Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

Wydział
Informatyki

UCZENIE KASKADY KLASYFIKATORÓW W CELU REDUKCJI OCZEKIWANEJ LICZBY WYZNACZANYCH CECH W ZADANIACH DETEKCJI OBIEKTÓW NA OBRAZACH

Rozprawa doktorska

Autor:
mgr inż. Dariusz Sychel

Promotor:
dr hab. inż. Przemysław Klęsk, prof. ZUT

Szczecin, 2021

Streszczenie

Uczenie kaskady klasyfikatorów w celu redukcji oczekiwanej liczby wyznaczanych cech w zadaniach detekcji obiektów na obrazach

Autor: mgr inż. Dariusz Sychel

Promotor: dr hab. inż. Przemysław Klęsk, prof. ZUT

Kaskady klasyfikatorów działające wewnątrz procedury detekcyjnej, wyznaczają i wykorzystują różną liczbę cech w zależności od zawartości analizowanego okna obrazu (lub ogólniej okna sygnału). Okna zawierające tło mogą zostać prawidłowo rozpoznane zazwyczaj przy wykorzystaniu kilku cech, podczas gdy okna zawierające poszukiwany obiekt mogą wymagać tysięcy cech. Niniejsza praca doktorska skupia się na wielkości opisującej średni koszt obliczeniowy działającej kaskady — tj. *wartości oczekiwanej liczby cech* używanej przez kaskadę. W pracy zaproponowano dwie techniki pozwalające na redukcję tej wartości oczekiwanej. Pierwsza z nich wprowadza tzw. *poluzowane wymagania* dla każdego etapu kaskady dzięki wykorzystaniu zapasu powstającego w trakcie uczenia kaskady pomiędzy nałożonymi wymaganiami na etap a faktycznymi uzyskanymi wartościami wskaźników FAR oraz czułości. Druga zaproponowana technika to algorytm oparty na przeszukiwaniu drzewa oraz metodzie *podziału i ograniczeń* (ang. *branch-and-bound*). Zaproponowane dwie metody są do siebie komplementarne i mogą być stosowane równocześnie.

Słowa kluczowe: kaskada klasyfikatorów, detekcja, poluzowane wymagania na etap, oczekiwana liczba cech, boosting, przeszukiwanie drzewa metodą podziału i ograniczeń

Abstract

Training of cascade of classifiers in order to reduce the expected number of extracted features in tasks of detecting objects in images

Author: mgr inż. Dariusz Sychel

Supervisor: dr hab. inż. Przemysław Klęsk, prof. ZUT

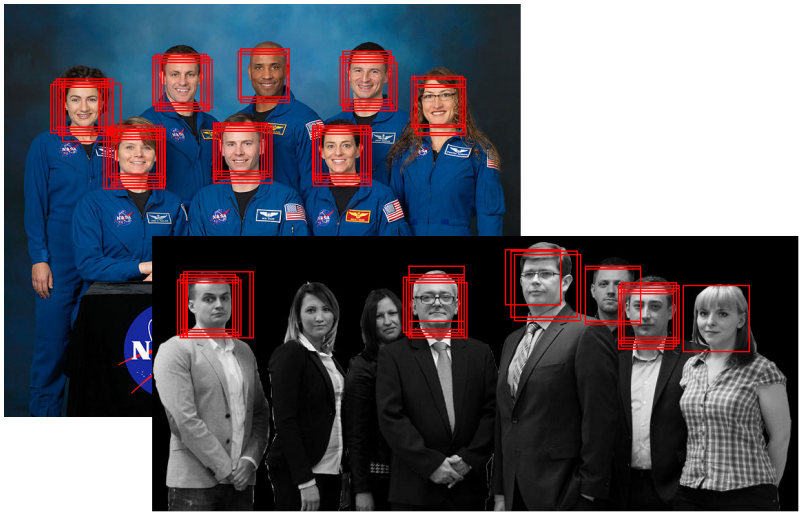
A cascade of classifiers, working inside a detection procedure, extracts and uses different number of features depending on the contents of the analyzed image window (or, more generally, signal window). Windows containing background fragments can be typically recognized correctly using only a few features, whereas windows containing target objects may require thousands of features. This work focuses on a quantity that describes the average computational cost of an operating cascade — the expected value of the *number of features* the cascade uses. In this doctoral dissertation, two methods, allowing to reduce this expected value, are proposed. The first method introduces the so-called *relaxed per-stage requirements* and takes advantage of the slack arising between constant requirements and actual rates of FAR and sensitivity observed during the training procedure. The second proposed method is an algorithm using a tree search approach to analyze cascade variants in a *branch-and-bound* fashion. Both proposed methods are complementary and can be applied together.

Keywords: cascade of classifiers, detection, relaxed per-stage requirements, expected number of features, boosting, branch-and-bound tree search

Spis treści

Wprowadzenie

Wraz z widocznym w obecnym świecie szybkim rozwojem technologii oraz jej wzrastającą dostępnością coraz częściej spotykamy się z zadaniem wykrywania obiektów na obrazach lub w sekwencjach wideo (czyli tzw. zadaniem detekcji). Mamy z nim do czynienia chociażby w smartfonach (wykrywanie twarzy w aplikacji aparatu fotograficznego (Chen, Shen i Sun, 2012)), w różnego rodzaju systemach monitoringu (wykrywanie sylwetek ludzkich (Li, Li i in., 2007)), czy w nowoczesnych lub nawet autonomicznych samochodach (wykrywanie innych pojazdów, znaków drogowych, sygnalizacji świetlnej, itd. (Kim, Yoo i Koo, 2018; Momin i Mujawar, 2015)). Jednym z algorytmów stosowanych do detekcji obiektów na obrazach jest *kaskada klasyfikatorów*.



Rysunek 1: Przykładowe wyniki działania procedury detekcyjnej (przed zgrupowaniem okien). (Źródło: opracowanie własne)

Pomysł kaskady klasyfikatorów został pierwotnie zaprezentowany przez Violę i Jonesa w artykule (Viola i Jones, 2001). Bazował on na obserwacji, że w procedurze detekcyjnej ogromną większość spośród rozpatrywanych okien obrazu stanowią okna negatywne (czyli okna nie zawierające interesujących nas obiektów). Procedura taka zazwyczaj bazuje na gęstym przeskanowaniu obrazu za pomocą tzw. *okna przesuwnego* (Rys. 1), przez co generuje dużą liczbę zapytań o odpowiedź klasyfikatora w krótkim czasie (Jiang i in., 2015; Papageorgiou i Poggio, 2000; Papageorgiou, Oren i Poggio, 1998; Sung i Poggio, 1998; Viola i Jones, 2001, 2004; Wojek i in., 2008). Niezależenie od rodzaju zadania (detekcja znaków drogowych, twarzy, pojazdów itp.) negatywy stanowią zwykle ponad 99% wszystkich okien. Na przykład w zadaniu detekcji twarzy na obrazie na kilka twarzy przy-

padać może 10^5 a nawet 10^6 negatywów w zależności od rozdzielczości obrazu i nastaw algorytmicznych (liczby skal przebiegu, rozmiaru okien, odstępów pomiędzy kolejnymi pozycjami okna przesuwnego, itp.). Innym scenariuszem, który warto rozpatrzyć są różnego rodzaju klasyfikacje wsadowe (batch classifications), gdzie zadaniem systemu jest regularne przetwarzanie otrzymywanych partii danych (logów systemowych, danych satelitarnych, medycznych, itp.).

Kaskady klasyfikatorów zostały zaprojektowane do działania właśnie w takich warunkach, to znaczy: (1) gdy wymagane jest przetworzenie dużej liczby zapytań, (2) oraz gdy mamy do czynienia ze znaczną nierównowagą klas. Drugi z warunków, nierównowaga klas, jest tak naprawdę dużą zaletą sprawiającą, że koncepcja kaskady ma sens. Koszt obliczeniowy w przypadku tego klasyfikatora jest zmienny i zależy od zawartości badanego okna. Obiekty, które są oczywistymi negatywami (nie zawierają poszukiwanego obiektu) powinny zostać rozpoznane szybko, to znaczy przy niedużej liczbie wyekstrahowanych cech oraz w krótkim czasie analizy. Powinno się tak dziać, ponieważ obiekty takie stanowią większość spośród analizowanych okien/próbek. Na przykład fragmenty obrazu zawierające tło powinny zostać rozpoznane i odrzucone bardzo szybko. Natomiast w przypadku okien zawierających poszukiwany obiekt lub obiekty go przypominające, klasyfikator powinien móc poświęcić więcej czasu na analizę oraz mieć możliwość wykorzystania większej liczby cech (setek czy nawet tysięcy). Efekt taki uzyskuje się poprzez podzielenie klasyfikatora na pewną liczbę etapów. Następnie, cechy są wyznaczane porcjami, etap po etapie. Jeżeli na którymś etapie uzyskamy negatywną odpowiedź klasyfikatora, to dalsze obliczenia zostają przerwane, a okno zostaje sklasyfikowane jako negatywne. Jeżeli na każdym etapie uzyskano odpowiedź pozytywną, badane okno zostaje sklasyfikowane jako pozytyw.

Każda nauczona kaskada klasyfikatorów posiada pewną charakterystyczną dla siebie *średnią* wartość kosztu obliczeniowego ponoszonego podczas pracy. Wartość ta może zostać matematycznie zdefiniowana jako pewna *wartość oczekiwana* (co zostanie przedstawione w Rozdziale 1) i jednoznacznie obliczona dla danej kaskady na podstawie: liczby cech używanych na kolejnych etapach, częstości fałszywych alarmów (FAR[1] — ang. *False Alarm Rate* lub *False Acceptance Rate*) na kolejnych etapach, czułości na kolejnych etapach, oraz łącznego rozkładu prawdopodobieństwa, z którego czerpane są dane. Co ciekawe — mimo, że prawdziwy rozkład prawdopodobieństwa leżący u podstaw danych jest w praktyce zazwyczaj nieznany (przez co dokładna wartość oczekiwana nie może zostać wyznaczona), to w przypadku zadań detekcji jesteśmy w stanie oszacować z dużą dokładnością tę wartość oczekiwaną, wykorzystując jedynie liczności cech na poszczególnych etapach kaskady oraz częstości fałszywych alarmów.

Sama procedura uczenia kaskady jest czasochłonna. Dla naprawdę złożonych problemów nauka może zająć dni czy nawet tygodnie. W swoich pionierskich pracach Viola i Jones (2001, 2004) stwierdzili, że nauka kaskady jest ciężkim kombinatorycznym zadaniem optymalizacji, obejmującym następujące istotne parametry: liczbę etapów kaskady, liczbę cech na kolejnych etapach (oraz wybór tych cech), progi decyzyjne dla etapów. Warto zauważyć, że problem ten nie został jeszcze ostatecznie rozwiązany. W swoich pracach Viola i Jones zaproponowali, aby narzucić liczbę etapów kaskady (oznaczaną w tej

[1] wartość tego wskaźnika jest równa tyle co: 1− specyficzność

pracy przez K) oraz *końcowe wymagania*, które muszą zostać spełnione, aby użytkownik zaakceptował taki klasyfikator. Wymagania te zostały zdefiniowane jako para liczb (A, D) reprezentujących odpowiednio: dopuszczalną maksymalną częstość fałszywych alarmów oraz minimalną akceptowalną czułość. Dzięki probabilistycznym właściwościom struktury kaskady, możliwe jest (stosując średnią geometryczną) wyznaczenie kolejnej pary liczb $(a_{\max}, d_{\min})$ reprezentującej tzw. *wymagania na etap*. Spełnienie tych wymagań na każdym etapie gwarantuje spełnienie wymagań końcowych dla całej kaskady. Takie właśnie podejście do uczenia kaskady zaproponowali Viola i Jones (2001, 2004).

Pomimo pojawienia się i rozwoju nowych technik głębokiego uczenia (ang. *deep learning*), aktualna literatura pokazuje, że kaskady klasyfikatorów są nadal szeroko stosowane w systemach detekcji oraz zadaniach klasyfikacji wsadowej. Abbas i in. (2017) zaproponowali metodę analizy tłumu, pozwalającą na zliczanie osób. W tym celu autorzy stworzyli system detekcji ludzkich głów oparty o cechy Haara i kaskadę klasyfikatorów. Setjo, Achmad i Djoko (2017) zaprezentowali kaskadę (także opartą na cechach Haara) pozwalająca na detekcję ludzi na obrazach termicznych. W pracy (Budiman i in., 2016) przedstawiono wykorzystanie kaskady w celu lokalizacji białych krwinek na obrazie. Autorzy osiągnęli precyzję 95% oraz czułość 74%. Dodatkowo metoda ta potrafiła rozróżnić białe krwinki od innych obiektów o podobnym kolorze. Li, Xu i in. (2016) opisali zadanie śledzenia wzroku wspomagane kaskadą. Podobne zastosowanie można znaleźć w pracy (Cuimei i in., 2017), gdzie autorzy połączyli kaskadę rozpoznającą twarze z dodatkowymi trzema słabymi klasyfikatorami służącymi do: rozpoznawania ust, oczu oraz dopasowania histogramu odcienia skóry. Interesujące połączenie cech Haara z transformacją Hougha (dla okręgów) zostało przedstawione w pracy (Fitriyani, Yang i Syafrudin, 2016). Zaproponowany system pozwalał na rozpoznanie, czy oczy wykrytej osoby są zamknięte czy otwarte z dokładnością 98%. Li, Lu i in. (2016) wykorzystali kaskadę z cechami w domenie częstotliwości bazującymi na rzeczywistej dyskretnej transformacji Fouriera. Chen, Liu i in. (2019) wykorzystali kaskadę klasyfikatorów w połączeniu z algorytmem propozycji regionów do detekcji pojazdów. Przygotowali oni w tym celu modyfikację cech Haara dostosowaną pod to zadanie (ang. *vehicle-based Haar features*). W pracy (Wang i in., 2019) możemy znaleźć zastosowanie kaskady do rozpoznawania wad w tkaninie. Martynow i in. (2019) użyli kaskady w celu poprawienia skuteczności algorytmu służącego do lokalizacji źrenic na obrazie oka. Zastosowanie kaskady umożliwiło poprawne działanie metody na obrazach z widocznymi rzęsami czy też refleksami świetlnymi. Warto także wspomnieć o pracy (Lu i in., 2018), w której autorzy zaproponowali wykorzystanie kaskad do detekcji ptasich gniazd na zdjęciach linii elektrycznych dużej mocy.

Na przestrzeni lat powstało wiele udoskonaleń dla nauki kaskad (Bourdev i Brandt, 2005; Gama i Brazdil, 2000; Li i Zhang, 2013; Pham i Cham, 2007; Vallez, Deniz i Bueno, 2015). W większości z nich skupiono się na różnych algorytmach selekcji cech, metodach próbkowania, czy też dostosowaniu kaskady do konkretnego zestawu cech (np. Haar, HOG[2], LBP[3], itp.). Co ważne, w większości z tych prac główny proces optymalizacyjny jest nadal oparty na spełnieniu *wymagań na etap*, przy założonej liczbie etapów. Niektórzy

[2]ang. *Histogram of oriented gradients*
[3]ang. *Local binary patterns*

autorzy w celu otrzymania zmodyfikowanej kaskady projektowali nowe algorytmy boostingowe, lepiej wpasowujące się w procedurę uczenia. Np. w pracach (Shen, Wang i Hengel, 2010; Shen, Wang, Paisitkriangkrai i in., 2013) zaproponowano tzw. „efektywne klasyfikatory węzłowe" (ang. *effective node classifiers*) skupiające się na dużej asymetrii w danych. Jednak w związku z matematycznymi trudnościami, oczekiwana liczba cech rzadko jest głównym kryterium optymalizacji.

Jednym z niewielu wyjątków w tym kontekście jest praca (Saberian i Vasconcelos, 2014). Autorzy wykorzystali metodę gradientu prostego w celu jawnej optymalizacji Lagranżjanu reprezentującego pewien kompromis pomiędzy wartością błędu dla kaskady a wartością oczekiwaną liczby użytych cech. Wykorzystali oni w tym celu sztuczkę pozwalającą przekształcić nieróżniczkowalny rekurencyjny wzór opisujący kaskadę na jego gładki odpowiednik, wykorzystując przybliżenie tangensem hiperbolicznym. W rozwiązaniu tym wszystkie etapy kaskady pozostają otwarte. Każdy krok gradientu prostego dodaje nowy słaby klasyfikator do któregoś z etapów albo tworzy nowy etap poprzez klonowanie ostatniego istniejącego. W związku z tym rozwiązanie to nie jest kierowane wymaganiami na etap. Posiada ono jednak pewne wady. Po pierwsze, gradient prosty może potencjalnie utknąć w lokalnym optimum, co jest prawdziwe w szczególności dla procedury uczącej kaskady (Sychel, Klęsk i Bera, 2018). Po drugie, ciężko jest zgadnąć z wyprzedzeniem dobrą wartość dla mnożnika Lagrange'a (który to mnożnik stanowi meta-parametr decydujący o sile kompromisu pomiędzy błędem a oczekiwaną liczbą cech). Po trzecie, przeprowadzana optymalizacja jest w pewnym sensie wyczerpująca. Należy bowiem sprawdzić wszystkie pochodne funkcjonalne bazując na dostępnych do dyspozycji cechach dla każdego otwartego etapu. Na końcu wybierana jest pochodna, która prowadzi do największej redukcji kryterium optymalizacji. Innymi słowy, zaproponowany algorytm staje się coraz bardziej złożony wraz z postępem nauki. Złożoność kolejnych kroków powyższego podejścia jest rzędu $O(Kn)$, gdzie K jest aktualną liczbą etapów, a n jest liczbą cech.

Warto zauważyć, że wydajność działającej kaskady jest niezależna od metody uczącej, a zależy bezpośrednio od liczby cech wykorzystywanej średnio na okno. W związku z tym istnieje bezpośrednie powiązanie między wartością oczekiwaną liczby cech a czasem detekcji.

W niniejszej pracy doktorskiej zaproponowano dwie współpracujące ze sobą techniki pozwalające na redukcję wartości oczekiwanej liczby cech kaskady. Pierwsza z nich dotyczy wymagań na etap. Sednem metody jest wykorzystywanie zapasu powstającego w trakcie uczenia kaskady pomiędzy nałożonymi wymaganiami na etap a faktycznymi uzyskanymi wartościami wskaźników FAR oraz czułości. Zapas ten używany jest do wyznaczenia nowych *poluzowanych wymagań* dla dalszych etapów kaskady. Druga zaproponowana technika to algorytm (lub raczej ogólna struktura algorytmiczna) oparty na przeszukiwaniu drzewa oraz metodzie *podziału i ograniczeń* (ang. *branch-and-bound*). W proponowanym podejściu kolejne poziomy drzewa odpowiadają kolejnym etapom kaskady. Węzły rodzeństwa reprezentują warianty tego samego etapu z różną liczbą zastosowanych cech. Przedstawione zostały odpowiednie wzory na dolne ograniczenie optymalizowanej wartości oczekiwanej. W trakcie trwającego poszukiwania, dolna granica jest na bieżąco obserwowana. Gdy wartość dla pewnej gałęzi przekroczy najlepszą odnalezioną do tej pory oczekiwaną,

gałąź taka zostaje przycięta. Gdy przeszukiwanie zostanie zakończone, jedna ze ścieżek od korzenia do któregoś z liści wskaże kaskadę z najmniejszą wartością oczekiwaną liczby cech. Oprócz dokładnej techniki przycinania, została także zaproponowana jej aproksymacja bazująca na predykcji wartości oczekiwanej.

Sama metoda podziału i ograniczeń jest użytecznym narzędziem w informatyce. Posiada ona wiele zastosowań (Lawler i Wood, 1966; Morrison i in., 2016). Wśród nich można wymienić chociażby poszukiwanie motywów regulatorowych w DNA w bioinformatyce (Jones i Pevzner, 2002), czy też algorytm przycinania α-β dla gier dwuosobowych (Brudno, 1963; Edwards i Hart, 1963; Newell i Simon, 1976; Samuel, 1959, 1967).

Treści w niniejszej rozprawie zostały zorganizowane w sposób następujący. Rozdział 1 zawiera niezbędne elementy teoretyczne oraz zastosowaną notację, w szczególności: opis klasycznej kaskady klasyfikatorów, boostingu oraz słabych klasyfikatorów, i wreszcie definicję oczekiwanego kosztu obliczeniowego kaskady. Rozdział 2 zawiera zestaw przykładów geometrycznych pokazujących nieoptymalność klasycznego podejścia. Rozdział 3 opisuje algorytmy utworzone w celu poradzenia sobie z tą nieoptymalności, w tym wspomniane powyżej poluzowane ograniczenia na etap oraz przeszukiwanie metodą podziału i ograniczeń. Rozdział 4 przedstawia wyniki przeprowadzonych badań. Ostatni rozdział (Rozdział 5) zawiera natomiast uwagi końcowe podsumowujące pracę oraz opis możliwych kierunków dalszego rozwoju badań.

Cel i teza pracy

Bezpośrednią motywacją do sformułowania celu i tezy niniejszej pracy jest obserwacja, że ścisłe przestrzeganie stałych wymagań cząstkowych (wymagań na każdy etap) dla klasyfikatorów wchodzących w skład kaskady zapewnia spełnienie wymagań końcowych, jednak wynikowa oczekiwana liczba wyznaczanych cech może nie być optymalna.

Cel pracy: *Opracowanie algorytmu uczenia kaskady klasyfikatorów redukującego oczekiwaną liczbę ekstrahowanych cech w porównaniu do algorytmu Violi i Jonesa przy zachowaniu końcowych wymagań dla czułości i specyficzności.*

Teza: *Poluzowanie wymagań cząstkowych z wykorzystaniem aktualizowanych średnich geometrycznych oraz technik przeszukiwania drzewa uczenia umożliwia redukcję wartości oczekiwanej liczby cech ekstrahowanych przez kaskadę klasyfikatorów.*

Rozdział 1

Wstęp teoretyczny

Rozdział ten zawiera opis podstawowych terminów oraz algorytmów niezbędnych do zrozumienia dalszych części pracy m.in.: definicję wartości oczekiwanej liczby cech dla kaskady i różne sposoby jej wyznaczania, opis klasyfikatorów wchodzących w skład kaskady oraz notację matematyczną.

1.1 Notacja

W tym punkcie przedstawiono notację obowiązującą w pracy oraz probabilistyczne znaczenie najważniejszych miar występujących w kaskadzie klasyfikatorów:

- K — liczba etapów kaskady,
- $(n_1, n_2, \ldots, n_K)$ — ciąg zawierający liczbę ekstrahowanych cech na poszczególnych etapach kaskady,
- $(a_1, a_2, \ldots, a_K)$ — FAR na kolejnych etapach (częstość fałszywych alarmów),
- $(d_1, d_2, \ldots, d_K)$ — czułość na kolejnych etapach (częstość wykryć pozytywnych),
- A — wymagany FAR dla całej kaskady,
- D — wymagana czułość dla całej kaskady,
- $a_{\max} = A^{1/K}$ — wymagany FAR na etap (wariant klasyczny — obliczany jako średnia geometryczna),
- $d_{\min} = D^{1/K}$ — wymagana czułość na etap (wariant klasyczny — obliczany jako średnia geometryczna),
- $F = (F_1, \ldots, F_K)$ — kaskada reprezentowana jako ciąg zespołowych klasyfikatorów na poszczególnych etapach,
- A_k — FAR zaobserwowany aż do k-tego etapu kaskady ($A_k = \prod_{1 \leqslant i \leqslant k} a_i$),
- D_k — czułość zaobserwowana aż do k-tego etapu kaskady ($D_k = \prod_{1 \leqslant i \leqslant k} d_i$),
- p — prawdopodobieństwo klasy pozytywnej (nieznane w praktyce),
- $1 - p$ — prawdopodobieństwo klasy negatywnej (nieznane w praktyce),
- $\mathcal{D} = \{(\mathbf{x}_i \; y_i)\}_{i=1,\ldots,m}$ — zbiór przykładów uczących,

- $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})$ — wektor cech opisujący i-ty przykład,
- $y_i \in \{-1, 1\}$ — etykieta klasy skojarzona z i-tym przykładem,
- $\mathcal{V}$ — zbiór przykładów walidacyjnych,
- $[s]$ — funkcja wskaźnikowa zwracająca 1, jeśli warunek s jest prawdziwy i 0 w przeciwnym razie,
- $||$ — operacja konkatenacji (np. dla kaskady $F = (F_1, F_2)$, $F||F_3 = (F_1, F_2, F_3)$),
- $\#$ — operacja mocy zbioru.

Rozważmy dowolny przykład uczący $(\mathbf{x}, y)$ pochodzący z łącznego rozkładu prawdopodobieństwa związanego z badanym zjawiskiem. Wówczas probabilistyczny sens istotnych dla kaskad wielkości można określić następująco. Końcowe wymaganie dla częstości fałszywych alarmów (FAR) oznacza, że zachodzić ma nierówność:

$$P\left(F(\mathbf{x}){=}+\,|y{=}-\right) \leqslant A. \tag{1.1}$$

Końcowe wymaganie dla częstości wykryć pozytywnych (czułości) oznacza, że zachodzić ma nierówność:

$$P\left(F(\mathbf{x}){=}+\,|y{=}+\right) \geqslant D. \tag{1.2}$$

Aktualne (tzn. obserwowane po k etapach procedury) częstości fałszywych alarmów oraz wykryć prawidłowych obserwowane podczas procedury uczącej są rozumiane odpowiednio jako:

$$\begin{aligned} a_k &= 1{-}P\left(F_k(\mathbf{x}){=}-\,|y{=}-, F_1(\mathbf{x}){=}\cdots{=}F_{k-1}(\mathbf{x}){=}+\right) \\ &= P\left(F_k(\mathbf{x}){=}+\,|y{=}-, F_1(\mathbf{x}){=}\cdots{=}F_{k-1}(\mathbf{x}){=}+\right), \\ d_k &= P\left(F_k(\mathbf{x}){=}+\,|y{=}+, F_1(\mathbf{x}){=}\cdots{=}F_{k-1}(\mathbf{x}) = +\right). \end{aligned} \tag{1.3}$$

W pracy tej klasie negatywnej '$-$' odpowiada wartość -1, natomiast klasie pozytywnej '$+$' wartość 1.

Należy wyjaśnić, że użyty w zapisach (1.1), (1.2), (1.3) operator prawdopodobieństwa $P(\cdot)$ nie będzie w kontekście algorytmicznym rozumiany w sposób ścisły. Tzn. prawdopodobieństwa wspomnianych zdarzeń będą tak naprawdę utożsamiane z ich częstościami zmierzonymi na odpowiednio dużym walidacyjnym zbiorze danych, nie zaś miarami prawdopodobieństwa obliczonymi względem łącznego rozkładu prawdopodobieństwa (który, jak zaznaczono wcześniej, jest nieznany w praktyce).

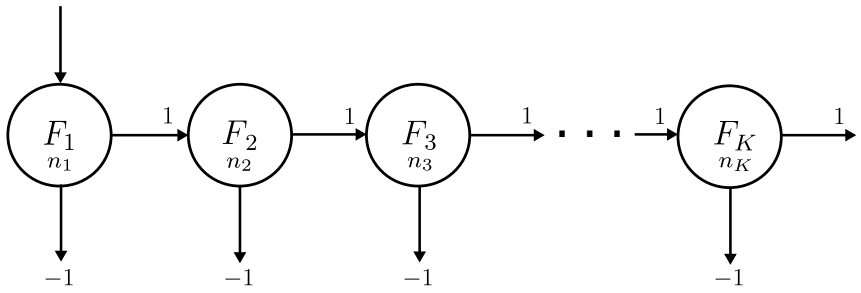
Należy także zaznaczyć, że pojawiająca się w dalszych treściach funkcja sgn jest dwuwartościowym wariantem pełnej funkcji signum (z pominięciem możliwości 0) i jest określona jako:

$$\operatorname{sgn}(\mathbf{x}) = \begin{cases} -1, & \mathbf{x} < 0; \\ 1, & \mathbf{x} \geqslant 0. \end{cases} \tag{1.4}$$

1.2 Kaskada klasyfikatorów

Kaskada klasyfikatorów (Viola i Jones, 2001, 2004) jest szczególnym klasyfikatorem zespołowym. To, co wyróżnia ją od innych algorytmów uczonych poprzez boosting, to fakt,

że kolejne silne klasyfikatory uwzględnione w kaskadzie uczone są na takim podzbiorze danych, w którym przykłady zostały sklasyfikowane jako pozytyw na etapie poprzednim. Każdy etap kaskady uczony jest zwykle tym samym algorytmem boostingowym. W oryginalnej pracy Violi i Jonesa autorzy zaproponowali wykorzystanie algorytmu Discrete AdaBoost, dzięki czemu klasyfikator ten jest zdolny do selekcji cech. W przeciwieństwie do typowych klasyfikatorów, w kaskadzie nadanie etykiety podanemu na wejście punktowi $\mathbf{x}$ nie wymaga stałego kosztu obliczeniowego. W przypadku kaskady jest to zależne od podobieństwa $\mathbf{x}$ do poszukiwanego obiektu np. twarzy. W momencie gdy punktowi danych, na danym etapie zostanie przypisana klasa negatywna, to jest on odrzucany z dalszej analizy (nie wchodzi na kolejny etap), co przedstawiono na Rys. 1.1. W związku z tym różne okna obrazu wymagają różnej liczby cech do sklasyfikowania ich zawartości, a czas klasyfikacji zależy bezpośrednio od średniej liczby cech przypadającej na okno.



Rysunek 1.1: Schemat kaskady klasyfikatorów. (Źródło: opracowanie własne)

W podejściu zaproponowanym w pracy (Viola i Jones, 2001) konieczne jest ustalenie *wymagań końcowych*, które nauczona kaskada powinna spełniać, są to odpowiednio maksymalny FAR oraz minimalna czułość. Charakterystyczne dla tego podejścia są stałe wymagania cząstkowe (wymagania na etap), które możemy obliczyć jako

$$a_{\max} = A^{\frac{1}{K}},\ d_{\min} = D^{\frac{1}{K}}. \tag{1.5}$$

Algorytm 1 przedstawia klasyczny proces uczenia kaskady klasyfikatorów. Warto zwrócić uwagę, że w końcowej linii pseudokodu zwrócona zostaje kaskada w postaci $(F_1, F_2, \ldots, F_k)$, zamiast $(F_1, F_2, \ldots, F_K)$. Jest to spowodowane tym, że procedura ucząca zgodna z tym podejściem może zostać zakończona wcześniej, gdy $k < K$, pod warunkiem, że końcowe wymagania (A, D) dla całej kaskady zostały spełnione, to znaczy $A_k \leqslant A$ i $D_k \geqslant D$.

Kolejnym fragmentem, na który należy zwrócić uwagę jest krok „Dostosuj próg decyzyjny", gdyż wymaga on dokładniejszego wyjaśnienia. Patrząc na rzeczywistoliczbowe odpowiedzi każdego z etapów można wybrać odpowiednie progi decyzyjne dla osiągnięcia zadanej czułości lub miary FAR. Decyzja dla etapu jest w rzeczywistości obliczana jako:

$$\operatorname{sgn}(F_k(\mathbf{x}) - \theta_k),$$

gdzie θ_k jest progiem decyzyjnym dla k-tego etapu. Załóżmy, że $(v_1, \ldots, v_{\#\mathcal{V}_P})$ oznacza ciąg posortowanych, $v_i \leqslant v_{i+1}$, rzeczywistoliczbowych odpowiedzi nowego etapu kaskady F_{k+1} uzyskanych dla przykładów pozytywnych $\mathcal{V}_P \subset \mathcal{V}$ w zbiorze walidacyjnym. Wtedy wymóg $d_{\min}$ na etap może zostać spełniony dzięki wybraniu progu jako odpowiedniej wartości

z tego ciągu, stojącej w przybliżeniu na pozycji $(1 - d_{\min}) \cdot \#\mathcal{V}_P$, tzn.:

$$\theta_{k+1} = v_{\lfloor (1-d_{\min}) \cdot \#\mathcal{V}_P \rfloor}. \tag{1.6}$$

Algorytm 1 Algorytm uczący kaskadę klasyfikatorów ze stałymi wymaganiami na etap (podejście klasyczne).

Algorytm TrainVJCascade($\mathcal{D}$, A, D, K, $\mathcal{V}$)

Ustal podzbiór przykładów pozytywnych $\mathcal{P}$ i negatywnych $\mathcal{N}$ w ramach zbioru $\mathcal{D}$.

$F := ()$. ▷ początkowa kaskada — pusty ciąg

$a_{\max} := A^{1/K}$, $d_{\min} := D^{1/K}$.

$A_0 := 1$, $D_0 := 1$, $k := 0$.

Dopóki $A_k > A$ **powtarzaj**

$n_{k+1} := 0$, $F_{k+1} := 0$, $A_{k+1} := A_k$.

$a_{k+1} := A_{k+1}/A_k$.

Dopóki $a_{k+1} > a_{\max}$ **powtarzaj**

Naucz nowy słaby klasyfikator f (wykorzystujący jedną wybraną cechę) używając przykładów w zbiorach $\mathcal{P}$ i $\mathcal{N}$.

$F_{k+1} := F_{k+1} + f$.

$n_{k+1} := n_{k+1} + 1$.

Dostosuj próg decyzyjny θ_{k+1} dla F_{k+1}, tak aby spełnił wymaganie $d_{\min}$.

Użyj kaskady $F \| F_{k+1}$ na zbiorze walidacyjnym $\mathcal{V}$, aby zmierzyć A_{k+1} i D_{k+1}.

$a_{k+1} := A_{k+1}/A_k$.

$F := F \| F_{k+1}$.

Jeżeli $A_{k+1} > A$ **to**

$\mathcal{N} := \emptyset$.

Użyj aktualnej kaskady F w celu uzupełnienia zbioru $\mathcal{N}$ fałszywymi wykryciami próbkowanymi z obrazów nie zawierających wykrywanych obiektów.

$k := k + 1$.

Zwróć $F = (F_1, F_2, \ldots, F_k)$.

1.3 Boosting

Każdy kolejny etap kaskady klasyfikatorów powstaje zwyczajowo poprzez wykonanie pewnego algorytmu boostingowego, w wyniku którego otrzymujemy zespół tzw. słabych klasyfikatorów.

