**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 inż. Joanna Jaworek-Korjakowska* |

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

Celem pracy jest zbudowanie narzędzia do 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 weryfiacyjnego 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 wysokopoziomowym 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.

Ciekawą alternatywą dla tych metod byłoby zastosowanie zliczania w czasie rzeczywistym 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.

Bazą do zbudowania tego typu 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. Zawartość pracy

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

Rozdział **??** zawiera opis implementacji programu i zastosowanych metod, po którym następuje krótkie podsumowanie uzyskanych wyników.

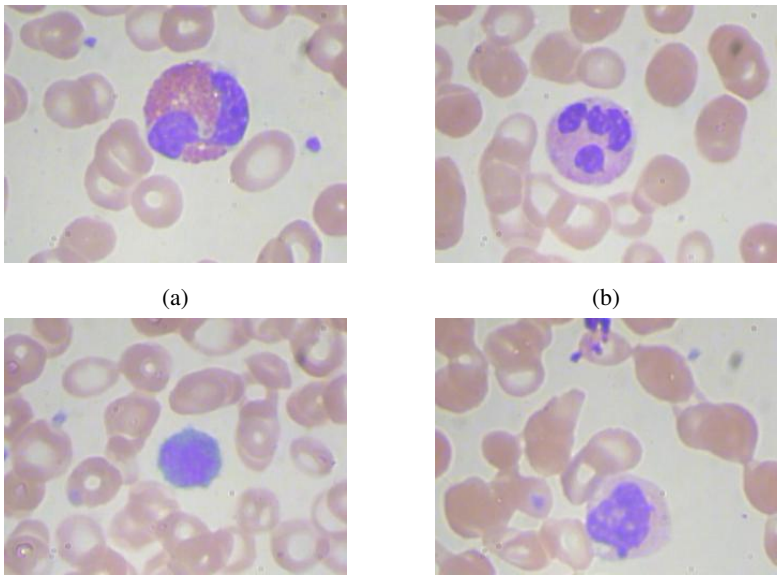
# 2. Część teoretyczna

## 2.1. Aspekt medyczny

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.

| **Nazwa** | Neutrofil | Eozynofil | Limfocyt | Monocyt |
|---|---|---|---|---|
| **Udział %**[**lymphocytes_percentage**] | 54-62 | 1-6 | 25-33 | 2-10 |
| **Średnica $\mu$m**[**lymphocytes_percentage**] | 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. [1] Poniżej przedstawiono przykładowe zjęcia pochodzące z bazy.



(a) (b)

(c) (d)

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

## 2.2. Sieci neuronowe

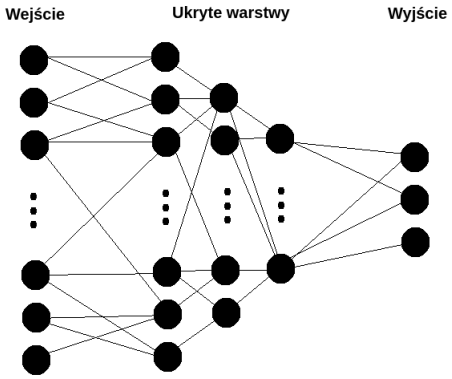
Sieć neuronowa jest to układ przetwarzania danych, składający się z warstw sztucznych neuronów, połączonych synapsami o konkretnych wagach. Neurony wykonują pewne operacje matematyczne na wejściowych danych, a wynik przesyłany jest do kolejnego rzędu neuronów lub do wyjścia układu. Funkcję, realizowaną przez całą sieć można zapisać wzorem: [2]

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

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.




**Rys. 2.2.** Wizualizacja przykładowej struktury sieci neurnowej.

Typy sieci neuronowych ze względu na kierunek przepływu danych:

- Jednokierunkowa *(ang. feedfoward)* - 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.

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

$$y_1 = X_0 W_1 + b_1 \tag{2.2}$$

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.

