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

1 Wstęp **4**

1.1 Cel i motywacja pracy 4

1.2 Zawartość pracy 4

2 Wybrane podstawy teoretyczne pracy **5**

2.1 Uczenie głębokie 5

2.1.1 Historia uczenia głębokiego 6

2.1.2 Zastosowania uczenia głębokiego 6

2.1.3 Różnice między uczeniem głębokim a innymi technikami uczenia maszynowego 7

2.1.4 Dalsze różnice między uczeniem głębokim a innymi technikami uczenia maszynowego 8

2.2 Sieci konwolucyjne 9

2.2.1 Jak opracowywane są sieci CNN? 9

2.3 Wady produkcyjne i ich rodzaje 12

2.4 Istniejące rozwiązania 14

2.4.1 Kontrola wizualna 14

2.4.2 Systemy wizyjne 15

2.4.3 Techniki uczenia maszynowego 15

2.4.4 Uczenie głębokie i sieci konwolucyjne 15

2.4.5 Internet Rzeczy (IoT) i analiza danych 15

2.4.6 Przegląd rozwiązań 16

2.5 Wybrane języki programowania 16

3 Projekt systemu **19**

3.1 Wymagania 19

3.1.1 Wymagania funkcjonalne 19

3.1.2 Wymagania niefunkcjonalne 20

3.2 Projekt rozwiązania 20

3.2.1 Wybór sieci konwolucyjnej 21

3.2.2 Korzyści z zastosowania uczenia głębokiego 21
3.2.3 Ogólny zarys rozwiązania 21
3.2.4 Import bibliotek 22
3.2.5 Wczytywanie danych 23
3.2.6 Ładowanie danych 24
3.2.7 Wstępne przetwarzanie danych 24
3.2.8 Podział danych na zestawy treningowe, walidacyjne i testowe 25

4 Implementacja **27**
4.1 Budowa modelu sieci neuronowej 27
4.2 Trenowanie modelu 28
4.3 Testowanie modelu na przykładach 30
4.3.1 Wczytywanie przykładowego obrazu 30
4.3.2 Przeskalowanie obrazu 31
4.3.3 Predykcja klasy obrazu 31
4.3.4 Interpretacja wyników 31
4.4 Zapisywanie i wczytywanie modelu 32
4.4.1 Zapisywanie modelu 32
4.4.2 Wczytywanie modelu 32
4.5 Ewaluacja modelu 33
4.5.1 Przygotowanie zbioru testowego 33
4.5.2 Ocena klasyfikacji 33
4.5.3 Wizualizacja wyników 34
4.6 Analiza wyników 34

5 Badanie, porównanie efektywności systemu **36**
5.1 Testy wydajności 36
5.2 Testy skuteczności 36
5.3 Analiza skuteczności 36

6 Prezentacja działania programu **37**

7 Podsumowanie **38**
7.1 Wnioski 38
7.2 Zrealizowane cele 38
7.3 Perspektywy 38

Rozdział 1. Wstęp

1.1 Cel i motywacja pracy

1.2 Zawartość pracy

Rozdział 2. Wybrane podstawy teoretyczne pracy

2.1 Uczenie głębokie

Uczenie głębokie to gałąź uczenia maszynowego, która skupia się na zastosowaniu sztucznych sieci neuronowych z wieloma warstwami ukrytymi. W przeciwieństwie do tradycyjnych metod uczenia maszynowego, takich jak regresja liniowa czy drzewa decyzyjne, uczenie głębokie automatycznie odkrywa reprezentacje danych na różnych poziomach abstrakcji. Jest to kluczowe dla rozwiązywania złożonych problemów, takich jak rozpoznawanie obrazów, przetwarzanie języka naturalnego, rozpoznawanie mowy czy analiza danych biologicznych. Uczenie głębokie to specjalna odmiana uczenia maszynowego, w ramach której sztuczne sieci neuronowe, czyli algorytmy stworzone tak, aby naśladować sposób działania ludzkiego mózgu, uczą się na podstawie obszernych zbiorów danych. Uczenie głębokie opiera się na wielowarstwowych sieciach neuronowych, które są algorytmami wzorowanymi na sposób funkcjonowania ludzkich mózgów. Szkolenie z dużą ilością danych kształtuje neurony w sieci neuronowej, co prowadzi do stworzenia modelu uczenia głębokiego zdolnego do przetwarzania nowych danych po przeszkoleniu. Modele uczenia głębokiego korzystają z informacji z różnych źródeł danych i analizują je w czasie rzeczywistym, bez potrzeby ingerencji człowieka. W procesie uczenia głębokiego, procesory graficzne (GPU) są optymalizowane do pracy z modelami szkoleniowymi, ponieważ mogą równocześnie wykonywać wiele obliczeń. Uczenie głębokie jest siłą napędową wielu technologii sztucznej inteligencji (AI), które mogą zautomatyzować i usprawnić zadania analityczne. Większość osób codziennie styka się z uczeniem głębokim, przeglądając internet czy korzystając z telefonów komórkowych. Wśród licznych zastosowań, uczenie głębokie jest wykorzystywane do generowania napisów dla filmów na YouTube, rozpoznawania mowy w telefonach i inteligentnych głośnikach, identyfikowania twarzy na zdjęciach czy tworzenia autonomicznych pojazdów. W miarę jak analitycy

