

Politechnika Łódzka
Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki
Instytut Informatyki Stosowanej

PRACA DYPLOMOWA INŻYNIERSKA

System umożliwiający digitalizację zapisów nutowych pieśni,
ułatwiający ich przechowywanie i przetwarzanie

A system for digitization of sheet music facilitating their storage and processing

Przemysław Woldon
Numer albumu: 195092

Opiekun pracy:
Dr. inż. Paweł Kapusta

Łódź, Wrzesień 2018

Spis treści

Streszczenie	3
Abstract	3
1 Wstęp	4
1.1 Cel pracy	4
2 Wykorzystane technologie	5
2.1 Biblioteka OpenCv	5
2.1.1 Zarys historyczny	5
2.1.2 Popularność projektu	7
2.1.3 Zastosowania biblioteki OpenCv	7
2.1.4 Reprezentacja obrazów	8
2.2 Konwolucyjne sieci neuronowe	9
2.2.1 Sieci konwolucyjne w języku Java	12
2.3 Tesseract OCR	14
2.4 Środowisko programistyczne	15
3 Część praktyczna	16
3.1 Budowa aplikacji	16
3.1.1 Ekstrakcja stron z plików pdf	17
3.1.2 Polepszenie jakości obrazów	18
3.1.3 Eliminacja nadmiarowych (czarnych) obszarów skanu	21
3.1.4 Podział skanu na dwie strony	26
3.1.5 Zasadnicze przygotowanie obrazu do analizy	31
3.1.6 Wyprostowanie strony	34
3.1.7 Główne przetwarzanie obrazu	38
3.1.8 Konfiguracja sieci neuronowej klasyfikującej linie	42
3.1.9 Detekcja tekstu	44
3.1.10 Detekcja i wycięcie liter	45
3.1.11 Konfiguracja sieci neuronowej klasyfikującej litery	46
3.1.12 Klasyfikacja odczytanych liter	48
3.1.13 Graficzny interfejs użytkownika	49
4 Podsumowanie	50
5 Bibliografia	51

Streszczenie

Niniejsza praca dyplomowa w sposób zwięzły opisuje prototyp systemu umożliwiającego pozyskanie tekstu z obrazów stron śpiewników kościelnych i jego dalsze możliwości rozwoju.

Na pracę pisemną składają się dwie zasadnicze części: teoretyczna i praktyczna. Część teoretyczna opisuje wybrane technologie, które zostały wykorzystane w systemie. Wykorzystanym językiem programowania w którym powstała aplikacja był język Java, który determinował wybór frameworku dla sztucznych sieci neuronowych – Deeplearning4J. Inne technologie które znalazły zastosowanie w systemie to OpenCv, wykorzystany do cyfrowego przetwarzania obrazów i system OCR Tesseract. Połączenie tych komponentów i zadbanie o ich właściwą współpracę umożliwiło prawidłowe działanie aplikacji. Część praktyczna w sposób przejrzysty przedstawia szczegóły implementacyjne algorytmów. W podsumowaniu autor w sposób krytyczny rekapituluje wykonaną pracę, zwraca uwagę na zalety i wady wypracowanego rozwiązania, jednocześnie podając sposoby na rozwiązywanie powstałych problemów oraz wskazuje drogi rozwoju projektu.

Słowa kluczowe: ?Partytura?, OpenCV, Przetwarzanie obrazów, Konwolucyjne Sieci Neuronowe, Tesseract OCR

Abstract

Keywords: ?Music sheet detection?, OpenCV, Image Processing, Convolutional Neural Network, Tesseract OCR

1 Wstęp

Informatyzacja dotyczy wielu obszarów naszego życia. Jednym z nich jest sztuka (w tym muzyka, muzyka sakralna). Obecnie coraz częściej w świątyniach możemy spotkać się z rzutnikami lub wyświetlaczami LED, na których ekranizowane są teksty pieśni. Cyfrowe zbiory pieśni dostarczane przez producentów wyświetlaczy nie są w stanie sprostać wymaganiom jakie stawia przed nimi: dynamiczny rozwój muzyki – powstawaniu nowych utworów, zróżnicowanym tradycjom lokalnym oraz indywidualnym upodobaniom muzyków kościelnych wykorzystującym różne śpiewniki. Czasochłonny proces budowy wspomnianych zbiorów często polega na ręcznym przepisywaniu tekstów pieśni z wykorzystywanych przez muzyków kościelnych śpiewników. Automatyzacja tego procesu pozwoli na łatwiejsze dopasowanie zbiorów do potrzeb muzyków kościelnych i tradycji lokalnych. Przed przystąpieniem do prac nad aplikacją autor nie spotkał się z wykorzystaniem na rynku polskim oprogramowania rozwiązującego powyższe problemy.

Dzięki pracy nad tą aplikacją autor mógł połączyć swoje dwie pasje – muzykę kościelną i informatykę. Pomyśl budowy aplikacji narodził się z zaobserwowania, że osoby odpowiedzialne za muzykę kościelną przeznaczają znaczącą liczbę godzin na ręczne przepisywanie śpiewników, poprawę błędów w już dostarczonych zbiorach. Zaangażowanie technik informatycznych do tego zadania umożliwia szybszą budowę własnych zbiorów pieśni. Budowa prototypu zapewniającego pełną funkcjonalność, daje możliwość jego opublikowania i dotarcia do szerokiego grona interesariuszy, co może stanowić punkt wyjściowy do dalszych prac nad aplikacją tak aby wypracowane rozwiązanie było użyteczne, niezawodne, wydajne i wspierane.

1.1 Cel pracy

Celem niniejszej pracy dyplomowej było skonstruowanie algorytmów przetwarzających strony śpiewników kościelnych tak, aby jak najlepiej przygotować je do optycznego rozpoznania zawartego w nich tekstu. W algorytmach wykorzystane zostały metody przetwarzania obrazów, które redukują szумy, usuwają obszary, które nie zawierają znaczących w analizie danych (eliminacja nadmiarowych informacji), prostują strony, wykrywają i usuwają znaki muzyczne, wykrywają znaki (cyfry, litery, znaki interpunkcyjne), zaś algorytmy cyfrowego rozpoznawania tekstu odczytują wykryte litery uwzględniając polskie znaki diakrytyczne. Dzięki zastosowaniu konwolucyjnych sieci neuronowych wypracowane rozwiązanie zostało uogólnione, a finalne algorytmy umożliwiają pracę na różnych danych wejściowych. Zaś budowa aplikacji uwzględniająca dobre praktyki programowania ułatwi jej dalszy rozwój.

2 Wykorzystane technologie

Założenia początkowe, które miał spełnić system możliwe były do osiągnięcia przy wykorzystaniu różnych technologii dostępnych na rynku. Wybrane zostały następujące narzędzia: do przetwarzania obrazów – biblioteka OpenCv, do budowy sztucznych sieci neuronowych – DL4J, do odczytania tekstu – Tesseract, zaś całość została napisana w języku programowania Java. Właściwe połączenie i wykorzystanie tych elementów pozwoliło na zrealizowanie założeń systemu. Technologie te zostały opisane w kolejnych podrozdziałach wraz z ich zaletami, które zadecydowały o ich wyborze.

2.1 Biblioteka OpenCv

Cyfrowe przetwarzanie obrazów (*ang. digital image processing*) jest dziedziną nauki zajmującą się reprezentacją obrazu w pamięci komputera, jego akwizycją i przetwarzaniem. Obecnie dostępnych jest wiele bibliotek umożliwiających pracę z obrazami, na przykład: ImageMagick, Generic Image Library, SIMD i inne. Najbardziej powszechną biblioteką jest jednak biblioteka OpenCv. Wieloletnia praca nad algorytmami przetwarzającymi obrazy pozwoliła zbudować optymalną i bardzo bogatą, bezkonkurencyjną funkcjonalność. Ponadto OpenCv posiada przejrzystą i zawierającą podstawy matematyczne dokumentację, zaś dynamicznie rozwijająca się społeczność zgromadzona wokół biblioteki ułatwia z nią pracę szczególnie nowym użytkownikom.

2.1.1 Zarys historyczny

Prace nad budową tej biblioteki rozpoczął jeden z pracowników firmy Intel – Gary Rost Bradski, zainspirowany środowiskiem akademickim, które wówczas posiadało bardzo bogatą infrastrukturę służącą do przetwarzania obrazów, jednak przeznaczoną dla użytku wewnętrznego. Studenci w obrębie jednej jednostki akademickiej dzielili się kodem zawierającym gotowe implementacje algorytmów, co znacząco ułatwiało pracę przy własnych projektach czy aplikacjach. Stąd głównym celem biblioteki OpenCv jest udostępnienie wszystkim zainteresowanym zagadnieniami przetwarzania obrazów i sztucznej inteligencji gotowej, jednolitej, darmowej infrastruktury pozwalającej na pracę, tak aby nie trzeba było ponownie "wynajdywać koła".

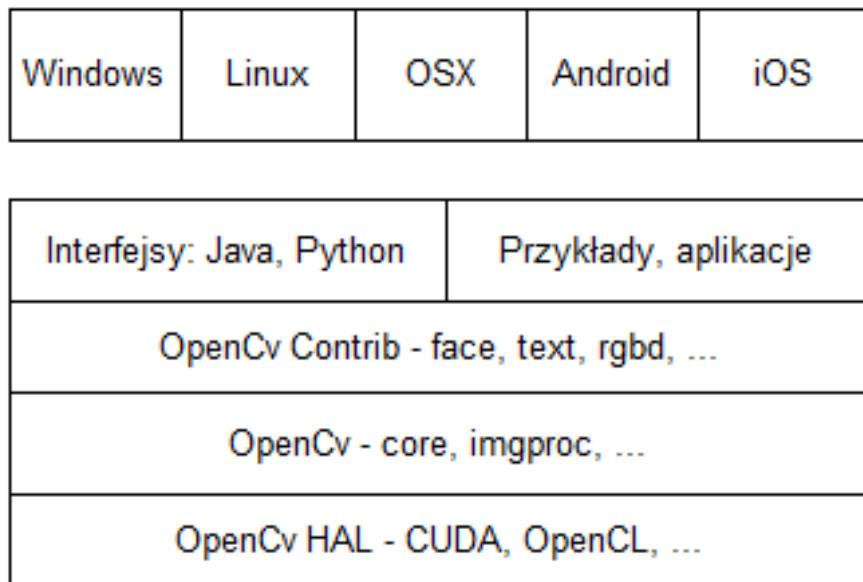
OpenCv zostało przedstawione po raz pierwszy w 1999 roku szerszemu gronu odbiorców. W wersji 1.0 kod biblioteki stanowiły wyłącznie najbardziej użyteczne przy przetwarzaniu obrazów algorytmy zaimplementowane w języku C. Od tego czasu biblioteka znacząco się

zmieniła.

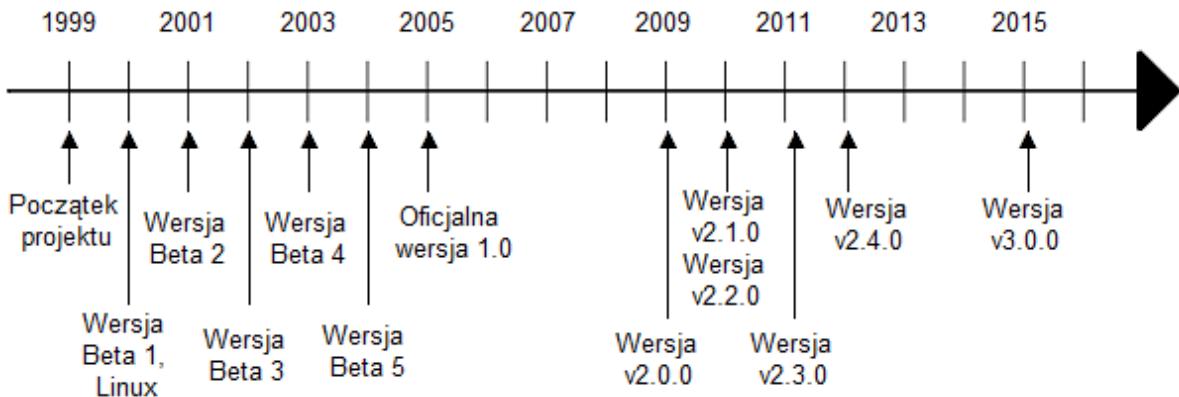
W wersji 2.0 znaczący wpływ na wydanie wywarły trendy obecne w prowadzeniu projektów, w których wytwarzają się oprogramowanie – repozytorium kodu w systemie kontroli wersji Git, gdzie możemy znaleźć najnowszą wersję biblioteki i najświeższe poprawki, testy jednostkowe czy booty ciągłej komplikacji. Zaimplementowano również interfejsy dla języków programowania takich jak C++ oraz Javy, Python'a, MATLAB-a. Od tego czasu nowe typy danych i funkcje metody implementowano w C++, a już napisane w języku C dopasowano do nowej technologii.

Nieustannie zwiększająca się liczba zaimplementowanych algorytmów użytkowych przy przetwarzaniu obrazów przyczyniła się do modułowej budowy biblioteki w wersji 3.0, wspiera ona budowanie i dołączanie do biblioteki własnych modułów, przedstawia się ona w sposób następujący:

- (warstwa wierzchnia) system operacyjny,
- interfejsy dla różnych języków i aplikacje,
- moduł *opencv_contrib* zawierający kod napisany i dołączony do biblioteki przez użytkowników,
- rdzeń OpenCv,
- optymalizacje sprzętowe (warstwa HAL *ang. hardware acceleration layer*).



Rysunek 1: Budowa modułowa OpenCv



Rysunek 2: Rozwój projektu w czasie

2.1.2 Popularność projektu

Projekt cieszy się bardzo dużym powodzeniem, które nieustannie rośnie. Świadczy o tym liczba pobrań wynosząca około 14 milionów razy, zaś miesięcznie około 200 tysięcy. Obecnie biblioteka zawiera ponad 2500 zoptymalizowanych algorytmów, które służą do przetwarzania obrazów również w czasie rzeczywistym i uczenia maszynowego, co znaczco ułatwia tworzenie aplikacji użytkownikom. Programiści piszący kod uwzględniali wymog przenośności OpenCv, a więc możliwości komplikacji na każdym odpowiednim kompilatorze języka C++, co wymusiło restrykcyjną zgodność ze standardem i ułatwiło obsługę różnych platform. Biblioteka dostępna jest na systemach operacyjnych takich jak: Windows, Linux, Mac OS X, do których dołączyły systemy operacyjne platform mobilnych (Android i iOS), co znaczco przyczyniło się do zwiększenia liczby użytkowników.

2.1.3 Zastosowania biblioteki OpenCv

Dynamiczny rozwój technologiczny otwiera przed biblioteką nowy horyzont użyteczności. OpenCv znajduje zastosowanie w wielu obszarach życia do tego stopnia że wykorzystanie projektu wydaje się rzeczą całkowicie naturalną, a wręcz niezauważalną. Biblioteka wykorzystywana jest w:

- skanowaniu kodów QR,
- monitoringu,
- rozpoznawaniu dźwięków i muzyki,
- obrazowaniu medycznym,
- robotyce,
- przemyśle – produkcji masowej i kontroli jakości,

- wojsku – bezzałogowe pojazdy, fotografie lotnicze,
- analizie obiektów,
- Google Street View

2.1.4 Reprezentacja obrazów

Obrazy można podzielić w najprostszy sposób na obrazy barwne i obrazy w skali szarości. Obrazy w bibliotece OpenCv reprezentowane są w postaci macierzy (Mat). Narzędzie to udostępnia bogatą gamę możliwości reprezentacji poszczególnego piksela, a więc i samego obrazu. Tak więc jeden piksel jest reprezentowany w następujący sposób – CV_xCy, gdzie:

- x – parametr określający typ
 - 8U – 8-bitowy char bez znaku
 - 16S – 16-bitowy short ze znakiem
 - 16U – 16-bitowy short bez znaku
 - 32S – 32-bitowa liczba całkowita ze znakiem
 - 32F – 32-bitowa liczba zmiennoprzecinkowa
 - 64F – 64-bitowa liczba zmiennoprzecinkowa
- y – parametr określający liczbę kanałów
 - 1 – jeden kanał
 - 2 – dwa kanały
 - 3 – trzy kanały

Jeśli użytkownik potrzebuje więcej kanałów może użyć funkcji CV_xC(n), gdzie n – liczba kanałów. Można również wybierać z obfitej przestrzeni reprezentacji obrazu np. RGB, RGBA, HSV i innych.

