



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI I ROBOTYKI

Praca dyplomowa inżynierska

*Klasyfikacja elementów morfometrycznych krwi przy pomocy głębokich
sieci neuronowych.*

*Classification of blood morphometric elements using deep neural
networks.*

Autor:

Ilona Tomkowicz

Kierunek studiów:

Automatyka i Robotyka

Opiekun pracy:

dr hab. inż. Joanna Jaworek-Korjakowska, prof. AGH

Kraków, 2019

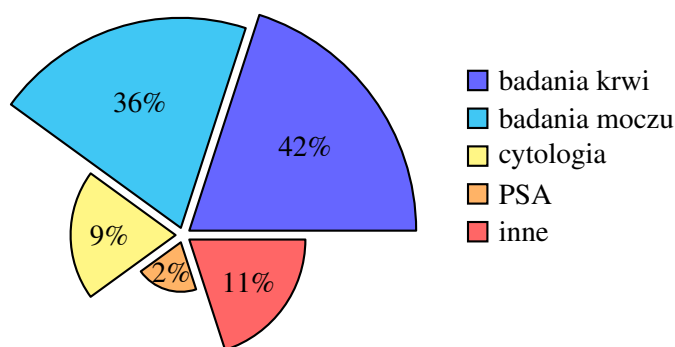
Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Spis treści

1. Wprowadzenie	7
1.1. Motywacja pracy	8
1.2. Cel i zrealizowane zadania	9
1.3. Zawartość pracy	9
2. Analiza problemu badawczego	11
2.1. Aspekt medyczny - analiza elementów krwi	11
2.2. Głębokie sieci neuronowe	12
2.2.1. Zasada działania sieci neuronowych	12
2.2.2. Początki sieci konwolucyjnych	13
2.2.3. Warstwy sieci konwolucyjnych	14
2.2.4. Analiza parametrów sieci konwolucyjnej	16
2.2.5. Uczenie	16
2.2.6. Sieci pretrenowane - opcjonalnie	17
2.3. Algorytmy klasyfikacji elementów morfologicznych	18
2.3.1. Sieć trenowana od podstaw z warstwą redukcyjną maksymalizującą [17]	18
2.3.2. Sieć trenowana od podstaw z warstwą dropoutu [18]	19
2.3.3. Pretrenowany model InceptionV3 [19]	21
3. System do klasyfikacji elementów morfologicznych	23
3.1. Przygotowanie danych	23
3.2. Powiększanie zbioru - opcjonalne, na razie nie używam	23
3.3. Dobór parametrów	24
3.3.1. Ograniczenie overfittingu	24
4. Analiza wyników	25
5. Podsumowanie	27
5.1. Kierunki dalszych badań	27

1. Wprowadzenie

Morfologia krwi (ang. *complete blood count*) jest jednym z najczęściej przeprowadzanych badań. Dostarcza ona informację o komórkach krwi pacjenta, w tym liczbę komórek każdego typu krwinek i wartość stężenia hemoglobiny. Zgodnie z zaleceniami powinno się wykonywać je przynajmniej raz do roku w celach profilaktycznych. Jest to też jedno z pierwszych badań stosowanych w diagnostyce schorzeń. Według raportu GUS 42% osób decydujących się na badanie laboratoryjne wybiera właśnie badanie krwi [1].



Biorąc pod uwagę stan ludności i ograniczoną liczbę personelu medycznego w szpitalach manualne wykonywanie tego typu badań jest problematyczne i zajmuje dużo czasu. Poprzez usunięcie czynnika ludzkiego można uzyskać większą poprawność i zwiększyć produktywność personelu medycznego. Celem pracy jest zbudowanie narzędzia do automatycznej klasyfikacji białych krwinek opartego na głęboko uczonych konwolucyjnych sieciach neuronowych. Na wejściu do sieci wprowadzane są zdjęcia pojedynczych krwinek, wykonane pod mikroskopem. W tym celu została wykorzystana baza danych na licencji MIT, zawierająca cztery klasy krwinek najliczniej występujące w składzie krwi. Każde zdjęcie jest oryginalnie przyporządkowane do odpowiedniej klasy, zgodnie z widniejącym na nim elementem morfologicznym.

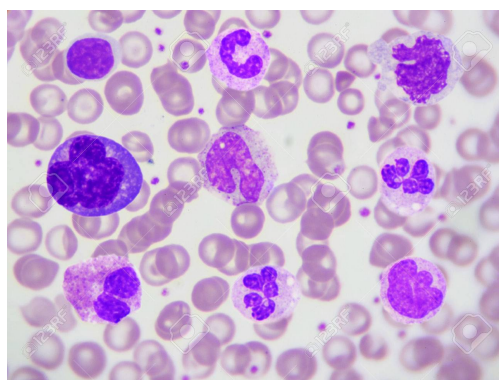
Przed użyciem bazę podzielono na rozłączne zbiory: uczący, weryfikacyjny i testowy. Za pomocą zbiorów uczącego i weryfikacyjnego przetrenowano i nastrojono klasyfikator. Dzięki danym ze zbioru testowego zawierającego zdjęcia nigdy nie wprowadzane na wejście sieci sprawdzono skuteczność zastosowanej metody.

