



Politechnika
Śląska

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, luty 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 8 lutego 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 8 lutego 2021

.....
(podpis promotora)

Spis treści

1 Wstęp	1
1.1 Cel pracy	2
1.2 Zakres pracy	3
1.3 Plan pracy	3
2 Analiza dziedziny	5
2.1 Analiza problemu	5
2.2 Sztuczne sieci neuronowe	6
2.2.1 Konwolucyjne sieci neuronowe	6
2.2.2 Przykładowe modele sieci	10
2.3 R-CNN	12
2.3.1 IoU	13
2.3.2 Algorytm wyszukiwania selektywnego	14
2.3.3 Wady modelu R-CNN	15
2.4 Inne architektury	16
3 Wymagania i narzędzia	17
3.1 Wymaganie funkcjonalne i niefunkcjonalne	17
3.1.1 Wymagania funkcjonalne	17
3.1.2 Wymagania niefunkcjonalne	18
3.2 Diagram przypadków użycia	18
3.3 Opis narzędzi	18
3.3.1 Python	18
3.3.2 Keras i Tensorflow	19
3.3.3 Skimage i Imageio	19

3.3.4	Abstract Base Classes	20
3.3.5	Tkinter	20
3.3.6	PASCAL VOC	20
3.4	Metodyka pracy nad projektowaniem oraz implementacją	20
4	Specyfikacja zewnętrzna	23
4.1	Wymagania sprzętowe i programowe	23
4.1.1	Python	23
4.1.2	Tensorflow	23
4.2	Sposób instalacji	24
4.2.1	Python	24
4.2.2	Instalacja narzędzia	24
4.3	Sposób aktywacji	24
4.4	Przykład działania	25
4.4.1	Wybór architektury	25
4.4.2	Wybór modelu	26
4.4.3	Wybór i format danych wejściowych	26
4.4.4	Ekstrakcja cech	27
4.4.5	Format zapisu	27
4.4.6	Inne opcje	29
5	Specyfikacja wewnętrzna	31
5.1	Przedstawienie idei	31
5.2	Architektura systemu	31
5.3	Opis najważniejszych klas oraz modułów	32
5.3.1	NetworkArchitecture.py	32
5.3.2	DataPrep.py	32
5.3.3	SelectiveSearch.py	33
5.3.4	RCNN.py	33
5.3.5	View.py	33
5.3.6	main.py	33
5.4	Opis najważniejszych metod i funkcji	33
5.4.1	create_annotations	33

5.4.2	get_region_proposal	34
5.4.3	warp_and_create_cnn_feature	34
5.4.4	extract_features_from_image	35
6	Weryfikacja i walidacja	37
6.1	Wyniki eksperymentalne	39
7	Podsumowanie i wnioski	41

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 40. zeszłego stulecia [1]. Idee te były sukcesywnie rozwijane oraz modyfikowane, lecz ograniczenia sprzętowe oraz trudność w dostępie do danych uniemożliwiały ich realne wykorzystanie. Dopiero na początku zeszłej dekady postępująca cyfryzacja oraz digitalizacja spowodowała znaczny wzrost ilości przechowywanych danych oraz ich większą dostępność. W tabeli 1.1 pokazano, jak zmieniały się rozmiary wybranych zbiorów danych przeznaczonych do zagadnień związanych z rozpoznawaniem rysów twarzy na przestrzeni lat. Łatwo zauważać 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 technologiami takich jak CUDA (*Compute Unified Device Architecture*), które umożliwiają łatwe wykorzystanie tej mocy, systemy oparte na sztucznej inteligencji osiągają coraz to lepsze wyniki i są w stanie wykonywać pewne zadania lepiej niż człowiek.

W ostatnich latach można zaobserwować zwiększający 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,

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

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	~ 1,250,000
Aff-Wild2 [13]	2019	~ 2,800,000

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 biologii[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 ekstrakcję 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 się 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. Zostaje 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 validacji 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. Pokrótko 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 programie, 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] pozwalają 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, zdawać się mogło, coraz bliżej.

Jednym z podstawowych problemów z dziedziny wizji komputerowej jest klas-

syfikacja. Polega ona na przypisaniu pewnej kategorii na podstawie obrazu. Za- zwyczaj 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 zidentyfikowała wszystkie obiekty, które się na nim znajdują, oraz wskazała w którym miejscu na zdjęciu te obiekty się znajdują.

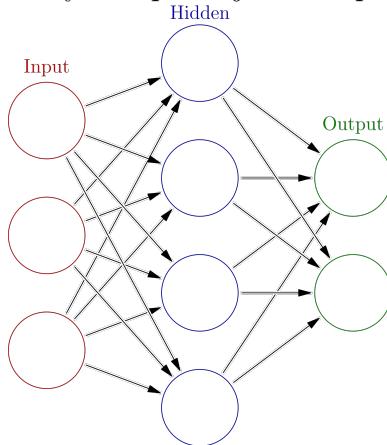
2.2 Sztuczne sieci neuronowe

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

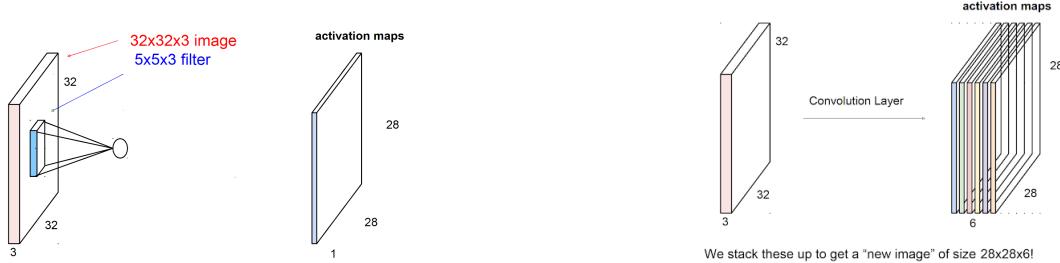


na określone pole widzenia. Każdy neuron ma swoje pole recepcyjne i reaguje na bodziec tylko w obrębie tego pola. Sieci konwolucyjne starają się odzworować sposób działania tych neuronów w korze wzrokowej. Zamiast patrzeć na obraz jako całość, każdy neuron jest odpowiedzialny za jego małą część. Neurony 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. próbującymi), a na ich końcach umieszcza się jedną lub kilka warstw w pełni połączonych. Zadaniem tych warstw w pełni połączonych jest dokonanie klasyfikacji na podstawie wyjścia z ostatniej warstwy 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

