Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Systemy internetowe wspomagania zarządzania

Predykcja chorób roślin

Michał Sołtysiak
Numer albumu 309486

promotor
dr inż. Mariusz Kamola

WARSZAWA 2022

Predykcja chorób roślin

Streszczenie. W pracy przedstawiono badania modeli do wykrywania chorób roślin na przykładzie wybranej choroby pomidora. Opisane modele do klasyfikacji wykorzystują zdjęcia liści, a w dalszych podejściach również dane środowiskowe, takie jak temperatura i wilgotność powietrza, czy ilość opadów atmosferycznych. Zastosowany wyjściowy model do klasyfikacji zdjęć to ResNet50. Praca zawiera także opisy zastosowanych metod przetwarzania wstępnego użytych przy tworzeniu zbioru danych z posiadanych przez autora zdjęć. Po weryfikacji stworzonych rozwiązań, w stopniu na jaki pozwalają dostępne dane, wyciągnięte zostają wnioski i przedstawione możliwe ścieżki dalszego rozwoju.

Słowa kluczowe: klasyfikacja, uczenie maszynowe, wykrywanie chorób, ResNet50, dane środowiskowe

Plant disease prediction

Abstract. This work presents results on training models for plant disease detection on the example of selected tomato disease. Described classification models use leaf photographies and, in some approaches, environmental data as well, i.e. air temperature, air humidity and rainfall. Created models are based on pretrained ResNet50 neural network. This publication also contains descriptions of applied preprocessing methods for creating a dataset from photographies owned by the author. After verification of presented solutions, as accurate as available data make it possible, the conclusions are drawn and future research directions are presented.

Keywords: classification, machine learning, disease detection, ResNet50, environmental data

Spis treści

1. Wstęp

1.1. Wprowadzenie do tematyki problemu

Zapobieganie chorobom roślin uprawnych oraz szybkie reagowanie na ich pierwsze objawy są kluczowe dla maksymalizacji uzyskiwanych plonów. W przypadku dużych upraw potrzeba dużo ludzkiej pracy, by w porę dostrzec pierwsze objawy choroby. Wielu plantatorów przyjmuje strategię prewencyjną, tzn. cyklicznych oprysków. Nietrudno zauważyć, że zarówno w pierwszej strategii, jak i drugiej, potrzebne są niemałe nakłady finansowe. W pierwszym przypadku jest to koszt dodatkowej pracy ludzkiej, a w drugim wyższe koszty środków ochrony roślin i pracy związanej z wykonaniem oprysków. Dodatkowo na niekorzyść drugiej strategii przemawia jej negatywny wpływ na środowisko naturalne.

1.2. Definicja problemu

Problem poruszany w tej pracy to predykcja chorób roślin. Dysponując technologią, która w sposób automatyczny wykrywa choroby roślin uprawnych na wczesnym etapie, jesteśmy w stanie ograniczyć żmudną ludzką pracę, koszty związane ze stosowaniem środków ochrony roślin oraz negatywny wpływ plantacji na środowisko naturalne. Posuwając się o krok dalej, czyli z pomocą modelu przewidując wystąpienie konkretnej choroby z dużym prawdopodobieństwem, uzyskujemy narzędzie pozwalające na jeszcze większe ograniczenie strat w zbiorach.

1.3. Cele pracy

Pierwszym celem pracy jest stworzenie użytecznego modelu do wykrywania wybranej choroby na podstawie zdjęć liści. Ze względu na posiadane dane wybór padł na chorobę pomidora - zarazę ziemniaka (ang. *late blight*), która jest wywoływana przez patogen *Phytophthora infestans*.

Kolejnym celem jest próba poprawienia klasyfikacji z pomocą danych środowiskowych odnoszących się do stanowiska, na którym rosła roślina oraz czasu wykonania zdjęcia.

W pracy omówione zostają w pierwszej kolejności podstawowe zagadnienia związane z uczeniem maszynowym, w tym metody uczenia, ewaluacji i regularyzacji modeli. W kolejnym rozdziale opisane zostały wykorzystane przez autora narzędzia. W rozdziale 4 opisano wykorzystane dane i zastosowane metody przetwarzania wstępnego. Rozdział 5 zawiera opis procesu uczenia modelu do klasyfikacji liści na podstawie zdjęć - od wyboru podejścia i modelu ogólnego zastosowania do weryfikacji uzyskanych rezultatów. Rozdział 6 został poświęcony próbom stworzenia modelu rozszerzonego o dodatkowe wejście wyliczane na podstawie danych środowiskowych powiązanych z czasem i miejscem wykonania zdjęć. W ostatnim rozdziale wyciągnięte zostały wnioski i zaproponowane ścieżki dalszego rozwoju.

2. Metody uczenia maszynowego

2.1. Wstęp

W ostatnich latach uczenie maszynowe (ang. *machine learning*) mocno zyskało na popularności, z czym idzie w parze dynamiczny rozwój tej dziedziny. Początkowo [1] metody uczenia maszynowego stosowane były do rozwiązywania problemów trudnych dla człowieka, lecz prostych dla komputera - dających się w nieskomplikowany sposób wyrazić za pomocą matematycznych zależności. Prawdziwym wyzwaniem są jednak problemy proste dla człowieka, lecz trudne w formalizacji, takie jak zagadnienia związane z analizą obrazu, czy rozpoznawaniem mowy.

2.2. Typy uczenia maszynowego

Pojęcie uczenia maszynowego jest bardzo szerokie - zostało stworzone wiele metod i algorytmów wykorzystywanych w tym celu. W tym podrozdziale zaprezentowane zostaną podstawowe techniki uczenia maszynowego.

Uczenie nadzorowane (ang. *supervised learning*) [1] to uczenie modelu na danych, do których zostały z góry przypisane etykiety. Innymi słowy jest to proces polegający na łączeniu wartości wejściowych z pewnego zbioru X z wartościami wyjściowymi z ustalonego zbioru Y na podstawie zbioru przykładów. Często nadawanie etykiet ze zbioru Y próbkom ze zbioru X dokonywane jest przez człowieka, ale w niektórych przypadkach proces ten można zautomatyzować.

Uczenie częściowo nadzorowane (ang. *semi-supervised learning*) to uczenie modelu na danych mieszanych, tzn. na przykładach z przydzielonymi etykietami i na przykładowych wejściach bez nich. Zazwyczaj w kontekście uczenia głębokiego celem jest ustalenie wspólnej reprezentacji dla obu zbiorów i wykorzystanie jej innym procesie uczenia.

Uczenie nienadzorowane (ang. *unsupervised learning*) [1] to uczenie modelu na danych bez nadanych etykiet. Polega ono na poszukiwaniu przez model w procesie uczenia pewnych cech charakteryzujących przetwarzane dane i dalsze posługiwanie się tymi cechami do opisu danych otrzymanych na wejściu.

Uczenie ze wzmocnieniem (ang. *reinforcement learning*) to typ uczenia, w którym agent musi metodą prób i błędów nauczyć się jak najlepiej wykonywać określone zadanie. Na przestrzeń, w której działa nałożone są pewne ograniczenia, ale sam agent nie otrzymuje żadnych wskazówek, w jaki sposób powinien dążyć do celu.

2.3. Sieci neuronowe

Sieci neuronowe (ang. *Neural Networks, NNs*) to struktury obliczeniowe inspirowane biologicznymi sieciami neuronowymi, z których zbudowane są mózgi zwierząt. Taka struktura złożona jest z połączonych ze sobą wierzchołków, nazywanych sztucznymi neuronami.

Podobnie jak w przypadku biologicznych neuronów, przepływa przez nie sygnał, wyrażany liczbą, który wpływa na wszystkie neurony, z którymi się łączy.

Modele budowane są z warstw zawierających neurony, gdzie w klasycznym podejściu neurony z warstwy n łączą się z neuronami z warstwy $n+1$. Każdy neuron przechowuje wagę, która jest dopasowywana w procesie uczenia. Warstwy, których jednostki nie mogą być obserwowane, nazywamy ukrytymi.

Z sieciami neuronowymi związane jest również pojęcie funkcji aktywacji. Jest to funkcja, która pozwala na obliczenie wartości wyjściowej z danego neuronu na podstawie otrzymanych wartości z neuronów z nim połączononych. Wsród popularnych funkcji aktywacji wymienić można m.in.:

- znormalizowana funkcja wykładnicza (softmax) - znormalizowana w ten sposób, by wszystkie wartości wyjściowe z warstwy sumowały się do 1, wzór dla i-tego wyjścia:

$$f_i(x) = \frac{e^{x_i}}{\sum_{i=1}^{K} \cdot e^{x_j}} \tag{2.1}$$

- tangens hiperboliczny - funkcja o zbiorze wartości $(-1, 1)$:

$$f(x) = \frac{2}{1+e^{-\beta x}} - 1 \tag{2.2}$$

  gdzie β jest ustalonym parametrem

- jednostronnnie obcięta funkcja liniowa (ReLU) - prosta w obliczaniu, przedział wartości $[0, +\infty)$:

$$f(x) = max(0, x) \tag{2.3}$$

Sieci neuronowe mogą być uczone zarówno w sposób nadzorowany, jak i nienadzorowany. W tej pracy wykorzystane zostało uczenie nadzorowane.

2.3.1. Głębokie sieci neuronowe

Głębokie sieci neuronowe (ang. *deep neural networks, DNNs*) to sieci neuronowe zawierające wiele warstw ukrytych, czyli warstw pomiędzy warstwą wejściową a wyjściową, w których wartości wyjściowe z neuronów nie są przeznaczone do interpretacji przez człowieka. Podstawowym typem, który można wyróżnić wśród sieci głębokich to sieci jednokierunkowe (ang. *feedforward neural networks*), gdzie jedynym kierunkiem przepływu sygnałów jest od warstwy wejściowej do wyjściowej. Innym rodzajem są rekurencyjne sieci

neuronowe (ang. *recurrent neural networks*), w których sygnał może poruszać się w każdym kierunku.

Spośród wielu typów sieci neuronowych, szczególną popularnością w zagadnieniach związanych z analizą obrazu cieszą się konwolucyjne sieci neuronowe (ang. *convolutional neural networks, CNNs*). Są to głębokie sieci jednokierunkowe, w których istotną rolę odgrywa wydobywanie cech (ang. *feature extraction*) z danych wejściowych, dzięki czemu nie jest konieczne zaawansowane przetwarzanie wstępne (ang. *preprocessing*) danych przed użyciem ich do uczenia modelu.

2.4. Weryfikacja jakości klasyfikacji

W przypadku uczenia maszynowego, gdzie model klasyfikujący powstaje w wyniku algorytmicznego procesu dopasowywania wag, szczególnie istotna jest weryfikacja jakości klasyfikacji. Taka ocena powinna być wiarygodna i możliwa do przeprowadzenia w sposób automatyczny. W tym podrozdziale opisane zostaną podstawowe metody i miary służące ewaluacji modeli.

2.4.1. Zbiór testowy

Podział zbioru danych na uczący i testowy ma niebagatelne znaczenie w trenowaniu modeli. Każdy uczony model potrzebuje weryfikacji - czy rzeczywiście klasyfikuje to co powinien i z jakim błędem. Szczególnie w przypadku trenowania wielu modeli i poszukiwania optymalnych parametrów uczenia potrzebny jest wiarygodny proces ich ewaluacji. Podstawową metodą jest podział próbek na dwa rozłączne zbiory: uczący i testowy. Wyłączenie z procesu uczenia próbek, którymi zamierzamy testować model jest bardzo ważne - celem modelu nie jest idealne dopasowanie się do próbek ze zbioru uczącego, ale jego zdolność do pewnej generalizacji. Nie ma jednych ustalonych proporcji podziału, ale zazwyczaj przyjmuje się, że zbiór testowy powinien stanowić przynajmniej 10% całego zbioru i przynajmniej tyle, by na podstawie uzyskanych wyników można było porównywać ze sobą modele.

2.4.2. K-krotny sprawdzian krzyżowy

Bardziej zaawansowaną metodą weryfikacji modelu jest k-krotny sprawdzian krzyżowy (ang. *cross-validation*). Polega on na dokonaniu podziału zbioru próbek na `k` części (na rozłączne podzbiory) i wykonaniu `k` iteracji uczenia i weryfikacji modelu. W każdej z iteracji `k-ty` zbiór jest wybierany jako testowy, a pozostałe `k-1` zbiorów jest wykorzystywane do uczenia. Wyniki klasyfikacji zbioru testowego ze wszystkich iteracji są na końcu uśredniane.

