POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY
INSTYTUT STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

PRACA DYPLOMOWA MAGISTERSKA
na kierunku Informatyka
specjalność: Inżynieria Komputerowa

Paweł NICER
Nr albumu: 197607

Rok akad.: 2008/2009
Warszawa, 30.11.2008

Analiza porównawcza wybranych klasyfikatorów w procesie rozpoznawania drukowanych znaków alfanumerycznych.

Zakres pracy:

1. *Wprowadzenie.*
2. *Omówienie wybranych klasyfikatorów i ich implementacji w OpenCV.*
3. *Przygotowanie zbiorów danych.*
4. *Analiza działania wybranych klasyfikatorów w oparciu o przygotowane zbiory danych.*
5. *Porównanie przetestowanych klasyfikatorów.*
6. *Próba stworzenia własnego klasyfikatora równoległo-kaskadowego.*
7. *Zastosowanie przetestowanych klasyfikatorów w praktyce.*
8. *Wnioski i podsumowanie.*

Kierujący pracą: dr inż. Witold Czajewski

Konsultant:

Termin złożenia pracy: 15.09.2009

Analiza porównawcza wybranych klasyfikatorów w procesie rozpoznawania drukowanych znaków alfanumerycznych.

Streszczenie

Praca poświęcona jest zagadnieniu klasyfikacji w zadaniu rozpoznawania znaków alfanumerycznych. W przeprowadzonych badaniach użyto klasyfikatorów: kNN, normalnego klasyfikatora Bayesa, binarnego drzewa decyzyjnego, lasu losowego, AdaBoost, MLP oraz SVM. Głównym celem pracy było porównanie powyższych klasyfikatorów i próba wybrania najlepszego spośród nich w rozpatrywanym zadaniu klasyfikacji. W tym celu określono pięć kryteriów oceny klasyfikatorów, a następnie przeprowadzono szereg badań umożliwiających zebranie niezbędnych danych.

Testowanie działania klasyfikatorów nie mogło zostać przeprowadzone bez wcześniejszego predefiniowania klas i zestawów cech opisujących klasyfikowane obiekty. W przypadku znaków alfanumerycznych podział na klasy był intuicyjny i oczywisty. Zaproponowanie odpowiedniego zestawu cech opisujących znaki, który zapewniałby wysoką skuteczność ich klasyfikacji przy niskim nakładzie obliczeniowym było kolejnym celem pracy. Przygotowano piętnaście różnych zbiorów cech a następnie wybrano spośród nich najlepsze zbiory dla każdego z badanych klasyfikatorów. Zbadano również wpływ normalizacji cech na uzyskiwane wyniki. Ponadto sprawdzono jaki efekt wywierała selekcja wyznaczonych cech na skuteczność i czas klasyfikacji. Selekcji dokonywano poprzez stopniowe odrzucanie cech o najmniejszej istotności - wyznaczanej przy pomocy binarnego drzewa decyzyjnego.

W oparciu o przeprowadzone wcześniej testy wszystkich klasyfikatorów i przy wykorzystaniu najefektywniejszych zestawów cech dla każdego z nich stworzone zostały klasyfikatory kaskadowe, równoległe i równoległo-kaskadowe. Klasyfikatory kaskadowe dzieliły proces klasyfikacji znaków na dwa etapy: w pierwszym specjalnie wyszkolony klasyfikator wskazywał czy dany znak jest literą czy też cyfrą, w drugim zaś w zależności od wyniku pierwszego etapu znak klasyfikowany był przy pomocy klasyfikatora rozpoznającego same litery lub same cyfry. Klasyfikatory równoległe składały się z kilku klasyfikatorów dokonujących jednocześnie klasyfikacji danego znaku i określających jego klasę na podstawie wspólnego głosowania. Ostatni typ stworzonych klasyfikatorów był połączeniem dwóch wcześniej opisanych. Proces klasyfikacji składał się z dwóch etapów jednakże na każdym etapie zamiast tylko jednego klasyfikatora decyzję podejmowano na podstawie głosowania

kilku klasyfikatorów. Uzyskane wyniki klasyfikacji zostały zestawione z wynikami otrzymanymi dla podstawowych wersji badanych klasyfikatorów.

Badane klasyfikatory zostały również przetestowane na zestawie 90 tablic rejestracyjnych w celu porównania otrzymanych wyników z wynikami uzyskanymi wcześniej w warunkach laboratoryjnych.

Wszystkie badania wykonane zostały przy wykorzystaniu otwartej biblioteki do przetwarzania obrazu OpenCV. Praca zawiera przykładowe implementacje badanych klasyfikatorów w języku C++, wraz z komentarzem, co może stanowić przydatny materiał dydaktyczny dla wszystkich zainteresowanych wykorzystaniem algorytmów klasyfikacji we własnych programach.

Comparative analysis of selected classifiers in alphanumeric character recognition.

Summary

This master thesis investigates selected classifiers in alphanumeric character recognition. Following classifiers were used: kNN, normal Bayes classifier, binary decision tree, random forest, AdaBoost, MLP and SVM. The main goal was to compare classifiers mentioned above and to indicate the best of them in run tests. For this purpose five evaluation criteria were created and a number of tests was performed in order to collect necessary data.

Predefinition of classes and features sets describing classified objects was a necessary step before the classifiers testing could begin. In case of alphanumerical characters division into classes was obvious. Another goal of this work was to propose a sufficient set of features describing characters which would ensure high classification rate at a low computation effort. Fifteen different sets of features were created and from among them the best set was chosen for each of the classifiers. The impact of features normalization on classification results was measured. Furthermore the effect of feature selection on classification rate and time was checked. Feature selections was done by gradual rejection of features with the lowest importance - which was calculated with binary decision tree.

Creation of cascade, parallel and parallel-cascade classifiers was based on previous classifiers tests. The best features sets were used to test them. Cascade classifier were designed to divide the classification process into two stages: in the first stage specially trained classifier was indicating whether the character was letter or digit, in the second one - depending on the outcome of the first stage – the character was classified with algorithm trained to recognize only letters or digits. Parallel classifier was build from few single classifiers and their classification was a result of voting of each single classifier used. The parallel-cascade classifier were a combination of two previous ones. Obtained results were compared with previous collected data.

The classifiers were also tested on a set of 90 car registration plates in order to compare obtained results with those obtained previously in the laboratory conditions.

All tests were performed using opensource library for computer vision – OpenCV. There are also examples of implementations of classifiers in C++ included, with comments, which may be a useful teaching material for all concerned in using classification algorithms in their own work.

Warszawa, dnia 15.09.2009 roku.

Politechnika Warszawska
Wydział Elektryczny

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa magisterska pt.

Analiza porównawcza wybranych klasyfikatorów w procesie rozpoznawania drukowanych znaków alfanumerycznych.

- została napisana przeze mnie samodzielnie
- nie narusza niczyich praw autorskich
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej.
Jestem świadom, że praca zawiera również rezultaty stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.
Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Imię i Nazwisko dyplomanta: Paweł Nicer
Podpis dyplomanta:

Spis treści

Wstęp

Klasyfikacja jako poznawanie i tworzenie otaczającego nas świata poprzez kojarzenie i wnioskowanie jest procesem myślowym właściwym człowiekowi, niezwykle trudnym do odtworzenia przez maszynę. W naukach informatycznych jest zagadnieniem z dziedziny sztucznej inteligencji – uczenia maszynowego, zadaniem realizowanym przy użyciu algorytmu klasyfikującego – tzw. klasyfikatora. Jego działanie polega na imitowaniu owego procesu. Ponieważ pełne odtworzenie go w takim zakresie, w jakim dokonuje się on w ludzkim mózgu jest niemożliwe, nie istnieje bowiem algorytm zdolny do samodzielnego wyodrębniania cech i tworzenia klas, zadanie klasyfikacji w naukach informatycznych jest ograniczone do – można by rzec – ostatniego jego etapu, czyli przyporządkowania danego obiektu do wybranej klasy. Zarówno cechy jak i klasy są uprzednio definiowane i opisywane przez programistę, sam zaś klasyfikator przygotowywany w trakcie „uczenia" czy też „szkolenia". Klasyfikacja taka jest de facto zautomatyzowaniem klasyfikacji dokonywanej przez człowieka, w tym przypadku rozumianej jako rozpoznawanie, grupowanie i hierarchizacja – i jako taka znajduje szerokie zastosowanie. Rozpoznawanie spamu przez programy do obsługi poczty elektronicznej, identyfikowanie tożsamości na podstawie tęczówki, wykrywanie nowotworów na zdjęciach medycznych, rozpoznawanie trendów na rynkach finansowych, wyszukiwanie wirusów na naszych komputerach czy też rozpoznawanie pisma ręcznego to tylko niektóre zagadnienia, dające pojęcie o tym, jak w wielu dziedzinach wykorzystywane są algorytmu klasyfikujące.

Tematem niniejszej pracy jest działanie wybranych klasyfikatorów w procesie rozpoznawania drukowanych znaków alfanumerycznych. Pośród przetestowanych algorytmów klasyfikacji znalazły się binarne drzewa decyzyjne, lasy losowe, AdaBoost, k najbliższych sąsiadów, perceptron wielowarstwowy, normalny klasyfikator Bayesa oraz SVM. Analizie poddana została ich skuteczność klasyfikacji, czasy szkolenia i klasyfikacji, liczba możliwych konfiguracji, ilość przestrzeni dyskowej wymaganej do przechowania utworzonego klasyfikatora a także wpływ normalizacji danych na uzyskiwane wyniki. Do obiektów poddanych klasyfikacji należało 26 dużych, drukowanych liter alfabetu łacińskiego oraz 10 drukowanych cyfr arabskich, zapisanych przy użyciu jednych z najpopularniejszych czcionek: Arial, Century, Comic Sans, Courier New, Georgia, Helvetica, Perpatua, Tahoma, Terminal, Times New Roman oraz Verdana. Znaki alfanumeryczne wybrane zostały z powodu swojego intuicyjnego podziału na klasy, a także regularnych, geometrycznych kształtów, które ułatwiły dobór opisujących je cech.

Badania przeprowadzone zostały przy wykorzystaniu otwartej biblioteki przetwarzania obrazów OpenCV w wersji 2.1. Należy ona do najpopularniejszych i najbardziej rozwiniętych

tego rodzaju bibliotek - w jej skład wchodzi ponad 500 zaawansowanych funkcji, a ich liczba cały czas rośnie. Napisana w języku C i C++ może zostać uruchomiona na różnych systemach operacyjnych: Linux, Windows, Mac OS X oraz Android. OpenCV oferuje także rozbudowany moduł Machine Learning – uczenia maszynowego, w którym projektanci położyli duży nacisk na rozpoznawanie wzorców oraz klasteryzację. Ponadto OpenCV jest zintegrowana z bibliotekami ROS (Robot Operating System) i PCL (Point Cloud Library), których celem jest dostarczenie uniwersalnego i wieloplatformowego oprogramowania dla inteligentnych maszyn i systemów, zdolnych do zawansowanej, choć jednak gorszej niż ludzka, percepcji otoczenia.

Zasadniczym celem przeprowadzonych badań było przetestowanie i porównanie najczęściej wykorzystywanych algorytmów klasyfikacji i wskazanie najefektywniejszego spośród nich w wybranym zadaniu. Kolejnym celem była próba stworzenia, w oparciu o otrzymane wyniki, klasyfikatora równoległo-kaskadowego składającego się z wybranych klasyfikatorów oraz zaproponowanie takiego zestawu cech opisujących znaki, który pozwoliłby osiągnąć zadowalającą skuteczność klasyfikacji przy niskim nakładzie obliczeniowym. Dodatkowym celem wykonanej pracy było pokazanie jak należy wykorzystać zaimplementowane w OpenCV metody klasyfikacji we własnym programie, gdyż z powodu bardzo skromnej dokumentacji, w dodatku dostępnej tylko w języku angielskim, nie jest to zadaniem prostym. Należy również zaznaczyć, iż celem autora nie było stworzenie programu OCR, tak więc podczas rozpoznawania znaków nie była przeprowadzana dodatkowa analiza słownikowa i kontekstowa, a rozpoznawane znaki występowały w postaci łatwej do segmentacji (czarny znak na białym tle).

Niniejsza praca składa się ze wstępu, czterech rozdziałów, podsumowania i bibliografii. Pierwszy rozdział zawiera opis działania wszystkich siedmiu wykorzystanych klasyfikatorów oraz ich implementacji w bibliotece OpenCV. W drugim rozdziale omówiony został sposób generowania zbiorów cech użytych do uczenia i testowania klasyfikatorów – zastosowane czcionki, zapis znaków, liczba stworzonych próbek oraz zbiory cech opisujących obiekty i metody ich wyznaczania. Trzeci rozdział przedstawia wyniki badań otrzymanych przy pomocy wybranych klasyfikatorów oraz przykładowe implementacje poszczególnych metod klasyfikacji. W czwartym rozdziale omówiony został sposób realizacji klasyfikatora równoległo-kaskadowego, a także umieszczone zostały pomiary skuteczności klasyfikacji zbioru tablic rejestracyjnych przy użyciu najskuteczniejszych klasyfikatorów.

1 Klasyfikatory i ich implementacja w OpenCV

1.1 Klasyfikator

Zadaniem klasyfikatora jest przyporządkowanie nieznanemu obiektowi, w oparciu o wyznaczony dla niego zbiór cech, jednej z predefiniowanych klas. W celu dokonania poprawnej klasyfikacji, klasyfikator musi zostać wcześniej odpowiednio przygotowany. Pierwszy etap nazywa się szkoleniem lub trenowaniem klasyfikatora i wymaga przygotowania specjalnego zbioru danych – zbioru uczącego. Zbiór ten składa się z wektorów cech klasyfikowanych obiektów i odpowiadających im klas. Na jego podstawie, w procesie szkolenia klasyfikator uczy się przyporządkowywać obiekty posiadające odpowiednie wartości cech do konkretnych klas. Drugim etapem przygotowywania klasyfikatora jest sprawdzenie jego skuteczności klasyfikacji na drugim zbiorze danych – zbiorze testowym. Zbiór testowy ma taką samą konstrukcję jak zbiór uczący, w jego skład wchodzą jednak obiekty, które nie pojawiły się w zbiorze uczącym. Klasyfikator, korzystając ze wcześniej nabytej wiedzy, dokonuje klasyfikacji nieznanych obiektów, a wyniki tej klasyfikacji porównywane są z faktycznymi klasami poszczególnych obiektów. Im więcej obiektów zostało poprawnie sklasyfikowanych, tym większa jest skuteczność klasyfikatora. Jeżeli skuteczność klasyfikacji jest niezadowalająca pojawia się konieczność zmiany parametrów klasyfikatora (jeżeli istnieje taka możliwość) lub zmiany zbioru uczącego - na zbiór składający się z większej liczby obiektów poszczególnych klas lub zbiór zawierający inne cechy [11].

Zbiory uczący i testowy tworzy się przeważnie na podstawie jednego zbioru danych referencyjnych. Często około 70% danych umieszczanych jest w zbiorze uczącym, reszta zaś w zbiorze testowym [11]. Większe i bardziej zróżnicowane zbiory uczące przeważnie pozwalają uzyskać klasyfikatory skuteczniejsze, choć czasami zbyt różnorodny zbiór nie daje się poprawnie sklasyfikować.

Bardzo ważną cechą klasyfikatora jest jego zdolność do generalizacji, czyli dokonywania poprawnej klasyfikacji na obiektach spoza zbioru uczącego. Czasami podczas szkolenia klasyfikator zbytnio dostosowuje się do klasyfikowania obiektów ze zbioru uczącego („ucząc się” także szumu zawartego w danych wejściowych), co powoduje spadek zdolności generalizacji. Zjawisko to nazywane jest przeuczeniem klasyfikatora.

1.2 K najbliższych sąsiadów (k nearest neighbours)

KNN zaliczany jest do grupy algorytmów „lazy learning", czyli algorytmów, dla których konstruowanie opisu funkcji docelowej klasyfikatora odbywa się na etapie klasyfikacji obiektu, a nie podczas szkolenia klasyfikatora. W praktyce oznacza to, iż uczenie klasyfikatora ogranicza się do zapisania w pamięci całego zbioru uczącego [4]. Klasyfikacja odbywa się poprzez porównanie klasyfikowanego obiektu ze wszystkimi zapamiętanymi obiektami ze zbioru uczącego i wybranie spośród nich k najbardziej podobnych obiektów. W celu oszacowania podobieństwa dwóch wektorów cech najczęściej używa się metryki euklidesowej lub ulicznej, wybierając obiekty, dla których odległość ta jest najmniejsza. Klasyfikowanemu obiektowi przypisywana jest klasa reprezentowana przez największą liczbę obiektów spośród k wybranych sąsiadów. W przypadku, gdy kilka klas reprezentowanych jest przez tą samą liczbę sąsiadów, wybierana jest klasa sąsiadów najbliższych klasyfikowanemu obiektowi. Parametr k należy wybrać eksperymentalnie w celu otrzymania jak najlepszej klasyfikacji dla danego zbioru danych. Dla małych wartości k algorytm jest bardziej podatny na szumy. Duże wartości k zwiększają czas wykonywania wymaganych obliczeń [12]. Nie zawsze zwiększanie parametru k daje lepsze wyniki klasyfikacji. Algorytm KNN traktuje wszystkie atrybuty wektora cech, jako jednakowo ważne co może prowadzić do nieskutecznej klasyfikacji w wyniku dominacji mniej znaczących cech nad cechami ważniejszymi. Aby uniknąć takiej sytuacji należy wybrać odpowiedni zbiór cech lub dokonać modyfikacji algorytmu tak, aby każda cecha miała przypisaną wagę opisującą jej ważność [7].

Najprostszą modyfikacją klasyfikatora kNN jest klasyfikator minimalno-odległościowy, który dokonuje klasyfikacji nieznanego obiektu na podstawie klasy znalezionego pierwszego najbliższego sąsiada (k = 1). Metoda ta sprawdza się w przypadku zbioru danych, w których istnieją wyraźne podziały obiektów na klasy, w przeciwnym wypadku klasyfikacja często może okazać się błędna (wówczas algorytm kNN dla k > 1 mógłby już sobie poradzić znacznie lepiej). Zaletą wyszukiwania tylko jednego sąsiada jest skrócenie czasu klasyfikacji nieznanego obiektu. Bardziej złożoną modyfikacją kNN jest metoda k najbliższych prototypów [5]. Polega ona na podzieleniu zbioru uczącego na podzbiory, a następnie wyznaczeniu uśrednionych wektorów cech dla obiektów z każdego z utworzonych podzbiorów. W efekcie tego działania cały zbiór uczący redukuje się do zbioru wektorów cech reprezentujących poszczególne podzbiory. Dzięki takiemu podejściu znacząco zmniejsza się ilość danych, które muszą być zapamiętane na etapie szkolenia klasyfikatora, ilość porównań, które muszą być wykonane na etapie klasyfikacji, a także podatność na szumy występujące w zbiorze danych uczących. Kluczowym zadaniem przy powyższej metodzie jest odpowiednie podzielenie zbioru uczącego na podzbiory tak, aby obiekty do siebie podobne

znalazły się w tych samych podzbiorach. Podział może być dokonany w oparciu o wiedzę posiadaną na temat zbioru uczącego (w przypadku niniejszej pracy może być to np. podział na klasy znaków) lub też przy wykorzystaniu automatycznych metod grupowania (np. k-means).

Klasyfikator kNN w OpenCV zaimplementowany jest w module Machine Learning pod nazwą `CvKNearest`. Możliwość konfiguracji klasyfikatora ograniczona jest jedynie do doboru parametru k. Poniżej znajduje się model jego klasy [20]:

```
class CvKNearest : public CvStatModel{
public:

    CvKNearest();
    virtual ~CvKNearest();

    CvKNearest( const CvMat* _train_data, const CvMat* _responses,
                const CvMat* _sample_idx=0, bool _is_regression=false,
              int max_k=32 );

    virtual bool train( const CvMat* _train_data,
                      const CvMat* _responses,
                         const CvMat* _sample_idx=0,
                      bool is_regression=false,
                         int _max_k=32, bool _update_base=false );

    virtual float find_nearest( const CvMat* _samples, int k,
                      CvMat* results,
                       const float** neighbors=0,
                      CvMat* neighbor_responses=0, CvMat* dist=0 )
      const;

    virtual void clear();
    int get_max_k() const;
    int get_var_count() const;
    int get_sample_count() const;
    bool is_regression() const;

protected:
    ...
};
```

Aby otrzymać działający klasyfikator należy stworzyć obiekt klasy `CvKNearest`. Danymi przekazywanymi do konstruktora są odpowiednio: macierz wektorów cech obiektów ze zbioru uczącego (kolejne wektory cech muszą znajdować się w kolejnych wierszach macierzy), macierz zawierająca klasy kolejnych obiektów ze zbioru uczącego, macierz wskazująca, które z obiektów ze zbioru uczącego mają być rozpatrywane (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zbioru uczącego), wartość `true` (dokonywana jest klasyfikacja), parametr `k`, wartość `false`, jeżeli ma zostać utworzony nowy klasyfikator lub `true`, jeśli wcześniej wyszkolony klasyfikator ma zostać doszkolony na nowym zbiorze uczącym. Konstruktor automatycznie uruchamia metodę `train()`, tak więc klasyfikator jest gotowy do użytku.

Klasyfikacji obiektu dokonuje się przy użyciu metody `find_nearest()`. Danymi przekazywanymi do metody są odpowiednio: macierz wektorów cech obiektów, które mają zostać poddane klasyfikacji, parametr k, pusta macierz wyników przeprowadzonej klasyfikacji (jeżeli klasyfikowany jest więcej niż jeden obiekt), która zostanie wypełniona podczas wykonywania metody, pusta tablica wskaźników do znalezionych podczas klasyfikacji najbliższych sąsiadów, pusta macierz klas znalezionych najbliższych sąsiadów i pusta macierz odległości znalezionych najbliższych sąsiadów od klasyfikowanych obiektów. Jeżeli na raz przeprowadzana jest klasyfikacja tylko jednego obiektu, metoda zwraca oszacowaną klasę obiektu.

Dzięki danym zwróconym w macierzach `neighbors_responses` i `dist`, można zweryfikować wybór klasy dokonany przez klasyfikator i ewentualnie dokonać korekty wyboru. Implementacja algorytmu kNN w OpenCV zawiera błąd i podczas wybierania najlepszej klasy spośród sąsiadów najbliższy sąsiad jest pomijany. W celu zapewnienia poprawności klasyfikacji nie powinno się korzystać z wyników zwróconych przez metodę `find_nearest()`, lecz wyznaczyć klasę obiektu klasyfikowanego na podstawie informacji o znalezionych najbliższych sąsiadach.

1.3 Klasyfikator Bayesa

Klasyfikator Bayesa klasyfikuje nieznane obiekty na podstawie określonego dla nich prawdopodobieństwa przynależenia do poszczególnych klas. Podstawą działania klasyfikatora jest twierdzenie Bayesa dotyczące prawdopodobieństw warunkowych, wyrażone w postaci wzoru [15]:

$$P(\omega_i|C) = \frac{P(\omega_i)P(C|\omega_i)}{\sum_{j=1}^{k} P(\omega_j)P(C|\omega_j)}, \tag{1.3.1}$$

gdzie ω_i jest hipotezą mówiąca o przynależności klasyfikowanego obiektu do *i*-tej klasy, $P(\omega_i)$ jest prawdopodobieństwem a priori hipotezy ω_i, $P(C|\omega_j)$ jest prawdopodobieństwem przynależności klasyfikowanego obiektu do *i*-tej klasy pod warunkiem posiadania przez niego wektora cech C, a k jest liczbą klas obiektów [12]. Ponieważ celem klasyfikatora jest wybranie najbardziej prawdopodobnej hipotezy ω_i poprzez porównanie i wybranie największego $P(\omega_i|C)$ (a nie dokładne wyliczenie poszczególnych wartości $P(\omega_i|C)$) mianownik równania 1.3.1 może zostać pominięty, gdyż jest on taki sam dla wszystkich przypadków, dzięki czemu równanie może zostać uproszczone do postaci [7]:

$$P(\omega_i|C) = P(\omega_i)P(C|\omega_i). \tag{1.3.2}$$

Prawdopodobieństwo $P(\omega_i)$ jest wyznaczane na podstawie analizy zbioru uczącego poprzez obliczenie ilorazu wszystkich obiektów danej klasy przez liczbę wszystkich obiektów w

zbiorze. Prawdopodobieństwo $P(C|\omega_i)$ jest szacowane na podstawie rozkładu wektorów cech poszczególnych klas w zbiorze uczącym. W szczególnym przypadku można dokonać założenia o braku zależności pomiędzy poszczególnymi cechami, dzięki czemu można przyjąć, iż zdarzenia losowe polegające na posiadaniu przez obiekt konkretnych wartości poszczególnych cech są od siebie niezależne. Klasyfikatory przyjmujące powyższe założenie nazywane są naiwnymi klasyfikatorami Bayesa. Ponieważ założenie o braku zależności pomiędzy poszczególnymi cechami jest przeważnie błędne zakłada się model dystrybucji wielowymiarowej, losowej – najczęściej rozkład normalny [12]. W takim przypadku $P(C|\omega_i)$ wynosi:

$$P(C|\omega_i) = \frac{1}{(2\pi)^{n/2}|M_i|^{1/2}} \exp\left(-\frac{1}{2}(C - s_i)^T * M_i^{-1} * (C - s_i)\right), \qquad (1.3.3)$$

gdzie s_i jest średnią wektora cech dla *i*-tej klasy a M_i macierzą kowariancji dla *i*-tej klasy. Stosując serię przekształceń równań 1.3.2 i 1.3.3 otrzymuje się:

$$P(\omega_i|C) = \ln\left(P(\omega_i)\right) - \frac{1}{2}\ln|M_i| - \frac{1}{2}(C - s_i)^T * M_i^{-1} * (C - s_i). \qquad (1.3.4)$$

W procesie uczenia klasyfikator wylicza macierze kowariancji M_i, zaś w procesie klasyfikacji danego obiektu wyznacza dla niego wszystkie $P(\omega_i|C)$ i przypisuje go do klasy o największym prawdopodobieństwie $P(\omega_i|C)$. Klasyfikator Bayesa opierający się na założeniu o normalnym rozkładzie cech nazywa się normalnym klasyfikatorem Bayesa.

Normalny klasyfikator Bayesa w OpenCV zaimplementowany jest w module Machine Learning pod nazwą `CvNormalBayesClassifier`. Klasyfikator nie posiada możliwości konfigurowania go. Poniżej znajduje się model jego klasy [20]:

```
class CvNormalBayesClassifier : public CvStatModel{
public:
    CvNormalBayesClassifier();
    virtual ~CvNormalBayesClassifier();

    CvNormalBayesClassifier( const CvMat* _train_data,
                      const CvMat* _responses,
                      const CvMat* _var_idx=0,
                      const CvMat* _sample_idx=0 );

    virtual bool train( const CvMat* _train_data,
                      const CvMat* _responses,
                      const CvMat* _var_idx = 0,
                      const CvMat* _sample_idx=0, bool update=false );

    virtual float predict( const CvMat* _samples, CvMat* results=0 ) const;
    virtual void clear();

    virtual void save( const char* filename, const char* name=0 );
    virtual void load( const char* filename, const char* name=0 );

    virtual void write( CvFileStorage* storage, const char* name );
    virtual void read( CvFileStorage* storage, CvFileNode* node );
protected:
    ...};
```

Aby otrzymać działający klasyfikator należy stworzyć obiekt klasy `CvNormalBayesClassifier`. Danymi przekazywanymi do konstruktora są odpowiednio: macierz wektorów cech obiektów ze zbioru uczącego (kolejne wektory cech muszą znajdować się w kolejnych wierszach macierzy), macierz zawierająca klasy kolejnych obiektów ze zbioru uczącego, macierz wskazująca, które z cech mają być brane pod uwagę (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zestawu cech) oraz macierz wskazująca, które z obiektów ze zbioru uczącego mają być rozpatrywane (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zbioru uczącego). Konstruktor automatycznie uruchamia metodę `train()`, tak więc klasyfikator jest gotowy do użytku.

Klasyfikacji obiektu dokonuje się przy użyciu metody `predict()`. Danymi przekazywanymi do metody są odpowiednio: macierz wektorów cech obiektów, które mają zostać poddane klasyfikacji oraz pusta macierz wyników przeprowadzonej klasyfikacji (jeżeli klasyfikowany jest więcej niż jeden obiekt), która zostanie wypełniona podczas wykonywania metody. Jeżeli klasyfikowany jest tylko jeden obiekt metoda zwraca wybraną dla niego klasę.

1.4 Binarne drzewo decyzyjne

Binarne drzewo decyzyjne z dużym powodzeniem może być używane jako klasyfikator. Jego implementacja charakteryzuje się stosunkowo niewielką złożonością obliczeniową i pamięciową. Dodatkowymi zaletami wynikającymi z zasady konstruowania drzewa decyzyjnego jest możliwość określenia ważności poszczególnych cech biorących udział w procesie klasyfikacji oraz zobrazowanie całego procesu w sposób czytelny dla człowieka [11]. Binarne drzewo decyzyjne przystosowane jest do klasyfikacji bardzo dużych zbiorów danych, a proces wyznaczania klasy dla nieznanego obiektu, ze względu na konstrukcję drzewa, jest krótki i mało skomplikowany obliczeniowo. Binarne drzewo decyzyjne opisuje w formie skierowanego, acyklicznego grafu zależność wyniku klasyfikacji od wartości poszczególnych cech klasyfikowanego obiektu. Składa się z korzenia, węzłów, gałęzi i liści. Korzeń jest głównym węzłem drzewa. Węzeł jest elementem, w którym wykonywany jest test wartości pojedynczych cech. Z każdego węzła wychodzą dwie gałęzie odpowiadające poszczególnym wynikom testu wykonanego w danym węźle. Gałęzie mogą prowadzić do innych węzłów lub do liści. Liście są zakończeniami drzewa i reprezentują poszczególne klasy [5].

Nauka klasyfikatora polega na zbudowaniu binarnego drzewa decyzyjnego dla dostarczonego zbioru uczącego. Algorytm realizujący to zadanie nazywa się algorytmem

zstępującym. Drzewo tworzone jest od korzenia (pierwszego węzła) poprzez kolejne połączone z nim węzły aż do zwieńczających je liści. W każdym węźle dokonywany jest podział zbioru na dwa podzbiory, tak więc na kolejnych poziomach drzewa analizowana jest coraz mniejsza ilość danych [8]. Kluczowym elementem tworzenia drzewa jest dobranie odpowiedniego testu podziału zbioru w węźle [7]. Jako kryterium oceny efektywności testu przyjmuje się niespójność powstałych przy jego użyciu dwóch nowych węzłów. Im bardziej klasy wektorów przydzielonych do danego węzła są jednorodne, tym mniejsza jest jego niespójność. Najczęściej stosowanymi miarami niespójności są [5]:

- niespójność entropijna:

$$i(P) = -\sum_{i=1}^{k} \frac{|P_i|}{|P|} \log \frac{|P_i|}{|P|}, \tag{1.4.1}$$

- niespójność błędnej klasyfikacji:

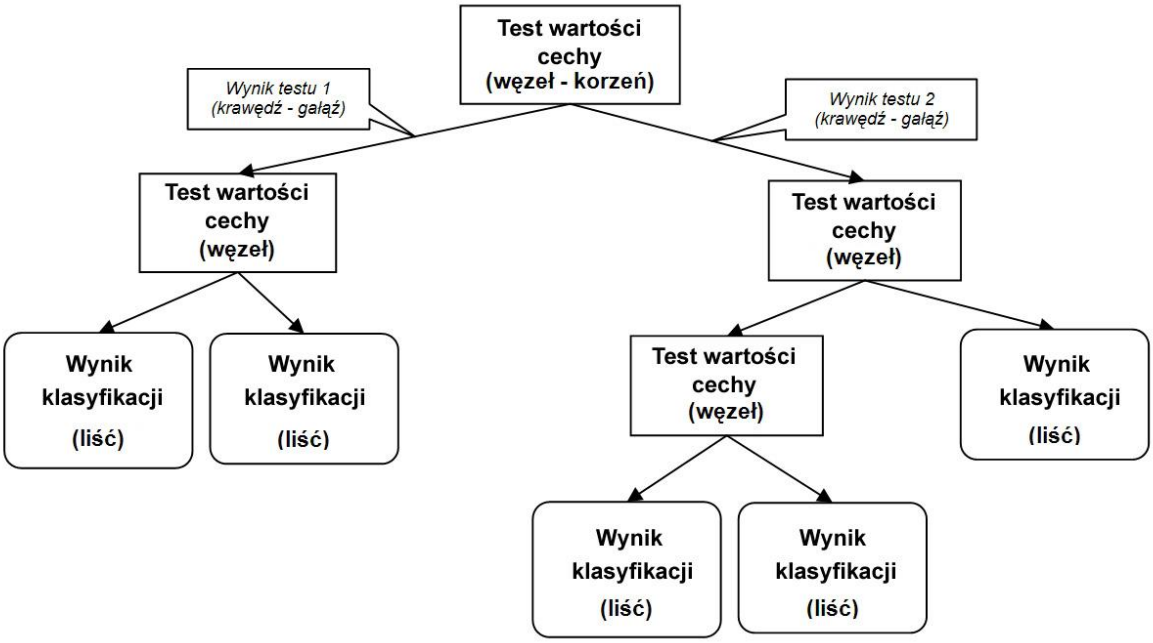
$$i(P) = 1 - \max_{i=1,\dots,k} \frac{|P_i|}{|P|}, \tag{1.4.2}$$

- niespójność Giniego:

$$i(P) = \frac{1}{2}\left(1 - \sum_{i=1}^{k} \left(\frac{|P_i|}{|P|}\right)^2\right), \tag{1.4.3}$$

gdzie P to zbiór uczący danego węzła, $i(P)$ to wartość miary niespójności węzła, P_i to podzbiór P składający się z samych wektorów i-tej klasy, k oznacza liczbę wszystkich klas zaś $|P|$ i $|P_i|$ to liczba elementów w poszczególnych zbiorach.

W momencie, gdy jeden z wyznaczonych podzbiorów składa się z samych obiektów jednej klasy tworzony jest liść i nadawana jest mu odpowiednia etykieta (odpowiadająca klasie wszystkich elementów ze zbioru). Czasami zbiór, który musi zostać podzielony składa się z elementów różnych klas i nie da się wyznaczyć testu, który rozdzieliłby go na dwa mniej zróżnicowane zbiory lub drzewo osiągnęło już zdefiniowaną przez projektanta maksymalną głębokość (liczbę następujących po sobie węzłów, znajdujących się na drodze od korzenia do liścia). W takiej sytuacji dalszy podział staje się niemożliwy i musi zostać utworzony liść. Należy wtedy podjąć decyzję jak sklasyfikować zbiór. Najprostszym, choć nie jedynym, rozwiązaniem jest przypisanie etykiecie kategorii dominującej wśród obiektów w zbiorze [5].



Rysunek 1. Przykład binarnego drzewa decyzyjnego. [5]

Zbudowane w ten sposób binarne drzewo decyzyjne w jak najlepszy sposób klasyfikuje zbiór danych uczących, jest do niego dopasowane, przez co zdolność klasyfikowania nieznanych obiektów może być niewystarczająca. W celu poprawienia skuteczności klasyfikacji nowych obiektów stosuje się mechanizm przycinania drzewa (ang. pruning) [7][11]. Polega on na analizie poszczególnych węzłów, odnajdowania tych najsłabszych i zastępowania ich odpowiednimi liśćmi. Operacja ta może być wykonywana zarówno na etapie budowania drzewa w trakcie podejmowania decyzji o doborze testu w węźle lub już na istniejącym drzewie. W efekcie zastosowania powyższej metody drzewo decyzyjne staje się mniejsze (część gałęzi została usunięta), gorzej klasyfikuje zbiór uczący, lecz jego zdolność do generalizacji wzrasta.

Binarne drzewo decyzyjne w OpenCV zaimplementowane jest w module Machine Learning pod nazwą `CvDTree`. Poniżej znajduje się model klasy [20]:

```
class CV_EXPORTS CvDTree : public CvStatModel{
public:
    CvDTree();
    virtual ~CvDTree();

    virtual bool train( const CvMat* _train_data, int _tflag,
                        const CvMat* _responses, const CvMat*
                        _var_idx=0,const CvMat* _sample_idx=0,
                        const CvMat* _var_type=0,
                        const CvMat* _missing_mask=0,
                        CvDTreeParams params=CvDTreeParams() );

    virtual bool train( CvMLData* _data,
                        CvDTreeParams_params=CvDTreeParams() );
```

```
    virtual float calc_error( CvMLData* _data, int type ,
                        std::vector<float> *resp = 0 );

    virtual bool train( CvDTreeTrainData* _train_data,
                        const CvMat* _subsample_idx );

    virtual CvDTreeNode* predict( const CvMat* _sample,
                        const CvMat* _missing_data_mask=0,
                        bool preprocessed_input=false ) const;

    virtual const CvMat* get_var_importance();
};
```

Konfiguracja klasyfikatora odbywa się poprzez obiekt klasy `CvDTreeParams`:

```
struct CV_EXPORTS CvDTreeParams{
    int   max_categories;
    int   max_depth;
    int   min_sample_count;
    int   cv_folds;
    bool  use_surrogates;
    bool  use_1se_rule;
    bool  truncate_pruned_tree;
    float regression_accuracy;
    const float* priors;

    CvDTreeParams() : max_categories(10), max_depth(INT_MAX),
                      min_sample_count(10), cv_folds(10),
                  use_surrogates(true), use_1se_rule(true),
                  truncate_pruned_tree(true), regression_accuracy(0.01f),
                  priors(0){}

    CvDTreeParams( int _max_depth, int _min_sample_count,
                   float _regression_accuracy, bool _use_surrogates,
                   int _max_categories, int _cv_folds,
                   bool _use_1se_rule, bool _truncate_pruned_tree,
                   const float* _priors ) :
        max_categories(_max_categories), max_depth(_max_depth),
        min_sample_count(_min_sample_count), cv_folds (_cv_folds),
        use_surrogates(_use_surrogates), use_1se_rule(_use_1se_rule),
        truncate_pruned_tree(_truncate_pruned_tree),
        regression_accuracy(_regression_accuracy),
        priors(_priors){}
};
```

Obiekt `CvDTreeParams` umożliwia ustawienie następujących parametrów:

- `max_depth` – definiuje maksymalną głębokość każdego z drzew w lesie,
- `min_sample_count` - oznacza minimalną liczbę obiektów, która musi przypadać na dany węzeł, aby mógł on zostać podzielony na kolejne dwa węzły,
- `regression_accuracy` – używane jest tylko w zadaniu regresji,
- `use_surrogates` – określa czy dla węzłów, oprócz optymalnych testów podziału, mają być wyznaczane również zamienne testy, używane, gdy brakuję części danych,

- `max_categories` - definiuje maksymalną ilość klastrów, na które dzielony jest zbiór danych podczas poszukiwania dla nie go testu podziału - wartość ta powinna być mniejsza od liczby wszystkich kategorii w zbiorze uczącym,
- `cv_folds` - określa ile razy ma być wykonana walidacja krzyżowa,
- `use_1se_rule` – ustawione na true zapewnia stworzenie drzewa odporniejszego na szumy, lecz mniej skutecznego, jeżeli drzewo będzie przycinane,
- `truncate_pruned_tree` – określa czy gałęzie odrzucone w procesie przycinania drzewa mają zostać skasowane (true) czy też informacje o nich mają zostać zachowane (false),
- `priors` - macierz wag poszczególnych klas ze zbioru uczącego. Im wyższa jest waga dla danej klasy, tym bardziej znaczący, w porównaniu do innych błędnych klasyfikacji, jest błąd klasyfikacji dla tej klasy.

W celu stworzenia działającego klasyfikatora należy utworzyć obiekt klasy `CvDTree`. Trenowanie klasyfikatora odbywa się przy pomocy metody `train()` przyjmującej następujące dane:

- macierz wektorów cech obiektów ze zbioru uczącego,
- flaga określająca czy kolejne wektory cech znajdują się w kolejnych wierszach czy kolumnach pierwszej macierzy,
- macierz zawierająca klasy kolejnych obiektów ze zbioru uczącego,
- macierz wskazująca, które z cech mają być brane pod uwagę (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zestawu cech),
- macierz wskazująca, które z obiektów ze zbioru uczącego mają być rozpatrywane (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zbioru uczącego),
- macierz określająca typy poszczególnych cech wektora cech (`CV_VAR_ORDERED`) oraz wyniku klasyfikacji (`CV_VAR_CATEGORICAL`),
- macierz wskazująca wybrakowane dane,
- obiekt klasy `CvRTParams`.

Klasyfikacji obiektu dokonuje się przy użyciu metody `predict()`. Danymi przekazywanymi do metody są odpowiednio: wektor cech obiektu, który ma zostać poddany klasyfikacji, macierz wskazująca wybrakowane dane. Metoda zwraca oszacowaną klasę obiektu.

1.5 Las losowy

Las losowy jest zbiorem drzew decyzyjnych. W procesie klasyfikacji nieznanego obiektu każde z drzew decyzyjnych dokonuje klasyfikacji niezależnie od pozostałych drzew. Następnie zliczane są wszystkie klasy określone przez poszczególne drzewa, a obiektowi przypisywana jest klasa, która została wskazana największą ilość razy. Istotną cechą lasu losowego jest jego niepodatność na przetrenowanie oraz wysoka efektywność nawet dla bardzo dużych zbiorów uczących składających się z bardzo rozbudowanych wektorów cech. Pozwala oszacować ważność poszczególnych cech biorących udział w procesie klasyfikacji. Tak samo jak drzewo decyzyjne, dokonuje skutecznej klasyfikacji w przypadku braku części danych wektora cech klasyfikowanego obiektu [6].

