POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Praca inżynierska

Narzędzie do ekstrakcji cech głębokich za pomocą konwolucyjnych sieci neuronowych

autor: Mikołaj Habarta

kierujący pracą: dr hab. inż. Michał Kawulok

Gliwice, styczeń 2021

załącznik nr 2 do zarz. nr 97/08/09

Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej*.

Gliwice, dnia 29 stycznia 2021

.....
(podpis)

.....
(poświadczenie wiarygodności
podpisu przez Dziekanat)

* podkreślić właściwe

Oświadczenie promotora

Oświadczam, że praca „Narzędzie do ekstrakcji cech głębokich za pomocą konwolucyjnych sieci neuronowych” spełnia wymagania formalne pracy dyplomowej inżynierskiej.

Gliwice, dnia 29 stycznia 2021

.....

(podpis promotora)

Spis treści

Rozdział 1

Wstęp

Na przestrzeni ostatniej dekady można zaobserwować gwałtowny rozwój dziedzin z zakresu uczenia maszynowego oraz sieci neuronowych. Pomimo pozornej nowości tych technologii, podstawy teoretyczne wielu z nich zostały opracowane już w latach latach 40. zeszłego stulecia [1]. Idee te były suckesywnie rozwijane oraz modyfikowane, lecz ograniczenia sprzętowe oraz trudność w dostępie do danych uniemożliwiały ich realne wykorzystanie. Dopiero na początku zeszłej dekady postępująca cyfyzacja oraz digitalizacja spowodowała znaczny wzorst ilości przechowywanych danych oraz ich większą dostępność. W tabeli 1.1 pokazano, jak zmieniały się rozmiary wybranych zbiorów danych przeznaczonych do zagadnień związanych z rozponawaniem rysów twarzy na przestrzeni lat. Łatwo zauważyć szybko zwiększające się rozmiary kolejnych baz danych, ze szczególnie gwałtownym wzrostem pomiędzy 2008 a 2014 rokiem. Dzięki dostępności coraz to większych zbiorów danych, ciągle rosnącej mocy obliczeniowej komputerów, oraz technologiach takich jak CUDA (*Compute Unified Device Architecture*), które umożliwiają łatwe wykorzystanie tej mocy, systemy oparte na sztucznej inteligencjii osiągają coraz to lepsze wyniki i są w stanie wykonywać pewne zadania lepiej niż człowiek.

W ostatnich latach można zaobserwować zwiększajacy się wpływ tych systemów na ludzkie życie w wielu różnych dziedzinach, takich jak np. diagnostyce chorób, samo-prowadzących się pojazdach, cyberbezpieczeństwie, czy marketingu. Stosunkowo niedawne odkrycia[2], [3] sugerują, że sztuczna inteligencja może być w stanie odciążyć specjalistów w dziedzinie diagnostyki chorób nowotworowych,

Nazwa	Rok powstania	Ilość obrazów
Yale Face Database[5]	1997	165
JAFFE Facial Expression Database[6]	1998	213
Face Recognition Grand Challenge Dataset[7]	2004	4007
CASIA 3D Face Database[8]	2007	4624
Bosphorus[9]	2008	4652
FaceScrub[10]	2014	107818
IMDB-WIKI[11]	2015	523051
Aff-Wild [12]	2017	$\sim$ 1,250,000
Aff-Wild2 [13]	2019	$\sim$ 2,800,000

Tablica 1.1: Rozmiary zbiorów danych służących do rozpoznawania twarzy na przestrzeni lat

a w przyszłości nawet w pewnym stopniu ich zastąpić. Warto tu również przytoczyć najnowszy przykład AlphaFold, systemu stworzonego przez Google, opartego o uczenie głębokie, który w październiku 2020 roku rozwiązał jedną z największych zagadek biologi[4]. Program nauczył się przewidywać trójwymiarową budowę białka na podstawie jego sekwencji aminokwasów, co było wyzwaniem dla biologów od 50 lat. To odkrycie pozwoliło również przyspieszyć pracę nad powstawaniem szczepionki na COVID-19.

Te dotychczasowe osiągnięcia systemów opartych o sztuczną inteligencję oraz potencjał ten dziedziny pozwala przypuszczać, że ich znaczenie w świecie będzie już tylko rosnąć.

1.1 Cel pracy

Celem pracy jest stworzenie uniwersalnego narzędzia, które ma umożliwić ekstrakcje wektorów cech głębokich w postaci serializowanej wraz z przypisanymi do nich etykietami w wybranym przez użytkownika formacie. Ekstrakcja jest dokonywana za pomocą konwolucyjnych sieci neuronowych służących do detekcji obiektów. Narzędzie powinno mieć możliwość wyboru architektury sieci, jak i dodania własnych architektur. Domyślną architekturą systemu, która zostanie zaimplementowana będzie architektura R-CNN. Narzędzie ma mieć możliwość użycia własnego zestawu danych w formacie PASCAL-VOC.

1.2 Zakres pracy

Zakres pracy obejmuje zgłębienie dziedziny wizji komputerowej oraz przegląd literatury technicznej. Kolejnym krokiem jest zrozumienie konwolucyjnych sieci neuronowych oraz modeli ich wykorzystujących do detekcji obiektów w obrazach, a następnie zapoznanie się bazą danych PASCAL-VOC oraz formatem przechowywanych tam danych. Kolejnym etapem jest przegląd oraz wybór odpowiedniej technologii, a następnie spisanie wymagań pracy oraz implementacja.

1.3 Plan pracy

Praca składa sie z 7 rozdziałów, które opisują teoretyczne oraz praktyczne ujęcie tematu.

Rozdział 1 zawiera wstęp do tematu oraz określenie celów projektu.

Rozdział 2 składa się z analizy zagadnienia detekcji obiektów w obrazach, przeglądu i porównanie dotychczas znanych rozwiązań i technologii.

W rozdziale 3 omówiono wymagania funkcjonalne i niefunkcjonalne oraz dokonano opisu zastosowanych narzędzi.

Rozdział 4 obejmuje specyfikacje zewnętrzną. Zostajie w nim opisany sposób instalacji oraz przykładowe scenariusze korzystania z narzędzia.

W rozdziale 5 można znaleźć opis architektury systemu oraz omówienie użytych modułów i bibliotek.

Rozdział 6 zawiera opis weryfikacji oraz walidacji systemu.