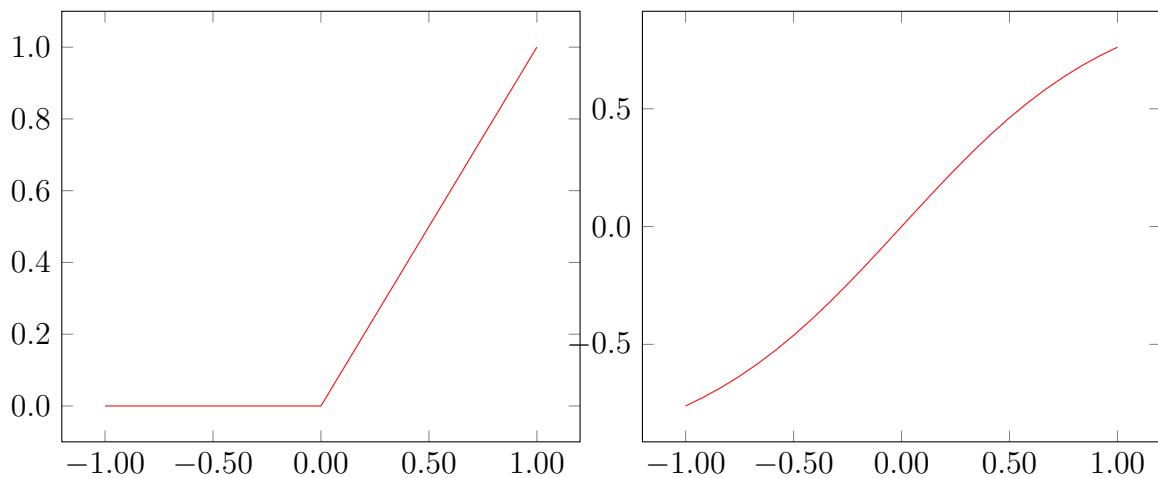
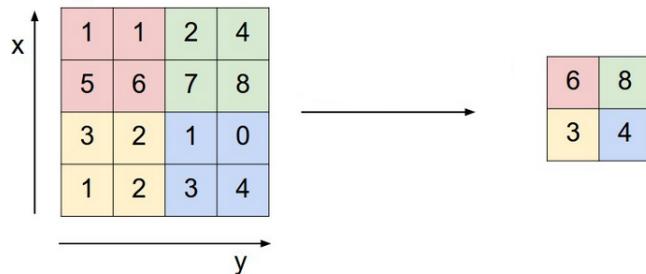
Warstwa próbkująca

Nazywana też czasem warstwą łączenia, najczęściej umieszczana jest pomiędzy dwoma warstwami konwolucyjnymi. Jej zadaniem jest redukcja rozmiaru otrzymanych map aktywacji, co zmniejsza ilość parametrów, których sieć musi się nauczyć. Ta warstwa opiera się u filtry najczęściej 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 funkcja rektyfikowanej jednostki liniowej (ReLU), definiowanej jako $\max(0, x)$ (rys. 2.5) pozwala osiągnąć 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 zmniejszenia 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 kontekstowych przypadkach.

Warstwa regularyzacji opuszczeń

Warstwa ta w sposób losowy wyklucza 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].

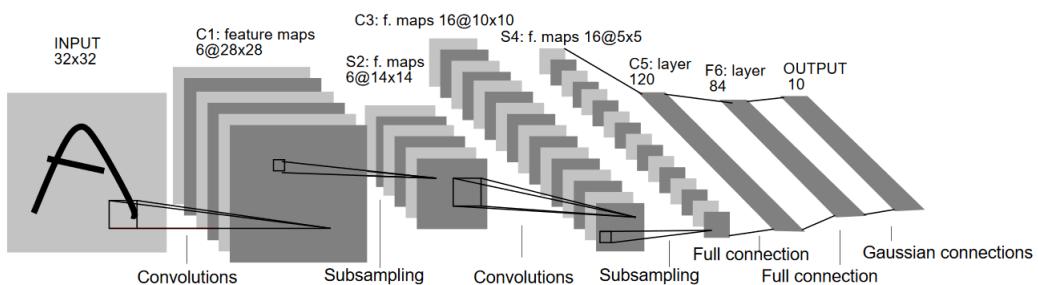
2.2.2 Przykładowe modele sieci

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 naukowych.

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

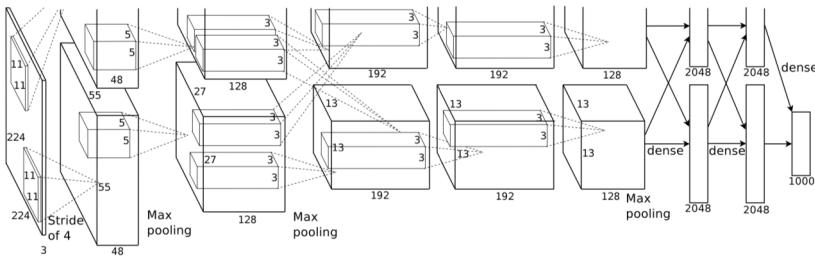


AlexNet[26]

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 wektorowych wspierających[27]. Sytuacja ta

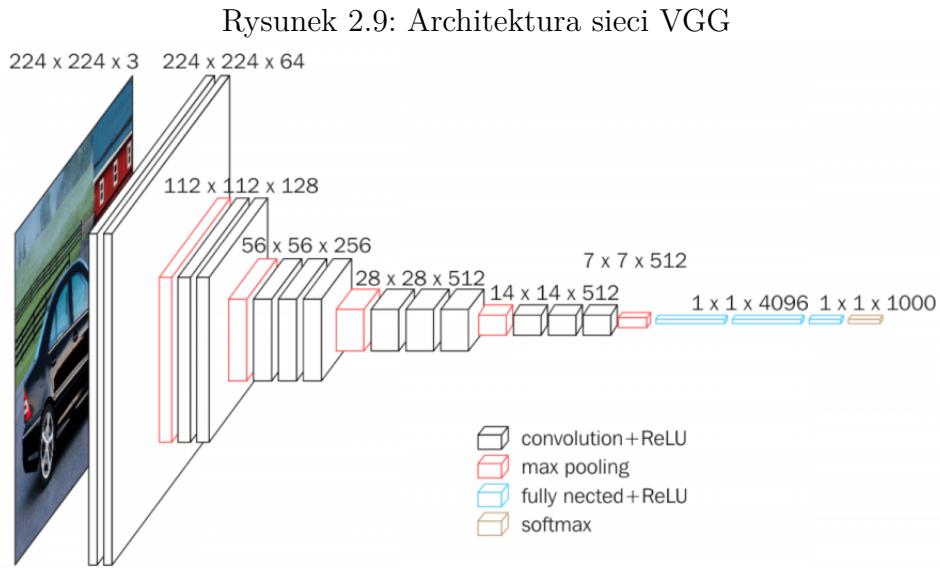
utrzymywała się aż do 2012 roku, kiedy to Alex Krizhevsky i in. stworzyli sieć AlexNet. Sieć osiągnęła najlepszy rezultat w konkursie ILSVRC, z błędem na poziomie 15,3%, ponad 10 punktów procentowych lepiej od drugiego miejsca. Ten świetny rezultat na nowo pobudził zainteresowanie technologią sieci konwolucyjnych, 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 opuszczanej 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 Zissemanna, pomimo tego, że zajęła 2 miejsce w konkursie ILSVRC, jest jedną z najbardziej rozpoznawalnych sieci w dziedzinie. Simonyan i Zissemann przyjęli inną strategię – zamiast zmieniać i testować różne wielkości warstw, użyli w niej warstw konwolucyjnych o stałym wymiarze 3x3 oraz próbujących 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.