Budowanie lasu losowego polega na niezależnym szkoleniu określonej liczby drzew decyzyjnych. Wszystkie drzewa decyzyjne tworzone są przy użyciu tego samego zestawu parametrów. Dla każdego drzewa ze zbioru uczącego klasyfikatora wydzielany jest podzbiór danych uczących. Liczba wybranych obiektów równa jest liczbie klas występujących w zbiorze uczącym. Poszczególne obiekty wybierane są losowo, w efekcie czego część obiektów może powtarzać się w różnych podzbiorach uczących, a część może nie występować w żadnym. Następnie drzewo szkolone jest w oparciu o około 76% [20] obiektów z podzbioru uczącego, pozostałe obiekty używane są na etapie dołączania drzewa do lasu w celu oszacowania błędu klasyfikacji (*OOB error*) oraz wyznaczenia ważności poszczególnych cech. Podczas szkolenia każdego z drzew, w momentach dokonywania podziału danych w węzłach, nie są brane pod uwagę wszystkie cechy obiektów a jedynie *m* losowo wybranych (każdy węzeł może rozpatrywać inny podzbiór cech). Wartość *m* jest stała dla wszystkich drzew i powinna być dużo mniejsza od liczby wszystkich cech.

Skuteczność klasyfikacji lasu losowego uzależniona jest od dwóch czynników. Pierwszym z nich jest korelacja pomiędzy dowolnymi dwoma drzewami. Im jest większa tym skuteczność klasyfikacji jest gorsza. Drugim jest skuteczność klasyfikacji każdego z drzew decyzyjnych. Zwiększanie skuteczności poszczególnych drzew zwiększa skuteczność całego lasu. Oba czynniki uzależnione są od parametru *m*. Zwiększając *m* zwiększa się korelację pomiędzy poszczególnymi drzewami oraz skuteczność klasyfikacji pojedynczych drzew. Zmniejszając *m* otrzymuje się efekt odwrotny. Odpowiednią wartość *m* należy dobrać eksperymentalnie.

Przydatną cechą lasu losowego jest możliwość wyznaczenia podobieństwa (ang. *proximity*) dwóch obiektów. Przy pomocy każdego z drzew decyzyjnych w lesie dokonuje się klasyfikacji obu obiektów. Podobieństwo wyrażone jest jako iloraz liczby klasyfikacji obu obiektów do tej samej klasy przez poszczególne drzewa i liczby wszystkich drzew w lesie.

Las losowy w OpenCV zaimplementowany jest w module Machine Learning pod nazwą `cvRTrees`. Poniżej znajduje się model klasy lasu losowego [20]:

```
class CV_EXPORTS CvRTrees : public CvStatModel{
public:
    CvRTrees();
    virtual ~CvRTrees();
    virtual bool train( const CvMat* _train_data, int _tflag,
                const CvMat* _responses, const CvMat* _var_idx=0,
                const CvMat* _sample_idx=0, const CvMat* _var_type=0,
                const CvMat* _missing_mask=0,
                CvRTParams params=CvRTParams() );

    virtual bool train( CvMLData* data, CvRTParams params=CvRTParams() );
    virtual float predict( const CvMat* sample, const CvMat* missing = 0 )
    const;
    virtual float predict_prob( const CvMat* sample,
                const CvMat* missing = 0 ) const;

    virtual void clear();
    virtual const CvMat* get_var_importance();
    virtual float get_proximity( const CvMat* sample1,
                const CvMat* sample2, const CvMat* missing1 = 0,
                const CvMat* missing2 = 0 ) const;

    virtual float calc_error( CvMLData* _data, int type ,
                std::vector<float> *resp = 0 );

    virtual float get_train_error();

    virtual void read( CvFileStorage* fs, CvFileNode* node );
    virtual void write( CvFileStorage* fs, const char* name ) const;

    CvMat* get_active_var_mask();
    CvRNG* get_rng();

    int get_tree_count() const;
    CvForestTree* get_tree(int i) const;

protected:
    virtual bool grow_forest( const CvTermCriteria term_crit );
    // array of the trees of the forest
    CvForestTree** trees;
    CvDTreeTrainData* data;
    int ntrees;
    int nclasses;
    double oob_error;
    CvMat* var_importance;
    int nsamples;

    CvRNG rng;
    CvMat* active_var_mask;
};
```

Konfiguracja klasyfikatora odbywa się poprzez obiekt klasy `CvRTParams`:

```
struct CV_EXPORTS CvRTParams : public CvDTreeParams{
    //Parameters for the forest
    bool calc_var_importance; // true <=> RF processes variable importance
    int nactive_vars;
    CvTermCriteria term_crit;

    CvRTParams() : CvDTreeParams( 5, 10, 0, false, 10, 0, false, false, 0
), calc_var_importance(false), nactive_vars(0){
        term_crit = cvTermCriteria( CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 50,
0.1 );
    }

    CvRTParams( int _max_depth, int _min_sample_count,
            float _regression_accuracy, bool _use_surrogates,
            int _max_categories, const float* _priors,
            bool _calc_var_importance,
            int _nactive_vars, int max_num_of_trees_in_the_forest,
            float forest_accuracy, int termcrit_type ) :
        CvDTreeParams( _max_depth, _min_sample_count, _regression_accuracy,
                       _use_surrogates, _max_categories, 0,
                       false, false, _priors ),
                      calc_var_importance(_calc_var_importance),
                      nactive_vars(_nactive_vars){
        term_crit = cvTermCriteria(termcrit_type,
            max_num_of_trees_in_the_forest, forest_accuracy);
    }
};
```

Obiekt `CvDTreeParams` umożliwia ustawienie następujących parametrów:

- `max_depth` - definiuje maksymalną głębokość każdego z drzew w lesie,
- `min_sample_count` - oznacza minimalną liczbę obiektów, która musi przypadać na dany węzeł, aby mógł on zostać podzielony na kolejne dwa węzły,
- `regression_accuracy` - używane jest tylko w zadaniu regresji,
- `use_surrogates` - określa czy dla węzłów oprócz optymalnych testów podziału mają być wyznaczane również zamienne testy, używane gdy brakuję części danych,
- `max_categories` - definiuje maksymalną ilość klastrów, na które dzielony jest zbiór danych podczas poszukiwania dla nie go testu podziału, wartość ta powinna być mniejsza od ilości wszystkich kategorii w zbiorze uczącym,
- `calc_var_importance` - wskazuje czy mają być określane ważności poszczególnych cech,
- `nactive_vars` - liczba losowo wybieranych obiektów *m*,
- `max_num_of_trees_in_the_forest` - określa maksymalną ilość drzew, z których składa się las,
- `forest_accuracy` - to akceptowalny błąd klasyfikacji lasu,
- `termcrit_type` - określa kryterium przerwania szkolenia klasyfikatora. Jeżeli przyjmuje on wartość `CV_TERMCRIT_ITER` klasyfikator skończy uczenie w momencie utworzenia maksymalnej ilości drzew, dla `CV_TERMCRIT_EPS` uczenie

zostanie przerwane, gdy błąd klasyfikatora będzie mniejszy lub równy akceptowalnemu błędowi klasyfikacji, zaś dla `CV_TERMCRIT_ITER|CV_TERMCRIT_EPS` uczenie zostanie przerwane, jeżeli spełniony zostanie jeden z powyższych warunków.

W celu stworzenia działającego klasyfikatora należy utworzyć obiekt klasy `CvRTrees`. Trenowanie klasyfikatora odbywa się przy pomocy metody `train()` przyjmującej następujące dane:

- macierz wektorów cech obiektów ze zbioru uczącego,
- flaga określająca czy kolejne wektory cech znajdują się w kolejnych wierszach czy kolumnach pierwszej macierzy,
- macierz zawierająca klasy kolejnych obiektów ze zbioru uczącego,
- macierz wskazująca, które z cech mają być brane pod uwagę (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zestawu cech),
- macierz wskazująca, które z obiektów ze zbioru uczącego mają być rozpatrywane (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zbioru uczącego),
- macierz określająca typy poszczególnych cech wektora cech (`CV_VAR_ORDERED`) oraz wyniku klasyfikacji (`CV_VAR_CATEGORICAL`),
- macierz wskazująca wybrakowane dane, oraz obiekt klasy `CvRTParams`.

Klasyfikacji obiektu dokonuje się przy użyciu metody `predict()`. Danymi przekazywanymi do metody są odpowiednio: wektor cech obiektu, który ma zostać poddany klasyfikacji oraz macierz wskazująca wybrakowane dane. Metoda zwraca oszacowaną klasę obiektu.

Do wyznaczenia podobieństwa dwóch obiektów służy metoda `get_proximity()`, która jako parametry przyjmuje dwa wektory cech obiektów.

1.6 AdaBoost

Metoda wzmacniania klasyfikatorów (ang. boosting) jest ogólną i efektywną metodą tworzenia skutecznych klasyfikatorów poprzez łączenie ze sobą wielu słabych klasyfikatorów. Najpopularniejszym algorytmem implementującym metodę wzmacniania klasyfikatorów jest algorytm AdaBoost. W przeciwieństwie do większości algorytmów klasyfikacji, AdaBoost pozwala dokonywać jedynie klasyfikacji binarnej (dwie klasy). Biorąc pod uwagę powyższy fakt, dobierane do konstrukcji klasyfikatora słabe klasyfikatory powinny mieć skuteczność klasyfikacji większą od 50%, czyli większą od przypadkowego

wyboru. Najczęściej do tej roli dobierane są proste drzewa decyzyjne – w niektórych przypadkach wystarczające mogą okazać się nawet drzewa składające się tylko z jednego węzła. Choć powszechnie uważa się, iż AdaBoost jest odporny na przeuczenie, szczegółowe badania wykazały błędność tego założenia dla bardzo dużych iteracji działania algorytmu [9].

Algorytm w pierwszej fazie swojego działania wszystkim obiektom ze zbioru uczącego przypisuje wagi $\omega_i = \frac{1}{N}$, gdzie i jest numerem konkretnego obiektu a N liczbą wszystkich obiektów w zbiorze. Następnie losowo wybiera h obiektów ze zbioru uczącego tworząc nowy zbiór C_1 i używa go jako zbioru uczącego do trenowania pierwszego słabego klasyfikatora K_1. Dla utworzonego klasyfikatora K_1wyznaczany jest błąd klasyfikacji E_1, jako suma wag obiektów źle sklasyfikowanych oraz współczynnik skalowania wag $\alpha_1$opisany wzorem [13]:

$$\alpha_k = \frac{1}{2}\ln(\frac{1-E_k}{E_k}), \tag{1.6.1}$$

Przy użyciu $\alpha_1$wyznaczane są nowe wagi dla wszystkich obiektów w zbiorze uczącym w następujący sposób:

$$\omega_i = \omega_i * \begin{cases} e^{-\alpha_k} & dla\ poprawnie\ sklasyfikowanego\ obiektu \\ e^{\alpha_k} & dla\ błędnie\ sklasyfikowanego\ obiektu \end{cases} \tag{1.6.2}$$

Po wyznaczeniu nowych wartości, wagi są normalizowane. W kolejnych iteracjach algorytm tworzy nowe podzbiory uczące $C_k$uwzględniając wagi przypisane poszczególnym obiektom. Obiekty z większymi wagami mają większą szansę dostać się do podzbioru uczącego. W ten sposób kolejne słabe klasyfikatory E_k przygotowywane są bardziej pod kątem klasyfikacji obiektów wcześniej błędnie sklasyfikowanych (trudniejszych przypadków). Dla każdego wytrenowanego słabego klasyfikatora K_k algorytm wyznacza E_k, α_k oraz zestaw wag dla obiektów w zbiorze uczącym. Algorytm wykonywany jest kolejne *T- 1* razy [7].

Klasyfikacja nieznanego obiektu polega na klasyfikacji obiektu T wytrenowanymi klasyfikatorami K_k sumując ich odpowiedzi $f_k(x)$ wymnożone przez odpowiadające im współczynniki skalowania α_k:

$$wybrana\ klasa = \sum_{k=1}^{T} \alpha_k\, f_k(x). \tag{1.6.3}$$

Klasyfikator AdaBoost w OpenCV zaimplementowany jest w module Machine Learning pod nazwą `cvBoost`. Słabymi klasyfikatorami użytymi do budowy klasyfikatora są drzewa decyzyjne. Poniżej znajduje się model klasy klasyfikatora AdaBoost [20]:

```
class CV_EXPORTS CvBoost : public CvStatModel{
public:
    // Boosting type
    enum { DISCRETE=0, REAL=1, LOGIT=2, GENTLE=3 };

    // Splitting criteria
    enum { DEFAULT=0, GINI=1, MISCLASS=3, SQERR=4 };

    CvBoost();
```

```
    virtual ~CvBoost();

    CvBoost( const CvMat* _train_data, int _tflag,
             const CvMat* _responses, const CvMat* _var_idx=0,
             const CvMat* _sample_idx=0, const CvMat* _var_type=0,
             const CvMat* _missing_mask=0,
             CvBoostParams params=CvBoostParams() );

    virtual bool train( const CvMat* _train_data, int _tflag,
             const CvMat* _responses, const CvMat* _var_idx=0,
             const CvMat* _sample_idx=0, const CvMat* _var_type=0,
             const CvMat* _missing_mask=0,
             CvBoostParams params=CvBoostParams(),
             bool update=false );

    virtual bool train( CvMLData* data,
             CvBoostParams params=CvBoostParams(),
             bool update=false );

    virtual float predict( const CvMat* _sample, const CvMat* _missing=0,
                        CvMat* weak_responses=0,
                        CvSlice slice=CV_WHOLE_SEQ,
                        bool raw_mode=false,
                        bool return_sum=false ) const;
    ...
}
```

Konfiguracja klasyfikatora odbywa się poprzez obiekt klasy `CvBoostParams`:

```
struct CV_EXPORTS CvBoostParams : public CvDTreeParams{
    int boost_type;
    int weak_count;
    int split_criteria;
    double weight_trim_rate;

    CvBoostParams();
    CvBoostParams( int boost_type, int weak_count, double weight_trim_rate,
                   int max_depth, bool use_surrogates, const float* priors
                 );
};
```

Obiekt `CvBoostParams` umożliwia ustawienie następujących parametrów:

- `boost_type` – określa typ modyfikacji algorytmu AdaBoost, może przyjmować cztery wartości: `CvBoost::DISCRETE`, `CvBoost::REAL`, `CvBoost::LOGIT`, `CvBoost::GENTLE`. Według dokumentacji najkorzystniejsza w zadaniu klasyfikacji jest modyfikacja `CvBoost::REAL`,
- `weak_count` – określa ile słabych klasyfikatorów ma zostać użytych podczas konstruowania klasyfikatora,
- `weight_trim_rate` – może być użyty do przyspieszenia szkolenia klasyfikatora. Obiekty ze zbioru uczącego, dla których wagi są mniejsze niż 1 - `weight_trim_rate` nie są brane pod uwagę podczas tworzenia nowych zbiorów uczących w kolejnych etapach uczenia klasyfikatora,

- split_criteria – może przyjmować jedną z czterech wartości: CvBoost::DEFAULT, CvBoost::GINI, CvBoost::MISCLASS lub CvBoost::SQERR i określa metodę wybierania optymalnych testów w poszczególnych węzłach konstruowanych drzew,
- max_depth, use_surrogates i priors - są to parametry drzew decyzyjnych i ich opis został umieszczony w rozdziale poświęconym drzewom decyzyjnym.

W celu stworzenia działającego klasyfikatora należy utworzyć obiekt klasy CvBoost. Trenowanie klasyfikatora odbywa się przy pomocy metody train() przyjmującej następujące dane:

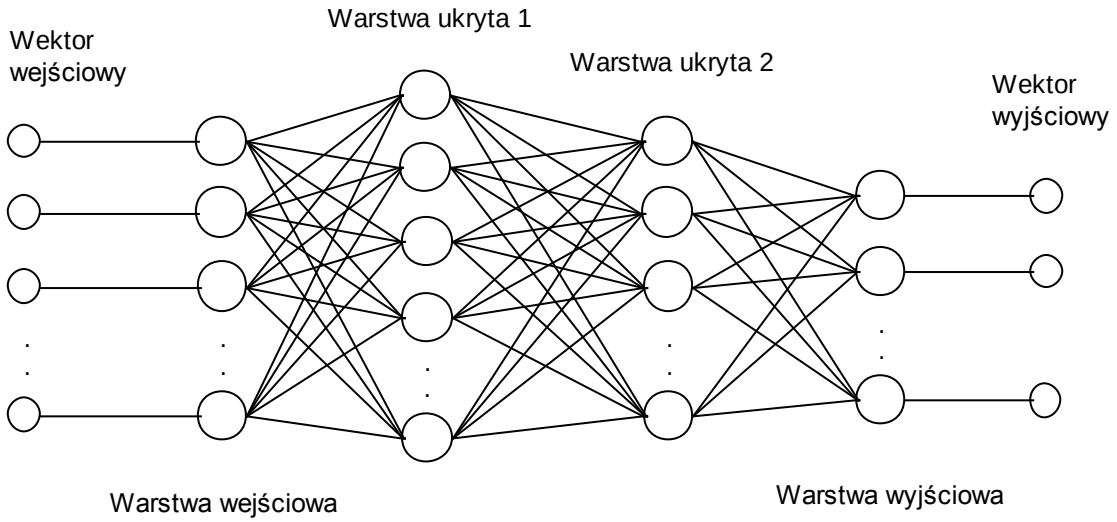
- macierz wektorów cech obiektów ze zbioru uczącego,
- flaga określająca czy kolejne wektory cech znajdują się w kolejnych wierszach czy kolumnach pierwszej macierzy,
- macierz zawierająca klasy kolejnych obiektów ze zbioru uczącego,
- macierz wskazująca, które z obiektów ze zbioru uczącego mają być rozpatrywane (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zbioru uczącego),
- macierz określająca typy poszczególnych cech wektora cech (CV_VAR_ORDERED) oraz wyniku klasyfikacji (CV_VAR_CATEGORICAL),
- macierz wskazująca wybrakowane dane,
- obiekt klasy CvBoostParam,
- wartość false, jeżeli ma zostać utworzony nowy klasyfikator lub true, jeśli wcześniej wyszkolony klasyfikator ma zostać doszkolony na nowym zbiorze uczącym.

Klasyfikacji obiektu dokonuje się przy użyciu metody predict(). Danymi przekazywanymi do metody są odpowiednio:

- macierz wektorów cech obiektów, które mają zostać poddane klasyfikacji,
- macierz wskazująca wybrakowane dane,
- pusta macierz odpowiedzi każdego ze słabych klasyfikatorów,
- zakres wskazujący, które ze słabych klasyfikatorów mają zostać użyte,
- wartość false, jeżeli wektory cech są znormalizowane lub true jeśli klasyfikatora ma sam dokonać normalizacji
- wartość true, jeśli funkcja ma zwrócić sumę odpowiedzi każdego ze słabych klasyfikatorów zamiast wybranej klasy.

1.7 MLP (perceptron wielowarstwowy)

Klasyfikator działający na zasadzie perceptronu wielowarstwowego jest siecią neuronową składająca się z warstw połączonych ze sobą neuronów nieliniowych. Składa się z warstwy wejściowej i wyjściowej. Kolejne warstwy znajdujące się pomiędzy nimi nazywane są warstwami ukrytymi. W szczególnym przypadku sieć może nie posiadać żadnych warstw ukrytych i nazywana jest wtedy siecią jednowarstwową [17]. Warstwa wejściowa zawsze składa się z tylu neuronów ile wartości znajduje się w wektorze cech zaś warstwa wyjściowa z tylu neuronów z ilu klas składa się zbiór uczący [17][18]. Liczba warstw ukrytych i znajdujących się w nich neuronów nie jest z góry określona i musi być dobrana do konkretnego zadania klasyfikacji [7]. Warstwy łączone są ze sobą tak, że kolejne warstwy są wyjściami dla wcześniejszych i wejściami dla następnych.



Rysunek 2. Struktura perceptronu wielowarstwowego.

Każdy z neuronów wchodzących w skład perceptronu wielowarstwowego posiada *n* wejść, jedno wyjście, funkcję aktywacji φ oraz określoną wartość progu ω_0. Każde jego wejście ma przypisaną odpowiednią wagę ω_i. Wartość wyjściowa neuronu *y* jest wynikiem funkcji aktywacji φ , której argumentem jest suma iloczynów wartości $x_i$znajdujących się na poszczególnych wejściach i odpowiadających im wag ω_i oraz progu ω_0 [5]:

$$y = \varphi(\omega_0 + \sum_{i=1}^{n} \omega_i x_i), \tag{1.7.1}$$

Funkcja aktywacji φ może przyjmować różne postacie, lecz powinna spełniać dwa warunki: dążyć do pewnych ustalonych wartości, gdy jej argument dąży do plus lub minus

nieskończoności oraz mieć nieskomplikowaną pochodną, aby jej obliczanie podczas procesu uczenia nie było zbyt złożone obliczeniowo. Najczęściej wybieraną funkcją jest funkcja sigmoidalna. Używane są jej dwie postaci. Pierwsza zwana unipolarną, przyjmuje wartości z zakresu [0,1] opisana jest wzorem:

$$\varphi(x) = \frac{1}{1+ e^{-\beta x}}, \tag{1.7.2}$$

i posiada pochodną w postaci:

$$\varphi'(x) = \beta x(1 - x), \tag{1.7.3}$$

Druga zwana bipolarną, przyjmuje wartości z zakresu [-1,1] opisana jest wzorem:

$$\varphi(x) = \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}}, \tag{1.7.4}$$

i posiada pochodną w postaci:

$$\varphi'(x) = \beta(x + 1)(x - 1), \tag{1.7.5}$$

Proces uczenia sieci polega na iteracyjnej modyfikacji wag poszczególnych neuronów tak, aby minimalizowana była funkcja energetyczna, która dla jednego neuronu może zostać przedstawiona w następującej postaci [14]:

$$E = \frac{1}{2}(y - z)^2, \tag{1.7.6}$$

gdzie *z* jest oczekiwaną wartością wyjścia neuronu a *y* faktyczną. Modyfikacja wag odbywa się w oparciu o regułę delta uwzględniającą minimalizację funkcji błędu średniokwadratowego dla zbioru wektorów wejściowych i ich oczekiwanych wartości wyjściowych. Błąd na wyjściu *i*-tego neuronu wynosi:

$$\varepsilon^i = z^i - y^i, \tag{1.7.7}$$

W każdej iteracji i uczenia sieci wyznaczany jest wektor zmian wag $\Delta\omega^i$, który służy do wyznaczenia wag w następnej iteracji:

$$\omega_j^{i+1} = \omega_j^i + \Delta\omega_j^i, \tag{1.7.8}$$

Współczynnik zmiany wag $\Delta\omega_j^i$ może być wyznaczany na wiele sposobów, najczęściej stosowanymi są metoda gradientowa i uczenie z momentu. Pierwszy sposób opisuje $\Delta\omega_j^i$ jako:

$$\Delta\omega_j^i = \eta\varepsilon^i\varphi'(y)x_j^i, \tag{1.7.9}$$

gdzie $0 < \eta < 1$ jest współczynnikiem uczenia, od którego zależy szybkość i dokładność uczenia neuronu. Im η jest większym tym sieć uczy się dłużej i staje się bardziej dokładna, lecz rośnie również ryzyko, iż sieć może utracić swoją zdolność generalizacji. Użycie zbyt małego współczynnika uczenia da w rezultacie mało skuteczną sieć. Metoda nazywa się gradientową, gdyż modyfikacja wag odbywa się w kierunku ujemnego gradientu funkcji energetycznej. Drugi sposób jest modyfikacją pierwszego i polega on na braniu pod uwagę nie tylko gradientu funkcji, ale również zmiany tych samych wag w iteracji wcześniejszej:

$$\Delta\omega_j^i = \eta_1 \varepsilon^i \varphi'(y) x_j^i + \eta_2 \Delta\omega_j^{i-1}, \tag{1.7.10}$$

gdzie:

$$\Delta\omega_j^{i-1} = \omega_j^i - \omega_j^{i-1}, \tag{1.7.11}$$

Współczynnik η_2 nazywany jest momentum. Metoda ta pozwala na szybsze osiągnięcie funkcji energetycznej a dodatkowo pozwala omijać minima lokalne i znajdować minima globalne. Aby metoda działała skutecznie należy odpowiednio dobrać współczynniki η_1 i η_2.

W sposób bezpośredni błąd na wyjściu poszczególnych neuronów danej warstwy można wyliczyć jedynie dla warstwy ostatniej. W celu określenia błędów na wyjściu warstw poprzedzających ostatnią stosuje się wiele różnych metod. Najpopularniejszą z nich jest metoda wstecznej propagacji. Polega ona na wyznaczaniu błędu wyjścia *i*-tego neurony na warstwie *n* - 1 poprzez obliczenie sumy iloczynów wag łączących ten neuron z neuronami warstwy *n* i odpowiednich składowych wektora błędu na wyjściu warstwy *n*:

$$\varepsilon_{n-1,i} = \sum_{j=1}^{k_n} \varepsilon_{n,j} \omega_{n,j \leftrightarrow n-1,i}, \tag{1.7.12}$$

gdzie $\varepsilon_{n-1,i}$ jest błędem na wyjściu *i*-tego neuronu warstwy *n*-1, $\varepsilon_{n,j}$ jest błędem na wyjściu kolejnych neuronów warstwy *n*, $\omega_{n,j \leftrightarrow n-1,i}$ jest wartością kolejnych wag łączących *i*-ty neuron warstwy *n*-1 z poszczególnymi neuronami warstwy *n* a k_n jest liczbą neuronów na warstwie *n*. Błędy wyznaczane są kolejno od przedostatniej warstwy do warstwy pierwszej.

W procesie szkolenia sieci neuronowej sposób wprowadzania kolejnych wektorów cech, które zostaną wykorzystane do nauki, ma wpływ na końcową strukturę sieci. Najczęściej stosowane są trzy sposoby prezentacji danych. Pierwszy polega na podawaniu na wejście sieci kolejnych wektorów cech ze zbioru uczącego i modyfikowaniu po tej operacji wag neuronów. Przy takim rozwiązaniu podczas każdej iteracji sieci wagi są zmieniane tyle razy ile zbiór uczący posiada obiektów. Drugi sposób jest modyfikacją pierwszego polegająca na zmianie wag neuronów tylko raz podczas każdej iteracji w oparciu o uśredniony błąd wyliczony podczas prezentacji wszystkich wektorów cech. Trzeci sposób jest również modyfikacją pierwszego polegająca na losowym wybieraniu wektorów cech ze zbioru uczącego i po ich prezentacji modyfikacji wag sieci.

Perceptron wielowarstwowy w OpenCV zaimplementowany jest w module Machine Learning pod nazwą `cvANN_MLP`. Poniżej znajduje się model jego klasy [20]:

```
class CV_EXPORTS CvANN_MLP : public CvStatModel{
public:
    CvANN_MLP();
    CvANN_MLP( const CvMat* _layer_sizes,
               int _activ_func=SIGMOID_SYM,
               double _f_param1=0, double _f_param2=0 );

    virtual ~CvANN_MLP();
```

```
    virtual void create( const CvMat* _layer_sizes,
                         int _activ_func=SIGMOID_SYM,
                         double _f_param1=0, double _f_param2=0 );

    virtual int train( const CvMat* _inputs, const CvMat* _outputs,
                  const CvMat* _sample_weights,
                  const CvMat* _sample_idx=0,
                  CvANN_MLP_TrainParams
                  _params = CvANN_MLP_TrainParams(),int flags=0 );

    virtual float predict( const CvMat* _inputs, CvMat* _outputs ) const;
    virtual void clear();

    // possible activation functions
    enum { IDENTITY = 0, SIGMOID_SYM = 1, GAUSSIAN = 2 };

    // available training flags
    enum { UPDATE_WEIGHTS = 1, NO_INPUT_SCALE = 2, NO_OUTPUT_SCALE = 4 };
    …
}
```

Konfiguracja klasyfikatora odbywa się poprzez wybranie funkcji aktywacji i jej parametrów oraz zdefiniowanie liczby warstw i znajdujących się na nich neuronów podczas tworzenia obiektu klasy a także poprzez obiekt klasy `CvANN_MLP_TrainParams`:

```
struct CV_EXPORTS CvANN_MLP_TrainParams{
    CvANN_MLP_TrainParams();
    CvANN_MLP_TrainParams( CvTermCriteria term_crit, int train_method,
                           double param1, double param2=0 );
    ~CvANN_MLP_TrainParams();

    enum { BACKPROP=0, RPROP=1 };

    CvTermCriteria term_crit;
    int train_method;

    // backpropagation parameters
    double bp_dw_scale, bp_moment_scale;

    // rprop parameters
    double rp_dw0, rp_dw_plus, rp_dw_minus, rp_dw_min, rp_dw_max;
};
```

OpenCV udostępnia dwie metody oszacowywania błędów na poszczególnych warstwach neuronów, `BACKPROP` czyli wsteczną propagację oraz `RPROP` będącą modyfikacją wcześniejszej zapewniającą większą szybkość uczenia. Parametry `param1` i `param2` dla metody BACKPROP odpowiadają odpowiednio współczynnikowi uczenia (`bp_dw_scale`) oraz momentum (`bp_moment_scale`) a dla `RPROP` `rp_dw0` i `rp_dw_min`. Pozostałe parametry metody RPROP nie są ustawiane przez konstruktor i należy jest ustawić samodzielnie. Kryterium zakończenia uczenia sieci `CvTermCriteria` należy podać, jako `CV_TERMCRIT_ITER` z określoną liczbą iteracji. Funkcja aktywacji

może przyjąć jedną z trzech postaci: tożsamościową `CvANN_MLP::IDENTITY`, sigmoidalną `CvANN_MLP::SIGMOID_SYM` oraz gausowską `CvANN_MLP::GAUSSIAN` (która nie jest do końca zaimplementowana). Liczbę warstw i ich neuronów definiuje się poprzez macierz 1xN wartości gdzie N jest ilością warstw a poszczególne wartości macierzy określają ilość neuronów na danej warstwie.

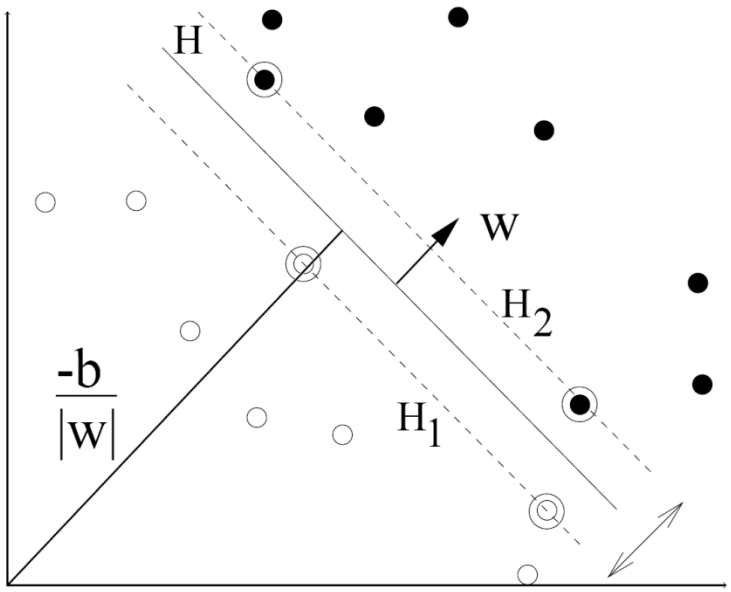
W celu stworzenia działającego klasyfikatora należy utworzyć obiekt klasy `CvANN_MLP` przekazując do konstruktora odpowiednio: macierz opisująca warstwy neuronów, typ funkcji aktywacji oraz dwa parametry funkcji aktywacji. Domyślną funkcją aktywacji jest funkcja sigmoidalna a parametry tej funkcji ustawiane są na 0. Zaimplementowana w bibliotece funkcja sigmoidalna dla parametrów zerowych zawsze przyjmować będzie wartość 0, dlatego też użycie takich parametrów jest niepoprawne. W praktyce okazuje się, że OpenCV dla zerowych parametrów funkcji aktywacji samo przypisuje im wartości 0.(6) oraz 1.7159. Utworzoną sieć należy następnie wytrenować przy użyciu funkcji `train()`, która przyjmuje następujące dane:

- macierz wektorów cech obiektów ze zbioru uczącego (wektory cech muszą być zapisane jako kolejne wiersze macierzy),
- macierz zawierająca klasy kolejnych obiektów ze zbioru uczącego,
- macierz zawierająca wagi dla poszczególnych obiektów ze zbioru uczącego (tylko dla metody `RPROP`),
- macierz wskazująca, które z obiektów ze zbioru uczącego mają być rozpatrywane (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zbioru uczącego),
- obiekt klasy `CvANN_MLP_TrainParams`,
- flagę przyjmująca wartość 0 lub 1 jeżeli wagi mają być modyfikowane w oparciu o wcześniej wyznaczone a nieprzeliczane od nowa, 2 jeżeli wektory wchodzące do danej warstwy mają być skalowane, 4 jeżeli wektory wychodzące z danej warstwy mają być skalowane.

Klasyfikacji nieznanego obiektu dokonuje się przy użyciu metody `predict()`, która jako dane otrzymuje: macierz wektorów cech obiektów, które mają zostać poddane klasyfikacji oraz pustą macierz wyników przeprowadzonej klasyfikacji (jeżeli klasyfikowany jest więcej niż jeden obiekt), która zostanie wypełniona podczas wykonywania metody. Jeżeli klasyfikowany jest tylko jeden obiekt metoda zwraca wybraną dla niego klasę.

1.8 SVM (maszyna wektorów nośnych)

SVM (*ang. Support Vector Machine*) w swej najprostszej formie jest liniowym klasyfikatorem binarnym. Jego działanie polega na wyznaczeniu hiperpłaszczyzny *H* dzielącej zbiór danych, na dwa rozdzielne podzbiory, w taki sposób, aby odległość hiperpłaszczyzny od najbliższych obiektów z obu podzbiorów była jak największa. Najbliższe hiperpłaszczyźnie obiekty z obu podzbiorów nazywane są wektorami nośnymi (ang. *support vectors*), zaś odległość między dwoma hiperpłaszczyznami (H_1 i H_2) wyznaczonymi przez wektory nośne obu klas nazywana jest marginesem separacji. Szerokość marginesu separacji wpływa na zdolność generalizacji klasyfikatora. Im margines jest szerszy tym zdolność generalizacji jest większa, co jednak powoduje gorszą klasyfikację danych uczących (część danych znajduje się wewnątrz marginesu i może zostać błędnie sklasyfikowana). Hiperpłaszczyzny H_1 i H_2 są równoległe do hiperpłaszczyzny *H* i oddalone od niej o tą samą odległość [12][19][1].



Rysunek 3. Przykład separacji liniowej zbioru białych i czarnych kropek. Kropki otoczone okręgami stanowią wektory nośne wyznaczające hiperpłaszczyzny H_1 i H_2. [1]

Poszukiwana hiperpłaszczyzna opisana jest równaniami [1]:

$$\omega \cdot x_i + b \geq 1 \; dla \; y_i = 1, \tag{1.8.1}$$

oraz

$$\omega \cdot x_i + b \leq -1 \; dla \; y_i = -1, \tag{1.8.2}$$

gdzie ω jest normalną do hiperpłaszczyzny, $\frac{|b|}{\|\omega\|}$ jest odległością hiperpłaszczyzny od środka układu współrzędnych, $\|\omega\|$ jest normą euklidesową ω, x_i jest punktem w układzie współrzędnych obrazującym *i*-ty obiekt a y_i jest klasą *i*-tego obiektu (przyjmującą wartość 1

lub -1). Margines separacji przyjmuje wartość $\frac{2}{\|\omega\|}$, tak więc poszukiwanie hiperpłaszczyzny *H* sprowadza się do wyznaczenia:

$$\min_{\omega} \frac{\|\omega\|^2}{2}, \tag{1.8.3}$$

przy warunkach ograniczających wynikających z równań 1.8.1 i 1.8.2:

$$y_i(\omega \cdot x_i + b) \geq dla\ i = 1, 2, \ldots, N. \tag{1.8.4}$$

Minimalizacja $\frac{\|\omega\|^2}{2}$ jest problemem optymalizacji kwadratowej z liniowymi ograniczeniami, dlatego też do jej rozwiązania wykorzystuje się metodę mnożników Lagrange'a. Poszukiwanie wartości 1.8.3 przy warunkach 1.8.4 zamienia się na poszukiwanie minimum funkcji:

$$L(\omega, b, \alpha) = \frac{\|\omega\|^2}{2} - \sum_{i=1}^{N} \alpha_i y_i (x_i \cdot \omega + b) + \sum_{i=1}^{N} \alpha_i, \tag{1.8.5}$$

gdzie α_i są mnożnikami Lagrange'a i spełniają warunek $\alpha_i \geq 0$. Równanie 1.8.5 można sprowadzić do postaci dualnej $L_D(\alpha)$, a problem poszukiwania minimum $L(\omega, b, \alpha)$ zamienić na problem poszukiwania maksimum $L_D(\alpha)$ gdzie:

$$L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i \cdot x_j, \tag{1.8.6}$$

przy założeniu:

$$\alpha_i \geq 0,\ \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{1.8.7}$$

oraz dodatkowych warunkach Karusha-Kuhna-Tuckera:

$$\alpha_i (y_i(\omega \cdot x_i + b) - 1) = 0. \tag{1.8.8}$$

Wartości α_i są niezerowe wyłącznie dla wektorów nośnych, dla pozostałych punktów przyjmują wartość 0.

Powyższe równania są prawdziwe jedynie dla danych separowalnych liniowo i utworzony na ich podstawie klasyfikator SVM dokonywał będzie błędnych klasyfikacji dla innych zbiorów danych. Aby umożliwić klasyfikację danych nie w pełni separowalnych liniowo do równań 1.8.1 i 1.8.2 wprowadza się dodatkowe zmienne $\xi_i \geq 0$ dla i = 1, 2, … N zmniejszające margines separacji:

$$\omega \cdot x_i + b \geq 1 - \xi_i\ dla\ y_i = 1, \tag{1.8.9}$$

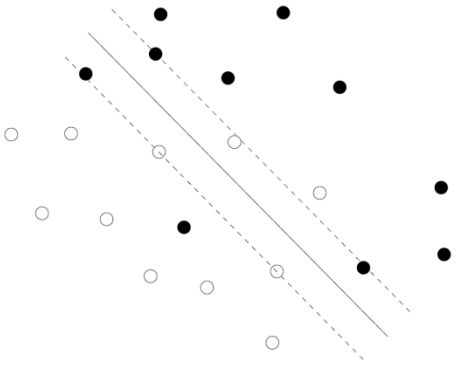
$$\omega \cdot x_i + b \leq -1 + \xi_i\ dla\ y_i = -1, \tag{1.8.10}$$

Jeżeli spełniony jest warunek $0 < \xi_i \leq 1$ to punkt danych (x_i, y_i) leży wewnątrz marginesu separacji, jednak po jego właściwej stronie. W przypadku, gdy $\xi_i > 1$ punkt znalazł się po niewłaściwej stronie hiperpłaszczyzny i nastąpiła niepoprawna klasyfikacja. W wyniku dodania zmiennych ξ_i wyrażenie 1.8.3 przyjmuje wartość:

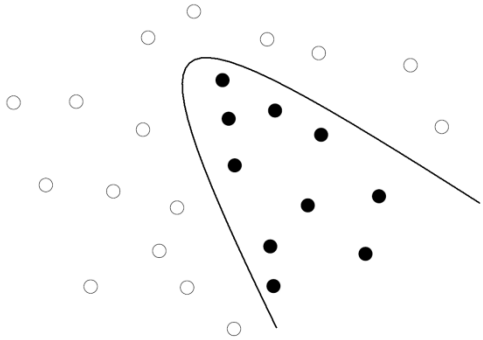
$$\min_{\omega} \left(\frac{\|\omega\|^2}{2} + C\left(\sum_i \xi_i\right)^k\right), \tag{1.8.11}$$

gdzie C jest współczynnikiem oceny straty związanej z każdą błędną klasyfikacją a jego wartość dobierana jest eksperymentalnie, zaś k najczęściej przyjmuje wartość równą 1, co upraszcza obliczenia [1]. Podczas poszukiwania maksimum funkcji $L_D(\alpha)$ (1.8.6) dochodzi dodatkowe ograniczenie dla parametru α_i:

$$\alpha_i \leq C. \tag{1.8.12}$$



Rysunek 4. Dane nie w pełni separowalne liniowo.



Rysunek 5. Dane nieseparowalne liniowo.

W praktyce wiele ze zbiorów danych poddawanych klasyfikacji nie jest separowalnych liniowo. W takim przypadku dane muszą zostać przetransformowane do wysoce wielowymiarowej przestrzeni. Im większy jest wymiar nowej przestrzeni tym większe jest prawdopodobieństwo znalezienia hiperpłaszczyzny umożliwiającej poprawne podzielenie danych. Równanie 1.8.6 przyjmuje postać:

$$L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \Phi(x_i) \cdot \Phi(x_j), \tag{1.8.13}$$

gdzie Φ jest funkcją transformującą zbiór danych do wysoce wielowymiarowej przestrzeni. Ponieważ wyznaczanie $\Phi(x_i)$, $\Phi(x_j)$ oraz ich iloczynu skalarnego dla wielowymiarowych przestrzeni jest bardzo skomplikowane obliczeniowo wyrażenie $\Phi(x_i) \cdot \Phi(x_j)$ w równaniu (1.8.13) zastępowane jest funkcją jądra $K(x_i, x_j)$. Dzięki tej operacji dokładna postać funkcji Φ nie musi być znana, a funkcje jądra mogą być dobrane tak, aby ich złożoność obliczeniowa była zadowalająca. Do najczęściej używanych funkcji jądra należą:

- Funkcje wielomianowe:

$$K(x_i, x_j) = (x_i \cdot x_j)^d, \tag{1.8.14}$$

- Funkcje sigmoidalne:

$$K(x_i, x_j) = tgh(\tau x_i \cdot x_j - \delta), \tag{1.8.15}$$

- RBF:

$$K(x_i, x_j) = \exp(\frac{\|x_i - x_j\|^2}{\delta^2}). \quad (1.8.16)$$

Skuteczność klasyfikatora SVM dla danego zbioru danych w znacznym stopniu uzależniona jest od doboru odpowiedniej funkcji jądra, a dobór ten musi zostać przeprowadzany eksperymentalnie.