Historyczną motywacją, która doprowadziła do opracowania algorytmów boostingu było pytanie: „czy zespół słabych klasyfikatorów może utworzyć pojedynczy silny klasyfikator?" (Kearns, 1988; Schapire, 1990). Słabymi klasyfikatorami zazwyczaj bywają: klasyfikatory jednoregułowe (decision stumps), płytkie drzewa decyzyjne, naiwne klasyfikatory Bayesa, proste sieci neuronowe, maszyny SVM. Wpływ słabych klasyfikatorów wchodzących w składa zespołu na ostateczny wynik klasyfikacji jest różny i zależy od ich skuteczności uzyskanej w trakcie nauki. W kolejnych rundach boostingu zmieniają się wagi

(priorytety) poszczególnych przykładów. Im więcej razy dany przykład był niepoprawnie sklasyfikowany, tym większa będzie jego waga, a co za tym idzie koszt, który do kryterium błędu wnosi tenże przykład. Tym samym kolejne rundy boostingu będą zmuszone starać się sklasyfikować ten przykład prawidłowo. Zaletami boostingu jest zdolność do automatycznej selekcji cech oraz wysoka odporność na przeuczenie.

Poniżej znajduje się opis dwóch podstawowych algorytmów boostingowych: Discrete AdaBoost (AdaBoost) wykorzystywanego w klasycznej kaskadzie oraz używany przez autora niniejszej pracy Real AdaBoost (RealBoost).

1.3.1 Discrete AdaBoost

W przypadku algorytmu Discrete AdaBoost słabe klasyfikatory wchodzące w skład wynikowego klasyfikatora są binarne i zwracają wartości ze zbioru $\{-1, 1\}$. Siła poszczególnych klasyfikatorów wyrażana jest za pomocą współczynników α_t, których wartość zależy od wartości błędu klasyfikacji na zbiorze uczącym ważonego wagami w_i przykładów uczących. Im ten błąd jest bliższy wartości 1/2, tym mniejsza jest siła klasyfikatora (Freund i Schapire, 1996, 1999).

Algorytm 2 Algorytm uczący klasyfikator Discrete AdaBoost.

Algorytm TrainAdaBoost($\mathcal{D}$) ▷ $\mathcal{D} = \{(\mathbf{x}_i\ y_i)\}_{i=1,\ldots,m}$

Przypisz jednakowe wartości wag dla każdego z przykładów $w_i := \frac{1}{m}, \quad i = 1, \ldots, m.$

Dla $t := 1, \ldots, T$ **powtarzaj**

Naucz słaby klasyfikator $f_t(\mathbf{x}) \in \{-1, 1\}$ uwzględniając wagi w.

Oblicz błąd $\epsilon_t := \sum_{i=1}^{m} w_i \left[f_t(\mathbf{x}_i) \neq y_i\right]$.

Oblicz siłę klasyfikatora $\alpha_t := \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$.

Zaktualizuj wagi $w_i := w_i e^{-\alpha_t f_t(\mathbf{x}_i) y_i}, \quad i = 1, \ldots, m.$

Znormalizuj wagi tak, aby $\sum_{i=1}^{m} w_i = 1$.

Zwróć $F_{\mathrm{DA}} = (f_1, f_2, \ldots, f_T)$.

Do obliczenia odpowiedzi tego klasyfikatora stosuje się dwuwartościową funkcję signum (1.4), gdzie argumentem jest ważona suma odpowiedzi słabych klasyfikatorów:

$$F_{\mathrm{DA}}(\mathbf{x}_i) = \operatorname{sgn}\left(\sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}_i)\right). \tag{1.7}$$

Wybór współczynnika siły klasyfikatora $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ motywowany jest kryterium błędu wykładniczego postaci:

$$Z_t = \sum_{i=1}^{m} w_i e^{-\alpha_t f_t(\mathbf{x}_i) y_i}. \tag{1.8}$$

Warto zaznaczyć, że powyższe kryterium lub też funkcja błędu stanowi górne i gładkie ograniczenie na tradycyjny błąd zero-jedynkowy:

$$\epsilon_t = \sum_{i=1}^{m} w_i \left[f_t(\mathbf{x}_i) \neq y_i\right]. \tag{1.9}$$

W celu zminimalizowania (1.8) należy przyrównać do zera pochodną ze względu na α_t oraz pogrupować odpowiednio wyrazy sumy:

$$\frac{\partial Z_t}{\partial \alpha_t} = 0, \qquad \frac{\partial}{\partial \alpha_t}\left(\sum_{f_t(\mathbf{x}_i)=y_i} w_i e^{-\alpha_t} + \sum_{f_t(\mathbf{x}_i)\neq y_i} w_i e^{\alpha_t}\right) = 0. \tag{1.10}$$

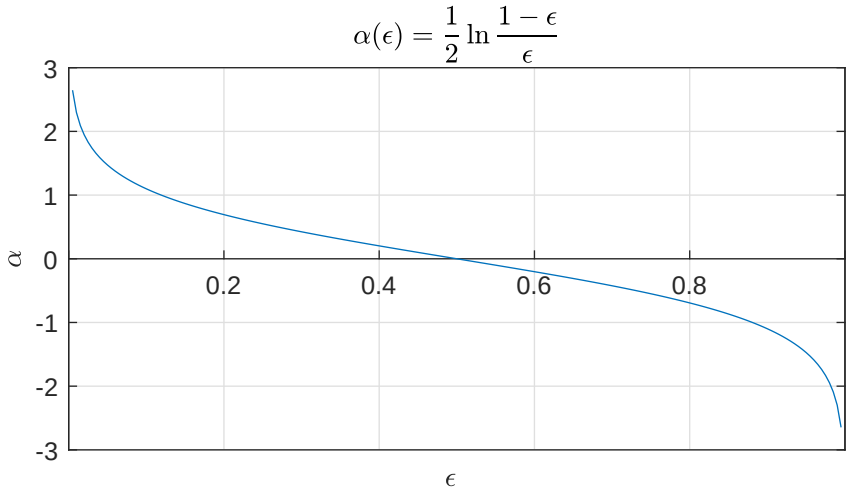
Wiedząc, że suma wag przykładów błędnie rozpoznanych wynosi ϵ_t otrzymujemy dalej:

$$\begin{aligned} \frac{\partial}{\partial \alpha_t}\Big((1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}\Big) &= 0, \\ (1-\epsilon_t)(-e^{-\alpha_t}) + \epsilon_t e^{\alpha_t} &= 0, \\ \epsilon_t e^{\alpha_t} &= -(1-\epsilon_t)(-e^{-\alpha_t}). \end{aligned} \tag{1.11}$$

Po zlogarytmowaniu stronami otrzymujemy widoczne w Algorytmie 2 wyrażenie:

$$\begin{aligned} \ln \epsilon_t + \alpha_t &= \ln(1-\epsilon_t) - \alpha_t, \\ 2\alpha_t &= \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right), \\ \alpha_t &= \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right). \end{aligned} \tag{1.12}$$

Rysunek 1.2 przedstawia wykres α jako funkcji ϵ. Jak możemy zauważyć, wraz z oddalaniem się wartości ϵ od punktu $\frac{1}{2}$ rośnie bezwzględna waga danego klasyfikatora. Dodatkowo warto zauważyć, że w przypadku, gdy dla któregoś ze słabych klasyfikatorów wartość $\epsilon > \frac{1}{2}$, to waga α zmieni jego odpowiedź na przeciwną. Czyli klasyfikator o błędzie np. 10% jest tak samo dobry jak zanegowany klasyfikator o błędzie 90%.



Rysunek 1.2: Wykres zależności między wartością ϵ a wartością α. (Źródło: opracowanie własne)

1.3.2 Real AdaBoost

W algorytmie Real AdaBoost słabe klasyfikatory są rzeczywistoliczbowe, a ich odpowiedź jest zazwyczaj przybliżeniem połowy przekształcenia logit:

$$f_t(\mathbf{x}) = \frac{1}{2}\ln\frac{\widehat{P}_w(y=1|\mathbf{x})}{\widehat{P}_w(y=-1|\mathbf{x})}, \tag{1.13}$$

gdzie $\widehat{P}_w$ stanowi oszacowanie rozkładu klas warunkowanego na $\mathbf{x}$ przy uwzględnieniu wag w (Friedman, Hastie i Tibshirani, 1998). Wynika to z faktu, że w przypadku tego klasyfikatora zamiast błędu zero-jedynkowego jako kryterium używany jest błąd wykładniczy. Błąd ten stanowi górne i gładkie ograniczenie na błąd zero-jedynkowy. Oczekiwany błąd wykładniczy względem nieznanego rozkładu łącznego $p(\mathbf{x}, y) = P(y|\mathbf{x})p(\mathbf{x})$, generującego dane, wynosi:

$$E_P\left(e^{-F(\mathbf{x})y}\right) = \int_{\mathbf{x}} \sum\nolimits_{y\in\{-1,1\}} e^{-F(\mathbf{x})y} P(y|\mathbf{x})p(\mathbf{x})\mathbf{dx}, \tag{1.14}$$

a funkcja minimalizująca ten błąd to właśnie połowa przekształcenia logit:

$$F(\mathbf{x}) = \frac{1}{2}\ln\frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})}. \tag{1.15}$$

Algorytm 3 Algorytm uczący klasyfikator Real AdaBoost.

Algorytm TrainRealBoost($\mathcal{D}$) ▷ $\mathcal{D} = \{(\mathbf{x}_i\ y_i)\}_{i=1,\ldots,m}$

Przypisz jednakowe wartości wag dla każdego z przykładów $w_i := \frac{1}{m}, \quad i = 1, \ldots, m.$

Dla $t := 1, \ldots, T$ **powtarzaj**

Używając wag w i oszacowań rozkładów warunkowych $\widehat{P}_w(y = \pm 1|\mathbf{x})$,

naucz słaby klasyfikator $f_t(\mathbf{x}) \in \mathbb{R}$ odpowiadający zgodnie ze wzorem (1.13).

Zaktualizuj wagi $w_i := w_i e^{-f_t(\mathbf{x}_i)y_i}, \quad i = 1, \ldots, m.$

Znormalizuj wagi tak, aby $\sum_{i=1}^{m} w_i = 1$.

Zwróć $F_{\mathrm{RA}} = (f_1, f_2, \ldots, f_T)$.

W związku z wprowadzoną zmianą nie ma potrzeby stosowania współczynników α_t do określenia siły poszczególnych słabych klasyfikatorów, ponieważ została ona niejako „wpleciona" w samą funkcję odpowiedzi, a decyzja klasyfikatora zespołowego obliczana jest jako:

$$F_{\mathrm{RA}}(\mathbf{x}_i) = \operatorname{sgn}\left(\sum_{t=1}^{T} f_t(\mathbf{x}_i)\right). \tag{1.16}$$

1.4 Słabe klasyfikatory

Według pomysłodawców boostingu (Freund i Schapire, 1999; Kearns, 1988; Schapire, 1990) jako słaby klasyfikator możemy w skrajnej postaci rozumieć klasyfikator niewiele lepszy od rzutu monetą. W klasycznej kaskadzie wg pomysłu Violi i Jonesa (Viola i Jones, 2001, 2004), jako słabe klasyfikatory wykorzystano klasyfikatory jednoregułowe (ang. *decision stumps*), które można rozumieć jako dwupoziomowe drzewa decyzyjne z jednym rozcięciem (korzeń i dwójka potomków). Lecz nie są one jedyną możliwością, równie dobrze mogą to być wspomniane w punkcie 1.3: płytkie drzewa decyzyjne, naiwne klasyfikatory Bayesa, proste sieci neuronowe, maszyny SVM, czy np. płaszczyzny decyzyjne (w szczególności nawet losowe) lub inne. W praktyce słaby klasyfikator powinien wykorzystywać niewielką liczbę cech, pozwalając tym sposobem algorytmowi boostingowemu na selekcję cech. Jest to szczególnie istotne z punktu widzenia budowy kaskady, ponieważ pozwala na zróżnicowanie liczby cech na poszczególnych jej etapach.

1.4.1 Klasyfikator jednoregułowy — decision stump

Klasyfikator ten jest dwupoziomowym drzewem decyzyjnym z dwoma terminalami (liśćmi). Swoją decyzję opiera na trzech wartościach: numerze cechy, wartości progu, oraz kierunku nierówności (tzw. polaryzacji). Wartości te są dobierane w taki sposób, aby zminimalizować błąd klasyfikacji na zbiorze uczącym, a w przypadku wykorzystania ich w algorytmie boostingowym minimalizacji ulega ważony błąd klasyfikacji (Ai i Langley, 1992). Odpowiedź klasyfikatora jednoregułowego dla punktu danych $\mathbf{x}_i$ ma następującą postać:

$$f(\mathbf{x}_i) = \begin{cases} 1, & \phi \cdot (\mathbf{x}_{i,j} - \theta) \geqslant 0; \\ -1, & \phi \cdot (\mathbf{x}_{i,j} - \theta) < 0, \end{cases} \tag{1.17}$$

gdzie j oznacza numer wybranej cechy, $\phi \in \{-1, 1\}$ kierunek nierówności, a θ wartość progu.

Aby zastosować „decision stumps" w algorytmie RealBoost, wartości $\{-1, 1\}$ należy zastąpić przybliżeniem połowy przekształcenia logit (1.13). W tym celu dla każdego z terminali należy oszacować $\widehat{P}_w(y = \pm|\mathbf{x})$, gdzie:

$$\widehat{P}_w(y = \pm 1|\mathbf{x}; j, \theta, \phi) = \begin{cases} \sum\limits_{\{i:\ \phi\cdot(\mathbf{x}_{i,j}-\theta)\geqslant 0,\ y_i=\pm 1\}} w_i, & \phi \cdot (\mathbf{x}_{i,j} - \theta) \geqslant 0; \\ \sum\limits_{\{i:\ \phi\cdot(\mathbf{x}_{i,j}-\theta)< 0,\ y_i=\pm 1\}} w_i, & \phi \cdot (\mathbf{x}_{i,j} - \theta) < 0. \end{cases} \tag{1.18}$$

1.4.2 Rozkłady normalne

Rozkłady normalne, stosowane jako słabe klasyfikatory, także działają w oparciu o jeden wyselekcjonowany atrybut. W trakcie nauki tego klasyfikatora dokonywana jest aproksymacja rozkładów atrybutów pod warunkiem klas[1] $p(x_j|y = \pm 1)$ przez rozkłady normalne (Rasolzadeh i in., 2006):

$$\widehat{p}_w(x_j|y = \pm 1) = \frac{1}{\sqrt{2\pi\sigma_{j\pm}^2}} \exp\left(\frac{-(x_j - \mu_{j\pm})^2}{2\sigma_{j\pm}^2}\right), \tag{1.19}$$

gdzie $\mu_{j\pm}$ i $\sigma_{j\pm}^2$ oznaczają średnie i wariancje odpowiednio dla klasy pozytywnej i negatywnej przy ustalonym j-tym atrybucie. W związku z koniecznością uwzględnienia wag przykładów (biorących udział w boostingu), średnie i wariancje z powyższego wzoru obliczamy jako:

$$\mu_{j-} = \frac{\sum\limits_{\substack{i=1 \\ y_i=-1}}^{m} w_i \mathbf{x}_{i,j}}{\sum\limits_{\substack{i=1 \\ y_i=-1}}^{m} w_i}, \qquad \mu_{j+} = \frac{\sum\limits_{\substack{i=1 \\ y_i=1}}^{m} w_i \mathbf{x}_{i,j}}{\sum\limits_{\substack{i=1 \\ y_i=1}}^{m} w_i}, \tag{1.20}$$

$$\sigma_{j-}^2 = \frac{\sum\limits_{\substack{i=1 \\ y_i=-1}}^{m} w_i \mathbf{x}_{i,j}^2}{\sum\limits_{\substack{i=1 \\ y_i=-1}}^{m} w_i} - \mu_{j-}^2, \qquad \sigma_{j+}^2 = \frac{\sum\limits_{\substack{i=1 \\ y_i=1}}^{m} w_i \mathbf{x}_{i,j}^2}{\sum\limits_{\substack{i=1 \\ y_i=1}}^{m} w_i} - \mu_{j+}^2. \tag{1.21}$$

[1] a dokładniej aproksymacja funkcji gęstości tych rozkładów

Odpowiedź nauczonego klasyfikatora wyznaczana jest poprzez twierdzenie Bayesa i w formie zlogarytmowanej jako:

$$\begin{aligned} f(\mathbf{x}_i) &= \frac{1}{2}\ln\frac{\widehat{p}_w(\mathbf{x}_{i,j}|y=1)\widehat{P}_w(y=1)}{\widehat{p}_w(\mathbf{x}_{i,j}|y=-1)\widehat{P}_w(y=-1)} \\ &= \frac{1}{2}\left(\frac{(\mathbf{x}_{i,j}-\mu_{j-})^2}{2\sigma_{j-}^2} - \frac{(\mathbf{x}_{i,j}-\mu_{j+})^2}{2\sigma_{j+}^2} + \ln\frac{\sigma_{j-}}{\sigma_{j+}} + \ln\frac{\widehat{P}_w(y=1)}{\widehat{P}_w(y=-1)}\right), \end{aligned} \tag{1.22}$$

gdzie j oznacza indeks cechy, dla którego kryterium wykładnicze jest najmniejsze, natomiast $\widehat{P}_w(y=\pm 1) = \sum\limits_{i:\ y_i=\pm 1} w_i$ są aktualnymi oszacowaniami prawdopodobieństw klas.

1.4.3 Kosze z odpowiedzią rzeczywistoliczbową

Kosze z odpowiedzią rzeczywistoliczbową przedstawione w pracy (Rasolzadeh i in., 2006) stanowią pewnego rodzaju usprawnienie rozkładów normalnych stosowanych jako słabe klasyfikatory. Kosze, w przeciwieństwie do rozkładów normalnych, pozwalają przybliżać wielomodalne rozkłady warunkowe funkcjami kawałkami stałymi. Klasyfikator ten można porównać do dwupoziomowego drzewa decyzyjnego z tą różnicą, że wprowadza się większą niż dwa liczbę terminali (liści). Progi decyzyjne są tutaj rozłożone równomiernie, dzieląc dziedzinę wybranego atrybutu na B równych fragmentów nazywanych koszami. Podobnie jak w przypadku modyfikacji „decision stumps" dla algorytmu RealBoost, każdy z koszy posiada swoje własne oszacowanie $\widehat{P}_w(y=\pm|\mathbf{x})$. Załóżmy, że $[a_1, a_2]$ reprezentuje przedział zmienności pewnej cechy. Chcąc obliczyć numer kosza $\beta(x) \in \{1,\ldots,B\}$, do którego należy dany argument x możemy posłużyć się wzorem:

$$\beta(x) = \begin{cases} 1, & x \leqslant a_1; \\ \lceil B \cdot \frac{x-a_1}{a_2-a_1} \rceil, & a_1 < x \leqslant a_2; \\ B, & x > a_2. \end{cases} \tag{1.23}$$

Znając indeks kosza odpowiedź klasyfikatora możemy obliczyć jako:

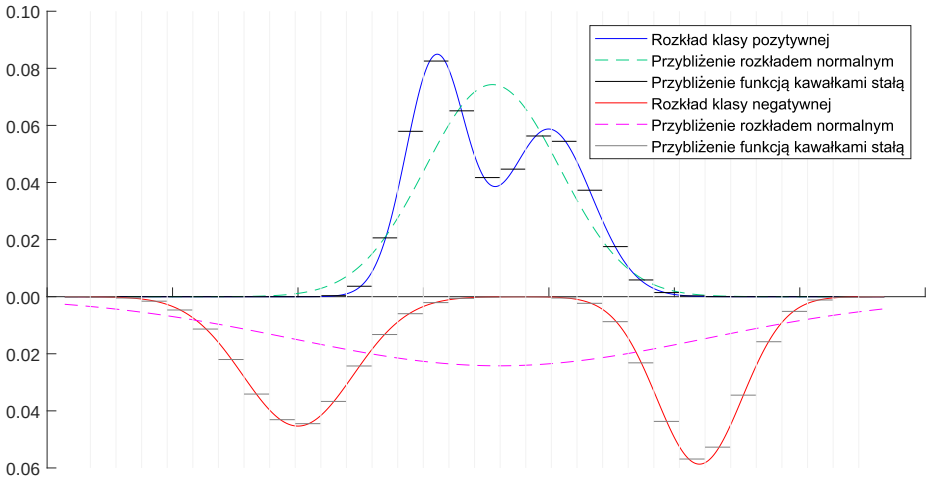
$$f(\mathbf{x}_i) = \frac{1}{2}\log\left(\frac{W_+(\beta(\mathbf{x}_{i,j}))}{W_-(\beta(\mathbf{x}_{i,j}))}\right), \tag{1.24}$$

$$W_+(b) = \sum_{\{i:\ y_i=1,\ \beta(\mathbf{x}_{i,j})=b\}} w_i, \qquad b=1,\ldots,B, \tag{1.25}$$

$$W_-(b) = \sum_{\{i:\ y_i=-1,\ \beta(\mathbf{x}_{i,j})=b\}} w_i, \qquad b=1,\ldots,B, \tag{1.26}$$

gdzie j oznacza numer wybranego atrybutu zaś $W_\pm(b)$ reprezentują sumy wag boostingowych pozytywów i negatywów w poszczególnych koszach.

Rys 1.3 ilustruje opisane powyżej różnice w sposobie aproksymacji funkcji gęstości rozkładu przez wspomniane klasyfikatory.



Rysunek 1.3: Zasada działania klasyfikatorów opartych o rozkłady normalne oraz koszy z odpowiedzią rzeczywistoliczbową. (Źródło: opracowanie własne)

1.5 Oczekiwana liczba ekstrahowanych cech

W niniejszym punkcie przedstawiono dwa dokładne podejścia i jedno przybliżone pozwalające na wyznaczenie oczekiwanej liczby ekstrahowanych cech dla kaskady klasyfikatorów (Sychel, Klęsk i Bera, 2018).

1.5.1 Wzór oparty na definicji

Należy przypomnieć, że zgodnie z przyjętą notacją n_k oznacza liczbę cech ekstrahowaną przez k-ty etap kaskady. Wiedząc, że kaskada kończy działanie po przebyciu pewnej liczby etapów, oraz że rozpoczęty etap musi się zakończyć (wszystkie cechy na danym etapie muszą zostać wyznaczone), możliwymi wynikami dla zmiennej losowej opisującej zdarzenia rozłączne są: $n_1,\ n_1+n_2,\ \ldots,\ n_1+n_2+\cdots+n_K$. Stąd, zgodnie z definicją wartości oczekiwanej, oczekiwana liczba cech może zostać obliczona następująco:

$$E\Big((n_1,\ldots,n_K)\Big) = \sum_{1\leqslant k\leqslant K}\Big(\sum_{1\leqslant i\leqslant k} n_i\Big)\Bigg(p\left(\prod_{1\leqslant i<k} d_i\right)(1-d_k)^{[k<K]} + (1-p)\left(\prod_{1\leqslant i<k} a_i\right)(1-a_k)^{[k<K]}\Bigg). \tag{1.27}$$

Dla przykładu, gdy rozpiszemy powyższy wzór dla $K=4$ otrzymamy:

$$\begin{aligned} E\Big((n_1,n_2,n_3,n_4)\Big) &= (n_1)(p(1-d_1)+(1-p)(1-a_1)) \\ &+ (n_1+n_2)(pd_1(1-d_2)+(1-p)a_1(1-a_2)) \\ &+ (n_1+n_2+n_3)(pd_1d_2(1-d_3)+(1-p)a_1a_2(1-a_3)) \\ &+ (n_1+n_2+n_3+n_4)(pd_1d_2d_3+(1-p)a_1a_2a_3). \end{aligned} \tag{1.28}$$

Jak widać we wzorze występują odpowiednie iloczyny prawdopodobieństw tj. czułości d_k lub FAR a_k w zależności od klasy (pozytywna / negatywna), oraz iloczyny te są przystawione do kolejnych możliwych wartości zmiennej losowej: $n_1,\ n_1+n_2,\ n_1+n_2+n_3,\ n_1+n_2+n_3+n_4$.

1.5.2 Wzór przyrostowy i jego aproksymacja

Grupując wyrażenia we wzorze (1.27) ze względu na n_k możemy otrzymać alternatywny, znacznie prostszy, wzór:

$$E\Big((n_1,\ldots,n_K)\Big) = \sum_{1\leqslant k\leqslant K} n_k \left(p \prod_{1\leqslant i<k} d_i + (1-p) \prod_{1\leqslant i<k} a_i \right). \tag{1.29}$$

Obydwa powyższe wzory posiadają jednak istotną wadę. Jest nią konieczność posiadania informacji o prawdziwym rozkładzie prawdopodobieństwa $(p, 1-p)$ dla używanych danych, który w praktycznych zastosowaniach jest często nieznany. Wiedząc jednak, że w zadaniach detekcji prawdopodobieństwo klasy pozytywnej p jest zazwyczaj bardzo małe (przeważnie $p < 10^{-4}$), wartość oczekiwana może zostać przybliżona z dużą dokładnością na podstawie sumy związanej tylko z klasą negatywną w następujący sposób:

$$\widehat{E}\Big((n_1,\ldots,n_K)\Big) = \sum_{1\leqslant k\leqslant K} n_k \prod_{1\leqslant i<k} a_i \approx E\Big((n_1,\ldots,n_K)\Big). \tag{1.30}$$

Co ciekawe, warto wspomnieć, że w oryginalnej pracy Violi i Jonesa (Viola i Jones, 2004) autorzy zaproponowali niepoprawny wzór do oszacowywania wartości oczekiwanej ekstrahowanych cech, mianowicie:

$$E_{\mathrm{VJ}}\Big((n_1,\ldots,n_K)\Big) = \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} r_i, \tag{1.31}$$

gdzie r_i reprezentuje „częstość wskazań pozytywnych" dla i-tego etapu, co odpowiada wyrażeniu:

$$E_{\mathrm{VJ}}\Big((n_1,\ldots,n_K)\Big) = \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} (pd_i + (1-p)a_i). \tag{1.32}$$

Mnożąc częstości wskazań pozytywnych dla etapów otrzymamy mieszane wyrazy postaci $d_i a_j$, które nie posiadają żadnego probabilistycznego sensu. Np. dla $k = 3$ iloczyn pod sumą wyniesie:

$$(pd_1 + (1-p)a_1)\,(pd_2 + (1-p)a_2)\,,$$

gdzie wyrażenia $d_1 a_2$ i $a_1 d_2$ nie mają sensu, ponieważ ten sam punkt danych $\mathbf{x}$ nie może zmienić swojej etykiety poruszając się wzdłuż kaskady.

Rozdział 2

Przykłady geometryczne

Klasyczny algorytm uczenia kaskady (o ustalonych cząstkowych wymaganiach na czułość i FAR dla każdego etapu) może prowadzić do nieoptymalnych rozwiązań pod względem oczekiwanej liczby ekstrahowanych cech. W celu weryfikacji powyższego stwierdzenia w niniejszym rozdziale zostały poddane analizie trzy odpowiednio skonstruowane przykłady geometryczne pozwalające zauważyć wspomnianą nieoptymalność oraz jej przyczyny (Sychel, Klęsk i Bera, 2018).

2.1 „Kostka w rogu"

Rozważmy rozkład danych zawarty w n-wymiarowej kostce jednostkowej, gdzie przykłady klasy pozytywnej zawarte są w prostopadłościanie umieszczonym w jednym z rogów kostki. Rozkład prawdopodobieństwa nad $(x_1, \ldots, x_n) \in [0,1]^n$ jest jednostajny dla całej dziedziny. Prostopadłościan zawierający pozytywy zdefiniujmy jako zbiór:

$$\mathcal{P} = \{(x_1, \ldots, x_n) \in [0,1]^n : 0 \leqslant x_j \leqslant w_j\},$$

gdzie $0 < w_1 < w_2 < \ldots < w_n \leqslant 1$ stanowią wymiary prostopadłościanu (uporządkowane rosnąco). Klasa negatywna stanowi dopełnienie klasy pozytywnej do całej kostki:

$$\mathcal{N} = [0,1]^n \setminus \mathcal{P}.$$

Opisany problem jest deterministyczny tzn. klasy można jednoznacznie rozdzielić cięciami prostopadłymi do osi układu. Na potrzeby tego zadania rozpatrzmy wariant algorytmu uczącego kaskadę klasyfikatorów działający bezpośrednio na ciągłych rozkładach, zamiast na skończonym zbiorze danych.

Załóżmy, że liczba etapów K oraz końcowe wymagania D, A (czułość, FAR) dla całej kaskady zostały narzucone. Każdy etap kaskady będzie składał się z pewnej grupy cięć prostopadłych do osi (podobnie jak w przypadku „decision stumpów"). Zbiór wyznaczony przez te cięcia będzie odpowiadał pozytywnej odpowiedzi klasyfikatora dla danego etapu. Przypuśćmy, że algorytm dodaje cięcia kolejno i pojedynczo. Algorytm 4 reprezentuje powyższą procedurę.

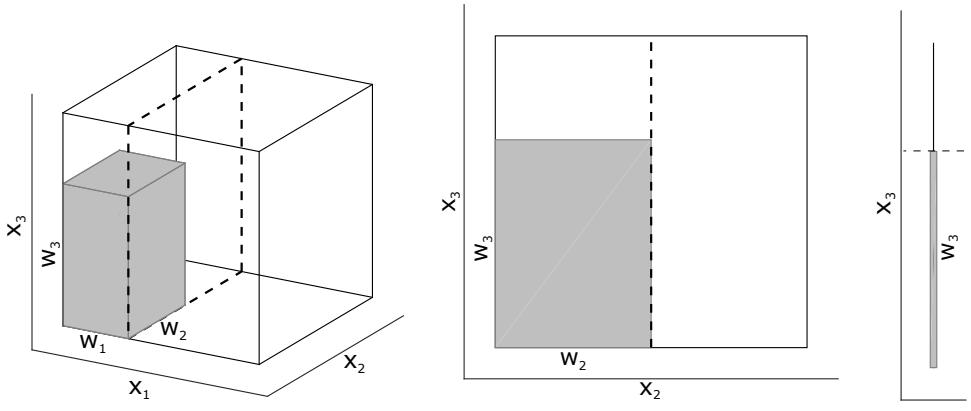
Algorytm 4 Uczenie kaskady w stylu podejścia Violi-Jonesa na potrzeby przykładu „kostka w rogu".

Algorytm TrainVJCascadeEx1(D, A, K)
$d_{\min} := D^{1/K}$, $a_{\max} := A^{1/K}$.
Dla $k := 1, \ldots, K$ **powtarzaj**
$n_k := 0$, $d_k := 0$, $a_k := 1$.
Dopóki ($d_k < d_{\min}$ lub $a_k > a_{\max}$) **powtarzaj**
Znajdź cechę x_j oraz punkt cięcia s_j (wzdłuż x_j), który zminimalizuje błąd klasyfikacji.
Dodaj cięcie (x_j, s_j) do aktualnego etapu.
$n_k := n_k + 1$.
Oblicz d_k, a_k.
Zwróć $(n_1, n_2, \ldots, n_K)$.

Rozpatrzmy przypadek, gdzie $D = 1$ (w związku z tym także $d_{\min} = 1$). Warunek ten niesie za sobą następujące konsekwencje:

1. cięcia muszą być wykonywane w punktach s_j o współrzędnych równych w_j,
2. kolejne wybrane cechy odpowiadają ich naturalnemu porządkowi $x_1, x_2, \ldots$.

Powód stojący za pierwszą konsekwencją jest taki, że dla $s_j < w_j$ stuprocentowa czułość nie może zostać osiągnięta, natomiast dla $s_j > w_j$ FAR zostanie niepotrzebnie zwiększony. Druga konsekwencja związana jest z tym, że kolejne wymiary prostopadłościanu zostały ułożone rosnąco ($w_1 < w_2 < \ldots < w_n$). Dlatego cięcie dla cechy x_{j_1} wykonane przed cięciem dla cechy x_{j_2}, gdzie $j_1 > j_2$, spowoduje wzrost miary FAR (co za tym idzie nie minimalizuje błędu). Rysunek 2.1 ilustruje rozkład danych i kolejne optymalne cięcia dla przykładu „kostka w rogu" i $n = 3$.



Rysunek 2.1: Przykład pierwszy „kostka w rogu": ilustracja dla $n = 3$. (Źródło: opracowanie własne)

2.1.1 Wzór na FAR dla „kostki w rogu”

Na potrzeby dalszej analizy wprowadzony zostanie wzór pozwalający na wyznaczenie wartości FAR. Wzór ten zakłada optymalną wartość $s_j = w_j$ i wyraża wartość FAR wynikającą z wykonania n_i nowych cięć (wykorzystania n_i nowych cech) zakładając, że n_0 cięć zostało już wykonanych.

$$\mathrm{FAR}(n_i; n_0) = \frac{\left(\prod_{j=1+n_0}^{n_0+n_i} w_j\right) \cdot \left(1 - \prod_{j=1+n_0+n_i}^{n} w_j\right)}{1 - \prod_{j=1+n_0}^{n} w_j} \tag{2.1}$$

Na przykład, jeśli $n = 3$ i zostanie wykonane 1 cięcie wynikowy FAR będzie równy:

$$\mathrm{FAR}(1; 0) = \frac{w_1 \left(1 - w_2 \cdot w_3\right)}{1 - w_1 \cdot w_2 \cdot w_3}.$$

Jeśli po wykonaniu pierwszego cięcia chcemy dokonać dwóch kolejnych FAR wyniesie:

$$\mathrm{FAR}(2; 1) = \frac{w_2 \cdot w_3 \left(1 - 1\right)}{1 - w_2 \cdot w_3} = 0.$$

Wartość $1-1$ występująca w liczniku jest wynikiem pustego iloczynu $\prod_{j=1+1+2}^{3} w_j$ równego zgodnie z definicją 1.

Wzór (2.1) pozwala nam ponownie zauważyć, że cechy na podstawie których dokonujemy cięć w tym przykładzie geometrycznym, muszą być wybierane zgodnie z ich naturalnym porządkiem. Jeśli algorytm wybrałby cechy w innej kolejności, to oba czynniki w liczniku wzrosłyby przy ustalonym mianowniku.

Należy także zauważyć, że wzór (2.1) pozwala obliczać FAR przyrostowo, tzn. dla wszystkich liczb naturalnych n_0, n_1, n_2 zachodzi

$$\mathrm{FAR}(n_1 + n_2; n_0) = \mathrm{FAR}(n_1; n_0) \cdot \mathrm{FAR}(n_2; n_0 + n_1). \tag{2.2}$$

Wykonanie $n_1 + n_2$ nowych cieć po wcześniejszym zrealizowaniu n_0 cięć jest równoważne do wykonania n_1 cięć, a następnie kolejnych n_2 cięć (gdy $n_0 + n_1$ poprzednich cięć zostało już zrealizowanych).