i badacze danych podejmują coraz bardziej zaawansowane projekty związane z uczeniem głębokim, wykorzystując architektury głębokich sieci neuronowych, ten rodzaj sztucznej inteligencji stanie się jeszcze bardziej powszechny w naszym codziennym życiu.

2.1.1 Historia uczenia głębokiego

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 wstecznej propagacji błędów (backpropagation), który umożliwił efektywne uczenie się sieci neuronowych.

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 (deep belief networks) 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.

2.1.2 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 (ang. attention mechanisms) 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 syntezowania mowy. Dzięki temu możliwe jest tworzenie inteligentnych asystentów głosowych, jak Amazon Alexa czy Google Assistant.

- **Uczenie ze wzmocnieniem:** Uczenie głębokie jest łączone z algorytmami uczenia ze wzmocnieniem, aby sterować robotami, pojazdami autonomicznymi, strategiami handlowymi na giełdzie czy systemami rekomendacji. Przykładem może być AlphaGo, system stworzony przez DeepMind, który pokonał mistrza świata w grze Go.
- **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.

2.1.3 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 (feature engineering) używane do uczenia modelu, uczenie głębokie automatycznie wykorzystuje hierarchiczne reprezentacje danych na różnych poziomach abstrakcji. 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. 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 (GPU). 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 (transfer learning). 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.

Różnice te sprawiają, że uczenie głębokie jest coraz częściej wykorzystywane w różnych dziedzinach nauki i przemysłu, zastępując lub uzupełniając inne techniki uczenia maszynowego.

2.1.4 Dalsze różnice między uczeniem głębokim a innymi technikami uczenia maszynowego

Oprócz wcześniej wymienionych różnic, uczenie głębokie różni się od innych technik uczenia maszynowego również pod względem:

- **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 dostępnych 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.

- **Współpraca z innymi metodami uczenia maszynowego:** Uczenie głębokie może być używane w połączeniu z innymi technikami uczenia maszynowego, aby stworzyć bardziej efektywne i wydajne systemy. Na przykład, sieci neuronowe mogą być stosowane

razem z drzewami decyzyjnymi czy algorytmami klasteryzacji w celu uzyskania lepszych wyników w przewidywaniu czy klasyfikacji.

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

2.2 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. 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.

2.2.1 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: uczenie, optymalizację oraz wnioskowanie.

Szkolenia

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 poprawnych predykcji na nowych, wcześniej niewidzianych danych. Proces szkolenia CNN składa się z kilku kroków:

- **Inicjalizacja wag:** Na początek, wagi sieci neuronowej są inicjalizowane losowo lub za pomocą specjalnych technik inicjalizacji.
- **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.
- **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.
- **Propagacja wsteczna (Backpropagation):** 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.
- **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.
- **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.

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.

Optymalizacja

Optymalizacja w konwolucyjnych sieciach neuronowych (CNN) polega na dostosowywaniu parametrów sieci w celu zminimalizowania funkcji kosztu i poprawy wydajności sieci. 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 kosztu 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

Wnioskowanie (ang. inference) w konwolucyjnych sieciach neuronowych (CNN) odnosi się do procesu, w którym wcześniej wytrenowana sieć jest stosowana do analizy nowych danych wejściowych i generowania predykcji. 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 nauczonej hierarchii cech oraz wag, które zostały optymalizowane podczas procesu treningu. Dzięki tym wagom sieć jest w stanie rozpoznać i klasyfikować nowe dane wejściowe. 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.

