie utworzony nowy obiekt kategorii, przyjmujący aktualny obraz.

5. Po wykonaniu wszystkich czynności z punktów od drugiego do czwartego, następuje wcześniej wspomniana adaptacja, która jest zaimplementowana w metodzie *adapt()*.

```
public void adaptat(int[] current_image){
        float suma_dol = 0.5f;

        for (int i = 0; i < this.v.size(); i++) {
            suma_dol += this.v.get(i) * current_image[i];
        }

        for (int i = 0; i < this.v.size(); i++) {
            this.v.set(i, this.v.get(i) * current_image[i]);
            this.w.set(i, this.v.get(i) / suma_dol);
        }
    }
```

Po zakończeniu adaptacji wag wczytywany jest kolejny obraz, pomijając punkt pierwszy.

Tworzenie sieci

```
Network siec1 = new Network();

                    •••


public void createNetwork(File[] f){
        String firstImage = f[0].getAbsolutePath();
        //Pierwszy neuron
        ConvertImage image = new ConvertImage(firstImage);
        if(siec1.getNeuron_list().size() == 0){
        siec1.addToNeuron_list(image.getImgBinaryTable());
    }
```

```
15          for (File f1 : f) {
16
17              String filename = f1.getAbsolutePath();
18              ConvertImage image1 = new ConvertImage(filename);
19              siec1.returnWinnerNeuron(image1.getImgBinaryTable());
20          }
21      }
```

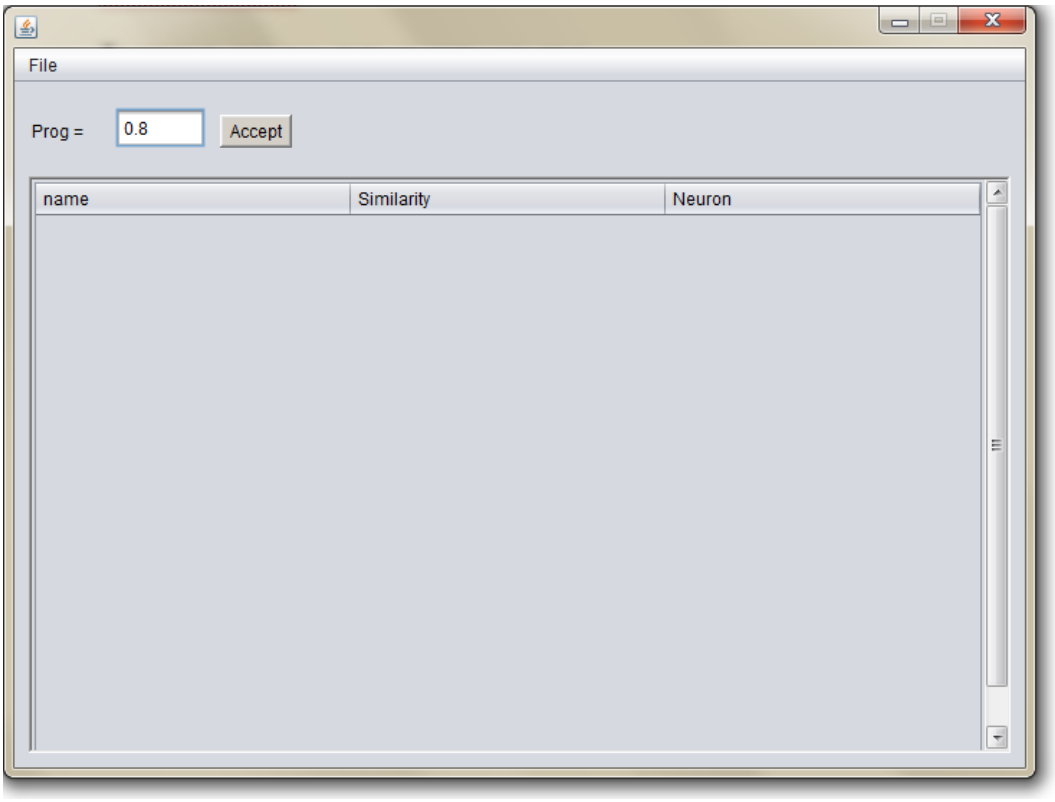
Tworzenie sieci rozpoczyna się od przygotowania obiektu *siec1* typu *Network*. Następnie obiekt ten zostaje wykorzystany w metodzie *createNetwork,* która odpowiada za właściwe tworzenie sieci. Pierwszym krokiem jest przygotowanie inicjalizującego obrazu, służącego za pierwszy neuron. Obrazy są wywoływane według tablicy ścieżek pobranych podczas wybierania plików, a sam obraz pobierany jest dopiero kiedy następuje jego użycie. Konstruktor klasy *ConvertImage* pobiera tablice binarną z wczytanego obrazu. Sprawdzana jest lista neuronów, w przypadku gdy jest pusta zostaje dodany pierwszy neuron do obiektu *siec1.*

Po przygotowaniu sieci i zainicjalizowaniu jej działania, rozpoczyna się pobieranie kolejnych obrazów z listy ścieżek, spośród których wyłaniane są zwycięskie neurony metodą *returnWinnerNeuron.*