2.1.2 Optymalna kaskada dla „kostki w rogu”

Możemy teraz przystąpić do poszukiwania optymalnej kaskady dla „kostki w rogu”, badając różne wartości K. W tym celu zostaną wykorzystane dwie techniki: dokładana (oparta na wyczerpującym przeszukiwaniu) oraz przybliżona (wykorzystująca metody numeryczne).

Niech N^* będzie optymalną liczbą cech wystarczającą do osiągnięcia narzuconej wartości A (miara FAR).

$$N^* = \min\{n_i : \mathrm{FAR}(n_i; 0) \leqslant A\}$$

Możliwe jest, że Algorytm 4 uczący kaskadę znajdzie

$$N = n_1 + \cdots + n_K > N^*$$

cech. Nadmiar ten może być spowodowany przez konieczność osiągnięcia wymagań cząstkowych ($d_{\min},\ a_{\max}$) przez każdy etap kaskady.

Metoda dokładna — wyczerpujące przeszukiwanie

Znając liczbę etapów poszukiwanej kaskady K można zauważyć, że liczba różnych kaskad zawierających N^* cech wynosi:

$$\binom{N^*-1}{K-1}. \tag{2.3}$$

Obserwacja ta zostanie wyjaśniona na dwa sposoby.

Wyjaśnienie 1: Każdy etap musi posiadać co najmniej jeden słaby klasyfikator. Kaskady kończące się zerami w ciągu $(n_1, \ldots, n_k)$ mogą zostać pominięte. Kaskady te posiadałyby większą wartość $E\Big((n_1, \ldots, n_k)\Big)$ niż kaskady nie kończące się zerem a wykorzystujące taką samą liczbę cech N^*. Wzór (2.3) może zostać zinterpretowany jako liczba sposobów na umieszczenie $K-1$ separatorów w napisie $2, 3, \ldots, N^*$ reprezentującym kolejne indeksy cech (pomijamy tu indeks 1, jako że pierwsza cecha zawsze wchodzi w skład pierwszego etapu). Na przykład, jeżeli $N^* = 7$ i $K = 3$ jedną z kaskad jakie możemy utworzyć jest:

$$\underline{2}, 3, \underline{4}, 5, 6, 7 \to 1|2, 3|4, 5, 6, 7 \to \begin{matrix}(n_1, & n_2, & n_3)\\ (1, & 2, & 4)\end{matrix},$$

gdzie podkreślniki oznaczają cechę, przed którą postawiony będzie separator. Inną kaskadą spełniającą powyższe warunki może być np. kaskada:

$$2, \underline{3}, \underline{4}, 5, 6, 7 \to 1, 2|3|4, 5, 6, 7 \to \begin{matrix}(n_1, & n_2, & n_3)\\ (2, & 1, & 4)\end{matrix}.$$

Wyjaśnienie 2: Rozpatrzmy różne przypisania ze zbioru słabych klasyfikatorów (związanych z pojedynczymi cechami) do zbioru etapów kaskady:

$$\{1, 2, \ldots, N^*\} \to \{1, 2, \ldots, K\}.$$

Jeśli na moment pominiemy ograniczenie mówiące, że na każdym etapie musi być co najmniej jeden klasyfikator, to liczba możliwych przypisań będzie równa liczbie N^*-elementowych kombinacji z powtórzeniami zbioru K-elementowego:

$$\binom{N^*+K-1}{N^*}.$$

Wspomniane ograniczenie może zostać wprowadzone poprzez podstawienie $N^* := N^* - K$ (ponieważ K klasyfikatorów, 1 na każdy etap, jest ustalonych w kaskadzie od początku). Prowadzi to do następującej liczby kaskad

$$\binom{N^*-K+K-1}{N^*-K} = \binom{N^*-1}{N^*-K} = \binom{N^*-1}{K-1},$$

co odpowiada wzorowi (2.3). Kodowanie przykładowej kaskady $(n_1, n_2, n_3) = (1, 2, 4)$ z poprzedniego objaśnienia może być teraz odzwierciedlone poprzez następujący napis przedstawiający kombinację z powtórzeniami:

$$*|**|****.$$

Algorytm 5 Wyczerpujące przeszukiwanie w poszukiwaniu optymalnej kaskady (na potrzeby przykładu „kostka w rogu").

Algorytm ExhaustiveCascades(A, D, K, N)
- $E^* := \infty$. ▷ najlepsza wartość oczekiwana do tej pory
- $(N_1^*, \ldots, N_K^*) := \emptyset$. ▷ najlepsza kaskada do tej pory
- $c := (1, \ldots, N-K)$. ▷ początkowa kombinacja
- **Dopóki** prawda **powtarzaj**
  - Dekoduj c w kaskadę $(n_1, \ldots, n_K)$.
  - Oblicz $(d_1, \ldots, d_K)$ i $(a_1, \ldots, a_K)$.
  - **Jeżeli** $\prod_{k=1}^{K} d_k \geqslant D$ i $\prod_{k=1}^{K} a_k \leqslant A$ **to**
    - Oblicz E zgodnie ze wzorem (1.30).
    - **Jeżeli** $E < E^*$ **to**
      - $E^* := E$.
      - $(N_1^*, \ldots, N_K^*) := (n_1, \ldots, n_K)$.
  - $c :=$NextCombination(c, $N-1$).
  - **Jeżeli** $c = (1, \ldots, N-K)$ **to** ▷ ponownie uzyskano początkową kombinację
    - **Zwróć** $(N_1^*, \ldots, N_K^*)$, E^*.

Algorytm 5 przedstawia procedurę wyczerpującego przeszukiwania kaskad. Funkcja NextCombination$(\cdot)$ reprezentuje iteracyjną procedurę generowania kombinacji dostępną w większości środowisk programistycznych[1]. Początkowa kombinacja (bez powtórzeń) jest dekodowana tworząc kaskadę o postaci $(N-K+1, 1, 1, \ldots, 1)$, następna do postaci $(N-K, 2, 1, \ldots, 1)$, proces ten jest powtarzany w pętli, aż do ostatniej kaskady $(1, 1, 1, \ldots, N-K+1)$.

Poniżej zamieszczone zostały wyniki eksperymentów dla przykładu „kostka w rogu" dla różnych wymiarowości (liczby cech): $n = 30$ oraz $n = 50$. W obu przypadkach kolejne wymiary prostopadłościanu zawierającego pozytywy tworzą następujący ciąg arytmetyczny: $w_1 = 0.8$, $\Delta = (1 - w_1)/n$ oraz $w_j = w_1 + (j-1)\Delta$ dla $j = 2, \ldots, n$.

Eksperymenty dla *n = 30*: Wyniki przedstawione w Tabeli 2.1 pozwalają na porównanie kaskad otrzymanych za pomocą Algorytmu 4 (w stylu VJ) z kaskadami uzyskanymi za pomocą Algorytmu 5 (wyczerpującego przeszukiwania). Obie implementacje zostały wykonane w Wolfram Mathematica 10.4[2]. W obu przypadkach wartości FAR a_k zostały obliczone przy użyciu wzoru (2.1). Całkowita liczba cech ($N = n_1 + \cdots + n_K$) w kaskadzie uzyskanej za pomocą pierwszego algorytmu, została wykorzystana jako wejście do drugiego algorytmu. Znajomość prawdopodobieństwa klasy pozytywnej dla tego eksperymentu $p = \prod_{k=1}^{n} w_k \approx 0.035633$ oraz ciągów (d_k) i (a_k) pozwala na dokładne obliczenie wartości oczekiwanej $E\Big((n_1, \ldots, n_K)\Big)$ (dla uproszczenia w tabelach oznaczona jako $E(\ldots)$).

Jak można zauważyć wszystkie kaskady uzyskane za pomocą algorytmu w stylu VJ

[1] W badaniach wykorzystano funkcję `NextKSubset[]` z Wolfram Mathematica 10.4.

[2] Czasy pracy zostały zmierzone dla procesora Intel Xeon CPU E3-1505M 2.8 GHz, 4 rdzenie / 8 wątków.

Tablica 2.1: Przykład 1 „kostka w rogu" ($n = 30$): kaskada w stylu VJ kontra optymalna kaskada znaleziona poprzez wyczerpujące przeszukiwanie.

nr	K	A	uzyskana kaskada (algorytm w stylu VJ)	N	$E(\dots)$	optymalna kaskada (wyczerpujące przeszukiwanie)	$E(\dots)$	liczba kaskad	czas [s]
1	2	10^{-2}	$(12, 11)$	23	13.2884	$(7, 16)$	10.9849	22	0.016
2	2	10^{-3}	$(18, 10)$	28	18.6085	$(8, 20)$	12.2173	27	0.016
3	3	10^{-2}	$(8, 9, 7)$	24	10.3641	$(4, 6, 14)$	8.74884	253	0.094
4	3	10^{-3}	$(12, 11, 5)$	28	13.5038	$(4, 7, 17)$	9.29271	351	0.141
5	4	10^{-2}	$(6, 6, 6, 5)$	25	8.78596	$(3, 4, 6, 10)$	7.62453	1540	0.516
6	4	10^{-3}	$(9, 9, 7, 4)$	29	11.2031	$(3, 4, 7, 15)$	8.21372	3276	1.188

(kierowanego wymaganiami cząstkowymi czułość / miara FAR) posiadają gorszą wartość oczekiwaną $E\Big((n_1, \dots, n_K)\Big)$ niż kaskady uzyskane poprzez wyczerpujące przeszukiwanie. Rożnica ta pojawia się pomimo wykorzystania identycznej liczby cech przez porównywane kaskady (oznaczonej przez N) i jest ona związana z rozłożeniem cech na poszczególnych etapach.

Kolejna tabela (Tabela 2.2) pokazuje, że w dwóch przypadkach wartość N, dzięki zastosowaniu wyczerpującego przeszukiwania, może zostać pomniejszona do N^*, przy zachowaniu końcowych wymagań. Wraz ze zmniejszeniem liczby cech poprawie uległa także wartość oczekiwana $E\Big((n_1, \dots, n_K)\Big)$.

Tablica 2.2: Przykład 1 „kostka w rogu" ($n = 30$): poprawiona całkowita liczba cech N^*.

nr	N	N^*	optymalna kaskada (wyczerpujące przeszukiwanie z użyciem N^*)	$E(\dots)$	liczba kaskad	czas [s]
3	24	23	$(4, 6, 13)$	8.59409	231	0.078
6	29	28	$(3, 4, 7, 14)$	8.12233	2 925	1.016

Eksperymenty dla $n = 50$ Zwiększenie wymiarowości $n = 50$ sprawiło, że prawdopodobieństwo klasy pozytywnej dla tych eksperymentów wynosi $p = \prod_{k=1}^{n} w_k \approx 0.00415707$. Podobnie jak w poprzedniej serii eksperymentów wyniki zostały zanotowane w dwóch tabelach. Tabela 2.3 porównuje kaskady w stylu VJ z optymalnymi (wykorzystującymi te same łączne liczby cech N). Tabela 2.4 przedstawia możliwość poprawy łącznej liczby cech do N^* i prezentuje powiązane z nią optymalne kaskady.

Tablica 2.3: Przykład 1 „kostka w rogu" ($n = 50$): kaskada w stylu VJ kontra optymalna kaskada znaleziona poprzez wyczerpujące przeszukiwanie.

nr	K	A	uzyskana kaskada (algorytm w stylu VJ)	N	$E(\dots)$	optymalna kaskada (wyczerpujące przeszukiwanie)	$E(\dots)$	liczba kaskad	czas [s]
1	5	10^{-2}	$(5, 5, 6, 6, 7)$	29	7.98656	$(2, 3, 4, 7, 13)$	7.08589	20 475	7.281
2	5	10^{-3}	$(7, 8, 9, 10, 8)$	42	9.62653	$(2, 3, 5, 9, 23)$	7.61736	101 270	44.766
3	6	10^{-2}	$(4, 4, 5, 5, 6, 6)$	30	7.38845	$(2, 2, 3, 4, 7, 12)$	6.72574	118 755	44.453
4	6	10^{-3}	$(6, 7, 8, 8, 8, 6)$	43	8.94712	$(2, 3, 3, 5, 9, 21)$	7.12384	850 668	387.328

Jak można zauważyć rozłożenie cech przez algorytm w stylu VJ ponownie okazało się nieoptymalne, a wartość oczekiwana $E\Big((n_1, \dots, n_K)\Big)$ dla tych kaskad znacznie odbiegała

od optymalnej (dla N cech). Wraz ze wzrostem wymiarowości n do 50 wzrosła też liczba kaskad, gdzie łączna liczba cech N w kaskadach odstawała od optymalnej N^*. W tym przypadku dotyczyło to wszystkich badanych kaskad. W efekcie ich wartość oczekiwana odstawała jeszcze bardziej od optimum niż wynikałoby to z Tabeli 2.3.

Tablica 2.4: Przykład 1 „kostka w rogu" ($n = 50$): poprawiona całkowita liczba cech N^*.

nr	N	N^*	optymalna kaskada (wyczerpujące przeszukiwanie z użyciem N^*)	$E(\dots)$	liczba kaskad	czas [s]
1	29	27	$(2, 3, 4, 6, 12)$	6.97378	14 950	5.125
2	42	41	$(2, 3, 5, 9, 22)$	7.58432	91 390	39.281
3	30	27	$(2, 2, 3, 4, 6, 10)$	6.59502	65 780	23.093
4	43	41	$(2, 3, 3, 5, 9, 19)$	7.0788	658 008	289.906

Metoda przybliżona — optymalizacja z wykorzystaniem metod numerycznych

Przeszkodę w bezpośredniej optymalizacji wartości oczekiwanej $E\Big((n_1, \dots, n_K)\Big)$ stanowią dwie rzeczy:

- całkowita liczba cech do dyspozycji jest kombinatorycznie rozłożona na liczności $(n_1, \dots, n_K)$,
- zależność pomiędzy częstościami fałszywych alarmów a_k a liczbą cech na poszczególnych etapach n_k jest w ogólności nieznana oraz nieciągła.

W tym punkcie przedstawiona zostanie metoda pozwalająca obejść wspomniane przeszkody, pozwalając na znalezienie optymalnej kaskady poprzez numeryczną optymalizację bez konieczności wyczerpującego przeglądania kaskad. Przy czym należy zaznaczyć, że metoda ta jest bezpośrednio dostosowana do przykładu „kostka w rogu" (a nie ogólna). Zastosowanie podejścia numerycznego jest możliwe dzięki wprowadzeniu *ciągłego* przybliżonego wariantu wzoru (2.1) na $\mathrm{FAR}(n_i; n_0)$. W efekcie pozwala to także na utworzenie ciągłych wariantów wzorów (1.29) oraz (1.30) na $E\Big((n_1, \dots, n_K)\Big)$.

Wyprowadzenie wspomnianych wzorów rozpocznijmy od zapisania alternatywnej reprezentacji wzoru (2.1).

$$\mathrm{FAR}(n_i; n_0) = \prod_{k=1}^{n} w_k^{[1+n_0 \leqslant k] \cdot [k \leqslant n_0 + n_i]} \frac{\left(1 - \prod_{k=1}^{n} w_k^{[1+n_0+n_i \leqslant k] \cdot [k \leqslant n]}\right)}{\left(1 - \prod_{k=1}^{n} w_k^{[1+n_0 \leqslant k] \cdot [k \leqslant n]}\right)}. \tag{2.4}$$

Należy zauważyć, że wszystkie iloczyny iterują teraz od $k = 1$ do $k = n$, a prawdziwe ograniczenia na indeksy k zostały przeniesione do wykładników w postaci odpowiednich funkcji wskaźnikowych. Wykorzystując funkcję sigmoidalną $\phi_\beta(x) = \frac{1}{1+e^{-\beta x}}$, gdzie β parametryzuje jej stromość, możliwe jest przybliżenie funkcji wskaźnikowej dla wystarczająco dużego β:

$$[a \leqslant b] \approx \phi_\beta \left(b - a + \frac{1}{2}\right). \tag{2.5}$$

Wartość $\frac{1}{2}$ została dodana ponieważ nierówność $a \leqslant b$ nie jest ścisła. Biorąc pod uwagę powyższe informacje przybliżona wersja wzoru na FAR może zostać zapisana jako:

$$\mathrm{FAR}(n_i; n_0) \approx \prod_{k=1}^{n} w_k^{\phi_\beta(k-1-n_0+\frac{1}{2})\phi_\beta(n_0+n_i-k+\frac{1}{2})} \frac{\left(1-\prod_{k=1}^{n} w_k^{\phi_\beta(k-1-n_0-n_i+\frac{1}{2})\phi_\beta(n-k+\frac{1}{2})}\right)}{\left(1-\prod_{k=1}^{n} w_k^{\phi_\beta(k-1-n_0+\frac{1}{2})\phi_\beta(n-k+\frac{1}{2})}\right)}. \tag{2.6}$$

Prawa strona wzoru (2.6) może teraz zostać wstawiona do wzorów na wartość oczekiwaną (1.29) lub (1.30), co pozwoli na przeprowadzenie numerycznej optymalizacji.

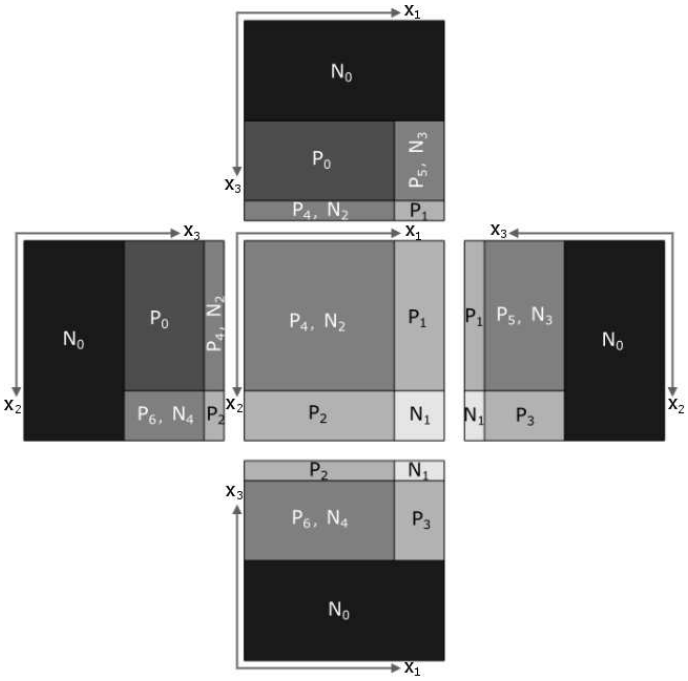
Tabela 2.5 przedstawia kaskady odnalezione numerycznie przy wykorzystaniu funkcji `NMinimize[·]` z Wolfram Mathematica przeprowadzającej optymalizację z ograniczeniami. Ograniczenia w tym wypadku to: $n_1 + \cdots + n_K = N$ i $n_k \in \mathbb{N}$. Parametr stromości β został ustawiony na 10.0. Wyniki w tabeli pokazują, że pomijając jeden przypadek wszystkie odnalezione kaskady były identyczne z tymi odnalezionymi za pomocą wyczerpującego przeszukiwania.

Tablica 2.5: Kaskady odnalezione numerycznie dzięki wykorzystaniu ciągłego przybliżenia wartości oczekiwanych (1.29) oraz (1.30).

n	K	A	optymalna kaskada (wyczerpujące przeszukiwanie)	kaskada dla kryterium (1.29) (`NMinimize[·]`)	czas [s]	kaskada dla kryterium (1.30) (`NMinimize[·]`)	czas [s]
30	2	10^{-2}	$(7, 16)$	$(7, 16)$	0.609	$(7, 16)$	0.620
30	2	10^{-3}	$(8, 20)$	$(8, 20)$	0.625	$(8, 20)$	0.625
30	3	10^{-2}	$(4, 6, 14)$	$(4, 6, 14)$	2.688	$(4, 6, 14)$	2.797
30	3	10^{-3}	$(4, 7, 17)$	$(4, 7, 17)$	2.750	$(4, 7, 17)$	2.800
30	4	10^{-2}	$(3, 4, 6, 10)$	$(3, 4, 6, 10)$	6.484	$(3, 4, 6, 10)$	6.422
30	4	10^{-3}	$(3, 4, 7, 15)$	$(3, 4, 7, 15)$	6.781	$(3, 4, 7, 15)$	6.703
50	5	10^{-2}	$(2, 3, 4, 7, 13)$	$(2, 3, 4, 7, 13)$	18.953	$(2, 3, 4, 7, 13)$	20.001
50	5	10^{-3}	$(2, 3, 5, 9, 23)$	$(2, 3, 5, 9, 23)$	19.313	$(2, 3, 5, 9, 23)$	19.578
50	6	10^{-2}	$(2, 2, 3, 4, 7, 12)$	$(2, 2, 3, 4, 7, 12)$	26.141	$(2, 2, 3, 4, 7, 12)$	26.875
50	6	10^{-3}	$(2, 3, 3, 5, 9, 21)$	$(2, 3, 4, 5, 9, 20)$	26.922	$(2, 3, 4, 5, 9, 20)$	27.297

2.2 „Kostka 3D”

Kolejny skonstruowany przykład geometryczny pozwoli pokazać nieoptymalność w kaskadzie klasyfikatorów uczonej standardowym algorytmem w stylu VJ przy niewielkiej wymiarowości problemu (kosztem zwiększonej złożoności). Tym razem przykład zawiera się dokładnie w trzech wymiarach jak pokazano na Rys. 2.2.



Rysunek 2.2: Przykład drugi „kostka 3D": wizualizacja. (Źródło: opracowanie własne)

Jak widać mamy tu do czynienia z sześcianem podzielonym na fragmenty z przykładami pozytywnymi, negatywnymi oraz na fragmenty „mieszane". Poprzez „mieszane" rozumiane są fragmenty, które nie mogą być rozdzielone cięciami prostopadłymi do osi w danej przestrzeni (zawierają one zarówno przykłady pozytywne jak i negatywne).

Rozkład prawdopodobieństw tym razem nie jest jednostajny. Dodatkowo na potrzeby tego przykładu wprowadzonych zostanie kilka nowych oznaczeń. Fragmenty z przykładami pozytywnymi zostaną oznaczone jako $\mathcal{P}_i$ a te z negatywnymi $\mathcal{N}_i$. Wartości prawdopodobieństwa klas dla poszczególnych fragmentów zostaną oznaczone jako $\mu(\mathcal{P}_i)$ lub $\mu(\mathcal{N}_i)$. W analizowanym przykładzie zostaną narzucone poniższe wartości (przy czym możliwe jest zastosowanie innych wariacji dających identyczne wnioski):

$$
\begin{aligned}
p = \mu(\mathcal{P}) &= 0.01, & 1 - p = \mu(\mathcal{N}) &= 0.99,\\
\mu(\mathcal{P}_0) &= 0.98\ \mu(\mathcal{P}), & \mu(\mathcal{N}_0) &= 0.975\ \mu(\mathcal{N}),\\
\mu(\mathcal{P}_1) = \mu(\mathcal{P}_2) = \mu(\mathcal{P}_3) &= 0.002\ \mu(\mathcal{P}), & \mu(\mathcal{N}_1) &= 0.002\ \mu(\mathcal{N}),\\
\mu(\mathcal{P}_4) &= 0.006\ \mu(\mathcal{P}), & \mu(\mathcal{N}_2) &= 0.001\ \mu(\mathcal{N}),\\
\mu(\mathcal{P}_5) = \mu(\mathcal{P}_6) &= 0.004\ \mu(\mathcal{P}), & \mu(N_3) = \mu(\mathcal{N}_4) &= 0.011\ \mu(\mathcal{N}).
\end{aligned}
$$

Podobnie jak w poprzednim przykładzie, jako słaby klasyfikator zostanie użyty klasyfikator jednoregułowy umożliwiający wykonanie cięcia prostopadłego do wybranej osi. W celu uproszczenia rozważań, cięć będzie można dokonać tylko wzdłuż granic między fragmentami. Ponadto w związku ze specyfiką problemu (podziałowi na prostopadłościenne fragmenty) jedna cecha może zostać wybrana kilkukrotnie.

2.2.1 Kaskada dla „kostki 3D”

Wymagania narzucone na kaskadę uczoną na danych z tego przykładu są następujące: $D = 0.9$, $A = 0.02$, $K = 2$. Skutkuje to narzuceniem na każdy etap wymagań cząstkowych równych: $d_{\min} \approx 0.9486833$, $a_{\max} \approx 0.1414214$. Poniższy opis przedstawi proces uczenia tak nastawionej kaskady algorytmem w stylu VJ, pokaże jej nieoptymalność oraz przedstawi optymalne rozwiązanie problemu.

Etap 1: W związku z wysoką wartością miary prawdopodobieństwa dla fragmentu $\mathcal{N}_0$ na początku wybrana zostanie cecha x_3 (jej wybór obciążony jest najmniejszym błędem i gwarantuje spełnienie wymagań przy minimalnej liczbie cech). Fragmenty dziedziny znajdujące się powyżej $\mathcal{N}_0$ zostaną sklasyfikowane jako pozytywy, natomiast $\mathcal{N}_0$ jako fragment negatywny. Ponieważ wszystkie fragmenty $\mathcal{P}_i$ zostały sklasyfikowane prawidłowo, to $d_1 = 1$ a $a_1 = 0.025$. Takie wartości zapewniają spełnienie wymagań cząstkowych i zakończenie uczenia tego etapu. Warto przy tym zaznaczyć, że wybór innej cechy spowodowałby wzrost wartości FAR lub spadek czułości zmuszając do wyboru kolejnej cechy.

Etap 2: Na kolejnym etapie mamy trzy możliwości: (I) ponownie wykorzystać x_3, (II) wykorzystać nową cechę (dokonać cięcia prostopadłego do x_1 lub x_2) lub (III) wykorzystać kombinację dwóch cech. Po odrzuceniu fragmentu N_0, zgodnie z własnościami kaskady, liczba negatywów ograniczy się do pozostałych 0.025 (zmiana ta będzie widoczna przy obliczaniu miary FAR dla tego etapu — a_2).

(I) Ponowny wybór cechy x_3 nie wpłynie na wartość oczekiwaną, ponieważ nie zwiększy się liczba cech wymaganych do sklasyfikowania przykładu. Wybór tej cechy sprawi, że punkty danych zawarte we fragmentach $\mathcal{P}_0,\ \mathcal{P}_3,\ \mathcal{P}_5,\ \mathcal{P}_6,\ \mathcal{N}_3,\ \mathcal{N}_4$ będą klasyfikowane jako pozytywy, a przykłady w obrębie pozostałych fragmentów jako negatywy. Co przełoży się na:

$$
\begin{aligned}
d_2 &= \frac{0.98 + 2 \cdot 0.004 + 0.002}{1} = 0.99 > d_{\min}, \\
a_2 &= \frac{2 \cdot 0.011}{0.025} = 0.88 > a_{\max}, \\
a_1 \cdot a_2 &> A.
\end{aligned}
$$

Zarówno wymagania końcowe jak i cząstkowe nie zostały więc spełnione.

(II) Wybór cechy x_1 spowoduje, że klasa pozytywna zostanie przypisana fragmentom $\mathcal{P}_0,\ \mathcal{P}_2,\ \mathcal{P}_4,\ \mathcal{P}_6,\ \mathcal{N}_2,\ \mathcal{N}_4$, a negatywna dla pozostałych. W związku z tym otrzymamy:

$$
\begin{aligned}
d_2 &= 0.9992 > d_{\min}, \\
a_2 &= 0.48 > a_{\max}, \\
d_1 \cdot d_2 &> D, \\
a_1 \cdot a_2 &< A.
\end{aligned}
$$

Warto w tym momencie zaznaczyć, że wybór cechy x_2 da identyczne wyniki, w związku z symetrią w kostce. Jak widać otrzymana w tym momencie kaskada spełniłaby wymagania końcowe, jednak wymagania cząstkowe nie zostały spełnione ($a_2 > a_{\max}$). Zatem algorytm ściśle przestrzegający wymagań na etap zostanie zmuszony do dodania kolejnej cechy (można to zauważyć analizując warunek dla wewnętrznej pętli w Algorytmie 1).

(IIIA) Aby zminimalizować FAR po wybraniu pary $x_1,\ x_2$ punkty przecięć musiałyby zostać tak dobrane, aby klasyfikator zwracał klasę pozytywną dla fragmentów $\mathcal{P}_0,\ \mathcal{P}_4,\ \mathcal{N}_2$. Pozostałe kombinacje pozwolą na poprawę czułości kosztem miary FAR. Wyniki dla tego przypadku to:

$$\begin{aligned} d_2 &= 0.986 > d_{\min}, \\ a_2 &= 0.04 < a_{\max}, \\ d_1 \cdot d_2 &> D, \\ a_1 \cdot a_2 &< A. \end{aligned}$$

Zatem zarówno wymagania końcowe jak i cząstkowe zostały spełnione.

(IIIB) Drugą możliwością jest para $x_1,\ x_3$ (para $x_2,\ x_3$ da identyczny wynik w związku ze wspomnianą symetrią). Jeśli chcemy zminimalizować wartość FAR, jako zawierające pozytywy muszą zostać oznaczone fragmenty $\mathcal{P}_0,\ \mathcal{P}_6,\ \mathcal{N}_4$. Co da w efekcie:

$$\begin{aligned} d_2 &= 0.984 > d_{\min}, \\ a_2 &= 0.44 > a_{\max}. \end{aligned}$$

Zatem wymagania cząstkowe nie zostały spełnione.

Podsumowanie: Spełnienie wymagań cząstkowych na poszczególnych etapach sprawiło, że niezbędne stało się dodanie kolejnej cechy do kaskady. Zatem wynikowa kaskada spełniająca te wymagania posiada łącznie 3 cechy rozłożone w konfiguracji $(n_1, n_2) = (1, 2)$, mianowicie x_3 na etapie pierwszym oraz x_1 i x_2 na etapie drugim. Możliwe jest jednak uzyskanie prostszego klasyfikatora. Jeśli zniesiemy konieczność przestrzegania wymagań na etap, możliwe staje się uzyskanie kaskady spełniającej zadane wymagania końcowe, posiadającej $(n_1,\ n_2) = (1,\ 1)$ cech, x_3 na pierwszym etapie oraz x_1 lub x_2 na etapie drugim.

2.2.2 Oczekiwana liczba cech dla „kostki 3D”

W tym momencie pozostało już tylko obliczenie oczekiwanych liczb cech dla wspomnianych kaskad oraz porównanie ich wartości. W przypadku kaskady w stylu VJ wartość ta wynosi:

$$\begin{aligned} E\Big((n_1, n_2)\Big) &= n_1 + n_2(pd_1 + (1-p)a_1) \\ &= 1 + 2(0.01 \cdot 1.0 + 0.99 \cdot 0.025) \\ &= 1.0695, \end{aligned}$$

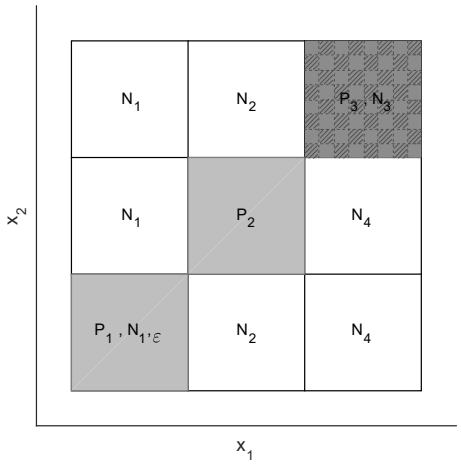
a w przypadku kaskady omijającej pułapkę:

$$\begin{aligned} E\Big((n_1, n_2)\Big) &= n_1 + n_2(pd_1 + (1-p)a_1) \\ &= 1 + 1(0.01 \cdot 1.0 + 0.99 \cdot 0.025) \\ &= 1.0348. \end{aligned}$$

Ścisłe przestrzeganie wymagań cząstkowych spowodowało wzrost liczby cech w końcowej kaskadzie mimo, że zadane wymagania końcowe zostały już przez nią spełnione. Efektem tego był wzrost oczekiwanej liczby cech, co w praktyce przekłada się na zwiększony czas klasyfikacji.

2.3 „Pułapka z szachownicą"

W ostatnim przykładzie geometrycznym dziedzina to ponownie n-wymiarowa kostka, przy czym większość granic decyzyjnych widoczna jest w podprzestrzeni x_1, x_2, jak przedstawiono na Rys. 2.3.



Rysunek 2.3: Przykład trzeci „pułapka z szachownicą": wizualizacja podprzestrzeni x_1, x_2. (Źródło: opracowanie własne)

Zbiór pozytywów dla tego przykładu został zdefiniowany jako:

$$\begin{aligned} \mathcal{P} =& \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3, \\ \mathcal{P}_1 =& \Big\{(x_1, \ldots, x_n) \in [0,1]^n \colon 0 \leqslant x_1, x_2 < \frac{1}{3} \text{ oraz } 0 \leqslant x_3 < 1 - \epsilon\Big\}, \\ \mathcal{P}_2 =& \Big\{(x_1, \ldots, x_n) \in [0,1]^n \colon \frac{1}{3} \leqslant x_1, x_2 < \frac{2}{3}\Big\}, \\ \mathcal{P}_3 =& \Big\{(x_1, \ldots, x_n) \in [0,1]^n : \frac{2}{3} \leqslant x_1, x_2 \leqslant 1 \text{ oraz } \textsc{szachownica}(x_3, \ldots, x_n; 8) = 1\Big\}, \end{aligned}$$

gdzie:

$$\textsc{szachownica}(z_1, \ldots, z_q; r) = (\lfloor z_1 \cdot r \rfloor + \ldots + \lfloor z_q \cdot r \rfloor) \bmod 2.$$

Natomiast zbiór negatywów jako $[0,1]^n \setminus \mathcal{P}$, co przekłada się na:

$$\begin{aligned}
\mathcal{N} =& \mathcal{N}_1 \cup \mathcal{N}_{1,\epsilon} \cup \mathcal{N}_2 \cup \mathcal{N}_3 \cup \mathcal{N}_4, \\
\mathcal{N}_1 =& \Big\{(x_1,\ldots,x_n) \in [0,1]^n \colon 0 \leqslant x_1 < \frac{1}{3} \text{ oraz } \frac{1}{3} \leqslant x_2 \leqslant 1\Big\}, \\
\mathcal{N}_{1,\epsilon} =& \Big\{(x_1,\ldots,x_n) \in [0,1]^n \colon \leqslant x_1, x_2 < \frac{1}{3} \text{ oraz } 1-\epsilon \leqslant x_3 \leqslant 1\Big\}, \\
\mathcal{N}_2 =& \Big\{(x_1,\ldots,x_n) \in [0,1]^n \colon \frac{1}{3} \leqslant x_1 < \frac{2}{3} \text{ oraz } (0 \leqslant x_2 < \frac{1}{3} \text{ or } \frac{2}{3} \leqslant x_2 \leqslant 1)\Big\}, \\
\mathcal{N}_3 =& \Big\{(x_1,\ldots,x_n) \in [0,1]^n \colon \frac{2}{3} \leqslant x_1, x_2 \leqslant 1 \text{ oraz } \textsc{szachownica}(x_3,\ldots,x_n;8) = 0\Big\}, \\
\mathcal{N}_4 =& \Big\{(x_1,\ldots,x_n) \in [0,1]^n \colon \frac{2}{3} \leqslant x_1 \leqslant 1 \text{ oraz } 0 \leqslant x_2 < \frac{2}{3}\Big\}.
\end{aligned}$$

Załóżmy ponownie, że cięcia będą wykonywane prostopadle do osi. Na podstawie rozłożenia danych dla tego przykładu można zauważyć, że: do wyizolowania $\mathcal{P}_2$ wystarczą dwie cechy (x_1 oraz x_2), do wyizolowania $\mathcal{P}_1$ trzy (x_1, x_2 oraz x_3), natomiast aby idealnie wyizolować $\mathcal{P}_3$ należy użyć wszystkich n cech. Jest to wymuszone przez takie rozmieszczenie pozytywów w $\mathcal{P}_3$ oraz negatywów z $\mathcal{N}_3$, aby tworzyły wzór szachownicy zdefiniowany dla pozostałych $n-2$ zmiennych $x_3,\ldots,x_n$. Szachownica ta tworzy pułapkę dla algorytmu ściśle przestrzegającego wymagań cząstkowych.

