

Politechnika Opolska
Wydział Informatyki

**Algorytm rozpoznawania niedoborów
pierwiastków podczas wegetacji roślin na
podstawie ich zdjęć**

studia stacjonarne II stopnia

specjalność Systemy inteligentne

Promotor:
Dr hab. inż. Mariusz Rząsa

Autor:
inż. Jakub Nieśmiała
nr indeksu: s101374

Opole, czerwiec 2025 r.

ALGORYTM ROZPOZNAWANIA NIEDOBORÓW PIERWIASTKÓW PODCZAS WEGETACJI ROŚLIN NA PODSTAWIE ICH ZDJĘĆ

Streszczenie

Celem pracy było opracowanie i porównanie algorytmów umożliwiających automatyczne rozpoznawanie niedoborów składników pokarmowych oraz objawów chorób na liściach roślin na podstawie ich zdjęć. Problem ten zyskał na znaczeniu w kontekście rosnącego zapotrzebowania na żywność oraz potrzeby optymalizacji produkcji rolnej w warunkach szklarniowych. W pracy wykorzystano publicznie dostępny zbiór danych PlantVillage, który został odpowiednio przetworzony (skalowanie, augmentacja, kodowanie etykiet). Oceniono skuteczność sześciu modeli klasyfikacyjnych: trzech opartych na głębokim uczeniu (Xception, ResNet50, własna konwolucyjna sieć neuronowa) oraz trzech klasycznych (KNN, SVM, Random Forest). Badania wykazały znaczną przewagę modeli opartych na transfer learningu (Xception, ResNet50), w stosunku do klasycznych algorytmów. Model Xception wykazał się najwyższą skutecznością oraz stabilnością. Zaproponowane rozwiązanie może stanowić podstawę do budowy eksperckiego systemu wspomagającego decyzje w rolnictwie precyzyjnym.

Algorithm for recognizing element deficiencies during plant vegetation based on their photos

S u m m a r y

The aim of this study was to develop and compare algorithms for the automatic recognition of nutrient deficiencies and disease symptoms on plant leaves based on their images. This issue has gained importance in the context of increasing food demand and the need to optimize agricultural production in greenhouse conditions. The publicly available PlantVillage dataset was used in the research, having been appropriately preprocessed (scaling, augmentation, label encoding). The effectiveness of six classification models was evaluated: three based on deep learning (Xception, ResNet50, a custom convolutional neural network) and three classical models (KNN, SVM, Random Forest). The study demonstrated a significant advantage of transfer learning-based models (Xception, ResNet50) over classical algorithms. The Xception model showed the highest accuracy and stability. The proposed solution can serve as a foundation for building an expert decision support system for precision agriculture.

Spis treści

1. Cel i zakres pracy	5
2. Wstęp	6
3. Charakterystyka problemu	8
3.1. Specyfika upraw szklarniowych	8
3.2. Znaczenie diagnozy niedoborów i chorób roślin.....	9
3.3. Wyzwania w rozpoznawaniu wizualnym	10
4. Przegląd literatury i istniejących rozwiązań.....	12
4.1. Tradycyjne metody diagnostyki roślin	12
4.2. Zastosowanie sztucznej inteligencji w rolnictwie	13
4.3. Modele klasyfikacji obrazów – przegląd technik	14
5. Opis danych i etap przygotowania.....	17
5.1. Źródła danych i struktura zbioru	17
5.2. Przekształcenia wstępne i augmentacja danych	19
5.3. Podział na zbiory treningowy, walidacyjny i testowy	23
6. Opis i implementacja modeli.....	25
6.1. Transfer learning – model Xception.....	25
6.2. Transfer learning – model ResNet50.....	26
6.3. Konwolucyjne sieci neuronowe (CNN)	28
6.4. Klasyfikator KNN	29
6.5. Maszyna wektorów nośnych (SVM)	30
6.6. Algorytm Random Forest	32
7. Eksperymenty i wyniki	34
7.1. Metryki ewaluacyjne i sposób oceny modeli	34
7.2. Wyniki badań	36
7.3. Zestawienie rezultatów	45
7.4. Analiza błędów	47
8. Wnioski i podsumowanie	50
8.1. Ocena skuteczności badanych modeli	50
8.2. Wnioski praktyczne i możliwe kierunki dalszych badań	51
9. Bibliografia	52

1. Cel i zakres pracy

Wzrost zapotrzebowania na żywność oraz ograniczone zasoby naturalne wymagają wdrażania nowoczesnych technologii w rolnictwie. Jednym z obiecujących kierunków jest wykorzystanie algorytmów analizy obrazu, w szczególności konwolucyjnych sieci neuronowych (CNN), do automatycznej diagnozy niedoborów składników pokarmowych i chorób liści roślin.

Dostępne rozwiązania często cechują się ograniczoną skutecznością ze względu na wąski zakres danych – obejmują zwykle pojedyncze gatunki roślin i nie odzwierciedlają różnorodnych warunków uprawy, a także ich skuteczność nie spełnia oczekiwań. Celem niniejszej pracy jest opracowanie i porównanie wybranych algorytmów uczenia maszynowego i głębokiego pod kątem ich skuteczności w diagnozie zdjęć liści jako zdrowych, wykazujących niedobory lub objawy chorób. Praca stanowi krok w stronę stworzenia eksperckiego systemu wspomagającego decyzje w rolnictwie precyzyjnym.

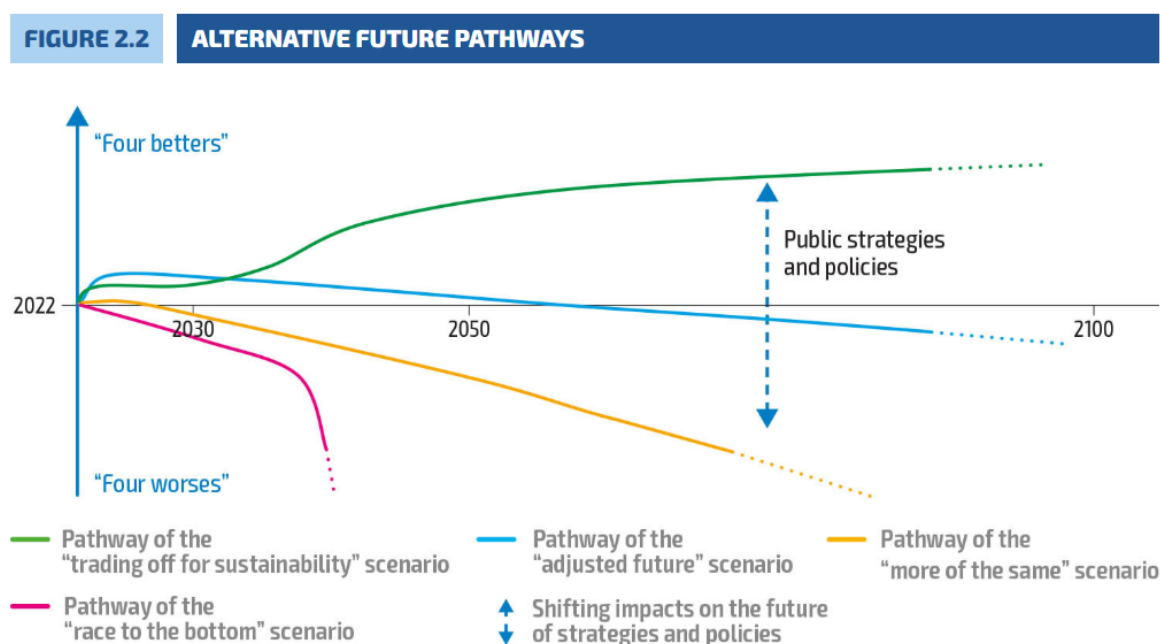
Zadaniem algorytmu jest rozpoznanie na podstawie zdjęcia liścia, czy roślina wykazuje objawy niedoboru składników pokarmowych (np. azotu, potasu, magnezu) lub symptomów chorób biotycznych (np. grzybowych). W odróżnieniu od klasycznych metod diagnostycznych opartych na analizie laboratoryjnej lub doświadczeniu człowieka, zaproponowane w pracy podejście wykorzystuje nowoczesne techniki analizy obrazu oraz sztuczną inteligencję do osiągnięcia automatycznej, powtarzalnej i skalowalnej diagnozy, która w przyszłości mogłaby stanowić podstawę eksperckiego systemu wspierającego decyzje w zakresie zarządzania uprawami w warunkach szklarniowych. Wyniki badania mogą przyczynić się do rozwoju nowoczesnych narzędzi wspomagających rolnictwo precyzyjne, umożliwiając lepsze zarządzanie zasobami i zwiększenie efektywności produkcji.

Zakres pracy obejmuje:

- Analizę dostępnych zbiorów danych obrazów liści roślin wykazujących różne typy niedoborów i chorób,
- Przygotowanie danych do trenowania modeli (czyszczenie, skalowanie, augmentacja),
- Implementację i trenowanie wybranych algorytmów klasyfikujących obrazy (m.in. CNN, KNN, SVM, Transfer Learning – Xception oraz ResNet, a także Random Forest),
- porównanie skuteczności modeli w kontekście kluczowych metryk ewaluacyjnych (dokładność, precyzja, czułość, F1-score) oraz czasu trenowania,
- wybór najlepszego modelu do potencjalnego wykorzystania w przyszłej aplikacji eksperckiej wspomagającej decyzje rolników.

2. Wstęp

Współczesne rolnictwo stoi w obliczu rosnących wyzwań związanych z globalnymi zmianami klimatycznymi, degradacją gleby, ograniczoną dostępnością zasobów naturalnych oraz rosnącym zapotrzebowaniem na żywność. Liczba ludności na świecie nieustannie rośnie, według danych dostępnych w serwisach statystycznych, globalna populacja przekracza już 8 miliardów, a prognozy demograficzne wskazują, że do 2050 roku świat może liczyć od 9,7 do nawet 10 miliardów mieszkańców. Taki gwałtowny wzrost oznacza, że zapotrzebowanie na żywność będzie rosło w zatrważającym tempie. Jednocześnie obserwuje się systemowy spadek powierzchni terenów rolnych, głównie w wyniku urbanizacji, ekspansji przemysłowej oraz przekształceń gruntowych, co jeszcze bardziej komplikuje zapewnienie bezpieczeństwa żywnościowego. Według Organizacji Narodów Zjednoczonych ds. Wyżywienia i Rolnictwa (FAO), aby zaspokoić potrzeby rosnącej populacji, globalna produkcja żywności musi wzrosnąć o około 70–100% do roku 2050^[1]. Na rysunku poniżej (rys. 2.1.) przedstawiono wykres z projekcją możliwych ścieżek rozwoju omawiany w raporcie FAO.



Rys. 2.1. Wykres przedstawiający cztery alternatywne ścieżki przyszłości dla systemów rolno-żywnościowych omawiany w raporcie FAO

Wykres przedstawia cztery alternatywne ścieżki rozwoju systemów rolno-żywnościowych. Pokazuje on różne scenariusze ewolucji, zależnie od podejmowanych działań i polityk publicznych. Pierwszy scenariusz, „*More of the Same*” (Więcej tego samego), zakłada kontynuację dotychczasowych trendów bez znaczących zmian, co może prowadzić do degradacji środowiska oraz pogłębiających się nierówności społecznych. Druga ścieżka, „*Adjusted Future*” (Dostosowana przyszłość), przewiduje umiarkowane korekty w obecnym systemie, z niewielkimi krokami w stronę zrównoważonego rozwoju, ale bez gruntownych reform. Trzeci scenariusz, „*Race to the Bottom*” (Wyścig na dno), przedstawia negatywną wizję przyszłości, w której złe decyzje i brak współpracy prowadzą do pogłębiających się

globalnych kryzysów społecznych, ekonomicznych i środowiskowych. Ostatnia ścieżka, „*Trading Off for Sustainability*” (Kompromisy na rzecz zrównoważonego rozwoju), ukazuje fundamentalne zmiany w podejściu do polityki gospodarczej i społecznej, skoncentrowane na długoterminowej stabilności, ochronie środowiska oraz równych warunkach życia dla wszystkich. Wybór ścieżki rozwoju zależy od decyzji podejmowanych przez społeczeństwo oraz polityków. Scenariusze kontynuacji obecnych trendów oraz niewielkich korekt mogą skutkować dalszymi problemami środowiskowymi i społecznymi, podczas gdy wyścig na dno niesie wizję katastrofalnych skutków globalnych. Kompromisy na rzecz zrównoważonego rozwoju wskazują na możliwość wdrożenia strukturalnych reform, które zapewnią stabilność ekologiczną, gospodarczą oraz społeczną w dłuższej perspektywie. Stawia to przed ludzkością jedno z najpoważniejszych wyzwań XXI wieku - konieczność zrównoważonego zwiększenia produkcji żywności przy ograniczonych zasobach.

To ogromne wyzwanie, jakim jest zapewnienie bezpieczeństwa żywnościowego, nie istnieje w oderwaniu od innych problemów – wręcz przeciwnie, towarzyszą mu liczne czynniki, które dodatkowo utrudniają jego realizację. Globalne zmiany klimatyczne prowadzą do szeregu niekorzystnych zjawisk, takich jak wzrost średnich temperatur, coraz bardziej nieregularne opady czy nasilające się okresy suszy. Takie warunki bezpośrednio wpływają na rolnictwo - ograniczają wydajność upraw, zmieniają długość i rytm sezonów wegetacyjnych oraz zwiększają ryzyko występowania chorób roślin i inwazji szkodników.

Skutkiem coraz bardziej ekstremalnych zjawisk pogodowych jest również degradacja gleby. Dochodzi do utraty jej struktury, obniżenia zawartości materii organicznej oraz pogorszenia naturalnych właściwości retencyjnych. W efekcie zmniejsza się urodzajność gleb i ich zdolność do magazynowania wody oraz składników pokarmowych, co w dłuższej perspektywie zagraża stabilności produkcji rolniczej^[2].

Kolejnym poważnym problemem jest nieprawidłowe nawożenie. Stosowanie nadmiernych lub źle dobranych dawek nawozów, które nie są przyswajane przez rośliny, prowadzi do ich wypłukiwania wraz z wodami opadowymi do rzek, jezior i mórz. To z kolei powoduje eutrofizację i zaburzenia w funkcjonowaniu całych ekosystemów wodnych. Równocześnie niekorzystne zmiany w chemii gleby - takie jak zakwaszenie, zasolenie, alkalizacja czy nagromadzenie toksycznych substancji - negatywnie wpływają na bioróżnorodność i stan środowiska^[3]. Do tego dochodzi rosnące ograniczenie dostępności kluczowych zasobów naturalnych, a w szczególności ziemi, wody i energii. Intensyfikacja upraw, nieodpowiedzialne gospodarowanie zasobami oraz przekształcenia gruntów rolnych wpływają na ich stopniowe wyczerpywanie. Równolegle rolnicy muszą zmagać się z niestabilnością cen surowców - zwłaszcza gazu ziemnego, który stanowi podstawowy składnik nawozów azotowych. Wahanie cen, wynikające z niestabilnej sytuacji geopolitycznej, rosnących kosztów energii i zakłóceń w łańcuchach dostaw, znacząco podnoszą koszty produkcji rolniczej i utrudniają inwestycje w nowoczesne, ekologiczne technologie^[4].

W obliczu rosnącej liczby ludności i zwiększonego zapotrzebowania na żywność, a także ograniczonych zasobów i kurczących się terenów rolnych, wyzwania stojące przed rolnictwem stają się coraz bardziej złożone. Konieczność zwiększenia globalnej produkcji żywności o 70–100% do 2050 roku (zgodnie z prognozami FAO), wymaga wdrożenia innowacyjnych i zrównoważonych rozwiązań. Jednakże, rolnictwo stoi również w obliczu dodatkowych trudności: negatywnego wpływu globalnych zmian klimatycznych, degradacji gleby, szkodliwych praktyk nawozowych, ograniczonej dostępności zasobów naturalnych oraz wahań cen surowców. Wprowadzenie precyzyjnych technologii zarządzania uprawami oraz zrównoważonych metod nawożenia wydaje się niezbędne, aby sprostać wyzwaniom XXI wieku, zapewniając równocześnie ochronę środowiska i długotrwałą produktywność gruntów.

3. Charakterystyka problemu

Poniższy rozdział skupia się na istotnych aspektach problematyki, takich jak: rola szybkiej diagnozy w utrzymaniu zdrowia roślin i poprawie wydajności upraw, specyfika środowiska upraw szklarniowych wymagających precyzyjnej kontroli, a także wyzwania związane z automatycznym rozpoznawaniem objawów chorobowych na liściach. Przedstawienie tych zagadnień pozwala lepiej zrozumieć kontekst, w którym opracowywany jest algorytm diagnostyczny opisany w dalszej części pracy.

3.1. Specyfika upraw szklarniowych

Uprawy szklarniowe odgrywają bardzo ważną rolę w globalnej produkcji żywności, umożliwiając całoroczne dostawy warzyw, owoców i ziół, niezależnie od warunków atmosferycznych. W obliczu zmian klimatycznych, degradacji gleby oraz rosnącego zapotrzebowania na żywność, coraz większy nacisk kładzie się na rozwój technologii szklarniowych, które pozwalają na optymalizację procesów produkcji i redukcję strat plonów.

Szklarnie umożliwiają precyzyjną kontrolę warunków środowiskowych, co skutkuje wyższą jakością i efektywnością upraw w porównaniu do rolnictwa tradycyjnego. Dzięki zastosowaniu komputerów klimatycznych, systemów nawadniania kropelkowego i doświetlania asymilacyjnego możliwe jest dostosowanie temperatury, wilgotności i dostępu do światła do specyficznych wymagań roślin. Takie rozwiązania pozwalają na znaczne zwiększenie plonów w stosunku do upraw polowych, jednocześnie zmniejszając wykorzystanie wody i nawozów chemicznych^[5].

Jednym z największych wyzwań upraw szklarniowych jest zarządzanie warunkami mikroklimatycznymi. Ograniczona cyrkulacja powietrza, wysoka wilgotność oraz kontrolowane warunki świetlne mogą sprzyjać rozwojowi patogenów. Szczególnie niebezpieczne są grzybowe choroby liści, takie jak mączniak prawdziwy czy szara pleśń, które mogą doprowadzić do niemal całkowitych strat w przypadku braku odpowiednich działań prewencyjnych.

Nowoczesne szklarnie wykorzystują innowacyjne technologie, takie jak systemy hydroponiczne i aeroponiczne, które eliminują konieczność stosowania gleby, a tym samym ograniczają ryzyko chorób glebowych. Takie rozwiązania umożliwiają również większą kontrolę nad składem odżywczym roślin, co przyczynia się do wyższej jakości i wartości odżywczej produktów.

Pomimo licznych zalet, produkcja szklarniowa wiąże się z pewnymi ograniczeniami. Wysokie koszty inwestycji oraz konieczność stałego monitorowania warunków środowiskowych stanowią poważne wyzwania dla producentów. Wprowadzenie zautomatyzowanych systemów diagnostycznych, opartych na sztucznej inteligencji i analizie obrazów, może jednak znacząco zwiększyć efektywność upraw poprzez szybką identyfikację niedoborów składników pokarmowych i chorób.

3.2. Znaczenie diagnozy niedoborów i chorób roślin

Niedobory składników pokarmowych oraz infekcje roślinne mają ogromny wpływ na obniżenie plonów, stanowiąc jeden z najczęściej występujących problemów współczesnego rolnictwa. Rozpoznawanie niedoborów składników pokarmowych oraz objawów chorób roślin na podstawie ich wyglądu zewnętrznego, a w szczególności liści, jest kluczowym zagadnieniem w rolnictwie precyzyjnym. Liście, jako główne organy asymilacyjne, często jako pierwsze wykazują symptomy zaburzeń fizjologicznych lub infekcji biotycznych, dlatego ich analiza stanowi cenne źródło informacji o stanie zdrowia całej rośliny. Wczesne wykrycie problemów pozwala na szybką interwencję, minimalizując straty plonów i poprawiając efektywność produkcji^[6].