Obrazy przechowywane jako pliki w pamięci komputera mają różne rozszerzenia, które wpływają na ich jakość, wielkość zajmowanego miejsca na dysku. Przykładowe formaty obrazów:

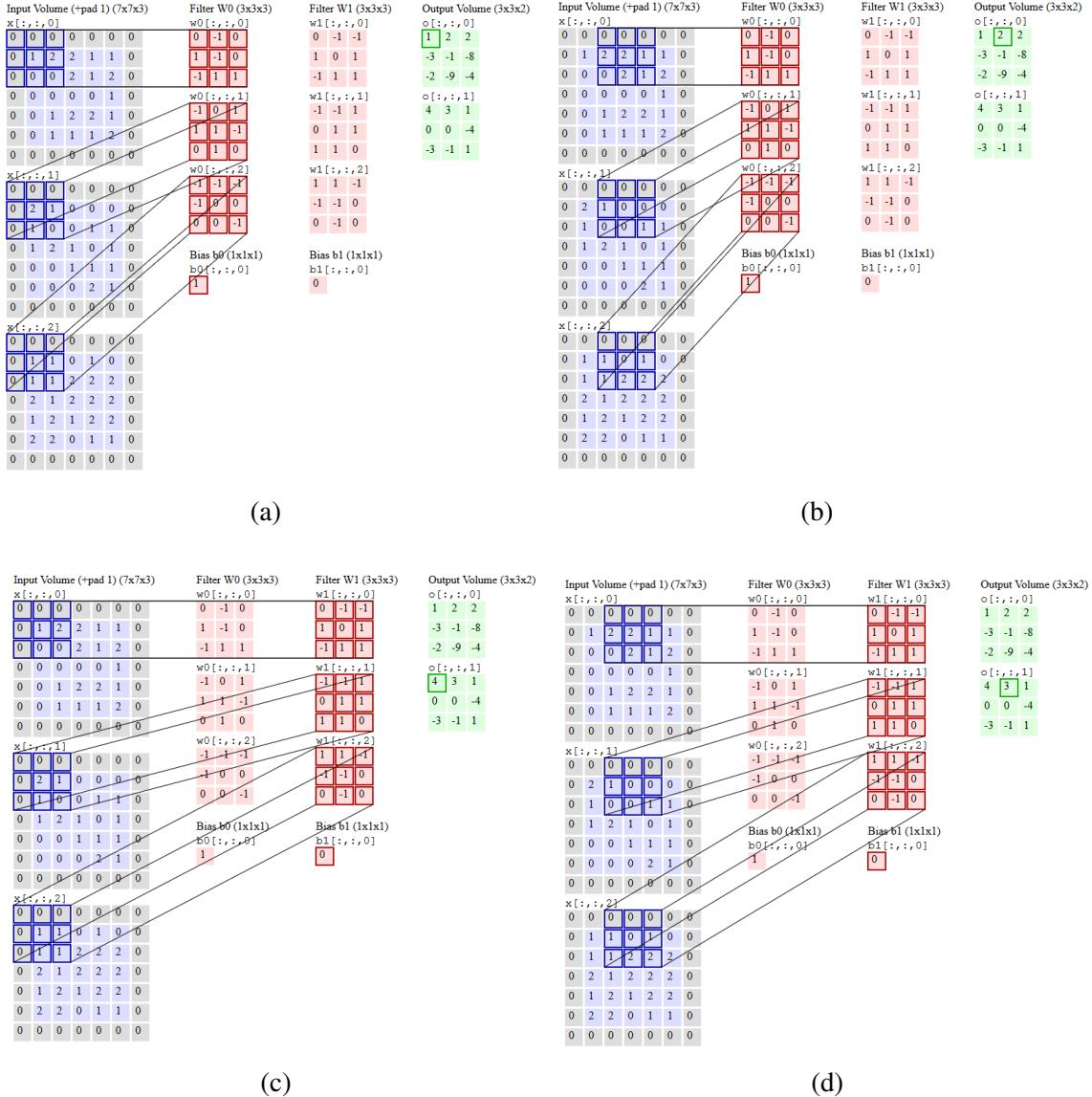
- TIFF (*ang. Tagged-Image File Format*) – format standardowy i najbardziej podstawowy, używa algorytmu kompresji bezstratnej;
- JPEG (*ang. Joint Photographic Experts Group*) – używa algorytmu kompresji stratnej;
- PNG (*ang. Portable Network Graphics*) – używa algorytmu kompresji bezstratnej;
- GIF (*ang. Graphics Interchange Format*) – używa algorytmu kompresji bezstratnej LZW (*Lemple-Zif-Welch*), pliki mają małe rozmiary, często jest wykorzystywany do tworzenia animacji;

2.2 Konwolucyjne sieci neuronowe

Działanie sztucznych sieci neuronowych (*Artificial neural network (ANN)*) opiera się na modelu pojedynczego neuronu (perceptronu), które zostają łączone w większe struktury. Matematyczną podstawę dla działania konwolucyjnych sieci neuronowych (*Convolutional neural networks (CNN)*) stanowi splot (konwolucja) – jest to operacja określana na dwóch funkcjach (np. $f(t)$, $g(t)$) w wyniku której otrzymujemy nową funkcję (np. $h(t)$); w przetwarzaniu obrazów jedną z funkcji jest analizowany obraz wejściowy, zaś druga filtr. Obraz wejściowy przechodzi przez wiele filtrów i każdy z nich mapuje cały obraz wejściowy biorąc pod uwagę jego części o rozmiarze filtra. Tak więc konwolucyjne sieci neuronowe uczą się obrazu, analizując go pewnymi wycinkami. Dokładnie filtr przechodzący przez obraz mnoży odpowiednie wartości a następnie sumuje ich iloczyny.

$$h[m, n] = (f \star g)[m, n] = \sum_j \sum_k f[j, k]g[m - j, n - k]$$

Przykład działania konwolucji został zaprezentowany na Rysunku 3. Przez każdy z kanałów obrazu wejściowego (kolor niebieski) przechodzą dwa filtry (kolor czerwony). Krok z jakim filtr przesuwa się wynosi dwa piksele (dwie kolumny) co można zaobserwować na parze (a) i (b) oraz parze (c) i (d). Wynik działania każdego z filtrów zapisywany jest do macierzy wynikowej (kolor zielonego).



Rysunek 3: Przykład działania konwolucji
 Źródło: <https://skymind.ai/wiki/convolutional-network>, data dostępu: 09.07.2018

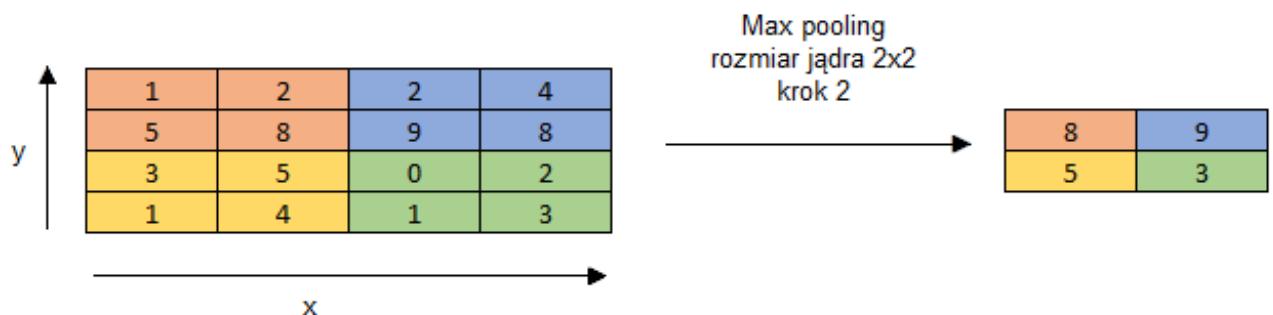
Następnie dane wejściowe przechodzą przez nieliniową transformację (funkcję aktywacji), która określa czy perceptron będzie aktywowany czy nie. Funkcja ta odgrywa znacząca rolę w dopasowaniu gradientów. Funkcje te są ciągłe i różniczkowalne (wyjątek stanowi funkcja ReLU w zerze).

Wśród funkcji aktywacji wyróżniamy funkcje takie jak: funkcja sigmoidalna, tangens hiperboliczny (*tanh*), rektyfikowana jednostka liniowa (*ang. The Rectified Linear Unit (ReLU)*), które mapują wartość wejściową w zakresach odpowiednio $(0, 1)$, $(-1, 1)$ i $(0, x)$. Funkcja softmax przetwarza wektor wejściowy na wektor wyjściowy według następującego wzoru będącego rozkładem prawdopodobieństwa, mówiącego o prawdopodobieństwie wystąpienia poszczególnej klasy.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

Suma wartości wektora wyjściowego wynosi jeden. Wektor wejściowy oraz wyjściowy mają taki sam rozmiar. Funkcja normalizuje K wymiarowy wektor w K wymiarowy wektor $\sigma(z)$.

Zjawiskiem występującym przy zastosowaniu konwolucyjnych sieci neuronowych jest pooling (Rysunek 4). Polega on na zmniejszeniu wymiarowości przetwarzanego obrazu. Zaletą zastosowania poolingu jest wyeksponowanie znaczących informacji, które zawiera obraz. Można wyróżnić average pooling i max pooling i inne. Pooling może być wykonany na każdym z kanałów obrazu z różnym rozmiarem jądra oraz różnym krokiem.



Rysunek 4: Max pooling

Pooling jest również metodą regularyzacji sieci neuronowej, zapobiegający jej przeuczeniu. Algorytmy służące regularyzacji stosuje się, aby sieć lepiej klasyfikowała dane testowe. Dzięki różnym techniką regularyzacji zmniejszany jest błąd generalizacji sieci neuronowej (jednak nie błąd szkoleniowy).

Inną efektywną drogą regularyzacji sieci neuronowych również zapobiegającą zjawisku przeuczenia się sieci jest dropout. Podczas treningu sieci zostają wykluczone, dezaktywowane nadmiarowe neurony z architektury sieci w sposób losowy, w efekcie może on uwzględniać np. neurony o najmniejszych wagach, które nie mają większego wpływu na ostateczny wynik działania sieci. Tak odrzucone neurony nie biorą już udziału w końcowym procesie wnioskowania. Obok dropout'u istnieją takie metody zapobiegające przeuczeniu się sieci jak: regularyzacja l1 i l2, zmniejszające wartość absolutną, pierwiastek kwadratowy wag, uznając, że modele mające zminimalizowane wagi osiągają lepsze wyniki. Regularyzacja, a w szczególności dropout pozytywnie wpływa na minimalizację błędu klasyfikacji sieci neuronowej¹

¹Na podstawie: https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/dropout_layer.html, dostęp 2018.08.03.

Zasadniczym metodą bez której uczenie się sieci neuronowych byłoby niemożliwe, a której implementacje jednocześnie służą optymalizacji sieci neuronowych (wielowarstwowych) jest algorytm wstecznej propagacji błędów (*ang. backpropagation*). Wagi poszczególnych połączeń mogą zostać wytrenowane, a więc dobrane automatycznie i stanowiące w przybliżeniu zestaw najbardziej optymalny. Wsteczna propagacja błędów polega na aktualizowaniu wartości wag na podstawie wartości błędu wag dotychczasowych. W celu poprawy efektywności wstecznej propagacji błędów często zaleca się następujące metody: zastosowanie losowych wag początkowych, kilkukrotne powtarzanie procesu uczenia. Sam algorytm uwzględnia następujące kroki zakładając losową generację wag początkowych: obliczenie odpowiedź sieci dla każdego z wektorów uczących, każdy z neuronów wyjściowych oblicza swój błąd, propagacja błędów do warstw wcześniejszych, na podstawie propagowanej wartości błędu każdy z neuronów modyfikuje wagę, powyższe kroki są powtarzane dopóki średni błąd nie przestanie maleć.

2.2.1 Sieci konwolucyjne w języku Java

Do budowy konwolucyjnych sieci neuronowych został wykorzystany framework Deeplearning4j. Działa on na wirtualnej maszynie Javy. Można go wykorzystywać z językiem programowania Java (DL4J) i Scala (DL4S). Narzędzie posiada bogatą dokumentację, repozytorium kodu na platformie GitHub gdzie znajdują się zaimplementowane gotowe przykłady użycia frameworku <https://github.com/deeplearning4j/nd4j>. Umożliwia konfigurację środowiska za pośrednictwem narzędzi automatyzujących budowę projektu, na przykład: Maven, Gradle, Ivy, SBT. Do operacji backendowych narzędzie udostępnia wybór pomiędzy GPU, a CPU. Framework posiada zaimplementowane mechanizmy budowy, regularyzacji, optymalizacji sieci neuronowych, różnorodne rodzaje warstw z których możemy budować architektury sieci (główne grupy) : Feed-Forward Layers, Output Layers, Convolutional Layers, Recurrent Layers, Unsupervised Layers, Other Layers, Graph Vertices, InputPreProcessors. W pracy wykorzystywane są następujące z udostępnionych warstw wraz z konfiguracją:

- ConvolutionLayer – warstwa wykonująca konwolucję
 - ConvolutionLayer.Builder (x, y) – określa wielkość jądra; x – jego szerokość w pikselach, a y – jego wysokość w pikselach,
 - stride (m, n) – określa krok z jakim jądro jest przesuwane na obrazie wejściowym; m – przesunięcie na osi odciętych w pikselach, a n – przesunięcie na osi odciętych w pikselach,
 - nOut (k) – określa liczbę filtrów przez które przechodzi obraz;
- SubsamplingLayer – warstwa odpowiadająca za pooling,

- SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.n) – n – określa rodzaj poolingu z pośród: MAX, AVG, SUM, PNORM;
- kernelSize (m, n) – określa wielkość jądra; m – jego szerokość w pikselach, a n – jego wysokość w pikselach,
- DenseLayer – warstwa standardowa biorąca pod uwagę wszystkie wyjścia z warstwy poprzedniej,
- OutputLayer – warstwa wyjściowa odpowiadająca za klasyfikację;

Narzędzie wspiera następujące parametry konfiguracji sieci lub poszczególnych warstw (Listing 1) : funkcja aktywacji , inicjalizacja wag początkowych, aktualizacje kroku uczenia, regularyzację i inne. Wszystkie dostępne warstwy i parametry konfiguracji sieci dostępne są na stronie projektu: <https://deeplearning4j.org/quickref#config>.

```

1 Map<Integer, Double> lrSchedule = new HashMap<Integer, Double>();
2   lrSchedule.put(0, 0.001); // iteracja #, wskaznik uczenia
3   lrSchedule.put(100, 0.0015);
4   lrSchedule.put(200, 0.0001);
5   lrSchedule.put(400, 0.00015);
6   lrSchedule.put(800, 0.00001);
7
8 MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
9   .seed(seed) //losowo generowana inicjalizacja wag
10  .l2(1e-4) //regularyzacja l2
11  .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
12  //algorytm optymalizacji
13  .updater(new Nesterovs(new MapSchedule(ScheduleType.ITERATION, lrSchedule)))
14  //okreslenie kroku uczenia sie sieci w zaleznosci od iteracji
15  .weightInit(WeightInit.XAVIER) //inicjalizacja wag
16  .list() //lista kolejnych warstw sieci
17  .layer(0, new ConvolutionLayer.Builder(5, 5)
18    .nIn(channels) //liczba kanalow obrazu
19    .stride(1, 1)
20    .nOut(25)
21    .activation(Activation.TANH)
22    .build())
23  .layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
24    .kernelSize(2, 2)
25    .stride(2, 2)
26    .build())
27  .layer(2, new DenseLayer.Builder().activation(Activation.RELU)
28    .nOut(500).build())
29  .layer(3, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
30    .nOut(outputNum)
31    .activation(Activation.SOFTMAX)
32    .build())
33  .setInputType(InputType.convolutionalFlat(600, 15, 1))
34  //600 - szerokosc, 15 - wysokosc, 1 - liczba kanalow obrazu wejsciowego
35  .backprop(true).pretrain(false).build();

```

```
36 //backprop - włączenie algorytmu propagacji wstecznej,  
37 //pretrain - w przypadku uczenia bez nadzoru sieci specjalnych  
38 //Restricted Boltzmann machine, autoencoders wartosc true  
39 //w innych przypadkach false
```

Listing 1: Przykładowa konfiguracja konwolucyjnej sieci neuronowej

2.3 Tesseract OCR

Tesseract jest systemem służącym do optycznego rozpoznawania znaków (OCR *ang. optical character recognition*) opracowanym przez firmę HP, obecnie projekt jest sponsorowany przez Google. System został napisany w języku C. Dostępne są interfejsy umożliwiające korzystanie z narzędzia, z poziomu innych języków np. Java (wrapper Tess4J).

Pierwsza wersja umożliwiała pracę z językiem angielskim, kolejne wydania powiększały listę pakietów językowych, najpierw o inne języki zachodnioeuropejskie, a później o inne języki używane w świecie. Na szczególną uwagę wśród nich zasługują japoński, chiński, i języki semickie czytane od prawej do lewej. Obecne narzędzie udostępnia wybór pakietu językowego uwzględniającego znaki diakrytyczne wśród około 130 możliwych. Oprócz obsługi różnych języków możliwa jest obsługa różnych czcionek.

W początkowej wersji Tesseract pozwalał na pracę z plikami o rozszerzeniu TIFF. Kolejne wersje systemu umożliwiają pracę z bogatszą gamą plików: JPEG, GIF, PNG, BMP, wielostronicowych plików TIFF dzięki wykorzystaniu biblioteki służącej do przetwarzania obrazów – Leptonica (wrapper Java – Lept4J). Dzięki współpracy dodatkowych bibliotek (GPL Ghostscript lub PDFBox) możemy odczytywać tekst zawarty w plikach PDF.

Dodatkowymi funkcjonalnościami, które zasługują na uwagę są: możliwość analizy układu strony, uzyskania informacji o położeniu rozpoznanego tekstu na stronie i formatowanie tekstu wyjściowego.

Projekt dostępny jest na wielu platformach: Windows, Linux, Mac OS X. Tesseract nie udostępnia graficznego interfejsu użytkownika, a sama konfiguracja systemu i przetwarzanie danych odbywa się z poziomu konsoli, powstały jednak systemy wykorzystujące tesseracta i posiadające GUI.

Oprogramowanie w pewnych okolicznościach jest wrażliwe na błędy, należy unikać rozmiaru tekstu mniejszego niż 20px, pochylonego lub obróconego, niepozytywnie na działanie wpływają również ciemne granice strony ponieważ mogą one zostać rozpoznane i dopasowane jako znaki. Mimo to silnik Tesseracta uznawany jest za jeden z najlepiej działających wśród

dostępnych systemów OCR.

2.4 Środowisko programistyczne

Projekt powstał w technologii Java. Utrzymująca się popularność tej technologii na pierwszym miejscu (która wciąż rośnie)² wśród języków programowania ułatwi dalszy rozwój i wsparcie aplikacji.

W wersji 8, wprowadzone zostały znaczące kamienie milowe takie jak: wyrażenia lambda oraz strumienie. Zostały one wykorzystane w kodzie źródłowym aplikacji, przez co znaczco przyśpieszyły działanie, wykonywanie poszczególnych metod i podniosły czytelność, zwięzłość kodu.

Wykorzystanym IDE (*ang. integrated development environment*) było Eclipse. Po utworzeniu projektu Maven, do pliku pom.xml dodano zależności umożliwiające wykorzystanie funkcjonalności Tesseract oraz zależności umożliwiające pracę z frameworkm DL4J. Ponadto do projektu została dołączona biblioteka OpenCv jako biblioteka użytkownika. Wykorzystanie technologii Maven w znaczący sposób ułatwiło zarządzanie zależnościami i wykorzystanymi bibliotekami w projekcie.

²<https://www.tiobe.com/tiobe-index>, dostęp 2018.07.22

3 Część praktyczna

Ten rozdział pracy dyplomowej traktuje o szczegółach implementacyjnych zbudowanego systemu. Poszczególne podrozdziały opisują kolejne algorytmy przez jakie przechodzi plik pdf lub graficzny w celu uzyskania końcowego efektu – pozyskania tekstu z obrazu. Znacząca część algorytmów została przedstawiona w postaci schematów blokowych (zawierających również fragmenty kodu napisanego w języku Java), odzwierciedlając one przede wszystkim przepływ sterowania i poszczególne operacje. W rozdziale tym znajdziemy również listingi, zostały one wykorzystane w miejscach gdzie wykorzystanie schematów blokowych ze względu na znaczącą liczbę operacji stałoby się nieczytelne. Wszystkie opisane algorytmy zostały scalone i zawarte w aplikacji okienkowej tak, aby użytkownik widział jedynie czy system wykonuje jakąś pracę i jej efekt końcowy.