### 2.2.2. Sieci pretrenowane

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. Sieci konwolucyjne

Konwolucyjne sieci neuronowe *(ang. convolutional neural networks, CNN)* są typem sieci neuronowej głęboko uczonej. Najczęściej używana w rozpoznawaniu obrazów, ze względu na inwariancję względem translacji oraz zdolność uczenia się wzorców lokalnych. [3] Dane wejściowe są w postaci tensora trójwymiarowego, a operacja konwolucji (oznaczona gwiazdką), która zachodzi w sieci może być opisana równaniem: [4]

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.3}$$

gdzie

$I$ – dane wejściowe,

$K$ – jądro *(ang. kernel)*,

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

### 2.3.1. Warstwy konwolucyjne

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)*. [4]

Poniżej 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 braną pod uwagę w dalszym działaniu.

Przykładowe funkcje aktywacyje:

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 |

– progowanie

$$Y < th, A = 0$$
$$Y \geq th, A = 1$$

gdzie

$Y$ – wynik sumy ważonej i błędu

$th$ – próg aktywacji

$A$ – aktywacja

– funkcja liniowa

$$A = cY$$

gdzie

$c$ – stała

– S-funkcja

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

– ReLu

$$A = max(0, Y)$$

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 redkcyjnej typu max pooling:

macierz wejściowa

| | | | | | |
|---|---|---|---|---|---|
| 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 |

wynik

| | |
|---|---|
| 10 | 8 |
| 5 | 9 |

Dzięki tej operacji uzyskuje się niezmienność wyjścia względem małych translacji wejścia. 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. [5]

Hiperparametermi, które można regulować w tej częsci sieci są rozmiar filtru, liczba filtrów, krok filtracji, dopełnianie zerami, funkcja aktywacyjna i funkcja redukcyjna.

### 2.3.2. Klasyfikator

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.

### 2.3.3. 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.

## 2.4. Wstępna obróbka danych

### 2.4.1. Skalowanie i standaryzacja ramek

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.

### 2.4.2. Powiększanie zbioru

W przypadku małych zbiorów danych, rozumianych jako zbiór liczący kilka tysięcy elementów często stosowaną praktyką jest poszerzanie zboru 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. 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. [6]

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*) [6]. Definiuje ona znormalizowany obszar losowego przemieszczenia $u(x, y)$, który dla każdego piksela w obrazie $(x, y)$ definiuje wektor przemieszczenia $R_w$:

$$R_w = R_0 + \alpha u \tag{2.4}$$

gdzie

$R_w$ – lokalizacja piksela w obrazie wyjściowym

$R_0$ – lokalizacja piksela w oryginalnym obrazie

$\alpha$ – wielkość przesunięcia w pikselach

Nie jest wskazane używanie zbioru powiększonego z użyciem dużych transformacji. Dla bazy MNIST przesunięcia większe niż $\alpha >= 8$ 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.[6]

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.

# 3. Część praktyczna

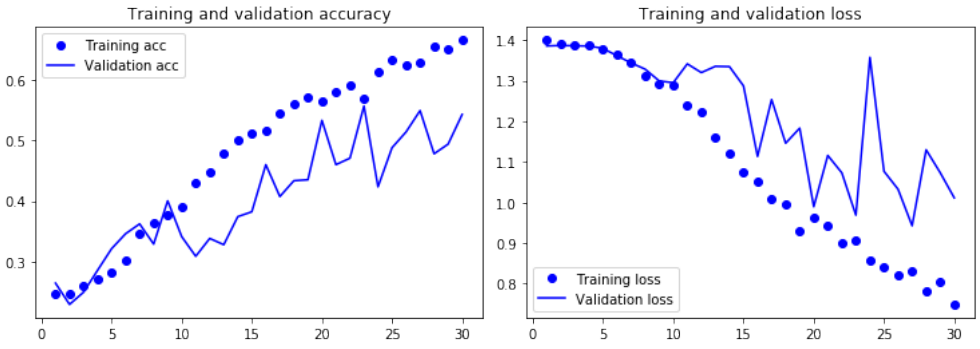
## 3.1. Prosta sieć konwolucyjna

### 3.1.1. Struktura

```
Model:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 118, 158, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 59, 79, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 59, 79, 32)        0         0.2
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 57, 77, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 38, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 28, 38, 64)        0         0.2
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 36, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 18, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 13, 18, 128)       0         0.2
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 16, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 8, 128)         0
_________________________________________________________________
```

```
dropout_4 (Dropout)          (None, 5, 8, 128)         0         0.2
_________________________________________________________________
flatten_1 (Flatten)          (None, 5120)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 5120)              0         0.5
_________________________________________________________________
dense_1 (Dense)              (None, 512)               2621952
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
=================================================================
Total params: 2,864,836
Trainable params: 2,864,836
Non-trainable params: 0
_________________________________________________________________
```

