UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki

Wojciech Denejko
nr albumu: 214 300

Rozpoznawanie tekstu w aplikacjach mobilnych

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
dr Tomasz Borzyszkowski

Gdańsk 2017

Streszczenie

Niniejsza praca ma na celu stworzenie aplikacji rozpoznającej tekst w języku polskim oraz angielskim, charakteryzującej się kompatybilnością z systemami iOS oraz Android. Do wytworzenia aplikacji zostanie użyte narzędzie Xamarin, które służy do tworzenia aplikacji wieloplatformowych. Zbadane zostaną różne metody połączenia technologii wieloplatformowej z istniejącymi rozwiązaniami OCR. Przedstawiona konwolucyjna sieć neuronowa zaprezentuje klasyfikacje polskiego alfabetu.

Integralną częścią pracy jest aplikacja OCRecognizer, w której zaimplementowano metody klasyfikacji obrazów. Program umożliwia zrobienie zdjęcia, a następnie przy użyciu kilku opcji, rozpoznanie tekstu.

Słowa kluczowe

C#, Xamarin, .NET, Uczenie maszynowe, Sieci neuronowe, kNN, Random Forest,

Spis treści

Wprowadzenie

Xamarin to platforma deweloperska służąca do tworzenia natywnych aplikacji mobilnych dla systemów iOS, Android oraz Windows, za pomocą wspólnej technologii .NET i języka C# . Dzięki temu możliwe jest uzyskanie do stu procent wspólnego kodu między różnymi platformami. Aplikacje napisane przy użyciu technologii Xamarin i C# mają pełny dostęp do interfejsów, API oraz możliwość tworzenia natywnych interfejsów użytkownika.

Ze względu na dynamiczny rozwój rynku IT, uczenie maszynowe staje się coraz bardziej popularne a algorytmy zyskują lepszą skuteczność dzięki dostępności danych oraz szybszych podzespołów komputerowych.

Urządzenia przenośne mają stosunkowo ograniczone zasoby w związku z tym istnieje problem powiązania tych dwóch dziedzin. Algorytmy systemów uczących się wymagają dużej mocy obliczeniowej. Aplikacje wieloplatformowe pozwalają zaoszczędzić czas na implementacji oraz skuteczniej tworzyć funkcjonalności rozpoznawania tekstu. Połączenie tej technologi z algorytmem służącym do klasyfikacji znaków w obrazie jest bardziej optymalne niż ich natywne odpowiedniki.

Celem pracy jest zbadanie istniejących rozwiązań służących do rozpoznawania tekstu oraz stworzenie sieci neuronowej pozwalającej na klasyfikację znaków pisanych charakterystycznych dla współczesnego języka polskiego. Ponieważ pozyskanie danych z polskimi znakami potrzebnych do trenowania sieci neuronowej stanowi problem, zostało stworzone narzędzie do odczytywania znaków z kartki papieru, a następnie zapisanie ich w formie obrazu 32x32 piksele, w skali szarości.

ROZDZIAŁ 1

Rozpoznawanie tekstu w aplikacjach wieloplatformowych

OCR (ang. Optical Character Recognition) jest to technika lub część oprogramowania służąca do rozpoznawania znaków oraz całych tekstów w pliku graficznym prezentowanym za pomocą pionowo-poziomej siatki odpowiednio kolorowanych pikseli. Przykładem takiej grafiki jest zdjęcie z aparatu cyfrowego.

Niegdyś pojęcie rozpoznawania znaków oznaczało samą klasyfikacje ciągów znaków drukowanych, które są łatwiejszym problemem do rozwiązania, dziś również pisma odręczne oraz cechy formatowania, takie jak krój pisma, stopień pisma lub układy tabelaryczne (formularze).

Techniki OCR są głównie wykorzystywane do cyfryzacji zasobów bibliotek, a także jako ułatwienie przy odczytywaniu dokumentacji napisanych pismem odręcznym, w aplikacjach mobilnych rozpoznawanie znaków pomaga w takich zadaniach jak tworzenie notatek, a następnie tłumaczenie ich na tekst drukowany. Niestety, w obu przypadkach istniejące rozwiązania OCR nie są tak skuteczne jak człowiek, zatem w przypadkach trudności z klasyfikacją znaku lub fragmentu tekstu niezbędna jest weryfikacja wyniku przez człowieka celem uniknięcia błędu.

Postęp w metodach OCR jest bardzo widoczny gdyż w obecnych czasach produkty potrafią rozpoznawać mało dokładne skany, wykonane telefonami komórkowymi z szumami na obrazkach, z tekstem napisanym pod nienaturalnymi kątami w wielu językach, pozostaje jednak problem rozpoznawania znaków pisma odręcznego.

Rozpoznawanie pisma jest możliwe dzięki zastosowaniu metod z dziedziny rozpoznawania wzorców, czyli pola badawczego w obrębie uczenia maszynowego. Metoda ta może być definiowana jako działanie polegające na pobieraniu danych i podejmowaniu dalszych czynności zależnych od kategorii do której należą te dane. By odpowiednio wyodrębnić poszczególne znaki z obrazu używane są biblioteki

pozwalające na profesjonalną obróbkę zdjęć pod zastosowania w celach uczenia maszynowego. Przykładem takiej biblioteki jest OpenCV. Następnie po wyodrębnieniu potrzebnych informacji na temat danego znaku obrazy są klasyfikowane jako poszczególne litery. Zwykle w tym procesie używane są sieci neuronowe.

Kompletny system rozpoznawania wzorców składa się z:

- zbioru danych, które oferują możliwość klasyfikacji lub opisu
- mechanizmu wydobywania cech, które najlepiej charakteryzują i separują daną klase, do której dany element zbioru danych należy
- mechanizmu przekształcenia elementu zbioru w symboliczną informacje, łatwiejszą do wykorzystania przez algorytm
- schematu decyzyjnego lub schematu opisu, który realizuje właściwą część procesu klasyfikacji w oparciu o wydobyte i przekształcone cechy obiektu.

