Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Projekt inżynierski

Wykorzystanie sieci neuronowych do detekcji ziaren w obrazach termowizyjnych

Autor: Maciej Ziaja
Kierujący pracą: dr inż. Sebastian Budzan

Gliwice, styczeń 2020

Streszczenie

Tematem pracy jest...

Deklaracja

Deklaruję że...

Podziękowania

Pragnę podziękować...

Spis treści

Rozdział 1

Wstęp

1.1 Motywacja projektu

1.2 Cel pracy

Rozdział 2

Założenia projektowe i wykorzystane narzędzia

2.1 Analizowane pyły rud miedzi

2.2 Rodzaje termowizji i idea wykorzystania pomiarów termowizyjnych

Pomiary termowizyjne polegają na rejestracji promieniowania cieplnego obiektów w celu ustalenia ich temperatury. Badania z wykorzystaniem termowizji można podzielić na dwie grupy: *termowizję pasywną* oraz *termowizję aktywną*.

Technika pasywna polega na rejestracji obrazów obiektów bez ingerencji w ich temperaturę w czasie trwania pomiarów. Można w ten sposób obserwować przepływ ciepła w urządzeniach technicznych, procesach przemysłowych oraz biologicznych. W rozważanym w pracy przypadku technika ta nie ma jednak zastosowania. W warunkach pokojowych badany materiał ma w całej swojej objętości podobną temperaturę co skutkuje obrazem równomiernego szumu w rejestrowanym obrazie.

Bardziej zaawansowaną techniką są pomiary z wykorzystaniem termowizji aktywnej. W tym trybie pomiarów badane obiekty nagrzewa się w początkowej fazie pomiaru, w powtarzalnych warunkach, a następnie rejestruje proces ich stygnięcia. Podczas dostarczania ciepła do obiektu oraz jego oddawania na obrazach termowizyjnych można zaobserwować strukturę badanego obiektu. Przedmioty o niejednorodnej i złożonej budowie nagrzewają się oraz stygną nierównomiernie co jest rejestrowane przez kamery termowizyjne. W ten sposób można dostrzec cechy obiektu niewidoczne gołym okiem, takie jak zmiany jego gęstości, składu oraz uszkodzenia struktury. Termowizję aktywną stosuje się szeroko w badaniach naukowych oraz przemyśle. Przykładowe aplikacje tej techniki to wykrywanie defektów

w produktach przemysłowych oraz wykrywanie części konstrukcji podatnych na zużycie. Termowizja aktywna jest techniką niedestruktywną oraz bezkontaktową, co stanowi jej zalety w analizie materiałowej. Technika ta wymaga jednak etapowego procesu pomiarowego, potrzebne jest przygotowanie instalacji grzewczej, a nagrzewanie i stygnięcie materiału może być czasochłonne. Przebieg zmian temperatury najlepiej rejestrować na materiałach wideo, aby zmaksymalizować ilość danych zgromadzonych w trakcie eksperymentu.

2.3 Kamera termowizyjna FLIR A320

Przedmiotem projektu jest analiza obrazów pochodzących z przemysłowej kamery termowizyjnej FLIR A320. Firma FLIR zajmuje się produkcją wysokiej jakości kamer i detektorów do celów profesjonalnych i przemysłowych. Używana kamera łączący wysoką jakość pomiaru z nowoczesnymi funkcjami integracji z oprogramowaniem komputerowym. Urządzenie komunikuje się z komputerem za pomocą kabla ethernetowego, pozwalając na kontrolę z poziomu oprogramowania oraz bibliotek programistycznych. Dodatkowo kamera posiada możliwość planowania automatycznych pomiarów, alarmów, oferuje funkcje analityczne oraz wbudowany serwer internetowy.

2.3.1 Opis sprzętowy wykorzystywanej kamery

Kamera ma postać podłużnego korpusu, do którego swobodnie można podłączać pasujące obiektywy. Na rysunku 2.1 przedstawiono zdjęcie wykorzystywanego urządzenia. Egzemplarz kamery znajdujący się w laboratorium termowizji Politechniki Śląskiej został wyposażony w obiektyw, pozwalający oglądać ziarna rud miedzi w powiększeniu. Kamera cechuje się następującymi parametrami:

- typ detektora: niechłodzony mikrobolometr
- rozdzielczość: 320 na 240 pikseli,
- częstotliwość odświeżania: 9Hz do 30Hz
- szerokość otworu: $f1.3$,
- autofokus z wbudowanym silnikiem,
- zakres pomiarowy temperatur:
  1. od -15°C do 50°C,
  2. od 0°C do 350°C,

Rysunek 2.1: Kamera termowizyjna FLIR A320

- dokładność: ±2°C lub ±2% odczytu,
- zakres wykrywanego widma promieniowania: 7.5µm do 13µm.

Kamera wykrywa temperaturę przez detektor zwany *bolometrem*, który mierzy energię niesioną przez fale elektromagnetyczne w spektrum podczerwieni. Kiedy fala pada na detektor kamery, temperatura komórek matrycy rośnie i zwiększa się ich rezystancja elektryczna, co wpływa na natężenie prądu przepływającego przez obwody w czujniku. Wartości natężenia prądu są mierzone, a na ich podstawie określana jest mierzona temperatura.

Zakres temperatur kamery jest odpowiedni do przeprowadzenia eksperymentów z pomiarami metodą termowizji aktywnej. Próbki planuje się podgrzewać do temperatury maksymalnie około 80°C, wartość ta mieści się w zakresie pracy urządzenia. Dokładność kamery jest zadowalająca, próbne materiały nagraniowe wskazały, że na zdjęciach widocznych jest wiele szczegółów i detali badanego materiału. Przy klasyfikacji obrazów i wzorców stygnięcia próbek jest to bardziej istotne niż liczbowa dokładność pomiarowa przyrządu.

W porównaniu ze zwykłymi, współczesnymi aparatami rozdzielczość kamery termowizyjnej może wydawać się bardzo mała. Należy sobie jednak uzmysłowić, że w standardowych aparatach piksele mają rozkład Bayera, a wartości składowych koloru są interpolowane. W kamerze termowizyjnej każdy piksel dokonuje pełnego pomiaru wartości temperatury, dlatego bezpośrednie porównanie rozdzielczości używanego przyrządu z popularnymi aparatami może być mylące. Oczywi-

ście większa rozdzielczość kamery byłaby pożądana, jednak jej obecne możliwości pozwalają na szczegółowe pomiary i obserwacje wielu detali ziaren rud miedzi.

2.3.2 Obiektyw kamery

Kamerę wyposażono w obiektyw zbliżeniowy FLIR T19741, pozwalający obserwować ziarna rud miedzy. Jest to sprzęt zaprojektowany przez producentów używanej kamery i dedykowany pracy z urządzeniami termowizyjnymi. Przyrząd przedstawiono na rysunku 2.2. Wybrany obiektyw jest przygotowany z myślą o obserwacji

Rysunek 2.2: Obiektyw FLIR T197415

drobnych detali powierzchni w dużym zbliżeniu. Używany model ma następujące parametry:

- ogniskowa: 18.2mm,
- powiększenie: 1×1,
- pole widzenia: 8mm na 6mm,
- odległość od płaszczyzny ostrzenia: 20mm,
- głębia ostrości: 0.3mm,
- przysłona: bez regulacji, równa otworowi systemu montażu kamery,
- budowa: trzy soczewki asferyczne.

Zgodnie ze specyfikacją producenta obiektyw ma powiększenie 1×1, co może wydawać się niedużą wartością. Należy mieć jednak świadomość, że zwykłe obiektywy zmniejszają obraz padający na matrycę, powszechnie przyjmuje się jako granicę makrofotografii powiększenie 1×1. W obserwacji ziaren i detali powierzchni nie jest jednak istotne powiększenie, ale to że używany obiektyw jest *zbliżeniowy*. Oznacza to, że ma on bardzo małą odległość ostrzenia, czyli można go przysunąć blisko obserwowanej powierzchni. Obiektyw zbliżeniowy pozwala na obserwację z dystansu 20mm, typowe obiektywy ostrzą z odległości parudziesięciu centymetrów do ponad metra. Dzięki temu obiektyw zbliżeniowy pozwala na obserwację bardzo drobnych detali powierzchni.

2.3.3 Oprogramowanie do obsługi kamery

Jedną z najważniejszych cech kamery jest łatwość jej integracji z oprogramowaniem komputerowym. Kamerę można obsługiwać za pomocą programu FLIR Tools. Pozwala on na podgląd obrazu oraz wykonywanie zdjęć termowizyjnych. Producent dostarcza również bibliotekę LabVIEW pozwalającą na zaawansowaną pracę z kamerą. Do obsługi stanowiska został napisany program używający tych bibliotek, który pozwala na nagrywanie materiałów wideo przy pomocy kamery. Nagrane materiały mają własnościowy format firmy FLIR, jednak można je odtwarzać w programie FLIR Tools. Program pozwala także na eksport stopklatek z nagrania, w postaci plików JPEG. Przy obsłudze narzędzia ważne jest ustawianie zakresu temperatur na obrazie. Wybrany zakres decyduje w jaki sposób wartości temperatury są mapowane na kolory w obrazie zawarte w tablicy LUT. Program oferuje tablice w skali szarości, takie zostały użyte w projekcie, możliwy jest także wybór tablic w postaci kolorowych gradientów. Wybór zakresu wpływa na wygląd wyświetlanego obrazu oraz eksportowanych stopklatek. Jego nieodpowiedni dobór może skutkować zbyt ciemnym, jasnym, lub mało kontrastowym obrazem. Aby zapewnić najlepsze wykorzystanie nagrań z kamery oraz powtarzalny charakter eksportu stopklatek korzystano z opcji automatycznego doboru zakresu temperatur, jaki jest wbudowany w program FLIR Tools.

2.4 Narzędzia programistyczne

2.4.1 Język programowania Python

Założenia projektu wymagają użycia języka programowania pozwalającego na zaawansowaną obróbkę obrazu oraz wydajne budowanie sieci neuronowych. W obu tych dziedzinach wiodącym językiem jest Python, posiadający bogaty zestaw bibliotek. Język ten oferuje dużą wygodę programowania, co jest istotne przy proto-

typowaniu oraz pracy badawczej. Jednocześnie Python jest zaopatrzony w wydajną bibliotekę obliczeń numerycznych NumPy. Oferuje ona klasy macierzy numerycznych oraz bogaty zestaw operacji matematycznych. Macierze biblioteki NumPy są mniej elastyczne niż zwykłe listy języka Python, jednak oferują dużo większą wydajność obliczeniową, między innymi dzięki wsparciu obliczeń wektorowych na różnych procesorach. Wiele innych bibliotek wykorzystuje pakiet NumPy, na przykład przy przetwarzaniu obrazów, co pozwala na wysoką wydajność obliczeń. Połączenie wygody programowania z zadowalającą wydajnością bibliotek numerycznych, sprawia że język Python jest dobrym wyborem do realizacji założeń projektu.

