AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt dyplomowy

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, 2020

Spis treści

1. Wstęp 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 morfometrii 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 jako podstawowe bloki konstrukcyjne 14

2.2.4. Analiza parametrów sieci konwolucyjnej 16

2.2.5. Uczenie głębokich sieci neuronowych 18

2.2.6. Używanie sieci wytrenowanych 19

2.3. Przykłady algorytmów do klasyfikacji elementów krwi 19

2.3.1. Sieć trenowana z warstwą redukcyjną maksymalizującą 20

2.3.2. Sieć trenowana z regularyzacją 24

2.3.3. Wytrenowana architektura sieci InceptionV3 26

3. System do klasyfikacji elementów morometrycznych krwi 29

3.1. Przygotowanie bazy danych 29

3.1.1. Zastosowanie metody augmentacji danych 29

3.1.2. Podział na podzbiory 30

3.2. Zaproponowana struktura sieci neuronowej 31

3.2.1. Cechy architektury zaczerpnięte z modelu VGGNet. 31

3.2.2. Cechy architektury zaczerpnięte z modelu U-Net. 32

3.2.3. Cechy architektury zaczerpnięte z modelu ResNet. 34

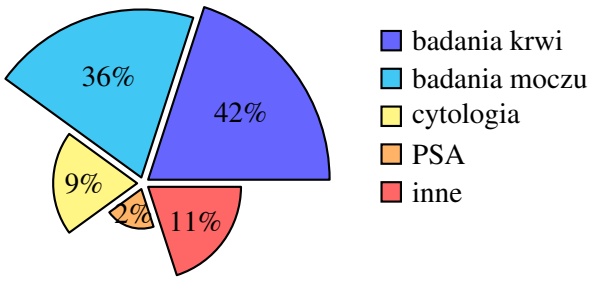
3.3. Dobór parametrów 36

3.3.1. Dobór optymalizatora i szybkości uczenia 37

3.3.2. Ograniczenie przeuczenia modelu 38
3.3.3. Wpływ przygotowania danych na jakość klasyfikacji 39
3.3.4. Wybór funkcji redukcyjnej 41
3.3.5. Wybór funkcji aktywacyjnej warstw konwolucyjnych 44
4. Analiza wyników 47
5. Podsumowanie 49
5.1. Kierunki dalszych badań 49

1. Wstęp

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 reportu GUS 42% osób decydujących się na badanie labolatoryjne wybiera właśnie badanie krwi (Rys. 1.1) [21].



Rys. 1.1. Typy wykonanych badań labolatoryjnych na podstawie [21].

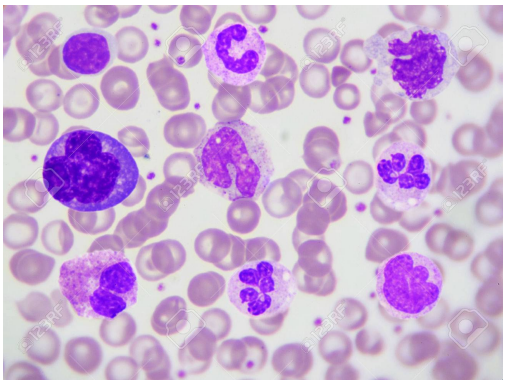
Biorąc pod uwagę stan ludności i ograniczoną liczbę personelu medycznego w szpitalach manualne wykonywanie tego typu badań jest problematyczne i zajmuje dużo czasu. Poprzez usunięcie czynnika ludzkiego można uzyskać większą poprawność i zwiększyć poduktywność personelu medycznego. **Celem pracy jest zbudowanie narzędzia do automatycznej klasyfikacji białych krwinek opartego na głębokich konwolucyjnych sieciach neuronowych. Oczekiwanym wynikiem pracy jest zbudowanie sieci neuronowej określającej z jak najlepszą doładnośćią 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.** 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 [26].

Przed użyciem bazę podzielono na rozłączne zbiory: uczący, walidacyjny i testowy. Za pomocą zbiorów uczącego i walidacyjnego wyuczono i dostrojono klasyfikator. Dzięki danym ze zbioru testowego,

zawierającego zdjęcia nigdy nie wprowadzane na wejście sieci, sprawdzono skuteczność zastosowanej metody.
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 wysokopoziomowym API biblioteki TensorFlow.

1.1. Motywacja pracy

Morfologia krwi to podstawowe badanie wykonywane w celu diagnostyki lub profilaktyki. Analizuje ono cechy i liczebność zarówno krwinek czerwonych, które stanowią ok.40% objętości krwi,jak i białych, stanowiących ok. 0,1% objętości krwi [38]. Wyniki badania krwinek białych mogą dać informację o zakażenu wirusowym, bakteryjnym lub pasożytniczym organizmu [37]. Ocena parametrów tego typu krwinek jest też pomocna w diagnostyce stanu immunologicznego, białaczki i chorób rozrostowych. Zwiększenie sumarycznej liczby leukocytów do ok. 20 000 świadczy o zakażeniu organizmu. Podwyższona liczba neutrofili oznacza ostre zakażenie bakteryjne lub stan po urazie z masywnym krwotokiem. W przypadku gdy pacjent cierpi na alergię lub jest zakażony pasożytami w organiźmie będzie za dużo eozynofili. Skutkiem zakażenia wirusowego i przewlekłych infekcji bakteryjnych jest wyższy poziom limfocytów. Wzrost liczby monocytów świadczy o chorobie zapalnej, a ich spadek o zaburzeniu odporności [37].
Znaczenie diagnostyczne mają bezwzględne liczby poszczególnych rodzajów krwinek białych, zawartych we krwi obwodowej [1]. Obecnie proces ten jest wykonywany manualnie za pomocą hemocymetru (Rys. 1.2) lub automatycznie z użyciem np. technologii VCS, pomiarem impedancji czy pomiarami z użyciem lasera. Precyzyjne pomiary liczby poszczególnych rodzajów krwinek białych możliwe są jedynie dzięki metodom automatycznym, przy pomocy analizatorów hematologicznych [1].



Rys. 1.2. Widok krwinek badanych pod mikroskopem [30].

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 zwolniłoby miejsce dla innych ważnych zadań, zwiększyło wydajność i poprawiło jakość analiz. W pełni zautomatyzowany proces zmniejszyłby indywidualne ryzyko błędów.
Bazę do zbudowania takiego narzędzia stanwiłaby 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.
W celu zrealizowania pracy inżynierskiej wykonano następujące zadania:

- analiza problemu badawczego, zapoznanie się z zagadnieniem badań morfologicznych,
- zapoznanie się z obecnym stanem wiedzy na temat sieci neuronowych głęboko uczonych oraz sieci konwolucyjnych i przegląd dostępnych rozwiązań podobnych problemów w lieraturze,
- wybór bazy danych i zapoznanie się z jej zawartością,
- zaplanowanie struktury sieci i implementacja modelu w języku Python,
- wybór modelu osiągającego najlepsze wyniki i przebadanie wpływu doboru 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 rodziale 2 przedstawiono teoretyczną analizę problemu badawczego zarówno w aspekcie medycznym jak i matematycznym. Zawiera on opis działania sieci głębokich ze szczególnym uwzględnieniem sieci konwolucyjnch. Na podstawie najnowszych publikacji wyodrębniono kilka najciekawszych rozwiązań klasyfikacji wizyjnej elementów morfometrycznych i zamieszczono w tym rozdziale.
Rozdział 3 zawiera opis implementacji rozwiązania oraz szczegóły strukturalne i parametryczne zastosowanych modeli sieci neuronowych. Następnie opisane są wyniki eksperymentów zmiany parametrów modelu i ich wpływ na jakość klasyfikacji.
Rozdział 4 obejmuje analizę uzyskanych wyników z wizualizacją wybranych filtrów sieci i map aktywacji dla wybranych zdjęć z bazy. W końcowym rozdziale następuje podsumowanie przeprowadzonego badania.

2. Analiza problemu badawczego

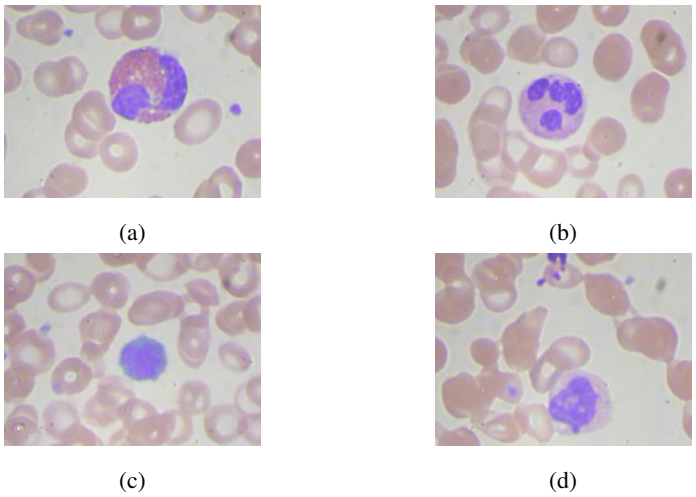
2.1. Aspekt medyczny - analiza morfometrii krwi

Krwinki białe, będące komórkami systemu odpornościowego, w zależności od funkcji pełnionej w organiźmie można podzielić na pięć grup, z których cztery mają znaczny udział procentowy w składzie krwi (2.1).

Tabela 2.1. Udział procentowy typów krwinek białych w składzie krwi [43].

Nazwa	Neutrofil	Eozynofil	Limfocyt	Monocyt
Udział %	54-62	1-6	25-33	2-10
Średnica μm	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. Neurofile mają jądra podzielone na segmenty, eozynofile jądra dwupłatowe, limfocyty są okrągłe z kulistymi jądrami, a monocyty z elipsodalnymi [2]. Rys. 2.1 przedstawia przykładowe zjęcia pochodzące z bazy.



(a) (b)

(c) (d)

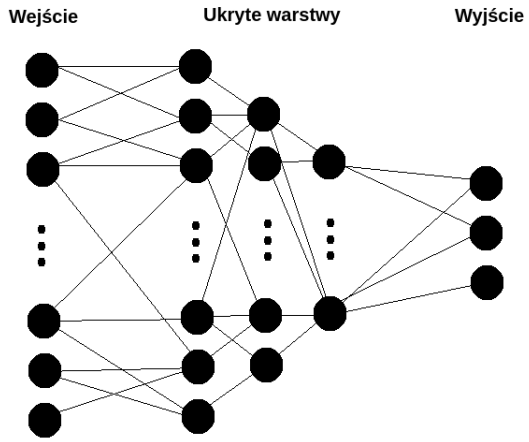
Rys. 2.1. Zdjęcia przedstawiające: (a) eozynofil, (b) neurofil, (c) limfocyt, (d) monocyt.

2.2. Głębokie sieci neuronowe

Cechą charakterystyczną głębokich seci neuronowych (ang. *deep learning 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 ukytych warstw [12]. Jedna z pierwszych sieci bazująca na reprezentacji wysokopoziomowej, maszyna Boltzmanna (ang. *restricted boltzmann machine*, RBM), została opisana w 2006 roku. Miała tylko trzy gęsto połączone ukryte warstwy, gdzie warstwę ukrytą należy rozumieć jako część sieci nie będącą wejściem ani wyjściem z układu [13]. 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 rozpoznawaniu 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 (Rys. 3.4). Neurony wykonują pewne operacje matematyczne na wejściowych danych, a wynik przesyłany jest do kolejnego rzędu neuronów lub do wyjścia układu. Schemat stryktury sieci został przedstawiony na Rys. 3.4).



Rys. 2.2. Wizualizacja przykładowej struktury sieci neurnowej.

Typy sieci neuronowych z podziałem na kierunek przepływu danych:

– Jednokierunkowa (ang. *feedfoward*) - dane w sieci przepływają tylko w kierunku od wejścia do wyjścia, a uczenie sieci odbywa się dzięki zastosowaniu propagacji wstecznej. Do tego typu należą sieci konwolucyjne.

– Rekurencyjna (ang. *recurrent*) - przepływ danych między dwoma połączonymi neuronami odbywa się w dowolnym kierunku, dopuszcza są też cykle.

– Ze sprzężeniem zwrotnym (ang. *regulatory feedback*) - dane przepływają od wejścia do wyjścia, a w celu poprawy wag sieci stosowane jest ujemne sprzężenie zwrotne.

– Samoorganizujące się mapy (ang. *self organizing maps*) - dane nie przepływają przez sieć. Sieć dopasowuje się do struktury zbiorów danych, na których jest uczona.

Funkcję, realizowaną przez całą sieć można zapisać wzorem (2.1) [31]:

$$Y = W_k X \tag{2.1}$$

gdzie,

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

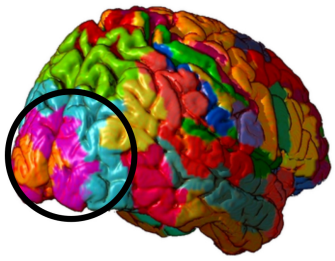
X – wektor danych wejściowych,

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) należą do najczęściej używanych głębokich sieci neuronowych w wizji komputerowej. Zbudowane na bazie perceptonu wielowarstwowego (ang. *multilayer perceton*, MLP) będącego najpopularniejszym typem ANN w latach 80' [7, 42]. 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 [23]. 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 [25]. Narząd ten zawiera liczne drobne i gęsto ułożone komórki nerwowe. Zajmuje trzy pola Brodmana) - obszary, na który została podzielona struktura mózgu (Rys. 2.3) [3].
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



Rys. 2.3. Struktura mózgu podzielona na pola Brodmanna z zaznaczoną korą wzrokową [9].

zajmujących się kojarzeniem obiektu z jego reprezentacją w pamięci. Z kolei trzeciorzędowa kora wzrokowa (ang. *middle temporal*) rozpoznaje ruch obieków, a przedczołowa (ang. *dorsomedial prefrontal cortex*) ruch samego obiektu rejestrującego obraz [18]. 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 kowolucyjne znalazły zastosowanie w rozpoznawaniu obrazów, ze względu na inwariancję względem translacji oraz zdolność uczenia się wzorców lokalnych [5]. Dane wejściowe są w postaci tensora trójwymiarowego, a operacja konwolucji (oznaczona gwiazdką), która zachodzi w warstwach sieci może być opisana równaniem (2.2), [7]:

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

gdzie,

I – dane wejściowe,

K – jądro (ang. *kernel*),

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

2.2.3. Warstwy jako podstawowe bloki konstrukcyjne

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*) [7].