Oczekiwanym wynikiem pracy jest zbudowanie sieci neuronowej określającej z jak najlepszą dokładnością jaki typ krwinki znajduje się na zdjęciu, a następnie modyfikacja zarówno parametrów sieci jak i danych wejściowych w celu zbadania wpływu zmian na działanie modelu.

Program został napisany w języku Python, który jest dobrze przystosowany do przetwarzania, analizy i modelowania danych. Do implementacji sieci użyta została biblioteka Keras, która jest wysokopoziomym API biblioteki TensorFlow.

1.1. Motywacja pracy

W celu prawidłowego zdiagnozowania chorób często konieczne jest zbadanie liczby krwinek białych danego typu. Obecnie proces ten jest wykonywany manualnie za pomocą hemocymetru lub automatycznie z użyciem np. technologii VCS, pomiarem impedancji czy pomiarami z użyciem laseru.



Rys. 1.1. Widok krwinek badanych pod mikroskopem [2].

Ciekawą alternatywą dla tych metod byłoby zastosowanie automatycznego zliczania komórek opartego na klasyfikacji przynależności do danego typu na podstawie analizy obrazów przez sieć neuronową. Byłaby to metoda nie wymagająca ingerencji czynnika ludzkiego, jak to ma miejsce w przypadku badania manualnego, a jednocześnie tańsza niż stosowane pomiary automatyczne. Zmniejszenie liczby ręcznych procesów i nadmiaru próbek podczas rutynowych badań w zwolniłoby miejsce dla innych ważnych zadań, zwiększyło wydajność i poprawiło jakość analiz. W pełni zautomatyzowany proces zmniejszyłby indywidualne ryzyko błędów.

Bazą do zbudowania takiego narzędzia byłaby sieć rozpoznająca typ krwinki na zdjęciu i właśnie tą częścią zajmuje się niniejszy projekt. W pracy zdecydowano się na sieć konwolucyjną głęboko uczoną i w zależności od parametrów zbadano precyzyjność jej działania. W tym celu przetestowano wiele kombinacji doboru składowych modelu, a poniżej opisano kilka najciekawszych przypadków.

1.2. Cel i zrealizowane zadania

Aby zbudować narzędzie do klasyfikacji krwinek białych najpierw przeprowadzono analizę przydatności takiego rozwiązania na rynku oraz sprawdzono jakie metody są obecnie wykorzystywane w badaniach krwi. Analiza wykazała zasadność stworzenia tego typu automatyzacji.

Następnym krokiem było zapoznanie się ze stanem obecnej wiedzy na temat sieci neuronowych głęboko uczonych oraz sieci konwolucyjnych i przegląd dostępnych rozwiązań podobnych problemów. Po wyciągnięciu wniosków z zebranych informacji przystąpiono do planowania i implementacji modelu sieci. Wybrano model osiągający najlepsze wyniki i przystąpiono do badania wpływu doboru jego parametrów na dokładność klasyfikacji.

Wynikiem końcowym pracy jest klasyfikator osiągający X% skuteczność na zbiorze testowym oraz wnioski wyciągnięte z badań nad zależnością skuteczności od doboru hiperparametrów.

1.3. Zawartość pracy

W rozdziale 2 przedstawiono teoretyczną analizę problemu badawczego wraz z kilkoma przykładowymi rozwiązaniami zadania klasyfikacji wizyjnej na podstawie najnowszych publikacji.

Rozdział 3 zawiera opis implementacji programu i zastosowanych metod, zaś rozdział 4 zestawienie i analizę uzyskanych wyników, po którym następuje podsumowanie przeprowadzonego badania.

2. Analiza problemu badawczego

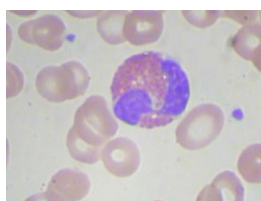
2.1. Aspekt medyczny - analiza elementów krwi

Krwinki białe, będące komórkami systemu odpornościowego, w zależności od funkcji pełnionej w organizmie można podzielić na pięć grup, z których cztery mają znaczny udział procentowy w składzie krwi.

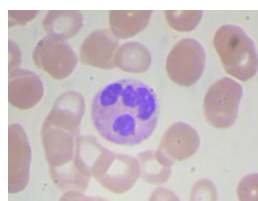
Tabela 2.1

Nazwa	Neutrofil	Eozynofil	Limfocyt	Monocyt
Udział %[3]	54-62	1-6	25-33	2-10
Średnica μm [3]	10-12	10-12	7-15	15-30

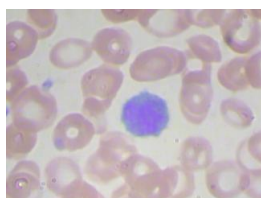
Baza wykorzystana w pracy zawiera zdjęcia w każdej z tych kategorii. Najważniejsze cechy, po których można rozpoznać daną klasę to wielkość komórki, kształt oraz typ jądra komórkowego. Neutrofile mają jądra podzielone na segmenty, eozynofile jądra dwupłatkowe, limfocyty są okrągłe z kulistymi jądrami, a monocyty z elipsoidalnymi [4]. Poniżej przedstawiono przykładowe zdjęcia pochodzące z bazy.