Miary prawdopodobieństwa dla tego przykładu zostały rozłożone zgodnie z poniższym opisem (inne wariacje są możliwe), przy czym parametry ϵ oraz α zostaną dobrane i omówione w dalszej części tego rozdziału:

$$\begin{aligned}
p = \mu(\mathcal{P}) &= 0.01, & 1-p = \mu(\mathcal{N}) &= 0.99, \\
\mu(\mathcal{P}_1) &= 0.05\ \mu(\mathcal{P}), & \mu(\mathcal{N}_1 \cup \mathcal{N}_{1,\epsilon}) &= 0.9\ \mu(\mathcal{N}), \\
\mu(\mathcal{P}_2) &= 0.85\ \mu(\mathcal{P}), & \mu(\mathcal{N}_{1,\epsilon}) &= \epsilon\ 0.9\ \mu(\mathcal{N}), \\
\mu(\mathcal{P}_3) &= 0.10\ \mu(\mathcal{P}), & \mu(\mathcal{N}_2 \cup \mathcal{N}_3 \cup \mathcal{N}_4) &= 0.1\ \mu(\mathcal{N}), \\
& & \text{gdzie } \mu(\mathcal{N}_3) &= \alpha\ 0.1\ \mu(\mathcal{N}) \quad (\text{np. } \alpha \geqslant 0.9).
\end{aligned}$$

Należy zauważyć wysoką wartość parametru α, która sprawia, że spora część z $0.1\mu(\mathcal{N})$ tj. miary prawdopodobieństwa negatywów znajdujących się poza regionem $\mathcal{N}_1$ oraz $\mathcal{N}_{1,\epsilon}$ zawiera się w szachownicy.

Algorytm 6 przedstawia pseudokod algorytmu uczenia kaskady dopasowanego do matematycznych własności rozpatrywanego przykładu. Ponownie uczenie jest kierowane przez wymagania cząstkowe czułości i miary FAR. Na kolejnych etapach klasyfikator wypróbowuje coraz bardziej złożone klasyfikatory bazujące na prostopadłych cięciach. Przebieg w pętli rozpoczyna się od pojedynczych cech, potem par cech, trójek cech, itd. Dla ustalonej kombinacji cech budowane jest drzewo decyzyjne, które może wykonać *dowolną* liczbę cięć (używając przy tym tylko wybranych cech) w celu minimalizacji błędu klasyfikacji. W momencie kiedy dana kombinacja cech pozwala na spełnienie wymagań $d_{\min}$ oraz $a_{\max}$, algorytm wstrzymuje dalsze wyszukiwanie i aktualny etap zostaje zamknięty.

Algorytm 6 Uczenie kaskady w stylu Viola Jones na potrzeby przykładu 3.

Algorytm TrainVJCascadeEx3(A, D, K)
  $d_{\min} := D^{1/K}$, $a_{\max} := A^{1/K}$.
  Dla $k := 1, \ldots, K$ **powtarzaj**
    $n_k := 0$, $d_k := 0$, $a_k := 1$.
    Dla $q := 1, \ldots, n$ **powtarzaj**
      Dla wszystkich kombinacji cech $x_{i_1}, \ldots, x_{i_d}$
        długości q $(1 \leqslant i_1 < \cdots < i_q \leqslant n)$ **powtarzaj**
        Zbuduj drzewo decyzyjne używając cech $x_{i_1}, \ldots, x_{i_q}$, które zminimalizują błąd klasyfikacji (przy dowolnej liczbie cięć).
        Oblicz d_k, a_k dla drzewa.
        Zapamiętaj drzewo, jeśli jest lepsze niż najlepsze do tej pory (dla aktualnego q).
      Jeżeli ($d_k \geqslant d_{\min}$ oraz $a_k \leqslant a_{\max}$) **to**
        Użyj najlepszego drzewa dla aktualnego etapu.
        $n_k := q$.
        Wyskocz z tej pętli.
  Zwróć $(n_1, n_2, \ldots, n_K)$.

2.3.1 Kaskada dla „pułapki z szachownicą"

Załóżmy, że na uczoną kaskadę zostały narzucone następujące wymagania końcowe: $K = 2$, $D = 0.9025$, $A = 0.01$. Co implikuje: $d_{\min} = D^{1/K} = 0.95$, $a_{\max} = A^{1/K} = 0.1$. Poniżej przedstawiono opis kolejnych etapów uczenia kaskady algorytmem w stylu VJ przy powyższych nastawach oraz opis momentu, kiedy algorytm wpadnie w pułapkę.

Etap 1: Można zauważyć, że aby spełnić wymagania dla tego etapu wystarczy wykorzystać jedną cechę x_1, co da następujący klasyfikator dla pierwszego etapu:

$$F_1(\mathbf{x}) = \begin{cases} -1, & \text{dla } 0 \leqslant x_1 < \frac{1}{3}, \\ 1, & \text{w przeciwnym razie.} \end{cases}$$

Przekłada się on na $d_1 = 0.95$ oraz $a_1 = 0.1$, co pozwala zamknąć aktualny etap przy $n_1 = 1$.

Etap 2: Wybranie jednej cechy na pierwszym etapie wymusi wybór wszystkich pozostałych cech na drugim etapie, ponieważ aby spełnić wymagania cząstkowe klasyfikator będzie musiał idealnie wyizolować fragment $\mathcal{P}_2$ (jest to obliczeniowo tanie, ponieważ tylko jedna cecha musi zostać dodana x_2) oraz pewien wycinek $\beta \in (0, 1)$ z fragmentu $\mathcal{P}_3$. W celu osiągnięcia $d_2 \geqslant d_{\min}$ następujący warunek musi zostać spełniony:

$$\frac{\mu(\mathcal{P}_2) + \beta\ \mu(\mathcal{P}_3)}{\mu(\mathcal{P}_2) + \mu(\mathcal{P}_3)} \geqslant d_{\min},$$

co daje $\beta \geqslant 0.525$. Należy zauważyć, że jeśli wszystkie z $n-2$ pozostałych cech $(x_3, \ldots, x_n)$ nie zostaną wybrane, a zamiast tego tylko pewien ich podzbiór, to obszar szachownicy nie zostanie idealnie rozpoznany (dzięki tej własności szachownica jest dogodna dla tego przykładu). Z kolei rozpoznanie jako pozytywów jedynie wycinka $\beta \cdot \mu(\mathcal{P}_3)$ z obszaru szachownicy spowoduje wprowadzenie $\beta \cdot \alpha \cdot \mu(\mathcal{N})$ fałszywych alarmów. Innymi słowy, aby zamknąć etap spełniony musiałby zostać warunek:

$$\frac{\beta\ \alpha\ 0.1\mu(\mathcal{N})}{0.1\mu(\mathcal{N})} \leqslant a_{\max},$$

co daje $\alpha \leqslant \frac{0.1}{0.525}$. Ponieważ wartość α została ustawiona na 0.9 warunek ten nie może zostać spełniony. W efekcie wszystkie pozostałe cechy $x_3, \ldots, x_n$ oraz x_2 muszą zostać wybrane, skutkując na drugim etapie klasyfikatorem o postaci:

$$F_2(\mathbf{x}) = \begin{cases} 1, & \text{for } \frac{1}{3} \leqslant x_1, x_2 < \frac{2}{3}, \\ 1, & \text{for } \frac{2}{3} \leqslant x_1, x_2 \leqslant 1 \text{ oraz } \textsc{szachownica}(x_3, \ldots, x_n; 8) = 1, \\ -1, & \text{w przeciwnym razie.} \end{cases}$$

Miary czułości oraz częstości fałszywych alarmów dla powyższego klasyfikatora wynoszą odpowiednio $d_2 = 1.0$ oraz $a_2 = 0.0$ co pozwala na zamknięcie ostatniego etapu.

2.3.2 Oczekiwana liczba cech dla „pułapki z szachownicą"

Pozostało obliczenie oraz porównanie wartości oczekiwanej dla powyższego klasyfikatora z jego odpowiednikiem omijającym pułapkę. Dla kaskady wpadającej w pułapkę wartość ta wynosi:

$$\begin{aligned} E\Big((n_1, n_2)\Big) &= n_1 + n_2(pd_1 + (1-p)a_1) \\ &= 1 + (n-1)(0.01 \cdot 0.95 + 0.99 \cdot 0.1) \\ &= 1 + 0.1085(n-1). \end{aligned}$$

Kaskada omijająca pułapkę: Aby ominąć pułapkę wystarczy na pierwszym etapie sklasyfikować cały obszar szachownicy $\mathcal{P}_3 \cup \mathcal{N}_3$ jako negatywny. Może to zostać osiągnięte na przykład przez kaskadę:

$$F_1(\mathbf{x}) = \begin{cases} 1, & \text{dla } 0 \leqslant x_1, x_2 < \frac{1}{3}, \\ 1, & \text{dla } \frac{2}{3} \leqslant x_1, x_2 \leqslant 1, \\ -1, & \text{w przeciwnym razie,} \end{cases}$$

$$F_2(\mathbf{x}) = \begin{cases} -1, & \text{dla } 0 \leqslant x_1, x_2 < \frac{1}{3} \text{ oraz } x_3 \geqslant 1 - \epsilon, \\ 1, & \text{w przeciwnym razie.} \end{cases}$$

Stąd dla pierwszego etapu otrzymamy: $n_1 = 2$, $d_1 = 0.9$ oraz $a_1 = \epsilon\ 0.9$. Natomiast dla drugiego etapu otrzymamy: $n_2 = 1$ (ponieważ x_3 jest jedyną nową zmienną), $d_2 = 1$ oraz $a_2 = 0$. Należy zaznaczyć, że odcięcie rejonu ϵ jest wymagane jeśli:

$$\frac{\epsilon\ 0.9\mu(\mathcal{N})}{\mu(\mathcal{N})} > A,$$

co daje $\epsilon > 0.01$.

Zatem wartość oczekiwana dla powyższej kaskady (dla przykładowej wartości $\epsilon = 0.02$) wynosi:

$$\begin{aligned} E\Big((n_1, n_2)\Big) &= n_1 + n_2(pd_1 + (1-p)a_1) \\ &= 2 + 1(0.01 \cdot 0.9 + 0.99\epsilon \cdot 0.9) \\ &= 2.009 + 0.891\epsilon \\ &=_{|\epsilon:=0.02} \ 2.02682. \end{aligned}$$

Jeśli porównamy obie kaskady, to zobaczymy, że kaskada omijająca pułapkę stanie się lepsza, gdy spełniony zostanie następujący warunek:

$$1 + 0.1085(n-1) > 2.02682,$$

dzieje się to gdy liczba cech (wymiarów) jest $n \geqslant 11$.

2.4 Wnioski dla przykładów geometrycznych

Powyżej przedstawione zostały trzy geometryczne przykłady demonstrujące jak uczenie kaskady z nałożonymi wymaganiami na każdy etap co do czułości oraz miary FAR może doprowadzić do nieoptymalności w koszcie obliczeniowym kaskady — a konkretnie w oczekiwanej liczbie cech $E\Big((n_1, \ldots, n_K)\Big)$ używanej przez kaskadę w momencie wykonywania procedury detekcyjnej.

Wprawdzie utworzone przykłady są sztuczne i motywowane pewnymi matematycznymi celami, to jednak podobne własności lub wariacje można łatwo spotkać w rzeczywistych danych, co potwierdziły testy na detektorach twarzy (Sychel, Klęsk i Bera, 2020a,b). Uogólniając wady uczenia przy ścisłym przestrzeganiu wymagań cząstkowych wskazane przez przedstawione przykłady, można sformułować następujące wnioski.

1. Etap spełniający wymagania $(d_{\min}, a_{\max})$ jest czasami zamykany przedwcześnie, co może spowodować wzrost liczby cech na kolejnych etapach. Czasami zwiększenie liczby cech na danym etapie (o ile nie za duże) może zaowocować zmniejszeniem wartości oczekiwanej $E\Big((n_1, \ldots, n_K)\Big)$ dla całej kaskady.

2. W wielu przypadkach istnieje możliwość zamknięcia etapu mimo nie spełnienia wymagań $(d_{\min}, a_{\max})$, a końcowe wymagania wciąż będą możliwe do osiągnięcia.

Wnioski te mogą pomóc uniknąć pułapek związanych z zastosowaniem zachłannego podejścia kierowanego wymaganiami na etap, pomimo że są one przeciwstawne (tzn. w ogólności nie do pogodzenia). Pierwszy z nich sugeruje, że czasami należy rozszerzać etap. Drugi zaś żeby zamykać etap przedwcześnie. Z punktu widzenia wzorów na wartość oczekiwaną drugi wniosek jest istotniejszy (jak pokazały przeprowadzane badania, zmniejszenie n_k na wczesnych etapach częściej prowadzi do zmniejszenia wartości oczekiwanej niż dodanie nowych cech), o ile idzie w parze z odpowiednio niedużymi częstościami a_k. Stąd z obu tych wniosków płynie pewna wskazówka na projektowanie algorytmów przeszukiwania (niekoniecznie wyczerpującego).

Rozdział 3

Proponowane rozwiązania

Rozdział ten prezentuje zestaw metod opracowanych na potrzeby pracy w celu redukcji wartości oczekiwanej uczonych kaskad w porównaniu z algorytmami ściśle przestrzegającymi wymagań cząstkowych. Dwie pierwsze z opisanych metod opierają się na wykorzystaniu różnicy, która może pojawić się na każdym etapie pomiędzy uzyskanymi wartościami $a_k,\ d_k$ a zadanymi. Metody te różnią się głównie sposobem wykorzystania tej różnicy tzn. zachłannością, z jaką to wykorzystanie jest przeprowadzane (Sychel, Klęsk i Bera, 2020b). Ostatnia z metod rozszerza tę koncepcję, wprowadzając uczenie oparte na przeszukiwaniu drzewa. Ponadto, w rozdziale tym przedstawiono dwa algorytmy pozwalające na przycinanie drzewa (powstającego w wyniku trzeciej metody), pozwalając w ten sposób na redukcję czasu nauki (Sychel, Klęsk i Bera, 2020a). Co ważne, każda z zaprezentowanych metod zatrzymuje się przy spełnieniu wymagań końcowych (podobnie jak klasyczne kaskady w stylu VJ), o ile oczywiście jest to możliwe dla danego zestawu cech.

3.1 Twierdzenie motywacyjne

Przedstawione poniżej twierdzenie wskazuje na możliwość poluzowania wymagań na etap, tym samym stanowi podstawową motywację autora do projektowania nowego typu algorytmów uczących kaskadę klasyfikatorów.

Twierdzenie 1: *Istnienie zapasu pomiędzy stałymi wymaganiami na etap* $(a_{max},\ d_{min})$ *a faktycznymi wskaźnikami* $(a_k,\ d_k)$, $k = 1, \ldots, K$, *zaobserwowanymi podczas uczenia kaskady —*

$$a_k = (1 - \epsilon_k)a_{max}, \qquad\qquad d_k = (1 + \delta_k)d_{min}, \tag{3.1}$$

gdzie ϵ_k, δ_k *reprezentują zmienne zapasu o małych wartościach — pozwala na wprowadzenie nowych poluzowanych wymagań dla każdego kolejnego etapu oraz przeprowadzenie procedury uczącej, która wciąż spełni wymagania końcowe* $(A,\ D)$ *dla nauczonej kaskady. W szczególności, gdy k-ty etap zostaje zakończony, następujące dwie pary poluzowanych ograniczeń (równomierne i zachłanne) mogą zostać zastosowane dla etapu* $k + 1$*:*

$$a_{k+1} \leqslant \frac{a_{max}}{(1 - \epsilon_{\leqslant k})^{1/(K-k)}}, \qquad\qquad d_{k+1} \geqslant \frac{d_{min}}{(1 + \delta_{\leqslant k})^{1/(K-k)}}, \tag{3.2}$$

lub

$$a_{k+1} \leqslant \frac{a_{max}}{1-\epsilon_{\leqslant k}}, \qquad d_{k+1} \geqslant \frac{d_{min}}{1+\delta_{\leqslant k}}, \tag{3.3}$$

gdzie

$$1-\epsilon_{\leqslant k} = \prod_{1 \leqslant i \leqslant k} (1-\epsilon_i)$$

oraz

$$1+\delta_{\leqslant k} = \prod_{1 \leqslant i \leqslant k} (1+\delta_i).$$

Przed przedstawieniem dowodu warto wypowiedzieć poniższe uwagi. Twierdzenie mówi, że zmienne reprezentujące zapas ϵ_k, δ_k są małymi liczbami, ale celowo nie wspomina o ich znaku. Intuicyjnie wydawać by się mogło, że $\epsilon_k, \delta_k \geqslant 0$. Jednak gdy poluzowane wymagania zostaną zastosowane w procedurze uczącej, część spośród ϵ_k, δ_k może być ujemna. Aby to zauważyć, rozpatrzmy przypadek dwóch pierwszych zmiennych ϵ_1, ϵ_2 oraz ograniczenia (3.3). Wynika z niego, że $a_2 = (1-\epsilon_2)a_{\max} \leqslant a_{\max}/(1-\epsilon_1)$ i w związku z tym:

$$\epsilon_2 \geqslant \frac{-\epsilon_1}{1-\epsilon_1}. \tag{3.4}$$

Jednocześnie zauważmy, że $\epsilon_1 \geqslant 0$ ponieważ $a_1 = (1-\epsilon_1)a_{\max}$ nie może przekroczyć oryginalnych ograniczeń $a_{\max}$. Zatem, nierówność (3.4) informuje nas, że zmienna ϵ_2 w szczególności może być ujemna. Co jednak ważne, "efektywne" zmienne swobodne $\epsilon_{\leqslant k}, \delta_{\leqslant k}$ (wynikające z iloczynu) muszą być nieujemne, tak aby poluzowanie w ograniczeniach (3.2), (3.3) rzeczywiście miało miejsce. Poniższy lemat stwierdza ten fakt i jest później wykorzystany do udowodnienia głównego twierdzenia.

Lemat 1: $\epsilon_{\leqslant k} \geqslant 0$ *oraz* $\delta_{\leqslant k} \geqslant 0$ *dla każdego* $k = 1, \ldots, K$ *niezależnie od zastosowanego ograniczenia: równomiernego* (3.2) *lub zachłannego* (3.3)*.*

Dowód: Wystarcza udowodnić lemat tylko dla zmiennej $\epsilon_{\leqslant k}$, ponieważ argumentacja dla $\delta_{\leqslant k}$ jest analogiczna, zmianie ulega jedynie kierunek wszystkich nierówności. Zacznijmy od ograniczenia równomiernego (3.2) i udowodnijmy lemat poprzez indukcję. Baza: $\epsilon_{\leqslant 1} = \epsilon_1 \geqslant 0$ wynika z ograniczenia na $a_1 = (1-\epsilon_1)a_{\max} \leqslant a_{\max}/\prod_{1 \leqslant i \leqslant 0}(1-\epsilon_i)^{1/(K-0)}$ ponieważ pusty iloczyn w mianowniku wynosi 1. Krok indukcyjny: załóżmy, że lemat jest prawdziwy dla wszystkich indeksów aż do k, to jest $\epsilon_{\leqslant k} \geqslant 0$. Zatem, aby zobaczyć, że $\epsilon_{\leqslant k+1} \geqslant 0$, wystarczy pokazać, że $1-\epsilon_{\leqslant k+1} = (1-\epsilon_{\leqslant k})(1-\epsilon_{k+1}) \leqslant 1$. Z ograniczenia (3.2) wiemy, że:

$$\begin{aligned}
a_{k+1} &\leqslant \frac{a_{\max}}{(1-\epsilon_{\leqslant k})^{1/(K-k)}} \\
(1-\epsilon_{k+1})a_{\max} &\leqslant \frac{a_{\max}}{(1-\epsilon_{\leqslant k})^{1/(K-k)}} \\
(1-\epsilon_{k+1})(1-\epsilon_{\leqslant k})^{1/(K-k)} &\leqslant 1 \\
(1-\epsilon_{k+1})(1-\epsilon_{\leqslant k}) &\leqslant 1 \\
1-\epsilon_{\leqslant k+1} &\leqslant 1.
\end{aligned} \tag{3.5}$$

Trzecie przejście jest prawdziwe ponieważ indukcyjne założenie $\epsilon_{\leqslant k} \geqslant 0$ implikuje, że wyrażenie $(1-\epsilon_{\leqslant k})^{1/(K-k)}$ jest ułamkiem podniesionym do ułamkowej potęgi. Zatem pominięcie potęgi pomniejsza lewą stronę nierówności. To dowodzi lematu dla ograniczenia równomiernego. W przypadku ograniczenia zachłannego (3.3) łatwo jest sprawdzić, że krok indukcyjny prowadzi bezpośrednio do nierówności: $(1-\epsilon_{k+1})(1-\epsilon_{\leqslant k}) \leqslant 1$ oraz, że baza jest także spełniona. □

Dowód Twierdzenia 1: Na mocy lematu 1 zauważmy, że gdy $\epsilon_{\leqslant k}, \delta_{\leqslant k} > 0$, to mianowniki w (3.2) oraz (3.3) spowodują, że nowe wymagania na etap mogą zostać spełnione łatwiej niż te oryginalne. Ograniczmy teraz dalsze rozważana jedynie do wskaźników fałszywych alarmów $a_1, \ldots, a_K$ (rozumowanie dla wskaźników czułości jest analogiczne). Załóżmy, że k etapów uczenia zostało już zakończonych. Aby spełnić wymagania końcowe następująca nierówność musi zostać spełniona

$$\begin{aligned} a_1 \cdots a_k \cdot a_{k+1} \cdots a_K &\leqslant A \\ \underbrace{(1-\epsilon_1)a_{\max} \cdots (1-\epsilon_k)a_{\max}}_{k \text{ początkowych etapów}} \cdot a_{k+1} \cdots a_K &\leqslant a_{\max}^K \\ a_{k+1} \cdots a_K &\leqslant \frac{a_{\max}^{K-k}}{\prod\limits_{1 \leqslant i \leqslant k} (1-\epsilon_i)}. \end{aligned} \tag{3.6}$$

Możemy teraz zauważyć dwa podejścia do napisania ograniczeń na pozostałe wskaźniki $a_{k+1}, \ldots, a_K$. Pierwsze podejście to pozwolić wszystkim pozostałym częstościom na równomierne wykorzystanie skumulowanego dotąd zapasu $\prod_{1 \leqslant i \leqslant k}(1-\epsilon_i) = (1-\epsilon_{\leqslant k})$. W tym celu pomyślmy o pewnym średnim czynniku a_*, takim że $a_*^{K-k} = a_{k+1} \cdots a_K$ w (3.6). Pierwiastek stopnia $K-k$ zastosowany obustronnie skutkuje wzorem (3.2) — równomierne ograniczenie. Rozwiązanie to może być interpretowane jako aktualizowana średnia geometryczna dla pozostałych częstości fałszywych alarmów. Drugie podejście to pozwolić na wykorzystanie całego zapasu przez kolejną (najbliższą) częstość a_{k+1}, zakładając pesymistycznie, że kolejne częstości $a_{k+2}, \ldots, a_K$ będą równe oryginalnym wymaganiom $a_{\max}$. Prowadzi to do nierówności $a_{k+1} a_{\max}^{K-k-1} \leqslant a_{\max}^{K-k}/(1-\epsilon_{\leqslant k})$ i skutkuje wzorem (3.3) — zachłanne ograniczenie. □

Jeśli chodzi o drugie podejście, to jego zachłanność może dodatkowo być wytłumaczona poprzez następującą konsekwencję poluzowanych wymagań (3.3).

Konsekwencja 1: *Jeżeli $(k+1)$-ty etap zostanie nauczony zgodnie z podejściem zachłannym oraz zaobserwowane wynikowe a_{k+1} nie jest mniejsze a dokładnie równe ograniczeniu $a_{max}/(1-\epsilon_{\leqslant k})$, wtedy wymaganie na kolejny etap dla a_{k+2} zacieśnia się z powrotem do oryginalnego ograniczenia a_{max}.*

Dowód: Wynik powstaje bezpośrednio z (3.6) poprzez: wyizolowanie a_{k+2}, podstawienie prawej strony równania (3.3) do a_{k+1} i podstawienie $a_i = a_{\max}$ dla każdego $i \geqslant k+3$. □

Analogiczna argumentacja jest prawdziwa dla każdych dwóch następujących po sobie czułości d_{k+1}, d_{k+2}.

3.2 Związek między poluzowanymi wymaganiami na etap a wartością oczekiwaną

Ponieważ poluzowane wymagania na etap są łatwiejsze do spełnienia (przez mniejszą liczbę cech) wydawać by się mogło, że prowadzą one bezpośrednio do redukcji $\widehat{E}(n)$. Niestety w ogólności nie jest to prawdziwe. Pomimo, że liczby n_k we wzorze (1.30) mogą w większości przypadków zostać zmniejszone, to należy zauważyć, że zmniejszenie ich może nieść ze sobą koszt w postaci zwiększenia wskaźników fałszywych alarmów a_k, które także występują we wzorze (1.30). Co więcej każde zwiększenie a_k przyczynia się w sposób multiplikatywny do wzrostu we wszystkich kolejnych składnikach wartości oczekiwanej.

Poniżej przedstawiony zostanie szczególny rezultat odnoszący się tylko do drugiego etapu kaskady klasyfikatorów. Pokazuje on warunki, jakie muszą zostać spełnione, aby zmniejszenie liczby cech na drugim etapie rzeczywiście przełożyło się na redukcję wartości oczekiwanej.

Na potrzeby tego rezultatu wprowadźmy następującą dodatkową notację. Niech $n' = (n'_1, \ldots, n'_K)$ oznacza liczby cech w kaskadzie nauczonej przy zastosowaniu poluzowanych wymagań na etap (dowolnego typu), natomiast $a' = (a'_1, \ldots, a'_K)$ reprezentuje częstości fałszywych alarmów dla tej kaskady. Przyjmijmy, że liczby $m_k = n_k - n'_k$ oznaczają różnice w liczbie cech na poszczególnych etapach względem klasycznej kaskady VJ (gdyby hipotetycznie jej uczenie zostało przeprowadzone na tych samych danych). Oczywiście chcielibyśmy, aby $m_k > 0$ na pewnej liczbie etapów, co świadczyłoby o potencjalnym zysku, jednakże należy pamiętać, że $m_k < 0$ jest także możliwe, zaś $m_k = 0$ reprezentuje brak zmiany. Dodatkowo, niech zmienne ε_k reprezentują zapas pomiędzy częstościami fałszywych alarmów osiągniętymi przez obie kaskady: $a'_k = (1+\varepsilon_k)a_k$. Należy zwrócić uwagę na notacyjną i znaczeniową różnicę pomiędzy powyższymi nowymi zmiennymi a ϵ_k, które związane były z zapasem w odniesieniu do oryginalnych wymagań. Innymi słowy, typową relacją jest $a_k \leqslant a'_k$, gdzie a'_k może zostać przedstawione zarówno jako $a'_k = (1+\varepsilon_k)a_k$ lub $a'_k = (1-\epsilon_k)a_{\max}$.

Twierdzenie 2: *Drugi etap kaskady klasyfikatorów nauczony przy pomocy poluzowanych wymagań na etap może przyczynić się do redukcji wartości oczekiwanej liczby cech (to znaczy $\widehat{E}(n') < \widehat{E}(n)$) wtedy, i tylko wtedy, gdy,*

$$m_2 > \sum_{2<k\leqslant K} n_k \varepsilon_{<k} \prod_{2\leqslant i<k} a_i - \sum_{2<k\leqslant K} m_k (1+\varepsilon_{<k}) \prod_{2\leqslant i<k} a_i, \tag{3.7}$$

gdzie $(1+\varepsilon_{<k}) = \prod_{1\leqslant i<k} (1+\varepsilon_i)$.

Wynik ten ma ciekawą interpretację. Zysk m_2 musi być większy niż różnica dwóch przyszłych oczekiwanych — wartości oczekiwanej kolejnych oryginalnych liczności n_k ważonych zmiennymi $\varepsilon_{<k}$ oraz wartości oczekiwanej kolejnych zysków (lub strat) m_k ważonych zmiennymi $1+\varepsilon_{<k}$.

Dowód: Rezultat wynika z definicji $\widehat{E}$, rozpoczynając wyprowadzenie od nierówności $\widehat{E}(n') < \widehat{E}(n)$, i zauważając, że $n'_1 = n_1$, $a'_1 = a_1$ (stąd także $\varepsilon_1 = 0$), ponieważ dla

pierwszego etapu obowiązują takie same wymagania w obu wariantach uczenia. Kiedy zredukujemy liczności n_1 po obu stronach

$$n_1 + n_2' a_1 + \sum_{2<k\leqslant K} n_k' \prod_{1\leqslant i<k} a_i' < n_1 + n_2 a_1 + \sum_{2<k\leqslant K} n_k \prod_{1\leqslant i<k} a_i$$

doprowadzając do

$$(n_2 - m_2)a_1 + \sum_{2<k\leqslant K} (n_k - m_k)a_1 \prod_{2\leqslant i<k} a_i(1+\varepsilon_i) < n_2 a_1 + \sum_{2<k\leqslant K} n_k a_1 \prod_{2\leqslant i<k} a_i,$$

a następnie pogrupujemy wyrażenia i podzielimy nierówność obustronnie przez a_1 otrzymamy ostateczny wynik. □

3.3 Uczenie kaskady z wykorzystaniem aktualizowanych średnich geometrycznych

3.3.1 Wyrażenie poluzowanych wymagań bez użycia zmiennej swobodnej

Na poziomie implementacji nie ma potrzeby bezpośredniego wyliczania wartości zmiennych ϵ_i, δ_i reprezentujących zapas. Zostały one wprowadzone jedynie w celu zaprezentowania pewnych matematycznych własności. Nowe wymagania na etap mogą zostać wyrażone z wykorzystaniem stałych A, D oraz częstości a_i, d_i zaobserwowanych do tej pory. To znaczy dla $i \leqslant k$ można łatwo sprawdzić, że następujące pary wzorów są równoważnymi odpowiednikami prawych stron odpowiednio nierówności (3.2) oraz (3.3).

$$d_{\min,k+1} = \left(\frac{D}{\prod\limits_{1\leqslant i\leqslant k} d_i}\right)^{\frac{1}{K-k}}, \qquad a_{\max,k+1} = \left(\frac{A}{\prod\limits_{1\leqslant i\leqslant k} a_i}\right)^{\frac{1}{K-k}}. \tag{3.8}$$

$$d_{\min,k+1} = \frac{D^{\frac{k+1}{K}}}{\prod\limits_{1\leqslant i\leqslant k} d_i}, \qquad a_{\max,k+1} = \frac{A^{\frac{k+1}{K}}}{\prod\limits_{1\leqslant i\leqslant k} a_i}. \tag{3.9}$$

W dalszej części pracy w rozdziale opisującym eksperymenty, w celu rozróżnienia między typami zastosowanych ograniczeń, zostaną wykorzystane skróty:
UGM (Updated Geometric Mean) — oznaczający aktualizowaną średnią geometryczną i reprezentujący ograniczenie (3.8),
UGM-G (Updated Geometric Mean - Greedy) — oznaczający aktualizowaną średnią geometryczną — wariant zachłanny i reprezentujący ograniczenie (3.9).

3.3.2 Algorytm uczący

Równania (3.8) oraz (3.9) można w prosty sposób połączyć z klasyczną procedurą uczącą w stylu VJ (Algorytm 1) uzyskując w ten sposób jej dwa nowe warianty. W tym celu wystarczy zastosować wybrane wzory, jako zamienniki dla stałych wymagań na etap $(a_{\max}, d_{\min})$, w każdej iteracji głównej pętli. Poniższy pseudokod (Algorytm 7) reprezentuje takie „poluzowane" uczenie kaskady.

Algorytm 7 Algorytm uczący kaskadę klasyfikatorów z poluzowanymi wymaganiami na etap.

Algorytm TrainRelaxedCascade($\mathcal{D}$, A, D, K, $\mathcal{V}$)

Ustal podzbiór przykładów pozytywnych $\mathcal{P}$ i negatywnych $\mathcal{N}$ w ramach zbioru $\mathcal{D}$.

$F := ()$. ▷ początkowa kaskada — pusty ciąg

$A_0 := 1$, $D_0 := 1$, $k := 0$.

Dopóki $A_k > A$ **powtarzaj**

$F_{k+1} :=$TrainStage($\mathcal{P}$, $\mathcal{N}$, K, k, $\mathcal{V}$, F).

$F := F \| F_{k+1}$.

Jeżeli $A_{k+1} > A$ **to**

$\mathcal{N} := \emptyset$.

Użyj aktualnej kaskady F w celu uzupełnienia zbioru $\mathcal{N}$ fałszywi wykryciami próbkowanymi z obrazów nie zawierających wykrywanych obiektów.

$k := k + 1$

Zwróć $F = (F_1, F_2, \ldots, F_k)$.

Algorytm TrainStage($\mathcal{P}$, $\mathcal{N}$, K, k, $\mathcal{V}$, F)

$n_{k+1} := 0$, $F_{k+1} := 0$, $A_{k+1} := A_k$.

Oblicz poluzowane wymagania na etap $(a_{\max,k+1}, d_{\min,k+1})$ używając (3.8) lub (3.9).