W rozdziale 7 zawarto podsumowanie całej pracy oraz wnioski z niej płynące. Wymieniono również największe trudności, które napotkano w czasie pracy nad projektem.

Rozdział 2

Analiza dziedziny

W tym rozdziale zostanie omówiony problem detekcji oraz klasyfikacji obiektów w obrazach. Pokrótce wyjaśniona zostanie zasada działania konwolucyjnych sieci neuronowych, ze zwięzłym opisem różnych rodzajów warstw, a następnie przedstawione zostanie kilka najważniejszych modeli sieci neuronowych. Opisana zostanie architektura R-CNN, która została zaimplementowana w programia, oraz algorytm wyszukiwania selektywnego, który również został zaimplementowany w ramach tej architektury. Aby móc uzyskać jakieś porównanie co do wydajności i ograniczeń architektury R-CNN, pokazane zostaną również inne architektury sieci, takie jak Fast R-CNN czy YOLO.

2.1 Analiza problemu

Człowiek postrzega świat głównie wizualnie. Szacuje się, że 80 % bodźców odbieranych przez człowieka to bodźce wzrokowe. Niektóre z teorii [14] pozwają przypuszczać, że wykształcenie oka było najważniejszym momentem w historii ewolucji oraz kluczowym elementem, który umożliwił powstanie inteligentnych form życia. Nic więc dziwnego, że temat tak znaczący dla człowieka otrzymuje proporcjonalnie dużo uwagi w dziedzinie sztucznej inteligencji. Umożliwinie maszynom zrozumienia wizualnych danych jest głównym celem, do którego spełnienia jesteśmy, zdawaćby się mogło, coraz bliżej.

Jednym z podstawowych problemów z dziedziny wizji komputerowej jest kla-

syfikacja. Polega ona na przypisaniu pewnej kategorii na podstawie obrazu. Zazwyczaj kategorie te to obiekty znajdujące się na zdjęciu. Chcemy więc, aby maszyna po zobaczeniu zdjęcia zawierającego jakiś obiekt psa skategoryzowała go jako 'pies'. Do problemu klasyfikacji możemy dołożyć jeszcze inny problem – detekcji. W ramach tego problemu oczekujemy, aby maszyna bo zobaczeniu jakiegoś zdjęcia zidentyfkowała wszystkie obiekty, które się na nim znajdują, oraz wskazała w którym miejscu na zdjęciu te obiekty się znajdują.

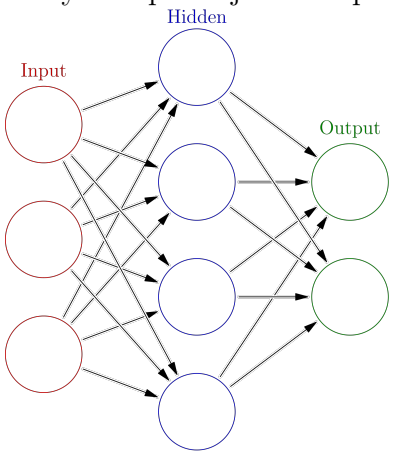
2.2 Sztuczne sieci neuronowe

Podstawą budowy sieci neuronowej jest węzeł, nazywany też czasem neuronem. Każdy węzeł ma swoje parametry – wagi. Każdy neuron przyjmuje pewne dane wejściowe oraz wytwarza dane wyjściowe, obydwa o stałym rozmiarze. Nauka sieci neuronowej polega na dobraniu odpowiednich parametrów za pomocą propagacji wstecznej dla każdego z neuronów tak, aby sieć na wyjściu zwracała oczekiwany rezultat. Neurony grupuje się w warstwy, które łączy się ze sobą. Na rysunku 2.1 przedstawiono prosty model sieci neuronowej, w którym każdy neuron jest połączony ze wszystkimi neuronami z następnej warstwy. Taki rodzaj warstw nazywa się warstwą w pełni połączoną, lub gęstą, a sieci stworzone z takich warstw sztucznymi sieciami neuronowymi. Taki model sieci otrzymuje dane wejściowe o stałym rozmiarze oraz produkuje dane wyjściowe o stałym rozmiarze. W przypadku problemu klasyfikacji danymi wejściowymi jest obraz, a danymi wyjściowymi – klasa obiektu. Ponieważ na wyjściu sieci otrzymujemy liczbę (wektor), to stosuje się kodowanie 1 z n, aby zamienić otrzymany wynik na odpowiednią klasę.

2.2.1 Konwolucyjne sieci neuronowe

Sztuczne sieci neuronowe dominowały w początkowych latach badań, jednak wraz z rozwojem dziedziny opracowano inne modele sieci, które miały służyć już bardziej konkretnym zadaniom. Modelem, który został stworzony do analizowania scen wizualnych był model konwolucyjny, nazywany też splotowym. Inspiracją do stworzenia tego modelu były odkrycia neurofizjologów Hubela i Wiesela z lat 50. i 60. zeszłego wieku. [15][16][17] Odkryli oni, że neurony w korze wzrokowej reagują

Rysunek 2.1: Przykład prostej sieci w pełni połączonej

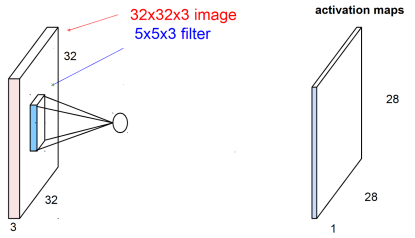


na określone pole widzenia. Każdy neuron ma swoje pole recepcyjne i reaguje na bodziec tylko w obrębie tego pola. Sieci konwolucyjne starają się odzworować sposób działania tych neuronów w korze wzrokowej. Zamiast patrzeć na obraz jako całość, każdy neuron jest odpowiedzialny za jego małą część. Neurony posiadająca pole recepcyjne nazywa się kernalami, albo filtrami, gdyż działają dokładnie jak klasyczne filtry, a warstwę filtrów nazywa się warstwą konwolucyjną. Sieci konwolucyjne składają się zazwyczaj z wielu warstw konwolucyjnych, przeplatanych innymi warstwami (np. próbkującymi), a na ich końcach umieszcza się jedną lub kilka warst w pełni połączonych. Zadaniem tych warstw w pełni połączonych jest dokonanie klasyfikacji na podstawie wyjścia z ostatniej warsty konwolucyjnej. To właśnie wyjście z ostatniej warstwy konwolucyjnej nazywane jest wektorem cech głębokich. Poniżej zostanie dokonany dokładniejszy opis warstwy konwolucyjnej, jak i również kilku innych rodzajów warstw, które są powszechnie używane w sieciach neuronowych.