ResNet[29]

Sieć ta stworzona została przez Kaiminga He i in. i wygrała konkurs ILSVRC w 2015 roku. Jest przykładem sieci szcątkowej, w której niektóre połącznia pomiędzy warstwami są pomijane. Takie rozwiązanie pozwala na lepszą skalowalność wraz ze zwiększaniem liczby warstw oraz eliminuje problem tzw. zanikającego gradientu. Sieć ta posiada olbrzymią liczbę 152 warstw, i ta głębokość 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 musiała 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 2.3.2. 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.

2.3.1 IoU

Podczas detekcji musimy zmierzyć się z jeszcze jednym zadaniem – musimy zlokalizować nasz obiekt na zdjeciu. Jako lokalizację przyjmuję 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 wyznaczmy 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:

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

Czyli iloraz części wspólnej oraz sumy obu obszarów. Próg, od którego uznamy, 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.

2.3.2 Algorytm wyszukiwania selektywnego

Algorytm ten[31] ma za zadanie wyznaczyć propozycje regionów, które będą później używane do detekcji obiektów. Na początku algorytm dokonuje segmentacji obrazu 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:

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

Gdzie c_i^k, c_j^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:

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

Gdzie t_i^k, t_j^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 $\text{size}(im)$ jest ono liczone jako:

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

Podobieństwo kształtu

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

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

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

Końcowe podobieństwo można ozyskać ze wzoru:

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

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

2.3.3 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 zauważać, że na etapie wyznaczania regionów nie następuje żadna nauka sieci – algorytm wyszukiwania selektywnego jest algorymem 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 zaoponował 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 sieć 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 znaczaco zmniejszamy złożoność obliczeniową, co skutkuje 9-krotnym przyspieszeniem etapu treningu sieci oraz aż 213-krotnym przyspieszeniem etapu testowania, przy jednoczesnym zwiększeniu precyzji sieci.

Faster R-CNN[34]

Wszystkie poprzednie metody używały algorytmu wyszukiwania selektywnego do wyznaczania 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 architektura 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 rozwiązania, 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.

Rozdział 3

Wymagania i narzędzia

W tym rozdziale zostanie dokonany opis wymagań funkcjonalnych oraz niefunkcjonalnych pracy oraz zaprezentowane zostaną diagramy przypadków użycia. Następnie omówiona będą użyte narzędzia oraz opisana zostanie metodyka pracy nad projektowaniem i implementacją.

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 zapisywania 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 ekstrakcja
- W ramach projektu zostanie zaimplementowana architektura R-CNN i będzie domyślnie używaną architekturą (jeżeli użytkownik nie wskaże innej).

3.1.2 Wymagania niefunkcjonalne

Projekt powinien działać na dowolnym systemie operacyjnym (Windows, macOS X, Linux). Szacowany czas wykonywania ekstrakcji może zająć nawet do kilkudziesięciu sekund na jeden obraz, dlatego dla dużych zbiorów obrazów proces może trwać nawet kilka dni.

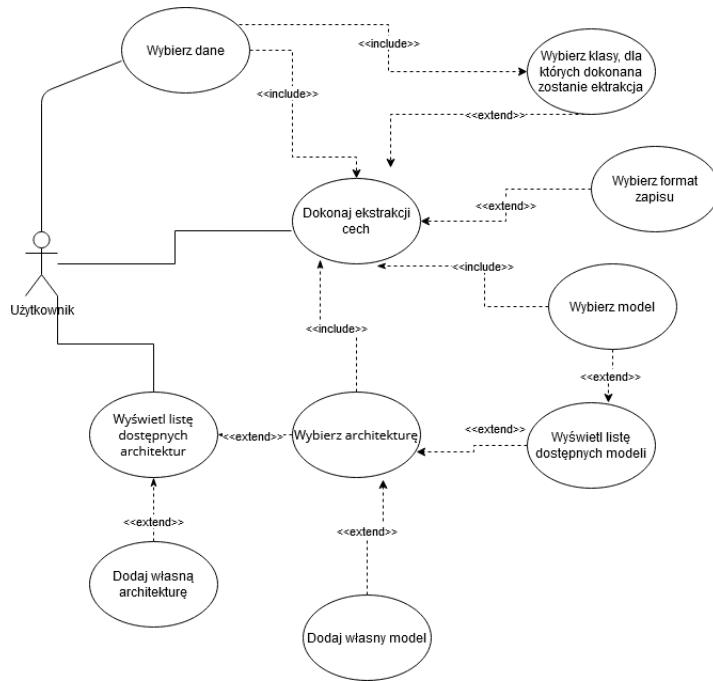
3.2 Diagram przypadków użycia

Na rysunku 3.1 przedstawiono diagram przypadków użycia, który obrazuje wszystkie funkcjonalności systemu.

3.3 Opis narzędzi

3.3.1 Python

Python jest wysokopoziomowym językiem, który ze względu na swój rozbudowany system bibliotek oraz zwięzłą i przejrzystą składnię znajduje swoje zastosowanie w wielu dziedzinach, szczególnie tych związanych z uczeniem maszynowym. Python nie wymusza jednego paradygmatu programowania, jak np. C#, lecz pozwala na użycie różnorodnych technik, takich jak programowanie obiektowe, pro-



Rysunek 3.1: Diagram przypadków użycia

gramowanie strukturalne czy programowanie funkcyjne. Całość tworzonego projektu została stworzona właśnie w tym języku.

3.3.2 Keras i Tensorflow

Tensorflow to darmowa i otwartoźródłowa biblioteka, służąca głównie do rozwiązywania zadań z obszaru uczenia maszynowego. Keras służy jako API (ang. *Application Programming Interface*) do biblioteki Tensorflow. Obie te biblioteki zostały użyte do implementacji konwolucyjnej sieci neuronowej. Oferują one możliwość wyborów modelów z już nauczonymi parametrami sieci (np. model VGG16, opisany w podrozdziale 2.2.2, czy ResNet opisany w 2.2.2).

3.3.3 Skimage i Imageio

Skimage oraz Imageio są bibliotekami języka python, które umożliwiają łatwe wczytywanie oraz przetwarzanie obrazów. W programie zostały one wykorzystane w implementacji algorytmu wyszukiwania selektywnego, jak i również do zmian

rozmiarów obrazów.

3.3.4 Abstract Base Classes

Abstract Base Classes (w skrócie – ABC) to moduł języka Python, który zapewnia infrastrukturę pozwalającą na zdefiniowanie klas abstrakcyjnych. Za jego pomocą tworzona jest abstrakcyjna klasa reprezentująca architekturę sieci.

3.3.5 Tkinter

Tkinter to biblioteka umożliwiająca stworzenia GUI (ang. *Graphical User Interface*) w Pythonie. Została użyta do stworzenia prostego interfejsu graficznego pomiędzy użytkownikiem a programem.

3.3.6 PASCAL VOC

W projekcie została wykorzystana baza danych PASCAL VOC 2012[36] zawierająca 11530 obrazów z wyznaczonymi 27450 regionami zainteresowań, wraz z odpowiadającym im etykietami. Ta baza danych jest publicznie dostępna i przez lata była używana w ramach corocznego konkursu z zadań dotyczących wizji komputerowej.

