

Politechnika Warszawska

W Y D Z I A Ł E L E K T R Y C Z N Y



Instytut Sterowania i Elektroniki Przemysłowej

Praca dyplomowa inżynierska

na kierunku Automatyka i Robotyka Stosowana

System oceny czystości chodników wykorzystujący głębokie sieci neuronowe wytrenowane na obrazach wygenerowanych przez sztuczną inteligencję

Michał Janicki

numer albumu 319148

promotor
dr inż. Witold Czajewski

Warszawa 2025

System oceny czystości chodników wykorzystujący głębokie sieci neuronowe wytrenowane na obrazach wygenerowanych przez sztuczną inteligencję

Streszczenie

W niniejszej pracy zaprezentowane zostało podejście do problemu klasyfikacji stopnia zabrudzenia chodników. Do tego celu wykorzystano dwa modele sieci neuronowych: ConvNeXt i ResNet152, a następnie douczono je metodą transfer learning. Jako zbioru danych uczących użyto jedynie zdjęć wygenerowanych przez modele generatywne. Przetestowano pod tym kątem wiele różnych modeli. Najlepiej radzącymi sobie z generacją okazały się modele Stable Diffusion w wersji 1.5 oraz Firefly Image 3. Obrazy generowano metodami image-to-image oraz inpainting. W dalszej części pracy użyto wymienionej bazy danych do uczenia sieci neuronowych klasyfikacji poziomu zabrudzenia chodników z podziałem na trzy klasy: *czyste*, *średnio-zabrudzone* i *bardzo-zabrudzone*. Dane wejściowe podlegały dodatkowo losowym transformacjom w celu powiększenia zbioru trenującego. Dobrano również optymalne hiperparametry nauczania. Model ConvNeXt okazał się lepszy w tym problemie uzyskując dokładność 96,1% na zbiorze walidacyjnym. Uzyskał on również dokładność 92,8% i 85,2% odpowiednio na dwóch użytych zbiorach testowych składających się ze zdjęć rzeczywistych chodników. Wynik ten jest zadowalający i zachęca do dalszych prac związanych z zastosowaniem połączenia generacji zdjęć i problemu klasyfikacji obrazów.

Słowa kluczowe: sztuczne sieci neuronowe, głębokie nauczanie, ConvNeXt, ResNet, modele generatywne, klasyfikacja obrazów

Sidewalk cleanliness assessment system utilizing deep neural networks trained on images generated by artificial intelligence

Abstract

In this thesis an approach to a problem of classification the level of dirtiness of sidewalks has been presented. For this purpose ConvNeXt and ResNet152 models have been used and fine-tuned with the usage of transfer learning. Only images generated with generative models were used as the training data set. Many different models were tested for this purpose. The best performing models were Stable Diffusion version 1.5 and Firefly Image 3. Images were generated using image-to-image and inpainting methods. In the further part of the work, the above-mentioned database was used to train neural networks to classify the level of dirtiness of sidewalks into three classes: *clean*, *medium-dirty* and *very dirty*. During the training, input data was additionally going under random transformations in purpose of data augmentation. Optimal learning hyperparameters have been selected. The ConvNeXt model turned out to be better in this problem, achieving the accuracy of 96,1% on the validation set. It also achieved the accuracy of 92,8% and 85,2% respectively on both test sets consisting of real photos of sidewalks. The result is satisfying and it encourages further works on the combination of image generation and image classification.

Keywords: artificial neural networks, deep learning, ConvNeXt, ResNet, generative models, image classification

