

**COLLEGIUM WITELONA**  
**Uczelnia Państwowa**

**Wydział Nauk Technicznych i Ekonomicznych**  
**Kierunek Inżynieria produkcji i logistyki**  
**Specjalność Przemysł 4.0**

**KAROL ZYGADŁO**

**Rozpoznawanie wad produkcyjnych z wykorzystaniem uczenia głębokiego**

**Praca dyplomowa magisterska**  
**napisana pod kierunkiem**  
**dr hab. inż. Robert Burduk**

**Legnica 2023 rok**

# **Spis treści**

<b>1 Wstęp</b>	<b>4</b>
1.1 Cel i motywacja pracy . . . . .	4
1.2 Zawartość pracy . . . . .	5
<b>2 Wybrane podstawy teoretyczne pracy</b>	<b>7</b>
2.1 Uczenie głębokie . . . . .	7
2.2 Zastosowania uczenia głębokiego . . . . .	8
2.3 Różnice między uczeniem głębokim a innymi technikami uczenia maszynowego . . . . .	8
2.4 Sieci konwolucyjne . . . . .	10
2.5 Jak opracowywane są sieci CNN? . . . . .	10
2.6 Wady produkcyjne i ich rodzaje . . . . .	13
2.7 Istniejące rozwiązania . . . . .	16
2.8 Wybrany język programowania - Python . . . . .	17
<b>3 Wymagania funkcjonalne oraz niefunkcjonalne</b>	<b>19</b>
3.1 Wymagania funkcjonalne . . . . .	19
3.2 Wymagania niefunkcjonalne . . . . .	19
<b>4 Projekt rozwiązania</b>	<b>21</b>
4.1 Wybór sieci konwolucyjnej . . . . .	21
4.2 Korzyści z zastosowania uczenia głębokiego . . . . .	21
4.3 Ogólny zarys rozwiązania . . . . .	22
4.4 Import bibliotek . . . . .	22
4.5 Wczytywanie danych . . . . .	24
4.6 Ładowanie danych . . . . .	24
4.7 Wstępne przetwarzanie danych . . . . .	25
4.8 Podział danych na zestawy treningowe, walidacyjne i testowe . . . . .	26
<b>5 Implementacja</b>	<b>28</b>
5.1 Budowa modelu sieci neuronowej . . . . .	28
5.2 Trenowanie modelu . . . . .	29

5.3	Wczytywanie przykładowego obrazu . . . . .	31
5.4	Przeskalowanie obrazu . . . . .	31
5.5	Predykcja klasy obrazu . . . . .	32
5.6	Interpretacja wyników . . . . .	32
5.7	Zapisywanie modelu . . . . .	33
5.8	Wczytywanie modelu . . . . .	33
<b>6</b>	<b>Badania</b>	<b>35</b>
6.1	Przedmiot badań . . . . .	35
6.2	Cele badań . . . . .	35
6.3	Środowisko badawcze . . . . .	35
6.4	Metoda badawcza . . . . .	36
6.5	Trenowanie modelu . . . . .	36
6.6	Walidacja . . . . .	37
6.7	Testowanie . . . . .	37
6.8	Materiały badawcze . . . . .	38
<b>7</b>	<b>Wyniki i analiza badań</b>	<b>46</b>
7.1	Wyniki dla modelu uczonego na 25 zdjęciach . . . . .	46
7.2	Wyniki dla modelu uczonego na 50 zdjęciach . . . . .	49
7.3	Wyniki dla modelu uczonego na 100 zdjęciach . . . . .	52
7.4	Wyniki dla modelu uczonego na 250 zdjęciach . . . . .	55
7.5	Wyniki dla modelu uczonego na 500 zdjęciach . . . . .	58
7.6	Podsumowanie wyników . . . . .	61
<b>8</b>	<b>Podsumowanie</b>	<b>63</b>
8.1	Wnioski . . . . .	63
8.2	Zrealizowane cele . . . . .	64
<b>Bibliografia</b>		<b>67</b>
<b>Spis rysunków</b>		<b>68</b>
<b>Spis tabel</b>		<b>70</b>

## **Rozdział 1. Wstęp**

### **1.1 Cel i motywacja pracy**

Celem niniejszej pracy magisterskiej jest wykorzystanie i zastosowanie metod uczenia głębokiego w celu rozpoznawania wad produkcyjnych na podstawie analizy zdjęć. Ze względu na coraz większe zapotrzebowanie na szybkie i efektywne rozwiązania w zakresie kontroli jakości, zastosowanie technik sztucznej inteligencji staje się niezbędne dla przemysłu. W szczególności, wykorzystanie uczenia głębokiego, jako jednego z najbardziej zaawansowanych podejść w dziedzinie sztucznej inteligencji, pozwala na znaczące zwiększenie skuteczności wykrywania wad produkcyjnych.

Motywacją do podjęcia tego tematu była chęć eksploracji i zrozumienia nowoczesnych technologii związanych z uczeniem maszynowym oraz ich praktyczne zastosowanie w kontekście przemysłowym. Wadliwe komponenty mogą prowadzić do znaczących strat finansowych oraz wpływać negatywnie na reputację przedsiębiorstwa. Automatyczne wykrywanie wad na wczesnym etapie procesu produkcyjnego może przyczynić się do zwiększenia efektywności, zmniejszenia kosztów oraz ograniczenia ilości produktów o niskiej jakości trafiających do odbiorców.

W ramach pracy magisterskiej opracowany zostanie system, który będzie w stanie analizować zdjęcia przedmiotów i automatycznie klasyfikować je jako wadliwe lub prawidłowe. System ten będzie oparty na technikach uczenia głębokiego, takich jak konwolucyjne sieci neuronowe, które są obecnie szeroko stosowane w różnych dziedzinach analizy obrazów. Praca będzie obejmować zarówno teorię, jak i praktyczne aspekty projektowania oraz implementacji.

W celu zilustrowania koncepcji i metod przedstawionych w pracy, zostanie przeprowadzone eksperymentalne zastosowanie opracowanego systemu do wybranego zbioru zdjęć re-

prezentujących obiekty z wadami produkcyjnymi oraz prawidłowymi komponentami. Wyniki tego eksperymentu posłużą jako dowód na skuteczność zastosowanego podejścia oraz jako punkt wyjścia do dalszej dyskusji na temat potencjalnych ulepszeń i przyszłych kierunków rozwoju.

## 1.2 Zawartość pracy

W niniejszej pracy przedstawiono zagadnienia związane z rozpoznawaniem wad produkcyjnych z wykorzystaniem uczenia głębokiego. Poniżej przedstawiono opis zawartości pracy według kolejnych rozdziałów:

- **Rozdział 1 - Wstęp:** W tym rozdziale omówiono cel i motywację pracy oraz przedstawiono zawartość pracy.
- **Rozdział 2 - Wybrane podstawy teoretyczne pracy:** W drugim rozdziale omówiono podstawy teoretyczne pracy, takie jak uczenie głębokie, sieci konwolucyjne, rodzaje wad produkcyjnych oraz istniejące rozwiązania związane z kontrolą jakości. Przedstawiono także język programowania wykorzystany w projekcie.
- **Rozdział 3 - Wymagania funkcjonalne oraz niefunkcjonalne:** W tym rozdziale przedstawiono wymagania funkcjonalne i niefunkcjonalne.
- **Rozdział 4 - Projekt systemu:** W tym rozdziale omówiono proces projektowania i implementacji wybranej sieci konwolucyjnej. Przedstawiono korzyści wynikające z zastosowania uczenia głębokiego oraz ogólny zarys proponowanego rozwiązania. Opisano również importowanie potrzebnych bibliotek, wczytywanie, ładowanie oraz wstępne przetwarzanie danych. Na koniec, przedstawiono podział danych na zestawy treningowe, walidacyjne i testowe.
- **Rozdział 5 - Implementacja:** W czwartym rozdziale omówiono budowę modelu sieci neuronowej oraz proces trenowania i testowania modelu na przykładach. Przedstawiono także sposób zapisywania i wczytywania modelu.
- **Rozdział 6 - Badania:** W piątym rozdziale opisano przedmiot, cele oraz metodę badań. Przedstawiono również środowisko badawcze oraz przebieg badań.

- **Rozdział 7** - Wyniki i analiza badań: W tym rozdziale przedstawiono wyniki dla modeli uczenia głębokiego wyuczonych na różnych liczbach zdjęć oraz przeprowadzono analizę tych wyników.
- **Rozdział 8** - Podsumowanie: W ostatnim rozdziale podsumowano pracę, przedstawiono wnioski oraz opisano zrealizowane cele. Wskazano także kierunki dalszego rozwoju projektu.

## **Rozdział 2. Wybrane podstawy teoretyczne pracy**

### **2.1 Uczenie głębokie**

Głębokie uczenie, odmiana uczenia maszynowego, koncentruje się na zastosowaniu wielowarstwowych sztucznych sieci neuronowych. W odróżnieniu od klasycznych metod uczenia maszynowego, takich jak regresja liniowa czy drzewa decyzyjne, głębokie uczenie automatycznie odkrywa reprezentacje danych na różnych poziomach abstrakcji, co jest kluczowe dla rozwiązania złożonych problemów (np. rozpoznawanie obrazów, przetwarzanie języka naturalnego, rozpoznawanie mowy czy analiza danych biologicznych) [1].

Głębokie uczenie opiera się na naśladowaniu ludzkich mózgów, ucząc się na podstawie obszernych zbiorów danych, a procesory graficzne są optymalizowane do pracy z tymi modelami. Wiele technologii sztucznej inteligencji, które zautomatyzowały i usprawniły zadania analityczne, opiera się na głębokim uczeniu [3].

Historia uczenia głębokiego sięga lat 40. XX wieku, kiedy to badacze zaczęli eksplorować koncepcje sztucznych neuronów, które próbowały naśladować biologiczne procesy zachodzące w ludzkim mózgu. W 1986 roku Rumelhart, Hinton i Williams wprowadzili algorytm wstępnej propagacji błędów, który umożliwił efektywne uczenie się sieci neuronowych [2].

W 2006 roku Hinton i Salakhutdinov opublikowali pracę, która przyczyniła się do powstania współczesnego uczenia głębokiego. Przedstawili w niej skonstruowane przez siebie głębokie sieci wstępnie uczące się i pokazali, że sieci te potrafią uczyć się reprezentacji danych na wielu poziomach abstrakcji. Od tego czasu uczenie głębokie zyskało na popularności, a rozwój technologii przyczynił się do udoskonalenia algorytmów oraz wykorzystania uczenia głębokiego w praktyce [2].

## 2.2 Zastosowania uczenia głębokiego

Uczenie głębokie znajduje zastosowanie w szerokim spektrum dziedzin naukowych i przemysłowych. Oto niektóre przykłady:

- **Rozpoznawanie obrazów:** Konwolucyjne sieci neuronowe (CNN) są wykorzystywane do klasyfikacji obrazów, identyfikacji obiektów na zdjęciach, segmentacji obrazów (np. wyodrębnianie elementów pierwszego planu) oraz rozpoznawania twarzy i wyrażeń.
- **Przetwarzanie języka naturalnego (NLP):** Rekurencyjne sieci neuronowe (RNN) oraz mechanizmy uwagi są stosowane do tłumaczenia maszynowego, generowania tekstu, analizy uczuć, rozpoznawania nazw własnych i ekstrakcji relacji między nimi, a także odpowiedzi na pytania oparte na kontekście.
- **Rozpoznawanie mowy:** Sieci neuronowe, takie jak RNN, są wykorzystywane do rozpoznawania mowy, konwersji mowy na tekst oraz syntezy mowy. Dzięki temu możliwe jest tworzenie inteligentnych asystentów głosowych, jak Amazon Alexa czy Google Assistant.
- **Uczenie ze wzmacnieniem:** Uczenie głębokie jest łączone z algorytmami uczenia ze wzmacnieniem, aby sterować robotami, pojazdami autonomicznymi, strategiami handlowymi na giełdzie czy systemami rekomendacji.
- **Medycyna i bioinformatyka:** Uczenie głębokie jest wykorzystywane do przewidywania struktury białek, rozpoznawania chorób na podstawie obrazów medycznych, analizy genomów oraz wczesnego diagnozowania i monitorowania chorób.
- **Sztuka i twórczość:** Algorytmy uczenia głębokiego są stosowane do generowania sztuki, muzyki, stylizowania zdjęć czy tworzenia filmów.
- **Bezpieczeństwo i monitoring:** Uczenie głębokie umożliwia analizę obrazów z kamer monitoringu, rozpoznawanie twarzy oraz detekcję podejrzanych zachowań.

## 2.3 Różnice między uczeniem głębokim a innymi technikami uczenia maszynowego

Uczenie głębokie wyróżnia się od innych technik uczenia maszynowego pod względem kilku kluczowych aspektów:

- **Reprezentacja danych:** W przeciwnym do tradycyjnych metod uczenia maszynowego, takich jak maszyny wektorów nośnych (SVM) czy drzewa decyzyjne, gdzie inżynierowie muszą ręcznie projektować cechy używane do uczenia modelu, uczenie głębokie automatycznie wykorzystuje hierarchiczne reprezentacje danych na różnych poziomach abstrakcji [3]. Pozwala to na efektywniejsze uczenie się złożonych zależności w danych.
- **Architektura sieci:** Sieci neuronowe stosowane w uczeniu głębokim mają wiele warstw ukrytych, co pozwala na uczenie się bardziej złożonych funkcji oraz modelowanie wyższego poziomu abstrakcji [10]. Inne metody uczenia maszynowego, takie jak regresja liniowa czy drzewa decyzyjne, zwykle mają mniejszą złożoność modelu i są mniej elastyczne w przetwarzaniu danych o wysokiej złożoności.
- **Skalowalność:** Uczenie głębokie może być stosowane do przetwarzania ogromnych zbiorów danych dzięki efektywnym algorytmom optymalizacji oraz rosnącej mocy obliczeniowej, w szczególności dzięki wykorzystaniu jednostek przetwarzania graficznego. Inne techniki uczenia maszynowego często napotykają trudności związane z działaniem na dużą skalę, zwłaszcza gdy wymagane jest ekstrahowanie cech z dużych zbiorów danych.
- **Transfer wiedzy:** Uczenie głębokie pozwala na wykorzystanie wiedzy uzyskanej z jednego zadania do innych zadań, co jest nazywane uczeniem transferowym. W tradycyjnym uczeniu maszynowym transfer wiedzy jest trudniejszy do osiągnięcia, ponieważ ręcznie zaprojektowane cechy są często specyficzne dla konkretnego zadania i trudno je przenieść na inne problemy. Uczenie transferowe w uczeniu głębokim umożliwia oszczędność czasu i zasobów, gdyż model może być przystosowany do nowego zadania z mniejszymi nakładami.
- **Tolerancja na szum:** Sieci neuronowe wykorzystywane w uczeniu głębokim są bardziej odporne na szum i brakujące dane niż tradycyjne metody uczenia maszynowego. Dzięki swojej zdolności do uczenia się hierarchicznych reprezentacji danych, uczenie głębokie jest w stanie zidentyfikować i uwzględnić istotne cechy mimo występowania zakłóceń czy niekompletności danych.
- **Wydajność w przypadku dużych zbiorów danych:** Uczenie głębokie wykazuje szczególnie dobrą wydajność w przypadku dużych zbiorów danych. W miarę jak ilość do-

stępnych danych rośnie, sieci neuronowe potrafią lepiej się dostosować i osiągać wyższą dokładność niż inne techniki uczenia maszynowego. Dla porównania, tradycyjne metody uczenia maszynowego mogą osiągać gorsze wyniki przy zwiększaniu ilości danych ze względu na ograniczenia ich modeli i zdolności przetwarzania.