Każdy pierwiastek odgrywa kluczową rolę w metabolizmie roślin, a jego niedobór prowadzi do zaburzeń fizjologicznych, co skutkuje zmniejszeniem powierzchni asymilacyjnej i ograniczeniem zdolności do efektywnego pobierania składników odżywczych^[7]. Wśród najważniejszych niedoborów można wymienić:

- **Niedobór azotu** – powoduje zahamowanie wzrostu roślin, chlorozy liści oraz spadek zawartości chlorofilu, co skutkuje redukcją plonów.
- **Niedobór fosforu** – prowadzi do słabego rozwoju systemu korzeniowego, ograniczając efektywność pobierania wody i składników odżywczych, co znacznie wpływa na zmniejszenie plonu i zdrowia roślin.
- **Niedobór potasu** – wpływa na odporność roślin na stresy środowiskowe oraz regulację gospodarki wodnej, zwiększając podatność na choroby i zmniejszając plony.
- **Niedobór magnezu** – zaburza proces fotosyntezy i syntezę białek, co prowadzi do przedwczesnego starzenia się roślin i redukcji biomasy.
- **Niedobór żelaza** – skutkuje chlorozą międzyżyłkową liści, co obniża zdolność roślin do absorpcji światła i prowadzi do zahamowania wzrostu.

Nieodpowiednie nawożenie nie tylko obniża wydajność upraw, ale także prowadzi do degradacji gleby, zwiększając ryzyko wystąpienia chorób roślin oraz kumulacji szkodliwych substancji w ekosystemie. Choroby roślin stanowią również poważne zagrożenie dla produkcji rolnej, wpływając negatywnie na jakość i ilość zbiorów. Ich wpływ zależy głównie od czynników środowiskowych, odporności upraw oraz czasu reakcji na pojawiające się objawy^[8]. Wśród najczęściej występujących infekcji można wyróżnić:

- **Grzybicze choroby liści** – mączniak prawdziwy, rdza zbóż, alternarioza oraz septorioza mogą zmniejszyć plony nawet o blisko połowę, szczególnie w warunkach wysokiej wilgotności i ograniczonej cyrkulacji powietrza.
- **Infekcje bakteryjne** – bakteryjne czernienie pszenicy, bakteriozy kapustnych czy bakteryjna kanciasta plamistość pomidora prowadzą do spadku jakości plonów, powodując znaczne straty.

- **Choroby wirusowe** – żółta karłowatość jęczmienia czy mozaika ogórka mogą powodować zahamowanie wzrostu, deformacje liści oraz utratę zdolności roślin do prawidłowego plonowania.

Wczesna diagnoza chorób oraz niedoborów składników pokarmowych ma kluczowe znaczenie dla zapewnienia stabilności produkcji rolniczej. Skuteczne utrzymywanie zdrowia roślin poprzez monitorowanie objawów i wdrażanie odpowiednich zabiegów może znacząco poprawić plony oraz zmniejszyć konieczność stosowania środków chemicznych. Tradycyjne metody identyfikacji chorób i niedoborów są często uzależnione od indywidualnej wiedzy eksperta, co niesie ryzyko błędnej diagnozy i opóźnienia interwencji. Ponadto, analiza laboratoryjna próbek roślin jest kosztowna i czasochłonna przez co dostępność precyzyjnych metod diagnostycznych jest znacznie ograniczona dla mniejszych gospodarstw rolnych.

Zastosowanie sztucznej inteligencji w diagnostyce wizualnej roślin może znacząco zwiększyć skuteczność oraz szybkość wykrywania objawów chorób i niedoborów. Algorytmy uczenia maszynowego oraz głębokie sieci neuronowe, umożliwiają automatyczne rozpoznawanie wzorców na zdjęciach liści, co pozwala na skalowalne i powtarzalne monitorowanie kondycji roślin w czasie rzeczywistym. W obliczu zmian klimatycznych, rosnącego zapotrzebowania na żywność oraz potrzeby optymalizacji produkcji, automatyzacja procesów diagnostycznych staje się niezbędna. Technologie oparte na sztucznej inteligencji mogą znacząco usprawnić zarządzanie uprawami, ograniczyć koszty, zminimalizować straty plonów i poprawić efektywność nawożenia oraz ochrony roślin.

3.3. Wyzwania w rozpoznawaniu wizualnym

Rozpoznawanie wizualne niedoborów składników pokarmowych oraz chorób roślin wiąże się z wieloma wyzwaniami natury technicznej, biologicznej oraz środowiskowej. Główne trudności wynikają z wysokiego stopnia złożoności oraz zmienności objawów występujących na powierzchni liści. Objawy wielu niedoborów pokarmowych oraz chorób mogą być do siebie bardzo podobne - np. chlorozę (żółknięcie liści) może powodować zarówno niedobór azotu, jak i infekcja grzybicza. Tego typu zbieżności znacząco utrudniają jednoznaczną klasyfikację wizualną.

Kolejnym wyzwaniem jest wpływ czynników środowiskowych na wygląd roślin. Warunki oświetleniowe, kąt padania światła, cień, wilgotność powietrza, a nawet kurz lub krople wody mogą zaburzać cechy istotne dla modelu klasyfikacyjnego. Zmienność odmian roślin również wpływa na zróżnicowanie wyglądu tych samych objawów - liście różnych odmian mogą mieć różny kształt, kolor i fakturę, co dodatkowo komplikuje zadanie rozpoznawania.

Problemem jest także jakość oraz dostępność danych. Wiele zbiorów danych zawiera obrazy w ograniczonej rozdzielczości lub nierównomiernie rozłożone klasy (tzw. niezbalansowane dane), co może prowadzić do błędów w predykcji i przeuczenia modelu na dominujących klasach. Zebranie dużej liczby dobrze opisanych przykładów z rzeczywistych upraw jest kosztowne i czasochłonne.

Nie bez znaczenia pozostają również ograniczenia modeli głębokiego uczenia. Chociaż sieci neuronowe wykazują wysoką skuteczność w zadaniach rozpoznawania obrazów, wymagają one dużych zbiorów danych do efektywnego trenowania, a ich działanie często bywa trudne do zinterpretowania przez użytkownika końcowego - co w kontekście rolnictwa precyzyjnego, gdzie decyzje muszą być uzasadnione, może stanowić barierę we wdrażaniu. Z tych powodów skuteczne rozpoznawanie wizualne wymaga nie tylko odpowiednio

dobrych modeli, ale również starannego przygotowania danych, metod zwiększania ich różnorodności (augmentacja danych), kalibracji wyników oraz integracji z wiedzą ekspercką.

4. Przegląd literatury i aktualnych rozwiązań

W ostatnich latach rosnące zainteresowanie automatyzacją procesów rolniczych doprowadziło do powstania licznych systemów wykorzystujących analizę obrazu do diagnozy chorób i niedoborów pokarmowych roślin. Choć wiele z tych rozwiązań wykazuje wysoką skuteczność w badaniach eksperymentalnych, nie znajdują zastosowania w rzeczywistych warunkach, zwłaszcza w środowiskach szklarniowych. W niniejszym rozdziale dokonano przeglądu tradycyjnych metod diagnozy, zastosowań sztucznej inteligencji w rolnictwie oraz przeglądu technik klasyfikacji obrazów, zwracając uwagę na ograniczenia istniejących zbiorów danych.

4.1. Tradycyjne metody diagnostyki roślin

Tradycyjne metody diagnostyki roślin od wielu dekad są fundamentem oceny stanu ich zdrowia. W literaturze podkreśla się, że klasyczne podejścia opierają się głównie na obserwacji wizualnej oraz analizach laboratoryjnych, które umożliwiają precyzyjne określenie składu chemicznego gleby i tkanek roślinnych^[9]. Choć metody te są często uznawane za bardzo dokładne, ich zastosowanie w dużej skali pozostaje ograniczone ze względu na wysokie koszty, długi czas oczekiwania na wyniki oraz zależność od subiektywnej oceny specjalistów.

W kontekście wczesnej detekcji problemów fizjologicznych roślin, szczególnie istotną rolę odgrywają współczynniki wegetacyjne (tzw. indeksy roślinne), które wyznacza się na podstawie analizy widma odbitego światła. Najczęściej wykorzystywanymi współczynnikami są NDVI (Normalized Difference Vegetation Index), GNDVI (Green NDVI) czy SAVI (Soil Adjusted Vegetation Index), które uwzględniają wartości odbicia w kanałach czerwonym, bliskiej podczerwieni oraz zielonym. Już na poziomie barwy liścia – mierzonej w przestrzeniach RGB czy HSV – możliwe jest zarejestrowanie pierwszych oznak stresu fizjologicznego, takich jak chloroza, nekroza, czy zaburzenia w transporcie składników. Zmiany te nie zawsze są dostrzegalne gołym okiem, ale mogą być szybko wykryte przez algorytmy analizy obrazu. Tym samym współczynniki te stanowią wczesny sygnał ostrzegawczy, często poprzedzający wyraźne objawy niedoborów lub chorób, co umożliwia wdrożenie działań interwencyjnych zanim dojdzie do utraty plonów.

Jednym z najbardziej rozpowszechnionych podejść jest wizualna ocena stanu zdrowia roślin. Metoda ta polega na dokładnym badaniu liści, łodyg i innych organów roślinnych przez doświadczonych agronomów i plantatorów. Charakterystyczne zmiany, takie jak przebarwienia, pojawianie się plam, deformacje czy zahamowania wzrostu, uważa się za wskaźniki niedoborów składników pokarmowych lub infekcji patogenami. W praktyce objawy te porównuje się z atlasami chorób roślin oraz specjalistycznymi podręcznikami, co umożliwia szybką identyfikację problemu i podjęcie odpowiednich działań interwencyjnych. Mimo, że ta metoda charakteryzuje się niskimi kosztami i jest relatywnie szybka, jej główną wadą pozostaje subiektywność oraz dostępność specjalistów.

Kolejnym istotnym elementem tradycyjnych metod diagnostycznych są analizy chemiczne oraz laboratoryjne. Precyzyjne określenie zawartości pierwiastków odżywczych w glebie i tkankach roślinnych odbywa się przy pomocy technik takich jak spektrofotometria, chromatografia czy analiza spektroskopowa. Dzięki tym metodom możliwe jest nie tylko

potwierdzenie niedoborów pokarmowych, ale również wykrycie nadmiaru niektórych składników, który również negatywnie wpływa na wzrost roślin. Analizy te, choć dostarczają wyników o bardzo wysokiej dokładności, są zazwyczaj czasochłonne oraz wymagają dostępu do specjalistycznego sprzętu laboratoryjnego i wykwalifikowanego personelu, co czyni je mniej przydatnymi przy rutynowym monitorowaniu dużych gospodarstw.

W ostatnich latach coraz większe zainteresowanie w diagnostyce roślin budzą metody molekularne. Techniki takie jak reakcja łańcuchowa polimerazy (PCR) czy testy serologiczne (np. ELISA) stosowane są do wykrywania patogenów na poziomie genetycznym, jeszcze zanim pojawią się widoczne objawy zachorowania. Choć diagnostyka molekularna daje bardzo precyzyjne wyniki, jest droga i wymaga odpowiednich warunków pracy, co ogranicza jej szerokie wdrożenie na poziomie codziennych praktyk rolniczych.

Podsumowując, tradycyjne metody diagnostyki roślin – zarówno wizualna ocena, jak i analizy chemiczne oraz molekularne – stanowią solidną podstawę dla oceny stanu zdrowia upraw. W praktyce jednak ich ograniczenia, wynikające z wysokich kosztów, długiego czasu oczekiwania na wyniki oraz subiektywności ocen, sprawiają, że w intensywnych systemach produkcyjnych, takich jak rolnictwo szklarniowe, poszukuje się alternatyw umożliwiających automatyzację i szybką analizę. Dlatego też coraz większe zainteresowanie budzi zastosowanie technologii opartych na automatycznej analizie obrazu, które mogą znacząco wspierać tradycyjne metody diagnostyczne, poprawiając precyzję oraz skalowalność procesów monitorowania stanu roślin.

4.2. Zastosowanie sztucznej inteligencji w rolnictwie

Sztuczna inteligencja znajduje coraz szersze zastosowanie w nowoczesnym rolnictwie, stanowiąc rewolucyjny element transformacji technologicznej tego sektora. Współczesne systemy oparte na SI integrują dane zbierane przez czujniki, kamery oraz satelity, umożliwiając inteligentne zarządzanie różnorodnymi procesami produkcyjnymi, takimi jak prognozowanie plonów, optymalizacja nawadniania czy zarządzanie gospodarstwem. Przykładowe zastosowania obejmują również detekcję chwastów i planowanie zabiegów herbicydowych, automatyczne rozpoznawanie dojrzałości owoców, sterowanie robotami rolniczymi i dronami, a także prognozowanie wystąpienia chorób roślin na podstawie danych pogodowych i biologicznych^[10].

Diagnoza niedoborów pierwiastków oraz chorób na podstawie analizy obrazów liści stanowi naturalne rozwinięcie powyższych rozwiązań. Automatyczna analiza obrazów umożliwia szybkie wykrycie subtelnych zmian w morfologii liści, które mogą wskazywać na pojawienie się problemów zdrowotnych roślin, jeszcze zanim wyraźnie ujawnią się symptomy w postaci poważnych uszkodzeń^[11]. Wczesne wykrycie tego typu anomalii pozwala na natychmiastowe podjęcie działań interwencyjnych, co jest szczególnie istotne w intensywnych systemach produkcji szklarniowej, gdzie minimalizacja strat plonowych i optymalizacja zasobów mają kluczowe znaczenie.

Warto zauważyć, że rozwój skutecznych modeli diagnostycznych^[12] opartych na sztucznej inteligencji jest w dużym stopniu uzależniony od dostępności odpowiednich zbiorów danych^[13]. Wiele wysokiej jakości zbiorów danych, obejmujących różnorodne gatunki roślin i szeroką gamę objawów chorób oraz niedoborów, stanowi własność prywatnych firm. Takie dane są często udostępniane wyłącznie odpłatnie, a koszty ich licencjonowania mogą być bardzo wysokie, co stanowi istotną barierę. Spośród dostępnych publicznie i darmowych zbiorów danych, znacząca część koncentruje się na pojedynczych gatunkach roślin. Przykładowo:

- **Banana Leaf Dataset** – zbiór skupiony wyłącznie na liściach bananowców, obejmujący kilka wyraźnych chorób takich jak *Black Sigatoka* i *Bacterial Wilt*. Choć skuteczność modeli na tym zbiorze jest wysoka, ich zastosowanie w innych uprawach okazuje się problematyczne z powodu braku uniwersalności.
- **Coffee Leaf Dataset** – zawiera zdjęcia zdrowych i porażonych liści kawowca, głównie objętych rdzą liści (*Coffee Leaf Rust*). Jest to przykład zbioru dobrze dopasowanego do konkretnego celu, ale bezużytecznego poza uprawą kawy.
- **Rice Leaf Disease Dataset** – zbiór skoncentrowany na ryżu, zawiera obrazy liści z typowymi chorobami jak *blast*, *brown spot* i *tungro*. Podobnie jak wcześniej, nie nadaje się do przeniesienia na inne rośliny.

Choć wyniki na tych zbiorach sugerują wysoką skuteczność, ograniczona różnorodność danych oraz wąski zakres gatunkowy powodują, że modele te nie znajdują zastosowania w warunkach szklarniowych, gdzie uprawy są bardziej zróżnicowane, a warunki środowiskowe silnie kontrolowane. Podobne ograniczenia obserwuje się w przypadku zbiorów takich jak OLID (Open Leaf Image Dataset), który zawiera zdjęcia zdrowych oraz chorych liści głównej roślinności Bangladeszu. Choć OLID prezentuje bardziej realistyczne warunki zdjęć dzięki danym pochodzącym z różnych źródeł, nie znajduje on praktycznego zastosowania w algorytmie przeznaczonym do upraw szklarniowych. Zbiór ten skupia się głównie na zdjęciach różnych gatunków drzew, a nie roślin szklarniowych, co ogranicza jego przydatność w wspomnianym kontekście. Dodatkowo, nie spełnia on wymagań niezawodności, niezbędnej do stworzenia systemu eksperckiego dla producentów szklarniowych. Problemy te obejmują m.in.:

- brak standaryzacji oświetlenia i odległości zdjęć,
- brak danych dla wielu popularnych gatunków szklarniowych,
- małą liczbę zdjęć dla rzadziej występujących, ale istotnych klas objawów.

W kontekście zastosowania szklarniowego **PlantVillage** pozostaje jednym z najlepszych zbiorów danych. Zbiór ten zawiera obrazy liści wielu gatunków roślin, w tym także tych powszechnie uprawianych w szklarniach. Należy jednak zaznaczyć, że dane zostały zgromadzone głównie w kontrolowanych warunkach laboratoryjnych. Jego główne ograniczenia to brak różnorodnych warunków świetlnych i środowiskowych, co może wpływać na skuteczność modeli uczonych na jego podstawie w rzeczywistych uprawach. Z tego względu wiele modeli, mimo obiecujących wyników na danych testowych, okazuje się zawodnych w praktyce.

4.3. Modele klasyfikacji obrazów – przegląd technik

Klasyfikacja obrazów stanowi jedno z kluczowych zastosowań sztucznej inteligencji i widzenia komputerowego, odgrywając istotną rolę w takich dziedzinach jak medycyna, systemy bezpieczeństwa, przemysł wizyjny oraz rolnictwo. W kontekście rolnictwa, techniki klasyfikacyjne znalazły szczególnie szerokie zastosowanie w diagnostyce stanu zdrowia roślin, a w szczególności w identyfikacji chorób i niedoborów składników odżywczych na podstawie zdjęć liści. Skuteczna klasyfikacja obrazów umożliwia automatyczne wykrywanie nieprawidłowości w strukturze, kolorze czy fakturze liścia, co może świadczyć o rozwijającym

się patogenie, deficycie pierwiastków lub stresie środowiskowym. Początkowo w analizie obrazów wykorzystywano tradycyjne algorytmy uczenia maszynowego, takie jak:

- **K-najbliższych sąsiadów (KNN)** – klasyfikator oparty na podobieństwie między wektorami cech obrazów, wykazujący dobrą skuteczność przy prostych zadaniach,
- **Maszyny wektorów nośnych (SVM)** – efektywny model przy separacji liniowej i nieliniowej, zwłaszcza w zadaniach binarnej klasyfikacji,
- **Lasy losowe (Random Forest)** – model oparty na zbiorze drzew decyzyjnych, odporny na przeuczenie i dobrze radzący sobie z danymi heterogenicznymi.

Choć podejścia klasyczne są proste w implementacji i wymagają mniejszych zasobów obliczeniowych, ich skuteczność jest ograniczona w sytuacjach, gdy dane są złożone, obejmują wiele klas lub zawierają zakłócenia, jak to często ma miejsce w przypadku zdjęć liści w środowiskach naturalnych lub szklarniowych.

W odpowiedzi na te ograniczenia, współczesne podejście do klasyfikacji obrazów coraz częściej opiera się na modelach głębokiego uczenia, a w szczególności na konwolucyjnych sieciach neuronowych (CNN)^[14]. Są to architektury, które uczą się automatycznie hierarchicznych reprezentacji cech – od prostych krawędzi i tekstur po złożone wzorce charakterystyczne dla określonych klas. Głębokie sieci neuronowe eliminują konieczność ręcznego projektowania cech, pozwalając na lepsze dopasowanie modelu do złożonych i różnorodnych danych wizualnych. Do najczęściej stosowanych architektur CNN w zadaniach klasyfikacji liści należą:

- **VGGNet** – sieć o głębokiej, warstwowej strukturze, wykorzystująca małe filtry konwolucyjne (3x3). Ceniona za prostotę i przewidywalność działania, jednak wymagająca dużej mocy obliczeniowej.
- **ResNet (Residual Network)** – wprowadza tzw. połączenia rezydualne (*skip connections*), umożliwiające trenowanie bardzo głębokich modeli bez problemu zaniku gradientu. Doskonale sprawdza się w klasyfikacjach o dużym zróżnicowaniu klas i danych.
- **Inception** – sieć oparta na równoległych filtrach różnej wielkości w jednej warstwie. Pozwala uchwycić cechy o różnych skalach jednocześnie, co jest szczególnie użyteczne w przypadku objawów chorobowych o zróżnicowanej morfologii.
- **EfficientNet** – nowoczesna i zoptymalizowana architektura łącząca wysoką dokładność z efektywnością obliczeniową^[15]. Dzięki zrównoważonej skalowalności parametrów sieci (szerokość, głębokość, rozdzielczość wejścia) znajduje zastosowanie w nowoczesnych systemach rolniczych
- **MobileNet** – lekka sieć przeznaczona do zastosowań mobilnych i wbudowanych, umożliwiającą wdrażanie systemów klasyfikacyjnych bezpośrednio na urządzeniach terenowych, takich jak smartfony, tablety czy drony^[16].