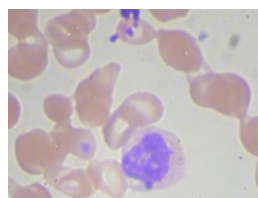
(a)



(b)



(c)



(d)

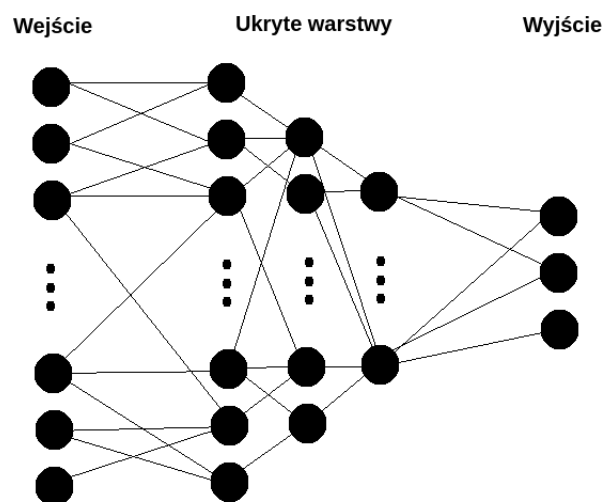
Rys. 2.1. Zdjęcia przedstawiające (a) eozynofil, (b) neutrofil, (c) limfocyt, (d) monocyt.

2.2. Głębokie sieci neuronowe

Cechą charakterystyczną głębokich sieci neuronowych (ang. *deep learning artificial neural networks*) jest uczenie się wysokopoziomowej reprezentacji wzorców. Struktura sieci składa się zwykle z kilku do kilkunastu warstw (ang. *layers*), chociaż czasem zdarzają się implementacje bardzo głębokich sieci z więcej niż 1000 ukrytych warstw [5]. Jedną z pierwszych sieci tego typu miała trzy gęsto połączone ukryte warstwy, gdzie warstwę ukrytą należy rozumieć jako część sieci nie będącą wejściem ani wyjściem z układu [6]. W rozpoznawaniu obrazów początkowe warstwy służą identyfikacji ogólnych i generycznych wzorców, jak rozpoznawanie krawędzi. Im głębsza warstwa tym bardziej kształty przez nie zapamiętywane przypominają reprezentacje obiektów znane człowiekowi, na przykład oczy, nos w przypadku rozpoznawania twarzy.

2.2.1. Zasada działania sieci neuronowych

Sztuczna sieć neuronowa (ang. *artificial neural network*, ANN) jest to układ przetwarzania danych, składający się z warstw sztucznych neuronów, połączonych synapsami o konkretnych wagach. Neurony wykonują pewne operacje matematyczne na wejściowych danych, a wynik przesyłany jest do kolejnego rzędu neuronów lub do wyjścia układu.



Rys. 2.2. Wizualizacja przykładowej struktury sieci neuronowej.

Typy sieci neuronowych ze względu na kierunek przepływu danych:

- Jednokierunkowa (ang. *feedforward*) - dane w sieci przepływają tylko w kierunku od wejścia do wyjścia. Do tego typu należą sieci konwolucyjne.
- Rekurencyjna - przepływ danych między dwoma połączonymi neuronami odbywa się w dowolnym kierunku.

Funkcję, realizowaną przez całą sieć można zapisać wzorem [7]:

$$Y = W_k X \quad (2.1)$$

gdzie,

W_k – macierz współczynników wagowych połączeń między neuronami. Ma wymiar $[k \times n]$, gdzie k - liczba warstw, n - liczba neuronów w jednej warstwie,

X – wektor danych wejściowych,

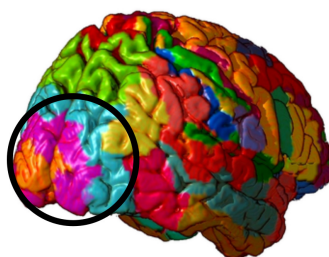
Y – wektor sygnałów wyjściowych.

Celem trenowania sieci neuronowej jest dobranie wartości w macierzy W_k tak, aby odwzorowała wektor X w wektor Y .

2.2.2. Początki sieci konwolucyjnych

Konwolucyjne sieci neuronowe (ang. *convolutional neural networks*, CNN) są typem sieci głęboko uczonej. Zbudowane na bazie perceptronu wielowarstwowego (ang. *multilayer perceptron*, MLP) [8] będącego najpopularniejszym typem ANN w latach 80' [9]. MLP są używane w większości modeli na końcu konwolucyjnej sieci neuronowej i może zawierać kilka tego typu warstw. Pełni rolę przekodowania wartości cech zwracanych przez warstwy konwolucyjne na klasy, do których przynależy obiekt [10]. Podczas gdy MLP charakteryzują się tym, że są w pełni połączone, co oznacza że każdy neuron z warstwy jest powiązany z każdym neuronem z kolejnej warstwy, CNN nie są już połączone tak gęsto. Pozwala to między innymi na ograniczenie w pewnym stopniu podatności na zjawisko nadmiernego dopasowania (ang. *overfitting*).

Powstanie sieci tego typu zostało zainspirowane budową części mózgu odpowiedzialnej za odbiór wrażeń wizyjnych - kory wzrokowej [11]. Narząd ten zawiera liczne drobne i gęsto ułożone komórki nerwowe. Zajmuje trzy pola Brodmanna - obszary, na który została podzielona struktura mózgu [12].



Rys. 2.3. Struktura mózgu podzielona na pola Brodmanna z zaznaczoną korą wzrokową [13].