Na Rys. 2.4 zaprezentowano przykładowe działanie konwolucji dla obrazu o głębokości 3 (np. RGB). Zastosowano filtr 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 dopełnianie

macierzy zerowymi wierszami i kolumnami na brzegach (ang. *padding - valid*), więc z tego powodu także następuje redukcja rozmiaru - do 2x2.

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

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

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

filtr: (3x3x3)

f1[:,:,1]

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

f1[:,:,2]

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

f1[:,:,3]

0	1	0
1	1	1
-1	1	0

wynik: (2x2x3)

a1[:,:,1]

1	4
4	4

a1[:,:,2]

-1	4
1	2

a1[:,:,3]

4	4
4	5

wyjście: (2x2x1)

y[:,:,1]

4	12
9	11

Rys. 2.4. Przykład działania konwolucji

.

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 braną pod uwagę w dalszym działaniu.

Funkcja redukcyjna zastępuje wartość wyjściową w danym węźle pewną warością obliczoną na podstawie wyjść sąsiednich neuronów. W ten sposób zmiejszana jest ilość próbek, a także parametrów sieci, co zmniejsza nakład obliczeniowy i redukuje overfitting. Przykład zasosowania funkcji redukcyjnej typu max pooling znajduje się na Rys. 2.5. Zastosowano podział na bloki 3x3. Dzięki tej operacji uzyskuje się niezmienność wyjścia względem małych translacji wejścia.

1	10	2	6	6	2
0	2	4	3	5	4
4	0	1	2	8	1
0	2	5	4	9	0
4	5	3	7	5	3
2	1	5	0	2	7

10	8
5	9

Rys. 2.5. Przykład działania redukcji max pooling

.

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 [5].
W celu zapobiegnięcia przetrenowaniu sieci używa się warstw typu dropout, które usuwają pewne połączenia między neuronami. Dzięki temu sieć uczy się cech bardziej ogólnych oraz będzie mniej podatna na osiąganie wysokiej skuteczności na zbiorze walidacyjnym, ale niskich na zbiorze testowym [36].
Temu samemu służą warstwy normalizacji wsadowej (ang. *batch normalisation*), które standaryzują dane wyjściowe z poprzedniej warstwy przez nadanie im postaci rozkładu normalnego N(0,1). W przeciwieństwie do dropoutu nie powoduje utraty niektórych informacji przez usunięcie połączeń, lecz dodaje szum do każdej funkcji aktywacyjnej. Skutkiem jej stosowania jest zwiększenie niezależności warstw od siebie, generalizacja działania sieci oraz zwiększenie współczynnika uczenia. Najlepszym podejściem jest używanie obu typów wastw zmniejszających ryzyko przetrenowania [16].

2.2.4. Analiza parametrów sieci konwolucyjnej

Sieć konwolucyjna ma wiele parametrów, które można regulować w celu dobrania jak najlepszego sposobu działania. Parametry te mają wpływ na skuteczność i mimo, że w przypadku doboru funkcji aktywacji czy redukcji da się powiedzieć które formuły mają większe prawdopodobieństwo powodzenia, to jednak nie ma ścisłych reguł ich doboru.
Parametry warstwy konwolucyjnej to ilość filtrów, ich rozmiar, krok filtracji, dopełnianie zerami bądź jego brak i dobór funkji aktywacyjnej. W przypadku warstw redukcyjnych, na przykład używanej w niniejszej pracy redukcji maksymalizującej, dobiera się wielkość bloków, krok i obecność dopełnienia zerami. Warstwa gęsto połączona ma regulowaną ilość segmentów i funkcję aktywacyjną, zaś w dropout ustala się współczynnik przepuszczalności danych.
Podczas propagacji wstecznej w fazie uczenia używany jest algorytm do znajdowania minimum funkcji błędu. Minimum to można znaleźć na wiele sposobów, jednak najskuteczniejszym sposobem do szybkiej zbieżności jest użycie ADAM lub innej techniki adaptacyjnej [40]. Poniżej przedstawiono przykładowe funkcje optymalizacyjne.

– Naszybszy spadek (2.3),

$$\theta_{i+1} = \theta_i - \alpha \nabla F(\theta_i) \tag{2.3}$$

gdzie,

θ – argument minimalizowanej funkjcji w i-tym kroku algorytmu,
α – szybkość uczenia (ang. *learining rate*),

F – funkcja błędu.

– najszybszy spadek z regulacją bezwładności uczenia (2.4),

$$\theta_{i+1} = \theta_i - V_i V_{i+1} = \gamma V_i + \alpha \nabla F(\theta_i) \tag{2.4}$$

gdzie,

γ – bezwładność uczenia (ang. *momentum*)

– adapcyjne oszacowanie momentu (ang. *adaptive moment estimation*, ADAM) (2.5), [20].

$$\theta_{i+1} = \theta_i - \frac{\alpha}{\sqrt{v_i} + \epsilon} m_i \tag{2.5}$$

gdzie,

m_i – pierwszy moment (wartość oczekiwana) gradientu funkcji,
v_i – drugi moment (wariancja) gradientu funkcji.

– apropagacja średniokwadratowa (ang. *root mean square propagation*, RMSprop) (2.6), [24].

$$\theta_{i+1} = \theta_i - \frac{\alpha}{\sqrt{v_i + \epsilon}} \nabla F(\theta_i) \tag{2.6}$$

Najlepiej działającymi funkcjami aktywacyjnymi są funkcje nieliniowe, gdyż pozwalają na reprezentację bardziej skomplikowanych odwzorowań. Ważną cechą, jaką powinna posiadać funkcja aktywacyjna jest różniczkowalność. Dzięki temu, że funkcja ma pochodną możliwa jest propagacja wsteczna. Ten krok zawiera w sobie liczenie gradientów błędów względem wag w celu optymalizacji działania sieci neuronowej i zredukowania wartości błędu do minimum. Przykładowe funkcje aktywacyje:

– progowanie (2.7),

$$\begin{aligned} Y < th, A = 0 \\ Y \geq th, A = 1 \end{aligned} \tag{2.7}$$

gdzie,

Y – wynik sumy ważonej i błędu,
th – próg aktywacji,
A – aktywacja.

– funkcja liniowa (2.8),

$$A = cY \tag{2.8}$$

gdzie,

c – stała,

– eksponencjalna (2.9),

$$A = e^Y \tag{2.9}$$

– tangens hiperboliczny (2.10),

$$A = tanh(Y) = \frac{e^Y - e^{-Y}}{e^Y + e^{-Y}} \tag{2.10}$$

– ReLu (2.11).

$$A = max(0, Y) \tag{2.11}$$

– S-funkcja (ang. *sigmoid*) (2.12),

$$A = \frac{1}{1 + e^{-Y}} \tag{2.12}$$

– softmax (2.13),

$$A = \frac{e^Y}{\sum_{j=0}^{k} e^{Y_j}} \tag{2.13}$$

gdzie,

k – ilość klas,

Przykładowymi funkcjami reducyjnymi są: maksimum, minimum, średnia, norma L^2, średnia ważona odległością od centralego piksela. Najczęściej stosowana jest jednak funkcja maksimum, gdyż daje najlepsze efekty [33].

2.2.5. Uczenie głębokich sieci neuronowych

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 popagation)*, która polega na przejściu przez sieć w kierunku od wejścia do wyjścia i obliczeniu liniowego kroku (2.14):

$$y_1 = X_0 W_1 + b_1 \tag{2.14}$$

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 podstwie 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*).
Najczęsciej 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. Używanie sieci wytrenowanych

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. Przenoszenie 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. Przykłady algorytmów do klasyfikacji elementów krwi

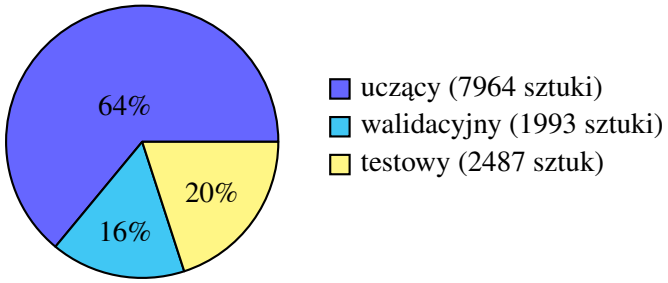
Podejścia do problemu klasyfikacji krwinek białych spotykane w literaturze wykazują pewne wspólne cechy charakterystyczne. W przypadku sieci trenowanych od podstaw (ang. *trained from scratch*) najlepsze efekty osiągały modele bazujące na warstwach konwolucji i normalizacji wsadowej. W ten sposób osiągnięto 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 z warstwą redukcyjną maksymalizującą

Przytoczony algorytm w całości bazuje na publikacji [10]. Poniżej przedstawiono najważniejsze fragmenty oraz autorski eksperyment na podstwie cytowanego modelu.

– Przygotowanie danych:

Oryginalnie baza została podzielona w stosunku 75:25 na zbiory uczący i walidacyjny. W publikcji nie ma zbioru testowego, co spowodowało brak informacji o jakości działania sieci. Z tego powodu więc eksperyment przedstawiony w artkule został przeze mnie powtórzony z dokładnością co do algorytmu, ale ze zmodyfiowanym podziałem bazy danych. Z pierwotnego zbioru treningowego o liczebności wydzielono 20% obrazów wyodrębiając zbiór walidacyjny (Rys. 2.6).



Rys. 2.6. Podział bazy danych.