W praktyce rolniczej, jednym z kluczowych problemów jest ograniczona dostępność dużych, odpowiednio opisanych zbiorów danych dotyczących roślin uprawnych. Z tego powodu coraz powszechniej stosuje się technikę transfer learning^[17], czyli uczenia transferowego. Polega ona na wykorzystaniu modeli wstępnie wytrenowanych na dużych

zbiorach danych ogólnych, takich jak ImageNet (zawierający miliony obrazów z tysięcy kategorii), a następnie ich dostosowaniu do konkretnego zadania diagnostycznego – np. rozpoznania niedoboru azotu lub objawów mączniaka prawdziwego na liściu pomidora. Uczenie transferowe pozwala skrócić czas trenowania modelu, znacząco ograniczyć wymagania ilościowe względem lokalnych zbiorów danych oraz poprawić dokładność klasyfikacji w warunkach ograniczonego dostępu do danych.

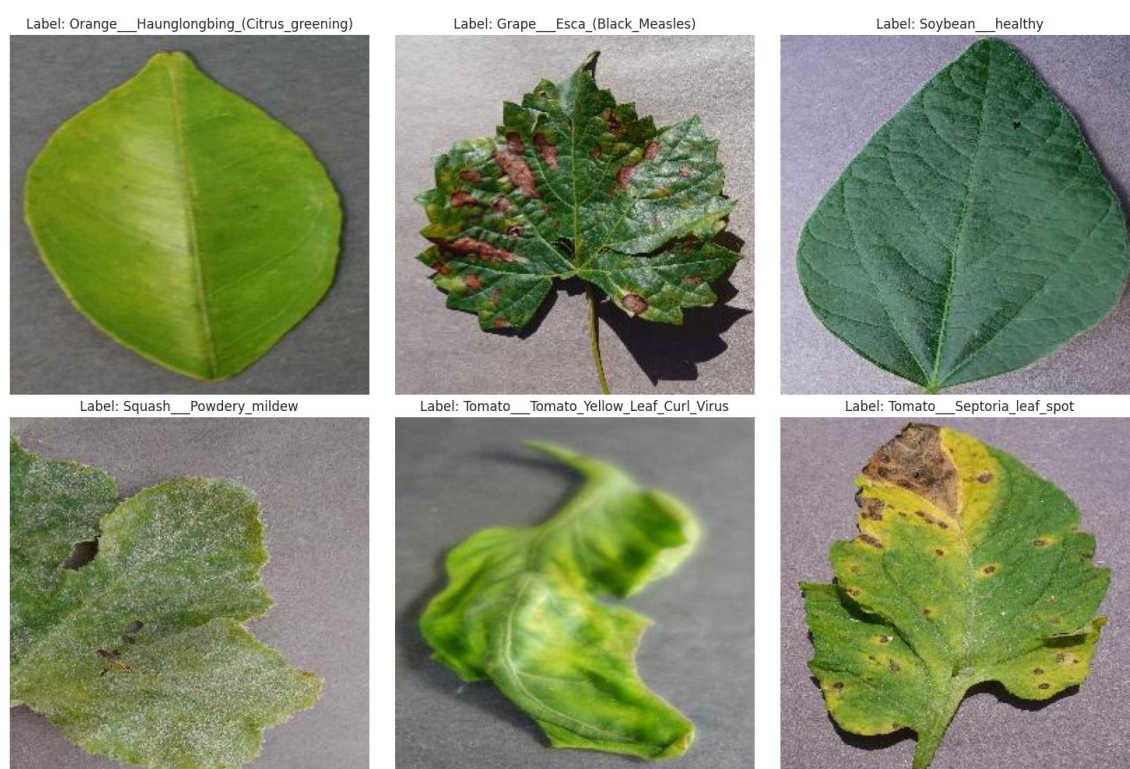
W kontekście diagnostyki szklarniowej, techniki te umożliwiają stworzenie praktycznego i skalowalnego rozwiązania, stanowiącego podstawę systemów eksperckich wspierających decyzje rolników bez konieczności gromadzenia dziesiątek tysięcy przykładów dla każdej klasy.

5. Opis danych i etap przygotowania

W niniejszym rozdziale przedstawiono dane wykorzystane do trenowania modeli klasyfikacyjnych. Opisano źródło pochodzenia zbioru oraz jego strukturę, a także etapy przygotowania danych, obejmujące wstępny preprocessing i zastosowanie technik augmentacji obrazu. W końcowej części omówiono sposób podziału danych na zbiory treningowy, walidacyjny i testowy, co stanowi istotny element procesu uczenia i oceny skuteczności modeli.

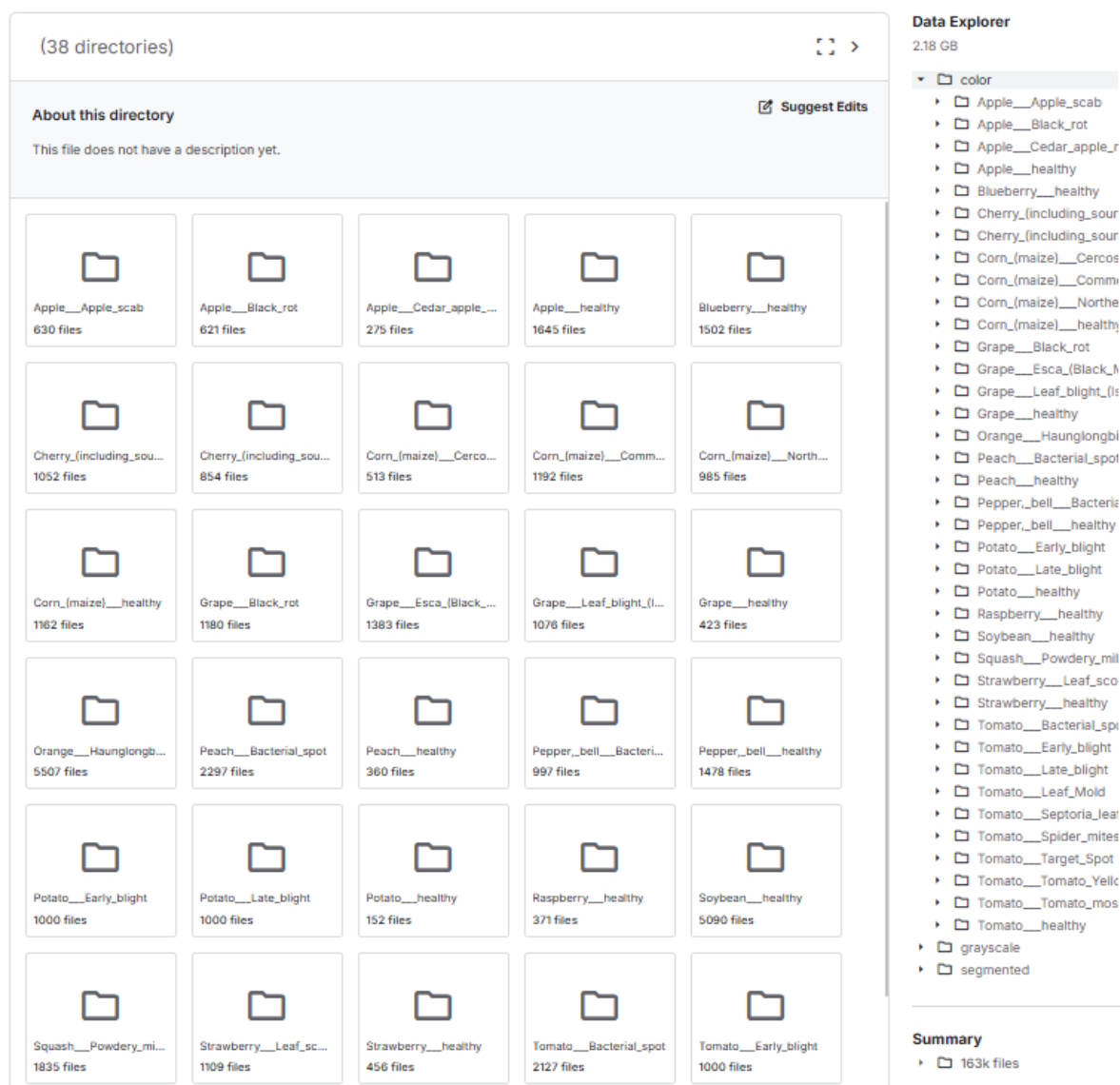
5.1. Źródła danych i struktura zbioru

Zbiór danych *PlantVillage* został udostępniony publicznie przez naukowców z Pennsylvania State University w ramach projektu mającego na celu opracowanie narzędzi wspomagających rolników w krajach rozwijających się^[18]. *PlantVillage* dostępny jest w serwisie Kaggle^[19], a jego otwarta wersja może być bezpłatnie używana w badaniach naukowych i projektach edukacyjnych. Obrazy w zbiorze zostały zebrane w warunkach laboratoryjnych i pół-polowych, głównie przy użyciu aparatów fotograficznych oraz smartfonów. Liście były odizolowywane od tła i fotografowane przy różnym oświetleniu, co zapewniało możliwie dobrą jakość wizualną materiału. Klasyfikacja chorób i niedoborów została dokonana przez ekspertów z dziedziny patologii roślin, co znacząco zwiększa wiarygodność etykiet przypisanych do danych obrazów. Na poniższym rysunku(rys. 5.1.1.) widoczne jest 6 losowych próbek danych ze zbioru wraz z etykietami:



Rys. 5.1.1. Sześć losowych próbek danych ze zbioru PlantVillage wyświetlone w Google Colab.

Zbiór zawiera 38 klas, z których każda reprezentuje konkretną roślinę oraz diagnozę np. *Tomato_Bacterial_spot* lub *Potato_Early_blight*. Struktura danych jest katalogowa, a każdy folder odpowiada jednej klasie diagnostycznej. W jego wnętrzu znajdują się zdjęcia liści przypisanych do tej klasy. Zdjęcie poniżej (rys. 5.1.2.) przedstawia strukturę zbioru danych oraz wszystkie katalogi reprezentujące klasy danych:



Rys. 5.1.2. Struktura zbioru danych oraz wszystkie katalogi reprezentujące klasy.

Takie uporządkowanie danych znacznie ułatwia automatyczne etykietowanie i wczytywanie danych do narzędzi typu *TensorFlow* czy *PyTorch*. Na tle innych ogólnodostępnych zbiorów danych roślinnych, *PlantVillage* wyróżnia się skalą, jakością oraz różnorodnością danych. W przeciwieństwie do zbiorów skupionych na pojedynczych roślinach (np. ryż, kawa, banan), *PlantVillage* obejmuje wiele gatunków uprawnych, z których część jest powszechnie spotykana w szklarniach, takich jak pomidor, dyniowate, truskawki oraz różne odmiany papyki. Z tego względu zbiór ten został uznany za najlepiej nadający się do realizacji algorytmu diagnostycznego przeznaczonego dla upraw szklarniowych, mimo że dane nie były pozyskiwane bezpośrednio w warunkach szklarniowych.

5.2. Przekształcenia wstępne i augmentacja danych

Przed przystąpieniem do trenowania modeli klasyfikacyjnych niezbędne było odpowiednie przygotowanie zbioru danych, tak aby nadawał się do efektywnego przetwarzania przez algorytmy uczenia maszynowego i głębokiego^[20]. Proces ten obejmował zarówno przekształcenia wstępne, jak i zastosowanie augmentacji danych w celu zwiększenia różnorodności i objętości zestawu treningowego.

Na potrzeby projektu opracowano funkcję `create_dataframe` (rys. 5.2.1.), której zadaniem było stworzenie uporządkowanej struktury danych wejściowych w postaci obiektu `DataFrame` z biblioteki `Pandas`. `DataFrame` to struktura przypominająca tabelę, w której dane przechowywane są w wierszach i kolumnach - co umożliwia wygodne filtrowanie, przetwarzanie oraz przekazywanie danych do dalszych etapów pipeline'u. Funkcja `create_dataframe` przeszukuje katalog główny zbioru obrazów, przypisuje każdemu zdjęciu jego ścieżkę dostępu oraz etykietę klasową (na podstawie nazwy katalogu nadrzędnego), a następnie zapisuje te informacje do ramki danych. Rozwiązanie to, umożliwiło łatwe zarządzanie danymi oraz wygodny podział na podzbiory (treningowy, walidacyjny i testowy).

```
# Pobranie datasetu za pomocą kagglehub
data = kagglehub.dataset_download("abdallahalidev/plantvillage-dataset")

# Ścieżka do danych (zmodyfikowana na katalog z "color" w dodatkowym folderze)
data = os.path.join(data, "plantvillage dataset/color")

# Sprawdzenie, czy dane zostały poprawnie załadowane
if not os.path.exists(data):
    print(f"Folder danych nie istnieje: {data}")
else:
    print(f"Dane znajdują się w folderze: {data}")

# Funkcja do tworzenia DataFrame z obrazami i etykietami
def create_dataframe(data_path):
    filepaths = [] # Lista na ścieżki do plików
    labels = [] # Lista na etykiety

    folds = os.listdir(data_path) # Lista podfolderów (klas)

    for fold in folds:
        f_path = os.path.join(data_path, fold) # Ścieżka do folderu klasy
        imgs = os.listdir(f_path) # Lista obrazów w folderze

        for img in imgs:
            img_path = os.path.join(f_path, img) # Ścieżka do obrazu
            filepaths.append(img_path) # Dodanie ścieżki do listy
            labels.append(fold) # Dodanie etykiety (nazwa folderu)

    # Tworzenie DataFrame z list
    fseries = pd.Series(filepaths, name='Filepaths')
    lseries = pd.Series(labels, name='Labels')
    return pd.concat([fseries, lseries], axis=1)

# Tworzenie DataFrame z danymi
df = create_dataframe(data)

# Wyświetlenie pierwszych 5 wierszy DataFrame
print(df.head())
```

Rys. 5.2.1. Fragment kodu przedstawiający funkcję `create_dataframe` przystosowującą dane do struktury `DataFrame`.

Po utworzeniu struktury DataFrame, dane zostały poddane wstępnej obróbce wizualnej w celu normalizacji danych wejściowych:

- **Zmiana rozmiaru** – obrazy zostały przeskalowane do jednolitych wymiarów, zgodnych z wymaganiami architektur modeli (np. 224×224 piksele w przypadku modeli Xception i ResNet).
- **Normalizacja pikseli** – wartości RGB zostały przekształcone do zakresu $[0, 1]$ lub znormalizowane względem statystyk zbioru ImageNet (w przypadku transfer learningu), co przyspiesza i stabilizuje proces uczenia^[21].
- **Konwersja etykiet do formatu numerycznego lub one-hot** – klasy tekstowe zostały zakodowane w sposób zrozumiały dla modelu klasyfikacyjnego.

Istotnym elementem przygotowania danych było również zastosowanie augmentacji danych. Celem augmentacji jest sztuczne zwiększenie liczby przykładów uczących poprzez wprowadzenie różnorodnych modyfikacji obrazu, co pozwala modelowi lepiej generalizować i unikać nadmiernego dopasowania do danych treningowych^[22]. Na rysunku poniżej (rys. 5.2.2.) przedstawiony jest fragment kodu, realizujący proces wstępnej obróbki wizualnej danych oraz funkcje realizującą augmentację:

```

bs=4 #batch size

# Augmentacja danych dla zbioru treningowego
train_datagen = ImageDataGenerator(
    rescale=1./255, # Skalowanie pikseli do przedziału [0, 1]
    rotation_range=20, # Losowy obrót obrazów
    width_shift_range=0.2, # Losowe przesunięcie w poziomie
    height_shift_range=0.2, # Losowe przesunięcie w pionie
    shear_range=0.2, # Losowe ścinanie
    zoom_range=0.2, # Losowe powiększenie
    horizontal_flip=True, # Losowe odbicie poziome
    fill_mode='nearest' # Uzupełnianie brakujących pikseli
)

# Tylko skalowanie dla walidacji i testów
valid_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Generatory danych
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepaths', # Kolumna ze ścieżkami do obrazów
    y_col='Labels', # Kolumna z etykietami
    target_size=(224, 224), # Rozmiar obrazu
    batch_size=bs,
    class_mode='categorical', # Klasyfikacja wieloklasowa
    shuffle=True,
    seed=42
)

valid_generator = valid_datagen.flow_from_dataframe(
    dataframe=valid_df,
    x_col='Filepaths',
    y_col='Labels',
    target_size=(224, 224),
    batch_size=bs,
    class_mode='categorical',
    shuffle=False
)

test_generator = test_datagen.flow_from_dataframe(
    dataframe=test_df,
    x_col='Filepaths',
    y_col='Labels',
    target_size=(224, 224),
    batch_size=bs,
    class_mode='categorical',
    shuffle=False
)

```

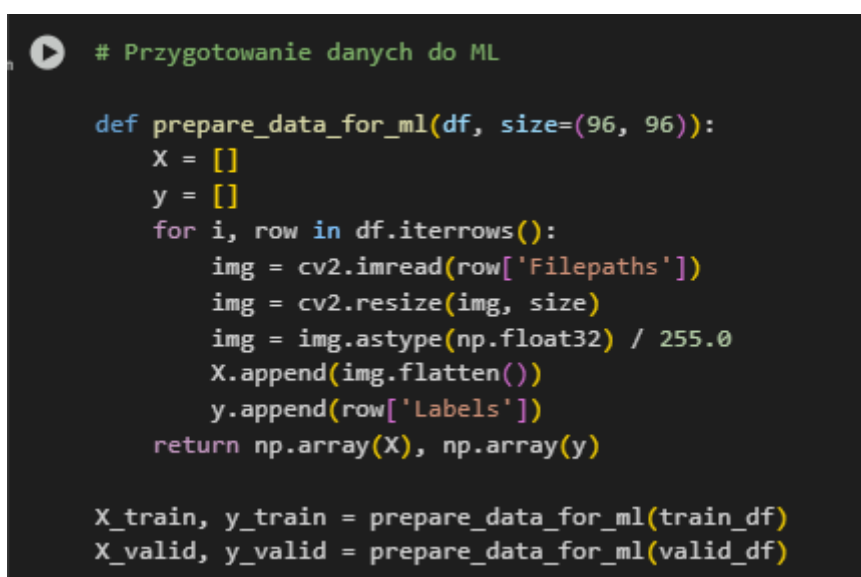
Rys. 5.2.2. Fragment kodu realizującego augmentację oraz proces wstępnej obróbki wizualnej danych

Augmentacja realizowana była przy użyciu generatora *ImageDataGenerator* z biblioteki *Keras*. Generator ten pozwala na dynamiczne tworzenie przekształconych wersji obrazów „w locie” podczas trenowania modelu, co znacząco ogranicza zużycie pamięci RAM i umożliwia obsługę dużych zbiorów danych bez konieczności ich wcześniejszego zapisu na dysku. Przekształcenia obejmowały m.in.:

- Losowe obroty obrazów (do ± 40 stopni)
- Przesunięcia w pionie i poziomie
- Losowe powiększenia (zoom)
- Odbicia lustrzane w poziomie
- Zmiany jasności i kontrastu

Zarówno przetworzone obrazy, jak i przypisane do nich etykiety, były następnie przekazywane do modeli w formie batched generatorów - co stanowiło gotowe wejście dla architektur głębokiego uczenia, takich jak Xception, ResNet50 oraz klasyczne CNN. Zastosowanie generatora danych nie tylko zwiększyło różnorodność danych treningowych, ale również pozwoliło na wydajne przetwarzanie danych w czasie rzeczywistym, co było szczególnie istotne przy ograniczonych zasobach sprzętowych.