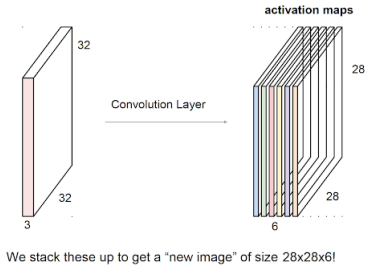
Warstwa konwolucyjna

Warstw konwolucyjna to zbiór kerneli (filtrów), zawierających parametry, które należy nauczyć. Kernele są zazwyczaj małych rozmiarów, mniejszych od rozmiaru obrazu wejściowego. Typowym rozmiarem kernela jest np. 3x3x3, który oznacza, że pokrywa on obszar 3 na 3 piksele, w kolorze (trzeci wymiar to kanały RGB).

Filtr jest następnie przesuwany przez cały obraz wejściowy, i w każdej jego pozycji obliczany jest iloczyn skalarny między nim a danymi wejściowymi (Rys. 2.2). W wyniku tego działania otrzymujemy pewną macierz, która nazywa się mapą aktywacji danego kernela. Mapy aktywacji wszystkich filtrów z danej warstwy są nakładane na siebie i tworzą trójwymiarowa macierz, która jest podawana na wyjściu warstwy konwolucyjnej, co pokazano na rysunku 2.3.



Rysunek 2.2: Filtr o wymiarach 5x5x3



Rysunek 2.3: Mapy aktywacji nakładane na siebie

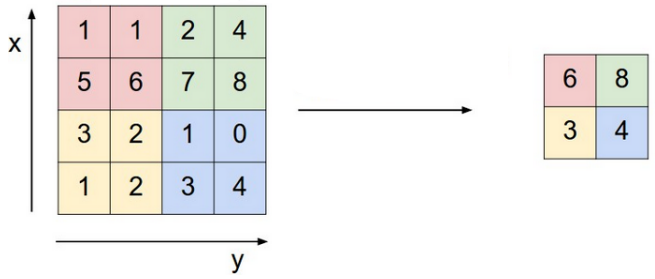
Warstwa próbkująca

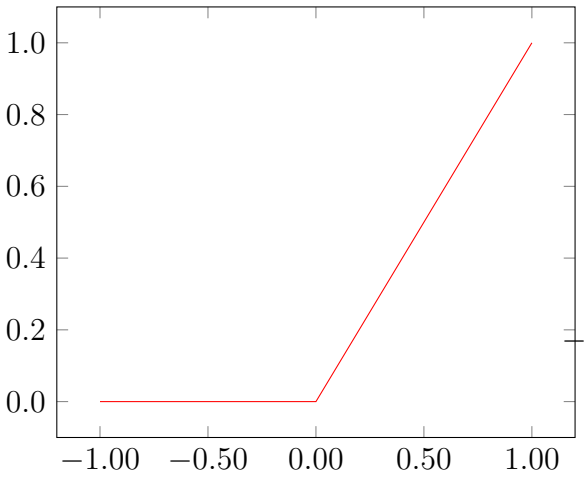
Nazywana też czasem warstwą łączenia, najcześciej umieszczana jest pomiędzy dwoma warstwami konwolucyjnymi. Jej zadaniem jest redukcja rozmiaru otrzymanych map aktywacji, co zmiejsza ilosć parametrów, których sieć musi się nauczyć. Ta warstwa opiera się u filtry najczęsciej o rozmiarach 2x2, które biorą maksymalną lub średnią wartość z każdego rejonu (rys. 2.4) i zmniejszają w ten sposób wysokość oraz szerokość danych, nie zmieniając jednak głębokości.

Warstwa aktywacyjne

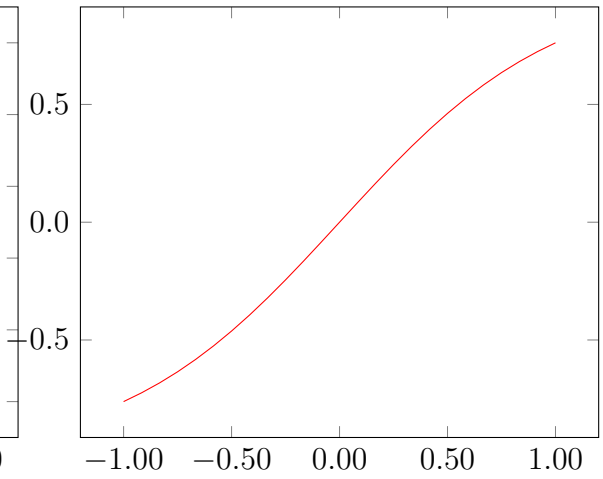
Ta warstwa to funkcja matematyczna, która decyduje o tym, czy neuron ma być aktywny, czy nie, na podstawie jego wartości. Powinna być to funkcja szybka do obliczenia, bo będzie wykonywana dla każdego nueronu w sieci. Początkowo często używaną funkcją był $\tanh(x)$ (rys. 2.6) , ale z czasem okazało się że fukncja rektyfikowanej jednostki liniowej (ReLu), definiowanej jako $\max(0, x)$(rys. 2.5) pozwala osiagnąć lepsze wyniki [18][19] i aktualnie jest najczęściej używaną funkcją aktywacji [20].

Rysunek 2.4: Max pooling





Rysunek 2.5: ReLu.



Rysunek 2.6: Tangens hiperboliczny.

Warstwa normalizujące

Warstwa ta została zaproponowana w celu zmiejszenia złożoności obliczeniowej poprzez normalizację aktywacji neuronów [21], jednak doświadczenia praktyczne sugerują, że ich wpływ jest znikomy, przez co stosowane są bardzo rzadko i tylko w kontretnych przypadkach.

Warstwa regularyzacji opuszczeń

Warstwa ta w sposób losowy wyłącza pewną część neuronów (najczęściej 50%), poprzez ustawienie ich wartości na 0, co sprawia, że nie będą aktywne. Może wydawać się to nieintuicyjne, jednak ta technika sprawia, że sieć musi być bardziej

elastyczna i nie może zawsze polegać na istniejących już połączeniach. Pozwala do zapobiegać nadmiernemu dopsowaniu sieci oraz sprawia, że sieć osiąga lepsze rezultaty[22][23][24].