Klasyfikator SVM w OpenCV zaimplementowany jest w module Machine Learning pod nazwą `CvSVM`. Poniżej znajduje się model jego klasy [20]:

```
class CvSVM : public CvStatModel{
public:
    // SVM type
    enum { C_SVC=100, NU_SVC=101, ONE_CLASS=102, EPS_SVR=103, NU_SVR=104 };

    // SVM kernel type
    enum { LINEAR=0, POLY=1, RBF=2, SIGMOID=3 };

    // SVM params type
    enum { C=0, GAMMA=1, P=2, NU=3, COEF=4, DEGREE=5 };

    CvSVM();
    virtual ~CvSVM();

    CvSVM( const CvMat* _train_data, const CvMat* _responses,
           const CvMat* _var_idx=0, const CvMat* _sample_idx=0,
           CvSVMParams _params=CvSVMParams() );

    virtual bool train( const CvMat* _train_data, const CvMat* _responses,
                        const CvMat* _var_idx=0,
                        const CvMat* _sample_idx=0,
                        CvSVMParams _params=CvSVMParams() );

    virtual bool train_auto( const CvMat* _train_data,
        const CvMat* _responses,
        const CvMat* _var_idx, const
        CvMat* _sample_idx, CvSVMParams _params,
         int k_fold = 10,
         CvParamGrid C_grid      = get_default_grid(CvSVM::C),
         CvParamGrid gamma_grid  = get_default_grid(CvSVM::GAMMA),
         CvParamGrid p_grid      = get_default_grid(CvSVM::P),
         CvParamGrid nu_grid     = get_default_grid(CvSVM::NU),
         CvParamGrid coef_grid   = get_default_grid(CvSVM::COEF),
         CvParamGrid degree_grid = get_default_grid(CvSVM::DEGREE) );

    virtual float predict( const CvMat* _sample ) const;
    virtual int get_support_vector_count() const;
    virtual const float* get_support_vector(int i) const;
    virtual CvSVMParams get_params() const { return params; };
    virtual void clear();

    static CvParamGrid get_default_grid( int param_id );

    virtual void save( const char* filename, const char* name=0 );
    virtual void load( const char* filename, const char* name=0 );

    virtual void write( CvFileStorage* storage, const char* name );
    virtual void read( CvFileStorage* storage, CvFileNode* node );
    int get_var_count() const { return var_idx ? var_idx->cols : var_all; }
```

```
protected:
    ...
};
```

Konfiguracja klasyfikatora odbywa się poprzez obiekt `CvSVMParams`:

```
struct CvSVMParams{
    CvSVMParams();
    CvSVMParams( int _svm_type, int _kernel_type,
                 double _degree, double _gamma, double _coef0,
                 double _C, double _nu, double _p,
                 CvMat* _class_weights, CvTermCriteria _term_crit );

    int         svm_type;
    int         kernel_type;
    double      degree; // for poly
    double      gamma;  // for poly/rbf/sigmoid
    double      coef0;  // for poly/sigmoid

    double      C;  // for CV_SVM_C_SVC, CV_SVM_EPS_SVR and CV_SVM_NU_SVR
    double      nu; // for CV_SVM_NU_SVC, CV_SVM_ONE_CLASS, and
CV_SVM_NU_SVR
    double      p; // for CV_SVM_EPS_SVR
    CvMat*      class_weights; // for CV_SVM_C_SVC
    CvTermCriteria term_crit; // termination criteria
};
```

Obiekt `CvBoostParams` umożliwia ustawienie następujących parametrów:

- `svm_type` – określa typ użytego SVM i może przyjmować wartości: `CvSVM::C_SVC` i `CvSVM::NU_SVC` – klasyfikacja dwu i wieloklasowa, `CvSVM::ONE_CLASS` – klasyfikator wyznacza granicę otaczającą zbiór uczący i odzielającą go od reszty przestrzeni, oraz `CvSVM::EPS_SVR i CvSVM::NU_SVR`, które używane są w zadaniau regresji,
- `kernel_type` – funkcją jądra: `CvSVM::LINEAR` – liniowa, `CvSVM::POLY` – wielomianowa, `CvSVM::RBF` - radialna, `CvSVM::SIGMOID` – sigmoidalna,
- `degree, gamma, coef0, C, nu, p` – paramtery odpowiedzialne za konfigurowanie poszczególnych metod klasyfikacji i regresji oraz funkcji jądra,
- `class_weights` – macierz wag dla poszczególnych klas używana przy oszacowywaniu wazności błędu klasyfikacji,
- `term_crit` – warunki zakończenia uczenia, tak samo jak w poprzednich klasyfikatorach.

W celu stworzenia działającego klasyfikatora należy utworzyć obiekt klasy `CvBoost`. Trenowanie klasyfikatora odbywa się przy pomocy metody `train()` przyjmującej następujące dane:

- macierz wektorów cech obiektów ze zbioru uczącego,
- macierz zawierająca klasy kolejnych obiektów ze zbioru uczącego,

- macierz wskazująca, które z cech mają być brane pod uwagę (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zestawu cech),
- macierz wskazująca, które z obiektów ze zbioru uczącego mają być rozpatrywane (pozostawienie tej zmiennej ustawionej domyślnie na 0 powoduje rozpatrywanie całego zbioru uczącego),
- obiekt `CvBoostParam`.

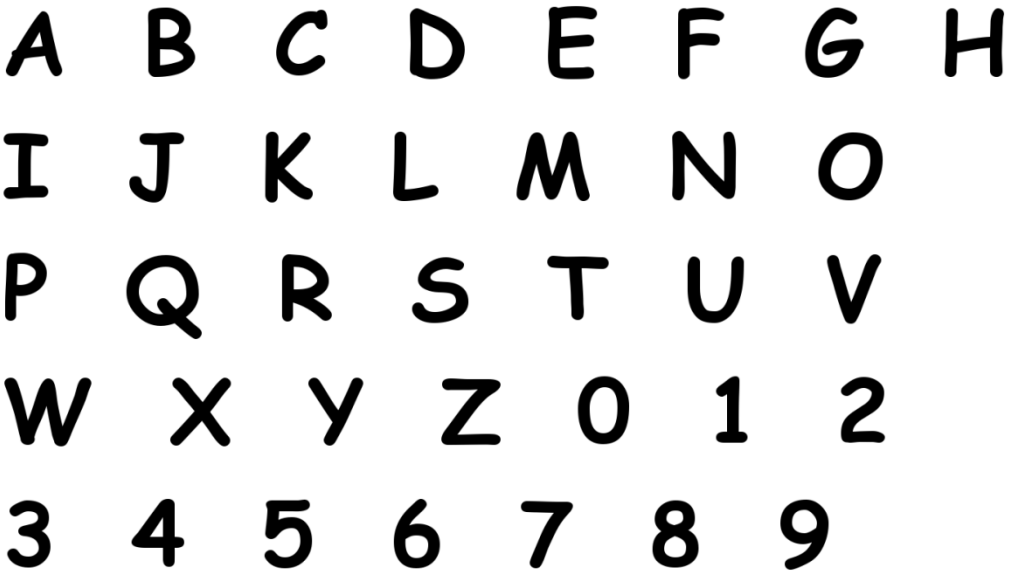
Parametry klasyfikatora w zależności od użytego typu SVM i funkcji jądra mogą również zostać wyznaczone automatycznie dla danego zbioru danych przy użyciu funkcji `train_auto()`. Najważniejszym parametrem powyższej funkcji jest paramter *k* określający liczbę walidacji krzyżowych przeprowadzonych podczas ustalania parametrów. Im jest on większy, tym teoretycznie dobrane paramtery są lepsze jednakże czas takiego szkolenia znacznie wzrasta. W niektórych przypadkach zwiększanie *k* nie powoduje dalszej zmiany wyznaczonych parametrów.

Klasyfikacji obiektu dokonuje się przy użyciu metody `predict()`, dla której jako argument podaje się wektor cech klasyfikowanego obiektu.

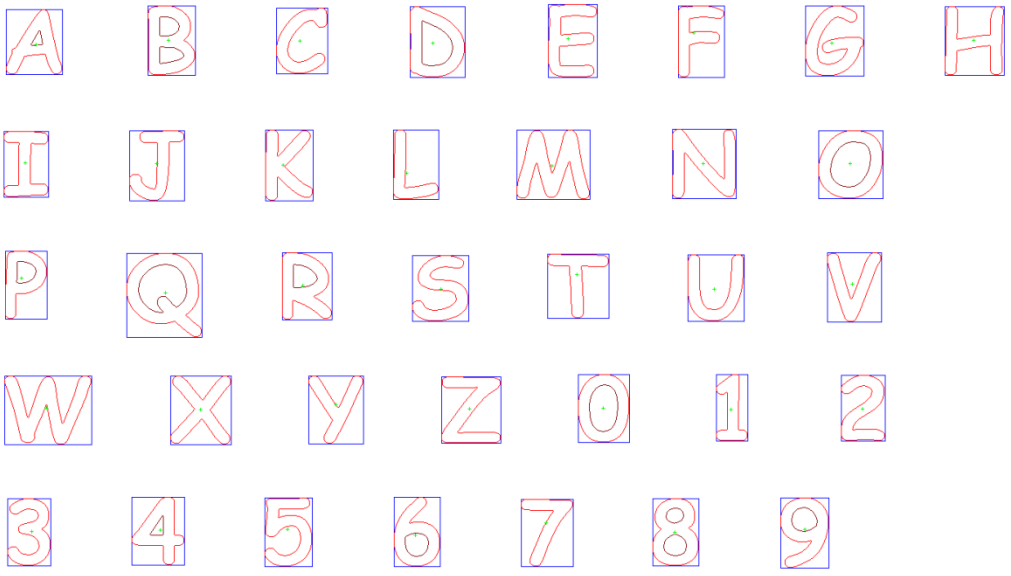
2 Zbiory danych

2.1 Przygotowanie danych

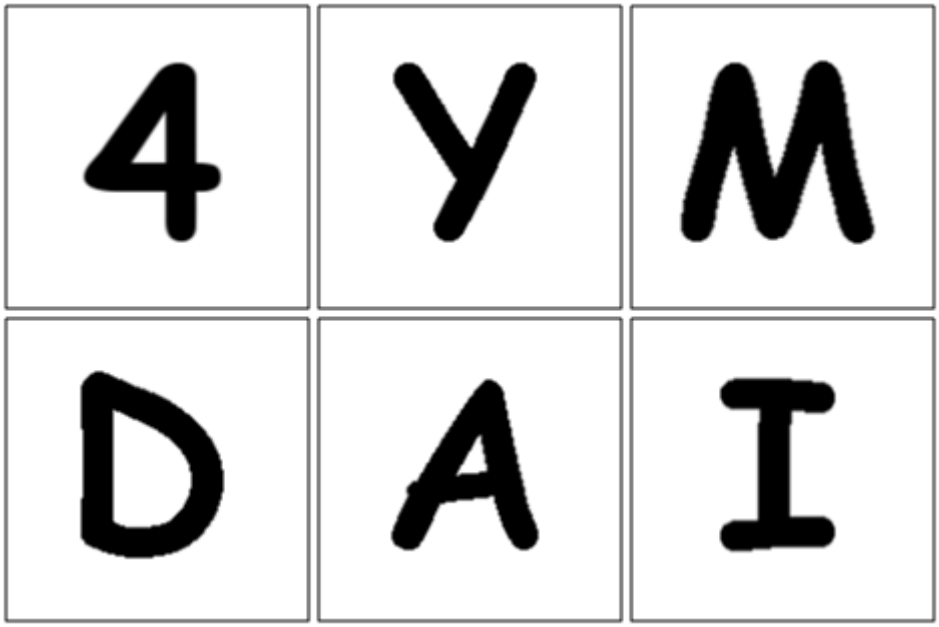
W przeprowadzonych badaniach klasyfikatory zostały przetestowane na zbiorze danych składającym się z 26 liter alfabetu łacińskiego oraz 10 cyfr arabskich. Spośród niezliczonej liczby czcionek komputerowych do uczenia klasyfikatorów wybrano siedem, kierując się ich popularnością i różnorodnością. Należą do nich: Arial, Century Gothic, Comic Sans, Tahoma, Terminal, Times New Roman oraz Verdana. Powyższych czcionek użyto również do testowania klasyfikatorów. W celu zbadania skuteczności klasyfikacji nieznanych typów czcionek zbiór testowy poszerzony został o kolejnych pięć: Georgia, Courier New, Helvetica, Perpatua. Wybrane czcionki należą do najpopularniejszych i najczęściej używanych, jednocześnie zapewniają różnorodność zbioru danych. Obrazy ze znakami przygotowano wykorzystując darmowy program graficzny Gimp w wersji 2.0. Do zapisu znaków alfanumerycznych użyto czcionek o wielkości 120 punktów, aby uwyraźnić charakterystyczne cechy poszczególnych liter i zapewnić wystarczającą rozdzielczość podczas skalowania w dół. W celu powiększenia i zróżnicowania zbioru danych każdy zestaw znaków zapisano dodatkowo w różnych wariantach użytych czcionek (pogrubiona, pochylona, pochylona i pogrubiona). Ponieważ segmentacja obrazu nie była celem przeprowadzonych badań dla ułatwienia litery i cyfry zapisano czarnym kolorem na białym tle. Każdy z zestawów znaków alfanumerycznych umieszczono w oddzielnym pliku TIFF (bez kompresji), a następnie przy pomocy napisanego programu wykrywającego kontury podzielono na 36 plików o wielkości 150x150 pikseli zawierających poszczególne znaki alfanumeryczne (znak bez skalowania umieszczony został w samym środku obrazu). Program zrealizowano w oparciu o funkcję OpenCV cvFindContours wywoływaną z parametrem CV_RETR_CCOMP, na wcześniej zbinaryzowanym obrazie znaku (ustawienie odpowiedniej wartości progowania pozwoliło odrzucić szum powstały podczas graficznego przetwarzania obrazu i ułatwiło wyszukiwanie konturów). Zbyt małe kontury zostały odrzucone, a te, które pozostały określały wykryte na obrazie litery i cyfry.



Rysunek 6. Obraz z 36 znakami alfanumerycznymi zapisanymi przy pomocy pogrubionej czcionki Comic Sans.



Rysunek 7. Obraz z Rysunku 1 po wykryciu konturów (zaznaczonych na czerwono). Niebieskie prostokąty wskazują wielkość i położenie poszczególnych znaków.



Rysunek 8. Wybrane znaki z Rysunku 1 po detekcji konturów i umieszczeniu ich w oddzielnych plikach o rozmiarze 150 na 150 pikseli.

W rezultacie przeprowadzonych operacji otrzymano 756 liter (czcionki Comic Sans i Tahoma posiadały tylko wariant pogrubienia zaś Terminal nie posiadał żadnego), które następnie poddano działaniu losowego szumu oraz dziesięciu różnym przekształceniom afinicznym otrzymując w ten sposób zbiór danych składający się z 9072 liter. Część cienkich liter w konsekwencji dodania szumu została rozdzielona na kilka kawałków, takie przypadki usunięto ze zbioru i nie rozpatrywano ich klasyfikacji. Aby zapewnić jednakową ilość znaków alfanumerycznych każdego rodzaju wybrano spośród każdego z nich 248 znaków. Zbiór ostateczny składał się więc z 8928 elementów.

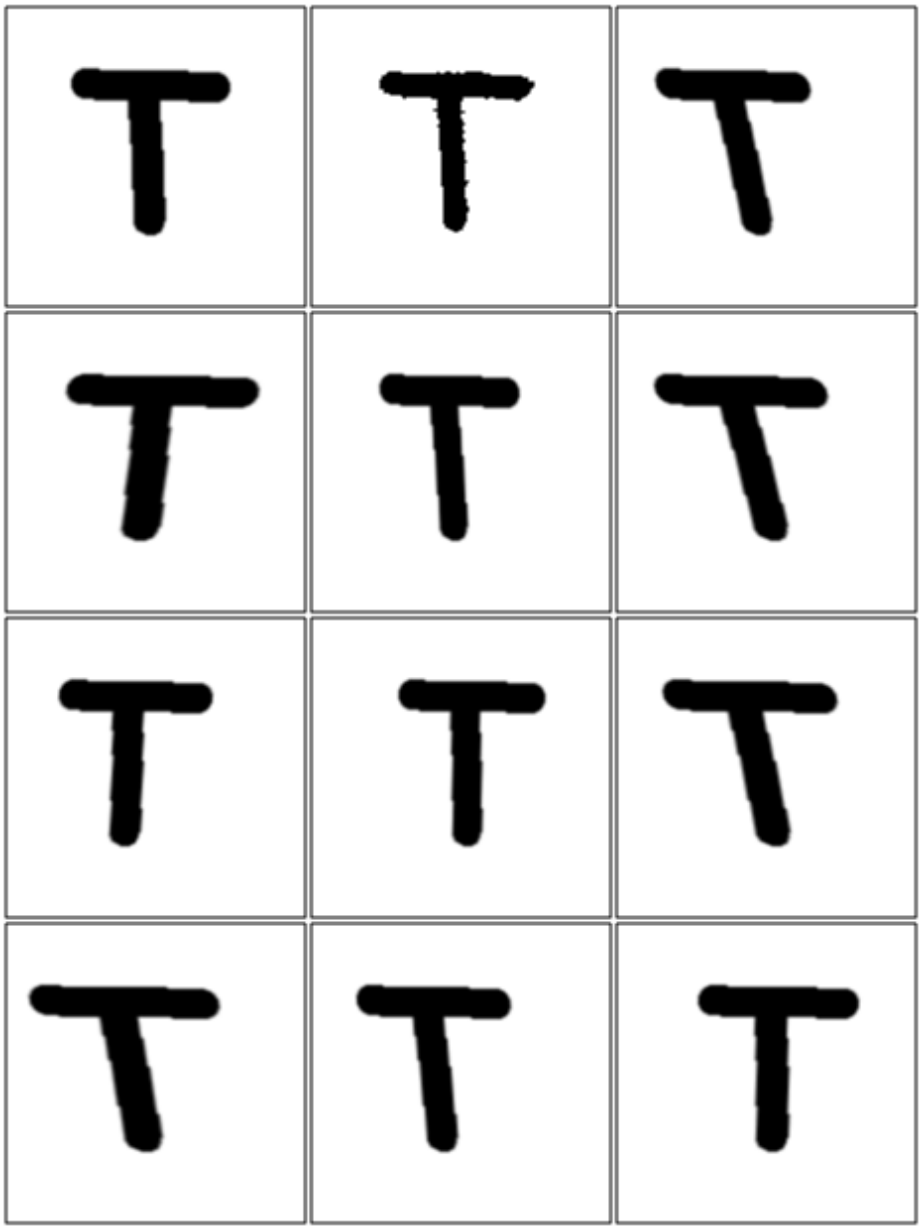
W celu zachowania informacji na temat rodzaju znaku oraz czcionki i jej wariantu zastosowano następujące nazewnictwo plików z wydzielonymi znakami:

ZNAK_WARIANT_CZCIONKA_MODYFIKACJA_NrMODYFIKACJI.tif,

Gdzie:

- ZNAK przyjmuje wartość z zakresu A-Z0-9,
- WARIANT to odpowiednio:
  - n – dla czcionki normalnej,
  - b – pogrubionej,
  - i – kursywy,
  - m – połączeniu kursywy z pogrubieniem,
- CZIONKA to odpowiednio:
  - 1 – Arial,
  - 2 – Century,
  - 3 – Comic Sans,
  - 4 – Tahoma,
  - 5 – Terminal,
  - 6 – Times New Roman,
  - 7 – Verdana,
- MODYFIKACJA to odpowiednio:
  - n – szum,
  - w – przekształcenie afiniczne,
  - o – oryginalna litera,
  - NrMODYFIKACJI przyjmuje wartość od 0 do 9 i występuje tylko wtedy, gdy dana litera zostało poddana jednemu z przekształceń afinicznych.

Przygotowany według powyższego opisu zbiór 8928 liter przetworzono programem do wyznaczania cech znaków.



Rysunek 9. Przykład zaszumienia i zniekształceń litery T (pogrubiony Comic Sans). Odpowiednio znak oryginalny, zaszumiony i poddany 10 transformacjom.

2.2 Cechy

2.2.1 Znaczenie cech

Najważniejszą i zarazem najtrudniejszą czynnością podczas tworzenia klasyfikatorów jest odpowiednie dobranie zestawu cech dla konkretnego zadania klasyfikacji. Wybór odpowiednich cech wymaga dobrej znajomości klasyfikowanych obiektów i w dużej mierze zależy od osoby przygotowującej zbiór[1] [10][11][16]. Cechy powinny być na tyle szczegółowe, by uchwycić charakterystyczne właściwości konkretnego obiektu a zarazem na tyle ogólne by dobrze opisywały wszystkie klasyfikowane obiekty. Zbyt mała ich ilość nie pozwoli na odpowiednie przydzielenie obiektu do oczekiwanej klasy zaś zbyt duża wydłuży czas szkolenia klasyfikatora, czas jego działania a także może pogorszyć jego umiejętność generalizowania [6]. Każda cecha powinna być możliwa do wyznaczenia dla każdego klasyfikowanego obiektu, jej brak może znacząco wpłynąć na skuteczność klasyfikacji danego obiektu a nawet całej klasy obiektów (jeżeli wiele z cech nie zostanie wyznaczonych). Poprawność wybranego zbioru cech można sprawdzić dopiero po wytrenowaniu klasyfikatora na zbiorze uczącym i przetestowaniu go na zbiorze testowym. Jeżeli wyniki nie są zadowalające należy zweryfikować wybrane cechy i wprowadzić poprawki lub też wyznaczyć inny zbiór cech.

2.2.2 Wybór cech

W przeprowadzonych badaniach zastosowano trzy różne zestawy cech. Ilość zestawów nie była określona arbitralnie, lecz ustaliła się w trakcie prowadzenia badań. Pierwsze dwa zestawy nie dawały wystarczająco dobrych wyników, trzeci zaś okazał się wystarczający. Skuteczność klasyfikacji przy użyciu każdego z nich opisano w części praktycznej, w tym rozdziale skupiono się zaś jedynie na sposobie ich wyznaczania.

2.2.2.1 Zestaw pierwszy – Momenty Hu

Na zestaw ten składało się 7 momentów Hu. Momenty te z definicji są niezależne od translacji, rotacji i skalowania obrazu. Jeżeli okazałyby się wystarczającym zestawem cech w zadaniu klasyfikacji znaków alfanumerycznych pozwoliłyby bardzo uprościć proces

[1] Należy wspomnieć o istnieniu metod umożliwiających częściową automatyzację doboru cech jednakże wymagają one również ingerencji człowieka i nie są omawiane w niniejszej pracy.

segmentacji tekstu przed klasyfikacją. Momenty Hu wyznacza się w oparciu o momenty centralne, które dla obrazu cyfrowego oblicza się następującym wzorem [20]:

$$\mu_{pq} = \sum_x \sum_y (x - x')^p \, (y - y')^q f(x, y) \, , \qquad (2.2.2.1.1)$$

gdzie f(x,y) jest obrazem, $x' = \frac{M_{10}}{M_{00}}$, $y' = \frac{M_{10}}{M_{00}}$, a M_{ij} jednym z momentów zwykłych dla obrazu cyfrowego w skali szarości wyrażonych równaniem:

$$M_{ij} = \sum_x \sum_y x^i y^i I(x, y), \qquad (2.2.2.1.2)$$

Kolejne momenty Hu opisują wzory:

$$I_1 = \eta_{20} + \eta_{02}, \qquad (2.2.2.1.2)$$

$$I_2 = (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2, \qquad (2.2.2.1.3)$$

$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2, \qquad (2.2.2.1.4)$$

$$I_4 = (\eta_{30} - 3\eta_{12})^2 + (\eta_{21} + \eta_{03})^2, \qquad (2.2.2.1.5)$$

$$I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} - \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} - \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2], \qquad (2.2.2.1.6)$$

$$I_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} - \eta_{12})(\eta_{21} - \eta_{03}), \qquad (2.2.2.1.7)$$

$$I_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2], \qquad (2.2.2.1.8)$$

Momenty Hu wyznaczone zostały przy użyciu funkcji dostarczonej przez OpenCV `cvGetHuMoments`.

2.2.2.2 Zestaw drugi – cechy geometryczne

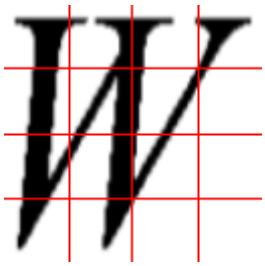
Na zestaw ten składało się 11 cech opisujących kształty geometryczne klasyfikowanych znaków, były to kolejno [2]:

1. Liczba zamkniętych konturów (otworów), które posiadał dany znak. Dla przykładu A posiadała jedną, B dwie a 1 zero - `cvFindContours`
2. Obwód zewnętrzny litery - `cvContourPerimeter`
3. Obwód otworów litery - `cvContourPerimeter`
4. Iloraz wysokości i szerokościki minimalnego prostokąta opisującego daną literę - `cvMinAreaRect2`
5. Iloczyn wysokości i szerokości minimalnego prostokąta opisującego daną literę.
6. Wysokość prostokąta opisującego daną literę.
7. Szerokość prostokąta opisującego daną literę.
8. Kąt minimalnej elipsy opisującej daną literę.

9. Iloczyn wysokości i szerokości minimalnej elipsy opisującej daną literę - `cvFitEllipse2`
10. Pole convex-hull - `cvConvexHull2`
11. Obwód convex-hull - `cvConvexHull2`

2.2.2.3 Zestaw trzeci – cechy gęstościowe (liczba zapalonych pikseli)

W celu wyznaczenia tego zestawu cech obraz ze znakiem zamieniany był na obraz czarno-biały i dzielony na NxM prostokątów. N określała liczbę prostokątów w poziomie a M w pionie. Dla każdego obszaru ograniczonego wyznaczonym prostokątem zliczane były wszystkie czarne piksele a następnie zapisywane jako wartości kolejnych cech. W ten sposób zmieniając parametry N i M można było generować różne zestawy cech.



Rysunek 10. Przykład podziału 4x4 dla litery W.

2.2.3 Ekstrakcja cech

Opisane powyżej zestawy cech wyznaczono, dla każdego znaku z przygotowanego wcześniej zbioru danych, przy użyciu programu Ekstrakcja Cech. Program wczytywał po kolei każdy obraz z zapisanym znakiem, dokonywał pomiaru jego cech a następnie zapisywał w pliku wynikowym wiersz zawierający klasę znaku, informacje o użytej czcionce, jej wariancie i zastosowanych modyfikacjach oraz wszystkie wyznaczone wartości oddzielone przecinkami.

Przykład drugiego zestawu cech dla litery A zapisanej pogrubioną czcionką Arial w oryginalnej postaci:
A,b,1,o, ,1,427.078,122.426,1.08654,11752,114,105,180.568,9491.92,7412,367.709

Proces wyznaczania cech dzielił się na 4 etapy. Wczytanie obrazu zawierającego znak i wyszukanie na nim konturów (ten sam mechanizm, który został użyty podczas tworzenia zbioru danych) w celu zlokalizowania znaku.

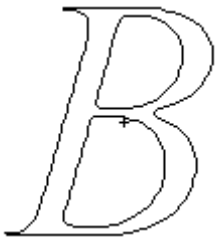
Rysunek 11. Wczytany obraz litery B.

Rysunek 12. Wykryty kontur litery B.

Utworzenie okna o rozmiarze 128x128 pikseli, proporcjonalne przeskalowanie znaku tak, aby jego dłuższy bok (szerokość lub wysokość) wynosił 122 pikseli i umieszczenie znaku dokładnie w środku okna.

Rysunek 13. Litera B po przeskalowaniu i umieszczeniu w środku okna 128x128.

Wyszukanie konturów na poprzednio przygotowanym oknie.



Rysunek 14. Kontur litery B po detekcji w oknie 128x128.

Wyznaczenie cech dla znalezionego konturu.

2.2.4 Otrzymane zbiory

Przy pomocy opisanych powyżej technik stworzono następujące zbiory:

1. Niezmienniki Hu (7 cech)
2. Niezmienniki Hu i otwory (8 cech)
3. Cechy geometryczne bez otworów (10 cech)
4. Cechy geometryczne i otwory (11 cech)
5. Cechy geometryczne i otwory wraz z niezmiennikami Hu (18 cech)
6. Cechy gęstościowe – podział 3x3 (9 cech)
7. Cechy gęstościowe – podział 4x4 (16 cech)
8. Cechy gęstościowe – podział 4x5 (20 cech)
9. Cechy gęstościowe – podział 5x5 (25 cechy)
10. Cechy gęstościowe – podział 6x6 (36 cech)
11. Cechy gęstościowe – podział 7x7 (49 cech)
12. Cechy gęstościowe – podział 8x8 (64 cechy)
13. Cechy gęstościowe – podział 9x9 (81 cech)
14. Cechy gęstościowe – podział 13x13 (cech)
15. Cechy gęstościowe – podział 16x16 (256 cech)

2.2.5 Przygotowanie zbioru uczącego i testowego

Szkolenie i testowanie klasyfikatora odbywa się w oparciu o dwa zbiory danych – uczący i testowy. Podczas testowania klasyfikatorów użyto dwóch typów zbiorów testujących. Pierwszy z nich stworzony został na podstawie zbioru danych, z którego powstał zbiór uczący (przy zachowanym założeniu rozłączności zbioru uczącego i testowego). W skład drugiego wchodziły znaki zapisane czcionkami, które nie występowały w zbiorze uczącym.

Najczęściej na zbiór uczący składa się około 70% zbioru danych, pozostałe elementy trafiają do zbioru testowego. Bardzo istotne jest odpowiednie rozdzielenie danych pomiędzy oba zbiory. Należy unikać sytuacji, w których znaki poszczególnych klas nie są w miarę równomiernie rozdzielone między zbiory uczące i testowe. Umieszczenie 90% znaków A w zbiorze testowym a jedynie 10% w zbiorze uczącym da w efekcie klasyfikator słabo rozpoznający litery A (za mało danych, aby wystarczająco opisać daną klasę). Odwrotna sytuacja nie da za to możliwości dobrego przetestowania skuteczności klasyfikatora (za mało

danych do testowania). Aby zapewnić zadowalające rozłożenie znaków między oba zbiory program Przygotuj Dane – służący do generowania zbiorów uczących i testowych, wykonywał następujące czynności. Dzielił zbiór danych na 36 podzbiorów. Każdy podzbiór składał się ze znaków danej klasy i miał tyle samo elementów (w przypadku głównego zbioru danych składającego się z 8928 liter było to 248 elementów). Z każdego z podzbiorów wybierał losowo 68% wszystkich znaków (169) i umieszczał je w zbiorze uczącym, pozostałe 32% (79) trafiało do zbioru testowego. W efekcie powyższych działań otrzymywano zbiór uczący składający się z 6084 znaków i zbiór testowy obejmujący 2844 znaków.

Należy zauważyć, że każda z wygenerowanych par zbiorów uczących i testowych była nieznacznie inna a powstałe w oparciu o nią klasyfikatory mogły dawać trochę inne wyniki (z przeprowadzonych testów wynika, że skuteczność może wahać się od około -0.5% do około +0,5%).

Ze względów praktycznych program łączył oba zbiory w jeden, na początku umieszczając cały zbiór uczący a po nim zbiór testowy. Programy szkolące klasyfikatory pierwsze 6084 wierszy traktowały jako dane uczące zaś kolejne 2844 jako dane testowe.

2.2.6 Normalizacja danych

Wartości liczbowe cech wyznaczonych dla używanych w badaniach zbiorów danych należały do przedziału od -1 do 1718. Podczas szkolenia klasyfikatora zbyt duża rozpiętość liczbowa danej cechy może powodować zmniejszenie jej przydatności w procesie klasyfikacji. W przypadku, gdy jedna z cech przyjmuje wartości znacznie większe niż pozostałe może dojść do sytuacji, w której znaczenie tej cechy fałszywie wzrośnie i zmniejszy wagę pozostałych, teoretycznie równie ważnych lub nawet ważniejszych. Przykładem może być porównanie ilości otworów danego znaku z jego wysokością. Praktycznie wszystkie znaki miały tę samą wysokość, która wynosiła 122 piksele, przez co była cechą mało ważną. Ilość otworów była zaś cechą bardzo istotną, ale przyjmującą wartość z zakresu od 0 do 2. Porównując obie cechy jedynie na podstawie ich wartości liczbowej wysokość zdominowałaby ilość otworów, co byłoby oczywiście sytuacją niepożądaną. Jednym ze sposobów uniknięcia powyższej sytuacji jest określenie wagi każdej z cech i przekazanie tej informacji klasyfikatorowi. Niestety implementacja większości klasyfikatorów w OpenCV nie daje takiej możliwości. Dodatkowo podejście takie wymagało ocenienia przydatności poszczególnych cech względem innych, co często jest zadaniem bardzo trudnym. Drugim sposobem, użytym w przeprowadzonych badaniach, była normalizacja wartości każdej z cech.

Normalizacja polega na odwzorowaniu jednego zbioru wartości w drugi zbiór wartości z zakresu od 0 do 1. Minimalna wartości odwzorowywanego zbioru zostaje zamieniona na 0, maksymalna na 1 a pozostałe wartości są odpowiednio przeliczane i przypisywana jest im wartość z zakresu (0,1).

Program Przygotuj Dane normalizował oddzielnie zbiór uczący i testowy. Najpierw dokonywał normalizacji zbioru uczącego, normalizując kolejno wszystkie cechy według wzoru:

$$C_i = \frac{c_i + |\min \quad C|}{|\min \quad C| + |\max \quad C|} \qquad (2.2.6.1),$$

gdzie C_i to wartość *i*-tej cechy w wektorze cech a $|\min \quad C|$ i $|\max \quad C|$ to odpowiednio wartości bewzględne minimalnej i maksymalnej wartości spośród wszystkich *i*-tych cech dla całego normalizowanego zbioru. Dzięki powyższej operacji wszystkie wartości cech były zawsze większe lub równe 0 i można było dokonać dalszej normalizacji. Kolejnym krokiem było wyznaczenie nowej wartość poprzez wyliczenie ilorazu pobranej wartości z sumą maksymalnej wartości cechy wyznaczonej wcześniej i znalezionej wartości absolutnej. Po znormalizowaniu zbioru uczącego dokonywał normalizacji zbioru testowego w oparciu o minimalne i maksymalne wartości cech wyznaczone podczas normalizacji zbioru uczącego. Takie podejście wymuszone było faktem teoretycznej nieznajomości wszystkich elementów zbioru testowego, który musiał być rozpatrywany jako pojedyncze znaki a nie ich zbiór.

Przykład wektora cech składającego się z momentów Hu,
przed normalizacją:
A,0.279452,0.00310253,0.0124125,0.000182388,-2.72596e-007,1.01494e-005,3.16225e-008
po normalizacji:
A,0.063,0,0.002,0,0.006,0.005,0.839

3 Testy klasyfikatorów

3.1 Klasyfikator Bayesa

Klasyfikator Bayesa w OpenCV zaimplementowany został w klasie `CvNormalBayesClassifier`. Ponieważ nie można go było konfigurować żadnymi parametrami, w celu utworzenia działającego klasyfikatora wystarczyło jedynie utworzyć obiekt klasy i uruchomić dla niego metodę `train()`. Testowanie stworzonego klasyfikatora przebiegało w następujący sposób - kolejne znaki ze zbioru uczącego i testowego podawane były jako argument metody predict(), a wyniki klasyfikacji porównywane były z klasami badanych znaków. Poniżej znajduje się przykładowy kod.

```
CvMat* cvmZbiorDanych;
CvMat* cvmZbiorUczacy;
CvMat* cvmKlasyZbioruDanych;
CvMat* cvmKlasyZbioruUczacego;
CvMat* cvmKlasyZbioruTestowego;

WczytajDane(&cvmZbiorDanych, &cvmKlasyZbioruDanych);

//tworzenie wektora klas zbioru uczacego
cvmKlasyZbioruUczacego = cvCreateMat(iWielkoscZbioruUczacego, 1, CV_32F);

for( i=0; i<iWielkoscZbioruUczacego; i++){
      float* fKlasa = (float*)( cvmKlasyZbioruUczacego->data.ptr + i*
      cvmKlasyZbioruUczacego->step);
      fKlasa[0] = cvmKlasyZbioruDanych->data.fl[i];
}

//tworzenie zbioru uczącego
cvGetRows(cvmZbiorDanych, &cvmZbiorUczacy, 0, iWielkoscZbioruUczacego);

//mierzenie czasu szkolenia
int iSzkolenieStart = clock();

//szkolenie
CvNormalBayesClassifier cvBayes = new cvBayes();
cvBayes.train(&cvmZbiorUczacy, cvmKlasyZbioruUczacego);

int iSzkolenieKoniec = clock();
int iCzasSzkolenia = iSzkolenieKoniec – iSzkolenieStart;

//testowanie klasyfikatora
int iPoprawnaKlasyfikacjaUczacy = 0;
int iPoprawnaKlasyfikacjaTestowy = 0;
iTestowanieStart = clock();

for( i=0; i< cvmZbiorDanych->rows; i++{
      CvMat cvmObiekt;
      cvGetRow(cvmZbiorDanych, &cvmObiekt, i);
```

```
        float fKlasa = cvBayes.predict(&cvmObiekt);

        int iKlasyfikacja = fabs(fKlasa – cvmKlasyZbioruDanych->data.fl[i]) <
        FLT_EPSILON ? 1 : 0;

        if(i < iWielkoscZbioruUczacego)
              iPoprawnaKlasyfikacjaUczacy += iKlasyfikacja;
        else
              iPoprawnaKlasyfikacjaTestowy += iKlasyfikacja;
}

int iTestowanieKoniec = clock();
int iCzasTestowania = iTestowanieKoniec – iTestowanieStart;

double dSkutecznoscDlaUczacego = (double) iPoprawnaKlasyfikacjaUczacy /
(double) iWielkoscZbioruUczacego * 100;
double dSkutecznoscDlaTestowego = (double) iPoprawnaKlasyfikacjaTestowy /
(double)(cvmZbiorDanych->rows -  iWielkoscZbioruUczacego) * 100;
```

Testowanie klasyfikatora Bayesa ograniczyło się jedynie do wytrenowania go dla różnych zbiorów danych i dokonania na tych zbiorach klasyfikacji. Skuteczność klasyfikacji danych uczących nie osiągnęła dla żadnego zbioru wartości 100%, lecz była zawsze większa od skuteczność klasyfikacji zbioru testowego i przeważnie większa od niej o około 0,5 do 0,9 punktu procentowego. Najlepsze wyniki klasyfikacji otrzymano dla zbioru 36 cech gęstościowych przy podziale *6x6*. Średni czas uczenia klasyfikatora dla tego zbioru wynosił 0,208 s, czas klasyfikacji wszystkich znaków ze zbioru 3,738 s a pojedynczego znaku 0,0004 s. Dla wybranego zbioru normalizacja danych poprawiła wyniki klasyfikacji jednak w bardzo nieznacznym stopniu – mediana klasyfikacji zbioru testowego dla danych znormalizowanych to 99,4% a dla nieznormalizowanych 99,3% czyli wynik poprawił się o 0,1 punktu procentowego. Najgorsza klasyfikacja zbioru testowego dla danych gęstościowych 6x6 wynosiła 99,1 % a najlepsza 99,7% co przekładało się na błędną klasyfikację 27 i 9 znaków spośród 2884. Dla mediany skuteczności klasyfikacji równej 99,4% błędna klasyfikacja wystąpiła dla 16 znaków. Taką skuteczność klasyfikacji klasyfikator osiągnął dla wariantów zbiorów numer 5, 6 i 8. Błędnie sklasyfikowane znaki dla zbioru numer 6 znajdują się w Tabeli 1 poniżej.

Tabela 1. Błędnie sklasyfikowane znaki i ich poprawne klasy z 6 zestawu 6x6 cech gęstościowych.

Znak	Wynik klasyfikacji	Opis próbki	Obraz próbki
0	O	0_b_3_w_2	
1	I	1_m_6_n	
1	R	1_b_4_w_2	
1	R	1_m_7_w_6	
1	R	1_b_7_w_2	
2	R	2_b_3_w_2	
5	B	5_b_3_w_2	
5	S	5_b_7_w_2	
6	B	6_b_7_w_2	
A	K	A_b_4_w_0	
F	B	F_b_6_w_0	
J	6	J_b_1_w_6	
J	1	J_n_5_w_0	
O	0	O_b_4_w_1	
P	R	P_b_6_w_0	
Z	1	Z_n_2_n	

Poniżej znajduje się uproszczona tabela wyników (Tabela 2) zawierająca średnie i mediany dla każdego zbioru danych zamiast wszystkich dziesięciu wyników, „n” oznacza dane znormalizowane, „un” nieznormalizowane.

Tabela 2. Wyniki testowania klasyfikatora na różnych zbiorach danych. Każdy wiersz przedstawia średnie czasy uczenia, klasyfikacji i klasyfikacji jednej litery, mediany skuteczności klasyfikacji zbioru uczącego i testowego oraz maksymalną i minimalną skuteczność klasyfikacji zbioru testowego dla 10 zestawów danego typu cech.