- **Złożoność modelu:** Uczenie głębokie oferuje większą elastyczność w kształtowaniu modeli, co pozwala na uczenie się bardziej złożonych wzorców i funkcji. W przeciwnym razie, metody uczenia maszynowego często korzystają z modeli o mniejszej złożoności, które mają ograniczoną zdolność do modelowania skomplikowanych zależności.

Różnice te sprawiają, że uczenie głębokie jest coraz bardziej popularne w świecie nauki i przemysłu. Pozwala ono na rozwiązywanie wielu problemów, które były trudne lub niemożliwe do rozwiązywania za pomocą innych technik uczenia maszynowego, przyczyniając się do szybkiego postępu w dziedzinie sztucznej inteligencji.

## 2.4 Sieci konwolucyjne

Sieci konwolucyjne (CNN) to specjalny rodzaj sieci neuronowych, w których zastosowano warstwy konwolucyjne. Warstwy te analizują obrazy za pomocą filtrów, które są przesuwane po danych wejściowych, generując mapy cech. Filtry te uczą się wykrywać lokalne wzorce, takie jak krawędzie, tekstury czy kształty [1]. Warstwy pooling (agregujące) są stosowane w celu zmniejszenia wymiarowości map cech, co prowadzi do redukcji ilości parametrów w sieci. Sieci konwolucyjne mogą być również wykorzystywane w połączeniu z innymi warstwami, takimi jak warstwy gęsto połączone (fully connected) czy rekurencyjne (RNN), w zależności od problemu, który mają rozwiązać. Sieci neuronowe konwolucyjne funkcjonują przez gromadzenie i przetwarzanie dużych zbiorów danych w postaci siatek, a następnie ekstrakcję kluczowych cech szczegółowych w celu klasyfikacji i rozpoznawania. CNN zazwyczaj składają się z trzech głównych typów warstw: konwolucyjnej, buforowej (pooling) i w pełni połączonej. Każda z tych warstw pełni inną funkcję, wykonuje określone zadanie na zebranych danych oraz uczy się coraz bardziej złożonych aspektów [8].

## 2.5 Jak opracowywane są sieci CNN?

Sieci neuronowe konwolucyjne (CNN) odgrywają kluczową rolę w uczeniu głębokim i umożliwiają szerokie zastosowania w różnorodnych sektorach na całym świecie. Tworzenie

CNN to czasochłonny i skomplikowany proces, który obejmuje trzy etapy: szkolenie, optymalizację oraz wnioskowanie [6].

**Szkolenie** (trening) w konwolucyjnych sieciach neuronowych (CNN) to proces, w którym sieć uczy się rozpoznawać wzorce i cechy na podstawie danych treningowych. Celem tego procesu jest optymalizacja wag sieci neuronowej, tak aby była w stanie dokonywać prawnych predykcji na nowych, wcześniej niewidzianych danych [5]. Proces szkolenia CNN składa się z kilku kroków:

- **Inicjalizacja wag:** Na początek, wagi sieci neuronowej są inicjalizowane losowo lub za pomocą specjalnych technik inicjalizacji [6].
- **Przekazanie danych do sieci (propagacja wprzód):** Dane treningowe, takie jak obrazy, są przekazywane przez sieć neuronową. W trakcie tego procesu dane przechodzą przez różne warstwy sieci, takie jak warstwy konwolucyjne, warstwy buforowe (pooling) oraz warstwy w pełni połączone. W każdej warstwie sieć ekstrahuje cechy z danych wejściowych, które są przekazywane do kolejnych warstw [6].
- **Obliczenie funkcji straty:** Na końcu sieci neuronowej obliczana jest wartość funkcji straty, która mierzy różnicę między predykcjami sieci a rzeczywistymi etykietami danych treningowych. Funkcja straty jest kluczowym elementem procesu uczenia, ponieważ określa, jak dobrze sieć radzi sobie z zadaniem [6].
- **Propagacja wsteczna:** Po obliczeniu funkcji straty, gradienty tej funkcji są obliczane wstecznie dla każdej warstwy sieci. Gradienty te informują o kierunku, w którym wagi sieci powinny zostać zaktualizowane, aby zmniejszyć wartość funkcji straty [6].
- **Aktualizacja wag:** Wagi sieci neuronowej są aktualizowane za pomocą algorytmu optymalizacji, takiego jak stochastyczny spadek gradientu (SGD), Adam czy RMSprop. Algorytm optymalizacji stosuje obliczone gradienty do wag sieci, modyfikując je w taki sposób, aby funkcja straty była mniejsza [6].
- **Iteracje i epoki:** Powyższe kroki są powtarzane wielokrotnie dla całego zbioru danych treningowych. Przejście przez cały zbiór danych to jedna epoka. Proces szkolenia zwykle obejmuje wiele epok, aż sieć osiągnie zadowalający poziom dokładności [6].

Po zakończeniu procesu szkolenia, CNN jest gotowa do przeprowadzania wnioskowania

na nowych danych. Wagi sieci zostały zoptymalizowane, dzięki czemu potrafi ona dokonywać poprawnych predykcji na podstawie cech wykrytych w danych wejściowych [3].

**Optymalizacja** w konwolucyjnych sieciach neuronowych (CNN) polega na dostosowywaniu parametrów sieci w celu zminimalizowania funkcji koszta i poprawy wydajności sieci [2]. Proces optymalizacji prowadzi do lepszego uczenia się przez sieć istotnych cech danych wejściowych, co przekłada się na lepsze wyniki predykcji. Optymalizacja w sieciach CNN obejmuje kilka aspektów:

- **Aktualizacja wag:** Optymalizacja polega na dostosowywaniu wag sieci neuronowej w odpowiedzi na błędy popełniane podczas procesu uczenia. Metody aktualizacji wag, takie jak stochastyczny gradient prosty (SGD) czy adaptacyjne metody optymalizacji (np. Adam, RMSprop), wykorzystują informacje o gradientach funkcji koszta do aktualizacji wag.
- **Dobór hiperparametrów:** Optymalizacja może obejmować dobór odpowiednich hiperparametrów, takich jak liczba warstw, liczba neuronów w każdej warstwie, współczynnik uczenia się, wielkość batcha czy funkcje aktywacji. Hiperparametry można dobrać za pomocą technik takich jak przeszukiwanie siatki, przeszukiwanie losowe czy optymalizacja bayesowska.
- **Regularyzacja:** Wprowadzenie technik regularyzacji, takich jak L1, L2 czy dropout, może pomóc w zapobieganiu nadmiernemu dopasowaniu sieci, co pozytywnie wpływa na jej zdolność generalizacji.
- **Architektura sieci:** Optymalizacja obejmuje również eksperymentowanie z różnymi architekturami sieci, aby znaleźć taką, która najlepiej radzi sobie z konkretnym zadaniem. Warto zbadać istniejące architektury, które odniosły sukces w podobnych problemach, takie jak VGG, ResNet czy Inception.
- **Techniki augmentacji danych:** Optymalizacja może również obejmować zastosowanie technik augmentacji danych, które polegają na wprowadzeniu różnych transformacji na danych wejściowych, takich jak obracanie, przeskalowanie, odbicie lustrzane czy zmiana oświetlenia. Augmentacja danych pomaga sieci lepiej generalizować na nowych danych.

**Wnioskowanie** w konwolucyjnych sieciach neuronowych (CNN) odnosi się do procesu, w którym wcześniej wytrenowana sieć jest stosowana do analizy nowych danych wejściowych i generowania predykcji [2]. W przypadku CNN, wnioskowanie często dotyczy zastosowań takich jak klasyfikacja obrazów, detekcja obiektów czy segmentacja semantyczna. W procesie wnioskowania, dane wejściowe, takie jak obraz, są przekazywane przez sieć CNN, która przeprowadza szereg operacji w różnych warstwach. Warstwy te obejmują warstwy konwolucyjne, warstwy buforowe (pooling) oraz warstwy w pełni połączone. Każda warstwa analizuje dane wejściowe i ekstrahuje istotne cechy, które są przekazywane do kolejnych warstw sieci. W trakcie wnioskowania sieć neuronowa korzysta z wcześniej naucznej hierarchii cech oraz wag, które zostały optymalizowane podczas procesu treningu. Dzięki tym wagom sieć jest w stanie rozpoznać i klasyfikować nowe dane wejściowe [2]. Na końcu sieci znajduje się warstwa wyjściowa, która generuje wynik końcowy, czyli predykcję. W przypadku klasyfikacji obrazów może to być prawdopodobieństwo przynależności obrazu do danej klasy. W detekcji obiektów sieć może generować współrzędne prostokątów otaczających obiekty oraz ich kategorie. Wnioskowanie jest zwykle znacznie szybsze niż proces uczenia, ponieważ nie wymaga obliczania gradientów ani aktualizacji wag. Jako że wagi są już wcześniej nauczone, sieć koncentruje się na przeprowadzeniu operacji na nowych danych wejściowych, aby wygenerować predykcję. W przypadku aplikacji w czasie rzeczywistym, takich jak rozpoznawanie mowy czy analiza wideo, szybkość wnioskowania jest kluczowym czynnikiem wpływającym na użyteczność i skuteczność sieci neuronowej [1].

## 2.6 Wady produkcyjne i ich rodzaje

Wady produkcyjne to niezgodności w procesie wytwarzania, które prowadzą do tego, że produkt nie spełnia wymagań jakościowych. Wady te mogą wynikać z różnych przyczyn, takich jak błędy w procesie produkcyjnym, zużycie sprzętu czy błędy ludzkie [4]. W zależności od charakterystyki produktu i procesu produkcyjnego można wyróżnić różne rodzaje wad produkcyjnych, takie jak:

- **Wady powierzchniowe:** Dotyczą one uszkodzeń lub nieprawidłowości na powierzchni produktu. Wady te obejmują pęknięcia, zadrapania, zmarszczki, przebarwienia czy zanieczyszczenia na powierzchni wyrobu. Mogą być spowodowane przez niewłaściwe parametry procesu, takie jak temperatury czy ciśnienia, niedostateczną kontrolę jakości

surowców lub problemy z utrzymaniem czystości w środowisku produkcyjnym [4].

- **Wady geometryczne:** Są to nieprawidłowości związane z wymiarami, kształtami czy wzajemnym położeniem elementów produktu. Wady te obejmują niewłaściwe wymiary, kształty czy położenie części w stosunku do siebie nawzajem. Mogą być spowodowane przez błędy w ustawieniach maszyn, niedokładności w narzędziach pomiarowych czy błędy w projektowaniu produktu [4].
- **Wady materiałowe:** Dotyczą one wad w strukturze materiału, takich jak pęcherze powietrza, porowatość czy zanieczyszczenia. Mogą być spowodowane przez niewłaściwy dobór surowców, problemy z mieszaniami materiałów czy nieodpowiednie warunki procesowe, takie jak szybkość chłodzenia czy czas utwardzania [4].
- **Wady funkcjonalne:** Są to problemy związane z działaniem produktu, wynikające z nieprawidłowego montażu, błędów w projektowaniu czy wadliwych komponentów. Wady te mogą prowadzić do awarii produktu, niebezpiecznych sytuacji czy niezadowolenia klientów. Przyczyną takich wad może być brak precyzyjności w procesie montażu, niewłaściwy dobór komponentów czy niedostateczna kontrola jakości w trakcie projektowania i produkcji [4].

Rozpoznanie i identyfikacja wad produkcyjnych są kluczowe dla utrzymania wysokiej jakości produktów i zadowolenia klientów. Poprzez monitorowanie procesów produkcyjnych, wprowadzenie systemów kontroli jakości oraz analizę przyczyn wad, przedsiębiorstwa mogą zminimalizować występowanie wad produkcyjnych i uniknąć ich negatywnego wpływu na produkty oraz reputację firmy [9]. Oto kilka sposobów, które mogą pomóc w redukcji wad produkcyjnych:

- **Staranne projektowanie produktu:** Dokładne projektowanie produktu z uwzględnieniem wymagań jakościowych oraz ograniczeń materiałowych i procesowych może pomóc w zapobieganiu wielu wadom produkcyjnym. Współpraca z ekspertami z różnych dziedzin może również pomóc w identyfikacji potencjalnych problemów na etapie projektowania [9].
- **Kontrola jakości surowców:** Przeprowadzanie kontroli jakości dostawców surowców i systematyczne sprawdzanie jakości materiałów przed rozpoczęciem produkcji może pomóc w wykryciu i eliminacji wad materiałowych [9].

- **Optymalizacja procesów produkcyjnych:** Analiza procesów produkcyjnych w celu identyfikacji i eliminacji źródeł wad może prowadzić do wydajniejszej i bardziej niezawodnej produkcji [9]. Obejmuje to zarówno monitorowanie maszyn i narzędzi, jak i przeglądanie procedur produkcyjnych, aby zapewnić ich zgodność z najlepszymi praktykami [9].
- **Szkolenie pracowników:** Zapewnienie odpowiedniego szkolenia i wsparcia dla pracowników może znacznie zmniejszyć ryzyko błędów ludzkich, które prowadzą do wad produkcyjnych. Regularne szkolenia, aktualizacje wiedzy i umiejętności oraz zrozumienie znaczenia jakości w procesie produkcyjnym są kluczowe dla utrzymania wysokiego standardu pracy [9].
- **Systematyczna kontrola jakości:** Wprowadzenie skutecznych systemów kontroli jakości na każdym etapie produkcji, od projektowania po montaż i wysyłkę, może pomóc w wykrywaniu i eliminacji wad produkcyjnych. Monitorowanie, inspekcje i testy jakościowe powinny być przeprowadzane regularnie, aby zapewnić ciągłe doskonalenie jakości produktów [9].
- **Analiza przyczyn wad i działania korygujące:** Gdy wady produkcyjne zostaną zidentyfikowane, ważne jest przeprowadzenie dokładnej analizy przyczyn i wprowadzenie działań korygujących, aby uniknąć powtarzania się tych samych problemów. Analiza przyczyn może obejmować badanie procesów, surowców, maszyn i praktyk pracy, a także analizę danych historycznych dotyczących jakości w celu identyfikacji wzorców i trendów [9].
- **Utrzymanie maszyn i sprzętu:** Regularne konserwacje, naprawy i modernizacje maszyn oraz sprzętu są niezbędne, aby zapewnić ich sprawne działanie i minimalizować ryzyko wystąpienia wad produkcyjnych związanych z zużyciem czy uszkodzeniami. Planowanie przeglądów i utrzymania oparte na harmonogramie może zapewnić ciągłą działalność produkcyjną i utrzymanie wysokiej jakości [9].
- **Wdrożenie technologii monitorujących i kontrolujących:** Implementacja zaawansowanych technologii, takich jak systemy wizyjne, sensory czy algorytmy uczenia maszynowego, może pozwolić na szybsze i bardziej precyzyjne wykrywanie wad produkcyjnych oraz monitorowanie procesów w czasie rzeczywistym. Dzięki temu możliwe

jest szybkie reagowanie na problemy i unikanie poważniejszych konsekwencji [9].