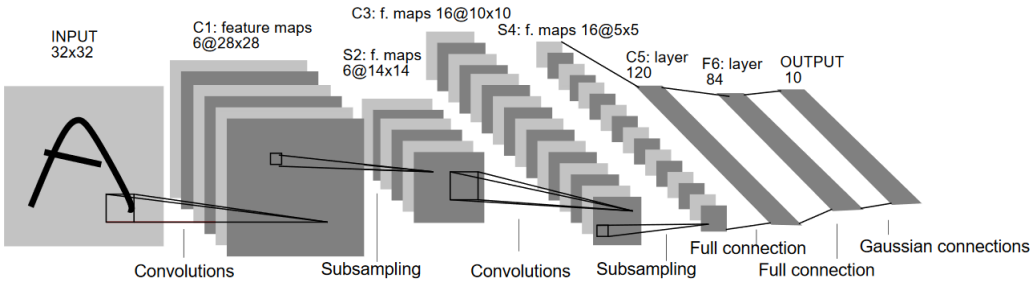
2.2.2 Przykladowe modele sieci

Na przestrzeni lat pojawiło się kilka modeli sieci neuronowych, które, czy to ze względu na swoją innowacyjność, czy na uzyskiwane wyniki, miały wielki wpływ na rozwój dziedziny i są powszechnie znane w środowiskach naukowach.

LeNet[25]

Jest to pierwsza udana implementacja konwolucyjnej sieci neuronowej. Stworzona w latach 90. przez Yanna LeCuna służyła do rozpoznawania ręcznie pisanych cyfr z kodów pocztowych. Składała się z 3 warstw konwolucyjnych na przemian z warstwami próbkującymi, oraz z jednej warstwy w pełni połączonej na samym końcu, co pokazano na rysunku 2.7.

Rysunek 2.7: Architektura sieci LeNet

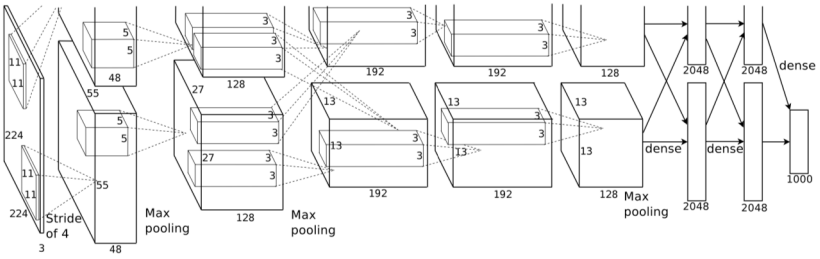


AlexNet[26]

Sieci konwolucyjne zyskały popularność w latach 90., jednak wymagały dużej mocy obliczeniowej, które przy ówczesnym poziomie techniki były trudno dostępne (warto przypomnieć, że technologia CUDA powstała dopiero w 2007 roku), przez co wypadły z łask na rzecz maszyn wektorów wspierajacych[27]. Sytuacja ta

utrzymywała się aż to 2012 roku, kiedy to Alex Krizhevsky i in. stworzyli sieć AlexNet. Sieć osiągnęła najlepszy rezultat w konkursie ILSVRC, z błędem na poziomie 15,3%, ponad 10 punktów procentowych lepiej od drugiego miejsca. Ten świetny rezultat na nowo pobudził zainteresowanie technologią sieci konwolucyjnych, a sieć ta jest uważana za jedną z najbardziej wpływowych w dziedzinie wizji komputerowej. Używa ona ReLu jako funkcji aktywacji, co nie było standardem w tym czasie. Zastosowana została warstwa regularyzacji opuszczeń z prawdopodobieństwem 50%, jak i również augmentacja danych, co zmniejszyło nadmierne dopasowanie, a użycie procesorów graficznych pozwoliło na szybsze wykonanie kosztownych obliczeń. Na rysunku 2.8 pokazano architekturę sieci AlexNet.

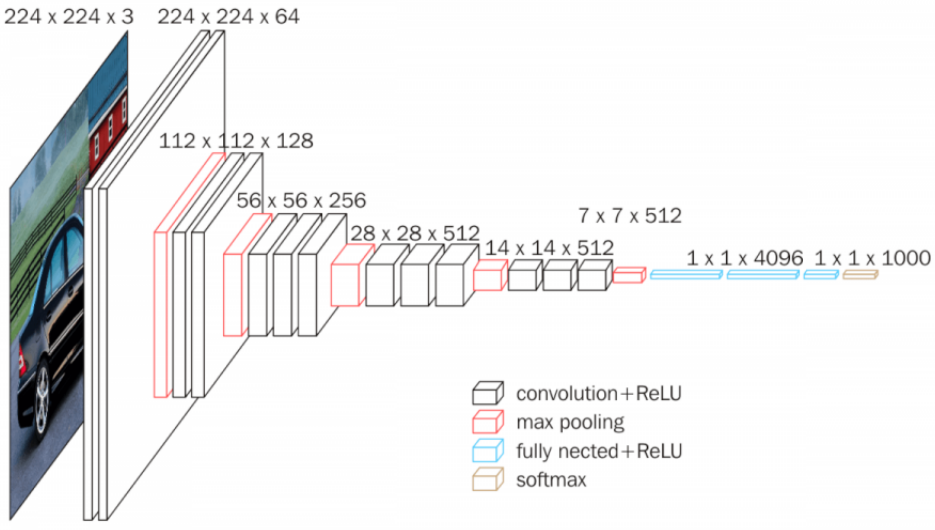
Rysunek 2.8: Architektura sieci AlexNet. Sieć została tutaj podzielona na dwie równoległe części, ponieważ obliczenia były dzielone pomiędzy dwie jednostki graficzne.



VGG16[28]

Stworzona w 2014 roku przez Simonyana i Zissemana, pomimo tego, że zajęła 2 miejsce w konkursie ILSVRC, jest jedną z najbardziej rozpoznawalnych sieci w dziedzinie. Simonyan i Zisseman przyjęli inną strategi – zamiast zmieniać i testować różne wielkości warstw, użyli w niej warst konwolucyjnych o stałym wymiarze 3x3 oraz próbkujących o rozmiarze 2x2, a testowali jedynie różne głębokości sieci. W ramach pracy stworzono kilka wariantów sieci o różnej głębokości. Najlepszy okazał się wariant z 16 warstwami (rys. 2.9). Ten model pokazał, że głębokość sieci jest kluczowym czynnikiem decydującym o dobrym rezultacie.