1.1. Przedstawienie problemu

Wsród istniejących rozwiązań mogących służyć jako narzędzie potrzebne do wytworzenia aplikacji mobilnej, która rozpozna polskie znaki pisma odręcznego nie istnieje łatwy sposób zastosowania rozwiązania pozwalającego na skuteczną klasyfikacje polskiego pisma. Brakuje również dostępnych danych wymaganych do skutecznej klasyfikacji w oparciu o przekształcone informacje. Aby rozwiązać ten problem należy stworzyć zbiór treningowy lub rozszerzenie istniejącego zbioru danych o polskie znaki alfabetu.

Dostępne biblioteki na rynku, takie jak TesseractAPI oraz Microsoft Computer Vision API oferują wysoką skuteczność w rozpoznawaniu polskich oraz angielskich obrazów tekstu drukowanego lecz zarazem brak możliwości rozpoznawania pisma odręcznego. Wymagane jest więc stworzenie systemu rozpoznawania wzorców, który pozwalałby na skuteczną klasyfikacje znaków pisma odręcznego.

Kolejnym problemem są znacząco ograniczone zasoby urządzeń mobilnych. Systemy rozpoznawania wzorców wymagają mocy obliczeniowej potrzebnej do przekształcenia obrazów w postać pozwalającą na wyodrębnianie cech, a następnie

przeprowadzenie procesu klasyfikacji. Rozwiązaniem tego problemu jest wykorzystanie systemu rozpoznawania wzorców jako serwisu internetowego działającego w oparciu o architekturę REST.

1.2. Sposób wytworzenia zbioru treningowego

Zbiór treningowy jest kontenerem krotek (przykładów, obserwacji, próbek), będących lista właściwości atrybutów opisowych (tzw. deskryptorów) i wybranego atrybutu decyzyjnego (ang. class label attribute). Głównym jego celem jest zbudowanie formalnego modelu zwanego klasyfikatorem. Wynikiem procesu klasyfikacji jest pewien otrzymany model (klasyfikator), który przydziela każdemu przykładowi wartość atrybutu decyzyjnego w oparciu o właściwości pozostałych atrybutów.

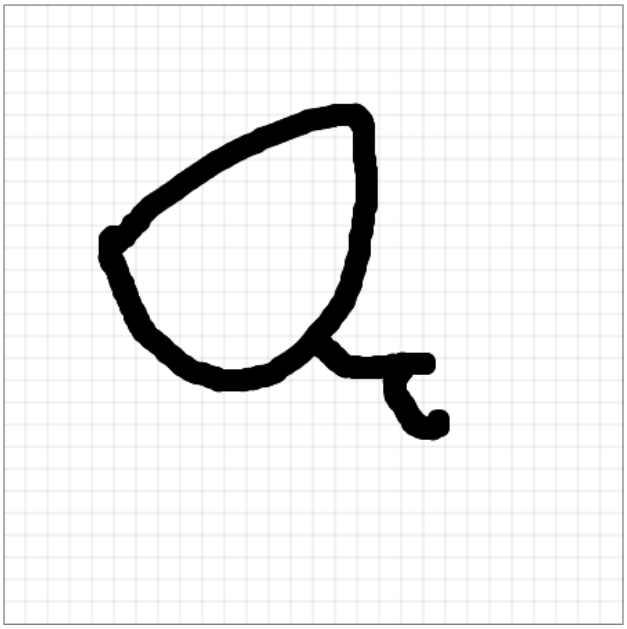
W przypadku systemu rozpoznawania wzorów zbiorem treningowym są zdjęcia obrazów zawierajace odpowiednio wszystkie litery polskiego alfabetu oraz cyfry. Wszystkie zdjęcia liter, które istnieją w zbiorze należy przeformatować do postaci najlepiej rozumianą przez wykorzystywane algorytmy.

Do transformacji zdjęć zastosowano EmguCV, jest to wieloplatformowa implementacja (ang. wrapper) w technologi .NET biblioteki OpenCV, pozwalająca na wykorzystanie funkcjonalności OpenCV w środowisku .NET we wszystkich jego językach programowania takich jak C#, VB, F#. Można ją zainstalować używając menadżera pakietów Nuget w programie Visual Sutdio, Xamarin Studio lub Unity, a więc jest również kompatybilna z platformami mobilnymi Android oraz iOS.

Transformacja zdjęcia przebiega następująco:

- Odczytaj zdjęcie w formacie .png
- Przeprowadź konwersje kolorów RGB na odcienie szarości
- Przetwórz obraz do formatu 28 x 28 pikseli
- Odczytaj stopień jasności każdego piksela w skali od 0 do 255 i zapisz je w tablicy

Rezultatem działania programu do konwersji zdjęć jest plik train.csv. Zawiera ona 785 kolumn. Pierwsza kolumna, nazwana ”label”, określa znak, który jest narysowany. Reszta kolumn zawiera informacje na temat jasności każdego piksela.



Rysunek 1.1. Przykład zdjęcia znaku

Każda kolumna w zbiorze treningowym ma ustawioną nazwę pixelx, gdzie x jest liczbą między 0 a 783. By znaleźć dany piksel na obrazie, należy rozłożyć x jako x = a * 28 + b, gdzie a i b to liczby między 0 a 27. Wtedy pixelx jest umieszczony w a-tym rzędzie b-tej kolumnie w macierzy 28 x 28, indeksowanej od zera. Na przykład, pixel31 wskazuje na to, piksel w czwartej kolumnie od lewej i drugim wierszu od góry. Tak jak pokazane na diagramie poniżej:

```
000  001  002  003  ...  026  027
028  029  030  031  ...  054  055
 |    |    |    |   ...   |    |
728  729  730  731  ...  754  755
756  757  758  759  ...  782  783
```