- **Adaptacyjne zarządzanie jakością:** Elastyczne podejście do zarządzania jakością, które pozwala na szybkie dostosowywanie się do zmieniających się warunków rynkowych, wymagań klientów czy pojawiających się problemów, może przyczynić się do efektywniejszego radzenia sobie z wadami produkcyjnymi. Współpraca z innymi działami, takimi jak badania i rozwój czy logistyka, może również pomóc w utrzymaniu wysokiej jakości w całym łańcuchu dostaw [9].

Wdrażając te strategie, przedsiębiorstwa mogą zredukować występowanie wad produkcyjnych i dostarczać produkty o wysokiej jakości, które spełniają oczekiwania klientów. Długoterminowe zobowiązanie do doskonalenia jakości i ciągłego ulepszania procesów produkcyjnych może prowadzić do lepszej konkurencyjności, lojalności klientów oraz wzrostu rentowności firmy [9].

## 2.7 Istniejące rozwiązania

W przemyśle wykorzystuje się wiele technik do wykrywania wad produkcyjnych. Poniżej przedstawiono opis wybranych istniejących rozwiązań:

- **Kontrola wizualna:** Tradycyjna kontrola wizualna polega na ręcznym sprawdzaniu produktów przez operatorów, którzy oceniają ich jakość wzrokowo. Kontrola ta może być czasochłonna, niewystarczająco precyzyjna i podatna na błędy ludzkie, a jej efektywność zależy od doświadczenia inspektorów. Chociaż metoda ta jest stosunkowo tania, może prowadzić do niezadowalającej jakości produktów oraz wysokich kosztów związanych z błędami w wykrywaniu wad.
- **Systemy wizyjne:** Systemy wizyjne to zaawansowane rozwiązania oparte na kamerach oraz algorytmach przetwarzania obrazów. Są one stosowane do automatycznego analizowania produktów pod kątem wad, takich jak nieprawidłowe wymiary czy defekty powierzchniowe. Systemy te mogą być bardziej precyzyjne niż kontrola wizualna, ale często wymagają ręcznego projektowania cech i mogą być trudne do skalowania, gdy liczba różnych produktów czy wad wzrasta.
- **Techniki uczenia maszynowego:** W celu wykrywania wad produkcyjnych wykorzystuje się również techniki uczenia maszynowego, takie jak maszyny wektorów nośnych

(SVM), drzewa decyzyjne czy klasteryzacja. Te metody pozwalają na automatyczne wykrywanie wad na podstawie wcześniej opracowanych cech. Jednakże, wymagają one często ręcznego projektowania cech, co może być czasochłonne i trudne w przypadku złożonych problemów.

- **Uczenie głębokie i sieci konwolucyjne:** Sieci konwolucyjne (CNN) są jednym z najbardziej zaawansowanych i skutecznych rozwiązań w dziedzinie wykrywania wad produkcyjnych. Dzięki automatycznemu uczeniu się reprezentacji danych, sieci te potrafią wykrywać wady na obrazach z wysoką precyzją i są łatwe do skalowania. Ponadto, uczenie transferowe pozwala na zastosowanie wiedzy uzyskanej z jednego zadania do innego, co ułatwia adaptację modeli do różnych rodzajów wad i procesów produkcyjnych [10].
- **Internet Rzeczy (IoT) i analiza danych:** Wykorzystanie technologii Internetu Rzeczy (IoT) w połączeniu z zaawansowanymi technikami analizy danych może przyczynić się do lepszego monitorowania i kontroli procesów produkcyjnych. Inteligentne czujniki i urządzenia IoT mogą zbierać dane ze wszystkich etapów produkcji, takie jak temperatura, ciśnienie, prędkość czy wibracje. Zebrane dane mogą być następnie analizowane w czasie rzeczywistym przy użyciu zaawansowanych algorytmów uczenia maszynowego, aby wykrywać nieprawidłowości w procesie produkcyjnym, które mogą prowadzić do wad.

## 2.8 Wybrany język programowania - Python

Python to dynamicznie typowany, interpretowany język programowania wysokiego poziomu, który został stworzony przez Guido van Rossumę i po raz pierwszy opublikowany w 1991 roku. Zyskał ogromną popularność w różnych dziedzinach, w tym w uczeniu maszynowym i uczeniu głębokim, ze względu na kilka istotnych cech [7].

- **Składnia:** Python cechuje się prostą i czytelną składnią, która promuje przejrzystość i łatwość zrozumienia kodu. Białe znaki, takie jak wcięcia, są istotne w Pythonie, co zmusza programistów do stosowania konsekwentnych, dobrze zorganizowanych stylów kodowania [7].
- **Biblioteki:** Python posiada bogaty ekosystem bibliotek wspierających uczenie maszynowe.

nowe i uczenie głębokie, co pozwala na tworzenie zaawansowanych modeli z mniejszym nakładem pracy [7].

- **Wsparcie społeczności:** Python cieszy się dużym wsparciem społeczności programistów, co oznacza, że istnieje wiele zasobów online, takich jak dokumentacja, kursy, tutoriale, blogi, fora dyskusyjne i konferencje, które mogą pomóc w nauce i rozwoju umiejętności związanych z uczeniem maszynowym w Pythonie [7].
- **Integracja z innymi językami programowania:** Python można łatwo zintegrować z innymi językami programowania, takimi jak C, C++ czy Fortran, co pozwala na tworzenie wydajnych rozwiązań hybrydowych. Można na przykład pisać kod w Pythonie, który wywołuje funkcje napisane w innych językach, aby zwiększyć wydajność kodu. Takie podejście jest często stosowane w bibliotekach uczenia maszynowego, które używają jądra napisanego w C++ czy Fortranie w celu przyspieszenia obliczeń [7].
- **Przenośność:** Python jest wieloplatformowy, co oznacza, że może być uruchamiany na różnych systemach operacyjnych, takich jak Windows, macOS czy Linux. Daje to programistom większą swobodę w wyborze środowiska pracy oraz pozwala na łatwe wdrożenie rozwiązań na różne platformy [7].

## **Rozdział 3. Wymagania funkcjonalne oraz niefunkcjonalne**

### **3.1 Wymagania funkcjonalne**

Wymagania funkcjonalne systemu obejmują następujące elementy:

1. **Wczytywanie i przetwarzanie danych wejściowych (obrazów):** System powinien być w stanie wczytać dane wejściowe w postaci obrazów oraz przetworzyć je w celu przygotowania do analizy przez model.
2. **Trenowanie modelu na zbiorze treningowym:** System powinien być wyposażony w model oparty na uczeniu głębokim, który jest trenowany na zbiorze treningowym, zawierającym obrazy uszkodzonych i prawidłowych elementów.
3. **Walidacja modelu na zbiorze walidacyjnym:** System powinien wykorzystywać zbiór walidacyjny, aby sprawdzić jakość modelu w trakcie procesu trenowania. Walidacja pozwala dostosować hiperparametry modelu, aby uniknąć nadmiernego dopasowania (overfitting).
4. **Testowanie modelu na zbiorze testowym:** Po zakończeniu trenowania, system powinien zostać przetestowany na zbiorze testowym, który zawiera obrazy nieznane dla modelu. Wyniki testów pozwolą ocenić ostateczną jakość modelu.
5. **Klasyfikacja obrazów na części uszkodzone i prawidłowe:** Głównym celem systemu jest klasyfikacja obrazów na części uszkodzone i prawidłowe, co pozwala zidentyfikować problemy z jakością w procesie produkcyjnym.

### **3.2 Wymagania niefunkcjonalne**

Wymagania niefunkcjonalne systemu odnoszą się do cech jakościowych, takich jak:

1. **Dokładność klasyfikacji:** System powinien osiągać wysoką dokładność klasyfikacji, aby skutecznie identyfikować uszkodzone i prawidłowe elementy.
2. **Czas uczenia modelu:** Czas trenowania modelu powinien być na tyle krótki, aby umożliwić szybkie dostosowanie modelu do nowych danych.
3. **Złożoność obliczeniowa modelu:** Model powinien być na tyle prosty, aby możliwe obliczenia nie obciążały nadmiernie zasobów sprzętowych, jednocześnie zachowując wysoką jakość klasyfikacji.
4. **Skalowalność systemu:** System powinien być skalowalny, co oznacza, że powinien być w stanie obsłużyć większe ilości danych oraz dostosować się do zmieniających się warunków (np. dodanie nowych klas obiektów do klasyfikacji).
5. **Współczynnik fałszywych pozytywów i fałszywych negatywów:** System powinien charakteryzować się niskim współczynnikiem fałszywych pozytywów (FP) i fałszywych negatywów (FN). Fałszywe pozytywy to przypadki, gdy system błędnie klasyfikuje uszkodzone elementy jako prawidłowe, natomiast fałszywe negatywy to błędna klasyfikacja prawidłowych elementów jako uszkodzone. Oba te rodzaje błędów mogą prowadzić do niekorzystnych skutków, takich jak przestój w produkcji, czy też przekroczenie progów jakościowych.

## Rozdział 4. Projekt rozwiązania

### 4.1 Wybór sieci konwolucyjnej

Sieci konwolucyjne są szczególnie odpowiednie dla problemów związanych z analizą obrazów, ponieważ potrafią automatycznie uczyć się cech na różnych poziomach abstrakcji. W przeciwnym razie, ręczne projektowanie cech obrazu, zwłaszcza w przypadku złożonych zadań klasyfikacji, może być żmudne i czasochłonne. CNN pozwala nam wykryć lokalne wzorce w obrazach, takie jak kształty i tekstury, które są istotne dla zadania klasyfikacji.

### 4.2 Korzyści z zastosowania uczenia głębokiego

Uczenie głębokie, jako poddziedzina uczenia maszynowego, oferuje wiele korzyści w kontekście klasyfikacji obrazów. Oto niektóre z nich:

- **Automatyczna ekstrakcja cech:** Uczenie głębokie pozwala na automatyczne wykrywanie istotnych cech w danych, eliminując potrzebę ręcznego projektowania cech. W przypadku klasyfikacji obrazów oznacza to, że sieci głębokie potrafią uczyć się hierarchicznych reprezentacji obrazów, co prowadzi do lepszej wydajności klasyfikacji.
- **Generalizacja:** Uczenie głębokie ma zdolność do generalizacji na nowe, niewidziane wcześniej dane. Oznacza to, że model wytrenowany na odpowiednio dużym i różnorodnym zbiorze danych może skutecznie klasyfikować obrazy, które nie były częścią jego zbioru treningowego.
- **Skalowalność:** Architektury uczenia głębokiego są elastyczne i łatwo skalowalne. Można je dostosować do różnych rozmiarów i rodzajów danych, co pozwala na efektywne rozwiązanie problemów o różnym stopniu złożoności.

- **Wydajność:** Dzięki zastosowaniu akceleratorów sprzętowych, takich jak GPU, proces uczenia głębokich sieci neuronowych można znacznie przyspieszyć, co prowadzi do szybszego rozwoju i wdrożenia modeli.

### 4.3 Ogólny zarys rozwiązania

Zaprogramowane rozwiązanie będzie oparte na sieci konwolucyjnej (CNN), która będzie trenowana na danych obrazowych przedstawiających uszkodzone i prawidłowe elementy. Przygotowany model będzie następnie testowany na zbiorze testowym w celu oceny jego dokładności oraz zdolności generalizacji.

W kolejnych podrozdziałach przedstawione zostaną szczegółowe etapy procesu projektowania rozwiązania, takie jak:

- Import bibliotek
- Wczytywanie danych
- Ładowanie danych
- Wstępne przetwarzanie danych
- Podział danych na zestawy treningowe, walidacyjne i testowe

Na podstawie uzyskanych wyników będzie można ocenić jakość opracowanego rozwiązania oraz jego potencjalne ograniczenia.

### 4.4 Import bibliotek

W tym podrozdziale omówione zostaną wykorzystane biblioteki oraz ich zastosowanie w projekcie. W naszym rozwiążaniu korzystamy z następujących bibliotek:

- **TensorFlow:** Główna biblioteka do uczenia głębokiego, która umożliwia definiowanie, trenowanie i ewaluację modeli sieci neuronowych. W projekcie wykorzystany jest TensorFlow do implementacji sieci konwolucyjnej (CNN) oraz zarządzania procesem uczenia i walidacji modelu.

- **OpenCV:** Popularna biblioteka do przetwarzania obrazów, która umożliwia wczytywanie, modyfikowanie i zapisywanie obrazów w różnych formatach. W projekcie wykorzystany jest OpenCV do wczytywania obrazów z dysku, ich przekształceń (np. skalowanie, obrót), a także sprawdzania poprawności danych obrazowych.
- **Matplotlib:** Biblioteka do generowania wysokiej jakości wykresów 2D i 3D. W projekcie wykorzystany jest Matplotlib do wizualizacji danych, takich jak wykresy dokładności i straty w trakcie procesu uczenia oraz prezentacji wyników klasyfikacji obrazów.
- **NumPy:** Biblioteka do obsługi macierzy wielowymiarowych, która oferuje wiele funkcji matematycznych o dużej wydajności. W projekcie wykorzystany jest NumPy do manipulacji danymi obrazowymi, przetwarzania macierzy oraz realizacji obliczeń matematycznych.

Do pracy z tymi bibliotekami należy je zimportować, korzystając z poniższego kodu:

```
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

Dzięki temu mamy dostęp do wszystkich funkcji i klas oferowanych przez te biblioteki, co pozwala na efektywną realizację projektu.

Kolejnym krokiem jest konfiguracja pamięci GPU, aby uniknąć wykorzystania całej dostępnej pamięci przez TensorFlow. Dzięki temu inni użytkownicy również będą mogli korzystać z GPU.

```
import tensorflow as tf
import os
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
tf.config.list_physical_devices('GPU')
```

## 4.5 Wczytywanie danych

W celu wczytania danych do modelu TensorFlow, należy zacząć od zorganizowania katalogów ze zdjęciami. Klasyfikacja obrazów odbywa się na podstawie nazw folderów, gdzie każdy folder odpowiada jednej klasie. W opisywanym, przypadku, są dwa foldery: 'uszkodzony' i 'prawidłowy'.

Podczas wczytywania danych, należy skontrolować jakość zdjęć, sprawdzając, czy można je załadować do biblioteki OpenCV oraz czy ich rozszerzenie jest zgodne z oczekiwany (JPEG, JPG, BMP, PNG). W przypadku wykrycia uszkodzonych lub nieobsługiwanych obrazów, są one usuwane. Poniżej przedstawiono fragment kodu odpowiedzialny za kontrolę i usuwanie wadliwych zdjęć:

```
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Zdjęcie posiada nieobsługiwane rozszerzenie {}'.format(
                    image_path))
                os.remove(image_path)
        except Exception as e:
            print('Wystąpił problem ze zdjęciem {}'.format(image_path))
```

## 4.6 Ładowanie danych

W kolejnym kroku trzeba załadować dane za pomocą funkcji `image_dataset_from_directory` z biblioteki TensorFlow. Funkcja ta automatycznie wczytuje zdjęcia z katalogów i przetwarza je do odpowiedniego formatu. Następnie konwertujemy dane na iterator, aby uzyskać dostęp do poszczególnych zdjęć i ich etykiet.