Rysunek 2.9: Architektura sieci VGG



ResNet[29]

Sieć ta stworzona została przez Kaiminga He i in. i wygrała konkurs ILSVRC w 2015 roku. Jest przykładem sieci szczątkowej, w której niektóre połącznia pomiędzy warstwami są pomijane. Takie rozwiązanie pozwala na lepszą skalowalość wraz ze zwiększaniem liczby warstw oraz eliminuje problem tzw. zanikającego gradientu. Sieć ta posiada olbrzymią liczbę 152 warstw, i ta glebokość w połączeniu z nową technologią sprawiła, że przez długi czas była ona szczytowym osiągnięciem technologii.

2.3 R-CNN

Zastosowanie konwolucyjnych sieci neuronowych w problemie klasyfikacji jest stosunkowo proste, ponieważ wymiary danych wejściowych oraz wyjściowych są stałe. W przypadku detekcji jednak pojawia się problem, ponieważ liczba obiektów na zdjęciach może być różna, więc nasza sieć musiałaby dawać wyniki o zmiennych rozmiarach, co jest sprzeczne z jej zasadą działania. Początkowym rozwiązaniem było użycie okna, które przesuwało się po obrazie w każdej pozycji oraz klasyfikowało zaznaczony obszar [30]. Okno musiało sprawdzić każdą możliwą lokalizację, dodatkowo musiało zmieniać rozmiar, co skutkowało ogromną ilością obliczeń.

W celu rozwiązania tego problemu Ross Girshick i in. zaproponowali w 2014 roku rozwiązanie – regionalną konwolucyjną sieć neuronową[27]. Rozwiązanie to zakłada, że najpierw z obrazu wydzielamy propozycje około 2000 regionów, w których jest duże prawdopodobieństwo, że znajduje się jakiś obiekt. Do wydzielania tych regionów, nazywanych też regionami zainteresowań (*RoI - Regions of Interest*), służy algorytm selektywnej selekcji, opisany dokładniej w podrozdziale 2.3.1. Następnie każdy z tych regionów służy jako dane wejściowe do konwolucyjnej sieci neuronowej, która wyznacza dla niego wektor cech głębokich. Następnie te wektory poddawane są klasyfikacji. W pierwotnej wersji klasyfikacja ta była przeprowadzana za pomocą maszyny wektorów wspierających, ale można używać innych metod w celu poprawy dokładności sieci.

Podczas detekcji musimy zmierzyć się z jeszcze jednym zadaniem – musimy zlokalizować nasz obiekt na zdjeciu. Jako lokalizację przyjmuje się wyznaczenie prostokątu, w którym znajduje się obiekt. Pojawia się więc problem, w jaki sposób ocenić, czy wyznaczony przez nas obszar pokrywa się z prostokątem zawierającym obiekt. Nie możemy oczekiwać, że wyznaczymy dokładnie identyczny obszar, ponieważ byłoby to wręcz niemożliwe. Wystarczy nam, że nasz obszar tylko w pewnym stopniu będzie pokrywał prostokąt. W celu ewaluacji tej miary używa się operatora [IoU] (przecięcie nad połączeniem), który używa współczynnika matematycznego, zwanego indeksem Jaccarda, definiowanego jako:

$$F(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

Czyli iloraz cześci wspólnej oraz sumy obu obszarów. Próg, od którego uznajemy, że wyznaczony obszar zawiera obiekt jest wyznaczny umownie (zazwyczaj jest to 0.5), a jego zmiana może znacząco wpływać na wynik sieci [27]. Obszary z miarą IoU powyżej tego progu są uznawane za próbki dodatnie, ponieważ znajdują się w nich jakieś obiekty, a poniżej tego progu – ujemne, czyli zawierające tło.

2.3.1 Algorytm wyszukiwania selektywnego

Algorytm ten[31] ma za zadanie wyznaczyć propozycje regionów, które będą później używane do detekcji obiektów. Na początku algorytm dokonuje segmentuje obraz na podstawie intensywności pikseli, bazując na zaproponowanej przez

Felzenszwalba i in. metodzie segmentacji z zastosowaniem teorii grafów [32]. Następnie obszary, które są do siebie podobne, są ze sobą łączone. Podobieństwo obszarów określa się na podstawie 4 cech: koloru, tekstury, rozmiaru i kształtu.

Podobieństwo koloru

Dla każdego regionu generowany jest histogram danego kanału barwy. Wszystkie kanały są następnie zestawiane razem w wektor o określonej długości n, a podobieństwo jest wyliczane według wzoru:

$$P_{koloru}(r_i, r_j) = \sum_{k=1}^{n} min(c_i^k, c_j^k) \tag{2.2}$$

Gdzie c_i^k, c_i^k są wartością k-tego przedziału histogramu dla regionów kolejno: r_i i r_j.

Podobieństwo tekstury

Dla każdego kanału liczonych jest 8 pochodnych Gaussa przy $\sigma = 1$. Na ich podstawie dla każdego regionu tworzony jest histogram, a podobieństwo tekstur jest liczone jako:

$$P_{tekstury}(r_i, r_j) = \sum_{k=1}^{n} min(t_i^k, t_j^k) \tag{2.3}$$

Gdzie t_i^k, t_i^k są wartością k-tego przedziału histogramu dla regionów kolejno: r_i i r_j.

Podobieństwo rozmiaru

To podobieństwo ma zachęcać mniejsze regiony do łączenia się ze sobą, jednocześnie pozwala unikać sytuacji, w której jeden region wchłania wszystkie inne. Dla obrazu o rozmiarze w pikselach $size(im)$ jest ono liczone jako:

$$P_{rozmiaru}(r_i, r_j) = 1 - \frac{size(r_i) + size(r_j)}{size(im)} \tag{2.4}$$

Podobieństwo kształu

Określa jak bardzo dwa regiony do siebie pasują. Jest zdefiniowane jako:

$$P_{kształtu}(r_i, r_j) = 1 - \frac{size(BB_{ij}) - size(r_i) + size(r_j)}{size(im)} \tag{2.5}$$

Gdzie BB_{ij} jest obwiednią dookoła regionów r_i i r_j.

