Gliwice, 12.06.2017 r.

Biologically Inspired Artificial Intelligence

Rozpoznawanie liter przy użyciu sieci neuronowych

Prowadzący: mgr inż. Marcin Wierzchanowski

Wykonanie: Alicja Lachman, Mateusz Ligus

GKiO3, sekcja 7

Informatyka, semestr 6

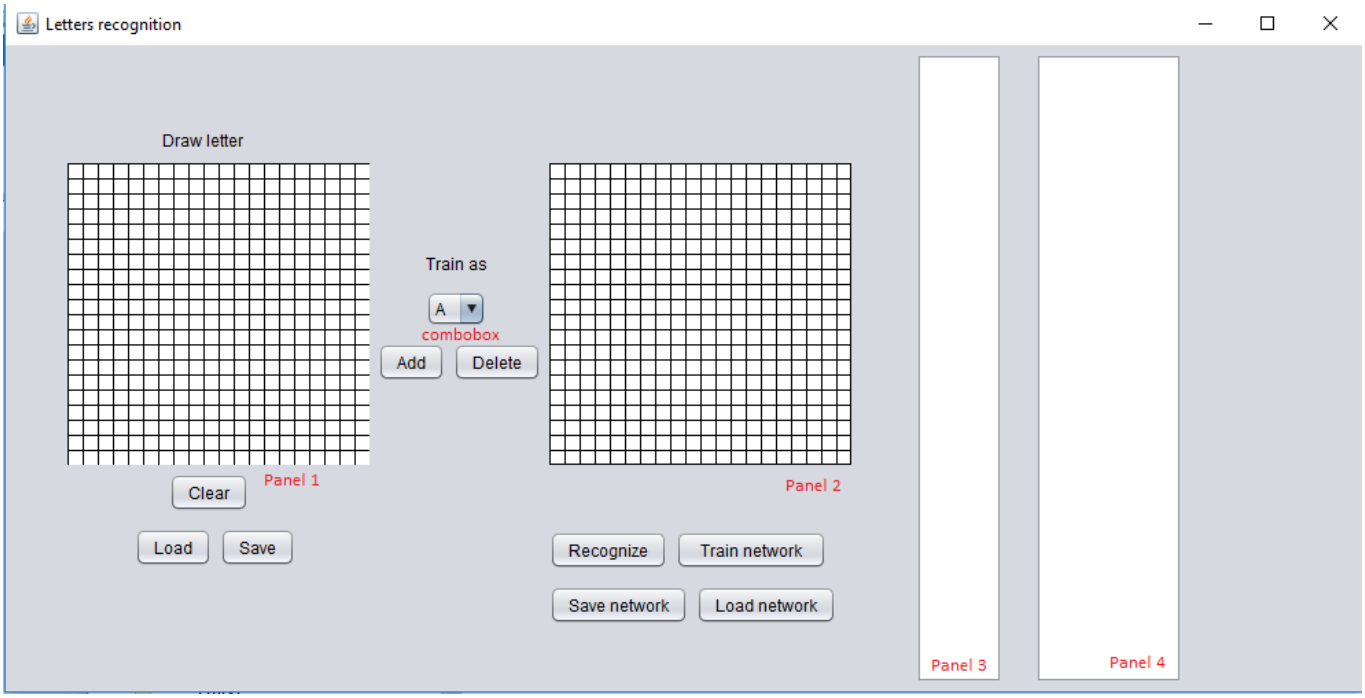
1. Temat projektu

Tematem projektu jest aplikacja umożliwiająca rozpoznawanie liter wprowadzonych do aplikacji przez użytkownika. Podczas prac nad projektem zdecydowano się na stworzenie aplikacji okienkowej w języku Java, z użyciem frameworka Swing, który pozwala na intuicyjne tworzenie okienek. Dodatkowo zdecydowano się na wykorzystanie biblioteki Encog. Biblioteka ta powstała w 2008 roku i jest zaawansowanym frameworkiem wspierającym między innymi tworzenie sieci neuronowych.