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:

```
_________________________________________________________________
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:

W warstwach konwolucyjnych użyto funkcji aktywacyjnej ReLu, z dopełnianiem ramek zerami na brzegach (ang. *padding - same*) i wielkością filtra zmieniającą się od 1 do 3. Warstwy normalizacji wsadowej mają bezwładność uczenia równą 0.85. Użyte warstwy redukcyjne z funkcją maksymalizującą obliczną na podmacierzach wielkości 2x2. Jako klasyfikatora użyto warstwy gęsto połączonej z czterema wyjściami, a jako funkcję aktywującą - sigmoid. Jest to niespójna decyzja, gdyż dla więcej niż dwóch wyjść w problemach niebinarnych powinno się używać funkcji softmax, a nie sigmoid.

Przyjęty algorytm optymalizacji to RMSprop z szybkością uczenia 2×10^{-5}, funkcja liczenia błędu to binarna entrpia krzyżowa (ang. *binary cross entropy*). Uczenie trwało 25 epok, każda epoka trwała 248 iteracji, a walidacja 62 iteracje.

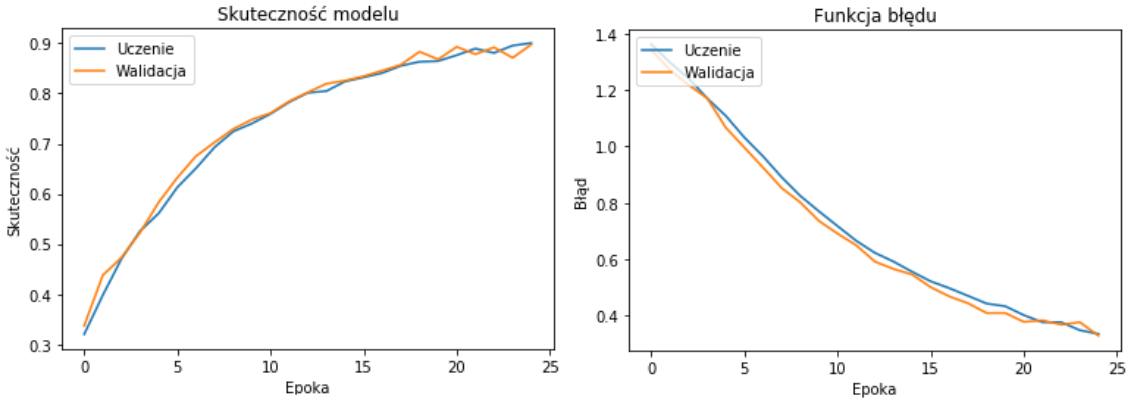
– Wyniki klasyfikacji przy użyciu podanej architektury:

Proces uczenia nie wykazuje nieprawidłowych cech wskazujących na przetrenowanie lub niedotrenowanie (Rys. 2.7). Skuteczność zbioru walidacyjnego nadąża za skutecznością zbioru treningowego, nie ma wyraźnych zmian skokowych skuteczności na przestrzeni epok, a krzywa uczenia ma przybliżony kształt funkcji eksponencjalnie rosnącej. Oznacza to poprawny przebieg uczenia.

Wyniki dotyczące skuteczności (2.2) pokrywają się z wynikami w publikacji w zakresie dokładności zbioru uczącego i walidacyjnego. Wygenerowana w eksperymencie macierz pomyłek zbioru walidacyjnego (2.3) i obliczone na jej podstawie parametry (4.3) potwierdzają teorię, że użyty klasyfikator jest błędny i sieć rozpoznaje jedynie dwie klasy z niezerową skutecznością.

Tabela 2.2. Skuteczność modelu.

typ zbioru	**treningowy**	**walidacyjny**	**testowy**
skuteczność [%]	90	89	68



Rys. 2.7. Zależność skuteczności i błędu od epoki trenowania w modelu z klasyfikatorem binarnym.

.

Tabela 2.3. Macierz pomyłek zbioru testowego modelu z klasyfikatorem binarnym.

		predykcja			
		E	L	M	N
klasa	E	0	607	0	16
	L	0	606	0	14
	M	0	607	0	13
	N	0	613	0	11

Tabela 2.4. Parametry mierzące jakość klasyfikacji na zbiorze testowym modelu z klasyfikatorem binarnym.

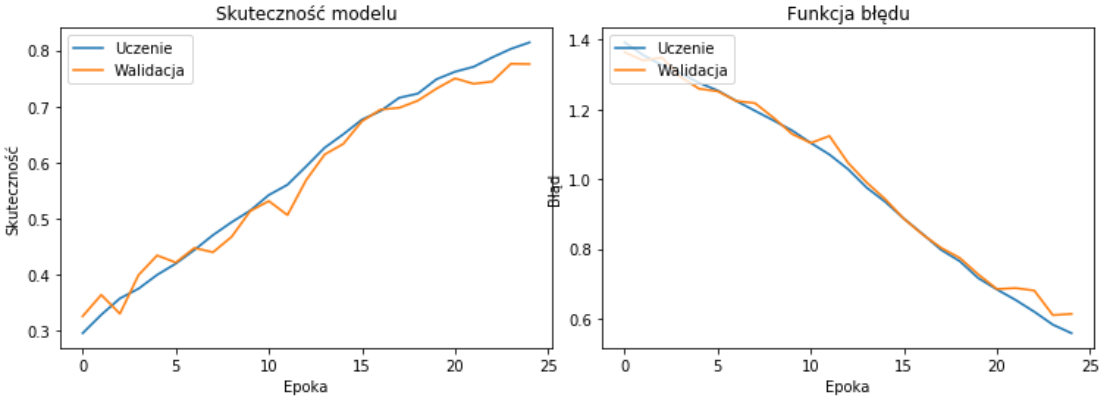
Parametr	**Precyzja**	**Czułość**	**Miara F1**	**Ilość próbek**
klasa eozynofil (E)	0.00	0.00	0.00	623
klasa limfocyt (L)	0.25	0.98	0.40	620
klasa monocyt (M)	0.00	0.00	0.00	620
klasa neutrofil (N)	0.20	0.02	0.03	624

– Modyfikacja oryginalnej sieci w celu poprawy klasyfikacji:

Zamieniono klasyfikator binarny na wieloklasowy z funkcją aktywacyjną softmax. Zmiana typu funkcji aktywacyjnej przy wyjściu z modelu wpłynęła w niewielkim stopniu negatywnie na proces uczenia (Rys. 2.8). Sumaryczne skuteczności na zbiorze treningowym jak i waliacyjnym spadły (2.5), ale macierz pomyłek dla zbioru walidacyjnego znacząco się poprawiła (2.6, 2.7).

Tabela 2.5. Skuteczność modelu z klasyfikatorem niebinarnym.

typ zbioru	treningowy	walidacyjny	testowy
skuteczność [%]	81	77	70



Rys. 2.8. Zależność skuteczności i błędu od epoki trenowania w modelu z klasyfikatorem niebinarnym

.

Tabela 2.6. Macierz pomyłek zbioru testowego modelu z klasyfikatorem niebinarnym.

		predykcja			
		E	L	M	N
klasa	E	144	167	173	139
	L	152	165	170	133
	M	140	179	160	141
	N	147	151	172	154

Tabela 2.7. Parametry mierzące jakość klasyfikacji na zbiorze testowym modelu z klasyfikatorem niebinarnym.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.25	0.23	0.24	623
klasa limfocyt (L)	0.25	0.27	0.26	620
klasa monocyt (M)	0.24	0.26	0.25	620
klasa neutrofil (N)	0.27	0.25	0.26	624

Sprawdzenie szczegółów dotyczących klasyfiacji, takich jak skuteczność dla poszczególnych klas jest bardzo ważnym krokiem ewaluacyji modelu. W podanym przypadku pozwoliło to na poprawę osiągnięć sieci, mimo że pozornie sumaryczna skuteczność na zbiorach treningowym i walidacyjnym zmalały. Mogą zostać poprawiona przez zwiększenie ilości epok treningu i strojenie parametrów sieci.

2.3.2. Sieć trenowana z regularyzacją

Przytoczony algorytm w całości bazuje na publikacji [28]. Poniżej przedstawiono najważniejsze fragmenty oraz autorski eksperyment na podstwie cytowanego modelu.

– Przygotowanie danych:

Oryginalnie baza została podzielona w stosunku 75:25 na zbiory uczący i walidacyjny. Zbiór testowy uzyskano oddzielając 10% danych walidacyjnych (**??**). W powtórzonym eksperymencie bazę podzielono tak jak w poprzednio rozpatrywanym algoytmie 2.3.1.

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:

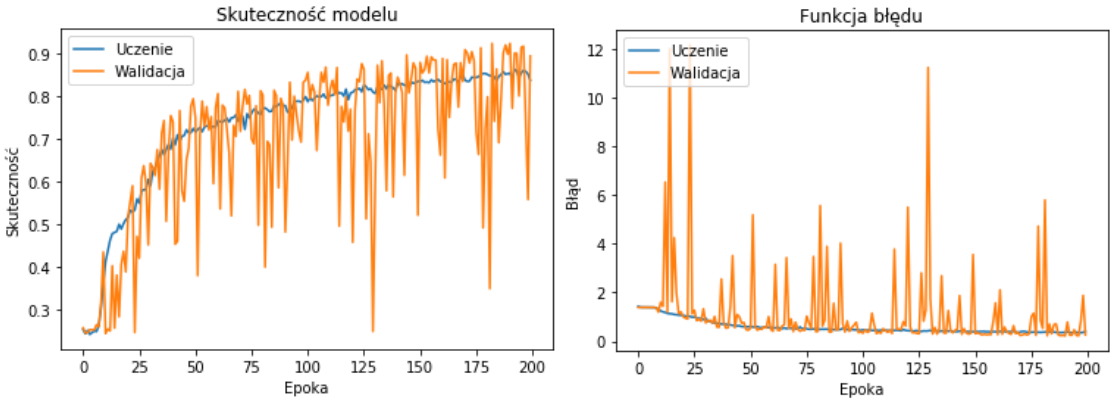
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 120, 160, 3)       0
_________________________________________________________________
sekwencja warstw, która powtarza się czterokrotnie:
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 60, 80, 16)        1216
batch_normalization_1 (Batch (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 sieci i treningu:

Liczba filtrów wastw konwolucyjnych zmieniała się o połowę od 16 do 4 przy stałej wielkości filtra 5x5. Zastosowano dopełnianie zerami, funkcję ReLu w celu aktywacji oraz dropout na poziomie 0,2. Model był trenowany przez 200 epok.

– Wyniki klasyfikacji przy użyciu podanej architektury:

Duża liczba warstw głęboko połączonych na końcu sieci wprowadza ryzyko przetrenowania, nawet w sytuacji użycia warstw odrzucających niektóre połączenia neuronowe. Aby zweryfikować wpływ warstw gęsto połączonych na proces uczenia powtórzono eksperyment w celu dokładniejszego przeanalizowania wyniku. Proces uczenia w analizowanym modelu jest chaotyczny i niestabilny, co można zaobserwować na Rys. 2.9. Przyczyną może być użycie dużej ilości warstw gęsto połączonych. W rezultacie model jest przetrenowany, czyli liczba epok trening, jak i ilość parametrów modelu jest zbyt duża. Najczęściej powodue to wysokie rezultaty podczas treningu, zaś niskie w stosunku do nowych danych - testowych (2.8).



Rys. 2.9. Zależność skuteczności i błędu od epoki trenowania modelu.

.

Tabela 2.8. Skuteczność modelu w powtórzonym eksperymencie.

typ zbioru	**treningowy**	**walidacyjny**	**testowy**
skuteczność [%]	86	92	77

Macierz pomyłek dla zbioru walidacyjnego (2.9) wskazuje na równomierną klasyfikację każdej z klas. Jakość klasyfikacji jest porównywalna z modelem z rozdziału 2.3.1 (2.10).

Tabela 2.9. Macierz pomyłek zbioru testowego.

		predykcja			
klasa		E	L	M	N
	E	219	168	103	133
	L	213	132	133	142
	M	203	168	115	134
	N	217	153	127	127

Tabela 2.10. Parametry mierzące jakość klasyfikacji na zbiorze testowym.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.26	0.35	0.30	623
klasa limfocyt (L)	0.21	0.21	0.21	620
klasa monocyt (M)	0.24	0.19	0.21	620
klasa neutrofil (N)	0.24	0.20	0.22	620

2.3.3. Wytrenowana architektura sieci InceptionV3

Przytoczony algorytm w całości bazuje na publikacji [17]. Poniżej przedstawiono najważniejsze fragmenty oraz autorski eksperment na podstawie cytowanego modelu.

– Przygotowanie danych:

Zbiór uczący i testowy z oryginalnej bazy zostały użyty w całości i bez modyfikacji. Jako zbiór testowy wykorzystano raz jeszcze zbiór walidacyjny. Taki sposób testowania nie daje faktycznej informacji o zdolności klasyfikacyjnej sieci, dlatego eksperyment musiał zostać powtórzony. Zastosowano taki sam podział bazy jak w rozdziale 2.3.1.

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

– Struktura sieci:

```
_________________________________________________________________
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 sieci i treningu:

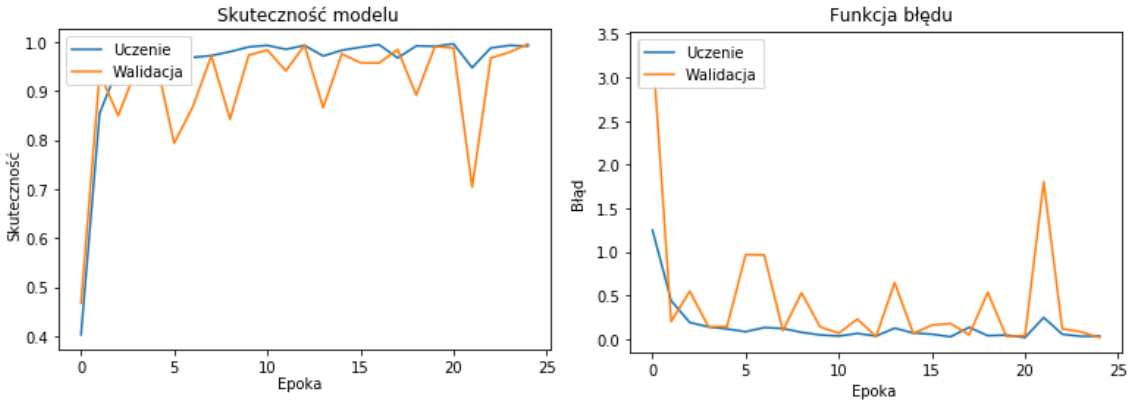
Model był trenowany przez 30 epok. W warstwach gęsto połączonych użyto aktywacji ReLu, a wartości dropoutu ustalono na od 0,7 do 0,3. Użyty optymalizator to ADAM z krokiem uczenia 5×10^{-5}.

– Wyniki klasyfikacji przy użyciu podanej architektury:

Uzyskane wyniki dotyczące skuteczności na zbiorze treningowym, walidacyjnym i także testowym są najwyższe w porównaniu z poprzednimi architekturami (2.11). Wyniki skuteczności dla zbioru walidacyjnego są mniej oscylują niż w modelu z regularyzacją (2.10). Skuteczność klasyfikacji jest porównywalna w każdej z klas, co przedstawiają tabele 2.12 i 2.13.

Tabela 2.11. Skuteczność modelu w powtórzonym eksperymencie.

typ zbioru	**treningowy**	**walidacyjny**	**testowy**
skuteczność [%]	99	99	86



Rys. 2.10. Zależność skuteczności i błędu od epoki trenowania modelu.

.

Tabela 2.12. Macierz pomyłek zbioru testowego.

		predykcja			
		E	L	M	N
klasa	E	179	154	115	175
	L	181	164	116	169
	M	208	136	115	161
	N	177	166	119	162

Tabela 2.13. Parametry mierzące jakość klasyfikacji na zbiorze testowym.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.24	0.29	0.26	623
klasa limfocyt (L)	0.26	0.26	0.26	620
klasa monocyt (M)	0.25	0.19	0.21	620
klasa neutrofil (N)	0.25	0.26	0.25	624

3. System do klasyfikacji elementów morometrycznych krwi

3.1. Przygotowanie bazy danych

Baza danych zawiera 12500 zdjęć krwinek białych w formacie JPEG wykonanych pod mikroskopem. Pogrupowane są w cztery foldery według przynależności do klas, każdy po około 3000 ramek. Uwzględnione typy krwnek to neutrofil, eozynofil, limfocyt, monocyt. Zbiór został stworzony z 410 obrazów przez operację powiększania zbioru (ang. *data augmentation*) [27].
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 standadowym 1 (3.1), [22].

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

gdzie,

- x – oryginalna wartość piksela,
- μ – średnia z wartości pikseli w ramce,
- σ – odchylenie standardowe wartości pikseli w ramce,
- z – wartość piksela po ustandaryzowaniu.