$a_{k+1} := A_{k+1}/A_k$.

Dopóki $a_{k+1} > a_{\max,k+1}$ **powtarzaj**

Naucz nowy słaby klasyfikator f (wykorzystujący jedną wybraną cechę) używając przykładów w zbiorach $\mathcal{P}$ i $\mathcal{N}$.

$F_{k+1} := F_{k+1} + f$.

$n_{k+1} := n_{k+1} + 1$.

Dostosuj próg decyzyjny θ_{k+1} dla F_{k+1}, tak aby spełnił wymaganie $d_{\min,k+1}$.

Użyj kaskady $F \| F_{k+1}$ na zbiorze walidacyjnym $\mathcal{V}$, aby zmierzyć A_{k+1} i D_{k+1}.

$a_{k+1} := A_{k+1}/A_k$.

Zwróć F_k.

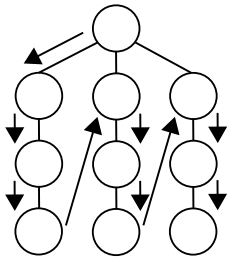
W treści Algorytmu 7 celowo została wyróżniona procedura TrainStage odpowiedzialna za naukę pojedynczego etapu kaskady. Procedura ta zostanie wykorzystana ponownie w dalszej części pracy, jako budulec dla bardziej ogólnego podejścia do nauki kaskad.

3.4 Nauka bazująca na przeszukiwaniu drzewa

W przypadku nauki ze ścisłym przestrzeganiem wymagań na etap w momencie, gdy nauka danego etapu zostanie zakończona nie może on już zostać zmieniony. W artykule (Saberian i Vasconcelos, 2014) zaproponowano odmienne podejście, w którym każdy etap może zostać rozszerzony poprzez dodanie nowego klasyfikatora w dowolnym momencie

nauki kaskady[1]. W rozdziale tym zostanie przedstawiona metoda pośrednia, znajdująca się gdzieś pomiędzy tymi dwoma podejściami. Z jednej strony jest ona bardziej elastyczna od nauki ze ścisłym przestrzeganiem wymagań na etap, z drugiej pozwala na uniknięcie nadmiernej złożoności obliczeniowej podejścia z (Saberian i Vasconcelos, 2014).

Zaproponowana metoda traktuje naukę kaskady jako proces przeszukiwania drzewa (Sychel, Klęsk i Bera, 2020a). Korzeń drzewa reprezentuje pustą kaskadę. Kolejne poziomy drzewa odpowiadają kolejnym etapom kaskady. Każdy węzeł nie będący liściem posiada nieparzystą liczbę potomków. Reprezentują oni różne warianty kolejnych etapów kaskady z lekko zmienioną liczbą zastosowanych cech. Potomkowie są przeszukiwani rekurencyjnie od lewej strony do prawej dopóki warunek stopu nie zostanie spełniony (Rys. 3.1). Należy rozumieć, że omawiane węzły nie są jakimiś sztucznie generowanymi obiektami na potrzeby przeszukiwania, a faktycznie reprezentują zespołowe klasyfikatory podlegające długiemu procesowi uczenia.



Rysunek 3.1: Uczenie kaskad jako przeszukiwanie drzewa w głąb. (Źródło: opracowanie własne)

Rozmiar drzewa będzie kontrolowany za pomocą dwóch całkowitoliczbowych parametrów L oraz C, o wartościach ustalonych przez użytkownika przed rozpoczęciem nauki. Aby rozmiar drzewa pozostał stosunkowo mały, drzewo zostanie rozgałęzione jedynie na L pierwszych poziomach. Na poziomach tych współczynnik rozgałęzienia będzie wynosił C, wartość ta musi być liczbą nieparzystą. Na dalszych poziomach współczynnik ten wyniesie jeden (Rys. 3.2). Podejście takie związane jest z faktem, że to właśnie początkowe etapy kaskady mają największy wpływ na wartość oczekiwaną, a zatem zmiany na początkowych etapach mogą przynieść największe korzyści. Po zakończeniu budowy drzewa, jedna ze ścieżek od korzenia do któregoś z liści będzie wskazywała najlepszą kaskadę, to znaczy taką, której wartość oczekiwana będzie najmniejsza. Warto zaznaczyć, że już dla nastaw $L = 1, C = 3$, stosowanych przez autora pracy jako domyślne, w połączeniu z zachłannymi poluzowanymi wymaganiami (3.9) możliwe było poprawienie wartości oczekiwanej dla kaskady.

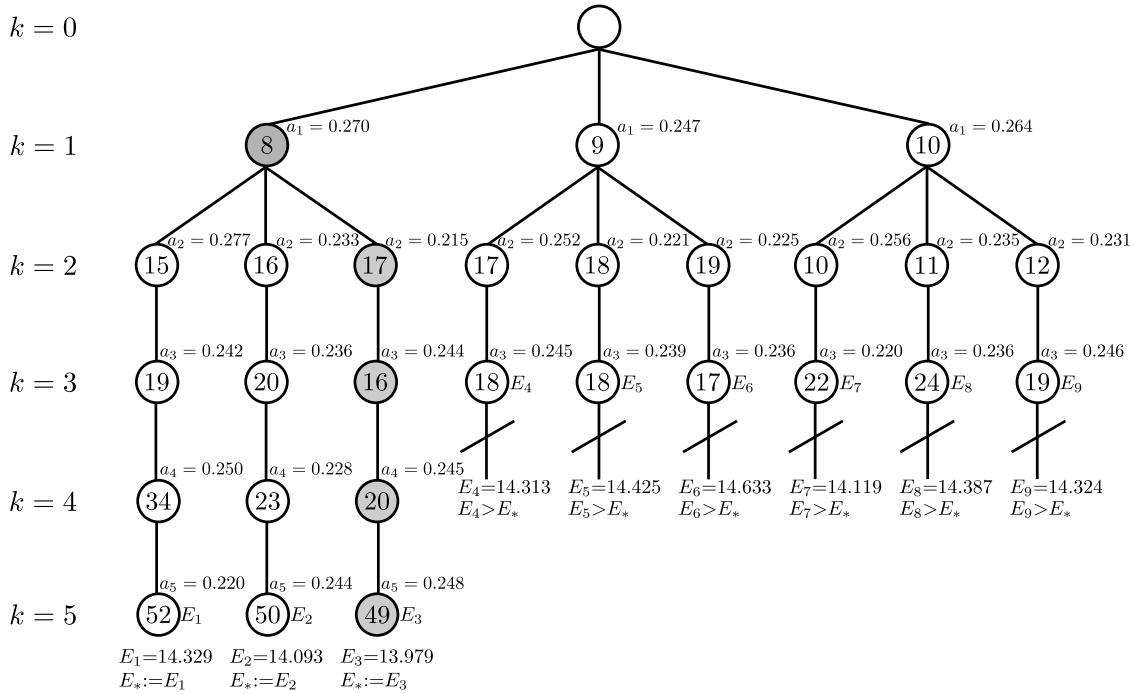
W celach notacyjnych potomkowie będący wariantami tego samego etapu będą posiadać dodatkowy podindeks. Na przykład klasyfikator $F_{1,0}$ oznacza główny wariant pierwszego etapu (wykorzystujący pewną liczbę cech) i jest on graficznie przedstawiany jako *środkowe* dziecko. Jego *lewe* rodzeństwo $F_{1,-1}, F_{1,-2}, \ldots$ oznacza klasyfikatory używające mniej cech (jedną mniej, dwie mniej, itd.). *Prawe* rodzeństwo $F_{1,+1}, F_{1,+2}, \ldots$ wykorzystuje więcej cech niż $F_{1,0}$ (jedną więcej, dwie więcej, itd.). Należy zaznaczyć, że oznaczenia te

[1] z drugiej strony, dodany klasyfikator nie może zostać usunięty

będą wykorzystywane jedynie lokalnie w ramach pojedynczego rekurencyjnego wywołania w związku z globalną niejednoznacznością takiego nazewnictwa.

3.4.1 Metoda podziału i ograniczeń (branch-and-bound)

Wadą opisanego rozwiązania jest znaczne wydłużenie czasu nauki, proporcjonalne do liczby ścieżek w drzewie, aby temu zaradzić został opracowany algorytm przycinania. Jego działanie możemy schematycznie zaobserwować na Rys. 3.2, a szczegółowy opis znajduje się poniżej.



Rysunek 3.2: Uczenie kaskady jako przeszukiwanie drzewa z przycinaniem — przykładowa ilustracja dla $L = 2$ i $C = 3$. (Źródło: opracowanie własne)

Wykorzystanie aktualnej cząstkowej wartości oczekiwanej — dokładna technika podziału i ograniczeń

W trakcie trwającego przeszukiwania drzewa połączonego z uczeniem kaskady, jesteśmy w stanie obserwować bieżące *częściowe* wartości dla optymalizowanej przez nas wartości oczekiwanej (wzór (1.30)). Załóżmy, że nowy $(k+1)$-wszy etap został właśnie ukończony, dając informację o n_{k+1} cechach przez niego wykorzystywanych. Poniższy wzór:

$$\begin{aligned}\widehat{E}\Big((n_1,\ldots,n_{k+1})\Big) &= \sum_{1\leqslant j\leqslant k} n_j \prod_{1\leqslant i<j} a_i + n_{k+1}\prod_{1\leqslant i<k+1} a_i \\ &= \widehat{E}\Big((n_1,\ldots,n_k)\Big) + n_{k+1}\prod_{1\leqslant i<k+1} a_i. \end{aligned} \tag{3.10}$$

wyraża częściową wartość oczekiwaną dla rozszerzonej kaskady w sposób przyrostowy, jako sumę poprzedniej częściowej oczekiwanej (dla k etapów) oraz nowego $(k+1)$-wszego

składnika. Skupiając się na metodzie podziału i ograniczeń staje się jasne, że jeśli z jakiegoś powodu częściowa oczekiwana jest większa od (lub równa) najlepszej odnalezionej do tej pory wartości oczekiwanej (pełnej — obejmującej wszystkie składniki), czyli $\widehat{E}((n_1, \ldots, n_{k+1})) \geqslant \widehat{E}^*$, to nie ma sensu dalej podążać tą ścieżka w dół drzewa. Innymi słowy, możemy ją przyciąć, ponieważ wzór (3.10) stanowi dolne ograniczenie dla ostatecznej nieznanej oczekiwanej. Zaprezentowany Rys. 3.2 stanowi symboliczną ilustrację przeszukiwania drzewa z przycinaniem. Intencją podanych na rysunku wielkości E_1, E_2, ... jest chronologiczne przedstawienie wartości oczekiwanych obserwowanych dla kolejnych odwiedzanych gałęzi (należy pamiętać, że drzewo jest przeszukiwane od lewej do prawej). Przekreślone linie reprezentują przycięte gałęzie, dla których kontynuacja nauki nie przyniosłaby poprawy wartości oczekiwanej.

Algorytm 8 Algorytm uczący kaskadę klasyfikatorów oparty na przeszukiwaniu drzewa z przycinaniem dokładnym.

Algorytm TrainTreeCascade($\mathcal{D}$, A, D, K, k, $\mathcal{V}$, F, C, L, F^*, $\widehat{E}^*$)
  Ustal podzbiór przykładów pozytywnych $\mathcal{P}$ i negatywnych $\mathcal{N}$ w ramach zbioru $\mathcal{D}$.
  Naucz etap dla środkowego dziecka: $F_{k+1,0}$:=TrainStage($\mathcal{P}$, $\mathcal{N}$, K, k, $\mathcal{V}$, F).
  Użyj kaskady $F\|F_{k+1,0}$ na zbiorze walidacyjnym $\mathcal{V}$, aby zmierzyć $A_{k+1,0}$ i $D_{k+1,0}$.
  Jeżeli $k > L$ **to**
    $C := 1$.
  Dla $c = -1, -2, \ldots, -\lfloor C/2 \rfloor$ **powtarzaj** ▷ lewe rodzeństwo
    Utwórz $F_{k+1,c}$ poprzez klonowanie $F_{k+1,c+1}$.
    Usuń ostatni słaby klasyfikator z $F_{k+1,c}$.
    Dostosuj próg decyzyjny $\theta_{k+1,c}$ dla $F_{k+1,c}$, aby spełnić wymaganie $d_{\min,k+1}$.
    Użyj kaskady $F\|F_{k+1,c}$ na zbiorze walidacyjnym $\mathcal{V}$, aby zmierzyć $A_{k+1,c}$ i $D_{k+1,c}$.
  Dla $c = 1, 2, \ldots, \lfloor C/2 \rfloor$ **powtarzaj** ▷ prawe rodzeństwo
    Utwórz $F_{k+1,c}$ poprzez klonowanie $F_{k+1,c-1}$.
    Naucz nowy słaby klasyfikator f wykorzystując $\mathcal{P}$ i $\mathcal{N}$.
    $F_{k+1,c} := F_{k+1,c} + f$.
    Dostosuj próg decyzyjny $\theta_{k+1,c}$ dla $F_{k+1,c}$, aby spełnić wymaganie $d_{\min,k+1}$.
    Użyj kaskady $F\|F_{k+1,c}$ na zbiorze walidacyjnym $\mathcal{V}$, aby zmierzyć $A_{k+1,c}$ i $D_{k+1,c}$.
  Dla $c = -\lfloor C/2 \rfloor, \ldots, 0, \ldots, \lfloor C/2 \rfloor$ **powtarzaj** ▷ wszyscy potomkowie
    Oblicz oczekiwaną $\widehat{E}$ dla kaskady $F\|F_{k+1,c}$ wykorzystując (3.10).
    Jeżeli $A_{k+1,c} > A$ and $\widehat{E} < \widehat{E}^*$ **to**
      Przygotuj nowy zbiór uczący $\mathcal{D}_{k+1,c}$ oraz nowy zbiór walidacyjny $\mathcal{V}_{k+1,c}$.
      (F^*, $\widehat{E}^*$):=TrainTreeCascade($\mathcal{D}_{k+1,c}$,A, D, K, $k+1$, $\mathcal{V}_{k+1,c}$,
        $F\|F_{k+1,c}$, C, L, F^*, E^*).
    W.p.r. Jeżeli $A_{k+1,c} \leqslant A$ oraz $\widehat{E} < \widehat{E}^*$ **to**
      $\widehat{E}^* := \widehat{E}$, $F^* := F\|F_{k+1,c}$.
      Zwróć (F^*, $\widehat{E}^*$).
  Zwróć (F^*, E^*).

Algorytm 8 przedstawia procedurę uczenia kaskady połączaną z techniką podziału

i ograniczeń służącą do przeszukiwania drzew. Algorytm ten został zdefiniowany rekurencyjnie, gdzie pojedyncze wywołanie rekurencyjne może zostać podsumowane w następujący sposób. Na wejściu przyjmowana jest częściowa kaskada F z k etapami. Następnie uczony jest główny wariant $F_{k+1,0}$ dla $(k+1)$-wszego etapu, określany jako *środkowe dziecko/potomek*. Następnie algorytm „rozgałęzia" etap poprzez utworzenie kopii środkowego potomka z mniejszą liczbą cech: $F_{k+1,-1}, F_{k+1,-2}, \ldots$ (*lewi potomkowie*), oraz kopii z dodatkowymi cechami: $F_{k+1,+1}, F_{k+1,+2}, \ldots$ (*prawi potomkowie*). Jak wspomniano wcześniej, drzewo jest rozgałęziane jedynie na L najwyższych poziomach a współczynnik rozgałęzienia wynosi C. Algorytm iteruje po wszystkich dzieciach i wykonuje rekurencyjne wywołania w celu nauczenia ich potomków, pod warunkiem że dolne ograniczenie (3.10) na pełną wartość oczekiwaną nie jest gorsze od najlepszej odnalezionej dotychczas oczekiwanej $\hat{E}^*$. Ścieżka rekurencyjna, reprezentująca pewną kaskadę, osiąga swój koniec w momencie, gdy końcowe wymagania (A, D) zostaną spełnione oraz jej oczekiwana jest ostro mniejsza od $\widehat{E}^*$ (początkowo, ustawione na ∞).

Początkowe wywołanie rekurencji ma postać:

$$\textsc{TrainTreeCascade}\left(\mathcal{D}, A, D, K, 0, \mathcal{V}, (), C, L, \text{null}, \infty\right),$$

i jako wynik zwraca najlepszą kaskadę F^* wraz z jej wartością oczekiwaną $\widehat{E}^*$.

Wykorzystanie aproksymacji wartości oczekiwanej — przybliżona technika podziału i ograniczeń

Patrząc wstecz na dolne ograniczenie (3.10), można zauważyć, że wartość nowego składnika wynika bezpośrednio i tak naprawdę wyłącznie z liczebności n_{k+1}. Jest tak, ponieważ wszystkie pozostałe parametry $a_1, \ldots, a_k$ biorące udział w tym składniku są znane jeszcze *przed* rozpoczęciem nauki $(k+1)$-ego etapu. Obserwacja ta jest użyteczna do tego, aby zachęcić algorytm do wykonania większej liczby odcięć podczas przeszukiwania, dzięki dokonaniu predykcji przyszłej częściowej wartości oczekiwanej.

Załóżmy, że zakończyliśmy naukę etapu $k+1$ i chcemy dokonać predykcji na temat częściowej wartości oczekiwanej dla etapu $k+2$, nie ucząc go. Nauka jakiegokolwiek etapu jest czasochłonna, stąd pojawia się możliwość znacznego zysku, jeśli nie zmarnujemy czasu na naukę etapu, który nie jest w stanie poprawić najlepszej osiągniętej do tej pory oczekiwanej. Warto zauważyć, że po zakończeniu nauki etapu $k+1$ dostajemy dwie nowe informacje opisujące kaskadę: n_{k+1} oraz a_{k+1}. Ta druga informacja nie jest potrzebna do obliczenia częściowej oczekiwanej (3.10) dla etapu $k+1$, ale jest potrzebna dla etapu $k+2$. W związku z tym jedyną niewiadomą nie pozwalającą nam na obliczenie dokładnej częściowej oczekiwanej dla etapu $k+2$ jest n_{k+2}. Jesteśmy jednak w stanie w prosty sposób aproksymować tę niewiadomą.

Badania kaskad na rzeczywistych danych pokazują, że liczności cech na kolejnych etapach $(n_k)_{k=1,\ldots,K}$ zazwyczaj tworzą ciąg niemalejący. Wprawdzie istnieją kontrprzykłady, jednak dla znaczącej większości przypadków jest prawdą, że $n_{k+1} \geqslant n_k$. W związku z tym teoretycznie wystarczyłoby ograniczyć z dołu n_{k+2} przez n_{k+1}. Jednakże, autor niniejszej pracy proponuje trochę bardziej bezpieczne sparametryzowane podejście — zakładając że:

$$n_{k+2} \geqslant \alpha\, n_{k+1}, \tag{3.11}$$

gdzie parametr α może zostać wybrany np. z przedziału $[0.5, 1.5]$. Poniższe wyprowadzenie demonstruje, na jakiej zasadzie będzie odbywała się omawiana predykcja częściowej wartości oczekiwanej:

$$\begin{aligned}\widehat{E}\Big((n_1,\ldots,n_{k+2})\Big) &= \widehat{E}\Big((n_1,\ldots,n_k)\Big) + n_{k+1}\prod_{1\leqslant i<k+1} a_i + n_{k+2}\prod_{1\leqslant i<k+2} a_i \\ &\approx \widehat{E}\Big((n_1,\ldots,n_k)\Big) + n_{k+1}\prod_{1\leqslant i<k+1} a_i + \alpha\, n_{k+1}\Big(\prod_{1\leqslant i<k+1} a_i\Big) a_{k+1} \\ &\equiv \widehat{E}_\alpha\Big((n_1,\ldots,n_{k+1},\alpha\, n_{k+1})\Big). \end{aligned} \tag{3.12}$$

W drugiej linii częstość a_{k+1} została celowo wyizolowana, aby pokazać, że iloczyn $\prod_{1\leqslant i<k+1} a_i$ dla aktualnego etapu może zostać ponownie wykorzystany przy predykcji oczekiwanej dla kolejnego etapu bez konieczności jego ponownego wyznaczania.

Przy doborze parametru α warto wziąć pod uwagę następujące fakty. Zmniejszając parametr α, zmniejszamy ryzyko błędnego przycięcia aktualnej gałęzi, jednocześnie zwiększamy szansę na niedoszacowanie wartości oczekiwanej, co może spowodować kontynuowanie nauki mimo znikomej szansy na poprawę oczekiwanej. Natomiast zwiększanie wartości α prowadzi do zwiększenia liczby przyciętych węzłów jednak niesie za sobą ryzyko pominięcia optymalnego rozwiązania. Należy także zaznaczyć, że zaproponowana tutaj predykcja jest przewidywaniem tylko na *jeden* etap w przód, ignorując wszystkie kolejne. Ponieważ składniki dla zignorowanych etapów także mają wpływ na ostateczną wartość oczekiwaną, sugeruje to, że wysokie wartości α powinny być także bezpiecznie, szczególnie dla początkowych etapów (tj. dla małych indeksów k).

Algorytm 9 prezentuje opisane powyżej podejście, to znaczy kaskadę uczoną w oparciu o przeszukiwanie drzewa z przycinaniem przybliżonym.

Algorytm 9 Algorytm uczący kaskadę klasyfikatorów oparty na przeszukiwaniu z przybliżonym algorytmem przycinania.

Algorytm TrainTreeCascadeApprox($\mathcal{D}$, A, D, K, k, $\mathcal{V}$, F, C, L, F^*, $\widehat{E}^*$, α)
⋮ ▷ początkowe kroki identyczne jak w TrainTreeCascade
Dla $c = -\lfloor C/2\rfloor, \ldots, 0, \ldots, \lfloor C/2\rfloor$ **powtarzaj** ▷ wszyscy potomkowie
Jeżeli $A_{k+1,c} > A$ **to**
Oblicz predykcję $\widehat{E}_\alpha$ dla kaskad $F\|F_{k+1,c}\|F_{k+2,\cdot}$ wykorzystując (3.12).
Jeżeli $\widehat{E}_\alpha < \widehat{E}^*$ **to**
Przygotuj nowy zbiór uczący $\mathcal{D}_{k+1,c}$ oraz nowy zbiór walidacyjny $\mathcal{V}_{k+1,c}$.
$(F^*, \widehat{E}^*)$:=TrainTreeCascadeApprox($\mathcal{D}_{k+1,c}$, A, D, K, $k{+}1$, $\mathcal{V}_{k+1,c}$, $F\|F_{k+1,c}$, C, L, F^*, E^*, α).
W.p.r.
Oblicz oczekiwaną $\widehat{E}$ dla kaskady $F\|F_{k+1,c}$ wykorzystując (3.10).
Jeżeli $\widehat{E} < \widehat{E}^*$ **to**
$\widehat{E}^* := \widehat{E}, \quad F^* := F\|F_{k+1,c}$.
Zwróć $(F^*, \widehat{E}^*)$.
Zwróć (F^*, E^*).

Rozdział 4

Badania

Przedstawione we wcześniejszych rozdziałach rozwiązania zostały przetestowane dla trzech zestawów cech: cech Haara w zadaniu detekcji twarzy, cech HOG w tym samym zadaniu oraz momentów Zernike'a w zadaniu detekcji litery „A". Podobnie jak w przypadku pozostałych wspomnianych rodzajów cech momenty Zernike'a także zostały wsparte obrazami całkowymi przygotowanymi przed procedurą detekcyjną (Bera, Klęsk i Sychel, 2018), co zagwarantowało stały czas ekstrakcji dla każdej z cech. Dodatkowo należy zaznaczyć, że w przypadku ostatniego eksperymentu obiekty miały być rozpoznawane niezależnie od ich obrotu.

W każdym ze wspomnianych badań posłużono się algorytmem RealBoost jako algorytmem boostingowym na rzecz uczenia kaskady. Rolę słabych klasyfikatorów w algorytmie RealBoost pełniły kosze z odpowiedzią rzeczywistoliczbową (Rasolzadeh i in., 2006). Każdy ze słabych klasyfikatorów opiera się więc na dokładnie jednej cesze.

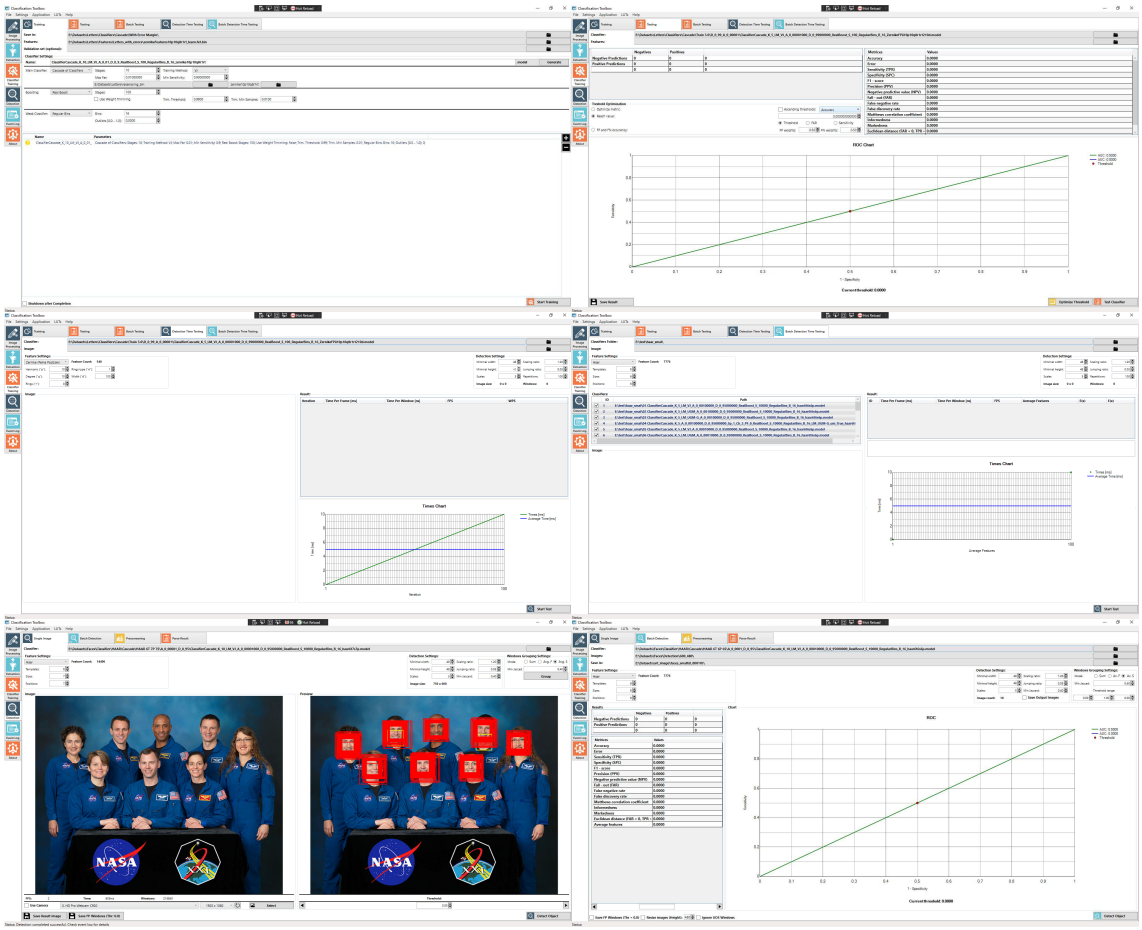
Wszystkie z przedstawionych poniżej kaskad zostały nauczone na maszynie wyposażonej w procesor Intel Xeon E5-2699 v4 CPU, 22/44 rdzenie/wątki. W związku z wysokim kosztem takich procesorów, który ogranicza ich praktyczne zastosowanie, pomiary czasów detekcji nauczonych klasyfikatorów dokonano na procesorze AMD Ryzen 7 3800X 8/8 rdzeni/wątków. Dodatkowo, aby uprościć interpretację wyników tabele przedstawiają jedynie wyniki uzyskane bez zastosowania zrównoleglenia, co oznaczono symbolem ST (*single thread*). Wyniki dla wielu wątków zostały natomiast przedstawione na wykresach umieszczonych poniżej tabel.

Przedstawione pomiary czasów nie zawierają czasu potrzebnego do przygotowania obrazów całkowych, ponieważ nie ma on związku z wyznaczaniem odpowiedzi kaskady ani z jej wartością oczekiwaną liczby cech.

Jeżeli chodzi o czas uczenia wymienionych poniżej kaskad wahał się on od 1 h dla kaskad spełniających FAR $A = 10^{-3}$ do nawet 3 miesięcy dla $A = 10^{-6}$.

Wykorzystane oprogramowanie zostało utworzone w języku C#, przy czym kluczowe dla obliczeń funkcje (np. wyznaczenie obrazów całkowych, ekstrakcja cech) zostały zaimplementowane w języku C++ jako biblioteki dll. Powstała na potrzeby pracy aplikacja pozwalała w wygodny sposób, za pomocą przygotowanego graficznego interfejsu użytkownika (Rys. 4.1), na m.in. uczenie oraz wczytywanie różnego rodzaju modeli, detekcję na pojedynczych obrazach oraz sekwencjach wideo (dostęp do kamery uzyskano dzięki zasto-

sowaniu biblioteki OpenCV), testowanie oraz pomiary czasów pojedynczego modelu lub też całej ich serii.



Rysunek 4.1: Przykładowe zrzuty ekranu przedstawiające aplikację utworzoną na potrzeby pracy. (Źródło: opracowanie własne)

4.1 Wykrywanie twarzy

Z dwóch wspomnianych zadań detekcja twarzy była tym trudniejszym pod względem dokładności. Jej trudność wynikała z dużej różnorodności w klasie przykładów pozytywnych: wyrazy twarzy, fryzury, kolory skóry, wiek, okulary, itp. W związku z tym zadanie to wymagało znacznie większej liczby cech niż w przypadku detekcji litery „A”.

Wykorzystane do nauki twarze zostały wycięte z 3 000 obrazów wyszukanych za pomocą silnika *Google Images*. Zbiór uczący składał się z 7 258 przykładowych twarzy, natomiast zbiór walidacyjny z 1 000. Zbiór testowy, na którym badana była skuteczność uzyskanych klasyfikatorów, zawierał 3 014 twarzy pochodzących ze zbioru danych uniwersytetu Essex (Ahmad, Najam i Ahmed, 2013; University of Essex, 1997). Zbiór negatywów dla zbioru testowego był stały i składał się z 1 000 000 przykładów. W celu redukcji czasu nauki liczba przykładów negatywnych w zbiorze uczącym oraz walidacyjnym była stopniowo zmniejszana wraz z kolejnymi etapami zgodnie z Tabelą 4.1. Czasy detekcji poszczególnych kaskad, zapisane poniżej, zostały określone na podstawie średniej z 1 000

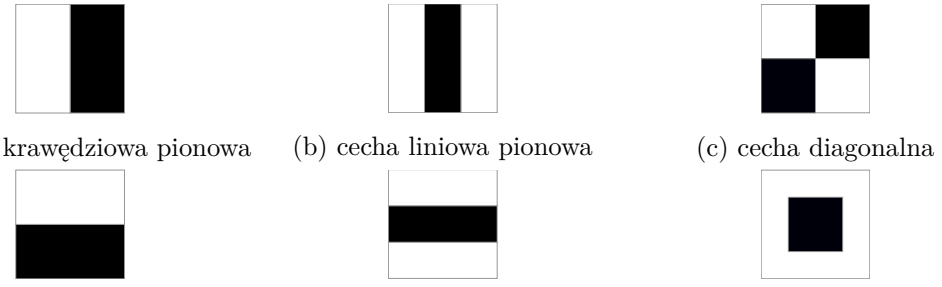
uruchomień procedury detekcyjnej na obrazie testowym.

Tablica 4.1: „Detekcja twarzy”: nastawy eksperymentu.

dane uczące		
parametr	wartość	dodatkowe informacje
liczba przykładów pozytywnych	7 258	okna z twarzą różne skale, kolory skóry, okulary, itp.
liczba przykładów negatywnych	139 373/42 742/27 742	na 1/2/kolejnych etapach
rozmiar zbioru uczącego	146 631/50 000/35 000	przykładów łącznie na 1/2/kolejnych etapach
dane walidacyjne		
liczba przykładów pozytywnych	1 000	okna z twarzą różne skale, kolory skóry, okulary, itp.
liczba przykładów negatywnych	40 000/ ≈ 24 000	na 1/kolejnych etapach
rozmiar zbioru walidacyjnego	41 000/25 000	przykładów łącznie na 1/kolejnych etapach
dane testowe		
liczba przykładów pozytywnych	3 014	okna z twarzą (University of Essex, 1997) różne skale, kolory skóry, okulary
liczba przykładów negatywnych	1 000 000	
rozmiar zbioru testowego	1 003 014	przykładów łącznie
procedura detekcyjna (skanowanie oknem przesuwnym)		
liczba powtórzeń	1000	na potrzeby uśrednienia pomiarów czasu
rozdzielczość obrazu	600 × 480	
liczba skal detekcyjnych	5	obraz przeskanowany z użyciem 5 różnych rozmiarów okien
współczynnik przyrostu okna	1.2	wysokość i szerokość okna wzrasta o ≈ 20% na każdą skalę
rozmiar najmniejszego okna	48 × 48	
rozmiar największego okna	100 × 100	
współczynnik przesunięcia okna	0.05	okno przesuwa się o ≈ 5% jego rozmiaru

4.1.1 Cechy Harra

Pierwszym zestawem cech wykorzystanym przez autora pracy do detekcji twarzy na obrazie były cechy Haara wsparte obrazami całkowymi (Crow, 1984; Lewis, 1995; Papageorgiou, Oren i Poggio, 1998; Viola i Jones, 2001). Przestrzeń ucząca składała się z 14 406 cech Haara, wygenerowanych dla 6 szablonów falki (widocznych na Rys. 4.2) w 7^2 skalach przy 7^2 punktach zakotwiczenia, podobnie jak miało to miejsce w pracach (Sychel, Klęsk i Bera, 2020a,b). Warto zaznaczyć, że od czasu pierwszej ze wspomnianych publikacji (Sychel, Klęsk i Bera, 2020a) drobnej zmianie uległ algorytm resamplingu, stąd różnice między przedstawionymi tam klasyfikatorami a poniższą pracą.