Informacja wizyjna jest przekazywana z jednego obszaru do drugiego, przy czym każdy kolejny obszar jest bardziej wyspecjalizowany niż poprzedni. Pola różnią się między sobą funkcjami, przez co neurony w danym obszarze wykonują tylko konkretne zadania. Przykładowo obszar, do którego w pierwszej

kolejności trafiają informacje wizyjne, nazwany bruzdą ostrogową zachowuje lokalizację przestrzenną widzianych obiektów. Przekazuje on informację do asocjacyjnej kory wzrokowej, która z kolei jest odpowiedzialna za rozpoznawanie kształtów, rozmiarów i kolorów, a potem do innych obszarów mózgu zajmujących się kojarzeniem obiektu z jego reprezentacją w pamięci. Z kolei trzeciorzędowa kora wzrokowa (ang. *middle temporal*) rozpoznaje ruch obiektów, a przedczołowa (ang. *dorsomedial prefrontal cortex*) ruch samego obiektu rejestrującego obraz [14]. Analogicznie działają sieci konwolucyjne, w których kolejne warstwy są odpowiedzialne za detekcję różnych typów wzorców, a dane są przekazywane między warstwami i analizowane na różnych poziomach abstrakcji.

Sieci konwolucyjne znalazły zastosowanie w rozpoznawaniu obrazów, ze względu na inwariancję względem translacji oraz zdolność uczenia się wzorców lokalnych [15]. Dane wejściowe są w postaci tensora trójwymiarowego, a operacja konwolucji (oznaczona gwiazdką), która zachodzi w warstwach sieci może być opisana poniższym równaniem [8]:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n). \quad (2.2)$$

gdzie,

I – dane wejściowe,

K – jądro (ang. *kernel*),

S – wyjście, mapa cech (ang. *feature map*).

2.2.3. Warstwy sieci konwolucyjnych

Neurony w sieci są pogrupowane w warstwy. Typowe obliczenia w warstwie CNN składają się z trzech etapów. W pierwszym przeprowadzane jest kilka równoległych konwolucji, których wyniki nazywamy liniowymi aktywacjami. W kolejnym etapie, nazywanym detekcyjnym, każda aktywacja liniowa poddawana jest działaniu nieliniowej funkcji aktywacji. Na koniec używana jest funkcja redukująca (ang. *pooling function*) [8].

W tabeli 2.2 zaprezentowano przykładowe działanie konwolucji dla obrazu o głębokości 3 (np. RGB) z zastosowaniem filtru 3x3 z krokiem równym 2, co znaczy że filtr jest stosowany co 2 piksele. Zmniejsza to rozmiar ramki do 3x3. Wyjściem tej operacji jest mapa cech (ang. *feature map*). Z każdym filtrem (inaczej jądrem, (ang. *kernel*) powiązana jest wartość błędu (ang. *bias*). Nie zastosowano dopełniania macierzy zerowymi wierszami i kolumnami na brzegach (ang. *padding - same*), więc z tego powodu także następuje redukcja rozmiaru - do 2x2.

Po przejściu przez konwolucję macierz poddawana jest funkcji aktywacyjnej. Jej celem jest obliczenie ważonej sumy macierzy, dodanie do niej wartości błędu i zdecydowanie czy dana wartość powinna być uznana za aktywną czyli brana pod uwagę w dalszym działaniu.

Funkcja redukcyjna zastępuje wartość wyjściową w danym węźle pewną wartością obliczoną na podstawie wyjść sąsiednich neuronów. W ten sposób zmniejszana jest ilość próbek, a także parametrów sieci,

Tabela 2.2. Przykładowe działanie konwolucji.

wejście: (5x5x3)

x[:, :, 1]
1 2 3 1 1
0 1 4 0 2
1 2 0 2 0
4 0 4 0 0
0 1 1 0 1

filtr: (3x3x3)

f1[:, :, 1]
1 0 -1
1 0 -1
1 0 -1

wynik: (2x2x3)

a1[:, :, 1]
1 4
4 4

wyjście: (2x2x1)

y[:, :, 1]
4 12
9 11

x[:, :, 2]

0 2 1 0 3
0 0 2 1 0
1 1 2 0 0
2 0 1 0 0
3 1 0 0 1

f1[:, :, 2]

1 -1 1
-1 0 -1
1 -1 1

a1[:, :, 2]

-1 4
1 2

x[:, :, 3]

3 0 4 1 0
0 1 2 0 2
2 3 1 0 2
0 1 2 4 1
3 1 3 1 0

f1[:, :, 3]

0 1 0
1 1 1
-1 1 0

a1[:, :, 3]

4 4
4 5

co zmniejsza nakład obliczeniowy i redukuje overfitting. Przykład zastosowania funkcji redukcyjnej typu max pooling znajduje się w tabeli 2.3. Dzięki tej operacji uzyskuje się niezmiennosć wyjścia względem małych translacji wejścia.

Tabela 2.3

macierz wejściowa						wynik
1	10	2	6	6	2	
0	2	4	3	5	4	
4	0	1	2	8	1	
0	2	5	4	9	0	10 8
4	5	3	7	5	3	5 9
2	1	5	0	2	7	

Na warstwy konwolucyjne nakładane są warstwy gęsto połączone (ang. *dense layers*), służące do klasyfikacji. Na ich wejściu wymagane są dane jednowymiarowe, a wyjściem konwolucji są dane trójwymiarowe. Z tego powodu łączy się je warstwą spłaszczającą (ang. *flatten layer*), która transformuje macierze cech w wektor cech.