3.1.1. Zastosowanie metody augmentacji danych

W przypadku małych zbiorów danych, rozumianych jako zbiór liczący kilka tysięcy elementów często stosowaną praktyką jest poszerzanie zboru danych. 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. overfiting*). 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 [44].
Jednym ze sposobów transformacji jest elastczna deformacja w przestrzenii 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 przestrzenii cech (*ang. feature-space augmentation*) [44]. Definiuje ona znormalizowany obszar losowego przemieszczenia $u(x, y)$, który dla każdego piksela w obrazie (x, y) definiuje wektor przemieszczenia R_w (3.2), [44]:

$$R_w = R_0 + \alpha u \tag{3.2}$$

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 $\alpha \geq 8$ pikseli skutkuje w pewnej części przypadków utratą informacje o przynależności do danej klasy. Jest to definiowane jako brak zdolności do rozpoznania i zaklasfikowania danej ramki przez człowieka [44].
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 nauczenia 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.
Baza danych użyta w pracy oryginalnie zawiera obrazy, będące wynikiem operacji powiększania zbioru. Oryginalna baza danych zawierała 410 obrazów krwinek białych. W wyniku powięszania zbiou powstała baza wkorzystywana w niniejszym projekcie. Z tego powodu nie jest wskazane dodatkowe przepowadzanie powiększania. Sumarcznie, użyta baza zawiera 12444 obrazy krwinek białych z czterech rozpatrywanych kategorii.

3.1.2. Podział na podzbiory

W celu wykorzystania danych do stworzenia sieci należy je podzielić na podzbiory: uczący - służący do wytrenowania sieci, walidacyjny - służący do nastrojenia sieci oraz testowy - użyty tylko raz na samym końcu gdy klasyfikator jest gotowy w celu sprawdzenia skuteczności sieci na nowych danych.
Sposób podziału bazy danych na zbiory ma znaczący wpływ na jakość klasyfikatora, jaki można uzyskać. W przpadku przeznaczenia dużej ilości danych do treningu łatwiejsze będzie uzyskanie dobrej skuteczności na danych teningowych. Jednak nie da to pewności czy sieć zadziała poprawnie dla nowych danych, gdyż zbiory walidacyjny i testowy będą ograniczone. W sytuacji odwrotnej, przy użyciu dużej ilości danych do strojenia i testów sieć może nie nauczyć się odpowiedniej ilości różnorodnych wzorców, gdyż zbiór treningowy został ograniczony.

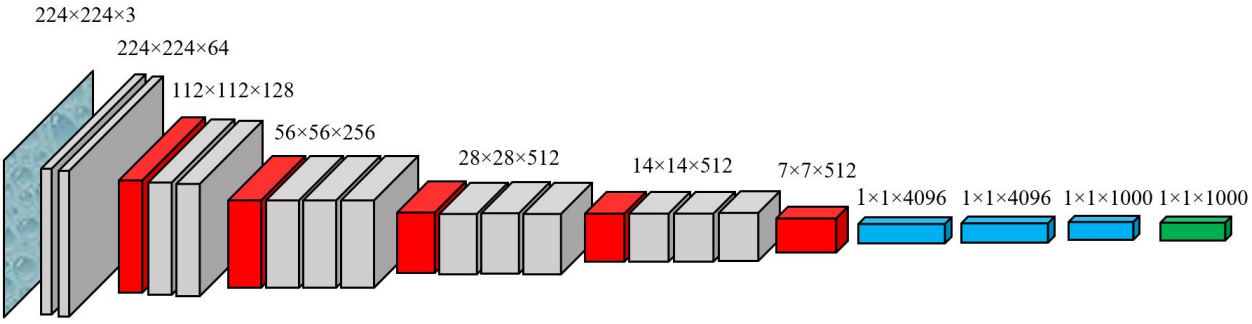
Początkowym podział bazy na zbiory został zrobion zgodnie z podejściem zastosowanym w rozdziale 2.3.1. Dzięki temu uzyskane wyniki będą mogły być porównywane.

3.2. Zaproponowana struktura sieci neuronowej

Struktura sieci neuronowej bazuje na wybranych cechach zaczerpiętych z architektur VGGNet, U-Net oraz ResNet. W kolejnych podrozdziałach zostały opisane poszczególne modele, których charakterystyczne wlasności wykorzystano w końcowym rozwiązaniu.

3.2.1. Cechy architektury zaczerpnięte z modelu VGGNet.

VGGNet jest to architektura sieci kowolucyjnej opublikowanej przez Visual Geometry Group na Universitecie Oksfordzkim w 2014 roku [35]. Wytrenowana na 1.3 milionach obrazów treningowych dzielących się na 1000 klas uzyskała 92,7% skuteczności na zbiorze testowym [8] [11]. Charakteryzuje się jednolitą budową i składa się jedynie z warstw konwolucyjnych przekładanych co dwie lub trzy warstwy redukcją maksymalizującą oraz trzech warstw gęsto połączonych, umieszczonych na końcu modelu (Rys. 3.1). Najczęściej w literaturze występuje w wersjach z 16 (VGG16) i 19 (VGG19) warstwami konwolucyjnymi.



Rys. 3.1. Wizualizacja architektury sieci VGG16 [29].

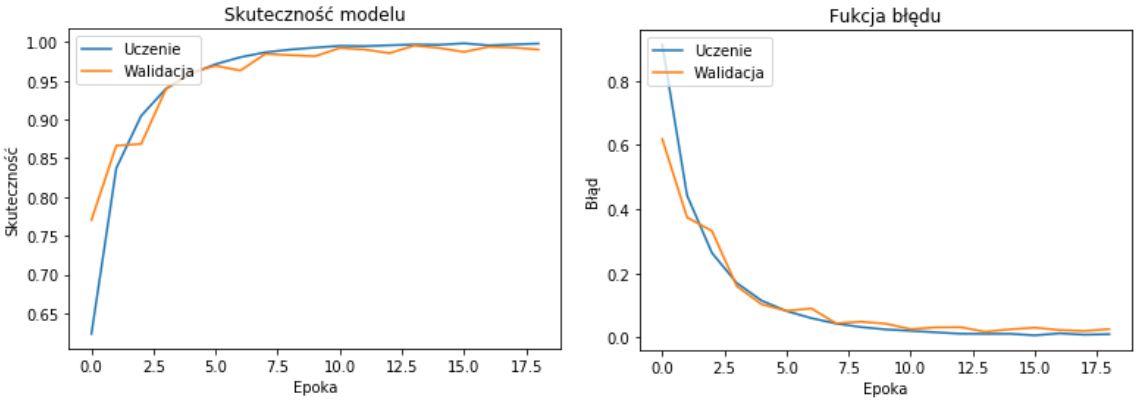
Biblioteka Keras udostępnia przetrenowany model sieci VGG16. Z uwagi na bardzo dobre rezultaty uzyskane przy użyciu przenoszenia uczenia w rozdziale 2.3.3, zdecydowano się na sprawdzenie jakie wyniki uzyska mniej skompikowany w budowie model. W tym celu rozmrożono trzy ostatnie warstwy konwolucyjne i ponownie wyuczono sieć na zdjęciach krwinek. Uzyskano dość dobre rezultaty na wszystkich zbiorach (3.1), a skuteczność klasyfikacji każdej klasy jest na podobnym poziomie (3.2). Wnioskując po przebiegu treningu, sieć jest poprawnie wyuczona (Rys. 3.2).

Tabela 3.1. Skuteczność modelu w powtórzonym eksperymencie.

typ zbioru	treningowy	walidacyjny	testowy
skuteczność [%]	99	99	71

Tabela 3.2. Parametry mierzące jakość klasyfikacji na zbiorze testowym.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.24	0.32	0.27	623
klasa limfocyt (L)	0.22	0.22	0.22	620
klasa monocyt (M)	0.20	0.07	0.10	620
klasa neutrofil (N)	0.23	0.30	0.26	624

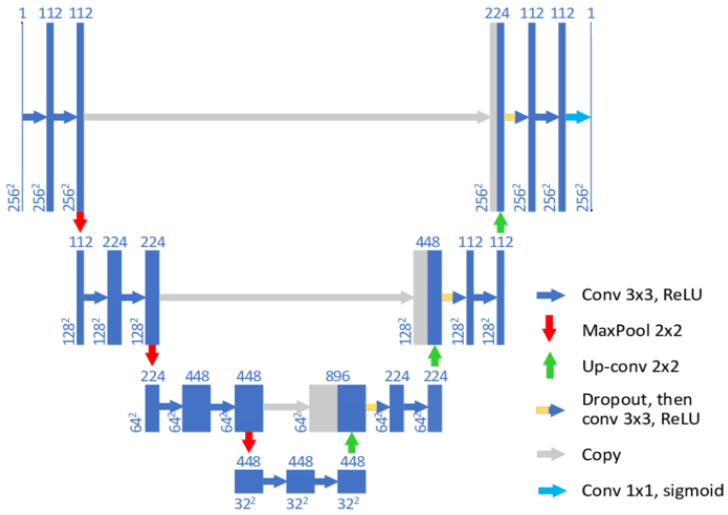


Rys. 3.2. Zależność skuteczności i błędu od epoki trenowania modelu VGG16.

Sieć VGG została zbudowana w celu zwiększenia jakości klasyfikacji nie przez ilość warstw, ale przez wzrost liczby filtrów w poszczególnych warstwach. Wielkość filtrów jest stała i wynosi 3x3, są stosowane z krokiem jednego piksela. Użycie tak dużej ilości filtrów do stosunkowo małej bazy o czterech klasach w sytuacji, gdy wszystkie wagi modelu byłyby edytowalne, mógłby spowodować przetrenowanie zbioru. Z tego powodu w budowanej sieci zdecydowano się na zmniejszenie ilości filtrów do liczby dwucyfrowej w pojedynczej warstwie. Z sieci VGGNet zaczerpnięto logikę ułożenia typów warstw, gdyż po co dwóch lub trzech warstwach konwolucyjnych pojawia się warstwa redukcji maksymalizującej. Zastosowano też taką samą liczbę warstw redukcyjnych.

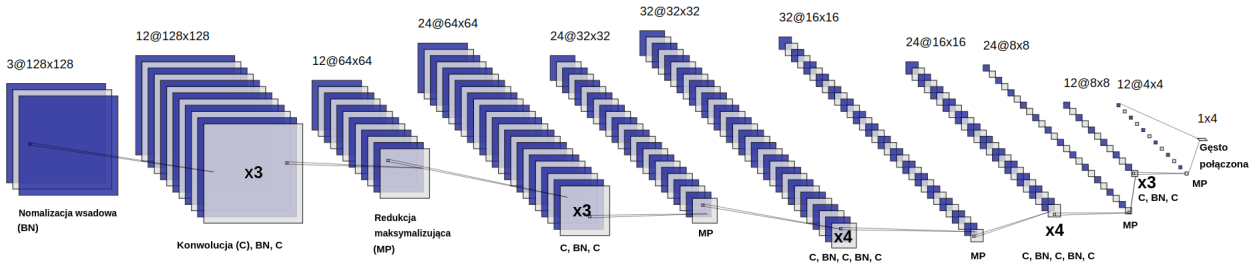
3.2.2. Cechy architektury zaczerpnięte z modelu U-Net.

Architektura zaprezentowana w rozdziale 2.3.1 wykazuje cechy struktury U-Net ze względu na kolejność ułożenia filtrów poszczególnej wielkości. Bazując na wiedzy, że użycie wspomnianej architektury dało wysokie rezultaty klasyfikacji, zastosowano podobne podejście. U-Net jest to sieć konwolucyjna opracowana na Uniwersytecie we Fryburgu do segmentacji obrazów biomedycznych [32]. Typowa sieć U-Net nie ma w swojej strukturze warstw gęsto połączonych, a funkcją aktywacyjną jest zawsze ReLU. Opiera się na fakcie, że sieć neuronowa jest w stanie odfiltrować cechy ważne dla rozpoznania obiektów obrazu przez sukcesywne zmniejszanie wielkości filtrów, a potem ponowne zwiększanie do początkowej wielkości (Rys. 3.3). Zmniejszanie wymiaru następuje przez użycie warstw redukcyjnych, a zwiększanie przez konkatenację warstw. Jest to operacja podobna do działania filru dolnoprzepustowego,



Rys. 3.3. Wizualizacja struktury sieci U-Net [34].

powodującego rozmycie obrazu, przez co traci się informację o szczegółach. Potem następuje ponowne odtworzenie danych w pierwotnej rozdzielczości na podstawie wygładzonego obrazu. Z architektury tej zaczerpnięto symetrię wielkości filtrów względem środkowej warstwy sieci.



Rys. 3.4. Wizualizacja struktury zaproponowanej sieci VGG-UNet.