2.3 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. 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.
- **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.
- **Wady materiałowe:** Dotyczą one wad w strukturze materiału, takich jak pęcherze powietrza, porowatość czy zanieczyszczenia. Mogą być spowodowane przez niewłaściwy dobór surowców, problemy z mieszaniem materiałów czy nieodpowiednie warunki procesowe, takie jak szybkość chłodzenia czy czas utwardzania.
- **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.

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. 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.

- **Kontrola jakości surowców:** Przeprowadzanie kontroli jakości dostawców surowców i systematyczne sprawdzanie jakości materiałów przed rozpoczęciem produkcji może pomóc w wykryciu i eliminacji wad materiałowych.

- **Optymalizacja procesów produkcyjnych:** Analiza procesów produkcyjnych w celu identyfikacji i eliminacji źródeł wad może prowadzić do wydajniejszej i bardziej niezawodnej produkcji. Obejmuje to zarówno monitorowanie maszyn i narzędzi, jak i przeglądanie procedur produkcyjnych, aby zapewnić ich zgodność z najlepszymi praktykami.

- **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.

- **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.

- **Analiza przyczyn wad i działania korygujące:** Gdy wady produkcyjne zostaną zidentyfikowane, ważne jest przeprowadzenie dogłębnej 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.

- **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.

- **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.

- **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.

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.

2.4 Istniejące rozwiązania

W przemyśle wykorzystuje się wiele technik do wykrywania wad produkcyjnych. Poniżej przedstawiamy obszerny, szczegółowy i techniczny opis istniejących rozwiązań:

2.4.1 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.

2.4.2 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.

2.4.3 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.

2.4.4 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.

2.4.5 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.

2.4.6 Przegląd rozwiązań

Wady produkcyjne można wykrywać na różne sposoby, a każde z istniejących rozwiązań ma swoje wady i zalety. Kontrola wizualna jest stosunkowo tania, ale czasochłonna i podatna na błędy. Systemy wizyjne i techniki uczenia maszynowego są bardziej zaawansowane, ale często wymagają ręcznego projektowania cech i mogą być trudne do skalowania.

Uczenie głębokie, a szczególnie sieci konwolucyjne, oferuje nowe możliwości w zakresie wykrywania wad produkcyjnych, zapewniając wysoką precyzję i łatwość skalowania. Ponadto, technologie IoT i analiza danych mogą przyczynić się do lepszego monitorowania i kontroli procesów produkcyjnych, co może prowadzić do szybszego wykrywania i eliminacji wad.

W związku z tym, rozwój i wdrożenie zaawansowanych technik uczenia głębokiego, sieci konwolucyjnych oraz technologii IoT w przemyśle może prowadzić do znacznego zmniejszenia liczby wad produkcyjnych oraz poprawy jakości produktów. W miarę jak te technologie będą się rozwijać, można spodziewać się, że będą one jeszcze bardziej efektywne i powszechnie stosowane w celu monitorowania jakości oraz wykrywania wad produkcyjnych.

2.5 Wybrane języki programowania

C++

C++ to język programowania o ogólnym przeznaczeniu, który pozwala na równoczesne korzystanie z różnych paradygmatów programowania, takich jak programowanie proceduralne, obiektowe i generyczne. C++ jest znany ze swojej wydajności i szerokiego zastosowania, zwłaszcza w projektowaniu systemów operacyjnych, gier i aplikacji o wysokiej wydajności. W kontekście uczenia maszynowego i uczenia głębokiego, C++ jest często stosowany do optymalizacji wydajności i budowy szybkich bibliotek, takich jak TensorFlow, które są następnie używane przez programistów Pythona.

R

R to język programowania oraz środowisko oprogramowania do statystyki i grafiki, które jest szeroko używane przez naukowców danych i statystyków. R posiada bogaty ekosystem pakietów do analizy danych, modelowania statystycznego, wizualizacji i uczenia maszynowego, takich jak dplyr, ggplot2 czy randomForest. Chociaż R nie jest tak popularny jak Python w zakresie uczenia głębokiego, istnieje kilka bibliotek, takich jak Keras-R czy MXNet-R, które pozwalają na korzystanie z sieci neuronowych w języku R.