2. Specyfikacja zewnętrzna

Aplikacja składa się z jednego głównego widoku, umożliwiającego wykorzystanie przez użytkownika wszystkich funkcjonalności aplikacji.

a. Opis elementów



Rys. 1 Widok aplikacji po uruchomieniu

Panel 1 – panel w którym użytkownik rysuje literę, która ma być dodana do bazy lub rozpoznana przez program. W początkowej fazie prac nad aplikacją panel ten umożliwiał swobodne rysowanie liter, które następnie były przekształcane na siatkę pikseli, jednak powodowało to powstawanie błędów podczas dodawania liter do programu. Zastosowanie siatki o takich samych właściwościach jak analizowane litery znacząco poprawiło działanie aplikacji.

Panel 2 – panel w którym wyświetlane są „wzorowe” litery, po rozpoznaniu danej litery przez program. Ma charakter czysto informacyjny, nie jest związany z działaniem sieci neuronowej. Dodatkowo umożliwia podgląd już zapisanych liter.

Panel 3 – panel wyświetlający listę próbek liter dodanych do programu. Można w nim śledzić, ile przykładów dla danej litery zostało dodanych do bazy, oraz czy wszystkie są poprawne, na przykład czy przypadkowo nie zapisaliśmy narysowanej litery ‘C’ jako literę ‘A’. Po kliknięciu w jeden z elementów listy, na panelu 2 zostanie wyświetlony zapisany sposób narysowania danej litery.

Panel 4 – panel 4 uaktualnia się po wywołaniu akcji rozpoznawania narysowanej litery. Wyświetlane są wtedy wszystkie litery z alfabetu wraz z odpowiadającym im rezultatem zwróconym z sieci neuronowej, z zakresu od 0 do 1. Litera z wartością najbardziej zbliżoną do 1 jest literą rozpoznaną przez sieć neuronową jako ta narysowana przez użytkownika w panelu 1.

Przycisk „Clear” – umożliwia wyczyszczenie panelu 1, aby można było narysować kolejną literę lub poprawić literę błędnie narysowaną.

Przycisk „Load” – umożliwia załadowanie z pliku zlokalizowanego w folderze aplikacji dodanych poprzednio liter.

Przycisk „Save” – zapisuje dodane w aplikacji litery do pliku tekstowego, aby następnie można było je ponownie załadować.

Combobox – służy do wyboru litery, która ma zostać dodana do bazy. Litera powinna zostać wybrana z listy przed naciśnięciem przycisku „Add”.

Przycisk „Add” – dodaje literę narysowaną w panelu 1 do listy, która później jest wykorzystywana w sieci neuronowej.

Przycisk „Delete” – usuwa aktualnie zaznaczony element z listy z panelu nr 3.