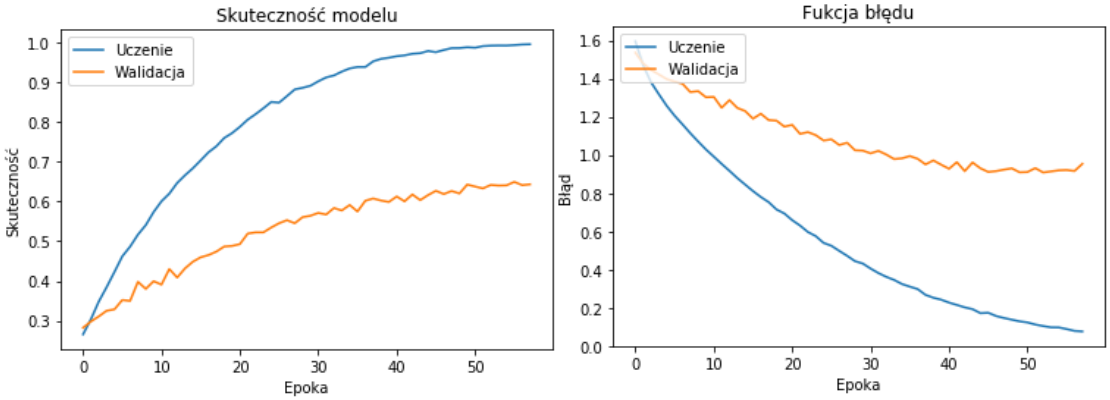
W żadnej ze wspomnianych architektur nie jest używana normalizacja wsadowa, zapewne dlatego, że zostały one opracowane zanim ten typ warstwy opisano i opublikowano w 2015 roku [15]. W zaproponowanym modelu przed każdorazowym wejściem do warstwy konwolucyjnej są poddawane normalizacji. Jest to technika poprawiająca stabilność i jakość sieci neuronowych. Wyniki skuteczności klasyfikacji architektury z Rys. 3.4 są niższe niż sieci uprzednio przetrenowanej z powodu znacznie mniejszej sumarycznej liczby danych przepływających przez sieć w trakcie uczenia. Jednak fakt, że ponad połowa próbek została poprawnie zaklasyfikowana (3.3) oraz nie ma klasy, która byłaby gorzej rozpoznawana niż inne (3.4) oznacza, że przy odpowiedniej edycji obecnego modelu można uzyskać wyższą skuteczność. Z wykresu treningu wynika, że opowiednia zmiana parametrów mogłaby pozwolić na wyższe osiągnięcia na zbiorze walidacyjnym, gdyż w obecnej postaci wyniki na tym zbiorze są mocno zaniżone (Rys. 3.5).

Tabela 3.3. Skuteczność modelu o strukturze VGG-UNet.

typ zbioru	treningowy	walidacyjny	testowy
skuteczność [%]	99	64	53

Tabela 3.4. Parametry mierzące jakość klasyfikacji na zbiorze testowym.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.26	0.29	0.27	623
klasa limfocyt (L)	0.24	0.23	0.23	620
klasa monocyt (M)	0.25	0.21	0.23	620
klasa neutrofil (N)	0.26	0.28	0.27	624

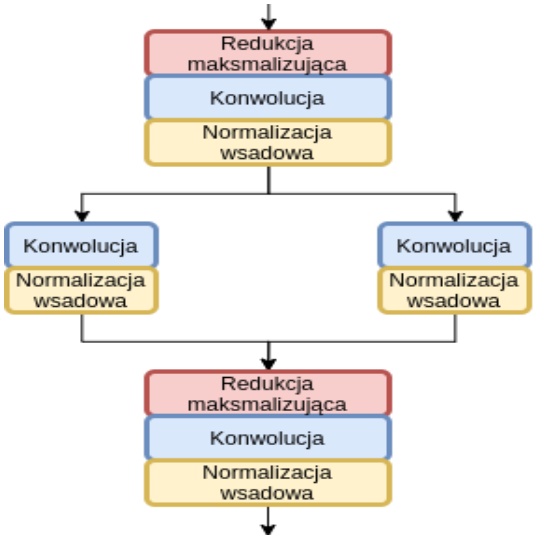


Rys. 3.5. Zależność skuteczności i błędu od epoki trenowania modelu o strukturze VGG-UNet.

3.2.3. Cechy architektury zaczerpnięte z modelu ResNet.

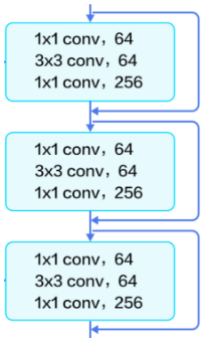
Warstwy w architekturze z rozdziału 2.3.1 nie były połączone typowo sekwencyjnie. W co drugiej warstwie konwolucyjnej stosowano rozdzielenie sieci na dwie podsieci o takiej samej liczbie filtrów, ale różnej wielkości jądra. Wyjścia z tych warstw konkatenowano i wsadzano na kolejną warstwę konwolucyjną. Fragment struktury tej sieci obrazuje Rys. 3.6.

Archiekturą, która także stosuje niesekwencyjne połączenie warstw jest rezydualna sieć (ang. *residual neural network*, ResNet). Jej konstrukcja jest podobna do połączeń neuronów piramidowych w korze mózgowej. Wejścia w niej są połączone ze skonktenowanym wjściami warstwy bezpośrednio poprzedzającej oraz warstwy położonej kilka poziomów wcześniej 3.7. W ten sposób omijane są połączenia z niektórymi warstwami. Najczęściej dotyczy to co drugiej lub co trzeciej warstwy. Pomijanie upraszcza sieć, co przyśpiesza proces uczenia. Inną korzyścią jest zmniejszenie problemu zanikajacych gradientów. Problem ten powstaje na etapie uczenia sieci korzystając z metod gradientowych. Poprawa wag w każdej iteracji jest proporcjnalna do pochodnej cząstkowej błędu funkcji względem obecnej wagi. Czasami może to prowadzić do sytuacji, gdzie gradient będzie tak mały, że waga nie zostanie zmieniona i postęp



Rys. 3.6. Wizualizacja fragmentu struktury sieci z rozdziału 2.3.1.

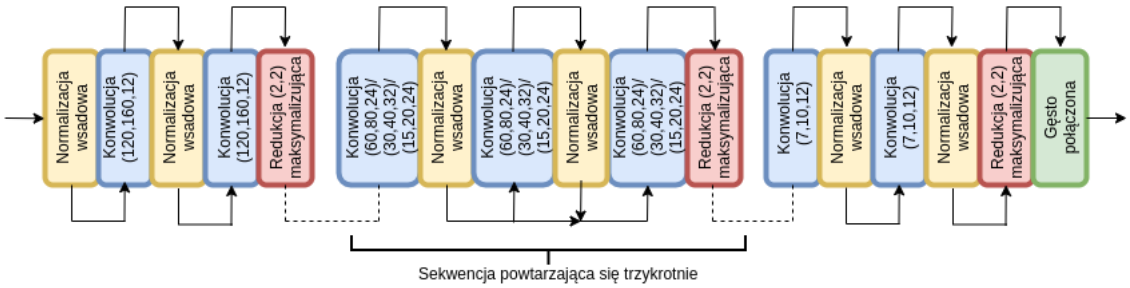
w nauce zostanie zatrzymany. Dzięki użyciu wag nie tylko z aktualnej warstwy, ale też z poprzedzającej [41]. W przypadku modelu VGG-UNet nie używano funkcji aktywacyjnej mogącej spowodować zanikanie gradientu, gdyż aktywacją była funkcja ReLU. Mimo to po dołączeniu do modelu z Rys. 3.4 połączenia typu ResNet, jak na Rys. 3.8, wyniki na zbiorze testowym znacząco się poprawiły (3.5, 3.6). Z wykresów zależności skuteczności i błędu od epoki trenowania (3.9) wynika, że klasyfikator należy dostroić, żeby zniwelować wahania wartości.



Rys. 3.7. Wizualizacja fragmentu struktury sieci ResNet50 [6].

Tabela 3.5. Skuteczność modelu z cechami VGG-UNet-ResNet.

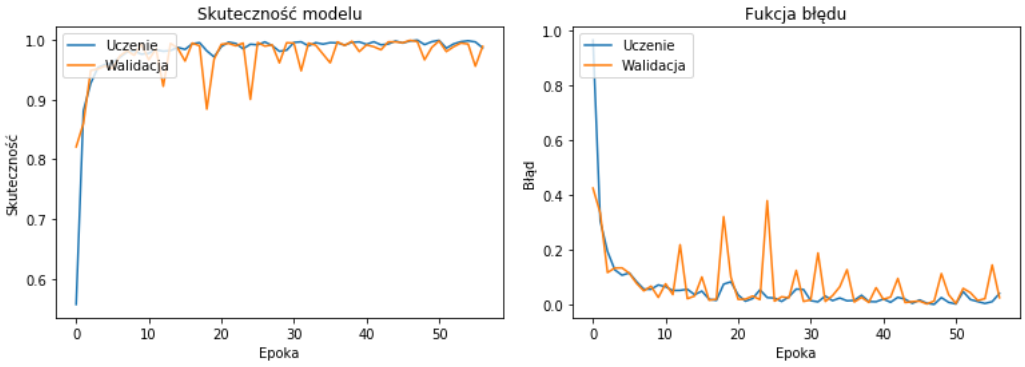
typ zbioru	treningowy	walidacyjny	testowy
skuteczność [%]	99	99	86



Rys. 3.8. Wizualizacja architektury zaproponowanej sieci VGG-UNet-ResNet.

Tabela 3.6. Parametry mierzące jakość klasyfikacji na zbiorze testowym modelu VGG-UNet-ResNet.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.23	0.22	0.23	623
klasa limfocyt (L)	0.22	0.22	0.22	620
klasa monocyt (M)	0.24	0.18	0.21	620
klasa neutrofil (N)	0.24	0.31	0.27	624



Rys. 3.9. Zależność skuteczności i błędu od epoki trenowania modelu o strukturze VGG-UNet-ResNet.

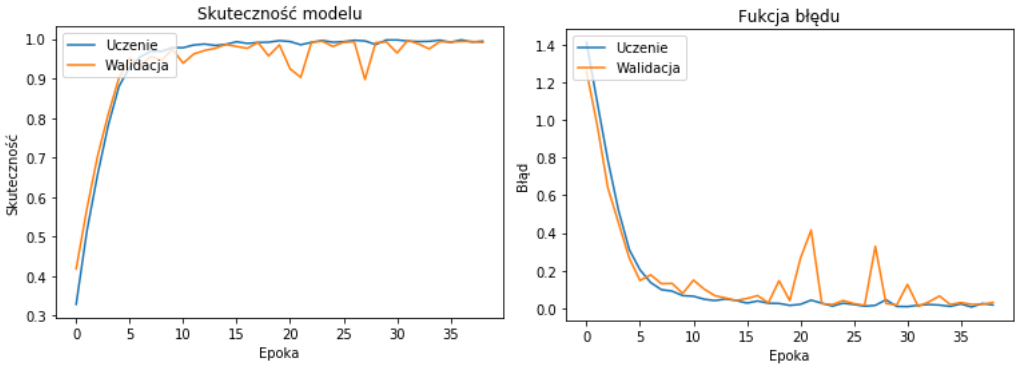
3.3. Dobór parametrów

Podczas uczenia zaproponowanego modelu widoczne są duże wahania skuteczności i błędu na zbiorze walidacyjnym (3.9). Jest to dowód na nadmierne dopasowanie, gdyż wykres opisuje w dużej części szum. Może to być spowodowane słabym dobraniem modelu do zadania i oznacza konieczność dostrojenia parametrów. Jeśli mimo prób doboru parametrów krzywa walidayjna nadal będzie zaszumiona, możliwe że baza danych musi zostać podzielona raz jeszcze, gdyż zbiór walidacyjny jest niereprezentacyjny [4].

Strojenie parametrów wykonuje się w celu uzyskania jak najszybszej zbieżności modelu do wyniku, więc im mniej epok zajmuje trening tym lepsza jest opracowna sieć. Jednocześnie należy przy tym zachować wysoką precyzję rozwiązania, która jest mierzona przez skuteczność klasyfikacji. Innym ważnym aspektem, jaki należy wziąć pod uwagę jest złożoność obliczeniowa, jeśli klasyfikator posiada wiele warstw i filtrów, to jego użycie będzie zajmowało dużo mocy obliczeniowej. Należy dążyć do jak prostej struktury, która będzie dawała dobre efekty. W tym celu zostanie przetestowanych kilka wariantów wybranych parametrów modelu, a wyniki porównane z uzyskanymi w rozdziale 3.2.3. Końcowy model zostanie zaprezentowany w rozdziale 4, gdzie zamieszczona zostanie analiza wyników klasyfikacji.

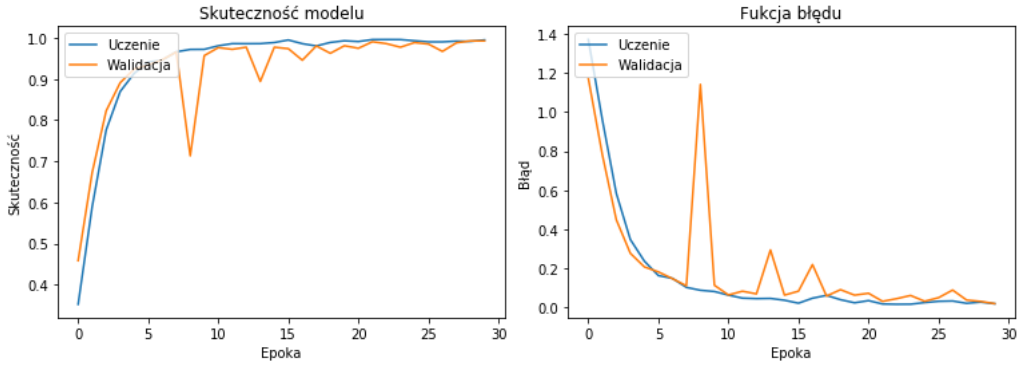
3.3.1. Dobór optymalizatora i szybkości uczenia

Jednym z powodów wahań skuteczności modelu na przestrzenii epok może być niepoprawnie dobrany krok w minimalizacji funkcji błędu. W pierwszym podejściu używany był optymalizator ADAM (2.5) z krokiem równym 1×10^{-3}. Jest to metoda optymalizacji bazująca na RMSprop (2.6), z użyciem dodatkowo stochastycznego spadku gradientowego z bezwładnością. Istnieją różne warjancje optymalizator ADAM, w tym NADAM, gdzie bezwładność liczona jest algorytmem Nesterova. Wykonano serię eksperymentów używając optymalizatorów: RMSprop, Adam i Nadam oraz różnej wielkości kroku. Dla wszystkich typów badanych optymalizatorów najlepsza wielkość kroku to 2×10^{-4}. Poniżej przedstawiono porównanie optymalizatorów z tym krokiem.