Ostatnia warstwa w pełni połączona powinna mieć wymiar równy liczbie klas, do których jest klasyfikowany zbiór danych oraz odpowiednią funkcję aktywacyjną. Dla klasyfikacji binarnej używana jest S-funkcja, a do niebinarnej funkcja softmax.

DOPISAC, to co w komentarzu

2.2.4. Analiza parametrów sieci konwolucyjnej

Parametrami, które można regulować w sieci są rozmiar filtru, liczba filtrów, krok filtracji, dopełnianie zerami, dobór funkcji aktywacyjnej i redukcyjnej. Parametry te mają wpływ na skuteczność działania sieci i mimo, że w przypadku funkcji da się powiedzieć które formuły mają większe prawdopodobieństwo powodzenia, to jednak nie ma ścisłych reguł ich doboru.

Przykładowe funkcje aktywacyjne:

- progowanie

$$\begin{aligned} Y < th, A &= 0 \\ Y \geq th, A &= 1 \end{aligned} \quad (2.3)$$

gdzie,

Y – wynik sumy ważonej i błędu,

th – próg aktywacji,

A – aktywacja.

- funkcja liniowa

$$A = cY \quad (2.4)$$

gdzie,

c – stała,

- S-funkcja

$$A = \frac{1}{1 + e^{-Y}} \quad (2.5)$$

- ReLu

$$A = \max(0, Y) \quad (2.6)$$

Przykładowymi funkcjami reducyjnymi są: maksimum, minimum, średnia, norma L^2 , średnia ważona odległością od centralnego piksela. Najczęściej stosowana jest jednak funkcja maksimum, gdyż daje najlepsze efekty [16].

2.2.5. Uczenie

Proces uczenia sieci neuronowej dzieli się na epoki. Liczba epok jest regulowalnym parametrem i od przyjętej wartości zależy jakość działania modelu. Zbyt mała liczba epok skutkuje niedotrenowaniem (model mógłby klasyfikować lepiej), a zbyt duża przetrenowaniem (model zna zbiór na którym trenował bardzo dobrze, ale słabo radzi sobie z nowymi zbiorami). Poniżej przedstawiono kolejne procesy zachodzące podczas jednej epoki.

Początkowo ustalane są wagi sieci W_k i błędów b_k przez inicjalizację małymi liczbami losowymi. W pierwszej części treningu odbywa się propagacja w przód (ang. *forward propagation*), która polega na przejściu przez sieć w kierunku od wejścia do wyjścia i obliczeniu liniowego kroku:

$$y_1 = X_0 W_1 + b_1 \quad (2.7)$$

gdzie,

X_1 – macierz wejściowa,

W_1 – macierz wag,

b_1 – błąd (ang. *bias*),

y_1 – pierwszy liniowy krok.

Następnie zbiór liniowych kroków przechodzi przez funkcje aktywacyjne, wprowadzając do modelu cechy nieliniowe i pozwalając na reprezentację bardziej skomplikowanych odwzorowań.

Po zakończonej propagacji w przód następuje etap propagacji wstecznej (ang. *backward propagation*), mający na celu poprawę wartości wag. Na podstawie funkcji błędu - różnicy między wyjściem z modelu (predykcją), a oczekiwanym wyjściem - szacuje się jakość rozwiązania. Używając pochodnej funkcji błędu względem wag minimalizuje się błąd metodą najszybszego spadku. Krok spadku jest determinowany przez parametr nazywany tempem uczenia (ang. *learning rate*).

Najczęściej nie wszystkie dane przepływają przez sieć jednocześnie. W przypadku dużych zbiorów danych dzieli się je na mniejsze podzbiory (ang. *batches*), które przepływają kolejno przez sieć. Liczebność tego typu podzbioru jest parametrem modelu i wpływa na jakość klasyfikacji. Liczba iteracji definiuje ile podzbiorów ma przejść przez sieć od wejścia do wyjścia układu i spowrotem, aby epoka została uznana za skończoną.

2.2.6. Sieci pretrenowane - opcjonalnie

Trening sieci neuronowej jest procesem czasochłonnym. Co więcej, wymaga zgromadzenia odpowiedniej ilości opisanych danych, co bywa problematyczne. Z tego powodu zaczęto szukać metod, dzięki którym będzie można ten proces uprościć i stosować te same narzędzia do różnych problemów. Przeniesienie uczenia (ang. *transfer learning*) jest stosowane w sieciach neronowych przez użycie pretrenowanych modeli. Tego typu model jest trenowany na dużym zbiorze danych i zawierającym nawet kilka milionów próbek i kilkadziesiąt tysięcy klas.

Korzystając z faktu, że coraz głębsze warstwy sieci uczą się i rozpoznają coraz bardziej skomplikowane i szczegółowe wzorce na obrazie można zedytować raz przetrenowany model do przeznaczenia ogólnego. Należy zamrozić początkowe warstwy - rozpoznające generyczne wzorce - aby nie nadpisać ich wag oraz na nich dołożyć kolejne warstwy mające za zadanie nauczenie się szczegółów typowych dla konkretnego zbioru zdjęć. Dzięki temu można użyć pretrenowanego modelu do rozpoznawania kształtów w bazie zdjęć niezwiązanych wcale z oryginalnym zbiorem, na którym został przetrenowany.