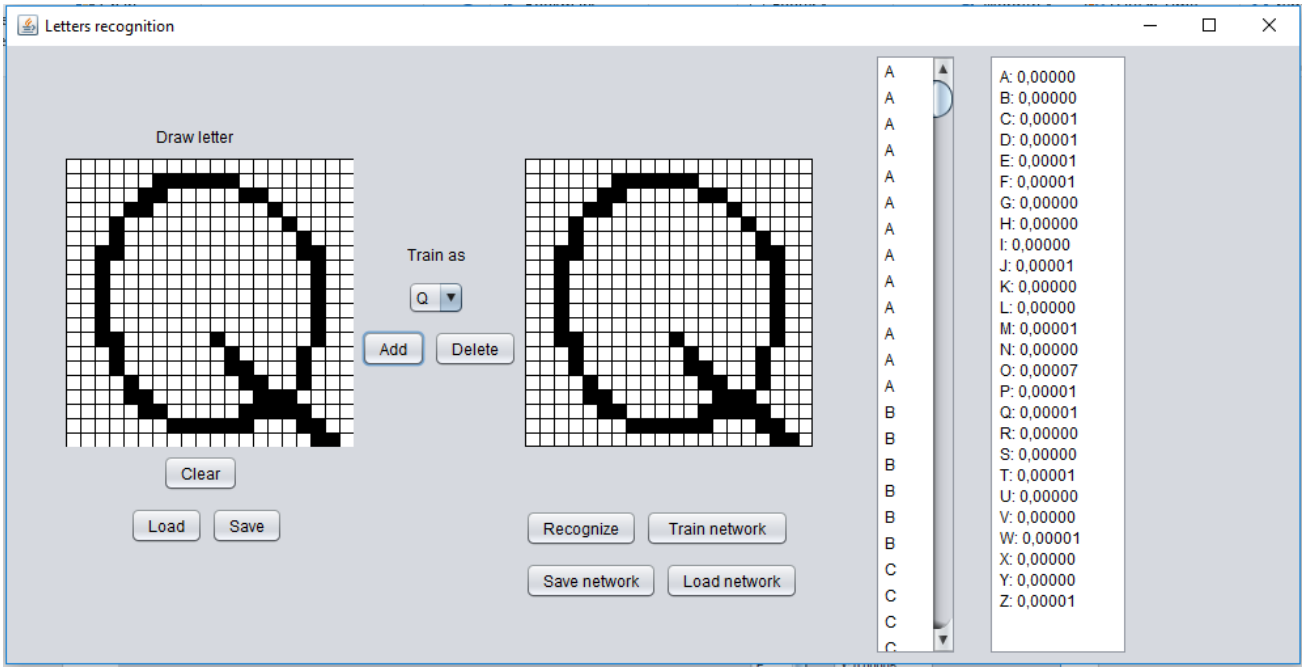
Przycisk „Recognize” – uruchamia rozpoznawanie przez sieć neuronową litery narysowanej przez użytkownika w panelu nr 1. Po rozpoznaniu litery wyświetlony zostanie komunikat która litera została rozpoznana, a na panelu nr 4 pojawi się lista wszystkich liter, wraz z rezultatem zwróconym przez sieć neuronową. Jeżeli sieć nie została jeszcze nauczona, zostanie wyświetlony odpowiedni komunikat.

Przycisk „Train network” – uruchamia proces nauczania sieci neuronowej. Po zakończeniu nauki wyświetlony zostanie komunikat.

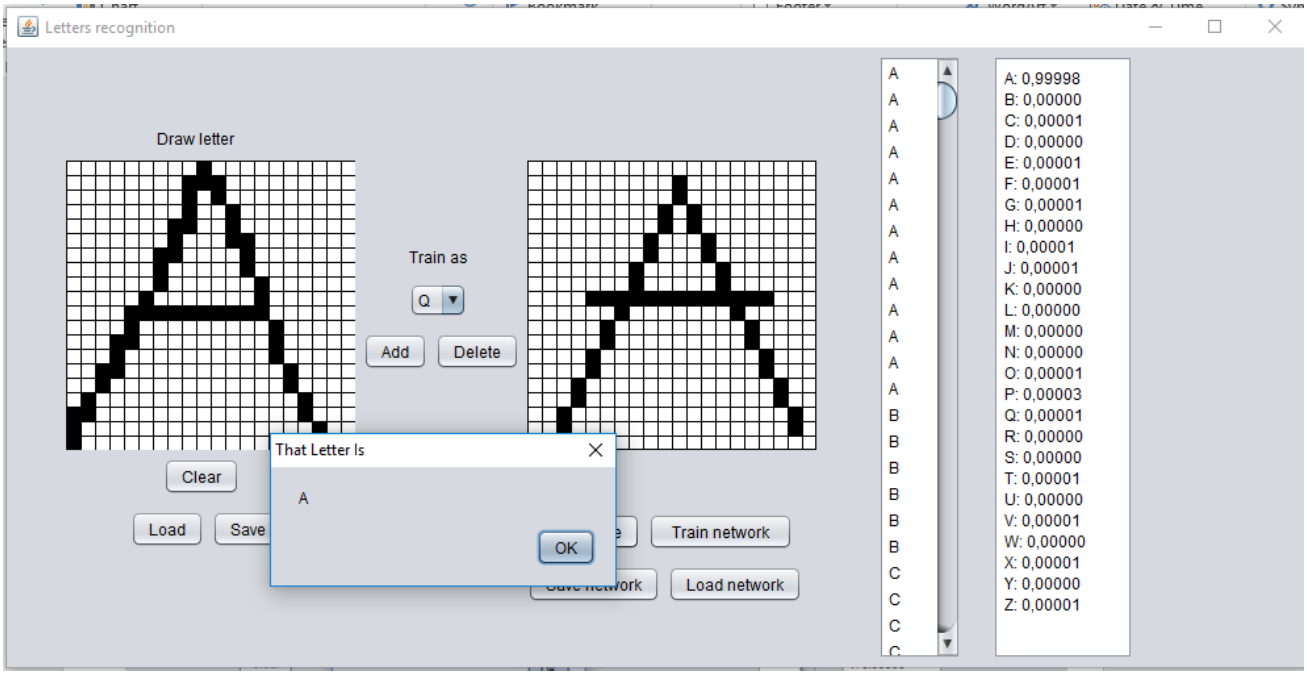
Przycisk „Save network” – uruchamia serializację i zapis nauczonej sieci neuronowej do pliku.