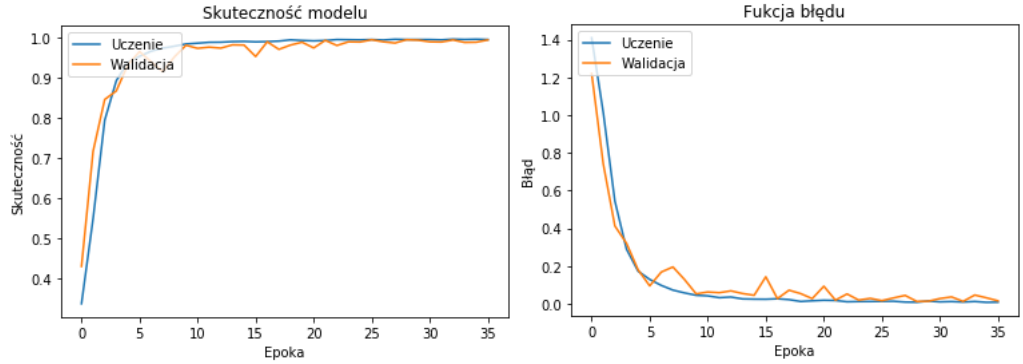


Rys. 3.10. Zależność skuteczności i błędu od epoki trenowania dla optymalizatora NADAM z krokiem 2×10^{-4}.

W przypadku algorytmu NADAM (3.10) wahania są rzadsze niż przy użyciu oryginalnego optymalizatora (3.9), jednak wykres nie wykazuje tendencji do stabilizacji. Uczenie z użyciem optymalizatora ADAM wykazuje pozytywną cechę zanikania wahań (3.11), więc wybór tego wariantu miałby sens. Jednak najmniejsze wahania i tendencję do ich wygłuszania można zaobserwować na wykresie optymalizatora RMSprop (3.12). Z tego powodu kolejne eksperymenty będą przeprowadzane z użyciem RMSprop i kroku 2×10^{-4}.



Rys. 3.11. Zależność skuteczności i błędu od epoki trenowania dla optymalizatora ADAM z krokiem 2×10^{-4}.



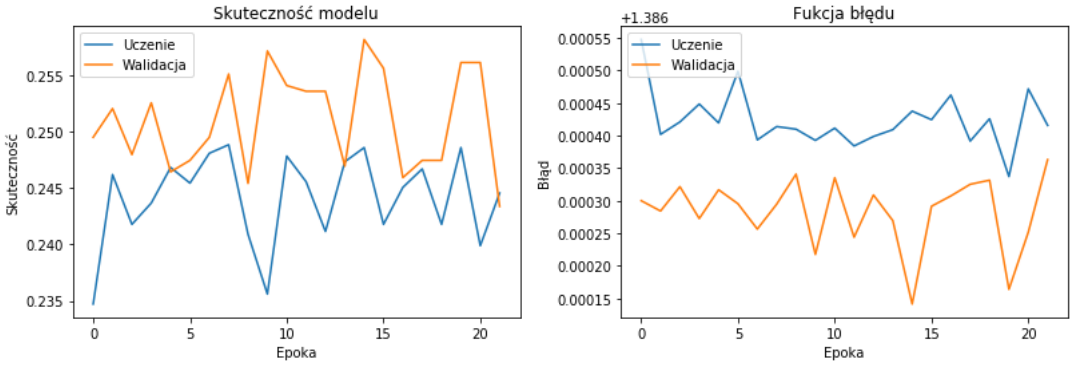
Rys. 3.12. Zależność skuteczności i błędu od epoki trenowania dla optymalizatora RMSprop z krokiem 2×10^{-4}.

3.3.2. Ograniczenie przeuczenia modelu

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.

Jednym ze sposobów na uniknięcie nadmiernego dopasowania jest używanie warstw regularyzacyjnych, takich jak normalizacja wsadowa. Oryginalnie w architekturach VGG16, U-Net i ResNet tego typu warstwa nie jest używana. Mimo to, w zaproponowanym modelu zdecydowano się na jej użycie, gdyż pozwala to na o wiele lepsze wyniki w porównaniu do sieci bez tej warstwy (3.13).

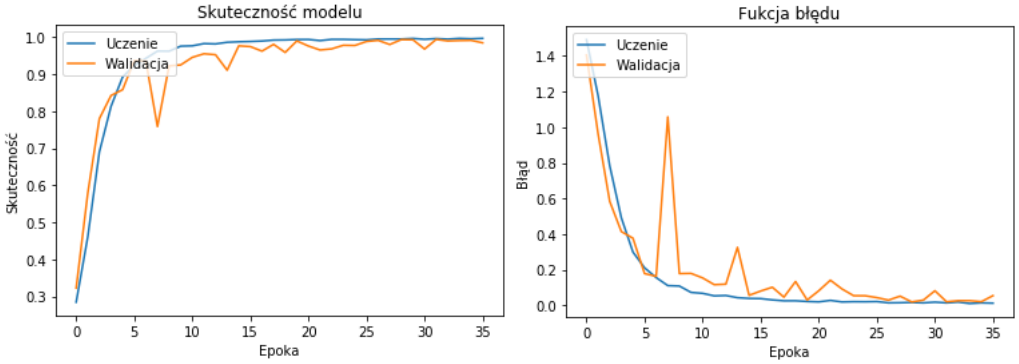
Innym sposobem, poza użyciem konkretnych warstw, jest przeprowadzenie treningu w taki sposób, aby nie dopuścić do przeuczenia modelu. W tym celu w każdej epoce nauki należy monitorować wartość błędu na zbiorze walidacyjnym i porównywać z błędem w poprzedniej epoce. Jeśli nie obserwuje się poprawy przez daną ilość epok, w omawianym modelu ta wartość wynosi 7, to trening zostaje przerwany. Ma to na celu zachowanie ogólności działania klasyfikatora i uniknięcie takiego ustawienia wag, aby model dopasował się jedynie do konkretnych danych w zbiorze uczącym.



Rys. 3.13. Zależność skuteczności i błędu od epoki trenowania modelu o strukturze VGG-UNet-ResNet bez warstw normalizacji wsadowej.

3.3.3. Wpływ przygotowania danych na jakość klasyfikacji

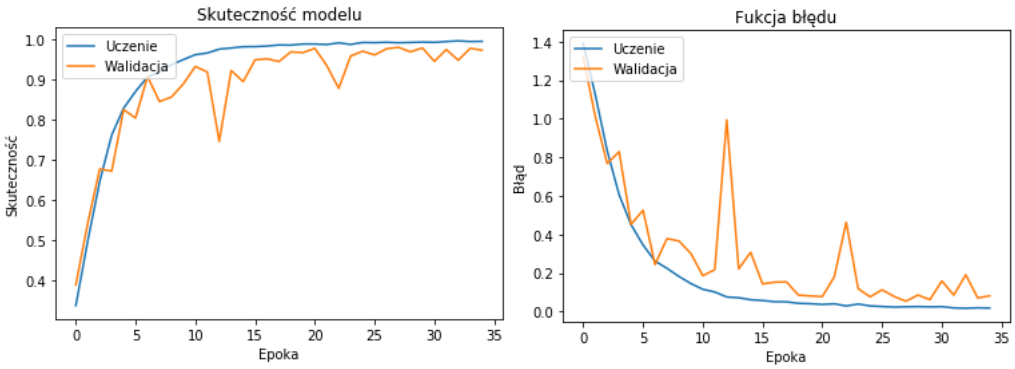
Oryginalnie dane przed przejściem przez sieć neuronową są normalizowane zgodnie z rozdziałem 3.1. Wykonano eksperyment, w którym pominięto krok normalizacji danych. Wyniki i jakość modelu nie uległy znacznemu pogorszeniu (3.14), z powodu użycia warstwy normalizacji wsadowej na początku sieci neuronowej. Kształt wejścia do modelu zachowano bez zmian: 120x160x3. Skuteczność na zbiorze walidacyjnym i treningowym wynosiła nadal 99%.



Rys. 3.14. Zależność skuteczności i błędu od epoki trenowania modelu przetrenowanego na ramkach RGB nie znormalizowanych.

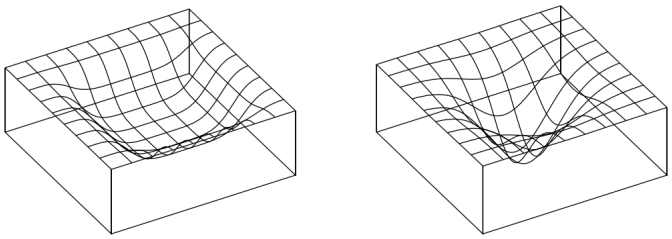
W kolejnym eksperymencie także nie wykonano normalizacji, a dodatkowo zmniejszono liczbę kanałów do jednego, czyli spłycono ilość informacji do jasności pikseli. Nie tylko pogorszyło to jakość procesu uczenia, ale też obniżyło skuteczność na zbiorze walidacyjnym o dwa punkty procentowe 3.15).

Dane wprowadzane do sieci neuronowej są dzielone na porcje (ang. *batches*), co odciąża maszynę liczącą i pozwala na przyspieszenie procesu uczenia. Każdy podzbiór: uczący, walidacyjny i testowy dzielony jest na porcje o tej samej liczebności. Początkowym podejściem był podział na porcje o zawartości 32 elementów. Typowo sieci uczą się szybciej na porcjach o małej liczebności oraz zbiegają do płaskich minimów (3.16) . Z kolei przy stosowaniu dużej liczbeności obserwuje się degradację jakości modelu, w



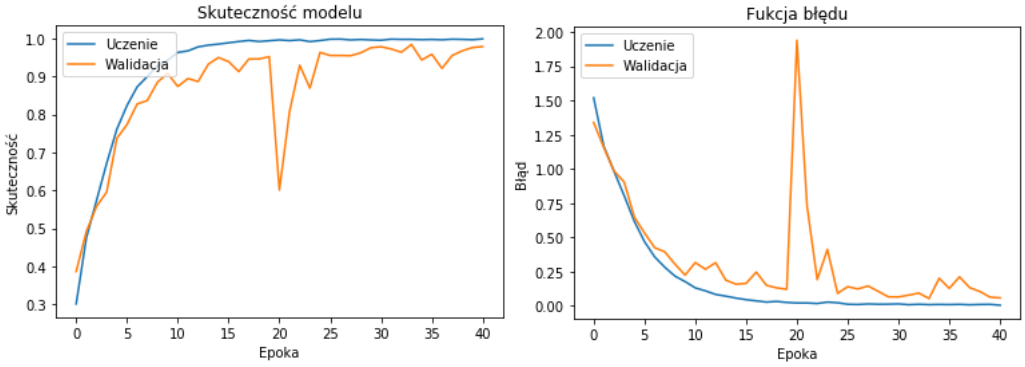
Rys. 3.15. Zależność skuteczności i błędu od epoki trenowania modelu przetrenowanego na ramkach Grayscale, nie znormalizowanych.

szczególności dotyczącej zdolności do generalizacji. Jest to spowodowane tendencją do zbieżności tego typu algorytmów do ostrych minimów, która nie sprzyja generalizacji rozwiązania [19].

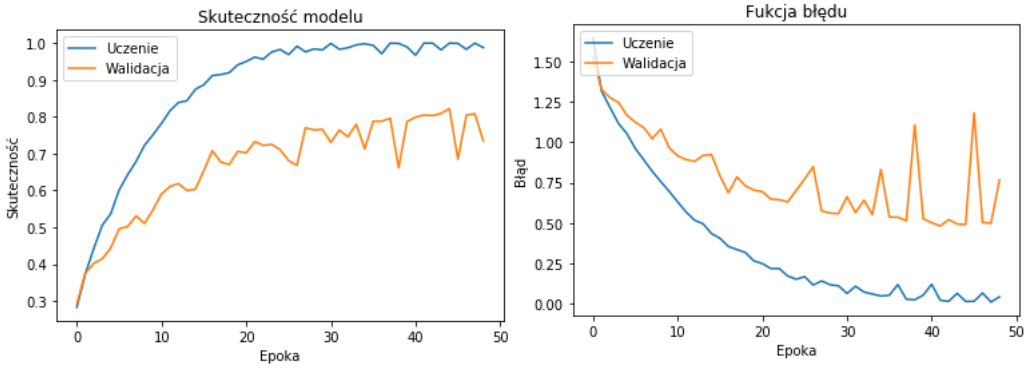


Rys. 3.16. Wizualizacja różnicy między minimum płaskim (z lewej), a ostrym (z prawej) [14].