```

import numpy as np
from matplotlib import pyplot as plt
data = tf.keras.utils.image_dataset_from_directory('data')
data_iterator = data.as_numpy_iterator()
batch = data_iterator.next()
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])

```

## 4.7 Wstępne przetwarzanie danych

Wstępne przetwarzanie danych polega na przygotowaniu danych wejściowych, tak aby model uczenia głębokiego mógł lepiej zrozumieć i wykorzystać informacje zawarte w tych danych. W przypadku tego projektu, wstępne przetwarzanie danych obejmuje przeskalowanie wartości pikseli do zakresu 0-1, co ułatwia uczenie się modelu.

Przeskalowanie wartości pikseli polega na podzieleniu wartości każdego piksela przez 255, co jest najwyższą wartością możliwą dla wartości piksela w formacie RGB. Wynik tego przekształcenia daje wartości zmiennoprzecinkowe w zakresie od 0 do 1, co jest preferowanym zakresem wartości dla wielu algorytmów uczenia głębokiego, w tym również dla sieci neuronowych.

W poniższym fragmencie kodu przeskalowanie wartości pikseli jest wykonywane na całym zbiorze danych za pomocą funkcji `map`. Funkcja ta przekształca dane przy użyciu podanej funkcji, w tym przypadku przeskalowując wartości pikseli dzieląc przez 255.

```

data = data.map(lambda x, y: (x / 255, y))
data.as_numpy_iterator().next()

```

Przeskalowanie wartości pikseli ma kilka zalet. Po pierwsze, przekształcone wartości są mniejsze, co może przyspieszyć proces uczenia się modelu. Po drugie, przeskalowanie wartości pikseli może również przyczynić się do poprawy zdolności generalizacji modelu, ponieważ wartości w podobnym zakresie mogą być łatwiej porównywane przez sieć neuronową.

## 4.8 Podział danych na zestawy treningowe, walidacyjne i testowe

Podział danych na zestawy treningowe, walidacyjne i testowe jest kluczowym elementem procesu uczenia się modelu głębokiego. Różne zestawy danych mają różne funkcje w procesie uczenia się:

- **Zestaw treningowy** służy do uczenia modelu. W trakcie uczenia model aktualizuje swoje wagi na podstawie błędów, które popełnia w prognozowaniu etykiet dla danych treningowych.
- **Zestaw walidacyjny** służy do monitorowania postępów modelu podczas uczenia. Na podstawie wyników na danych walidacyjnych, możemy dostosować parametry modelu, takie jak liczba epok, funkcje kosztu czy hiperparametry optymalizatora. Zestaw walidacyjny pomaga również w wykrywaniu problemów, takich jak nadmierne dopasowanie (overfitting).
- **Zestaw testowy** służy do oceny końcowej wydajności modelu po zakończeniu uczenia. Wyniki na danych testowych dają nam informacje na temat zdolności generalizacji modelu do danych, których nie widział wcześniej.

Aby podzielić dane na odpowiednie zestawy, najpierw należy określić ich rozmiary. W tym przypadku, 70% danych zostanie przeznaczone na trening, 20% na walidację i 10% na testowanie. Następnie trzeba skorzystać z metod `take`, `skip` i `take` dostępnych dla obiektu `dataset` w TensorFlow, aby wydzielić odpowiednie części danych.

```
train_size = int(len(data).7)
val_size = int(len(data).2) + 1
test_size = int(len(data)*.1) + 1
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size + val_size).take(test_size)
```

Warto zauważyć, że podział danych może być również wykonywany za pomocą funkcji `train_test_split` z biblioteki scikit-learn, jednak w tym przypadku korzystamy z funkcji dostępnych w TensorFlow.

Wybór odpowiedniego podziału danych zależy od wielu czynników, takich jak liczba dostępnych danych, złożoność problemu oraz złożoność modelu. Zbyt mały zestaw treningowy

może prowadzić do niedouczenia modelu, podczas gdy zbyt mały zestaw walidacyjny lub testowy może prowadzić do niedokładnej oceny wydajności modelu.

## Rozdział 5. Implementacja

### 5.1 Budowa modelu sieci neuronowej

W celu rozwiązania problemu rozpoznawania wad produkcyjnych, zastosowano sieć neuronową opartą na architekturze konwolucyjnej (CNN). Sieci tego typu są powszechnie używane w problemach analizy obrazów, ponieważ potrafią efektywnie wykrywać lokalne wzorce i cechy na obrazach. W tym przypadku, sieć będzie w stanie nauczyć się rozpoznawania różnych wad produkcyjnych na podstawie analizy dostarczonych zdjęć.

Model sieci neuronowej został zaimplementowany przy użyciu biblioteki TensorFlow w języku Python. Poniżej przedstawiono kod źródłowy użyty do budowy modelu:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,
MaxPooling2D, Dense, Flatten, Dropout
model = Sequential()
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape
=(256,256,3)))
model.add(MaxPooling2D())
model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Sieć składa się z trzech warstw konwolucyjnych z funkcją aktywacji ReLU oraz warstwami MaxPooling po każdej z nich. Warstwy konwolucyjne uczą się wykrywać lokalne

cechy na obrazach, a MaxPooling pomaga w redukcji wymiarowości, co przyspiesza uczenie i zmniejsza ryzyko przeuczenia.

Po trzech warstwach konwolucyjnych, sieć przechodzi przez warstwę Flatten, która spłaszcza dane do jednowymiarowego wektora, co pozwala na przekazanie ich do warstw gęstych (Dense). W tym przypadku, użyto jednej warstwy gęstej z 256 neuronami i funkcją aktywacji ReLU.

Na końcu, sieć kończy się warstwą gęstą z jednym neuronem i funkcją aktywacji sigmoidalną. Ta warstwa odpowiada za generowanie wyników klasyfikacji, gdzie wartość bliska 0 wskazuje na uszkodzony produkt, a wartość bliska 1 na prawidłowy.

Model został skompilowany z użyciem optymalizatora Adam, funkcji straty BinaryCrossentropy oraz metryki dokładności:

```
model.compile('adam', loss=tf.losses.BinaryCrossentropy(),
metrics=['accuracy'])
```

Zastosowanie optymalizatora Adam pomaga w szybszym i skuteczniejszym uczeniu się sieci, gdyż dostosowuje szybkość uczenia na podstawie zmian gradientu. Funkcja straty BinaryCrossentropy jest odpowiednia do problemów klasyfikacji binarnej, takich jak ten, ponieważ pozwala na ocenę różnic między prawdziwymi etykietami a prognozowanymi przez sieć.

## 5.2 Trenowanie modelu

Trenowanie modelu sieci neuronowej to proces optymalizacji wag w sieci, aby osiągnąć jak najlepszą wydajność w rozpoznawaniu wad produkcyjnych na podstawie zdjęć. W implementacji wykorzystano dane treningowe i walidacyjne do uczenia modelu oraz oceny jego wydajności podczas trenowania. W tej sekcji omówione zostanie trenowanie modelu, analiza wydajności oraz sposób monitorowania procesu uczenia.

Wykorzystano metodę `fit` dostarczoną przez bibliotekę Keras, aby przeszkościć model sieci neuronowej. Poniższy kod przedstawia sposób trenowania modelu przez 5 epok, używając danych treningowych oraz walidacyjnych:

```
hist = model.fit(train, epochs=5,
validation_data=val, callbacks=[tensorboard_callback])
```

Parametr epochs określa liczbę pełnych przebiegów przez dane treningowe. W każdej epoce model próbuje zminimalizować funkcję straty na danych treningowych, dostosowując wagi sieci neuronowej. Użycie danych walidacyjnych pozwala na obserwację procesu uczenia się i wykrycie ewentualnego przeuczenia modelu. Jeśli model zaczyna się przeuczać, dokładność na danych walidacyjnych zacznie maleć, podczas gdy dokładność na danych treningowych nadal będzie rosnąć.

TensorBoard to narzędzie do wizualizacji uczenia sieci neuronowych, które pozwala na monitorowanie różnych metryk, takich jak funkcja straty i dokładność. W implementacji używa TensorBoard jako wywołania zwrotnego podczas trenowania, co pozwala na automatyczne zapisywanie danych do logów:

```
logdir='logs'  
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

Logi TensorBoard można następnie wyświetlić w przeglądarce, aby uzyskać interaktywną wizualizację procesu uczenia. Aby uruchomić TensorBoard, należy wpisać w terminalu poniższe polecenie, a następnie otworzyć wyświetlony adres URL w przeglądarce:

```
tensorboard --logdir=logs
```

Po zakończeniu trenowania modelu, można ocenić jego wydajność na podstawie historii uczenia. Poniższy kod przedstawia sposób tworzenia wykresów straty oraz dokładności dla danych treningowych i walidacyjnych:

```
fig = plt.figure()  
plt.plot(hist.history['loss'], color='teal',  
label='strata treningowa')  
plt.plot(hist.history['val_loss'], color='orange',  
label='strata walidacji')  
fig.suptitle('Strata', fontsize=20)  
plt.legend(loc="upper left")  
plt.show()  
fig = plt.figure()  
plt.plot(hist.history['accuracy'], color='teal',  
label='dokladnosc treningowa')
```

```
plt.plot(hist.history['val_accuracy'], color='orange',
label='dokladnosc walidacji')
fig.suptitle('Dokladnosc', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```

Wykresy te pozwalają na analizę wydajności modelu w czasie trenowania. Na wykresie straty obserwujemy spadek wartości funkcji straty zarówno dla danych treningowych, jak i walidacyjnych. Jeśli model jest odpowiednio uczone, strata na danych walidacyjnych powinna stabilizować się na niskim poziomie.

Wykres dokładności przedstawia, jak dobrze model radzi sobie z klasyfikacją próbek na danych treningowych i walidacyjnych. W miarę jak model się uczy, dokładność powinna rosnąć, aż osiągnie pewien poziom, po którym może wystąpić przeuczenie. W przypadku przeuczenia, dokładność na danych treningowych będzie nadal rosnąć, podczas gdy dokładność na danych walidacyjnych będzie maleć.

### 5.3 Wczytywanie przykładowego obrazu

W tym oraz w kolejnych podrozdziałach omówione zostanie testowanie modelu na przykładach obrazów, sprawdzając jego zdolność do klasyfikacji zdjęć jako uszkodzonych lub prawidłowych elementów. Testowanie modelu odbywa się na podstawie dostarczonego kodu źródłowego. Pierwszym krokiem w testowaniu modelu jest wczytanie przykładowego obrazu, który zostanie następnie przekazany do modelu w celu klasyfikacji. Wczytujemy obraz za pomocą biblioteki OpenCV:

```
img = cv2.imread('testing/failure_4.png')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

### 5.4 Przeskalowanie obrazu

Przed przekazaniem obrazu do modelu, konieczne jest jego przeskalowanie do wymiarów, na których model został wytrenowany. W naszym przypadku, model został przeszko-

ny na obrazach o wymiarach 256x256 pikseli. Przeskalowanie obrazu odbywa się za pomocą funkcji `tf.image.resize`:

```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```

## 5.5 Predykcja klasy obrazu

Następnie przekazujemy przeskalowany obraz do modelu, aby uzyskać predykcję klasy. Przed przekazaniem obrazu do modelu, konieczne jest jego normalizowanie przez podzielenie wartości pikseli przez 255:

```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

## 5.6 Interpretacja wyników

Wynik predykcji (`yhat`) to wartość od 0 do 1, która wskazuje na przynależność obrazu do klasy 'prawidłowy'. Aby uzyskać konkretną klasę obrazu, ustalamy próg (np. 0,5) i sprawdzamy, czy wartość predykcji przekracza ten próg:

```
if yhat > 0.5:
    print(f'Wskazany obraz został sklasyfikowany jako czesc prawidlowa')
else:
    print(f'Wskazany obraz został sklasyfikowany jako czesc uszkodzona')
```

W ten sposób możemy przetestować model na różnych przykładach obrazów i ocenić jego zdolność do klasyfikacji uszkodzonych i prawidłowych elementów. Testowanie modelu na przykładach jest istotne dla praktycznego zastosowania modelu w rzeczywistych scenariuszach, gdzie musi on poprawnie sklasyfikować różnorodne obrazy przedstawiające uszkodzone i prawidłowe elementy.

## 5.7 Zapisywanie modelu

Aby zachować wytrenowany model i umożliwić jego dalsze wykorzystanie, należy zapisać jego strukturę i parametry. Dzięki temu będziemy mogli wczytać model i użyć go do klasyfikacji obrazów w przyszłości bez konieczności przeprowadzania procesu treningowego ponownie. Zapisanie modelu w TensorFlow odbywa się za pomocą metody `save`:

```
model.save(os.path.join('models', 'imageclassificationversionlive.h5'))
```

Model zostaje zapisany w formacie HDF5, który przechowuje zarówno architekturę modelu, jak i nauczone wagи.

## 5.8 Wczytywanie modelu

Aby wczytać zapisany model, używamy funkcji `load_model` z biblioteki TensorFlow:

```
new_model = load_model(os.path.join('models', 'imageclassificationversionlive.h5'))
```

Wczytany model można następnie wykorzystać do przewidywania klasy obrazów, podobnie jak przed zapisaniem:

```
img = cv2.imread('testing/failure_4.png')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
resize = tf.image.resize(img, (256, 256))
plt.imshow(resize.numpy().astype(int))
plt.show()
yhat = model.predict(np.expand_dims(resize/255, 0))
if yhat > 0.5:
    print(f'Wskazany obraz został sklasyfikowany jako czesc prawidlowa')
else:
    print(f'Wskazany obraz został sklasyfikowany jako czesc uszkodzona')
```

Dzięki zapisywaniu i wczytywaniu modelu mamy możliwość przechowywania i wykorzystywania wyników treningu w dowolnym momencie, co pozwala na efektywniejsze wy-

korzystanie zasobów obliczeniowych oraz sprawne wdrażanie modeli do praktycznych zastosowań.

## **Rozdział 6. Badania**

### **6.1 Przedmiot badań**

Przedmiotem badań są zdjęcia przedstawiające różne wady produkcyjne oraz model głębokiego uczenia służący do ich rozpoznawania. Zdjęcia zostały podzielone na dwie klasy: prawidłowe i uszkodzone. W ramach badań wykorzystano 20 zdjęć testowych, z których 10 przedstawia prawidłowe części, a pozostałe 10 to zdjęcia uszkodzonych elementów. Eksperymenty przeprowadzono na modelach wyuczonych na różnej liczbie zdjęć, konkretnie na 25, 50, 100, 250 i 500 zdjęciach, aby zbadać wpływ liczby zdjęć użytych do treningu na skuteczność modelu.

### **6.2 Cele badań**

Celem badań jest ocena skuteczności modeli głębokiego uczenia w rozpoznawaniu wad produkcyjnych na zdjęciach, w zależności od liczby zdjęć użytych do wyuczenia modelu. W ramach badań analizowana jest dokładność modeli w rozpoznawaniu zarówno prawidłowych, jak i uszkodzonych części. Badania te mają na celu ocenić, czy model jest w stanie skutecznie klasyfikować zdjęcia w zależności od liczby danych uczących, co może prowadzić do wniosków dotyczących optymalnej liczby zdjęć potrzebnych do skutecznego nauczenia modelu.

### **6.3 Środowisko badawcze**

Do przeprowadzenia badań użyto następujących narzędzi, bibliotek i środowiska programistycznego:

- TensorFlow
- Keras

- OpenCV
- Python

Tabela 6.1 zawiera szczegółowe informacje na temat konfiguracji sprzętowej i systemu operacyjnego maszyny, na której przeprowadzono badania.