Istnieje również pojęcie stratyfikowanego sprawdzianu krzyżowego, gdzie dodatkowo nakłada się ograniczenie dotyczące podziału zbiorów: proporcje pomiędzy klasami w zbiorze testowym i w zbiorze uczącym powinny być takie same. Jest to szczególnie istotne, gdy w zbiorze danych występują duże dysproporcje w reprezentacji poszczególnych klas.

2.5. Miary jakości klasyfikacji

2.5.1. Dokładność

Chyba najbardziej intuicyjną miarą używaną do weryfikacji jakości klasyfikacji jest dokładność (ang. *accuracy*). Jest to stosunek liczby wszystkich poprawnych klasyfikacji do liczby wszystkich dokonanych klasyfikacji, czyli w ilu przypadkach model trafnie przewidział klasę.

2.5.2. Macierz błędów

Macierz błędów (ang. *confusion matrix*) jest prostym konceptem, a jednocześnie przy jej użyciu można wyliczyć wartości wszystkich podstawowych miar jakości klasyfikacji. W tabeli 2.1 przedstawiona została prosta macierz błędów. Takie macierze stosowane są do każdej z klas z osobna - wtedy wartość TAK dotyczy tej klasy, a NIE wszystkich pozostałych.

Tak więc w macierzy błędów dla klasy A wartości *True Positive* będą oznaczać wszystkie poprawne klasyfikacje modelu do tej klasy, a *False Negative* wszystkie poprawnie uznane za nienależące do klasy A. Wartość *False Positive* to liczba próbek nienależących do klasy A, które zostały uznane przez model za reprezentantów klasy A. Wartość *True Negative* to liczba przypadków, w których model dla próbek klasy A przypisał inną klasę.

Tabela 2.1. Macierz błędów dla pojedynczej klasy modelu

	wartość prawdziwa: TAK	**wartość prawdziwa: NIE**
predykcja: TAK	True Positive (TP)	False Positive (FP)
predykcja: NIE	True Negative (TN)	False Negative (FN)

2.5.3. F1-score

Bardzo popularną miarą jakości klasyfikacji jest wskaźnik F1-score, który określa jak dobrze model radzi sobie z rozpoznawaniem danej klasy. Jest to szczególnie przydatna miara w przypadku uczenia modelu klasyfikującego do wielu klas - sugerując się jedynie wartością dokładności, może umknąć nam fakt, że jedna z klas jest słabo rozpoznawana przez model. Wszystkie wartości potrzebne do wyliczenia wartości tego wskaźnika znajdziemy w macierzy błędów, co unaocznia wzór 2.4.

$$F_1 score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.4}$$

gdzie

$$precision = \frac{TP}{TP + FP} \tag{2.5}$$

$$recall = \frac{TP}{TP + FN} \tag{2.6}$$

Precyzja (ang. *precision*) jest miarą określającą jak dobrze model radzi sobie z rozpoznawaniem danej klasy, tzn. nie nadaje etykiet tej klasy obiektom, które do niej nie należą.

Czułość (ang. *recall*) jest miarą mówiącą jak wiele obiektów spośród należących do danej klasy zostało poprawnie rozpoznanych przez model.

2.5.4. Uśredniony F1-score

Wskaźnik F1-score jest bardzo użyteczną miarą jakości klasyfikacji, ale jest oddzielnie liczony dla każdej z klas. Aby prosto porównywać pomiędzy sobą modele, wygodnie jest mieć jedną liczbę oceniającą trafność przydzielanych przez nie etykiet. Najprostszym rozwiązaniem jest F1-average (F1-macro), czyli średnia arytmetyczna wartości F1-score dla wszystkich klas. Bardziej wiarygodnym wskaźnikiem jest F1-weighted, czyli średnia ważona wartości F1-score dla wszystkich klas, co opisuje wzór 2.7:

$$F_1 weighed = r_0 \cdot F_1(k_0) + r_1 \cdot F_1(k_1) + ... + r_n \cdot F_1(k_n) \tag{2.7}$$

gdzie

$F_1(k_i)$ - wartość $F_1 score$ dla klasy k_i
$r_i = \frac{TP_i+TN_i}{TP_i+TN_i+FP_i+FN_i}$ - reprezentacja klasy k_i w zbiorze

Warto zauważyć, że dla równej reprezentacji klas w zbiorach uczącym i testowym wartości F1-macro oraz F1-weighted będą sobie równe. Dlatego szczególnie przydatne jest korzystanie ze wskaźnika F1-weighted w przypadku zbiorów o różnym wsparciu dla poszczególnych klas.

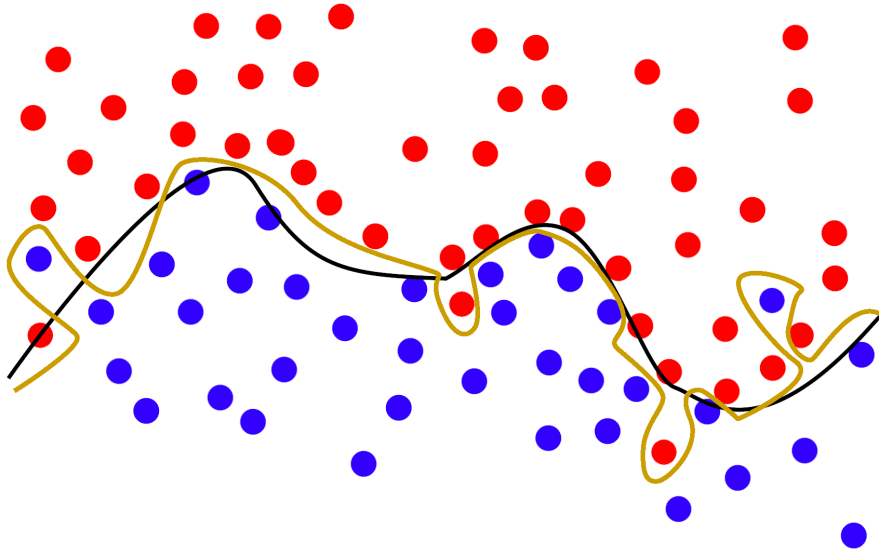
2.6. Problemy związane z uczeniem modeli

2.6.1. Nadmierne dopasowanie do danych uczących

Nadmierne dopasowanie (ang. *overfitting*) modelu do danych uczących to jeden z niepożądanych efektów uczenia maszynowego. Występuje zazwyczaj, gdy model ma zbyt wiele parametrów w stosunku do danych, na których jest uczony. Dodatkowym czynnikiem prowadzącym do przeuczenia jest zbyt długie trenowanie modelu. Problem nadmiernego dopasowania polega na tym, że pomimo świetnych wyników klasyfikacji dla próbek ze zbioru uczącego, model słabo radzi sobie z klasyfikacją próbek z niego nie pochodzących. Zwizualizowane zostało to na rysunku 2.1.

2.7. Metody regularyzacji w uczeniu głębokim

Jak już zostało to zaznaczone w poprzednich podrozdziałach, istotna w uczeniu maszynowym z nadzorem jest zdolność wytrenowanych modeli do pewnej generalizacji, czyli znalezienia ogólniejszych reguł przypisywania etykiet próbkom wejściowym na podstawie ograniczonej liczby przykładów. By to osiągnąć, dążymy do uzyskania jak najlepszych wyników predykcji na zbiorze testowym, kosztem pogorszenia wyników dla zbioru uczą-



Rysunek 2.1. Nadmierne dopasowanie (overfitting). Linią czarną zaznaczono rozgraniczenie na klasy w prawidłowym (generalizującym) modelu, a linią złotą rozgraniczenie na klasy będące efektem nadmiernego dopasowania.

cego. Działania mające na celu osiągnięcie modelu o takich własnościach nazywamy regularyzacją.

Pragnąc zregularyzować tworzony model mamy do wyboru cały wachlarz metod i rozwiązań [1]. W tym podrozdziale przedstawione zostaną wybrane z nich, używane w uczeniu głębokim.

2.7.1. Wzbogacanie danych

Dość oczywistym sposobem na wytrenowanie modelu, który będzie lepiej generalizował, jest uczenie go na większym zbiorze danych. Dla niektórych problemów zasadne jest sztuczne generowanie danych do nauki, często jednak otrzymujemy skończony zbiór danych. Zastosowanie wzbogacenia danych (ang. *data augmentation*) pozwala nam niejako "rozmnożyć"dane.

Poprzez stosowanie obrotów, przycięć, czy różnego rodzaju filtrów (np. wpływających na kolory), jesteśmy w stanie z jednego zdjęcia uzyskać wiele próbek do nauki. Oczywiście dobór przekształceń i filtrów powinien być przemyślany - np. w przypadku rozpoznawania cyfr arabskich stosowanie obrotów poskutkuje najpewniej problemami modelu z rozpoznawaniem cyfr 6 i 9.

2.7.2. Porzucanie

Porzucanie (ang. *dropout*) to bardzo wydajna obliczeniowo i skuteczna metoda regularyzacji, którą można z powodzeniem stosować dla wielu różnych typów modeli. Polega ona na pomijaniu w trakcie uczenia losowo wybranych wyjść z warstwy, czy też tymczasowym porzucaniu losowo wybranych połączeń między wartstwami. Uzyskiwany efekt można porównać do szumu, który ma zapobiegać zbytniemu dopasowywaniu się do danych uczących.

2.8. Podsumowanie

W tym rozdziale wytłumaczone zostały podstawowe pojęcia z zakresu uczenia maszynowego, a także rodzaje sieci neuronowych. W szczegółach opisane zostały sposoby weryfikacji jakości klasyfikacji modeli oraz miary służące ich ewaluacji. Zaprezentowane zostały również częste problemy związane z procesem uczenia maszynowego.

3. Wykorzystane narzędzia

W tym rozdziale opisane zostaną narzędzia wykorzystane w toku prowadzonych badań. Cała część programistyczna została zrealizowana w języku Python, który ma ugruntowaną pozycję w środowisku uczenia maszynowego, o czym świadczy chociażby mnogość dostępnych otwartoźródłowych bibliotek i kursów. Dodatkowym czynnikiem, który wpłynął na ten wybór, była znajomość tego języka programowania przez autora.

3.1. Keras i TensorFlow

3.1.1. TensorFlow

TensorFlow [2] to otwartoźródłowa biblioteka programistyczna do uczenia maszynowego. Pierwsza wersja stworzonej przez Google biblioteki ukazała się w listopadzie 2015 roku i może być używana w wielu językach, oprócz Pythona m.in. w JavaScripcie, C++ i Javie. Głównym zastosowaniem TensorFlow są głębokie sieci neuronowe. Biblioteka pozwala na wykorzystanie zarówno mocy obliczeniowej wielowątkowych procesorów, jak i jednostek GPU z kart graficznych.

3.1.2. Keras

Keras [3] jest popularną biblioteką otwartoźródłową dla języka Python, która udostępnia m.in. wysokopoziomowe API dla TensorFlow. Wydana po raz pierwszy w 2015 roku biblioteka udostępniała interfejs dla wielu bibliotek niższego poziomu, wśród nich poza TensorFlow również Microsoft Cognitive Toolkit, Theano i PlaidML. Od wersji 2.4 (2020 r.) wspiera jedynie TensorFlow. Keras umożliwia budowanie nawet bardzo złożonych modeli w relatywnie prosty sposób. Udostępnia też klasy do przetwarzania wstępnego (ang. *preprocessing*) i wzbogacania (ang. *augmentation*) danych wejściowych. Ze względu na popularność tej biblioteki w internecie znaleźć można wiele materiałów, kursów oraz szybko rozwiązać napotykane problemy z pomocą np. społeczności StackOverflow. Wszystkie wymienione zalety wpłynęły na decyzję o wyborze tej właśnie biblioteki do przeprowadzania treningów modeli na potrzeby tej pracy. Wykorzystane w niej elementy Kerasa to m.in.:

- klasa `ImageDataGenerator` do dynamicznej augmentacji zdjęć w procesie uczenia
- model ResNet50 udostępniany przez pakiet Keras Applications
- funkcyjne API do budowy modeli
- zapisywanie modelu na dysk i jego późniejsze odczytywanie (`save_model` i `load_model`)
- klasa `tf.keras.utils.Sequence`, której rozszerzenie umożliwiło podawanie dwóch wejść do uczenia modeli z rodziału 6

3.2. OpenCV

OpenCV [4] to biblioteka zawierająca szereg funkcji służących do przetwarzania obrazu. Pierwsza wersja światło dzienne ujrzała już w roku 2000. Głównym jej zastosowaniem

jest przetwarzanie obrazu w czasie rzeczywistym. Co prawda sama biblioteka napisana jest w C/C++, ale posiada adaptery dla języków Java, Python i nie tylko. Ze względu na mnogość oferowanych funkcji, popularność oraz łatwą dostępność adaptera w języku Python to właśnie OpenCV zostało użyte do przetwarzania wstępnego zdjęć w trakcie przygotowywania zbioru uczącego.