2.4.2 Biblioteka przetwarzania obrazu Scikit-image

Język Python oferuje bogaty zestaw bibliotek przetwarzania obrazu. Najbardziej popularną z nich jest biblioteka OpenCV napisana w języku C++. Zdecydowano się jednak na wybór mniej znanego pakietu Scikit-image. Jest to biblioteka nastawiona bardziej na obliczenia naukowe i badawcze oraz napisana bezpośrednio z myślą o języku Python. Scikit-image, w porównaniu z OpenCV, oferuje bardziej spójny interfejs programisty oraz lepiej wykorzystuje charakterystykę języka Python. Dodatkowo wybrana biblioteka jest częścią zestawu Scikit, w skład którego wchodzi pakiet Scikit-learn służący do uczenia maszynowego. Może on być przydatny przy tworzeniu sieci neuronowej, a spójność bibliotek z pakietu Scikit jest niewątpliwie zaletą. Wybrana biblioteka charakteryzuje się również dobrą dokumentacją oraz zestawem przydatnych przykładów.

2.4.3 Interfejs sieci neuronowych Keras i biblioteka TensorFlow

Wybór bibliotek głębokiego uczenia w języku Python jest bardzo szeroki. Do najpopularniejszych pakietów należą: Keras, TensorFlow oraz PyTorch. Zdecydowano się na wybór interfejsu biblioteki Keras. Jest to pakiet nastawiony na elastyczność i możliwość eksperymentowania, oferuje wysokopoziomową, matematyczną warstwę abstrakcji opisu sieci neuronowych. Keras nie jest jednak pełnym rozwiązaniem, a raczej abstrakcyjnym interfejsem. Używanie go wymaga wyboru wewnętrznego silnika biblioteki. Zdecydowano się na wybór domyślnej opcji i użycie zaplecza pakietu TensorFlow. W standardowy sposób interfejsu Keras używa się, po osobnym zainstalowaniu jego pakietu modułów i wskazaniu mu silnika sieci neuronowych. Biblioteka TensorFlow udostępnia jednak wewnętrzny interfejs interfejsu Keras, co pozwala na zwiększenie wydajności obliczeniowej w połączeniu dwóch bibliotek. Przy tworzeniu oprogramowania wzięto to pod uwagę i użyto bardziej wydajnej metody importowania interfejsu Keras.

Rozdział 3

Analiza i ekstrakcja danych z kamery termowizyjnej

3.1 Proces pomiarowy i budowa zbioru danych

Zgodnie z opisem technik termowizyjnych przedstawionym w sekcji 2.2 zdecydowano się na przeprowadzenie pomiarów za pomocą termowizji aktywnej. Aby zrealizować pomiary uprzednio przygotowano stanowisko laboratoryjne. Kamera termowizyjna została umieszczona na statywie, a do ogrzewania próbek zdecydowano się wykorzystać lampę halogenową.

W procesie termowizji aktywnej istotna jest charakter procesu nagrzewania materiału. Przy przygotowaniu pomiarów należało zdecydować przy jakim warunku zakończyć przekazywanie ciepła do próbki. Rozważono dwie możliwości:

a) ogrzewanie próbek do osiągnięcia ustalonej temperatury,

b) ogrzewanie próbek przez określony, stały czas.

Zdecydowano się na metodę b), ze względu na wygodę jej realizacji. Doprowadzenie każdej próbki do tej samej temperatury wymagałoby pomiarów w czasie nagrzewania, co jest bardziej wymagające do realizacji. Zgodnie ze wstępnymi obserwacjami nagrzewanie materiału przez określony czas pozwala na obserwację jego cech unikalnych i wzorców zachowania podczas stygnięcia. Następnie należy wybrać czas nagrywania materiałów wideo kamerą. Na podstawie wstępnych obserwacji i próbnych nagrań zdecydowano się na ogrzewanie próbek przez jedną minutę oraz rejestrację ich stygnięcia przez cztery minuty. Taka konfiguracja daje przy badanych pyłach rud miedzi ostry i szczegółowy obraz w początkowej fazie nagrywania oraz widocznie rozmazany i mniej kontrastowy materiał pod koniec stygnięcia próbek. Charakter procesu przejścia między tymi stanami pozwoli na klasyfikację badanych próbek. Ostatnia decyzja kształtująca charakter pomiarów

dotyczy chwili przechwytywania stopklatek z pozyskanych materiałów wideo. Na podstawie obserwacji zdecydowano się eksportować 5 klatek na początku każdej minuty nagrania. Po ustaleniu planu eksperymentu pomiarowego przystąpiono do jego wykonania. Zgodnie z opisem badanych materiałów w sekcji 2.1, zgromadzono materiały wideo dla czterech klas ziaren rudy miedzi. Ze względu na czasochłonność pomiarów dla każdej klasy materiału nagrano trzy materiały wideo. Przy nagrywaniu stygnięcia tej samej klasy materiały pomiędzy pomiarami próbkę poddawano przemieszaniu, aby uniknąć powtarzania struktur ułożenia ziaren dla tej samej klasy rud miedzi. Z pozyskanych nagrań wyeksportowano stopklatki używając programu FLIR Tools, pamiętając o używaniu algorytmu automatycznego doboru zakresu temperatur zgodnie z opisem w podsekcji 2.3.3.

Proces pomiarowy w protypowych warunkach był obarczony brakiem dużej precyzji i powtarzalności. Obserwacja uzyskanych nagrań pokazała, że obrazy tej samej klasy ziaren niekoniecznie osiągały tę samą temperaturę po ogrzewaniu przez ustalony czas. Problemem okazało się również dostosowanie ostrości obrazu z kamery. Jak wytłumaczono w podsekcji 2.3.2 wykorzystywany aparat cechuje się bardzo małą głębią ostrości, co powoduje że drobne ruchy kamery mogą spowodować utratę czytelności obrazu. Z kolei proces pomiarowy wymagał ciągłego przenoszenia próbki między stanowiskiem do podgrzewania materiału oraz nagrywania filmów. Należy mieć na uwadze że w czasie układania próbek pod kamerą oraz poprawiania ostrości postępowało stygnięcie ziaren, co sprzyjało brakowi powtarzalności pomiarów. Ze względu na przypadkowe utraty ostrości przy nagrywaniu oraz zbyt długi czas przenoszenia i przygotowania próbki do nagrywania, eksperyment wymagał czasem powtórnego nagrania próbki. Ocenę jakości procesu pomiarowego przestawiono na podstawie jego użyteczności w klasyfikacji ziaren, pod koniec pracy. Wnioski na temat stosowności i rozmiaru zbioru danych przedstawiono w rozdziale 5.

3.2 Analiza zebranych obrazów termowizyjnych

Zgodnie z zamysłem pomiarów przedstawionym w sekcji 3.1 z każdego nagrania wybrano pięć stopklatek. Podczas eksperymentu uzyskano łącznie dwanaście pomiarów, zawierających sumarycznie 60 zdjęć. W ramach uczenia maszynowego jest to bardzo mały zbiór danych, biorąc jednak pod uwagę wstępno-badawczy charakter pracy oraz czasochłonność procesu pomiarowego zdecydowano, że jest to rozmiar zadawalający do pierwszych prób klasyfikacji. Zebrane materiały mają format JPEG, do oznaczania zdjęć przyjęto schemat nazw jak w przykładzie: 115_E11R_1, gdzie człony nazwy oznaczają kolejno:

- automatyczny numer nagrania w programie FLIR Tools,

- klasę próbki,
- minutę nagrania.

3.2.1 Prezentacja przykładowej serii pomiarowej

Jak opisano w rozdziale 3.1 jedna próbka w zbiorze danych składa się z serii pięciu zdjęć o malejącym kontraście i szczegółowości. Na rysunku 3.1 przedstawiono przykładową próbkę 104 klasy E5R. Widoczny jest proces stygnięcia materiału. Skala po prawej stronie obrazów ma malejące na kolejnych obrazach wartości co pokazuje że następuje zmniejszenie temperatury na całym obrazie. Dodatkowo obraz staje się coraz mniej wyraźny i kontrastowy. Ze względu na budowę materiału nagrzana próbka emituje ciepłoze swoich zróżnicowanych struktur w niejednorodny sposób. Wraz z ochłodzeniem próbki jej temperatura się wyrównuje i kamera termowizyjna rejestruje coraz mniej szczegółów. Detale i elementy charakterystyczne obrazu zlewają się na kolejnych zdjęciach, wraz z opadaniem temperatury na zdjęciu pojawia się także coraz więcej szumów.

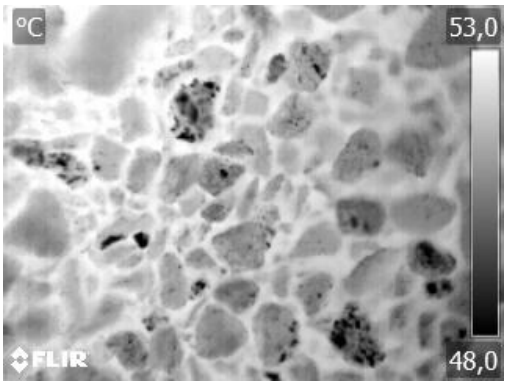
3.2.2 Przetwarzanie danych wizyjnych

3.2.3 Poprawa jakości obrazu

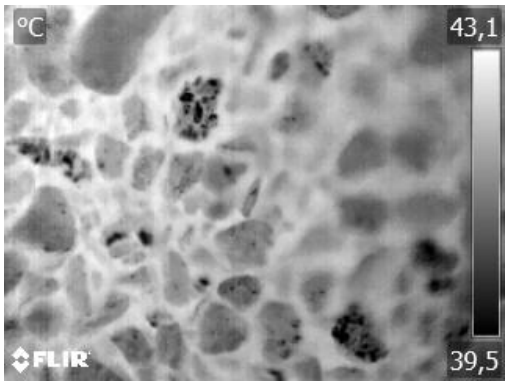
3.2.4 Automatyczny odczyt zakresu pomiarowego temperatur z obrazu

Jak wspomniano w sekcji 2.3.3 jednym z kluczowych czynników decydujących o wyglądzie obrazów pochodzących z kamery jest zakres temperatur mapowany na kolory w obrazie. Niestety aplikacja FLIR Tools nie pozwala na eksport zakresu temperatur wraz ze zdjęciami w formie liczbowej. W czasie zapisu zdjęć oprogramowanie dodaje na nich interfejs z aktywną skalą pomiarową, jednak jest on graficznie naniesiony na obraz. Aby ułatwić w przyszłości pracę z materiałami z kamery opracowano dodatkowo mechanizm ekstrakcji zakresu temperatur z zdjęć pochodzących z kamery.