3.1 Budowa aplikacji

Algorytmy przetwarzające obrazy zostały umieszczone w odpowiednich klasach, ułatwiających posługiwanie się nimi. Rozwiążanie to wychodzi na przeciw podstawowym zasadą obiektowości takimi jak: modularyzacja, hermentyzacja, abstrakcja. W aplikacji można wyróżnić następujące funkcjonalności i odpowiadające im klasy:

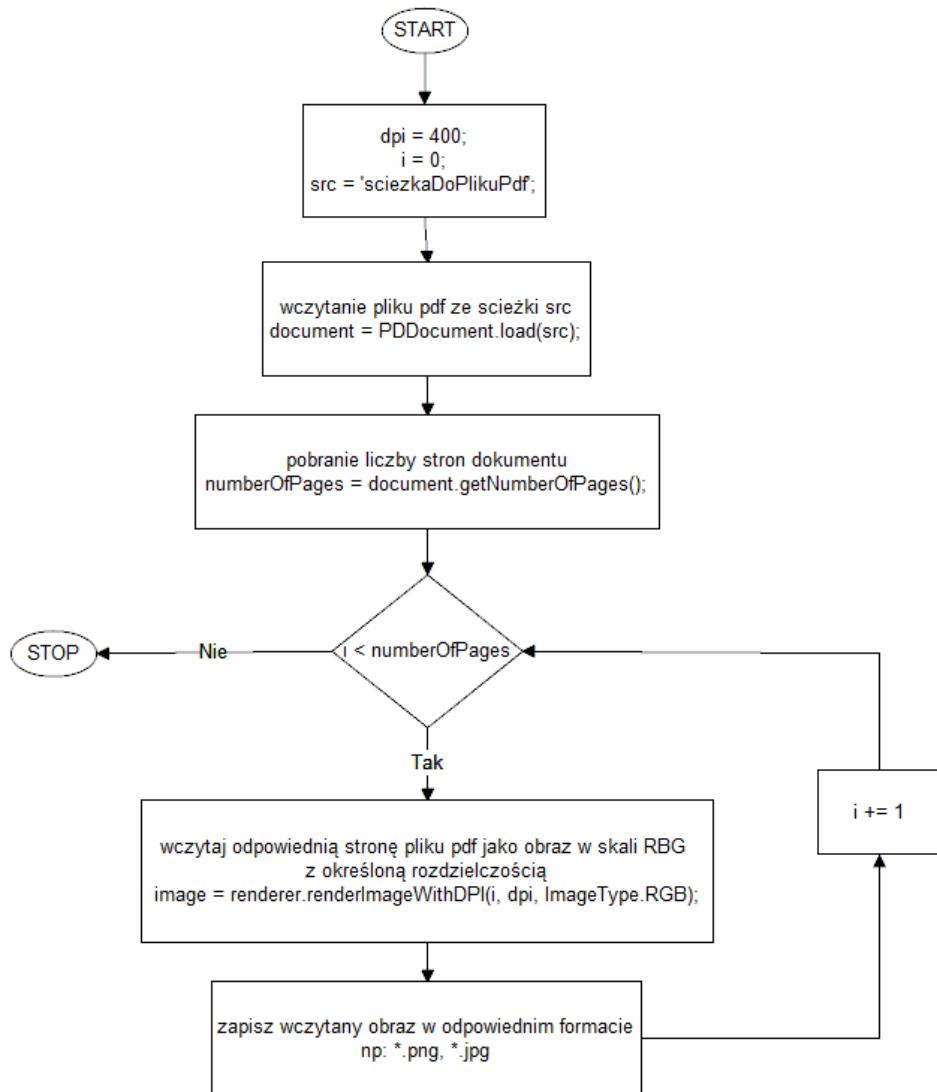
1. ekstrakcja stron z pliku pdf do obrazów (jedna strona pliku pdf odpowiada jednemu obrazowi) – klasa GetImgFromPdf;
2. polepszenie jakości obrazu – klasa ImproveImg;
3. usunięcie z obrazu obszarów które nie należą do stron – klasa CutBlackArea;
4. podział obrazu na dwie strony – klasa DivideToPage;
5. przygotowanie obrazu do analizy poprzez budowę zbiorów linii – klasa PrepareImgToAnalyze;
6. wyprostowanie obrazu – klasa ToStraightenUp;
7. zasadnicze przetwarzanie obrazu, detekcja obszarów zawierających znaki muzyczne i teksty – klasa MajorProcessing;
8. wykrycie tekstu – klasa DetectText;
9. rozpoznanie oraz wycięcie konturów obrazu które tworzą litery, znaki interpunkcyjne, etc. – klasa DetectLetter;
10. model, konfiguracja, uczenie się konwolucyjnych sieci neuronowych, odpowiednio klasyfikującej litery, linie czy jest linią tekstu czy nut – klasa LetterClassifier i klasa MusicSheetClassifier;
11. wykorzystanie zserializowanych konwolucyjnych sieci neuronowych w celu przetworzenia danych wejściowych i zwróceniu wyniku, odpowiednio tekstu, wartości logicznej –

klasa ReadLetterClassifier i klasa MusicSheetClassifierImg;

Szczegóły implementacyjne powyższych funkcjonalności zostały dokładniej opisane w kolejnych podrozdziałach. Wyjaśnienia oprócz schematów blokowych zawierają również listingi kodu w języku Java, który został podany w celu zwięzlejszego przedstawienia działania algorytmu.

3.1.1 Ekstrakcja stron z plików pdf

Przy pomocy biblioteki Apache PDFBox dołączonej do projektu jako plik z rozszerzeniem jar, z plików pdf można pobrać strony konwertując je do kolorowych plików graficznych z określoną rozdzielczością – Rysunek 5. Metody które wykonują powyższe algorytmy znajdują się w klasie GetImgFromPdf.



Rysunek 5: Algorytm pobierania obrazów z plików pdf

3.1.2 Polepszenie jakości obrazów

Algorytmy polepszające jakość obrazu (Rysunek 6) zostały zaimplementowane w klasie ImproveImg. Redukują szумy, dzięki wykorzystaniu metod rozmycia i progowania udostępnionych przez bibliotekę OpenCv, które pracują na obrazie wczytanym w skali szarości.

Operacja rozmycia nosi również nazwę wygładzania, przyczyną wykonania zabiegu rozmycia na obrazie mogą być różne, w znaczącej większości przypadków powodem jest chęć usunięcia występujących szumów. Do wyboru (w bibliotece OpenCv) mamy kilka funkcji, posługujących się różnymi rodzajami filtrów dolnoprzepustowych, a więc wykorzystującymi różne jądra, będące de facto współczynnikami tych filtrów. Algorytm wykorzystuje funkcję: *Imgproc.medianBlur(src, dst, ksize)*; gdzie *ksize* jest rozmiarem jądra, które jest nieliniowe oraz:

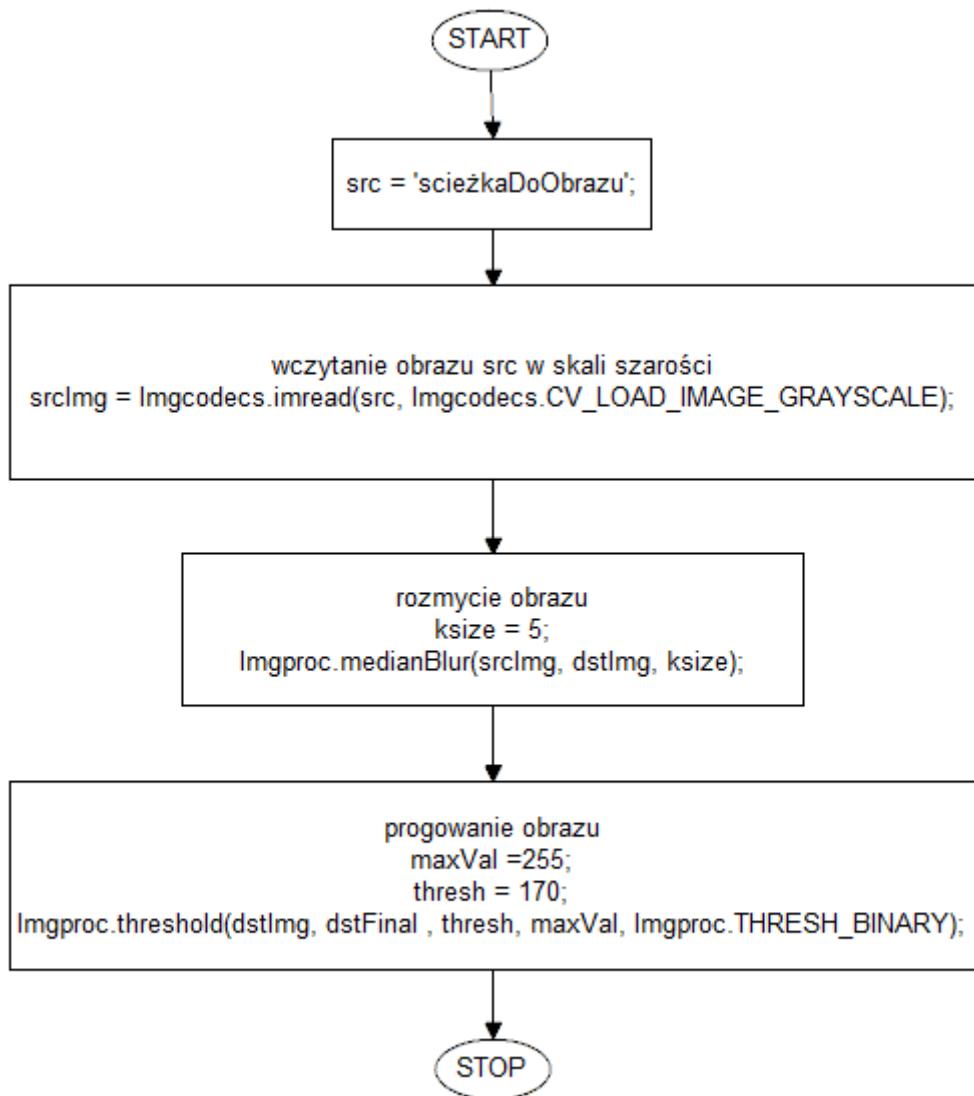
$$ksize \in \mathbb{N} \wedge ksize \in \mathbb{Z} \wedge ksize \pmod{2} = 1$$

Filtr ten wartości poszczególnych pikseli zmienia na medianę, która zostaje obliczona uwzględniając wartości pikseli tworzących określony prostokąt w którego centrum jest zmieniany piksel.

Jedną z najprostszych metod segmentacji obrazu jest operacja progowania. Biblioteka udostępnia dwa warianty progowań: proste i adaptacyjne. Wśród progowań prostych możemy wybrać jedną z następujących możliwości (*thresholdType*): BINARY, BINARY_INV, TRUNC, TOZERO, TOZERO_INV. Algorytm wykorzystuje proste progowanie binarne (BINARY), które wartość każdego z pikseli obrazu (*src*) porównuje z zadanym progiem (*threshold*) i jeśli jego wartość jest od niego większa to temu pikselowi przypisywana jest wartość *maxValue*, następnie obraz zapisywany jest w nowej macierzy (*dst*).

*Imgproc.threshold(src, dst, threshold, maxValue, thresholdType/*Imgproc.THRESH_BINARY*/);*

$$dst(x,y) = \begin{cases} maxValue & \text{dla } src(x,y) > threshold \\ 0 & \text{dla } src(x,y) \leqslant threshold \end{cases}$$



Rysunek 6: Algorytm polepszenia jakości obrazów

GLORIA
Choral gregorianiski

Glo - ri - a in ex - cel - sis De - o. Et in ter - ra pax ho - mi - ni - bus
 bo - nae vo - lun - ta - tis. Lau - da - mus te. Be - ne - di - ci - mus te.
 A - do - ra - mus te. Glo - ri - fi - ca - mus te. Gra - ti - as a - gi - mus ti - bi
 pro - pter ma - gnam glo - ri - am tu - am. Do - mi - ne De - us Rex cae - le - stis,
 De - us Pa - ter o - mni - po - tens. Do - mi - ne Fi - li u - ni - ge - ni - te,

- 2 -

Je - su Chri - ste Do - mi - ne De - us, A - gnu - s De - i, Fi - li - us Pa - tri - sis.
 Qui tol - lis pec - ca - ta mun - - - di, mi - se - re - re no - bis.
 Qui tol - lis pec - ca - ta mun - di, sus - ci - pe de - pre - ca - ti - o - nem no - stram.
 Qui se - des ad dex - te - ram Pa - tri - sis, mi - se - re - re no - bis. Quo - ni - am tu so - lis san - ctus.
 Tus - o - lus Do - mi - nus. Tus - o - lus Al - ti - si - mu - sis, Je - su Christe. Cum San - cto Spi - ri - tu,
 in glo - ri - a De - i Pa - - - tri - sis. A - - - - men.

- 3 -

Rysunek 7: Obraz przed zastosowaniem algorytmu

GLORIA
Choral gregorianiski

Glo - ri - a in ex - cel - sis De - o. Et in ter - ra pax ho - mi - ni - bus
 bo - nae vo - lun - ta - tis. Lau - da - mus te. Be - ne - di - ci - mus te.
 A - do - ra - mus te. Glo - ri - fi - ca - mus te. Gra - ti - as a - gi - mus ti - bi
 pro - pter ma - gnam glo - ri - am tu - am. Do - mi - ne De - us Rex cae - le - stis,
 De - us Pa - ter o - mni - po - tens. Do - mi - ne Fi - li u - ni - ge - ni - te,

- 2 -

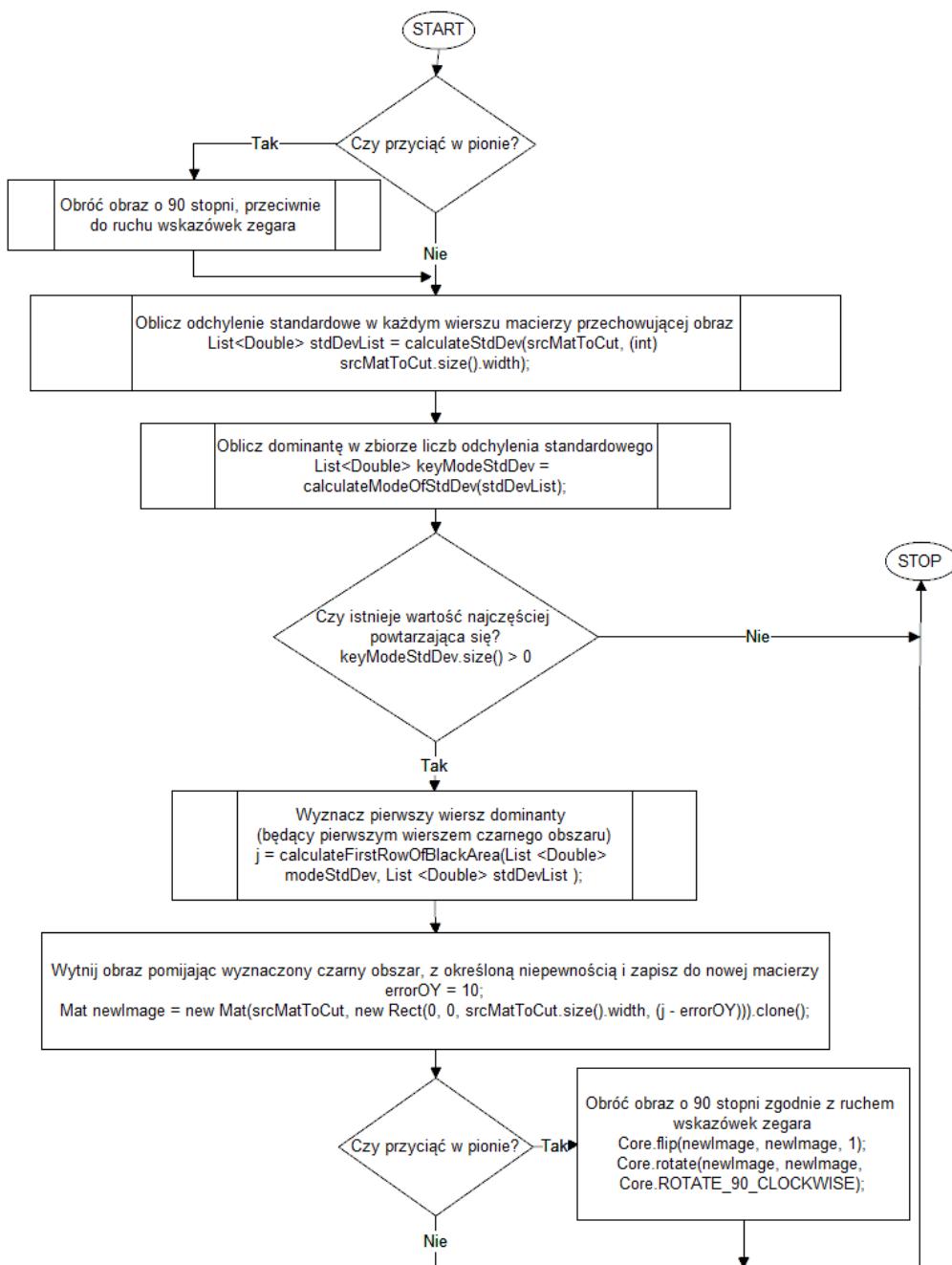
Je - su Chri - ste Do - mi - ne De - us, A - gnu - s De - i, Fi - li - us Pa - tri - sis.
 Qui tol - lis pec - ca - ta mun - - - di, mi - se - re - re no - bis.
 Qui tol - lis pec - ca - ta mun - di, sus - ci - pe de - pre - ca - ti - o - nem no - stram.
 Qui se - des ad dex - te - ram Pa - tri - sis, mi - se - re - re no - bis. Quo - ni - am tu so - lis san - ctus.
 Tus - o - lus Do - mi - nus. Tus - o - lus Al - ti - si - mu - sis, Je - su Christe. Cum San - cto Spi - ri - tu,
 in glo - ri - a De - i Pa - - - tri - sis. A - - - - men.

- 3 -

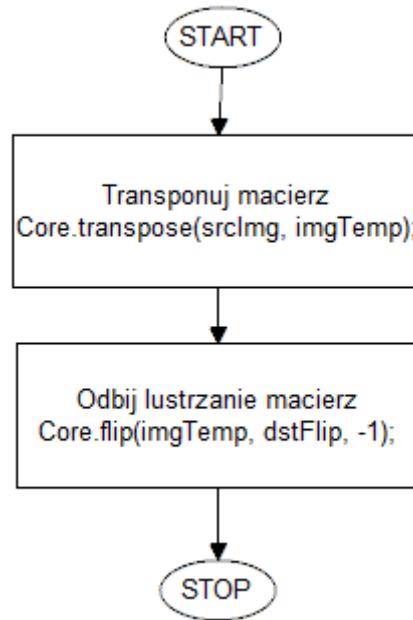
Rysunek 8: Obraz po zastosowaniu algorytmu

3.1.3 Eliminacja nadmiarowych (czarnych) obszarów skanu

Algorytm wykrywający i usuwający czarne obszary skanu (uzupełnienie skanu do określonego formatu np. A4) (Rysunek 9) zostały zaimplementowane w klasie CutBlackArea. Algorytm wykorzystuje do tego wartości odchylenia standardowego w poziomych liniach obrazu (Listing 2) i wartości dominanty (na jej podstawie obliczając początek czarnego obszaru (Listing 3)). Przy założeniu, że powierzchnie te znajdują się po lewej stronie i na dole obrazu. Algorytm umożliwia wycięcie jednego (dolnego) lub obu obszarów.