### 3.1.2. Trening




(a) (b)

**Rys. 3.1.** Wyniki treningu modelu (a) dokładność, (b) błąd.

### 3.1.3. Strojenie

Nie wygląda na to, żeby to był dobry kierunek. W sumie to nie wiem jeszcze co należałoby zmienić żeby go dostroić. Na pewno mamy nadpróbkowanie. Może dołożyć BatchNormalisation i więcej Dropoutu?

### 3.1.4. Testowanie

class0: Percentage of correctly classified frames: 0.0

class1: Percentage of correctly classified frames: 100.0

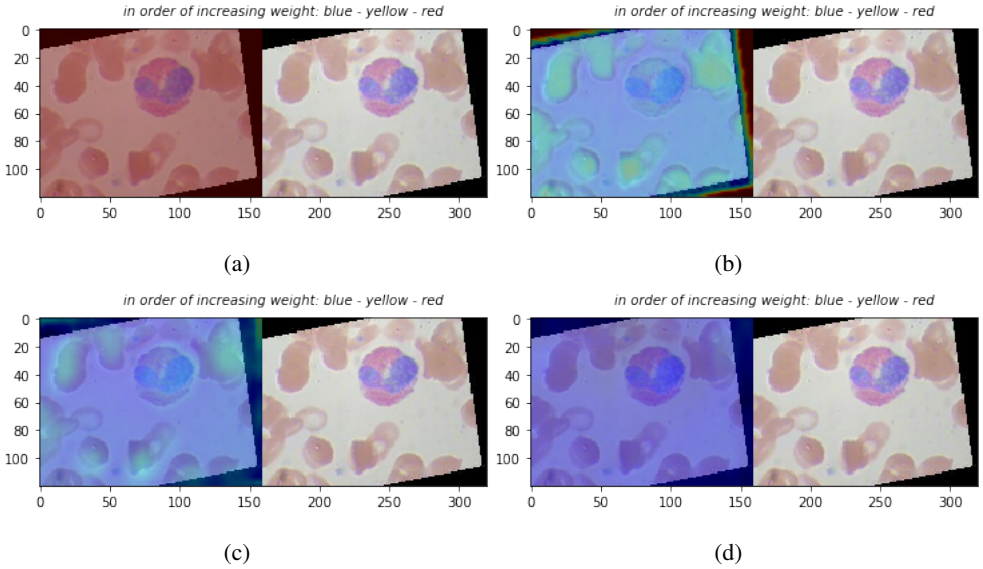
class2: Percentage of correctly classified frames: 0.0

class3: Percentage of correctly classified frames: 0.0

Evaluation on test data

125/125 [==============================] - 679s 5s/step

test loss, test acc: [1.3867403992122562, 0.2492963441785771]




**Rys. 3.2.** Obszary aktywacji w kolejnch warstwach (a) warstwa 1, (b) warstwa 2, (c) warstwa 3, (d) warstwa 4.

### 3.1.5. Wpływ hiperparametrów

Gdy znajdziesz dobry model tego typu.