Końcowe podobieństwo można ozyskać ze wzoru:

$$P(r_i, r_j) = a_1 P_{koloru} + a_2 P_{tekstury} + a_3 P_{rozmiaru} + a_4 P_{kształtu} \tag{2.6}$$

Gdzie $a_i \in \{0, 1\}$ określa, czy miara podobieństwa jest użyta.

2.3.2 Wady modelu R-CNN

Pomimo swojej użyteczności, model R-CNN nie jest pozbawiony wad. Konieczność wykonania obliczeń dla każdego z 2000 regionów sprawia, że model działa bardzo wolno, przez co wytrenowanie go może zająć duże ilości czasu. Dodatkowo nie może znaleźć on zastosowania w sytuacjach czasu rzeczywistego (np. analiza obrazu z kamery), przetworzenie każdego obrazu zajmuje średnio 40 sekund. Należy też zauważyć, że na etapie wyznaczania regionów nie następuje żadna nauka sieci – algorytm wyszukiwania selektywnego jest algorytmem stałym i niezależnym od parametrów sieci. Kolejne rozwiązania starają się rozwiązać te problemy.

2.4 Inne architektury

Fast R-CNN[33]

Ross Girshick rok po publikacji swojej pracy opisującej R-CNN zaoproponował jej ulepszoną wersję – Fast R-CNN. Jak sama nazwa wskazuje, rozwiązanie to jest szybsze od swojej pierwszej wersji. Zamiast wyznaczać dużą liczbę regionów, na których następnie sięc dokonuje obliczeń, w tym modelu wrzucamy wejściowy obraz do sieci konwolucyjnej, a dopiero na otrzymanej z sieci mapie aktywacji dokonujemy podziału na regiony, które następnie klasyfikujemy. Dzięki temu rozwiązaniu

znacząco zmniejszamy złożoność obliczeniową, co skutkuje 9-krotnym przyspieszeniem etapu treningu sieci oraz aż 213-krotnym przyspiesznym etapu testowania, przy jednoczesnym zwiększeniu precyzji sieci.

Faster R-CNN[34]

Wszystkie poprzednie metody używały algorytmu wyszukiwania selektywnego do wyznacznia propozycji regionów, jednak wraz ze wzrostem szybkości innych elementów modelu, ten algorytm stał się „wąskim gardłem" obliczeniowym, dlatego postanowiono z niego zrezygnować. W ten sposób powstała architekura Faster R-CNN, jeszcze szybsza od swoich poprzedniczek, w której propozycje regionów są wyznaczane również przez równoległą sieć neuronową. Dzięki temu udało się jeszcze bardziej przyspieszyć działanie sieci, do poziomu 200 ms na obraz.

YOLO[35]

Architektura YOLO (*You Only Look Once* – patrzy się tylko raz) stara się traktować problem detekcji jako problem regresji. Zamiast na cały obraz, sieć patrzy tylko na te fragmenty, w których jest bardziej prawdopodobne, że występuje jakiś obiekt. Wszystkie zadanie – lokalizację obiektów oraz klasyfikację wykonuje tylko jedna sieć, co drastycznie zwiększa szybkość tego rozwiązanie, pozwalając na zastosowanie go w celu detekcji w czasie rzeczywistym (z prędkością 45 klatek na sekundę). Wadą tego rozwiązania jest mniejsza precyzja sieci – często nie zauważa obiektów, szczególnie jeżeli są dość małe.

2.5 PASCAL VOC

PASCAL VOC[36][37] składa się z dwóch cześci – publicznie dostępnej bazy danych, zawierajacych prawie 20 tysięcy zdjęć wraz z odpowiadającymi im adnotacjami, oraz z corocznego konkursu, organizowanego od 2005 roku, w którym różne grupy badawcze mogą zgłosić swoje rozwiązanie i spróbować swoich sił w którymś z pięciu wyzwań: klasyfikacji, detekcji, segmentacji, klasyfikacji akcji oraz rozpoznania ułożenia ciała. Dzięki takim konkursom (jak również wcześniej wspominany ILSVRC) można dokonać przeglądu najnowocześniejszych rozwiązań w dziedzinie.

Dodatkowo udostępniane przez organizatorów dane są powszechnie dostępne, co pozwala każdemu zainteresowanemu dziedziną na wykorzystywanie ich we własnych projektach, tak jak właśnie zostana wykorzystane w ramach tej pracy.

Rozdział 3

Wymagania i narzędzia

W tym rozdziale zostanie dokonany opis wymagań funkcjonalnych oraz niefunkcjonalnych pracy, oraz zaprezentowane zostaną diagramy przypadkow użycia. Następnia omówiona zostana użyte narzędzia oraz opisana zostania metodyka pracy nad projektowaniem i implenetacją.

3.1 Wymaganie funkcjonalne i niefunkcjonalne

3.1.1 Wymagania funkcjonalne

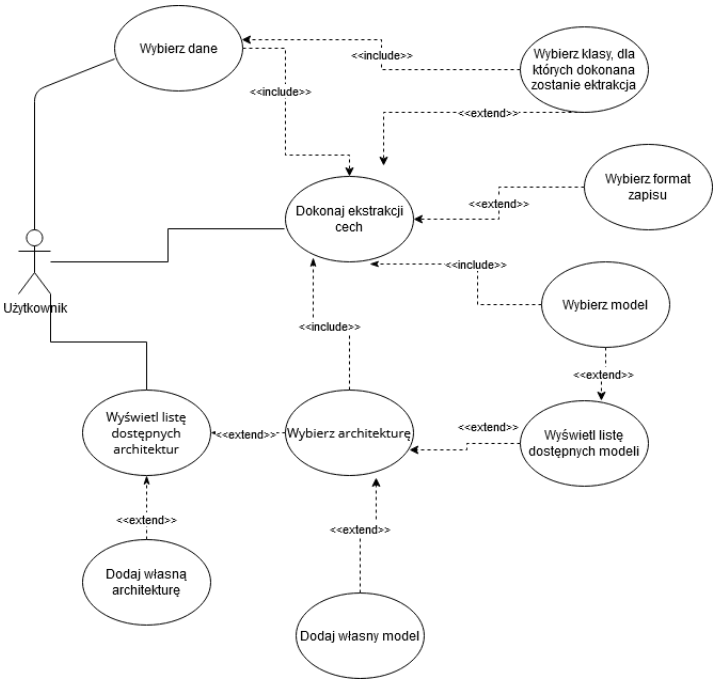
Projekt powinien realizować następujące funkcjonalności:

- Użytkownik powinien móc dokonać wyboru architektury sieci.
- Powinna istnieć możliwość łatwego dodania innych architektur sieci.
- W ramach architektury, użytkownik ma mieć możliwość wczytania swojego modelu z pliku, wyświetlenia listy dostępnych modeli oraz wyboru modelu sieci.
- Dla określonych danych wejściowych, program ma mieć możliwość zapisania cech głębokich ektrahowanych przez sieć, wraz z odpowiadającymi im etykietami. Poprzez cechy głębokie rozumie się ostatnią warstwę sieci konwolucyjnej, którą sieć oblicza przed dokonaniem klasyfikacji.