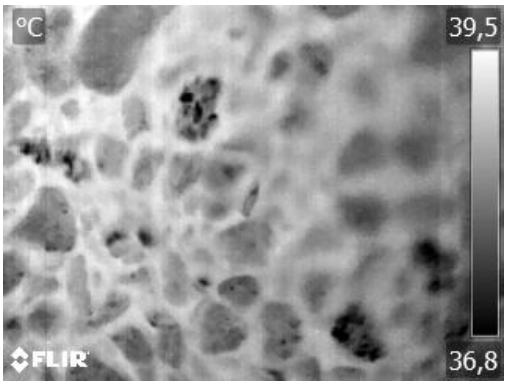
W celu konstrukcji funkcji odczytu wartości liczbowych z obrazu posłużono się gotową siecią neuronową zaprojektowaną do detekcji tekstu na zdjęciach. Zdecydowano się na użycie popularnej biblioteki *Pytesseract*. Aby poprawnie odczytać wartości z obrazu najpierw przycięto je tak by w kadrze znajdowała się tylko odczytywana liczba. Ponieważ przy eksporcie zdjęć program FLIR Tools nakłada interfejs na zdjęcia w identyczny sposób, kadrowanie obrazu jest takie same dla każdej próbki pomiarowej. Wycięte kadry są bardzo małej rozdzielczości, aby ułatwić sieci rozpoznawanie liczb zdecydowano się przeskalować je w górę. W czasie skalowania



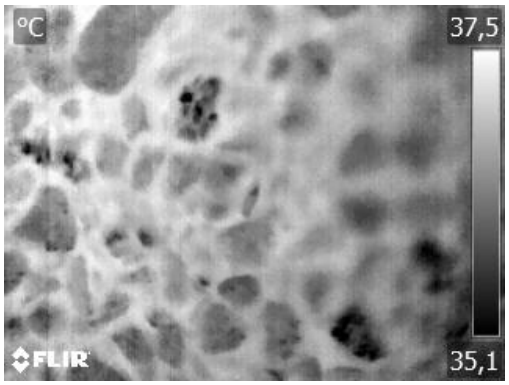
(a) Obraz z próbki 104_E5R_0



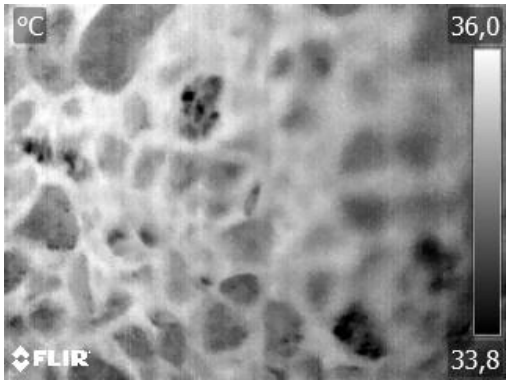
(b) Obraz z próbki 104_E5R_1



(c) Obraz z próbki 104_E5R_2



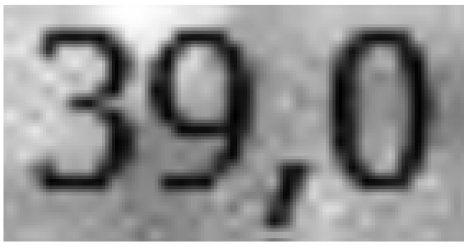
(d) Obraz z próbki 104_E5R_3



(e) Obraz z próbki 104_E5R_4

Rysunek 3.1: Zdjęcia procesu stygnięcia w przykładowej próbce 104 klasy E5R

włączono mechanizm anty aliasingu aby wyrównać krawędzie cyfr. Ponieważ używana sieć uznaje za tło kolor biały oraz poszukuje liczb w kolorze czarnym barwy na zdjęciu odwrócono. Następnie obraz poddano binaryzacji metodą *otsu*. Jest to popularna i wydajna metoda binaryzacji, jej efektywność jest maksymalna kiedy ilość pikseli tła oraz pierwszego planu jest zbliżona, dlatego poprawne kadrowanie liczb sprzyja jakości ich binaryzacji[2]. Na rysunku 3.2 przedstawiono kolejne etapy przygotowania obrazu do rozpoznania liczb. Implementację opisanego mechanizmu odczytywania zakresu temperatur ze zdjęć przedstawiono na listingu 1.



(a) Przeskalowany kadr z liczbą

(b) Kadr z liczbą po binaryzacji

Rysunek 3.2: Przygotowanie zakresu temperatur do odczytu przez sieć neuronową

3.3 Poszukiwanie zależności użytecznych w klasyfikacji

Aby móc klasyfikować dane należy zastanowić się nad cechami które je odróżniają. Na rysunku 3.3 przedstawiono porównanie stygnięcia dwóch rodzajów próbek: E5R oraz E6R. W klasyfikacji użyteczne będą dane które są unikalne dla danej klasy. Ponieważ dane stanowią serię obrazów postępującego studzenia materiału, aby wykorzystać pełnię możliwości zebranych zdjęć warto szukać cech charakterystycznych dla przebiegu procesu chłodzenia.

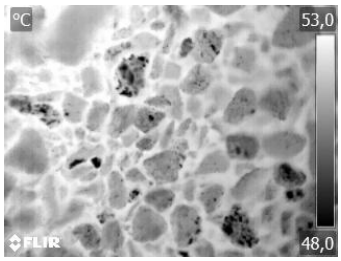
3.3.1 Wybór cech obrazu użytecznych w klasyfikacji

Po przyjrzeniu się rysunkowi 3.3 widoczne jest, że w próbce E6R temperatura ziaren zaczęła wyrównywać się szybciej. Na podstawie obserwacji zebranych danych rozpatrzono następujące możliwości obserwacji cech charakterystycznych materiałów:

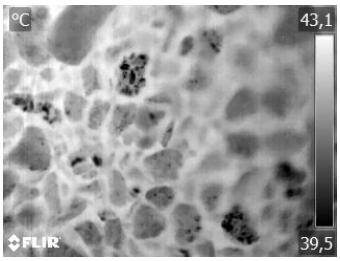
a) klasyfikacja zdjęć w całości jako macierzy danych przez złożoną sieć konwolucyjną,

```
def get_temperature_bounds(img, bounds=(((6, 24), (283, 318)),
                                        ((219, 236), (283, 318)))):
    '''Extract temperature values from FLIR UI on image.'''
    img = invert(img)
    temp_txt = []
    for bound in bounds:
        bound_img = img[slice(*bound[0]), slice(*bound[1])]
        bound_img = rescale(bound_img, 4, anti_aliasing=True)
        thr = threshold_otsu(bound_img)
        img_txt = bound_img > thr
        img_txt = Image.fromarray(img_txt)
        temp = pytesseract.image_to_string(img_txt)
        if temp is not '':
            temp = float(temp)
        else:
            temp = 0
        temp_txt.append(temp)
    return temp_txt
```

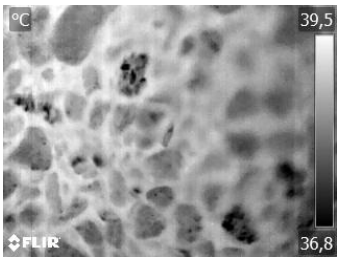
Listing 1: Funkcja języka Python do odczytywania zakresu temperatur ze zdjęć z kamery



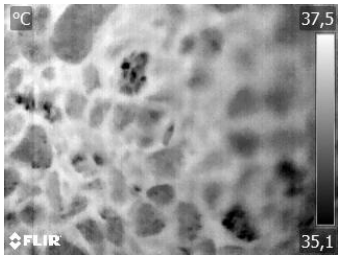
(a) Próbka 104_E5R_0



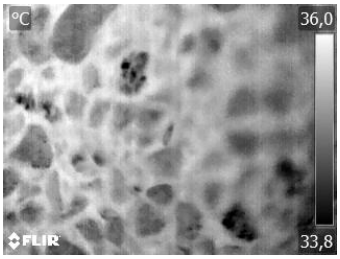
(b) Próbka 104_E5R_1



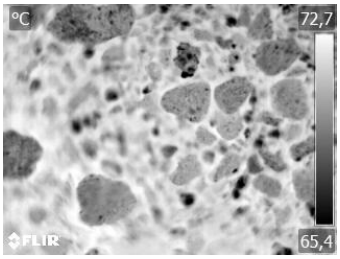
(c) Próbka 104_E5R_2



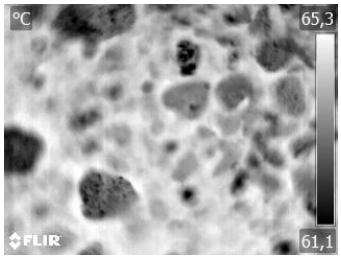
(d) Próbka 104_E5R_3



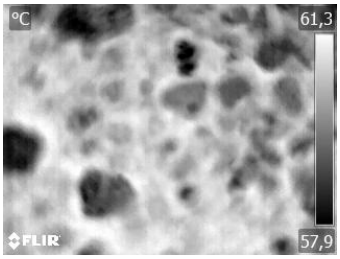
(e) Próbka 104_E5R_4



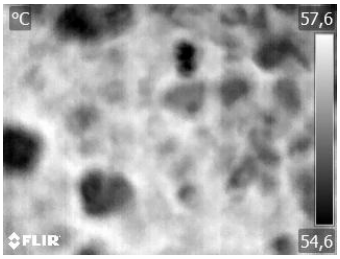
(f) Próbka 117_E6R_0



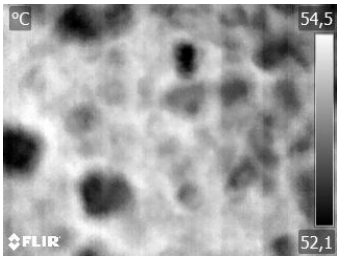
(g) Próbka 117_E6R_1



(h) Próbka 117_E6R_2



(i) Próbka 117_E6R_3



(j) Próbka 117_E6R_4

Rysunek 3.3: Porównanie procesu stygnięcia próbek klasy E5R oraz E6R

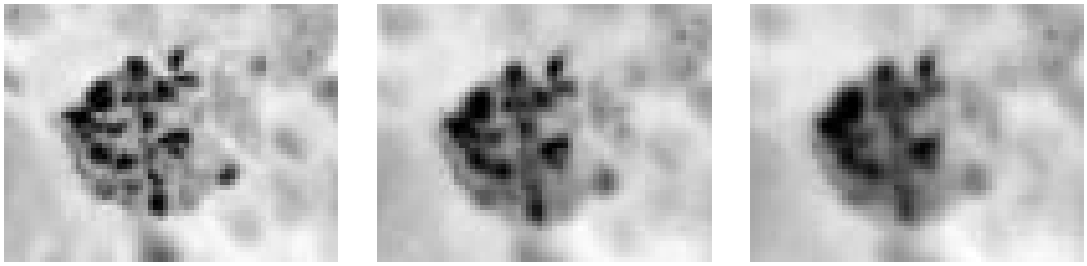
b) analiza częstotliwościowa obrazów w celu śledzenia tempa rozmycia kolejnych zdjęć,

c) użycie macierzy GLCM jako wejścia sieci neuronowych,

d) wykrywanie krawędzi ziaren i wyznaczanie reprezentacji liczbowej ich kształtów oraz powierzchni,

e) śledzenie zlewania się i zanikania małych detali na obrazie.