Tabela 6.1: Specyfikacja techniczna maszyny użytej do badań

Komponent	Specyfikacja
Procesor	Intel Core i7-12700K
Pamięć RAM	32 GB DDR4
Karta graficzna	Nvidia RTX 3080 Ti
System operacyjny	Fedora 37, Fedora 38

Źródło: opracowanie własne

## 6.4 Metoda badawcza

W celu oceny skuteczności modeli zastosowano następujące metody:

- **Analiza straty i dokładności:** porównanie wartości straty (błędu) i dokładności (poprawnych klasyfikacji) dla danych treningowych i walidacyjnych w trakcie procesu uczenia. Wartości te są zapisywane podczas każdej epoki treningowej, co pozwala na obserwację, jak model się uczy, i może pomóc w identyfikacji problemów, takich jak przetrenowanie.
- **Analiza przypadków sukcesów i błędów:** szczegółowa analiza poszczególnych przypadków, w których model dokonał poprawnej lub błędnej klasyfikacji. Przeglądanie tych przypadków może pomóc w zrozumieniu, jakie cechy zdjęć wpływają na decyzje modelu i gdzie model może się mylić.

## 6.5 Trenowanie modelu

Trenowanie modelu polega na uczeniu sieci neuronowej na podstawie dostarczonego zbioru danych uczących. W trakcie procesu trenowania, model optymalizuje swoje wag, aby zminimalizować stratę wynikającą z różnicy między prognozami modelu a rzeczywistymi etykietami danych uczących.

Fragment kodu odpowiadający za trenowanie modelu:

```
hist = model.fit(train, epochs=15, validation_data=val,
callbacks=[tensorboard_callback])
```

## 6.6 Walidacja

Walidacja to proces oceny wydajności modelu na podzbiorze danych, który nie był używany do trenowania modelu. Walidacja pozwala na monitorowanie postępów uczenia się modelu, a także na wykrywanie sytuacji, w których model jest przetrenowany. W trakcie walidacji model nie jest aktualizowany, a jedynie oceniany na podstawie danych walidacyjnych.

## 6.7 Testowanie

Testowanie polega na ocenie wydajności nauczonego modelu na zupełnie nowym, nieznanym zbiorze danych (zbiór testowy). Proces ten pozwala na rzeczywistą ocenę, jak dobrze model radzi sobie z prognozowaniem na danych, które nie były używane ani do trenowania, ani do walidacji. Wyniki testowania mogą dać szacunkową informację na temat tego, jak model poradzi sobie z danymi napotkanymi w rzeczywistych zastosowaniach.

Fragment kodu odpowiadający za testowanie modelu:

```
img = cv2.imread('testing/failure_10.png')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
yhat = model.predict(np.expand_dims(resize/255, 0))
if yhat > 0.5:
    print(f'Wskazany obraz zostal sklasyfikowany jako czesc prawidlowa')
else:
    print(f'Wskazany obraz zostal sklasyfikowany jako czesc uszkodzona')
```

## **6.8 Materiały badawcze**

W niniejszym podrozdziale przedstawione zostaną materiały badawcze wykorzystane do przeprowadzenia eksperymentów. Wykorzystano 20 zdjęć testowych podzielonych na dwie klasy: prawidłowe i uszkodzone. Każda z klas zawiera 10 zdjęć. Materiały te zostały wygenerowane przy użyciu sztucznej inteligencji na wzór zdjęć z rzeczywistego procesu produkcyjnego.

Zdjęcia te symulują różne sytuacje, które mogą wystąpić w rzeczywistości, takie jak różne rodzaje wad, różnorodność kształtów i rozmiarów części oraz różne warunki oświetleniowe. Wykorzystanie takich zdjęć do przeprowadzenia badań pozwala na ocenę skuteczności modeli głębokiego uczenia w warunkach zbliżonych do rzeczywistych, co może prowadzić do bardziej wiarygodnych wyników i przydatnych wniosków.

Wykorzystanie zdjęć wygenerowanych przez sztuczną inteligencję pozwala również na kontrolowanie liczby danych uczących oraz dokładne dopasowanie stopnia trudności zdjęć. W ten sposób można zbadać wpływ liczby zdjęć użytych do treningu na skuteczność modelu oraz zrozumieć, jakie cechy zdjęć wpływają na decyzje modelu i gdzie model może się mylić.

W pracy przedstawiono zbiór zdjęć produktów użytych do testowania. Zdjęcia podzielone są na dwie kategorie: prawidłowe produkty oraz uszkodzone produkty.

Na zdjęciach prawidłowych produktów (Rys. 6.1–6.10) prezentowane są różne typy produktów spełniających wymagane kryteria jakości. Ich celem jest prezentacja dobrych praktyk produkcyjnych oraz jako punkt odniesienia dla porównania z uszkodzonymi produktami.

Natomiast na zdjęciach uszkodzonych produktów (Rys. 6.11–6.20) przedstawiono przykłady wadliwych produktów, które nie spełniają wymagań jakościowych. Mogą one zawierać pęknięcia, złamania, nierówności, deformacje czy inne wady, które wpływają na ich funkcjonalność i estetykę.

Rysunek 6.1: Prawidłowe - zdjęcie 1



Źródło: opracowanie własne

Rysunek 6.2: Prawidłowe - zdjęcie 2



Źródło: opracowanie własne

Rysunek 6.3: Prawidłowe - zdjęcie 3



Źródło: opracowanie własne

Rysunek 6.4: Prawidłowe - zdjęcie 4



Źródło: opracowanie własne

Rysunek 6.5: Prawidłowe - zdjęcie 5



Źródło: opracowanie własne

Rysunek 6.6: Prawidłowe - zdjęcie 6



Źródło: opracowanie własne

Rysunek 6.7: Prawidłowe - zdjęcie 7



Źródło: opracowanie własne

Rysunek 6.8: Prawidłowe - zdjęcie 8



Źródło: opracowanie własne

Rysunek 6.9: Prawidłowe - zdjęcie 9



Źródło: opracowanie własne

Rysunek 6.10: Prawidłowe - zdjęcie 10



Źródło: opracowanie własne

Rysunek 6.11: Uszkodzone - zdjęcie 1



Źródło: opracowanie własne

Rysunek 6.12: Uszkodzone - zdjęcie 2



Źródło: opracowanie własne

Rysunek 6.13: Uszkodzone - zdjęcie 3



Źródło: opracowanie własne

Rysunek 6.14: Uszkodzone - zdjęcie 4



Źródło: opracowanie własne

Rysunek 6.15: Uszkodzone - zdjęcie 5



Źródło: opracowanie własne

Rysunek 6.16: Uszkodzone - zdjęcie 6



Źródło: opracowanie własne

Rysunek 6.17: Uszkodzone - zdjęcie 7



Źródło: opracowanie własne

Rysunek 6.18: Uszkodzone - zdjęcie 8



Źródło: opracowanie własne

Rysunek 6.19: Uszkodzone - zdjęcie 9



Źródło: opracowanie własne

Rysunek 6.20: Uszkodzone - zdjęcie 10



Źródło: opracowanie własne

## Rozdział 7. Wyniki i analiza badań

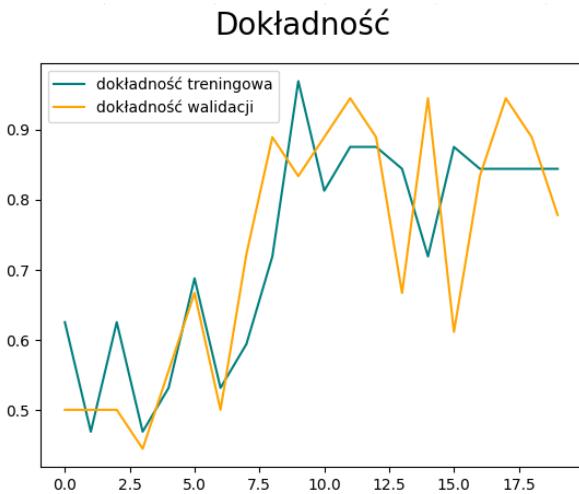
### 7.1 Wyniki dla modelu uczonego na 25 zdjęciach

Wykres dokładności modelu, który był uczyony na 25 zdjęciach przez 20 epok, prezentuje interesujący profil uczenia się (Rys. 7.1). Początkowe wartości dokładności, poniżej 0.7 do 7 epoki, sugerują, że model miał na początku trudności z nauką na dostarczonych danych. Taka sytuacja jest całkiem normalna, zwłaszcza na początku procesu uczenia.

Następnie w epokach od 7 do 15 obserwujemy wahania wartości dokładności, które kształtują się w granicach od 0.7 do 0.9. Ten wzrost i spadek dokładności może świadczyć o tym, że model próbował znaleźć optymalne rozwiązanie w przestrzeni parametrów. Wahania mogą być spowodowane także niewielką ilością danych uczących, co może prowadzić do większej zmienności wyników.

W końcu, wartości dokładności walidacyjnej i testowej są blisko siebie, oscylując między 0.8 a 0.9. To jest dobry znak, sugerujący, że model dobrze się generalizuje i jest w stanie skutecznie przewidywać wyniki na nowych danych. Jednakże, warto pamiętać, że pomimo wysokiej dokładności na końcu, model był uczyony na stosunkowo niewielkim zbiorze danych, co może ograniczać jego zdolność do generalizacji na bardziej zróżnicowanych danych.

Rysunek 7.1: Wykres dokładności dla modelu uczonego na 25 zdjęciach



Źródło: opracowanie własne

Wykres straty dla modelu uczonego na 25 zdjęciach przez 20 epok pokazuje, że model poprawiał swoje wyniki na przestrzeni epok, co jest pożądanym zachowaniem (Rys. 7.2).

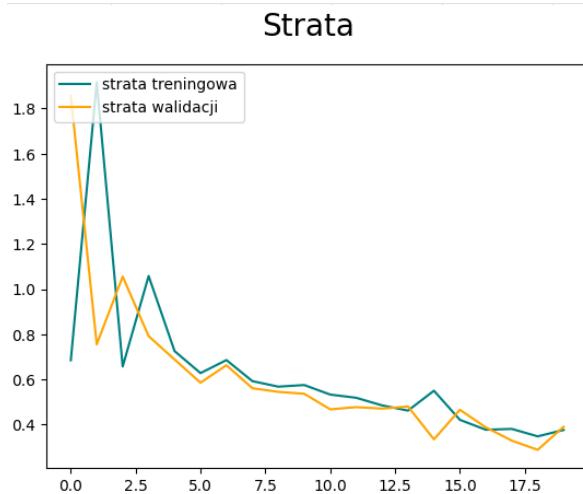
Na początku procesu uczenia, strata treningowa zaczęła od wartości 0.8, a następnie skoczyła do 1.8. Strata walidacyjna od początku wynosiła 1.8. Wysokie początkowe wartości straty są typowe dla początkowych etapów uczenia sieci neuronowych. Model na początku jest jeszcze "niewykształcony" i ma tendencję do generowania dużych błędów.

W kolejnych epokach, wartości straty treningowej i walidacyjnej stopniowo malały, co świadczy o tym, że model poprawiał swoją zdolność do prognozowania etykiet w danych treningowych i walidacyjnych. To jest pozytywny wynik, pokazujący, że model uczył się od swoich błędów i poprawiał swoje przewidywania.

Spadek straty w takim samym tempie zarówno dla danych treningowych, jak i walidacyjnych, sugeruje, że model dobrze się generalizuje i nie ma oznak przetrenowania. Przetrenowanie zazwyczaj objawia się dużym spadkiem straty treningowej i niewielkim spadkiem lub nawet wzrostem straty walidacyjnej.

W końcowej, 20 epoce, straty wynosiły poniżej 0.4, co wskazuje na znaczną poprawę w porównaniu do początkowych epok. Jednak nadal jest to stosunkowo wysoka wartość straty i możliwe, że dalsze uczenie lub optymalizacja modelu mogłaby doprowadzić do jej dalszego zmniejszenia.

Rysunek 7.2: Wykres straty dla modelu uczonego na 25 zdjęciach



Źródło: opracowanie własne

W tabelach zostały przedstawione wyniki testów na zdjęciach z produktami prawidłowymi i uszkodzonymi dla modelu uczonego na 25 zdjęciach. Wartości od 0 do 0.5 zostały przypisane do klasy uszkodzone, a wartości powyżej 0.5 do klasy prawidłowe. Poprawność wyników została oceniona i odpowiednio zapisana.

Analiza wyników testów dla modelu uczonego na 25 zdjęciach pokazuje, że model osiąga zróżnicowaną skuteczność w ocenie zdjęć z produktami prawidłowymi i uszkodzonymi.

Dla zdjęć z produktami prawidłowymi (Tabela 7.1), model poprawnie ocenił 9 na 10 przypadków, co daje 90% skuteczności. Tylko w teście numer 2 model niepoprawnie ocenił element jako "Uszkodzony", gdy był prawidłowy.

W przypadku zdjęć z produktami uszkodzonymi (Tabela 7.2), model osiągnął niższą skuteczność, poprawnie oceniając 6 na 10 przypadków, co daje 60% skuteczności. Błędne oceny wystąpiły w testach 1, 3, 6 oraz 10, gdzie model ocenił uszkodzone produkty jako "Prawidłowe".

Ogólna skuteczność modelu uczonego na 25 zdjęciach wynosi 75% (15 poprawnych ocen na 20 prób). Można zauważyć, że model ma tendencję do lepszego rozpoznawania prawidłowych elementów niż uszkodzonych.

Należy jednak zauważyc, że model był uczony tylko na 25 zdjęciach, co jest stosunkowo małą liczbą przykładów uczących.

Tabela 7.1: Wyniki testów na zdjęciach z produktami prawidłowymi dla modelu uczonego na 25 zdjęciach

Test	Wynik - Prawidłowe	Ocena	Poprawność oceny
1	0.9185279	Prawidłowy	Tak
2	0.16176112	Uszkodzony	Nie
3	0.84514433	Prawidłowy	Tak
4	0.8351546	Prawidłowy	Tak
5	0.8647871	Prawidłowy	Tak
6	0.72340083	Prawidłowy	Tak
7	0.8165024	Prawidłowy	Tak
8	0.81942993	Prawidłowy	Tak
9	0.7422247	Prawidłowy	Tak
10	0.8814195	Prawidłowy	Tak

Źródło: opracowanie własne

Tabela 7.2: Wyniki testów na zdjęciach z produktami uszkodzonymi dla modelu uczonego na 25 zdjęciach

Test	Wynik - Uszkodzone	Ocena	Poprawność oceny
1	0.54751855	Prawidłowy	Nie
2	0.24815	Uszkodzony	Tak
3	0.67537665	Prawidłowy	Nie
4	0.2928825	Uszkodzony	Tak
5	0.1816359	Uszkodzony	Tak
6	0.5924341	Prawidłowy	Nie
7	0.33428523	Uszkodzony	Tak
8	0.19617	Uszkodzony	Tak
9	0.25611275	Uszkodzony	Tak
10	0.74181026	Prawidłowy	Nie

Źródło: opracowanie własne

## 7.2 Wyniki dla modelu uczonego na 50 zdjęciach

Wykres dokładności modelu uczonego na 50 zdjęciach przez 20 epok pokazuje ogólnie pozytywną tendencję wzrostu dokładności w czasie treningu (Rys. 7.3).