Dane	Mediana skuteczności klasyfikacji zbioru uczącego [%]	Min. skuteczność klasyfikacji zbioru testowego [%]	Max. skuteczność klasyfikacji zbioru testowego [%]	Mediana skuteczność klasyfikacji zbioru testowego [%]	Śr. czas uczenia [s]	Śr. czas klasyfikacji [s]	Śr. Czas klasyfikacji jednej litery [ms]
7hu n	30,5	27,6	35,0	28,4	0,013	0,909	0,102
7hu un	37,7	33,8	38,1	36,1	0,009	0,910	0,102
7hu, dziury n	41,2	38,7	45,0	40,0	0,009	0,927	0,104
7hu, dziury un	47,5	43,9	48,1	46,0	0,011	0,924	0,103
10 cech ksztalu n	89,5	86,7	88,9	88,3	0,016	1,072	0,120
10 cech ksztalu un	87,5	85,2	88,3	86,3	0,017	1,059	0,119
10 cech ksztaltu, dziury n	92,3	89,9	91,6	90,9	0,019	1,144	0,128
10 cech ksztaltu, dziury un	90,5	87,5	91,4	89,2	0,017	1,148	0,129
10 cech ksztaltu, dziury, 7hu n	96,7	94,9	96,4	95,6	0,038	1,637	0,183
10 cech ksztaltu, dziury, 7hu un	95,9	93,4	95,5	94,7	0,037	1,645	0,184
3x3 n	95,7	94,3	95,4	94,8	0,013	0,977	0,109
3x3 un	95,5	93,6	95,0	94,6	0,014	0,974	0,109
4x4 n	98,0	96,8	98,0	97,6	0,047	1,417	0,159
4x4 un	95,5	94,3	95,5	95,0	0,044	1,422	0,159
4x5 n	99,4	98,9	99,5	99,2	0,060	1,736	0,194
4x5 un	98,8	98,0	99,5	98,7	0,064	1,744	0,195
5x5 n	99,4	98,6	99,4	98,9	0,094	2,249	0,252
5x5 un	99,3	97,2	99,3	98,8	0,092	2,255	0,253
6x6 n	99,9	99,1	99,7	99,4	0,208	3,738	0,419
6x6 un	99,9	98,6	99,6	99,3	0,208	3,753	0,420
7x7 n	100,0	98,7	99,5	99,1	0,409	6,303	0,706
7x7 un	99,6	97,9	98,8	98,5	0,417	6,308	0,707
8x8 n	100,0	98,2	99,2	99,0	0,750	9,747	1,092
8x8 un	99,9	97,4	98,8	98,3	0,748	9,725	1,089
9x9 n	2,8	2,8	2,8	2,8	1,306	14,788	1,656
9x9 un	100,0	97,0	98,4	97,7	1,305	14,814	1,659
13x13 n	2,8	2,8	2,8	2,8	9,153	57,817	6,476
13x13 un	2,8	2,8	2,8	2,8	9,085	58,361	6,537
16x16 n	2,8	2,8	2,8	2,8	25,933	131,572	14,737
16x16 un	2,8	2,8	2,8	2,8	25,905	128,614	14,406

Dane znormalizowane dla wszystkich zbiorów cech prócz *7hu* i *7hu*, *otwory* dały nieznacznie lepsze wyniki niż nieznormalizowane. Dodanie informacji o otworach poprawiło skuteczność klasyfikacji dla zbiorów *7hu* (o około 10 punktów procentowych), *10 cech kształtu* (o około 3 punkty procentowe). Same *7hu* jak i *10 cech kształtu* okazało się niewystarczające do poprawnego rozpoznawania znaków. Dopiero połączenie ze sobą obu zbiorów i dodanie informacji o otworach pozwoliło uzyskać lepsze wyniki, choć wciąż niewystarczające (95,6% zbioru testowego to 125 błędnych klasyfikacji). Dla cech gęstościowych otrzymano znacznie lepsze wyniki niż dla pozostałych zbiorów cech.

Najmniejszy przyjęty podział (*3x3* czyli 9 cech) zapewnił wyniki porównywalne z cechami *10 cech ksztaltu, otwory, 7hu* (w sumie 18 cech) a klasyfikator szkolił się trzy razy szybciej, testowanie zaś trwało prawie dwa razy krócej. Zadowalająca skuteczność klasyfikacji pojawia się już przy podziale *4x5* i nieznacznie zmieniała się aż do przedziału *6x6* gdzie była największa. Dla podziałów *7x7* i *8x8* zaczęła powoli spadać. Dla podziałów *9x9, 13x13* i *16x16* wyniosła tyle samo, czyli 2,8% co zapewne było wynikiem błędu działania normalnego klasyfikatora Bayesa w OpenCV, spowodowanego zbyt dużą ilością cech i niemożliwością znalezienia prawidłowości między nimi. Ciekawym przypadkiem jest nieznormalizowany zbiór *9x9* cech gęstościowych, dla którego skuteczność klasyfikacji zbioru testowego wyniosła 97,7% chociaż dla tego samego zbioru znormalizowanych danych wynosiła ona 2,8%. Wraz ze wzrostem liczby cech zawsze wzrastał czas szkolenia klasyfikatora, lecz nie zawsze wzrastała skuteczność klasyfikacji.

Dla najlepiej klasyfikowanego zbioru cech gęstościowych *6x6* wykonano testy klasyfikatora na zbiorach testowych zapisanych innymi czcionkami niż zbiory uczące – wyniki ujęto w Tabeli 3 poniżej.

Tabela 3. Wyniki klasyfikacji dodatkowego zbioru testowego 6x6 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji
1	dane 6x6 testowe.cechyn	88,4
2	dane 6x6 testowe.cechyn	88,6
3	dane 6x6 testowe.cechyn	89,5
4	dane 6x6 testowe.cechyn	87,4
5	dane 6x6 testowe.cechyn	88,1
6	dane 6x6 testowe.cechyn	87,9
7	dane 6x6 testowe.cechyn	88,4
8	dane 6x6 testowe.cechyn	88,2
9	dane 6x6 testowe.cechyn	87,9
10	dane 6x6 testowe.cechyn	88,6

Minimalna skuteczność klasyfikacji wynosiła 87,4%, maksymalna 89,5% a mediana 88,3% co skutkowało odpowiednio 73, 60 i 67 źle sklasyfikowanymi znaki spośród 576. Mediana skuteczności klasyfikacji była mniejsza od skuteczność klasyfikacji podstawowych zbiorów testowych o 11,1 punktu procentowego. Dla danych nieznormalizowanych mediana skuteczności wynosiła 87,9%.

Ostatnim wykonanym testem normalnego klasyfikatora Bayesa była próba stworzenia trzech współdziałających ze sobą klasyfikatorów w celu dokonania poprawnej klasyfikacji. Poniżej znajdują się tabele (Tabela 4, Tabela 5, Tabela 6) przedstawiające wyniki kolejno dla klasyfikatora podejmującego decyzję, czy dany znak jest litera czy cyfrą, klasyfikatora klasyfikującego litery i klasyfikatora klasyfikującego cyfry.

Tabela 4. Wyniki klasyfikacji dla klasyfikatora rozróżniającego cyfry od liter na zbiorze 6x6 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 6x6.cechyn	86,9	87,1
2	dane 6x6.cechyn	86,4	85,3
3	dane 6x6.cechyn	86,9	86,2
4	dane 6x6.cechyn	87,2	86,4
5	dane 6x6.cechyn	86,4	86,8
6	dane 6x6.cechyn	87,2	86,2
7	dane 6x6.cechyn	86,3	86,4
8	dane 6x6.cechyn	86,6	86,9
9	dane 6x6.cechyn	86,5	86,7
10	dane 6x6.cechyn	86,7	86,0

Tabela 5. Wyniki klasyfikacji dla klasyfikatora rozpoznającego cyfry na zbiorze 6x6 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 6x6 same cyfry.cechyn	100,0	99,5
2	dane 6x6 same cyfry.cechyn	100,0	99,6
3	dane 6x6 same cyfry.cechyn	100,0	99,7
4	dane 6x6 same cyfry.cechyn	100,0	99,5
5	dane 6x6 same cyfry.cechyn	100,0	99,4
6	dane 6x6 same cyfry.cechyn	100,0	100,0
7	dane 6x6 same cyfry.cechyn	100,0	99,9
8	dane 6x6 same cyfry.cechyn	100,0	99,6
9	dane 6x6 same cyfry.cechyn	100,0	99,6
10	dane 6x6 same cyfry.cechyn	100,0	99,9

Tabela 6. Wyniki klasyfikacji dla klasyfikatora rozpoznającego litery na zbiorze 6x6 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 6x6 same litery.cechyn	100,0	99,7
2	dane 6x6 same litery.cechyn	100,0	99,7
3	dane 6x6 same litery.cechyn	100,0	99,5
4	dane 6x6 same litery.cechyn	100,0	99,4
5	dane 6x6 same litery.cechyn	100,0	99,8
6	dane 6x6 same litery.cechyn	100,0	99,7
7	dane 6x6 same litery.cechyn	100,0	99,8
8	dane 6x6 same litery.cechyn	100,0	99,8
9	dane 6x6 same litery.cechyn	100,0	99,7
10	dane 6x6 same litery.cechyn	100,0	99,7

Minimalna skuteczność klasyfikacji dla klasyfikatora podejmującego decyzję wynosiła 85,3%, maksymalna 87,1%, a mediana 86,4%, przełożyło się odpowiednio na 424, 372 i 392 źle sklasyfikowanych znaków. Dla klasyfikatora liter minimalna skuteczność klasyfikacji dla zbioru testowego wynosiła 99,4%, maksymalna 99,8%, a mediana 99,7%, czyli odpowiednio 13, 4 i 6 źle sklasyfikowanych liter zaś dla klasyfikatora cyfr skuteczność minimalna wynosiła 99,4%, maksymalna 100%, a mediana 99,6%, w rezultacie czego źle sklasyfikowanych zostało odpowiednio 5, 0 i 3 cyfr. Pomimo bardzo dobrych wyników klasyfikatorów liczb i cyfr (średnio 10 źle sklasyfikowanych znaków) słaba skuteczność rozróżniania liczb od cyfr czyni powyższe rozwiązanie nieskutecznym.

W wyniku przeprowadzonych eksperymentów ustalono, iż najlepszym spośród przetestowanych zbiorów cech dla normalnego klasyfikatora Bayesa był zbiór cech gęstościowych *6x6*. Otrzymane dla niego wyniki klasyfikacji zbioru uczącego, testowego, oraz dodatkowego zbioru testowego wynosiły odpowiednio: 99,9%, 99,4% i 88,3%.

3.2 K najbliższych sąsiadów (k nearest neighbours)

Klasyfikator KNN w OpenCV zaimplementowany został w klasie `CvKNearest`. W celu utworzenia działającego klasyfikatora należało jedynie utworzyć obiekt klasy. Testowanie stworzonego klasyfikatora przebiegało w następujący sposób - kolejne znaki ze zbioru uczącego i testowego podawane były jako argument metody predict(), a wyniki klasyfikacji porównywane były z klasami badanych znaków. Poniżej znajduje się przykładowy kod.

```
CvMat* cvmZbiorDanych;
CvMat* cvmZbiorUczacy;
CvMat* cvmKlasyZbioruDanych;
CvMat* cvmKlasyZbioruUczacego;
CvMat* cvmKlasyZbioruTestowego;

WczytajDane(&cvmZbiorDanych, &cvmKlasyZbioruDanych);

//tworzenie wektora klas zbioru uczacego
cvmKlasyZbioruUczacego = cvCreateMat(iWielkoscZbioruUczacego, 1, CV_32F);

for( i=0; i<iWielkoscZbioruUczacego; i++){
      float* fKlasa = (float*)( cvmKlasyZbioruUczacego->data.ptr + i*
      cvmKlasyZbioruUczacego->step);
      fKlasa[0] = cvmKlasyZbioruDanych->data.fl[i];
}

//tworzenie zbioru uczącego
cvGetRows(cvmZbiorDanych, &cvmZbiorUczacy, 0, iWielkoscZbioruUczacego);
```

```
//mierzenie czasu szkolenia
int iSzkolenieStart = clock();

CvKNearest cvknKNN = CvKNearest(&cvmZbiorUczacy, cvmKlasyZbioruUczacego, 0,
false, K);

int iSzkolenieKoniec = clock();
int iCzasSzkolenia = iSzkolenieKoniec – iSzkolenieStart;

//testowanie klasyfikatora
int iPoprawnaKlasyfikacjaUczacy = 0;
int iPoprawnaKlasyfikacjaTestowy = 0;
CvMat* cvmNajblizsziSasiedzi = cvCreatMat(1, K, CV_32FC1);
CvMat* cvmOdlegloscSasiadow = cvCreatMat(1, K, CV_32FC1);
iTestowanieStart = clock();

for( i=0; i< cvmZbiorDanych->rows; i++{
      CvMat cvmObiekt;
      cvGetRow(cvmZbiorDanych, &cvmObiekt, i);

      float   fKlasa   =   cvknKNN.find_nearest(&cvmObiekt,   K,   0,   0,
      cvmNajblizsziSasiedzi, cvmOdlegloscSasiadow);

      int iKlasyfikacja = fabs(fKlasa – cvmKlasyZbioruDanych->data.fl[i]) <
      FLT_EPSILON ? 1 : 0;

      if(i < iWielkoscZbioruUczacego)
            iPoprawnaKlasyfikacjaUczacy += iKlasyfikacja;
      else
            iPoprawnaKlasyfikacjaTestowy += iKlasyfikacja;
}

int iTestowanieKoniec = clock();
int iCzasTestowania = iTestowanieKoniec – iTestowanieStart;

double dSkutecznoscDlaUczacego = (double) iPoprawnaKlasyfikacjaUczacy /
(double) iWielkoscZbioruUczacego * 100;
double dSkutecznoscDlaTestowego = (double) iPoprawnaKlasyfikacjaTestowy /
(double)(cvmZbiorDanych->rows -  iWielkoscZbioruUczacego) * 100;
```

Konfiguracja klasyfikatora kNN ograniczyła się jedynie do dobrania odpowiedniej ilości rozpatrywanych sąsiadów. Dla klasyfikatora minimalno-odległościowego z definicji k przyjęła wartość 1. Przeprowadzone badania na pierwszym zestawie danych wykazały, iż najskuteczniejsza klasyfikacja występuje dla k równego 3. Wraz ze wzrostem parametru k wzrastał czas klasyfikacji a skuteczność klasyfikacji malała. Dodatkowo zaimplementowany w OpenCV algorytm kNN podczas klasyfikacji nieznanego obiektu w przypadkach spornych (np. znalezionych 3 różnych sąsiadów, lub dwóch sąsiadów jednej klasy i dwóch sąsiadów drugiej klasy) pomijał najbliższego sąsiada i rozważał następnych. Błąd ten został wyeliminowany poprzez ignorowanie wyniku zwróconego przez metodę `find_nearest()` i samodzielną analizę wszystkich znalezionych sąsiadów i na jej podstawie wybranie poprawnej klasy.

Pierwszym testowanym typem klasyfikatora kNN był klasyfikator minimalno-odległościowy. Najlepsze wyniki klasyfikacji otrzymano dla znormalizowanego zbioru 256 cech gęstościowych przy podziale *16x16*. Średni czas uczenia klasyfikatora dla wybranego zbioru wynosił 18,7 ms, czas klasyfikacji wszystkich znaków ze zbioru 103,29 s a pojedynczego znaku 11,57 ms. Najgorsza klasyfikacja zbioru testowego dla danych gęstościowych *16x16* wynosiła 99,2 % a najlepsza 99,8% co przekładało się na błędną klasyfikację 24 i 7 znaków spośród 2884. Dla mediany skuteczności klasyfikacji równej 99,6% błędna klasyfikacja wystąpiła dla 12 znaków. Zbliżoną skuteczność klasyfikacji klasyfikator osiągnął dla wariantów zbiorów numer 1, 5 i 6. Kolejne źle sklasyfikowane znaki dla zbioru numer 6 zostały przedstawione w Tabeli 7.

Tabela 7. Błędnie sklasyfikowane znaki i ich poprawne klasy z 6 zestawu 16x16 cech gęstościowych dla klasyfikatora minimalno-odległościowego.

Znak	Wynik klasyfikacji	Opis próbki	Obraz próbki
0	O	n_3_w_2	
0	O	n_3_w_6	
1	I	b_6_w_5	
1	I	b_6_n	
E	F	m_6_n	
J	1	n_5_w_2	
J	1	n_5_w_5	
O	0	i_6_w_8	
O	0	b_1_w_3	
R	P	n_2_w_2	
T	I	b_6_n	

Poniżej znajduje się uproszczona tabela wyników dla klasyfikatora minimalno-odległościowego (Tabela 8) zawierająca średnie i mediany dla każdego zbioru danych zamiast wszystkich dziesięciu wyników, „n” oznacza dane znormalizowane, „un” nieznormalizowane.

Tabela 8. Wyniki testowania klasyfikatora minimalno-odległościowego na różnych zbiorach danych. Każdy wiersz przedstawia średnie czasy uczenia, klasyfikacji i klasyfikacji jednej litery, mediany skuteczności klasyfikacji zbioru uczącego i testowego oraz maksymalną i minimalną skuteczność klasyfikacji zbioru testowego dla 10 zestawów danego typu cech.

Dane	Mediana skuteczności klasyfikacji zbioru uczącego [%]	Min. skuteczność klasyfikacji zbioru testowego [%]	Max. skuteczność klasyfikacji zbioru testowego [%]	Mediana skuteczność klasyfikacji zbioru testowego [%]	Śr. czas uczenia [ms]	Śr. czas klasyfikacji [s]	Śr. Czas klasyfikacji jednej litery [ms]
7hu n	60,7	42,4	53,8	44,6	1,5	5,681	0,636
7hu un	100,0	59,0	62,1	60,5	1,6	5,672	0,635
7hu, otwory n	88,8	56,2	68,2	63,0	3,2	4,645	0,520
7hu, otwory un	100,0	69,0	73,0	71,4	0	4,644	0,520
10 cech ksztaltu n	100,0	78,4	81,0	80,5	3,1	6,339	0,710
10 cech ksztaltu un	100,0	61,6	63,7	62,9	3,2	6,325	0,708
10 cech ksztaltu, otwory n	100,0	83,1	85,4	83,8	4,8	7,227	0,809
10 cech ksztaltu, otwory un	100,0	60,3	63,9	62,9	<< 1	7,236	0,810
10 cech ksztaltu, otwory, 7hu n	100,0	83,4	85,8	84,9	<< 1	9,409	1,054
10 cech ksztaltu, otwory, 7hu un	100,0	61,5	65,4	62,4	<< 1	9,427	1,056
3x3 n	100,0	93,6	94,5	93,9	<< 1	5,669	0,635
3x3 un	100,0	94,0	95,0	94,5	6,2	5,661	0,634
4x4 n	100,0	97,0	98,0	97,7	1,6	7,755	0,869
4x4 un	100,0	97,1	98,0	97,8	4,6	7,745	0,868
4x5 n	100,0	97,5	98,6	98,3	1,6	9,320	1,044
4x5 un	100,0	97,7	98,7	98,5	1,5	9,310	1,043
5x5 n	100,0	98,6	99,2	98,9	1,5	12,761	1,429
5x5 un	100,0	98,5	99,3	99,0	3,1	12,764	1,430
6x6 n	100,0	98,9	99,5	99,4	1,5	16,317	1,828
6x6 un	100,0	98,9	99,5	99,3	3,2	16,308	1,827
7x7 n	100,0	99,1	99,7	99,4	3,2	22,075	2,473
7x7 un	100,0	99,1	99,6	99,4	3,1	22,081	2,473
8x8 n	100,0	99,1	99,8	99,5	4,7	27,442	3,074
8x8 un	100,0	99,1	99,8	99,5	6,3	27,455	3,075
9x9 n	100,0	99,3	99,7	99,5	7,9	34,795	3,897
9x9 un	100,0	99,2	99,7	99,5	1,6	34,781	3,896

13x13 n	100,0	99,3	99,5	99,4	11,0	69,781	7,816
13x13 un	100,0	99,1	99,4	99,3	11,2	69,716	7,809
16x16 n	100,0	99,2	99,8	99,6	18,7	103,292	11,569
16x16 un	100,0	99,1	99,7	99,5	17,2	103,334	11,574

Dla wszystkich zbiorów poza znormalizowanymi *7hu* i *7hu, otwory* skuteczność klasyfikacji danych uczących wynosiła 100%. Otrzymane wyniki klasyfikacji zbioru uczącego dla powyższych zbiorów wskazują na to, iż wektory cech różnych znaków po znormalizowaniu były takie same. Dla zbiorów zawierających dane kształtu normalizacja danych w znacznym stopniu poprawiła wyniki klasyfikacji zbioru testowego: o 17,6 punktu procentowego dla *dane 10 cech kształtu*, o 21 punktów procentowych dla *dane 10 cech ksztaltu, otwory* oraz 22,5 punktu procentowego dla *dane 10 cech ksztaltu, otwory, 7hu.* Dla zbiorów cech gęstościowych mniejszych od *6x6* dane nieznormalizowane przynosiły nieznacznie lepsze wyniki klasyfikacji zbioru testowego: 0,6 punktu procentowego dla 3x3 oraz 0,1 punktu procentowego dla *4x4*, *4x5*, *5x5*. Dla pozostałych zbiorów cech gęstościowych wyniki dla danych znormalizowanych były lepsze o 0,1 punktu procentowego. Normalizacja danych nie miała wpływu na czas szkolenia klasyfikatora ani na czas klasyfikacji. Zwiększanie liczby cech poprawiało wyniki klasyfikacji zbioru testowego jednocześnie wydłużając czas szkolenia i klasyfikacji.

Dla najlepiej klasyfikowanego zbioru cech gęstościowych *16x16* wykonano testy klasyfikatora minimalno-odległościowego na zbiorach testowych zapisanych innymi czcionkami niż zbiory uczące – wyniki ujęto w Tabeli 9 poniżej.

Tabela 9. Wyniki klasyfikacji dodatkowego zbioru testowego 16x16 cech gęstościowych dla klasyfikatora minimalno-odległościowego.

Zestaw	Dane	Skuteczność klasyfikacji [%]
1	dane 16x16.cechyn	94,6
2	dane 16x16.cechyn	94,6
3	dane 16x16.cechyn	95,1
4	dane 16x16.cechyn	94,1
5	dane 16x16.cechyn	95,3
6	dane 16x16.cechyn	95,3
7	dane 16x16.cechyn	95,1
8	dane 16x16.cechyn	95,1
9	dane 16x16.cechyn	95,8
10	dane 16x16.cechyn	95,1

Minimalna skuteczność klasyfikacji wynosiła 94,1%, maksymalna 95,8% a mediana 95,1% co skutkowało odpowiednio 34, 24 i 28 źle sklasyfikowanymi znaki spośród 576. Mediana skuteczności klasyfikacji była mniejsza od skuteczność klasyfikacji podstawowych zbiorów

testowych o 4,5 punktu procentowego. Dla danych nieznormalizowanych mediana skuteczności klasyfikacji wynosiła 94,3%.

Ostatnim wykonanym testem klasyfikatora minimalno-odległościowego była próba stworzenia trzech współdziałających ze sobą klasyfikatorów w celu dokonania poprawnej klasyfikacji. Poniżej znajdują się tabele (Tabela 10, Tabela 11, Tabela 12) przedstawiające wyniki kolejno dla klasyfikatora podejmującego decyzję, czy dany znak jest litera czy cyfrą, klasyfikatora klasyfikującego litery i klasyfikatora klasyfikującego cyfry.

Tabela 10. Wyniki klasyfikacji dla klasyfikatora minimalno-odległościowego rozróżniającego cyfry od liter na zbiorze 16x16 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 16x16.cechyn	100,0	99,7
2	dane 16x16.cechyn	100,0	99,5
3	dane 16x16.cechyn	100,0	99,7
4	dane 16x16.cechyn	100,0	99,6
5	dane 16x16.cechyn	100,0	99,6
6	dane 16x16.cechyn	100,0	99,5
7	dane 16x16.cechyn	100,0	99,6
8	dane 16x16.cechyn	100,0	99,7
9	dane 16x16.cechyn	100,0	99,5
10	dane 16x16.cechyn	100,0	99,8

Tabela 11. Wyniki klasyfikacji dla klasyfikatora minimalno-odległościowego rozpoznającego cyfry na zbiorze 16x16 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 16x16 same cyfry.cechyn	100,0	99,9
2	dane 16x16 same cyfry.cechyn	100,0	100,0
3	dane 16x16 same cyfry.cechyn	100,0	100,0
4	dane 16x16 same cyfry.cechyn	100,0	100,0
5	dane 16x16 same cyfry.cechyn	100,0	99,9
6	dane 16x16 same cyfry.cechyn	100,0	99,9
7	dane 16x16 same cyfry.cechyn	100,0	99,7
8	dane 16x16 same cyfry.cechyn	100,0	99,7
9	dane 16x16 same cyfry.cechyn	100,0	100,0
10	dane 16x16 same cyfry.cechyn	100,0	99,7

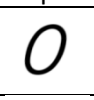
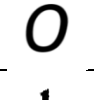
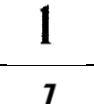
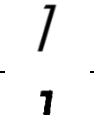
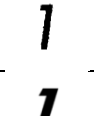
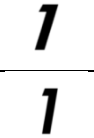
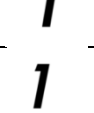
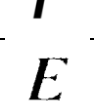
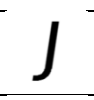
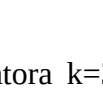
Tabela 12. Wyniki klasyfikacji dla klasyfikatora minimalno-odległościowego rozpoznającego litery na zbiorze 16x16 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 16x16 same litery.cechyn	100,0	99,9
2	dane 16x16 same litery.cechyn	100,0	99,8
3	dane 16x16 same litery.cechyn	100,0	99,8
4	dane 16x16 same litery.cechyn	100,0	99,9
5	dane 16x16 same litery.cechyn	100,0	99,9
6	dane 16x16 same litery.cechyn	100,0	99,8
7	dane 16x16 same litery.cechyn	100,0	100,0
8	dane 16x16 same litery.cechyn	100,0	99,9
9	dane 16x16 same litery.cechyn	100,0	99,7
10	dane 16x16 same litery.cechyn	100,0	99,8

Minimalna skuteczność klasyfikacji dla klasyfikatora podejmującego decyzję wynosiła 99,5%, maksymalna 99,8%, a mediana 99,6%, przełożyło się odpowiednio na 15, 5 i 12 źle sklasyfikowanych znaków. Dla klasyfikatora cyfr minimalna skuteczność klasyfikacji dla zbioru testowego wynosiła 99,7%, maksymalna 100%, a mediana 99,9%, czyli odpowiednio 2, 0 i 1 źle sklasyfikowanych cyfr zaś dla klasyfikatora liter skuteczność minimalna wynosiła 99,7%, maksymalna 100%, a mediana 99,8%, w rezultacie czego źle sklasyfikowanych zostało odpowiednio 6, 0 i 4 liter. Wyniki klasyfikatora liczb i cyfr były praktycznie idealne a klasyfikatora podejmującego decyzję bardzo wysokie. Powyższe rozwiązanie okazało się prawie tak samo skutecznym jak rozwiązanie podstawowe.

Drugim testowanym typem klasyfikatora kNN był klasyfikator dla k=3. Najlepsze wyniki klasyfikacji otrzymano dla znormalizowanego zbioru 81 cech gęstościowych przy podziale *9x9*. Średni czas uczenia klasyfikatora dla wybranego zbioru wynosił 96,9 ms, czas klasyfikacji wszystkich znaków ze zbioru 36,7 s a pojedynczego znaku 4,1 ms. Najgorsza klasyfikacja zbioru testowego dla danych gęstościowych *9x9* wynosiła 99,2 % a najlepsza 99,6% co przekładało się na błędną klasyfikację 24 i 10 znaków spośród 2884. Dla mediany skuteczności klasyfikacji równej 99,4% błędna klasyfikacja wystąpiła dla 18 znaków. Zbliżoną skuteczność klasyfikacji klasyfikator osiągnął dla wariantów zbiorów numer 2, 6 i 9. Kolejne źle sklasyfikowane znaki dla zbioru numer 9 zostały przedstawione w Tabeli 13.

Tabela 13. Błędnie sklasyfikowane znaki i ich poprawne klasy z 9 zestawu 9x9 cech gęstościowych dla klasyfikatora k=3.

Znak	Wynik klasyfikacji	Sąsiedzi	Opis próbki	Obraz próbki
0	O	0 O O	i_2_w_2	
0	O	O O O	i_2_w_6	
0	O	0 O O	m_7_w_7	
0	O	O O O	n_2_w_2	
0	O	O O O	n_3_w_2	
1	I	I I I	b_6_n	
1	I	I I I	i_2_w_2	
1	I	1 I I	m_2_n	
1	I	I I I	m_2_w_2	
1	I	1 I I	m_2_w_4	
1	I	1 I I	m_2_w_9	
E	F	F F F	m_6_n	
I	1	I 1 1	b_6_w_0	
J	3	J 3 3	i_7_w_2	
J	1	J 1 1	n_5_w_2	
Q	G	O G Q	i_2_n	

Poniżej znajduje się uproszczona tabela wyników dla klasyfikatora k=3 (Tabela 14) zawierająca średnie i mediany dla każdego zbioru danych zamiast wszystkich dziesięciu wyników, „n” oznacza dane znormalizowane, „un” nieznormalizowane.

Tabela 14. Wyniki testowania klasyfikatora k=3 na różnych zbiorach danych. Każdy wiersz przedstawia średnie czasy uczenia, klasyfikacji i klasyfikacji jednej litery, mediany skuteczności klasyfikacji zbioru uczącego i testowego oraz maksymalną i minimalną skuteczność klasyfikacji zbioru testowego dla 10 zestawów danego typu cech.

Dane	Mediana skuteczności klasyfikacji zbioru uczącego [%]	Min. skuteczność klasyfikacji zbioru testowego [%]	Max. skuteczność klasyfikacji zbioru testowego [%]	Mediana skuteczność klasyfikacji zbioru testowego [%]	Śr. czas uczenia [ms]	Śr. czas klasyfikacji [s]	Śr. czas Klasyfikacji jednej litery [ms]
7hu n	58,3	42,9	52,5	44,6	1,6	7,056	0,790
7hu un	87,4	55,6	57,6	56,7	<< 0	7,014	0,786
7hu, otwory n	82,1	55,7	64,6	61,3	3,1	5,159	0,578
7hu, otwory un	88,5	65,3	69,2	68,1	3,2	5,213	0,584
10 cech kształtu n	92,2	76,9	79,4	78,8	46,8	6,875	0,770
10 cech kształtu un	88,1	59,4	61,8	60,8	1,5	6,952	0,779
10 cech kształtu, otwory n	94,5	80,8	84,1	82,7	50	8,386	0,939
10 cech kształtu, otwory un	88,1	58,6	61,9	61,0	1,6	8,539	0,956
10 cech kształtu, otwory, 7hu n	94,9	81,3	84,4	83,2	9,3	10,242	1,147
10 cech kształtu, otwory, 7hu un	88,2	59,6	62,1	60,6	91,9	10,152	1,137
3x3 n	97,6	91,9	93,3	92,8	23,5	7,098	0,795
3x3 un	97,7	92,1	94,1	93,2	56,1	6,327	0,709
4x4 n	99,0	96,5	97,6	97,0	71,9	8,592	0,962
4x4 un	99,0	96,6	97,8	97,2	4,7	8,542	0,957
4x5 n	99,3	97,2	98,1	97,8	81,2	10,847	1,215
4x5 un	99,3	97,3	98,2	97,8	78,1	10,877	1,218
5x5 n	99,5	98,3	98,9	98,6	82,9	13,522	1,515
5x5 un	99,5	98,3	98,9	98,6	78,1	13,528	1,515
6x6 n	99,6	98,8	99,2	99,1	57,6	18,014	2,018
6x6 un	99,6	98,6	99,2	99,1	12,5	17,689	1,981
7x7 n	99,7	99,0	99,5	99,1	75,1	24,286	2,720
7x7 un	99,7	98,7	99,3	99,2	18,5	23,564	2,639
8x8 n	99,7	98,9	99,6	99,4	124,8	29,061	3,255
8x8 un	99,7	98,9	99,5	99,3	139,2	29,083	3,257
9x9 n	99,8	99,2	99,6	99,4	96,9	36,723	4,113
9x9 un	99,7	99,0	99,6	99,3	98,5	36,845	4,127
13x13 n	99,7	98,9	99,3	99,1	14,1	73,164	8,195
13x13 un	99,6	98,8	99,2	99,0	21,6	77,549	8,686
16x16 n	99,7	99,1	99,5	99,3	35,9	112,997	12,656
16x16 un	99,7	98,9	99,4	99,2	28,2	112,405	12,590

Skuteczność klasyfikacji zbioru uczącego dla żadnego ze zbiorów cech nie sięgnęła 100%. Dla zbiorów zawierających dane kształtu normalizacja danych w znacznym stopniu poprawiła wyniki klasyfikacji zbioru testowego: o 18 punktu procentowego dla *dane 10 cech kształtu,* o 21,7 punktu procentowego dla *dane 10 cech ksztaltu, otwory* oraz 22,6 punktu procentowego dla *dane 10 cech ksztaltu, otwory, 7hu.* Dane nieznormalizowane dla zbioru cech gęstościowych *3x3* polepszyły skuteczność klasyfikacji o 0,4 punktu procentowego a dla zbiorów *5x5* i *7x7* o 0,1 punktu procentowego zaś dla zbiorów *9x9, 13x13* i *16x16* pogorszyły wyniki o 0,1 punktu procentowego. Dla zbiorów cech gęstościowych *4x5* i *8x8* normalizacja nie wpłynęła na wyniki. Zwiększanie liczby cech stopniowo poprawiało skuteczność klasyfikacji aż do 81 cech.

Dla najlepiej klasyfikowanego zbioru cech gęstościowych *9x9* wykonano testy klasyfikatora k=3 na zbiorach testowych zapisanych innymi czcionkami niż zbiory uczące – wyniki ujęto w Tabeli 15 poniżej.

Tabela 15. Wyniki klasyfikacji dodatkowego zbioru testowego 9x9 cech gęstościowych dla klasyfiaktora k=3.

Zestaw	Dane	Skuteczność klasyfikacji [%]
1	dane 9x9.cechyn	96,0
2	dane 9x9.cechyn	96,2
3	dane 9x9.cechyn	95,3
4	dane 9x9.cechyn	96,0
5	dane 9x9.cechyn	95,5
6	dane 9x9.cechyn	95,3
7	dane 9x9.cechyn	96,5
8	dane 9x9.cechyn	96,4
9	dane 9x9.cechyn	96,2
10	dane 9x9.cechyn	95,3

Minimalna skuteczność klasyfikacji wynosiła 95,3%, maksymalna 96,5% a mediana 96,0% co skutkowało odpowiednio 27, 20 i 23 źle sklasyfikowanymi znaki spośród 576. Mediana skuteczności klasyfikacji była mniejsza od skuteczność klasyfikacji podstawowych zbiorów testowych o 3,9 punktu procentowego. Dla danych nieznormalizowanych mediana skuteczności klasyfikacji wynosiła 95,5%.

Ostatnim wykonanym testem klasyfikatora k=3 była próba stworzenia trzech współdziałających ze sobą klasyfikatorów w celu dokonania poprawnej klasyfikacji. Poniżej znajdują się tabele (Tabela 16, Tabela 17, Tabela 18) przedstawiające wyniki kolejno dla klasyfikatora podejmującego decyzję, czy dany znak jest litera czy cyfrą, klasyfikatora klasyfikującego litery i klasyfikatora klasyfikującego cyfry.

Tabela 16. Wyniki klasyfikacji dla klasyfikatora k=3 rozróżniającego cyfry od liter na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9.cechyn	99,8	99,4
2	dane 9x9.cechyn	99,7	99,7
3	dane 9x9.cechyn	99,8	99,4
4	dane 9x9.cechyn	99,8	99,5
5	dane 9x9.cechyn	99,8	99,5
6	dane 9x9.cechyn	99,8	99,6
7	dane 9x9.cechyn	99,8	99,4
8	dane 9x9.cechyn	99,7	99,4
9	dane 9x9.cechyn	99,8	99,6
10	dane 9x9.cechyn	99,8	99,5

Tabela 17. Wyniki klasyfikacji dla klasyfikatora k=3 rozpoznającego cyfry na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same cyfry.cechyn	100,0	100,0
2	dane 9x9 same cyfry.cechyn	100,0	100,0
3	dane 9x9 same cyfry.cechyn	100,0	100,0
4	dane 9x9 same cyfry.cechyn	100,0	100,0
5	dane 9x9 same cyfry.cechyn	100,0	100,0
6	dane 9x9 same cyfry.cechyn	100,0	100,0
7	dane 9x9 same cyfry.cechyn	100,0	100,0
8	dane 9x9 same cyfry.cechyn	100,0	100,0
9	dane 9x9 same cyfry.cechyn	100,0	100,0
10	dane 9x9 same cyfry.cechyn	100,0	99,9

Tabela 18. Wyniki klasyfikacji dla klasyfikatora k=3 rozpoznającego litery na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same litery.cechyn	99,9	99,8
2	dane 9x9 same litery.cechyn	99,9	100,0
3	dane 9x9 same litery.cechyn	99,9	99,9
4	dane 9x9 same litery.cechyn	99,9	99,9
5	dane 9x9 same litery.cechyn	100,0	99,9
6	dane 9x9 same litery.cechyn	99,9	99,8
7	dane 9x9 same litery.cechyn	99,9	100,0
8	dane 9x9 same litery.cechyn	100,0	99,9
9	dane 9x9 same litery.cechyn	100,0	99,6
10	dane 9x9 same litery.cechyn	99,9	99,9

Minimalna skuteczność klasyfikacji dla klasyfikatora podejmującego decyzję wynosiła 99,4%, maksymalna 99,7%, a mediana 99,5%, przełożyło się odpowiednio na 18, 9 i 15 źle sklasyfikowanych znaków. Dla klasyfikatora cyfr minimalna skuteczność klasyfikacji dla zbioru testowego wynosiła 99,9%, maksymalna 100%, a mediana 100%, czyli odpowiednio 1, 0 i 0 źle sklasyfikowanych cyfr zaś dla klasyfikatora liter skuteczność minimalna wynosiła 99,6%, maksymalna 100%, a mediana 99,9%, w rezultacie czego źle sklasyfikowanych zostało odpowiednio 8, 0 i 3 liter. Wyniki klasyfikatora liczb i cyfr były praktycznie idealne a klasyfikatora podejmującego decyzję bardzo wysokie. Powyższe rozwiązanie okazało się prawie tak samo skutecznym jak rozwiązanie podstawowe.

3.3 Binarne drzewo decyzyjne

Binarne drzewo decyzyjne w OpenCV zaimplementowane zostało w klasie `CvDTree`. W celu utworzenia działającego klasyfikatora należało utworzyć obiekt klasy i wywołać dla niego metodę `train()` dostarczając jej jako jeden z argumentów obiekt klasy `CvDTreeParams` służący do konfigurowania parametrów klasyfikatora. Testowanie stworzonego klasyfikatora przebiegało w następujący sposób - kolejne znaki ze zbioru uczącego i testowego podawane były jako argument metody predict(), a wyniki klasyfikacji porównywane były z klasami badanych znaków. Poniżej znajduje się przykładowy kod.

```
CvMat* cvmZbiorDanych;
CvMat* cvmZbiorUczacy;
CvMat* cvmKlasyZbioruDanych;
CvMat* cvmKlasyZbioruUczacego;
CvMat* cvmKlasyZbioruTestowego;

WczytajDane(&cvmZbiorDanych, &cvmKlasyZbioruDanych);

//tworzenie wektora klas zbioru uczacego
cvmKlasyZbioruUczacego = cvCreateMat(iWielkoscZbioruUczacego, 1, CV_32F);

for( i=0; i<iWielkoscZbioruUczacego; i++){
      float* fKlasa = (float*)( cvmKlasyZbioruUczacego->data.ptr + i*
      cvmKlasyZbioruUczacego->step);
      fKlasa[0] = cvmKlasyZbioruDanych->data.fl[i];
}

//tworzenie zbioru uczącego
cvGetRows(cvmZbiorDanych, &cvmZbiorUczacy, 0, iWielkoscZbioruUczacego);

//typy cech
CvMat* cvTypyCech = cvCreateMat(iIloscCech+1, 1, CV_8U);
cvSet(cvTypyCech, cvScalarAll(CV_VAR_ORDERED));
cvSetReal1D(cvTypyCech, iIloscCech, CV_VAR_CATEGORICAL);
```

```
CvDTree cvdTree = CvDTree();

//parametry drzewa
CvDTreeParams cvdtParametry = CvDTreeParams(36, 1, 0, false, 8, 1, false,
false, NULL);

//mierzenie czasu szkolenia
int iSzkolenieStart = clock();
//szkolenie
cvdTree.train(&cvmZbiorUczacy, CV_ROW_SAMPLE,  cvmKlasyZbioruUczacego, 0,
0, cvTypyCech, 0, cvdtParametry);

int iSzkolenieKoniec = clock();
int iCzasSzkolenia = iSzkolenieKoniec - iSzkolenieStart;

//testowanie klasyfikatora
int iPoprawnaKlasyfikacjaUczacy = 0;
int iPoprawnaKlasyfikacjaTestowy = 0;
iTestowanieStart = clock();

for( i=0; i< cvmZbiorDanych->rows; i++{
      CvMat cvmObiekt;
      cvGetRow(cvmZbiorDanych, &cvmObiekt, i);

      float fKlasa = cvdTree.predict(&cvmObiekt)->value;

      int iKlasyfikacja = fabs(fKlasa - cvmKlasyZbioruDanych->data.fl[i]) <
      FLT_EPSILON ? 1 : 0;

      if(i < iWielkoscZbioruUczacego)
            iPoprawnaKlasyfikacjaUczacy += iKlasyfikacja;
      else
            iPoprawnaKlasyfikacjaTestowy += iKlasyfikacja;
}

int iTestowanieKoniec = clock();
int iCzasTestowania = iTestowanieKoniec - iTestowanieStart;

double dSkutecznoscDlaUczacego = (double) iPoprawnaKlasyfikacjaUczacy /
(double) iWielkoscZbioruUczacego * 100;
double dSkutecznoscDlaTestowego = (double) iPoprawnaKlasyfikacjaTestowy /
(double)(cvmZbiorDanych->rows -  iWielkoscZbioruUczacego) * 100;
```

Pierwszym krokiem podczas testowania klasyfikatora było wyznaczenie jego następujących parametrów: `max_depth`, `min_sample_count`, `max_categories`, `cv_folds` i `use_1se_rule`. Wartości dobierane były eksperymentalnie a badania przeprowadzone zostały w oparciu o pierwszy zestaw zbiorów. Parametrom `max_categories` i `max_depth` wstępnie zostały przypisane te same wartości, równe ilości klas w zbiorach danych (36 różnych znaków), `cv_folds` i `use_1se_rule` odpowiedzialne za przycinanie powstałego drzewa zostały wyzerowane, zaś

min_sample_count ustawiono wartość 2. Testy wykazały, iż zwiększanie parametru min_sample_count pogarszało skuteczność klasyfikacji zarówno zbioru uczącego jak i testowego. Zwiększanie wartości max_depth i max_categories ponad wartość 36 zgodnie z oczekiwaniami nie przynosiło żadnych zmian w skuteczności klasyfikacji. Dla większych max_categories wydłużał się czas szkolenia, co wynikało z implementacji algorytmu w OpenCV, o czym informowała dokumentacja. Zmniejszanie max_categories nie miało wpływu zarówno na skuteczność klasyfikacji jak i na czas uczenia, minimalna wartość nie mogła być jednak mniejsza niż 2. Dla max_depth większego od 23 wyniki pozostawały bez zmian, zwiększanie parametru nie zmieniało także czasu szkolenia. Najlepsze wyniki otrzymywano dla min_sample_count równego 1, chociaż według dokumentacji 2 powinno być najmniejszą wartością tego parametru. Użycie agresywniejszej metody przycinania drzewa poprzez ustawienie use_1se_rule na 1 pogorszyło wyniki klasyfikacji o średnio 15 punktów procentowych. Zastosowanie krzyżowej walidacji wyników (cv_folds) jednokrotnie nie zmieniło skuteczności klasyfikacji zaś dla większej liczby krzyżowych walidacji skuteczność klasyfikacji zbioru uczącego spadała średnio o 0,1 punktu procentowego a zbioru testowego wzrosła o 0,05 punktu procentowego. Na podstawie przeprowadzonych badań wybrano następujące wartości parametrów: max_depth równe 36, min_sample_count równe 1, max_categories równe 8, cv_folds równe 1 i use_1se_rule równe 0.