Wszystkie przedstawione opcje mają uzasadnienie i mogą sprawdzić się dobrze jako podstawa klasyfikacji. Należy jednak ocenić której z nich użyć w pierwszej próbie konstrukcji klasyfikatora. Metoda a), z użyciem sieci konwolucyjnych może wykorzystywać najnowsze rozwiązania w dziecinie uczenia maszynowego, jednak przy jej użyciu na przeszkodzie może stać bardzo mały rozmiar zbioru uczącego. Na niewielu zgromadzonych obrazach znajduje się wiele informacji i szumów, a złożoność jednego zdjęcia jest na tę chwilę nieproporcjonalna do wielkości zbioru danych. Pomysł ten można spróbować zrealizować po rozszerzeniu pomiarów. Kolejna opcja b) z użyciem analizy częstotliwościowej wymaga złożonych operacji matematycznych i może być wrażliwa na szumu na obrazie. Po analizie innych opcji zdecydowano, że istnieją bardziej obiecujące alternatywy. Macierz glcm (*Gray-Level Co-Occurrence Matrix*), na której może bazować opcja c), to tablica zawierająca informacje o relacjach wszystkich par pikseli na obrazie. Pozwala ona na analizę takich wartości jak: kontrast, korelacja, energia oraz homogeniczność. Jest to opcja dająca możliwość analizy dużej ilości informacji, z pewnością warta rozpatrzenia, jednak dosyć skomplikowana. Na obrazie można także wykrywać kształty ziaren za pomocą filtrów detekcji krawędzi. Opcję tę testowano przy pomocy filtra *Canny*. Krawędzie ziaren okazały się jedank trudne do wykrycia kształtów i dalszej segmentacji ze względu na małą rodzielczość oraz duże upakowanie ziaren. Operacje morfologiczne domykania kształtów powodowały bardzo duże zmiany w obrazie i zlewały ziarna. Rozwój takiego podejścia przy analizowanych obrazach wymaga zaawansowanej i ostrożnej obróbki zdjęć. Ostatnia opcja e) wynika z obserwacji detali na obrazach. Na przedstawionych zdjęciach próbek można zauważyć drobne ciemne punkty, które są obszarami o wolniejszej wymianie ciepła z otoczeniem niż reszta powierzchni ziaren. Na rysunku 3.4 przedstawiono zbliżenie na grupę takich detali, w czterech częściach procesu stygnięcia. Analizując próbki można zauważyć, że wraz ze stygnięciem ciemne punkty w grupach zlewają się, a następnie zanikają. Dodatkowo ich liczba na poszczególnych klasach materiałów jest różna. Zdecydowano się na wybór metody polegającej na śledzeniu liczby tych punktów i ich zaniku. Taka analiza wiąże się z przetwarzaniem obrazów i utworzeniem algorytmów śledzenia detali. Opcja ta wydaje się jednak obiecująca, ponieważ nawet



(a) Grupa detali w próbce 119_E5R_0 (b) Grupa detali w próbce 119_E5R_1 (c) Grupa detali w próbce 119_E5R_2

Rysunek 3.4: Zbliżenie na charakterystyczne grupy detali materiału

podczas wstępnej obserwacji próbek można dopatrywać się zależności miedzy klasami a obecnością omawianych detali. Można również zaobserwować, że w czasie stygnięcia na kolejnych zdjęciach pojawiają się sporadycznie także nowe detale. Zjawisko ich powstawania jest jednak pomijalne przy zdecydowanej tendencji do zanika i rozmywania plam, którą omówiono. Fakt pojawiania się nowych detali wzięto pod uwagę przy późniejszym procesie projektowania algorytmów ich śledzenia.

3.3.2 Wybór algorytmu detekcji ziaren

Zgodnie z rozważaniami przedstawionymi w podsekcji 3.3.1 w celu klasyfikacji ziaren zdecydowano się na obserwację ilości ciemnych, drobnych detali na obrazach. Należy więc wybrać metodę detekcji charakterystycznych punktów. W wykrywaniu omawianych detali użyteczne są algorytmy wykrywania plam na podstawie analizy pochodnych wartości na obrazie. Biblioteka Scikit-image udostępnia trzy algorytmy tego typu wykorzystujące:

a) laplasjan funkcji Gaussa,

b) różnicę funkcji Gaussa,

c) wyznacznik Hesjanu.

Algorytmy te pozwalają na wykrycie na obrazie plam o kształcie zbliżonym do kolistego, oraz oszacowanie ich promienia. Stosuje się je na przykład w analizie obrazów astronomicznych do detekcji i zliczania ciał niebieskich na zdjęciach wykonanych z użyciem teleskopów. Badane detale, przedstawione na rysunku 5, mają kształt zbliżony do kolistego, więc próba użycia rozpatrywanych algorytmów jest uzasadniona. Należy porównać dostępne warianty detekcji i wybrać algorytm, który działa najskuteczniej na zgromadzonych próbkach.

Metoda bazująca na laplasjanie funkcji Gaussa jest najdokładniejsza, ale także najwolniejsza. Funkcja Gaussa, której wykres ma charakterystyczny kształt krzywej dzwonowej jest dana wzorem 3.1, gdzie:

- σ to odchylenie standardowe,
- μ to wartość średnia.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{3.1}$$

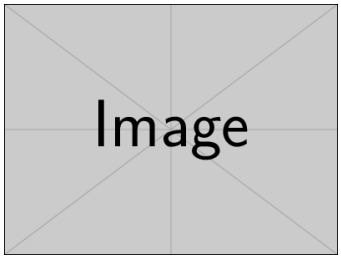
Laplasjan to operator różniczkowy drugiego rzędu. Omawiany algorytm oblicza wartości funkcji Gaussa dla coraz większego odchylenia standardowego i układa je w sześcianie. Poszukiwane plamy to lokalne maksima w tym sześcianie. Wadą tego rozwiązania jest bardzo wolne wykrywanie dużych plam z powodu złożoności obliczeniowej.

Różnica funkcji Gaussa jest metodą podobną do poprzedniej. Ponownie rozmywa ona obrazu z narastającymi odchyleniami standardowymi z użyciem funkcji Gaussa. Następnie różnice rozmytych obrazów są układane w sześcianie, gdzie maksima to plamy. Metoda ta jest szybsza i mniej dokładna od algorytmu bazującego na laplasjanie funkcji Gaussa, ale podobnie jak ona jest wolna w wykrywaniu dużych elementów.

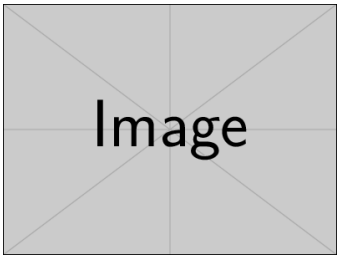
Ostatnia metoda jest najszybsza, ale najmniej dokładna. Polega ona na wyszukiwaniu maksimów w macierzy Hesjanu, jest to macierz drugich pochodnych cząstkowych. Postać takiej macierzy w n-wymiarowej przestrzeni zmiennych x przedstawia wzór 3.2. Prędkość tej metody nie zależy od wielkości wykrywanych plam, ale małe elementy mogą nie zostać przez nią wykryte.

$$H = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{3.2}$$

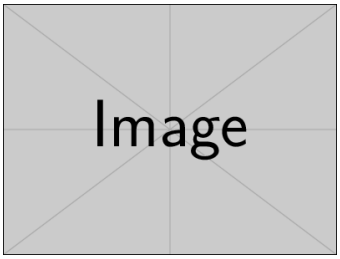
Porównanie działania wymienionych metod przedstawiono na grafice 3.5. Jak można było spodziewać się po opisie funkcji najlepsza okazała się metoda Laplasjanu funkcji Gaussa. Funkcję korzystającą z wyznacznika Hesjanu należy odrzucić, ponieważ w rozważanym przypadku istotne jest wykrywanie małych plam. Z tego samego powodu korzystne jest użycie najbardziej dokładnej funkcji. Ponieważ program nie powinien wykrywać dużych elementów nie ma ryzyka zbyt powolnych obliczeń na obszernych plamach.