Java

Java to język programowania o ogólnym przeznaczeniu, oparty na koncepcji obiektowości. Java jest szeroko stosowana w różnych dziedzinach, takich jak rozwój aplikacji webowych, mobilnych, desktopowych czy systemów wbudowanych. W uczeniu maszynowym i uczeniu głębokim, Java oferuje różne biblioteki i ramy pracy, takie jak Deeplearning4j, Weka czy Apache Mahout, które umożliwiają stosowanie zaawansowanych technik analizy danych i modelowania. Java jest szczególnie przydatna dla organizacji, które już używają technologii opartych na Javie, dzięki czemu mogą łatwo integrować rozwiązania związane z uczeniem maszynowym z istniejącą infrastrukturą.

Julia

Julia to nowoczesny język programowania o wysokiej wydajności, który został zaprojektowany z myślą o obliczeniach naukowych, analizie danych i uczeniu maszynowym. Julia łączy szybkość i wydajność języków, takich jak C++ czy Fortran, z prostotą i elastycznością języków, takich jak Python czy R. Julia oferuje rosnący ekosystem bibliotek do uczenia maszynowego i uczenia głębokiego, takich jak Flux.jl czy MLJ.jl. Język ten zdobywa popularność wśród naukowców i inżynierów danych ze względu na swoją zdolność do przyspieszania obliczeń i możliwość współpracy z innymi językami programowania.

Python: szczegółowy opis techniczny

Python to dynamicznie typowany, interpretowany język programowania wysokiego poziomu, który został stworzony przez Guido van Rossuma 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.

- **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.
- **Biblioteki:** Python posiada bogaty ekosystem bibliotek wspierających uczenie maszynowe i uczenie głębokie, co pozwala na tworzenie zaawansowanych modeli z mniejszym nakładem pracy. Przykłady bibliotek to:
  - TensorFlow: opracowane przez Google, pozwala na tworzenie i wdrożenie modeli uczenia maszynowego i uczenia głębokiego.
  - PyTorch: opracowany przez Facebook AI Research, jest popularnym narzędziem do uczenia maszynowego i głębokiego, szczególnie w badaniach naukowych.

  - scikit-learn: to biblioteka do uczenia maszynowego, która zawiera wiele prostych i wydajnych narzędzi do analizy danych i modelowania.
  - Keras: to wysokopoziomowy interfejs API do tworzenia sieci neuronowych, który może działać na TensorFlow, Theano lub CNTK.
  - NumPy: to biblioteka do obsługi wielowymiarowych tablic i macierzy, a także szerokiej gamy funkcji matematycznych.
  - Pandas: to biblioteka do manipulacji i analizy danych, która oferuje struktury danych, takie jak DataFrame i Series, umożliwiające łatwe zarządzanie danymi.

- **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.

- **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.

- **Jupyter Notebook**: Jupyter Notebook to popularne narzędzie do tworzenia interaktywnych notatników, które umożliwiają tworzenie i udostępnianie dokumentów zawierających zarówno kod, jak i bogate treści medialne, takie jak wykresy, obrazy czy filmy. Jupyter Notebook jest szeroko wykorzystywany w uczeniu maszynowym, analizie danych i wizualizacji, co przyczynia się do jeszcze większej popularności Pythona w tych dziedzinach.

- **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.

Rozdział 3. Projekt systemu

3.1 Wymagania

W tej sekcji opisane zostaną wymagania funkcjonalne i niefunkcjonalne systemu klasyfikacji obrazów. Wymagania funkcjonalne są bezpośrednio związane z funkcjami, które system powinien realizować, natomiast wymagania niefunkcjonalne dotyczą cech jakościowych systemu.

3.1.1 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.

3.1.2 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.

Podsumowując, wymagania funkcjonalne i niefunkcjonalne mają na celu zapewnienie, że opracowany system klasyfikacji obrazów jest skuteczny, wydajny i skalowalny, oraz że może być używany w różnych kontekstach przemysłowych związanych z kontrolem jakości.