Skuteczność klasyfikacji danych uczących dla większości zbiorów wynosiła 100%. Najlepsze wyniki klasyfikacji otrzymano dla zbioru 81 cech gęstościowych przy podziale *9x9*. Średni czas uczenia klasyfikatora dla tego zbioru wynosił 0,776 s, czas klasyfikacji wszystkich znaków ze zbioru 0,01 s a pojedynczego znaku 1 ns. Dla wybranego zbioru normalizacja danych nie wpłynęła na wyniki. Najgorsza klasyfikacja zbioru testowego dla danych gęstościowych *9x9* wynosiła 93,1 % a najlepsza 94,8%, co przekładało się na błędną klasyfikację 195 i 148 znaków spośród 2884. Dla mediany skuteczności klasyfikacji równej 93,5% błędna klasyfikacja wystąpiła dla 186 znaków. Zbliżoną skuteczność klasyfikacji klasyfikator osiągnął dla wariantów zbiorów numer 3, 4, 8, 9 i 10.

Poniżej znajduje się uproszczona tabela wyników (Tabela 19) zawierająca średnie i mediany dla każdego zbioru danych zamiast wszystkich dziesięciu wyników, „n” oznacza dane znormalizowane, „un” nieznormalizowane.

Tabela 19. Wyniki testowania klasyfikatora na różnych zbiorach danych. Każdy wiersz przedstawia średnie czasy uczenia, klasyfikacji i klasyfikacji jednej litery, mediany skuteczności klasyfikacji zbioru uczącego i testowego oraz maksymalną i minimalną skuteczność klasyfikacji zbioru testowego dla 10 zestawów danego typu cech.

Dane	Mediana skuteczności klasyfikacji zbioru uczącego[%]	Min. skuteczność klasyfikacji zbioru testowego [%]	Max. skuteczność klasyfikacji zbioru testowego[%]	Mediana skuteczność klasyfikacji zbioru testowego [%]	Śr. czas uczenia [s]	Śr. czas klasyfikacji [s]	Śr. czas klasyfikacji jednej litery [ns]
7hu n	62,5	45,8	56,9	47,0	0,176	0,008	0,885
7hu un	100,0	72,7	75,8	74,5	0,137	0,008	0,885
7hu, otwory n	89,0	54,8	69,1	66,6	0,174	0,013	1,400
7hu, otwory un	100,0	80,2	83,8	81,9	0,138	0,006	0,694
10 cech ksztaltu n	99,6	71,8	74,2	72,6	0,166	0,006	0,672
10 cech ksztaltu un	99,5	71,9	73,9	72,6	0,175	0,009	1,042
10 cech ksztaltu, otwory n	100,0	75,1	79,9	77,5	0,170	0,003	0,358
10 cech ksztaltu, otwory un	100,0	75,0	78,9	76,8	0,175	0,011	1,198
10 cech ksztaltu, otwory, 7hu n	100,0	83,1	85,7	84,4	0,242	0,006	0,706
10 cech ksztaltu, otwory, 7hu un	100,0	87,1	88,7	88,2	0,247	0,006	0,706
3x3 n	100,0	81,0	84,0	81,9	0,142	0,010	1,064
3x3 un	100,0	80,9	84,3	81,6	0,141	0,013	1,411
4x4 n	100,0	84,3	86,4	85,6	0,222	0,005	0,526
4x4 un	100,0	84,3	86,2	85,5	0,219	0,011	1,232
4x5 n	100,0	86,3	89,2	87,2	0,261	0,008	0,851
4x5 un	100,0	86,4	89,3	87,1	0,263	0,005	0,515
5x5 n	100,0	91,5	93,1	92,3	0,291	0,009	1,053
5x5 un	100,0	91,5	93,1	92,3	0,292	0,005	0,526
6x6 n	100,0	91,0	92,8	92,0	0,411	0,006	0,717
6x6 un	100,0	90,9	92,8	92,0	0,408	0,010	1,064
7x7 n	100,0	91,8	93,8	92,7	0,505	0,011	1,232
7x7 un	100,0	91,9	93,7	92,7	0,502	0,006	0,717
8x8 n	100,0	92,1	94,4	93,1	0,748	0,008	0,885
8x8 un	100,0	92,1	94,4	93,1	0,750	0,008	0,874
9x9 n	100,0	93,1	94,8	93,5	0,776	0,010	1,064
9x9 un	100,0	93,2	94,8	93,5	0,775	0,006	0,706
13x13 n	100,0	92,5	94,0	93,0	1,550	0,011	1,232
13x13 un	100,0	92,5	93,9	93,0	1,552	0,009	1,053
16x16 n	100,0	90,7	94,0	93,1	3,034	0,011	1,243
16x16 un	100,0	90,6	94,0	93,1	3,031	0,009	1,053

Dane znormalizowane dla wszystkich zbiorów cech prócz *7hu*, *7hu*, *otwory*, *10 cech ksztaltu*, *otwory*, *7hu* dały zbliżone lub takie same wyniki jak dane nieznormalizowane. Dodanie informacji o otworach poprawiło skuteczność klasyfikacji dla zbioru

znormalizowanego *7hu* o około 19 punktów procentowych, nieznormalizowanego *7hu* o około 6 punktów procentowych, a dla *10 cech kształtu* (o około 5 punktów procentowych). Żaden ze zbiorów cech nie zapewnił odpowiedniej skuteczności klasyfikacji zbioru testowego, chociaż klasyfikacja zbiorów uczących w większości przypadków wynosiła 100%. Zbiór *10 cech ksztaltu, otwory, 7hu* dał lepsze wyniki niż część zbiorów cech gęstościowych, dopiero dla podziału *5x5* klasyfikacja zbioru testowego była skuteczniejsza. Najwyższą skuteczność otrzymano dla zbioru *9x9, chociaż* jest ona zbliżona do skuteczności klasyfikacji zbiorów *8x8, 13x13* i *16x16*. Wraz ze wzrostem liczby cech zawsze wzrastał czas szkolenia klasyfikatora zaś skuteczność klasyfikacji wzrastała do pewnego momentu a potem utrzymywała się na w miarę stałym poziomie.

Dla najlepiej klasyfikowanego zbioru cech gęstościowych *9x9* wykonano testy klasyfikatora na zbiorach testowych zapisanych innymi czcionkami niż zbiory uczące – wyniki ujęto w Tabeli 20 poniżej.

Tabela 20. Wyniki klasyfikacji dodatkowego zbioru testowego 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji
1	dane 9x9,cechyn	76,0
2	dane 9x9,cechyn	78,1
3	dane 9x9,cechyn	78,8
4	dane 9x9,cechyn	79,3
5	dane 9x9,cechyn	76,9
6	dane 9x9,cechyn	78,8
7	dane 9x9,cechyn	79,2
8	dane 9x9,cechyn	77,6
9	dane 9x9,cechyn	76,4
10	dane 9x9,cechyn	80,0

Minimalna skuteczność klasyfikacji wynosiła 76%, maksymalna 80% a mediana 78,5% co skutkowało odpowiednio 138, 115 i 124 źle sklasyfikowanymi znaki spośród 576. Mediana skuteczności klasyfikacji była mniejsza od skuteczność klasyfikacji podstawowych zbiorów testowych o 15 punktów procentowych. Dla danych nieznormalizowanych mediana skuteczności klasyfikacji również wynosiła 78,5%.

Ostatnim wykonanym testem binarnego drzewa decyzyjnego była próba stworzenia trzech współdziałających ze sobą klasyfikatorów w celu dokonania poprawnej klasyfikacji. Poniżej znajdują się tabele (Tabela 21, Tabela 22, Tabela 23) przedstawiające wyniki kolejno dla klasyfikatora podejmującego decyzję, czy dany znak jest litera czy cyfrą, klasyfikatora klasyfikującego litery i klasyfikatora klasyfikującego cyfry.

Tabela 21. Wyniki klasyfikacji dla klasyfikatora rozróżniającego cyfry od liter na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	9x9.cechyn	100,0	95,9
2	9x9.cechyn	100,0	94,7
3	9x9.cechyn	100,0	95,5
4	9x9.cechyn	100,0	95,5
5	9x9.cechyn	100,0	94,6
6	9x9.cechyn	100,0	95,7
7	9x9.cechyn	100,0	95,4
8	9x9.cechyn	100,0	95,0
9	9x9.cechyn	100,0	95,5
10	9x9.cechyn	100,0	95,0

Tabela 22. Wyniki klasyfikacji dla klasyfikatora rozpoznającego cyfry na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	9x9 same cyfry.cechyn	100,0	97,7
2	9x9 same cyfry.cechyn	100,0	97,3
3	9x9 same cyfry.cechyn	100,0	97,7
4	9x9 same cyfry.cechyn	100,0	97,7
5	9x9 same cyfry.cechyn	100,0	98,0
6	9x9 same cyfry.cechyn	100,0	97,2
7	9x9 same cyfry.cechyn	100,0	97,3
8	9x9 same cyfry.cechyn	100,0	97,7
9	9x9 same cyfry.cechyn	100,0	97,7
10	9x9 same cyfry.cechyn	100,0	98,5

Tabela 23. Wyniki klasyfikacji dla klasyfikatora rozpoznającego litery na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same litery.cechyn	100,0	96,4
2	dane 9x9 same litery.cechyn	100,0	95,1
3	dane 9x9 same litery.cechyn	100,0	95,9
4	dane 9x9 same litery.cechyn	100,0	95,3
5	dane 9x9 same litery.cechyn	100,0	96,3
6	dane 9x9 same litery.cechyn	100,0	96,6
7	dane 9x9 same litery.cechyn	100,0	95,8
8	dane 9x9 same litery.cechyn	100,0	96,2
9	dane 9x9 same litery.cechyn	100,0	96,3
10	dane 9x9 same litery.cechyn	100,0	95,8

Minimalna skuteczność klasyfikacji dla klasyfikatora podejmującego decyzję wynosiła 94,6%, maksymalna 95,9%, a mediana 95,4%, przełożyło się odpowiednio na 155, 118 i 131 źle sklasyfikowanych znaków. Dla klasyfikatora cyfr minimalna skuteczność klasyfikacji dla zbioru testowego wynosiła 97,2%, maksymalna 98,5%, a mediana 97,7%, czyli odpowiednio 22, 12 i 18 źle sklasyfikowanych cyfr zaś dla klasyfikatora liter skuteczność minimalna wynosiła 95,1%, maksymalna 96,6%, a mediana 95,9%, w rezultacie czego źle sklasyfikowanych zostało odpowiednio 100, 69 i 85 liter. W oparciu i przeprowadzone badania stwierdzono, iż powyższe rozwiązanie jest mniej skuteczne niż podstawowe podejście.

Binarne drzewo decyzyjne pozwala wyliczyć ważność każdej z cech w procesie klasyfikacji. Wyznaczone wartości dla zbioru *9x9* są takie same dla zbioru znormalizowanego i nieznormalizowanego i znajdują się poniżej w Tabeli 24.

Tabela 24. Ważności cech dla 9x9 cech gęstościowych.

Cecha	Ważność cech	Cecha	Ważność cech	Cecha	Ważność cech
1	0,0207	28	0,0000	55	0,0000
2	0,0257	29	0,0135	56	0,0069
3	0,0314	30	0,0143	57	0,0000
4	0,0310	31	0,0264	58	0,0143
5	0,0243	32	0,0346	59	0,0198
6	0,0158	33	0,0202	60	0,0090
7	0,0287	34	0,0363	61	0,0120
8	0,0119	35	0,0174	62	0,0061
9	0,0111	36	0,0000	63	0,0000
10	0,0000	37	0,0000	64	0,0000
11	0,0004	38	0,0136	65	0,0043
12	0,0147	39	0,0423	66	0,0018
13	0,0203	40	0,0111	67	0,0032
14	0,0229	41	0,0509	68	0,0000
15	0,0040	42	0,0254	69	0,0000
16	0,0009	43	0,0129	70	0,0115
17	0,0000	44	0,0199	71	0,0024
18	0,0000	45	0,0000	72	0,0000
19	0,0000	46	0,0000	73	0,0000
20	0,0008	47	0,0080	74	0,0220
21	0,0091	48	0,0190	75	0,0095
22	0,0074	49	0,0070	76	0,0003
23	0,0350	50	0,0253	77	0,0119
24	0,0086	51	0,0185	78	0,0177
25	0,0413	52	0,0119	79	0,0196
26	0,0077	53	0,0141	80	0,0114
27	0,0000	54	0,0000	81	0,0000

W wyniku przeprowadzonych eksperymentów ustalono, iż najlepszym spośród przetestowanych zbiorów cech dla binarnego drzewa decyzyjnego był zbiór cech

gęstościowych *9x9*. Otrzymane dla niego wyniki klasyfikacji zbioru uczącego, testowego, oraz dodatkowego zbioru testowego wynosiły odpowiednio: 100%, 93,5% i 78,5%.

3.4 Las losowy

Las losowy w OpenCV zaimplementowany został w klasie `CvRTrees`. W celu utworzenia działającego klasyfikatora należało utworzyć obiekt klasy i wywołać dla niego metodę `train()` dostarczając jej jako jeden z argumentów obiekt klasy `CvRTParams` służący do konfigurowania parametrów klasyfikatora. Testowanie stworzonego klasyfikatora przebiegało w następujący sposób - kolejne znaki ze zbioru uczącego i testowego podawane były jako argument metody predict(), a wyniki klasyfikacji porównywane były z klasami badanych znaków. Poniżej znajduje się przykładowy kod.

```
CvMat* cvmZbiorDanych;
CvMat* cvmZbiorUczacy;
CvMat* cvmKlasyZbioruDanych;
CvMat* cvmKlasyZbioruUczacego;
CvMat* cvmKlasyZbioruTestowego;

WczytajDane(&cvmZbiorDanych, &cvmKlasyZbioruDanych);

//tworzenie wektora klas zbioru uczacego
cvmKlasyZbioruUczacego = cvCreateMat(iWielkoscZbioruUczacego, 1, CV_32F);

for( i=0; i<iWielkoscZbioruUczacego; i++){
      float* fKlasa = (float*)( cvmKlasyZbioruUczacego->data.ptr + i*
      cvmKlasyZbioruUczacego->step);
      fKlasa[0] = cvmKlasyZbioruDanych->data.fl[i];
}

//tworzenie zbioru uczącego
cvGetRows(cvmZbiorDanych, &cvmZbiorUczacy, 0, iWielkoscZbioruUczacego);

//typy cech
CvMat* cvTypyCech = cvCreateMat(iIloscCech+1, 1, CV_8U);
cvSet(cvTypyCech, cvScalarAll(CV_VAR_ORDERED));
cvSetReal1D(cvTypyCech, iIloscCech, CV_VAR_CATEGORICAL);

CvRTrees cvrTrees = CvRTrees();

//parametry drzewa
CvRTParams cvrtParametry = CvRTParams (36, 1, 0, false, 8, NULL, false, 0,
0, 0, 0);
cvrtParametry.term_crit.max_iter = 100;
cvrtParametry.term_crit.epsilon = 0.01;
cvrtParametry.term_crit.type = CV_TERMCRIT_ITER|CV_TERMCRIT_EPS;

//mierzenie czasu szkolenia
int iSzkolenieStart = clock();
```

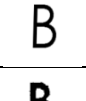
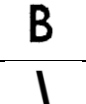
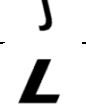
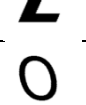
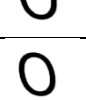
```
//szkolenie
cvrTrees.train(&cvmZbiorUczacy, CV_ROW_SAMPLE,  cvmKlasyZbioruUczacego, 0,
0, cvTypyCech, 0, cvrtParametry);

int iSzkolenieKoniec = clock();
int iCzasSzkolenia = iSzkolenieKoniec – iSzkolenieStart;

//testowanie klasyfikatora
int iPoprawnaKlasyfikacjaUczacy = 0;
int iPoprawnaKlasyfikacjaTestowy = 0;
iTestowanieStart = clock();

for( i=0; i< cvmZbiorDanych->rows; i++{
      CvMat cvmObiekt;
      cvGetRow(cvmZbiorDanych, &cvmObiekt, i);

      float fKlasa = cvrTrees.predict(&cvmObiekt);

      int iKlasyfikacja = fabs(fKlasa – cvmKlasyZbioruDanych->data.fl[i]) <
      FLT_EPSILON ? 1 : 0;

      if(i < iWielkoscZbioruUczacego)
            iPoprawnaKlasyfikacjaUczacy += iKlasyfikacja;
      else
            iPoprawnaKlasyfikacjaTestowy += iKlasyfikacja;
}

int iTestowanieKoniec = clock();
int iCzasTestowania = iTestowanieKoniec – iTestowanieStart;

double dSkutecznoscDlaUczacego = (double) iPoprawnaKlasyfikacjaUczacy /
(double) iWielkoscZbioruUczacego * 100;
double dSkutecznoscDlaTestowego = (double) iPoprawnaKlasyfikacjaTestowy /
(double)(cvmZbiorDanych->rows -  iWielkoscZbioruUczacego) * 100;
```

Pierwszym krokiem podczas testowania klasyfikatora było wyznaczenie jego następujących parametrów: `max_depth`, `min_sample_count`, `max_categories`, `max_num_of_trees_in_the_forest`, `nactive_vars`, `forest_accuracy`. Wartości dobierane były eksperymentalnie a badania przeprowadzone zostały w oparciu o pierwszy zestaw zbiorów. Pierwsze trzy parametry odpowiedzialne za konfigurację binarnych drzew decyzyjnych ustalone zostały w oparciu o uprzednio przeprowadzone dla nich badania. Początkowo parametrowi `forest_accuracy` przypisana została wartość 0,01, maksymalna ilość drzew wynosiła 100 a `nactive_vars` wynosiło 0, co domyślnie przypisywało mu wartość pierwiastka kwadratowego ilości cech. Jako kryterium stopu przyjęto osiągnięcie maksymalnej ilości drzew lub akceptowalnego błędu klasyfikacji lasu. Badania wykazały, iż zwiększanie liczby drzew powoduje nieznaczne zwiększenie skuteczności klasyfikacji wydłużając jednak zauważalnie czas uczenia i klasyfikacji. Porównanie lasu składającego się ze 100 drzew i lasu składającego się z 1700 drzew

wskazało, iż skuteczność klasyfikacji drugiego była większa o średnio 0,2 punktu procentowego zaś czas uczenia był dłuższy o średnio 485 s a czas klasyfikacji o 34 s, gdzie dla pierwszego lasu średni czas uczenia wynosił 31 s a czas klasyfikacji 1,4 s. Jedynie dla połowy zbiorów cech osiągnięty został akceptowalny błąd klasyfikacji, dla pozostałych szkolenie klasyfikatora przerywane było z powodu osiągnięcia maksymalnej ilości drzew w lesie. Zmniejszenie akceptowalnego błędu do 0,001 zwiększyło skuteczność klasyfikacji średnio o 0,1 punktu procentowego i wydłużyło czas uczenia średnio o 16 s. Dodatkowo dla najlepiej klasyfikowanego podczas wszystkich testów zbioru cech gęstościowych *13x13* wynik pozostał zawsze ten sam (choć wydłużały się czasy), niezależnie od zwiększania liczby drzew i zmniejszania akceptowalnego błędu. Parametr `nactive_vars` na etapie ustalania powyższych parametrów nie był modyfikowany, gdyż jego dobór uzależniony był od ilości cech w zbiorze a te były różne dla poszczególnych zbiorów. Dla wybranego zbioru *16x16* najlepszą wartością okazała się wartość domyślna czyli 16. Zwiększanie i zmniejszanie parametru `nactive_vars` skutkowało skróceniem czasu szkolenia i klasyfikowania trzykrotnie, ale spadała także skuteczność klasyfikacji zbioru testowego. Na podstawie przeprowadzonych badań wybrano następujące wartości parametrów: `max_depth` równe 36, `min_sample_count` równe 1, `max_categories` równe 8, `max_num_of_trees_in_the_forest` równe 100, `nactive_vars` równe 0, `forest_accuracy` równe 0,01.

Dla większości zbiorów skuteczność klasyfikacji danych uczących wynosiła 100%. Najlepsze wyniki klasyfikacji otrzymano dla zbioru 81 cech gęstościowych przy podziale *9x9,* należy jednak wspomnieć, iż wyniki dla zbiorów gęstościowych większych od *6x6* były praktycznie takie same. Średni czas uczenia klasyfikatora dla wybranego zbioru wynosił 14 s, czas klasyfikacji wszystkich znaków ze zbioru 0,33 s a pojedynczego znaku 0,04 ms. Normalizacja danych nie wpłynęła na wyniki. Najgorsza klasyfikacja zbioru testowego dla danych gęstościowych *9x9* wynosiła 99,1 % a najlepsza 99,6% co przekładało się na błędną klasyfikację 26 i 12 znaków spośród 2884. Dla mediany skuteczności klasyfikacji równej 99,4% błędna klasyfikacja wystąpiła dla 17 znaków. Zbliżoną skuteczność klasyfikacji klasyfikator osiągnął dla wariantów zbiorów numer 4 i 6. Kolejne źle sklasyfikowane znaki dla zbioru numer 6 zostały przedstawione w Tabeli 25.

Tabela 25. Błędnie sklasyfikowane znaki i ich poprawne klasy z 6 zestawu 9x9 cech gęstościowych.

Znak	Wynik klasyfikacji	Opis próbki	Obraz próbki
0	O	0_i_7_w_6	
0	O	0_b_3_w_2	
0	O	0_n_3_w_2	
1	I	1_b_6_n	
1	I	1_i_6_n	
1	I	1_m_2_n	
B	8	B_n_2_w_8	
B	8	B_b_2_n	
J	I	J_n_5_w_0	
L	Z	L_m_7_w_2	
O	0	O_n_4_w_3	
O	0	O_b_4_w_3	
O	0	O_b_4_w_1	
Q	O	Q_i_2_n	
T	I	T_n_2_n	
T	Z	T_i_2_n	
Y	V	Y_b_3_w_2	

Poniżej znajduje się uproszczona tabela wyników (Tabela 26) zawierająca średnie i mediany dla każdego zbioru danych zamiast wszystkich dziesięciu wyników, „n” oznacza dane znormalizowane, „un” nieznormalizowane.

Tabela 26. Wyniki testowania klasyfikatora na różnych zbiorach danych. Każdy wiersz przedstawia średnie czasy uczenia, klasyfikacji i klasyfikacji jednej litery, mediany skuteczności klasyfikacji zbioru uczącego i testowego oraz maksymalną i minimalną skuteczność klasyfikacji zbioru testowego dla 10 zestawów danego typu cech.

Dane	Mediana skuteczności klasyfikacji zbioru uczącego [%]	Min. skuteczność klasyfikacji zbioru testowego [%]	Max. skuteczność klasyfikacji zbioru testowego [%]	Mediana skuteczność klasyfikacji zbioru testowego [%]	Śr. czas uczenia [s]	Śr. czas klasyfikacji [s]	Śr. czas Klasyfikacji jednej litery [ms]
7hu n	63,2	49,5	63,3	50,5	13,316	2,510	0,281
7hu un	100	82,6	85,2	84,5	9,858	1,855	0,208
7hu, dziury n	50,4	41,4	54,6	45,6	13,631	1,362	0,153
7hu, dziury un	99,1	88,1	90,1	89,3	9,952	1,695	0,190
10 cech ksztalu n	100	83,0	86,0	85,1	11,477	1,713	0,192
10 cech ksztalu un	100	82,8	85,9	85,0	11,475	1,702	0,191
10 cech ksztaltu, dziury n	100	86,4	89,0	87,8	11,653	1,660	0,186
10 cech ksztaltu, dziury un	100	86,4	89,0	88,0	11,688	1,645	0,184
10 cech ksztaltu, dziury, 7hu n	100	92,3	96,1	93,8	15,492	1,595	0,179
10 cech ksztaltu, dziury, 7hu un	100	95,2	97,0	96,3	14,777	1,503	0,168
3x3 n	100	92,7	94,4	93,4	10,099	1,666	0,187
3x3 un	100	92,1	94,2	93,5	10,096	1,672	0,187
4x4 n	100	96,4	97,5	96,7	13,520	1,550	0,174
4x4 un	100	96,4	97,4	96,9	13,531	1,553	0,174
4x5 n	100	97,0	98,0	97,7	15,206	1,505	0,169
4x5 un	100	97,0	98,0	97,7	15,200	1,508	0,169
5x5 n	100	98,0	99,0	98,6	15,113	1,208	0,135
5x5 un	100	98,0	99,0	98,6	15,111	1,210	0,135
6x6 n	100	99,0	99,4	99,1	15,531	0,884	0,099
6x6 un	100	99,0	99,4	99,1	16,028	0,919	0,103
7x7 n	100	99,0	99,5	99,3	11,284	0,439	0,049
7x7 un	100	99,0	99,5	99,3	11,353	0,440	0,049
8x8 n	100	99,1	99,6	99,3	15,461	0,434	0,049
8x8 un	100	99,1	99,6	99,3	15,686	0,437	0,049
9x9 n	100	99,1	99,6	99,4	14,037	0,333	0,037
9x9 un	100	99,1	99,6	99,4	14,281	0,345	0,039
13x13 n	100	99,2	99,6	99,3	26,458	0,312	0,035
13x13 un	100	99,2	99,6	99,3	26,306	0,314	0,035
16x16 n	100	99,0	99,5	99,4	41,512	0,275	0,031
16x16 un	100	99,0	99,5	99,4	41,706	0,274	0,031

Dane nieznormalizowane dla zbiorów cech *7hu, 7hu, otwory, 10 cech ksztaltu, otwory, 7hu, 3x3 i 4x4* dały lepsze wyniki niż dane znormalizowane (dla dwóch pierwszych zbiorów odpowiednio aż o 34 i 43,7 punktów proco netowych), dla pozostałych zbiorów normalizacja nie wpływała na skuteczność klasyfikacji. Dodanie informacji o otworach pogorszyło skuteczność klasyfikacji dla zbioru *7hu* o około 5 punktów procentowych, zaś dla *10 cech kształtu* poprawiło o około 2 punkty procentowe. Zbiór *10 cech ksztaltu, otwory, 7hu* dał lepsze wyniki niż zbiór cech gęstościowych *3x3*. Najwyższą skuteczność otrzymano dla zbiorów *9x9* oraz *16x16*, chociaż jest ona zbliżona do skuteczności klasyfikacji zbiorów *8x8* i *13x13*. Wraz ze wzrostem liczby cech zawsze wzrastał czas szkolenia klasyfikatora zaś skuteczność klasyfikacji wzrastała do pewnego momentu a potem utrzymywała się na w miarę stałym poziomie. Jako najlepszy zbiór wybrany został *9x9*, ponieważ czas jego szkolenia był krótszy niż zbioru *16x16*.

Dla najlepiej klasyfikowanego zbioru cech gęstościowych *9x9* wykonano testy klasyfikatora na zbiorach testowych zapisanych innymi czcionkami niż zbiory uczące – wyniki ujęto w Tabeli 27 poniżej.

Tabela 27. Wyniki klasyfikacji dodatkowego zbioru testowego 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji [%]
1	dane 9x9.cechyn	95,3
2	dane 9x9.cechyn	95,1
3	dane 9x9.cechyn	95,7
4	dane 9x9.cechyn	95,7
5	dane 9x9.cechyn	95,8
6	dane 9x9.cechyn	95,8
7	dane 9x9.cechyn	96,0
8	dane 9x9.cechyn	95,5
9	dane 9x9.cechyn	95,1
10	dane 9x9.cechyn	95,8

Minimalna skuteczność klasyfikacji wynosiła 95,1%, maksymalna 96,0% a mediana 95,7% co skutkowało odpowiednio 28, 23 i 25 źle sklasyfikowanymi znaki spośród 576. Mediana skuteczności klasyfikacji była mniejsza od skuteczność klasyfikacji podstawowych zbiorów testowych o 3,7 punktu procentowego.

Ostatnim wykonanym testem lasu losowego była próba stworzenia trzech współdziałających ze sobą klasyfikatorów w celu dokonania poprawnej klasyfikacji. Poniżej znajdują się tabele (Tabela 28, Tabela 29, Tabela 30) przedstawiające wyniki kolejno dla klasyfikatora podejmującego decyzję, czy dany znak jest litera czy cyfrą, klasyfikatora klasyfikującego litery i klasyfikatora klasyfikującego cyfry.

Tabela 28. Wyniki klasyfikacji dla klasyfikatora rozróżniającego cyfry od liter na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9.cechyn	100,0	99,1
2	dane 9x9.cechyn	100,0	99,5
3	dane 9x9.cechyn	100,0	98,7
4	dane 9x9.cechyn	100,0	98,9
5	dane 9x9.cechyn	100,0	98,9
6	dane 9x9.cechyn	100,0	98,9
7	dane 9x9.cechyn	100,0	98,9
8	dane 9x9.cechyn	100,0	98,7
9	dane 9x9.cechyn	100,0	98,7
10	dane 9x9.cechyn	100,0	98,8

Tabela 29. Wyniki klasyfikacji dla klasyfikatora rozpoznającego cyfry na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same cyfry.cechyn	100,0	99,6
2	dane 9x9 same cyfry.cechyn	100,0	99,5
3	dane 9x9 same cyfry.cechyn	100,0	99,7
4	dane 9x9 same cyfry.cechyn	100,0	99,9
5	dane 9x9 same cyfry.cechyn	100,0	99,6
6	dane 9x9 same cyfry.cechyn	100,0	100,0
7	dane 9x9 same cyfry.cechyn	100,0	99,1
8	dane 9x9 same cyfry.cechyn	100,0	99,9
9	dane 9x9 same cyfry.cechyn	100,0	99,9
10	dane 9x9 same cyfry.cechyn	100,0	99,9

Tabela 30. Wyniki klasyfikacji dla klasyfikatora rozpoznającego litery na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same litery.cechyn	100,0	99,7
2	dane 9x9 same litery.cechyn	100,0	99,5
3	dane 9x9 same litery.cechyn	100,0	99,7
4	dane 9x9 same litery.cechyn	100,0	99,3
5	dane 9x9 same litery.cechyn	100,0	99,6
6	dane 9x9 same litery.cechyn	100,0	99,5
7	dane 9x9 same litery.cechyn	100,0	99,7
8	dane 9x9 same litery.cechyn	100,0	99,4
9	dane 9x9 same litery.cechyn	100,0	100,0
10	dane 9x9 same litery.cechyn	100,0	99,9

Minimalna skuteczność klasyfikacji dla klasyfikatora podejmującego decyzję wynosiła 98,7%, maksymalna 99,5%, a mediana 98,9%, przełożyło się odpowiednio na 39, 13 i 32 źle sklasyfikowanych znaków. Dla klasyfikatora cyfr minimalna skuteczność klasyfikacji dla zbioru testowego wynosiła 99,1%, maksymalna 100%, a mediana 99,8%, czyli odpowiednio 7, 0 i 2 źle sklasyfikowanych cyfr zaś dla klasyfikatora liter skuteczność minimalna wynosiła 99,3%, maksymalna 100%, a mediana 99,6%, w rezultacie czego źle sklasyfikowanych zostało odpowiednio 14, 0 i 7 liter. Pomimo bardzo dobrej skuteczności klasyfikatorów liter i cyfr, niska skuteczność klasyfikatora dokonującego rozróżnienia czy dany znak jest literą czy cyfrą czyniła powyższe rozwiązanie gorszym od podejścia podstawowego.

W wyniku przeprowadzonych eksperymentów ustalono, iż najlepszym spośród przetestowanych zbiorów cech dla lasu losowego był zbiór cech gęstościowych *9x9*. Otrzymane dla wybranego zbioru wyniki klasyfikacji zbioru uczącego, testowego, oraz dodatkowego zbioru testowego wynosiły odpowiednio: 100%, 99,4% i 95,7%.

3.5 AdaBoost

AdaBoost w OpenCV zaimplementowany został w klasie `cvBoost`. W celu utworzenia działającego klasyfikatora należało utworzyć obiekt klasy i wywołać dla niego metodę `train()` dostarczając jej jako jeden z argumentów obiekt klasy `CvBoostParams` służący do konfigurowania parametrów klasyfikatora. Ponieważ klasyfikator AdaBoost umożliwiał jedynie klasyfikację binarną wczytany zbiór danych musiał zostać odpowiednio zmodyfikowany. Każdy wektor cech i odpowiadająca mu klasa znaku zostały zastąpione przez 36 tych samych wektorów cech lecz z przypisanymi wartościami klasy równymi 0 lub 1 – 1 dla *i*-tego wektora odpowiadającego numerowi klasy badanego znaku, 0 – dla pozostałych wektorów. Testowanie stworzonego klasyfikatora przebiegało w następujący sposób - kolejne znaki ze zbioru uczącego i testowego podawane były jako argument metody predict(), a wyniki klasyfikacji porównywane były z klasami badanych znaków. Poniżej znajduje się przykładowy kod.

```
CvMat* cvmZbiorDanych;
CvMat* cvmZbiorUczacy;
CvMat* cvmKlasyZbioruDanych;
CvMat* cvmKlasyZbioruUczacego;
CvMat* cvmKlasyZbioruTestowego;

WczytajDane(&cvmZbiorDanych, &cvmKlasyZbioruDanych);

//konwersja danych
cvmKlasyZbioruUczacego = cvCreateMat(iWielkoscZbioruUczacego, 1, CV_32F);
```

```
CvMat* cvmNowyZbiorUczacy = cvCreateMat(cvmZbiorDanych->rows * 36,
iIloscCech+1, CV_32F);
cvmKlasyZbioruUczacego = cvCreateMat(cvmZbiorDanych->rows * 36, 1, CV_32S);

for(i = 0; i < cvmZbiorDanych->rows; i++){
      float* fWiersz = (float*)( cvmZbiorDanych ->data.ptr +
                                    cvmZbiorDanych->step*i);

      for(j = 0; j < 36; j++){
            float* fNowyWiersz = (float*)(cvmNowyZbiorUczacy->data.ptr +
                        cvmNowyZbiorUczacy->step*(i*36+j));

            for(k = 0; k < iIloscCech; k++)
                fNowyWiersz [k] = fWiersz [k];

            fNowyWiersz [iIloscCech] = (float)j;

            int iKlasa = 0;
            if(j >= 27 && j <= 36)
                  iKlasa = j + 22;
            else
                  iKlasa = j + 'A';

            cvmKlasyZbioruUczacego ->data.i[i*36 + j] =
                        cvmKlasyZbioruDanych ->data.fl[i] == iKlasa;
      }
}

//typy cech
CvMat* cvTypyCech = cvCreateMat(iIloscCech+2, 1, CV_8U);
cvSet(cvTypyCech, cvScalarAll(CV_VAR_ORDERED));
cvSetReal1D(cvTypyCech, iIloscCech, CV_VAR_CATEGORICAL);
cvSetReal1D(cvTypyCech, iIloscCech+1, CV_VAR_CATEGORICAL);

CvBoost cvbAdaBoost;

//mierzenie czasu szkolenia
int iSzkolenieStart = clock();

//szkolenie
cvbAdaBoost.train(cvmNowyZbiorUczacy, CV_ROW_SAMPLE,
                  cvmKlasyZbioruUczacego, 0, 0, cvTypyCech, 0,
                  CvBoostParams(CvBoost::REAL, 100, 0.95, 5, false, 0 ));

int iSzkolenieKoniec = clock();
int iCzasSzkolenia = iSzkolenieKoniec - iSzkolenieStart;

//testowanie klasyfikatora
int iPoprawnaKlasyfikacjaUczacy = 0;
int iPoprawnaKlasyfikacjaTestowy = 0;
CvMat* cvmObiektTmp = cvCreateMat(1, iIloscCech + 1, CV_32F);
CvMat* cvmOdpowiedzi = cvCreateMat(1,
                        cvbAdaBoost.get_weak_predictors()->total, CV_32F);
iTestowanieStart = clock();

for( i=0; i< cvmZbiorDanych->rows; i++{
      double dMaxSuma = -DBL_MAX;
      int iKlasa = 0;
      CvMat cvmObiekt;
      cvGetRow(cvmZbiorDanych, &cvmObiekt, i);
```

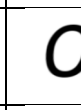
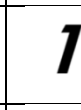
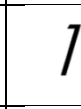
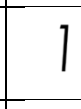
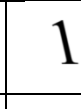
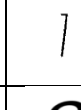
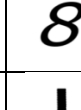
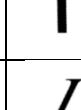
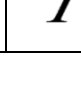
```
	for(k = 0; k < iIloscCech; k++)
		cvmObiektTmp->data.fl[k] = cvmObiekt.data.fl[k];

	for(j = 0; j < 36; j++){
		cvmObiektTmp->data.fl[iIloscCech] = (float)j;
		cvbAdaBoost.predict(cvmObiektTmp, 0, cvmOdpowiedzi);
		double dSuma = cvSum(cvmOdpowiedzi).val[0];

		if(dMaxSuma < dSuma){
		    dMaxSuma = dSuma;

		      if(j >= 26 && j <= 35)
			    best_class = j + 22;
		      else
			    best_class = j + 'A';
		}
	}

	int iKlasyfikacja = fabs(iKlasa - cvmKlasyZbioruDanych->data.fl[i]) <
	FLT_EPSILON ? 1 : 0;

	if(i < iWielkoscZbioruUczacego)
		iPoprawnaKlasyfikacjaUczacy += iKlasyfikacja;
	else
		iPoprawnaKlasyfikacjaTestowy += iKlasyfikacja;
}

int iTestowanieKoniec = clock();
int iCzasTestowania = iTestowanieKoniec - iTestowanieStart;

double dSkutecznoscDlaUczacego = (double) iPoprawnaKlasyfikacjaUczacy /
(double) iWielkoscZbioruUczacego * 100;
double dSkutecznoscDlaTestowego = (double) iPoprawnaKlasyfikacjaTestowy /
(double)(cvmZbiorDanych->rows -  iWielkoscZbioruUczacego) * 100;
```

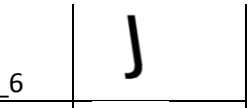
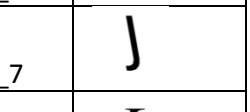
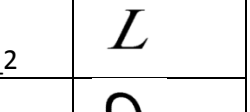
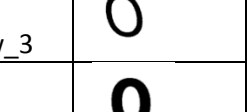
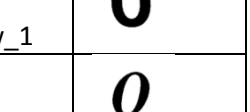
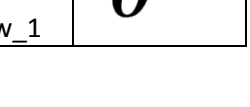
Pierwszym krokiem podczas testowania klasyfikatora było ustalenie jego następujących parametrów: `max_depth`, `boost_type`, `weak_count`, `split_criteria`, `weight_trim_rate`. Wartości dobierane były eksperymentalnie a badania przeprowadzone zostały w oparciu o pierwszy zestaw zbiorów. Parametr `max_depth` i `weak_count` odpowiadały za głębokość i ilość drzew decyzyjnych użytych podczas konstruowania klasyfikatora. Parametrowi `weak_count` przypisano wartość 100. Pierwszą przyjętą wartością parametru `max_depth` było 36 zgodnie z wynikami otrzymanymi podczas testów drzew decyzyjnych. Czas szkolenia zbiorów dla `max_depth` 36 był bardzo długi (np. dla zbioru *16x16* 27266s), tak więc wartość parametru była stopniowo zmniejszana aż do wartości 5 (czas szkolenia dla zbioru *16x16* wyniósł 6316s). Zyskano w ten sposób znaczne przyspieszenie procesu uczenia klasyfikatora (ponad 4 razy szybciej) kosztem nieznacznie gorszej skuteczności klasyfikacji zbioru testowego – nie więcej niż 0,1 punktu procentowego. Najlepsze wyniki uzyskano dla `boost_type GENTLE`, chociaż dokumentacja wskazuje, iż do zadania klasyfikacji najlepiej nadaje się `REAL`. Jako

`spli_criteria` użyto wartości `DEFAULT`. Dla parametru `weight_trim_rate` przyjęto wartość 0,95 – zmniejszanie tej wartości niezauważalnie wpływało na skrócenie czasu uczenia a zarazem zmniejszało skuteczność klasyfikacji zbioru testowego.