(a) Laplasjan funkcji Gaussa



(b) Różnica funkcji Gaussa



(c) Wyznacznik Hesjanu

Rysunek 3.5: Porównanie bibliotecznych algorytmów wykrywania plam w obrazie

Aby wybrana funkcja laplasjanu funkcji Gaussa wykrywała, zgodnie zamierzeniami tylko małe plamy należy podać jej odpowiednie parametry, co uczyniono już na etapie porównania metod detekcji. Wywołanie omawianej funkcji ma postać przedstawioną na listingu 2. Funkcji podano dwa dodatkowe argumenty, które są istotne dla pożądanego działania. Argument `max_sigma=2` ogranicza odchylenie standardowe obliczanych funkcji Gaussa, przez co wykrywane są tylko małe elementy. Drugi argument `threshold=0.1` decyduje o poziomie powyżej jakiego punkt jest uznany za maksimum w sześcianie laplasjanów. Domyślna wartość tego argumentu `threshold=2.0` okazała się za duża, zmniejszono jej wartość aby wykrywać bardziej subtelne detale. Na podstawie opisanego wywołania metody Laplasjanu funkcji Gaussa opracowano funkcję zwracającą położenie i promienie wykrytych plam.

```
def find_blobs(img):
    '''
    Find blobs in given image and get list of their positions and
    radiuses.
    '''
    # Detect blobs with Difference of Gaussian
    blobs = blob_dog(img, max_sigma=2, threshold=0.1)
    # Get blobs radiuses from each kernel sigma
    blobs[:, 2] = blobs[:, 2] * sqrt(2)
    return blobs
```

Listing 2: Funkcja języka Python do wykrywania detali w obrazie

3.3.3 Algorytm śledzenia ziaren w serii zdjęć

Użycie funkcji bibliotecznych opisanych w podsekcji 3.3.2 pozwala na detekcję detali na pojedynczym zdjęciu. Aby wykorzystać detekcję plam w serii pięciu zdjęć które reprezentują stygnięcie jednej próbki należy opracować zestaw funkcji śledzący zmiany na obrazie. Na tym etapie prac niejednoznaczne jest jaki sposób obserwacji i zliczania detali będzie najlepszy do późniejszego prototypowania sieci neuronowej klasyfikującej klasę ziaren. Dlatego zdecydowano się na napisanie funkcji zliczania plam w serii, tak by możliwe były trzy warianty pracy algorytmu:

I. zliczanie wszystkich detali na każdym etapie stygnięcia,

II. zliczanie na kolejnych etapach stygnięcia jedynie tych detali, które były obecne od początku stygnięcia,

III. zliczanie na kolejnych etapach pozostałego procentu ziaren, które były obecne od początku stygnięcia.

W celu realizacji planu śledzenia detali utworzono funkcję języka Python. Jej kod przedstawiono na listingu 3. Aby umożliwić wariantowość algorytmu funkcja posiada opcjonalny parametr, który pozwala włączyć lub wyłączyć śledzenie jedynie ziaren, które były obecne od początku stygnięcia. Przedstawiony kod wykorzystuje funkcję wykrywania detali w zdjęciu przedstawioną na listingu 2. Jeżeli

```
def find_blob_series(imgs, only_remaining=True):
    '''
    Return list of list of blobs found in each of given images.
    '''
    stages = []
    remaining = None
    for img in imgs:
        new_blobs = find_blobs(img)
        if remaining is not None and only_remaining:
            remaining = find_remaining_blobs(new_blobs, remaining)
        else:
            remaining = new_blobs
        stages.append(remaining)
    return stages
```

Listing 3: Funkcja języka Python do śledzenia detali w serii zdjęć

opcja detekcji jedynie ziaren obecnych od początku jest aktywna zostaje wykorzystana utworzona funkcja porównania ziaren wykrytych na obecnym etapie z ziarnami obecnymi poprzednio. Po analizie każdego zdjęcia nowy zestaw ziaren staje

się zbiorem poprzednim w kolejnej iteracji. Dzieje się tak, by uwzględnić fakt że podczas zlewania się ziaren środki ciężkości i promienie detali ulegają zmianie. Dlatego kolejne grupy plam nie są zawsze porównywane z grupą z początku serii, a z zestawem z poprzedniego analizowanego zdjęcia.

Na listingu 4 przedstawiono funkcję testującą które z detali znajdują się na kolejnych etapach stygnięcia. Przyjmuje ona listy plam wykrytych w dwóch kolejnych

```
def find_remaining_blobs(new_blobs, old_blobs):
    '''
    Return list of blobs present in both lists, where blob
    is considerd same if is in proximity of 2 times it's radius.
    '''
    remaining = []
    for new_blob in new_blobs:
        yn, xn, rn = new_blob
        for old_blob in old_blobs:
            yo, xo, ro = old_blob
            if inside_circle(xn, yn, xo, yo, 2 * ro):
                remaining.append(new_blob)
    return unique(remaining)
```

Listing 4: Funkcja języka Python do wykrywania tych samych detali w kolejnych obrazach

etapach iteracji. Następnie następuje porównanie każdego ziarna z nowej i starej próbki. Jeżeli nowy punkt znajduje się w odległości do dwóch promieni od środka starej plamy to zostaje on uznany za powtarzający się. Powtarzające się punkty są dołączane do zwracanej listy. Aby zrealizować ten zamysł zaimplementowano funkcję przedstawioną na listingu 5. Kod tej funkcji wynika wprost z równania

```
def inside_circle(x, y, a, b, r):
    '''
    Return True if point (x, y) lies inside circle
    with center of (a, b) and radius r.
    '''
    return (x - a) * (x - a) + (y - b) * (y - b) < r * r
```

Listing 5: Funkcja języka Python sprawdzająca czy dany punkt leży wewnątrz podanego okręgu

matematycznego okręgu, punkty znajdują się w jego wnętrzu jeśli spełniają nierówność przedstawioną we wzorze, gdzie:

- (x, y) to współrzędne danego punktu,
- (a, b) to współrzędne środka okręgu,
- r to promień okręgu.

$$(x - a)^2 + (y - b)^2 < r^2 \tag{3.3}$$

Ponieważ wiele punktów może znajdować się blisko siebie istnieje ryzyko, że zostaną one zliczone wiele razy. Z tego powodu z listy przed jej zwróceniem z funkcji należy usunąć duplikaty. W tym celu utworzono funkcje pomocniczą widoczną na listingu 6. Najprostszym sposobem eliminacji powtarzających się wartości w języki

```
def unique(multilist):
    '''Get list without repeating values.'''
    return list(set(tuple(i) for i in multilist))
```

Listing 6: Funkcja języka Python usuwająca duplikaty z listy

Python jest konwersja listy na zbiór, który jest agregatem unikalnych nieuszeregowanych wartości. Aby zwrócić z funkcji listę należy z powrotem skonwertować zbiór na typ listowy. W naszym przypadku argument funkcji jest listą, której elementy są listami zawierającymi współrzędne oraz promień każdej plamy. Lista list jest nie podlega konwersji do zbioru ponieważ typ jej elementów nie jest hashowalny. Dlatego każdy punkt na liście zamieniono przed konwersją w zbiór na typ krotki. W ten sposób wszystkie konieczne przekształcenia są możliwe. Aby zrealizować pomysł zliczania procentu plam pozostałych w kolejnych etapach stygnięcia zaimplementowano funkcje przedstawioną na listingu. 7.

```
def percent_of_remaining_blobs_in_stages(stages):
    '''
    In each stage calculate the percentage of blobs that are
    present in the first stage and remain at given stage.
    '''
    num_of_blobs = [len(stage) for stage in stages]
    return [remaining / num_of_blobs[0] for remaining in num_of_blobs]
```

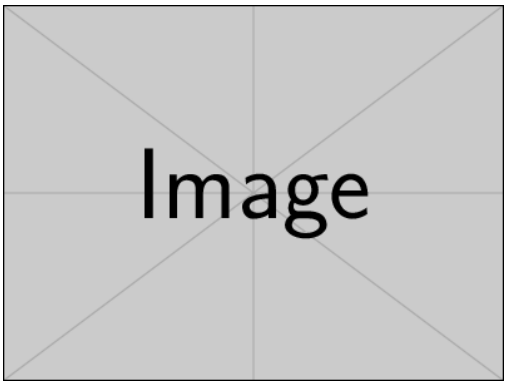
Listing 7: Funkcja języka Python obliczająca ile procent ziaren z początku stygnięcia pozostało w jego kolejnych etapach

Przyglądając się zaimplementowanym funkcjom można docenić wybór języka Python do realizacji projektu. Elastyczność języka Python, w szczególności jego list oraz wygoda dynamicznego typowania pozwoliły na szybkie i eleganckie zaimplementowanie potrzebnych funkcji. Wykorzystano bardzo zwięzły i ekspresyjny element języka nazywany *wyrażeniami listowymi*. Pozwala on na efektywne stosowanie jednolinijkowych pętli do przeglądania i modyfikowania obiektów iterowalnych. Użyteczność tego mechanizmu można zaobserwować w przedstawionych listingach. Podczas budowania funkcji dbano i ich zwięzłość, dobry podział realizowanych zadań oraz ograniczenie długości bloków kodu. Dzięki temu program zachowuje zasady dobrego programowania. Każdą funkcję opatrzono w specjalny komentarz *docstring* będący standardem w świecie języka Python. Funkcje umieszczono w odpowiednich plikach, które zawierają także przykłady ich użycia i obrazują działania oraz porównują efekty. Kod przykładów posłużył do generowania wykresów i obrazów zawartych w pracy. Pełen listing programów znajduje się w załącznikach.

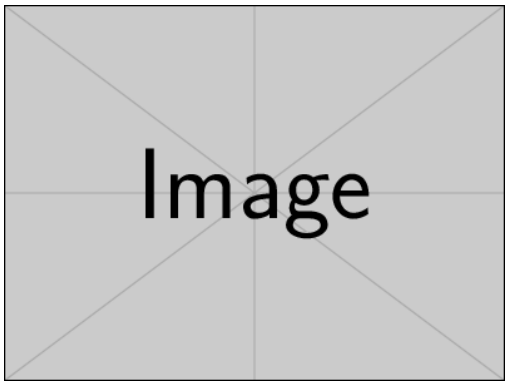
Przedstawiony zestaw funkcji daje możliwość ekstrakcji liczby plam w serii zdjęć na trzy sposoby. Efekty działania utworzonego kodu przetestowano na obrazach pomiarowych. Działanie sposobów zliczania ziaren I. oraz II. przedstawiono na rysunku 3.6. Zaznaczono na nim wykryte detale, oraz wyróżniono wśród nich te które są śledzone od początku etapu stygnięcia. Dodatkowo metodę śledzenia plam i ich zanikania przedstawiono na rysunku 3.7. Pokazuje on położenie plam wykrytych w kolejnych etapach stygnięcia. Wykryte punkty zaznaczono na obrazie z początku tego procesu. Zmieniające się kolory obrazują detale wykryte w kolejnych chwilach. Pozawala to zaobserwować przemieszczanie się środków ciężkości i promieni plam podczas stygnięcia i zlewania się. Działanie metody III., zliczającej procent detali, obecnych od początku stygnięcia zobrazowano w tabeli 3.1. Działanie zliczania przedstawiono przedstawiono w niej dla każdej rozpatrywanej klasy ziaren.