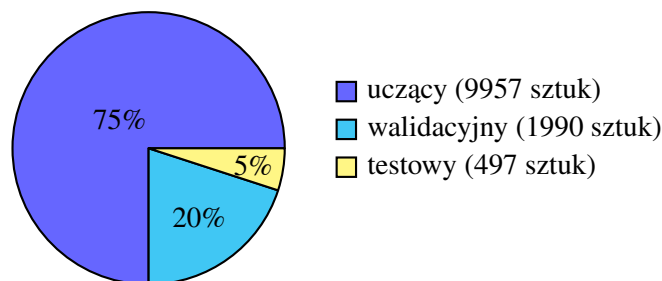
2.3. Algorytmy klasyfikacji elementów morfologicznych

Podejścia do problemu klasyfikacji krwinek białych spotykane w literaturze wykazują pewne wspólne cechy charakterystyczne. W przypadku sieci trenowanych od podstaw (ang. *trained from scratch*) najlepsze efekty osiągały modele bazujące na warstwach konwolucji i normalizacji wsadowej (ang. *batch normalisation*). W ten sposób osiągnięto nawet 80% skuteczności na zbiorze testowym. Mimo, że skuteczność sieci budowanych od podstaw jest wysoka, to użycie sieci pretrenowanych daje jeszcze lepsze rezultaty, bo ponad 85%. Poniżej przedstawiono najważniejsze informacje o wybranych, najskuteczniejszych algorytmach.

2.3.1. Sieć trenowana od podstaw z warstwą redukcyjną maksymalizującą [17]

– Przygotowanie danych:

Oryginalnie baza została podzielona w stosunku 75:25 na zbiory uczący i walidacyjny. W publikacji nie ma zbioru testowego, co spowodowało brak informacji o jakości działania sieci. Z tego powodu więc eksperyment przedstawiony w artykule został przeze mnie powtórzony z dokładnością co do algorytmu, ale ze zmodyfikowanym podziałem bazy danych. Pierwotny zbiór walidacyjny o liczebności 2487 elementów podzielono w stosunku 80:20 na zbiory walidacyjny i testowy.



Ramki zostały przeskalowane do zakresu wartości pikseli [0:1] i rozmiaru 128x128. Następnie przetasowano każdy ze zbiorów i podzielono je na podzbiory (ang. *batches*) o liczebności 32 ramek każdy.

– Struktura sieci (DODAĆ TUTAJ TABULATOR DLA TEJ TABELI CZY CO TO JEST):

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	(None, 128, 128, 3)	0
conv2d_1 (Conv2D)	(None, 128, 128, 12)	912
batch_normalization_2 (BatchNor	(None, 128, 128, 12)	48

sekwencja warstw, która powtarza się pięciokrotnie:

conv2d_2 (Conv2D)	(None, 128, 128, 12)	156
batch_normalization_3 (BatchNor	(None, 128, 128, 12)	48
conv2d_3 (Conv2D)	(None, 128, 128, 12)	156
conv2d_4 (Conv2D)	(None, 128, 128, 12)	1308
batch_normalization_4 (BatchNor	(None, 128, 128, 12)	48
batch_normalization_5 (BatchNor	(None, 128, 128, 12)	48
concatenate_1 (Concatenate)	(None, 128, 128, 24)	0
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 24)	0

.....

conv2d_17 (Conv2D)	(None, 4, 4, 12)	444
batch_normalization_18 (BatchNo	(None, 4, 4, 12)	48
conv2d_18 (Conv2D)	(None, 4, 4, 12)	156
conv2d_19 (Conv2D)	(None, 4, 4, 12)	1308
batch_normalization_19 (BatchNo	(None, 4, 4, 12)	48
batch_normalization_20 (BatchNo	(None, 4, 4, 12)	48
concatenate_6 (Concatenate)	(None, 4, 4, 24)	0
glob_average_pooling2d_1 (Globa	(None, 24)	0

dense_1 (Dense)	(None, 4)	100
-----------------	-----------	-----

=====
Total params: 35,858

Trainable params: 35,102

Non-trainable params: 756

– Parametry sieci i treningu:

Przyjęty optyimizator (CZYLI CO?) to RMSprop, szybkość uczenia to $2e-5$, funkcja liczenia błędu to (ang. *binary crossentropy*). Uczenie trwało 25 epok, każda epoka trwała 310 iteracji, a walidacja 62 iteracje.

– Wyniki eksperymentu:

Wyniki powtórzonego eksperymentu pokrywają się z wynikami w publikacji w zakresie dokładności zbioru uczącego i walidacyjnego.

train acc: 92%

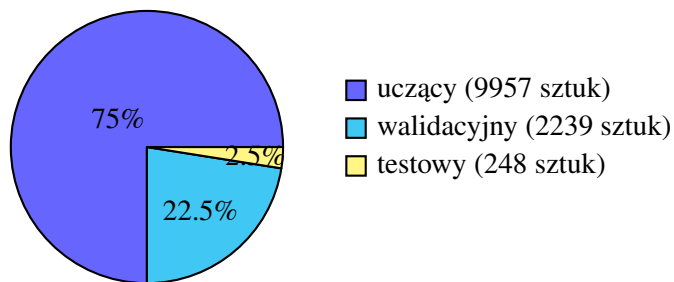
val acc: 90%

test acc: 75%

2.3.2. Sieć trenowana od podstaw z warstwą dropoutu [18]

– Przygotowanie danych:

Oryginalnie baza została podzielona w stosunku 75:25 na zbiory uczący i walidacyjny. Zbiór testowy uzyskano oddzielając 10% danych walidacyjnych.