3.3. NumPy

NumPy [5] jest jedną z najpopularniejszych bibliotek dla języka Python stworzoną w 2005/2006 roku na bazie istniejącej od 1995 roku biblioteki Numeric i konkurującej z nią Numarray. Pozwala na tworzenie obiektów macierzowych i tabelarycznych oraz implementuje wiele operacji, które można na takich obiektach dokonywać. Umożliwia również podawanie explicite typów liczbowych, spośród których oprócz typu m.in. całkowitoliczbowego, zmiennoprzecinkowego, czy zmiennoprzecinkowego podwójnej precyzji możemy również wybrać liczbę bitów w reprezentacji liczby (np. 16, 32, 64). Oprócz wygody przetwarzania danych, której nie dają pythonowe listy, można też bardziej zapanować nad pamięcią zajmowaną przez macierze liczbowe, co w przypadku dużych danych ma znaczenie niebagatelne. Sam Keras również korzysta z NumPy, co znacząco wpłynęło na korzystanie z tej biblioteki w kodzie.

4. Przygotowanie danych

4.1. Pozyskanie zbioru uczącego

Pozyskanie zbioru uczącego do wytrenowania modelu na potrzeby tej pracy okazało się zadaniem niełatwym. Poszukiwane dane nie należą do dostępnych powszechnie i w przystępnej formie, jak np. historyczne dane z rynków finansowych, czy zdjęcia zwierząt. Stworzenie zbioru do nauki przy pomocy tylko komputera i zasobów czasowych jest praktycznie niemożliwe. Web scraping, czyli zbieranie danych dostępnych w internecie za pomocą programu poruszającego się przy pomocy dostarczonej listy adresów URL oraz napotykanych odnośników, można zastosować jedynie w bardzo ograniczonym zakresie. Takim zastosowaniem może być konwersja formatu dostępnych danych na pożądany - np. danych meteorologicznych dla zdjęć ze znanym czasem i miejscem wykonania. W tym podrozdziale opisane zostaną wszystkie źrodła danych, które zostały wykorzystane do opisywanych badań.

4.1.1. PlantVillage

Pierwszym wykorzystanym przez autora pracy zbiorem danych był znaleziony w Internecie zbiór PlantVillage [6]. Zawiera on ponad 54 tys. zdjęć liści zdrowych i chorych podzielonych na 38 kategorii ze względu na gatunki i choroby. Samych kategorii dotyczących pomidora jest aż 10 - jedna odpowiadająca zdrowej roślinie i dziewięć odpowiadających wybranym chorobom dotykającym tych roślin. W tejże pracy wykorzystane zostały dwie kategorie: zdrowy pomidor o liczbie zdjęć 1591 oraz pomidor zainfekowany zarazą ziemniaka (ang. *late blight*) o liczbie zdjęć 1909.

Rysunek 4.1. Przykładowe zdjęcia ze zbioru PlantVillage

4.1.2. Zbiór własny

W okresie od 15 czerwca 2021 do 24 sierpnia 2021 podjęte zostały próby zbierania danych z trzech stanowisk na terenie Warszawy, na każdym stanowisku po 10 roślin. Zbierane były dane środowiskowe w postaci temperatury (w $°C$) i względnej wilgotności powietrza (w %) w odstępach piętnastominutowych oraz ręcznie wykonywane były zdjęcia liści co 2-4 dni. Podczas przygotowywania własnego zbioru danych wystąpiły liczne trudności, takie jak:

- ręczna klasyfikacja próbek
  - — obarczona błędem subiektywnej oceny
  - — problem ze zdjęciami liści z bardzo wczesnymi objawami - od którego momentu oznaczać jako chore, a które np. wyłączać z uczenia jako niejednoznaczne
- choroba w trakcie monitorowanego okresu nie wystąpiła
- wykonanywanie zdjęć liści bez oddzielania ich od rośliny, a jednocześnie by na zdjęciu znajdował się tylko wybrany liść

Szczególnie trudny był etap zbierania zdjęć, ponieważ nie było wiadome a priori jakie konkretnie zdjęcia będą użyteczne. Podjęta została decyzja, że będą to zdjęcia liści złożonych wykonywane na jednolitym tle. Trudnym do przewidzenia było jednak, że w trakcie realizacji tej pracy wystąpią trudności z wykorzystaniem wykonanych zdjęć, takie jak:

- ekstremalnie różne oświetlenie
- występowanie na zdjęciach cieni rzucanych zarówno przez fotografowany liść, jak i inne liście
- połączenie zdjęć tych samych liści w czasie - podział zdjęć występował na daty i na rośliny, ale już nie na konkretne liście
- przesłanianie niektórych listków przez inne z tego samego liścia złożonego

Spośród trzech stanowisk tylko na jednym wystąpiła zaraza ziemniaka. Spośród zdjęć z tego stanowiska naturalnie zdecydowana większość to zdjęcia zdrowych liści. Pokazuje to, że tak naprawdę niewiele zdjęć ze wszystkich wykonanych mogło zostać użytych do trenowania modeli, gdy reprezentacja klas w zbiorze uczącym jest taka sama. W tabeli 4.1 przedstawione zostały liczby przetworzonych wstępnie listków, które znalazły się w finalnym zbiorze danych. Podane liczby próbek to fragmenty oryginalnych zdjęć, bez wzbogacania.

Tabela 4.1. Liczba próbek w finalnym zbiorze danych

Klasa	Liczba próbek	Procent próbek
zdrowy	166	58.04%
lekko chory	87	30.42%
definitywnie chory	33	11.54%
Razem	**286**	**100%**

4.1.3. Stacja meteo Warszawa

Przydatnym źródłem danych okazała się Stacja meteo Warszawa[1]. Udostępniane przez nią publicznie historyczne dane meteorologiczne z rozdzielczością odczytów na poziomie kilku minut pozwoliły na rozszerzenie własnego zbioru danych o wartości opadów atmosferycznych (w mm) i nasłonecznienia (promieniowania słonecznego, w W/m^2).

[1] www.meteo.waw.pl

Rysunek 4.2. Przykładowe zdjęcia zawierające kłopotliwe przy wyodrębnianiu tła cienie oraz ilustrujące problem zmiennego oświetlenia

4.2. Wyodrębnienie tła zdjęcia

Pierwszym wyzwaniem związanym z przetwarzaniem wstępnym zdjęć było usunięcie tła ze zdjęć i zastąpienie go jednakowym. Celem takiego działania jest to, by model klasyfikujący skupiał się na liściach, a nie na innych widocznych elementach tworzących tło zdjęcia. Pierwsze próby klasyfikacji takich zdjęć modelem wytrenowanym na danych ze zbioru danych PlantVillage pokazały, że zdjęcia liści na białym źle są źle klasyfikowane przez model - wytrenowany na zdjęciach z szumem w tle model uznawał wszystkie zdjęcia na białym tle za chore. Po podmianie tła na podobny szum przypisywane nowym próbkom etykiety były w znaczącej większości poprawne. Następnie rozpoczęły się próby automatyzacji procesu, który początkowo wykonywany był ręcznie. W tym celu zostały napisane skrypty w języku Python z wykorzystaniem biblioteki OpenCV. Zważając na to, że posiadane zdjęcia liści wykonywane były na białym tle, to próby rozpoczęto od tworzenia maski poprzez progowanie (ang. *thresholding*) [7] zdjęcia konwertowanego do skali szarości (listing 4.1).

Listing 4.1. Progowanie w skali szarości

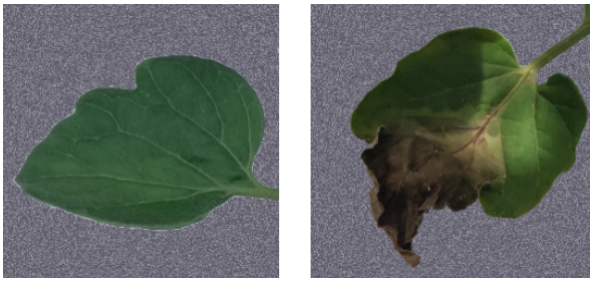
```
_min = 100
_max = 150
grayscale = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
ret, mask = cv.threshold(grayscale, _min, _max,
                         cv.THRESH_BINARY_INV)
```

Po uzyskaniu maski można było już w prosty sposób, wykorzystując mieszanie (ang. *blending*), podłożyć pod liście dowolne tło. Rezultaty były obiecujące, lecz znaczne różnice w oświetleniu oraz rzucane cienie powodowały, że na niektórych zdjęciach części liści były uznawane za tło, a cień na tle jako liść. Lepsze wyniki zostały uzyskane przy progowaniu zdjęcia w skali HSV (listing 4.2).

Listing 4.2. Progowanie w skali HSV

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)

# Define range of green color in HSV
lower_green = np.array([30, 0, 0])
upper_green = np.array([100, 255, 255])

# Threshold the HSV image to get only green colors
new_mask = cv.inRange(hsv, lower_green, upper_green)
```

Choć i w tym wypadku nie udało się dobrać jednego zbioru parametrów, który zadziałałby dobrze dla wszystkich zdjęć, to jednak okazała się o wiele skuteczniejsza. Procent zadowalająco przetworzonych zdjęć w przypadku metody ze skalą HSV wyniósł ok. 90%, podczas gdy w przypadku metody ze skalą szarości było to około 60%. Ta metoda została użyta finalnie do przetwarzania wstępnego w połączeniu z ręcznym korygowaniem nieprawidłowości.



Rysunek 4.3. Przykładowe zdjęcia zdrowego i chorego listka po podmianie tła

4.3. Próby automatycznego podziału liści złożonych

Wykonane zdjęcia obejmowały cały liść złożony, który składał się z wielu listków. Istotnym zadaniem było wycięcie tych listków do osobnych plików. Tak jak poprzednio, wykorzystany został w tym celu Python i OpenCV. Za pomocą operacji `threshold`, `erode` oraz `dilate` z odpowiednimi parametrami udało się wyeliminować ze zdjęcia łodyżki. Następnie, wykrywając kontury i obliczając `boundingRect`, wyznaczone zostały współrzędne używane dalej do wycinania fragmentów ze zdjęcia (oryginalnego). Przedstawia to fragment kodu z listingu 4.3.

Listing 4.3. Próba automatycznego podziału liścia złożonego na listki

```
# Removing thin stems
kernel = np.ones((7,7), np.uint8)
erosion = cv.erode(mask, kernel, iterations = 2)
dilation = cv.dilate(erosion, kernel, iterations = 10)

# Contours recognition for groups of pixels left
threshold = 50
canny_output = cv.Canny(dilation, threshold, threshold * 2)
contours, _ = cv.findContours(canny_output,
                              cv.RETR_EXTERNAL,
                              cv.CHAIN_APPROX_SIMPLE)