Rysunek 9: Algorytm usuwający czarne obszary z obrazu



Rysunek 10: Obrót o 90 stopni przeciwnie do ruchu wskazówek zegara

```

1 public List<Double> calculateStdDev(Mat srcImg, int width) {
2     List<Double> stdDevList = new ArrayList<Double>();
3     for (int j = 0; j < srcImg.rows(); j += 1) {
4         Mat temp = new Mat(srcImg, new Rect(0, j, width, 1)).clone();
5         MatOfDouble mean = new MatOfDouble();
6         MatOfDouble stddev = new MatOfDouble();
7         Core.meanStdDev(temp, mean, stddev);
8         stdDevList.add(stddev.get(0, 0)[0]);
9         System.gc();
10    }
11    return stdDevList;
12 }
  
```

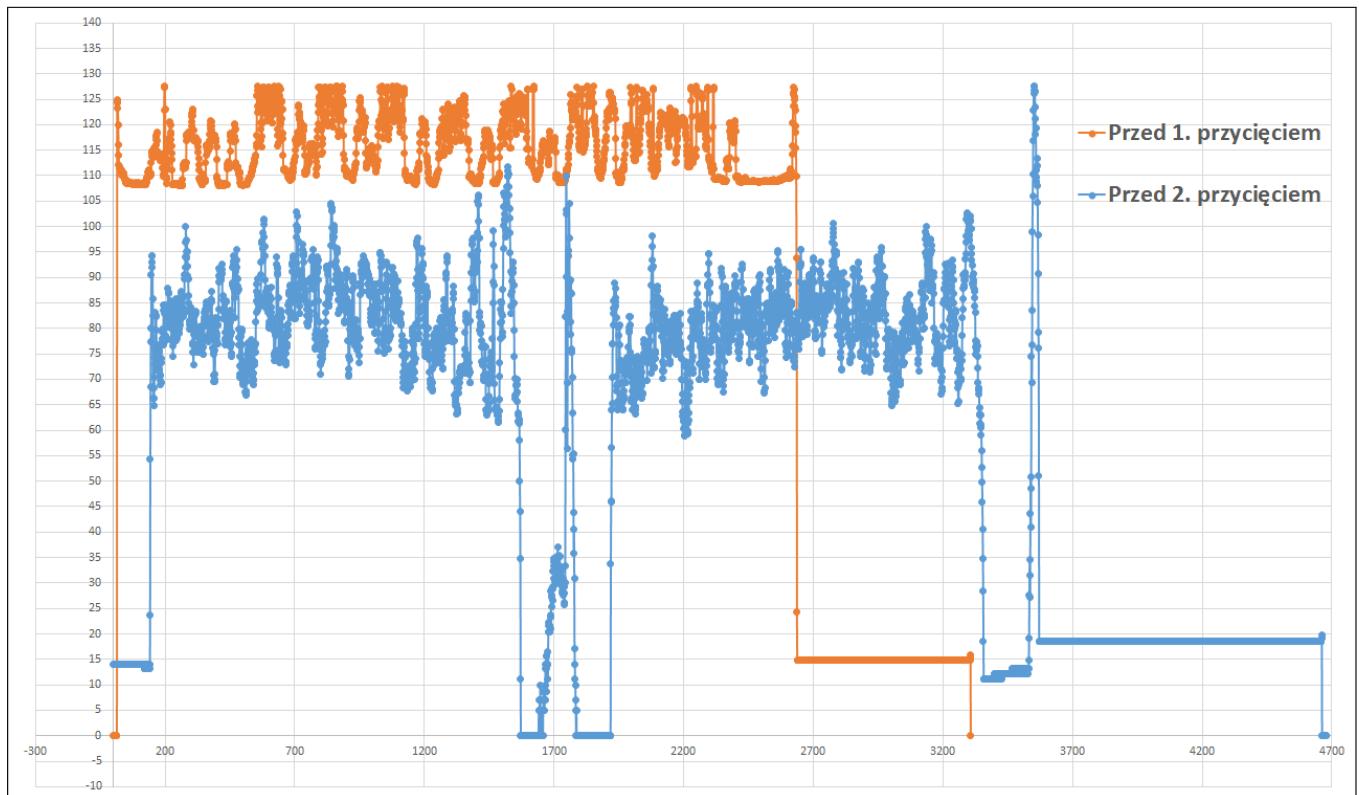
Listing 2: Metoda obliczająca odchylenie standardowe każdej linii

```

1 public int calculateFirstRowOfBlackArea(List<Double> keyModeStdDev, List<Double> stdDevList) {
2     int k = 0;
3     if (!keyModeStdDev.isEmpty()) {
4         for (k = stdDevList.size() - 1; k >= 0; k -= 1) {
5             if (stdDevList.get(k).equals(keyModeStdDev.get(0))) {
6                 break;
7             }
8         }
9         for (int a = k; a >= 0; a -= 1) {
10            if (!stdDevList.get(a).equals(keyModeStdDev.get(0))) {
11                k = a;
12                break;
13            }
14        }
15    }
16    return k;
17 }

```

Listing 3: Metoda obliczająca początek czarnego obszaru



Rysunek 11: Odchylenie standardowe w funkcji wierszy macierzy

Na podstawie wykresu (Rysunek 11) można zauważyć, że wiersze obrazu (w skali szarości) o małej zmienności wartości pikseli, będą dawały niewielkie wartości odchylenia standardowego. Jeśli nie równa się ono zeru, ale jest odpowiednio małe można założyć, że badany wiersz ma jedną barwę, a wartość ta wynika z drobnych zakłóceń i niedoskonałości obrazu.

Do obliczenia odchylenia standardowego została użyta funkcja udostępniona w bibli-

tece OpenCv:*Core.meanStdDev(Mat src, MatOfDouble mean, MatOfDouble stdDev)*. Oblicza ona średnią i odchylenie standardowe macierzy podanej jako parametr src (w algorytmie jest to jeden wiersz obrazu), które zależy od wszystkich kanałów macierzy. Tutaj jest to jeden kanał – obraz przed obliczeniami został przetransformowany do skali szarości według poniższego algorytmu:

```

1 public void toGrayScale(Mat srcImg) {
2     for (int y = 0; y < srcImg.rows(); y += 1) {
3         for (int x = 0; x < srcImg.cols(); x += 1) {
4             double[] rgb[] = srcImg.get(y, x);
5             double greyScale = (rgb[0] + rgb[1] + rgb[2]) / 3;
6             srcImg.put(y, x, greyScale, greyScale, greyScale);
7         }
8     }
9 }
```

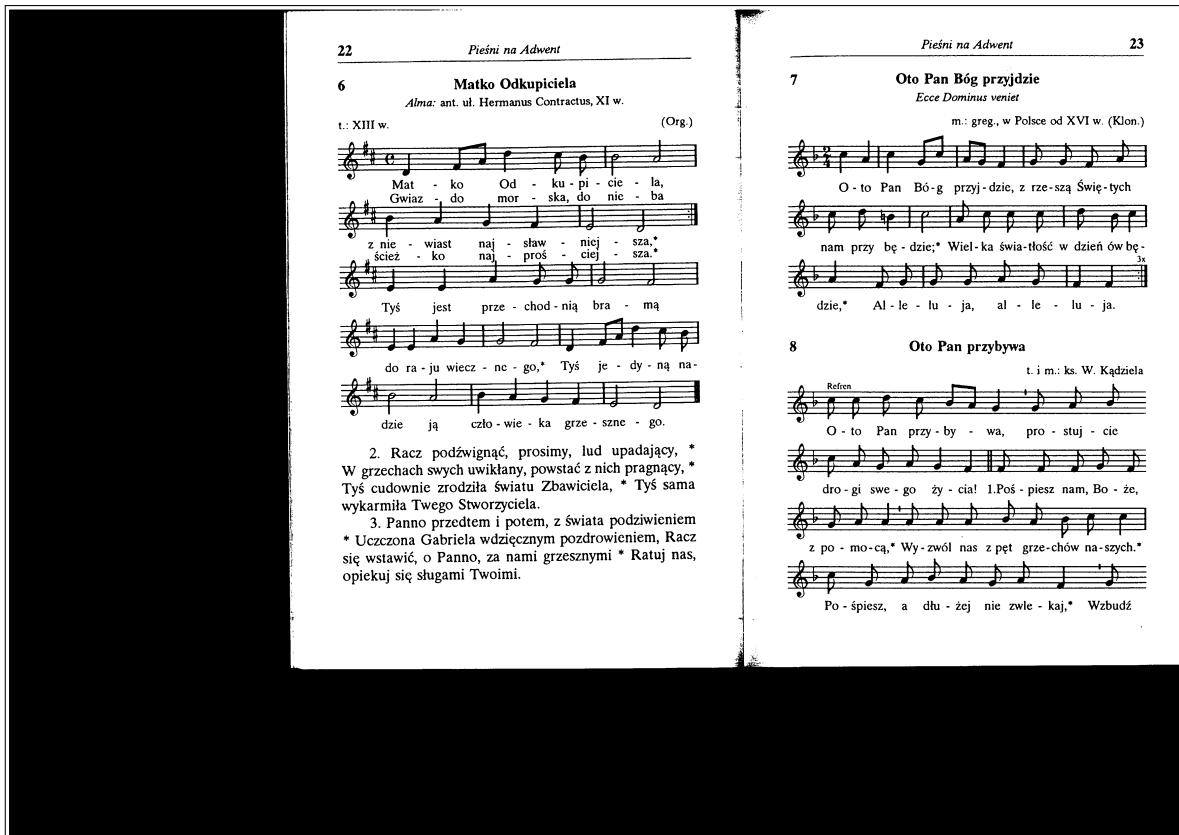
Listing 4: Metoda transformująca obraz barwny do skali szarości

Za obliczenie wartości najczęściej występującej odchylenia standardowego odpowiada funkcja: *public List<Double> calculateModeOfStdDev(List<Double> stdDevList)* (Listing 5) wykorzystująca nowe rozwiązania wprowadzone w Javie wersji 8, a więc strumienie i wyrażenia lambda, co znacząco przyśpiesza czas obliczeń oraz przejrzystość napisanego kodu.

```

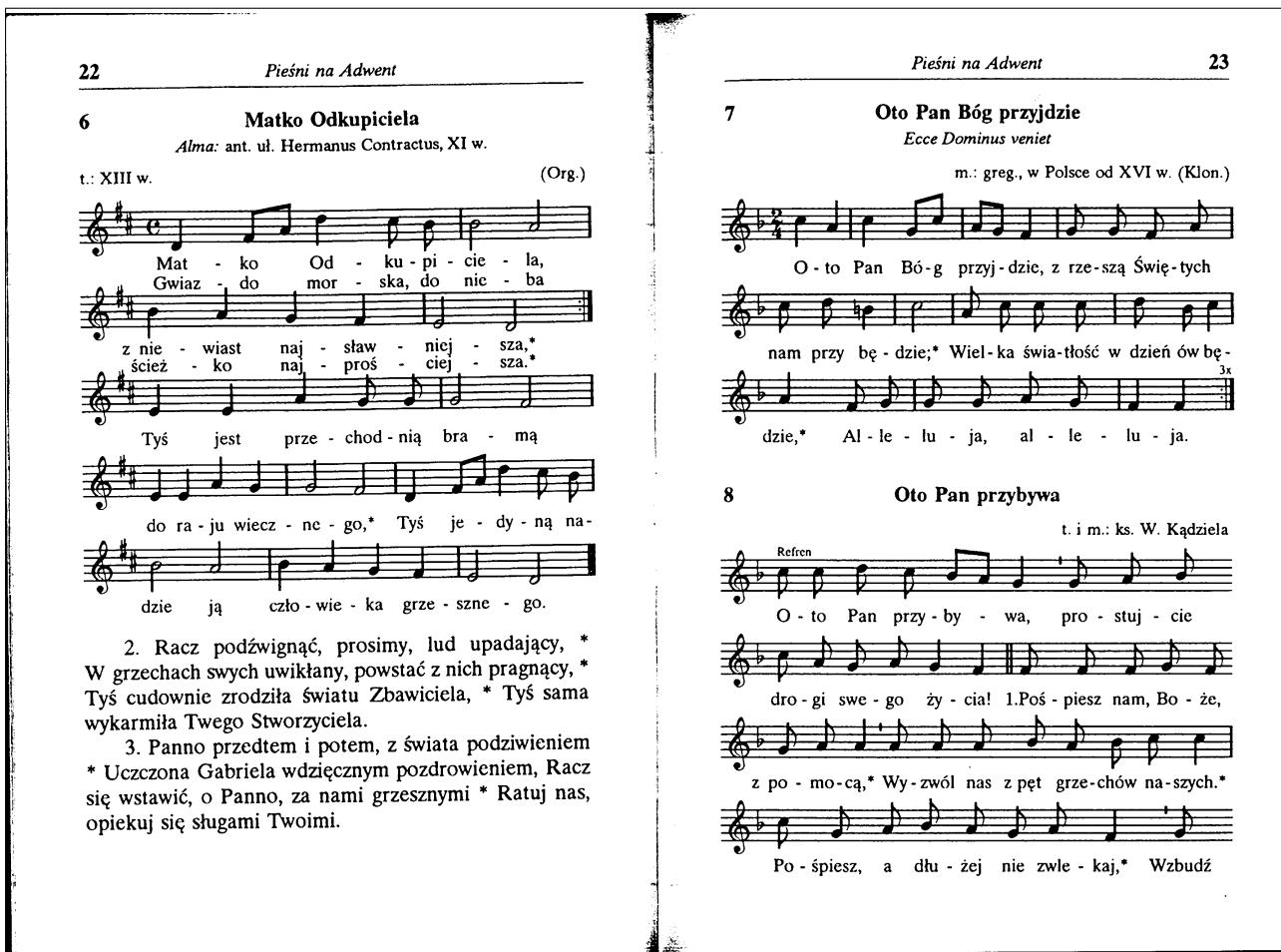
1 public List<Double> calculateModeOfStdDev(List<Double> stdDevList) {
2     Map<Double, Integer> calculateRepresentationOfValue = new HashMap<Double, Integer>();
3     for (double d : stdDevList) {
4         calculateRepresentationOfValue.compute(d, (k, v) -> v == null ? 1 : v + 1);
5     }
6     Map<Double, Integer> modeSpecificScope = calculateRepresentationOfValue.entrySet()
7         .stream()
8         .filter(map -> map.getKey() < maxStdDev)
9         .collect(Collectors.toMap(p -> p.getKey(), p -> p.getValue()));
10
11    List<Double> keyModeStdDev = new ArrayList<Double>();
12    if (modeSpecificScope.size() > 0 && modeSpecificScope != null) {
13        long modeStdDev = modeSpecificScope.values().stream()
14            .max(Comparator.naturalOrder()).get();
15        keyModeStdDev = modeSpecificScope.entrySet().stream()
16            .filter(e -> e.getValue() == modeStdDev)
17            .map(Map.Entry::getKey).collect(Collectors.toList());
18    }
19    return keyModeStdDev;
20 }
```

Listing 5: Metoda obliczająca dominantę odchylenia standardowego



Rysunek 12: Obraz przed usunięciem nadmiarowych obszarów

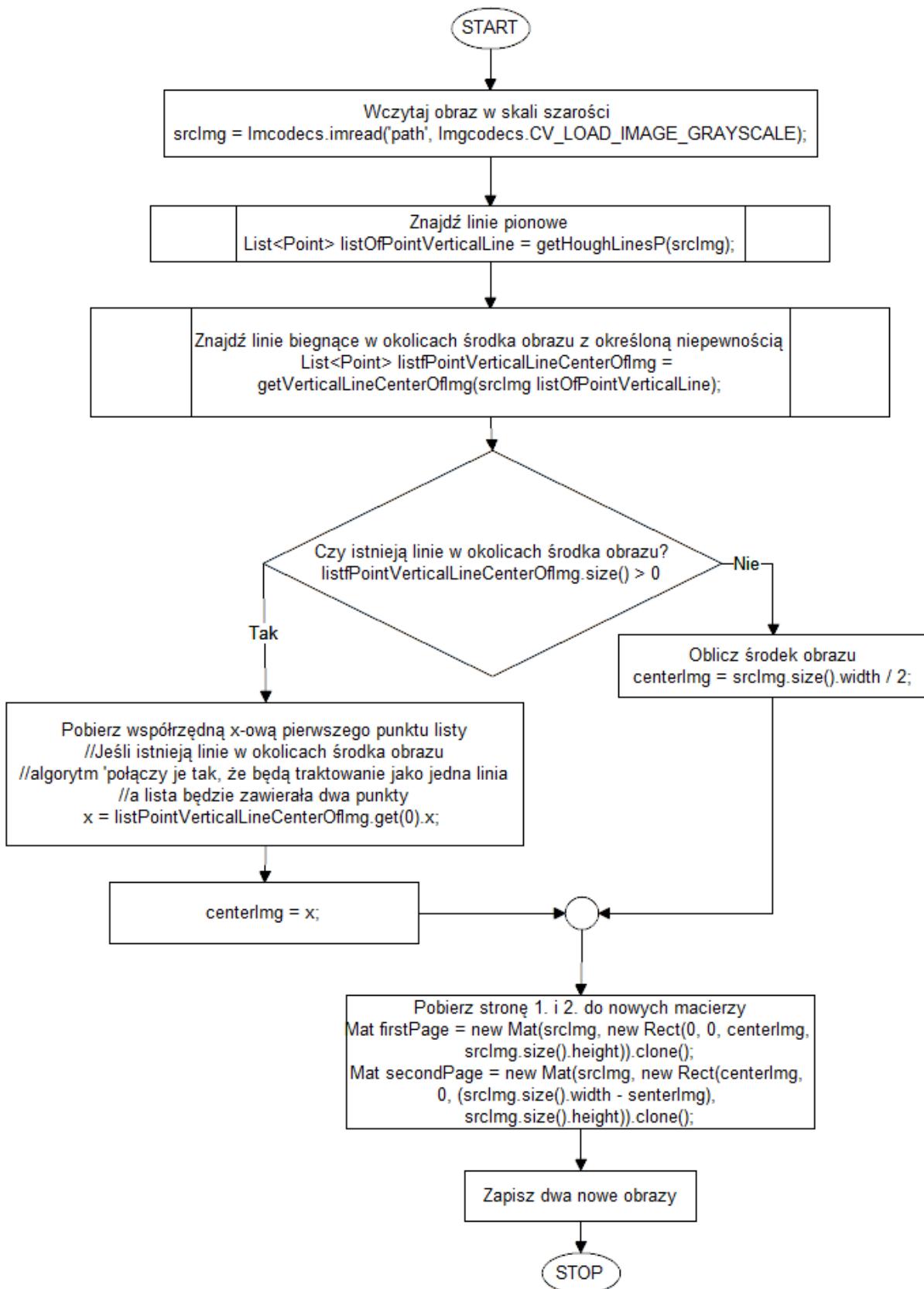
Rysunek 13: Obraz po przycięciu obszaru znajdującego się na dole