Normalizacja danych dla cech gęstościowych nie wpływała na wyniki klasyfikacji zaś dla pozostałych cech skutkowała obniżeniem skuteczności. Najlepsze wyniki klasyfikacji otrzymano dla zbioru 81 cech gęstościowych przy podziale *9x9*. Ze względu na długi czas szkolenia zbiorów *13x13* i *16x16* oraz ich skuteczność mniejszą niż dla zbioru *9x9* w przeprowadzonych badaniach zostały uwzględnione tylko zestawy numer 2, 3, 4 i 10. Średni czas uczenia klasyfikatora dla wybranego zbioru wynosił 1929 s, czas klasyfikacji wszystkich znaków ze zbioru 11,5 s a pojedynczego znaku 1,3 ms. Najgorsza klasyfikacja zbioru testowego dla danych gęstościowych *9x9* wynosiła 99,4% a najlepsza 99,7% co przekładało się na błędną klasyfikację 17 i 9 znaków spośród 2884. Dla mediany skuteczności klasyfikacji równej 99,5% błędna klasyfikacja wystąpiła dla 14 znaków. Zbliżoną skuteczność klasyfikacji klasyfikator osiągnął dla wariantów zbiorów numer 5, 6 i 9. Kolejne źle sklasyfikowane znaki dla zbioru numer 5 zostały przedstawione w Tabeli 31.

Tabela 31. Błędnie sklasyfikowane znaki i ich poprawne klasy z 5 zestawu 9x9 cech gęstościowych .

Znak	Wynik klasyfikacji	Opis próbki	Obraz próbki
0	O	0_n_3_w_2	
1	7	1_m_2_w_2	
1	I	1_i_2_w_2	
1	I	1_i_2_w_1	
1	I	1_n_6_w_0	
1	I	1_i_2_n	
8	B	8_i_7_w_2	
I	1	I_m_1_w_0	
I	Y	I_i_6_w_2	

J	1	J_n_5_w_6	
J	1	J_n_5_w_7	
L	Z	L_i_6_w_2	
O	0	O_n_4_w_3	
O	0	O_b_4_w_1	
O	0	O_m_6_w_1	

Poniżej znajduje się uproszczona tabela wyników (Tabela 32) zawierająca średnie i mediany dla każdego zbioru danych zamiast wszystkich dziesięciu wyników, „n” oznacza dane znormalizowane, „un” nieznormalizowane.

Tabela 32. Wyniki testowania klasyfikatora na różnych zbiorach danych. Każdy wiersz przedstawia średnie czasy uczenia, klasyfikacji i klasyfikacji jednej litery, mediany skuteczności klasyfikacji zbioru uczącego i testowego oraz maksymalną i minimalną skuteczność klasyfikacji zbioru testowego dla 10 zestawów danego typu cech.

Dane	Mediana skuteczności klasyfikacji zbioru uczącego [%]	Min. skuteczność klasyfikacji zbioru testowego [%]	Max. skuteczność klasyfikacji zbioru testowego [%]	Mediana skuteczność klasyfikacji zbioru testowego [%]	Śr. czas uczenia [s]	Śr. czas klasyfikacji [s]	Śr. czas klasyfikacji jednej litery [ms]
7hu n	56,7	47,2	56,3	48,9	153,788	10,564	1,183
7hu un	89,8	76,4	78,7	77,6	172,150	10,742	1,203
7hu, otwory n	74,7	60,0	70,0	65,9	173,000	10,711	1,200
7hu, otwory un	95,5	84,1	87,3	85,3	189,402	10,764	1,206
10 cech ksztaltu n	92,4	77,6	80,1	78,9	227,317	10,639	1,192
10 cech ksztaltu un	93,6	79,8	81,3	80,5	226,819	10,523	1,179
10 cech ksztaltu, otwory n	95,4	83,9	85,8	85,1	238,710	10,469	1,173
10 cech ksztaltu, otwory un	95,8	83,7	86,1	85,3	240,644	10,427	1,168
10 cech ksztaltu, otwory, 7hu n	99,3	91,3	94,9	92,5	395,647	10,527	1,179

10 cech kształtu, otwory, 7hu un	99,7	95,4	96,7	96,0	410,800	10,605	1,188
3x3 n	99,2	90,6	92,5	91,5	211,131	10,675	1,196
3x3 un	99,2	90,6	92,2	91,5	214,547	11,172	1,251
4x4 n	100,0	96,6	97,3	97,1	381,452	10,922	1,223
4x4 un	100,0	96,6	97,3	97,1	377,527	10,953	1,227
4x5 n	100,0	97,5	98,2	98,0	474,605	10,936	1,225
4x5 un	100,0	97,5	98,2	98,0	473,905	10,845	1,215
5x5 n	100,0	98,7	99,4	99,2	600,273	10,880	1,219
5x5 un	100,0	98,7	99,4	99,2	598,552	10,966	1,228
6x6 n	100,0	99,1	99,5	99,3	870,852	11,117	1,245
6x6 un	100,0	99,1	99,5	99,3	892,401	11,363	1,273
7x7 n	100,0	99,2	99,6	99,4	1165,750	11,005	1,233
7x7 un	100,0	99,2	99,6	99,4	1166,405	11,208	1,255
8x8 n	100,0	99,2	99,7	99,5	1528,769	11,322	1,268
8x8 un	100,0	99,2	99,7	99,5	1543,630	11,289	1,264
9x9 n	100,0	99,4	99,7	99,5	1929,055	11,483	1,286
9x9 un	100,0	99,4	99,7	99,5	1921,569	11,495	1,288
13x13 n	100,0	99,5	99,8	99,5	4176,887	12,145	1,360
13x13 un	100,0	99,5	99,8	99,5	4171,540	12,090	1,354
16x16 n	100,0	99,3	99,7	99,6	6616,820	12,582	1,409
16x16 un	100,0	99,3	99,7	99,6	6511,785	12,762	1,429

Dla danych nieznormalizowane ze zbiorów cech innych niż cechy gęstościowe otrzymano lepsze wyniki niż dla danych znormalizowanych - odpowiednio o 28,7 punktu procentowego na zbiorze *7hu*, 19,7 punktu procentowego na zbiorze *7hu*, *otwory*, 1,6 punktu procentowego na *10 cech kształtu*, 0,2 punktu procentowego na *10 cech kształtu, otwory* i 3,5 punktu procentowego na *10 cech kształtu, otwory, 7hu*. Dodanie informacji o otworach polepszyło skuteczność klasyfikacji dla zbioru *7hu* o 7,7 punktu procentowego, zaś dla *10 cech kształtu* 4,8 punktu procentowego. Zbiór *10 cech kształtu*, *otwory*, *7hu* dał lepsze wyniki niż zbiór cech gęstościowych *3x3*. Wraz ze wzrostem liczby cech zawsze wzrastał czas szkolenia klasyfikatora zaś skuteczność klasyfikacji wzrastała aż do zbioru *9x9* potem zaś maleć utrzymywała się na zbliżonym poziomie.

Dla najlepiej klasyfikowanego zbioru cech gęstościowych *9x9* wykonano testy klasyfikatora na zbiorach testowych zapisanych innymi czcionkami niż zbiory uczące – wyniki ujęto w Tabeli 33 poniżej.

Tabela 33. Wyniki klasyfikacji dodatkowego zbioru testowego 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji [%]
1	dane 9x9.cechyn	95,3
2	dane 9x9.cechyn	95,1
3	dane 9x9.cechyn	95,7
4	dane 9x9.cechyn	95,7
5	dane 9x9.cechyn	95,8
6	dane 9x9.cechyn	95,8
7	dane 9x9.cechyn	96,0
8	dane 9x9.cechyn	95,5
9	dane 9x9.cechyn	95,1
10	dane 9x9.cechyn	95,8

Minimalna skuteczność klasyfikacji wynosiła 95,1%, maksymalna 96% a mediana 95,7% co skutkowało odpowiednio 28, 23 i 25 źle sklasyfikowanymi znaki spośród 576. Mediana skuteczności klasyfikacji była mniejsza od skuteczność klasyfikacji podstawowych zbiorów testowych o 3,8 punktu procentowego.

Ostatnim wykonanym testem AdaBoost była próba stworzenia trzech współdziałających ze sobą klasyfikatorów w celu dokonania poprawnej klasyfikacji. Poniżej znajdują się tabele (Tabela 34, Tabela 35, Tabela 36) przedstawiające wyniki kolejno dla klasyfikatora podejmującego decyzję, czy dany znak jest litera czy cyfrą, klasyfikatora klasyfikującego litery i klasyfikatora klasyfikującego cyfry.

Tabela 34. Wyniki klasyfikacji dla klasyfikatora rozróżniającego cyfry od liter na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9.cechyn	100,0	99,5
2	dane 9x9.cechyn	100,0	99,8
3	dane 9x9.cechyn	100,0	99,4
4	dane 9x9.cechyn	100,0	99,6
5	dane 9x9.cechyn	100,0	99,5
6	dane 9x9.cechyn	100,0	99,5
7	dane 9x9.cechyn	100,0	99,6
8	dane 9x9.cechyn	100,0	99,5
9	dane 9x9.cechyn	100,0	99,5
10	dane 9x9.cechyn	100,0	99,5

Tabela 35. Wyniki klasyfikacji dla klasyfikatora rozpoznającego cyfry na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same cyfry.cechyn	100,0	100,0
2	dane 9x9 same cyfry.cechyn	100,0	100,0
3	dane 9x9 same cyfry.cechyn	100,0	100,0
4	dane 9x9 same cyfry.cechyn	100,0	100,0
5	dane 9x9 same cyfry.cechyn	100,0	100,0
6	dane 9x9 same cyfry.cechyn	100,0	100,0
7	dane 9x9 same cyfry.cechyn	100,0	99,9
8	dane 9x9 same cyfry.cechyn	100,0	100,0
9	dane 9x9 same cyfry.cechyn	100,0	100,0
10	dane 9x9 same cyfry.cechyn	100,0	100,0

Tabela 36. Wyniki klasyfikacji dla klasyfikatora rozpoznającego litery na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same litery.cechyn	100	99,903
2	dane 9x9 same litery.cechyn	100	99,854
3	dane 9x9 same litery.cechyn	100	99,805
4	dane 9x9 same litery.cechyn	100	99,757
5	dane 9x9 same litery.cechyn	100	99,951
6	dane 9x9 same litery.cechyn	100	99,805
7	dane 9x9 same litery.cechyn	100	99,951
8	dane 9x9 same litery.cechyn	100	99,951
9	dane 9x9 same litery.cechyn	100	99,903
10	dane 9x9 same litery.cechyn	100	99,951

Minimalna skuteczność klasyfikacji dla klasyfikatora podejmującego decyzję wynosiła 99,4%, maksymalna 99,8%, a mediana 99,5%, przełożyło się odpowiednio na 17, 5, 13 źle sklasyfikowanych znaków. Dla klasyfikatora cyfr minimalna skuteczność klasyfikacji dla zbioru testowego wynosiła 99,9%, maksymalna 100%, a mediana 100%, czyli odpowiednio 1, 0 i 0 źle sklasyfikowanych cyfr zaś dla klasyfikatora liter skuteczność minimalna wynosiła 99,8%, maksymalna 99,95%, a mediana 99,9%, w rezultacie czego źle sklasyfikowanych zostało odpowiednio 5, 1 i 2 liter. Skuteczność klasyfikacji dla poszczególnych klasyfikatorów była bardzo wysoka jednak powyższe rozwiązanie nie daje lepszych efektów niż rozwiązanie podstawowe.

W wyniku przeprowadzonych eksperymentów ustalono, iż najlepszym spośród przetestowanych zbiorów cech dla drzew decyzyjnych był zbiór cech gęstościowych *9x9*. Otrzymane dla wybranego zbioru wyniki klasyfikacji zbioru uczącego, testowego, oraz dodatkowego zbioru testowego wynosiły odpowiednio: 100%, 99,5% i 95,7%.

3.6 MLP (perceptron wielowarstwowy)

MLP w OpenCV zaimplementowany został w klasie `cvANN_MLP`. W celu utworzenia działającego klasyfikatora należało utworzyć obiekt klasy i wywołać dla niego metodę `train()` dostarczając jej jako jeden z argumentów obiekt klasy `CvANN_MLP_TrainParams` służący do konfigurowania parametrów klasyfikatora. Ze względu na implementację MLP wszystkie klasy musiały przyjmować wartości liczbowe z zakresu od 0 do liczby klas pomniejszonej o jeden. Testowanie stworzonego klasyfikatora przebiegało w następujący sposób - kolejne znaki ze zbioru uczącego i testowego podawane były jako argument metody predict(), a wyniki klasyfikacji porównywane były z klasami badanych znaków. Poniżej znajduje się przykładowy kod.

```
CvMat* cvmZbiorDanych;
CvMat* cvmZbiorUczacy;
CvMat* cvmKlasyZbioruDanych;
CvMat* cvmKlasyZbioruUczacego;
CvMat* cvmKlasyZbioruTestowego;

WczytajDane(&cvmZbiorDanych, &cvmKlasyZbioruDanych);

cvmKlasyZbioruUczacego = cvCreateMat(iWielkoscZbioruUczacego, 1, CV_32S);

for(i = 0; i < iWielkoscZbioruUczacego; i++){
      int iKlasa = 0;

      if(cvmKlasyZbioruDanych->data.fl[i] >= 48
                  && cvmKlasyZbioruDanych ->data.fl[i] <= 57)
            iKlasa = cvRound(cvmKlasyZbioruDanych ->data.fl[i]) - 22;
      else
            iKlasa = cvRound(cvmKlasyZbioruDanych ->data.fl[i]) - 'A';

      float* fWektor = (float*)( cvmKlasyZbioruUczacego->data.ptr
                                    + i* cvmKlasyZbioruUczacego->step);

      for(j = 0; j < 36; j++)
            fWektor [j] = 0.f;

      fWektor[iKlasa] = 1.f;
}

cvGetRows(cvmZbiorDanych, &cvmZbiorUczacy, 0, iWielkoscZbioruUczacego);

int iWarstwy[10];
iWarstwy[0] = cvmZbiorDanych->cols;
```

```
for(k=0;k<iIloscWarstw;k++)
      iWarstwy[k+1]= iWarstwyUkryte[k];

iWarstwy [k+1] = 36;

CvMat cvmWarstwy =
                  cvMat( 1, iIloscWarstw +2, CV_32S, iWarstwy);

cvANN_MLP cvamMlp.create(&cvmWarstwy);

//mierzenie czasu szkolenia
int iSzkolenieStart = clock();

//szkolenie
cvamMlp.train(&cvmZbiorUczacy, cvmKlasyZbioruUczacego, 0, 0,
                  CvANN_MLP_TrainParams(
                  cvTermCriteria(CV_TERMCRIT_ITER, iIloscIteracji, 0.01),
                  CvANN_MLP_TrainParams::BACKPROP,0.001));

int iSzkolenieKoniec = clock();
int iCzasSzkolenia = iSzkolenieKoniec – iSzkolenieStart;

//testowanie klasyfikatora
int iPoprawnaKlasyfikacjaUczacy = 0;
int iPoprawnaKlasyfikacjaTestowy = 0;
CvMat* cvmOdpMLP = cvCreateMat(1, 36, CV_32F );
iTestowanieStart = clock();

for( i=0; i< cvmZbiorDanych->rows; i++{
      CvMat cvmObiekt;
      int iKlasa = 0;
      cvGetRow(cvmZbiorDanych, &cvmObiekt, i);
      CvPoint cvpPunkt = {0,0};
      mlp.predict(&cvmObiekt, cvmOdpMLP);
      cvMinMaxLoc(cvmOdpMLP, 0, 0, 0, &cvpPunkt, 0);

      if(cvpPunkt.x >= 26 && cvpPunkt.x <= 35)
            iKlasa = cvpPunkt.x + 22;
      else
            iKlasa = cvpPunkt.x + 'A';

      int iKlasyfikacja = fabs(iKlasa – cvmKlasyZbioruDanych->data.fl[i]) <
      FLT_EPSILON ? 1 : 0;

      if(i < iWielkoscZbioruUczacego)
            iPoprawnaKlasyfikacjaUczacy += iKlasyfikacja;
      else
            iPoprawnaKlasyfikacjaTestowy += iKlasyfikacja;
}

int iTestowanieKoniec = clock();
int iCzasTestowania = iTestowanieKoniec – iTestowanieStart;

double dSkutecznoscDlaUczacego = (double) iPoprawnaKlasyfikacjaUczacy /
(double) iWielkoscZbioruUczacego * 100;
double dSkutecznoscDlaTestowego = (double) iPoprawnaKlasyfikacjaTestowy /
(double)(cvmZbiorDanych->rows -  iWielkoscZbioruUczacego) * 100;
```

Stworzenie skutecznego klasyfikatora MLP jest bardzo skomplikowane. Nie istniej żadna skuteczna metoda pozwalająca ustalić strukturę sieci przed rozpoczęciem badań. Liczba warstw i neuronów musi zostać dobrana eksperymentalnie i uzależniona jest od rodzaju danych poddawanych klasyfikacji [11]. Kolejnymi zmiennymi wpływającymi na działanie sieci jest momentum, współczynnik uczenia, ilość iteracji uczenia oraz funkcja aktywacji (uzależniona od jej parametrów). Liczba kombinacji powyższych parametrów jest nieskończona a wykonanie pełnych testów niemożliwe. Z tego powodu przeprowadzone badania nie są kompletne i wyczerpujące, przybliżają jedynie możliwe do osiągnięcia wyniki.

Jako funkcję aktywacji przyjęto funkcję sigmoidalną o parametrach $\alpha = 1$ i $\beta = 1$. Pozostałe funkcje aktywacji nie były testowane. Przeprowadzone badania wykazały, iż dla sieci z trzema warstwami ukrytymi i 50 neuronami na każdej, wyniki klasyfikacji dla przygotowanych zbiorów były zadowalające. Dla powyższych parametrów najlepsze wyniki osiągnięto dla współczynniku uczenia 0,01 i momentum 0,1. Dokumentacja OpenCV zaleca użycie współczynnika klasyfikacji równego 0,1 jednak dla takiej wartości sieć nie była w stanie klasyfikować żadnego z przygotowanych zbiorów (skuteczność uczenia i testowania wynosiła 0, musiał więc wystąpić błąd zaimplementowanego algorytmu). Zmniejszanie współczynnika uczenia poniżej 0,01 nieznaczenie wydłużało czas uczenia i nie przynosiło widocznych zmian w skuteczności klasyfikacji, podobnie momentum. Jako ilość iteracji przyjęto 1000 powtórzeń. Sieć o opisanych powyżej parametrach przetestowano na pierwszych trzech zestawach zbiorów cech. Poniżej znajduje się uproszczona tabela wyników (Tabela 37) zawierająca średnie i mediany dla każdego zbioru danych zamiast wszystkich trzech wyników (z pominięciem zbiorów zawierających 10 cech kształtu, dla których wyniki wahały się nawet o 82 punktów procentowych), „n” oznacza dane znormalizowane, „un” nieznormalizowane.

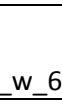
Zbiory cech składające się z momentów hu zostały bardzo słabo sklasyfikowane, podobnie zbiory zawierające 10 cech kształtu (część tych zbiorów została jednak sklasyfikowana na poziomie 96%). Spośród zbiorów cech gęstościowych wybrano podział *8x8* i sprawdzono jaka struktura sieci byłaby dla niego najlepsza. Przetestowano warianty sieci z jedną, dwiema i trzeba warstwami. Liczba neuronów na pierwszej warstwie zmieniała się od 10 do 100, a na drugiej od 10 do 50 zaś na trzeciej od 30 do 70, zaś liczba iteracji od 100 do 900. Na podstawie przeprowadzonych badań zdecydowano się wybrać sieć trzy warstwową z odpowiednio 70, 50 i 70 neuronami i 400 iteracjami. Nie była to sieć dająca największą skuteczność klasyfikacji, jednak była ona mniejsza jedynie o 0,1 punktu procentowego zaś czas jej szkolenia był prawie dwukrotnie krótszy od najefektywniejszej sieci. Dodatkowo mniejsza liczba iteracji zmniejszała ryzyko przeuczenia sieci.

Tabela 37. Wyniki testowania klasyfikatora na różnych zbiorach danych. Każdy wiersz przedstawia średnie czasy uczenia, klasyfikacji i klasyfikacji jednej litery, mediany skuteczności klasyfikacji zbioru uczącego i testowego oraz maksymalną i minimalną skuteczność klasyfikacji zbioru testowego dla 3 zestawów danego typu cech.

Dane	Mediana skuteczności klasyfikacji zbioru uczącego [%]	Min. skuteczność klasyfikacji zbioru testowego [%]	Max. skuteczność klasyfikacji zbioru testowego [%]	Mediana skuteczność klasyfikacji zbioru testowego [%]	Śr. czas uczenia [s]	Śr. czas klasyfikacji [ms]	Śr. czas klasyfikacji jednej litery [ms]
7hu n	2,8	2,8	14,3	2,8	2164,765	468,667	0,052
7hu un	2,8	2,8	3,3	2,8	2133,755	463,000	0,052
7hu, otwory n	3,6	2,9	4,8	3,6	2173,213	469,000	0,053
7hu, otwory un	2,8	2,7	4,7	2,8	2177,422	468,333	0,052
3x3 n	99,8	96,4	98,0	96,5	2242,427	520,667	0,058
3x3 un	99,9	95,3	98,2	97,5	2099,177	468,333	0,052
4x4 n	2,8	2,8	2,8	2,8	2249,021	489,667	0,055
4x4 un	2,8	2,8	2,8	2,8	2251,229	484,333	0,054
4x5 n	100,0	99,3	99,6	99,4	2178,631	494,667	0,055
4x5 un	100,0	98,7	99,7	99,3	2293,479	505,333	0,057
5x5 n	100,0	99,4	99,8	99,8	2160,375	500,000	0,056
5x5 un	100,0	99,5	99,9	99,8	2055,490	505,333	0,057
6x6 n	100,0	99,4	99,8	99,6	1710,078	536,333	0,060
6x6 un	100,0	99,6	99,9	99,7	2097,203	521,000	0,058
7x7 n	100,0	99,5	99,7	99,7	2219,016	577,667	0,065
7x7 un	100,0	99,5	99,8	99,7	2348,683	562,667	0,063
8x8 n	100,0	99,6	99,9	99,7	2610,146	593,667	0,066
8x8 un	100,0	99,6	99,9	99,7	2817,209	609,000	0,068
9x9 n	100,0	99,4	99,7	99,7	2278,719	635,333	0,071
9x9 un	100,0	99,4	99,7	99,7	2728,396	635,333	0,071
13x13 n	100,0	99,3	99,7	99,5	3632,802	812,333	0,091
13x13 un	100,0	99,2	99,7	99,6	2421,484	807,333	0,090
16x16 n	100,0	99,4	99,7	99,6	3984,771	1036,667	0,116
16x16 un	100,0	99,5	99,7	99,5	4221,469	1031,000	0,115

W Tabeli 38 zamieszono błędnie sklasyfikowane znaki dla zbioru *8x8* cech gęstościowych z 9 zestawu.

Tabela 38. Błędnie sklasyfikowane znaki i ich poprawne klasy z 9 zestawu 8x8 cech gęstościowych .

Znak	Odgadnięty znak	Opis próbki	Obraz próbki
0	O	0_i_2_w_6	
1	I	1_n_1_n_	
I	X	I_m_6_w_2	
O	0	O_i_6_w_1	
O	0	O_b_4_w_3	
O	0	O_n_4_w_3	

Dla najlepiej klasyfikowanego zbioru cech gęstościowych *8x8* wykonano testy klasyfikatora na zbiorach testowych zapisanych innymi czcionkami niż zbiory uczące – wyniki ujęto w Tabeli 39 poniżej.

Tabela 39. Wyniki klasyfikacji dodatkowego zbioru testowego 8x8 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji [%]
1	dane 8x8.cechyn	93,6
2	dane 8x8.cechyn	94,6
3	dane 8x8.cechyn	94,3
4	dane 8x8.cechyn	92,9
5	dane 8x8.cechyn	93,8
6	dane 8x8.cechyn	94,3
7	dane 8x8.cechyn	93,6
8	dane 8x8.cechyn	94,1
9	dane 8x8.cechyn	93,8
10	dane 8x8.cechyn	94,3

Minimalna skuteczność klasyfikacji wynosiła 92,9%, maksymalna 94,6% a mediana 93,9% co skutkowało odpowiednio 41, 31 i 35 źle sklasyfikowanymi znaki spośród 576. Mediana skuteczności klasyfikacji była mniejsza od skuteczność klasyfikacji podstawowych zbiorów testowych o 5,8 punktu procentowego.

Ostatnim wykonanym testem MLP była próba stworzenia trzech współdziałających ze sobą klasyfikatorów w celu dokonania poprawnej klasyfikacji. Poniżej znajdują się tabele (Tabela 40, Tabela 41, Tabela 42, Tabela 43) przedstawiające wyniki kolejno dla

klasyfikatora podejmującego decyzję, czy dany znak jest litera czy cyfrą, klasyfikatora klasyfikującego litery i klasyfikatora klasyfikującego cyfry.

Tabela 40. Wyniki klasyfikacji dla klasyfikatora rozróżniającego cyfry od liter na zbiorze 8x8 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 8x8 n	99,98	99,5
2	dane 8x8 n	99,98	99,5
3	dane 8x8 n	99,97	99,3
4	dane 8x8 n	99,95	99,4
5	dane 8x8 n	99,95	99,6
6	dane 8x8 n	100,00	99,2
7	dane 8x8 n	100,00	99,3
8	dane 8x8 n	100,00	99,5
9	dane 8x8 n	100,00	99,3
10	dane 8x8 n	99,95	99,5

Tabela 41. Wyniki klasyfikacji dla klasyfikatora rozpoznającego cyfry na zbiorze 8x8 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 8x8 same cyfry n	100,0	100,0
2	dane 8x8 same cyfry n	100,0	100,0
3	dane 8x8 same cyfry n	100,0	100,0
4	dane 8x8 same cyfry n	100,0	99,9
5	dane 8x8 same cyfry n	100,0	100,0
6	dane 8x8 same cyfry n	100,0	100,0
7	dane 8x8 same cyfry n	100,0	100,0
8	dane 8x8 same cyfry n	100,0	100,0
9	dane 8x8 same cyfry n	100,0	99,9
10	dane 8x8 same cyfry n	100,0	100,0

Tabela 42. Wyniki klasyfikacji dla klasyfikatora rozpoznającego litery na zbiorze 8x8 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 8x8 same litery n	100,0	100,0
2	dane 8x8 same litery n	100,0	100,0
3	dane 8x8 same litery n	100,0	100,0
4	dane 8x8 same litery n	100,0	100,0
5	dane 8x8 same litery n	100,0	100,0
6	dane 8x8 same litery n	100,0	100,0
7	dane 8x8 same litery n	100,0	99,9
8	dane 8x8 same litery n	100,0	99,9
9	dane 8x8 same litery n	100,0	100,0
10	dane 8x8 same litery n	100,0	100,0

Minimalna skuteczność klasyfikacji dla klasyfikatora podejmującego decyzję wynosiła 99,2%, maksymalna 99,6%, a mediana 99,5%, przełożyło się odpowiednio na 22, 11 i 16 źle sklasyfikowanych znaków. Dla klasyfikatora cyfr i klasyfikatora liter mediana skuteczność klasyfikacji dla zbioru testowego wynosiła 100%. Badania wykazały, iż powyższe podejście przyniosło bardzo dobre efekty, skuteczność klasyfikacji była mniejsza od podstawowego podejścia jedynie o 0,2 punktu procentowego.

W wyniku przeprowadzonych eksperymentów ustalono, iż najlepszym spośród przetestowanych zbiorów cech dla MLP był zbiór cech gęstościowych *8x8*. Otrzymane dla wybranego zbioru wyniki klasyfikacji zbioru uczącego, testowego, oraz dodatkowego zbioru testowego wynosiły odpowiednio: 100%, 99,7% i 93,9%.

3.7 SVM (maszyna wektorów nośnych)

Klasyfikator SVM w OpenCV zaimplementowany został w klasie `CvSVM`. W celu utworzenia działającego klasyfikatora należało utworzyć obiekt klasy i wywołać dla niego metodę `train()` lub `train_auto()` dostarczając jej jako jeden z argumentów obiekt klasy `CvSVMParams` służący do konfigurowania parametrów klasyfikatora. Testowanie stworzonego klasyfikatora przebiegało w następujący sposób - kolejne znaki ze zbioru uczącego i testowego podawane były jako argument metody predict(), a wyniki klasyfikacji porównywane były z klasami badanych znaków. Poniżej znajduje się przykładowy kod.

```
CvMat* cvmZbiorDanych;
CvMat* cvmZbiorUczacy;
CvMat* cvmKlasyZbioruDanych;
CvMat* cvmKlasyZbioruUczacego;
CvMat* cvmKlasyZbioruTestowego;

WczytajDane(&cvmZbiorDanych, &cvmKlasyZbioruDanych);

//tworzenie wektora klas zbioru uczacego
cvmKlasyZbioruUczacego = cvCreateMat(iWielkoscZbioruUczacego, 1, CV_32S);

for(i = 0; i < iWielkoscZbioruUczacego; i++){
      int iKlasa = 0;

      if(cvmKlasyZbioruDanych->data.fl[i] >= 48
                  && cvmKlasyZbioruDanych ->data.fl[i] <= 57)
            iKlasa = cvRound(cvmKlasyZbioruDanych ->data.fl[i]) - 22;
      else
            iKlasa = cvRound(cvmKlasyZbioruDanych ->data.fl[i]) - 'A';

      float* fWektor = (float*)(cvmKlasyZbioruUczacego->data.ptr
                                    + i* cvmKlasyZbioruUczacego->step);

      fWektor[0] = iKlasa;
```

```
}

//tworzenie zbioru uczącego
cvGetRows(cvmZbiorDanych, &cvmZbiorUczacy, 0, iWielkoscZbioruUczacego);

//ustawianie parametrów
CvSVMParams cvsvmParametry = CvSVMParams();
cvsvmParametry.svm_type = 100;
cvsvmParametry.kernel_type = 2;
cvsvmParametry.gamma = 0.03375;
cvsvmParametry.C = 312.5;

//mierzenie czasu szkolenia
int iSzkolenieStart = clock();

//szkolenie
CvSVM cvsSVM.train(&cvmZbiorUczacy, cvmKlasyZbioruUczacego, 0, 0,
cvsvmParametry);

int iSzkolenieKoniec = clock();
int iCzasSzkolenia = iSzkolenieKoniec – iSzkolenieStart;

//testowanie klasyfikatora
int iPoprawnaKlasyfikacjaUczacy = 0;
int iPoprawnaKlasyfikacjaTestowy = 0;
iTestowanieStart = clock();

for( i=0; i< cvmZbiorDanych->rows; i++{
      CvMat cvmObiekt;
      cvGetRow(cvmZbiorDanych, &cvmObiekt, i);
      int iKlasa = cvsSVM.predict(&cvmObiekt);

      if(iKlasa >= 26 && iKlasa <= 35)
            iKlasa = iKlasa + 22;
      else
            iKlasa = iKlasa + 'A';

      int iKlasyfikacja = fabs(iKlasa – cvmKlasyZbioruDanych->data.fl[i]) <
      FLT_EPSILON ? 1 : 0;

      if(i < iWielkoscZbioruUczacego)
            iPoprawnaKlasyfikacjaUczacy += iKlasyfikacja;
      else
            iPoprawnaKlasyfikacjaTestowy += iKlasyfikacja;
}

int iTestowanieKoniec = clock();
int iCzasTestowania = iTestowanieKoniec – iTestowanieStart;

double dSkutecznoscDlaUczacego = (double) iPoprawnaKlasyfikacjaUczacy /
(double) iWielkoscZbioruUczacego * 100;
double dSkutecznoscDlaTestowego = (double) iPoprawnaKlasyfikacjaTestowy /
(double)(cvmZbiorDanych->rows -  iWielkoscZbioruUczacego) * 100;
```

Pierwszym krokiem podczas testowania klasyfikatora było wybranie jednej z dwóch dostępnych metod klasyfikacji a dla niej typu jądra i odpowiednich parametrów. W tym celu wyszkolono klasyfikatory przy pomocy funkcji `train_auto()` dla metody C_SVC i NU_SVC i różnych funkcji jądra: LINEAR, POLY, RBF, SIGMOID. Najkrótszy czas uczenia uzyskano przy użyciu funkcji LINEAR a wyniki klasyfikacji zbioru testowego przekraczały 99%. Dla RBF czas uczenia był dłuższy (we wszystkich przypadkach mniejszy niż 50s) a wyniki klasyfikacji zbioru uczącego i testowego nieznacznie lepsze. Dla funkcji POLY i SIGMOID klasyfikator szkolił się bardzo długo (powyżej 3 godzin). W wyniku przeprowadzonych badań wybrano metodę C_SVC i funkcję jądra RBF. Przy użyciu funkcji `train_auto()` dla 10 walidacji krzyżowych na zbiorach cech gęstościowych wyznaczono parametry gamma i C. Dla wszystkich zbiorów wyznaczane parametry były takie same, jedynie dla zbiorów *13x13* i *16x16* parametr C był mniejszy. Pomimo tego wynik klasyfikacji dla zbiorów *13x13* i *16x16* przy użyciu wartości parametrów wyznaczonych dla innych zbiorów gęstościowych był taki sam jak wynik dla parametrów wyznaczonych dla tych zbiorów.

Z przeprowadzonych badań wynika, iż normalizacja danych, na których operuje klasyfikator SVM jest konieczna. Najwyższą skuteczność otrzymano dla zbiorów *16x16* i *9x9, chociaż* jest ona zbliżona do skuteczności klasyfikacji zbiorów *8x8,* i *13x13*. Wraz ze wzrostem liczby cech zawsze wzrastał czas szkolenia klasyfikatora zaś skuteczność klasyfikacji wzrastała do pewnego momentu a potem utrzymywała się na w miarę stałym poziomie. Jako najlepszy zbiór wybrany został zbiór cech gęstościowych *9x9,* ponieważ czas uczenie był ponad dwukrotnie krótszy niż czas uczenia dla zbioru *16x16*. Dla większości zbiorów cech gęstościowych skuteczność klasyfikacji danych uczących wynosiła 100%. Dla wybranego zbioru 81 cech gęstościowych *9x9* średni czas uczenia klasyfikatora wynosił 15 s, czas klasyfikacji wszystkich znaków ze zbioru 16 s a pojedynczego znaku 1,8 ms. Najgorsza klasyfikacja zbioru testowego dla danych gęstościowych *9x9* wynosiła 99,5 % a najlepsza 99,9% co przekładało się na błędną klasyfikację 17 i 3 znaków spośród 2884. Dla mediany skuteczności klasyfikacji równej 99,8% błędna klasyfikacja wystąpiła dla 7 znaków. Zbliżoną skuteczność klasyfikacji klasyfikator osiągnął dla wariantów zbiorów numer 2, 5, 8 i 9. Kolejne źle sklasyfikowane znaki dla zbioru numer 9 zostały przedstawione w Tabeli 43.

Tabela 43. Błędnie sklasyfikowane znaki i ich poprawne klasy z 9 zestawu 9x9 cech gęstościowych.

Znak	Wynik klasyfikacji	Opis próbki	Obraz próbki
0	O	0_i_1_w_6	
0	O	0_i_2_w_6	
0	O	0_n_2_w_6	
0	O	0_n_2_w_2	
1	I	1_b_6_n	
1	I	1_m_2_n	
1	I	1_m_2_w_2	

Poniżej znajduje się uproszczona tabela wyników (Tabela 44) zawierająca średnie i mediany dla każdego zbioru danych zamiast wszystkich dziesięciu wyników, „n” oznacza dane znormalizowane, „un” nieznormalizowane.

Tabela 44. Wyniki testowania klasyfikatora na różnych zbiorach danych. Każdy wiersz przedstawia średnie czasy uczenia, klasyfikacji i klasyfikacji jednej litery, mediany skuteczności klasyfikacji zbioru uczącego i testowego oraz maksymalną i minimalną skuteczność klasyfikacji zbioru testowego dla 10 zestawów danego typu cech.

Dane	Mediana skuteczności klasyfikacji zbioru uczącego [%]	Min. skuteczność klasyfikacji zbioru testowego [%]	Max. skuteczność klasyfikacji zbioru testowego [%]	Mediana skuteczność klasyfikacji zbioru testowego [%]	Śr. czas uczenia [s]	Śr. czas klasyfikacji [s]	Śr. czas klasyfikacji jednej litery [ms]
7hu n	8,5	6,6	14,2	8,5	12,769	24,175	2,708
7hu un	23,8	21,3	25,2	23,3	17,125	17,314	1,939
7hu, otwory n	20,2	14,2	21,9	19,4	16,338	16,780	1,879
7hu, otwory un	33,6	30,7	36,1	33,6	17,274	12,576	1,409
10 cech ksztaltu n	80,8	78,4	80,7	79,8	13,189	7,150	0,801
10 cech ksztaltu un	100,0	4,8	5,8	5,4	26,506	28,412	3,182

10 cech kształtu, otwory n	84,6	83,1	85,1	83,9	13,655	6,775	0,759
10 cech kształtu, otwory un	100,0	4,9	5,8	5,3	27,320	28,848	3,231
10 cech kształtu, otwory, 7hu n	88,6	87,1	88,2	87,5	14,435	8,217	0,920
10 cech kształtu, otwory, 7hu un	100,0	4,8	6,1	5,1	29,712	30,907	3,462
3x3 n	93,6	91,1	92,8	92,3	12,222	4,025	0,451
3x3 un	100,0	6,0	6,4	6,1	21,388	27,784	3,112
4x4 n	98,8	97,9	98,7	98,0	11,858	4,008	0,449
4x4 un	100,0	4,1	5,7	5,2	21,534	29,985	3,358
4x5 n	99,3	98,5	99,2	98,9	11,141	3,997	0,448
4x5 un	100,0	4,7	5,7	4,9	23,301	31,222	3,497
5x5 n	99,3	98,7	99,5	99,1	9,364	4,327	0,485
5x5 un	100,0	4,9	6,0	5,3	24,808	33,110	3,709
6x6 n	99,8	99,5	99,8	99,7	10,720	6,033	0,676
6x6 un	100,0	4,4	5,4	4,6	30,480	37,018	4,146
7x7 n	99,9	99,4	99,8	99,7	11,195	8,291	0,929
7x7 un	100,0	4,5	5,4	5,0	35,501	41,664	4,667
8x8 n	99,9	99,7	99,9	99,7	12,838	11,547	1,293
8x8 un	100,0	4,3	5,3	4,6	42,042	46,232	5,178
9x9 n	100,0	99,5	99,9	99,8	14,867	15,966	1,788
9x9 un	100,0	4,4	5,5	4,9	49,261	52,537	5,885
13x13 n	100,0	99,6	99,8	99,7	27,092	25,877	2,898
13x13 un	100,0	4,5	5,3	4,8	86,148	80,634	9,032
16x16 n	100,0	99,4	99,9	99,8	41,633	40,578	4,545
16x16 un	100,0	4,6	5,5	5,1	132,044	108,120	12,110

Dla zbioru cech gęstościowych *9x9* wykonano testy klasyfikatora na zbiorach testowych zapisanych innymi czcionkami niż zbiory uczące – wyniki ujęto w Tabeli 45 poniżej.

Tabela 45. Wyniki klasyfikacji dodatkowego zbioru testowego 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji [%]
1	dane 9x9.cechyn	98,8
2	dane 9x9.cechyn	98,6
3	dane 9x9.cechyn	98,6
4	dane 9x9.cechyn	98,8
5	dane 9x9.cechyn	99,0
6	dane 9x9.cechyn	98,5
7	dane 9x9.cechyn	98,7
8	dane 9x9.cechyn	98,7
9	dane 9x9.cechyn	98,8
10	dane 9x9.cechyn	98,7

Minimalna skuteczność klasyfikacji wynosiła 98,5%, maksymalna 99% a mediana 98,7% co skutkowało odpowiednio 8, 6 i 7 źle sklasyfikowanymi znaki spośród 576. Mediana skuteczności klasyfikacji była mniejsza od skuteczność klasyfikacji podstawowych zbiorów testowych o 1,1 punktu procentowego. Dla pozostałych zbiorów cech, zgodnie z oczekiwaniami, skuteczność klasyfikacji była niższa niż dla wybranego *9x9*.

Ostatnim wykonanym testem SVM była próba stworzenia trzech współdziałających ze sobą klasyfikatorów w celu dokonania poprawnej klasyfikacji. Poniżej znajdują się tabele (Tabela 46, Tabela 47, Tabela 48, Tabela 49) przedstawiające wyniki kolejno dla klasyfikatora podejmującego decyzję, czy dany znak jest litera czy cyfrą, klasyfikatora klasyfikującego litery i klasyfikatora klasyfikującego cyfry.