W przypadku klasycznych modeli uczenia maszynowego, takich jak KNN, SVM czy Random Forest, niezbędne było dodatkowe przygotowanie danych różniące się od podejścia stosowanego w głębokim uczeniu. Modele te wymagają, aby dane wejściowe były reprezentowane jako wektory cech w postaci jednowymiarowych tablic numerycznych. W tym celu zastosowano specjalną funkcję pomocniczą *prepare_data_for_ml()* przedstawioną na zdjęciu poniżej(rys. 5.2.3):



```
# Przygotowanie danych do ML

def prepare_data_for_ml(df, size=(96, 96)):
    X = []
    y = []
    for i, row in df.iterrows():
        img = cv2.imread(row['Filepaths'])
        img = cv2.resize(img, size)
        img = img.astype(np.float32) / 255.0
        X.append(img.flatten())
        y.append(row['Labels'])
    return np.array(X), np.array(y)

X_train, y_train = prepare_data_for_ml(train_df)
X_valid, y_valid = prepare_data_for_ml(valid_df)
```

Rys. 5.2.3. Fragment kodu deklarujący funkcję *prepare_data_for_ml()*.

Funkcja *prepare_data_for_ml()* przekształca obrazy przechowywane w strukturze DataFrame do formatu odpowiedniego dla klasycznych algorytmów uczenia maszynowego. Każdy obraz z kolumny *Filepaths* jest wczytywany za pomocą biblioteki *cv2*, skalowany do rozmiaru 96×96 pikseli, a następnie normalizowany (przekształcany do zakresu [0, 1]). Po przekształceniu obraz zostaje spłaszczony (*flatten*) do postaci jednowymiarowego wektora długości 96×96×3 czyli 27 648. Takie podejście pozwala na zastosowanie modeli, które operują na klasycznych macierzach cech^[23], a nie na danych wielowymiarowych (np. 3D tensors)

typowych dla CNN. Wektor X zawiera więc numeryczne reprezentacje obrazów, natomiast y - przypisane do nich etykiety klasowe.

Następnie, aby przekształcić tekstowe etykiety klas w wartości numeryczne, zastosowano narzędzie `LabelEncoder()` z biblioteki `sklearn.preprocessing`. Proces ten, znany jako kodowanie etykiet, zamienia kategorie tekstowe (np. "Tomato_Bacterial_spot", "Potato_Healthy") na odpowiadające im liczby całkowite (np. 0, 1, 2, ...). Modele klasyczne nie obsługują zmiennych kategoriycznych bez wcześniejszego zakodowania ich do postaci liczbowej, dlatego operacja ta jest niezbędna. Na obrazie poniżej (rys. 5.2.4.) przedstawiony jest fragment kodu realizujący kodowanie etykiet dla modeli uczenia maszynowego:

```
# Zakodowanie etykiet
encoder = LabelEncoder()
y_train_enc = encoder.fit_transform(y_train)
y_valid_enc = encoder.transform(y_valid)
```

Rys. 5.2.4. Fragment kodu realizujący kodowanie etykiet zbioru danych dla klasycznych modeli ML.

Kod ten zapewnia spójną konwersję etykiet w zbiorze treningowym i walidacyjnym, przy czym metoda `fit_transform()` uczy się unikalnych klas na podstawie `y_train`, a `transform()` stosuje tę samą mapę kodowania do zbioru walidacyjnego. Dzięki temu dane są w pełni gotowe do trenowania klasycznych modeli klasyfikacyjnych, zapewniając zgodność zarówno pod względem formatu cech, jak i etykiet wyjściowych.

5.3. Podział na zbiory treningowy, walidacyjny i testowy

Podział danych na odrębne zbiory treningowy, walidacyjny i testowy stanowi istotnym elementem trenowania oraz oceny skuteczności modeli klasyfikujących^[24]. Celem takiego podziału jest zapobieganie przeuczeniu modelu oraz umożliwienie jego obiektywnej ewaluacji na danych których jeszcze „nie widział”^[25].

Zbiór treningowy stanowi największą i najważniejszą część danych, która jest wykorzystywana do nauki modelu. To właśnie na jego podstawie model iteracyjnie dostosowuje swoje wewnętrzne parametry, takie jak wagi i biasy, aby nauczyć się rozpoznawać wzorce i zależności niezbędne do prawidłowego wykonywania zadania klasyfikacji^[26]. Jest to główny zasób informacyjny, z którego model czerpie wiedzę i buduje swoje zdolności predykcyjne.

Zbiór walidacyjny pełni funkcję wewnętrznego sprawdzianu postępów modelu w trakcie procesu treningu. Po każdej epoce uczenia lub w określonych interwałach, model jest poddawany ocenie na tym zbiorze. Uzyskane wyniki na danych walidacyjnych są kluczowe do optymalizacji hiperparametrów modelu, takich jak szybkość uczenia czy złożoność architektury sieci. Pozwalają również na monitorowanie zjawiska przeuczenia, czyli sytuacji, w której model zaczyna zapamiętywać dane treningowe zamiast uczyć się ogólnych wzorców. Monitorowanie wydajności na zbiorze walidacyjnym umożliwia podejmowanie świadomych decyzji o przerwaniu treningu lub wprowadzeniu modyfikacji w architekturze modelu, zanim dojdzie do utrwalenia niepożądanych zachowań. Model korzysta z tego zbioru do oceny swojej wydajności, ale nie modyfikuje swoich wag bezpośrednio na jego podstawie.

Zbiór testowy stanowi ostateczny i całkowicie niezależny sprawdzian dla wytrenowanego modelu. Jest to zbiór danych, który pozostaje całkowicie poza procesem

zarówno treningu, jak i walidacji, co gwarantuje bezstronną ocenę. Wyniki uzyskane na zbiorze testowym dostarczają najbardziej realistycznego obrazu tego, jak model będzie generalizował i radził sobie z nowymi, nieznanymi wcześniej danymi w rzeczywistych zastosowaniach. Jest to kluczowa miara prawdziwej skuteczności i niezawodności modelu. Na poniższym rysunku (rys. 5.3.1.) przedstawiono kod realizujący podział zbioru danych na zbiór treningowy, walidacyjny oraz testowy:

```
# Podział danych na zbiory treningowy, walidacyjny i testowy
train_df, dummy_df = train_test_split(df, train_size=0.8, shuffle=True, random_state=42)
valid_df, test_df = train_test_split(dummy_df, train_size=0.5, shuffle=True, random_state=42)
```

Rys. 5.3.1. Fragment kodu dzielący zbiór na część treningową, walidacyjną oraz testową.

Na przedstawionym fragmencie kodu zastosowano podział w proporcji 80% (trening), 10% (walidacja) i 10% (test), zapewniając tym samym odpowiedni balans pomiędzy ilością danych do nauki i oceny, przy jednoczesnym zachowaniu reprezentatywności klas w każdym z zestawów^[27].

6. Opis i implementacja modeli

Poniższy rozdział zawiera opis wybranych modeli uczenia maszynowego i głębokiego, które zostały zaimplementowane i porównane w ramach badań. Każdy podrozdział zawiera zarówno omówienie teoretycznych podstaw działania danego modelu, jak i szczegółowy opis jego implementacji w środowisku Google Colab, z uwzględnieniem zastosowanych warstw, parametrów oraz technik optymalizacyjnych.

W badaniu porównano łącznie sześć modeli – trzy reprezentujące podejście głębokiego uczenia oraz trzy klasyczne algorytmy uczenia maszynowego. Wśród modeli głębokiego uczenia zastosowano dwie architektury wykorzystujące uczenie transferowe: **Xception** oraz **ResNet50**, które zestawiono z własnoręcznie zaprojektowaną konwolucyjną siecią neuronową (CNN). Dla porównania, jako klasyczne algorytmy uczenia maszynowego wykorzystano: **KNN** (k-najbliższych sąsiadów), **SVM** (maszynę wektorów nośnych) oraz **Random Forest**. Celem porównania było zbadanie, które z podejść cechuje się najwyższą skutecznością w klasyfikacji obrazów liści roślin uprawnych wykazujących objawy niedoborów składników pokarmowych lub chorób.

6.1. Transfer learning – model Xception

Model Xception został wybrany jako reprezentant nowoczesnych architektur wykorzystujących transfer learning w klasyfikacji obrazów. Dzięki swojej strukturze opartej na separowalnych konwolucjach głębokich^[28], model ten pozwala na uzyskanie wysokiej dokładności przy relatywnie niewielkiej liczbie parametrów. W kontekście diagnostyki wizualnej objawów na liściach roślin, Xception oferuje dużą elastyczność i zdolność do wychwytywania złożonych wzorców, co czyni go odpowiednim kandydatem do zastosowań w rolnictwie precyzyjnym. Model został zaadaptowany do zadania klasyfikacji zdjęć liści na podstawie danych z PlantVillage.

Xception (ang. Extreme Inception) to głęboka konwolucyjna sieć neuronowa zaproponowana przez François Chollata, stanowiąca rozwinięcie architektury Inception. Główna innowacja polega na zastąpieniu klasycznych modułów Inception warstwami tzw. separowalnych głębokich konwolucji (depthwise separable convolutions). Taka struktura umożliwia znaczne zmniejszenie liczby parametrów modelu przy zachowaniu wysokiej skuteczności klasyfikacyjnej. Xception bazuje na założeniu, że mapowanie przestrzeni kanałowej i przestrzennej cech można oddzielić - najpierw przetwarzając każdy kanał osobno, a następnie mieszając informacje między kanałami. Jest to architektura, która osiąga wysoką skuteczność na zbiorach takich jak ImageNet i doskonale nadaje się do transfer learningu^[29]. Poniżej przedstawiono fragment kodu (rys. 6.1.1.) implementujący metodę transfer learningu opartą o model Xception:

```
[ ] # Model 1: Transfer Learning - Xception
base_model = tf.keras.applications.xception.Xception(weights='imagenet', include_top=False, input_shape=(224, 224, 3), pooling='max')

model = Sequential([
    base_model, # Pretrenowany model
    BatchNormalization(), # Normalizacja wsadowa
    Dense(256, activation='relu'), # Warstwa gęsta
    Dropout(0.5), # Dropout dla regularizacji
    Dense(38, activation='softmax') # Warstwa wyjściowa (38 klas)
])

# Kompilacja modelu z dodatkowymi metrykami
model.compile(
    Adamax(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy', Precision(name='precision'), Recall(name='recall'), F1Score()]
)

# Trenowanie modelu
start_time = time.time()
xception_history = model.fit(
    x=train_generator,
    validation_data=valid_generator,
    epochs=5, # Liczba epok
    verbose=1,
    validation_steps=None,
    shuffle=False
)
xception_time = time.time() - start_time
```

Rys. 6.1.1. Fragment kodu implementujący metodę transfer learningu opartą o model Xception.

Model Xception został zaimplementowany z wykorzystaniem biblioteki *TensorFlow Keras*, używając funkcji *tf.keras.applications.xception.Xception*. Jako bazę zastosowano wstępnie wytrenowaną wersję modelu z wagami imagenet. Oryginalna końcowa warstwa klasyfikacyjna została usunięta (*include_top = False*), a na wyjściu modelu zastosowano agregację cech przez globalne maksymalne poolingowanie (*pooling = 'max'*), co zmniejsza wymiar i zapobiega przeuczeniu. Do modelu dodano kolejne warstwy:

- BatchNormalization - normalizacja aktywacji mająca na celu stabilizację uczenia,
- Dense(256, activation = 'relu') - warstwa gęsta z funkcją aktywacji ReLU, zwiększająca zdolność modelu do reprezentowania złożonych zależności,
- Dropout(0.5) - mechanizm regularizacji zapobiegający przeuczeniu,
- Dense(38, activation = 'softmax') - warstwa wyjściowa z aktywacją softmax, dopasowana do 38 klas danych.

Model skompilowano z optymalizatorem **Adamax** o współczynniku uczenia 0.001, funkcją straty *categorical_crossentropy* oraz metrykami *accuracy*, *Precision*, *Recall* i *F1Score*. Trening prowadzono przez 5 epok z użyciem generatora danych *train_generator*.

6.2. Transfer learning – model ResNet50

Drugim modelem wykorzystującym transfer learning w ramach przeprowadzonych badań jest ResNet50 – jedna z najczęściej stosowanych i sprawdzonych architektur głębokich sieci neuronowych^[30]. Dzięki zastosowaniu połączeń rezydualnych (ang. skip connections), ResNet50 umożliwia skuteczne trenowanie bardzo głębokich modeli bez problemu zaniku gradientu^[31]. W projekcie model ten został użyty jako baza, na której zbudowano własny

klasyfikator dostosowany do danych diagnostycznych. Włączenie ResNet50 do porównania pozwala ocenić, jak sprawdza się klasyczna architektura głębokiego uczenia w zadaniach związanych z analizą obrazu w środowisku szklarniowym.

ResNet50 to model głębokiej sieci konwolucyjnej zawierający 50 warstw, oparty na architekturze Residual Network. Kluczową cechą ResNet jest zastosowanie tzw. połączeń rezydualnych, które przeciwdziałają zanikowi gradientu przy trenowaniu bardzo głębokich modeli. Pozwalają one na skuteczne trenowanie sieci liczących nawet setki warstw.

Model ten osiąga wysokie wyniki klasyfikacyjne i często jest wykorzystywany jako baza w transfer learningu dla zadań klasyfikacji obrazów. Poniżej przedstawiono fragment kodu (rys. 6.2.1.) implementujący metodę transfer learningu opartą o model ResNet:

```
[ ] # Model 2: Transfer Learning - ResNet50
base_resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3), pooling='avg')

resnet_model = Sequential([
    base_resnet,
    BatchNormalization(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(38, activation='softmax')
])

resnet_model.compile(optimizer=Adam(learning_rate=0.0005),
                    loss='categorical_crossentropy',
                    metrics=['accuracy', Precision(name='precision'), Recall(name='recall'), F1Score()])

start_time = time.time()
resnet_history = resnet_model.fit(
    train_generator,
    validation_data=valid_generator,
    # na 7 malo bylo
    epochs=9,
    verbose=1
)
resnet_time = time.time() - start_time
```

Rys. 6.2.1. Fragment kodu implementujący metodę transfer learningu opartą o model ResNet

Model ResNet50 również zaimplementowano z wykorzystaniem *TensorFlow Keras*. Bazowy model ResNet50 został wczytany z wagami imagenet, z pominięciem oryginalnych warstw końcowych (*include_top = False*) i zastosowaniem uśredniania cech na końcu (*pooling = 'avg'*). Nadbudowa modelu obejmuje następujące warstwy:

- BatchNormalization - stabilizacja aktywacji i przyspieszenie konwergencji,
- Dense(256, activation = 'relu') - warstwa gęsta ucząca nieliniowych reprezentacji,
- Dropout(0.5) - losowe zerowanie połączeń w celu ograniczenia nadmiernego dopasowania,
- Dense(38, activation = 'softmax') - warstwa wyjściowa dostosowana do liczby klas.

Model skompilowano przy użyciu optymalizatora *Adam* z obniżonym współczynnikiem uczenia 0.0005, oraz z metrykami accuracy, precision, recall i F1Score. Trening prowadzono przez 9 epok na danych wejściowych pochodzących z *train_generator* i *valid_generator*.

6.3. Konwolucyjne sieci neuronowe (CNN)

Trzecim modelem głębokiego uczenia w badaniu jest własnoręcznie zaprojektowana konwolucyjna sieć neuronowa (CNN), zbudowana i trenowana od podstaw^[32]. Celem włączenia tego modelu było stworzenie punktu odniesienia dla metod transfer learningu, w szczególności sprawdzenie, czy model stworzony specjalnie dla konkretnego zbioru danych może konkurować z dużymi, wstępnie wytrenowanymi architekturami. Zaletą takiego podejścia jest pełna kontrola nad strukturą sieci i możliwość optymalizacji pod kątem konkretnego zastosowania. Analiza tego modelu pozwala także na lepsze zrozumienie wpływu poszczególnych warstw i parametrów na skuteczność klasyfikacji.

Klasyczne CNN to modele głębokiego uczenia zaprojektowane do analizy danych obrazowych. Kluczowym elementem są warstwy konwolucyjne, które uczą się hierarchii cech, a także pooling, który zmniejsza wymiarowość danych. W odróżnieniu od transfer learningu, model ten trenowany jest od podstaw, co pozwala na lepsze dopasowanie do konkretnego zadania, ale wymaga większej ilości danych. Poniżej przedstawiono fragment kodu (rys. 6.3.1.) implementujący własną sieć CNN:

```
[ ] # Model 3: Custom CNN
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(38, activation='softmax')
])

cnn_model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy', Precision(name='precision'), Recall(name='recall'), F1Score()])

start_time = time.time()
cnn_history = cnn_model.fit(
    train_generator,
    validation_data=valid_generator,
    # na 7 dalej rosło
    epochs=9,
    verbose=1
)
cnn_time = time.time() - start_time
```

Rys. 6.3.1. Fragment kodu implementujący własną sieć CNN.

Własna sieć CNN została zbudowana od podstaw przy użyciu *tf.keras.Sequential*. Jako dane wejściowe przyjmowano obrazy o rozdzielczości $224 \times 224 \times 3$. Architektura sieci składała się z następujących warstw:

- Conv2D(32, (3, 3), activation = 'relu') - konwolucja z 32 filtrami, wykrywająca lokalne cechy obrazu,
- MaxPooling2D(pool_size = (2, 2)) - zmniejszenie rozmiaru przestrzennego cech,
- Conv2D(64, (3, 3), activation='relu') - kolejna warstwa konwolucyjna z większą liczbą filtrów,
- MaxPooling2D(pool_size = (2, 2)) - dalsza redukcja przestrzenna,
- Flatten() – przekształcenie danych do postaci jednowymiarowego wektora,
- Dense(128, activation = 'relu') - w pełni połączona warstwa ukrywająca,
- Dropout(0.5) - regularizacja przez losowe wyłączanie neuronów,
- Dense(38, activation = 'softmax') - warstwa wyjściowa do klasyfikacji 38 klas.

Model skompilowano z optymalizatorem *Adam*, funkcją straty *categorical_crossentropy* oraz metrykami accuracy, precision, recall i F1Score. Trening odbywał się przez 9 epok na generatorze train_generator.

6.4. Klasyfikator KNN

Model KNN (k-najbliższych sąsiadów) został wybrany jako jeden z klasycznych algorytmów uczenia maszynowego do porównania z podejściami głębokiego uczenia. Mimo swojej prostoty, KNN często osiąga zaskakująco dobre wyniki w zadaniach klasyfikacyjnych, szczególnie przy odpowiednim przygotowaniu danych wejściowych^[33]. W kontekście diagnozowania objawów niedoborów i chorób roślin na podstawie obrazów, model ten pozwala ocenić, na ile skuteczne mogą być techniki oparte wyłącznie na podobieństwie cech pikselowych bez złożonego modelowania struktury obrazu.

KNN to klasyczny, leniwy algorytm klasyfikacji, który przypisuje etykietę na podstawie większości głosów k najbliższych sąsiadów w przestrzeni cech. Nie wymaga procesu trenowania, ale jest kosztowny obliczeniowo przy dużych zbiorach. Poniżej przedstawiono fragment kodu(rys. 6.4.1.) implementujący klasyfikator KNN:

```

# Model 4: KNN

# Zakodowanie etykiet
encoder = LabelEncoder()
y_train_enc = encoder.fit_transform(y_train)
y_valid_enc = encoder.transform(y_valid)

# Katalog zapisu
save_dir = "/content/drive/MyDrive/praca_magisterska/wyniki/KNN/"
os.makedirs(save_dir, exist_ok=True)

# Lista wartości n_neighbors do przetestowania
neighbor_values = [1, 3, 5, 7, 9, 12]

# Słownik do przechowywania metryk
all_metrics = {
    "k": [],
    "accuracy": [],
    "precision": [],
    "recall": [],
    "f1_score": [],
    "training_time (s)": []
}

# Pętla testująca różne wartości k
for k in neighbor_values:
    knn = KNeighborsClassifier(n_neighbors=k)

    start_time = time.time()
    knn.fit(X_train, y_train_enc)
    training_time = time.time() - start_time

```

Rys. 6.4.1. Fragment kodu implementujący klasyfikator KNN.