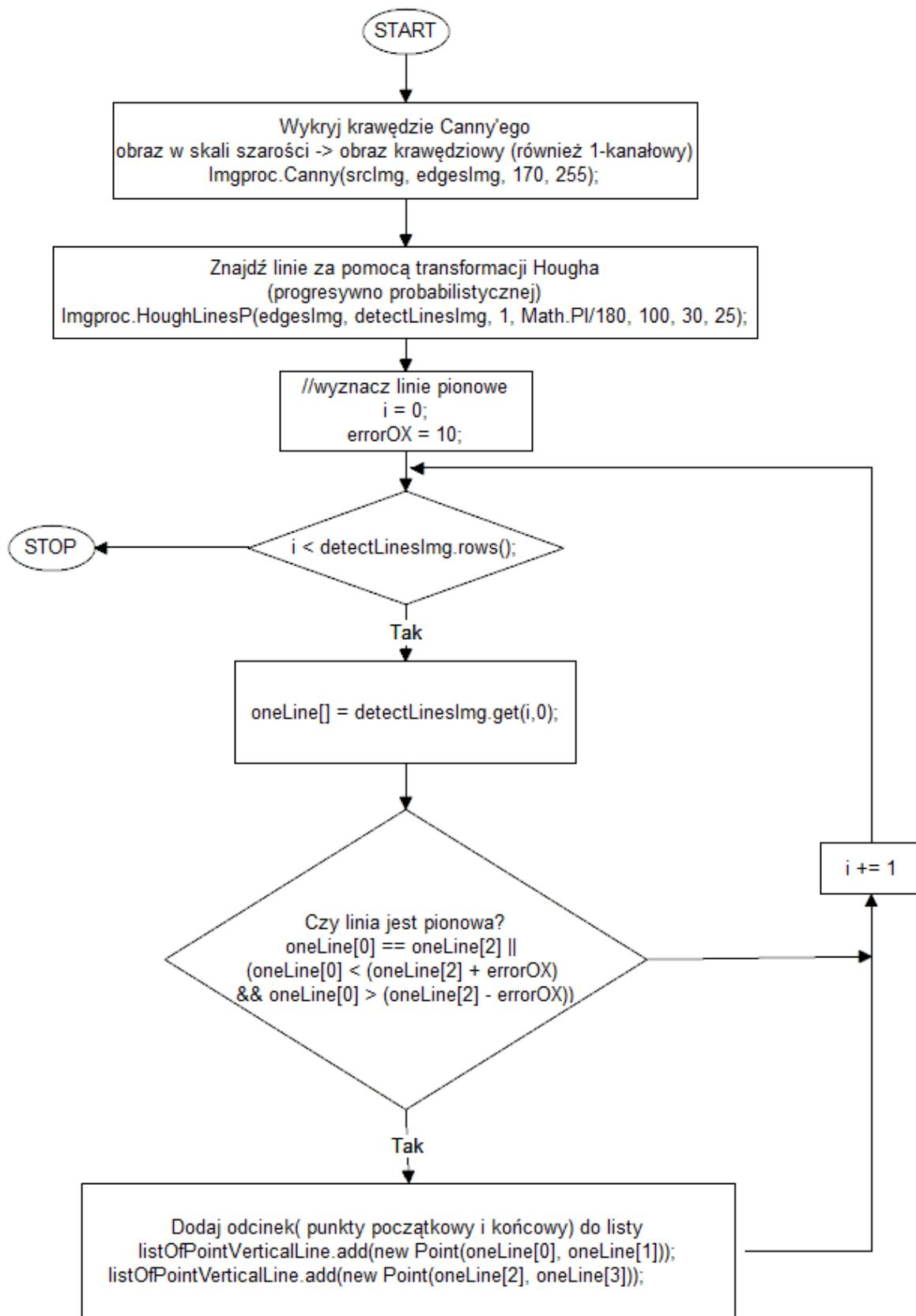
Rysunek 14: Obraz po przycięciu obszaru znajdującego się po lewej stronie

3.1.4 Podział skanu na dwie strony

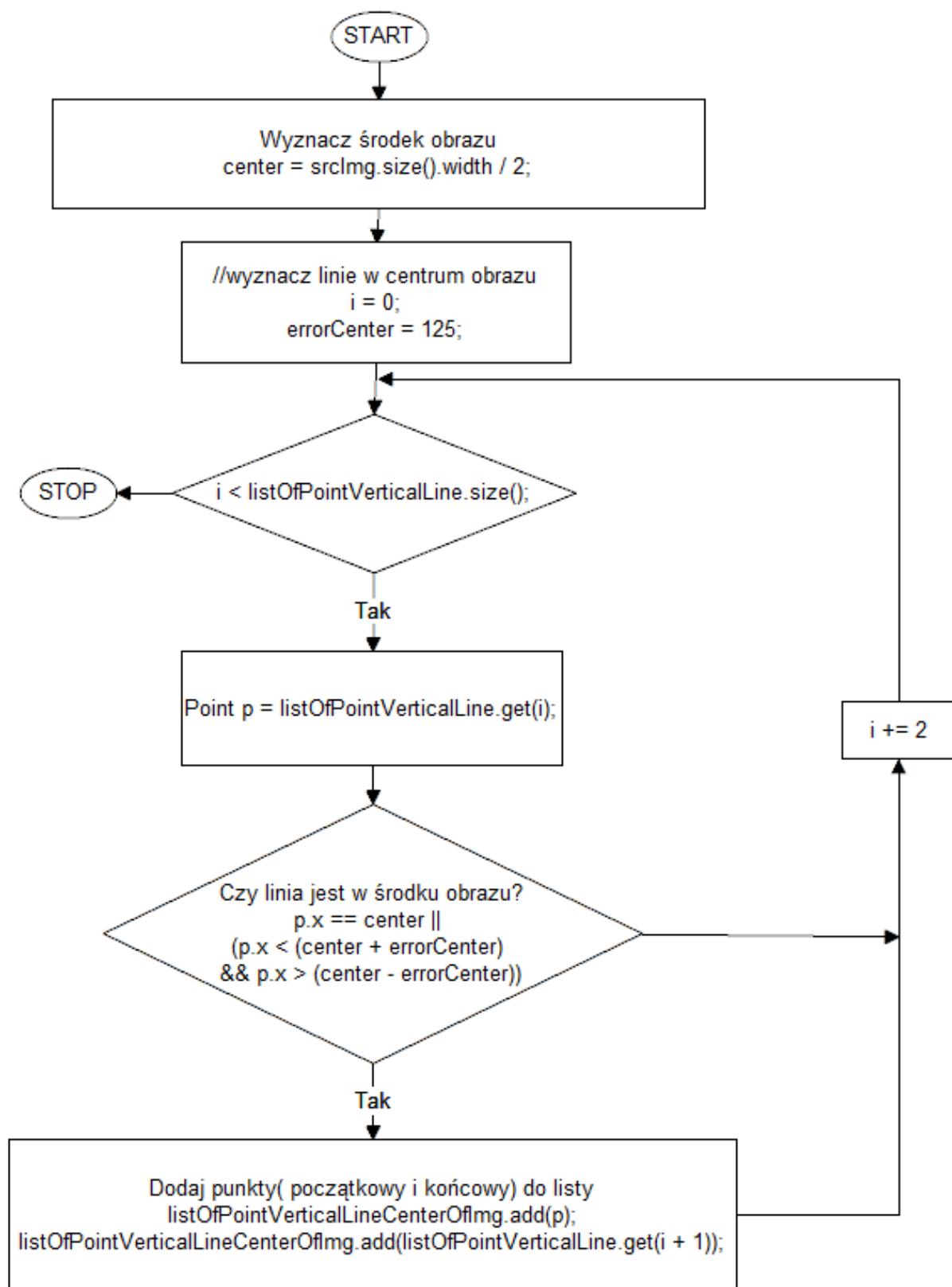
Algorytmy dzielące skan zawierający dwie strony śpiewnika na dwa osobne obraz, z których każdy stanowi jedną stronę zostały zaimplementowane w klasie DivideToPage (Rysunek 15). Algorytm wykorzystuje do tego niedoskonałość skanu, który w książkach szczególnie tych o znacznej liczbie stron odciska na ich łączeniu charakterystyczny czarny lub zaciemniony obszar pionowy. Jeżeli wykrycie tego obszaru nie jest możliwe (niewielka liczba przypadków) przyjęto założenie, że na cały obszar obrazu przypadają dwie strony śpiewnika i następnie zostaje on podzielony na dwie części gdzie granicą jest środkowa kolumna macierzy przechowującej obraz.



Rysunek 15: Algorytm dzielący obraz na dwie części, strony



Rysunek 16: Algorytm znajdujący linie pionowe



Rysunek 17: Algorytm znajdujący linie pionowe w okolicach środka obrazu

Metoda podziału strony na dwie części korzysta z dwóch zasadniczych funkcji zaimplementowanych w bibliotece OpenCv – detektora krawędzi Canny’ego oraz transformacji (liniowej) Hougha. Algorytmy te ze względu na osiąganie dobrych efektów są powszechnie wykorzystywane z zagadnieniu przetwarzania obrazów.

W 1986 został zaproponowany detektor krawędzi Canny’ego. Metoda ta jest udoskonalonym filtrem Laplace’a. Spełnia następujące założenia:

- wykrycie maksymalnej liczby krawędzi rzeczywistych, a więc minimalizacją wykrywania krawędzi fałszywych, jak i nie pomijanie krawędzi rzeczywistych;
- wykryte krawędzie zlokalizowane są jak najbliżej krawędzi rzeczywistych;
- pojedynczej odpowiedzi – generowanie krawędzi o szerokości jednego piksela, która jest rzeczywistą krawędzią, a nie szumem.

Algorytm ten składa się z kilku kroków. Pierwszym z nich jest wygładzanie, które odbywa się przez rozmycie filtrem Gausa (konwolucja). Następnie dla każdego z pikseli obrazu wyznaczany jest gradient, do operacji tej używane są filtry gradientowe, następnie dla każdego z pikseli obrazu ustalana jest wartość i kierunek gradientu. Kolejnym krokiem jest zerowanie (eliminacja) pikseli o niemaksymalnych wartościach gradientu. Ostatnim etapem jest progowanie z histerezą. Metoda ta (progowania z histerezą) opiera się na zdefiniowaniu dwóch progów dolnego i górnego. Później przyporządkowany jest gradient piksela w zależności od zakwalifikowanego w przedziale ograniczonym progami. Piksel może zostać odrzucony jeśli jego wartość nie przekracza dolnego progu, zaliczony do krawędzi jeśli jego wartość przekracza wartość progu górnego. Zaś jeśli wartość gradientu piksela znajduje się w przedziale ograniczonym progami to jeśli styka się z pikselem którego wartość gradientu przekracza gorny próg to zaliczany jest do krawędzi w przeciwnym wypadku zostaje odrzucony. Zaleca się następujące proporcje między progami 2:1, 3:1.

Biblioteka OpenCv dostarcza kilka możliwości (kilka funkcji) detekcji kształtów na obrazie, jedną z nich jest transformacja liniowa Hougha. U podstaw tej metody leży obserwacja, że wszelkie punkty obrazu binarnego mogą być częścią pewnego zbioru prostych, zdefiniowanych przy użyciu dwóch parametrów: kąta nachylenia prostej a i punktu w którym przecina się z osią b. Ponieważ zastosowanie takiego podejścia (płaszczyzny akumulacyjnej (a, b)) nie daje najlepszych rezultatów, ze względu na zakres kąta nachylenia będącego w przedziale $(-\infty, +\infty)$, praktyczniejsze zastosowanie znajduje metoda przedstawienia prostej wykorzystująca współrzędne biegunowe (ρ, θ). Wtedy wyznaczana linia przechodzi przez badany punkt i jest prostopadła do prostej określonej wzorem:

$$\rho = x \cos \theta + y \sin \theta$$

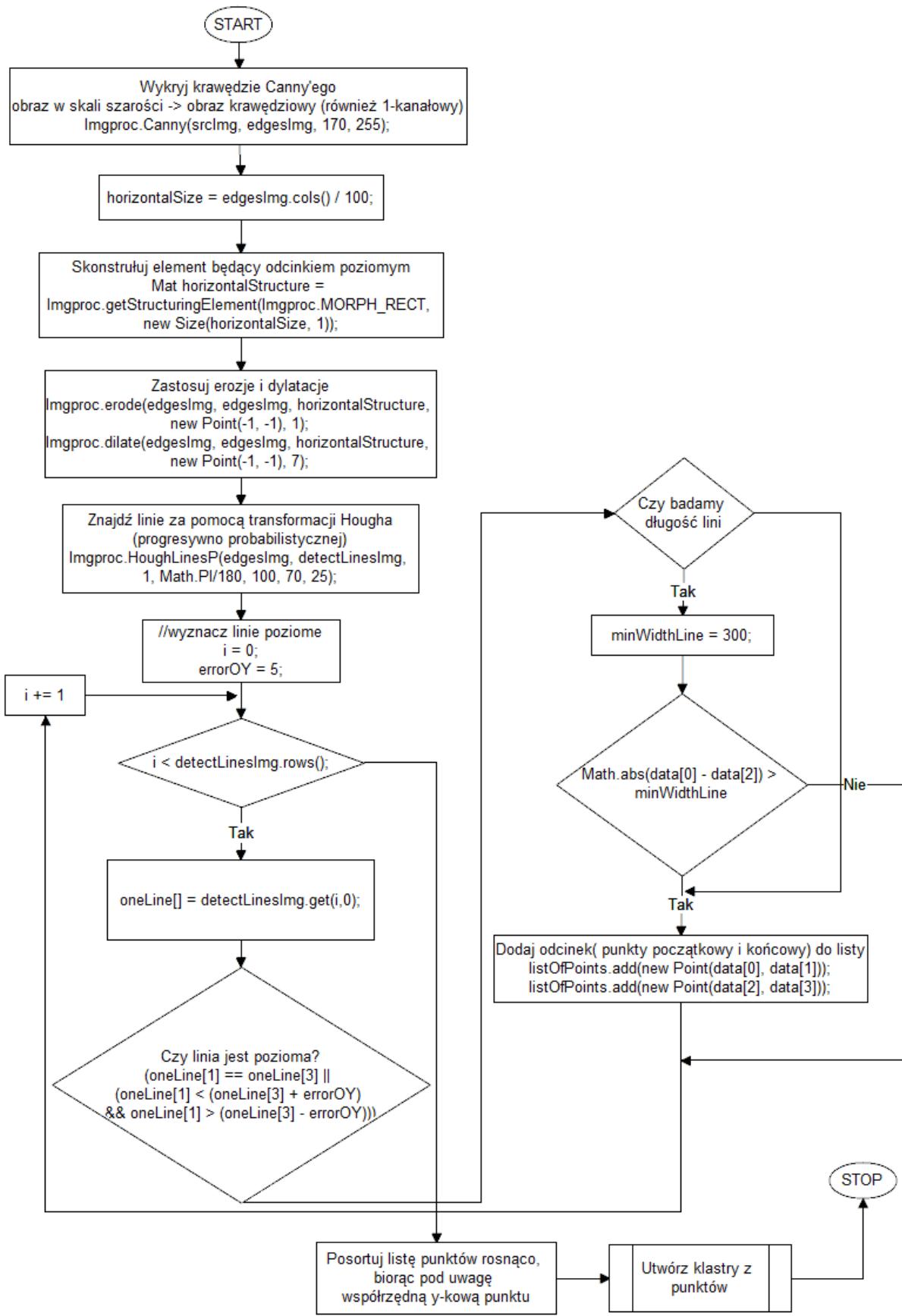
która jest wyprowadzona z tego punktu do początku układu współrzędnych.