Na początku, wartości dokładności rozpoczęły się na niskim poziomie 0.2. Jest to typowe dla wczesnych etapów uczenia sieci neuronowych, kiedy model nie jest jeszcze w stanie dobrze klasyfikować danych. W kolejnych epokach, dokładność treningowa rosła stopniowo, choć z drobnymi wachaniami.

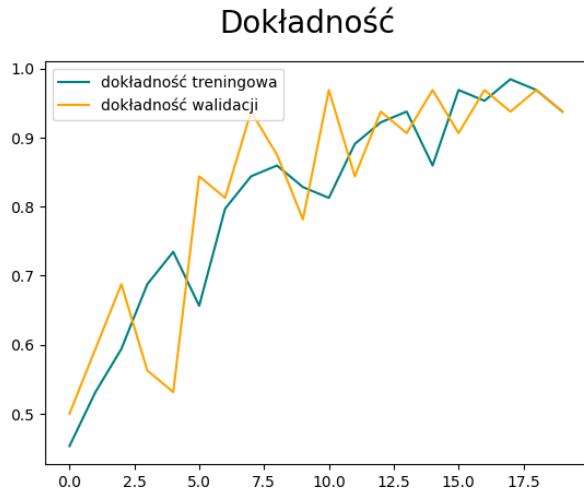
Między epoką 3 a 5 zaobserwowano istotny wzrost dokładności treningowej, z 0.4 do około 0.8. W kolejnych epokach wachania dokładności treningowej były mniejsze, wynoszące

około 0.1.

W przypadku dokładności walidacyjnej, wzrost był bardziej stopniowy, a wachania były mniejsze niż dla dokładności treningowej. To jest pożądane zachowanie, ponieważ oznacza to, że model dobrze generalizuje swoją wiedzę na dane walidacyjne, a nie tylko "zapamiętuje" dane treningowe.

W końcowych epokach, wartości dokładności treningowej i walidacyjnej wahały się między 0.9 a 1. Wysoka dokładność wskazuje, że model jest efektywny w rozpoznawaniu wad produkcyjnych na podstawie dostarczonych danych. Niemniej jednak, warto zwrócić uwagę na możliwość przetrenowania, zwłaszcza gdy dokładność osiąga wartość 1.

Rysunek 7.3: Wykres dokładności dla modelu uczonego na 50 zdjęciach



Źródło: opracowanie własne

Wykres straty modelu uczonego na 50 zdjęciach przez 20 epok pokazuje, że model poprawiał się w trakcie procesu uczenia, co jest zgodne z oczekiwanyim zachowaniem (Rys. 7.4).

Na początku procesu uczenia, wartości straty były dość wysokie, wynosząc około 0.8. Wysoka strata na początku jest typowym zjawiskiem podczas uczenia sieci neuronowych, kiedy model jest jeszcze słabo dopasowany do danych.

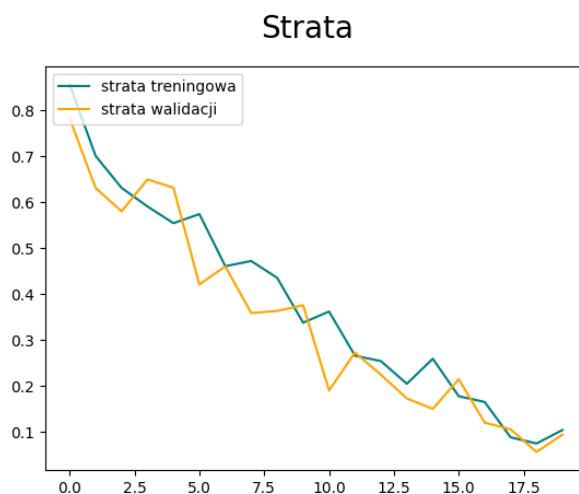
Jednakże, z każdą kolejną epoką, wartości straty systematycznie malały. To wskazuje, że model poprawiał swoje predykcje, minimalizując różnice między prognozowanymi a rzeczywistymi etykietami.

Spadek straty był stosunkowo jednolity, co sugeruje, że proces uczenia przebiegał stabilnie. Nie zaobserwowano dużych skoków czy gwałtownych zmian w wartościach straty, co mogłoby sugerować problemy, takie jak zbyt duży współczynnik uczenia.

W końcowej, 20. epoce, wartości straty osiągnęły poziom około 0.1. Jest to znaczne zmniejszenie w porównaniu do początkowych wartości, co sugeruje, że model jest efektywny w rozpoznawaniu wad produkcyjnych na podstawie dostarczonych danych.

Wykres straty pokazuje, że model skutecznie uczył się i poprawiał swoje predykcje w trakcie procesu uczenia. Jednakże, niskie wartości straty na końcu procesu uczenia mogą sugerować, że model jest potencjalnie przetrenowany.

Rysunek 7.4: Wykres straty dla modelu uczonego na 50 zdjęciach



Źródło: opracowanie własne

W tabelach zostały przedstawione wyniki testów na zdjęciach z produktami prawidłowymi i uszkodzonymi dla modelu uczonego na 50 zdjęciach. Wartości od 0 do 0.5 zostały przypisane do klasy uszkodzone, a wartości powyżej 0.5 do klasy prawidłowe. Poprawność wyników została oceniona i odpowiednio zapisana.

Analiza wyników testów dla modelu uczonego na 50 zdjęciach pokazuje, że model osiąga lepszą skuteczność w ocenie zdjęć z produktami uszkodzonymi.

Dla zdjęć z produktami prawidłowymi (Tabela 7.3), model poprawnie ocenił 6 na 10 przypadków, co daje 60% skuteczności. Błędne oceny wystąpiły w testach 3, 4, 7 i 9 gdzie model niepoprawnie ocenił element jako "Uszkodzony", gdy był prawidłowy.

W przypadku zdjęć z produktami uszkodzonymi (Tabela 7.4), model osiągnął większą skuteczność niż model uczonego na 25 zdjęciach, poprawnie oceniając 8 na 10 przypadków, co daje 80% skuteczności. Błędne oceny wystąpiły w testach 1 i 8, gdzie model ocenił uszkodzone produkty jako "Prawidłowe".

Ogólna skuteczność modelu uczonego na 50 zdjęciach wynosi 70% (14 poprawnych ocen

na 20 prób), jest to wynik mniejszy niż dla modelu uczonego na 25 zdjęciach. Jednak warto zauważać, że model uczonego na 50 zdjęciach osiągnął wyższą skuteczność w ocenie zdjęć z produktami uszkodzonymi.

Tabela 7.3: Wyniki testów na zdjęciach z produktami prawidłowymi dla modelu uczonego na 50 zdjęciach

Test	Wynik	Ocena	Poprawność
1	0.84990263	Prawidłowy	Tak
2	0.9798704	Prawidłowy	Tak
3	0.03639716	Uszkodzony	Nie
4	0.05641427	Uszkodzony	Nie
5	0.9491898	Prawidłowy	Tak
6	0.7077886	Prawidłowy	Tak
7	0.2013224	Uszkodzony	Nie
8	0.88786936	Prawidłowy	Tak
9	0.04853454	Uszkodzony	Nie
10	0.9503239	Prawidłowy	Tak

Źródło: opracowanie własne

Tabela 7.4: Wyniki testów na zdjęciach z produktami uszkodzonymi dla modelu uczonego na 50 zdjęciach

Test	Wynik	Ocena	Poprawność
1	0.99400294	Prawidłowy	Nie
2	0.00386766	Uszkodzony	Tak
3	0.01674952	Uszkodzony	Tak
4	0.00414581	Uszkodzony	Tak
5	0.02043912	Uszkodzony	Tak
6	0.00069023	Uszkodzony	Tak
7	0.01667762	Uszkodzony	Tak
8	0.9827842	Prawidłowy	Nie
9	0.01187711	Uszkodzony	Tak
10	0.42052767	Uszkodzony	Tak

Źródło: opracowanie własne

### 7.3 Wyniki dla modelu uczonego na 100 zdjęciach

Wykres dokładności dla modelu uczonego na 100 zdjęciach przez 20 epok pokazuje stopniową poprawę dokładności zarówno na danych treningowych, jak i walidacyjnych w trakcie procesu uczenia (Rys. 7.5).

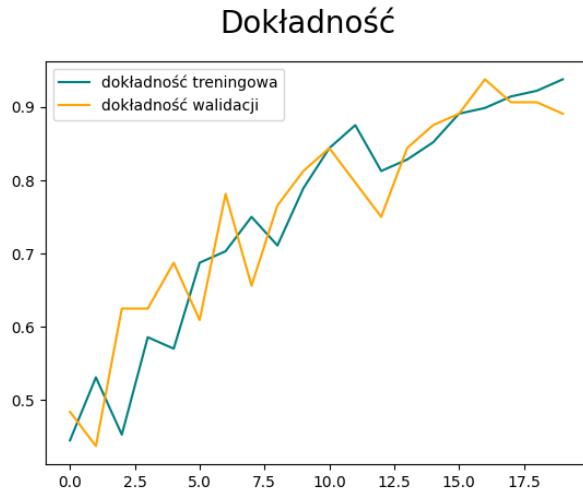
Na początku, dokładność modelu wynosiła około 0.3, co oznacza, że model prawidłowo klasyfikował około 30% próbek. Stopniowy wzrost dokładności wskazuje na to, że model

poprawiał swoje zdolności predykcyjne z każdą kolejną epoką.

Nie zaobserwowano znaczących skoków dokładności, co sugeruje, że proces uczenia przebiegał bez większych problemów. Istotne jest, że dokładność na danych walidacyjnych rosła równomiernie z dokładnością na danych treningowych. To jest dobrym znakiem, ponieważ sugeruje, że model nie był przetrenowany.

W końcowych epokach, dokładność modelu wahała się między 0.8 a 0.9. To oznacza, że model poprawnie klasyfikował około 80-90% próbek. Jest to dość wysoki poziom dokładności, szczególnie biorąc pod uwagę, że model był uczony na stosunkowo małym zestawie danych.

Rysunek 7.5: Wykres dokładności dla modelu uczonego na 100 zdjęciach



Źródło: opracowanie własne

Wykres straty dla modelu uczonego na 100 zdjęciach przez 20 epok pokazuje, jak model poprawiał się w trakcie procesu uczenia (Rys. 7.6). Wysokie początkowe wartości straty (1.6 dla treningowej i 1.3 dla walidacyjnej) wskazują, że na początku procesu uczenia model popełniał wiele błędów w prognozowaniu klas obrazów. To jest spodziewane, ponieważ na początku uczenia, wagi modelu są zazwyczaj inicjalizowane losowo, co prowadzi do losowych prognoz.

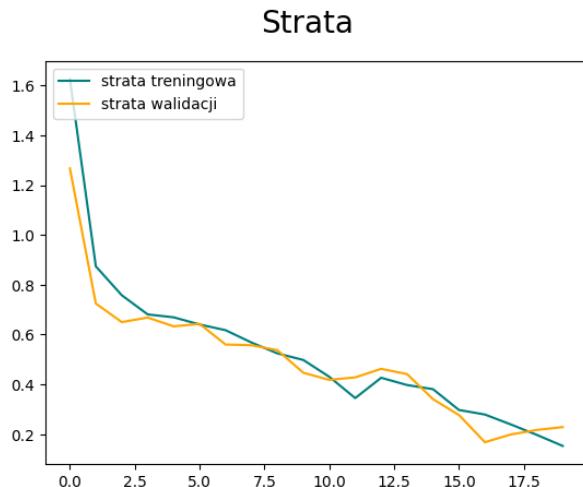
Jednak z każdą kolejną epoką, wartości straty zarówno na danych treningowych, jak i walidacyjnych stopniowo malały. To jest dobry znak, ponieważ pokazuje, że model poprawiał swoje zdolności predykcyjne w trakcie uczenia.

Równomierny spadek wartości straty dla danych treningowych i walidacyjnych jest pozytywnym zjawiskiem. Oznacza to, że model był w stanie dobrze generalizować nauczoną

wiedzę na nieznane wcześniej dane.

W końcowych epokach, wartości straty wynosiły między 0.4 a 0.2. To oznacza, że model był w stanie poprawnie przewidzieć klasy obrazów z relatywnie niskim błędem.

Rysunek 7.6: Wykres straty dla modelu uczonego na 100 zdjęciach



Źródło: opracowanie własne

W tabelach zostały przedstawione wyniki testów na zdjęciach z produktami prawidłowymi i uszkodzonymi dla modelu uczonego na 100 zdjęciach. Wartości od 0 do 0.5 zostały przypisane do klasy uszkodzone, a wartości powyżej 0.5 do klasy prawidłowe. Poprawność wyników została oceniona i odpowiednio zapisana.

Analiza wyników testów dla modelu uczonego na 100 zdjęciach pokazuje, że model osiąga lepszą skuteczność w ocenie zdjęć z produktami prawidłowymi niż uszkodzonymi, jednak ogólna skuteczność jest lepsza niż w przypadku modeli uczonych na 25 i 50 zdjęciach.

Dla zdjęć z produktami prawidłowymi (Tabela 7.5), model poprawnie ocenił 7 na 10 przypadków, co daje 70% skuteczności. Błędne oceny wystąpiły w testach 6, 7 i 9, gdzie model niepoprawnie ocenił element jako "Uszkodzony", gdy był prawidłowy.

W przypadku zdjęć z produktami uszkodzonymi (Tabela 7.6), model osiągnął większą skuteczność niż w przypadku modeli uczonych na 25 i 50 zdjęciach, poprawnie oceniając 8 na 10 przypadków, co daje 80% skuteczności. Błędne oceny wystąpiły w testach 1 i 8, gdzie model ocenił uszkodzone produkty jako "Prawidłowe".

Ogólna skuteczność modelu uczonego na 100 zdjęciach wynosi 75% (15 poprawnych ocen na 20 prób), tak samo jak dla modelu uczonego na 50 zdjęciach. Jednak warto zauważyć, że model uczonego na 100 zdjęciach osiągnął wyższą skuteczność w ocenie zdjęć z

produktami uszkodzonymi niż model uczonego na 25 zdjęciach.

Podobnie jak dla modeli uczonych na mniejszej liczbie zdjęć, zwiększenie liczby zdjęć uczących oraz dalsze strojenie hiperparametrów modelu mogłyby pomóc w poprawie ogólnej skuteczności, szczególnie w rozpoznawaniu uszkodzonych elementów.

Tabela 7.5: Wyniki testów na zdjęciach z produktami prawidłowymi dla modelu uczonego na 100 zdjęciach

Test	Wynik - Prawidłowe	Ocena	Poprawność
1	0.9864047	Prawidłowy	Tak
2	0.812722	Prawidłowy	Tak
3	0.8298837	Prawidłowy	Tak
4	0.8440731	Prawidłowy	Tak
5	0.9554931	Prawidłowy	Tak
6	0.44777694	Uszkodzony	Nie
7	0.2929827	Uszkodzony	Nie
8	0.73627317	Prawidłowy	Tak
9	0.20005107	Uszkodzony	Nie
10	0.94812363	Prawidłowy	Tak

Źródło: opracowanie własne

Tabela 7.6: Wyniki testów na zdjęciach z produktami uszkodzonymi dla modelu uczonego na 100 zdjęciach

Test	Wynik - Uszkodzone	Ocena	Poprawność
1	0.9690722	Prawidłowy	Nie
2	0.04128061	Uszkodzony	Tak
3	0.01292709	Uszkodzony	Tak
4	0.01372818	Uszkodzony	Tak
5	0.0035921	Uszkodzony	Tak
6	0.00410713	Uszkodzony	Tak
7	0.09529652	Uszkodzony	Tak
8	0.59629834	Prawidłowy	Nie
9	0.1106549	Uszkodzony	Tak
10	0.40220806	Uszkodzony	Tak

Źródło: opracowanie własne

## 7.4 Wyniki dla modelu uczonego na 250 zdjęciach

Wykres dokładności dla modelu uczonego na 250 zdjęciach pokazuje, że model generalizował dobrze swoją zdolność do klasyfikacji obrazów (Rys. 7.7).