# Converting contours into start coordinates, width
# and height of rectangles bounding each contour
contours_poly = [None] *  len(contours)
bounding_rect = [None] *  len(contours)
for i, c in  enumerate(contours):
    contours_poly[i] = cv.approxPolyDP(c, 3, True)
    bounding_rect[i] = cv.boundingRect(contours_poly[i])
```

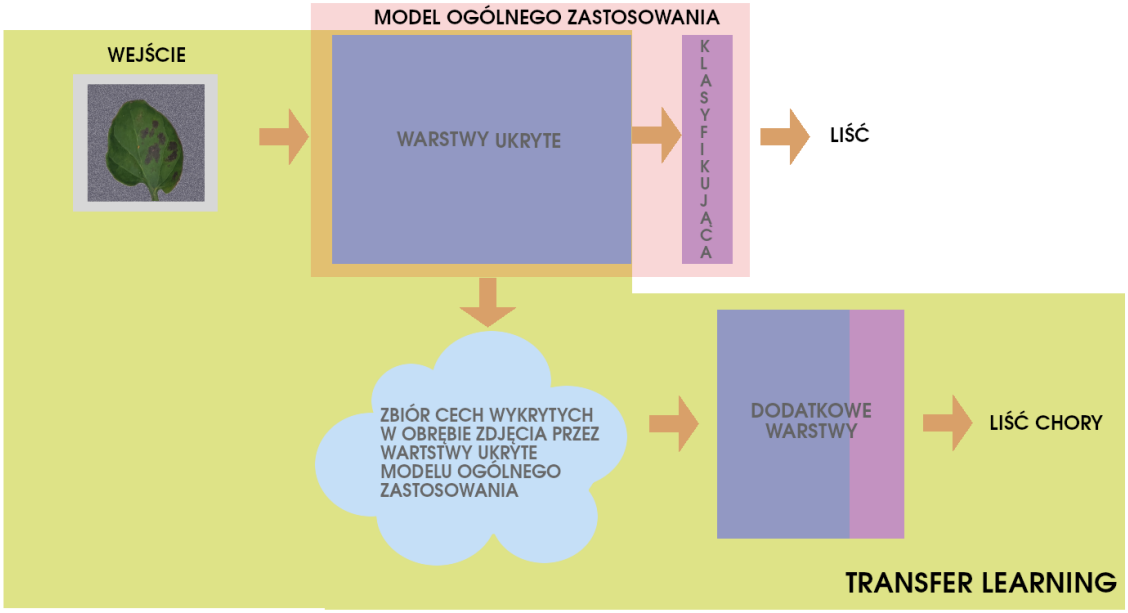
Niestety dobór parametrów dla jednego zdjęcia okazał się zupełnie nietrafiony dla innych. Ze względu na trudności związane z doborem parametrów do wyżej opisanej metody oraz częste nachodzenie na siebie liści na zdjęciach automatyzacja tego procesu została porzucona.

Rysunek 4.4. Automatycznie wycięty fragment zdjęcia, widoczne dwa problemy: nachodzenie na siebie listków oraz nieujęcie cienia w masce

5. Uczenie modelu do klasyfikacji liści

5.1. Transfer learning

Transfer learning [8], tłumaczony jako nauczanie metodą transferu, to technika w uczeniu maszynowym, który skupia się na wykorzystywaniu wiedzy zdobytej w procesie uczenia modelu rozwiązującego problem A do rozwiązania podobnego problemu B. Dla przykładu model wyuczony do rozpoznawania samochodów osobowych może zostać douczony tak, by rozpoznawał samochody ciężarowe. Na rysunku 5.1 koncept ten został przedstawiony na przykładzie zdjęcia chorego listka pomidora.



Rysunek 5.1. Schemat obrazujący koncept nauczania metodą transferu

5.1.1. Uzasadnienie wyboru

Decyzja o wykorzystaniu konceptu transfer learningu zamiast budowy modelu i trenowania go od początku miała wiele podstaw. Najważniejszą z nich był relatywnie nieduży zbiór danych uczących, który nie pozwoliłby na wytrenowanie wiarygodnego modelu. Innym problemem związanym z potencjalnym trenowaniem modelu od początku byłaby potrzebna do takiego uczenia moc obliczeniowa, którą autor nie dysponował. Wyjście od gotowego modelu ogólnego zastosowania, który został wcześniej wytrenowany na bardzo dużym zbiorze zdjęć i solidnie zweryfikowany przez wielu badaczy, wydało się dobrym pomysłem.

5.2. Wybór modelu ogólnego zastosowania

Następnym krokiem był wybór modelu ogólnego zastosowania. Przegląd dostępnych prac naukowych z tematyki rozpoznawania chorób roślin na postawie zdjęć pomógł w

przyjrzeniu się skuteczności i cechom charakterystycznym najpopularniejszych z takich modeli.

W pracy [9] z 2016 roku porównane zostały rezultaty klasyfikacji zdjęć liści roślin zdrowych i chorych ze zbioru danych PlantVillage przez sieci neuronowe AlexNet oraz GoogLeNet. W swoim rozwiązaniu autorzy resetują wagi wybranych warstw i następnie uczą oba modele ogólnego zastosowania. Przeprowadzone badania wykazały, że wytrenowana sieć GoogleLeNet zazwyczaj osiągała lepsze wyniki w klasyfikacji od sieci AlexNet. Różnica ta szczególnie widoczna była przy mniejszych zbiorach treningowych, jednak zarówno sieć AlexNet z 2012 roku, jak i GoogLeNet z 2014 roku osiągały przy zastosowaniu transfer learningu mocno zadowalające wyniki. Wskaźnik uśredniony F1-score dla modeli douczanych kolorowymi zdjęciami ze zbioru PlantVillage wynosił powyżej 97%.

W artykule naukowym [10] z 2019 roku natomiast porównanie modeli ogólnego zastosowania w kontekście wykrywania chorób roślin było szersze. Porównane zostały najpopularniejsze z nich: AlexNet, GoogLeNet, VGG16, VGG19, ResNet50, ResNet101, InceptionV3, Inception ResNetV2 oraz SqueezeNet. W wyniku uczenia metodą transferu modele mające klasyfikować do ośmiu klas chorobowych (chorób i szkodników drzew owocowych uprawianych w Turcji, m.in. brzoskwini, wiśni, czy moreli) osiągnęły dokładność (ang. *accuracy*) od 89.23% dla Inception ResNetV2 do 96.41% dla VGG16. Wyniki dla modeli ResNet50 i ResNet101 były kolejnymi najlepszymi, z dokładnością na poziomie 95.38%.

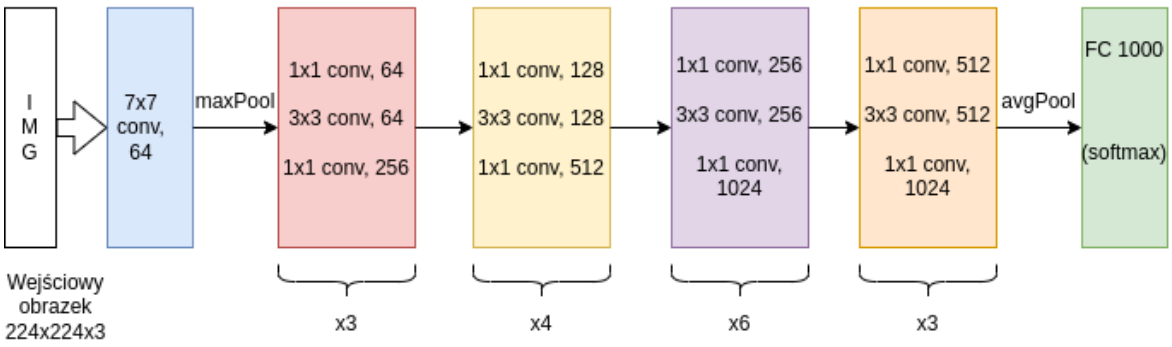
Wybrany został model ResNet50, który jako model stosunkowo nowy osiągał zadowalające wyniki. Na decyzję wpłynęła również jego dostępność w pakiecie Keras Applications oraz mnogość materiałów w internecie.

5.3. ResNet50

ResNet50 to konwolucyjna sieć neuronowa. W swojej publikacji konferencyjnej [11] autorzy opisują stworzony przez nich model ResNet, który uczony na ogólnodostępnym zbiorze ImageNet w 2015 roku zajął pierwsze miejsce w konkursie rozpoznawania obrazu *"ImageNet Large Scale Visual Recognition Challenge"*. ResNet50 jest 50-warstwowym wariantem spośród wielu opisanych w wyżej wymienionej publikacji, który klasyfikuje do 1000 klas ze zbioru ImageNet, w którym zawarte są m.in. zwierzęta, rośliny, czy przedmioty codziennego użytku. Autorzy testowali wiele wariantów sieci, wśród których były zarówno płytsze - posiadające 18 czy 34 warstwy, jak i głębsze - chociażby o liczbie warstw 101 lub 152. Keras Applications, z którego korzystałem, zawiera 3 wersje modelu ResNet: ResNet50, ResNet101 oraz ResNet152.

5.3.1. Struktura modelu

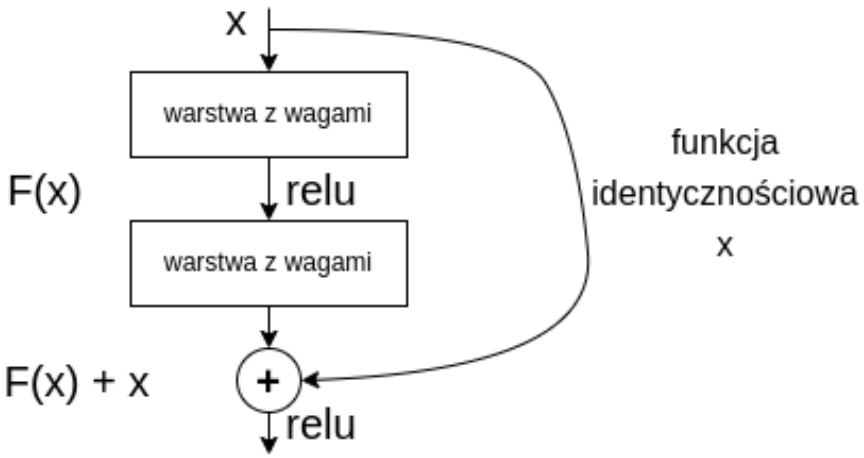
Na rysunku 5.2 przedstawiona została struktura modelu ResNet50. Przyjmowane wejście ma rozmiar 224 x 224 x 3, czyli są to bitmapy o szerokości 224px i wysokości 224px w skali RGB. Na schemacie dla czytelności przedstawiono schemat blokowy, każdy z bloków, poza ostatnim, składa się z warstw konwolucyjnych. Po każdej konwolucji, przed aktywacją, jest stosowana normalizacja wsadowa (ang. *batch normalization*) w celu regularyzacji modelu.



Rysunek 5.2. Struktura modelu ResNet50

W modelu ResNet50 na szczególną uwagę zasługują połączenia skrótowe (ang. *shortcut connections*), które prowadzą z wejścia do warstwy $2n$ do wyjścia warstwy $2n+2$. Użycie połączenia skrótowego przedstawiono na rysunku 5.3, gdzie zostało opisane jako funkcja identycznościowa. Wprowadzenie takich dodatkowych wejść z wcześniejszych wartsw modelu ma na celu, jak piszą autorzy, zapobieganie zanikającym/eksplodującym gradientom. Problem niestabilnego gradientu przejawia się dużymi różnicami w tempie uczenia pomiędzy poszczególnymi warstwami modelu.

Na końcu znajduje się warstwa gęsta o rozmiarze wyjściowym 1000. Implementacja z Kerasa pozwala na podanie innego rozmiaru wyjściowego lub pominięcie tej ostatniej warstwy w tworzonym modelu i skorzystanie z przedostatniej, o strukturze konwolucyjnej.

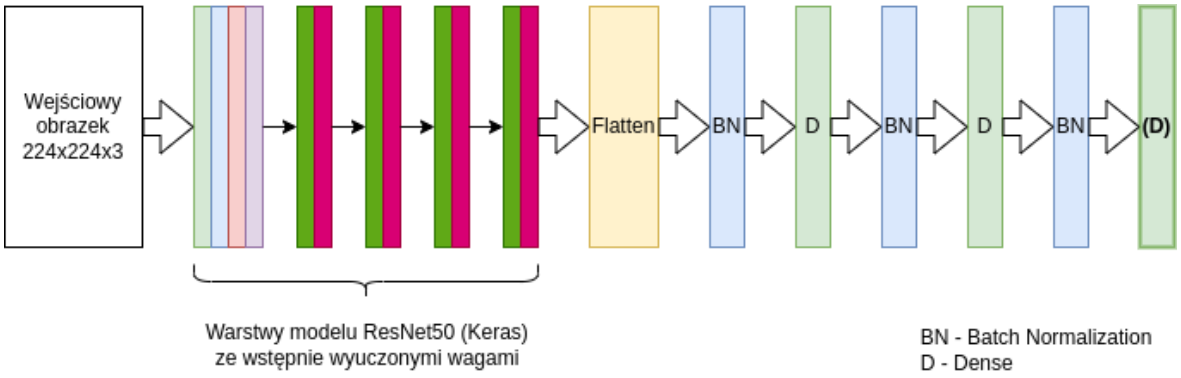


Rysunek 5.3. Schemat bloku w uczeniu rezydualnym (ang. *residual learning*)

5.4. Nadbudowa modelu ResNet50

Przygotowując się do zadania, jakim była nadbudowa modelu ogólnego zastosowania, autor zaczął od zapoznania się z dostępnymi w internecie kursami pokazującymi jak wyuczyć model do własnego zastosowania na bazie ResNet50. Na szczególną uwagę zasłużył zwięzły i konkretny przewodnik [12], który stał się punktem wyjścia do dalszych prac.

5.4.1. Pierwsza wersja modelu



Rysunek 5.4. Schemat pierwszej wersji modelu do klasyfikacji zdjęć liści

W pierwszym podejściu wykorzystano trzy typy warstw: spłaszczającą (`Flatten`), normalizującą (`BatchNormalization`) oraz gęstą (`Dense`). Warstwa spłaszczająca była konieczna, by dane wychodzące z modelu ResNet zostały przekazane w jednowymiarowym formacie do dalszych warstw, które tego wymagały. Warstwy `BatchNormalization` w założeniu miały spełniać funkcję regularyzacji sieci. Na rysunku 5.4 przedstawiony został schemat omawianego modelu. Pierwsza warstwa gęsta modelu miała rozmiar 1024, druga 512, a ostatnia (klasyfikująca) rozmiar 2. Do modelu ResNet (`pretrained`) załadowane zostały standardowe pretrenowane wagi, użyty został model bez ostatniej warstwy (klasyfikującej), a pozostałe z nich zostały zamrożone (`layer.trainable = False`). Stworzony z użyciem Kerasa model przedstawiony jest na listingu 5.4.

Listing 5.4. Pierwsza wersja modelu do klasyfikacji liści