Spis treści

1 Wprowadzenie	1
1.1 Cel pracy	1
1.2 Układ pracy	2
2 Stan wiedzy i przegląd istniejących rozwiązań	3
2.1 Matematyczny model neuronu	3
2.2 Głębokie nauczanie	5
2.3 Splotowe sieci neuronowe	6
2.4 ConvNeXt	7
2.5 Transfer Learning	9
2.6 Sieci dyfuzyjne	9
2.7 System oceny czystości chodników	11
2.8 Porównanie modeli oceniających czystość nawierzchni	12
3 Wykorzystane narzędzia i biblioteki	13
3.1 Python	13
3.2 PyCharm	13
3.3 Pytorch	14
3.4 TensorBoard	15
3.5 Git LFS	16
3.6 Hugging Face	16
3.6.1 Stable Diffusion v1.5	16
3.6.2 Stable Diffusion v2.1	16
3.6.3 Stable Diffusion Inpainting	17
3.6.4 Kluczowe argumenty	17
3.7 Adobe Firefly	17
3.8 Scikit-learn	18
3.9 Pillow	19
4 Przygotowanie bazy zdjęć	21
4.1 Generowanie obrazów za pomocą Adobe Firefly	21
4.2 Generacja zabrudzonych chodników za pomocą modeli Stable Diffusion	23

4.2.1	Metoda text-to-image	23
4.2.2	Metoda image-to-image	25
4.2.3	Metoda inpainting	28
4.3	Generacja zdjęć większych grup liści	31
4.4	Generacja zdjęć czystych chodników	32
4.5	Podsumowanie generacji zdjęć chodników	33
4.6	Problemy podczas generacji zdjęć	34
4.7	Zbiór rzeczywistych zdjęć chodników	34
5	Implementacja rozwiązania	37
5.1	Przygotowanie bazy danych do nauczania	37
5.1.1	Klasyfikacja danych	37
5.1.2	Podział i transformacje danych	38
5.2	Przygotowanie modelu sieci neuronowej do nauczania	39
5.2.1	ConvNeXt	39
5.2.2	ResNet	40
5.3	Optymalizacja hiperparametrów nauczania	40
5.3.1	Dobór rozmiaru paczki uczącej	40
5.3.2	Dobór współczynnika uczenia	41
5.4	Wyniki nauczania sieci neuronowych	41
5.4.1	Sieć ConvNeXt	41
5.4.2	Sieć ResNet152	45
5.5	Próba poprawy wyników	49
6	Podsumowanie i wnioski	51
Bibliografia		53
Wykaz skrótów i symboli		55
Spis rysunków		57
Spis tabel		61
Spis załączników		63

Rozdział 1

Wprowadzenie

Pierwsze prace nad sztucznymi sieciami neuronowymi sięgają połowy XX wieku, jednak dopiero niedawno wzrosło zainteresowanie tym tematem. Przełomowym momentem było wprowadzenie głębokiego nauczania, odłamu nauczania maszynowego, i konwolucyjnych sieci neuronowych, które zrewolucjonizowały rynek i pokazały nowe możliwości w praktycznym zastosowaniu sztucznych sieci neuronowych. Było to spowodowane rozwojem procesorów graficznych, pozwalających na wykonywanie dużej liczby operacji zmiennoprzecinkowych i zwiększenie wydajności komputerów. Do szybkiego rozwoju przyczyniło się również wyzwanie ImageNet, w którym co roku nagrodę otrzymywał model sieci, który najlepiej radził sobie z problemem klasyfikacji obrazów należących do zbioru ImageNet. Model AlexNet, zwycięski w 2012 roku, stał się podstawą do projektowania architektury nowych, lepszych i bardziej wydajnych sieci. Bardzo duże zainteresowanie budzą ostatnio modele generatywne do tworzenia obrazów. Ich nadzwyczajna popularność była jednym z powodów podjęcia się niniejszej pracy. Dodatkową motywacją była chęć rozwoju w dziedzinie głębokiego nauczania, spowodowana jej dynamicznym rozwojem w ostatnich latach.

1.1 Cel pracy

Po wstępnej analizie opracowywanego problemu sformułowano dwa główne cele pracy:

Celem praktycznym, który można uznać za cel główny pracy, jest uzyskanie sprawnego klasyfikatora, który będzie w stanie w zadowalającym stopniu odróżniać stopnie zabrudzenia chodników miejskich. Architektura modelu bazowego, który zostanie w tym celu douczony, musi być stosunkowo nowa i opracowana na przestrzeni ostatnich kilku lat. Wszystkie zdjęcia, które będą brać udział w procesie nauczania sieci, muszą zostać sztucznie stworzone przez modele generatywne.