Próbka	Minuta 0	Minuta 1	Minuta 2	Minuta 3	Minuta 4
104_E5R	1.0	0.52	0.29	0.23	0.20
106_E11R	1.0	0.12	0.02	0.00	0.00
107_E6R	1.0	0.20	0.12	0.02	0.02
111_E1XP	1.0	0.05	0.01	0.01	0.01

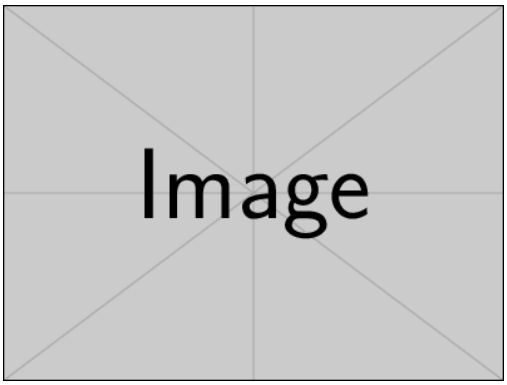
Tablica 3.1: Procent detali wykrytych w kolejnych etapach stygnięcia, które były obecne na jego początku



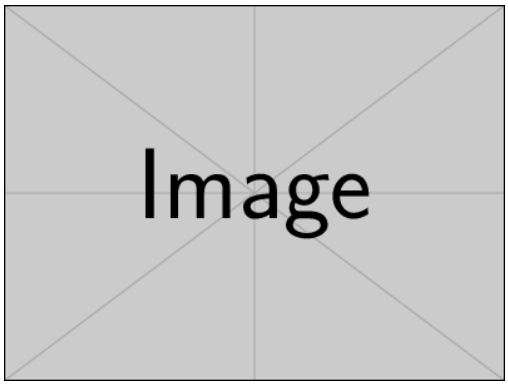
(a) Obraz z próbki 104_E5R_0



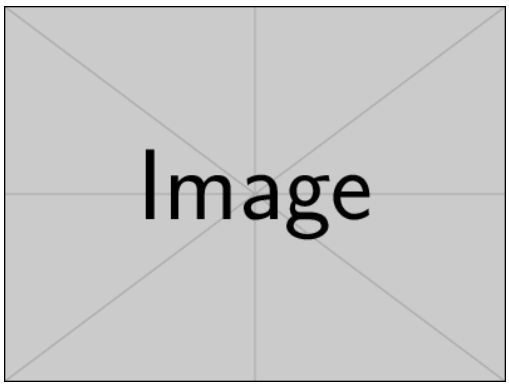
(b) Obraz z próbki 104_E5R_1



(c) Obraz z próbki 104_E5R_2

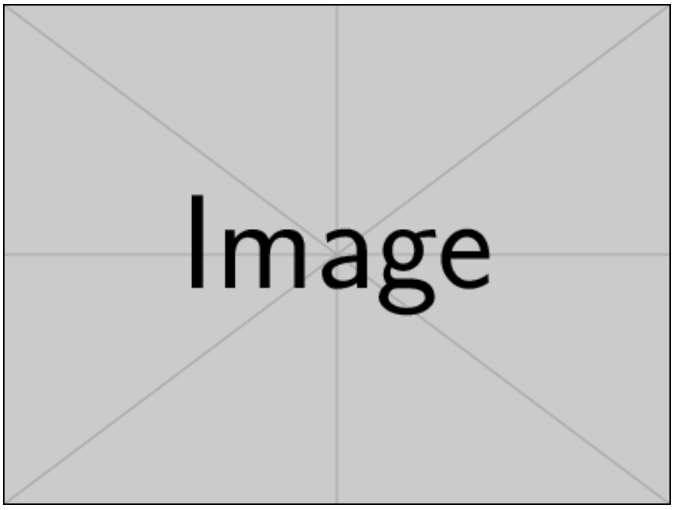


(d) Obraz z próbki 104_E5R_3



(e) Obraz z próbki 104_E5R_4

Rysunek 3.6: Zliczanie śledzonych detali w próbce dwoma sposobami



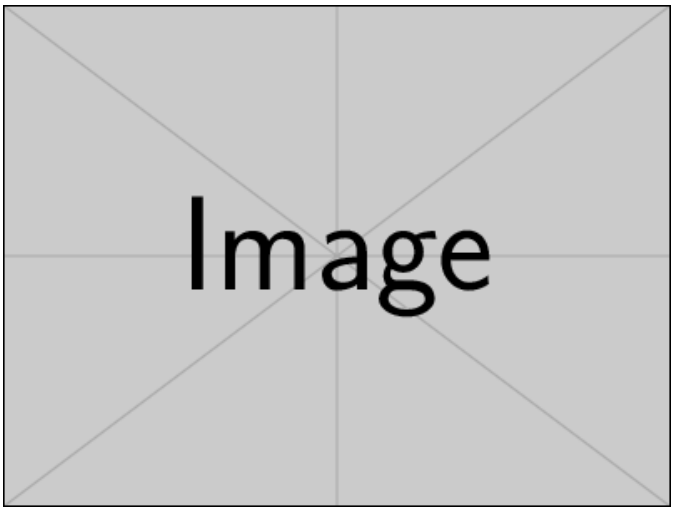
Rysunek 3.7: Wykryte detale, śledzone od początku stygnięcia, zaznaczone na pierwszym obrazie w serii pomiarowej

3.3.4 Wizualizacja zebranych cech i ocena ich użyteczności

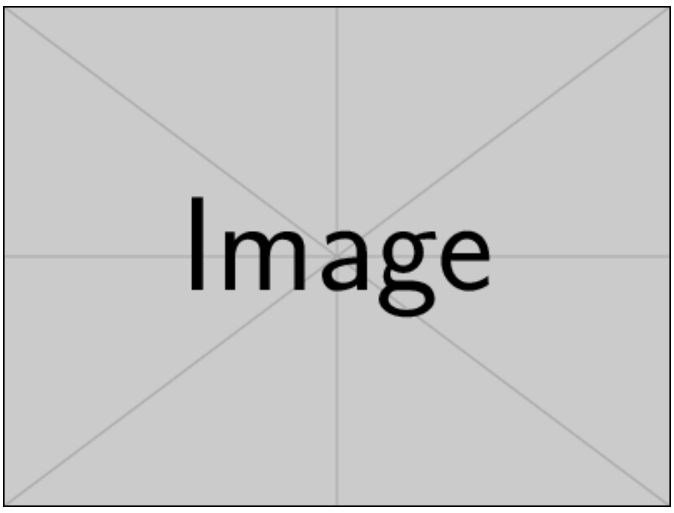
W podsekcji 3.3.3 przedstawiono opracowane algorytmy śledzenia ilości detali w rozpatrywanych obrazach stygnięcia ziaren rud miedzi. Opracowane metody mają sens przy klasyfikacji, jeśli pozyskane cechy są charakterystyczne dla rozpatrywanych klas. Aby oszacować użyteczność pozyskanych cech należy przeprowadzić ich wizualizację. W tym celu stworzono wykresy ilości wykrytych plam dla poszczególnych metod ich zliczania.

Rysunek 3.8 przedstawia wykres procesu zanikania plam dla wariantu z wykrywaniem wszystkich detali na każdym etapie stygnięcia. Wykreślone krzywe dla różnych klas materiałów przecinają się, w wykreślonych wartościach trudno na pierwszy rzut oka dojrzeć jakiekolwiek zależności i uporządkowanie. Ilość charakterystycznych detali może być cechą rud miedzi, ze względu na ich różny skład, jednak w analizowanym przypadku nie jest to cecha dająca nadzieję na dobre wyniki klasyfikacji. Powodem takiej sytuacji jest duża losowość tego typu danych. Niezależnie od własności materiału, jego ułożenie podczas pomiaru jest przypadkowe co ma wpływ na bezwzględną liczbę zliczonych detali. Na ilość widocznych plam wpływa również sposób ustawienia ostrości kamery. Po przyjrzeniu się krzywym można jednak domniemywać, że pewną cechą charakterystyczną jest nachylenie krzywych. Wskazuje to, że bardziej widoczne mogą być względne cechy czasowe, a nie bezwzględna liczba zliczonych detali.

Drugi sposób zliczania ziaren przedstawiono na rysunku 3.9. W tym przypadku zliczano jedynie detale obecne na obrazie od początku stygnięcia. Ten wariant śledzenia ilości plam daje dużo lepsze perspektywy na klasyfikację ziaren. Dla



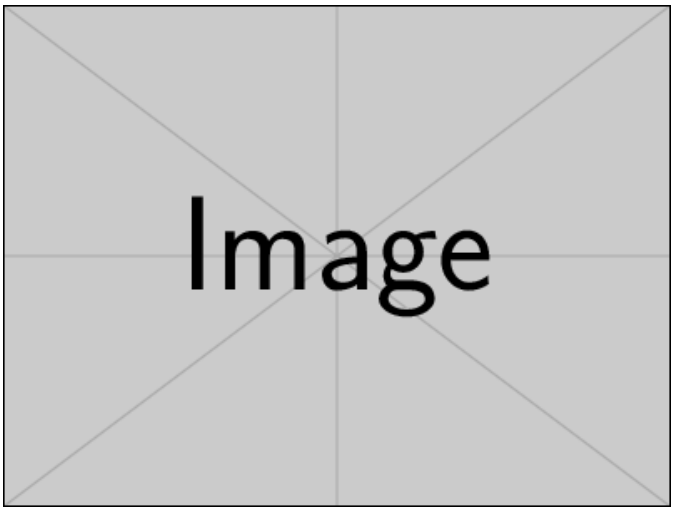
Rysunek 3.8: Ilość wykrytych detali na każdym etapie stygnięcia, w wariancie zliczającym wszystkie plamy



Rysunek 3.9: Ilość wykrytych detali na każdym etapie stygnięcia, w wariancie zliczającym wszystkie plamy

różnych typów próbek widoczne są charakterystyczne przebiegi krzywych. Obserwacja dotycząca nachylenia krzywych staje się bardziej uzasadniona, pochodne poszczególnych klas wydają się zbliżone. Wyeliminowanie pojawiających się detali pomogła w obserwacji procesu stygnięcia i zmniejszyła czynniki losowe. Na podstawie analizy nagrań można domniemywać, że nowe ziarna pojawiały się w wyniku zaobserwowanych delikatnych ruchów materiału. Te mogły wynikać z drgań stanowiska pomiarowego, istotnym czynnikiem może być także mała głębia ostrości obiektywu. W późniejszych etapach stygnięcia na obrazach pojawia się także coraz więcej szumów, które mogą zostać wykryte przez algorytm. Śledzenie detali, które znajdują się na obrazie od początku eliminuje ten problem.