Model KNN został zaimplementowany przy pomocy klasy *KNeighborsClassifier* z biblioteki *scikit-learn*. Obrazy wejściowe zostały wcześniej przekształcone do wektorów cech o długości $224 \times 224 \times 3$ i zakodowane etykietami numerycznymi przy użyciu *LabelEncoder*. Model nie posiada warstw typowych dla sieci neuronowych – jego działanie opiera się na obliczaniu odległości między próbkami. Przetestowano kilka wartości parametru *n_neighbors*, w tym: 1, 3, 5, 7, 9 i 12. Dla każdej wartości model był trenowany (przez *fit()*), a czas treningu oraz metryki (*accuracy*, *precision*, *recall*, *f1_score*) zapisywano osobno.

6.5. Maszyna wektorów nośnych (SVM)

SVM to klasyczny, lecz bardzo silny algorytm klasyfikacyjny, który znajduje zastosowanie szczególnie w zadaniach, gdzie dostępne są dane wysokowymiarowe, ale w niewielkiej liczbie próbek^[34]. Z uwagi na swoją zdolność do tworzenia optymalnych

hiperplanów oddzielających klasy, SVM może stanowić efektywną alternatywę dla bardziej złożonych modeli głębokiego uczenia. SVM (Support Vector Machine) to nadzorowany klasyfikator, który próbuje znaleźć hiperplan maksymalnie oddzielający klasy w przestrzeni cech. Jest skuteczny przy danych o wysokim wymiarze i dobrze radzi sobie nawet przy niewielkiej liczbie próbek. Poniżej przedstawiono fragment kodu (rys. 6.5.1.) implementujący SVM:

```
[ ] # Model 5: SVM

# Zakodowanie etykiet
encoder = LabelEncoder()
y_train_enc = encoder.fit_transform(y_train)
y_valid_enc = encoder.transform(y_valid)

# SVM
svm = SVC(kernel='rbf')
start_time = time.time()
svm.fit(X_train, y_train_enc)
svm_time = time.time() - start_time

y_pred_svm = svm.predict(X_valid)

svm_metrics = {
    "accuracy": accuracy_score(y_valid_enc, y_pred_svm),
    "precision": precision_score(y_valid_enc, y_pred_svm, average='macro'),
    "recall": recall_score(y_valid_enc, y_pred_svm, average='macro'),
    "f1_score": f1_score(y_valid_enc, y_pred_svm, average='macro'),
    "training_time (s)": svm_time
}
```

Rys. 6.5.1. Fragment kodu implementujący SVM.

Model SVM (Support Vector Machine) został zaimplementowany przy użyciu klasy *SVC* z biblioteki *scikit-learn*. Dane wejściowe (obrazy) zostały spłaszczone do jednowymiarowych wektorów o długości 150 528 ($224 \times 224 \times 3$), a etykiety zakodowano numerycznie za pomocą *LabelEncoder*. W implementacji zastosowano jądro radialne (RBF), które umożliwia tworzenie nieliniowych granic decyzyjnych w przestrzeni cech^[35]. Wybrano domyślne parametry jądra, a sam model został wytrenowany z użyciem metody *fit()* na zbiorze treningowym.

W przeciwieństwie do modelu KNN, gdzie testowano różne wartości parametru *k*, w przypadku SVM nie przeprowadzano pętli testowej z różnymi hiperparametrami. Wynika to z faktu, że jądro RBF posiada wbudowaną zdolność dopasowania się do struktury danych - automatycznie kształtuje granicę decyzyjną, opartą na rozkładzie obserwacji w przestrzeni cech. Po zakończeniu treningu wykonano predykcję na zbiorze walidacyjnym, a następnie obliczono metryki skuteczności: *accuracy*, *precision*, *recall* i *f1_score*, wykorzystując funkcje z pakietu *sklearn.metrics*. Czas treningu modelu został również zmierzony i zapisany jako jedna z metryk porównawczych.

6.6. Algorytm Random Forest

Random Forest to przedstawiciel tzw. algorytmów zespołowych (ensemble learning), które łączą decyzje wielu prostych modeli (drzew decyzyjnych), aby zwiększyć ogólną skuteczność klasyfikacji^[36]. Wybór tego modelu do porównania z CNN i metodami transfer learningu wynika z jego odporności na przeuczenie oraz skuteczności w pracy z nieprzekształconymi cechami. Model ten dobrze radzi sobie w zadaniach o dużej liczbie cech wejściowych^[37], co czyni go atrakcyjnym kandydatem do diagnozy na podstawie spłaszczonej danych obrazowych. Jego działanie opiera się na mechanizmie głosowania wielu niezależnych klasyfikatorów, co wpływa pozytywnie na stabilność wyników. Random Forest to zespół drzew decyzyjnych, które tworzą głosy dla ostatecznej klasyfikacji. Każde drzewo jest trenowane na losowym podzbiorze danych i cech, co poprawia odporność na przeuczenie. Model dobrze radzi sobie z danymi heterogenicznymi i jest intuicyjny w interpretacji. Poniżej przedstawiono fragment kodu(rys. 6.6.1.) implementujący algorytm Random Forest:


```

# Model 6: Random Forest

# Zakodowanie etykiet
encoder = LabelEncoder()
y_train_enc = encoder.fit_transform(y_train)
y_valid_enc = encoder.transform(y_valid)

# Katalog zapisu
save_dir = "/content/drive/MyDrive/praca_magisterska/wyniki/RandomForest/"
os.makedirs(save_dir, exist_ok=True)

# Lista wartości n_estimators do przetestowania
n_estimators_values = [25, 50, 100, 200, 300, 400, 500]

# Słownik do przechowywania metryk
all_metrics = {
    "n": [],
    "accuracy": [],
    "precision": [],
    "recall": [],
    "f1_score": [],
    "training_time (s)": []
}

# Pętla testująca różne wartości n_estimators
for n in n_estimators_values:
    rf_model = RandomForestClassifier(n_estimators=n, random_state=42)

    start_time = time.time()
    rf_model.fit(X_train, y_train_enc)
    training_time = time.time() - start_time

```

Rys. 6.6.1. Fragment kodu implementujący algorytm Random Forest.

Model Random Forest stworzono z użyciem *RandomForestClassifier* z biblioteki *scikit-learn*. Dane wejściowe były znormalizowane i spłaszczone, a etykiety zakodowane numerycznie. Testowano różne liczby drzew decyzyjnych: 25, 50, 100, 200, 300, 400 i 500 (*n_estimators*), co miało na celu ocenę wpływu złożoności modelu na jego skuteczność i czas trenowania. Dla każdej wartości *n_estimators* trenowano model przy użyciu *fit()*, mierząc czas treningu. Po zakończeniu obliczano metryki klasyfikacyjne (accuracy, precision, recall, f1_score) i zapisywano je w odpowiednich słownikach wyników.

7. Eksperymenty i wyniki

W poniższym rozdziale opisano przebieg eksperymentów mających na celu porównanie skuteczności wybranych modeli uczenia maszynowego i głębokiego w kontekście klasyfikacji obrazów liści roślin. Celem badań była ocena, na ile poszczególne modele potrafią prawidłowo identyfikować objawy chorób oraz niedoborów składników pokarmowych na podstawie danych obrazowych. Eksperymenty przeprowadzono w środowisku Google Colab, z wykorzystaniem odpowiednio przygotowanego i przetworzonego zbioru zdjęć.

Dla każdego modelu obliczono zestaw standardowych metryk klasyfikacyjnych, które następnie posłużyły do analizy porównawczej pod względem skuteczności oraz kosztów obliczeniowych. W badaniach uwzględniono zarówno modele głębokiego uczenia, takie jak Xception, ResNet50 oraz własna architektura CNN, jak i klasyczne algorytmy uczenia maszynowego - KNN, SVM, a także Random Forest.

7.1. Metryki ewaluacyjne i sposób oceny modeli

Do oceny skuteczności działania modeli wykorzystano cztery popularne metryki klasyfikacyjne: accuracy, precision, recall oraz F1-score^[38]. Metryki te pozwalają lepiej zrozumieć, jak dobrze model rozpoznaje poszczególne klasy^[39], szczególnie w zadaniu wieloklasowej klasyfikacji obrazów^[40]. Ich wartość obliczana jest na podstawie czterech możliwych wyników klasyfikacji, które opisano następującymi symbolami:

- **TP (True Positive)** - liczba przykładów poprawnie sklasyfikowanych jako należące do danej klasy (prawdziwe pozytywne),
- **TN (True Negative)** - liczba przykładów poprawnie sklasyfikowanych jako nienależące do danej klasy (prawdziwe negatywne),
- **FP (False Positive)** - liczba przykładów błędnie zaklasyfikowanych jako należące do danej klasy (fałszywe pozytywne),
- **FN (False Negative)** - liczba przykładów błędnie zaklasyfikowanych jako nienależące do danej klasy (fałszywe negatywne).

Na tej podstawie obliczane są następujące metryki:

- **Accuracy (dokładność)** - to najprostsza miara skuteczności, mówi, jaki procent wszystkich przykładów model sklasyfikował poprawnie. Bierze pod uwagę zarówno poprawne przewidywania pozytywne (TP), jak i negatywne (TN).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Jest to dobra metryka, gdy klasy są zrównoważone, ale może być myląca, gdy niektóre klasy występują dużo częściej niż inne.

- **Precision (precyzja)** - mówi, jak wiele z przewidywań danej klasy było rzeczywiście trafnych. Odpowiada na pytanie: „Jeśli model przewidział daną klasę, to jak często miał rację?”.

$$Precision = \frac{TP}{TP + FP}$$

Wysoka precyzja oznacza, że model rzadko zgłasza fałszywe rozpoznania (fałszywe alarmy).

- **Recall (czułość)** - metryka ta pokazuje, ile przypadków z danej klasy model zdołał poprawnie wykryć. Odpowiada na pytanie: „Ile z rzeczywistych przypadków danej klasy model rozpoznał?”.

$$Recall = \frac{TP}{TP + FN}$$

Wysoka czułość oznacza, że model rzadko przeocza prawdziwe przypadki.

- **F1-score** - to metryka, która łączy precision i recall w jedną wartość, stanowiąc ich średnią harmoniczną. Jest użyteczna, gdy ważna jest równowaga między precyzją a czułością, np. przy diagnozowaniu objawów, gdzie zarówno fałszywe alarmy, jak i przeoczenia są niepożądane.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Modele głębokiego uczenia (Xception, ResNet50, CNN) były trenowane przy użyciu generatorów danych (*train_generator* i *valid_generator*), a następnie ewaluowane przy użyciu metody *evaluate()* z biblioteki *TensorFlow*. Funkcja ta zwracała wartości zdefiniowanych wcześniej metryk: *accuracy*, *precision*, *recall* oraz *f1_score*. Dodatkowo dla tych modeli zapisywano historię procesu uczenia, która zawierała informacje o zmianach metryk w kolejnych epokach. Dane te umożliwiły późniejsze wygenerowanie wykresów oraz analizę modeli.

Dla modeli klasycznych (KNN i Random Forest) zastosowano pętle testowe, w których dla różnych wartości parametrów (*n_neighbors* w KNN oraz *n_estimators* w Random Forest) trenowano modele i zapisywano ich wyniki. Dla każdego wariantu modelu mierzono czas treningu oraz obliczano metryki przy użyciu funkcji *accuracy_score*, *precision_score*, *recall_score* oraz *f1_score* z biblioteki *sklearn.metrics*.

Wyniki ewaluacji oraz czasy trenowania zostały automatycznie zapisywane w formie plików .json lub obrazów .png do dedykowanych katalogów na Dysku Google. Dla każdego modelu utworzono osobny katalog wynikowy, co ułatwiło późniejszą analizę porównawczą oraz organizację danych.

W przypadku modelu SVM, z uwagi na zastosowanie jądra radialnego (RBF), nie przeprowadzano pętli testowej z różnymi konfiguracjami hiperparametrów. Jądro RBF automatycznie dopasowuje się do rozkładu danych, co pozwala modelowi skutecznie kształtować granicę decyzyjną bez potrzeby eksplorowania wielu wariantów. W przeciwieństwie do np. KNN, w którym istotne jest dobranie optymalnej liczby sąsiadów, tutaj priorytetem było zastosowanie elastycznego jądra umożliwiającego nieliniową separację klas.

7.2. Wyniki badań

W wyniku przeprowadzonych eksperymentów uzyskano konkretne wartości metryk skuteczności danych modeli. Poniżej przedstawiono szczegółowe rezultaty dla każdego z sześciu analizowanych modeli, wraz z opisem przebiegu procesu uczenia oraz osiągniętych wyników na zbiorze testowym. Dla każdego modelu oceniano skuteczność na podstawie czterech kluczowych metryk: accuracy, precision, recall oraz f1_score.

• Xception

Model Xception trenowano z ustawionym parametrem *verbose=1*, co pozwoliło uzyskać pełny raport po każdej epoce (rys. 7.2.1.). Log zawierał wartości metryk treningowych i walidacyjnych: accuracy, f1_score, precision, recall, loss, a także czas przetwarzania. Dzięki temu możliwe było dokładne śledzenie postępów modelu w trakcie uczenia.

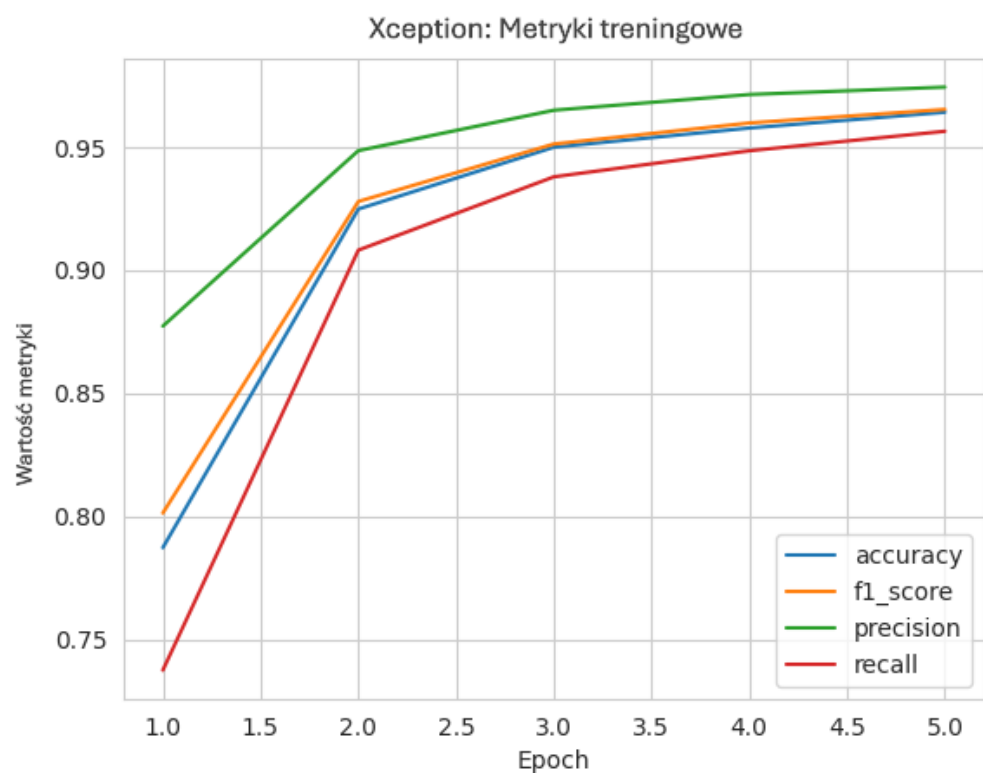
```

Download data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 5s 0us/step
Epoch 1/5
18861/18861 1157s 102ms/step - accuracy: 0.6491 - f1_score: 0.6632 - loss: 1.4327 - precision: 0.7868 - recall: 0.5782 - val_accuracy: 0.9591 - val_f1_score: 0.9593 - val_loss: 0.3328 - val_precision: 0.9683 - val_recall: 0.9582
Epoch 2/5
18861/18861 959s 88ms/step - accuracy: 0.9169 - f1_score: 0.9189 - loss: 0.3186 - precision: 0.9413 - recall: 0.8975 - val_accuracy: 0.9755 - val_f1_score: 0.9757 - val_loss: 0.1693 - val_precision: 0.9762 - val_recall: 0.9751
Epoch 3/5
18861/18861 943s 87ms/step - accuracy: 0.9475 - f1_score: 0.9483 - loss: 0.2118 - precision: 0.9636 - recall: 0.9335 - val_accuracy: 0.9788 - val_f1_score: 0.9783 - val_loss: 0.1795 - val_precision: 0.9790 - val_recall: 0.9777
Epoch 4/5
18861/18861 930s 86ms/step - accuracy: 0.9551 - f1_score: 0.9567 - loss: 0.1784 - precision: 0.9691 - recall: 0.9445 - val_accuracy: 0.9781 - val_f1_score: 0.9788 - val_loss: 0.4326 - val_precision: 0.9781 - val_recall: 0.9779
Epoch 5/5
18861/18861 914s 84ms/step - accuracy: 0.9634 - f1_score: 0.9646 - loss: 0.1439 - precision: 0.9735 - recall: 0.9558 - val_accuracy: 0.9691 - val_f1_score: 0.9698 - val_loss: 0.1846 - val_precision: 0.9694 - val_recall: 0.9687

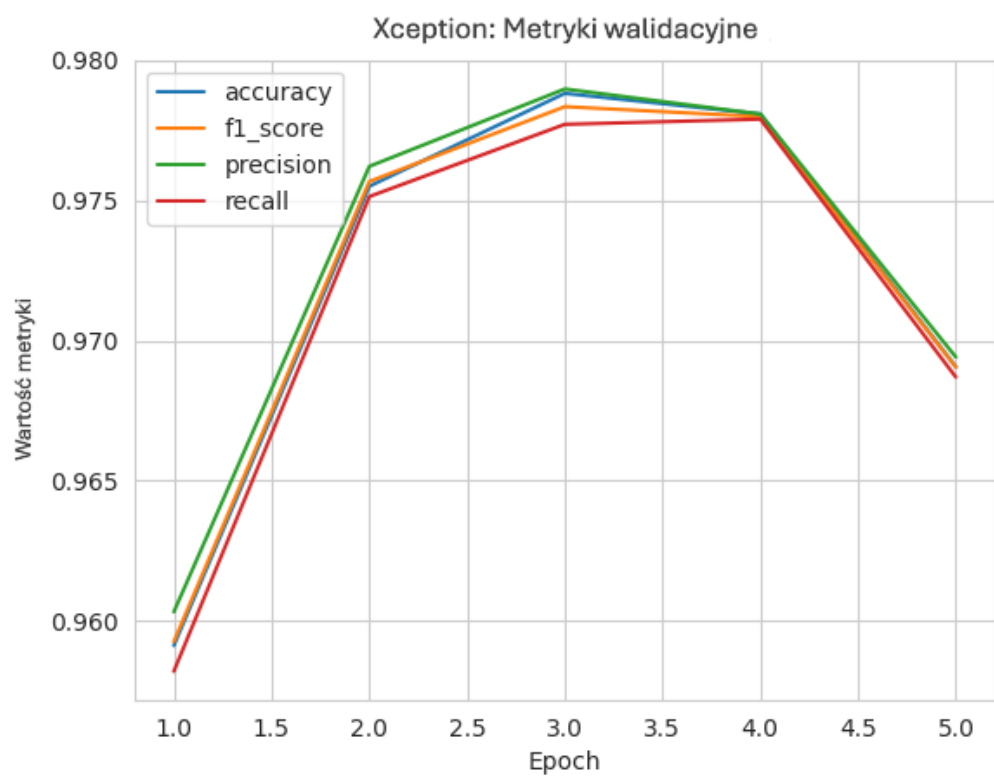
```

Rys. 7.2.1. Raport z trenowania modelu Xception.

Poniżej zamieszczono wykresy przedstawiające, jak zmieniały się wartości metryk w kolejnych epokach, osobno dla danych treningowych (rys. 7.2.2.) i walidacyjnych (rys. 7.2.3.):



Rys. 7.2.2. Wykres metryk treningowych dla modelu Xception.



Rys. 7.2.3. Wykres metryk walidacyjnych dla modelu Xception.

Analiza wykresu walidacyjnego pokazuje, że model osiągnął najwyższe wartości metryk po trzeciej epoce treningu. Dodanie kolejnych epok skutkowało przetrenowaniem i coraz to mocniejszym spadkiem skuteczności.