Przycisk „Load network” – ładuje do aplikacji uprzednio zapisaną do pliku nauczoną sieć neuronową.

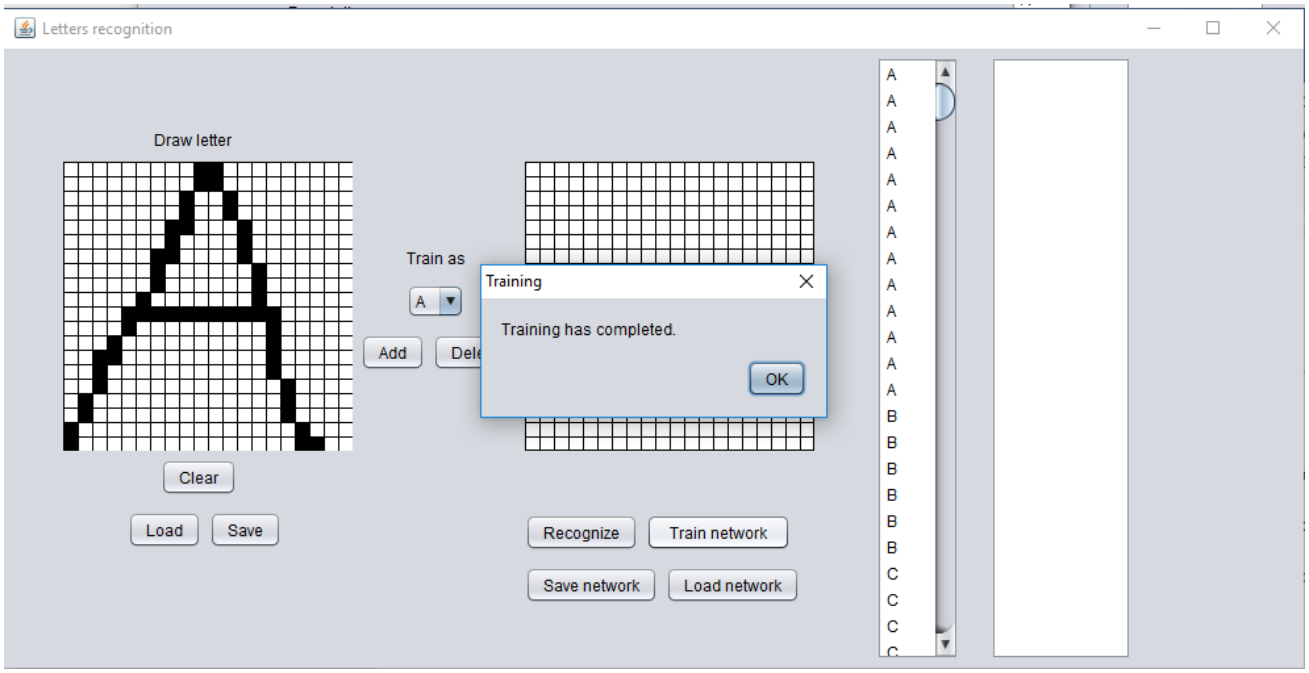
b. Przykłady z działania aplikacji



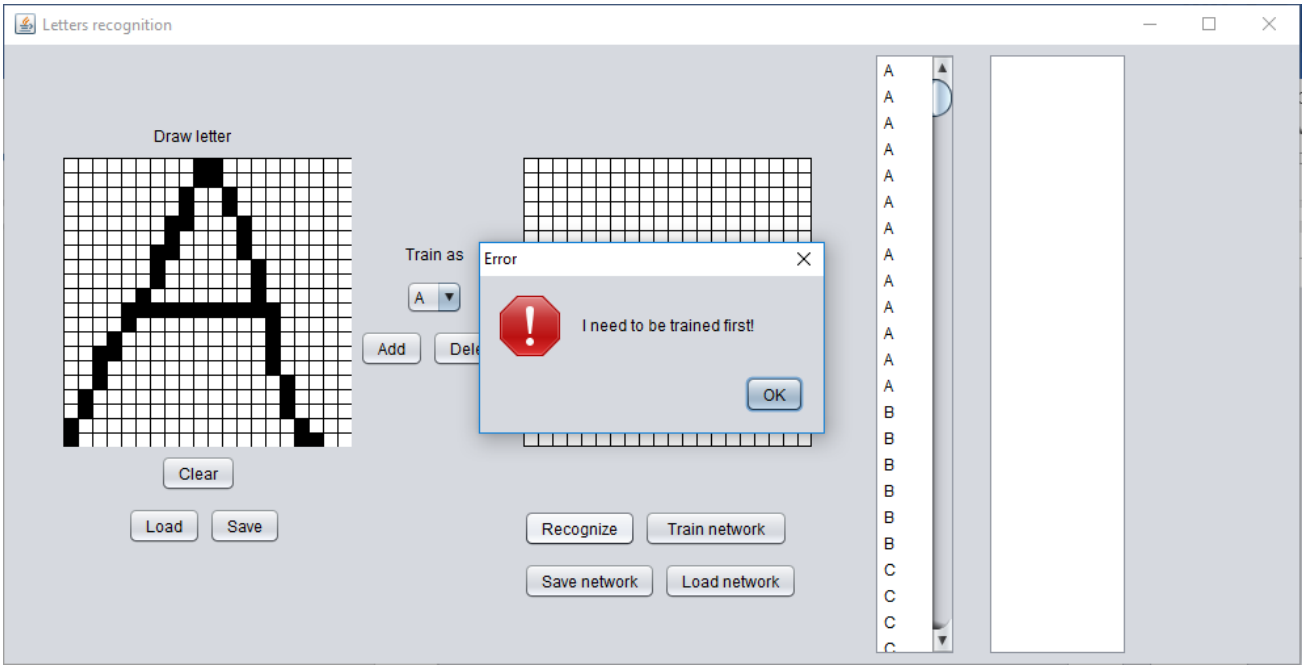
Rys. 2 Przykładowy widok aplikacji po dodaniu do listy litery Q.



Rys. 3 Przykładowy widok aplikacji po rozpoznaniu litery A.



Rys. 4 Przykładowy widok aplikacji po zakończeniu nauczania sieci neuronowej.



Rys.5 Przykładowy widok aplikacji po próbie rozpoznania litery bez uprzedniego nauczenia sieci neuronowej.

c. Instrukcja obsługi programu

1. Uruchomić aplikację.
2. Załadować listę wprowadzonych wcześniej próbek liter, poprzez kliknięcie przycisku „Load”
   Lub
   Dodanie kolejno próbek liter poprzez narysowanie litery w panelu nr 1, wybraniu odpowiedniej litery z comboboxa oraz kliknięcie przycisku „Add”. Proces ten należy powtórzyć wielokrotnie dla każdej litery, w celu uzyskania jak najlepszej dokładności sieci neuronowej.
3. Kliknąć przycisk „Train network”
   Lub
   Kliknąć przycisk „Load network”.
4. Narysować w panelu nr 1 dowolną literę oraz kliknąć przycisk „Recognize”.
5. Wyświetlony zostanie komunikat o rozpoznanej literze, a na panelu nr 4 z boku aplikacji pokazane zostanie z jaką dokładnością została rozpoznana dana litera.