Ramki zostały znormalizowane do zakresu wartości pikseli [0:1] i odchylenia standardowego równego 1. Zmniejszono ich rozmiar o połowę do 160x120. Następnie podzielono je na podzbiory o liczebności 16 ramek każdy.

– Struktura sieci:

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 120, 160, 3)	0
sekwencja warstw, która powtarza się czterokrotnie:		
conv2d_1 (Conv2D)	(None, 60, 80, 16)	1216
batch_normalization_1 (Batch Normalization)	(None, 60, 80, 16)	64
dropout_1 (Dropout)	(None, 60, 80, 16)	0
.....		
flatten_1 (Flatten)	(None, 320)	0
sekwencja warstw, która powtarza się trzykrotnie:		
dense_1 (Dense)	(None, 32)	10272
dropout_5 (Dropout)	(None, 32)	0
.....		
dense_4 (Dense)	(None, 4)	36
Total params: 16,732		
Trainable params: 16,668		
Non-trainable params: 64		

– Parametry treningu:

Model był trenowany przez 200 epok.

– Wyniki eksperymentu:

tabela:

train acc: 99%

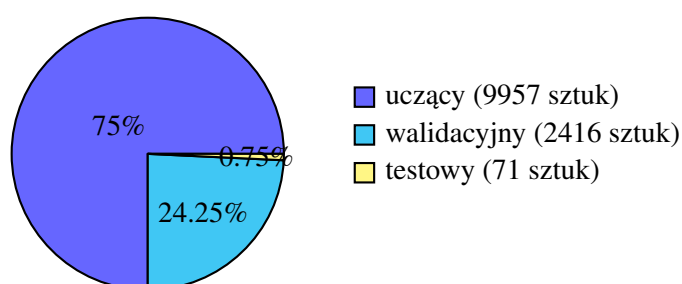
val acc: 97%

test acc: 83%

2.3.3. Pretrenowany model InceptionV3 [19]

– Przygotowanie danych:

Zbiór uczący i testowy z oryginalnej bazy zostały użyte w całości i bez modyfikacji. Jako zbiór testowy wykorzystano mały podzbiór dołączony do bazy, którego liczebność stanowi niecałe 3% zbioru walidacyjnego.



c Ramki zostały użyte w oryginalnym rozmiarze 320x240, bez normalizacji i standaryzacji wartości pikseli. Podzielono je na podzbiory o liczebności 32 ramki każdy.

– Struktura sieci:

Znaczącą różnicą jest zastosowanie klasyfikatora pięciowyjściowego, co pozwala na przypisanie do ramki klasy "żadna".

Layer (type)	Output Shape	Param #
inception_v3 (Model)	(None, 6, 8, 2048)	21802784
global_average_pooling2d_1	(None, 2048)	0
sekwencja warstw, która powtarza się trzykrotnie:		
dense_1 (Dense)	(None, 512)	1049088
dropout_5 (Dropout)	(None, 512)	0
.....		
dense_4 (Dense)	(None, 5)	10272
Total params: 16,732		
Trainable params: 16,668		
Non-trainable params: 64		

- Parametry treningu:

Model był trenowany przez 30 epok.

- Wyniki eksperymentu:

tabela:

train acc: 99%

val acc: 87%

test acc: 86%

3. System do klasyfikacji elementów morfologicznych

3.1. Przygotowanie danych

Ramki wczytane bezpośrednio z bazy źródłowej do programu są w formacie RGB. Piksele oryginalnych obrazów mogą przyjmować wartości z zakresu od 0-255. Dla lepszego działania sieci neuronowej zaleca się normalizację wartości pikseli do małego zakresu, najlepiej 0-1 oraz ustandaryzowanie tak, aby można było traktować dane wejściowe jako rozkład Gaussa o średniej 0 i odchyleniu standardowym 1 [20].

$$z = \frac{x - \mu}{\sigma} \quad (3.1)$$

gdzie,

x – oryginalna wartość piksela,

μ – średnia z wartości pikseli w ramce,

σ – odchylenie standardowe wartości pikseli w ramce,

z – wartość piksela po ustandaryzowaniu.

3.2. Powiększanie zbioru - opcjonalne, na razie nie używam

W przypadku małych zbiorów danych, rozumianych jako zbiór liczący kilka tysięcy elementów często stosowaną praktyką jest poszerzanie zbioru danych (*ang. data augmentation*). Ma ona na celu bezpośrednio powiększenie ilości danych wprowadzanych do modelu, a pośrednio polepszenie rezultatów klasyfikacji. Jednym z korzystnych efektów tego działania jest redukcja zjawiska nadmiernego dopasowania (*ang. overfitting*). Objawia się ona zmniejszeniem różnicy między błędem zbioru, na którym się trenuje model a błędem zbioru, na którym model jest testowany. Szczególnie dużą poprawę w tym aspekcie obserwuje się właśnie dla sieci typu CNN [21].