```
model = Sequential()
model.add(pretrained_resnet)
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(1024, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(2, activation='softmax'))
```

Początkowo używanym do uczenia zbiorem danych był podzbiór PlantVillage (klasy `Tomato_healthy` oraz `Tomato_Late_blight`). Oryginalne zdjęcia listków były dynamicznie wzbogacane w trakcie uczenia z użyciem udostępnianej przez Keras klasy `ImageDataGenerator`. Stosowane przekształcenia to odbicia lustrzane, obroty, przybliżenia oraz przesunięcia.

Następnie zbiór uczący rozszerzony został o przetworzone wstępnie zdjęcia z własnego zbioru (również dynamicznie wzbogacane w trakcie uczenia).

Początkowymi parametrami wybranymi do uczenia były optymalizator (ang. *optimizer*) `adam`, funkcja straty (ang. *loss function*) `categorical_crossentropy`, rozmiar wsadu (ang. *batch size*) 8 i liczba epok uczenia (ang. *epoches*) 8. Dla zdjęć ze zbioru PlantVillage model ten zachowywał się dobrze (średnia dokładność klasyfikacji powyżej 95%), ale dla próbek z własnego zbioru klasyfikacje były zdecydowanie mniej trafne. Macierze błędów dla zbiorów testowych złożonych ze zdjęć z PlantVillage i z własnego zbioru zostały przedstawione odpowiednio w tabelach 5.1 i 5.2 (dane dla jednego z wyuczonych modeli). Przyczyną takich rezultatów mógł być odmienny charakter zdjęć z własnego zbioru lub zbytnie dopasowywanie się (ang. *overfitting*) uczonego modelu do próbek ze zbioru PlantVillage.

Tabela 5.1. Macierz błędów dla zbioru testowego złożonego ze zdjęć z PlantVillage (pierwszy model klasyfikacji)

		Prawdziwa klasa	
		chory	zdrowy
Wynik klasyfikacji	chory	144	1
	zdrowy	13	156

Tabela 5.2. Macierz błędów dla zbioru testowego złożonego ze zdjęć z własnego zbioru (pierwszy model klasyfikacji)

		Prawdziwa klasa	
		chory	zdrowy
Wynik klasyfikacji	chory	14	0
	zdrowy	6	20

Dołączenie próbek z własnego zbioru do zbioru uczącego poskutkowało znaczącą poprawą rezultatów. Średnia dokładność klasyfikacji wzrosła z 81.5% do 86%.

5.4.2. Ulepszona wersja modelu

Wyniki uczenia modeli o budowie opisanej powyżej pozostawiły miejsce do poprawy, w związku z czym w dalszej kolejności podjęte zostały próby uczenia i ewaluacji modeli o zmodyfikowanej strukturze.

Ze względu na niewielką liczbę danych do nauki w pierwszej kolejności dokonane zostały zmiany rozmiarów warstw gęstych i zweryfikowano, że znacząca ich redukcja nie wpłynęła negatywnie na jakość klasyfikacji.

Następnie, by zapobiegać nadmiernemu dopasowywaniu się modelu do próbek ze zbioru uczącego, jego struktura została wzbogacona o warstwy porzucenia (`Dropout`). Wprowadzenie tych modyfikacji istotnie poprawiło wyniki klasyfikacji dla zdjęć z własnego zbioru - średnia dokładność klasyfikacji wzrosła z 86% do 88.4%.

Struktura modelu po zmianach została przedstawiona na listingu 5.5.

Listing 5.5. Ulepszona wersja modelu do klasyfikacji liści