INTERFEJS PROGRAMU

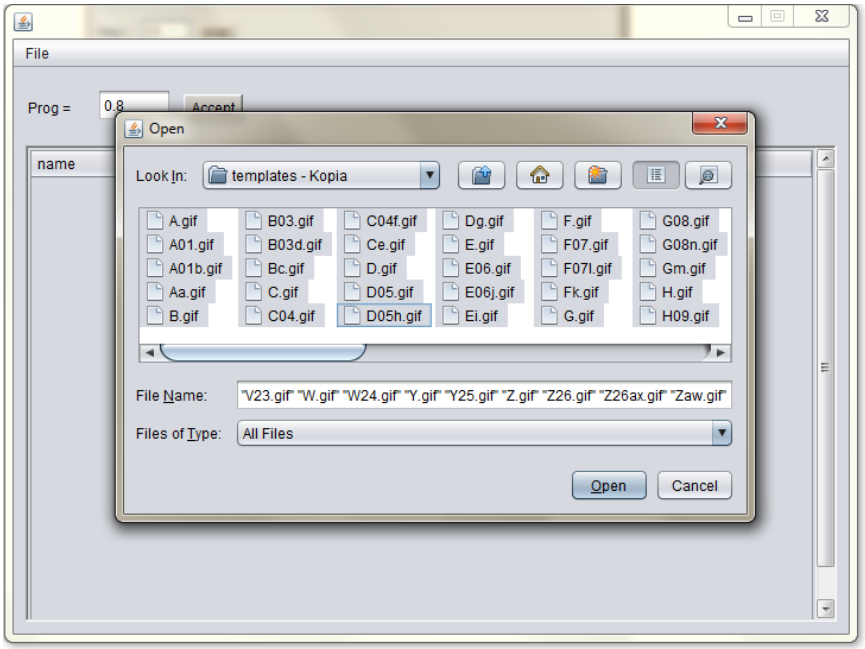
Interfejs programu składa się z trzech części – Panel Menu, edycja progu czujności oraz tabela wyników. Panel Menu posiada sekcje File, w której możemy wczytać obrazy do uczenia sieci. Edycja progu czujności jest możliwa jest poprzez wpisanie w edytowalne okienko wartości od 0-1, którą trzeba potwierdzić przyciskiem „Accept". Domyślnie próg jest ustawiony na 0.8, i jest on już wczytany do programu. Tabela wyników składa się z trzech kolumn przedstawiające odpowiednio: name – nazwa wczytanego pliku, similarity – podobieństwo wczytanego pliku do wcześniejszych obrazów, neuron – numer kategorii do której przyporządkowano obraz.

Okno główne programu:

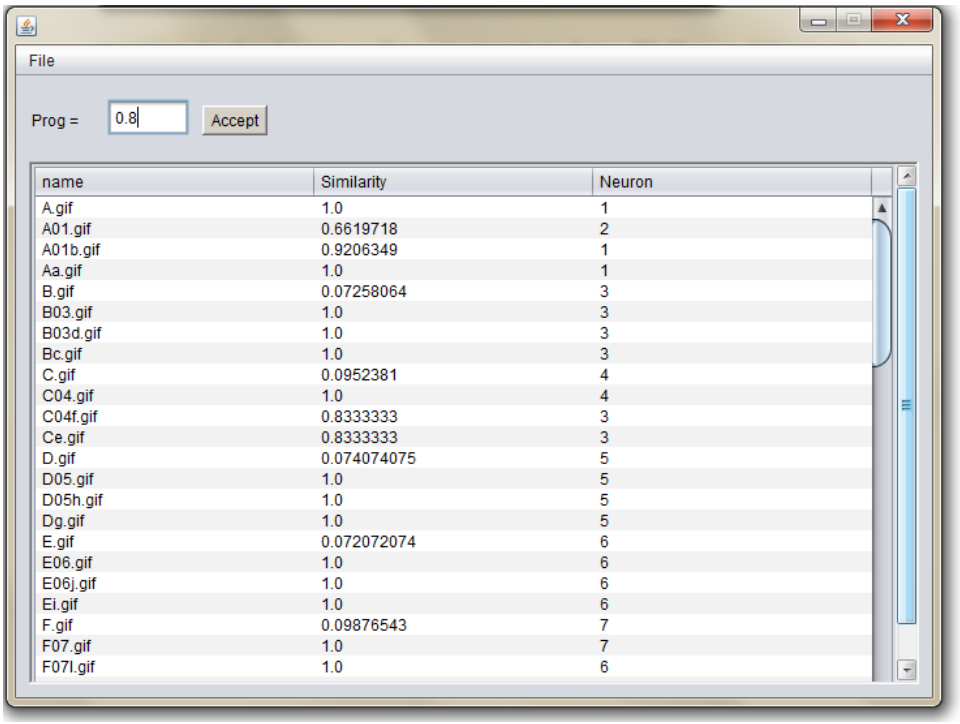


Importowanie obrazów jest zrealizowane przy użyciu Swing JFileChooser, który umożliwia pobieranie wielu plików naraz, co usprawnia proces wczytywania.

Eksplorator plików:



Przykładowe użycie:



BIBLIOGRAFIA

Jacek Maciej Żurada, Mariusz Barski i Wojciech Jędruch Wyd. Naukowe PWN(1996): *Sztuczne sieci neuronowe: podstawy teorii i zastosowania,* ISBN-9788301121068