3. Specyfikacja wewnętrzna i opis biblioteki

Sieci neuronowe zbudowane są z warstw podobnych neuronów. Sieć neuronowa zawiera co najmniej warstwę wejściową i warstwę wyjściową. Sieć neuronowa zawsze zwraca jakiś wynik, niekoniecznie w 100% poprawny, ale jak najbardziej zbliżony do poprawnego.

Warstwa wejściowa

Warstwa wejściowa jest pierwszą warstwą w sieci neuronowej. Zawiera ściśle określoną liczbę neuronów, z których każdy opisuje jeden z atrybutów wykorzystywanych w klasyfikacji, regresji lub clusteringu sieci neuronowej.

W programie do rozpoznawania liter warstwa wejściowa składa się z takiej ilości neuronów, która odpowiada ilości pól na siatce obrazującej przetworzoną literę (wysokość * szerokość siatki). Dane wejściowe sieci neuronowej to tablica obiektów typu double. Wielkość tej tablicy odpowiada ilości neuronów wejściowych. Każdemu pikselowi przyporządkowane są odpowiednio wartości -0.5 lub 0.5 – pierwsza w przypadku piksela pustego, druga dla wypełnionego.

Biblioteka Encog udostępnia interfejs MLData, opisujący dane w formie tablicy double, które następnie mogą być użyte w sieci neuronowej. Podstawową klasą implementującą interfejs MLData jest BasicMLData.

Warstwa wyjściowa

Warstwa wyjściowa jest końcową warstwą w sieci neuronowej. Warstwa ta dostarcza wyniki, po uprzednim przetworzeniu ich przez wcześniejsze warstwy. Ta warstwa również reprezentowana jest jako tablica elementów typu double, poprzez implementację interfejsu MLData. Sieci neuronowe posiadają bardzo dobre zdolności rozpoznawania wzorców, w związku z tym powinny być w stanie zwrócić pożądany wynik nawet wówczas, gdy dane wejściowe są lekko zniekształcone. W przygotowanej aplikacji warstwa wyjściowa składa się z 26 neuronów – każdy z nich odpowiada jednej literze alfabetu. Wartości neuronów wyjściowych są z zakresu 0.0 do 1.0, neuron z najwyższą wartością uznawany jest za „zwycięzcę” klasyfikacji, a tym samym określa jaka litera została rozpoznana. W programie jako funkcję aktywującą zastosowano funkcję aktywującą Elliota.

Ukryte warstwy

Warstwy ukryte stosowane są aby poprawić efektywność pracy sieci neuronowej. W pierwszej kolejności definiowane są warstwy wejściowa i wyjściowa, następnie można przejść do definiowania optymalnej struktury ukrytych warstw. Jest to ważny moment w tworzeniu sieci neuronowej, ponieważ zbyt skomplikowana struktura może uczyć się zbyt wolno, a zbyt prosta – nie osiągnąć oczekiwanych rezultatów. Testy różnych konfiguracji rozpoczęto, zgodnie z zaleceniami autora biblioteki, od jednej warstwy ukrytej z liczbą neuronów równą połowie liczby neuronów wejściowych. Ostatecznie w programie zastosowano jedną warstwę ukrytą ze 100 neuronami, korzystającą z funkcji aktywującej Elliota.