Biblioteka OpenCv udostępnia kilka różnych implementacji transformacji liniowej Hough'a: standardową, wieloskalową, progresywno probabilistyczną. W algorytmie została wykorzystana ta trzecia, dająca najlepsze efekty biorąc pod uwagę czas obliczeń i dokładność algorytmu. Dodatkowymi zaletami użycia metody:

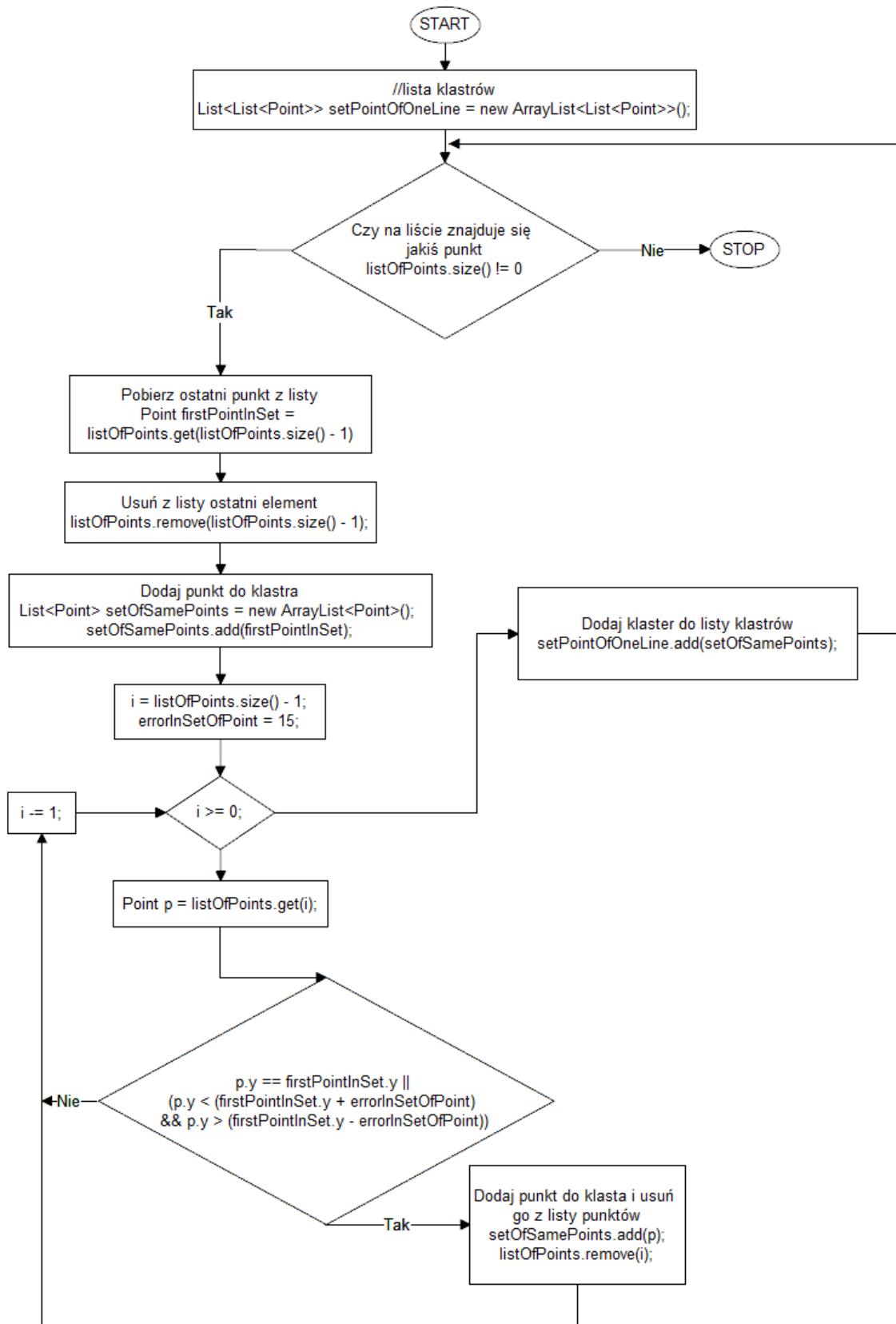
Imgproc.HoughLinesP(srcImg, lines, rho, theta, threshold, minLineLength, maxLineGap);
są argumenty *minLineLength* - minimalna długość linii, *maxLineGap* - wymagany odstęp pomiędzy współliniowymi odcinkami, również macierz wynikowa *lines* posiada cztery kolumny, które odpowiadają współrzędnym początku i końca linii (w następującej kolejności x_0, y_0, x_1, y_1).

3.1.5 Zasadnicze przygotowanie obrazu do analizy

Kluczowe algorytmy analizujące obraz zostały zaimplementowane w klasie *PrepareImageToAnalyze* mają one za zadanie pozyskać pożądane (określone przez parametry) właściwości obrazu, głównie jest to wyznaczenie zbioru linii (Rysunek 18). Wywołanie metody tej klasy *houghLines(boolean)* buduje zbiór punktów które tworzą linie. Punkty są łączone w klastry, dany klaster jest jedną linią i zwracana jest lista tych klastrów, linii (Rysunek 19). Argument logiczny decyduje czy bada się dodatkowo długość linii.



Rysunek 18: Algorytm znajdujący linie poziome



Rysunek 19: Algorytm budujący listę klastrów

Algorytm wykorzystuje zasadnicze przekształcenia morfologiczne udostępnione w bibliotece OpenCv – dylatacje i erozje, które mają szerokie zastosowanie w zagadnieniu przetwarzania obrazów.

Dylatacja jest to konwolucja obrazu i jądra, w której wszystkie piksele zostają zastąpione na lokalne maksimum każdego z pikseli, które obejmuje jądro. Jest to operacja nieliniowa, która powoduje rozrastanie się jasnego obszaru i zazwyczaj powoduje wypełnienie zagłębień przy przyjęciu założenia, że jądro jest jądem wypukłym i wypełnionym.

Erozja jest to operacja przeciwna do operacji dylatacji, a więc piksele zostają zastąpione na lokalne minimum, jest to również operacja nieliniowa, która powoduje zmniejszenie się czarnego obszaru oraz usuwa wypukłości, przy tych samych założeniach.

3.1.6 Wyprostowanie strony

Metody klasy ToStraightenUp pomagają zniwelować w pewnym stopniu wpływ ludzki na jakość przetwarzania obrazu. Umożliwiają one wyprostowanie danego zdjęcia jeśli zostało ono niewłaściwie, krzywo umieszczone na skanerze (Rysunek 20). Klasa ta wykorzystuje również metody klasy PrepareImgToAnalyze dzięki której buduje zbiór linii, z których każda ma taką podobną współrzędną y–kową początku i końca. I na podstawie średniej z różnic tej współrzędnej wykrytych linii określany jest kąt o jaki należy obrócić obraz, tak aby został on wyprostowany (Listing 6).

Algorytm wykorzystuje funkcję zaimplementowaną w bibliotece OpenCv przekształcającą obraz. Jest to funkcja przekształcenia aficznego:

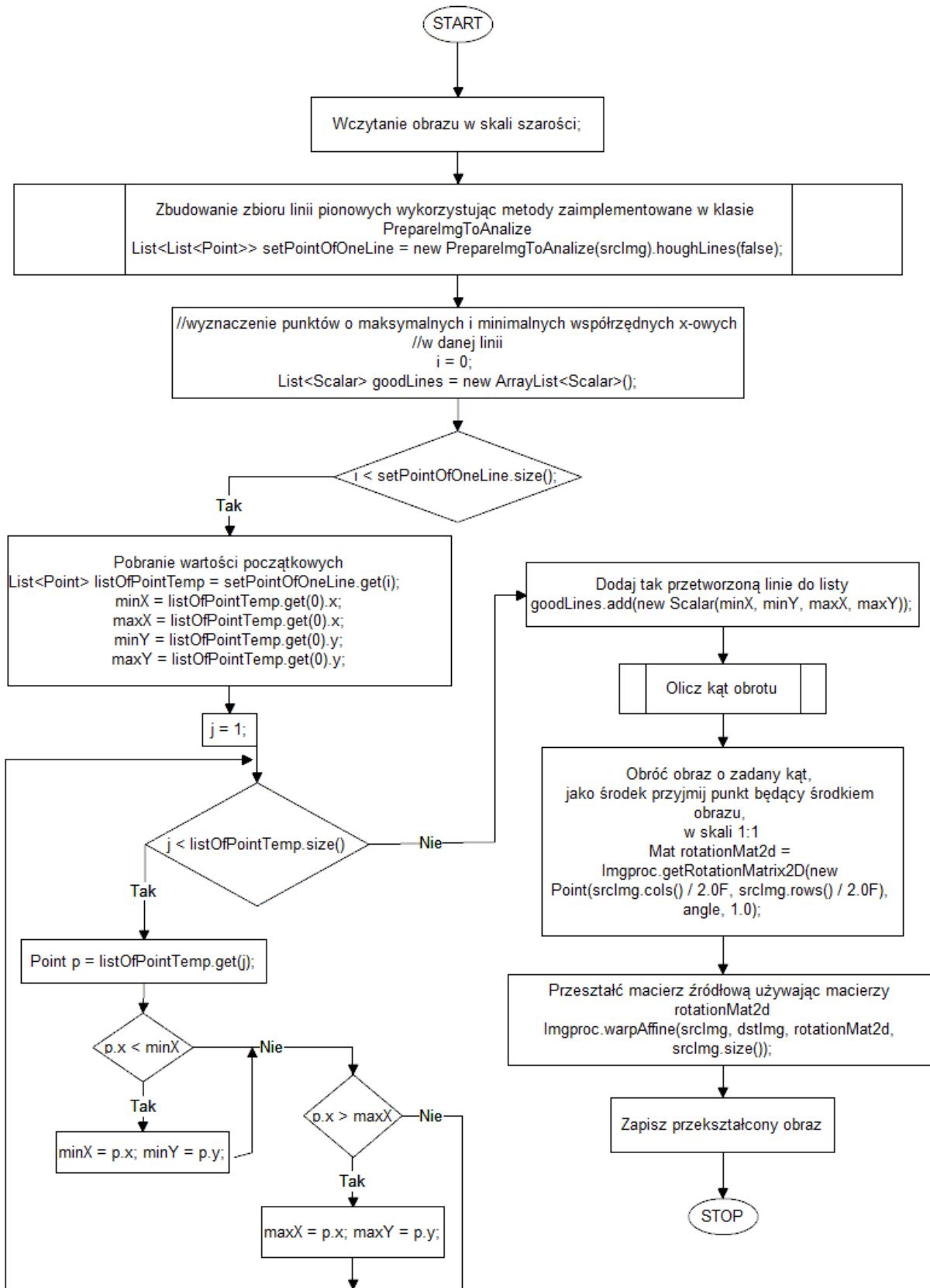
```
Imgproc.warpAffine(srcImg, dstImg, M, dsize);
```

która punkt macierzy źródłowej *srcImg* przekształca do macierzy wynikowej *dstImg* według macierzy określającej jakiego rodzaju jest to przekształcenie *M* mającej dwa wiersze i trzy kolumny według następującego wzoru:

$$dst(x, y) = src(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23}).$$

Ponieważ prawa strona równania często nie jest liczbą całkowitą, wartość odpowiedniego punktu macierzy wynikowej oblicza się stosując interpolację. Metoda ta dostarcza również innych wariantów wywołania z kolejnymi argumentami takimi jak:

- *flags* – wybór metody interpolacji, dostępne są następujące typy:
 - INTER_NEAREST – najbliższego sąsiada,
 - INTER_LINEAR – dwuliniowa,
 - INTER_AREA – przepróbkowanie obszaru pikseli,
 - INTER_CUBIC – interpolacja dwusześcianowa,
 - INTER_LANCZOS4 – interpolacja wykorzystująca metodę Lanczosa biorąca pod uwagę sąsiednie punkty w obszarze o wymiarach 8x8, można dołączyć dodatkową opcję przy użyciu operatora lub WRAP_INVERSE_MAP – przekształcenie z *dstImg* do *srcImg*,
- *borderMode* – metoda użyta do ekstrapolacji pikseli znajdujących się na zewnątrz obrazu, dostępne są następujące typy:
 - BORDER_CONSTANT – dodaje piksele o stałej wartości,
 - BORDER_WRAP – dodaje piksele przez pobranie ich z przeciwej strony,
 - BORDER_REPLICATE – dodaje piksele przez skopiowanie ich z krawędzi,
 - BORDER_REFLECT – dodaje piksele przez lustrzane odbicie,
 - BORDER_REFLECT_101 – dodaje piksele przez lustrzane odbicie przy tym nie duplikując pikseli tworzących krawędź,
 - BORDER_DEFAULT – jest to alias dla BORDER_REFLECT_101,
- *borderValue* – parametr używany w przypadku stałej granicy, wartością domyślną jest zero.



```

1 public double calculateAngle(double width) {
2     double meanRotate = 0;
3     int counterLine = 0;
4     for(Scalar p : goodLines) {
5         meanRotate += Math.abs((p.val[3] - p.val[1]));
6         counterLine += 1;
7     }
8     meanRotate /= counterLine;
9     double angleRotTan = /*-*/1 * meanRotate) / width;
10    return Math.toDegrees(Math.atan(angleRotTan));
11 }

```

Listing 6: Algorytm obliczający kąt obrotu

SPUŚCIE NAM NA ZIEMSKIE NIWY

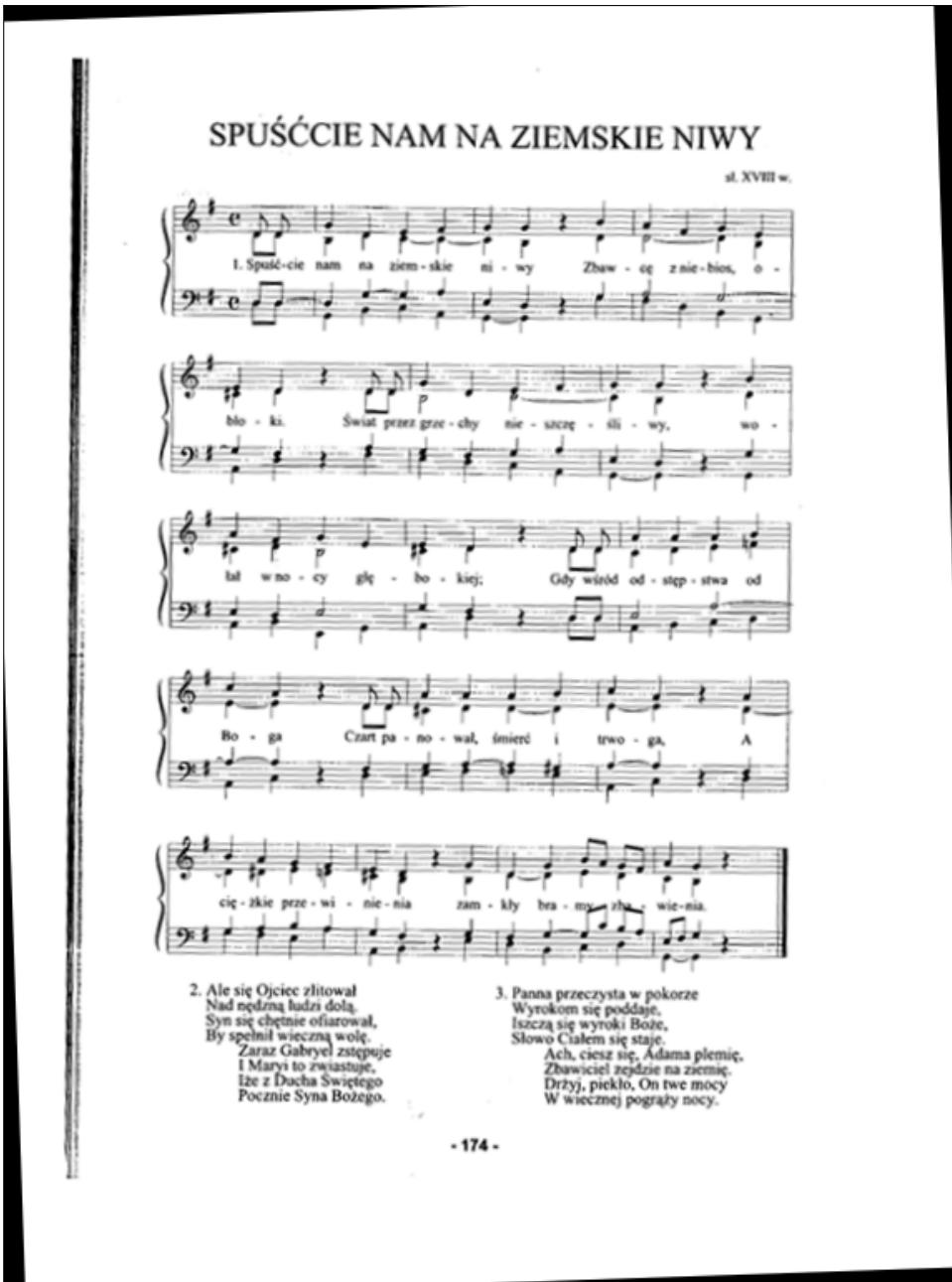
sl. XVIII w.

2. Ale się Ojciec złotował
Nad nędzną ludzią dolą.
Syn się chętnie ofiarował,
By spełnił wieczną wolę.
Zaraz Gabryel zstępue
I Maryi to zwiazuje
Ize z Ducha Świętego
Pocznie Syna Bożego.

3. Panna przeczysta w pokorze
Wyrokiem się poddaje,
Iszczą się wyroki Boże,
Słowo Ciałem się staje.
Ach, ciesz się, Adama plemię,
Zbawiciel zejdzie na ziemię.
Drży, piekło, On twe mocy
W wiecznej pograły nocy.