Tabela 46. Wyniki klasyfikacji dla klasyfikatora rozróżniającego cyfry od liter na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9.cechyn	97,0	96,9
2	dane 9x9.cechyn	99,4	98,9
3	dane 9x9.cechyn	98,6	97,9
4	dane 9x9.cechyn	97,0	96,3
5	dane 9x9.cechyn	98,7	98,1
6	dane 9x9.cechyn	98,4	98,1
7	dane 9x9.cechyn	99,2	99,1
8	dane 9x9.cechyn	97,9	97,8
9	dane 9x9.cechyn	99,6	99,2
10	dane 9x9.cechyn	97,6	97,2

Tabela 47. Wyniki klasyfikacji dla klasyfikatora rozróżniającego cyfry od liter na zbiorze 9x9 cech gęstościowych przy użyciu parametrów wyznaczonych specjalnie dla tego dwuklasowego zbioru danych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9.cechyn	99,6	99,4
2	dane 9x9.cechyn	99,3	99,5
3	dane 9x9.cechyn	99,6	99,2
4	dane 9x9.cechyn	99,5	99,5
5	dane 9x9.cechyn	99,5	99,4
6	dane 9x9.cechyn	99,5	99,6
7	dane 9x9.cechyn	99,6	99,2
8	dane 9x9.cechyn	99,4	98,9
9	dane 9x9.cechyn	99,6	99,2
10	dane 9x9.cechyn	99,5	99,3

Tabela 48. Wyniki klasyfikacji dla klasyfikatora rozpoznającego cyfry na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same cyfry.cechyn	100,0	100,0
2	dane 9x9 same cyfry.cechyn	100,0	100,0
3	dane 9x9 same cyfry.cechyn	100,0	100,0
4	dane 9x9 same cyfry.cechyn	100,0	100,0
5	dane 9x9 same cyfry.cechyn	100,0	100,0
6	dane 9x9 same cyfry.cechyn	100,0	100,0
7	dane 9x9 same cyfry.cechyn	100,0	100,0
8	dane 9x9 same cyfry.cechyn	100,0	100,0
9	dane 9x9 same cyfry.cechyn	100,0	100,0
10	dane 9x9 same cyfry.cechyn	100,0	100,0

Tabela 49. Wyniki klasyfikacji dla klasyfikatora rozpoznającego litery na zbiorze 9x9 cech gęstościowych.

Zestaw	Dane	Skuteczność klasyfikacji zbioru uczącego [%]	Skuteczność klasyfikacji zbioru testowego [%]
1	dane 9x9 same litery.cechyn	100,0	100,0
2	dane 9x9 same litery.cechyn	100,0	100,0
3	dane 9x9 same litery.cechyn	100,0	99,9
4	dane 9x9 same litery.cechyn	100,0	100,0
5	dane 9x9 same litery.cechyn	100,0	100,0
6	dane 9x9 same litery.cechyn	100,0	100,0
7	dane 9x9 same litery.cechyn	100,0	100,0
8	dane 9x9 same litery.cechyn	100,0	99,9
9	dane 9x9 same litery.cechyn	100,0	99,9
10	dane 9x9 same litery.cechyn	100,0	100,0

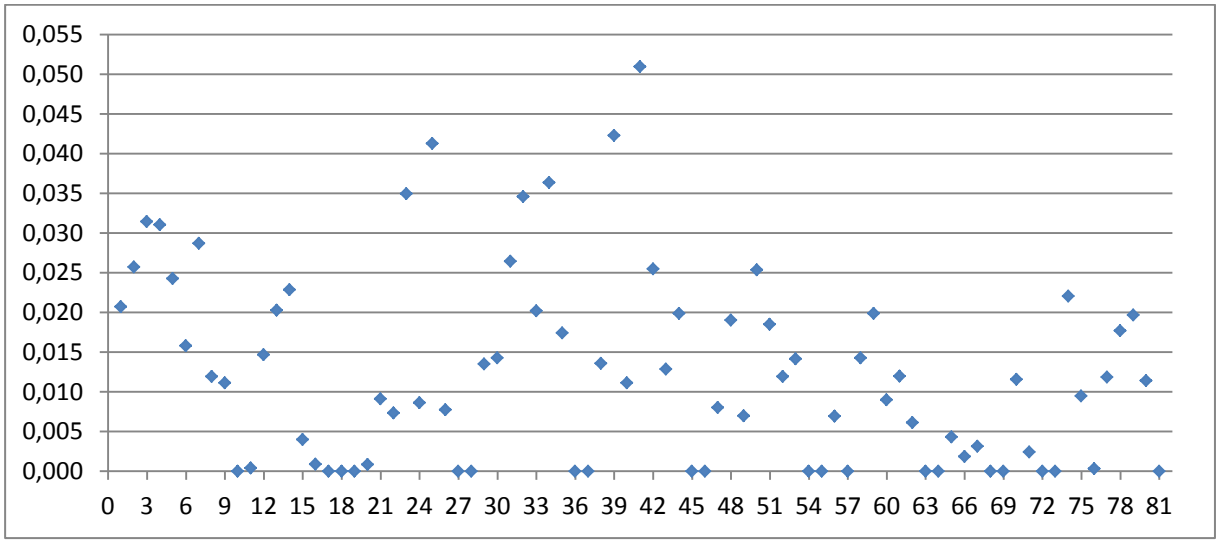
Minimalna skuteczność klasyfikacji dla klasyfikatora podejmującego decyzję wynosiła 96,3%, maksymalna 99,2%, a mediana 98,0%, przełożyło się odpowiednio na 105, 23 i 58 źle sklasyfikowanych znaków. Zastosowanie specjalnie dobranych parametrów do klasyfikacji zbioru dwuklasowego przyniosło znaczną poprawę wyników klasyfikacji, minimalna skuteczność wzrosła do 98,9%, maksymalna do 99,6 a mediana do 99,3% dzięki czemu liczba źle sklasyfikowany znaków spadła do odpowiednio 32, 11 i 20. Dla klasyfikatora cyfr mediana skuteczność klasyfikacji dla zbioru testowego wynosiła 100 zaś dla klasyfikatora liter skuteczność minimalna wynosiła 99,9%, maksymalna 100%, a mediana 100%, w rezultacie czego źle sklasyfikowanych zostało odpowiednio 3, 0 i 0 liter. Pomimo bardzo dobrej skuteczności klasyfikatorów liter i cyfr, niska skuteczność klasyfikatora dokonującego

rozróżnienia czy dany znak jest literą czy cyfrą czyniła powyższe rozwiązanie gorszym od podejścia podstawowego.

W wyniku przeprowadzonych eksperymentów ustalono, iż najlepszym spośród przetestowanych zbiorów cech dla SVM był zbiór cech gęstościowych *9x9*. Otrzymane dla wybranego zbioru wyniki klasyfikacji zbioru uczącego, testowego, oraz dodatkowego zbioru testowego wynosiły odpowiednio: 100%, 99,8% i 98,7%.

3.8. Ważność cech danych gęstościowych

W przeprowadzonych badaniach dla zbiorów cech gęstościowych wykorzystywane zostały wszystkie wyznaczone cechy. Na podstawie analizy obrazów znaków z naniesioną na nie siatką reprezentującą poszczególne podziały na obszary cech, autor zauważył, iż część z obszarów dla większości lub wszystkich znaków pozostaje pusta. Obszary takie teoretycznie nie wpływały na klasyfikację, gdyż nie wnosiły żadnych dodatkowych informacji na temat znaku. Usunięcie ich nie powinno więc było mieć żadnego negatywnego efektu na działanie klasyfikatora a mogłoby skrócić czas uczenia i klasyfikacji. W celu wyznaczenia ważności poszczególnych cech wykorzystano binarne drzewa decyzyjne a jako badany zbiór wybrano zbiór cech gęstościowych *9x9*, który był najlepszym zbiorem danych dla większości klasyfikatorów. Ważności każdej z 81 cech, wyznaczonych dla zbioru uczącego U_0 składającego się z 6084 znaków ze zbioru podstawowego, zostały zamieszczone na Rysunku 15.



Rysunek 15. Ważności 81 cech gęstościowych dla zbioru uczącego U_0 składającego się z 6084 znaków ze zbioru podstawowego.

Na podstawie wyznaczonych ważności 81 cech dla zbioru U_0 wybrano trzy podzbiory cech. Pierwszy składał się z 19 cech, dla których ważność była większa od 0,02, drugi z 27 cech, dla których ważność była większa od 0,015, trzeci zaś z 42 cech, dla których ważność była większa od 0,01. Przy pomocy wyznaczonych zestawów cech, wytrenowano klasyfikatory MLP, SVM, binarne drzewo decyzyjne, AdaBoost oraz kNN dla k =3 i przetestowano ich skuteczność klasyfikacji dla zbioru podstawowego i dodatkowego U_1. Wyniki klasyfikacji zostały przedstawione w Tabeli 50 poniżej.

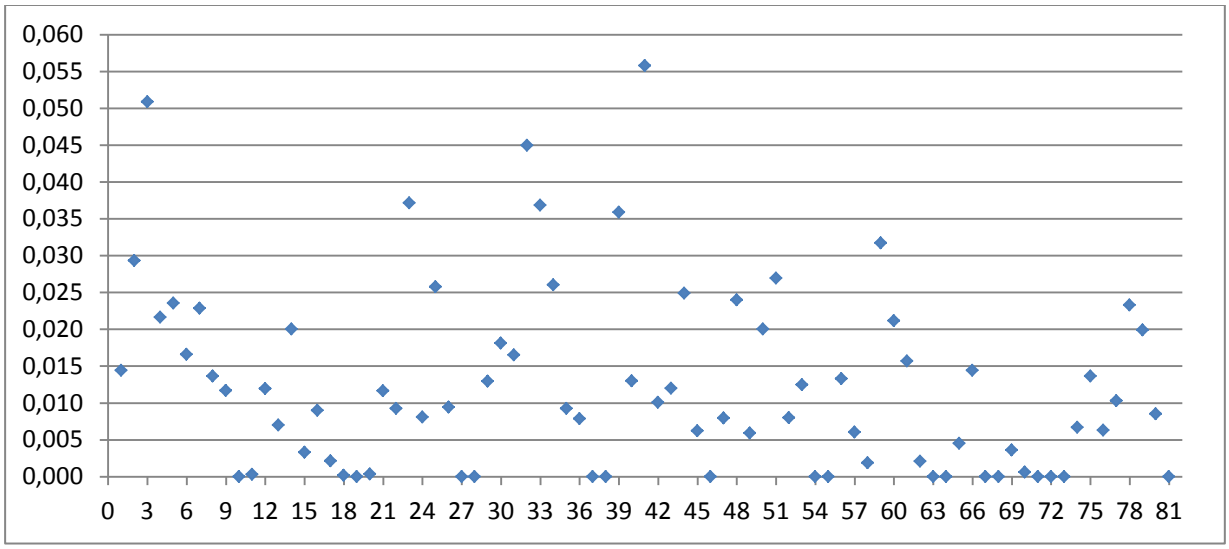
Tabela 50. Wyniki klasyfikacji zbioru U_1podstawowego i dodatkowego przy użyciu klasyfikatorów wyszkolonych w oparciu o pełny zestaw 81 cech oraz o trzy podzestawy składające się z 19, 27 oraz 42 cech wyznaczone w oparciu o zbiór U_0.

Zbiór	Liczba cech	SVM	MLP	AdaBoost	kNN k=3	Binarne drzewo decyzyjne
Podstawowy uczący	81	99,95%	100,00%	100,00%	99,69%	100,00%
Podstawowy uczący	42	99,92%	100,00%	100,00%	99,74%	100,00%
Podstawowy uczący	27	99,85%	100,00%	100,00%	99,52%	100,00%
Podstawowy uczący	19	95,27%	79,44%	99,46%	96,92%	100,00%
Podstawowy testowy	81	99,86%	99,90%	99,59%	99,47%	93,95%
Podstawowy testowy	42	99,97%	99,90%	99,65%	99,54%	94,16%
Podstawowy testowy	27	99,82%	99,72%	99,40%	99,16%	93,39%
Podstawowy testowy	19	94,20%	77,11%	94,83%	92,79%	87,17%
Dodatkowy	81	95,66%	93,06%	95,83%	95,49%	77,43%
Dodatkowy	42	95,83%	93,93%	95,53%	95,83%	78,47%
Dodatkowy	27	91,32%	89,41%	93,92%	91,49%	75,00%
Dodatkowy	19	81,25%	60,42%	79,86%	76,73%	66,84%

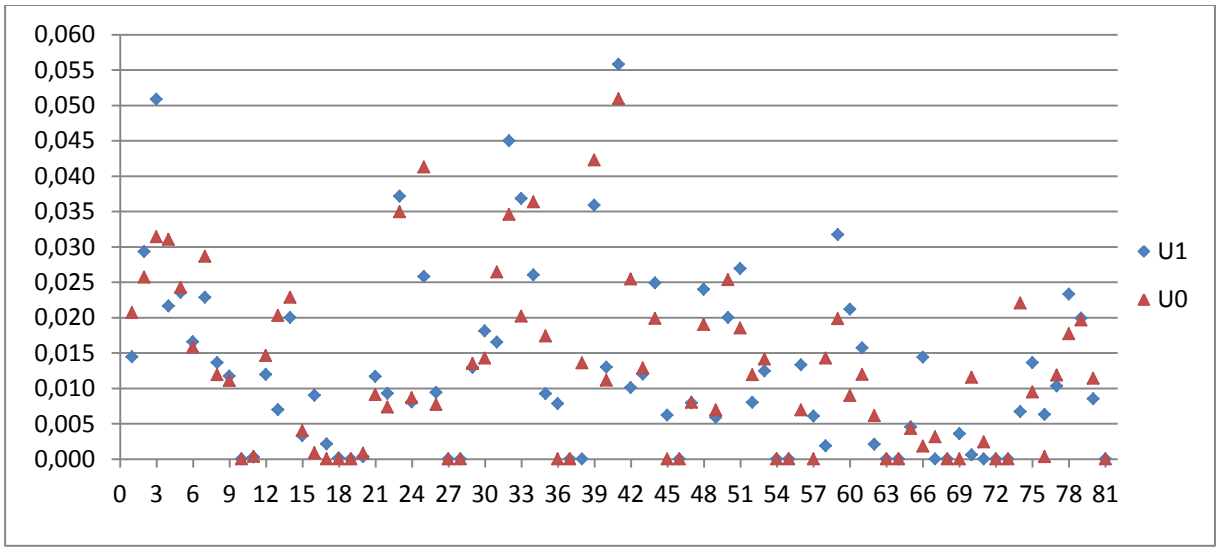
Badania wykazały, iż ograniczenie liczby cech do 42 dla klasyfikatorów SVM, kNN i binarnego drzewa decyzyjnego skutkowało nieznacznym zwiększeniem skuteczności klasyfikacji. MLP podobnie jak pozostałe klasyfikatory, przy zbyt małej liczbie cech dokonywały gorszej klasyfikacji, dla podstawowego zbioru testowego wynik uzyskany dla 42 cech był taki sam jak dla 81 cech, zaś dla zbioru dodatkowego lepsze wyniki uzyskano dla 42 cech. Klasyfikator AdaBoost najlepiej klasyfikował wszystkie zbiory dla 81 cech..

Użyte zestawy cech wyznaczone zostały na podstawie ważności cech dla zbioru U_0 przetestowane zaś zostały dla zbioru U_1. Następny test polegał na wyznaczeniu w podobny sposób zestawów cech na podstawie ich ważność dla zbioru U_1 (Rysunek 16). Prócz progów ważności cech równych 0,02, 0,015, 0,01 użyto dwóch dodatkowych: 0,005 oraz > 0. Liczba cech dla poszczególnych zakresów ważności okazała się inna niż przy wcześniej wygenerowanych podzbiorach, odpowiednio: 20, 25, 39, 54 i 64. Jak widać na Rysunku 17

ważności poszczególnych cech dla zbiorów U_0 i U_1były różne co uzasadnione było innym wymieszanie obu zbiorów uczących.



Rysunek 16. Ważność 81 cech dla zbioru uczącego U_1 składającego się z 6084 znaków ze zbioru podstawowego.



Rysunek 17. Ważność 81 cech dla zbioru uczącego U_0 oraz **U_1.**

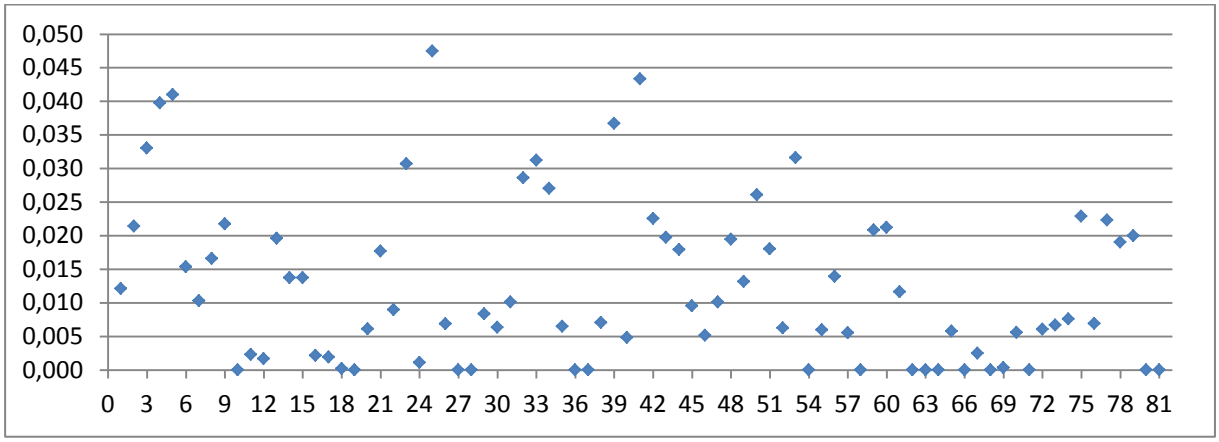
Wyniki klasyfikacji zbioru U_1 przy użyciu zbiorów cech wyznaczonych w oparciu o ważności cech dla zbioru U_1 (Tabela 51) nie pokryły się z tymi dla wcześniej przeprowadzonych testów. Binarne drzewo decyzyjne osiągnęło najlepsze wyniki dla wszystkich 81 cech, chociaż dla progu ważności 0,01 (39 cech) skuteczność klasyfikacji dodatkowego zbioru była taka sama jak dla 81 cech. kNN dla k=3 zbiór uczący klasyfikował najlepiej dla progu ważności 0,005 (54 cechy), zbiór testowy dla progu 0,01 zaś zbiór dodatkowy dla progu większego od 0. AdaBoost najlepszej klasyfikacji dokonało dla cech z ważnością większą od 0,005. MLP zbiór testowy najlepiej klasyfikowało dla wszystkich 81 cech zaś dodatkowy zbiór dla progu większego od 0. SVM zbiór testowy najlepiej klasyfikowało dla progu 0,01

zaś dodatkowy zbiór dla wszystkich 81 cech. Na podstawie uzyskanych wyników nie można było wskazać żadnej prawidłowości pomiędzy progiem ważności cech a uzyskanymi wynikami klasyfikacji dla wszystkich klasyfikatorów.

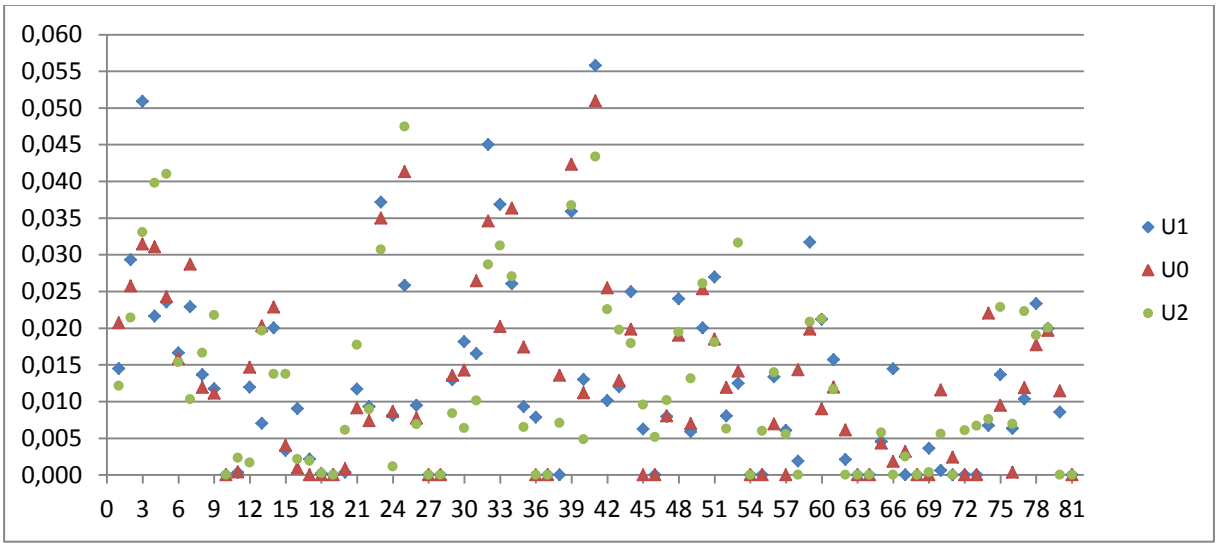
Tabela 51. Wyniki klasyfikacji zbioru U_1 przy użyciu klasyfikatorów wyszkolonych w oparciu o pełny zestaw 81 cech oraz o pięć podzestawów składającychsię z 20, 25, 3, 54 oraz 64 cech.

Zbiór	Liczba cech	SVM	MLP	AdaBoost	kNN k=3	Binarne drzewo decyzyjne
Podstawowy uczący	81	99,95%	100,00%	100,00%	99,69%	100,00%
Podstawowy uczący	64	99,95%	100,00%	100,00%	99,72%	100,00%
Podstawowy uczący	54	99,95%	100,00%	100,00%	99,75%	100,00%
Podstawowy uczący	39	99,87%	100,00%	100,00%	99,70%	100,00%
Podstawowy uczący	25	99,75%	100,00%	100,00%	99,52%	100,00%
Podstawowy uczący	20	99,51%	100,00%	100,00%	99,39%	100,00%
Podstawowy testowy	81	99,86%	99,90%	99,58%	99,47%	93,95%
Podstawowy testowy	64	99,86%	99,86%	99,58%	99,54%	93,95%
Podstawowy testowy	54	99,90%	99,86%	99,61%	99,54%	93,95%
Podstawowy testowy	39	99,97%	99,83%	99,58%	99,61%	94,06%
Podstawowy testowy	25	99,79%	99,58%	99,26%	99,30%	93,35%
Podstawowy testowy	20	99,51%	99,05%	98,77%	98,80%	92,51%
Dodatkowy	81	95,66%	93,06%	95,83%	95,49%	77,43%
Dodatkowy	64	95,14%	94,27%	95,49%	95,66%	77,43%
Dodatkowy	54	94,62%	93,58%	96,01%	95,49%	77,26%
Dodatkowy	39	94,62%	91,32%	94,44%	95,14%	77,43%
Dodatkowy	25	91,67%	90,28%	90,80%	90,97%	74,31%
Dodatkowy	20	89,76%	86,81%	90,92%	87,32%	75,17%

Kolejny test wykonano na nowym zbiorze U_2 oraz podzestawach cech utworzonych w oparciu o wyznaczone ważności dla tego zbioru (wyniki w Tabeli 52). Ważność cech została przedstawiona na Rysunku 18 zaś na Rysunku 19 umieszone zostało zestawienie ważności cech dla wszystkich trzech zbiorów U, dla których były one wyznaczane. Ważność poszczególnych cech dla różnych zbiorów uczących (z kilkoma wyjątkami, jak np. zerowe cechy 18, 19, 37, 38, 54, 55 lub wysoce znaczące jak 39 i 41) była przeważnie różna przez co nie można było wyznaczyć jednego zbioru ważnych cech dla wszystkich zbiorów.



Rysunek 18. Ważność 81 cech dla zbioru uczącego U_2 składającego się z 6084 znaków ze zbioru podstawowego.



Rysunek 19. Ważność 81 cech dla zbioru uczącego U_0, U_1 oraz U_2.

Wyniki klasyfikacji zbioru U_2 przy użyciu zestawów cech wyznaczonych dla tego zbioru zostały umieszczone w Tabeli 52. Jedynie dla klasyfikatora AdaBoost wyniki klasyfikacji w zależności od zbioru cech dla zbioru U_2 pokryły się z tymi dla zbioru U_1 - najlepsze wyniki uzyskano dla cech o ważności większej od 0,005.

Przeprowadzone badania wykazały, iż zmniejszenie liczby cech poprzez odrzucenie tych, które dostarczają mniej danych o badanym obiekcie, mogło zwiększyć skuteczność klasyfikacji wybranych klasyfikatorów. Ważności poszczególnych cech wyznaczane przy użyciu binarnego drzewa decyzyjnego były inne dla każdego z użytych zbiorów uczących, tak więc można założyć, iż dobór podzbioru cech powinien być wykonywany przed każdym rozpoczęciem trenowania klasyfikatora dla nowego zbioru uczącego. Przeprowadzone testy dla binarnego drzewa decyzyjnego potwierdziły, iż w jego przypadku odrzucenie cech o zerowej ważności nie wpływa na skuteczność klasyfikacji zmniejszając jednak nieznacznie czas szkolenia klasyfikatora. Wyznaczane w badaniach podzbiory cech pozwalały poprawić

skuteczność klasyfikacji wszystkich klasyfikatorów, jednakże jedynie w przypadku klasyfikatora AdaBoost jeden podzbiór cech gwarantował poprawę klasyfikacji zarówno zbioru podstawowego testowego oraz dodatkowego – dla innych klasyfikatorów lepsze wyniki uzyskiwano dla różnych podzbiorów cech dla zbioru testowego podstawowego oraz dodatkowego. Wyniki testów przeprowadzone dla zbioru U_1 przy użyciu podzbiorów cech wyznaczonych w oparciu o zbiór U_0wykazały, iż istnieje możliwość dobrania takiego podzbioru cech, który pozwoliłby poprawić skuteczność klasyfikacji zarówno zbioru podstawowego uczącego i testowego jak i zbioru dodatkowego – choć dobór ten nie może być realizowany przez binarne drzewo decyzyjne lecz przez jedne z wielu metod doboru optymalnych zbiorów cech.

Tabela 52. Wyniki klasyfikacji zbioru U_2 przy użyciu klasyfikatorów wyszkolonych w oparciu o pełny zestaw 81 cech oraz o pięć podzestawów składającychsię z 19, 29, 38, 56 oraz 65 cech.

Zbiór	Liczba cech	SVM	MLP	AdaBoost	kNN k=3	Binarne drzewo decyzyjne
Podstawowy uczący	81	100,00%	100,00%	100,00%	99,75%	100,00%
Podstawowy uczący	65	100,00%	100,00%	100,00%	99,84%	100,00%
Podstawowy uczący	56	100,00%	100,00%	100,00%	99,82%	100,00%
Podstawowy uczący	38	99,97%	100,00%	100,00%	99,75%	100,00%
Podstawowy uczący	29	99,93%	100,00%	100,00%	99,75%	100,00%
Podstawowy uczący	19	99,52%	99,98%	100,00%	99,39%	100,00%
Podstawowy testowy	81	99,79%	99,82%	99,58%	99,19%	93,67%
Podstawowy testowy	65	99,72%	99,58%	99,58%	99,26%	93,67%
Podstawowy testowy	56	99,68%	99,75%	99,61%	99,40%	93,53%
Podstawowy testowy	38	99,75%	99,68%	99,58%	99,40%	93,14%
Podstawowy testowy	29	99,58%	99,51%	99,26%	99,19%	93,07%
Podstawowy testowy	19	98,84%	98,66%	98,77%	98,45%	91,60%
Dodatkowy	81	95,66%	92,54%	95,83%	95,83%	78,65%
Dodatkowy	65	95,49%	91,84%	94,49%	95,66%	78,65%
Dodatkowy	56	94,79%	92,88%	96,01%	95,49%	78,82%
Dodatkowy	38	93,93%	91,67%	94,44%	94,27%	76,91%
Dodatkowy	29	93,58%	90,28%	90,80%	93,06%	76,56%
Dodatkowy	19	86,81%	84,55%	90,97%	86,46%	74,48%

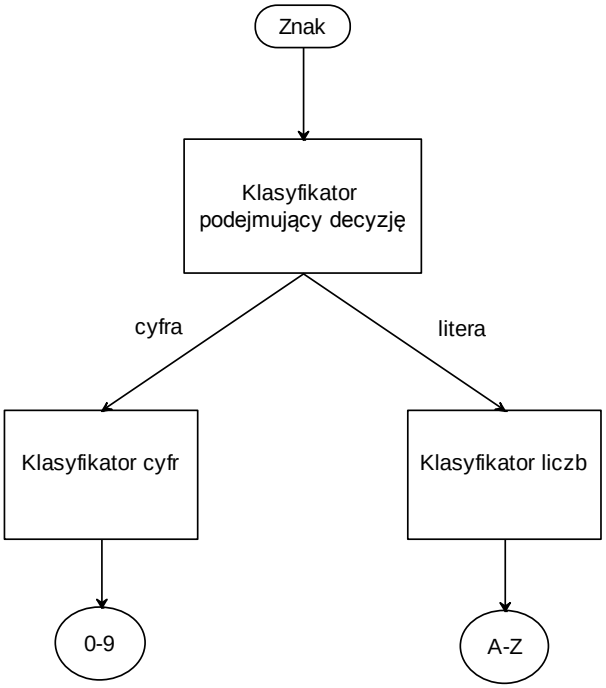
4 Testy klasyfikatora równoległego i równoległo-kaskadowego

4.1. Opis klasyfikatorów

W poprzednim rozdziale opisane zostały badania przeprowadzone dla poszczególnych typów klasyfikatorów. Klasyfikacja znaków odbywała się głównie w oparciu o ocenę dokonaną przez jeden klasyfikator – nazywany klasyfikatorem całościowym. Przetestowane zostało również inne podejście do tworzenia klasyfikatora. Zamiast używać jednego klasyfikatora do klasyfikacji znaków alfanumerycznych, stworzono trzy klasyfikatory tego samego typu, z których każdy pełnił inną funkcję. Pierwszy podejmował decyzję, czy dany znak był literą czy też cyfrą, zaś pozostałe dwa klasyfikowały odpowiednio same cyfry lub same litery. Klasyfikator ten nazwany został klasyfikatorem kaskadowym. Przeprowadzone doświadczenia wykazały, iż w każdym z analizowanych przypadków klasyfikator kaskadowy klasyfikował gorzej niż klasyfikator całościowy. Niezadowalająca skuteczność klasyfikacji wynikała przeważnie z niedostatecznej skuteczności podczas podejmowania decyzji czy dany znak jest literą czy też cyfrą – nieprzekraczającą 99,63%. W celu poprawienia efektywności klasyfikatora kaskadowego należało więc usprawnić jego zdolność do rozróżniania liter od cyfr.

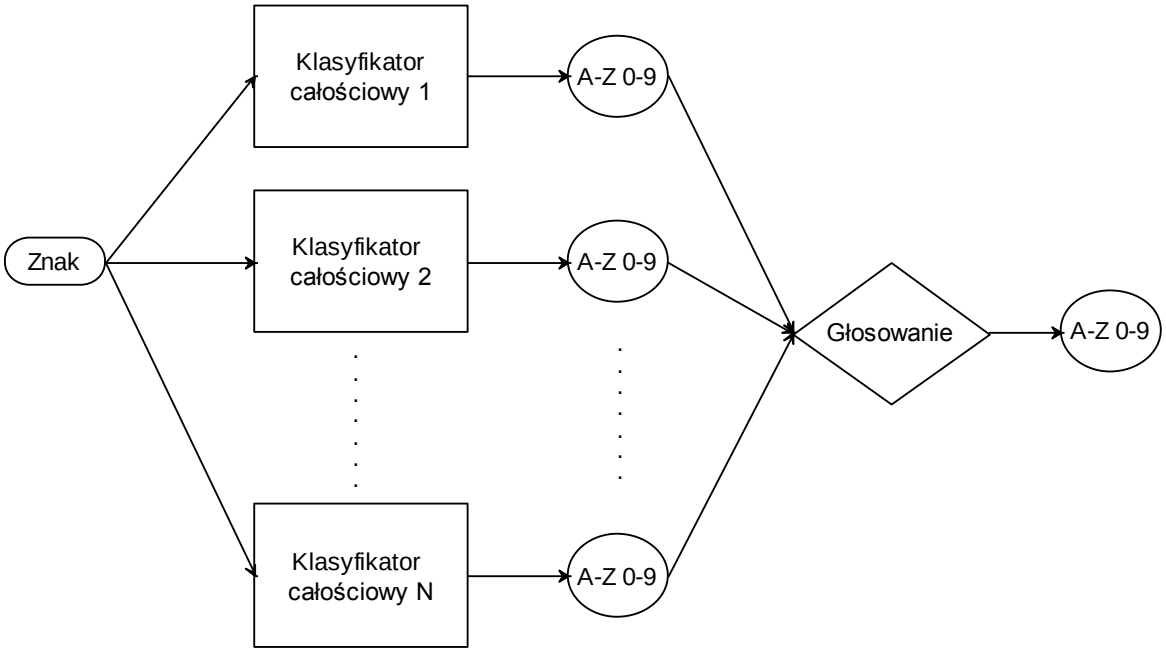
Tabela 53. Mediany skuteczności podejmowania decyzji, klasyfikacji cyfr oraz klasyfikacji liter dla wszystkich badanych klasyfikatorów.

Klasyfikator	Zbiór	Mediana skuteczności rozróżniania cyfr od liter [%]	Mediana skuteczności klasyfikacji liter [%]	Mediana skuteczności klasyfikacji cyfr [%]
Bayes	6x6 n	86,41	99,71	99,62
kNN k=1	16x16 n	99,63	99,83	99,87
kNN k=3	9x9 n	99,49	99,85	100,00
Binarne drzewo decyzyjne	9x9 n	95,45	96,03	97,72
Las losowy	9x9 n	98,88	99,64	99,81
AdaBoost	9x9 n	99,54	99,90	100,00
MLP	8x8 n	99,46	100,00	100,00
SVM	9x9 n	99,30	100,00	100,00

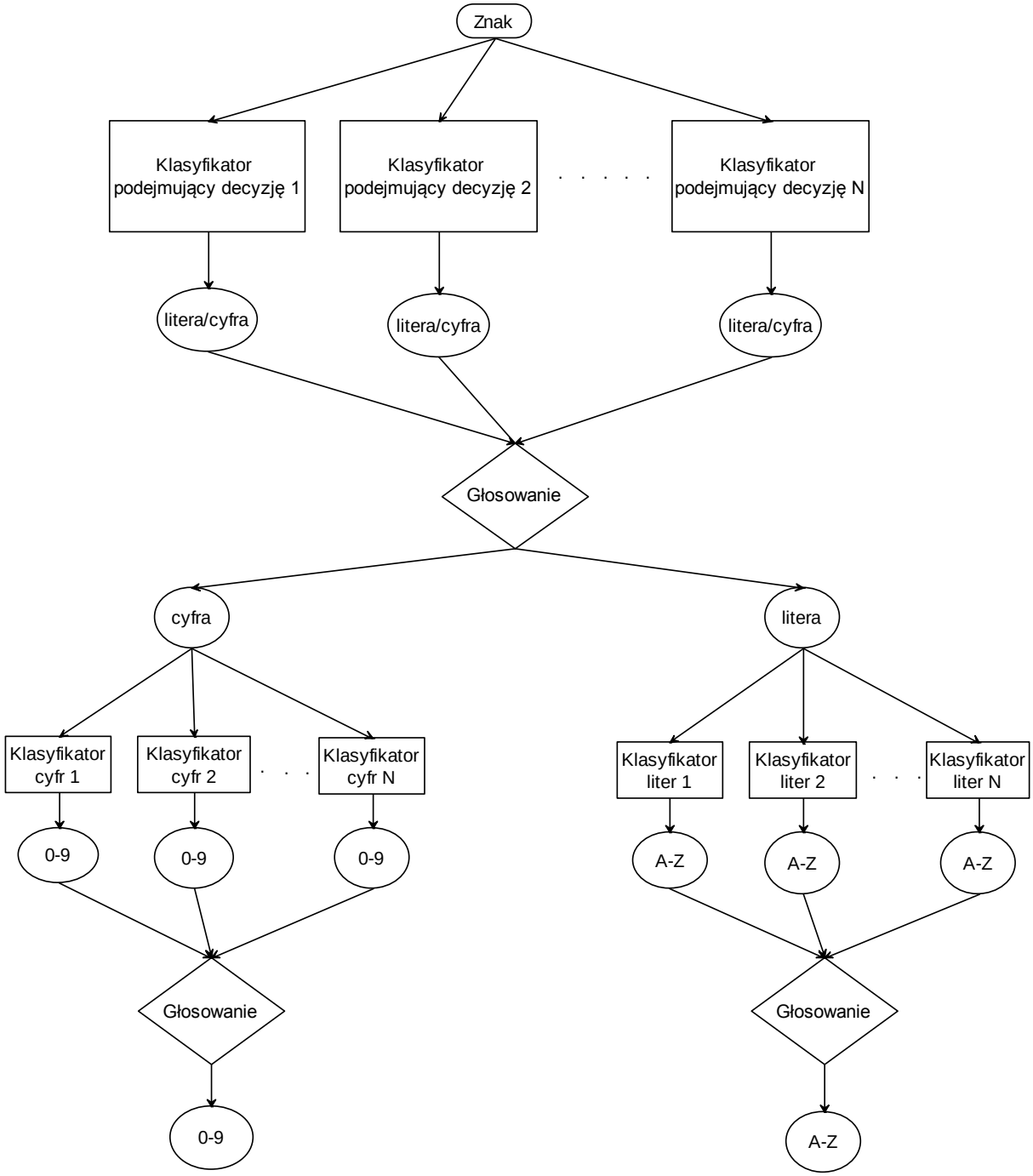


Rysunek 20. Schemat klasyfikatora kaskadowego.

Przeprowadzone testy wykazały, iż zbiory błędnie sklasyfikowanych znaków dla każdego z klasyfikatorów nie były takie same. Bazując na tej obserwacji autor zdecydował się zastąpić jeden klasyfikator kilkoma różnymi klasyfikatorami a wynik klasyfikacji uzależnić od przeprowadzonego przez nie „głosowania" większościowego. Zbudowany w ten sposób klasyfikator nazwany został klasyfikatorem równoległym. Klasyfikator składający się z kilku różnych klasyfikatorów dokonujących tej samej klasyfikacji nazwany został klasyfikatorem równoległym. Klasyfikator kaskadowy używający na każdym etapie klasyfikatora równoległego zamiast klasyfikatora całościowego nazwany został klasyfikatorem równoległo-kaskadowym.



Rysunek 21. Schemat klasyfikatora równoległego.



Rysunek 22. Schemat klasyfikatora równoległo-kaskadowego.

4.2. Konstrukcja klasyfikatorów i ich testy na zbiorze podstawowym i dodatkowym

Do budowy klasyfikatora równoległo-kaskadowego użyto klasyfikatorów SVM, MLP i AdaBoost, ze względu na ich wysoką skuteczność odróżniania liter od cyfr oraz klasyfikacji liter i cyfr. Powyższe klasyfikatory zostały wytrenowane na zbiorze uczącym składającym się z 6084 znaków. Jako wynik klasyfikacji przyjmowano klasę wskazaną przez przynajmniej dwa klasyfikatory. W przypadku niewystąpienia takiej sytuacji klasa określana była na

podstawie odpowiednio: AdaBoost podczas rozróżniania cyfr od liter, MLP podczas klasyfikowania liter oraz SVM podczas klasyfikowania cyfr.

Powyższy klasyfikator równoległo-kaskadowy przetestowany został na dwóch zbiorach danych, nazwanych podstawowym i dodatkowym. W skład pierwszego wchodziło 8955 znaków (w tym 6084 ze zbioru uczącego) zapisanych tymi samymi czcionkami co zbiór uczący. Drugi składał się z 576 znaków zapisanych innymi czcionkami niż zbiór uczący. Podczas klasyfikacji zbioru podstawowego zostało błędnie sklasyfikowanych 7 znaków – skuteczność 99,92% , zaś dla zbioru dodatkowego 28 – skuteczność 95,10%. Dla porównania wyniki klasyfikacji tych samych zbiorów klasyfikatorami kaskadowymi zostały zamieszczone w Tabeli 54 poniżej.

Tabela 54. Wyniki klasyfikacji zbioru podstawowego i dodatkowego klasyfikatorami kaskadowymi SVM, MLP i AdaBoost oraz klasyfikatorem równoległo-kaskadowym AdaBoost-MLP-SVM.

Klasyfikator	Liczba błędnie sklasyfikowanych znaków dla zbioru podstawowego	Skuteczność klasyfikacji zbioru podstawowego [%]	Liczba błędnie sklasyfikowanych znaków dla zbioru dodatkowego	Skuteczność klasyfikacji zbioru dodatkowego [%]
SVM	57	99,36	33	94,27
MLP	20	99,78	49	91,49
AdaBoost	12	99,87	35	93,92
Równoległo-kaskadowy AdaBoost-MLP-SVM	7	99,92	28	95,10

Klasyfikator równoległo-kaskadowy zapewnił więc lepszą skuteczność klasyfikacji niż same klasyfikatory kaskadowe. Znaczne różnice otrzymano w porównaniu z klasyfikatorami kaskadowymi SVM i MLP, jednakże AdaBoost osiągnął zbliżone wyniki.

Oba zbiory – podstawowy i dodatkowy, zostały sklasyfikowane przy pomocy klasyfikatorów całkowitych SVM, MLP i AdaBoost aby porównać ich skuteczność z zaproponowanym klasyfikatorem równoległo-kaskadowym, wyniki umieszczono w Tabeli 55 poniżej.

Tabela 55. Wyniki klasyfikacji zbioru podstawowego i dodatkowego klasyfikatorami całościowymi SVM, MLP i AdaBoost.