```
model = Sequential()
model.add(pretrained_resnet)
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(16, activation='relu'))  # or 32
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(2, activation='softmax'))
```

Sprawdzona została jakość klasyfikacji modeli o rozmiarze przedostatniej warstwy gęstej 16 i 32. W zależności od dobranych parametrów wyniki pozostawały na tym samym poziomie lub okazywały się nieznacznie lepsze dla modeli z rozmiarem warstwy 32. W żadnym z przypadków ta przewaga, mierzona dokładnością (ang. *accuracy*), nie wynosiła więcej niż 0.6%. Przy średniej dokładności na poziomie 88.4% było to niewiele, w związku z czym wybrano mniejszy rozmiar tej warstwy. Celem była dalsza redukcja liczby wag, by pojedyncze z nich zyskały na znaczeniu i miały szansę zostać dobrze dobrane w procesie uczenia na niewielkiej liczbie danych.

5.5. Weryfikacja rozwiązania

5.5.1. Opis procesu weryfikacji

Weryfikacja modelu przeprowadzona została z pomocą skryptu napisanego w języku Python. Zdefiniowane zostały parametry, które dla ustalonej struktury warstwowej były dobierane spośród ustalonego zbioru wartości:

- optymalizator (optimizer)
- funkcja straty (loss function)
- rozmiar wsadów (batch size)
- współczynnik porzucania (dropout rate)
- liczba epok (epoches)

Dla każdej kombinacji wartości powyższych parametrów uczonych było 10 modeli, a rezultaty uśredniane. Klasyfikacje weryfikujące wykonywane były dla dwóch zbiorów danych. Pierwszy to zbiór testowy pochodzący z losowego podziału danych przeznaczonych do uczenia na zbiór treningowy i sprawdzający o rozmiarze około 170 próbek na każdą z klas. Drugi to ustalony zbiór testowy składający się jedynie ze zdjęć z własnego

zbioru o rozmiarze 20 próbek dla każdej z klas. Reprezentacja zdjęć z tego zbioru w danych uczących nie przekraczała 7%. Obliczane miary jakości klasyfikacji to:

- dokładność (ang. *accuracy*)
- wskaźnik F1 dla każdej z klas
- średnia arytmetyczna wskaźników F1
- średnia ważona wskaźników F1

5.5.2. Dobór parametrów

Rozważane wartości parametrów do uczenia częściowo pochodziły z artykułów naukowych, jak np. z [13], a częściowo z informacji znalezionych w internecie. W tym drugim przypadku wyznaczono zbiory rozsądnych wartości dla transfer learningu i spośród nich wybrane zostały wartości możliwie oddalone od siebie. Ustalone zbiory wartości dla parametrów (udostępniane przez bibliotekę Keras):

- optymalizator: `adam, RMSprop`
- funkcja straty: `categorical_crossentropy, binary_crossentropy, hinge`
- rozmiar wsadów: `8, 16, 32`
- współczynnik porzucania[2]: `0.2, 0.35, 0.5`
- liczba epok: `8, 10, 12`

5.5.3. Wyniki

Wyniki zostały podzielone według każdego z parametrów oddzielnie, to znaczy wszystkie wyniki modeli dla każdego zbioru parametrów są grupowane według wartości wybranego parametru i następnie uśredniane. Oprócz tego porównany zostanie wynik uzyskany dla najlepszej kombinacji parametrów z wynikiem uzyskanym dla modelu złożonego z najlepszych parametrów przy rozpatrywaniu ich oddzielnie. Dla czytelności użyty został symbol `[A]` dla oznaczenia wyników dla zbioru testowego uzyskanego z podziału na zbiór uczący i sprawdzający, a symbol `[B]` dla wyników dla ustalonego zbioru złożonego wyłącznie z próbek z własnego zbioru danych.

Tabela 5.3. Porównanie optymalizatorów

Miara	**adam**	**RMSprop**
[A] accuracy	0.961	**0.966**
[B] accuracy	0.868	**0.895**
[B] F1-score zdrowy	0.868	**0.897**
[B] F1-score chory	0.859	**0.891**

[2] Ten współczynnik w modelu przedstawionym wcześniej był ustalony dla obu warstw na 0.5 i to on był modyfikowany (dla obu warstw była podawana ta sama wartość)

Jak widać w tabeli 5.3, oba sprawdzane optymalizatory pozwalały na osiągnięcie satysfakcjonujących wyników, jednak średnio `RMSprop` działał lepiej.

Tabela 5.4. Porównanie liczby epok uczenia

Miara	**8 epok**	**10 epok**	**12 epok**
[A] accuracy	0.962	0.964	**0.965**
[B] accuracy	0.88	0.881	**0.884**
[B] F1-score zdrowy	**0.883**	0.882	0.881
[B] F1-score chory	0.869	0.872	**0.881**

Z danych ukazanych w tabeli 5.4 wynika, że wraz ze wzrostem liczby epok uczenia trafność klasyfikacji się poprawia. Można też zauważyć, że wskaźnik F1-score dla klasy liści zdrowych spada nieznacznie (o 0.002) przy zwiększeniu liczby epok z 8 do 12, ale ten sam wskaźnik dla klasy liści chorych rośnie o 0.012. Choć nie jest to spektakularny wzrost, to jednak jednak jest sześciokrotnie większy od spadku F1-score dla klasy liści zdrowych. Co więcej, dla modeli uczonych przez 12 epok średnia wartość wskaźników dla obu klas jest równa, co jest pożądanym wynikiem.

Tabela 5.5. Porównanie funkcji straty

Miara	**binary_crossentropy**	**categorical_crossentropy**	**hinge**
[A] accuracy	0.963	0.963	**0.964**
[B] accuracy	0.872	0.879	**0.884**
[B] F1-score zdrowy	0.862	0.869	**0.895**
[B] F1-score chory	0.875	**0.882**	0.867

W tabeli 5.5 pokazane zostały uśrednione wyniki klasyfikacji dla trenowanych modeli z różnymi funkcjami straty. W tym miejscu konieczne jest małe doprecyzowanie: modele z funkcją straty `hinge` miały dobraną inną funkcję aktywacji na ostatniej warstwie (klasyfikującej) - `tanh` zamiast używanej dla wszystkich pozostałych modeli funkcji `softmax`. Ta zmiana podyktowana była tym, by wyjściowa wartość z modelu była w przedziale $[-1,1]$. Funkcja `hinge` średnio zachowywała się najlepiej, biorąc pod uwagę miarę dokładności. Przyglądając się jednak bliżej wskaźnikom F1-score, można dostrzec że dla funkcji `categorical_crossentropy` klasyfikacja chorych próbek była istotnie lepsza, a wartości tych wskaźników dla tej funkcji straty dla obu klas były bardziej zbliżone: różnica 0.013, podczas gdy dla funkcji `hinge` różnica wyniosła 0.028.

Tabela 5.6. Porównanie rozmiarów wsadu

Miara	**8**	**16**	**32**
[A] accuracy	0.963	0.962	**0.964**
[B] accuracy	**0.882**	0.865	0.82
[B] F1-score zdrowy	**0.882**	0.863	0.82
[B] F1-score chory	**0.875**	0.858	0.816

Przyglądając się danym w tabeli 5.6, można zauważyć, że średnia dokładność klasyfikacji dla zbioru `[A]` była bardzo zbliżona dla weryfikowanych rozmiarów wsadu, wręcz na granicy błędu statystycznego. Dla zbioru `[B]` wyniki klasyfikacji ulegają pogorszeniu wraz ze zwiększaniem rozmiaru wsadu. Może być to związane z niewielkim rozmiarem tego zbioru, co jednocześnie zadecydowało o uznaniu rozmiaru 8 jako najbardziej odpowiedniego do dalszych badań.

Tabela 5.7. Porównanie współczynnika porzucania

Miara	**0.2**	**0.35**	**0.5**
[A] accuracy	0.961	0.963	**0.966**
[B] accuracy	0.878	0.881	**0.885**
[B] F1-score zdrowy	0.868	0.88	**0.896**
[B] F1-score chory	**0.882**	0.874	0.868

Z danych w tabeli 5.7 można wnioskować, że wyższa wartość współczynnika porzucania dawała średnio w wyniku wyższą dokładność klasyfikacji. Niestety, pomimo rosnącej dokładności, wskaźnik F1-score dla klasy liści chorych maleje.

Tabela 5.8. Porównanie najlepszych zbiorów parametrów rozumianych na dwa sposoby

Miara	**Zbiór X**	**Zbiór Y**
[A] accuracy	0.97	0.968
[B] accuracy	0.918	0.89
[B] F1-score zdrowy	0.922	0.901
[B] F1-score chory	0.912	0.875

Na koniec, co zostało przedstawione w tabeli 5.8, porównano wyniki dla dwóch zbiorów parametrów:

- zbiór X: optymalizator `RMSprop`, funkcja straty `category_crossentropy`, rozmiar wsadu 8, liczba epok 8, współczynnik porzucania 0.5
- zbiór Y: optymalizator `RMSprop`, funkcja straty `hinge`, rozmiar wsadu 8, liczba epok 12, współczynnik porzucania 0.5

Pierwszy z nich (X) to kombinacja parametrów, dla której modele uzyskiwały najlepsze rezultaty klasyfikacji. Drugi zbiór (Y) to parametry uznane za najbardziej obiecujące w wyniku dokonywanych w tym podrozdziale analiz. Jak nietrudno zauważyć, 3 z 5 parametrów mają takie same wartości, a różnica występuje w liczbie epok uczenia oraz funkcji straty.

Dodatkowo w tabeli 5.9 zamieściłem przedziały średnich wartości miar uzyskanych dla wszystkich rozważanych kombinacji parametrów.

Tabela 5.9. Przedziały wartości uzyskanych dla wszystkich kombinacji parametrów

Miara	**Min**	**Max**
[A] accuracy	0.952	0.97
[B] accuracy	0.823	0.918
[B] F1-score zdrowy	0.8	0.922
[B] F1-score chory	0.766	0.912

6. Rozszerzenie modelu klasyfikującego liście o wejście z informacją o danych środowiskowych

6.1. Sposób realizacji

W tym podrozdziale opisany zostanie sposób realizacji rozszerzenia modelu z poprzedniego rozdziału o dodatkowe wejście.

Struktura dotychczasowego modelu klasyfikującego liście została zmieniona w taki sposób, by przyjmował dodatkowe wejście. Taki model może być douczany w podobny sposób jak to miało miejsce wcześniej, ale tym razem danymi wejściowymi do jego nauki nie będą same zdjęcia, tylko pary złożone ze zdjęcia i danych środowiskowych. Wprowadzenie innych danych niż zdjęcie do modelu ResNet nie ma większego sensu, ale takie wejście z powodzeniem może być wprowadzone do warstwy gęstej znajdującej się już za warstwami ResNetu.

Największym problemem z wytrenowaniem modelu rozszerzonego był bardzo mały zbiór uczący. Dla zdjęć ze zbioru PlantVillage nie ma dostępnych danych środowiskowych, w związku z czym uczenie musiało odbywać się wyłącznie na próbkach z własnego zbioru danych. Zbiór ten jest co najmniej rząd wielkości mniejszy od zbioru PlantVillage, który był dostępny do uczenia modelu z rozdziału 5. Aby przezwyciężyć te trudności, starano się wybrać podejście, które miało szanse zadziałać. Próbek było zbyt mało, by uczyć na nich model w podobny sposób. W związku z tym została podjęta decyzja, by spróbować poprawić wyniki wyuczonego na samych liściach modelu poprzez douczenie kolejną porcją próbek zawierających dane środowiskowe. W tym podejściu można było też łatwo regulować stopień douczania modelu - od najmniejszych porcji do wieloepokowych treningów. Tym samym można było mieć też wpływ na stopień pogorszenia lub poprawy jakości klasyfikacji dokonywanych przez model, a co istotne punktem wyjścia był w oczywisty sposób wynik modelu o strukturze z poprzedniego rozdziału.

6.2. Uwzględnienie danych środowiskowych w zbiorze uczącym

6.2.1. Wybór metody

Opisany w poprzednim podrozdziale sposób realizacji to jeszcze nie wszystko. Nadal należało ustalić sposób, czy też format, w jakim dane o środowisku będą trafiać do modelu. Gwoli przypomnienia, dane dostępne w zbiorze to wartości parametrów środowiskowych umiejscowione na osi czasu z rozdzielczością piętnastominutową. Dostępne parametry to: temperatura i wilgotność powietrza, ilość opadów atmosferycznych oraz nasłonecznienie.

Zbiór uczący złożony ze zdjęć z własnego zbioru i odpowiadających im danych środowiskowych był stosunkowo niewielki. Początkowe plany, zakładające uczenie modelu wielowartościowym wektorem danych o środowisku, musiały zostać odsunięte na bok. Z pomocą przyszły gotowe modele ryzyka zakażenia roślin dla konkretnych chorób, gdzie zmiennymi były właśnie syntetyczne indeksy bieżących warunków środowiskowych. Niewątpliwą zaletą było w tym wypadku ich dopracowanie na bazie wieloletnich badań i

doświadczeń. Taki model daje w wyniku jedną liczbę będącą wskaźnikiem ryzyka infekcji, co jest wartością łatwo interpretowalną.

6.2.2. Wybór modelu ryzyka zachorowania

Przejrzane zostały różne modele ryzyka zachorowania dla roślin uprawnych. W przypadku wybranej choroby - zarazy ziemniaka - zawsze we wzorach pojawiały się wartości związane z temperaturą powietrza, jego wilgotnością oraz opadami atmosferycznymi. Wybrano model opisany na stronie Uniwersytetu Kaliforni [14], który korzysta dokładnie z tych trzech parametrów środowiskowych. Poniżej przytoczony został wzór służący do obliczania wskaźnika ryzyka zachorowania (IPI - ang. *Infection Potential Index*):

$$IPI = max(T_{index} \cdot RH_{index}, T_{index} \cdot R_{index}) \tag{6.1}$$

gdzie

$$T_{index} = (-2.19247 + 0.259906 \cdot T - 0.000139 \cdot T^3 - 6.095832 \cdot 10^{-6} \cdot T^4) \cdot Fc \tag{6.2}$$

$Fc = 0.35 + 0.05 \cdot T_{min}$ (współczynnik korekcji)
T - średnia dzienna temperatura [$^\circ C$]
T_{min} - minimalna dzienna temperatura [$^\circ C$]

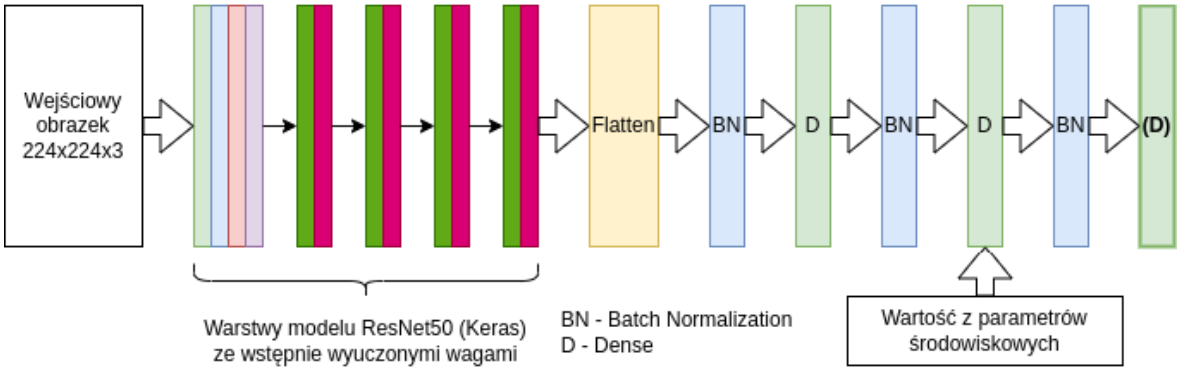
$$RH_{index} = -34.9972725 + 0.751 \cdot RH - 0.003909 \cdot RH^2 \tag{6.3}$$

RH - średnia dzienna względna wilgotność powietrza [%]

$$R_{index} = 0.006667 + 0.194405 \cdot R + 0.0002239 \cdot R^2 \tag{6.4}$$

R - suma opadów atmosferycznych z ostatnich 48 godzin [mm]

Wśród innych modeli można było napotkać takie zmienne jak GAP (ang. *Good Agriculture Practices*) uwzględniony we wzorze w książce [15], czyli wskaźnik odzwierciedlający stosowanie dobrych praktyk ogrodniczych w trakcie uprawy (jak np. zachowanie odpowiedniej odległości między roślinami). Zamieszczone tam modele co prawda dotyczą choroby zarazy ziemniaka dla ziemniaka i bardzo podobnej do zarazy ziemniaka choroby pomidora, ale nietrudno zauważyć znaczące podobieństwo pomiędzy tymi dwoma wzorami. Z kolei we wzorze przedstawionym w artykule [16] pojawiają się dodatkowe parametry: nasłonecznienie (promieniowanie słoneczne) oraz zawilżenie liścia (ang. *leaf wetness*).



Rysunek 6.1. Struktura modelu rozszerzonego o dodatkowe wejście (podejście pierwsze)

6.3. Pierwsze podejście

6.3.1. Struktura modelu

W pierwszym podejściu wartość wskaźnika ryzyka zachorowania została włączona do modelu jako dodatkowe wejście do ostatniej ukrytej warstwy gęstej (rozmiar 16), co przedstawione zostało na rys. 6.1. Tak więc w nowym modelu do warstwy `Dense(16)` prowadziło 129 wejść - 128 z poprzedniej warstwy gęstej w modelu klasyfikacji liści oraz jedno dodatkowe, które miało odzwierciedlać warunki środowiskowe. Zostało to dokonane z pomocą dostępnej w Kerasie warstwy `concatenate`. Konieczne było użycie udostępnianego funkcyjnego API, by zbudować nowy model, co widać na listingu 6.6.

Listing 6.6. Pierwsza wersja modelu z dodatkowym wejściem

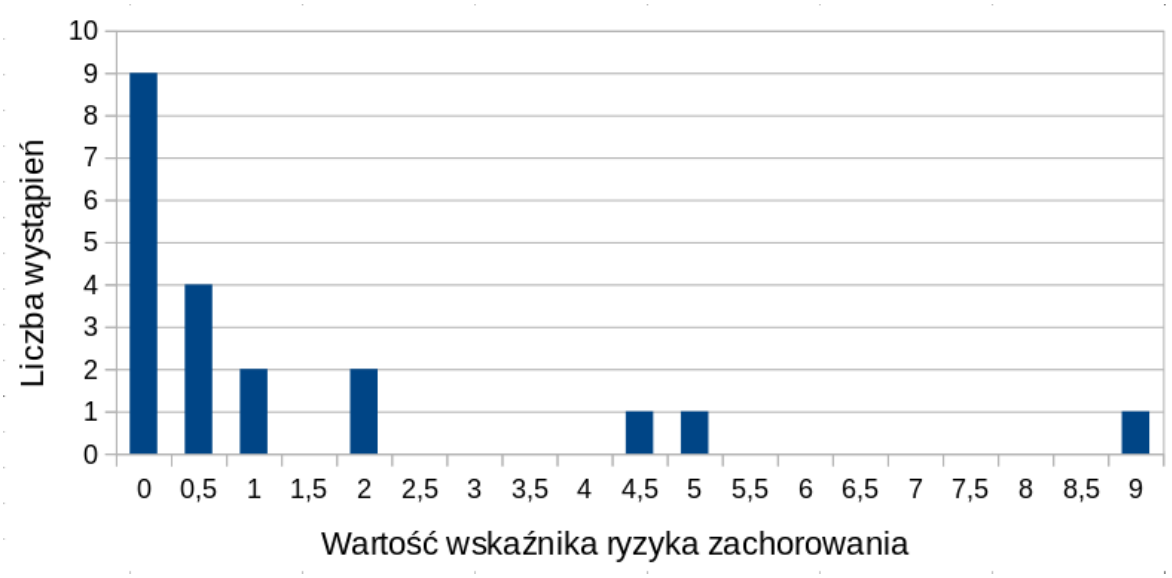
```
model = Flatten()(pretrained_resnet.output)
model = BatchNormalization()(model)
model = Dense(128, activation='relu')(model)
model = Dropout(dropout)(model)
model = BatchNormalization()(model)

_model = Model(inputs=pretrained.input, outputs=model)
data_input = Input(shape=(1,))
data_model = Dense(1)(data_input)
_data_model = Model(inputs=data_input, outputs=data_model)

merge = concatenate([_model.output, _data_model.output])

model2 = Dense(16, activation='relu')(merge)
model2 = Dropout(dropout)(model2)
model2 = BatchNormalization()(model2)
model2 = Dense(num_classes, activation='softmax')(model2)
```