Aplikacją generującą zbiór treningowy jest program TrainingSetGenerator, kod przeprowadzający transformacje załączony jest poniżej:

```
using System;
using System.Collections.Generic;
```

```
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Emgu.CV;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;

namespace ImageResizer
{
    class Program
    {
        static void Main(string[] args)
        {
            const string imagesPath = @"IMAGEPATH";
            const string csvFilePath = @"CSVFILEPATH";
            var dirInfo = new DirectoryInfo(imagesPath)
            .GetDirectories("*", SearchOption.AllDirectories);

            var csv3 = new StringBuilder();
            csv3.Append("label");
            csv3.Append(',');
            for (int i = 0; i < 784; i++)
            {
                csv3.Append("pixel" + i);
                csv3.Append(',');
            }

            File.AppendAllText(csvFilePath, csv3 + "\n");
```

```
foreach (var directoryInfo in dirInfo)
{
    var files = directoryInfo
    .GetFiles("*", SearchOption.AllDirectories);
    Console.WriteLine("In folder : "
    + directoryInfo.FullName);
    for (int i = 0; i < 1920; i++)
    {
        var csv = new StringBuilder();
        var csv2 = new StringBuilder();
        var fileName = files[i]
        .DirectoryName
        .Replace(imagesPath + @"\", string.Empty);
        csv.Append(fileName);
        csv.Append(',');
        var originalImage =
        new Image<Gray, byte>(files[i].FullName)
        .Not();
        var img = originalImage
        .Resize(28, 28, Inter.Linear);
        for (var k = 0; k < img.Height; k++)
        {
            for (var j = 0; j < img.Width; j++)
            {
                csv.Append(img[k, j].Intensity);
                csv.Append(',');
            }
        }

        csv2.AppendLine(csv.ToString());
        File.AppendAllText(csvFilePath, csv2
        .ToString());
```

```
                }
            }

            Console.WriteLine("DONE!");
            Console.ReadLine();

        }
    }

    public class CsvRow : List<string>
    {
        public string LineText { get; set; }
    }

    public class CsvFileWriter : StreamWriter
    {
        public CsvFileWriter(Stream stream)
            : base(stream)
        {
        }

        public CsvFileWriter(string filename)
            : base(filename)
        {
        }

        public void WriteRow(CsvRow row)
        {
            StringBuilder builder = new StringBuilder();
            bool firstColumn = true;
            foreach (string value in row)
            {
```

```
                if (!firstColumn)
                    builder.Append(',');
                if (value.IndexOfAny(new char[] { '"', ',' }) !=
                    builder.AppendFormat("\"{0}\"", value.Replace(
                else
                    builder.Append(value);
                firstColumn = false;
            }
            row.LineText = builder.ToString();
            WriteLine(row.LineText);
        }
    }
}
```

1.3. Algorytm k-NN

Algorytm k-najbliższych sąsiadów (ang. k nearest neighbours) - algorytm regresji nieparametrycznej najczęściej używany w statystyce do prognozowania pewnej wartości zmiennej losowe.

Założenia:

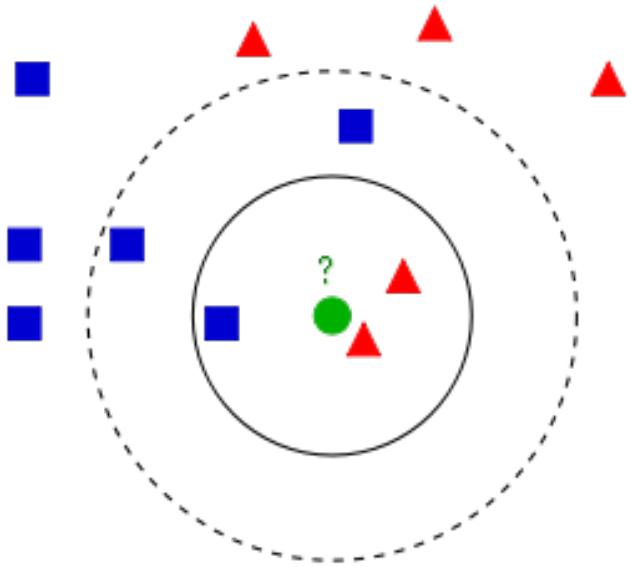
- Dany jest zbiór teningowy, który stworzony został w oparciu o narzędzie TraningSetGenerator.
- Dana jest obserwacja C, zawierająca wektor zmiennych pixelo ... pixel783, dla której chcemy prognozować wartość zmiennej objaśnianej label.

Ilustracja przedstawiająca przykład działania algorytmu k najbliższych sąsiadów:

Algorytm działa następująco:

- Porównaj wartości zmiennych objaśniających dla obserwacji C, z każdym wektorem w zbiorze treningowy.

- Wyborze k (ustalonej z góry liczby) najbliższych do C obserwacji ze zbioru treningowego.
- Uśrednieniu wartości zmiennej objaśnianej dla wybranych obserwacji, w wyniku czego uzyskujemy prognozę.



Rysunek 1.2. Przykład problemu k-NN

Dla k = 3, niewiadoma oznaczona zielonym punktem będzie sklasyfikowana jako czerwony trójkąt w oparciu o trzech najbliższych sąsiadów, jednak jeśli k = 5, zostałaby sklasyfikowana jako niebieski kwadrat ponieważ algorytm działałby w oparciu o pięciu sąsiadów. Najbliżsi sąsiedzi są określani przy pomocy metryki euklidesowej określonej wzorem:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_i - q_i)^2 + ... + (p_n - q_n)^2}.$$

```
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import random


def plot_digit(pixels, label):
    img = pixels.reshape((28,28))
    plt.imshow(img,cmap='gray')
    plt.title(label)
    plt.show()

labeled_images = pd.read_csv('output.csv')
images = labeled_images.iloc[0:5000,1:]
labels = labeled_images.iloc[0:5000,:1]
train_images, test_images, train_labels, test_labels = train_test_

knn = KNeighborsClassifier(n_neighbors=10, algorithm="kd_tree")
knn.fit(train_images, train_labels.values.ravel())
print knn.score(test_images,test_labels)


def test_prediction(index):
    predic = knn.predict(test_images.iloc[index:index+1])[0]
```

```
    actual = test_labels.iloc[index]['label']
    return (predic, actual)

index = random.randint(0, len(test_images)-1)
predic, actual = test_prediction(index)

pixels = test_images.iloc[index].as_matrix()
label = "Predicted: {0}, Actual: {1}".format(predic, actual)

plot_digit(pixels, label)
```

1.4. Random Forest

Algorytm Random Forest to metoda klasyfikacji polegająca na tworzeniu wielu drzew decyzyjnych na podstawie zestawu danych. Idea tego klasyfikatora polega na zbudowaniu zgromadzeniu najlepszych z losowych drzew decyzyjnych, w klasycznych drzewach decyzji, losowe drzewa budowane są na zasadzie podzbiorów analizowanych cech w węźle, które dobierane są losowo.