Wyniki ostatniej metody polegającej na procentowym ustaleniu zaniku ilości detali została przedstawiona na rysunku 3.10. Widoczne jest odseparowanie róż-



Rysunek 3.10: Ilość wykrytych detali, przy procentowym zliczaniu detali, które występowały w serii od początku stygnięcia

nych klas rud miedzi, różne typy próbek charakteryzują się odmiennymi postępami rozmycia detali w czasie. Śledzenie ilości detali w stosunku względnym dało najlepsze rezultaty oraz pozwoliło na pewną normalizację danych, która była trudna do realizacji przy bezwzględnym zliczaniu wszystkich plam na obrazie. Należy także zauważyć, że przedstawione krzywe przedstawione na rysunkach 3.9 oraz 3.10 mają kształt zbliżony do krzywych eksponencjalnych. Jest to obserwacja wskazująca, że pozyskane cechy dobrze oddają naturę procesu stygnięcia. Proces opadania temperatury ciał opisuje *prawo stygnięcia Newtona*, które ma postać równania różniczkowego, przedstawionego we wzorze 3.4, gdzie:

- $T(t)$ to funkcja temperatury ciała w czasie,

- T_R to temperatura otocznia,
- k to współczynnik liczbowy, charakterystyczny dla danego ciała.

$$\frac{dT\left(t\right)}{dt} = -k\left(T\left(t\right) - T_R\right) \tag{3.4}$$

Rozwiązanie równania przedstawia wzór 3.5, jak widać ma ono charakter eksponencjalny, do którego zbliżone są wykresy na rysunkach 3.9 oraz 3.10.

$$T(t) - T_R = \Delta T(t) = \Delta T(0)\ e^{-kt} \tag{3.5}$$

Analiza pozyskanych cech wskazuje że istnieje możliwość wykorzystania ich do klasyfikacji rud miedzi. Dalsza ocena wyników pracy będzie zależała od jakości działania sieci neuronowej stworzonej do rozpoznawania przygotowanych danych.

Rozdział 4

Prototypowanie sieci neuronowej klasyfikującej ziarna miedzi

4.1 Budowa prototypu sieci neuronowej

Po zakończeniu procesu budowy i eksploracji danych można przystąpić do budowy sieci neuronowej odpowiedzialnej za rozpoznawanie ziaren rud miedzi. Przedstawiony problem jest wymaga klasyfikacji wieloklasowej wielowymiarowych danych. Problemy tego typu można rozwiązywać zarówno za pomocą klasycznych algorytmów uczenia maszynowego jak i uczenia głębokiego. Zdecydowano się na wybór sieci neuronowych do klasyfikacji ziaren. Jest to rozwiązanie najbardziej nowoczesne i elastyczne. Zgodnie z opisem narzędzi programistycznych w podsekcji 2.4.3, proces budowania sieci oparto na interfejsie Keras, z zapleczem TensorFlow.

4.1.1 Dobór struktury sieci

Kluczowym czynnikiem decydującym o architekturze sieci jest posiadany zbiór danych. Wejściowy zestaw cech klas jest pięciowymiarowy, do klasyfikacji takich danych wystarczająca powinna być standardowa sieć neuronowa. Nie ma potrzeby używania bardziej złożonych sieci rekurencyjnych, czy konwolucyjnych.

Projektowanie sieci neuronowych wymaga podjęcia szeregu wyborów, wśród nich można wyróżnić następujące decyzje dotyczące:

- ilości warstw sieci,
- typu warstw w sieci,
- ilości neuronów w poszczególnych warstwach sieci,
- rodzajów funkcji aktywacji w neuronach poszczególnych warstw sieci.

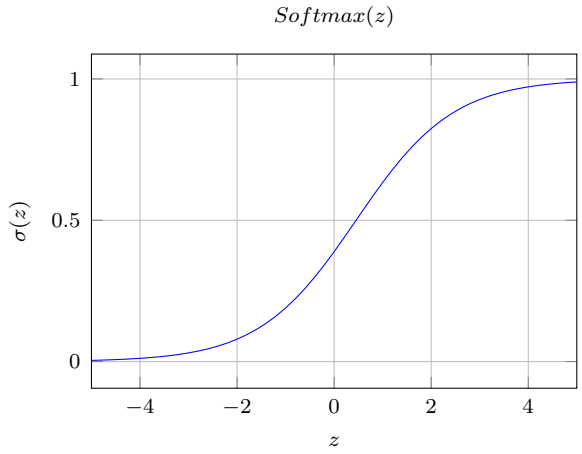
Definiując model należy rozpatrzeć przedstawione cechy budowanej sieci. Nie istnieje jednoznaczny sposób bezpośredniego określenia najlepszego klasyfikatora dla danego problemu. Przy projektowaniu sieci należy kierować się przesłankami teoretycznymi, doświadczeniem oraz testując i porównując różne rozwiązania.

Konstrukcję klasyfikatora rozpoczęto od wyboru ilości warstw. W standardowej konstrukcji każda sieć posiada warstwę wejściową oraz wyjściową, których parametry są związane odpowiednio z wektorem cech oraz reprezentacją klas na wyjściu sieci. Ponadto do aproksymacji dowolnej funkcji nieliniowej wystarczająca jest sieć składająca się z dwóch warstw. Nie oznacza to jednak, że w praktyce dwie warstwy wystarczą by rozwiązać dowolny problem. Aby zwiększać efektywność sieci pomiędzy warstwą wejściową, wyjściową umieszcza się zestaw warstw nazywanych *ukrytymi*. Zazwyczaj posiadają one najwięcej neuronów spośród warstw w sieci. Mimo że jedna warstwa ukryta pozwala teoretycznie na rozwiązanie dowolnego problemu, w praktyce opłacalne jest stosowanie większej ilości warstw, co poparte jest licznymi przykładami[3]. Biorąc pod uwagę złożoność rozpatrywanego problemu zdecydowano się na strukturę czterowarstwową. Jest to rozmiar często spotykany w tego typu sieciach, późniejsze testy pokazały, że jest to ilość warstw dająca dobre rezultaty. Sprawdzono, że zastosowanie jednej warstwy ukrytej powodowało pogorszenie działania sieci.

Kolejnym krokiem projektowania sieci jest decyzja o ilości neuronów w poszczególnych warstwach. W przypadku pierwszej i ostatniej warstwy jest to wartość prosta do określenia. Zaleca się, aby wejście sieci miało liczbę neuronów równą wymiarowi używanego wektora cech, który w rozpatrywanym przypadku ma długość równą pięć. Ostatnia warstwa powinna mieć tyle neuronów ile wynosi liczba rozpoznawanych klas, tak by każde wyjście sieci oznaczało prawdopodobieństwo przynależności próbki do odpowiedniej klasy. W analizowanym zbiorze ziaren znajdują się cztery typy rud miedzi i tyle neuronów powinno znajdować się w warstwie wyjściowej sieci. W warstwach ukrytych umieszczono większą liczbę neuronów, eksperymentalnie stwierdzono, że sieć osiąga dobre wyniki dla 256 neuronów w pierwszej warstwie ukrytej i 128 w drugiej. Zmniejszenie liczby neuronów o połowę w kolejnej warstwie ukrytej jest często spotykanym zabiegiem.

Następnie rozpatrzono dostępne funkcje aktywacji poszczególnych warstw. Dla warstwy wyjściowej należy wybrać funkcję o wartościach od zera do jeden. Przy klasyfikacji zazwyczaj stosuje się funkcję softmax, jej wykres przedstawiono na rysunku 4.1. W przypadku pozostałych warstw wybór funkcji aktywacji jest mniej oczywisty. Rozpatrzono trzy rodziny funkcji:

- funkcję sigmoid,
- funkcje z grupy jednostek liniowych (ang. *linear unit*):
  - ReLU (ang. *Rectified Linear Unit*),

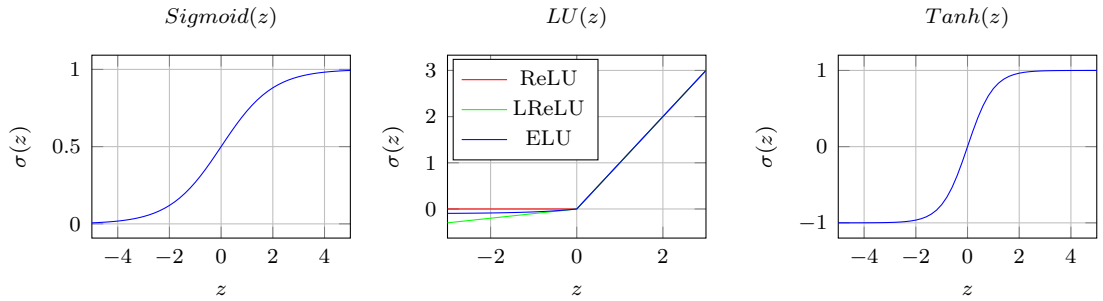


Rysunek 4.1: Funkcja aktywacji softmax

  - ReLU z wyciekiem (ang. *Leaky Rectified Linear Unit)*
  - ELU (ang. *Exponential Linear Unit*),

- funkcję tangens hiperboliczny.

Rozpatrywane funkcje aktywacji przedstawiono na rysunku 4.2. Funkcja sigmoid



(a) Funkcja aktywacji tangens hiperboliczny

(b) Rodzina funkcji aktywacji jednostek liniowych

(c) Funkcja aktywacji tangens hiperboliczny

Rysunek 4.2: Dostępne funkcje aktywacji rozpatrywane do użycia w warstwach ukrytych

jest często stosowaną, klasyczną funkcją aktywacji. Jej wadą jest jednak zanikanie wartości pochodnej. Obecnie częściej stosowane są funkcję z rodziny jednostek liniowych, szczególnie z wyciekiem, czyli małymi wartościami dla ujemnych argumentów. Przetestowano również funkcję tangens hiperboliczny, która ma kształt

podobny do funkcji sigmoid. Mimo, że obecnie funkcje jednostek liniowych są najbardziej popularne w rozpatrywanym przypadku sieć osiągała najlepsze wyniki dla funkcji tangens.