(a) cecha krawędziowa pionowa (b) cecha liniowa pionowa (c) cecha diagonalna

(d) cecha krawędziowa pozioma (e) cecha liniowa pozioma (f) cecha centralna

Rysunek 4.2: Szablony falek dla cech Haara. (Źródło: opracowanie własne)

Rysunek 4.3: „Detekcja twarzy”: przykłady detekcji. Żółte ramki oznaczają fałszywe wykrycia (cechy Haara — 14406). (Źródło: opracowanie własne)

Rys. 4.3 prezentuje przykładowe wyniki detekcji dla dwóch wybranych klasyfikatorów (pierwszy dla $A = 10^{-5}$, a drugi dla $A = 10^{-6}$) o najkorzystniejszych (najmniejszych) wartościach oczekiwanych liczby ekstrahowanych cech. Czerwone ramki oznaczają prawidłowo wykryte twarze, żółte natomiast to fałszywe alarmy. Jak możemy zauważyć mimo redukcji wartości oczekiwanej klasyfikatory działają poprawnie.

Przed przejściem do szczegółowego omówienia wyników liczbowych (dokładności i pomiarów czasowych) poniżej pokazano wizualne efekty (Rysunki 4.4, 4.5) działania najlepszej otrzymanej kaskady w zadaniu detekcji dla nastaw $K = 10$, $A = 10^{-6}$, $D = 0.95$.

Rysunek 4.4: „Detekcja twarzy”: przykłady detekcji, dla najlepszej otrzymanej kaskady dla nastaw $K = 10$, $A = 10^{-6}$ (cechy Haara — 14406). (Źródło: opracowanie własne)

Rysunek 4.5: „Detekcja twarzy”: przykłady detekcji, dla najlepszej otrzymanej kaskady dla nastaw $K = 10$, $A = 10^{-6}$ (cechy Haara — 14406). (Źródło: opracowanie własne)

Pomimo, że do zastosowań przemysłowych potrzebne są klasyfikatory posiadające bardzo niski FAR (np. rzędu 10^{-6}), w celu wykonania jak największej liczby eksperymentów porównujących wartości oczekiwane (i z uwagi na czasochłonność uczenia kaskady) autor przeprowadzał uczenie także dla większych wartości FAR: 10^{-3}, 10^{-4}, 10^{-5}.

Należy zaznaczyć, że w przypadku wszystkich tabel raportujących wyniki kaskad, każda z kaskad jest podana jako ciąg liczebności cech na poszczególnych etapach (n_k)

wraz z podanymi poniżej częstościami fałszywych alarmów (a_k). Dodatkowo w pierwszym wierszu podana jest kaskada w stylu VJ stanowiąca referencję dla pozostałych.

Tabela 4.2 przedstawia wyniki dla kaskad pięcioetapowych. Jak możemy zaobserwować zastosowanie zaproponowanych metod pozwoliło na redukcję wartości oczekiwanej, co przełożyło się na zmniejszenie czasu klasyfikacji. Jednym wyjątkiem była metoda UGM, która przy narzuconym wymaganiu $A = 10^{-3}$ uzyskała gorszą kaskadę od tej otrzymanej dla klasycznego podejścia. Przypadek ten stanowi jednak wyjątek i dla innych wartości A oraz K ta sama metoda wykazywała zyski. Najlepszą z testowanych metod okazała się metoda oparta o drzewo przeszukiwań (TREE-C3-L1-UGM-G), pozwalając na redukcję czasu nawet o 9% (przy zastosowaniu tylko jednego wątku). Warto także zauważyć stosunkowo nieduże różnice wskaźników dokładności badanych na zbiorze testowym. Wartości czułości często były wyższe dla proponowanych podejść, a w najgorszym wypadku spadek czułości nie przekroczył 1%. Natomiast wartość FAR dla $A = 10^{-3}$ pogorszyła się maksymalnie o 253 fałszywe wykrycia na 1 000 000 negatywów, a w przypadku $A = 10^{-4}$ różnica między opisanymi metodami a klasycznym podejściem nie przekroczyła 34 fałszywych wykryć.

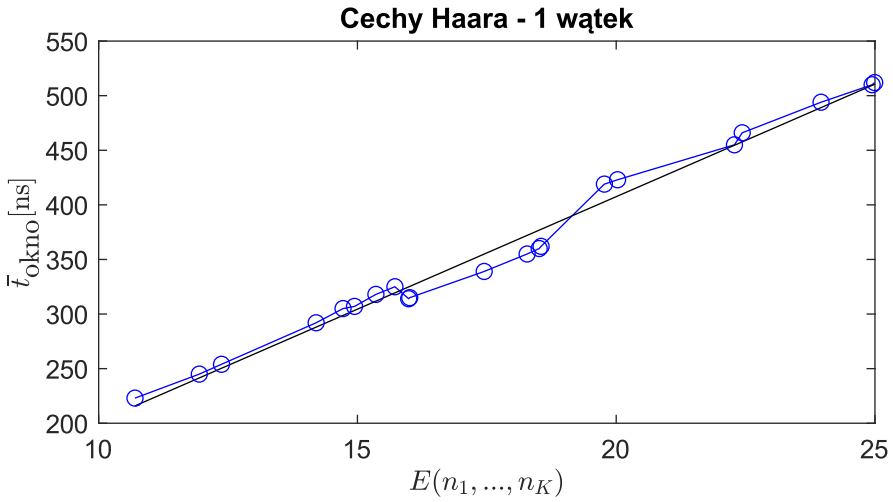
Tablica 4.2: „Detekcja twarzy” — kaskady z $K = 5$ etapami dla różnych metod uczenia (cechy Haara — 14406).

Algorytm uczący	Kaskada	Wartość oczekiwana	Walidacja		Test			Czas detekcji		
			FAR	czułość	FAR	czułość	$\bar{n}$	obraz [ST][ms]	okno [ST][ns]	Δ
			Wymagania: $\mathbf{10^{-3}} =$ **0.001000**	**0.9500**				(okna na obraz: 130 917)		
VJ	9 18 26 30 38 0.2468, 0.2214, 0.2299, 0.2468, 0.2370	15.3571	0.000735	0.9520	0.000711	0.9572	16.21	41.6	318	+0.0%
UGM	9 17 32 29 29 0.2468, 0.2516, 0.2450, 0.2303, 0.2798	15.7243	0.000980	0.9510	0.000980	0.9555	16.58	42.4	325	+2.2%
UGM-G	9 16 21 22 39 0.2468, 0.2548, 0.2234, 0.2635, 0.2606	14.7220	0.000964	0.9510	0.000966	0.9575	15.50	39.9	305	−4.1%
TREE-C3-L1 UGM-G	8 16 22 25 40 0.2703, 0.2329, 0.2188, 0.2713, 0.2646	14.2021	0.000988	0.9510	0.000847	0.9615	14.91	38.2	292	−8.2%
			Wymagania: $\mathbf{10^{-4}} =$ **0.000100**	**0.9500**				(okna na obraz: 130 917)		
VJ	18 38 40 67 82 0.1516, 0.1582, 0.1452, 0.1533, 0.1509	24.9947	0.000081	0.9520	0.000051	0.9456	26.85	67.0	512	+0.0%
UGM	18 38 40 52 81 0.1516, 0.1582, 0.1452, 0.1656, 0.1694	24.9455	0.000098	0.9510	0.000079	0.9466	26.80	66.7	510	−0.4%
UGM-G	18 32 33 60 100 0.1516, 0.1647, 0.1487, 0.1685, 0.1843	23.9587	0.000115	0.9510	0.000081	0.9509	25.50	64.6	494	−3.5%
TREE-C3-L1 UGM-G	17 27 28 80 61 0.1654, 0.1344, 0.1746, 0.1563, 0.1631	22.4353	0.000099	0.9510	0.000083	0.9366	24.04	61.0	466	−9.0%

W przypadku Tabeli 4.3 zostały zestawione ze sobą większe dziesięcioetapowe kaskady. Ponownie zaproponowane rozwiązania pozwoliły na uzyskanie krótszego czasu klasyfikacji bez znacznej utraty jakości klasyfikatora. W większości przypadków najskuteczniejszą okazała się metoda oparta o drzewo przeszukiwań (TREE-C3-L1-UGM-G). Przegrała ona jedynie z algorytmem UGM w ostatnim eksperymencie ($A = 10^{-6}$). Przy czym zastosowanie dokładniejszego przeszukiwania (tj. wyższych nastaw dla parametrów C i L) powinno i w tym przypadku zmienić zaistniałą sytuację na korzyść drzew. Także i tym razem różnice w miarach dokładności na zbiorze testowym nie odbiegały znacznie od siebie pomiędzy klasyfikatorami w poszczególnych seriach eksperymentów. Podobnie jak poprzednio, analizując wyniki dla zbioru testowego możemy zauważyć, że spadek wartości oczekiwanej niekoniecznie oznacza gorszy klasyfikator. W wielu przypadkach klasyfikator o wyższej wartości FAR, nadrabiał wyższą czułość na zbiorze testowym.

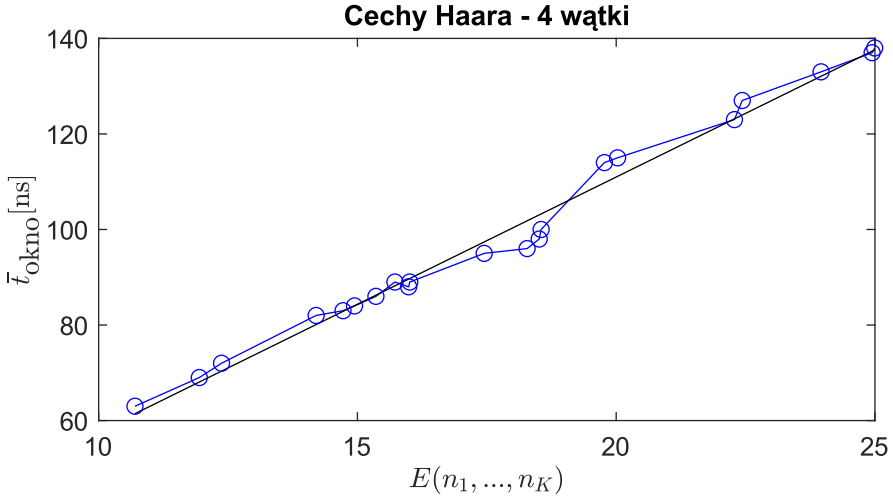
Tablica 4.3: „Detekcja twarzy” — kaskady z $K = 10$ etapami dla różnych metod uczenia (cechy Haara — 14406).

Algorytm uczący	Kaskada	Wartość oczekiwana	Walidacja FAR	Walidacja czułość	Test FAR	Test czułość	Test $\bar{n}$	Czas detekcji obraz [ST][ms]	Czas detekcji okno [ST][ns]	Δ
			Wymagania: $\mathbf{10^{-3}} =$ **0.001000**	**0.9500**				(okna na obraz: 130 917)		
VJ	4 6 16 13 11 17 14 21 24 45 0.4521, 0.4491, 0.4849, 0.4204, 0.4966, 0.4722, 0.4522, 0.4933, 0.4990, 0.4679	12.3741	0.000505	0.9560	0.000512	0.9552	13.91	33.3	254	+0.0%
UGM	4 6 14 10 13 10 13 18 29 17 0.4521, 0.4491, 0.5040, 0.5093, 0.5151, 0.5061, 0.4726, 0.5298, 0.5161, 0.5586	11.9447	0.000980	0.9500	0.001009	0.9575	13.34	32.1	245	−3.5%
UGM-G	4 6 5 14 8 12 17 16 14 34 0.4521, 0.4435, 0.5892, 0.5058, 0.4905, 0.5225, 0.5042, 0.4746, 0.5431, 0.4516	10.7018	0.000899	0.9510	0.000902	0.9522	11.51	29.2	223	−12.2%
TREE-C3-L1 UGM-G	4 6 5 14 8 12 17 16 14 34 0.4521, 0.4435, 0.5892, 0.5058, 0.4905, 0.5225, 0.5042, 0.4746, 0.5431, 0.4516	10.7018	0.000899	0.9510	0.000902	0.9522	11.51	29.2	223	−12.2%
			Wymagania: $\mathbf{10^{-4}} =$ **0.000100**	**0.9500**				(okna na obraz: 130 917)		
VJ	6 13 24 16 34 43 38 46 33 41 0.3546, 0.3914, 0.3939, 0.3945, 0.3256, 0.3615, 0.3907, 0.3977, 0.3865, 0.3940	16.0108	0.000060	0.9550	0.000057	0.9492	17.38	41.2	315	+0.0%
UGM	6 13 24 16 34 40 34 32 42 47 0.3546, 0.3914, 0.3939, 0.3945, 0.3256, 0.4000, 0.3926, 0.4397, 0.4533, 0.3927	15.9898	0.000086	0.9510	0.000076	0.9542	17.35	41.1	314	−0.3%
UGM-G	6 12 15 21 34 19 30 47 29 40 0.3546, 0.4135, 0.4104, 0.4010, 0.4093, 0.3991, 0.3940, 0.3970, 0.4057, 0.3941	14.9437	0.000099	0.9510	0.000105	0.9545	16.26	40.2	307	−2.5%
TREE-C3-L1 UGM-G	6 12 15 21 34 19 30 47 29 40 0.3546, 0.4135, 0.4104, 0.4010, 0.4093, 0.3991, 0.3940, 0.3970, 0.4057, 0.3941	14.9437	0.000099	0.9510	0.000105	0.9545	16.26	40.2	307	−2.5%
			Wymagania: $\mathbf{10^{-5}} =$ **0.000010**	**0.9500**				(okna na obraz: 130 917)		
VJ	9 20 32 40 37 79 61 95 192 132 0.2768, 0.2924, 0.2911, 0.2904, 0.2996, 0.2995, 0.2909, 0.2851, 0.3066, 0.3137	18.5492	0.000005	0.9550	0.000005	0.9393	20.35	47.4	362	+0.0%
UGM	9 20 32 38 41 44 64 78 63 81 0.2768, 0.2924, 0.2911, 0.3241, 0.3031, 0.3339, 0.3311, 0.3197, 0.3254, 0.2975	18.5122	0.000008	0.9510	0.000016	0.9403	20.12	47.1	360	−0.6%
UGM-G	9 19 25 35 29 46 56 46 60 83 0.2768, 0.3529, 0.3200, 0.3005, 0.3202, 0.3053, 0.3032, 0.3581, 0.3103, 0.2965	18.2789	0.000009	0.9510	0.000007	0.9382	19.52	46.5	355	−1.9%
TREE-C3-L1 UGM-G	8 18 16 50 38 40 62 70 105 75 0.3233, 0.2893, 0.3310, 0.3168, 0.3053, 0.3339, 0.2964, 0.3001, 0.2946, 0.3773	17.4493	0.000010	0.9510	0.000007	0.9409	18.91	44.4	339	−6.4%
			Wymagania: $\mathbf{10^{-6}} =$ **0.000001**	**0.9500**				(okna na obraz: 130 917)		
VJ	14 22 48 50 61 108 156 152 197 75 0.2178, 0.2417, 0.2462, 0.2451, 0.2509, 0.2501, 0.2433, 0.2502, 0.2478, 0.2499	22.2850	0.000001	0.9550	0.000001	0.9307	25.35	58.3	455	+0.0%
UGM	14 14 35 33 84 104 111 134 104 64 0.2178, 0.2513, 0.2437, 0.2339, 0.2375, 0.2554, 0.2682, 0.2434, 0.2811, 0.2804	19.7720	0.000001	0.9510	0.000002	0.9320	22.48	54.9	419	−7.9%
UGM-G	14 14 33 50 70 86 118 135 135 110 0.2178, 0.2513, 0.2837, 0.2552, 0.2346, 0.2576, 0.2611, 0.2366, 0.2547, 0.2613	20.0267	0.000001	0.9510	0.000004	0.9270	22.81	55.3	423	−7.0%
TREE-C3-L1 UGM-G	14 14 33 50 70 86 118 135 135 110 0.2178, 0.2513, 0.2837, 0.2552, 0.2346, 0.2576, 0.2611, 0.2366, 0.2547, 0.2613	20.0267	0.000001	0.9510	0.000004	0.9270	22.81	55.3	423	−7.0%



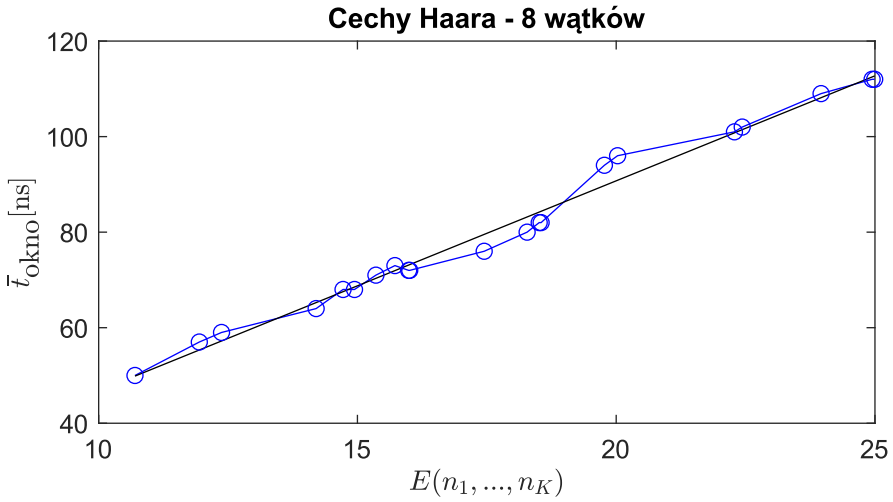
Rysunek 4.6: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Haara (14 406 cech) — 1 wątek. (Źródło: opracowanie własne)

Rys. 4.6 przedstawia zależność pomiędzy wartością oczekiwaną liczby cech a średnim czasem potrzebnym na przeanalizowanie pojedynczego okna (bez zrównoleglenia) dla każdego z przedstawionych powyżej klasyfikatorów. Jak możemy zauważyć dla cech Haara, istnieje liniowa zależność pomiędzy wartością oczekiwaną a czasem klasyfikacji. Delikatne odstępstwa od prostej wynikają z niewielkich rozbieżności między średnią liczbą cech dla wykorzystanego obrazu testowego a wartością oczekiwaną klasyfikatora. W przypadku jednego wątku średni czas na okno wahał się między 223 ns a 512 ns.



Rysunek 4.7: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Haara (14 406 cech) — 4 wątki. (Źródło: opracowanie własne)

Zwiększenie liczby wątków do czterech nie wpłynęło na kształt zależności, pozostała ona liniowa (Rys. 4.7). Natomiast odnotowano znaczny spadek średniego czasu klasyfikacji okna. Średnie czasy spadły ponad trzykrotnie i wynosiły od 63 ns do 138 ns.



Rysunek 4.8: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Haara (14 406 cech) — 8 wątków. (Źródło: opracowanie własne)

Dalsze zwiększenie liczby wątków do 8 (Rys. 4.8) spowodowało spadek czasów potrzebnych na analizę pojedynczego okna do przedziału $[50; 112]$ ns. Natomiast sama zależność pozostała liniowa.

Tablica 4.4: „Detekcja twarzy” — FPS dla różnej liczby wątków ($K = 10$, $A = 10^{-5}$, HAAR).

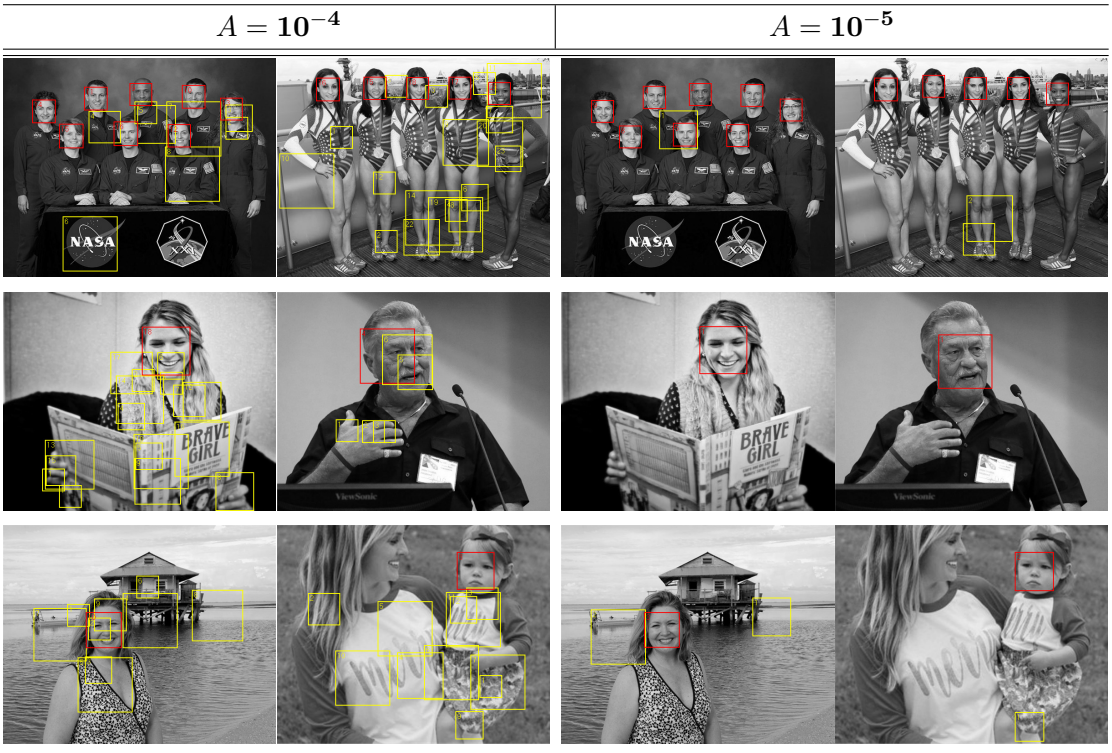
Algorytm uczący	1-wątek			4-wątki			8-wątków		
	obraz [ms]	FPS	Δ	obraz [ms]	FPS	Δ	obraz [ms]	FPS	Δ
VJ	47.4	21.1		13.1	76.3		10.7	93.5	
UGM	47.1	21.2	+0.1	12.8	78.1	+1.8	10.7	93.5	+0.0
UGM-G	46.5	21.5	+0.4	12.6	79.4	+3.1	10.5	95.2	+1.7
TREE-C3-L1-UGM-G	44.4	22.5	+1.4	12.4	80.6	+4.3	9.9	101.0	+7.5

Tabela 4.4 pokazuje zysk mierzony w klatkach na sekundę (FPS), uzyskany dzięki zastosowaniu nowych algorytmów uczenia kaskad dla przykładowego zestawu kaskad dla $K = 10$, $A = 10^{-5}$. O ile w przypadku jednego wątku zysk ten był stosunkowy nieduży i wyniósł maksymalnie 1 FPS (klasyfikator był wstanie przetworzyć około 180 tys. okien na sekundę więcej), to już w przypadku czterech wątków udało się przetworzyć cztery klatki na sekundę więcej, a dla ośmiu wątków wartość ta wyniosła 7 FPS (około miliona okien na sekundę więcej).

4.1.2 Cechy Haara — zbiór pomniejszony

Kolejny etap badań miał na celu sprawdzenie czy przy zbiorze o zmniejszonej liczbie cech utworzone rozwiązania nadal będą miały korzystny wpływ na wartość oczekiwaną. Tym razem wykorzystano zbiór zawierający 7 776 cech Haara, wygenerowanych dla 6 szablonów falki w 6^2 skalach przy 6^2 punktach zakotwiczenia. W przypadku tego eksperymentu pominięto kaskady dla wartości FAR na poziomie $A = 10^{-6}$, w związku z długim

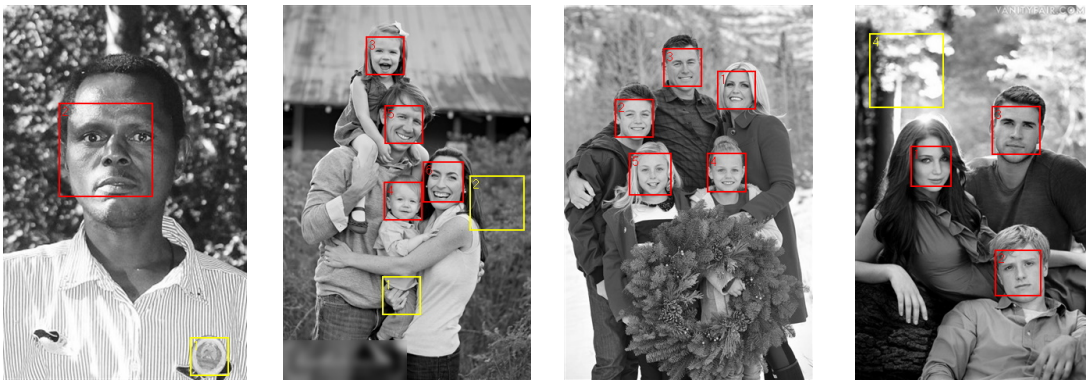
czasem ich nauki.



Rysunek 4.9: „Detekcja twarzy”: przykłady detekcji. Żółte ramki oznaczają fałszywe wykrycia (cechy Haara — 7776). (Źródło: opracowanie własne)

Podobnie jak w poprzednim badaniu na Rys. 4.9 możemy znaleźć przykładowe wyniki detekcji dla dwóch klasyfikatorów (pierwszy dla $A = 10^{-4}$, a drugi dla $A = 10^{-5}$) o najkorzystniejszych (najmniejszych) wartościach oczekiwanych. Czerwone ramki oznaczają prawidłowo wykryte twarze, żółte fałszywe alarmy.

Poniżej pokazano wizualne efekty (Rysunki 4.10, 4.11) działania najlepszej otrzymanej kaskady w zadaniu detekcji dla nastaw $K = 10$, $A = 10^{-5}$, $D = 0.95$.



Rysunek 4.10: „Detekcja twarzy”: przykłady detekcji, dla najlepszej otrzymanej kaskady dla nastaw $K = 10$, $A = 10^{-5}$ (cechy Haara — 7776). (Źródło: opracowanie własne)

Rysunek 4.11: „Detekcja twarzy”: przykłady detekcji, dla najlepszej otrzymanej kaskady dla nastaw $K = 10$, $A = 10^{-5}$ (cechy Haara — 7776). (Źródło: opracowanie własne)

W przypadku Tabeli 4.5 ponownie mamy do czynienia z kaskadami pięcioetapowymi. W przypadku tych eksperymentów najskuteczniejszą metodą okazała się ta bazująca na przeszukiwaniu drzewa ex aequo z metodą UGM-G (dla domyślnych nastaw tej pierwszej TREE-C3-L1-UGM-G). Wartości wskaźników jakości klasyfikacji tym razem także były do siebie zbliżone, przy nawet 7.4% poprawy w czasie detekcji dla bardziej złożonej serii kaskad. Natomiast dla najprostszych badanych kaskad redukcja wartości oczekiwanej była niewielka i przyniosła zmniejszenie czasu detekcji w wysokości 0.5%.

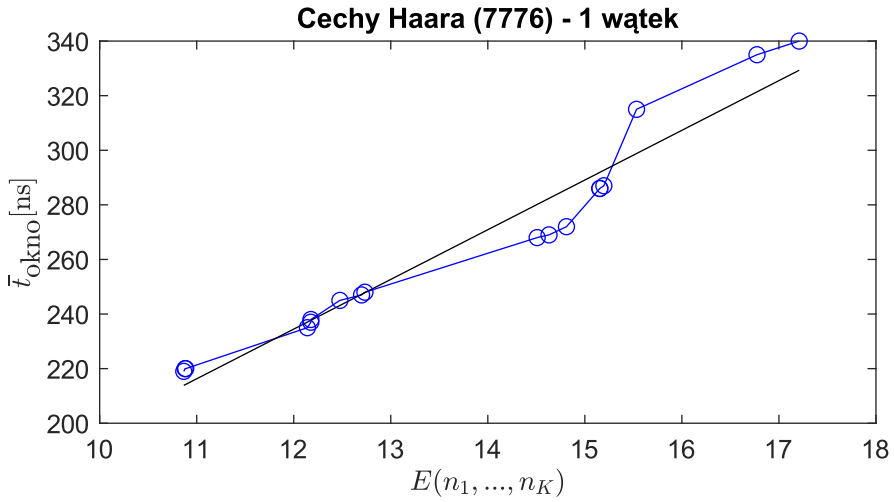
Tablica 4.5: „Detekcja twarzy" — kaskady z $K = 5$ etapami dla różnych metod uczenia (cechy Haara — 7776).

Algorytm uczący	Kaskada	Wartość oczekiwana	Walidacja FAR	Walidacja czułość	Test FAR	Test czułość	Test $\bar{n}$	Czas detekcji obraz [ST][ms]	Czas detekcji okno [ST][ns]	Δ
			Wymagania: $\mathbf{10^{-3}}$ = **0.001000**	**0.9500**				(okna na obraz: 130 917)		
VJ	6 13 21 31 34 0.2452, 0.2271, 0.2420, 0.2474, 0.2253	10.8868	0.000751	0.9520	0.000873	0.9320	11.78	28.8	220	+0.0%
UGM	6 13 21 31 32 0.2452, 0.2271, 0.2420, 0.2471, 0.2903	10.8800	0.000967	0.9510	0.001159	0.9326	11.77	28.8	220	+0.0%
UGM-G	6 13 20 28 40 0.2452, 0.2271, 0.2639, 0.2637, 0.2551	10.8667	0.000988	0.9510	0.001057	0.9399	11.72	28.7	219	−0.5%
TREE-C3-L1 UGM-G	6 13 20 28 40 0.2452, 0.2271, 0.2639, 0.2637, 0.2551	10.8667	0.000988	0.9510	0.001057	0.9399	11.72	28.7	219	−0.5%
			Wymagania: $\mathbf{10^{-4}}$ = **0.000100**	**0.9500**				(okna na obraz: 130 917)		
VJ	12 33 54 69 126 0.1200, 0.1566, 0.1438, 0.1425, 0.1427	17.2091	0.000055	0.9520	0.000085	0.9443	18.76	44.5	340	+0.0%
UGM	12 31 42 53 90 0.1200, 0.1656, 0.1615, 0.1752, 0.1737	16.7741	0.000098	0.9510	0.000119	0.9277	18.14	43.9	335	−1.5%
UGM-G	12 21 31 57 91 0.1200, 0.1980, 0.1631, 0.1583, 0.1452	15.5328	0.000089	0.9510	0.000115	0.9370	16.59	41.2	315	−7.4%
TREE-C3-L1 UGM-G	12 21 31 57 91 0.1200, 0.1980, 0.1631, 0.1583, 0.1452	15.5328	0.000089	0.9510	0.000115	0.9370	16.59	41.2	315	−7.4%

Kolejna tabela (Tabela 4.6) przedstawia kaskady dziesięcioetapowe. Najlepszą metodą okazała się ta bazująca na przeszukiwaniu drzew, przy czym dla kaskady $A = 10^{-4}$ konieczne było zbudowanie większego drzewa ($L = 3,\ C = 7$). W przypadku pozostałych dwóch serii kaskad ($A = 10^{-3}$ oraz $A = 10^{-5}$) metoda ta okazała się skuteczna już dla domyślnie stosowanych parametrów, wyprzedzając metodę UGM-G. Warto zaznaczyć, że eksperyment ten (a konkretnie wariant dla $A = 10^{-4}$) jest jedynym w całej pracy, gdzie kaskada w stylu VJ była w stanie pokonać zarówno kaskady uczone przy pomocy poluzowanych wymagań UGM oraz UGM-G, a dla drzewa konieczne było zastosowanie dokładniejszego przeszukiwania. Za przyczynę takiego stanu rzeczy można domniemywać zmniejszoną przestrzeń cech Haara. Jeśli chodzi o czas detekcji w przypadku tych kaskad udało się uzyskać poprawę o maksymalnie 11.9%, dla $A = 10^{-3}$, a dla serii o najniższej zadanej wartości FAR poprawa wyniosła 5.2%. Jeśli chodzi natomiast o miary jakości klasyfikacji ponownie nie ma między nimi dużych różnic.

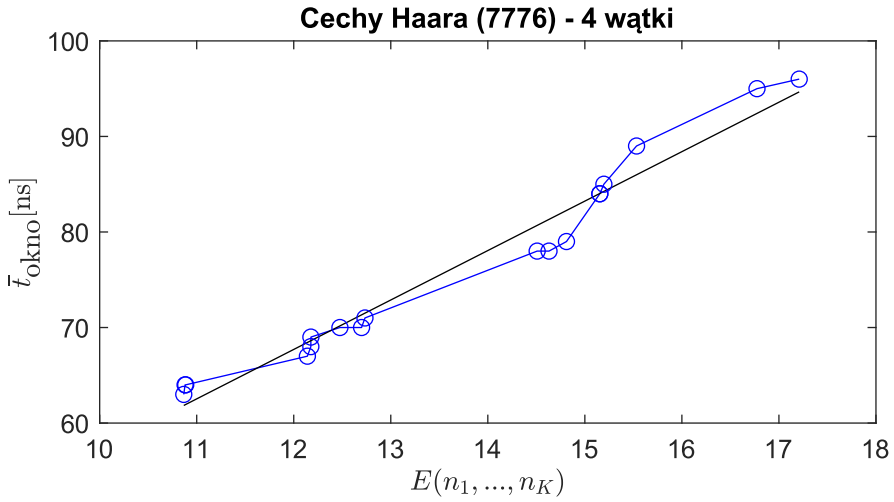
Tablica 4.6: „Detekcja twarzy” — kaskady z $K = 10$ etapami dla różnych metod uczenia (cechy Haara — 7776).