- **ResNet50**

Raport dla modelu ResNet50 trenowanego przez 9 epok(rys. 7.2.4):

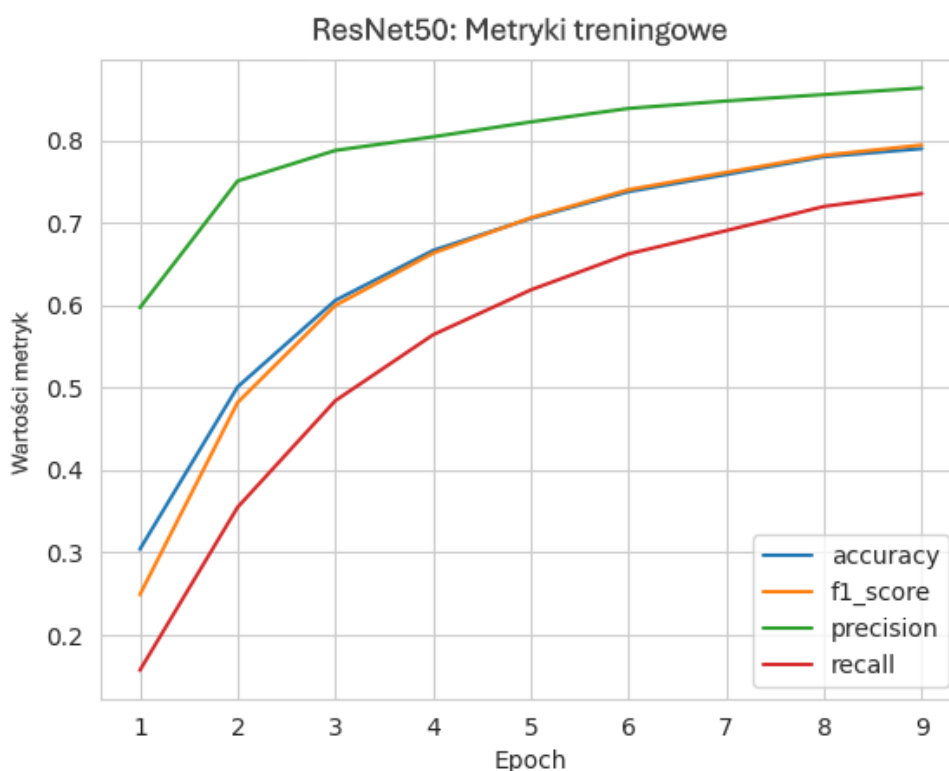
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 3s 0us/step
Epoch 1/9
10861/10861 ----- 1101s 95ms/step - accuracy: 0.2115 - f1_score: 0.1503 - loss: 3.5117 - precision: 0.4027 - recall: 0.0930 - val_accuracy: 0.5223 - val_f1_score: 0.5118 - val_loss: 3.3414 - val_precision: 0.6083 - val_recall: 0.4418
Epoch 2/9
10861/10861 ----- 1066s 94ms/step - accuracy: 0.4629 - f1_score: 0.4355 - loss: 1.9345 - precision: 0.7279 - recall: 0.3112 - val_accuracy: 0.4816 - val_f1_score: 0.4781 - val_loss: 16.1799 - val_precision: 0.5094 - val_recall: 0.4585
Epoch 3/9
10861/10861 ----- 1018s 94ms/step - accuracy: 0.5830 - f1_score: 0.5719 - loss: 1.4442 - precision: 0.7783 - recall: 0.4523 - val_accuracy: 0.6473 - val_f1_score: 0.6515 - val_loss: 2.4606 - val_precision: 0.6788 - val_recall: 0.6263
Epoch 4/9
10861/10861 ----- 1013s 93ms/step - accuracy: 0.6568 - f1_score: 0.6523 - loss: 1.1582 - precision: 0.8026 - recall: 0.5495 - val_accuracy: 0.7587 - val_f1_score: 0.7573 - val_loss: 1.8861 - val_precision: 0.7928 - val_recall: 0.7256
Epoch 5/9
10861/10861 ----- 1015s 93ms/step - accuracy: 0.6967 - f1_score: 0.6979 - loss: 1.0043 - precision: 0.8191 - recall: 0.6081 - val_accuracy: 0.7858 - val_f1_score: 0.7880 - val_loss: 1.4964 - val_precision: 0.7962 - val_recall: 0.7799
Epoch 6/9
10861/10861 ----- 1007s 93ms/step - accuracy: 0.7268 - f1_score: 0.7282 - loss: 0.9327 - precision: 0.8333 - recall: 0.6468 - val_accuracy: 0.8287 - val_f1_score: 0.8293 - val_loss: 1.2693 - val_precision: 0.8356 - val_recall: 0.8230
Epoch 7/9
10861/10861 ----- 1017s 94ms/step - accuracy: 0.7517 - f1_score: 0.7545 - loss: 0.8291 - precision: 0.8439 - recall: 0.6823 - val_accuracy: 0.8013 - val_f1_score: 0.8018 - val_loss: 26.8467 - val_precision: 0.8028 - val_recall: 0.8009
Epoch 8/9
10861/10861 ----- 1054s 95ms/step - accuracy: 0.7752 - f1_score: 0.7763 - loss: 0.7581 - precision: 0.8506 - recall: 0.7139 - val_accuracy: 0.8703 - val_f1_score: 0.8716 - val_loss: 1.0133 - val_precision: 0.8777 - val_recall: 0.8656
Epoch 9/9
10861/10861 ----- 1043s 96ms/step - accuracy: 0.7858 - f1_score: 0.7905 - loss: 0.7387 - precision: 0.8624 - recall: 0.7298 - val_accuracy: 0.8861 - val_f1_score: 0.8864 - val_loss: 1.7271 - val_precision: 0.8328 - val_recall: 0.8089

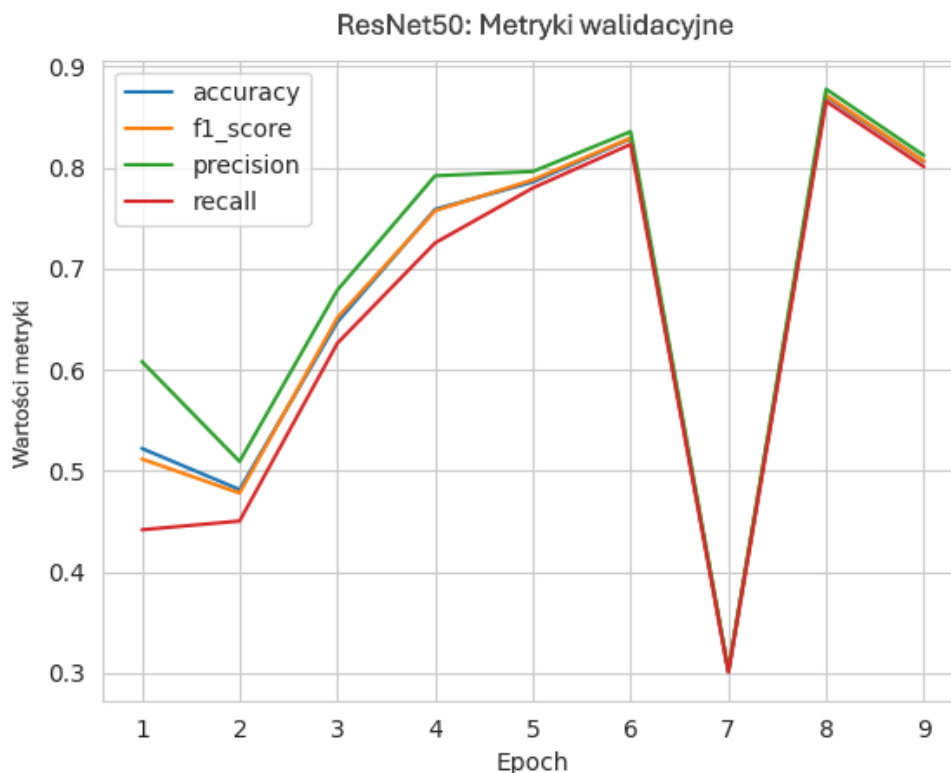
```

Rys. 7.2.4. Raport z trenowania modelu ResNet50.

Wykresy przedstawiające, jak zmieniały się wartości metryk w kolejnych epokach, osobno dla danych treningowych(rys. 7.2.5.) i walidacyjnych(rys. 7.2.6.):



Rys. 7.2.5 Wykres metryk treningowych dla modelu ResNet50.



Rys. 7.2.6 Wykres metryk walidacyjnych dla modelu ResNet50.

Model ucząc się zwiększał swoją skuteczność aż do sześciu epok, po których nastąpiło przetrenowanie i gwałtowny spadek wartości metryk widoczny na wykresie walidacyjnym. W kolejnych epokach model osiągnął nowy najwyższy wynik metryk (dla $epoch = 8$), a następnie jego skuteczność ponownie zaczęła spadać. Model ten wykazuje niestabilność trenowany przez większą liczbę epok.

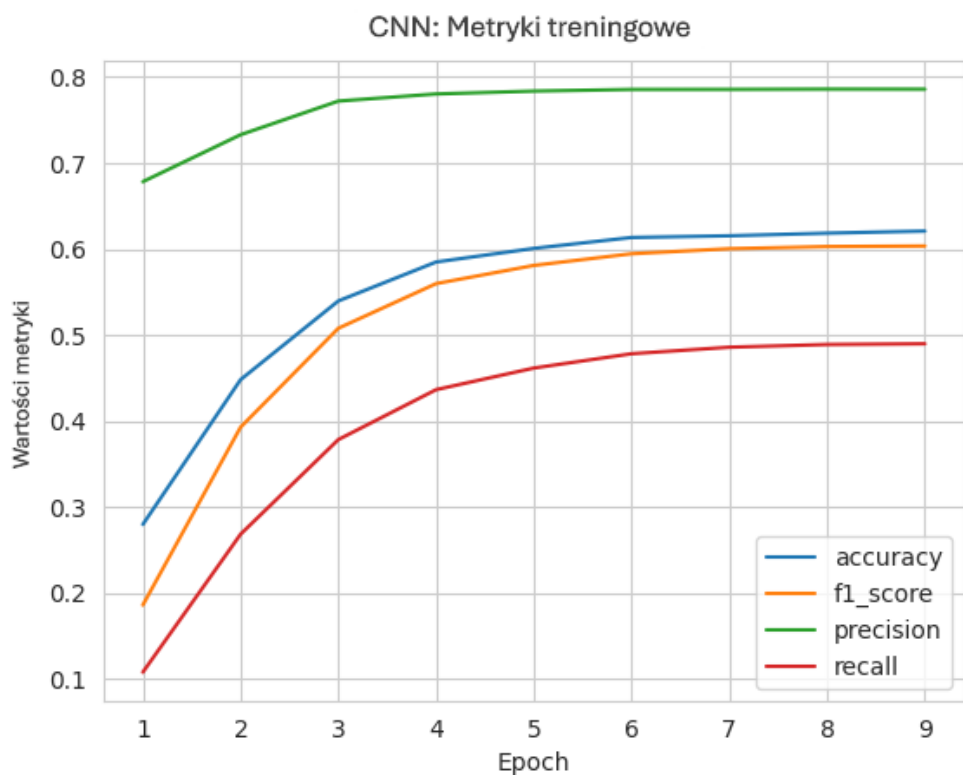
- CNN

Raport dla własnego modelu CNN trenowanego przez 9 epok(rys. 7.2.7):

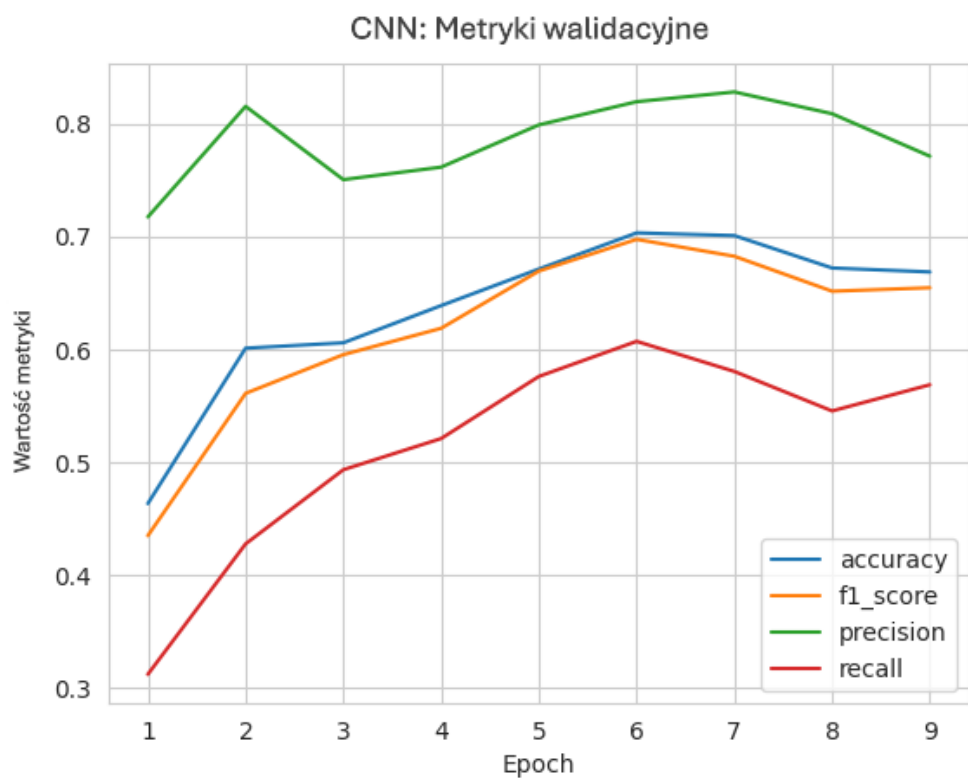
Epoch 1/9	879s 80ms/step	accuracy: 0.2042	f1_score: 0.1088	loss: 3.0228	precision: 0.6237	recall: 0.0559	val_accuracy: 0.4635	val_f1_score: 0.4351	val_loss: 1.8382	val_precision: 0.7176	val_recall: 0.3122
Epoch 2/9	626s 58ms/step	accuracy: 0.4157	f1_score: 0.3551	loss: 2.0416	precision: 0.7143	recall: 0.2364	val_accuracy: 0.6813	val_f1_score: 0.5612	val_loss: 1.4172	val_precision: 0.8157	val_recall: 0.4278
Epoch 3/9	615s 57ms/step	accuracy: 0.5263	f1_score: 0.4924	loss: 1.6517	precision: 0.7698	recall: 0.3621	val_accuracy: 0.6861	val_f1_score: 0.5956	val_loss: 1.4252	val_precision: 0.7567	val_recall: 0.4936
Epoch 4/9	634s 58ms/step	accuracy: 0.5889	f1_score: 0.5513	loss: 1.4748	precision: 0.7781	recall: 0.4266	val_accuracy: 0.6390	val_f1_score: 0.6189	val_loss: 1.2781	val_precision: 0.7618	val_recall: 0.5212
Epoch 5/9	637s 59ms/step	accuracy: 0.5970	f1_score: 0.5778	loss: 1.4134	precision: 0.7831	recall: 0.4578	val_accuracy: 0.6715	val_f1_score: 0.6697	val_loss: 1.1944	val_precision: 0.7992	val_recall: 0.5762
Epoch 6/9	621s 57ms/step	accuracy: 0.6140	f1_score: 0.5952	loss: 1.3535	precision: 0.7877	recall: 0.4783	val_accuracy: 0.7035	val_f1_score: 0.6978	val_loss: 1.0575	val_precision: 0.8198	val_recall: 0.6074
Epoch 7/9	618s 57ms/step	accuracy: 0.6176	f1_score: 0.6020	loss: 1.3444	precision: 0.7884	recall: 0.4869	val_accuracy: 0.7011	val_f1_score: 0.6828	val_loss: 1.1000	val_precision: 0.8284	val_recall: 0.5887
Epoch 8/9	624s 57ms/step	accuracy: 0.6181	f1_score: 0.6018	loss: 1.3369	precision: 0.7878	recall: 0.4869	val_accuracy: 0.6724	val_f1_score: 0.6518	val_loss: 1.1940	val_precision: 0.8091	val_recall: 0.5457
Epoch 9/9	630s 58ms/step	accuracy: 0.6214	f1_score: 0.6056	loss: 1.3305	precision: 0.7880	recall: 0.4918	val_accuracy: 0.6689	val_f1_score: 0.6549	val_loss: 1.3020	val_precision: 0.7715	val_recall: 0.5689

Rys. 7.2.7. Raport z trenowania własnego modelu CNN.

Wykresy przedstawiające, w jakiś sposób zmieniały się wartości metryk w kolejnych epokach, osobno dla danych treningowych(rys. 7.2.8.) i walidacyjnych(rys. 7.2.9.):



Rys. 7.2.8 Wykres metryk treningowych dla własnego modelu CNN.




Rys. 7.2.9 Wykres metryk walidacyjnych dla własnego modelu CNN.

Model zwiększał swoją skuteczność aż do szóstej epoki po której wystąpiło przetrenowanie i spadek wartości metryk. Metryki (accuracy, f1_score oraz recall) osiągają najwyższe wartości po sześciu epokach trenowania. Mimo, że w kolejnej epoce rośnie lekko wartość precision to pozostałe metryki ulegają pogorszeniu, co może wpływać na pogorszenie ogólnej skuteczności modelu. Do dalszych porównań użyty został model CNN trenowany na *epoch* = 6.

- **KNN**

Model KNN nie posiada historii treningu w epokach, jak modele głębokiego uczenia. Zamiast tego został przetestowany w pętli dla różnych wartości parametru *k* - [1, 3, 5, 7, 9, 12]. Dla każdej wartości *k* model został dopasowany do danych treningowych, a następnie poddany ewaluacji na zbiorze testowym. Obliczono metryki: accuracy, precision, recall i f1_score, a także zmierzono czas treningu(rys. 7.2.10.).



```
[KNN Test Set Evaluation for k = 1]
accuracy: 0.5519
precision: 0.5803
recall: 0.4307
f1_score: 0.4388
training_time (s): 0.4991

[KNN Test Set Evaluation for k = 3]
accuracy: 0.5339
precision: 0.6076
recall: 0.4064
f1_score: 0.4085
training_time (s): 0.5137

[KNN Test Set Evaluation for k = 5]
accuracy: 0.5361
precision: 0.6138
recall: 0.4048
f1_score: 0.4059
training_time (s): 0.6847

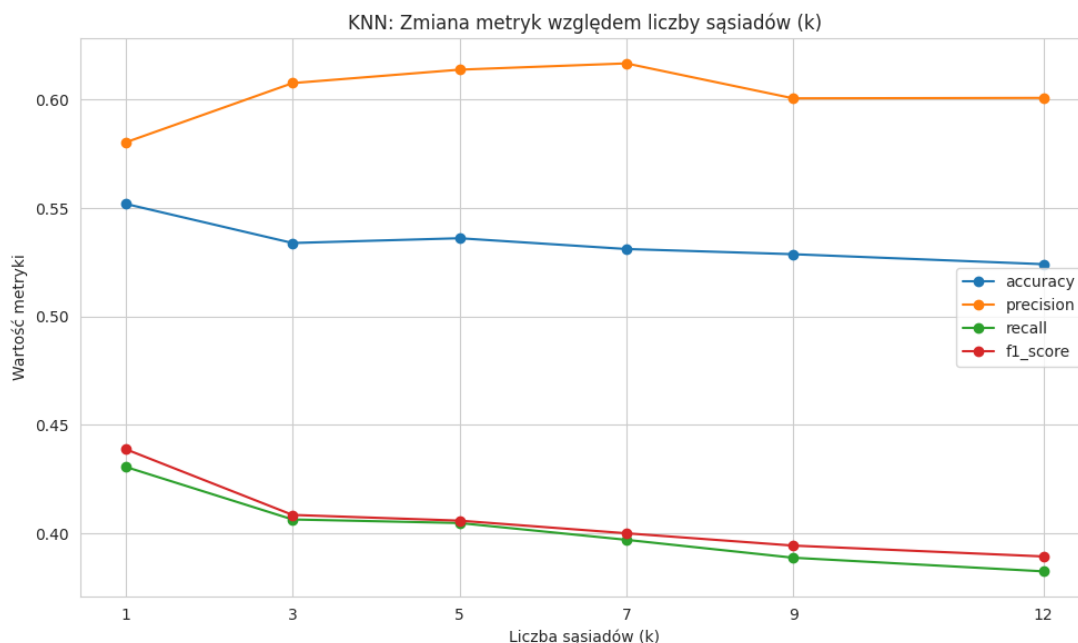
[KNN Test Set Evaluation for k = 7]
accuracy: 0.5311
precision: 0.6167
recall: 0.3970
f1_score: 0.4000
training_time (s): 0.6709

[KNN Test Set Evaluation for k = 9]
accuracy: 0.5287
precision: 0.6006
recall: 0.3888
f1_score: 0.3944
training_time (s): 0.4793

[KNN Test Set Evaluation for k = 12]
accuracy: 0.5241
precision: 0.6008
recall: 0.3825
f1_score: 0.3894
training_time (s): 0.4610
```

Rys. 7.2.10. Wyniki ewaluacji dla różnych wartości k podczas testowania modelu KNN

Wykres przedstawiający zmianę skuteczności modelu względem różnych wartości liczby sąsiadów(rys. 7.2.11.):



Rys. 7.2.11. Wykres zmiany metryk względem liczby sąsiadów dla modelu KNN.

Metryki accuracy, recall oraz f1_score osiągają maksymalny wynik przy liczbie sąsiadów równej jeden, a z każdym zwiększeniem tego parametru wartości metryk spadają. Wyjątkiem jest precision, które rośnie do siódmej epoki, po czym zaczyna spadać. Mimo niższej wartości precyzji, do dalszych porównań użyty został model o liczbie sąsiadów równej jeden, ze względu na maksymalną wartość trzech pozostałych metryk.

- **SVM**

Model SVM z jądrem radialnym (RBF) nie był trenowany w epokach ani w pętli z różnymi parametrami. Trening został przeprowadzony jednorazowo na pełnym zbiorze treningowym z domyślnymi ustawieniami. Dzięki właściwościom jądra RBF model automatycznie dopasowuje granicę decyzyjną do danych, co eliminuje potrzebę testowania wielu konfiguracji. Po treningu wykonano ewaluację na zbiorze testowym (rys. 7.2.12.), obliczając metryki: accuracy, precision, recall oraz f1_score.