```
full_model = Model(inputs=[_model.input,
                           _data_model.input],
                   outputs=[model2])
```

Parametry tego modelu, takie jak optymalizator, funkcja straty, współczynnik porzucania czy rozmiar wsadu zostały ustawione jak w przypadku modelu o najlepszych wynikach z rozdziału 5, tzn. optymalizator `RMSprop`, funkcja straty `category_crossentropy`, współczynnik porzucania 0.5, rozmiar wsadu 8 i liczba epok trenowania 8.

6.3.2. Wstępne uczenie

Próbek z przypisanymi do nich wartościami opisującymi stan środowiska naturalnego jest bardzo mało, więc model o zmienionej strukturze należy najpierw wytrenować na samych zdjęciach listków pomidorów, jak to miało miejsce w poprzednim rozdziale. Model wymaga jednak wprowadzenia wartości na dodatkowe wejście, w związku z czym zdecydowałem, by na dodatkowym wejściu podawać szum o rozkładzie wartości zbliżonym do rozkładu wartości rzeczywistych, które zostały wyliczone dla posiadanego własnego zbioru danych. Histogram tych wartości został przedstawiony na rysunku 6.2.



Rysunek 6.2. Histogram wartości wskaźnika ryzyka zachorowania

6.3.3. Wybór parametrów

Część parametrów modelu została już ustalona wcześniej - na etapie jego budowy i wstępnego uczenia. Nadal jednak należy zdecydować jak długo douczać model danymi z prawdziwymi wartościami ryzyka zachorowania (liczba epok) oraz w jaki sposób obliczać wprowadzaną do modelu dodatkową wartość. Ze względu na niewielką liczbę danych liczba epok nie powinna być zbyt duża, w związku z czym sprawdzono jak zmienia się działanie modelu przy liczbie epok 3, 5, 8. W przypadku wartości ryzyka zachorowania,

wprowadzanej dodatkowo do modelu, wzór do jego obliczenia został już wcześniej wybrany. Kwestią otwartą pozostawało nadal, dla jakiego punktu w czasie powinna być wyliczana ta wartość - dla czasu wykonania zdjęcia, czy dla poprzedniego dnia, czy w jeszcze inny sposób. Zważając na to, że od czasu infekcji do pierwszych widocznych objawów może minąć nawet parę dni, to pochylenie się nad tym problemem zdaje się jak najbardziej zasadne. Zweryfikowano kilka sposobów obliczania tej wartości:

- wartość wskaźnika ryzyka zachorowania dla czasu wykonania zdjęcia (D)
- wartość wskaźnika ryzyka zachorowania dla tej samej godziny dzień wcześniej (D-1)
- wartość wskaźnika ryzyka zachorowania dla tej samej godziny dwa dni wcześniej (D-2)
- średnia arytmetyczna z wartości wskaźnika ryzyka zachorowania dla D, D-1 i D-2
- suma wartości wskaźnika ryzyka zachorowania dla D, D-1 i D-2

6.3.4. Weryfikacja rozwiązania

Każda kombinacja parametrów została zweryfikowana na 10 wytrenowanych z ich użyciem modelach, a wyniki modeli zostały uśrednione. Użyty zbiór danych zawierał po 87 próbek dla każdej z dwóch klas: zdrowy oraz chory (lekko chory). Zbiór ten był w sposób losowy dzielony przed uczeniem każdego z modeli na uczący (80% próbek) i testowy (20%). Tak samo, jak w przypadku modelu klasyfikacji samych zdjęć, wejściowe dane były dynamicznie wzbogacane (por. podrozdział 5.4.1).

Miary przedstawione w tabelach mają dopiski "przed"lub "po", co oznacza odpowiednio średnie wyniki uzyskane przed douczaniem prawdziwymi wartościami wskaźnika ryzyka zachorowania i po nim. Istotnie gorsze wyniki klasyfikacji modelu wyjściowego ("przed") w stosunku do tych z rozdziału 5 wynikają z tego, że zamiast zdjęć liści definitywnie chorych w procesie douczania i ewaluacji wykorzystywane są zdjęcia liści z nieznacznymi objawami (początkowymi).

Tabela 6.1. Porównanie liczby epok uczenia

Miara	**3 epoki**	**5 epok**	**8 epok**
accuracy (przed)	0.654	0.631	0.653
accuracy (po)	0.681	0.668	0.689
accuracy (zmiana [%])	+4.1	+5.9	+5.5
F1-score zdrowy (przed)	0.637	0.613	0.64
F1-score zdrowy (po)	0.647	0.627	0.648
F1-score zdrowy (zmiana [%])	+1.6	+2.3	+1.3
F1-score chory (przed)	0.667	0.645	0.661
F1-score chory (po)	0.706	0.697	0.717
F1-score chory (zmiana [%])	+5.8	+8.1	+8.5

Jak wynika z tabeli 6.1, już 3 epoki douczania modelu skutkują zauważalnym poprawieniem wyników klasyfikacji. Szczególną poprawę, wraz ze wzrostem liczby epok, widać

we wskaźniku F1-score dla klasy liści chorych. Większą poprawę F1-score dla klasy liści w przypadku 5 epok można tłumaczyć stosunkowo niskim poziomem tego wskaźnika przed douczaniem.

Tabela 6.2. Porównanie rodzajów dodatkowego wejścia

Miara	**r(D)**[3]	**r(D-1)**	**r(D-2)**	**średnia**[4]	**suma**[5]
accuracy (przed)	0.648	0.655	0.639	0.643	0.645
accuracy (po)	0.67	0.682	0.676	0.676	0.691
accuracy (zmiana [%])	+3.4	+4.1	+5.8	+5.1	+7.1
F1-score zdrowy (przed)	0.624	0.638	0.62	0.627	0.64
F1-score zdrowy (po)	0.629	0.643	0.644	0.639	0.648
F1-score zdrowy (zmiana [%])	+0.8	+0.8	+3.9	+1.9	+1.3
F1-score chory (przed)	0.667	0.667	0.653	0.655	0.648
F1-score chory (po)	0.698	0.709	0.7	0.703	0.722
F1-score chory (zmiana [%])	+4.6	+6.3	+7.2	+7.3	+11.4

W tabeli 6.2 przedstawione zostało porównanie wyników klasyfikacji ze względu na rodzaj wprowadzonej informacji o ryzyku zachorowania. Jak widać, dla każdej z funkcji (w ujęciu średnim) douczanie poskutkowało pewną poprawą jakości klasyfikacji modelu. Największe polepszenie klasyfikacji osiągnięto dla sumy wartości wskaźnika ryzyka zachorowania dla czasu wykonania zdjęcia i dwóch poprzedzających dni, a najmniejsze dla wskaźnika ryzyka zachorowania liczonego dla czasu wykonania zdjęcia.

Suma i średnia arytmetyczna niosą w gruncie rzeczy bardzo podobną informację, a różnica pomiędzy wynikami dla tych dwóch wejść jest znacząca. Może to wskazywać na to, że douczanie modelu nie było wystarczające, a było ograniczone przez wielkość zbioru uczącego.

Dodatkowo policzone zostały średnie F1-score, łącząc w ten sposób wartości dla obu klas w jedną miarę. W ten sposób oceniono każdą z 15 sprawdzanych kombinacji parametrów. We wszystkich przypadkach uzyskano poprawę względem wyjściowej wartości wskaźnika (przed douczaniem).

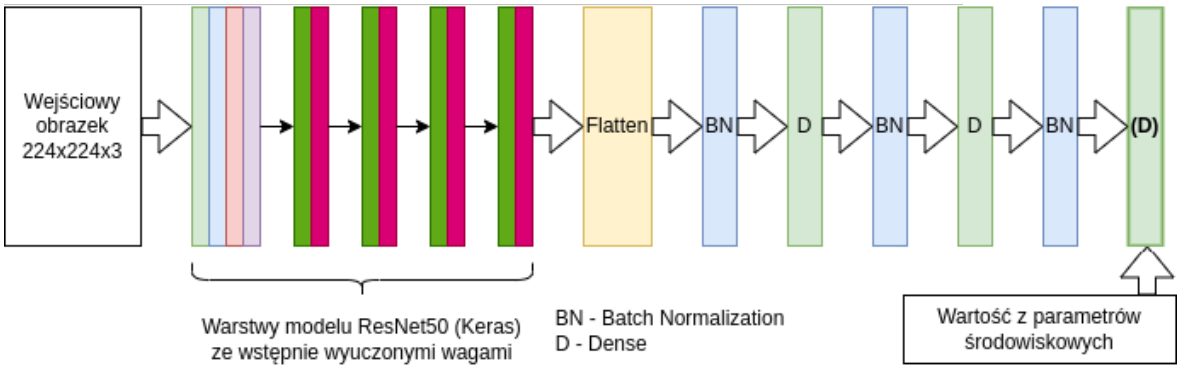
Największą poprawę osiągnięto przy użyciu jako wejścia wartości wskaźnika ryzyka zachorowania policzonej dla tej samej godziny 2 dni wcześniej i uczenia przez 8 epok. Wskaźnik F1-score po douczaniu wzrósł średnio o 8.3% (z wartości 0.638 do 0.691).

6.4. Drugie podejście

6.4.1. Struktura modelu

W drugim podejściu sprawdzono jakie wyniki uda się uzyskać, włączając dodatkowe wejście do ostatniej warstwy - klasyfikującej - zamiast do ostatniej ukrytej warstwy gęstej. Tym razem różnica względem modelu z poprzedniego rozdziału to dodatkowe, siedemnaste wejście do warstwy klasyfikującej (`Dense(2)`). Tak więc do ostatniej warstwy modelu

[3] r - funkcja wskaźnika ryzyka zachorowania, D - dzień wykonania zdjęcia
[4] średnia arytmetyczna dla wartości $r(D)$, $r(D-1)$ i $r(D-2)$
[5] $r(D)+r(D-1)+r(D-2)$



Rysunek 6.3. Struktura modelu rozszerzonego o dodatkowe wejście (podejście drugie)

wchodzi 16 wejść powstałych z przejścia zdjęcia przez pierwszą część modelu oraz (jak w pierwszym podejściu) jedno dodatkowe, które miało odzwierciedlać warunki środowiskowe. Zmieniony model wygląda tak, jak zostało to przedstawione na rysunku 6.3.

Zarówno parametry modelu, jak i sposób wstępnego uczenia, nie uległy zmianie i są identyczne jak w podejściu pierwszym.

6.4.2. Weryfikacja rozwiązania

Weryfikacja rozwiązania została przeprowadzona w taki sam sposób, w jaki zostało to opisane w podrozdziale o pierwszym podejściu.

Tabela 6.3. Porównanie liczby epok uczenia

Miara	**3 epoki**	**5 epok**	**8 epok**
accuracy (przed)	0.685	0.695	0.686
accuracy (po)	0.699	0.695	0.684
accuracy (zmiana [%])	+2.0	+0.0	-0.3
F1-score zdrowy (przed)	0.659	0.683	0.662
F1-score zdrowy (po)	0.647	0.629	0.591
F1-score zdrowy (zmiana [%])	-1.8	-7.9	-10.7
F1-score chory (przed)	0.705	0.702	0.705
F1-score chory (po)	0.735	0.736	0.740
F1-score chory (zmiana [%])	+4.3	+4.8	+5.0

Jak wynika z danych w tabeli 6.3, najlepsze średnie wyniki klasyfikacji zostały osiągnięte dla 3 epok uczenia. Przy tym wyborze średnia poprawa dokładności wyniosła 2%, a wartość wskaźnika F1-score dla klasy liści chorych wzrasta o 4.3%. Niestety maleje wartość wskaźnika F1-score dla klasy liści zdrowych. Dla uczenia przez 3 epoki spadek wynosi 1.8%, ale dla 8 epok jest to aż 10.7%.

Na podstawie danych przedstawionych w tabeli 6.4 można wnioskować, że żadna z funkcji nie pomaga w istotnej poprawie wyników klasyfikacji w modelu o nowej strukturze.

[6] r - funkcja wskaźnika ryzyka zachorowania, D - dzień wykonania zdjęcia
[7] średnia arytmetyczna dla wartości $r(D)$, $r(D-1)$ i $r(D-2)$
[8] $r(D)+r(D-1)+r(D-2)$

Tabela 6.4. Porównanie rodzajów dodatkowego wejścia

Miara	**r(D)**[6]	**r(D-1)**	**r(D-2)**	**średnia**[7]	**suma**[8]
accuracy (przed)	0.686	0.692	0.688	0.697	0.678
accuracy (po)	0.685	0.701	0.685	0.710	0.683
accuracy (zmiana [%])	-0.1	-1.3	-0.4	+1.9	+0.7
F1-score zdrowy (przed)	0.67	0.672	0.663	0.676	0.659
F1-score zdrowy (po)	0.613	0.627	0.607	0.65	0.614
F1-score zdrowy (zmiana [%])	-8.5	-6.7	-8.4	-3.8	-6.8
F1-score chory (przed)	0.699	0.708	0.709	0.711	0.693
F1-score chory (po)	0.729	0.747	0.734	0.749	0.726
F1-score chory (zmiana [%])	+4.3	+5.5	+3.5	+5.3	+4.8

Najlepiej wypadła średnia arytmetyczna wartości wskaźnika ryzyka zachorowania, gdzie douczane modele średnio o 1.9% dokładniej (accuracy) klasyfikowały próbki. Niestety i w tym przypadku wzrost F1-score dla klasy liści chorych (+5.3%) został okupiony spadkiem F1-score dla klasy liści zdrowych (-3.8%).

Policzono średnie F1-score, by ocenić każdą z 15 sprawdzanych kombinacji parametrów. W 7 przypadkach uzyskano poprawę, a w 8 jakość klasyfikacji spadła.