3.2 Projekt rozwiązania

W tej sekcji przedstawimy projekt proponowanego rozwiązania do klasyfikacji obrazów przedstawiających uszkodzone i prawidłowe elementy. Wybieramy sieć konwolucyjną (CNN) jako kluczową technologię do rozwiązania tego problemu ze względu na jej zdolność do skutecznego uczenia się hierarchicznych reprezentacji danych przestrzennych.

3.2.1 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 naszego zadania klasyfikacji.

3.2.2 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.

3.2.3 Ogólny zarys rozwiązania

Proponowane 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 podsekcjach przedstawimy 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 będziemy mogli ocenić jakość opracowanego rozwiązania oraz jego potencjalne ograniczenia. W miarę potrzeby będziemy również analizować możliwości dalszego rozwoju i optymalizacji modelu.

3.2.4 Import bibliotek

W tej części omówimy wykorzystane biblioteki oraz ich zastosowanie w projekcie. W naszym rozwiązaniu korzystamy z następujących bibliotek:

- **TensorFlow:** Główna biblioteka do uczenia głębokiego, która umożliwia definiowanie, trenowanie i ewaluację modeli sieci neuronowych. W projekcie używamy 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 używamy 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 używamy 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 używamy NumPy do manipulacji danymi obrazowymi, przetwarzania macierzy oraz realizacji obliczeń matematycznych.

Do pracy z tymi bibliotekami importujemy je, 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 nam 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. Warto zauważyć, że instalacja TensorFlow GPU umożliwia korzystanie z przyspieszenia obliczeń na 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')
```

3.2.5 Wczytywanie danych

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

Podczas wczytywania danych, kontrolujemy jakość zdjęć, sprawdzając, czy można je załadować do biblioteki OpenCV oraz czy ich rozszerzenie jest zgodne z oczekiwanymi (np. 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('Zdjecie posiada nieobslugiwane rozszerzenie {}'.format(
    image_path))
os.remove(image_path)
except Exception as e:
```

```
print('Wystapil problem ze zdjeciem {}'.format(image_path))
```

3.2.6 Ładowanie danych

W kolejnym kroku ładujemy 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])
```

3.2.7 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 porównywalne przez sieć neuronową.

Warto zauważyć, że przeskalowanie wartości pikseli jest jednym z wielu możliwych kroków wstępnego przetwarzania danych. W zależności od specyfiki problemu i danych wejściowych, inne kroki wstępnego przetwarzania mogą obejmować augmentację danych (na przykład obroty, przesunięcia, czy odbicia obrazów), konwersję obrazów do skali szarości, normalizację danych, czy też zastosowanie filtrów obrazu. Dobór odpowiednich metod wstępnego przetwarzania danych jest kluczowy dla uzyskania dobrych wyników uczenia się modelu.

3.2.8 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 określamy ich rozmiary. W tym przypadku, 70

```
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. Dobrze jest eksperymentować z różnymi proporcjami podziału danych, aby znaleźć optymalny podział dla konkretnego problemu.

Rozdział 4. Implementacja

4.1 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.

4.2 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 przeszkolić 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 użyto TensorBoard jako wywołania zwrotnego (ang. callback) 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='dokładność treningowa')
plt.plot(hist.history['val_accuracy'], color='orange',
label='dokładność walidacji')
fig.suptitle('Dokładność', 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.

Podsumowując, trenowanie modelu sieci neuronowej to proces uczenia modelu na danych treningowych oraz oceny jego wydajności na danych walidacyjnych. Użycie TensorBoard pozwala na monitorowanie tego procesu w czasie rzeczywistym, a analiza wykresów straty oraz dokładności pozwala na ocenę jakości uczenia modelu. Dostosowanie liczby epok oraz parametrów modelu może poprawić jego wydajność i ograniczyć przeuczenie.

4.3 Testowanie modelu na przykładach

W tej sekcji omówimy 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.