```

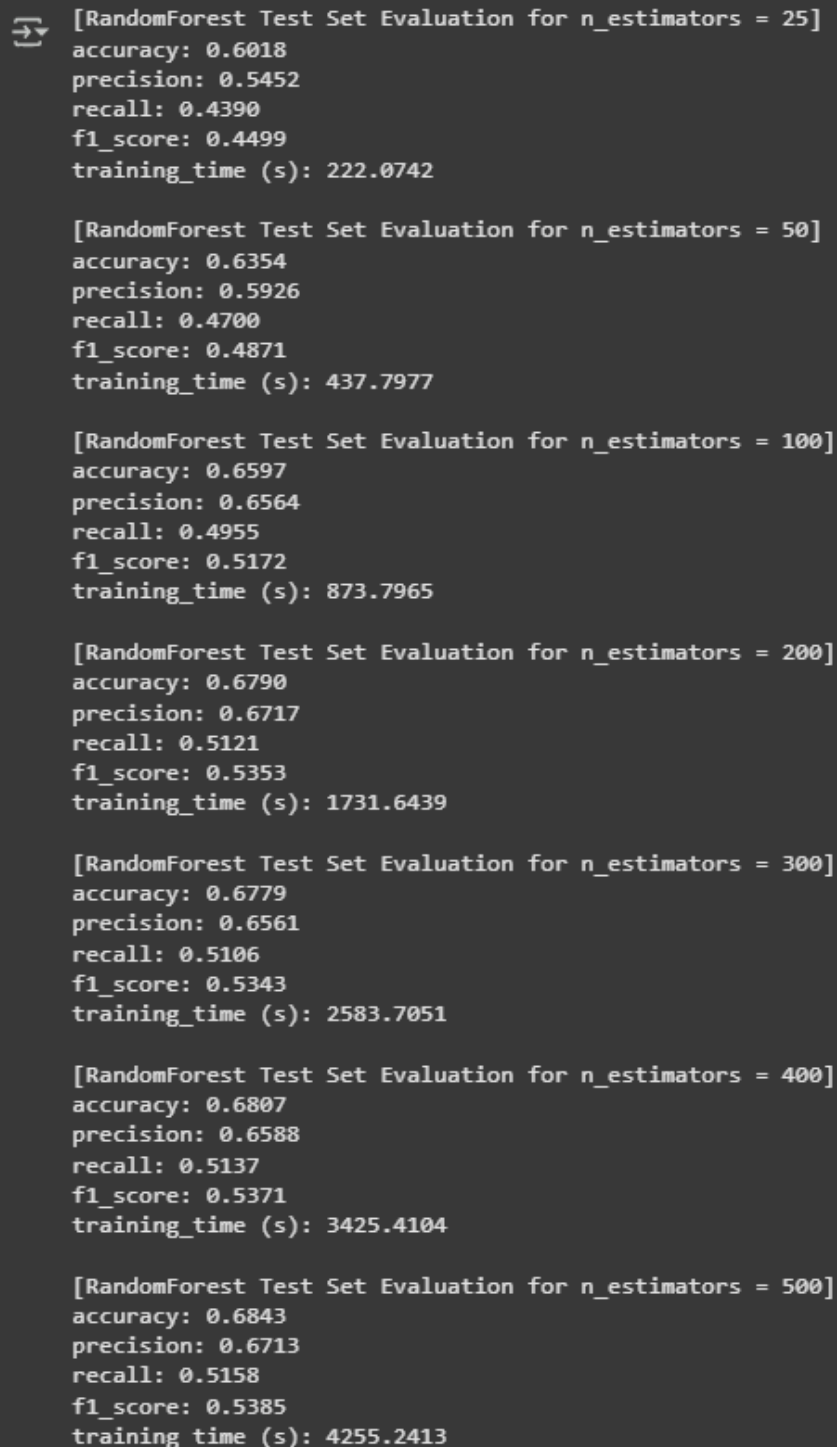
[Terminal Icon] [SVM Test set Evaluation]
accuracy: 0.7647
precision: 0.7134
recall: 0.6414
f1_score: 0.6538
training_time (s): 5810.8977

```

Rys. 7.2.12. Wynik ewaluacji dla modelu SVM.

- **RandomForest**

Model Random Forest również nie był trenowany na epokach. Zamiast tego, został przetestowany w pętli dla różnych wartości parametru *n_estimators*, czyli liczby drzew w lesie - [25, 50, 100, 200, 300, 400, 500]. Dla każdej wartości model został wytrenowany na danych treningowych i poddany ewaluacji na zbiorze testowym (rys. 7.2.13.). Obliczono metryki: accuracy, precision, recall i f1_score.



```
[RandomForest Test Set Evaluation for n_estimators = 25]
accuracy: 0.6018
precision: 0.5452
recall: 0.4390
f1_score: 0.4499
training_time (s): 222.0742

[RandomForest Test Set Evaluation for n_estimators = 50]
accuracy: 0.6354
precision: 0.5926
recall: 0.4700
f1_score: 0.4871
training_time (s): 437.7977

[RandomForest Test Set Evaluation for n_estimators = 100]
accuracy: 0.6597
precision: 0.6564
recall: 0.4955
f1_score: 0.5172
training_time (s): 873.7965

[RandomForest Test Set Evaluation for n_estimators = 200]
accuracy: 0.6790
precision: 0.6717
recall: 0.5121
f1_score: 0.5353
training_time (s): 1731.6439

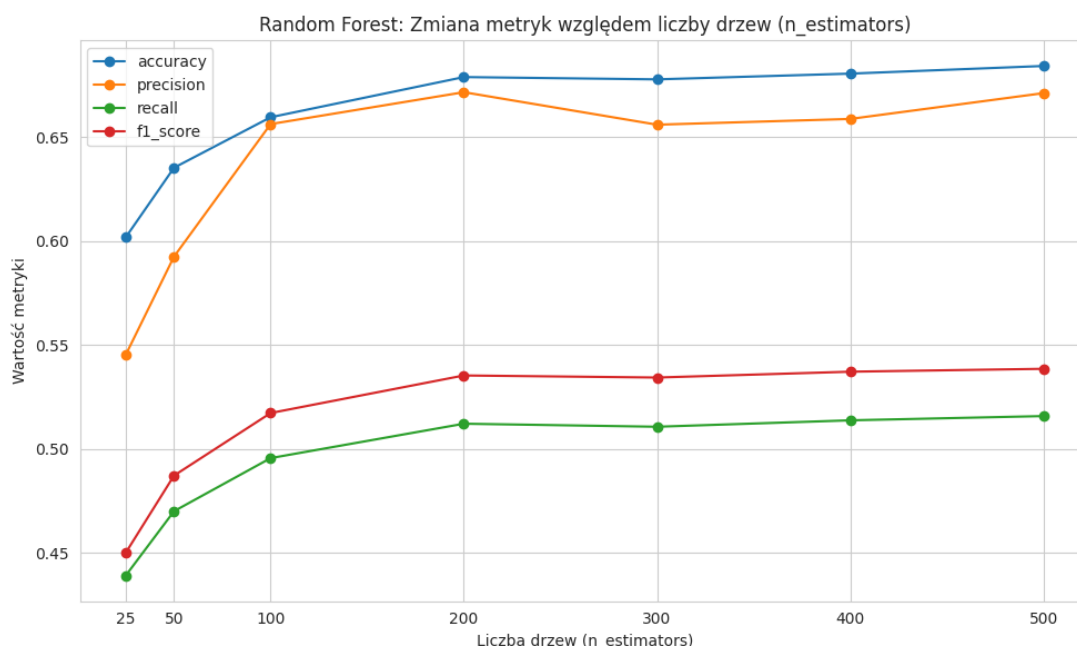
[RandomForest Test Set Evaluation for n_estimators = 300]
accuracy: 0.6779
precision: 0.6561
recall: 0.5106
f1_score: 0.5343
training_time (s): 2583.7051

[RandomForest Test Set Evaluation for n_estimators = 400]
accuracy: 0.6807
precision: 0.6588
recall: 0.5137
f1_score: 0.5371
training_time (s): 3425.4104