Proces tworzenia sieci neuronowej w aplikacji:

Aby nauczyć sieć neuronową, skorzystano z obiektu typu BasicMLDataSet. Jest on listą par typu BasicMLDataPair, zawierających dane wejściowe (tablica double przedstawiające narysowane litery jako wypełnione i puste piksele), oraz wartość idealną, w przypadku aplikacji jest to tablica 26-elementowa, zawierająca wartość 1.0 w miejscu odpowiadającym danej literze.

```
final MLDataSet trainingSet = new BasicMLDataSet();
        for (int i = 0; i < letterListModel.size(); i++) {
            final MLData item = new BasicMLData(inputNeuron);
            int idx = 0;
            final DownsampledData downsampledData = (DownsampledData) this.letterListModel
                    .getElementAt(i);
            for (int y = 0; y < downsampledData.getHeight(); y++) {
                for (int x = 0; x < downsampledData.getWidth(); x++) {
                    item.setData(idx++, downsampledData.getDataForPixel(x, y) ? .5 : -.5);
                }
            }
            MLData ideal = getIdealForLetter(downsampledData.getLetter());
            trainingSet.add(new BasicMLDataPair(item, ideal));
        }
```

Aby utworzyć nową sieć neuronową, skorzystano z klasy BasicNetwork z biblioteki Encog. Następnie zdefiniowano trzy warstwy – warstwę wejściową, warstwę ukrytą oraz warstwę wyjściową.

Poszczególne warstwy zdefiniowane są w poprzez przekazanie 3 parametrów do konstruktora klasy BasicLayer – funkcji aktywującej, wartości boolowskiej określającej, czy dana warstwa zawiera tzw. Neuron bias, oraz liczbę neuronów w danej warstwie. Warstwa wyjściowa nie posiada neuronu bias, ponieważ jest on połączony tylko z następną warstwą. Warstwa wyjściowa jest ostatnią warstwą, więc taki neuron jest zbędny. Warstwa wejściowa nie posiada funkcji aktywującej, ponieważ funkcje te wpływają na dane pochodzące z warstw poprzednich, których warstwa wejściowa nie posiada. Domyślną funkcją aktywującą jest tangens hiperboliczny. Funkcje aktywujące są dołączane do warstw i wykorzystywane do skalowania danych wyjściowych z warstwy.

Następnie należy wywołać metodę finalizeStructure(), która informuje sieć neuronową, że jej struktura jest gotowa i nie będzie już zmieniana. Wywołanie funkcji reset() powoduje zrandomizowanie wag w połączeniach pomiędzy warstwami. Sieci neuronowe zaczynają od losowych wag. W trakcie nauki wagi te są zmieniane w taki sposób, aby uzyskać pożądany wynik. Z uwagi na tę losowość, wyniki danej sieci neuronowej dla różnych treningów mogą się różnić, w przypadku uzyskania niezadowalających efektów można powtórzyć proces uczenia.

Kolejnym etapem jest rozpoczęcie nauki. Uczenie polega na takim doborze wag neuronów by w efekcie końcowym błąd popełniany przez sieć był mniejszy od zadanego. Nauka trwa określoną ilość iteracji, można również obserwować stopień błędu dla danej iteracji. W aplikacji skorzystano z treningu typu Resilient Propagation. Jest to jeden z treningów propagacyjnych, które są treningami nadzorowanymi. Oznacza to, że do algorytmu dostarczany jest zestaw danych zawierający dane wejściowe oraz wartości idealne, oczekiwane dla poszczególnych danych wejściowych. Algorytmy propagacyjne przechodzą przez szereg iteracji, z których każda powinna zminiejszyć stopień błędu sieci. Stopień błędu jest procentową różnicą pomiędzy faktycznym wynikiem zwracanym przez sieć, a wartością idealną dostarczoną do sieci. W każdej iteracji, dla każdego elementu z zestawu danych wejściowych zostanie wprowadzona korekta w macierzy wag. Dla każdego elementu wykonywane są 2 kroki. W pierwszym, tzw. „forward pass", dane są prezentowane sieci neuronowej tak, jakby nie odbywała się właśnie nauka. Następnie obliczany jest błąd pomiędzy wartością zwróconą przez sieć, a wartością oczekiwaną. Zapamiętywany jest wynik dla każdej z warstw. Następnie wykonywany jest tzw. „backward pass", rozpoczynający się od warstwy wyjściowej, aż do warstwy wejściowej. Sprawdzany jest błąd dla każdego neuronu, a następnie obliczany jest gradient błędu przy pomocy pochodnej funkcji aktywującej (z tego powodu dla treningów propagacyjnych można stosować tylko funkcje aktywujące posiadające pochodne).

```
        network = new BasicNetwork();
        int hiddenLayerNeuronsCount = 100;
        network.addLayer(new BasicLayer(null, true, Config.DOWNSAMPLE_HEIGHT
                * Config.DOWNSAMPLE_WIDTH));
        network.addLayer(new BasicLayer(new ActivationElliott(), true,
hiddenLayerNeuronsCount));
        network.addLayer(new BasicLayer(new ActivationElliott(), false, 26));
        network.getStructure().finalizeStructure();
        network.reset();

        final Propagation train = new ResilientPropagation(network, trainingSet);
        int epochsCount = EPOCHS_COUNT;

        for (int epoch = 1; epoch < epochsCount; epoch++) {
            train.iteration();
            System.out.println("epoch #" + epoch + " error: " + train.getError());
        }
```