3.4 Metodyka pracy nad projektowaniem oraz implementcją

Pracę rozpoczęto od ogólnego zapoznania się z dziedziną oraz zrozumienia przedstawionego zagadnienia. Dzięki powszechnie dostępnym źródłom, takim jak artykuły internetowe oraz wykłady innych uczelni udostępniane na serwisach internetowych, szybko pojęto podstawy teoretyczne uczenia maszynowego oraz zasady działania konwolucyjnych sieci neuronowych.

Kolejnym krokiem było ustalenie wymagań projektu i wyznaczenie efektu końcowego. W celu lepszej wizualizacji wymagań stworzono również diagram przypadków użycia, ilustrujący przykładowe użycia systemu. Wymagania były następnie

sukcesywnie doprecyzowane, dzięki czemu przedstawiony problem stawał się łatwiejszy do zrozumienia.

Następnym krokiem była analiza źródeł odnoszących się już do głównego zagadnienia pracy. Dokonano jej poprzez lekturę publikacji naukowych oraz przeglądu znanych rozwiązań. Zapoznano się również z podobymi projektami dostępnymi publicznie i ich rozwiązaniami, dzięki czemu uzyskano lepszy punkt odniesienia. Następnie dokonano wyboru odpowiednich narzędzi oraz zapoznano się z ich dokumentacją techniczną. Posiadając tą wiedzę teoretyczną, jasno przedstawiony cel oraz znajomość wybranych narzędzi, przystąpiono do implementacji projektu. Stworzono podstawowy model, do którego następnie dodawane były kolejne funkcjonalności, aż do osiągnięcia wyznaczonego efektu końcowego.

Rozdział 4

Specyfikacja zewnętrzna

W tym rozdziale znajduje się opis specyfikacji zewnętrznej projektu. Opisane zostaną wymaganie sprzętowe, sposób instalacji oraz aktywacji narzędzia. Następnie przedstawiony zostanie sposób obsługi, przykład działania oraz scenariusze korzystania z systemu.

4.1 Wymagania sprzętowe i programowe

4.1.1 Python

- System operacyjny:
 - Windows 7 lub nowszy
 - Mac OS X 10.11 lub nowszy, 64-bitowy
 - Linux RHEL 6/7, 64-bitowy
- Procesor x86 64-bit (Intel/AMD)
- 4GB pamięci RAM
- 5GB wolnej przestrzeni dyskowej

4.1.2 Tensorflow

- Python 3.5 lub nowszy

- System operacyjny:
 - Windows 7 lub nowszy, 64-bitowy
 - Mac OS X 10.12.6 lub nowszy, 64-bitowy
 - Ubuntu 16.04 lub nowszy, 64-bitowy
- Procesor obsługujący instrukcje AVX

4.2 Sposób instalacji

4.2.1 Python

W celu uruchomienia narzędzia należy posiadać zainstalowanego Pythona w wersji 3.5 lub nowszej. Można to zrobić poprzez oficjalną stronę internetową, w zakładce Downloads, <https://www.python.org/downloads/>. Należy pobrać plik odpowiedni dla swojego systemu operacyjnego, a następnie uruchomić go i przejść przez proces instalacji.

4.2.2 Instalacja narzędzia

Kod narzędzia należy pobrać z publicznego repozytorium <https://github.com/Mikohab450/FeatureExtraction.git>. Można to zrobić na wiele sposobów, pobierając kod bezpośrednio ze strony w formacie .zip, lub klonując repozytorium za pomocą narzędzia git w terminalu systemowym używając komendy:

```
$ git clone https://github.com/Mikohab450/FeatureExtraction.git
```

4.3 Sposób aktywacji

Następnie należy zainstalować wszystkie biblioteki potrzebne do działanie programu. Można to zrobić za pomocą wbudowanego w Pythona systemu zarządzania bibliotekami pip. W systemie Windows należy wpisać komendę:

```
C:\ > py -m pip install -r .../ FeatureExtraction / FeatureExtraction / requirements
```

A w systemach Linux oraz Mac OS X:

```
$ python -m pip install -r .../FeatureExtraction/FeatureExtraction/requirements
```

Gdzie w miejscu ... należy wpisać ścieżkę, do której narzędzie zostało pobrane.
Następnie narzędzie można uruchomić wpisując w terminalu Windowsa

```
C:\> py .../FeatureExtraction/FeatureExtraction/Main.py
```

Analogicznie dla pozostałych systemów:

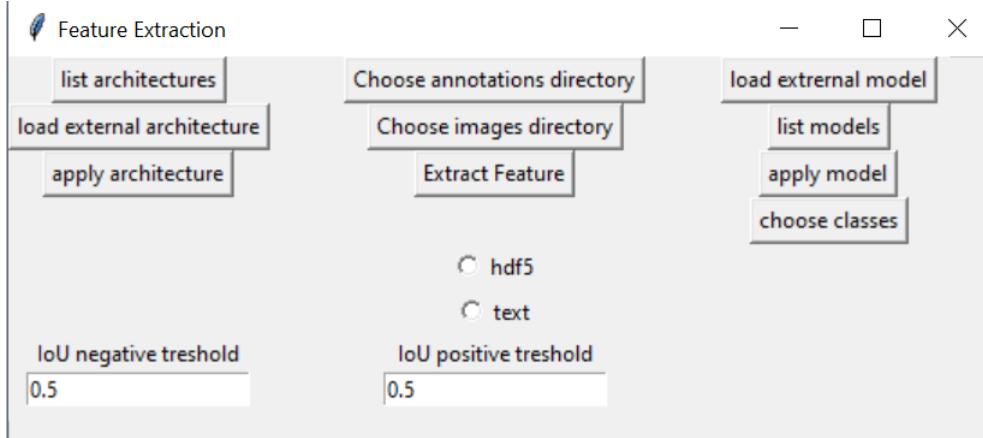
```
$ python .../FeatureExtraction/FeatureExtraction/Main.py
```

bądź klikając dwukrotnie plik **Main.py**.

4.4 Przykład działania

Po uruchomieniu narzędzia pokaże nam się prosty interfejs, przedstawiony na rysunku 4.1. Za jego pomocą możemy wykorzystywać różne funkcjonalności systemu.

Rysunek 4.1: Interfejs programu



4.4.1 Wybór architektury

W celu wybrania architektury sieci należy zaznaczyć architekturę z listy przy pomocy przycisku *list architectures*, a następnie zainicjalizować ją klikając *apply architecture*. Jeżeli inicjalizacja przebiegła pomyślnie, to wyświetlane zostanie

okienko informujące o tym, jaka architektura jest używana. W przeciwnym razie wyświetlna zostanie okienko informujące o wystąpieniu błędu. W przypadku kliknięcia na *apply architecture* bez wcześniejszego wybierania architektury z listy, użyta zostanie domyślnej architektury RCNN. Jeżeli użytkownik chce wybrać architekturę, której nie ma na liście, musi ją najpierw załadować przy pomocy przycisku *load external architecture*. Po kliknięciu niego będzie miał możliwość wybrania pliku Pythona (modułu), z którego zostanie wczytana architektura. Aby zewnętrzna architektura została wczytana poprawnie, musi spełniać następujące warunki:

- Architektura musi być klasą dziedziczącą po klasie NetworkArchitecture oraz implementującą wszystkie jej metody.
- Nazwa pliku oraz nazwa klasy muszą być identyczne.