- 174 -

Rysunek 21: Obraz przed wyprostowaniem



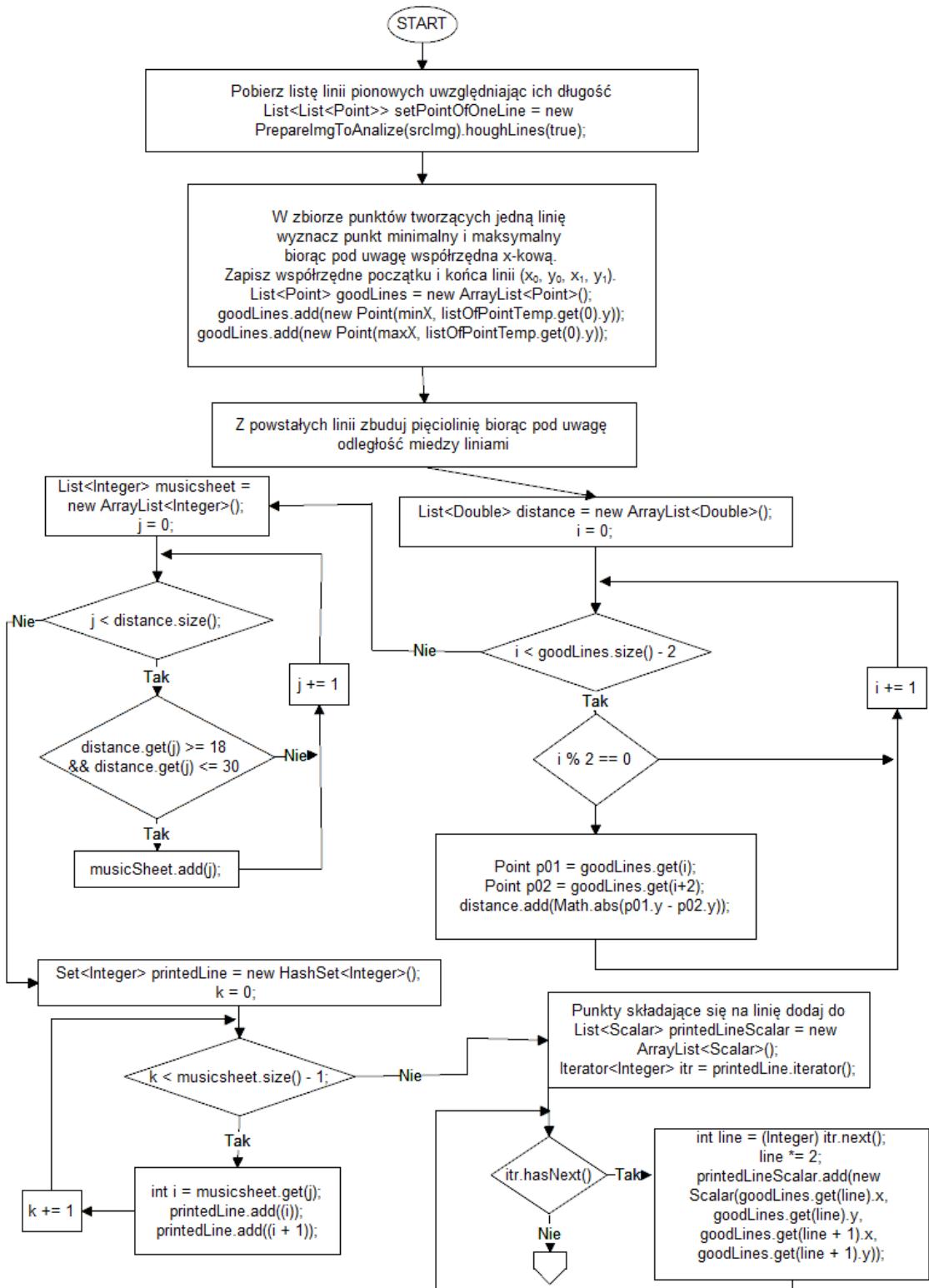
- 174 -

Rysunek 22: Obraz po wyprostowaniu

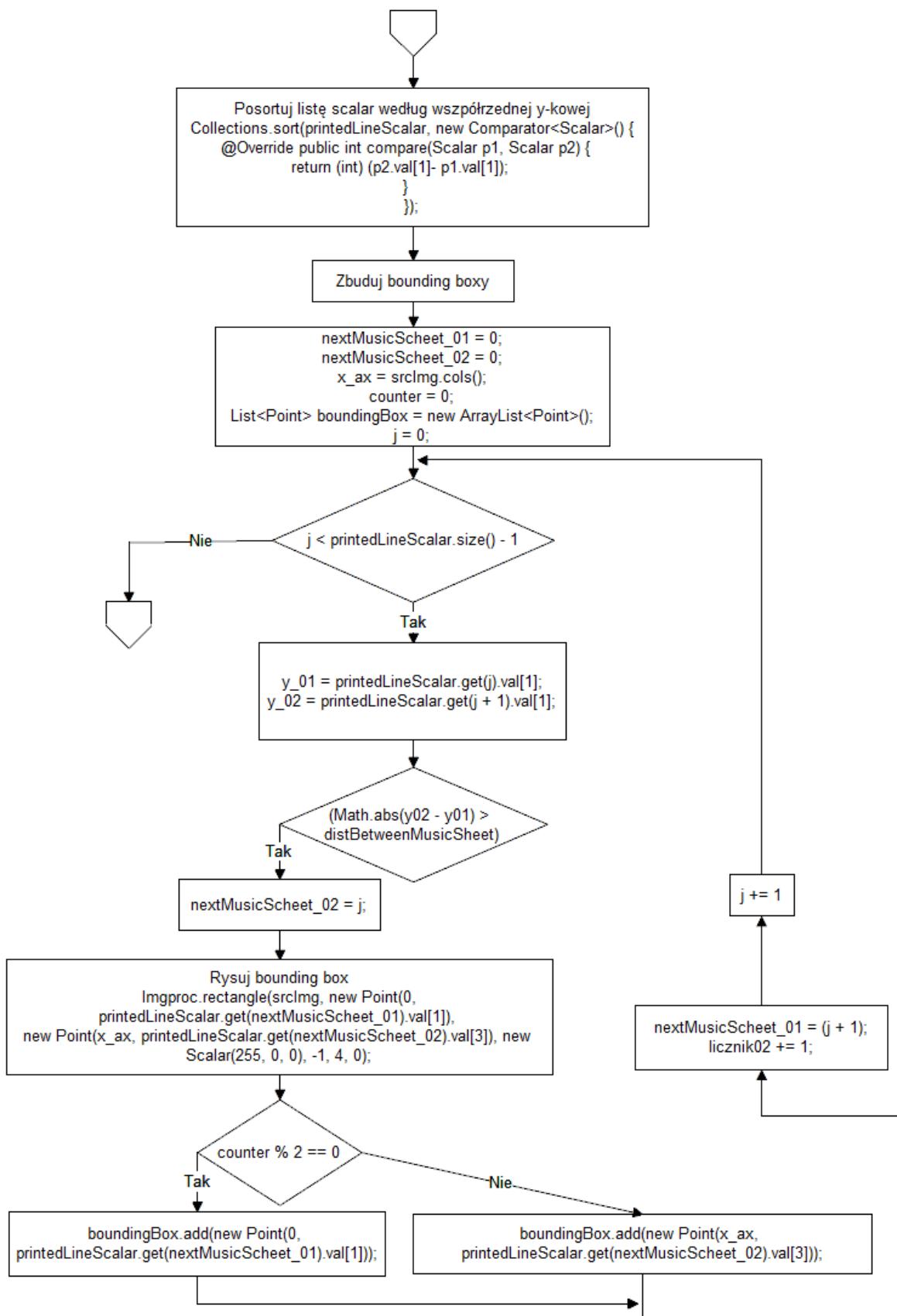
3.1.7 Główne przetwarzanie obrazu

Tak przygotowany obraz jest już wystarczająco przysposobiony, aby można było wykryć linie zawierające tekst oraz te zawracające nuty. Metody klasy MajorProcessing, w której zostały zaimplementowane algorytmy również używają metod klasy PrepareImgToAnalize. Wykryte linie transformatą Hougha zostają pogrupowane w zbiory tworzące pięciolinie. Odpowiednio w zależności od budowy śpiewnika z jedna pięciolinią (klucz wiolinowy) z tekstem umieszczonym pod nią lub z dwiema pięcioliniami oraz tekstem umieszczonym pomiędzy nimi. Następnie wykryte pięciolinie zostają zamarkowane utwarzając bounding boxy i wraz z tekstem znajdu-

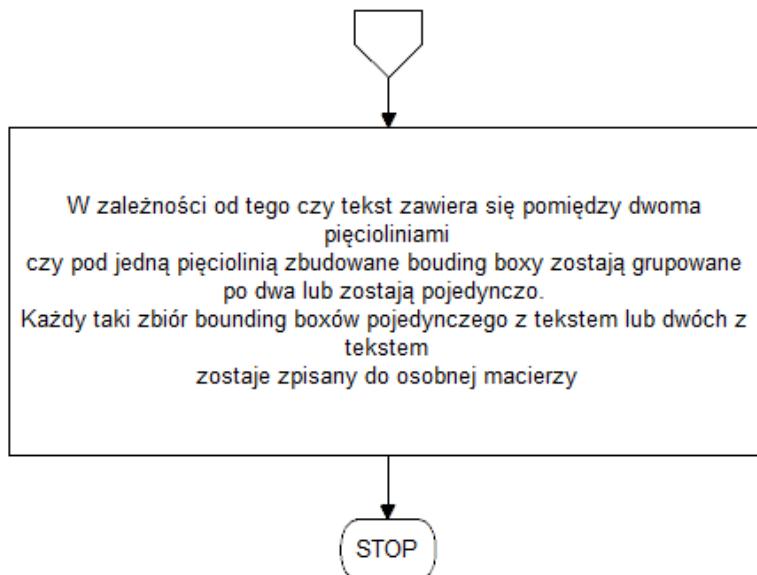
jącym się pod lub między nimi wycięte z obrazu i zapisane do osobnych plików czy macierzy przechowujących obrazy (Rysunek 23, 24, 25), a więc przygotowane do dalszego przetworzenia i ostatecznego uzyskania z nich tekstu.



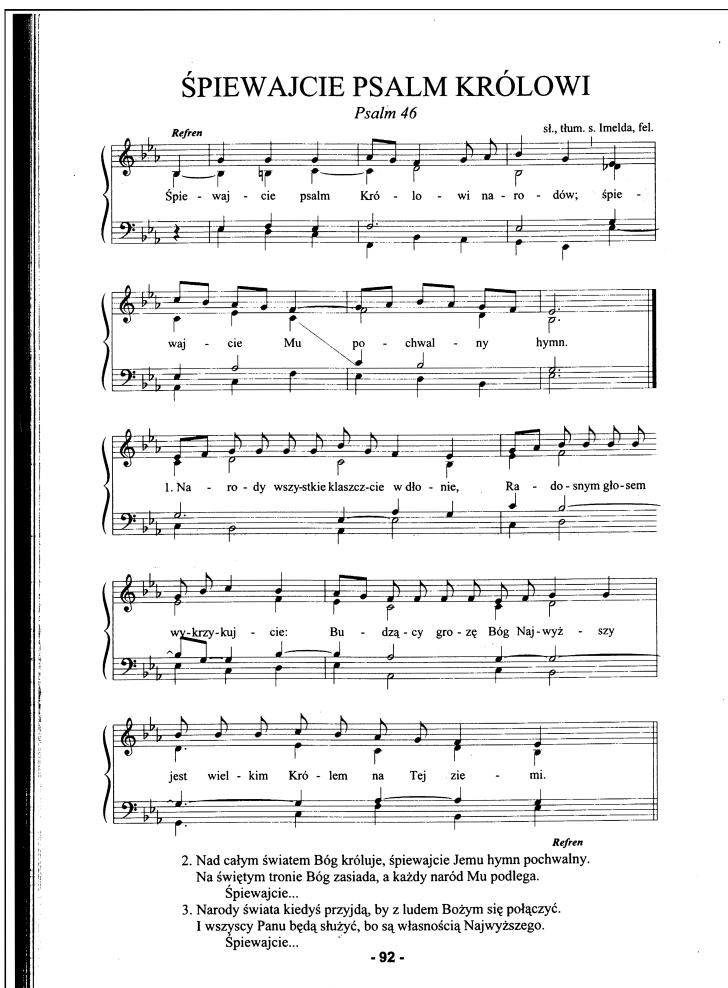
Rysunek 23: Główny algorytm budujący bounding boxy cz. 1



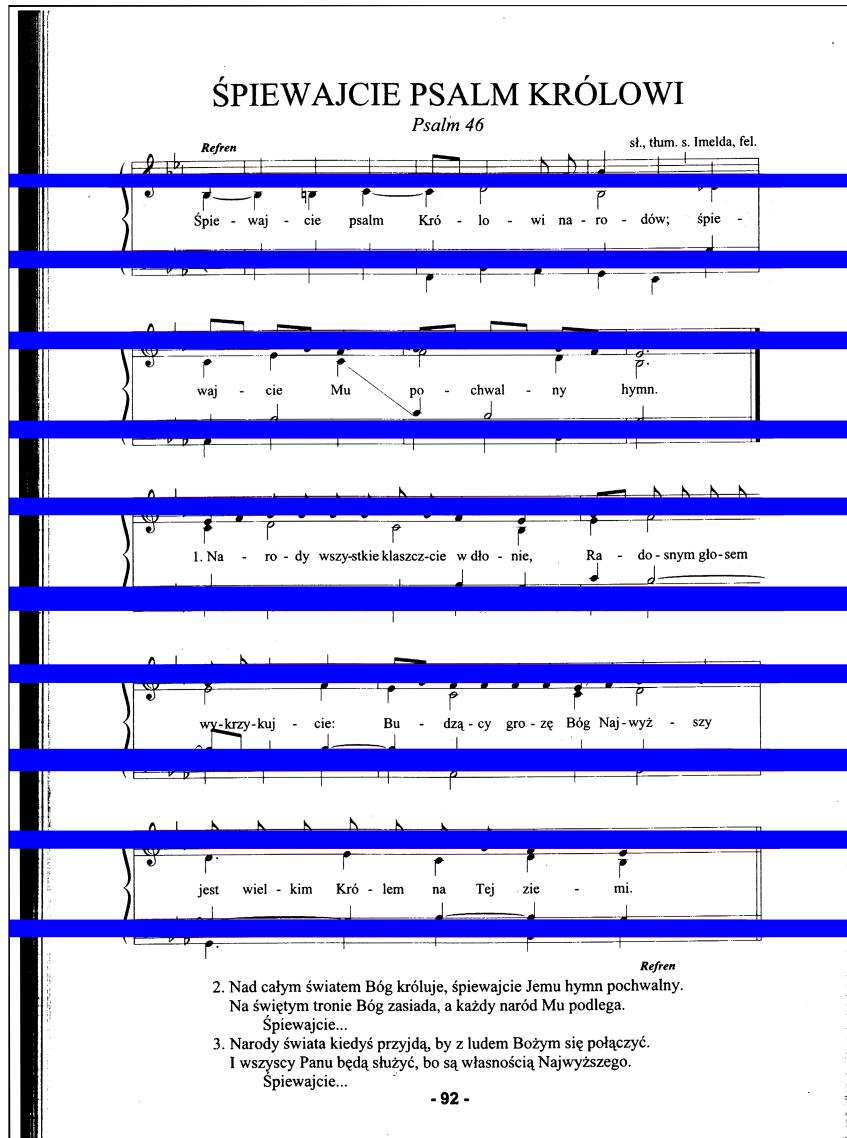
Rysunek 24: Główny algorytm budujący bounding boxy cz. 2



Rysunek 25: Główny algorytm budujący bounding boxy cz. 3



Rysunek 26: Obraz przed wykryciem pięciolinii



Rysunek 27: Obraz po wykryciu pięciolinii

3.1.8 Konfiguracja sieci neuronowej klasyfikującej linię

Model konwolucyjnej sieci neuronowej, który klasyfikuje obrazy wejściowe na te zawierające tekst i te zawierające linie muzyczne (Listing 7) został zawarty w klasie MusicSheetClassifier. Dane na których sieć się uczy zostały wcześniej przygotowane jako jednokanałowe zdjęcia o rozmiarach 600px x 15px.

```

1 int height = 600;
2 int width = 15;
3 int channels = 1; // single channel for grayscale images
4 int outputNum = 2; // 2 class classification
5 int batchSize = 128;
6 int nEpochs = 64;
7 int seed = 1234;
8
9 Map<Integer, Double> lrSchedule = new HashMap<Integer, Double>();
10 lrSchedule.put(0, 0.001); // iteration #, learning rate
11 lrSchedule.put(100, 0.0015);
12 lrSchedule.put(200, 0.0001);
13 lrSchedule.put(400, 0.00015);
14 lrSchedule.put(800, 0.00001);
15 lrSchedule.put(1200, 0.000015);
16 lrSchedule.put(1500, 0.0001);
17
18 MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
19     .seed(seed)
20     .l2(0.0015 * 0.005)
21     .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
22     .updater(new Nesterovs(new MapSchedule(ScheduleType.ITERATION, lrSchedule)))
23     .weightInit(WeightInit.XAVIER)
24     .list()
25     .layer(0, new ConvolutionLayer.Builder(5, 5)
26         .nIn(channels)
27         .stride(1, 1)
28         .nOut(20)
29         .activation(Activation.TANH)
30         .build())
31     .layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
32         .kernelSize(2, 2)
33         .stride(2, 2)
34         .build())
35     .layer(2, new ConvolutionLayer.Builder(1, 1)
36         .stride(1, 1) // nIn need not specified in later layers
37         .nOut(100)
38         .activation(Activation.IDENTITY)
39         .build())
40     .layer(3, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
41         .kernelSize(1, 1)
42         .stride(1, 1)
43         .build())
44     .layer(4, new DenseLayer.Builder().activation(Activation.RELU)
45         .nOut(500).build())
46     .layer(5, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
47         .nOut(outputNum)
48         .activation(Activation.SOFTMAX)
49         .build())
50     .setInputType(InputType.convolutionalFlat(600, 15, 1))
51     .backprop(true).pretrain(false).build());

```

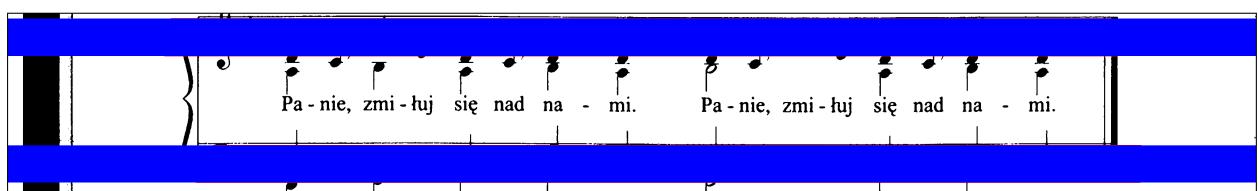
Listing 7: Konfiguracja sieci neuronowej klasyfikującej linię

# of classes:	liczba klas	2
Accuracy:	dokładność	0,9806
Precision:	precyzja	0,9768
Recall:	wycofanie	0,9808
F1 Score:	rezultat f1	0,9851

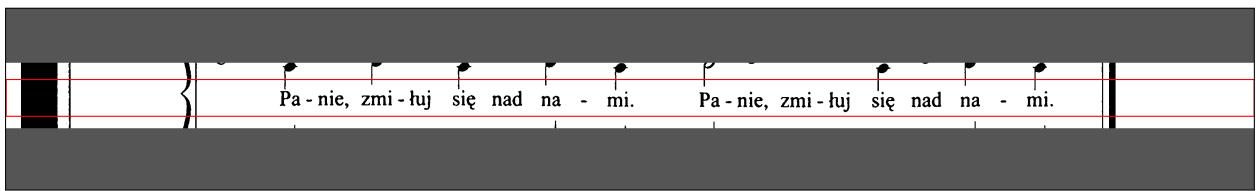
Właściwie przygotowane zbiory uczące pozwalają osiągnąć sieci bardzo dobre wyniki. Nieco gorsze wyniki sieć uzyskuje pracując na obrazach przekazanych przez algorytm budujący klastry. Może to być spowodowane koniecznością zmiany rozmiarów obrazów i pogorszeniu ich jakości. Rozwiązaniem tego problemu będzie ponowne przygotowanie zbioru obrazów uczących o zbliżonych rozmiarach do tych ze skanów śpiewników. Aktualnie sieć odgrywa drugoplanową rolę w algorytmie, jedynie go wspomagając. Polepszenie jej działania i niezawodności pozwoli na modyfikacje algorytmu tak aby klasyfikacja wykonywana przez sieć miała większa rolę niż dotychczas.