- Użytkownik ma mieć możliwość wyboru formatu, w którym zostaną zapisane (tekst lub hdf5).
- Powinna istnieć możliwość wyboru dowolnych danych wejściowych, zgodnych z formatem PASCAL VOC, i to na tych danych zostanie wykonana ekstrakcja.
- Powinna istnieć możliwość wybrania klas, dla których zostanie dokonana esktrakcja
- W ramach projektu zostanie zaimplementowana architektura R-CNN i będzie ona domyślnie używaną architekturą (jeżeli użytkownik nie wskaże innej).

3.1.2 Wymagania niefunkcjonalne

3.2 Diagram przypadków użycia

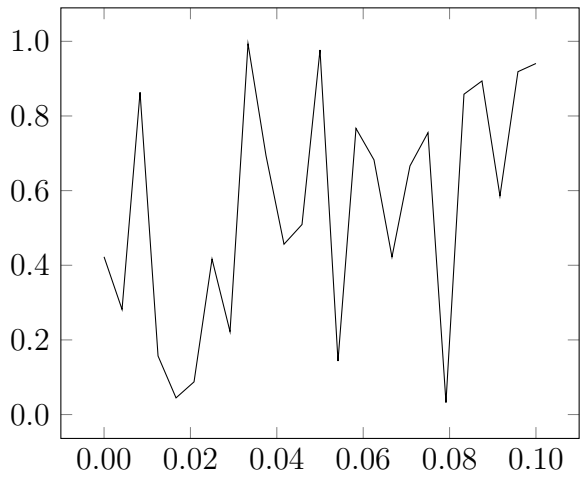


Rysunek 3.1: Diagram przypadków użycia

Rozdział 4

Specyfikacja zewnętrzna

- wymagania sprzętowe i programowe
- sposób instalacji
- sposób aktywacji
- kategorie użytkowników
- sposób obsługi
- administracja systemem
- kwestie bezpieczeństwa
- przykład działania
- scenariusze korzystania z systemu (ilustrowane zrzutami z ekranu lub generowanymi dokumentami)



Rysunek 4.1: Podpis rysunku po rysunkiem.

Rozdział 5

Specyfikacja wewnętrzna

- przedstawienie idei
- architektura systemu
- opis struktur danych (i organizacji baz danych)
- komponenty, moduły, biblioteki, przegląd ważniejszych klas (jeśli występują)
- przegląd ważniejszych algorytmów (jeśli występują)
- szczegóły implementacji wybranych fragmentów, zastosowane wzorce projektowe
- diagramy UML

Krótka wstawka kodu w linii tekstu jest możliwa, np. **descriptor**, a nawet **descriptor_gaussian**. Dłuższe fragmenty lepiej jest umieszczać jako rysunek, np. kod na rysunku 5.1, a naprawdę długie fragmenty – w załączniku.

```
1  class descriptor_gaussian : virtual public descriptor
2  {
3     protected:
4        /** core of the gaussian fuzzy set */
5        double _mean;
6        /** fuzzyfication of the gaussian fuzzy set */
7        double _stddev;
8
9     public:
10       /** @param mean core of the set
11           @param stddev standard deviation */
12       descriptor_gaussian (double mean, double stddev);
13       descriptor_gaussian (const descriptor_gaussian & w);
14       virtual ~descriptor_gaussian();
15       virtual descriptor * clone () const;
16
17       /** The method elaborates membership to the gaussian
             fuzzy set. */
18       virtual double getMembership (double x) const;
19
20 };
```

Rysunek 5.1: Klasa **descriptor_gaussian**.

Rozdział 6

Weryfikacja i walidacja

- sposób testowania w ramach pracy (np. odniesienie do modelu V)
- organizacja eksperymentów
- przypadki testowe zakres testowania (pełny/niepełny)
- wykryte i usunięte błędy
- opcjonalnie wyniki badań eksperymentalnych

Rozdział 7

Podsumowanie i wnioski

- uzyskane wyniki w świetle postawionych celów i zdefiniowanych wyżej wymagań
- kierunki ewentualnych danych prac (rozbudowa funkcjonalna ...)
- problemy napotkane w trakcie pracy

Tablica 7.1: Opis tabeli nad nią.

	metoda						
			alg. 3			alg. 4, $\gamma = 2$	
ζ	alg. 1	alg. 2	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$	$\beta = 0.1$	$\beta = -0.1$
0	8.3250	1.45305	7.5791	14.8517	20.0028	1.16396	1.1365
5	0.6111	2.27126	6.9952	13.8560	18.6064	1.18659	1.1630
10	11.6126	2.69218	6.2520	12.5202	16.8278	1.23180	1.2045
15	0.5665	2.95046	5.7753	11.4588	15.4837	1.25131	1.2614
20	15.8728	3.07225	5.3071	10.3935	13.8738	1.25307	1.2217
25	0.9791	3.19034	5.4575	9.9533	13.0721	1.27104	1.2640
30	2.0228	3.27474	5.7461	9.7164	12.2637	1.33404	1.3209
35	13.4210	3.36086	6.6735	10.0442	12.0270	1.35385	1.3059
40	13.2226	3.36420	7.7248	10.4495	12.0379	1.34919	1.2768
45	12.8445	3.47436	8.5539	10.8552	12.2773	1.42303	1.4362
50	12.9245	3.58228	9.2702	11.2183	12.3990	1.40922	1.3724

Bibliografia

[1] W. S. M. W. P. i in., "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, pp. 115–133, 1943.

[2] S. M. i in., "International evaluation of an ai system for breast cancer screening," *Nature*, vol. 577, no. 7788, p. 89–94, 2020.