Algorytm uczący	Kaskada	Wartość oczekiwana	Walidacja FAR	Walidacja czułość	Test FAR	Test czułość	Test $\bar{n}$	Czas detekcji obraz [ST][ms]	Czas detekcji okno [ST][ns]	Czas detekcji Δ
			Wymagania: $\mathbf{10^{-3}}$ = **0.001000**	**0.9500**				(okna na obraz: 130 917)		
VJ	5 10 10 14 16 21 19 25 22 0.4460, 0.5011, 0.4604, 0.4435, 0.4494, 0.4522, 0.4867, 0.4560, 0.4734	14.6307	0.000974	0.9590	0.001063	0.9456	15.78	35.2	269	+0.0%
UGM	5 10 10 13 9 17 21 25 25 13 0.4460, 0.5011, 0.4604, 0.5113, 0.5070, 0.5068, 0.4847, 0.4878, 0.5121, 0.5830	14.5083	0.000954	0.9510	0.001052	0.9502	15.56	35.1	268	−0.4%
UGM-G	5 6 10 6 14 20 14 19 21 25 0.4460, 0.5433, 0.4907, 0.4821, 0.5362, 0.4551, 0.5422, 0.5123, 0.4848, 0.4919	12.6991	0.000927	0.9510	0.000945	0.9309	12.87	32.3	247	−8.2%
TREE-C3-L1 UGM-G	6 5 8 9 15 9 20 16 27 33 0.3717, 0.4983, 0.6586, 0.4775, 0.4963, 0.5352, 0.4964, 0.5133, 0.5040, 0.4872	12.1762	0.000968	0.9510	0.000999	0.9489	12.71	31.0	237	−11.9%
			Wymagania: $\mathbf{10^{-4}}$ = **0.000100**	**0.9500**				(okna na obraz: 130 917)		
VJ	6 8 12 14 19 31 49 64 57 64 0.3717, 0.3695, 0.3876, 0.3432, 0.3943, 0.3840, 0.3917, 0.3930, 0.3725, 0.3940	12.1768	0.000062	0.9550	0.000079	0.9383	12.85	31.2	238	+0.0%
UGM	6 7 15 15 21 28 34 36 48 56 0.3717, 0.3993, 0.3560, 0.3943, 0.3904, 0.3890, 0.3908, 0.4245, 0.4201, 0.4425	12.4758	0.000098	0.9510	0.000121	0.9419	13.40	32.1	245	+2.9%
UGM-G	6 7 13 18 29 28 25 40 34 46 0.3717, 0.3993, 0.4032, 0.4005, 0.3577, 0.4557, 0.4013, 0.3490, 0.4419, 0.4051	12.7339	0.000098	0.9510	0.000099	0.9263	13.51	32.5	248	+4.2%
TREE-C3-L1 UGM-G	6 7 13 18 29 28 25 40 34 46 0.3717, 0.3993, 0.4032, 0.4005, 0.3577, 0.4557, 0.4013, 0.3490, 0.4419, 0.4051	12.7339	0.000098	0.9510	0.000099	0.9263	13.51	32.5	248	+4.2%
TREE-C7-L3 UGM-G	3 9 9 14 21 26 28 55 77 64 0.6335, 0.2715, 0.3549, 0.4086, 0.3948, 0.3971, 0.3990, 0.3831, 0.4162, 0.3993	12.1408	0.000099	0.9510	0.000132	0.9370	13.02	30.8	235	−1.3%
			Wymagania: $\mathbf{10^{-5}}$ = **0.000010**	**0.9500**				(okna na obraz: 130 917)		
VJ	7 16 18 34 35 50 114 101 113 135 0.3043, 0.3093, 0.3155, 0.3161, 0.3146, 0.3123, 0.3114, 0.3062, 0.3045, 0.3113	15.1961	0.000008	0.9550	0.000019	0.9366	17.22	37.6	287	+0.0%
UGM	7 16 18 34 35 45 90 96 88 113 0.3043, 0.3093, 0.3155, 0.3161, 0.3146, 0.3133, 0.2990, 0.3184, 0.3196, 0.3392	15.1544	0.000010	0.9510	0.000014	0.9436	17.10	37.4	286	−0.3%
UGM-G	7 16 18 30 41 46 111 85 112 131 0.3043, 0.3093, 0.3155, 0.3349, 0.3101, 0.3228, 0.3061, 0.3115, 0.3321, 0.3171	15.1543	0.000010	0.9510	0.000013	0.9423	17.13	37.4	286	−0.3%
TREE-C3-L1 UGM-G	6 15 22 23 36 35 50 64 72 72 0.3717, 0.2524, 0.3061, 0.3230, 0.3130, 0.3310, 0.3052, 0.3073, 0.2802, 0.3860	14.8107	0.000010	0.9510	0.000014	0.9340	16.04	35.6	272	−5.2%



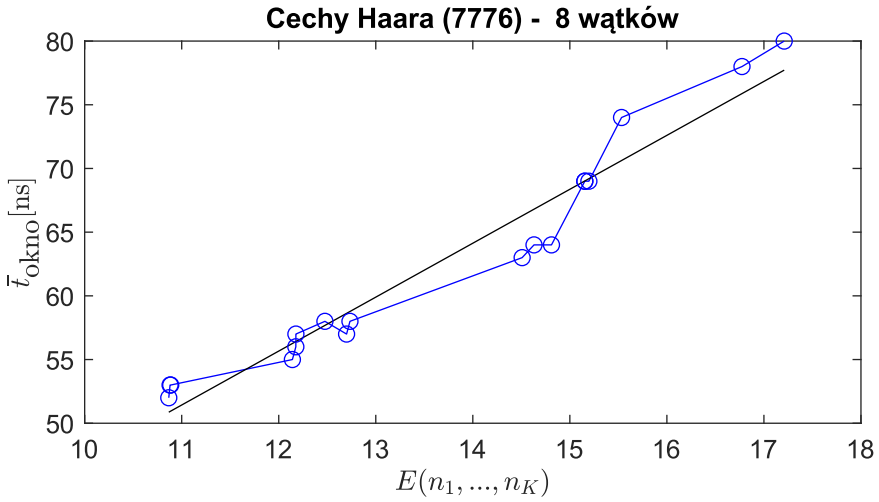
Rysunek 4.12: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Haara (7 776 cech) — 1 wątek. (Źródło: opracowanie własne)

Tak jak w przypadku większej przestrzeni cech także tutaj możemy wyraźnie zauważyć liniową zależność między wartością oczekiwaną a czasem klasyfikacji (Rys. 4.12). Dla jednego wątku czasy wahały się pomiędzy 219 ns a 340 ns.



Rysunek 4.13: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Haara (7 776 cech) — 4 wątki. (Źródło: opracowanie własne)

Zwiększenie liczby wątków do czterech pozwoliło na ponad trzykrotną redukcję czasów potrzebnych na klasyfikację pojedynczego okna. Czasy te wyniosły od 63 ns do 96 ns (Rys. 4.13). Zastosowanie zrównoleglenia ponownie nie wpłynęło na zależność między wartością oczekiwaną a czasem klasyfikacji.



Rysunek 4.14: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Haara (7 776 cech) — 8 wątków. (Źródło: opracowanie własne)

Dalsze zwiększenie liczby wątków do ośmiu pozwoliło na redukcję czasu klasyfikacji od 52 ns dla $E(n_1, ..., n_K) = 10.8667$ do 80 ns dla $E(n_1, ..., n_K) = 17.2091$ bez istotnego wpływu na przebieg zależności (Rys. 4.14).

Tablica 4.7: „Detekcja twarzy” — FPS dla różnej liczby wątków ($K = 10$, $A = 10^{-5}$, HAAR — 7776).

Algorytm	1-wątek			4-wątki			8-wątków		
uczący	obraz [ms]	FPS	Δ	obraz [ms]	FPS	Δ	obraz [ms]	FPS	Δ
VJ	37.6	26.6		11.1	90.1		9.0	111.1	
UGM	37.4	26.7	+0.1	11.0	90.9	+0.8	9.0	111.1	+0.0
UGM-G	37.4	26.7	+0.1	11.0	90.9	+0.8	9.0	111.1	+0.0
TREE-C3-L1-UGM-G	35.6	28.1	+1.5	10.3	97.1	+7.0	8.4	119.0	+7.9

Tabela 4.7 prezentuje różnicę w możliwościach obliczeniowych uzyskanych klasyfikatorów na przykładzie kaskad $K = 10, A = 10^{-5}$. O ile metody bazujące na poluzowanych wymaganiach na etap w przypadku tej serii kaskad dały zbliżone wyniki do kaskady w stylu VJ, to już w przypadku drzewa udało się przetworzyć około 200 tys. dodatkowych okien, co przełożyło się na 1.5 klatki na sekundę więcej. Zwiększając liczbę wątków udało się poprawić ten wynik do około miliona okien na sekundę więcej (7.9 FPS).

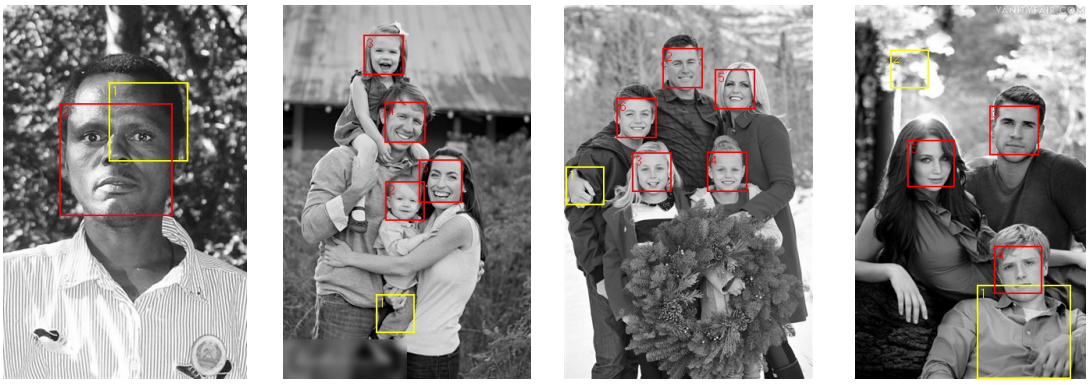
4.1.3 Cechy HOG

W przypadku cech HOG (Dalal i Triggs, 2005) przestrzeń ucząca była znacznie mniejsza i składała się z 400 cech, badane okno dzielono na siatkę składającą się z 5^2 komórek, a orientacje gradientów z przedziału $[0, 2\pi]$ zostały zdyskretyzowane na 16 podprzedziałów.

Rysunek 4.15: „Detekcja twarzy”: przykłady detekcji. Żółte ramki oznaczają fałszywe wykrycia (cechy HOG — 400). (Źródło: opracowanie własne)

Rys. 4.15 przedstawia wyniki działania klasyfikatorów opartych o cechy HOG. Także w tym przypadku wybrane zostały dwa klasyfikatory o najniższej wartości oczekiwanej, jeden spośród kaskad uczonych w celu spełnienia ograniczania na FAR $A = 10^{-5}$ a drugi dla $A = 10^{-6}$. Jak możemy zauważyć klasyfikator poprawnie rozpoznają twarze, a wynik ich działania jest zgodny z oczekiwaniami dla wskazanych nastaw.

Ponownie szczegółowe wyniki zostaną poprzedzone przedstawieniem wizualnych efektów (Rysunki 4.16, 4.17) działania najlepszej otrzymanej kaskady w zadaniu detekcji tym razem dla nastaw $K = 10$, $A = 10^{-6}$, $D = 0.95$.



Rysunek 4.16: „Detekcja twarzy”: przykłady detekcji, dla najlepszej otrzymanej kaskady dla nastaw $K = 10$, $A = 10^{-6}$ (cechy HOG — 400). (Źródło: opracowanie własne)

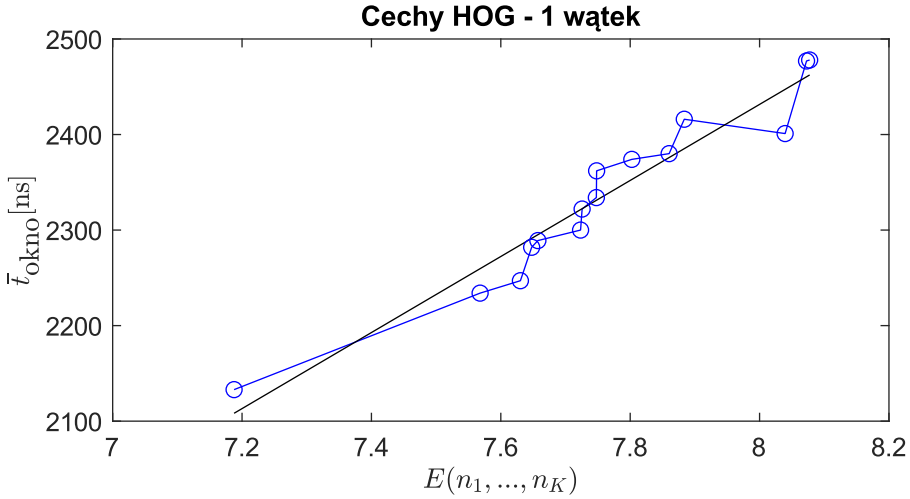
Rysunek 4.17: „Detekcja twarzy”: przykłady detekcji, dla najlepszej otrzymanej kaskady dla nastaw $K = 10$, $A = 10^{-6}$ (cechy HOG — 400). (Źródło: opracowanie własne)

Tablica 4.8: „Detekcja twarzy” — kaskady z $K = 10$ etapami dla różnych metod uczenia (cechy HOG — 400).

Algorytm uczący	Kaskada	Wartość oczekiwana	Walidacja FAR	Walidacja czułość	Test FAR	Test czułość	Test $\bar{n}$	Czas detekcji obraz [ST][ms]	Czas detekcji okno [ST][ns]	Czas detekcji Δ
			Wymagania: $\mathbf{10^{-3}} =$ **0.001000**	**0.9500**				(okna na obraz: 130 917)		
VJ	4 4 5 6 7 6 8 12 7 0.4341, 0.4908, 0.4361, 0.4121, 0.3578, 0.4976, 0.3803, 0.4092, 0.4698	7.8025	0.000498	0.9600	0.000640	0.9565	7.70	310.8	2374	+0.0%
UGM	4 4 5 5 6 6 8 7 6 8 0.4341, 0.4908, 0.4361, 0.5110, 0.5207, 0.4457, 0.5325, 0.5280, 0.5166, 0.5185	7.8602	0.000830	0.9500	0.000993	0.9479	7.68	311.6	2380	+0.3%
UGM-G	4 4 5 4 6 5 5 7 8 10 0.4341, 0.4908, 0.4361, 0.5549, 0.4345, 0.4991, 0.5967, 0.5901, 0.4782, 0.4571	7.7477	0.000860	0.9510	0.001044	0.9382	7.56	305.6	2334	−1.7%
TREE-C3-L1 UGM-G	3 4 4 5 3 7 7 6 8 9 0.5497, 0.4180, 0.4702, 0.3726, 0.7848, 0.4584, 0.4552, 0.5840, 0.4626, 0.5450	7.1878	0.000971	0.9510	0.001121	0.9482	6.56	279.2	2133	−10.2%
			Wymagania: $\mathbf{10^{-4}} =$ **0.000100**	**0.9500**				(okna na obraz: 130 917)		
VJ	5 6 8 10 9 13 18 18 13 19 0.2713, 0.3718, 0.3519, 0.3925, 0.3791, 0.3841, 0.3743, 0.3692, 0.3927, 0.3591	8.0395	0.000040	0.9550	0.000065	0.9668	8.36	314.3	2401	+0.0%
UGM	5 5 7 6 8 11 10 12 11 9 0.2713, 0.4046, 0.4015, 0.3706, 0.3573, 0.4397, 0.4440, 0.4213, 0.3646, 0.5679	7.6303	0.000099	0.9510	0.000105	0.9618	7.62	294.2	2247	−6.4%
UGM-G	5 4 6 8 8 8 9 11 12 13 0.2713, 0.4299, 0.5305, 0.3379, 0.3960, 0.3782, 0.4777, 0.3994, 0.3753, 0.4118	7.5680	0.000092	0.9510	0.000146	0.9671	7.52	292.5	2234	−7.8%
TREE-C3-L1 UGM-G	5 4 6 8 8 8 9 11 12 13 0.2713, 0.4299, 0.5305, 0.3379, 0.3960, 0.3782, 0.4777, 0.3994, 0.3753, 0.4118	7.5680	0.000092	0.9510	0.000146	0.9671	7.52	292.5	2234	−7.8%
			Wymagania: $\mathbf{10^{-5}} =$ **0.000010**	**0.9500**				(okna na obraz: 130 917)		
VJ	5 7 8 9 9 14 19 19 35 37 0.2713, 0.2554, 0.2858, 0.2959, 0.3115, 0.2993, 0.3130, 0.3152, 0.2972, 0.2974	7.7257	0.000005	0.9550	0.000004	0.9579	8.21	304.0	2322	+0.0%
UGM	5 7 8 9 9 12 20 16 18 19 0.2713, 0.2554, 0.2858, 0.2959, 0.3115, 0.3391, 0.3260, 0.3623, 0.3215, 0.3783	7.7233	0.000009	0.9510	0.000013	0.9721	8.17	301.1	2300	−0.9%
UGM-G	5 7 7 8 11 12 16 17 24 29 0.2713, 0.2554, 0.3616, 0.3875, 0.2915, 0.3314, 0.3032, 0.3501, 0.3170, 0.3095	7.7481	0.000010	0.9510	0.000033	0.9632	8.18	309.2	2362	+1.7%
TREE-C3-L1 UGM-G	6 4 8 9 10 11 12 16 21 18 0.1661, 0.4607, 0.3484, 0.3291, 0.3522, 0.3215, 0.2711, 0.3477, 0.3077, 0.3205	7.6573	0.000009	0.9510	0.000013	0.9691	7.89	299.7	2289	−1.4%
			Wymagania: $\mathbf{10^{-6}} =$ **0.000001**	**0.9500**				(okna na obraz: 130 917)		
VJ	6 9 10 14 17 20 43 34 56 48 0.1661, 0.2382, 0.2346, 0.2455, 0.2379, 0.2318, 0.2416, 0.2389, 0.2396, 0.2391	8.0774	0.000000	0.9550	0.000001	0.9602	8.73	324.4	2478	+0.0%
UGM	6 9 10 14 17 15 22 31 31 49 0.1661, 0.2382, 0.2346, 0.2455, 0.2379, 0.2659, 0.2620, 0.2749, 0.3102, 0.2742	8.0726	0.000001	0.9510	0.000003	0.9555	8.70	324.3	2477	−0.0%
UGM-G	6 8 7 10 16 22 20 22 49 32 0.1661, 0.2781, 0.3100, 0.2771, 0.2040, 0.3016, 0.1973, 0.3169, 0.2361, 0.2772	7.8836	0.000001	0.9510	0.000003	0.9658	8.42	316.3	2416	−2.5%
TREE-C3-L1 UGM-G	5 8 7 9 15 14 30 30 48 62 0.2713, 0.1490, 0.3236, 0.2731, 0.2630, 0.2643, 0.2229, 0.2758, 0.2402, 0.2325	7.6477	0.000001	0.9510	0.000001	0.9532	8.20	298.8	2282	−7.9%

Powyższa Tabela 4.8 zawiera porównanie dziesięcioetapowych kaskad wykorzystujących cechy HOG. Pomimo zmiany typu cech nadal najskuteczniejszą metodą pozostała ta oparta na przeszukiwaniu drzewa, poprawiając wartość oczekiwaną dla każdej z widocznych tutaj serii kaskad. Jeśli chodzi o wartości miar jakości klasyfikacji tutaj także, pomijając nieliczne wyjątki, były do siebie zbliżone, a czasami gorsze wartości FAR były nadrabiane wyższą czułością. W przypadku czasu klasyfikacji maksymalna poprawa wyniosła 10.2%. Dla klasyfikatorów o najniższych wskaźnikach FAR, co za tym idzie najdokładniejszych spośród nauczonych, udało się zredukować czas klasyfikacji o 7.9%. Oba wspominanie wyniki były możliwe do uzyskania dzięki zastosowaniu techniki opartej o przeszukiwanie drzewa w połączeniu z zachłannymi poluzowanymi wymaganiami na etap.

Warto zaznaczyć, że dla kaskad o wymaganiu $A = 10^{-5}$ mamy do czynienia z jednym z nielicznych przypadków, gdy *dodanie* cechy na początkowych etapach pozwoliło na zmniejszenie finalnej wartości oczekiwanej (postępowanie odwrotne do zachłannego).



Rysunek 4.18: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech HOG — 1 wątek. (Źródło: opracowanie własne)

Dla cech HOG ponownie można zauważyć liniową zależność między wartością oczekiwaną liczby cech a średnim czasem klasyfikacji okna (Rys. 4.18). Dla tego badania średni czas analizy na okno wahał się pomiędzy 2 133 ns a 2 478 ns. Jeśli chodzi o wzrost liczby przetwarzanych okien na sekundę był on najniższy dla kaskad $A = 10^{-5}$, gdzie udało się przetworzyć około 6 tys. okien więcej, a najwyższy dla kaskad $A = 10^{-3}$, gdzie wyniósł prawie 50 tys. okien. Dla pozostałych dwóch nastaw udało się przetworzyć około 30 tys. okien na sekundę więcej. W związku z zastosowaną implementacją tym razem nie zastosowano zrównoleglenia.

4.1.4 Przycinanie drzew

Kolejny eksperyment ma na celu sprawdzenie skuteczności algorytmów przycinania drzewa oraz wpływu doboru metody wyznaczania wymagań na etap dla drzewa na jego wartość oczekiwaną (Tabela 4.9). W eksperymencie tym zostały ze sobą zestawione ka-

skady uzyskane przy wykorzystaniu rożnych nastaw dla drzewa oraz różnych sposobów wyznaczania wymagań na etap (klasyczne w stylu VJ, UGM oraz UGM-G). Na podstawie widocznych tutaj wyników zdecydowano się na stosowanie, jako wariantu domyślnego, drzewa połączonego z zachłannymi poluzowanymi wymaganiami (UGM-G) — kombinacja taka pozwalała na uzyskanie najlepszych wartości dla wartości oczekiwanej. Jeżeli chodzi o algorytmy przycinania, należy zacząć od przypomnienia, że czym dalej węzeł znajduje się w kaskadzie tym jego nauka staje się bardziej czasochłonna. W związku z tym usunięcie nawet pojedynczych węzłów może znacznie skrócić czas nauki.

Jak można zauważyć na podstawie Tabeli 4.9, już dla dokładnego przycinania udało się uzyskać znaczną poprawę w liczbie odwiedzonych węzłów, co przekładało się na zmniejszenie czasu nauki kaskad. Przy zastosowaniu przycinania z przybliżonymi predykcjami wartości oczekiwanej oraz małej wartości $\alpha = 0.8$, liczba analizowanych węzłów uległa dalszej poprawie pozwalając na pominięcie od 1 do nawet 5 węzłów więcej. Zwiększenie wartości α pozwoliło na pominięcie nawet o 9 węzłów więcej. Co ważne wszystkie uzyskane kaskady były identyczne z tymi uzyskanymi w przypadku pełnego przeszukiwania drzewa.

Tablica 4.9: „Detekcja twarzy” (cechy Haara) — porównanie podejść opartych na drzewie przeszukiwań dla różnych ustawień parametrów C, L oraz różnych metod uczenia (VJ, UGM, UGM-G).

Algorytm uczący	Kaskada	Wartość oczekiwana	Walidacja		Nauczone węzły		
			FAR	czułość	przycinanie dokładne	aproksymacja	
			Wymagania:				
			$\mathbf{10^{-3}} =$ **0.001000**	**0.9500**		$\alpha = 0.8$	$\alpha = 1.2$
VJ	9 18 26 30 38 0.2468, 0.2214, 0.2299, 0.2468, 0.2370	15.3571	0.000735	0.9520			
TREE-C3-L1 VJ	8 16 22 35 55 0.2703, 0.2329, 0.2188, 0.2419, 0.2487	14.3735	0.000828	0.9520	12/15	11/15	10/15
TREE-C3-L1 UGM	8 16 22 27 37 0.2703, 0.2329, 0.2188, 0.2601, 0.2439	14.2127	0.000873	0.9510	12/15	11/15	10/15
TREE-C3-L1 UGM-G	8 16 22 25 40 0.2703, 0.2329, 0.2188, 0.2713, 0.2646	14.2021	0.000988	0.9510	12/15	11/15	11/15
TREE-C3-L2 VJ	8 16 22 35 55 0.2703, 0.2329, 0.2188, 0.2419, 0.2487	14.3735	0.000828	0.9520	30/39	26/39	23/39
TREE-C3-L2 UGM	8 16 22 27 37 0.2703, 0.2329, 0.2188, 0.2601, 0.2439	14.2127	0.000873	0.9510	29/39	25/39	24/39
TREE-C3-L2 UGM-G	8 16 22 25 40 0.2703, 0.2329, 0.2188, 0.2713, 0.2646	14.2021	0.000988	0.9510	30/39	26/39	25/39
TREE-C5-L1 VJ	7 18 23 30 40 0.2770, 0.2134, 0.2500, 0.2426, 0.2408	13.9309	0.000863	0.9520	17/25	15/25	15/25
TREE-C5-L1 UGM	7 18 21 26 43 0.2770, 0.2134, 0.2535, 0.2577, 0.2456	13.7815	0.000948	0.9510	17/25	15/25	14/25
TREE-C5-L1 UGM-G	7 18 17 33 30 0.2770, 0.2134, 0.2648, 0.2514, 0.2452	13.6241	0.000965	0.9510	16/25	15/25	13/25
VJ	4 6 16 13 11 17 14 21 24 45 0.45, 0.45, 0.48, 0.42, 0.50, 0.47, 0.45, 0.49, 0.50, 0.47	12.3741	0.000505	0.9560			
TREE-C3-L1 VJ	4 6 16 13 11 17 14 21 24 45 0.45, 0.45, 0.48, 0.42, 0.50, 0.47, 0.45, 0.49, 0.50, 0.47	12.3741	0.000505	0.9560	24/30	23/30	22/30
TREE-C3-L1 UGM	3 6 5 10 11 14 24 15 15 18 0.76, 0.34, 0.45, 0.50, 0.48, 0.45, 0.52, 0.50, 0.54, 0.51	11.5081	0.000882	0.9510	19/30	17/30	17/30
TREE-C3-L1 UGM-G	4 6 5 14 8 12 17 16 14 34 0.45, 0.44, 0.59, 0.51, 0.49, 0.52, 0.50, 0.47, 0.54, 0.45	10.7018	0.000899	0.9510	24/30	23/30	23/30
TREE-C3-L2 VJ	3 6 5 10 11 15 17 30 40 19 0.76, 0.34, 0.45, 0.50, 0.48, 0.45, 0.46, 0.48, 0.49, 0.50	11.5850	0.000683	0.9550	49/84	44/84	42/84
TREE-C3-L2 UGM	3 6 5 10 11 14 24 15 15 18 0.76, 0.34, 0.45, 0.50, 0.48, 0.45, 0.52, 0.50, 0.54, 0.51	11.5081	0.000882	0.9510	44/84	39/84	35/84
TREE-C3-L2 UGM-G	4 6 5 14 8 12 17 16 14 34 0.45, 0.44, 0.59, 0.51, 0.49, 0.52, 0.50, 0.47, 0.54, 0.45	10.7018	0.000899	0.9510	56/84	55/84	48/84
TREE-C5-L1 VJ	4 6 16 13 11 17 14 21 24 45 0.45, 0.45, 0.48, 0.42, 0.50, 0.47, 0.45, 0.49, 0.50, 0.47	12.3741	0.000505	0.9560	38/50	37/50	36/50
TREE-C5-L1 UGM	3 6 5 10 11 14 24 15 15 18 0.76, 0.34, 0.45, 0.50, 0.48, 0.45, 0.52, 0.50, 0.54, 0.51	11.5081	0.000882	0.9510	33/50	30/50	29/50
TREE-C5-L1 UGM-G	4 6 5 14 8 12 17 16 14 34 0.45, 0.44, 0.59, 0.51, 0.49, 0.52, 0.50, 0.47, 0.54, 0.45	10.7018	0.000899	0.9510	40/50	39/50	36/50

4.2 Wykrywanie litery „A” (dane syntetyczne)

Drugim rozpoznawanym obiektem była litera „A”. Tym razem obrazy zostały wygenerowane syntetycznie na podstawie zbioru czcionek komputerowych przedstawionych w pracy (de Campos i in., 2009). Litery ze wspomnianego zbioru były losowo rozmieszczane na przygotowanym zestawie teł, a litery „A” były traktowane jako poszukiwany obiekt w gronie wszystkich liter. Ponadto litery te mogły występować w obrocie o dowolny kąt, przy czym w zbiorze uczącym i walidacyjnym kąt obrotu był ograniczony do ±45°, natomiast w zbiorze testowym występował pełny zakres kątowy. Rysunek 4.19 przedstawia część z materiałów graficznych na podstawie, których generowano obrazy (przykłady pozytywne, przykłady negatywne, tła).

Rysunek 4.19: Przykładowe obiekty i tła użyte w zadaniu detekcji „syntetycznych liter A”. Pozytywy: litery „A” (a), negatywy: inne litery (b), użyte tła (c). (Źródło: Bera, Klęsk i Sychel (2018))

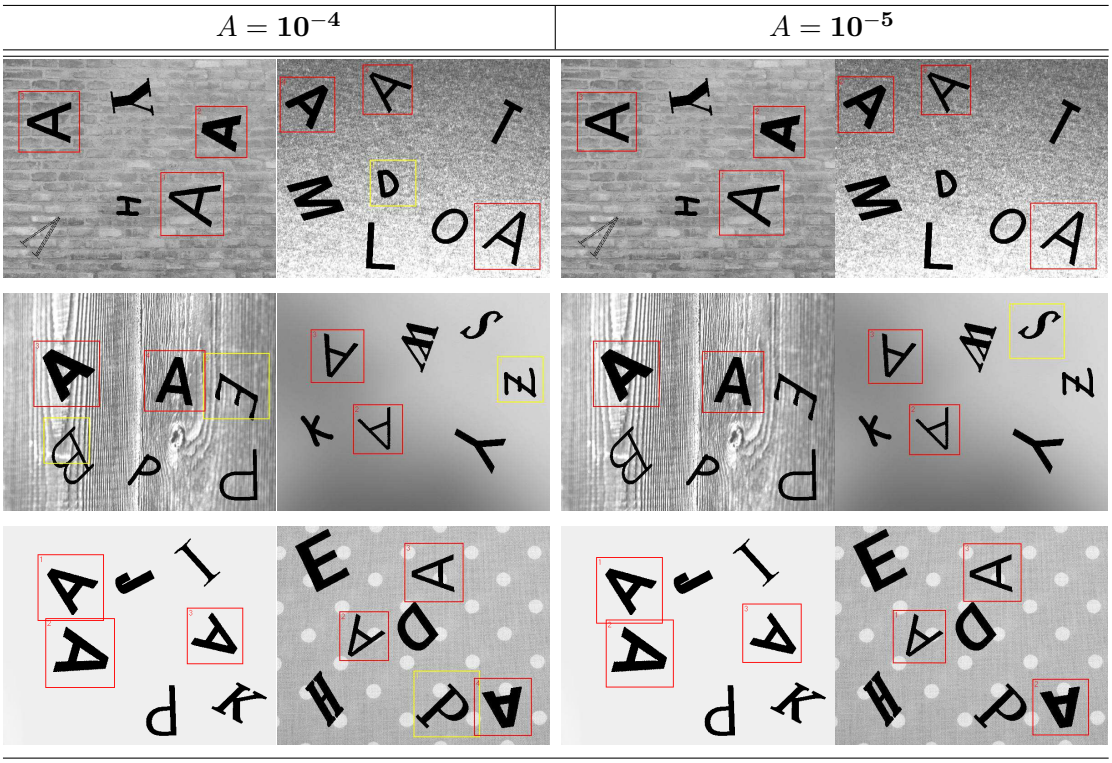
Tabela 4.10 przedstawia nastawy wykorzystane w tym eksperymencie. Ponownie liczba przykładów negatywnych była obniżana wraz z kolejnymi etapami w celu redukcji czasu nauki. W nastawach procedury detekcyjnej występuje istotna różnica — tym razem minimalny rozmiar okna wynosi 100×100 pikseli co przekłada się na mniejszą liczbę przeanalizowanych okien na obraz.

Tablica 4.10: „Syntetyczne litery A”: nastawy eksperymentu.

dane uczące		
parametr	wartość	dodatkowe informacje
liczba przykładów pozytywnych	25 208	okna z literą „A”
liczba przykładów negatywnych	350 000/ ≈ 25 000/ ≈ 10 000	na 1/2/kolejnych etapach
rozmiar zbioru uczącego	375 208/50 000/35 000	przykładów łącznie na 1/2/kolejnych etapach
dane walidacyjne		
liczba przykładów pozytywnych	1 092	okna z literą „A”
liczba przykładów negatywnych	35 000/ ≈ 24 000	na 1/kolejnych etapach
rozmiar zbioru walidacyjnego	36 092/25 000	przykładów łącznie na 1/kolejnych etapach
dane testowe		
liczba przykładów pozytywnych	20 000	okna z literą „A”
liczba przykładów negatywnych	1 000 000	
rozmiar zbioru testowego	1 020 000	przykładów łącznie
procedura detekcyjna (skanowanie oknem przesuwnym)		
liczba powtórzeń	1000	na potrzeby uśrednienia pomiarów czasu
rozdzielczość obrazu	600×480	
liczba skal detekcyjnych	5	obraz przeskanowany z użyciem 5 różnych rozmiarów okien
współczynnik przyrostu okna	1.2	wysokość i szerokość okna wzrasta o ≈ 20% na każdą skalę
rozmiar najmniejszego okna	100×100	
rozmiar największego okna	208×208	
współczynnik przesunięcia okna	0.05	okno przesuwa się o ≈ 5% jego rozmiaru

4.2.1 Momenty Zernike'a

Ponieważ obiekty mają zostać wykryte niezależnie od ich obrotu tym razem zostały zastosowane cechy obrotowo niezmiennicze bazujące na momentach Zernike'a wsparte obrazami całkowymi (Bera, Klęsk i Sychel, 2018). W przeciwieństwie do prac (Sychel, Klęsk i Bera, 2020a,b) skorzystano z ich nowszej implementacji rozszerzonej o redukcję błędów numerycznych (Klęsk, Bera i Sychel, 2020). W przypadku momentów Zernike'a liczba cech wyniosła 540[1].