## 3.2. MobileNet, pretrenowana sieć konwolucyjna

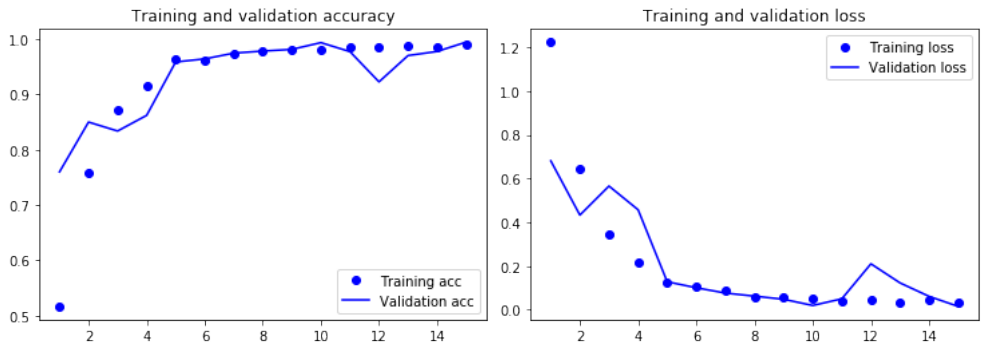
### 3.2.1. Struktura

Jeśli przyjmiemy, że sekwencję warstw: Konwolucyjna, Normalizacja Batcha i ReLu nazywamy A, sekwencję warstw: Depthwise, Normalizacja Batcha i ReLu nazywamy B, zaś sekwencję B,A,B,A nazywamy C, to MobileNet ma strukturę:

```
Model:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 120, 160, 3)       0
conv1_pad (ZeroPadding2D)    (None, 121, 161, 3)       0
A
```

```
B
A
conv_pad_2 (ZeroPadding2D)    (None, 61, 81, 64)        0
B
A
B
conv_pad_4 (ZeroPadding2D)    (None, 31, 41, 128)       0
C
conv_pad_6 (ZeroPadding2D)    (None, 16, 21, 256)       0
C
C
C
conv_pad_12 (ZeroPadding2D)   (None, 8, 11, 512)        0
C
flatten_1 (Flatten)           (None, 15360)             0
dropout_1 (Dropout)           (None, 15360)             0
dense_1 (Dense)               (None, 256)               3932416
batch_normalization_1 (Batch  (None, 256)               1024
dense_2 (Dense)               (None, 4)                 1028
=================================================================
Total params: 7,163,332
Trainable params: 7,140,932
Non-trainable params: 22,400
_________________________________________________________________
```

### 3.2.2. Trening



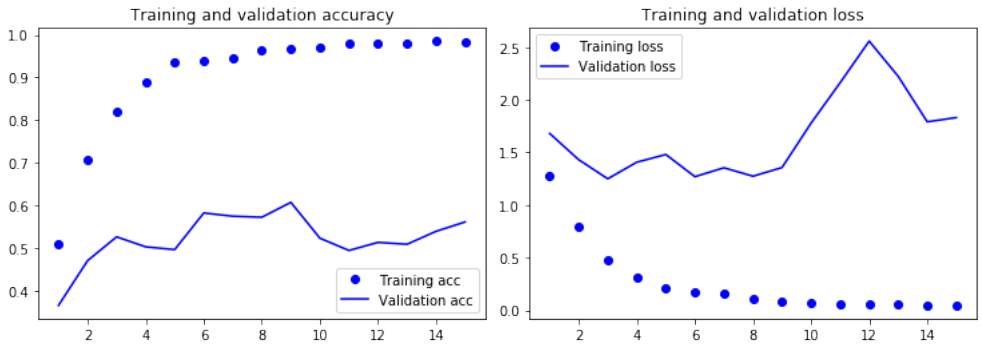

(a) (b)

**Rys. 3.3.** Wyniki treningu modelu (a) dokładność, (b) błąd.

### 3.2.3. Strojenie

Model nie wymagał strojenia ani wielokrotnego trenowania, bo przy pierwszym podejściu doszedł do 99% na zbiorze walidacyjnym oraz funkcja błędu malała wraz ze wzrostem skuteczności. Parametry:

```
frame_size = (120, 160)
activation = 'relu'
output_activation = 'softmax'
optimizer = optimizers.RMSprop(lr=1e-4)#optimizers.Adam(lr = 1e-4)
loss = 'categorical_crossentropy'
metrics = ['acc']
rescale = 1./255
batch_size = 20
epochs=15
validation_set_percentage=80
freezed_layers = range(0) <---- wszystko trainable, bez sensu i pewnie to właś
powód spadku dokładności na nowych danych w porównaniu z treningiem
freeze 21: w 7 epoce 90%/60%
freeze 27: w 7 epoce 95%/50%
freeze 9:  w 7 epoce 93%/57%
Jednak nie, też nie działa, ostatnia kombinacja poniżej:
```