Opisany model sieci neuronowej należy zdefiniować za pomocą interfejsu Keras. Do inicjalizacji sieci o standardowej liniowej strukturze służy funkcja `Sequential()`, która może przyjąć listę warstw w sieci. Funkcja `Dense(units)` tworzy warstwę łącząca każdy neuron z każdym wyjściem poprzedniej warstwy. Jej parametry definiuje ilość neuronów w warstwie oraz ich funkcję aktywacji. Funkcję języka Python implementują opisywaną sieć, za pomocą przedstawionych elementów biblioteki Keras, przedstawiono na listingu 8.

```
def default_grain_classifier_model():
    '''
    Get default uncompiled model for grain classifcation,
    based on 5 step cooling process using number of blobs.
    '''
    model = keras.Sequential([
        keras.layers.Dense(5, activation='tanh'),
        keras.layers.Dense(256, activation='tanh'),
        keras.layers.Dense(128, activation='tanh'),
        keras.layers.Dense(4, activation='softmax')
    ])
    return model
```

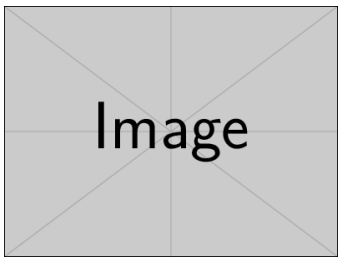
Listing 8: Funkcja języka Python definiująca model sieci neuronowej

4.1.2 Trening sieci neuronowej

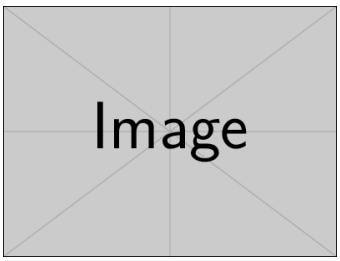
Kolejnym krokiem w budowie klasyfikatora ziaren jest trening sieci neuronowej. Przed rozpoczęciem uczenia sieci należy podzielić dostępne dane na zbiór treningowy oraz testowy. Aby ułatwić podział Biblioteka Scikit-learn udostępnia funkcję `train_test_split(*arrays, **options)`, która zwraca podane zbiory, podzielone na części treningowe i testowe. Odpowiednie wykorzystanie funkcji wymaga użycia jej parametrów opcjonalnych. Argument `stratify` sprawia, że podzielone zbiory zawierają takie same proporcje klas. Taka metoda podziału nazywana jest *losowaniem warstowym*. Uaktywnienie tej opcji jest szczególnie ważne, ze względu na mały rozmiar posiadanego zbioru danych. Gdyby dane dzielić w pełni przypadkowe istniałoby ryzyko nadmiernej reprezentacji klasy w danej grupie oraz jej braku w innej. Kolejnym istotnym parametrem jest `test_size`, który decyduje o rozmiarze zbioru testowego. Ponieważ w zbiorze są trzy egzemplarze każdej

klasy odpowiednie jest wydzielenie do testów jednej trzeciej próbek. Ostatni parametr `random_state`, to ziarno generatora losowego podziału zbiorów. Aby móc porównywać działanie sieci pomiędzy wielokrotnymi uruchomieniami programu należy wyeliminować z niego czynniki przypadkowe i podać funkcji stałe ziarno, co zapewni powtarzalny podział zbioru. Oczywiście wydzielenie w narzucony sposób zbioru testowego, szczególnie w przypadku tak małej ilości danych, nie daje obiektywnej oceny modelu. Jest on jednak wystarczający do budowy i testowania pierwszego prototypu sieci. W sekcji 4.2 przedstawiono konstrukcję i wyniki działania bardziej miarodajnego procesu walidacji.

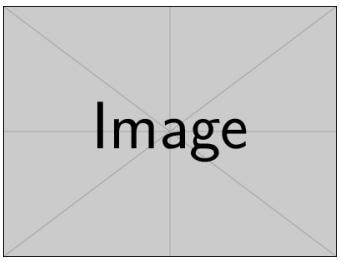
Aby móc do niego przystąpić do treningu należy określić parametry uczenia sieci. Konfiguracja tego procesu odbywa się przez wywołanie metody `Compile()`. Metoda przyjmuje parametry algorytmu optymalizacji sieci, miary błędu oraz metryki. Argument `optimizer` przyjmuje nazwę stosowanego algorytmu uczenia sieci. Rozważono dwa popularne metody treningu: sgd (ang. *stochastic gradient descent)* oraz adam (ang. *adaptive moment estimation*). Metoda sgd jest najbardziej popularnym i podstawowym sposobem uczenia, jednak algorytm adam, jest rozią-zaniem nowszym, polecanym w problemach klasyfikacji. Zgodnie z tymi przesłankami optymalizator adam okazał się dawać najlepsze rezultaty i to on został wybrany do uczenia sieci. Parametr `loss` przyjmuje nazwę sposobu liczenia błędu w sieci. Przykładem miary błędu używanej w sieciach neuronowych jest popularny błąd średniokwadratowy. Dla problemów klasyfikacji lepszy jest jednak błąd obliczany metodą *rzadkiej kategoryzacyjnej entropii krzyżowej* (ang. *sparse categorical crossentropy*). Jest to złożona metoda wykorzystująca funkcję logarytmiczną. Metoda kategoryzacyjnej entropii okazała się najlepszą funkcją błędu dla analizowanego przypadku. Ostatni parametr metryki to wartość obliczana w celu oszacowania poprawności działania sieci. Wybrano standardową metrykę dokładności sieci, czyli stosunku poprawnie zakwalifikowanych próbek do wszystkich analizowanych. Ostatnim uczenia etapem jest realizacja treningu sieci, dokonywana za pomocą metody `model.fit()`. Metoda przyjmuje argumenty zbioru treningowego wraz z zestawem etykiet oraz liczbę epok treningu, zwraca obiekt zawierający przebieg treningu. Na rysunku 4.3 przedstawiono dokładność i błąd w kolejnych epokach treningu. Na ich podstawie określono liczbę że odpowiednia liczba epok wynosi 300. Opisany plan treningu sieci realizuje kod przedstawiony na listingu 9. Działanie sieci można sprawdzić na zbiorze testowym. Należy mieć na uwadze, że przy małej ilości danych i przedstawiony podziale na zbiory treningowy oraz testowy nie będzie to test miarodajny. Jest on jednak akceptowalny przy pierwszych testach sieci podczas jej konstrukcji. Wyniki działania sieci przedstawia tabela 4.2. Przedstawione wartości błędu oraz dokładności potwierdzają przypuszczenia na temat użyteczności różnych metod zliczania ziaren, które przedstawiono w podsekcji 3.3.4.



(a) Przebieg treningu sieci dla metody zliczania wszystkich ziaren



(b) Przebieg treningu sieci dla metody zliczania śledzonych ziaren



(c) Przebieg treningu sieci dla metody procentowego zliczania śledzonych ziaren

Rysunek 4.3: Przebieg treningu sieci dla różnych metod zliczania ziaren

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
    stratify = y,
    test_size = 0.33,
    random_state = 1)

model = default_grain_classifier_model()

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=256)
```

Listing 9: Kod treningu sieci neuronowej klasyfikującej ziarna miedzi

metoda zliczania ziaren	wskaźnik	
	błąd	dokładność
zliczanie wszystkich ziaren	1.45	0.25
zliczanie śledzonych ziaren	7.50	0.5
procentowe zliczanie śledzonych ziaren	0.63	0.75

Tablica 4.1: Wskaźniki oceny działania sieci na zbiorze testowym

4.2 Walidacja i ocena działania sieci

Po stworzeniu prototypu klasyfkiatora przedstawionego w rozdziale 4.1.1 należy przeprowadzić walidację zbudowanej sieci neuronowej. Najpopularniejszą metodą miarodajnego testu sieci jest *k-krotny sprawdzian krzyżowy* (ang. *k-fold cross-validation*). Metoda ta polega na podziale dostępnych danych na k części, i kolejnym wydzielaniu jednej z części danych jako zbioru testowego. Walidacje powtarza się k-krotnie, tak by każda część była wykorzystana jako dane testowe. Na końcu testu wyniki są uśrednianie, przez co dają one dobry pogląd na ogólną zdolność sieci do generalizacji rozwiązywanego problemu.

Biblioteka keras nie posiada wbudowanej funkcji realizującej sprawdzian krzyżowy, dlatego należy przygotować ją samodzielnie. W tym celu użyto biblioteki Scikit-learn, która oferuje funkcje podziału zbiorów danych. Jedną z nich jest funkcja zwracająca iterowalny zestaw k-krotnych podziałów zbioru danych. Jak przedstawiono w sekcji 4.1.2, ze względu na mały rozmiar zbioru danych, przy podziale należy zastosować losowanie warstwowe. Na listingu 10 przedstawiono funkcję pozwalającą na walidację modelu klasyfikatora biblioteki keras z użyciem sprawdzianu krzyżowego. Funkcja przyjmuje argumenty w postaci skompilowanego modelu sieci oraz zbioru danych i etykiet. Zwracana jest macierz, w której rzędy oznaczają wyniki testów dla kolejnych podziałów zbioru danych. Sposób wyko-

```
def network_cross_validation(model, X, y):
    '''Compute cross validation fold scores for given keras model.'''
    eval_scores = []
    for train_index, test_index in
        StratifiedKFold(n_splits = 3).split(X, y):

        x_train, x_test= X[train_index], X[test_index]
        y_train, y_test= y[train_index], y[test_index]

        model.fit(x_train, y_train, epochs=300, verbose=0)
        eval_scores.append(model.evaluate(x_test, y_test))
    return eval_scores
```

Listing 10: Funkcja języka Python definiująca model sieci neuronowej

rzystania zbudowanej funkcji walidacji modelu przedstawia listing 11. Na końcu procesu oceny sieci wyniki kolejnych kroków sprawdziany krzyżowego należy uśrednić.

```
X = np.array(X)
y = np.array(y)

model = default_grain_classifier_model()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

eval = network_cross_validation(model, X, y)

print('Folds scores: (loss, acc)\n', eval)
eval = np.array(eval)
print('Cross validation mean score (loss, acc):\n',
      eval.mean(axis=0), '\n')
```

Listing 11: Wykorzystanie funkcji sprawdzianu krzyżowego do oceny działania sieci

metoda zliczania ziaren	wskaźnik	
	błąd	dokładność
zliczanie wszystkich ziaren	6,59	0,41
zliczanie śledzonych ziaren	3,99	0,50
procentowe zliczanie śledzonych ziaren	1,37	0,91

Tablica 4.2: Wskaźniki oceny działania sieci uzyskane metodą sprawdzianu krzyżowego

Rozdział 5

Podsumowanie

Spis rysunków

Spis tablic

Spis listingów

Bibliografia

[1] François Chollet, i in. Keras. `https://github.com/fchollet/keras`, 2015.

[2] Bülent Sankur Mehmet Sezgin. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–165, 2004.

[3] Rober J Marksll Russell Reed. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. The MIT Press.