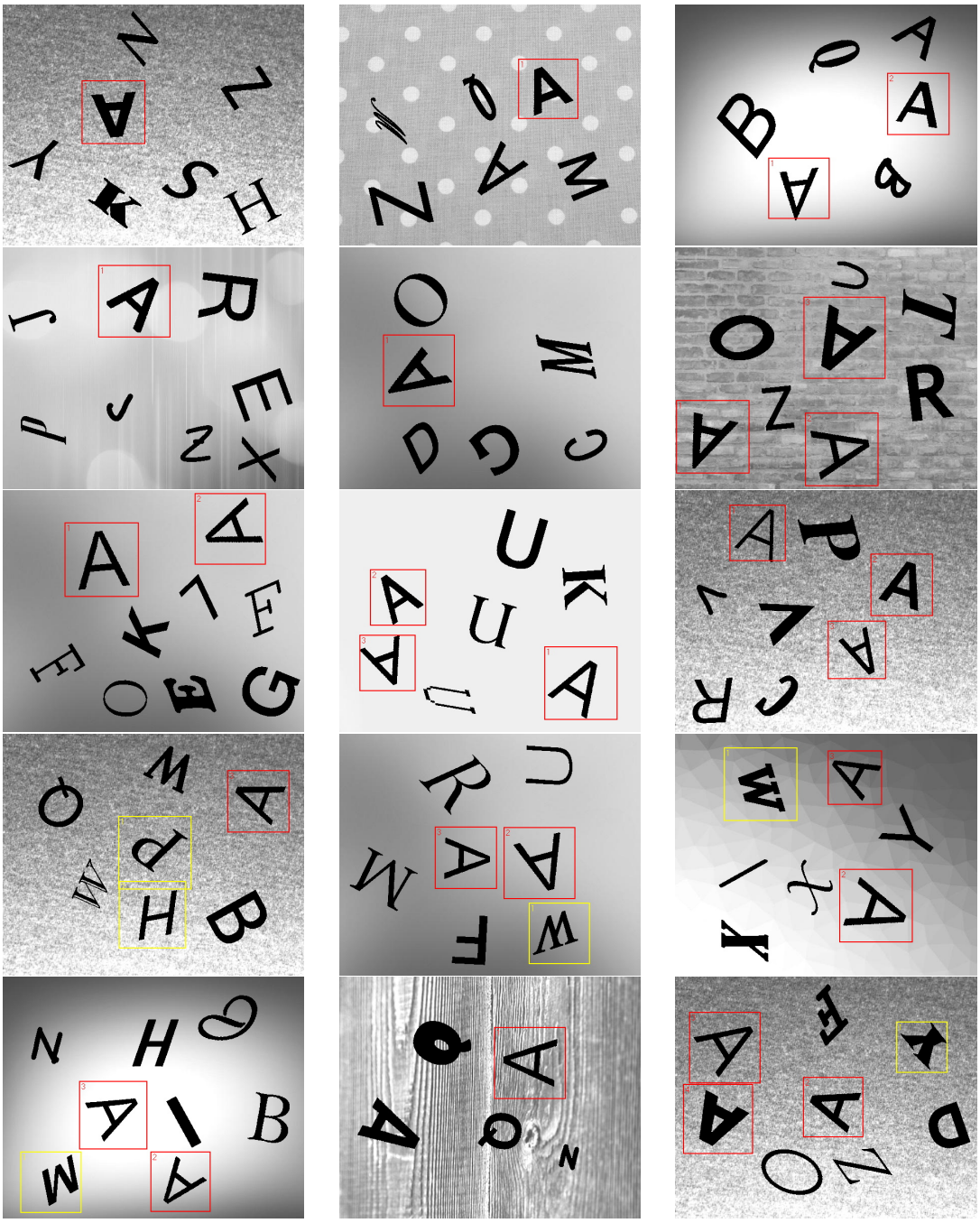


Rysunek 4.20: „Syntetyczne litery A": przykłady detekcji. Żółte ramki oznaczają fałszywe wykrycia. (Źródło: opracowanie własne)

Rys. 4.20 przedstawia przykładowe wyniki detekcji. Wyniki te pochodzą z dwóch klasyfikatorów o najniższej wartości oczekiwanej, pierwszy z nich miał uzyskać FAR nie większy niż $A = 10^{-4}$, a drugi niż $A = 10^{-5}$. Jak można zauważyć uzyskane klasyfikatory poradziły sobie z detekcją litery „A" niezależnie od kąta jej obrotu.

Szczegółowe wyniki zostaną poprzedzone przedstawieniem wizualnych efektów (Rysunki 4.21) działania najlepszej otrzymanej kaskady w zadaniu detekcji dla nastaw $K = 10$, $A = 10^{-5}$, $D = 0.95$. Podobnie jak: dane uczące, dane walidacyjne oraz testowe, poniższe obrazy zostały wygenerowane syntetycznie. Ich utworzenie polegało na wylosowaniu tła, a następnie rozmieszczeniu w losowych miejscach obiektów (zarówno przykładów pozytywnych jak i negatywnych) w taki sposób, aby na siebie nie nachodziły. W tym przypadku dla obiektów dozwolony był pełny zakres kątowy (Bera, Klęsk i Sychel, 2018).

[1]Liczba cech (540) wynika z następujących nastaw: ρ=10, ϱ=10, R=8 — nastawy te są związane z maksymalnym stopniem radialnym i harmonicznym wielomianów Zernike'a oraz z tzw. pierścieniami, więcej szczegółów jest dostępnych w pracy (Bera, Klęsk i Sychel, 2018).

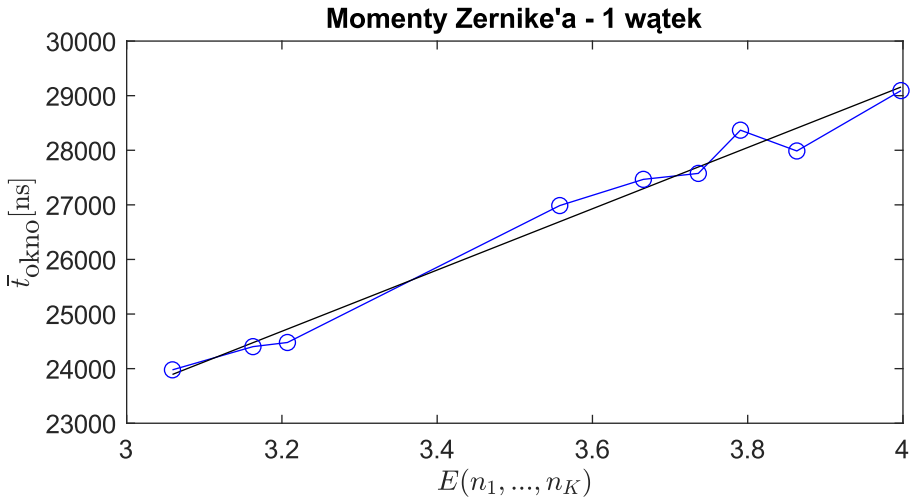


Rysunek 4.21: „Syntetyczne litery A”: przykłady detekcji, dla najlepszej otrzymanej kaskady dla nastaw $K = 10$, $A = 10^{-5}$. (Źródło: opracowanie własne)

Tabela 4.11 zawiera porównanie kaskad dziesięcioetapowych utworzonych na cele tego zadania. Jeśli chodzi o skuteczność uzyskanych kaskad ponownie osiągnięte wartości czułości oraz miary FAR są do siebie zbliżone. Co ciekawe, czułość wszystkich kaskad okazała się znacznie wyższa od zadanej minimalnej D. W tym eksperymencie zwycięskimi metodami okazały się ex aequo metoda UGM-G oraz przeszukiwanie drzew (przy domyślnych nastawach). W przypadku klasycznego podejścia w stylu VJ nauka wszystkich kaskad zakończyła się już po dziewięciu etapach. Zysk w czasie klasyfikacji wynikający z zastosowania tych metod zawierał się pomiędzy 2% a 5.2%.

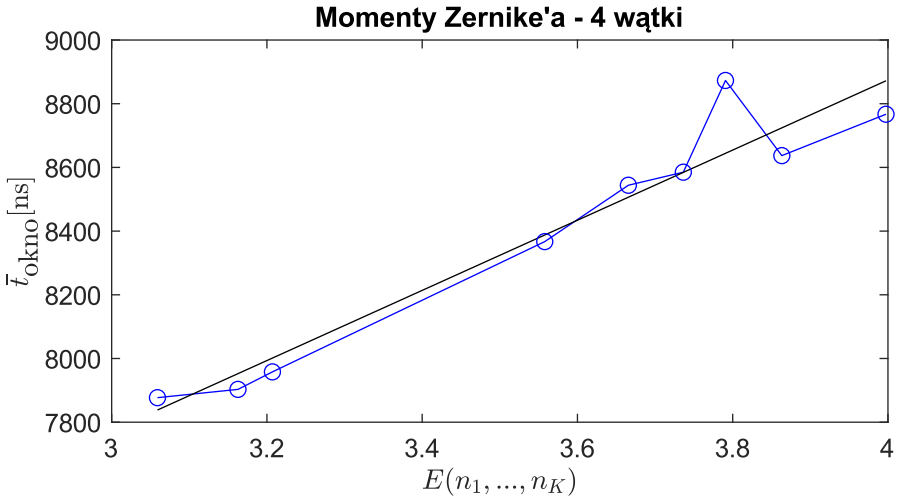
Tablica 4.11: „Syntetyczne litery A” — kaskady z $K = 10$ etapami dla różnych metod uczenia.

Algorytm uczący	Kaskada	Wartość oczekiwana	Walidacja FAR	Walidacja czułość	Test FAR	Test czułość	Test $\bar{n}$	Czas detekcji obraz [ST][ms]	Czas detekcji okno [ST][ns]	Czas detekcji Δ
			Wymagania: $\mathbf{10^{-3}} =$ **0.001000**	**0.9500**				(okna na obraz: 18 752)		
VJ	2 1 2 2 3 3 7 8 15 0.4565, 0.3982, 0.4090, 0.4159, 0.4681, 0.4926, 0.4827, 0.4826, 0.4704	3.2071	0.000781	0.9643	0.000521	0.9744	3.35	459.0	24479	+0.0%
UGM	2 1 2 2 3 3 5 5 7 8 0.4565, 0.3982, 0.3983, 0.3691, 0.4619, 0.5482, 0.6154, 0.5344, 0.5900, 0.7082	3.1627	0.000930	0.9505	0.000596	0.9622	3.30	457.6	24403	−0.3%
UGM-G	2 1 1 2 1 2 3 4 5 10 0.4565, 0.3982, 0.5713, 0.3860, 0.7685, 0.5096, 0.4631, 0.4791, 0.5128, 0.5548	3.0590	0.000991	0.9505	0.000647	0.9696	3.17	449.6	23978	−2.0%
TREE-C3-L1 UGM-G	2 1 1 2 1 2 3 4 5 10 0.4565, 0.3982, 0.5713, 0.3860, 0.7685, 0.5096, 0.4631, 0.4791, 0.5128, 0.5548	3.0590	0.000991	0.9505	0.000647	0.9696	3.17	449.6	23978	−2.0%
			Wymagania: $\mathbf{10^{-4}} =$ **0.000100**	**0.9500**				(okna na obraz: 18 752)		
VJ	3 2 4 5 11 26 22 49 79 0.1442, 0.3550, 0.3871, 0.3423, 0.3890, 0.3963, 0.3900, 0.3809, 0.6314	3.7907	0.000098	0.9588	0.000089	0.9817	5.60	532.0	28368	+0.0%
UGM	3 2 2 5 11 11 13 19 50 28 0.1442, 0.3550, 0.4336, 0.3030, 0.4491, 0.5011, 0.4672, 0.5156, 0.5196, 0.5274	3.6656	0.000100	0.9505	0.000094	0.9772	4.80	515.1	27467	−3.2%
UGM-G	3 1 1 2 4 7 15 17 32 31 0.1442, 0.6294, 0.5180, 0.3919, 0.3882, 0.5219, 0.3773, 0.3915, 0.4426, 0.3934	3.5579	0.000096	0.9505	0.000089	0.9717	4.52	506.1	26987	−4.9%
TREE-C3-L1 UGM-G	3 1 1 2 4 7 15 17 32 31 0.1442, 0.6294, 0.5180, 0.3919, 0.3882, 0.5219, 0.3773, 0.3915, 0.4426, 0.3934	3.5579	0.000096	0.9505	0.000089	0.9717	4.52	506.1	26987	−4.9%
			Wymagania: $\mathbf{10^{-5}} =$ **0.000010**	**0.9500**				(okna na obraz: 18 752)		
VJ	3 3 4 13 47 65 108 102 131 0.1442, 0.3043, 0.2421, 0.2859, 0.3160, 0.3159, 0.3116, 0.2946, 0.3033	3.9975	0.000008	0.9606	0.000039	0.9810	8.19	545.6	29093	+0.0%
UGM	3 3 3 10 14 26 40 59 49 44 0.1442, 0.3043, 0.3433, 0.3178, 0.3514, 0.3342, 0.3294, 0.3528, 0.3684, 0.4074	3.8633	0.000010	0.9505	0.000025	0.9778	5.96	524.8	27987	−3.8%
UGM-G	3 1 2 8 6 15 66 115 60 114 0.1442, 0.6294, 0.2978, 0.2862, 0.4051, 0.2755, 0.3510, 0.3080, 0.3270, 0.3252	3.7364	0.000010	0.9505	0.000032	0.9768	7.20	517.1	27576	−5.2%
TREE-C3-L1 UGM-G	3 1 2 8 6 15 66 115 60 114 0.1442, 0.6294, 0.2978, 0.2862, 0.4051, 0.2755, 0.3510, 0.3080, 0.3270, 0.3252	3.7364	0.000010	0.9505	0.000032	0.9768	7.20	517.1	27576	−5.2%



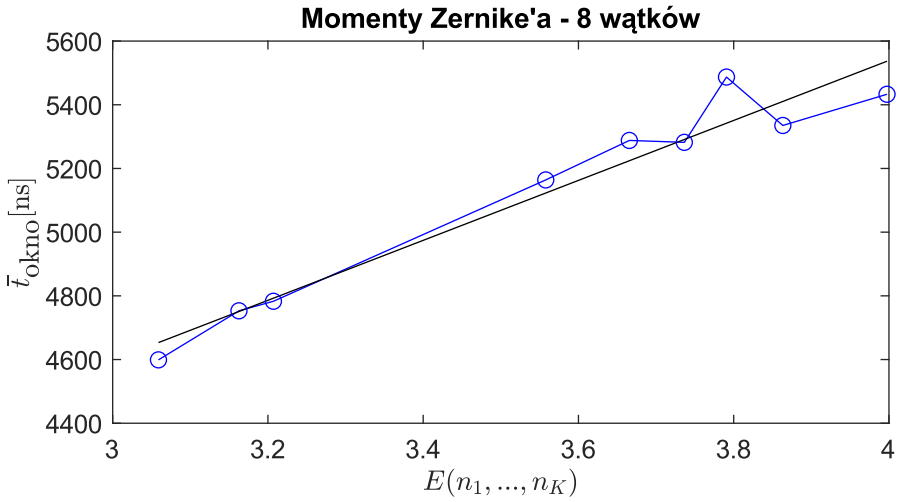
Rysunek 4.22: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Zernike'a — 1 wątek. (Źródło: opracowanie własne)

Ponieważ zastosowanie obrazów całkowych sprawiło, że koszty obliczenia dowolnych momentów są niezależne od liczby pikseli w oknie, to i w tym przypadku zależność między czasem klasyfikacji a wartością oczekiwaną okazała się liniowa (Rys. 4.22). Jako że cechy te są znacznie bardziej kosztowne do wyznaczania niż poprzednie przedstawione w tej pracy, to widoczne tutaj czasy są znacznie wyższe i wahają się między 23 978 ns a 29 093 ns dla jednego wątku.



Rysunek 4.23: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Zernike'a — 4 wątki. (Źródło: opracowanie własne)

Zwiększenie liczby wątków do czterech, pozwoliło na około trzykrotne zmniejszenie czasów klasyfikacji. Czas ten po zastosowaniu zrównoleglenia wahał się między 7 873 ns a 8 767 ns. Zależność między czasem a wartością oczekiwaną pozostała wyraźnie liniowa, chociaż można dostrzec drobną anomalię.



Rysunek 4.24: Wykres zależności między wartością oczekiwaną $E(n_1, ..., n_K)$ a średnim czasem potrzebnym do rozpatrzenia jednego okna $\bar{t}_{\text{okno}}$ dla cech Zernike'a — 8 wątków. (Źródło: opracowanie własne)

Dalsze zwiększanie liczby wątków (do ośmiu) poskutkowało redukcją czasów klasyfikacji do przedziału $[4599; 5433]$ ns, dając około pięciokrotną poprawę. Także i w tym przypadku zwiększenie liczby wątków nie miało wpływu na zależność między czasem a wartością oczekiwaną, nadal możemy dostrzec korzystny wpływ redukcji oczekiwanej na czas detekcji obiektów.

Tablica 4.12: „Syntetyczne litery A" — FPS dla różnej liczby wątków ($K = 10$, $A = 10^{-5}$, Momenty Zernike'a).

Algorytm	1-wątek			4-wątki			8-wątków		
uczący	obraz [ms]	FPS	Δ	obraz [ms]	FPS	Δ	obraz [ms]	FPS	Δ
VJ	545.6	1.8		164.4	6.1		101.9	9.8	
UGM	524.8	1.9	+0.1	162.0	6.2	+0.1	100.0	10.0	+0.2
UGM-G	517.1	1.9	+0.1	161.0	6.2	+0.1	99.0	10.1	+0.3
TREE-C3-L1-UGM-G	517.1	1.9	+0.1	161.0	6.2	+0.1	99.0	10.1	+0.3

Tabela 4.12 pokazuje korzyści płynące z redukcji wartości oczekiwanej w połączeniu ze zrównolegleniem. Dla przejrzystości ponownie do porównania wybrano kaskady dziesięcioetapowe dla $A = 10^{-5}$. Ponieważ momenty Zernike'a w połączeniu z algorytmem redukcji błędów numerycznych są stosunkowo ciężkie obliczeniowo, konieczne byłoby dalsze zwiększanie liczby wątków w celu uzyskania różnicy między przedstawionymi metodami na poziomie kilku klatek. Jednak nadal jesteśmy w stanie zauważyć korzyści wynikające z redukcji wartości oczekiwanej. Przy wykorzystaniu jednego wątku zastosowanie metody UGM-G lub TREE-C3-L1-UGM-G pozwoliło na przetworzenie dodatkowych 2 tys. okien. Zwiększenie liczby wątków do ośmiu powiększyło tę różnicę do prawie 6 tys. okien, co stanowi około $\frac{1}{3}$ okien w badanych w tym zadaniu obrazach.

4.3 Podsumowanie

Utworzone techniki redukcji wartości oczekiwanej liczby cech okazały się skuteczne i pozwoliły na istotne zwiększenie możliwości obliczeniowych uzyskanych przy ich pomocy kaskad. Poprawa ta była szczególnie widoczna w połączeniu ze zrównolegleniem obliczeń, które znacznie wzmocniło różnice pomiędzy metodami w liczbie okien możliwych do przetworzenia na sekundę.

Tablica 4.13: Porównanie opracowanych metod pod względem uzyskanej wartości oczekiwanej.

Algorytm uczący	Wartość oczekiwana		
	najniższa	remis	najwyższa
VJ	0	0	14
UGM	1	0	2
UGM-G	0	8	2
TREE-UGM-G	9	8	0

Tabela 4.13 porównuje ze sobą metody uczenia kaskady ze względu na uzyskane przez nie wartości oczekiwane. W powyższym zestawieniu nie znalazły się kaskady uzyskane na potrzeby testów algorytmów przycinania drzewa, ponieważ w eksperymencie tym stosowana była tylko jedna metoda (TREE) przy wykorzystaniu różnych nastaw. Najgorzej ze wszystkich wypadała metoda klasyczna w stylu VJ, która uzyskała najwyższe wartości oczekiwane dla 14 spośród 18 kaskad. Najlepszą metodą okazałą się ta oparta o przeszukiwanie drzew (mimo niewielkiego rozmiaru stosowanych drzew) uzyskując 9 razy najniższą wartość oczekiwaną, a 8 razy zwycięski remis ex aequo z metodą UGM-G. Drugie miejsce zajęła metoda UGM-G uzyskując w 8 przypadkach wynik identyczny do algorytmu TREE-UGM-G. Najgorzej z proponowanych metod wypadła metoda UGM, uzyskując najmniejszą wartość oczekiwaną tylko 1 raz. Przy czym gdyby w zestawieniu brała udział jedynie ta metoda oraz klasyczna, to metoda UGM uzyskałaby znaczną przewagę (15 zwycięstw na 18 przypadków). Należy także zaznaczyć, że 0 w kolumnie najmniejszej liczby oczekiwanej dla metody UGM-G, nie wynika z jej niskiej skuteczności, lecz z faktu, że była ona stosowana także w drzewach, w związku z tym odnalezienie przez tę metodę kaskady o najmniejszej wartości oczekiwanej oznaczało automatyczny remis z podejściem opartym na drzewach. Równie ważny jest fakt, że w powyższych eksperymentach stosowano bardzo małe drzewa ($L = 1, C = 3$) w celu ograniczenia czasu nauki. Możliwe jest uzyskanie nawet większej przewagi dla techniki opartej na drzewach w wyniku stosowania dokładniejszego przeszukiwania, co pokazał np. eksperyment dla cech Haara widoczny w Tabeli 4.6, gdzie konieczne było zastosowanie nastaw ($L = 3, C = 7$) w celu pokonania klasycznej kaskady. W związku z tym jakość uzyskanego rozwiązania w dużej mierze zależy od czasu, jaki jesteśmy gotowi poświęcić na naukę klasyfikatora oraz jakości algorytmów przycinania, które pozwalają ten czas zredukować.

Rozdział 5

Wnioski

Celem niniejszej rozprawy doktorskiej było opracowanie algorytmu uczenia kaskady klasyfikatorów redukującego oczekiwaną liczbę ekstrahowanych cech w porównaniu do algorytmu Violi i Jonesa przy zachowaniu końcowych wymagań dla czułości i specyficzności. Postawiony cel został zrealizowany. W poniższej pracy zaproponowano dwie techniki wychodzące z różnych motywacji, które mogą pracować niezależnie, ale mogą też współpracować i w wybranym połączeniu stanowią takowy algorytm.

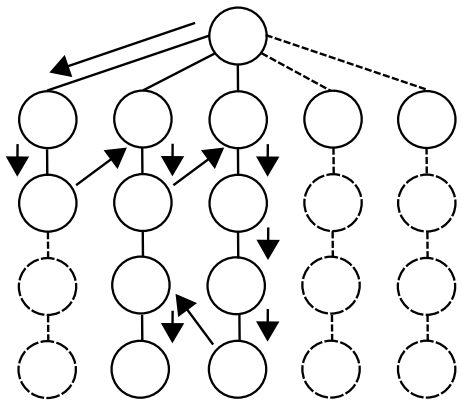
Pierwsza technika opierała się na wykorzystaniu aktualizowanych średnich geometrycznych i posiadała dwa warianty. Nie wpływały one na czas uczenia klasyfikatora jednak w sporadycznych przypadkach potrafiły zwrócić kaskadę o gorszej wartości oczekiwanej (dla UGM były to 3 przypadki, natomiast dla UGM-G 2 przypadki na 18 nauczonych kaskad). Druga proponowana technika polegała na sprowadzeniu nauki kaskady do przeszukiwania drzewa. Podejście to gwarantuje redukcję wartości oczekiwanej, jednak znacząco wpływa na czas nauki. W związku z tym rozmiar drzewa w zaproponowanym rozwiązaniu jest ściśle kontrolowany. Dodatkowo została zaproponowana technika przycinania drzewa (wpisująca się w algorytmy klasy „branch-and-bound") oraz jej aproksymacja pozwalająca zmniejszyć czas nauki kaskady. Warto zauważyć, że w większości przypadków (z pominięciem jednego) już dla drzewa o współczynniku rozgałęzienia $C = 3$ rozgałęzionego jedynie na pierwszym poziomie ($L = 1$) udało się zredukować wartość oczekiwaną liczby cech.

Punktem wyjścia do przeprowadzonych w pracy badań było podanie prawidłowych wzorów na wartość oczekiwaną liczby cech używanych przez pracującą kaskadę. W przypadku drugiego wzoru wyprowadzona została także jego aproksymacja pozwalająca na obliczanie wspomnianej wartości w zadaniach, gdzie prawdopodobieństwo klasy pozytywnej jest nieznane, ale wiemy, że jest ono bardzo małe. Przykładem takich zadań jest detekcja obiektów na obrazie.

Praca ta zawiera także trzy przykłady geometryczne wskazujące przyczyny powstawania nieoptymalności w kaskadach uczonych przy ścisłym przestrzeganiu stałych wymagań na etap, tak jak ma to np. miejsce w algorytmie zaproponowanym przez Violę i Jonesa.

5.1 Kierunki dalszych badań

Kolejnym krokiem badawczym dla autora może być opracowanie nowego algorytmu przeszukiwania drzewa. W obecnej postaci algorytm swoim działaniem przypomina przeszukiwanie w głąb, co niesie za sobą szereg konsekwencji mogących negatywnie wpłynąć na czas nauki. Po pierwsze, niezależnie od tego, jak duża będzie wartości oczekiwana dla kaskady w skrajnej lewej gałęzi, to musi ona zostać nauczona w całości. Po drugie, jeśli wartość oczekiwana będzie malała z gałęzi na gałąź (zaczynając od lewej strony i kierując się w prawo), to algorytm przycinania nie zadziała i wszystkie kaskady zawarte w drzewie zostaną nauczone (przy czym zaistnienie takiej sytuacji jest mało realne). Rozwiązaniem powyższego problemu może być zastosowanie techniki przeszukiwania opartej o algorytm *best-first search*, gdzie jako kryterium pozwalające na wybór kolejnego węzła posłużyć może wartość oczekiwana. Wraz z nowym algorytmem przycinania rozwiązanie takie powinno wyeliminować wymienione wady.



Rysunek 5.1: Zarys pomysłu: uczenie kaskad oparte o algorytm best first search, wraz z koncepcją przycinania drzewa. (Źródło: opracowanie własne)

Rysunek 5.1 przedstawia zarys wspomnianego pomysłu. Węzły znajdujące się za przerywanymi liniami przedstawiają etapy kaskady, które dzięki algorytmowi przycinania nigdy nie zostałyby nauczone (jedyny wyjątek stanowią dwa ostatnie węzły pierwszego etapu, zgodnie ze specyfiką obecnego algorytmu uczenia w przypadku wystąpienia rozwidlenia uczone są wszystkie węzły prowadzące do analizowanego rodzica). Strzałki w powyższym drzewie określają kolejność, wg której wznawiana jest nauka kolejnych etapów zawartych w nim kaskad, dla którego węzła zostaną utworzeni jego potomkowie.

Spis tablic

Spis rysunków

Spis algorytmów

Bibliografia

Abbas, S. S. A. i in. (mar. 2017). "Crowd detection and management using cascade classifier on ARMv8 and OpenCV-Python". W: *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, s. 1–6.

Ahmad, F., Najam, A. i Ahmed, Z. (2013). "Image-based Face Detection and Recognition: "State of the Art"". W: *IJCSI International Journal of Computer Science Issues* 9.

Ai, W. I. i Langley, P. (1992). "Induction of One-Level Decision Trees". W: *Proceedings of the Ninth International Conference on Machine Learning*. Morgan Kaufmann, s. 233–240.

Bera, A., Klęsk, P. i Sychel, D. (2018). "Constant-Time Calculation of Zernike Moments for Detection with Rotational Invariance". W: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Bourdev, L. i Brandt, J. (2005). "Robust Object Detection via Soft Cascade". W: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*. CVPR '05. IEEE Computer Society, s. 236–243.

Brudno, A. L. (1963). "Bounds and Valuations for Abridging the Search of Estimates". W: *Problems of Cybernetics* 10.

Budiman, R. A. M. i in. (paź. 2016). "Localization of white blood cell images using Haar Cascade classifiers". W: *2016 1st International Conference on Biomedical Engineering (IBIOMED)*, s. 1–5.

Chen, B., Shen, J. i Sun, H. (2012). "A fast face recognition system on mobile phone". W: *2012 International Conference on Systems and Informatics (ICSAI2012)*, s. 1783–1786.

Chen, X., Liu, L. i in. (2019). "Vehicle detection based on visual attention mechanism and adaboost cascade classifier in intelligent transportation systems". W: *Optical and Quantum Electronics* 51.8.

Crow, F. (sty. 1984). "Summed-area Tables for Texture Mapping". W: *SIGGRAPH '84 Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. T. 18. 3. ACM, s. 207–212.

Cuimei, L. i in. (paź. 2017). "Human face detection algorithm via Haar cascade classifier combined with three additional classifiers". W: *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, s. 483–487.

Dalal, N. i Triggs, B. (2005). "Histograms of oriented gradients for human detection". W: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. T. 1, s. 886–893.

de Campos, T. E. i in. (2009). "Character recognition in natural images". W: *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, s. 273–280.

Edwards, D. J. i Hart, T. P. (1963). "The alpha-beta heuristic". W: *Technical Report*. MIT.

Fitriyani, N. L., Yang, C. i Syafrudin, M. (paź. 2016). "Real-time eye state detection system using haar cascade classifier and circular hough transform". W: *2016 IEEE 5th Global Conference on Consumer Electronics*, s. 1–3.

Freund, Y. i Schapire, R. E. (1996). "Experiments with a New Boosting Algorithm". W: *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*. T. 96, s. 148–156.

Freund, Y. i Schapire, R. E. (1999). "A Short Introduction to Boosting". W: *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, s. 1401–1406.

Friedman, J., Hastie, T. i Tibshirani, R. (1998). "Additive Logistic Regression: a Statistical View of Boosting". W: *Annals of Statistics* 28, s. 2000.

Gama, J. i Brazdil, P. (2000). "Cascade Generalization". W: *Machine Learning* 41.3, s. 315–343.

Jiang, X. i in. (2015). "Flexible sliding windows with adaptive pixel strides". W: *Signal Processing* 110, s. 37–45.

Jones, N. C. i Pevzner, P. A. (2002). *An Introduction to Bioinformatics Algorithms*. MIT Press.

Kearns, M. (1988). "Thoughts on Hypothesis Boosting". (niepublikowane).

Kim, J., Yoo, J. i Koo, J. (2018). "Road and Lane Detection Using Stereo Camera". W: *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, s. 649–652.

Klęsk, P., Bera, A. i Sychel, D. (2020). "Reduction of Numerical Errors in Zernike Invariants Computed via Complex-Valued Integral Images". W: *Computational Science – ICCS 2020*. Springer International Publishing.

Lawler, E. L. i Wood, D. E. (1966). "Branch-And-Bound Methods: A Survey". W: *Operations Research* 14.4, s. 699–719.

Lewis, J. P. (mar. 1995). "Fast Template Matching". W: *Vision Interface* 95, s. 120–123.

Li, C., Li, Y. i in. (2007). "Moving Human Body Detection in Video Sequences". W: *2007 International Conference on Machine Learning and Cybernetics*. T. 4, s. 2188-2–192.

Li, J. i Zhang, Y. (2013). "Learning SURF Cascade for Fast and Accurate Object Detection". W: *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '13. IEEE Computer Society, s. 3468–3475.

Li, L., Lu, W. i in. (maj 2016). "A frequency domain feature based cascade classifier and its application to fault diagnosis". W: *2016 Chinese Control and Decision Conference (CCDC)*, s. 5957–5961.

Li, Y., Xu, X. i in. (czer. 2016). "Eye-gaze tracking system by haar cascade classifier". W: *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*, s. 564–567.

Lu, J. i in. (2018). "Detection of Bird's Nest in High Power Lines in the Vicinity of Remote Campus Based on Combination Features and Cascade Classifier". W: *IEEE Access* 6, s. 39063–39071.

Martynow, M. i in. (2019). "Pupil detection supported by Haar feature based cascade classifier for two-photon vision examinations". W: *2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA)*, s. 54–59.

Momin, B. F. i Mujawar, T. M. (2015). "Vehicle detection and attribute based search of vehicles in video surveillance system". W: *2015 International Conference on Circuits, Power and Computing Technologies (ICCPCT-2015)*.

Morrison, D. R. i in. (2016). "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning". W: *Discrete Optimization* 19, s. 79–102.

Newell, A. i Simon, H. A. (mar. 1976). "Computer Science as Empirical Inquiry: Symbols and Search". W: *Communications of the ACM* 19.3, s. 113–126.

Papageorgiou, C. i Poggio, T. (2000). "A Trainable System for Object Detection". W: *International Journal of Computer Vision* 38, s. 15–33.

Papageorgiou, C. P., Oren, M. i Poggio, T. (1998). "A general framework for object detection". W: *Sixth International Conference on Computer Vision*, s. 555–562.

Pham, M. i Cham, T. (2007). "Fast training and selection of Haar features using statistics in boosting-based face detection". W: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, s. 1–7.

Rasolzadeh, B. i in. (2006). "Response Binning: Improved Weak Classifiers for Boosting". W: *IEEE Intelligent Vehicles Symposium*, s. 344–349.

Saberian, M. i Vasconcelos, N. (2014). "Boosting Algorithms for Detector Cascade Learning". W: *Journal of Machine Learning Research* 15, s. 2569–2605.

Samuel, A. L. (1959). "Some Studies in Machine Learning Using the Game of Checkers". W: *IBM Journal of Research and Development* 3.3, s. 210–229.

Samuel, A. L. (1967). "Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress". W: *IBM Journal of Research and Development* 11.6, s. 601–617.

Schapire, R. E. (1990). "The Strength of Weak Learnability". W: *Machine Learning* 5.2, s. 197–227.

Setjo, C. H., Achmad, B. i Djoko, F. (sierp. 2017). "Thermal image human detection using Haar-cascade classifier". W: *2017 7th International Annual Engineering Seminar (InAES)*, s. 1–6.

Shen, C., Wang, P. i Hengel, A. van den (2010). "Optimally Training a Cascade Classifier". W: *arXiv: Computer Vision and Pattern Recognition*.

Shen, C., Wang, P., Paisitkriangkrai, S. i in. (2013). "Training Effective Node Classifiers for Cascade Classification". W: *International Journal of Computer Vision* 103.3, s. 326–347.

Sung, K. i Poggio, T. (1998). "Example-based learning for view-based human face detection". W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.1, s. 39–51.

Sychel, D., Klęsk, P. i Bera, A. (2018). "Notes on Expected Computational Cost of Classifiers Cascade: A Geometric View". W: *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM*. SciTePress.

Sychel, D., Klęsk, P. i Bera, A. (2020a). "Branch-and-Bound Search for Training Cascades of Classifiers". W: *Computational Science – ICCS 2020*. Springer International Publishing.

Sychel, D., Klęsk, P. i Bera, A. (2020b). "Relaxed Per-Stage Requirements for Training Cascades of Classifiers". W: *Frontiers in Artificial Intelligence and Applications – ECAI 2020*. T. 325. IOS Press, s. 1523–1530.

University of Essex (1997). *Face Recognition Data*. `https://cswww.essex.ac.uk/mv/allfaces/faces96.html`. [dostęp 11-maj-2019].

Vallez, N., Deniz, O. i Bueno, G. (2015). "Sample selection for training cascade detectors". W: *PLOS One* 10 (7).

Viola, P. i Jones, M. (2001). "Rapid Object Detection using a Boosted Cascade of Simple Features". W: *Conference on Computer Vision and Pattern Recognition (CVPR'2001)*. IEEE, s. 511–518.

Viola, P. i Jones, M. (2004). "Robust Real-time Face Detection". W: *International Journal of Computer Vision* 57.2, s. 137–154.

Wang, Y. i in. (2019). "Woven Fabric Defect Detection Based on the Cascade Classifier". W: *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, s. 1–5.

Wojek, C. i in. (2008). "Sliding-Windows for Rapid Object Class Localization: A Parallel Technique". W: *Pattern Recognition*. Springer, s. 71–81.