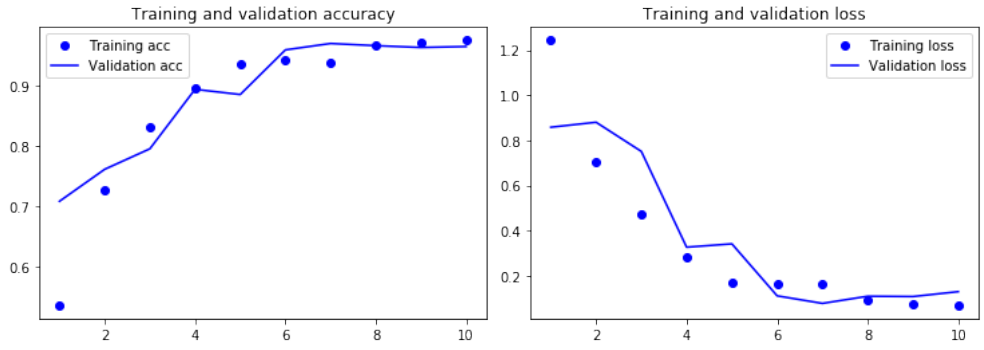

(a) (b)

**Rys. 3.4.** Wyniki treningu modelu (a) dokładność, (b) błąd.

Zamiast strojenia przetrenowano nowy model (bez zamrożenia warstw). Na przetasowanych danych i ze zmienioną liczbą epok na 10 (żeby uniknąć overfittingu który wystąpił w 12 epoce). Wyniki treningu:

### 3.2.4. Testowanie

Dla pierwszej wersji (15 epok): Gdy wynik predykcji dla danej klasy przekracza 25% i predykcja jest zgodna z przynależnością do klasy to uznaje się za ramkę dobrze zakwalifikowaną.




(a) (b)

**Rys. 3.5.** Wyniki treningu modelu (a) dokładność, (b) błąd.

class0: Percentage of correctly classified frames: 18.973214285714285
class1: Percentage of correctly classified frames: 92.82511210762333
class2: Percentage of correctly classified frames: 12.584269662921349
class3: Percentage of correctly classified frames: 12.026726057906458
Dla 10 epok: class0: Percentage of correctly classified frames: 23.4375
class1: Percentage of correctly classified frames: 95.96412556053812
class2: Percentage of correctly classified frames: 2.0224719101123596
class3: Percentage of correctly classified frames: 0.6681514476614699

### 3.2.5. Wpływ hiperparametrów

Gdy znajdziesz dobry model tego typu.

## 3.3. Sieć o architekturze U-Net dla segmentacji jednej klasy - to chyba bez sensu

### 3.3.1. Struktura

### 3.3.2. Trening

### 3.3.3. Strojenie

### 3.3.4. Testowanie

### 3.3.5. Wpływ hiperparametrów

# Bibliografia


[1] *Leukocyte also known as macrophages functions and percentage breakdown*. ISBN 0-8153-4072-9. Edinburgh: Churchill Livingstone, 2002.

[2] Tadeusiewicz R. „Sieci neuronowe". W: Kraków, Wykład plenarny XXXII Zjazdu Fizyków Polskich, 1993, s. 5.

[3] *Deep learning with Python*. ISBN:9781617294433. Manning, 2018.

[4] *Deep learning*. ISBN:0262035618 9780262035613. The MIT Press, 2016.

[5] Scherer D.; Müller A.C.; Behnke S. „Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". W: Artificial Neural Networks (ICANN), 20th International Conference on. Thessaloniki, Greece, 2010, 92–101.

[6] Sebastien C. Wong; Adam Gatt; Victor Stamatescu; Mark D. McDonnell. „Understanding data augmentation for classification: when to warp?" W: Gold Coast, QLD, Australia: Institute of Electrical i Electronic Engineering, 2016, s. 2.