Cechy algorytmu Random Forest:

- działa skutecznie na dużych zbiorach treningowych
- utrzymuje dokładność w przypadku gdy dane są nie kompletne lub jest ich mało
- daje oszacowanie, które zmienne są istotne w klasyfikacji
- lasy drzew mogą być zapisane i wykorzystane w przyszłości dla innego zbioru danych
- nie jest podany na przeuczenie (ang. overfitting)

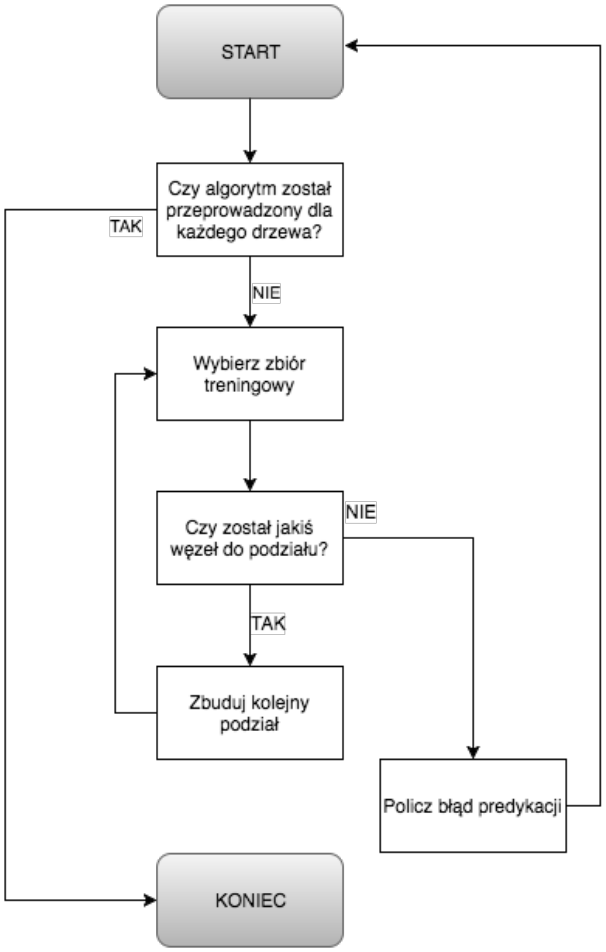
Algorytm działa następująco:

- Losujemy ze zwracaniem z n-elementowego zbioru treningowego n wektorów. Na podstawie takiej próby zostanie stworzone drzewo.

- W każdym węźle podział odbywa się poprzez wylosowanie bez zwracania m spośród p atrybutów, następnie w kolejnym węźle k spośród m atrybutów
- Proces budowania drzewa bez przycinania trwa, jeśli to możliwe do momentu uzyskania w liściach elementów z tylko jednej klasy.

Proces klasyfikacji:

- Dany wektor obserwacji jest klasyfikowany przez wszystkie drzewa, ostatecznie zaklasyfikowany do klasy, w której wystąpił najczęściej.
- W przypadku elementów niewylosowanych z oryginalnej podpróby, każdy taki i-ty element zostaje poddany klasyfikacji przez drzewa, w których budowie nie brał udziału. Taki element zostaje następnie przyporządkowany klasie, która osiągana była najczęściej.



Rysunek 1.3. Diagram przepływu algorytmu Random Forest

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import random


def plot_digit(pixels, label):
    img = pixels.reshape((28,28))
```

```
    plt.imshow(img,cmap='gray')
    plt.title(label)
    plt.show()

labeled_images = pd.read_csv('output.csv')
images = labeled_images.iloc[0:5000,1:]
labels = labeled_images.iloc[0:5000,:1]
train_images, test_images, train_labels, test_labels = train_test_

clf = RandomForestClassifier(n_jobs=2)
clf.fit(train_images, train_labels.values.ravel())
print clf.score(test_images,test_labels)


def test_prediction(index):
    predic = clf.predict(test_images.iloc[index:index+1])[0]
    actual = test_labels.iloc[index]['label']
    return (predic, actual)

index = random.randint(0, len(test_images)-1)
predic, actual = test_prediction(index)

pixels = test_images.iloc[index].as_matrix()
label = "Predicted: {0}, Actual: {1}".format(predic, actual)