4.3.1 Wczytywanie przykładowego obrazu

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()
```

4.3.2 Przeskalowanie obrazu

Przed przekazaniem obrazu do modelu, konieczne jest jego przeskalowanie do wymiarów, na których model został wytrenowany. W naszym przypadku, model został przeszkolony 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()
```

4.3.3 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))
```

4.3.4 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 zostal sklasyfikowany jako czesc prawidlowa')
else:
print(f'Wskazany obraz zostal 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 klasyfikować różnorodne obrazy przedstawiające uszkodzone i prawidłowe elementy.

4.4 Zapisywanie i wczytywanie 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.

4.4.1 Zapisywanie modelu

Zapisanie modelu w TensorFlow odbywa się za pomocą metody `save`:

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

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

4.4.2 Wczytywanie modelu

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

```
new_model = load_model(os.path.join('models', '
    imageclassicationversionlive.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 zostal sklasyfikowany jako czesc prawidlowa')
else:
print(f'Wskazany obraz zostal sklasyfikowany jako czesc uszkodzona')
```

Dzięki zapisywaniu i wczytywaniu modelu mamy możliwość przechowywania i wykorzystywania wyników treningu w dowolnym momencie, co pozwala na efektywniejsze wykorzystanie zasobów obliczeniowych oraz sprawne wdrażanie modeli do praktycznych zastosowań.

4.5 Ewaluacja modelu

W niniejszym podrozdziale omówiony zostanie proces oceny jakości klasyfikacji przeprowadzonej przez model. Ewaluacja modelu pozwala na ocenę jego skuteczności oraz możliwość identyfikacji obszarów, które mogą wymagać poprawy. Poniżej opisane zostaną kroki procesu ewaluacji.

4.5.1 Przygotowanie zbioru testowego

Zbiór danych został podzielony na trzy części: dane treningowe (70% całego zbioru), dane walidacyjne (20% całego zbioru) oraz dane testowe (10% całego zbioru). Dane testowe zostały wykorzystane do ostatecznej oceny modelu.

```
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)
```

4.5.2 Ocena klasyfikacji

Aby ocenić jakość klasyfikacji, wykorzystano następujące miary: precyzja, czułość (recall) oraz F1-score. W celu obliczenia tych miar, użyto zbioru testowego oraz funkcji `classification_r` z pakietu `scikitlearn`.

```
from sklearn.metrics import classification_report

y_test = []
y_pred = []

for images, labels in test:
predictions = model.predict(images)
```

```
predictions = np.round(predictions).reshape(-1)
y_test.extend(labels.numpy())
y_pred.extend(predictions)

report = classification_report(y_test, y_pred, target_names=['
    uszkodzony', 'prawidlowy'], output_dict=True)
print(report)
```

4.5.3 Wizualizacja wyników

W celu lepszego zrozumienia wyników ewaluacji, przedstawiono je w postaci wykresu słupkowego. Wykorzystano do tego biblioteki `seaborn` oraz `pandas`.

```
import seaborn as sns
import pandas as pd

report_df = pd.DataFrame(report).transpose()
report_df = report_df[:-3]

plt.figure(figsize=(8, 4))
sns.barplot(x=report_df.index, y='f1-score', data=report_df)
plt.ylim(0, 1)
plt.title('F1-score dla poszczegolnych klas')
plt.xlabel('Klasy')
plt.ylabel('F1-score')
plt.show()
```

Na podstawie powyższego kodu, tworzony jest wykres słupkowy prezentujący wartości F1-score dla poszczególnych klas (uszkodzony i prawidłowy).

4.6 Analiza wyników

Wyniki ewaluacji modelu, przedstawione za pomocą miar precyzji, czułości oraz F1-score, wskazują na skuteczność modelu w klasyfikacji obrazów przedstawiających uszkodzone i prawidłowe elementy. F1-score, będący miarą uwzględniającą zarówno precyzję, jak i czułość, pozwala na ocenę jakości modelu w przypadku niezrównoważonych zbiorów danych.

Warto zwrócić uwagę na wyniki osiągnięte dla poszczególnych klas. W przypadku kla-

syfikacji elementów uszkodzonych i prawidłowych, wartości F1-score wskazują na zadowalającą skuteczność modelu. Jednakże, wartości te mogą się różnić w zależności od danych treningowych i walidacyjnych, co może wpłynąć na ogólną ocenę modelu.

Rozdział 5. Badanie, porównanie efektywności systemu

5.1 Testy wydajności

5.2 Testy skuteczności

5.3 Analiza skuteczności

Rozdział 6. Prezentacja działania programu

Rozdział 7. Podsumowanie

7.1 Wnioski

7.2 Zrealizowane cele

7.3 Perspektywy

Listings

Spis rysunków

Spis tabel

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 pieczątka Wydziału)

* niepotrzebne skreślić