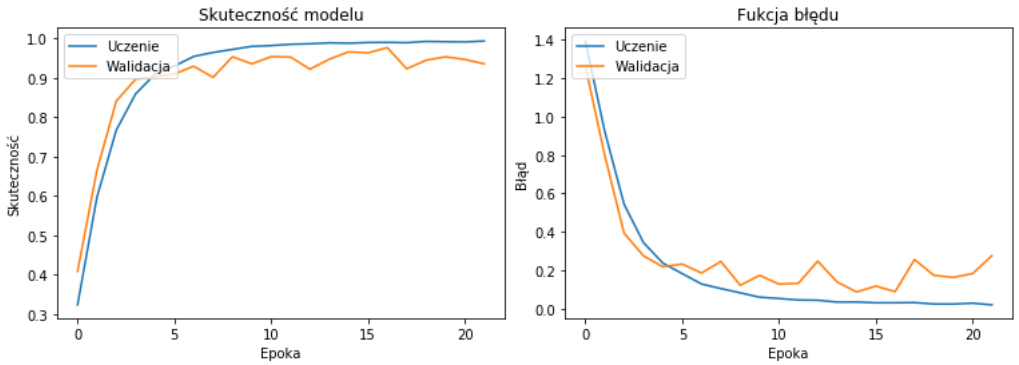
W niektórych przypadkach powiększenie batchy ma pozytywy wpływ. Na przykład dla sieci AlexNet uczonej na obrazach ImageNet zwiększenie liczebności z 512 do 4096 poskutkowało trzykrotnym przyśpieszeniem procesu uczenia [45]. W przypadku omawianego modelu podział na porcje o większej liczebności niż 32 skutkuje niedouczeniem sieci (ang. *underfitting*), zarówno dla 128 jak i 512 ramek (3.17, 3.18). Dla liczebności miejszej niż 32 też występuje taki sam problem (3.19). Na tej podstawie można wnioskować, że wybór 32 ramek w porcji jest najlepszym podziałem.



Rys. 3.17. Zależność skuteczności i błędu od epoki modelu trenowanego z użyciem bazy podzielonej na batche o liczebności 128 ramkek.



Rys. 3.18. Zależność skuteczności i błędu od epoki modelu trenowanego z użyciem bazy podzielonej na batche o liczebności 512 ramkek.

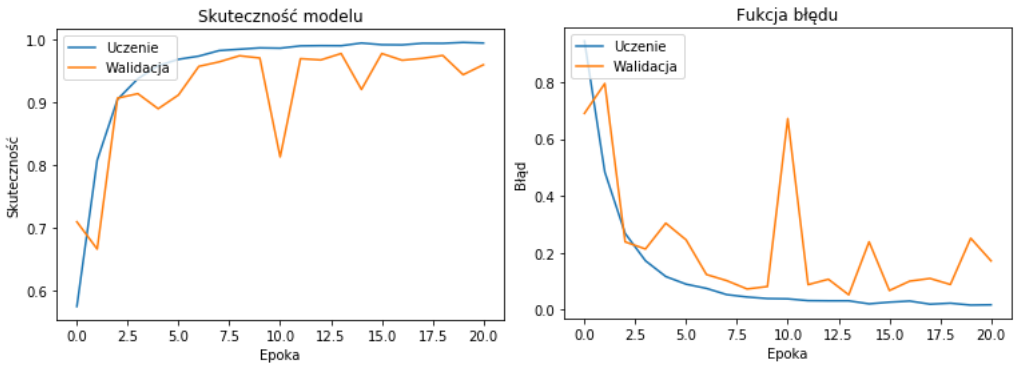


Rys. 3.19. Zależność skuteczności i błędu od epoki modelu trenowanego z użyciem bazy podzielonej na batche o liczebności 16 ramkek.

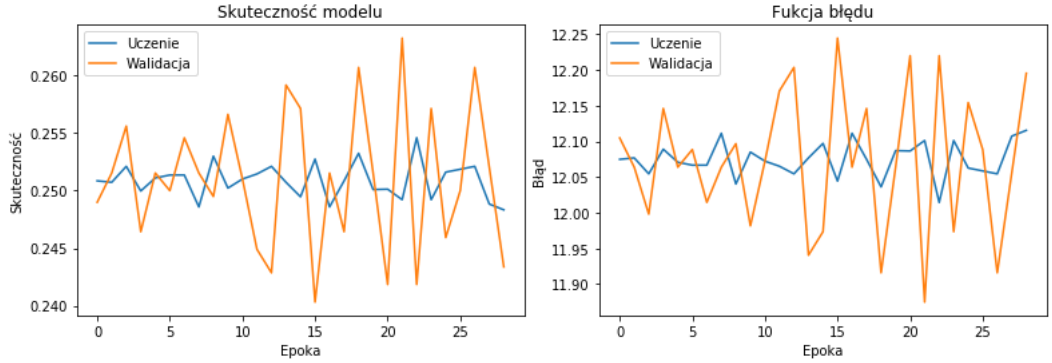
3.3.4. Wybór funkcji redukcyjnej

Dotychczas stosowanym typem warstwy redukcyjnej była maksymalizacja o oknie wielkości 2x2 stosowana z krokiem dwóch pikseli. Oznacza to, że wybiera ona najjaśniejsze piksele z obrazu. Jest więc

przydatna do klasyfikacji obiektów jasnych na ciemniejszym od nich tle. W eksperymencie sprawdzono jaki wpływ na przebieg uczenia będzie miało zamienienie wszystkich warstw reducyjnych na minimalizację lub średnią arytmetyczną, także o wielkości 2x2.



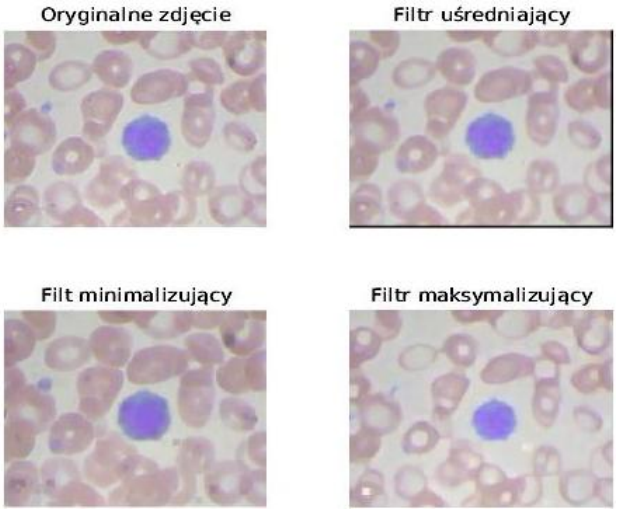
Rys. 3.20. Zależność skuteczności i błędu od epoki modelu trenowanego z użyciem warstw redukcji uśredniającej.



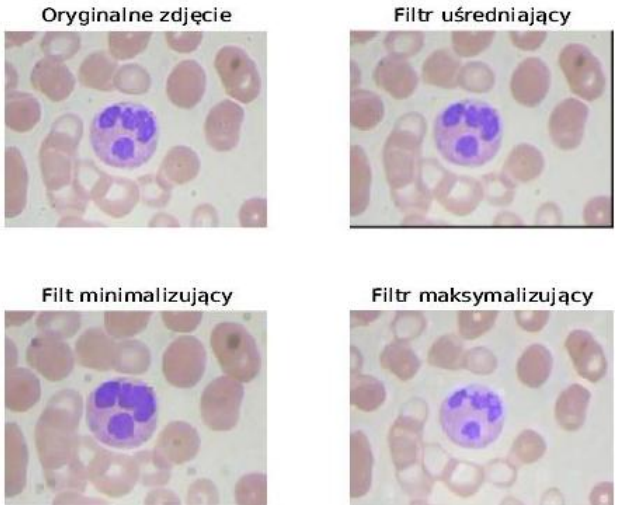
Rys. 3.21. Zależność skuteczności i błędu od epoki modelu trenowanego z użyciem warstw redukcji uśredniającej.

Użycie średniej arytmetycznej jedynie nieznacznie zabrzyło proces, powodując wahania skuteczności wlidacyjnej i niedotrenowanie (3.20). Skuteczność na zbiorze walidacyjnym spadła, ale nie drastycznie. Powodem takiego zachowania jest fakt, że redukcja uśredniająca wygładza obraz i powoduje, że detale mogą zostać pominięte. Ramki są nadal klasyfikowane poprawnie, ale w mniejszej ilości niż miało to miejsce w przypadku redukcji aksymalizujące. Z kolei warstwy minimalizacyjne całkowicie zablokowały proces uczenia, który zatrzymał się w minimum lokalnym dającym około 25% skuteczności. Filtr typu min podkreśla ciemne piksele. Sprawia to, że krwinki nie mające jądra, widoczne w tle zdjęcia stają się bardziej wyraźne (3.22). Zaciemnia to obraz i sprawia, że rozpoznanie typu krwinki staje się trudniejsze. Ponadto jądro, które jest ciemne na tle reszty krwinki zostaje powiększone we wszystkich typach krwinek (3.23), przez to algorytm myli jądra dwupłatowe i posegmentowane z jądrami kulistymi i klasyfikuje wszystkie typy krwinek jako neutrofil (3.7). Ramki, które zostały poddane działaniu filtru

maksymalizującego zachowują bardziej czytelną informację dotyczącą typu jądra, co szczególnie jest widoczne w jądrach niekulistych (3.23).



Rys. 3.22. Zdjęcie limfocytu z bazy po pięciokrotnym nałożeniu filtrów 2x2.



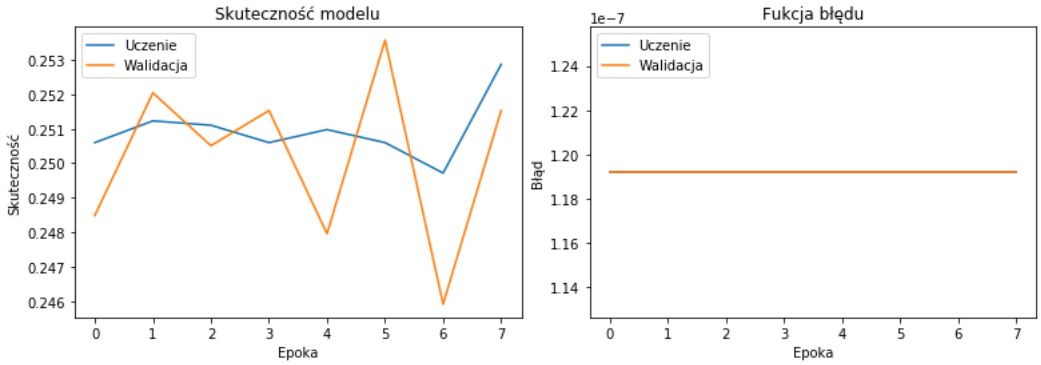
Rys. 3.23. Zdjęcie neutrofilu z bazy po pięciokrotnym nałożeniu filtrów 2x2.

Tabela 3.7. Parametry mierzące jakość klasyfikacji na zbiorze walidacyjnym modelu z warstwą redukcyjną minimalizującą.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.00	0.00	0.00	499
klasa limfocyt (L)	0.00	0.00	0.00	497
klasa monocyt (M)	0.00	0.00	0.00	496
klasa neutrofil (N)	0.25	1.00	0.40	500

3.3.5. Wybór funkcji aktywacyjnej warstw konwolucyjnych

W modelu użytą funkcją aktywacyjną dla wszystkich warstw konwolucyjnych jest ReLU (2.11). W celu dobrania optymalnej aktwacji sprawdzono działanie sieci także dla aktywacji liniowej (2.8) z współczynnikiem c = 1, eksponencjalnej (2.9) i tangensa hiperbolicznego (2.10).



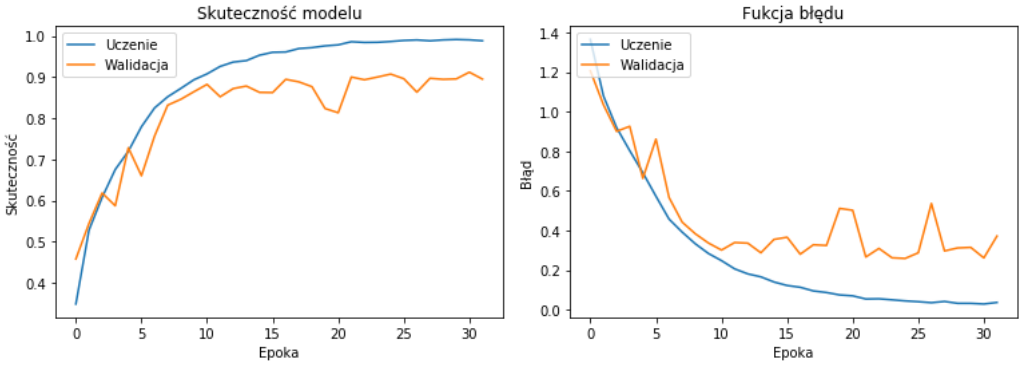
Rys. 3.24. Zależność skuteczności i błędu od epoki modelu trenowanego z użyciem aktywacji eksponencjalnej w warstwach konwolucyjnych.

Tabela 3.8. Parametry mierzące jakość klasyfikacji na zbiorze walidacyjnym modelu z warstwą aktywacyjną eksponencjalną.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.25	1.00	0.40	499
klasa limfocyt (L)	0.00	0.00	0.00	497
klasa monocyt (M)	0.00	0.00	0.00	496
klasa neutrofil (N)	0.00	0.00	0.00	500

Najmniej zadowalające wyniki dało użycie funkcji eksponencjalnej, gdyż algorytm utyka w minimum lokalnym już w pierwszej epoce i rozpoznawaje poprawnie jedynie klasę eozynofil. Zmiana wielkości kroku przy poszukiwaniu minimum nie przyniosła pozytywnych rezultatów. Zmiany wag w kolejnych epokach nie skutkowały zmianą wartości funkcji błędu (3.24). Parametry macierzy pomyłek zbioru wlidacyjnego wskazują na to, że aktywacja eksponencjalna prowadzi do przyporządkowania zawsze tej samej wartości wyjściowej niezależnie od wejścia (3.8).