[3] K. D. i in., "Effect of artificial intelligence-based triaging of breast cancer screening mammograms on cancer detection and radiologist workload: a retrospective simulation study," *The Lancet*, vol. 2, no. 9, pp. e468–e474, 2020.

[4] R. F. Service, "'the game has changed.'ai triumphs at protein folding," 2020.

[5] "The yale face database." `http://cvc.cs.yale.edu/cvc/projects/yalefaces/yalefaces.html`. [data dostępu: 2021-01-28].

[6] M. Lyons, S. Akamatsu, M. Kamachi, and J. Gyoba, "Coding facial expressions with gabor wavelets," in *Proceedings Third IEEE international conference on automatic face and gesture recognition*, pp. 200–205, IEEE, 1998.

[7] K. W. Bowyer, K. Chang, and P. Flynn, "A survey of approaches and challenges in 3d and multi-modal 3d+ 2d face recognition," *Computer vision and image understanding*, vol. 101, no. 1, pp. 1–15, 2006.

[8] "Bit face databases." `http://english.ia.cas.cn/db/201610/t20161026_169405.html`. [data dostępu: 2021-01-28].

[9] A. Savran, N. Alyüz, H. Dibeklioğlu, O. Çeliktutan, B. Gökberk, B. Sankur, and L. Akarun, "Bosphorus database for 3d face analysis," in *European workshop on biometrics and identity management*, pp. 47–56, Springer, 2008.

[10] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *2014 IEEE international conference on image processing (ICIP)*, pp. 343–347, IEEE, 2014.

[11] R. Rothe, R. Timofte, and L. V. Gool, "Dex: Deep expectation of apparent age from a single image," in *IEEE International Conference on Computer Vision Workshops (ICCVW)*, December 2015.

[12] S. Zafeiriou, D. Kollias, M. A. Nicolaou, A. Papaioannou, G. Zhao, and I. Kotsia, "Aff-wild: valence and arousal'in-the-wild'challenge," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 34–41, 2017.

[13] D. Kollias and S. Zafeiriou, "Expression, affect, action unit recognition: Aff-wild2, multi-task learning and arcface," *arXiv preprint arXiv:1910.04855*, 2019.

[14] D.-E. Nilsson, "Eye evolution and its functional basis," *Visual neuroscience*, vol. 30, no. 1-2, pp. 5–20, 2013.

[15] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[16] D. H. Hubel and T. Wiesel, "Shape and arrangement of columns in cat's striate cortex," *The Journal of physiology*, vol. 165, no. 3, pp. 559–568, 1963.

[17] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[18] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[19] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.

[20] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[21] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[23] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 8609–8613, IEEE, 2013.

[24] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[30] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," 2013.

[31] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[32] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[33] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[34] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[36] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[37] "The pascal visual object classes homepage." `http://host.robots.ox.ac.uk/pascal/VOC/`. [data dostępu: 2021-01-28].

Dodatki

Spis skrótów i symboli

RoI Region zainteresowań (ang. *Region of Interest*)

IoU Przecięcie nad połączeniem (ang. *Intersection over Union*)

CUDA ang. *Compute Unified Device Architecture*

ILSVRC TODO

DNA kwas deoksyrybonukleinowy (ang. *deoxyribonucleic acid*)

MVC model – widok – kontroler (ang. *model–view–controller*)

N liczebność zbioru danych

μ stopnień przyleżności do zbioru

$\mathbb{E}$ zbiór krawędzi grafu

$\mathcal{L}$ transformata Laplace'a

Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego, należy je przenieść do załącznika.

```
1  partition fcm_possibilistic::doPartition
2                                  (const dataset & ds)
3  {
4      try
5      {
6          if (_nClusters < 1)
7              throw std::string ("unknown number of clusters");
8          if (_nIterations < 1 and _epsilon < 0)
9              throw std::string ("You should set a maximal
                   number of iteration or minimal difference --
                   epsilon.");
10         if (_nIterations > 0 and _epsilon > 0)
11             throw std::string ("Both number of iterations and
                   minimal epsilon set -- you should set either
                   number of iterations or minimal epsilon.");
12
13         auto mX = ds.getMatrix();
14         std::size_t nAttr = ds.getNumberOfAttributes();
15         std::size_t nX    = ds.getNumberOfData();
16         std::vector<std::vector<double>> mV;
17         mU = std::vector<std::vector<double>> (_nClusters);
18         for (auto & u : mU)
```

```
19          u = std::vector<double> (nX);
20      randomise(mU);
21      normaliseByColumns(mU);
22      calculateEtas(_nClusters, nX, ds);
23      if (_nIterations > 0)
24      {
25          for (int iter = 0; iter < _nIterations; iter++)
26          {
27              mV = calculateClusterCentres(mU, mX);
28              mU = modifyPartitionMatrix (mV, mX);
29          }
30      }
31      else if (_epsilon > 0)
32      {
33          double frob;
34          do
35          {
36              mV = calculateClusterCentres(mU, mX);
37              auto mUnew = modifyPartitionMatrix (mV, mX);
38
39              frob = Frobenius_norm_of_difference (mU, mUnew)
                    ;
40              mU = mUnew;
41          } while (frob > _epsilon);
42      }
43      mV = calculateClusterCentres(mU, mX);
44      std::vector<std::vector<double>> mS =
            calculateClusterFuzzification(mU, mV, mX);
45
46      partition part;
47      for (int c = 0; c < _nClusters; c++)
48      {
49          cluster cl;
```

```
50            for (std::size_t a = 0; a < nAttr; a++)
51            {
52               descriptor_gaussian d (mV[c][a], mS[c][a]);
53               cl.addDescriptor(d);
54            }
55            part.addCluster(cl);
56         }
57         return part;
58      }
59      catch (my_exception & ex)
60      {
61         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               ex.what());
62      }
63      catch (std::exception & ex)
64      {
65         throw my_exceptionn (__FILE__, __FUNCTION__, __LINE__
               , ex.what());
66      }
67      catch (std::string & ex)
68      {
69         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               ex);
70      }
71      catch (...)
72      {
73         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               "unknown expection");
74      }
75   }
```

Zawartość dołączonej płyty

Do pracy dołączona jest płyta CD z następującą zawartością:

- praca (źródła LaTeXowe i końcowa wersja w `pdf`),
- źródła programu,
- dane testowe.

Spis rysunków

Spis tablic