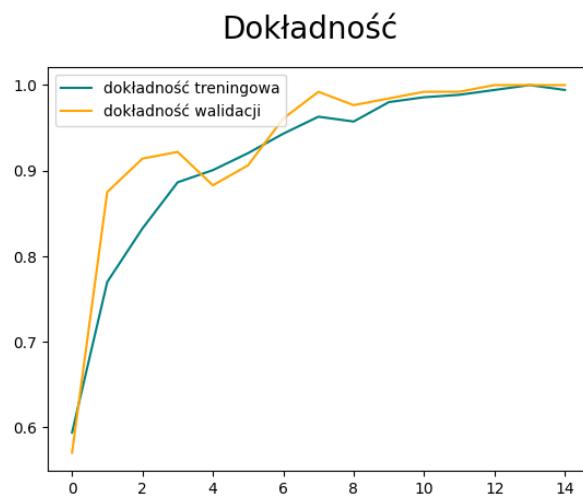
Wartości dokładności rozpoczynają się od 0.5 i stopniowo rosną wraz z kolejnymi epokami, co wskazuje na poprawę zdolności klasyfikacyjnych modelu. Wartości dokładności,

zarówno dla danych treningowych, jak i walidacyjnych, są do siebie bardzo bliskie przez cały czas uczenia, co jest dobrym znakiem. Oznacza to, że model nie tylko dobrze uczył się na danych treningowych, ale był również w stanie dobrze generalizować nauczoną wiedzę na nieznane wcześniej dane (dane walidacyjne).

Fakt, że wykres ma kształtłyżwy, sugeruje, że model szybko nauczył się klasyfikować obrazy w początkowych epokach, a potem jego dokładność stopniowo się poprawiała, ale w wolniejszym tempie.

W końcowych epokach, wartości dokładności są bardzo blisko 1, co wskazuje na bardzo wysoką zdolność modelu do poprawnej klasyfikacji obrazów. To jest znakomity wynik, który wskazuje na skuteczność zastosowanego podejścia.

Rysunek 7.7: Wykres dokładności dla modelu uczonego na 250 zdjęciach



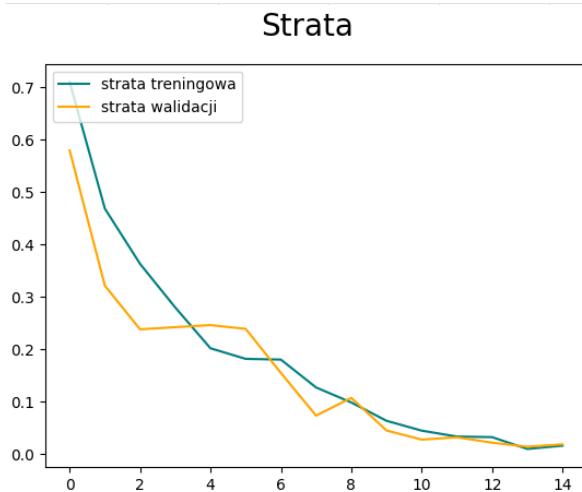
Źródło: opracowanie własne

Na początku, wartość straty dla zarówno zbioru treningowego jak i walidacyjnego były dość wysokie (0.7 dla treningowego i 0.6 dla walidacyjnego), co jest typowe dla początkowej fazy uczenia modelu (Rys. 7.8). Jest to oczekiwane, ponieważ model dopiero rozpoczyna uczenie się z danych i jego predykcje są jeszcze dalekie od prawidłowych.

Jednak z każdą kolejną epoką, wartości straty dla obu zestawów danych systematycznie maleją, co wskazuje na to, że model poprawiał swoje predykcje. Wartości straty dla danych treningowych i walidacyjnych maleją w podobnym tempie i są bardzo blisko siebie przez cały czas, co jest pozytywnym znakiem. Oznacza to, że model nie tylko dobrze uczył się z danych treningowych, ale również był w stanie skutecznie generalizować swoją wiedzę na nieznane wcześniej dane (dane walidacyjne).

W końcowych epokach, wartości straty są bardzo niskie (między 0.1 a 0), co wskazuje na to, że model jest w stanie dokonywać bardzo precyzyjnych predykcji. Niskie wartości straty na końcu procesu uczenia wskazują na wysoką dokładność modelu.

Rysunek 7.8: Wykres straty dla modelu uczonego na 250 zdjęciach



Źródło: opracowanie własne

W tabelach zostały przedstawione wyniki testów na zdjęciach z produktami prawidłowymi i uszkodzonymi dla modelu uczonego na 250 zdjęciach. Wartości od 0 do 0.5 zostały przypisane do klasy uszkodzone, a wartości powyżej 0.5 do klasy prawidłowe. Poprawność wyników została oceniona i odpowiednio zapisana.

Analiza wyników testów dla modelu uczonego na 250 zdjęciach przez 15 epok pokazuje, że model osiąga lepszą skuteczność w ocenie zdjęć z produktami uszkodzonymi niż prawidłowymi, a ogólna skuteczność jest lepsza niż w przypadku modeli uczonych na 25, 50 i 100 zdjęciach.

Dla zdjęć z produktami prawidłowymi (Tabela 7.7), model poprawnie ocenił 8 na 10 przypadków, co daje 80% skuteczności. Błędne oceny wystąpiły w testach 2 i 6, gdzie model niepoprawnie ocenił element jako "Uszkodzony", gdy był prawidłowy.

W przypadku zdjęć z produktami uszkodzonymi (Tabela 7.8), model osiągnął większą skuteczność niż w przypadku modeli uczonych na 25, 50 i 100 zdjęciach, poprawnie oceniając wszystkie 10 przypadków, co daje 100% skuteczności.

Ogólna skuteczność modelu uczonego na 250 zdjęciach wynosi 90% (18 poprawnych ocen na 20 prób), co jest wyższym wynikiem w porównaniu z modelami uczonego na 25, 50 i 100 zdjęciach.

Podobnie jak dla modeli uczonych na mniejszej liczbie zdjęć, zwiększenie liczby zdjęć uczących oraz dalsze strojenie hiperparametrów modelu mogłoby pomóc w poprawie ogólnej skuteczności, szczególnie w rozpoznawaniu prawidłowych elementów.

Tabela 7.7: Wyniki testów na zdjęciach z produktami prawidłowymi na 250 zdjęciach

Test	Wynik - Prawidłowe	Ocena	Poprawność
1	0.99992514	Prawidłowy	Tak
2	0.00049097	Uszkodzony	Nie
3	0.99903715	Prawidłowy	Tak
4	0.9983879	Prawidłowy	Tak
5	0.97826487	Prawidłowy	Tak
6	0.2164898	Uszkodzony	Nie
7	0.9998957	Prawidłowy	Tak
8	0.99057996	Prawidłowy	Tak
9	0.7195368	Prawidłowy	Tak
10	0.9921462	Prawidłowy	Tak

Źródło: opracowanie własne

Tabela 7.8: Wyniki testów na zdjęciach z produktami uszkodzonymi na 250 zdjęciach

Test	Wynik - Uszkodzone	Ocena	Poprawność
1	0.14923318	Uszkodzony	Tak
2	2.871099e-08	Uszkodzony	Tak
3	9.253375e-05	Uszkodzony	Tak
4	4.3932796e-06	Uszkodzony	Tak
5	1.2867513e-08	Uszkodzony	Tak
6	1.791459e-08	Uszkodzony	Tak
7	0.00028085	Uszkodzony	Tak
8	0.00098794	Uszkodzony	Tak
9	2.8661802e-06	Uszkodzony	Tak
10	0.00594379	Uszkodzony	Tak

Źródło: opracowanie własne

## 7.5 Wyniki dla modelu uczonego na 500 zdjęciach

Wykres dokładności modelu, który był trenowany na 500 zdjęciach przez 15 epok, wskazuje na bardzo dobrą zdolność modelu do klasyfikacji obrazów (Rys. 7.9).

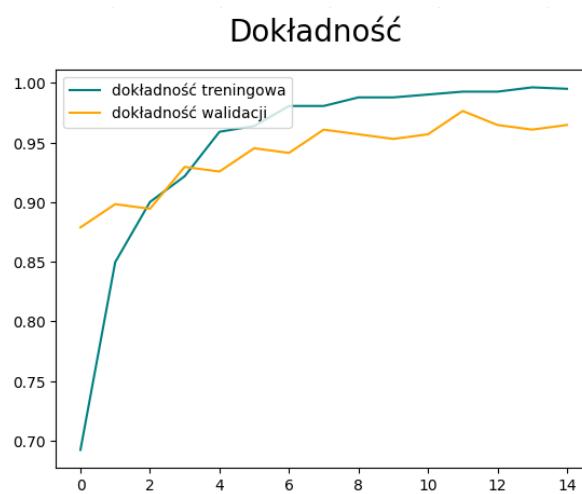
Na początku procesu uczenia, dokładność dla danych treningowych wynosiła 0.7, natomiast dla danych walidacyjnych - 0.89. Różnica między nimi może wynikać z różnic w

dystrybucji danych, jednak obie wartości są dość wysokie już na tym etapie, co sugeruje, że model już na początku był w stanie dobrze klasyfikować obrazy.

W miarę postępu procesu uczenia, dokładność modelu na danych treningowych i walidacyjnych stale rosła, co jest oznaką skutecznego uczenia. Warto zauważyć, że pomimo drobnych wahań, ogólny trend wzrostowy był konsekwentny i stopniowy, bez znaczących skoków, co wskazuje na stabilność procesu uczenia.

W końcowych epokach, dokładność modelu na danych treningowych zbliżała się do 1, co oznacza, że model prawie idealnie klasyfikował obrazy z zestawu treningowego. Dokładność na danych walidacyjnych wynosiła 0.95, co jest również bardzo wysokim wynikiem i sugeruje, że model dobrze generalizuje swoją wiedzę na nowe, nieznane wcześniej dane.

Rysunek 7.9: Wykres dokładności dla modelu uczonego na 500 zdjęciach



Źródło: opracowanie własne

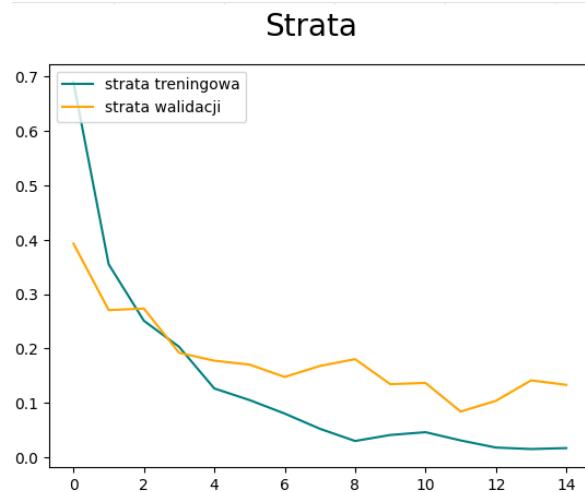
Wykres straty modelu, który był trenowany na 500 zdjęciach przez 15 epok, wskazuje na efektywny proces uczenia (Rys. 7.10).

Na początku procesu uczenia, strata dla danych treningowych wynosiła 0.7, natomiast dla danych walidacyjnych - 0.4. Ta różnica może wynikać z różnic w dystrybucji danych między zestawem treningowym i walidacyjnym. Warto zauważyć, że pomimo tej różnicy, wartości straty były na akceptowalnym poziomie już na początku procesu uczenia, co sugeruje, że model już na tym etapie był w stanie dość dobrze klasyfikować obrazy.

W miarę postępu procesu uczenia, strata modelu na danych treningowych i walidacyjnych stale malała, co jest oznaką skutecznego uczenia. Ogólny trend spadkowy był konsekwentny i stopniowy, co wskazuje na stabilność procesu uczenia.

W końcowych epokach, strata modelu na danych treningowych zbliżała się do 0, co oznacza, że model prawie idealnie klasyfikował obrazy z zestawu treningowego. Strata na danych walidacyjnych wynosiła 0.2, co jest również niskim wynikiem i sugeruje, że model dobrze generalizuje swoją wiedzę na nowe, nieznane wcześniej dane.

Rysunek 7.10: Wykres straty dla modelu uczonego na 500 zdjęciach



Źródło: opracowanie własne

W tabelach zostały przedstawione wyniki testów na zdjęciach z produktami prawidłowymi i uszkodzonymi dla modelu uczonego na 500 zdjęciach. Wartości od 0 do 0.5 zostały przypisane do klasy uszkodzone, a wartości powyżej 0.5 do klasy prawidłowe. Poprawność wyników została oceniona i odpowiednio zapisana.

Analiza wyników testów dla modelu uczonego na 500 zdjęciach pokazuje, że model osiąga znakomitą skuteczność w ocenie zdjęć zarówno z produktami prawidłowymi, jak i uszkodzonymi. Ogólna skuteczność jest znacznie lepsza niż w przypadku modeli uczonych na 25, 50, 100 i 250 zdjęciach.

Dla zdjęć z produktami prawidłowymi (Tabela 7.9), model poprawnie ocenił wszystkie 10 przypadków, co daje 100% skuteczności. Nie wystąpiły żadne błędne oceny.

W przypadku zdjęć z produktami uszkodzonymi (Tabela 7.10), model również osiągnął 100% skuteczność, poprawnie oceniając wszystkie 10 przypadków. Nie wystąpiły żadne błędne oceny.

Ogólna skuteczność modelu uczonego na 500 zdjęciach wynosi 100% (20 poprawnych ocen na 20 prób), co jest znacznie wyższym wynikiem w porównaniu z modelami uczonymi na 25, 50, 100 i 250 zdjęciach.

Wyniki te wskazują, że zwiększenie liczby zdjęć uczących oraz dalsze strojenie hiperparametrów modelu przyczyniło się do znaczającej poprawy ogólnej skuteczności.