Lepszą jakość klasyfikacji uzyskano przy użyciu jednej z najprostszych aktywacji - liniowej. W tym typie aktywacji wartość na wejściu jest przekazywana bezpośrednio na wyjście, gdyż współczynnik skalowania wynosi 1. Model rozpoznaje poprawnie pewną ilość danych z każdej klasy, jednak w porównaniu z aktywacją ReLU radzi sobie gorzej (3.9). W procesie uczenia można zaobserwować underfitting (3.25). Wybrany typ aktywacji nie odwzorowuje skomplikowanych typów funkcji i nie wprowadza elementów nieliniowych do modelu, co jest głównym celem użycia aktywacji. Z pewnością ten klasyfikator ma niższą złożoność obliczeniową, ale jest ograniczony w zakresie skomplikowania funkcji, których jest w

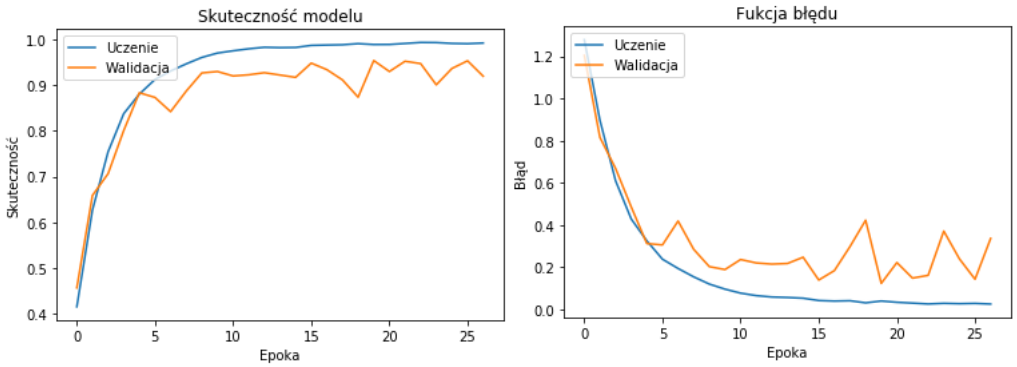


Rys. 3.25. Zależność skuteczności i błędu od epoki modelu trenowanego z użyciem aktywacji liniowej w warstwach konwolucyjnych.

Tabela 3.9. Parametry mierzące jakość klasyfikacji na zbiorze walidacyjnym modelu z warstwą aktywacyjną liniową.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.25	0.22	0.24	499
klasa limfocyt (L)	0.21	0.20	0.21	497
klasa monocyt (M)	0.23	0.24	0.24	496
klasa neutrofil (N)	0.25	0.27	0.26	500

stanie się nauczyć. Przez to jest słabszym narzędziem i jego działanie podobne jest do działania liniowych modeli regresji. Nie spełnia on więc odpowiednio swojej funkcji w sieci głębokiej.



Rys. 3.26. Zależność skuteczności i błędu od epoki modelu trenowanego z użyciem aktywacji tangensa hiperbolicznego w warstwach konwolucyjnych.

Aktywacja z użyciem tangensa hiperbolicznego dała jeszcze lepsze efekty. Mimo, że nadal w tej wersji występuje duże niedotrenowanie (3.26), to precyzja klasyfikacji na zbiorze walidacyjnym wzrosła w porównaniu do aktywacji liniowej (3.10). Tangens hiperboliczny jest narażony na występowanie zjawiska zanikania gradientu, dlatego zaleca się stosowanie aktywacji ReLU [39]. Z teo powodu, że żadna

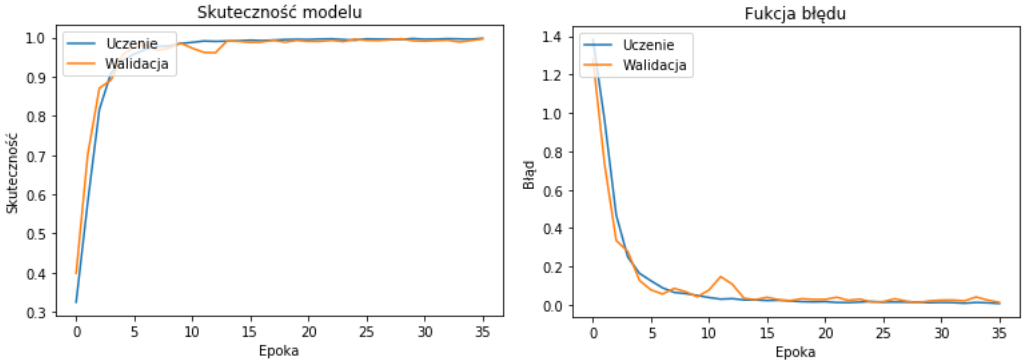
Tabela 3.10. Parametry mierzące jakość klasyfikacji na zbiorze walidacyjnym modelu z warstwą aktywacyjną tangensa hiperbolicznego.

Parametr	Precyzja	Czułość	Miara F1	Ilość próbek
klasa eozynofil (E)	0.26	0.22	0.24	499
klasa limfocyt (L)	0.26	0.27	0.26	497
klasa monocyt (M)	0.25	0.24	0.25	496
klasa neutrofil (N)	0.26	0.29	0.27	500

z aktywacji nie dała lepszych rezultatów niż pierwotne podejście funkcją aktywacyjną pozostaje ReLU (3.12).

4. Analiza wyników

Ostateczny model klasyfikatora po dobraniu odpowiedniej architektury i dostrojeniu parametrów został przetrenowany przez 36 epok, a ostatnia poprawa wartości funkcji błedu nastąpiła w 29 epoce. Przebieg procesu uczenia został przedstawiony na Rys. 4.1. Krzywa walidacji prawie w każdym punkcie nadąża za krzywą uczenia, co oznacza stabilność algorytmu. Wartość skuteczności modelu na zbiorze testowym wynosi 85% (4.1).



Rys. 4.1. Zależność skuteczności i błędu od epoki w procesie uczenia opracowanego klasyfikatora.

Tabela 4.1. Skuteczność opracowanego klasyfikatora.

typ zbioru	**treningowy**	**walidacyjny**	**testowy**
skuteczność [%]	99	99	85

Do klasyfikacji używany jest model uczony przez 29 epok. Dane wejściowe są w obrazami RGB o wymiarach 120x160, przeskalowanymi do zakresu wartości pikseli 0-1. Ramki podzielone są na porcje po 32 obrazy, co daje 248 iteracji w każdej epoce uczenia zgodnie ze wzorem (4.1). Testowanie odbywa się w 62 iteracjach, zgodnie ze wzorem (4.2). Takie podejście pozwala na przejście wszystkich danych przez system w każdej epoce, zarówno podczas uczenia, jak i testowania.

$$i = \left\lfloor \frac{x}{b} \right\rfloor = \left\lfloor \frac{7965}{32} \right\rfloor = 248 \tag{4.1}$$

gdzie,

i – ilość iteracji w jednej epoce treningu,

x – ilość ramek treningowych,

b – ilość ramek w porcji.

$$j = \left\lfloor \frac{y}{b} \right\rfloor = \left\lfloor \frac{1992}{32} \right\rfloor = 62 \tag{4.2}$$

gdzie,

j – ilość iteracji teście,

y – ilość ramek testowych,

b – ilość ramek w porcji.

Na podstawie ogólnej macierzy pomyłek (4.2) można wyprowadzić macierze dla każdej klasy osobno i wyprowadzić parametry pozwalające na dokładniejszą analizę jakości klasyfikacji.

Tabela 4.2. Macierz pomyłek zbioru testowego opracowanego klasyfikatora.

		predykcja			
klasa		E	L	M	N
	E	141	144	141	197
	L	134	149	130	207
	M	145	128	141	206
	N	140	137	140	207

Tabela 4.3. Parametry mierzące jakość klasyfikacji na zbiorze testowym opracowanego klasyfikatora.

Parametr	**Precyzja**	**Czułość**	**Miara F1**	**Ilość próbek**
klasa eozynofil (E)	0.25	0.23	0.24	623
klasa limfocyt (L)	0.27	0.24	0.25	620
klasa monocyt (M)	0.26	0.23	0.24	620
klasa neutrofil (N)	0.25	0.33	0.29	624

5. Podsumowanie

5.1. Kierunki dalszych badań

Bibliografia

[1] Anna Raszeja-Specht Andrzej Szutowicz. „Diagnostyka laboratoryjna". W: 2009.

[2] Alberts B i in. „Leukocyte also known as macrophages functions and percentage breakdown". W: Edinburgh: Churchill Livingstone, 2002.

[3] Korbinian Brodmann. „Vergleichende Lokalisationslehre der Großhirnrinde : in ihren Prinzipien dargestellt auf Grund des Zellenbaues". W: 1985.

[4] Jason Brownlee. *https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/*.

[5] François Chollet. „Deep Learning with Python". W: 2017.

[6] Siddharth Das. *https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5*.

[7] „Deep learning". W: The MIT Press, 2016.

[8] „Deep Learning With Keras". W: Packt Publishing, 2017.

[9] Mark Dow. *http://lcni.uoregon.edu/~dow/Space_software/renderings.html*.

[10] Maksim Drobchak. „*https://www.kaggle.com/drobchak1988/blood-cell-images-acc-92-val-acc-90*". W: 2019.

[11] Muneeb ul Hassan. „VGG16 – Convolutional Network for Classification and Detection". W: 2018.

[12] Kaiming He i in. „Deep Residual Learning for Image Recognition". W: 2015.

[13] Geoffrey E. Hinton, Simon Osindero i Yee Whye Teh. „A Fast Learning Algorithm for Deep Belief Nets". W: 2006.

[14] Ferenc Huszár. *https://www.inference.vc/everything-that-works-works-because-its-bayesian-2/*. 2017.

[15] Christian Ioffe Sergey; Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". W: 2015.

[16] Sergey Ioffe i Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". W: *ArXiv* (2015).

[17] Luke Sung Uk Jung. „*https://www.kaggle.com/jcruxsu/blood-cell-85-kernal-with-inception-v3*". W: 2019.

[18] Gopal Kalpande. „*https : / / medium . com / @gopalkalpande / biological - inspiration - of - convolutional-neural-network-cnn-9419668898ac*". W: 1979.

[19] Nitish Shirish Keskar i in. „On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". W: 2016.

[20] Diederik P. Kingma i Jimmy Ba. „Adam: A Method for Stochastic Optimization". W: 2014.

[21] Departament Badań Społecznych Główny Urząd Statystyczny w Krakowie. „Zdrowie i ochrona zdrowia w 2016 r". W: 2019.

[22] E. Kreyszig. „Advanced Engineering Mathematics". W: 1979.

[23] Alex Krizhevsky, Ilya Sutskever i Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks". W: *NIPS*. 2012.

[24] Thomas Kurbiel i Shahrzad Khaleghian. „Training of Deep Neural Networks based on Distance Measures using RMSProp". W: 2017.

[25] Masakazu Matsugu i in. „Subject independent facial expression recognition with robust face detection using a convolutional neural network". W: 2003.

[26] Paul Mooney. *https://www.kaggle.com/paultimothymooney/blood-cells*.

[27] Paul Mooney. „*https://www.kaggle.com/paultimothymooney/blood-cells*". W: 2017.

[28] nh4cl. „*https://www.kaggle.com/placidpanda/deep-learning-from-scratch-insights*". W: 2018.

[29] Nshafiei. *https://en.everybodywiki.com/File:VGG_structure.jpg*.

[30] Jarun Ontakrai. *https://www.123rf.com/photo_82668549_white-blood-cells-in-in-blood-smear-analyze-by-microscope.html*.

[31] Tadeusiewicz R. „Sieci neuronowe". W: Kraków, Wykład plenarny XXXII Zjazdu Fizyków Polskich, 1993.

[32] Olaf Ronneberger, Philipp Fischer i Thomas Brox. „U-Net: Convolutional Networks for Biomedical Image Segmentation". W: 2015.

[33] Dominik Scherer, Andreas C. Müller i Sven Behnke. „Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". W: *ICANN*. 2010.

[34] Ari Silburt i in. „Lunar crater identification via deep learning". W: 2019.

[35] Karen Simonyan i Andrew Zisserman. „Very deep convolutional networks for large-scale image recognition". W: 2014.

[36] Nitish Srivastava i in. „Dropout: a simple way to prevent neural networks from overfitting". W: 2014.

[37] Ewelina Stefanowicz. *https : // www . hellozdrowie . pl / wyniki - morfologii - krwi - interpretacja - odchylenia-od-normy/*.

[38] *https://krwiodawcy.org/krew-i-grupy-krwi*.

[39] Anish Singh Walia. *https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f/*.

[40] Anish Singh Walia. „*https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f*". W: 2017.

[41] Chi-Feng Wang. *https : / / towardsdatascience . com / the - vanishing - gradient - problem - 69bf08b15484*.

[42] Philip D. Wasserman i Tom J. Schwartz. „Neural networks. II. What are they and why is everybody so interested in them now?" W: 1988.

[43] Paul R. Wheater, H. George Burkitt i Victor G. Daniels. „Functional histology: A text and colour atlas". W: 1979.

[44] Sebastien C. Wong i in. „Understanding Data Augmentation for Classification: When to Warp?" W: 2016.

[45] Yang You, Igor Gitman i Boris Ginsburg. „Scaling SGD Batch Size to 32K for ImageNet Training". W: (2017).