Aby rozpoznać narysowaną literę, wywoływana jest metoda compute() na sieci neuronowej. Jako parametr przyjmuje ona obiekt typu BasicMLData, czyli w naszym przypadku tablicę wartości pikseli dla narysowanej litery. Sieć neuronowa zwraca obiekt MLData zawierający tablicę z 26 wartościami typu double. Indeks, którego wartość jest najbardziej zbliżona do 1 jest indeksem rozpoznanej litery.

```
        final MLData input = new BasicMLData(Config.DOWNSAMPLE_HEIGHT *
Config.DOWNSAMPLE_WIDTH);

        int idx = 0;

        final DownsampledData downsampledData =
downsampledDataJPanel1.getDownsampledData();

        for (int y = 0; y < downsampledData.getHeight(); y++) {

            for (int x = 0; x < downsampledData.getWidth(); x++) {

                input.setData(idx++, downsampledData.getDataForPixel(x, y) ? .5 : -.5);

            }

        }

        MLData result = network.compute(input);

        displayResults(result);


        final char best = getWinnerLetter(result);
```

Biblioteka Encog umożliwia również serializowanie wytrenowanych sieci neuronowych i zapis do pliku. Jest to przydatna opcja, ponieważ dla dużej liczby próbek oraz skomplikowanej struktury sieci czas nauki może być długi. Ponadto biblioteka ta pozwala na zastosowanie wielowątkowości w procesie uczenia sieci.

4. Wnioski

Podczas prac nad projektem zapoznaliśmy się z możliwiościami biblioteki Encog oraz ogólnymi konceptami związanymi z sieciami neuronowymi. Możliwości wykorzystania sieci neuronowych w pracach naukowych są ogromne. Szeroki wybór opcji w bibliotece Encog ułatwiał testowanie optymalnego rozwiązania dla aplikacji – w łatwy sposób można było zmienić ilość epok, neuronów w warstwie ukrytej czy funkcji aktywujących w poszczególnych warstwach. Umożliwiło to zapoznanie się z wieloma dostępnymi metodami trenowania sieci. Początkowe założenia dotyczące projektu zmieniały się w trakcie prac nad programem. Zrezygnowano z wprowadzania próbek liter w formie obrazów ładowanych z dysku oraz zmieniono sposób rysowania liter na ekranie. Zmiany te były spowodowane chęcią skupienia się na samym zagadnieniu sieci neuronowych, a nie na innych zagadnieniach programistycznych, nie związanych ściśle z tematyką przedmiotu. Wykonany program dość dobrze radzi sobie z rozpoznawaniem narysowanych liter. Problematyczne okazuje się rozpoznawanie liter podobnych, tj. C i D, C i O, O i Q, R i K.Aby zwiększyć dokładność sieci neuronowej, należy dostarczyć jej większą ilość odpowiednio przygotowanych danych wejściowych.