Tabela 7.9: Wyniki testów na zdjęciach z produktami prawidłowymi dla modelu uczonego na 500 zdjęciach

Test	Wynik - Prawidłowe	Ocena	Poprawny
1	0.99994624	Prawidłowy	Tak
2	0.86573	Prawidłowy	Tak
3	0.99992496	Prawidłowy	Tak
4	0.99304414	Prawidłowy	Tak
5	0.9994097	Prawidłowy	Tak
6	0.74320	Prawidłowy	Tak
7	0.99879295	Prawidłowy	Tak
8	0.98510396	Prawidłowy	Tak
9	0.9975974	Prawidłowy	Tak
10	0.9990083	Prawidłowy	Tak

Źródło: opracowanie własne

Tabela 7.10: Wyniki testów na zdjęciach z produktami uszkodzonymi dla modelu uczonego na 500 zdjęciach

Test	Wynik - Uszkodzone	Ocena	Poprawny
1	7.884345e-05	Uszkodzony	Tak
2	4.773199e-07	Uszkodzony	Tak
3	2.2078943e-06	Uszkodzony	Tak
4	6.3885636e-06	Uszkodzony	Tak
5	9.216427e-06	Uszkodzony	Tak
6	2.3838764e-09	Uszkodzony	Tak
7	1.6288888e-07	Uszkodzony	Tak
8	0.00364679	Uszkodzony	Tak
9	9.597346e-06	Uszkodzony	Tak
10	0.00247978	Uszkodzony	Tak

Źródło: opracowanie własne

## 7.6 Podsumowanie wyników

Wyniki badań wskazują, że jakość modelu ulega poprawie wraz ze wzrostem liczby zdjęć wykorzystanych do treningu (Tabela 7.11). Model trenowany na 25 zdjęciach osiąga poprawność na poziomie 75%, natomiast model trenowany na 50 zdjęciach uzyskuje nieco niższą poprawność wynoszącą 70%. W przypadku modelu trenowanego na 100 zdjęciach poprawność wynosi 75%. Liczba zdjęć wykorzystana podczas tych badań była niska, dlatego też można zauważać, że oceny w tych modelach były niestabilne. W przypadku modelu trenowanego na 250 zdjęć uzyskano poprawność wynoszącą 75%.

wanego na 25 zdjęciach, model miał tendencje do lepszego rozpoznawania prawidłowych elementów. Natomiast w przypadku modelu trenowanego na 50 zdjęciach model lepiej rozpoznawał uszkodzone elementy.

Znacząca poprawa wydajności modelu następuje przy zastosowaniu 250 zdjęć do treningu, gdzie poprawność wynosi 90%. Ostatecznie, model trenowany na 500 zdjęciach osiąga 100% poprawności, co świadczy o dobrej zdolności do klasyfikacji zdjęć.

Wnioskiem z przeprowadzonych testów jest zatem to, że większa liczba zdjęć użytych do treningu skutkuje lepszymi wynikami modelu. Warto zwrócić uwagę na korzyści płynące z zastosowania większego zbioru danych, szczególnie gdy dąży się do uzyskania wysokiej poprawności klasyfikacji.

Tabela 7.11: Podsumowanie wyników testów

<b>Model</b>	<b>Liczba zdjęć</b>	<b>Prawidłowe</b>	<b>Uszkodzone</b>	<b>Poprawność [%]</b>
25 zdjęć	20	9	6	75
50 zdjęć	20	6	8	70
100 zdjęć	20	7	8	75
250 zdjęć	20	8	10	90
500 zdjęć	20	10	10	100

Źródło: opracowanie własne

## Rozdział 8. Podsumowanie

### 8.1 Wnioski

W wyniku przeprowadzonych badań można wysnuć następujące wnioski:

1. Liczba zdjęć użytych do wyuczenia modelu ma znaczący wpływ na jego dokładność w rozpoznawaniu wad produkcyjnych. Wraz ze wzrostem liczby zdjęć użytych do nauki, dokładność modelu zwykle się poprawia.
2. Model wyuczony na 500 zdjęciach osiągnął najwyższą dokładność w testach, co sugeruje, że większa liczba danych uczących przyczynia się do lepszej generalizacji modelu.
3. W przypadku małych zbiorów danych, model może być narażony na przetrenowanie, co skutkuje słabszymi wynikami na danych testowych.
4. Potrzeba dalszych badań nad optymalizacją architektury modelu oraz eksploracją innych technik przetwarzania wstępnego danych, takich jak augmentacja danych, aby poprawić wydajność modelu.
5. Zastosowanie różnych technik uczenia transferowego może przyczynić się do zwiększenia skuteczności modelu, szczególnie w przypadku mniejszych zbiorów danych.

W oparciu o przeprowadzone badania, można zidentyfikować następujące kierunki dalszego rozwoju projektu:

- Eksploracja innych architektur sieci głębokiego uczenia, takich jak sieci ResNet, DenseNet lub Inception, które mogą osiągać lepsze wyniki w rozpoznawaniu wad produkcyjnych.

- Zastosowanie technik augmentacji danych, aby zwiększyć liczbę dostępnych danych uczących oraz poprawić generalizację modelu.
- Implementacja mechanizmów regularyzacji, takich jak dropout, aby zmniejszyć ryzyko przetrenowania modelu, zwłaszcza w przypadku małych zbiorów danych.
- Analiza wpływu różnych parametrów uczenia, takich jak rozmiar wsadu, współczynnik uczenia czy optymalizator, na dokładność modelu.
- Badanie zastosowania uczenia transferowego, wykorzystując wytrenowane na dużych zbiorach danych modele do ekstrakcji cech, co może skrócić czas uczenia oraz poprawić skuteczność modelu na mniejszych zbiorach danych.
- Eksploracja zastosowań modelu w innych dziedzinach, takich jak analiza jakości innych produktów, gdzie istnieje potrzeba wykrywania wad na podstawie zdjęć.

W kontekście praktycznym, wyniki badań oraz analizy mogą być wykorzystane do planowania i implementacji systemów kontroli jakości w różnych gałęziach przemysłu. Poprawa wydajności modelu w wykrywaniu wad produkcyjnych może przyczynić się do redukcji kosztów związanych z wadliwymi produktami oraz poprawy ogólnej jakości oferowanych produktów. W dłuższej perspektywie, opracowanie efektywnego i dokładnego modelu rozpoznawania wad może także pomóc w automatyzacji procesów kontroli jakości, co pozwoli na optymalizację zasobów i zwiększenie efektywności produkcji.

## 8.2 Zrealizowane cele

W ramach niniejszej pracy pt. "Rozpoznawanie wad produkcyjnych z wykorzystaniem uczenia głębokiego" osiągnięto szereg celów, które przyczyniły się do zrozumienia i wykorzystania technik uczenia głębokiego w celu wykrywania wad produkcyjnych. Poniżej przedstawiamy szczegółowy opis zrealizowanych celów:

- **Przegląd literatury :** Przeprowadzono obszerny przegląd literatury, który obejmował analizę podstawowych koncepcji uczenia głębokiego, sieci neuronowych, konwolucyjnych sieci neuronowych (CNN) oraz przegląd zastosowań tych technik w różnych dziedzinach przemysłu. Analiza literatury pozwoliła na zrozumienie podstawowych zagadnień związanych z uczeniem głębokim oraz na identyfikację kluczowych technik i metod, które mogą być wykorzystane w celu rozpoznawania wad produkcyjnych.

- **Przygotowanie danych uczących** Zebrano i przygotowano zbiór danych uczących, który obejmował zdjęcia przedstawiające produkty z wadami oraz produkty prawidłowe. Przeprowadzono wstępne przetwarzanie danych, takie jak skalowanie, kadrowanie i normalizację, aby ułatwić proces uczenia modelu. Przygotowanie odpowiedniego zbioru danych uczących stanowi podstawę dla późniejszego etapu uczenia modelu oraz oceny jego skuteczności.
- **Budowa i uczenie modeli** Zbudowano modele konwolucyjnych sieci neuronowych (CNN) z różnymi parametrami, takimi jak liczba warstw, rozmiar filtrów, funkcje aktywacji czy optymalizatory. Przeprowadzono uczenie modeli na różnych zbiorach danych uczących, obejmujących 25, 50, 100 oraz 500 zdjęć, co pozwoliło na analizę wpływu liczby danych uczących na skuteczność modelu.
- **Ocena i analiza wyników** Przeprowadzono testy modeli na danych testowych, które nie były wykorzystywane podczas uczenia. Analizowano skuteczność modeli na podstawie ich zdolności do poprawnego rozpoznawania produktów z wadami i produktów prawidłowych. Wyniki testów posłużyły do oceny ogólnej skuteczności modeli oraz do identyfikacji obszarów wymagających dalszych usprawnień.
- **Wnioski i kierunki dalszego rozwoju** Na podstawie przeprowadzonych badań sformułowano wnioski dotyczące zastosowania uczenia głębokiego w rozpoznawaniu wad produkcyjnych oraz zidentyfikowano kierunki dalszego rozwoju projektu. Wskazano między innymi, że liczba zdjęć użytych do wyuczenia modelu ma znaczący wpływ na jego dokładność, a model wyuczony na większej liczbie zdjęć osiąga lepsze wyniki. Zauważono również, że dalsze badania nad optymalizacją architektury modelu oraz eksploracją innych technik przetwarzania wstępnego danych mogą przyczynić się do poprawy wydajności modelu. W ramach dalszego rozwoju projektu zidentyfikowano następujące kierunki:
  - Eksploracja innych architektur sieci głębokiego uczenia, takich jak sieci ResNet, DenseNet lub Inception, które mogą osiągać lepsze wyniki w rozpoznawaniu wad produkcyjnych.
  - Zastosowanie technik augmentacji danych, aby zwiększyć liczbę dostępnych danych uczących oraz poprawić generalizację modelu.

- Implementacja mechanizmów regularyzacji, takich jak dropout, aby zmniejszyć ryzyko przetrenowania modelu, zwłaszcza w przypadku małych zbiorów danych.
- Analiza wpływu różnych parametrów uczenia, takich jak rozmiar wsadu, współczynnik uczenia czy optymalizator, na dokładność modelu.

## Bibliografia

- [1] Yoshua Bengio, Ian Goodfellow, Aaron Courville, Deep Learning Systemy uczące się, Wydawnictwo Naukowe PWN, Warszawa, 2018.
- [2] Mariusz Flasiński, Wstęp do sztucznej inteligencji, Wydawnictwo Naukowe PWN, Warszawa, 2018.
- [3] Joel Grus, Data science od podstaw. Analiza danych w Pythonie, Helion, Gliwice, 2022.
- [4] Ryszard Knosala, Inżynieria produkcji, PWE Polskie Wydawnictwo Ekonomiczne, Warszawa 2017.
- [5] Robert A. Kosiński, Sztuczne sieci neuronowe, Wydawnictwo Naukowe PWN, Warszawa, 2017.
- [6] Yuxi Hayden Liu, Python Uczenie maszynowe w przykładach TensorFlow 2 PyTorch i scikitlearn, Helion, Gliwice, 2022.
- [7] Mark Lutz, Python. Wprowadzenie, Helion, Gliwice, 2022.
- [8] Sebastian Raschka, Vahid Mirjalili, Python Machine learning i deep learning, Helion, Gliwice, 2021.
- [9] Kazimierz Szatkowski, Nowoczesne zarządzanie produkcją, Wydawnictwo Naukowe PWN, Warszawa 2023.
- [10] Jerzy Surma, Hakowanie sztucznej inteligencji, Wydawnictwo Naukowe PWN, Warszawa, 2022.

## **Spis rysunków**

6.1	Prawidłowe - zdjęcie 1 . . . . .	39
6.2	Prawidłowe - zdjęcie 2 . . . . .	39
6.3	Prawidłowe - zdjęcie 3 . . . . .	39
6.4	Prawidłowe - zdjęcie 4 . . . . .	40
6.5	Prawidłowe - zdjęcie 5 . . . . .	40
6.6	Prawidłowe - zdjęcie 6 . . . . .	40
6.7	Prawidłowe - zdjęcie 7 . . . . .	41
6.8	Prawidłowe - zdjęcie 8 . . . . .	41
6.9	Prawidłowe - zdjęcie 9 . . . . .	41
6.10	Prawidłowe - zdjęcie 10 . . . . .	42
6.11	Uszkodzone - zdjęcie 1 . . . . .	42
6.12	Uszkodzone - zdjęcie 2 . . . . .	42
6.13	Uszkodzone - zdjęcie 3 . . . . .	43
6.14	Uszkodzone - zdjęcie 4 . . . . .	43
6.15	Uszkodzone - zdjęcie 5 . . . . .	43
6.16	Uszkodzone - zdjęcie 6 . . . . .	44
6.17	Uszkodzone - zdjęcie 7 . . . . .	44
6.18	Uszkodzone - zdjęcie 8 . . . . .	44
6.19	Uszkodzone - zdjęcie 9 . . . . .	45
6.20	Uszkodzone - zdjęcie 10 . . . . .	45
7.1	Wykres dokładności dla modelu uczonego na 25 zdjęciach . . . . .	47
7.2	Wykres straty dla modelu uczonego na 25 zdjęciach . . . . .	48
7.3	Wykres dokładności dla modelu uczonego na 50 zdjęciach . . . . .	50

7.4	Wykres straty dla modelu uczonego na 50 zdjęciach . . . . .	51
7.5	Wykres dokładności dla modelu uczonego na 100 zdjęciach . . . . .	53
7.6	Wykres straty dla modelu uczonego na 100 zdjęciach . . . . .	54
7.7	Wykres dokładności dla modelu uczonego na 250 zdjęciach . . . . .	56
7.8	Wykres straty dla modelu uczonego na 250 zdjęciach . . . . .	57
7.9	Wykres dokładności dla modelu uczonego na 500 zdjęciach . . . . .	59
7.10	Wykres straty dla modelu uczonego na 500 zdjęciach . . . . .	60

## **Spis tabel**

6.1	Specyfikacja techniczna maszyny użytej do badań . . . . .	36
7.1	Wyniki testów na zdjęciach z produktami prawidłowymi dla modelu uczonego na 25 zdjęciach . . . . .	49
7.2	Wyniki testów na zdjęciach z produktami uszkodzonymi dla modelu uczonego na 25 zdjęciach . . . . .	49
7.3	Wyniki testów na zdjęciach z produktami prawidłowymi dla modelu uczonego na 50 zdjęciach . . . . .	52
7.4	Wyniki testów na zdjęciach z produktami uszkodzonymi dla modelu uczonego na 50 zdjęciach . . . . .	52
7.5	Wyniki testów na zdjęciach z produktami prawidłowymi dla modelu uczonego na 100 zdjęciach . . . . .	55
7.6	Wyniki testów na zdjęciach z produktami uszkodzonymi dla modelu uczonego na 100 zdjęciach . . . . .	55
7.7	Wyniki testów na zdjęciach z produktami prawidłowymi na 250 zdjęciach .	58
7.8	Wyniki testów na zdjęciach z produktami uszkodzonymi na 250 zdjęciach .	58
7.9	Wyniki testów na zdjęciach z produktami prawidłowymi dla modelu uczonego na 500 zdjęciach . . . . .	61
7.10	Wyniki testów na zdjęciach z produktami uszkodzonymi dla modelu uczonego na 500 zdjęciach . . . . .	61
7.11	Podsumowanie wyników testów . . . . .	62

## **OŚWIADCZENIE**

Oświadczam, że:

1. pracę niniejszą przygotowałem(am) samodzielnie; wszystkie dane, istotne myśli i sformułowania pochodzące z literatury (przytoczone dosłownie lub niedosłownie) są opatrzone odpowiednimi odsyłaczami; praca ta nie była w całości ani w części przez nikogo przedkładana do żadnej oceny i nie była publikowana;
2. wyrażam zgodę / nie wyrażam zgody na udostępnianie mojej pracy dyplomowej.

Data .....

.....

imię i nazwisko

Stwierdzam autentyczność podpisu

.....

(podpis pracownika i pieczętka Wydziału)

\* niepotrzebne skreślić