Jeżeli warunki te zostaną spełnione, to po wybraniu pliku pojawi się on na liście architektur i będzie można go wybrać.

4.4.2 Wybór modelu

Analogicznie jak przy wyborze architektury, aby wybrać model należy zaznaczyć jeden z listy *list models*, a następnie zainicjalizować przy pomocy *apply model*. Należy pamiętać, że przed wybraniem modelu należy wybrać architekturę. Domyślnym modelem jest VGG16 i to on zostanie użyty jeżeli nie wybierzemy innego modelu z listy. Własny model można załadować klikając *load external model*. Model powinien być w formacie TensorFlow SavedModel lub Keras H5 i mieć wejście o wymiarze 224x224x3.

4.4.3 Wybór i format danych wejściowych

Dane wejściowe składają się z dwóch części – folderu ze zdjęciami w dowolnym formacie oraz folderu z odpowiadającymi im adnotacjami. Adnotacje powinny mieć taką samą nazwę pliku (bez rozszerzenia) jak zdjęcie, którego dotyczą. Folder z zdjęciami możemy wybrać klikając w przycisk *Choose images directory*, a folder z adnotacjami poprzez *Choose annotations directory*. Na rysunku 4.2 przedstawiono

przykładowe zdjęcie w folderze wejściowym, wzięta z bazy danych PASCAL VOC, a na rysunku 4.3 pokazano plik zawierający adnotacje do tego zdjęcia.

Rysunek 4.2: Wybrany obraz z bazy PASCAL VOC



4.4.4 Ekstrakcja cech

Po wyborze architektury, modelu oraz folderów z danymi wejściowymi należy zaznaczyć format zapisu, a następnie można przystąpić do ekstrakcji cech klikając w *Extract features*. Jeżeli którykolwiek z poprzednich kroków nie został wykonany bądź został wykonany niepoprawnie (np. wystąpił błąd podczas jego wykonywania), to ekstrakcja się nie wykona i program zwróci stosowny komunikat. Po zakończeniu udanej ekstrakcji program poinformuje o tym użytkownika, wyświetlając okno z informacją *Features saved!*. W zależności od wybranego formatu zapisu, w katalogu głównym programu pojawi się plik *Features.txt* dla formatu tekstowego, lub pliki *annotations.h5* oraz *activations.h5* dla formatu hdf5.

4.4.5 Format zapisu

W przypadku pliku tekstowego, format zapisu wygląda następująco:

Nazwa klasy	IoU	xmin	ymin	xmax	ymax	ID pliku	wektor
-------------	-----	------	------	------	------	----------	--------

```
1 <annotation>
2   <folder>VOC2012</folder>
3   <filename>2007_000799.jpg</filename>
4   <source>
5     <database>The VOC2007 Database</database>
6     <annotation>PASCAL VOC2007</annotation>
7     <image>flickr </image>
8   </source>
9   <size>
10    <width>500</width>
11    <height>441</height>
12    <depth>3</depth>
13  </size>
14  <object>
15    <name>horse </name>
16    <bndbox>
17      <xmin>95</xmin>
18      <ymin>23</ymin>
19      <xmax>500</xmax>
20      <ymax>441</ymax>
21    </bndbox>
22  </object>
23  <object>
24    <name>person </name>
25    <bndbox>
26      <xmin>230</xmin>
27      <ymin>225</ymin>
28      <xmax>500</xmax>
29      <ymax>441</ymax>
30    </bndbox>
31  </object>
32 </annotation>
```

Rysunek 4.3: Etykieta opisująca rysunek 4.2

Gdzie $xmin$, $ymin$, $xmax$, $xmax$ oznaczają cztery wiechołki prostokątu, z którego ektrahowano cechy. W tablicy 4.1 przedstawiono fragment próbki danych wyektrahowanych ze zdjęcia z rysunku 4.2. Ponieważ wektor cech mają 4096 liczb, pozwolono sobie je tutaj skrócić do tylko dwóch pierwszych. W przypadku zapisu w formacie hdf5 etykiety zapisywane są w pliku *annotations.h5*, a wektory w pliku *activations.h5*.

Tablica 4.1: Fragment przykładowych wyektrahowanych danych

horse	0.76	0	0	223	223	2007_000799	0.614785432	2.24941	[...]
horse	0.77	0	0	223	218	2007_000799	0.634443283	2.2379	[...]
horse	0.66	43	55	223	168	2007_000799	0.713880411	2.0658	[...]
person	0.73	43	101	180	122	2007_000799	0.74330669	2.33210	[...]
background	0.02	97	157	14	38	2007_000799	1.035556	1.999945	[...]
background	0.12	99	152	12	30	2007_000799	0.7487751	2.020392	[...]

4.4.6 Inne opcje

Wybór klas

Klikając w przycisk *choose classes* możemy dokonać wyboru klas, które podlegają ekstrakcji. Należy pamiętać, że najpierw należy wybrać folder z etykietami, a dopiero później ograniczać klasy. W przypadku wykonania tych kroków w odwrotnej kolejności wyświetlona lista z klasami będzie pusta.

Wybór progów IoU

Wpisując liczby w pola *IoU positive threshold* i *IoU negative threshold* można wybrać jaki będzie próg oceny, czy dana etykieta jest lub nie jest obiektem (więcej o tym w rozdziale 2.3.1). Wartości te powinny być z przedziału 0-1, logiczny również jest założenie że próg próbek pozytywnych będzie większy lub równy progiowi próbek negatywnych, choć nie jest to konieczne. Domyślnie obie zmienne przyjmują wartość 0.5.

Rozdział 5

Specyfikacja wewnętrzna

Niniejszy rozdział zawiera opis idei zastosowanego rozwiązania. Opisane zostaną również architektura systemu, użyte klasy, najważniejsze algorytmy oraz komponenty.

5.1 Przedstawienie idei

Idea projektu zakładała stworzenie uniwersalnego narzędzia, które zawierałoby w sobie zaimplementowaną architekturę sieci, lecz umożliwiała łatwą rozbudowę o inne architektury. W tym celu połączono mechanikę dziedziczenia oraz dynamicznego ładowania modułów. Do implementacji architektury potrzebne były trzy elementy: odczyt danych, wyznaczenie regionów oraz sieć konwolucyjna. Przy implementacji architektury zadbane o to, aby odczyt danych oraz algorytm selektywnej selekcji służący do wyznaczanie regionów były zaimplementowane jako osobne moduły, dzięki czemu działają niezależnie od reszty rozwiązania.

5.2 Architektura systemu

Główna klasą narzędzia klasa `View`, jest odpowiedzialna za wyświetlenie interfejsu graficznego i wykonywanie zadanych funkcjonalności. Działa ona na obiekcie `NetworkArchitecture`, który reprezentuje architekturę sieci, oraz korzysta z modułu `DataPrep.py` w celu przetworzenia adnotacji i zapisania ich w odpowiednim

formacie.