plot_digit(pixels, label)
```

1.5. Wielowarstwowe sieci neuronowe

Siecią neuronową nazywa się programową lub sprzętową strukturę modeli, realizującą obliczenia lub przetwarzającą sygnały poprzez rzędy elementów, zwanych sztucznymi neuronami. Emulują one niektóre spośród zaobserwowanych właści-

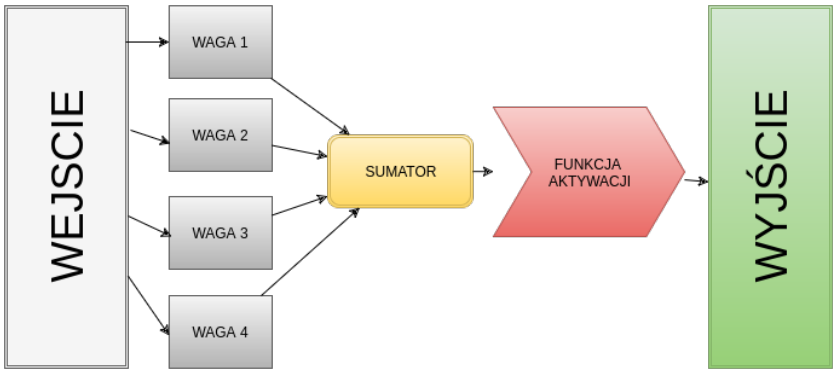
wości biologicznych układów nerwowych. Sztuczne sieci neuronowe są swoistym systemem inspirowanym przez to, w jaki sposób gęsto połączone między sobą struktury mózgu, odbierają i przetwarzają dane które docierają w różny sposób z otoczenia. Kluczowym elementem jest zatem struktura systemu przetwarzania informacji. Sieć taka składa się z dużej liczby rozlegle połączonych ze sobą elementów przetwarzających, które są powiązane ze sobą ważonymi połączeniami.

Cechą charakterystyczna sieci neuronowych od algorytmów realizujących przetwarzanie informacji przy użyciu algorytmów jest umiejetność generalizacji, czyli zdolność uogólniania wiedzy dla nieznanych wcześniej wzorców. Innym atutem jest także zdolność do aproksymacji wartości funkcji wielu zmiennych w przeciwieństwie do interpolacji, która jest możliwa do uzyskania używając przetwarzania algorytmicznego.

Uczenie sieci neuronowych zmienia liczbowe wartości wag znajdujących się pomiędzy neuronami. Następuje to poprzez bezpośrednią ekspozycje rzeczywistego zestawu danych, gdzie algorytm uczący modeluje wagi polaczeń. Ze względu na opisane powyżej cechy i zalety, obszar zastosowań sieci neuronowych jest rozległy:

- Rozpoznawanie wzorców
- Klasyfikowanie obiektów
- Prognozowanie i ocena ryzyka ekonomicznego
- Prognozowanie zmian cen rynkowych
- Ocena zdolności kredytowej
- Ocena wniosków ubezpieczeniowych
- Rozpoznawanie wzorów podpisów
- Diagnostyka medyczna
- Prognozowanie sprzedaży
- Analizowanie zachowań klienta w supermarketach

Podstawowym elementem sieci neuronowej jest neuron. Jego schemat został opracowany przez McCullocha i Pittsa w roku 1943, został on oparty na budowie biologicznej komórki nerwowej.



Rysunek 1.4. Schemat sztucznego neuronu

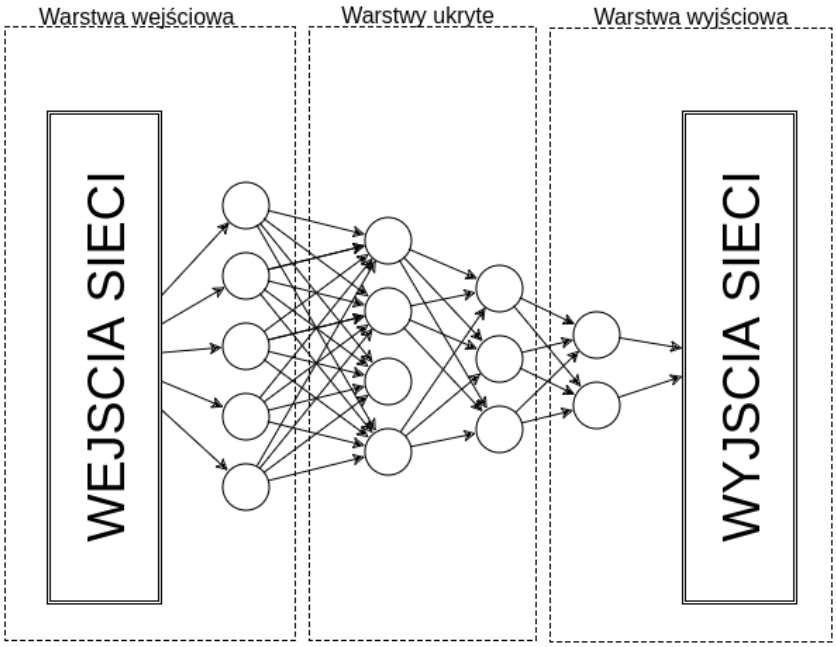
Do wejść doprowadzane są sygnały z wejść sieci lub neuronów warstwy poprzedniej. Każdy sygnał mnożony jest przez odpowiadającą mu wartość liczbowa zwana waga. Wpływa ona na percepcje danego sygnału wejściowego i jego udział w sygnale wyjściowym przez neuron. Waga może być dodania lub ujemna, jeżeli nie ma polaczenia miedzy neuronami to waga jest równa zero. Zsumowane iloczyny wag i sygnałów są argumentem funkcji zwanej funkcja aktywacji neuronu.

Wartość funkcji aktywacji jest wyjściem neuronu i propagowana jest do neuronów warstwy następnej. Może ona przybierać jedną z trzech postaci:

- nieliniowa
- liniowa
- skoku jednostkowego

Należy zauważyć, iż jest to podział bardziej formalny niż merytoryczny. Różnice funkcjonalne między tymi typami raczej nie występują, natomiast można stosować je naprzemiennie w różnych warstwach sieci.

Najbardziej popularnym typem sieci neuronowej jest sieć wielowarstwowa (ang. Multi-Layer Neural Network). Jej cecha charakterystyczna jest występowanie co najmniej jednej warstwy ukrytej neuronów, pośredniczącej w przekazywaniu sygnałów pomiędzy wejściami a wyjściami sieci.



Rysunek 1.5. Schemat budowy sieci wielowarstwowej

Do rozpoznania polskich znaków pisma odręcznego użyta została sieć posiadająca trzy warstwy.

- Warstwa wejściowa sieci składa się z neuronów zawierających informacje na temat każdego piksela. Zbiór treningowy składa się z obrazów 28 x 28 pikseli. Zgodnie z tym założeniem pierwsza warstwa sieci składa się z 784 neuronów. Każdy z nich przechowuje wartość skali szarości piksela, gdzie 0.0 oznacza kolor biały, a 1.0 czarny.

- Druga warstwa zawiera n neuronów, liczba n jest używana w kontekscie eksperymentalnym.

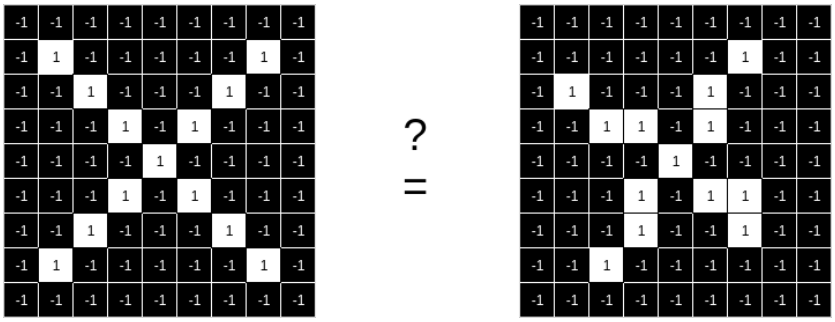
- Ostatnia warstwa, zawiera 74 neurony, ponieważ w Polski alfabet składa się z 32 liter, rozpatrywane są zarówno litery wielkie jak i małe oraz cyfry. Implementacja sieci:

```
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt

train = pd.read_csv('output.csv')
y = np.array(train.pop('label'))
x = np.array(train)/255.
plt.imshow(x[10].reshape(28,28), cmap='Greys', interpolation='nearest')


def plot_digit(pixels, label):
    img = pixels.reshape((28,28))
    plt.imshow(img,cmap='gray')
    plt.title(label)
    plt.show()

split = 50000
x0 = x[:split]; x1 = x[split:]
y0 = y[:split]; y1 = y[split:]
mlp = MLPClassifier(solver='sgd', activation='relu',
                    hidden_layer_sizes=(100,30),
                    learning_rate_init=0.3, learning_rate='adaptive', al
                    momentum=0.9, nesterovs_momentum=True,
                    tol=1e-4, max_iter=200,
                    shuffle=True, batch_size=300,
                    early_stopping = False, validation_fraction = 0.15,
                    verbose=True)
mlp.fit(x0,y0)
y_val = mlp.predict(x1)
accuracy = np.mean(y1 == y_val)
print accuracy
```

```
label = "Predicted: {0}, Actual: {1}".format(y1[100], y_val[100])

plot_digit(x[100], label)
```

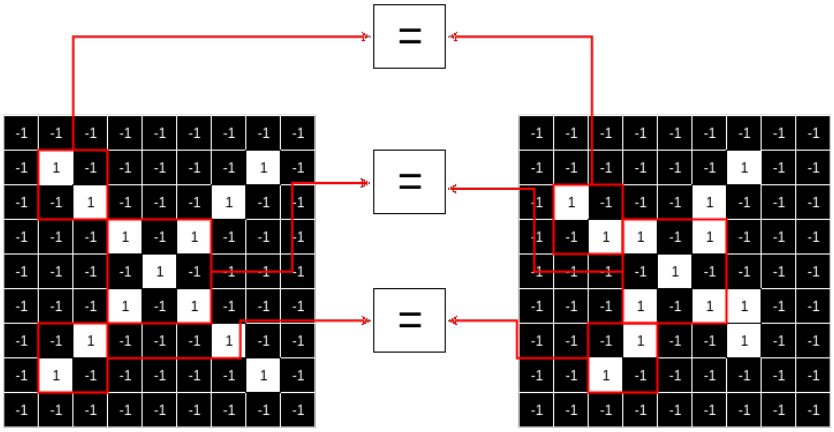
1.6. Konwolucyjne sieci neuronowe - CNN

Konwolucyjne sieci neuronowe (ang. Convolutional Neural Networks) są podobne do klasycznych sieci neuronowych. Aby dokładnie przeanalizować budowę oraz działanie CNN przedstawiony zostanie problem klasyfikacji dwóch liter X i O. Ten przykład demonstruje charakterystyczne reguły konwolucji.



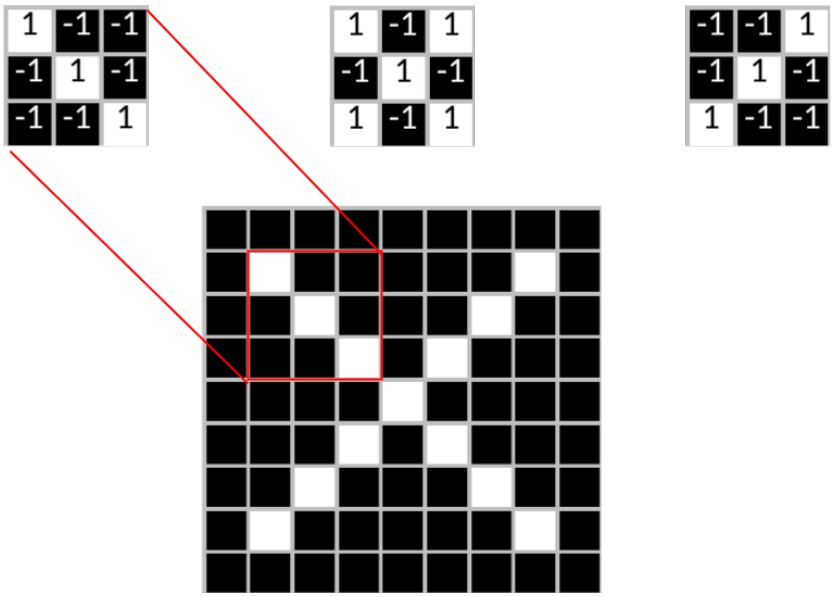
Rysunek 1.6. Problem klasyfikacji

CNN porównuje obrazy w kawałkach. Każda część nazywana jest cecha (ang. feature). Następnie zdjęcia przeszukiwane są na podobnych pozycjach by uzyskać maksymalną liczbę cech wspólnych. Sieci konwolucyjne wspolpracuja na podobieństwach niż na obrazie o pełnym rozmiarze.



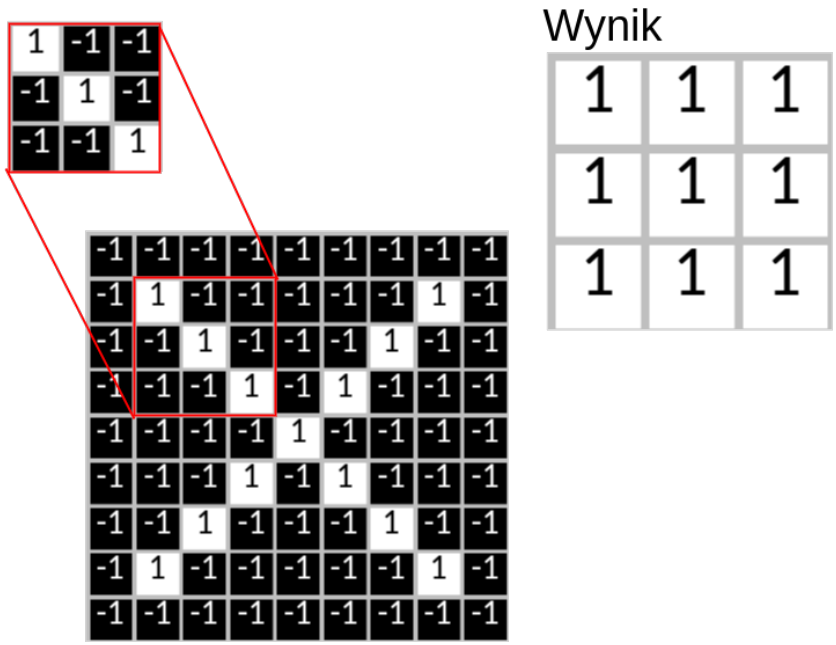
Rysunek 1.7. Wybrane cechy i ich odpowiedniki w zdjęciu do klasyfikacji

Każdą cechę można scharakteryzować jako mniejsze zdjęcie lub dwuwymiarową tablice wartości. W przypadku litery X, cechami beda ukośne linie i znak krzyża.



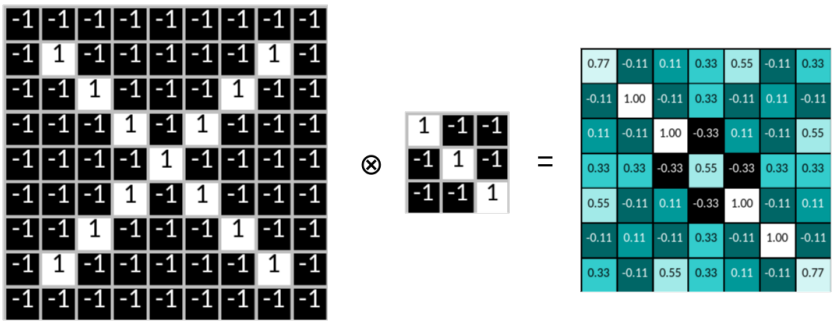
Rysunek 1.8. Wartości liczbowe pikseli różnych cech