3.1.9 Detekcja tekstu

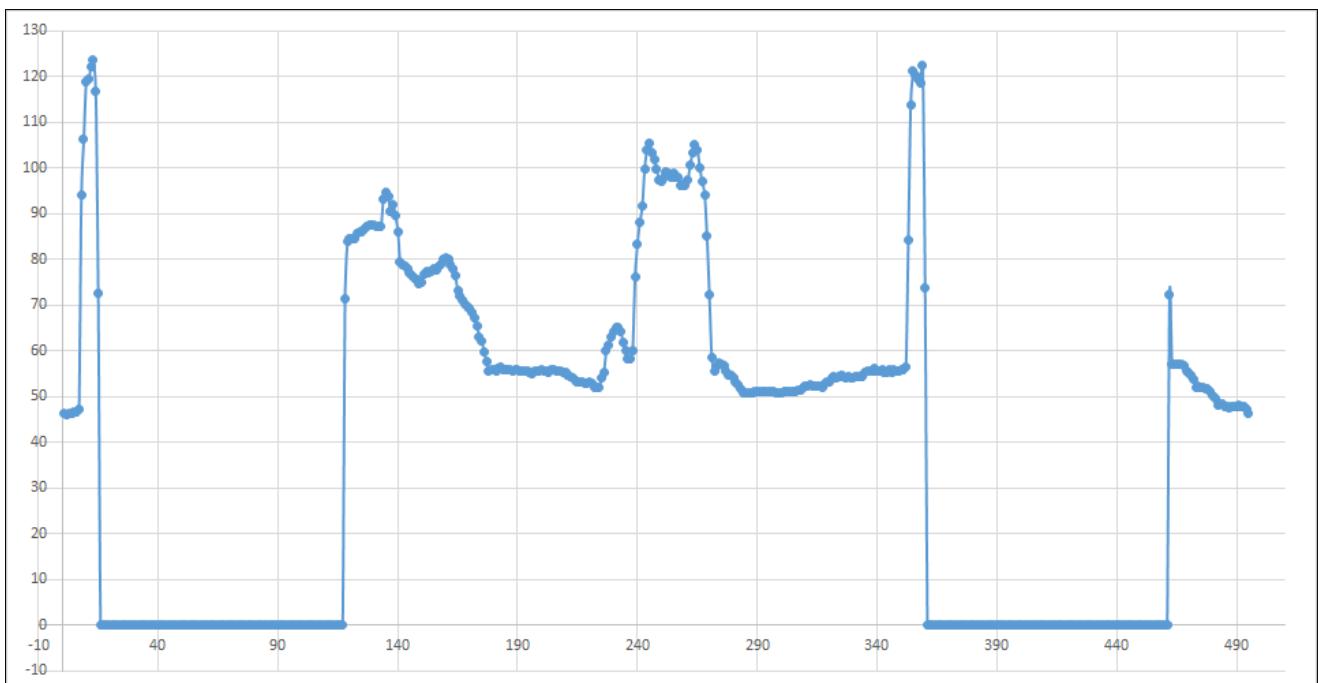
Powstałe w wyniku poprzednich algorytmów obrazy mogą zostać dalej przetwarzane tak aby zbudowane dotychczas boundung boxy zostały maksymalnie rozszerzone. Operacja ta została zaimplementowana w klasie DetectText. Algorytm wykorzystuje do tego wartości odchylenia standardowego, co umożliwia praktycznie całkowite usunięcie znaków muzycznych (nut i pięciolinii). Obszar w którym prawdopodobnie znajduje się tekst zostaje wycięty i jest obrazem wejściowym dla konwolucyjnej sieci neuronowej (opisanej w podrozdziale 3.1.8) która podejmuje decyzje czy jest on linią tekstu czy nut. Jeśli sieć zwróci wartość pozytywną (obraz jest tekstem; prawdopodobieństwo większe niż 0.9), zostaje on wycięty i poddany dalszej obróbce, zaś jeśli sieć zwróci wartość negatywną (obraz nie jest tekstem; prawdopodobieństwo mniejsze lub równe 0.9) dalszemu przetwarzaniu zostaje poddany cały obraz.



Rysunek 28: Obraz przed rozszerzeniem bounding boxów



Rysunek 29: Obraz po rozszerzeniu bounding boxów i zamarkowaniu miejsca gdzie znajduje się tekst



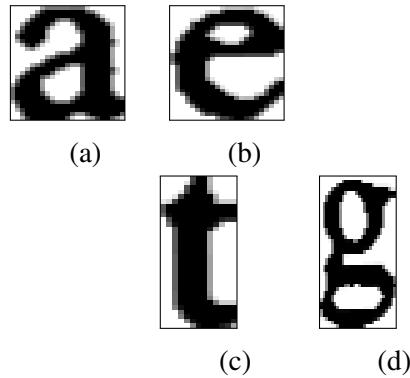
Rysunek 30: Wykres odchylenia standardowego poszczególnych linii

Na podstawie wykresu (Rysunek 30) można zauważyć że tekst znajduje się pomiędzy dwoma większymi obszarami o wartościach zerowych, dokładniej pomiędzy liniami od około 240 do około 270, odchylenie standardowe wierszy zawierających tekst wahaje się w zakresie od około 95 do około 105. Obszar pomiędzy tekstem a zamarkowanymi pięcioliniami ma niezbyt duże odchylenie standardowe ponieważ mogą się w tych liniach znajdować nuty dodane, ogonki nut jednak nie będzie ich nigdy zbyt wiele i rozrzut w tych liniach nie będzie większy od rozrzutu w liniach w których znajdują się litery.

3.1.10 Detekcja i wycięcie liter

Aby odczytać zawarty na obrazie tekst konieczna jest rozpoznanie pojedynczych znaków i ich odpowiednie przechowanie (właściwa kolejność). Algorytmy klasy DetectLetter wykrywają krawędzie w obrazie wejściowym, następnie grupują je w zbiory o taki samych współrzędnych x-owych, dzięki czemu jedna grupa konturów stanowi jeden znak (literę, znak interpunkcyjny,

etc.), grupy są sortowane rosnąco według współrzędnych x-owych. Z każdego zbioru konturów zostają wyznaczone skrajne współrzędne $\max_x, \min_x, \max_y, \min_y$. Na podstawie tych wartości z obrazu wejściowego zostaje wycięty kontur zawierający się w tym prostokącie. Tak wyciętemu obrazowi zostaje nadana jednoznaczna nazwa według jego indeksu w zbiorze posortowanym, według rosnącego rozmieszczenia na osi OX obrazu.



Rysunek 31: Przykładowe wycięte litery

3.1.11 Konfiguracja sieci neuronowej klasyfikującej litery

Do klasyfikacji liter została użyta konwolucyjna sieć neuronowa której model znajduje się w klasie LetterClassifier (Listing 8). Sieć uczy się rozpoznawać litery abecadła, w tym również polskich znaków diakrytycznych, majuskuł i minuskuł (łącznie 64 znaki, klasy), na podstawie wcześniej przygotowanych jednokanałowych zdjęć, o rozmiarach 50px x 25px, Konfiguracja sieci różni się od poprzedniej (Listing 7) jedynie kilkoma parametrami i dodatkowymi dwoma warstwami: ConvolutionLayer.Builder(x, y) i SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MA

```

1 int height = 50;
2 int width = 25;
3 int channels = 1; // single channel for grayscale images
4 int outputNum = 64; // 64 letters
5 int batchSize = 128;
6 int nEpochs = 64;
7 int seed = 1234;
8
9 Map<Integer, Double> lrSchedule = new HashMap<>();
10 lrSchedule.put(0, 0.001); // iteration #, learning rate
11 lrSchedule.put(100, 0.0015);
12 lrSchedule.put(200, 0.0001);
13 lrSchedule.put(400, 0.00015);
14 lrSchedule.put(800, 0.00001);
15 lrSchedule.put(1200, 0.000015);
16 lrSchedule.put(1500, 0.0001);

```

```

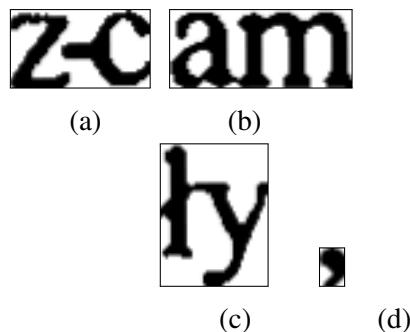
17
18 MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
19     .seed(seed)
20     .l2(0.0015 * 0.005)
21     .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
22     .updater(new Nesterovs(new MapSchedule(ScheduleType.ITERATION, lrSchedule)))
23     .weightInit(WeightInit.XAVIER)
24     .list()
25     .layer(0, new ConvolutionLayer.Builder(5, 5)
26         .nIn(channels)
27         .stride(1, 1)
28         .nOut(20)
29         .activation(Activation.TANH)
30         .build())
31     .layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
32         .kernelSize(4, 4)
33         .stride(2, 2)
34         .build())
35     .layer(2, new ConvolutionLayer.Builder(3, 3)
36         .stride(1, 1) // nIn need not specified in later layers
37         .nOut(100)
38         .activation(Activation.TANH)
39         .build())
40     .layer(3, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
41         .kernelSize(2, 2)
42         .stride(1, 1)
43         .build())
44     .layer(4, new ConvolutionLayer.Builder(1, 1)
45         .stride(1, 1)
46         .nOut(200)
47         .activation(Activation.TANH)
48         .build())
49     .layer(5, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
50         .kernelSize(1, 1)
51         .stride(1, 1)
52         .build())
53     .layer(6, new DenseLayer.Builder().activation(Activation.RELU)
54         .nOut(500).build())
55     .layer(7, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
56         .nOut(outputNum)
57         .activation(Activation.SOFTMAX)
58         .build())
59     .setInputType(InputType.convolutionalFlat(50, 25, 1))
60     .backprop(true).pretrain(false).build();

```

Listing 8: Konfiguracja sieci neuronowej klasyfikującej litery

# of classes:	liczb klas	64
Accuracy:	dokładność	0,7387
Precision:	precyzja	0,7098
Recall:	wycofanie	0,5776
F1 Score:	rezultat f1	0,6347

Trening sieci neuronowej daje dość zadowalające wyniki. Sieć bardzo dobrze rozpoznaje (zwraca wysokie prawdopodobieństwo > 0.9) pojedyncze litery podobne do tych ze zbioru na którym się uczy. Podczas pracy ze zbiorzem danych ze śpiewników nie zawsze algorytm rozpoznawania konturów pojedynczych liter zwraca zadowalające wyniki, często gdy kontury liter zachodzą na siebie zwraca kilka liter w jednym obrazie lub dodatkowych znaków co uniemożliwia poprawne rozpoznanie przez sieć.



Rysunek 32: Przykładowe znaki, litery wycięte niepoprawnie

3.1.12 Klasyfikacja odczytanych liter

Algorytmy klasy ReadLetterClassifier odczytują przygotowany przez klasę DetectLetter obraz zawierający znak, następnie obraz ten zostaje analizowany przez system OCR Tesseract oraz konwolucyjną sieć neuronową. Wybranie najbardziej prawdopodobnej litery odbywa się przez porównanie wyniku zwróconego przez system OCR i trzech największych prawdopodobieństw zwróconych przez sieć. Tak odczytany cały wykryty tekst znajdujący się na stronie zostaje zapisany do pliku tekstowego.

Przykładowy odczytany tekst: **Nie wiem czy jednak umieszczać ten przykładowy tekst :)**

"ILH HŁ»» gzFII—I,=iChwałaTobiesłowoBożechwałaTobiesłowoBożeIłlełt»

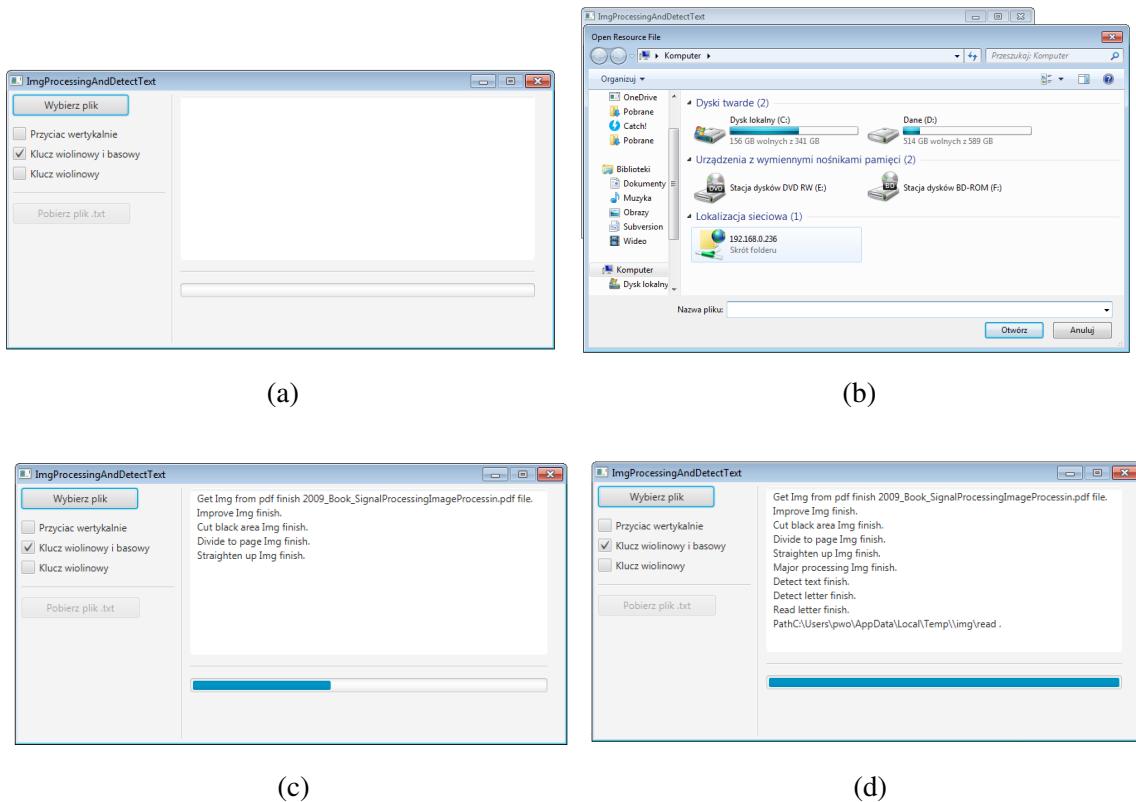
———— „AA BtChnenwCue wałaTobiesłowoBoże 222121ałaTobiesłowoBoże

›ie S5łłowoBożechwałaTobie›”J«eui;IW7 LA FLFA CI

W,,chwałaTobie,,ł3i hia IłłaTobie,,u i,,«——IłłowoBo

EIOWOAj0Boze „uiB n,, l> H;1N ki
 DJllAllelujaallelu—jazJJA;JJtl AllJ elui „>
 Iall lelujaA thJ j FJr I—”

3.1.13 Graficzny interfejs użytkownika



Rysunek 33: GUI

Wszystkie opracowane rozwiązania zostały udostępnione poprzez prosty graficzny interfejs użytkownika. Po uruchomieniu programu pojawia się okienko (Rysunek 33 (a)), następnie użytkownik określa kilka parametrów, które wykorzystują algorytmy (czy dodatkowo ma być usunięty czarny obszar po lewej stronie, czy tekst znajduje się pomiędzy dwoma liniami muzycznymi czy pod linią muzyczną). Kolejnym krokiem jest wybór pliku z zasobów komputera, (Rysunek 33 (b)). Teraz wybrany plik zostaje przetworzony przez wszystkie wcześniej opisane algorytmy. Okienko wyświetla informację jaki algorytm został wykonany i zmienia się niebieski pasek postępu. Gdy zdjęcie przejdzie przez wszystkie algorytmy pasek postępu osiąga wartość maksymalną i zostaje wyświetlona ścieżka do wyjściowych plików .txt.

4 Podsumowanie

W niniejszej pracy dyplomowej zostały zrealizowane główne jej założenia. Opracowane algorytmy umożliwiają pozyskanie tekstu ze stron zawierających znaki muzyczne. Wśród uzyskanego tekstu możemy odnaleźć poszczególne sylaby, a nie rzadko i wyrazy. Identyfikacją liter składających się na poszczególne wyrazy może zostać jeszcze polepszona, przede wszystkim przez wykorzystanie dokładniejszych algorytmów wycinających poszczególne litery, powiększenie zbioru liter (majuskuł, minuskuł) i linii (tekstu, pięciolinii) na których uczą się sieci neuronowe. Dzięki zastosowaniu sztucznej inteligencji opracowane algorytmy są bardziej uniwersalne i umożliwiają pracę ze znaczącą liczbą powszechnie wykorzystywanych śpiewników wpisujących się w następujące dwa schematy [pięciolina tekst pięciolina], [pięciolina tekst]. Dzięki wytworzeniu oprogramowania zgodnie z zasadami programowania zorientowanego obiektywnie, z łatwością możemy do powyższych dwóch struktur dodawać kolejne w różnych konfiguracjach na przykład [pięciolina tekst pięciolina pięciolina].

Kolejnym z celów pracy, który został zrealizowany to budowa graficznego interfejsu użytkownika, wybrana technologia i architektura systemu umożliwiła dość proste przejście z aplikacji konsolowej do aplikacji posiadającej prosty interfejs graficzny. Aplikacja może, a nawet powinna być rozwijana również w kierunku budowy bardziej rozbudowanego GUI, umożliwiającego użytkownikowi dobór parametrów (dla algorytmów) dla wybranego pliku poprzez udostępnienie rezultatów poszczególnych algorytmów w czasie rzeczywistym.

Dzięki pracy nad budową aplikacji mogłem rozwinąć swoje zdolności algorytmiczne i programistyczne. Zapoznanie się z podstawami teoretycznymi systemów przetwarzających tekst, działania sztucznych sieci neuronowych, algorytmów przetwarzających obrazy oraz z narzędziami związanymi z tymi zagadnieniami – Tesseract, OpenCv, DL4J pozwoliło autorowi na zdobycie i rozwój swojej wiedzy i umiejętności. **Umiejętności zdobyte podczas pisania pracy dyplomowej będą stanowiły ciekawą pozycję dla przyszłych pracodawców. nie wiem jak to lepiej ująć ze to całkiem ciekawie może się prezentować w cv**

5 Bibliografia

Źródła książkowe:

- Rączkowski F., Śpiewajmy Bogu, Płock, Hejnał, 2012;
- Siedlecki J., Śpiewnik kościelny, Wyd. XL (poprawione), Kraków, Instytut Teologiczny Księży Misjonarzy, 2011;
- Kaehler A., Bradski G., OpenCV 3. Komputerowe rozpoznawanie obrazu w C++ przy użyciu biblioteki OpenCV, Gliwice, Helion, 2017;
- Shanmugamani R., Deep Learning for Computer Vision, Birmingham, Packt Publishing, 2018

Źródła internetowe:

Data dostępu 2018.07.15

- <https://opencv.org>
- <https://docs.opencv.org/3.1.0/index.html>
- <https://docs.opencv.org/java/3.1.0>
- <http://tess4j.sourceforge.net>
- <https://deeplearning4j.org/index.html>
- <https://nd4j.org/doc>
- <https://depiesml.wordpress.com/category/deeplearning4j>
- <https://ksopyla.com/python/operacja-splotu-przetwarzanie-obrazow>
- <https://github.com/Kulbear/deep-learning-nano-foundation/wiki>
- <http://www.szkolazpasja.pl/rastrowa>