5.3 Opis najważniejszych klas oraz modułów

5.3.1 NetworkArchitecture.py

Moduł zawiera klasę `NetworkArchitecture`, która jest klasą abstrakcyjną reprezentującą architekturę sieci. Klasy abstrakcyjne nie są wbudowane w Pythona, dlatego aby osiągnąć ten efekt klasa ta dziedziczy bo klasie **ABCMeta** z pakietu **abc** opisanego w podrozdziale 3.3.4. Klasa ta zawiera następujące metody:

- `extract_features_from_image`
- `choose_model`
- `load_model`

W celu stworzenia klasy dziedziczącej należy te metody w niej poprawnie zaimplementować. Dodatkowo klasa ta posiada dwa pola `CNN_model` oraz `list_of_models`. W pierwszym przechowywany jest sieć konwolucyjna, drugie przechowuje listę dostępnych modeli. Przy implementacji klasy dziedziczącej należy pamiętać o tym, żeby tych właśnie pól używać przy wczytywaniu i inicjalizacji modeli, ponieważ w przeciwnym razie klasa dziedzicząca nie będzie kompatybilna z resztą programu.

5.3.2 DataPrep.py

Moduł ten zawiera funkcje pozwalające na przetworzenie wejściowego folderu z adnotacjami oraz zapisanie tylko tych istotnych adnotacji do pliku csv. Nie zapisujemy wszystkich danych z adnotacji, ponieważ mogą one zawierać również dodatkowe informacje o obiekcie, które są potrzebne do innych zadań (np. do segmentacji czy rozpoznawania akcji). Do zadań klasyfikacji potrzebujemy jedynie nazwe obiektu oraz obwolutę, w której się znajduje, dlatego odczytujemy tylko te dane a następnie zapisujemy je w formacie, z którego łatwo będzie je później odczytać.

5.3.3 SelectiveSearch.py

Moduł zawiera funkcje odpowiedzialną na implementację algorytmu wyszukiwania selektywnego, opisanego w podrozdziale 2.3.2. Do obliczania podobieństwa użyto wizualnego deskryptora nazywanego lokalnymi wzorami binarnymi (ang. Local Binary Patterns), co jest drobym odstępstwem od wcześniej opisanej metody, lecz zasadniczo nie zmienia jej działania.

5.3.4 RCNN.py

Moduł zawiera klasę **RCNN**, która implementuje tą właśnie architekturę sieci, opisaną dokładniej w rozdziale 2.3. Klasa ta dziedziczy po klasie **NetworkArchitecture**.

5.3.5 View.py

W tym module znajduje się klasa **View.py**, która odpowiada za interfejs graficzny programu. Dziedziczy ona po klasie **tkinter.Frame** i jest singletonem, tworzonym i wywoływanym przez główny program.

5.3.6 main.py

Moduł służący do uruchomienia projektu. Inicjalizuje instancję klasy **View** i tworzy główną pętlę programu.

5.4 Opis najważniejszych metod i funkcji

5.4.1 **create_annotations**

Funkcja **create_annotations** pokazana na rysunku 5.1, znajduje się module **DataPrep.py** i pobiera jako argument ścieżkę do folderu z adnotacjami, a następnie za pomocą funkcji **extract_single_xml_file** odczytuje potrzebne informacje ze wszystkich plików z tego folderu i zapisuje je w liście, która jest następnie konwertowana do typu **DataFrame** biblioteki pandas. Ten typ jest przeznaczony do przechowywania danych w postaci tabelarycznej i ma wbudowaną metodę **to_csv**

umożliwiającą zapis do formatu csv. Używając tej metody zapisujemy potrzebne etykiety w folderze roboczym, aby móc później z nich skorzystać. Funkcja zwraca listę z nazwami wszystkich klas odczytanych z adnotacji.

```

1 def create_annotations(dir_anno):
2     df_anno = []
3     class_names=[]
4     for fnm in os.listdir(dir_anno):
5         if not fnm.startswith('.'):
6             tree = ET.parse(os.path.join(dir_anno,fnm))
7             row = extract_single_xml_file(tree, class_names)
8             row[ "fileID" ] = fnm.split( ".") [0]
9             df_anno.append(row)
10    df_anno = pd.DataFrame(df_anno)
11    df_anno.to_csv("etykiety.csv",index=False)
12    return class_names}
```

Rysunek 5.1: Definicja funkcji `create_annotations`

5.4.2 `get_region_proposal`

Funkcja `get_region_proposal`, która znajduje się w module **SelectiveSearch**, pokazana jest na rysunku 5.2. Przyjmuje ona jako argument wejściowy obraz, dla którego regiony mają być wyznaczone. Zwraca listę słowników zawierających wyznaczone regiony.

5.4.3 `warp_and_create_cnn_feature`

Metoda te zdefiniowana jest w klasie **RCNN** i oblicza cechy głębokie dla otrzymanej listy obrazów `image`. Dla każdego obrazu z listy najpierw zmienia jego rozmiar za pomocą metody `warp` tak, aby jego wymiary były kompatybilne z wejściem sieci konwolucyjnym. Obraz jest przepuszczany przez sieć poprzez użycie wbudowanej metody `predict`, która operuje na modelu sieci neuronowej, który jest obiektem klasy **Model** z biblioteki *Keras*.

```

1 def get_region_proposal(img_8bit, min_size = 500):
2     img = image_segmentation(img_8bit, min_size = min_size)
3     R = extract_region(img)
4     tex_grad = calc_texture_gradient(img)
5     hsv = calc_hsv(img)
6     R = augment_regions_with_histogram_info(tex_grad, img,
7         R, hsv, tex_grad)
8     del tex_grad, hsv
9     neighbours = extract_neighbours(R)
10    S = calculate_similarlity(img, neighbours)
11    regions = merge_regions_in_order(S, R, imsize = img.shape
12        [0] * img.shape[1])
13    no_duplicates = remove_duplicates(regions)
14    return(no_duplicates)

```

Rysunek 5.2: Definicja funkcji `get_region_proposal`

```

1 def warp_and_create_cnn_feature(self, image):
2     for irow in range(len(image)):
3         image[irow] = self.warp(image[irow], self.
4             warped_size)
5     image = np.array(image)
6     feature = self.CNN_model.predict(image)
7     return(feature)

```

Rysunek 5.3: Definicja funkcji `warp_and_create_cnn_feature`

5.4.4 `extract_features_from_image`

Metoda `extract_features_from_image` działa na obiekcie klasy **RCNN** i służy do wyznaczenia odpowiednich regionów z odpowiadającymi im adnotacjami, a następnie wyliczenia dla nich wektorów cech. Funkcja przyjmuje następujące parametry:

- `img_dir` – ścieżka do folderu zawierającego zdjęcia.
- `classes` – słownik z nazwami klas, dla których dokonana ma być ekstrakcja. Każda nazwa klasy jest kluczem, której odpowiada wartość typu `tkiner.BooleanVar`

- **IoU_cutoff_object** – wartość współczynnika IoU, powyżej której etykieta ma być zakwalifikowana jako obiekt. Domyślnie przyjmuje wartość 0.5.
- **IoU_cutoff_not_object** – wartość współczynnika IoU, poniżej której etykieta ma być zakwalifikowana jako tło. Domyślnie przyjmuje wartość 0.5.