Kiedy rozpatrywany jest nowy obraz, sieć poszukuje cech zdjęcia w każdej możliwej pozycji. Obliczając wartości pasujące do cech tworzymy filtr. Działania matematyczne kryjące się za tym działaniem nazywane są splotem całkowitym. Aby obliczyć parę cech obrazu, należy pomnożyć każdą wartość piksela cechy przez odpowiadający mu piksel danego zdjęcia. Następnie należy dodać wszystkie wyniki z poprzednich operacji i podzielić przez łączną liczbę pikseli cechy. Jeśli oba piksele były białe, a ich wartość była reprezentowana przez 1, wynik wynosi 1 - piksel jest biały, w przeciwnym wypadku piksel jest czarny a jego wartość jest równa -1. Jeżeli wszystkie wartości cechy są takie same, wynikiem dodawania i podzielenia przez liczbę pikseli będzie 1.

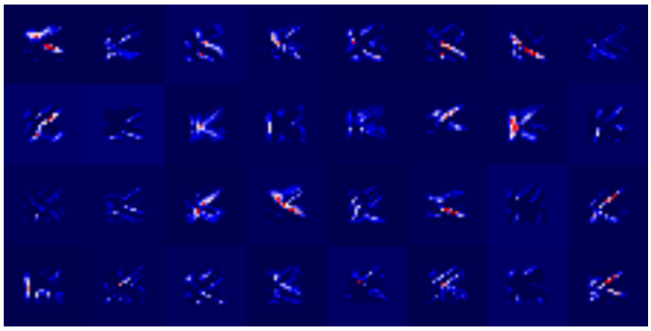


Rysunek 1.9. Wynik przykładowej operacji splotu

Następnym krokiem jest powtórzenie operacji splotu na kompletnym obrazie używając wszystkich dostępnych cech. Wynikiem jest szereg obrazów z filtrami, na każdym z dostępnych wyników nałożony jest jeden filtr. W konwulsyjnych sieciach neuronowych, warstwa ta nazywana jest warstwą konwolucyjną bądź też splotową (ang. convolution layer).

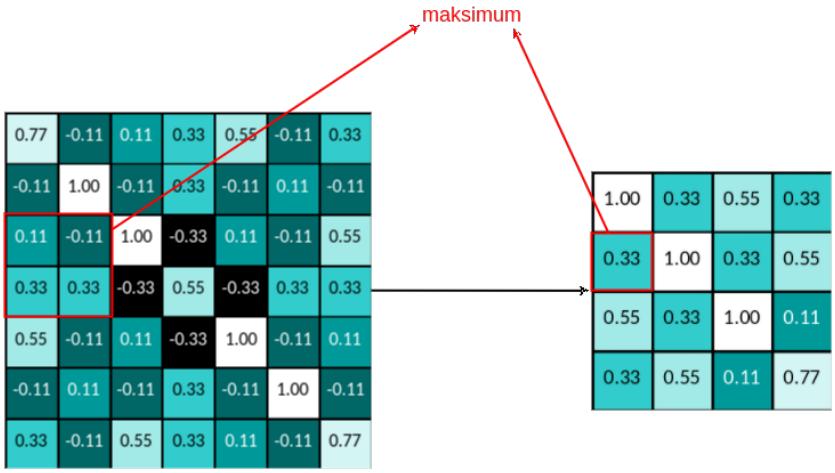


Rysunek 1.10. Wynik operacji splotu danej cechy



Rysunek 1.11. Przykład warstwy konwolucyjnej dla litery k

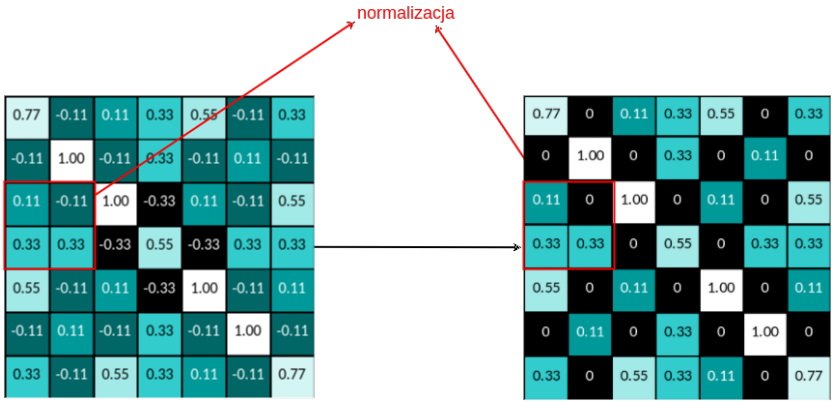
Kolejnym narzędziem oferowanym przez CNN jest warstwa sumowania (ang. pooling layer). Pooling daje możliwość zmniejszenia dużych obrazów bez utraty ważnych informacji na ich temat. Operacje wykonywane w tej warstwie to podział zdjęcia z poprzedniej warstwy na mniejsze, a następnie pobranie maksimum z danej części. Operacje te należy powtórzyć na całym zdjęciu.



Rysunek 1.12. Przykład sumowania

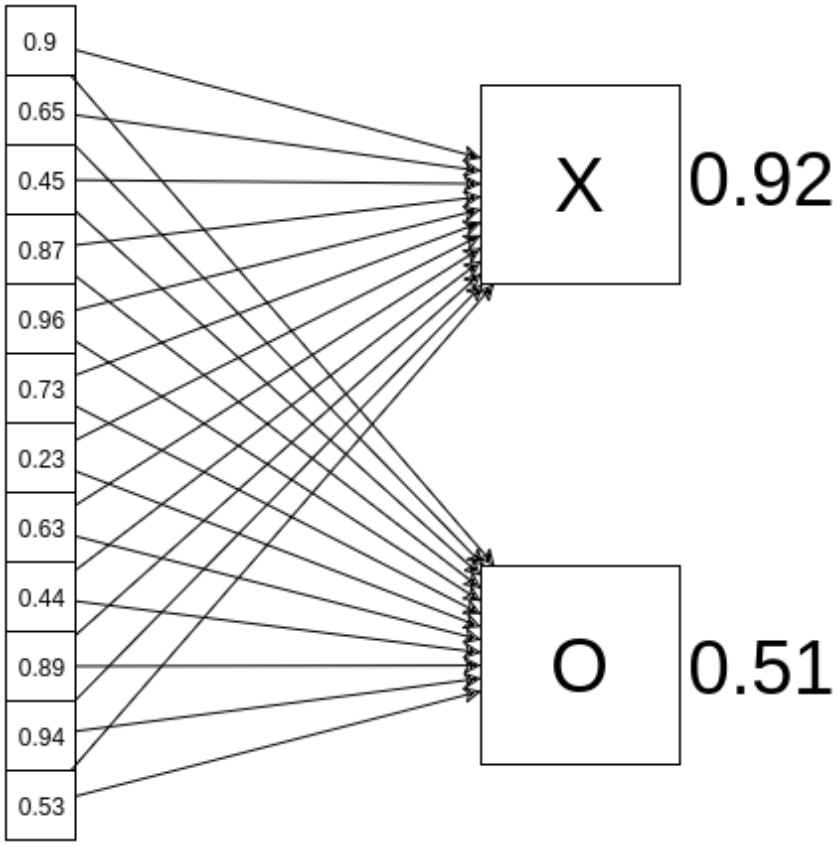
Po sumowaniu, wielkość danego zdjęcia jest cztery razy mniejsza. Ponieważ przetrzymywane są wartości maksymalne z każdego okna, przechowywane zostają najlepsze pasujące wartości danej cechy w oknie. To oznacza, że nie ważne jest gdzie dana cecha pasuje do momentu, aż znajdziemy pasującą odpowiedź w oknie.