Jednym ze sposobów transformacji jest elastyczna deformacja w przestrzeni danych (*ang. data-space elastic deformation*). Obrazy w oryginalnym zbiorze danych zostają poddane losowym transformacjom, z założeniem, że zachowane zostają informacje o przynależności do danej klasy. Daje najlepsze rezultaty w porównaniu do poszerzania w przestrzeni cech (*ang. feature-space augmentation*) [21]. Definiuje ona

znormalizowany obszar losowego przemieszczenia $u(x, y)$, który dla każdego piksela w obrazie (x, y) definiuje wektor przemieszczenia R_w [21]:

$$R_w = R_0 + \alpha u \quad (3.2)$$

gdzie,

R_w – lokalizacja piksela w obrazie wyjściowym,

R_0 – lokalizacja piksela w oryginalnym obrazie,

α – wielkość przesunięcia w pikselach.

Nie jest wskazane używanie zbioru powiększonego z użyciem dużych transformacji. Dla bazy MNIST przesunięcia większe niż $\alpha \geq 8$ pikseli skutkuje w pewnej części przypadków utratą informacji o przynależności do danej klasy. Jest to definiowane jako brak zdolności do rozpoznania i zaklasyfikowania danej ramki przez człowieka [21].

W przypadku rozpoznawania typów komórek augmentacja z zastosowaną zmianą skali może skutkować pogorszeniem dokładności klasyfikacji, gdyż na każdy rodzaj komórki ma swoją typową wielkość. Skorzystanie z tej cechy do nauczania się rozpoznawania elementów z pewnością podnosi poziom precyzji działania algorytmu. Zaburzenie tej cechy przez manipulację skalą zdjęcia będzie skutkować utraceniem tej informacji. Zbiór danych zostanie powiększony, jednak stanie się to kosztem utraty pewnych pomocnych danych.

3.3. Dobór parametrów

3.3.1. Ograniczenie overfittingu

CNN są sieciami posiadającymi poza warstwami konwolucyjnymi i redukcyjnymi warstwy w pełni połączone (ang. *fully-connected network*). Charakteryzują się one tym, że każdy neuron posiada połączenie z dowolnym innym neuronem w poprzedniej warstwie. To sprawia, że są podatne na zjawisko nadmiernego dopasowania.

4. Analiza wyników

5. Podsumowanie

5.1. Kierunki dalszych badań

Bibliografia

- [1] Departament Badań Społecznych Główny Urząd Statystyczny w Krakowie. „Zdrowie i ochrona zdrowia w 2016 r”. W: 2019.
- [2] Jarun Ontakrai. https://www.123rf.com/photo_82668549_white-blood-cells-in-in-blood-smear-analyse-by-microscope.html.
- [3] Paul R. Wheeler, H. George Burkitt i Victor G. Daniels. „Functional histology: A text and colour atlas”. W: 1979.
- [4] Alberts B i in. „Leukocyte also known as macrophages functions and percentage breakdown”. W: Edinburgh: Churchill Livingstone, 2002.
- [5] Kaiming He i in. „Deep Residual Learning for Image Recognition”. W: 2015.
- [6] Geoffrey E. Hinton, Simon Osindero i Yee Whye Teh. „A Fast Learning Algorithm for Deep Belief Nets”. W: 2006.
- [7] Tadeusiewicz R. „Sieci neuronowe”. W: Kraków, Wykład plenarny XXXII Zjazdu Fizyków Polskich, 1993.
- [8] „Deep learning”. W: The MIT Press, 2016.
- [9] Philip D. Wasserman i Tom J. Schwartz. „Neural networks. II. What are they and why is everybody so interested in them now?” W: 1988.
- [10] Alex Krizhevsky, Ilya Sutskever i Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks”. W: *NIPS*. 2012.
- [11] Masakazu Matsugu i in. „Subject independent facial expression recognition with robust face detection using a convolutional neural network”. W: 2003.
- [12] Korbinian Brodmann. „Vergleichende Lokalisationslehre der Großhirnrinde : in ihren Prinzipien dargestellt auf Grund des Zellenbaues”. W: 1985.
- [13] Mark Dow. http://lcn1.uoregon.edu/~dow/Space_software/renderings.html.
- [14] Gopal Kalpande. „<https://medium.com/@gopalkalpande/biological-inspiration-of-convolutional-neural-network-cnn-9419668898ac>”. W: 1979.
- [15] François Chollet. „Deep Learning with Python”. W: 2017.

- [16] Dominik Scherer, Andreas C. Müller i Sven Behnke. „Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition”. W: *ICANN*. 2010.
- [17] Maksim Drobchak. „<https://www.kaggle.com/drobchak1988/blood-cell-images-acc-92-val-acc-90>”. W: 2019.
- [18] nh4cl. „<https://www.kaggle.com/placidpanda/deep-learning-from-scratch-insights>”. W: 2018.
- [19] Luke Sung Uk Jung. „<https://www.kaggle.com/jcruxsu/blood-cell-85-kernal-with-inception-v3>”. W: 2019.
- [20] E. Kreyszig. „Advanced Engineering Mathematics”. W: 1979.
- [21] Sebastien C. Wong i in. „Understanding Data Augmentation for Classification: When to Warp?”. W: 2016.