[RandomForest Test Set Evaluation for n_estimators = 500]
accuracy: 0.6843
precision: 0.6713
recall: 0.5158
f1_score: 0.5385
training_time (s): 4255.2413
```

Rys. 7.2.13. Wynik ewaluacji RandomForest dla różnych wartości liczby drzew w lesie.

Wykres przedstawiający zmianę skuteczności modelu względem różnych wartości liczby drzew w lesie(rys. 7.2.14.):



Rys. 7.2.14. Wykres zmiany metryk względem liczby drzew w lesie dla modelu RandomForest.

Wraz ze zwiększaniem liczby drzew, skuteczność modelu systematycznie rosła, aż wartość parametru $n_estimators$ nie osiągnęła 200 drzew. Po przekroczeniu tej wartości dochodzi do lekkiego załamania, a następnie wypłaszczenia wykresów metryk, co sugeruje osiągnięcie punktu nasycenia. Dalsze zwiększanie liczby drzew nie przynosiło już wyraźnych korzyści.

7.3. Zestawienie rezultatów

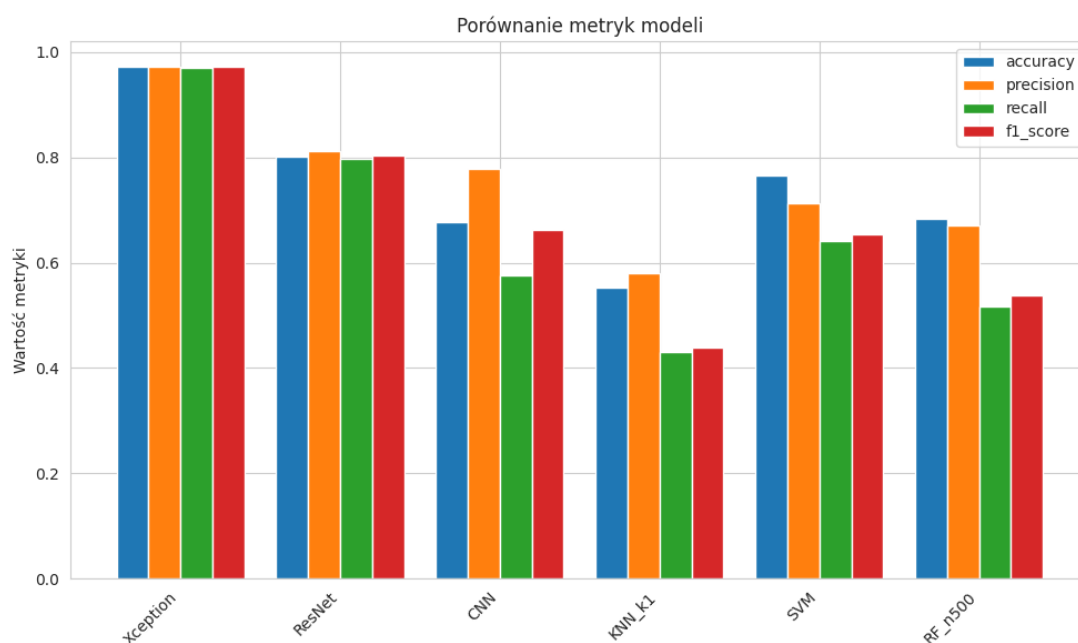
Dzięki przeprowadzonym badaniom, nad każdym z rozważanych modeli, udało się wskazać ich najlepsze warianty pod względem skuteczności klasyfikacji. Dla modeli głębokiego uczenia, takich jak Xception, ResNet50 i własna sieć CNN, optymalna liczba epok została określona na podstawie analizy wykresów metryk walidacyjnych. W przypadku modelu Xception najwyższe wartości metryk osiągnięto po trzeciej epoce treningu. Z kolei model ResNet50 uzyskał najlepsze wyniki po ośmiu epokach choć podczas treningu wykazywał się niestabilnością, natomiast własna sieć CNN - po sześciu. W przypadku klasycznych algorytmów uczenia maszynowego, również udało się wyznaczyć najbardziej efektywne konfiguracje. Dla modelu KNN najwyższą skuteczność uzyskano przy liczbie sąsiadów równej jeden. Model SVM, wykorzystujący jądro radialne (RBF), został wytrenowany bez pętli optymalizacyjnej - mimo to osiągnął stabilne i zadowalające wyniki. W przypadku modelu Random Forest najlepsze rezultaty uzyskano przy liczbie drzew wynoszącej 500, dalsze zwiększanie tej wartości nie przynosiło już zauważalnych korzyści.

Na podstawie uzyskanych danych wybrano powyższe wersje jako reprezentatywne i poddano je dalszemu porównaniu. Ich metryki zostały zestawione w jednej tabeli zbiorczej(tabela 7.3.1.), a także zaprezentowane graficznie na wykresie porównawczym(rys. 7.3.1. oraz rys. 7.3.2.). Takie podejście pozwoliło w przejrzysty sposób ocenić różnice w skuteczności poszczególnych modeli oraz wskazać ten, który najlepiej nadaje się do

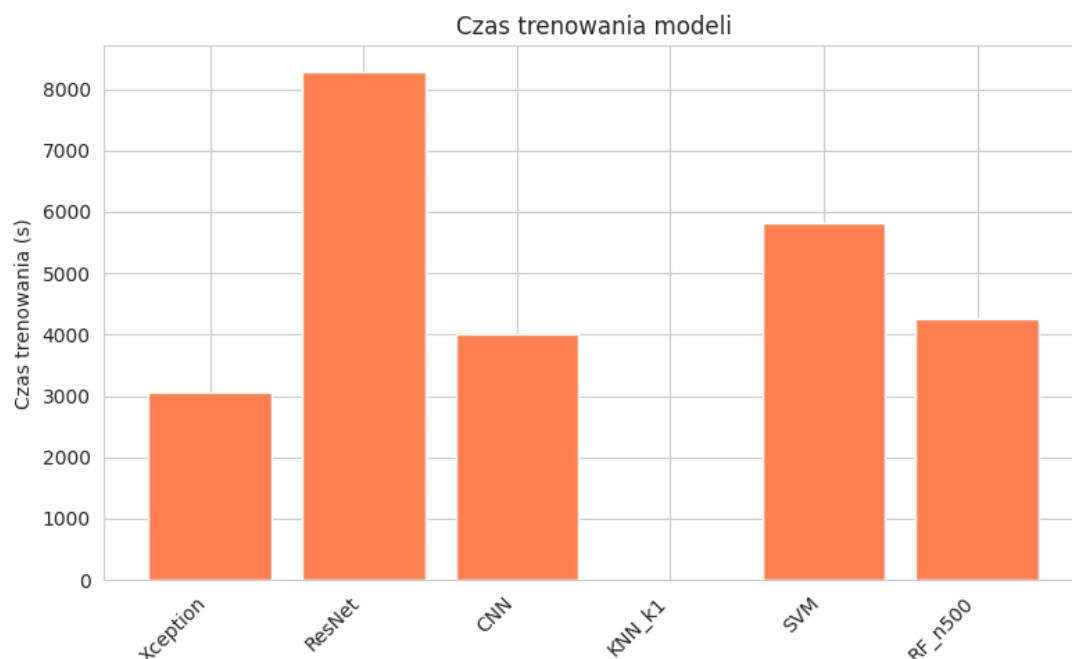
praktycznego wykorzystania w systemach wspomagających diagnozowanie niedoborów i chorób roślin na podstawie obrazu.

Model	Accuracy	Precision	Recall	F1_score	Czas trenowania [s]
Xception	0,9788	0,979	0,9777	0,9783	3059
ResNet50	0,8703	0,8777	0,8656	0,8716	8291
CNN	0,7035	0,8198	0,6074	0,6978	4012
KNN	0,5519	0,5803	0,4307	0,4388	0,38
SVM	0,7647	0,7134	0,6414	0,6538	5810
RandomForest	0,6843	0,6713	0,5158	0,5385	4255

Tabela 7.3.1. Zestawienie metryk skuteczności oraz czasów trenowania wszystkich modeli.



Rys. 7.3.1. Wykres porównujący metryki wszystkich modeli.

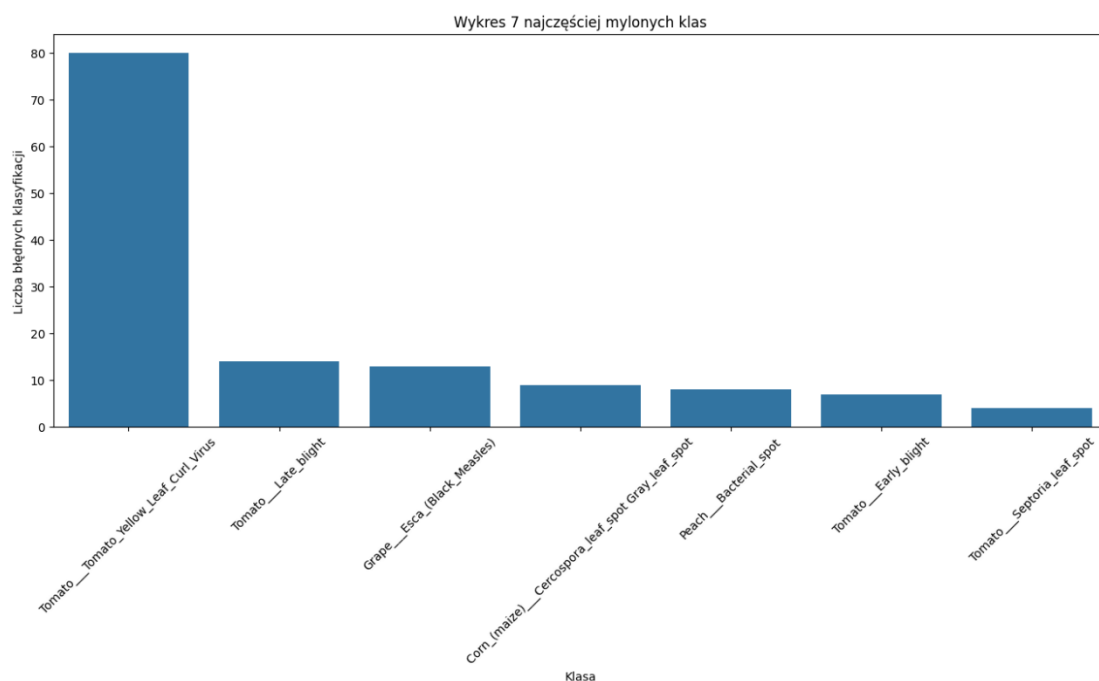


Rys. 7.3.2. Wykres porównujący czasy trenowania wszystkich modeli.

Dane z tabeli oraz wykresów zestawiających różne modele zostaną omówione w następnym rozdziale.

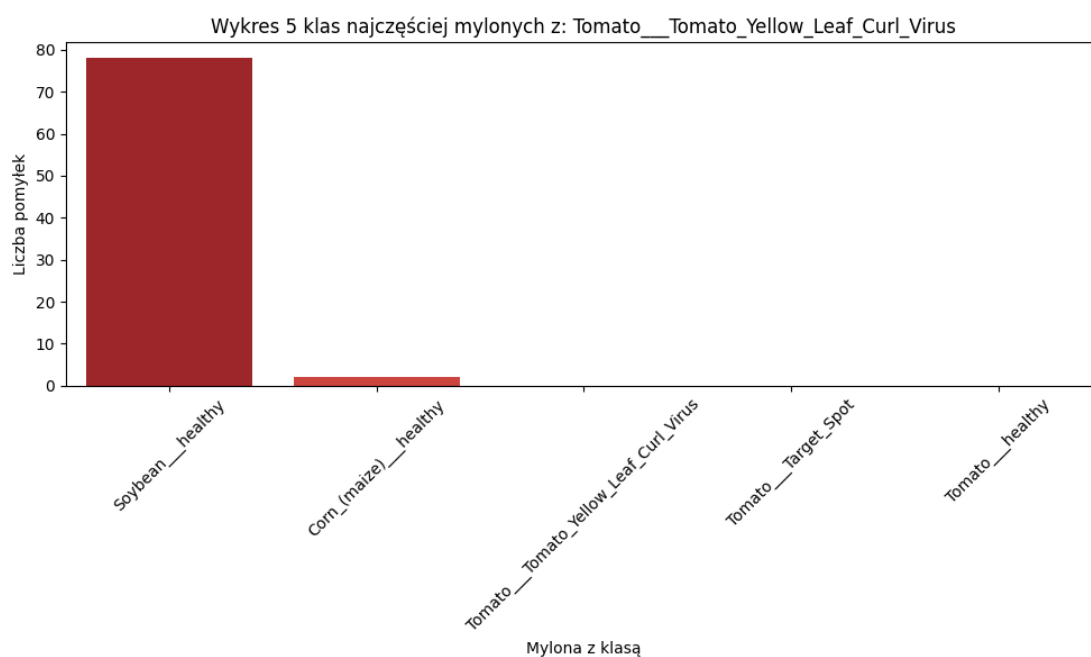
7.4. Analiza błędów

Model bazujący na architekturze Xception osiągnął najwyższą skuteczność spośród wszystkich testowanych algorytmów, zarówno pod względem dokładności klasyfikacji, jak i stabilności. Niemniej jednak, jak pokazuje wykres najczęściej mylonych klas (rys. 7.4.1.), model nie jest wolny od błędów i wykazuje istotne trudności w prawidłowej klasyfikacji etykiety *Tomato*___*Tomato_Yellow_Leaf_Curl_Virus*.



Rys. 7.4.1. Wykres najczęściej mylonych klas przez model Xception.

Analiza wskazuje, że klasa ta jest zdecydowanie najczęściej błędnie klasyfikowana - liczba błędnych predykcji dla tej etykiety wynosi 78 przypadków, co znacznie odbiega od innych klas. Dalsze badanie przedstawione na wykresie najczęściej błędnie przypisywanych etykiet dla klasy *Tomato__Tomato_Yellow_Leaf_Curl_Virus* (rys. 7.4.2.), pokazuje, że obrazy z tej klasy są najczęściej mylone z klasą *Soybean__healthy*. Tego typu błędy mają charakter krytyczny, ponieważ oznaczają, że model uznaje rośliny zainfekowane wirusem za zdrowe, co w praktyce rolniczej może prowadzić do zaniechania interwencji, rozprzestrzeniania się patogenu w całej szklarni i poważnych strat.



Rys. 7.4.2. Wykres 5 klas najczęściej mylonych z etykietą TYLCV.

Jedną z głównych przyczyn tego rodzaju błędów może być przetrenowanie modelu na klasach dominujących w zbiorze danych. W sytuacji, gdy obrazy przedstawiające zdrowe rośliny soi występują znacznie częściej niż inne klasy, model może nabyć tendencję do przypisywania niepewnych przypadków właśnie do tej klasy. Dodatkowo, możliwym źródłem problemu jest niewystarczająca liczba przykładów klasy TYLCV lub zbyt małe zróżnicowanie. Obrazy należące do tej klasy mogły być zbyt jednorodne – zarówno pod względem wizualnym, jak i warunków fotografowania – co mogło ograniczyć zdolność modelu do skutecznej generalizacji.

Należy również uwzględnić fakt, że objawy wirusa TYLCV, szczególnie w początkowej fazie, bywają subtelne i mogą przypominać cechy zdrowych liści innych gatunków roślin, takich jak soja. Jeśli dodatkowo model analizuje całość obrazu bez wyodrębnienia samego liścia, może nieświadomie uczyć się cech tła, koloru lub jasności, zamiast skupiać się na rzeczywistych objawach chorobowych. Brak mechanizmów skupiających uwagę modelu, takich jak warstwy attention, może również powodować, że nie identyfikuje on najistotniejszych fragmentów liścia. Poniżej przedstawiono przykłady błędnie sklasyfikowanych obrazów *Tomato__Tomato_Yellow_Leaf_Curl_Virus* (rys. 7.4.3.):

Błędne predykcje dla klasy: Tomato__Tomato_Yellow_Leaf_Curl_Virus



Rys. 7.4.3. przykłady błędnie sklasyfikowanych obrazów *Tomato__Tomato_Yellow_Leaf_Curl_Virus*.

8. Wnioski i podsumowanie

Poniższy rozdział zawiera końcowe podsumowanie wyników przeprowadzonych badań oraz najważniejsze wnioski wynikające z analizy skuteczności zastosowanych modeli. Omówiono również możliwości praktycznego wykorzystania opracowanego rozwiązania w rolnictwie precyzyjnym oraz wskazano ograniczenia i kierunki dalszego rozwoju systemu.

8.1. Ocena skuteczności badanych modeli

Model Xception osiągnął najwyższe wyniki spośród wszystkich analizowanych algorytmów, zarówno pod względem skuteczności klasyfikacyjnej (accuracy: 0,9788, F1-score: 0,9783), jak i efektywności czasowej (czas trenowania: 3059 sekund). Kluczowym aspektem technicznym, który mógł zadecydować o jego przewadze nad pozostałymi modelami, jest zastosowanie separowalnych konwolucji głębokich (depthwise separable convolutions). Ten rodzaj warstw pozwala na bardziej efektywne i precyzyjne przetwarzanie informacji wizualnej przy jednoczesnym zmniejszeniu liczby parametrów, co przekłada się na mniejsze ryzyko przeuczenia i lepszą generalizację. Architektura Xception pozwala więc na uchwycenie złożonych i subtelnych wzorców w danych obrazowych przy zachowaniu wysokiej wydajności obliczeniowej.

Należy jednak zauważyć, że mimo ogólnej wysokiej skuteczności, model Xception wykazuje objawy przetrenowania względem jednej klasy, co potwierdzono w analizie błędów. Obrazy z klasy *Tomato__Tomato_Yellow_Leaf_Curl_Virus* są bardzo często błędnie klasyfikowane jako *Soybean__healthy*, co może prowadzić do krytycznych konsekwencji w praktyce - szczególnie w warunkach szklarniowych, gdzie zaniechanie interwencji w przypadku choroby wirusowej może skutkować szybkim rozprzestrzenieniem się infekcji i znacznymi stratami plonów.

Z kolei model ResNet50, mimo że również opiera się na transfer learningu i osiągnął przyzwoite wyniki (accuracy: 0,8703, F1-score: 0,8716), wykazał się większą niestabilnością podczas trenowania, szczególnie przy zwiększaniu liczby epok. W praktyce zaobserwowano, że model był wrażliwy na przetrenowanie – po przekroczeniu pewnej liczby iteracji jego dokładność zaczynała spadać, co świadczy o trudności w optymalizacji tak głębokiej architektury bez odpowiednio dużego zbioru danych.

Własnoręcznie zaprojektowana konwolucyjna sieć neuronowa (CNN) osiągnęła umiarkowane rezultaty (accuracy na poziomie 0,7035 i F1-score 0,6978) oraz trenowała się około 4000 sekund. Choć wynik ten może być satysfakcjonujący przy mniejszej dostępności zasobów lub danych, nie dorównuje on modelom transfer learningu pod względem jakości klasyfikacji.

Spośród klasycznych algorytmów, SVM (Support Vector Machine) osiągnął najlepsze wyniki (accuracy: 0,7647 i F1-score: 0,6538) jednak jego czas trenowania wyniósł aż 5810 sekund, co czyni go nieopłacalnym, zwłaszcza przy wyraźnie niższych wynikach niż CNN. Modele KNN oraz Random Forest osiągnęły zdecydowanie najgorsze wyniki. KNN wykazał się niską dokładnością (0,5519) i najniższym wskaźnikiem recall (0,4307), mimo niemal zerowego czasu trenowania, co jest naturalną cechą tego algorytmu. Z kolei Random Forest

osiągnął tylko nieco wyższą skuteczność (F1-score: 0,5385), przy znaczącym czasie trenowania sięgającym 4255 sekund.

8.2. Wnioski praktyczne i możliwe kierunki dalszych badań

Osiągnięte wyniki wskazują na dużą skuteczność modeli opartych na transfer learningu, w szczególności architektury Xception, w klasyfikacji obrazów liści pod kątem niedoborów i chorób. Zastosowane podejście pozwala na automatyczną ocenę kondycji roślin w sposób skalowalny, szybki i powtarzalny, co jest szczególnie istotne w warunkach produkcji szklarniowej, gdzie liczy się czas reakcji i precyzja diagnozy. Mimo wysokiej dokładności ogólnej, analiza błędów wykazała, że model może popełniać poważne pomyłki w przypadku niektórych klas – zwłaszcza klasy *Tomato__Tomato_Yellow_Leaf_Curl_Virus*, która była wielokrotnie błędnie klasyfikowana jako *Soybean__healthy*.

Wyniki te pokazują, że dalsze prace powinny koncentrować się nie tylko na zwiększaniu ogólnej skuteczności klasyfikacyjnej, ale przede wszystkim na minimalizowaniu ryzyka błędów krytycznych, takich jak nierozpoznanie objawów choroby wirusowej. Problem ten może być częściowo spowodowany nierównomiernym rozkładem klas w zbiorze danych, gdzie klasy przedstawiające zdrowe liście soi występowały znacznie częściej niż obrazy roślin zainfekowanych TYLCV. Może to prowadzić do uprzywilejowania klasy dominującej w procesie uczenia i zbyt pochopnego przypisywania do niej przypadków niejednoznacznych.

W celu ograniczenia tego zjawiska warto rozważyć zastosowanie wag klasowych podczas trenowania modelu, co umożliwi nadanie większego znaczenia klasom niedoreprezentowanym. Dobrym rozwiązaniem byłoby także rozszerzenie zbioru danych o nowe obrazy przedstawiające objawy TYLCV w różnych stadiach rozwoju i w odmiennych warunkach środowiskowych, co zwiększyłoby różnorodność cech wizualnych i poprawiłoby zdolność modelu do generalizacji.

Kolejnym możliwym kierunkiem poprawy skuteczności diagnostycznej jest zastosowanie metod umożliwiających skupienie uwagi modelu na obszarach najbardziej istotnych z punktu widzenia klasyfikacji. Można tu wymienić wprowadzenie warstw typu attention lub zastosowanie segmentacji obrazu liścia przed klasyfikacją. W tym drugim przypadku eliminacja tła mogłaby zmniejszyć wpływ nieistotnych informacji, takich jak kolor otoczenia czy obecność innych obiektów na zdjęciu, które mogą zaburzać proces decyzyjny sieci neuronowej.

Usprawnieniem może być również rozszerzenie procesu fine-tuningu modelu bazowego Xception na większą liczbę warstw konwolucyjnych, co pozwoli lepiej dopasować niskopoziomowe cechy obrazu do specyfiki danych pochodzących z upraw szklarniowych. Warto także wykorzystywać techniki wizualizacji, takie jak Grad-CAM, które pozwalają sprawdzić, które obszary obrazu były decydujące w procesie klasyfikacji. Tego rodzaju analiza może pomóc w identyfikacji przypadków, w których model kieruje się błędnymi przesłankami i umożliwić lepsze dostrojenie jego architektury.

W perspektywie długoterminowej, skuteczność systemu diagnostycznego mogłaby zostać zwiększona przez integrację modelu klasyfikacyjnego z dodatkowymi źródłami danych, takimi jak czujniki środowiskowe monitorujące temperaturę, wilgotność czy skład atmosfery w szklarni. Połączenie informacji obrazowych z danymi kontekstowymi pozwoliłoby na stworzenie kompleksowego systemu wspomagającego decyzje rolników, który nie tylko diagnozuje aktualny stan roślin, ale także prognozuje możliwe zagrożenia i sugeruje optymalne zabiegi pielęgnacyjne.

9. Bibliografia

- [1] Organizacja Narodów Zjednoczonych ds. Wyżywienia i Rolnictwa (FAO). Projekcje dotyczące Wyżywienia i Rolnictwa do 2050 r. – Alternatywne ścieżki do 2050 r., <https://www.fao.org/global-perspectives-studies/fofa/en/>, 07.04.2025.
- [2] Bank Ochrony Środowiska: Artykuł omawiający degradację gleby jako problem lokalny i globalny, uwzględniający aspekty ekologiczne i ekonomiczne., <https://www.bosbank.pl/EKO/tresci-ekologiczne/degradacja-gleby-problem-lokalny-i-globalny>, 08.04.2025.
- [3] Instytut Uprawy Nawożenia i Gleboznawstwa – Państwowy Instytut Badawczy: Raport dotyczący środowiskowych skutków zakwaszenia gleb użytkowanych rolniczo oraz metod ich ograniczania, https://www.iung.pl/sir/zeszyt66_4.pdf, 09.04.2025.
- [4] MBF Group. Nawozy: Sytuacja na rynku – analiza prognoz cenowych, <https://mbfgroup.pl/nawozy-sytuacja-na-rynku-w-2025-oraz-analiza-prognoz-cenowych/>, 12.04.2025.
- [5] Sterowanie nawadnianiem i klimatem w szklarniach: Serwis Adviser omawia zastosowanie komputerów klimatycznych do precyzyjnej kontroli warunków środowiskowych w szklarniach, co pozwala na optymalizację produkcji. Strona producenta, <https://www.phu-adviser.pl/oferta-systemow-nawadniania/sterowanie-nawadnianiem-i-klimat-w-szklarniach/>, 12.04.2025
- [6] Zagrożenia dla plonów – identyfikacja i zwalczanie chorób roślin, artykuł na stronie, <https://tuagro.pl/zagrozenia-dla-plonow-jak-identyfikowac-i-zapobiegac-chorobom-roslin/>, 18.04.2025.
- [7] Składniki pokarmowe – ich rola w metabolizmie roślin, artykuł na stronie, <https://agrokultura.org/skladniki-pokarmowe-wzajemne-relacje-i-ich-dostepnosc/>, 18.04.2025.
- [8] Artykuł omawiający ryzyko strat w produkcji rolniczej, w tym wpływ chorób i niedoborów na plony, <https://rnr.sggw.edu.pl/article/download/8645/7624>, 18.04.2025.
- [9] EWA STOMPOR-CHYZAN, Diagnostyka makroskopowa chorób grzybowych – praca naukowa omawiająca metody rozpoznawania chorób grzybowych na roślinach, <https://journals.ur.edu.pl/pol-j-sust-dev/article/download/10589/9557/20304>, 18.04.2025.
- [10] Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147, 70–90, <https://doi.org/10.1016/j.compag.2018.02.016>, 22.04.2025.
- [11] Liakos, K. G., Busato, P., Moshou, D., Pearson, S., & Bochtis, D. (2018). Machine learning in agriculture: A review. *Sensors*, 18(8), 2674, <https://doi.org/10.3390/s18082674>, 22.04.2025.
- [12] Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, 145, 311–318, <https://doi.org/10.1016/j.compag.2018.01.009>, 28.04.2025.
- [13] Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7, 1419, <https://doi.org/10.3389/fpls.2016.01419>, 28.04.2025.

- [14] Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9), 2352–2449, https://doi.org/10.1162/neco_a_00990, 28.04.2025.
- [15] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. *Proceedings of the 36th International Conference on Machine Learning (ICML)*, <https://arxiv.org/abs/1905.11946>, 28.04.2025.
- [16] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. *CVPR 2018*, <https://arxiv.org/abs/1801.04381>, 02.05.2025.
- [17] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, <https://ieeexplore.ieee.org/document/5206848>, 02.05.2025.
- [18] Hughes, D. P. (2020). PlantVillage Project, Pennsylvania State University, <https://plantvillage.psu.edu>, 02.05.2025.
- [19] PlantVillage Dataset, Kaggle. <https://www.kaggle.com/datasets/emmarex/plantdisease>, 07.05.2025.
- [20] McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 51–56.
- [21] Abadi, M., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. <https://www.tensorflow.org>, 07.05.2025.
- [22] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1), 60.A
- [23] Géron, A. (2019). *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow*. Wydanie II. Helion. 13(2), 1322–1479.
- [24] V7 Labs – Splitting the dataset helps prevent overfitting and enables robust model evaluation, <https://www.v7labs.com/blog/train-validation-test-set>, 14.05.2025.
- [25] Analytics Vidhya (Medium) – Train/Test/Val ensures generalization to unseen data, <https://medium.com/analytics-vidhya/training-neural-networks-for-dummies-04-pt-1-train-validate-test-split-9b1141496c63>, 14.05.2025.
- [26] Analytics Vidhya (Medium) Data Splitting (Train-Test-Validation) in Machine Learning, <https://medium.com/%40rohanmistry231/data-splitting-train-test-validation-in-machine-learning-2d5d1927fa69>, 14.05.2025.
- [27] DataScience: 80-20 or 80-10-10 for training machine learning models? <https://datascience.stackexchange.com/questions/69907/80-20-or-80-10-10-for-training-machine-learning-models>, 14.05.2025.
- [28] Computer Vision Foundation - Deep Learning with Depthwise Separable Convolutions(Xception), 19.05.2025. https://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf,
- [29] Cornell University - Xception: Deep Learning with Depthwise Separable Convolutions, <https://ieeexplore.ieee.org/document/8099678>, 19.05.2025.
- [30] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. <https://arxiv.org/abs/1512.03385>, 19.05.2025.
- [31] Kien, C. T. (2022). Skip Connection and Explanation of ResNet. Medium. <https://chautuankien.medium.com/skip-connection-and-explanation-of-resnet-afabe792346c>, 21.05.2025.
- [32] Dokumentacja techniczna biblioteki Keras, https://www.tensorflow.org/api_docs/python/tf/keras, 21.05.2025.

- [33] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Duchesnay É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- [34] Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20(3), 273–297.
- [35] Dokumentacja techniczna jądra RBG modelu SVM, https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#sphx-glr-auto-examples-svm-plot-rbf-parameters-py, 02.06.2025.
- [36] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32.
- [37] Lucca Portes C., Simon B., Jean Paul B., Laurent H. - Random Forest Kernel for High-Dimension Low Sample Size Classification, <https://arxiv.org/pdf/2310.14710>, 02.06.2025.
- [38] Scikit-learn – opis funkcji `accuracy_score`, `precision_score`, `recall_score`, `f1_score`, i sposób użycia, <https://scikit-learn.org/stable/api/sklearn.metrics.html>, 03.06.2025.
- [39] Machine Learning Mastery - How to Calculate Precision, Recall, F1, and More for Deep Learning Models, <https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models>, 03.06.2025.
- [40] Margherita Grandini, Enrico Bagli, Giorgio Visani. Metrics for Multi-Class Classification: an Overview, <https://arxiv.org/abs/2008.05756>, 03.06.2025.