Dodatkowym narzędziem jest warstwa normalizacyjna (ang. Rectified Linear Units Layer, ReLU), to ważny proces w którym wszystkie ujemne wartości zostają zastąpione zerami. Wspomaga to funkcjonowaniu sieci trzymając wszystkie wartości na których wykonywane są operacje by te były równe o lub dodatnie.



Rysunek 1.13. Przykład normalizacji danych

Ostatnim narzędziem dostępnym do wykorzystania jest warstwa FCL (ang. Fully Connected Layer). Przyjmuje ona przefiltrowane obrazy, a następnie tłumaczy je na głosy. W przypadku klasyfikacji znaków pisma odręcznego należy zdecydować do której kategorii należy znak X czy O. Kiedy nowy obraz zostaje przekazany do klasyfikacji, wykonywane są operacje na wszystkich wymienionych warstwach, a ostatecznie FCL decyduje do której grupy należy znak.



Rysunek 1.14. Przykład klasyfikacji znaku X

1.7. Podsumowanie

Używając zbioru treningowego stworzonego za pomocą narzędzi z sekcji 1.2 i stworzenia klasyfikatorów problemu rozpoznawania znaków polskiego pisma odręcznego opisanego w sekcji 1.1 przeprowadzone zostały badania wykorzystujące algorytmy wymienione w sekcjach 1.3, 1.4 oraz sieci neuronowe 1.5, 1.6. uzyskane zostały następujące rezultaty:

Nazwa	Dokładność	Czas
kNN	38%	12.3s
Random Forest	28%	9.0s
Sieć neuronowa	73%	159s
CNN	85%	3969s

Tabela 1.1. Wyniki poszczególnych klasyfikatorów

Według wyników badań, najlepszą metodą klasyfikacji znaków polskiego pisma odręcznego jest wykorzystanie konwolucyjnych sieci neuronowych. Implementacja tej metody zostanie użyta w dalszych badaniach.

ROZDZIAŁ 2

Implementacja aplikacji do rozpoznawania tekstu

2.1. Xamarin.Android i Xamarin.iOS

2.2. Xamarin.Forms

2.3. Microsoft Computer Vision API

2.4. Microsoft Azure for Machine Learning

2.5. Tensorflow

ROZDZIAŁ 3

Metryki oraz testy

3.1. Testy wydajnościowe

3.2. Testy zgodności

3.3. Testy użyteczności

3.4. Cross Validation

3.5. Macierze błędu

3.6. Metryki wyliczane z kodu źródłowego

3.7. Macierze wyliczane z diagramów

3.8. Macierze pomiaru wspólnego kodu

ROZDZIAŁ 4

Podsumowanie i wnioski

4.1. Wady oraz zalety aplikacji wieloplatformowych

4.2. Uczenie maszynowe w aplikacjach mobilnych

4.3. Koszt

Zakończenie

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

.....	.....
data	podpis