Metoda ta odczytuje etykiety z pliku *etykiety.csv* zapisanego w folderze roboczym. Następnie iteruje po kolejnych etykietach każdego obrazu, znajdując odpowiadający obraz w folderze **img_dir**. W celu przyspieszenia obliczeń każdy obraz jest skalowany do mniejszych rozmiarów przy pomocy funkcji **warp**. Następnie obliczane są propozycje regionów, po czym rozpoczyna się iterację po wszystkich obiektach znajdujących się w obrazie. Używając zmiennej **classes** sprawdza się, czy obiekt powinien być brany pod uwagę przy ekstrakcji. Jeżeli nazwie obiektu odpowiada wartość *False*, to pętla jest przerywana i następuje przejście do kolejnego obiektu. W przeciwnym wypadku obliczenia są kontynuowane. Ponieważ obraz jest przeskalowany, to wymiary prostokątów zawierających obiekty, które zostały odczytane w pliku csv są niewłaściwe i należy je również przeskalać. Po przeskalowaniu trzymujemy zmienne *true_xmin*, *true_ymin*, *true_xmax*, *true_ymax*, które opisują cztery wierzchołki prostokąta w zmienionym już rozmiarze. Następnie iterujemy po wszystkich propozycjach regionów, obliczając wartość współczynnika IoU za pomocą funkcji **get_IOU**, poczynając porównujemy ten współczynnik do zmiennych **IoU_cutoff_object** oraz **IoU_cutoff_not_object**. Na podstawie tych porównań umieszczamy wierzchołki regionu wraz z nazwą klasy, obliczonym współczynnikiem IoU oraz nazwą akurat przetwarzanego obrazu do listy **info_pos** dla próbek pozytywnych, lub **info_neg** dla negatywnych. Dodatkowo na podstawie regionu wyznaczany jest fragment obrazu, który zapisywany jest odpowiednio w **image_pos** bądź **image_neg**. Po zakończeniu wszystkich iteracji łączone są listy **image_pos** z **image_neg** oraz **info_pos** z **info_neg**. Następnie połączone listy z regionami są przekazywane do funkcji **warp_and_create_cnn_feature**, która wylicza dla nich cechy. Funkcja zwraca krotkę zawierającą stworzone etykiety oraz odpowiadające im cechy.

Rozdział 6

Weryfikacja i walidacja

Podczas tworzenia projektu każdy moduł i funkcjonalności był systematycznie testowane pod względem poprawności funkcjonowania

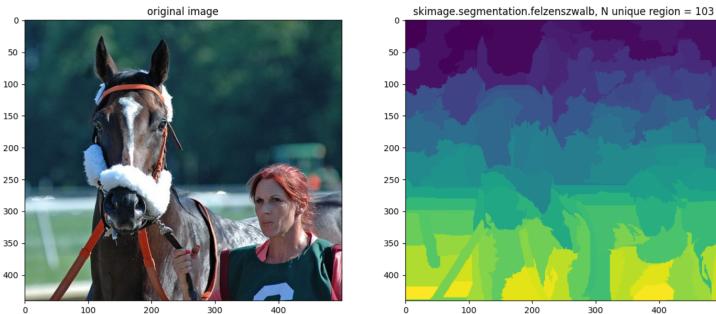
Moduły DataPrep oraz SelectiveSearch

Obydwa te moduły zostały stworzone na początku pracy nad narzędziem i były one testowane jeszcze przed przejściem do następnych etapów pracy. W celu weryfikacji poprawności ich działania, wizualizowano rezultaty działań, dzięki czemu można było zobaczyć, czy otrzymane wyniki pokrywają się z oczekiwaniemi. Na rysunku 6.1 pokazano działanie algorytmu segmentacji metodą Felzenszwalba, a na rysunku 6.2 wyświetcono regiony wyznaczone przez algorytm wyszukiwania selektywnego. Jednym z problemów napotkanym w tym obszarze pracy było wyznaczania kilku regionów o identycznych wymiarach. Nie był to co prawda błąd uniemożliwiający działanie narzędzie, lecz raczej niechciane zachowanie, które zlikwidowano poprzez dodanie funkcji usuwającej duplikaty z listy regionów.

Moduł View

W celu przetestowania funkcjonalności dodawania zewnętrznych architektur stworzono testową klasę **exampleModel**. Następnie sprawdzono, czy po wybraniu pliku z tą klasą moduł jest ładowany poprawnie, cz jest ona wyświetlana na liścia architektur oraz czy po zainicjalizowaniu jej jest tworzona instancja jej klasy. Podczas testowania tego elementu zauważono nieoczekiwane zachowanie systemu

Rysunek 6.1: Wizualizacja segmentacji metodą Felzenszwalba.



Rysunek 6.2: Wizualizacja wyznaczonych regionów.



– w przypadku wyjścia z okna dialogowego podczas wczytywania pliku system pomimo tego, że żaden plik nie został wybrany, próbował otworzyć plik o pustej nazwie, co powodowało błąd. Dodanie warunku sprawdzającego, czy jakiś plik został wybrany, rozwiązało problem.

Podczas testowania funkcjonalności wyświetlania list napotkano na kolejną słabość systemu. Każde naciśnięcie przycisku wyświetlajacego listę otwierało nowe okno z listą, można więc było otwierać nieograniczoną ilość okien. Dodatkowo w przypadku wielu otwartych okien z listami próba wybrania elementu z jednej z nich kończyła się błędem systemu. Problem rozwiązało poprzez ograniczenie ilości wyświetlanych okien, tak aby tylko jedno okno z listą mogło być otwarcie w danym

momencie.

6.1 Wyniki eksperymentalne

W celu oszacowania szybkości działania narzędzia oraz ilości zużywanej pamięci, przeprowadzono szereg testów dla różnych ilości danych wejściowych. Wybrano losowo ze zbioru danych PASCAL VOC określoną liczbę zdjęć, a następnie zmierzono czas wykonywania algorytmu wyszukiwania selektywnego (t_1), ekstrakcji cech (t_2), zapisu do pliku (t_3) oraz ich sumę (t_c). Zmierzono również ilość pamięci, która była w użyciu przez proces podczas zakończenia obliczeń. Wyniki przedstawiono w tabeli 6.1. Można zaobserwować, że czas zapisu (t_3) w formacie hdf5 jest szybszy od zapisu w formacie tekstowym. Nie jest to zaskoczeniem, ponieważ format hdf5 został stworzony specjalnie do efektywnego operowania na dużych zbiorach danych, i wykonuje takie operacje bardziej optymalnie. Nie udało się niestety przeprowadzić badań nad większą liczbą obrazów, ponieważ ilość dostępnej pamięci była niewystarczająca. Jak można łatwo policzyć, wyznaczając ok. 400 regionów na jeden obraz, każdy region o wymiarach 224x224x3, otrzymujemy dla pięciu obrazów ponad 300 milionów liczb. Doliczyć do tego należy jeszcze sieć neuronową, która również zawiera mnóstwo parametrów (użyty w tym przykładzie VGG16 zajmuje ponad 400MB) i otrzymujemy ogromne zapotrzebowanie na pamięć. Warto zaznaczyć, że wszystkie obliczenia były dokonywane tutaj za pomocą technologii CUDA – możliwość zrównoleglenia obliczeń na karcie graficznej pozwala skrócić czas wykonywania obliczeń ponad 6-krotnie w porównaniu do wykonywania ich na zwykłym procesorze.