Klasyfikator	Liczba błędnie sklasyfikowanych znaków dla zbioru podstawowego	Skuteczność klasyfikacji zbioru podstawowego [%]	Liczba błędnie sklasyfikowanych znaków dla zbioru dodatkowego	Skuteczność klasyfikacji zbioru dodatkowego [%]
SVM	5	99,94	26	95,49
MLP	7	99,92	38	93,40
AdaBoost	11	99,88	25	95,66

Klasyfikator równoległo-kaskadowy klasyfikował lepiej zbiór podstawowy niż klasyfikatory AdaBoost i tak samo jak MLP i gorzej niż SVM, natomiast zbiór dodatkowy klasyfikował gorzej niż SVM i AdaBoost zaś lepiej niż MLP. Nie można więc jednoznacznie orzec, które z rozwiązań jest bardziej skuteczne.

Kolejnym przebadanym klasyfikatorem był klasyfikator równoległy składający się z klasyfikatorów całościowych SVM, MLP i AdaBoost. W przypadku, gdy co najmniej dwa klasyfikatory nie wskazały tej samej klasy, klasa znaku określana była na podstawie oceny dokonanej przez SVM. Powyższy klasyfikator równoległy podczas klasyfikacji zbioru podstawowego sklasyfikował błędnie 3 znaki – skuteczność 99,97% , zaś podczas klasyfikacji zbioru dodatkowego 24 – skuteczność 95,8%. Tym samym klasyfikator równoległy okazał się skuteczniejszy od wszystkich wcześniej przetestowanych rozwiązań.

Analiza błędnie sklasyfikowanych znaków i odpowiedzi poszczególnych klasyfikatorów biorących udział w ustalaniu klasy znaków wykazała, iż w części przypadków jeden z klasyfikatorów wskazywał dobrą klasę lub dobrze identyfikował znak jako cyfrę lub literę, lecz jego ocena była odrzucana przez pozostałe dwa klasyfikatory. Próbą poprawienia tej sytuacji było dodanie kolejnych dwóch klasyfikatorów biorących udział w głosowaniu, co mogłoby poprawić klasyfikację w trudnych przypadkach. Wybranymi dodatkowymi klasyfikatorami były klasyfikatory kNN, jeden dla k=3 a drugi minimalno-odległościowy. Wyniki klasyfikacji zbioru podstawowego i dodatkowego przy użyciu klasyfikatora kaskadowego kNN dla k=3, kaskadowego minimalno-odległościowego i klasyfikatorów całościowych: kNN dla k=3 i minimalno-odległościowego, zostały przedstawione w Tabeli 56 i Tabeli 57 poniżej.

Tabela 56. Wyniki klasyfikacji zbioru podstawowego i dodatkowego klasyfikatorami kaskadowymi kNN dla k=3 i minimalno-odległościowym.

Klasyfikator	Liczba błędnie sklasyfikowanych znaków dla zbioru podstawowego	Skuteczność klasyfikacji zbioru podstawowego [%]	Liczba błędnie sklasyfikowanych znaków dla zbioru dodatkowego	Skuteczność klasyfikacji zbioru dodatkowego [%]
kNN k=3	15	99,83	31	94,62
kNN k=1	9	99,90	24	95,83

Tabela 57. Wyniki klasyfikacji zbioru podstawowego i dodatkowego klasyfikatorami całościowymi kNN dla k=3 i minimalno-odległościowym.

Klasyfikator	Liczba błędnie sklasyfikowanych znaków dla zbioru podstawowego	Skuteczność klasyfikacji zbioru podstawowego [%]	Liczba błędnie sklasyfikowanych znaków dla zbioru dodatkowego	Skuteczność klasyfikacji zbioru dodatkowego [%]
kNN k=3	13	99,85	23	96,01
kNN k=1	11	99,88	29	94,97

Jako wynik klasyfikacji dla klasyfikatora równoległo-kaskadowego składającego się z pięciu różnych klasyfikatorów (AdaBoost, SVM, MLP, kNN dla k=3 i minimalno-odległościowy) przyjmowano klasę wskazaną przez przynajmniej trzy z nich. W przypadku niewystąpienia takiej sytuacji wybierana była klasa wskazana przez dwa klasyfikatory. Jeżeli w wyniku głosowania dwie różne klasy otrzymały po dwa głosy, wybierana była odpowiednio klasa wskazana przez: klasyfikator minimalno-odległościowy podczas podejmowania decyzji, MLP podczas klasyfikacji liter i SVM podczas klasyfikacji cyfry. W sytuacji gdy każdy klasyfikator wskazał inną klasę wybierana była klasa oszacowana przez: klasyfikator minimalno-odległościowy podczas podejmowania decyzji, MLP podczas klasyfikacji liter i SVM podczas klasyfikacji cyfry. Powyższy klasyfikator równoległo-kaskadowy podczas testów sklasyfikował błędnie 6 znaków ze zbioru podstawowego – skuteczność 99,93% , oraz 26 znaków ze zbioru dodatkowego – skuteczność 95,50%. Jego skuteczność nie okazała się więc znacząco lepsza od klasyfikatora równoległo kaskadowego AdaBoost-MLP-SVM, zaś czas klasyfikacji zwiększył się prawie trzykrotnie. Dodatkowo powyższy klasyfikator nie okazał się skuteczniejszy od przetestowanego wcześniej klasyfikatora równoległego.

W ostatnim przeprowadzonym badaniu przetestowano działanie klasyfikatora równoległego składającego się z pięciu różnych klasyfikatorów całościowych: AdaBoost, SVM, MLP, kNN dla k=3 i minimalno-odległościowego. Wybór klasy przebiegał w ten sam sposób co w klasyfikatorze równoległo kaskadowym składającym się z pięciu różnych klasyfikatorów – oczywiście zamiast etapów rozróżniania cyfr od liter oraz klasyfikacji cyfr lub liter, występował tylko jeden etap klasyfikacji znaku. Podczas testów otrzymano następujące wyniki: 4 źle sklasyfikowane znaki ze zbioru podstawowego – skuteczność 99,96% oraz 20 źle sklasyfikowanych znaków ze zbioru dodatkowego – skuteczność 96,5%. W porównaniu z klasyfikatorem równoległym SVM-MLP-AdaBoost, testowany klasyfikator podczas rozpoznawania zbioru podstawowego pomylił się o jeden raz więcej, ale za to o cztery razy mniej podczas klasyfikacji zbioru dodatkowego. Można więc przypuszczać, iż testowany klasyfikator w praktyce okaże się skuteczniejszy gdyż przeważnie rozpoznawane

będą znaki niezapisane czcionkami użytymi w zbiorze podstawowym, a zbiór dodatkowy klasyfikowany był o jeden punkt procentowy lepiej. Średni czas klasyfikacji testowanego klasyfikatora wynosił 19 ms czyli ponad pięć razy dłużej niż klasyfikatora równoległego SVM-MLP-AdaBoost.

Należy również nadmienić, iż rozróżnienie cyfry 0 od litery O w wielu testowanych przypadkach było bardzo trudne – nawet dla człowieka. Pominięcie błędnych klasyfikacji cyfry 0 jako litery O oraz litery O jako cyfry 0 spowodowało polepszenie wyników klasyfikacji, co zostało przestawione w Tabeli 58.

Tabela 58. Wyniki klasyfikacji zbioru podstawowego i dodatkowego, z pominięciem błędnych klasyfikacji 0 i O - przy użyciu wszystkich przetestowanych klasyfikatorów.

Klasyfikator	Typ	Liczba błędnie sklasyfikowanych znaków dla zbioru podstawowego	Skuteczność klasyfikacji zbioru podstawowego [%]	Liczba błędnie sklasyfikowanych znaków dla zbioru dodatkowego	Skuteczność klasyfikacji zbioru dodatkowego [%]
NN-3NN-SVM-MLP-AdaBoost	równoległy	1	99,99	16	97,2
SVM-MLP-AdaBoost	równoległy	1	99,99	20	96,5
MLP	calosciowy	1	99,99	32	94,4
SVM	calosciowy	3	99,97	22	96,2
NN-3NN-SVM-MLP-AdaBoost	równoległo kaskadowy	4	99,96	22	96,2
AdaBoost-MLP-SVM	równoległo kaskadowy	4	99,96	24	95,8
kNN k=1	kaskadowy	6	99,93	20	96,5
AdaBoost	calosciowy	6	99,93	21	96,4
kNN k=1	calosciowy	6	99,93	25	95,7
kNN k=3	calosciowy	7	99,92	18	96,9
kNN k=3	kaskadowy	9	99,90	24	95,8
AdaBoost	kaskadowy	9	99,90	31	94,6
MLP	kaskadowy	17	99,81	45	92,2
SVM	kaskadowy	44	99,51	29	95,0

4.3. Testy na zbiorach znaków z tablic rejestracyjnych

Dla pięciu najskuteczniejszych klasyfikatorów zostały przeprowadzone dodatkowe testy przy użyciu zbioru składającego się ze znaków alfanumerycznych pochodzących ze zdjęc 93 tablic rejestracyjnych. W procesie automatycznej segmentacji obrazu z tablicami zostało wyodrębnionych 606 znaków spośród 608. Otrzymane znaki różniły się od siebie wielkością,

część z nich była niewyraźna, zaszumiona, przekrzywiona lub zlewała się z innymi elementami tablicy rejestracyjnej (Rysunek 23). Nie wszystkie znaki reprezentowane były przez tą samą liczbę próbek – dokładne liczby próbek przypadające na każdy znak zostały zamieszczone w Tabeli 59 poniżej.

Tabela 59. Liczby wystąpień poszczególnych znaków na 93 badanych tablicach rejestracyjnych.

Znak	Liczba	Znak	Liczba	Znak	Liczba
0	27	C	4	O	1
1	33	D	2	P	2
2	38	E	23	Q	0
3	28	F	10	R	19
4	36	G	10	S	12
5	29	H	3	T	7
6	34	I	3	U	9
7	30	J	8	V	2
8	30	K	12	W	51
9	30	L	45	X	2
A	17	M	8	Y	3
B	20	N	6	Z	13

Rysunek 23. Zestawienie zdjęć 93 tablic rejestracyjnych.

Rysunek 24. Zbinaryzowane znaki wydzielone z 93 tablic rejestracyjnych przedstawionych na Rysunku 18.

Wyniki klasyfikacji powyższego zbioru danych okazały się znacznie gorsze od wyników klasyfikacji zbioru podstawowego i dodatkowego (Tabela 60). Szczegółowa analiza wyników wykazała, iż najgorzej klasyfikowanymi znakami były: litera A, M i W (w tym przypadku wszystkie klasyfikacje były błędne, klasyfikatory W klasyfikowały przeważnie jako N lub 4, A zawsze jako 4, zaś M przeważnie jako N lub 4, w zależności od klasyfikatora). Dodatkowo znaczna część cyfr 5 klasyfikowana była jako litera S. Oczywiście nie były to jedyne źle klasyfikowane znaki, lecz dla pozostałych odsetek błędnych klasyfikacji można było w większości przypadków uznać za akceptowalny.

Tabela 60. Wyniki klasyfikacji znaków z tablic rejestracyjnych przy użyciu 6 najefektywniejszych klasyfikatorów.

Klasyfikator	Typ	Liczba błędnie sklasyfikowanych znaków z tablic	Skuteczność klasyfikacji zbioru znaków z tablic [%]
NN-3NN-SVM-MLP-AdaBoost	równoległy	178	70,63
SVM-MLP-AdaBoost	równoległy	184	69,64
MLP	calosciowy	161	73,43
SVM	calosciowy	209	65,51
NN-3NN-SVM-MLP-AdaBoost	równoległo-kaskadowy	168	72,28
AdaBoost-MLP-SVM	równoległo-kaskadowy	162	73,27

Testy wykazały, iż pomimo bardzo wysokiej skuteczności badanych klasyfikatorów podczas klasyfikacji zbioru podstawowego i dodatkowego, ich skutenoczność przy zbiorze znaków z tablic rejestracyjnych była gorsza nawet o ponad 26 punktów procentowych. Dodatkowo najlepszym klasyfikatorem okazał się MLP, zaś klasyfikatory równoległo-kaskadowe osiągnęły wyniki jedynie nieznacznie gorsze, co nie pokrywało się ze wcześniejszymi testami. Nie udało się niestety jednoznacznie określić, co było powodem tak słabej klasyfikacji – składało się na to zapewne kilka czynników. Krój znaków użytych do trenowania klasyfikatorów, w niektórych przypadkach znacznie odbiegał od kroju znaków z tablic rejestracyjnych. Najlepszym przykładem była litera W – w zbiorze podstawowym jak i dodatkowym zewnętrzne ramiona litery rozchodziły się pod większym kątem a wewnętrzne ramiona stykały się o wiele wyżej (Rysunek 25).

Rysunek 25. Część liter W użytych do trenowania klasyfikatorów.

Znaki z tablic rejestracyjnych często nie były idealnie ustawione w poziomie, tak jak miało to miejsce w zbiorze podstawowym i dodatkowym oraz często były zniekształcone przez prespektywę. Ich wielkość także nie była odpowiednia przez co wymagały znacznego powiększenia (2-5 razy) co powodowało zniekształcenia i szumy. Dodatkowo litery z tablic rejestracyjnych poddane zostały kompresji JPEG, gdzie zbiór podstawowy i dodatkowy zapisane były przy pomocy kompresji bezstratnej TIFF.

4.4. Wnioski

Spośród zbadanych klasyfikatorów dla zbioru podstawowego i dodatkowego najwyższą skutecznością charakteryzowały się klasyfikatory równoległe. Klasyfikatory równoległo-kaskadowe klasyfikowały znaki lepiej od klasyfikatorów kaskadowych i całościowych – z jednym wyjątkiem, całościowym SVM. W większości przypadków klasyfikatory całościowe

okazywały bardziej dokładne od klasyfikatorów kaskadowych, jednakże te drugie klasyfikowały szybciej. Dla zbioru znaków z tablic rejestracyjnych wyniki klasyfikacji były znacznie gorsze (nawet o 26 punktów procentowych) a dodatkowo najlepszym klasyfikatorem okazał się klasyfikator MLP i nieznacznie od niego gorsze klasyfikatory równoległo-kaskadowe, co nie pokrywało się z wcześniejszymi wynikami. Powodem takiej sytuacji była zapewne zła jakość klasyfikowanych znaków, nie pozwalająca precyzyjnie wyznaczyć porządanych cech. W tym przypadku, gdy skuteczność klasyfikacji samych liter i samych syfr nie była tak wysoka jak przy zbiorze podstawowym i dodatkowym, ujawniła się oczekiwana przewaga klasyfikatora równoległo-kaskadowego nad pozostałymi klasyfikatorami.

Wadą klasyfikatorów kaskadowych i równoległo kaskadowych była konieczność wyznaczania różnych wektorów cech dla badanego znaku, co w efekcie wydłużało czas jego klasyfikacji (mierzony w badaniach czas klasyfikacji nie obejmował generowania zestawów). Klasyfikatory równoległo-kaskadowy oraz równoległy składające się z pięciu różnych klasyfikatorów wymagały wygenerowania w sumie 401 cech, gdzie klasyfikator całościowy SVM charakteryzujący się zbliżoną skutecznością wymagał tylko 81. Dodatkowym problem podczas używania klasyfikatorów kaskadowych i równoległo-kaskadowych było ich douczanie. Biorąc pod uwagę fakt, iż jedynie klasyfikatory MLP i AdaBoost mogły zostać doszkolone, wszystkie pozostałe musiały być szkolone od nowa w oparciu o zbiór uczący uzupełniony o nowe znaki. W przypadku klasyfikatora równoległo-kaskadowego składającego się z pięciu różnych klasyfikatorów, dodanie nowego znaku wiązało się z wyszkoleniem lub douczeniem dziesięciu klasyfikatorów (pięć rozróżniających znaki od cyfr oraz pięć klasyfikujących cyfrę lub literę), zaś dla klasyfikatora całościowego SVM należało wyszkolić tylko jeden. Należy również wspomnieć o konieczności przechowywania wyszkolonych klasyfikatorów i ich zbiorów uczących a także czasie koniecznym na wczytanie zapisanych klasyfikatorów aby były gotowe do działania. W większości przypadków kwestia dodatkowego miejsca przestrzeni dyskowej lub operacyjnej mogła zostać pominięta. Istotny mógł się jednak okazać czas przygotowywania klasyfikatorów do pracy. Jeżeli program klasyfikujący uruchamiany był za każdym razem od nowa dla każdego znaku czas przygotowywania klasyfikatorów był kilkudziesięciokrotnie dłuższy niż sama klasyfikacja i w takim przypadku mniejsza ilość klasyfikatorów wyraźnie wpływała na efektywność programu. Podsumowując, w przypadku gdy czas i zasoby systemowe nie były istotne klasyfikatory równoległe i równoległo-kaskadowe stawały się poszukiwanym rozwiązaniem. W przeciwnym wypadku należało skorzystać z klasyfikatorów całościowych lub ewentualnie z klasyfikatora równoległego AdaBoost-SVM-MLP.

Podsumowanie

Przeprowadzone przez autora badania objęły osiem różnych klasyfikatorów. Każdy z nich (za wyjątkiem MLP) został wytrenowany i przetestowany na przynajmniej 340 zbiorach danych [2], co łącznie trwało ponad 160 godzin. Podczas testów mierzona była skuteczność klasyfikacji zbiorów uczących oraz testowych, czas uczenia oraz klasyfikacji znaku, zdolność wytrenowanego klasyfikatora do rozpoznawania znaków zapisanych innymi czcionkami niż zbiory uczące, a także czas wczytywania wytrenowanego klasyfikatora oraz ilość przestrzeni dyskowej wymaganej do jego zapisania. Badany był również wpływ normalizacji danych na uzyskane wyniki, a także stopień konfigurowalności klasyfikatora.

Jako podstawowe kryterium oceny klasyfikatora przyjęto jego skuteczność klasyfikowania zbioru testowego oraz dodatkowego zbioru testowego zapisanego innymi czcionkami niż zbiór uczący. Skuteczność klasyfikacji zbioru uczącego, choć również istotna obrazowała jedynie zdolność klasyfikatora do zapamiętywania danych nie zaś do generalizacji. Badania wykazały, iż skuteczność analizowanych klasyfikatorów zmieniała się w zależności od użytych zbiorów cech. Dlatego też dobranie odpowiedniego zbioru było zadaniem kluczowym a dokonane porównanie klasyfikatorów przeprowadzone zostało w oparciu o najlepsze zestawy cech dla każdego z nich. Najmniej korzystnym zestawem cech okazał się znormalizowany *7hu* – w tym przypadku brak normalizacji poprawiał skuteczność klasyfikacji nawet o kilkanaście punktów procentowych. Najefektywniej klasyfikował go las losowy osiągając 100% na zbiorze uczącym i 84,5% na zbiorze testowym, zaś najgorzej MLP, który sklasyfikowało zbiór na poziomie 3%. Dodanie informacji o otworach znacząco poprawiło skuteczność klasyfikacji, szczególnie dla klasyfikatorów innych niż las losowy (z pominięciem MLP, który tak samo jak w poprzednim przypadku nie potrafił sklasyfikować tego zestawu). Pomimo ogólnej poprawy skuteczności wszystkich klasyfikatorów, to las losowy uzyskały najlepsze wyniki, odpowiednio 99,1% oraz 89,3%. Kolejny zbiór *10 cech kształtu* w przeciwieństwie do *7hu* był skuteczniej klasyfikowany przy zastosowaniu normalizacji. Ponownie najskuteczniejszym klasyfikatorem okazał się las losowy klasyfikujący zbiór uczący ze skutecznością 100% a testowy 85,1%, zaś najmniej efektywnym MLP ze skutecznością 19% dla obu zbiorów. Podobnie jak przy *7hu* dodanie informacji o otworach poprawiło skuteczność klasyfikacji choć już nie w tak znacznym stopniu bo jedynie o kilka punktów procentowych. Najlepsze wyniki uzyskano dla lasu losowego – 100% zbiór uczący, 87,8% zbiór testowy, najgorsze zaś dla MLP, który w zależności od zestawu danych uczących potrafił osiągnąć nawet skuteczność większą od 97%

[2] Bayes, minimalno-odległościowy, kNN k=3, binarne drzewo decyzyjne oraz las losowy przetestowane zostały na 380 zbiorach, zaś MLP na 85.

lecz dla pozostałych nie przekraczającą 20%. Połączenie cech *7hu, 10 cech kształtu* oraz informacji o otworach pozwoliło dalej zwiększyć skuteczność klasyfikacji dla wszystkich klasyfikatorów. W tym przypadku korzystniejsze wyniki uzyskano dla danych nieznormalizowanych. Tak jak przy poprzednich zbiorach najlepszą efektywność osiągnął las losowy, odpowiednio 100% dla zbioru uczącego i 96,3% dla zbioru testowego zaś najgorszą MLP. Nieznacznie gorszą skuteczność od lasu losowego otrzymano dla klasyfikatora AdaBoost – 99,7% dla zbioru uczącego oraz 96% dla zbioru testowego. Ostatnim typem badanych cech były cechy gęstościowe przy zastosowaniu różnych podziałów. W większości analizowanych przypadków nawet mało zagęszczone zestawy jak *3x3* oraz *4x4* przynosiły lepsze efekty niż użycie wcześniej omówionych zbiorów cech.[3] Informacje o najlepszych zbiorach cech oraz skuteczności ich klasyfikacji dla poszczególnych klasyfikatorów znajdują się w Tabeli 61 poniżej.

Tabela 61. Najlepsze zbiory cech i skuteczności ich klasyfikacji dla poszczególnych klasyfikatorów.

Klasyfikator	Zbiór cech	Mediana skuteczności klasyfikacji zbioru uczącego [%]	Mediana skuteczności klasyfikacji zbioru testowego [%]
Bayes	6x6	99,9	99,4
kNN dla k=3	9x9	99,8	99,4
Las losowy	9x9	100	99,4
AdaBoost	9x9	100	99,5
MLP	8x8	100	99,7
SVM	9x9	100	99,8
Minimalno-odległościowy	16x16	100	99,6
Binarne drzewo decyzyjne	9x9	100	93,5

Podczas przeprowadzonych testów zaobserwowano, iż dla wszystkich klasyfikatorów prócz klasyfikatora minimalno-odległościowego oraz AdaBoost, skuteczność klasyfikacji rosła stopniowo, wraz ze wzrostem zagęszczenia podziału a po osiągnięciu pewnego zagęszczenia zaczynała spadać. Można więc przypuszczać, iż zwiększanie gęstości a co za tym idzie liczby cech dla większości klasyfikatorów nie wpływa korzystnie na skuteczność klasyfikacji.[4] Wyjątkiem okazał się klasyfikator minimalno-odległościowy, którego skuteczność rosła wraz z gęstością podziału[5], oraz AdaBoost, dla którego skuteczność

[3] Dla zbioru *10 cech kształtu, 7hu, otwory* Drzewo decyzyjne binarne uzyskiwało lepsze wyniki klasyfikacji niż dla zbiorów *3x3*, *4x4*, *4x5*, zaś AdaBoost dla *4x4*

[4] Przeprowadzone badania nie pozwoliły jednoznacznie stwierdzić czy zbyt duża ilość cech powoduje zmniejszenie skuteczności klasyfikacji poszczególnych klasyfikatorów czy też jest to jedynie wynikiem zbyt gęstego podziału niepozwalającego skutecznie uchwycić subtelnych różnic w znakach.

[5] Ponieważ zbiór *16x16* był najgęstszym użytym zbiorem należałoby wykonać dodatkowe badania dla większych zbiorów w celu potwierdzenia wniosku.

utrzymywałą się na w miarę stałym poziomie. Jako najlepszy zestaw cech wybrano *9x9* cech gęstościowych, dla którego otrzymano najwyższe skuteczności klasyfikacji dla pięciu klasyfikatorów a dodatkowo dla minimalno-odległościowego i MLP były one jedynie nieznacznie gorsze od uzyskanych na najlepszych dla nich zestawach. Normalizacja danych nie wpływała znacząco na wyniki a dla zbiorów gęstszych niż *5x5* wyniki nie różniły się praktycznie wcale. Wyjątkiem okazał się klasyfikator SVM, który nie był w stanie klasyfikować zbiorów danych, które nie zostały wcześniej znormalizowane. Najodpowiedniejszymi zestawami cech okazały się więc cechy gęstościowe i w oparciu o nie dokonane zostało dalsze porównywanie klasyfikatorów.

Według przeprowadzonych badań najskuteczniejszej klasyfikacji zbioru uczącego i testowego dokonywały klasyfikatory SVM, MLP i minimalno-odległościowy (wyniki przedstawiono w Tabeli 61). Pomimo bardzo zbliżonych wyników dla zbiorów podstawowych, klasyfikator SVM okazał się najskuteczniejszym klasyfikatorem znacznie skuteczniej klasyfikując dodatkowy zbiór testowy – 98,7%, podczas gdy minimalno-odległościowy osiągnął 95,1% a MLP tylko 93,9%. Należy zauważyć, iż klasyfikator MLP choć bardzo dobrze klasyfikujący zbiór uczący i testowy, sklasyfikował dodatkowy zbiór testowy gorzej niż większość pozostałych klasyfikatorów – mniej skuteczne okazały się jedynie Bayes i binarne drzewo decyzyjne.

Jako drugie istotne kryterium oceny klasyfikatorów przyjęto czas szkolenia klasyfikatora i czas klasyfikacji jednego znaku. Analizując zebrane pomiary zauważono, iż wraz ze wzrostem liczby cech wzrastał czas klasyfikacji dla wszystkich klasyfikatorów prócz binarnego drzewa decyzyjnego i lasu losowego. Dla pierwszego czas klasyfikacji zmieniał się niezależnie od liczby cech, choć dla cech gęstościowych pojawiła się pewna regularność – dla parzystej liczby cech czas był krótszy niż dla nieparzystej (np. czas dla zbioru *3x3* był dłuższy niż dla zbioru *16x16*). Dla drugiego wraz ze wzrostem liczby cech czas klasyfikacji malał. Średnie czasy uczenia i klasyfikacji jednego znaku dla poszczególnych klasyfikatorów zostały umieszczone w Tabeli 62 poniżej.

Tabela 62. Średnie czasy uczenia i klasyfikacji jednego znaku dla poszczególnych klasyfikatorów.

Klasyfikator	Średni czas uczenia [s]	Średni czas klasyfikacji jednego znaku [ms]
Bayes	0,21	0,42
kNN dla k=3	0,10	4,11
Las losowy	14,00	0,04
AdaBoost	1929,00	1,30
MLP	2610,00	0,07
SVM	14,87	1,79
Minimalno-odległościowy	0,02	11,57
Binarne drzewo decyzyjne	0,78	0,001

Najszybciej szkolącym się klasyfikatorem był klasyfikator minimalno-odległościowy, dla którego średni czas szkolenia wynosił 0,02 s najdłużej zaś szkolił się MLP – 2610 s. Najskuteczniejszy klasyfikator, czyli SVM uczył się 14,87 s. Najszybszej klasyfikacji jednego znaku dokonywało binarne drzewo decyzyjne – 0,001 ms, najdłuższej zaś klasyfikator minimalno-odległościowy – 11,57 ms. SVM osiągnęło jeden z gorszych czasów klasyfikacji równy 1,79 ms. Istotność czasu szkolenia jako kryterium klasyfikacji uwidaczniała się w sytuacji wymagającej doszkolenia istniejącego już klasyfikatora do rozpoznawania nowych znaków. W praktyce operacja ta dla większości klasyfikatorów polegała na dodaniu do zbioru uczącego danych nowego znaku i ponownym wyszkoleniu klasyfikatora, co z każdym dodanym w ten sposób znakiem oznaczało dłuższy czas szkolenia.[6] W przypadku najskuteczniejszego klasyfikatora SVM operacja ta zajmowała co najmniej 14,9 s. Choć czasy szkolenia AdaBoost i MLP były bardzo długie, oba klasyfikatory posiadały opcję douczania. Dzięki niej klasyfikator nie musiał być szkolony od początku, co bardzo skracało czas potrzebny do nauczenia się nowego znaku a dodatkowo wykluczało konieczność przechowywania wraz z klasyfikatorem zbioru uczącego. Czas douczania jednego znaku dla klasyfikatora AdaBoost wynosił około 47 ms zaś dla MLP około 250 ms.[7]

Klasyfikatorem udostępniającym największą liczbę konfiguracji był niekwestionowanie MLP – kombinacja liczby neuronów, warstw ukrytych, funkcji aktywacji i ich parametrów oraz wartości momentum i współczynnika uczenia była nieskończona. W tym przypadku ogromna konfigurowalność była zarówno mocną i jak i słabą stroną klasyfikatora. Metodą prób i błędów można było osiągnąć zadowalające wyniki w rozsądnym czasie, jednakże dobranie parametrów optymalnych byłoby bardzo trudne i czasochłonne oraz wymagałoby przeprowadzenia obszernych badań. Drugim wysoce konfigurowalnym klasyfikatorem okazał się SVM udostępniający dwie metody klasyfikacji modyfikowane kilkoma parametrami i korzystające z czterech różnych funkcji jądra. W wykonanych testach użyto funkcji automatycznie wyznaczającej parametry poszczególnych metod klasyfikacji, która znacznie ułatwiła konfigurowanie klasyfikatora.[8] Dla binarnego drzewa decyzyjnego w przeprowadzonych badaniach osiem dostępnych parametrów w nieznacznym stopniu wpływało na osiągnięte wyniki klasyfikacji. Sytuacja wyglądała podobnie dla lasu losowego i AdaBoost, które bazowały na binarnym drzewie decyzyjnym. Klasyfikator kNN

[6] Wymienione średnie czasy szkolenia klasyfikatorów zostały zmierzone dla zbioru uczącego składającego się z 6084 znaków.

[7] Badania dla zbiorów uczących składających się z 1800, 4032 oraz 6084 znaków wykazały, iż czas douczania jest taki sam dla każdego z rozpatrzonych zbiorów uczących. Można więc przypuszczać, iż czas douczania nie jest zależny od liczby znaków w zbiorze uczącym.

[8] Można przypuszczać, iż automatyczne wyznaczanie parametrów nie zwraca najlepszych wartości jakie mogłyby być dobrane dla danego zadania a co za tym idzie istnieje możliwość polepszenia uzyskanych wyników.

konfigurowany był jedynie jednym parametrem – liczbą rozpatrywanych sąsiadów, zaś klasyfikator minimalno-odległościowy oraz Bayesa nie mogły być konfigurowane wcale.

Ilość wymaganej przestrzeni dyskowej do przechowywania wyszkolonego klasyfikatora a także czas jego wczytywania uznane zostały za najmniej ważne cechy klasyfikatora. [9] Zgodnie z przypuszczeniami czas wczytywania był ściśle powiązany z wielkością zapisanego klasyfikatora - im ta była większa, tym wczytywanie trwało dłużej. Z racji swojej implementacji klasyfikator kNN (a więc również minimalno-odległościowy) nie mógł zostać zapisany, jako poszukiwaną wielkość przyjęto więc rozmiar pliku z danymi uczącymi zaś czas określono jako czas wczytywania tego pliku. [10] Wyniki pomiarów zostały umieszczone w Tabeli 63 poniżej.

Tabela 63. Rozmiary zapisanego klasyfikatora oraz czas jego wczytywania dla różnych zbiorów gęstościowych dla poszczególnych klasyfikatorów. Oznaczenie „[n]" przy zbiorze oznacza najlepszy zbiór danych dla danego klasyfikatora.

Klasyfikator	Zbiór	Rozmiar zapisanego klasyfikatora [kB]	Czas wczytywania [ms]
MLP	3x3	327	63
MLP	8x8	449	94
MLP	16x16	877	188
Boost	3x3	2461	344
Boost	9x9	2570	359
Boost	16x16	2499	344
SVM	3x3	906	203
SVM	9x9	2152	547
SVM	16x16	7347	2062
kNN dla k=3	3x3	536	281
kNN dla k=3	9x9	2892	1735
kNN dla k=3	16x16	7139	4875
Bayes	3x3	241	47
Bayes	6x6	2998	567
Bayes	16x16	123516	24047
Las losowy	3x3	66772	9125
Las losowy	9x9	13629	1859
Las losowy	16x16	10145	1391
Minimalno-odległościowy	3x3	536	281
Minimalno-odległościowy	9x9	2892	1735
Minimalno-odległościowy	16x16	7139	4875
Binarne drzewo decyzyjne	3x3	513	79
Binarne drzewo decyzyjne	9x9	204	62
Binarne drzewo decyzyjne	16x16	220	47

[9] W obecnych czasach dostępne komputery posiadają bardzo duże zasoby pamięci dyskowej a wczytywanie zapisanego klasyfikatora powinno odbywać się tylko raz na cały etap działania klasyfikatora lub programu.

[10] Funkcja wczytująca dane z pliku nie była optymalizowana pod względem czasu wykonywania, tak więc przy dobraniu odpowiedniej implementacji czasy wczytywania mogłyby ulec skróceniu.

Przeprowadzone badania wykazały, iż czas wczytywania binarnego drzewa decyzyjnego malał wraz ze wzrostem liczby cech, zaś rozmiar zapisanego klasyfikatora raz rósł raz malał. Dla klasyfikatora AdaBoost rozmiar zapisanego pliku oraz czas wczytywania był bardzo zbliżony niezależnie od zestawu cech. Dla klasyfikatora Bayesa zaobserwowano znaczny wzrost rozmiaru pliku podczas zwiększania liczby cech. Podobnie wzrastał czas wczytywania klasyfikatora. Dla lasu losowego w przeciwieństwie do pozostałych klasyfikatorów wraz ze zwiększaniem liczby cech obserwowano zmniejszanie rozmiaru i skracanie czasu wczytywania klasyfikatora. Rozmiary i czas wczytywania dla klasyfikatora kNN dla k=3 oraz minimalno-odległościowego były takie same. Dla swojego najlepszego zestawu cech najdłużej wczytywał się klasyfikator minimalno-odległościowy – 4875 ms, zaś najkrócej binarne drzewo decyzyjne. Najwięcej miejsca potrzebował las losowy – 13629 kB, zaś najmniej binarne drzewo decyzyjne.

W przeprowadzonych testach klasyfikacja cyfry 0 jako litery O oraz litery O jako cyfry 0 traktowana była jako klasyfikacja błędna i tym samym wpływała negatywnie na wyznaczaną skuteczność klasyfikatora. Z racji tego, iż oba znaki, w wielu przypadkach, były ciężkie do odróżnienia nawet przez człowieka, niepoprawna ich klasyfikacja mogłaby nie zostać rozpatrywana jako błąd. Przy takim założeniu skuteczność klasyfikacji wzrosłaby - szczególnie dla MLP oraz SVM.

Skuteczność klasyfikacji wszystkich klasyfikatorów mogła zostać nieznacznie poprawiona poprzez odrzucenie części najmniej znaczących cech z wybranego zbioru *9x9* cech gęstościowych. Choć użycie do tego celu binarnego drzewa decyzyjnego nie dało zadowalających rezultatów, pokazało iż odpowiedni proces selekcji cech może korzystnie wpłynąć na działanie klasyfikatora, nie tylko zwiększając jego skuteczność ale również skracając czas jego szkolenia i działania.

Pomimo bardzo wysokiej skuteczności badanych klasyfikatorów podczas klasyfikacji zbioru podstawowego i dodatkowego (powyżej 99%) skuteczność klasyfikacji znaków pobranych z tablic rejestracyjnych nie przekroczyła 75%. Tak niska efektywność klasyfikatorów wywołana była szeregiem czynników - przede wszystkim niezadowalającą jakością próbek, ich rotacją, zbyt małym rozmiarem oraz innym krojem niż znaki należące do zbioru uczącego klasyfikatorów. Przeprowadzone testy pokazały więc, iż klasyfikatory, w celu uzyskania jak najlepszej skuteczności klasyfikacji dla danego zbioru danych, powinny być szkolone na obiektach jak najbardziej zbliżonych do badanych obiektów. Szczególnie dobrze zależność tą widać było podczas klasyfikacji litery W z tablic rejestracyjnych. Pomimo tego, iż wyszkolone klasyfikatory bezbłędnie klasyfikowały litery W ze zbioru podstawowego a także bardzo dobrze ze zbioru dodatkowego (gdzie znaki zapisane były innymi czcionkami niż te ze zbioru podstawowego) nie sklasyfikowały poprawnie ani jednej

litery W ze zbioru znaków z tablic rejestracyjnych – krój tej litery odbiegał znacznie od kroju liter zawartych w zbiorze uczącym, przez co pomimo dużej zdolności generalizacji badanych klasyfikatorów żadnemu z nich nie udało się poprawnie sklasyfikować tego znaku.

Porównanie przebadanych klasyfikatorów wykazało, iż żaden z nich nie był lepszy od pozostałych dla wszystkich rozpatrywanych kryteriów. W przypadku, gdy najistotniejsza była jedynie skuteczność klasyfikacji a pozostałe kryteria nie były ważne najlepszym klasyfikatorem wybrano SVM. Jeżeli kluczowa w konkretnym zadaniu była szybkość klasyfikacji pojedynczego znaku las losowy lub ewentualnie MLP było poszukiwanym klasyfikatorem. AdaBoost i MLP okazywały się efektywniejsze od SVM w sytuacjach wymagających szybkiego doszkalania klasyfikatora do rozpoznawania nowych znaków. Klasyfikator kNN a w szczególności minimalno-odległościowy z racji najszybszego szkolenia i braku konieczności konfiguracji nadawał się świetnie do oceniania przydatności różnych zbiorów cech. Rozpatrywane w rozdziale czwartym klasyfikatory równoległe i równoległo-kaskadowe pozwoliły uzyskać większą skuteczność klasyfikacji niż klasyfikatory podstawowe kosztem zwiększenia wymaganych zasobów pamięci, złożoności obliczeniowej i czasu. Choć, w zaproponowanej formie, klasyfikator równoległo-kaskadowy nie okazał się najskuteczniejszym klasyfikatorem, uzyskane wyniki wskazały, iż opisane podejście do problemu jest skuteczne i wymaga jedynie dopracowania metody rozróżniania znaków od cyfr. W przypadku klasyfikacji innych obiektów niż cyfry i litery klasyfikator równoległo-kaskadowy prawdopodobnie okazałby się skuteczniejszy od klasyfikatora całościowego. Należy również zwrócić uwagę na wyniki testów klasyfikacji znaków z tablic rejestracyjnych, podczas których klasyfikator równoległo-kaskadowy był drugim najlepszym (po MLP) klasyfikatorem.

Wyniki przeprowadzonych badań jak i przykładowe implementacje wybranych algorytmów mogą stanowić bazę do dalszych badań klasyfikatorów nie tylko w zadaniu klasyfikacji znaków alfanumerycznych, lecz i dla innych zagadnień.

Bibliografia

[1] Burges, C. J. C.: A tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery, 1998, nr 2, s.121-167.

[2] Bradski, G., Kaehler, A.: Learning OpenCV, Cambridge: O'Reilly, 2008.

[3] Breiman, L., Cutler, A.: Random Forests. Machine Learning, 2001, nr 45, s.5-32

[4] Cunningham, P., Delany, S. J.: k-Nearest Neighbour Classifiers Technical Report UCD-CSI-2007-4, 2007 [dostęp 06 czerwca 2011]. Dostępny w Internecie: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.100.1131&rep=rep1&type=pdf

[5] Czajewski, W., Iwanowski, M., Sławiński, M.: Inteligentne maszyny i systemy, skrypt, Warszawa 2011, s.111-134 [dostęp 4 maja 2011]. Dostępny w Internecie: ftp://vlab.ee.pw.edu.pl/kierunek%20Automatyka%20i%20Robotyka/wyklad/Imis_skrypt.pdf

[6] Dash, M., Liu, H.: Feature Selection for Classification. Intelligent Data Analysis, 1997, nr 1, s.131-156.

[7] Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, New York, 2001.

[8] Ho, K. M., Scott, P. D.: Binary Decision Trees Technical Report CSM-313 [dostęp 15 czerwca 2011]. Dostępny w Internecie: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.52.564&rep=rep1&type=pdf

[9] Jiang, W.: Process Consistency for AdaBoost, 2000, [dostęp 21 sierpnia 2011]. Dostępny w Internecie: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.32.5108&rep=rep1&type=pdf

[10] Koller, D., Sahami, D.: Toward Optimal Feature Selection. W: Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, 1996, s.284-292.

[11] Kotsiantis, S. B.: Supervised Machine Learning: A Review of Classification Techniques. Informatica, 2007, nr 31, s.249-268.

[12] Marques de Sa, J. P.: Pattern Recognition: Concepts, Methods and Applications, Wydawn. Springer, 2001.

[13] Meir, R., Ratsch, G.: An Introductions to Boosting and Leveraging, 2003 [dostęp 23 sierpnia 2011], Dostępny w Internecie: http://webee.technion.ac.il/~rmeir/Publications/MeiRae03.pdf

[14] Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine Learning, Neural and Statistical Classification, New York, 1994.

[15] Mitchell, T. M.: Generative and Discriminative Classifiers: Naive Bayes and Logicstic Regression, 2010 [dostęp 11 czerwca 2011]. Dostępny w Internecie: http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

[16] Langley, P.: Selection of Relevant Features in Machine Learning. W: Proceedings of the AAAI Fall Symposium on Relevance, New Orleans, 1994, s.140-144.

[17] Osowski, S.: Sieci neuronowe, Warszawa: Oficyna Wydawnicza Politechniki Warszawskiej, 1994.

[18] Tadeusiewicz, R., Flasiński, M.: Rozpoznawanie obrazów, Warszawa: Państwowe Wydawnictwo Naukowe, 1991.

[19] Wagner, P.: Statistical Machine Learning with OpenCV, 2011, [dostęp 10 września 2011]. Dostępny w Internecie: http://www.bytefish.de/pdf/machinelearning.pdf

[20] Dokumentacja OpenCV 2.1 [dostęp 1 kwietnia 2011]. Dostępny w Internecie: http://opencv.willowgarage.com/documentation/cpp/index.html