Największą poprawę osiągnięto przy użyciu sumy jako wejścia i uczenia przez 3 epoki. Wskaźnik F1-score po douczaniu wzrósł średnio o 3%.

Największe pogorszenie wystąpiło dla zastosowania funkcji ryzyka zachorowania obliczonego dla chwili wykonania zdjęcia oraz przy uczeniu przez 8 epok. Wskaźnik F1-score po douczaniu spadł średnio o 4.8%.

6.5. Podsumowanie

W tym rozdziale szczegółowo opisane zostały struktury dwóch modeli klasyfikacji zdjęć rozszerzonych o dodatkowe wejście ze wskaźnikiem obliczanym na podstawie danych środowiskowych oraz została zweryfikowana ich zdolność do podniesienia jakości klasyfikacji roślin na zdrowe i chore we wczesnym stadium choroby.

W przypadku pierwszego podejścia uzyskane wyniki są obiecujące. Średnia poprawa wskaźnika F1-score dla liści zdrowych jest niewielka, ale dla liści chorych jest już znacząca. Pozwala to domniemywać, że wskaźnik ryzyka zachorowania odgrywa zauważalną rolę w zwiększeniu wykrywalności zachorowania. Niemniej jednak hipoteza ta wymaga dokładniejszej weryfikacji z użyciem większego zbioru danych.

7. Podsumowanie pracy

7.1. Podsumowanie wykonanej pracy

W pierwszej części pracy opisany został proces tworzenia i uczenia modelu do klasyfikacji zdjęć liści wybranej rośliny - pomidora - na zdrowe i zarażone patogenem *Phytophthora infestans*, wywołującym chorobę zwaną zarazą ziemniaka (ang. *late blight*).

W rozdziale 4 przedstawione zostały metody przetwarzania wstępnego, których zastosowanie było konieczne do stworzenia użytecznego zbioru danych z własnych zdjęć liści. Część z nich udało się bez większych przeszkód zautomatyzować, w innych przypadkach okazało się to trudniejsze do osiągnięcia.

Rozdział 5 poświęcony został tworzeniu i trenowaniu modelu do klasyfikacji zdjęć liści. Opisany został koncept transfer learningu i jego zastosowanie przy tworzeniu własnego modelu na bazie wyuczonego wstępnie modelu ResNet50. Dla publicznie dostępnego zbioru PlantVillage osiągnięcie satysfakcjonujących wyników nie było trudne przy uczeniu i testowaniu modelu tylko na próbkach z tego zbioru (podzielonego). Średnia dokładność klasyfikacji wynosiła powyżej 95%. Dla próbek z własnego zbioru, po modyfikacji struktury i poszukiwaniu optymalnych parametrów, udało się osiągnąć średnią dokładność na poziomie 91.8%.

W rozdziale 6 opisane zostały próby rozszerzenia modelu klasyfikacji zdjęć liści o wymiar danych środowiskowych. Ze względu na bardzo małą liczbę próbek w zbiorze do uczenia zostały one wprowadzone do modelu pod postacią jednej wartości - wskaźnika ryzyka zachorowania, wyliczanego na podstawie wzoru ze znalezionego modelu prognostycznego. W pierwszym podejściu uzyskano pewną poprawę w wykrywaniu liści chorych (lekko chorych), zaś w drugim dokładność klasyfikacji modelu w procesie uczenia szybko ulegała pogorszeniu. W pierwszym podejściu wzrost wskaźnika średniego F1-score wyniósł średnio 8.3% (z wartości 0.638 osiąganej przed douczaniem do 0.691).

7.2. Wnioski

W pracy udało się, z wykorzystaniem modelu ResNet50 i zbioru danych PlantVillage, osiągnąć zadowalające wyniki klasyfikacji. Następnie z powodzeniem przetworzone wstępnie zostały zdjęcia liści pomidorów z własnego zbioru uczącego w taki sposób, by były poprawnie klasyfikowane przez wyuczony model. Struktura modelu została ulepszona i finalnie dla zdjęć z własnego zbioru udało się uzyskać niewiele gorszą trafność klasyfikacji niż dla zdjęć z PlantVillage. Przy zastosowaniu wskaźnika ryzyka zachorowania jako dodatkowego wejścia do modelu udało się w jednym z podejść uzyskać zauważalną poprawę jakości klasyfikacji, szczególnie dla klasy liści (lekko) chorych.

Uzyskane wyniki modelu rozszerzonego o wymiar danych środowiskowych należy potwierdzić przy użyciu większego zbioru danych. Wtedy zasadne będzie również poszukiwanie optymalnych parametrów do uczenia.

7.3. Możliwości rozwoju

7.3.1. Przetwarzanie wstępne

Automatyzacji przetwarzania wstępnego nie zostało w tej pracy poświęcone aż tyle czasu, jak jest to możliwe. Pełna automatyzacja tego procesu istotnie ułatwiłaby stworzenie dużego zbioru danych. Co więcej, mogła by być wykorzystana w docelowym rozwiązaniu, gdzie obraz z kamery w miejscu uprawy byłby automatycznie przetwarzany i, wraz z danymi z czujników, podawany jako wejście do modelu stwierdzającego chorobę lub duże ryzyko jej wystąpienia. Podkładanie tła przed wykonaniem zdjęcia wydaje się trudne w realizacji automatycznej, ale może użycie aparatu o małej głębi ostrości pozwoliłoby na uzyskanie zdjęcia, z którego wyodrębnienie liścia byłoby prostym zadaniem.

7.3.2. Model klasyfikacji zdjęć

Mając duży zbiór danych można próbować innych podejść, niż zademonstrowane w tej pracy. Jednym z pomysłów jest ocena stopnia nasilenia objawów - od liści z delikatnymi zmianami do liści w całości zmienionych przez chorobę - i podział na kilka klas chorobowych ze względu na tę ocenę lub uciąglenie wyjścia modelu (wskaźnik zachorowania wyznaczany w modelu regresyjnym). Innym pomysłem jest zmiana obiektu klasyfikacji - zamiast pojedynczych listków można użyć całych liści złożonych, krzaków lub nawet zdjęć całych stanowisk uprawnych. Jeszcze inne możliwe podejście to zastosowanie więcej niż jednego zdjęcia na wejściu do modelu.

7.3.3. Model rozszerzony o dane środowiskowe

Będąc w posiadaniu dużego zbioru danych zawierającego zdjęcia i powiązane z nimi dane o środowisku, można w pierwszej kolejności zweryfikować wyniki uzyskane dla stworzonych modeli w rozdziale 6. Dla odpowiednio dużych danych można próbować tworzyć modele o innych wejściach dodatkowych, np. wektorach zawierających zagregowane dane o wilgotności i temperaturze powietrza, czy ilości opadów atmosferycznych w dniach poprzedzających wykonanie zdjęcia.

7.3.4. Model prognozy zachorowania

Bardzo użytecznym byłby model stwierdzający zachorowanie jeszcze przed wystąpieniem oczywistych objawów widocznych gołym okiem. Model taki, mający na wejściu pewne dane o środowisku oraz zdjęcia roślin, mógłby określać ryzyko, że dana roślina jest chora lub wkrótce zachoruje. Dane środowiskowe miały by pełnić podobną funkcję jak w modelu opisanym wzorem 6.1, czyli określać ryzyko zachorowania. Jeśli zaś chodzi o zdjęcia, to można mieć nadzieję na wykrycie przez model innych zależności między stanem liścia widocznym na zdjęciu a ryzykiem zachorowania niż powszechnie znane objawy. Chociażby stan liścia świadczący o pewnym osłabieniu rośliny mógłby wpłynąć na wynik klasyfikacji - jak wiadomo, osłabione rośliny są bardziej podatne na zachorowanie.

Bibliografia

[1] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] *TensorFlow - Wikipedia*, Dostęp zdalny (10.08.2022): https://en.wikipedia.org/wiki/TensorFlow.

[3] *Keras - Wikipedia*, Dostęp zdalny (10.08.2022): https://en.wikipedia.org/wiki/Keras.

[4] *OpenCV - Wikipedia*, Dostęp zdalny (10.08.2022): https://en.wikipedia.org/wiki/OpenCV.

[5] *NumPy - Wikipedia*, Dostęp zdalny (10.08.2022): https://en.wikipedia.org/wiki/NumPy.

[6] D. P. Hughes i M. Salathe, *An open access repository of images on plant health to enable the development of mobile disease diagnostics*, 2015. DOI: 10.48550/ARXIV.1511.08060. adr.: https://arxiv.org/abs/1511.08060.

[7] *OpenCV Docs - Thresholding tutorial*, Dostęp zdalny (10.08.2022): https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html.

[8] *Transfer learning - Wikipedia*, Dostęp zdalny (10.08.2022): https://en.wikipedia.org/wiki/Transfer_learning.

[9] S. Mohanty, D. Hughes i M. Salathe, „Using Deep Learning for Image-Based Plant Disease Detection", *Frontiers in Plant Science*, t. 7, kw. 2016. DOI: 10.3389/fpls.2016.01419.

[10] M. türkoğlu i D. Hanbay, „Plant disease and pest detection using deep learning-based features", *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, t. 27, s. 1636–1651, maj 2019. DOI: 10.3906/elk-1809-181.

[11] K. He, X. Zhang, S. Ren i J. Sun, „Deep Residual Learning for Image Recognition", czer. 2016, s. 770–778. DOI: 10.1109/CVPR.2016.90.

[12] K. C. Aguas, *A guide to transfer learning with Keras using ResNet50*, Dostęp zdalny (10.08.2022): https://medium.com/@kenneth.ca95/a-guide-to-transfer-learning-with-keras-using-resnet50-a81a4a28084b, 2020.

[13] K. Zhang, Q. Wu, A. Liu i X. Meng, „Can Deep Learning Identify Tomato Leaf Disease?", *Advances in Multimedia*, t. 2018, s. 1–10, wrz. 2018. DOI: 10.1155/2018/6710865.

[14] *Tomato Late Blight Model*, Dostęp zdalny (10.08.2022): http://ipm.ucanr.edu/DISEASE/DATABASE/tomatolateblight.html.

[15] K. K. Patel, G. Doctor, A. Patel i P. Lingras, *Soft Computing and Its Engineering Applications: Third International Conference, IcSoftComp 2021, Changa, Anand, India, December 10-11, 2021, Revised Selected Papers*. Berlin: Springer Nature, 2022, s. 321–322.

[16] A.-G. R. Hjelkrem, H. Eikemo, V. H. Le, A. Hermansen i R. Nærstad, „A process-based model to forecast risk of potato late blight in Norway (The Nærstad model): model development, sensitivity analysis and Bayesian calibration", *Ecological Modelling*,

t. 450, s. 109 565, 2021, ISSN: 0304-3800. DOI: https://doi.org/10.1016/j.ecolmodel.2021.109565. adr.: https://www.sciencedirect.com/science/article/pii/S0304380021001344.

Wykaz symboli i skrótów

API – Application Programming Interface
GPU – Graphics Processing Unit
HSV – Hue, Saturation, Value
IPI – Infection Potential Index
FN – False Negative
FP – False Positive
ResNet – Residual neural network
RGB – Red, Green, Blue
TN – True Negative
TP – True Positive
URL – Uniform Resource Locator

Spis rysunków

Spis tabel

Spis listingów