Tablica 6.1: Czas wykonania w sekundach oraz zużycie pamięci w zależności od liczby obrazów oraz formatu zapisu. Do badań użyto modelu VGG16.

<i>Obrazy</i>	Format	t_1	t_2	t_3	t_c	\bar{t}_c	Pamięć(GB)
1	text	0.73	11.66	3.18	14.84	15.86	1.235
		0.98	13.24	3.64	16.88		1.30
	hdf5	0.87	10.74	1.01	11.75	11.60	1.36
		0.72	10.59	0.86	11.45		0.96
2	text	0.93	17.25	4.35	21.60	21.06	0.83
		1.01	16.02	4.50	20.52		0.91
	hdf5	1.02	15.12	0.91	16.33	16.20	0.71
		0.93	14.45	0.71	16.06		1.12
3	text	3.98	21.77	8.51	30.28	35.38	0.93
		2.86	31.4	9.00	40.48		1.36
	hdf5	2.80	50.9	3.30	54.23	54.57	1.37
		2.91	51.2	3.67	54.90		1.39
5	text	4.31	50.4	12.28	62.71	76.40	1.56
		5.15	59.16	30.93	90.08		1.81
	hdf5	4.65	66.98	5.86	72.84	77.27	1.73
		4.82	75.88	5.83	81.70		1.64

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.

ζ	metoda							
			alg. 3			alg. 4, $\gamma = 2$		
	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. Kollia, M. A. Nicolaou, A. Papaioannou, G. Zhao, and I. Kotšia, “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. Kollia 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] “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 *Compute Unified Device Architecture*

ILSVRC TODO

API Interfejs Programowania Aplikacji (ang. *Application Programming Interface*)

GUI Graficzny Interfejs Użytkownika (ang. *Graphical User Interface*)

X

Źródła

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

```
1 def extract_features_from_image(self, img_dir, classes,
2     IoU_cutoff_object = 0.5, IoU_cutoff_not_object = 0.5):
3     anno = pd.read_csv("etykiety.csv")
4     image_pos, image_neg, info_pos, info_neg =
5         [], [], [], []
6     for irow in range(anno.shape[0]):
7         row = anno.iloc[irow, :]
8         path = os.path.join(img_dir, row["fileID"] + "."
9             "jpg")
10        image = imageio.imread(path)
11        orig_h, orig_w, _ = image.shape
12        img = self.warp(image, self.warped_size)
13        orig_nh, orig_nw, _ = img.shape
14        regions = ss.get_region_proposal(img, min_size
15            =50)[::-1]
16        for ibb in range(row["nobj"]):
17            name = row["bbx_{}_.name".format(ibb)]
18            if not classes[name].get():
19                break;
20            multx, multy = orig_nw/orig_w, orig_nh/
21                orig_h
22            true_xmin = row["bbx_{}_.xmin".format(
```

```
    ibb )]* multx
18      true_ymin      = row[ "bbx_{ }_ymin" . format(
        ibb )]* multy
19      true_xmax      = row[ "bbx_{ }_xmax" . format(
        ibb )]* multx
20      true_ymax      = row[ "bbx_{ }_ymax" . format(
        ibb )]* multy
21      object_found_TF = 0
22      _image1 = None
23      for r in regions:
24          prpl_xmin , prpl_ymin , prpl_width ,
25              prpl_height = r[ "rect" ]
26          IoU = ss.get_IOU(prpl_xmin , prpl_ymin ,
27              prpl_xmin + prpl_width , prpl_ymin +
28                  prpl_height ,
29                      true_xmin , true_ymin ,
30                          true_xmax ,
31                          true_ymax)
32          img_bb = np.array(img[prpl_ymin :
33              prpl_ymin + prpl_height ,
34                  prpl_xmin :
35                      prpl_xmin +
36                          prpl_width])
37          if IoU > IoU_cutoff_object:
38              found_object=[name, prpl_xmin ,
39                  prpl_ymin , prpl_width ,
40                      prpl_height ,row[ "fileID" ]]
41              info_pos.append(found_object)#
42                  encode( 'utf-8')
43              image_pos.append(img_bb)
44              background = [ "background" ,
45                  prpl_xmin , prpl_ymin , prpl_width
46                      , prpl_height ,row[ "fileID" ]]
```

```
34         if background in info_neg:
35             back_indx=info_neg.index(
36                 background)#if the
37                 regions figures as the
38                 background sample,
39                 delete it
40             del info_neg[back_indx] #
41                 from both annotations
42             del image_neg[back_indx] #
43                 and images list
44             elif IoU < IoU_cutoff_not_object:
45                 background=["background", prpl_xmin
46                               , prpl_ymin, prpl_width,
47                               prpl_height, row['fileID']]
48             if background not in info_neg:
49                 info_neg.append(background)
50                 image_neg.append(img_bb)
51             images = image_pos+image_neg
52             infos= np.array(info_pos+info_neg, dtype=h5py.
53                             string_dtype())
54             features = self.warp_and_create_cnn_feature(images)
55             return (infos,features)
```

Zawartość dołączonej płyty

Do pracy dołączona jest płyta CD z następującą zawartością:

- praca (źródła L^AT_EXowe i końcowa wersja w pdf),
- źródła programu,
- dane testowe.

Spis rysunków

2.1	Przykład prostej sieci w pełni połączonej	7
2.2	Filtr o wymiarach 5x5x3	8
2.3	Mapy aktywacji nakładane na siebie	8
2.4	Max pooling	9
2.5	ReLU.	9
2.6	Tangens hiperboliczny.	9
2.7	Architektura sieci LeNet	10
2.8	Architektura sieci AlexNet. Sieć została tutaj podzielona na dwie równoległe części, ponieważ obliczenia były dzielone pomiędzy dwie jednostki graficzne.	11
2.9	Architektura sieci VGG	12
3.1	Diagram przypadków użycia	19
4.1	Interfejs programu	25
4.2	Wybrany obraz z bazy PASCAL VOC	27
4.3	Etykieta opisująca rysunek 4.2	28
5.1	Definicja funkcji <code>create_annotations</code>	34
5.2	Definicja funkcji <code>get_region_proposal</code>	35
5.3	Definicja funkcji <code>warp_and_create_cnn_feature </code>	35
6.1	Wizualizacja segmentacji metodą Felzenszwalba.	38
6.2	Wizualizacja wyznaczonych regionów.	38

Spis tabelic

1.1	Rozmiary zbiorów danych służących do rozpoznawania twarzy na przestrzeni lat	2
4.1	Fragment przykładowych wyekstrahowanych danych	29
6.1	Czas wykonania w sekundach oraz zużycie pamięci w zależności od liczby obrazów oraz formatu zapisu. Do badań użyto modelu VGG16.	40
7.1	Opis tabeli nad nią.	42