Celem badawczym jest sprawdzenie, czy korzystając wyłącznie ze zdjęć wygenerowanych, nauczona sieć neuronowa będzie w stanie poprawnie klasyfikować prawdziwe zdjęcia chodników.

1.2 Układ pracy

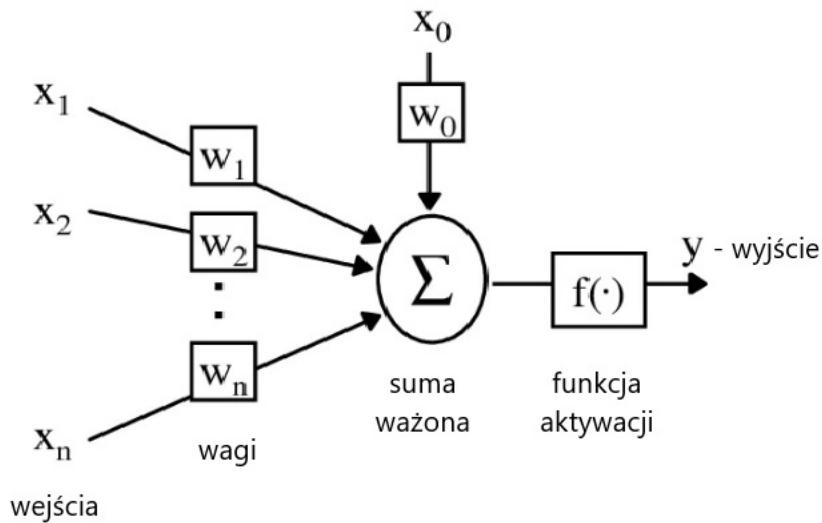
W pierwszym rozdziale przedstawiono zarys tematyczny pracy wraz z motywacją do podjęcia tematu, wyszczególniono cele projektu i ogólną zawartość każdego rozdziału. W kolejnym rozdziale opisano stan wiedzy poruszanego problemu wraz z przeglądem istniejących prac o podobnej tematyce. W trzecim rozdziale wymieniono i scharakteryzowano narzędzia oraz biblioteki, z jakich korzystano podczas pracy. W następnym rozdziale opisano proces zbierania bazy danych uczących, na których zostały wytrenowane oba modele sieci neuronowych oraz bazy zdjęć rzeczywistych, które posłużyły jako test sprawności klasyfikatorów na prawdziwych obrazach. W rozdziale piątym opisano proces uczenia sieci neuronowych wraz z wynikami testów, ich analizą i próbami poprawy uzyskanych rezultatów. Ostatni rozdział zawiera podsumowanie wyników, wnioski z nich płynące oraz propozycje dalszych prac i badań w tematyce, którą zajmuje się niniejsza praca.

Rozdział 2

Stan wiedzy i przegląd istniejących rozwiązań

2.1 Matematyczny model neuronu

Sztuczne sieci neuronowe swoje podstawy czerpią z naturalnych mechanizmów działania układów nerwowych żywych organizmów. Sygnały przekazywane pomiędzy kolejnymi neuronami mogą być zarówno wzmacniane, jak i tłumione, co jest możliwe przez skomplikowane procesy chemiczno-elektryczne zapewniające transmisję sygnałów wewnątrz systemu nerwowego. Chcąc opisać model matematyczny, można przyjąć pewne wagi występujące na wejściach komórek nerwowych, przez które mnożone są sygnały wejściowe i na tej podstawie można stwierdzić wpływ kolejnych sygnałów wejściowych na stan komórki nerwowej. Sygnały te mogą przyjmować wartości dodatnie i ujemne, a na to, czy komórka nerwowa zostanie pobudzona, wpływa ogólny bilans tych sygnałów. W przypadku gdy bilans ten jest na korzyść wejść tłumionych, to na wyjściu komórki nie zostaną zaobserwowane żadne zmiany. Gdy jednak bilans wejść będzie dodatni, nastąpi pobudzenie komórki i przekazanie dalej impulsu nerwowego. Jeden z pierwszych modeli matematycznych neuronu został opisany w roku 1943 i mianowany neuronem McCullocha-Pittsa [11]. Schemat tego neuronu został przedstawiony na rysunku 1.

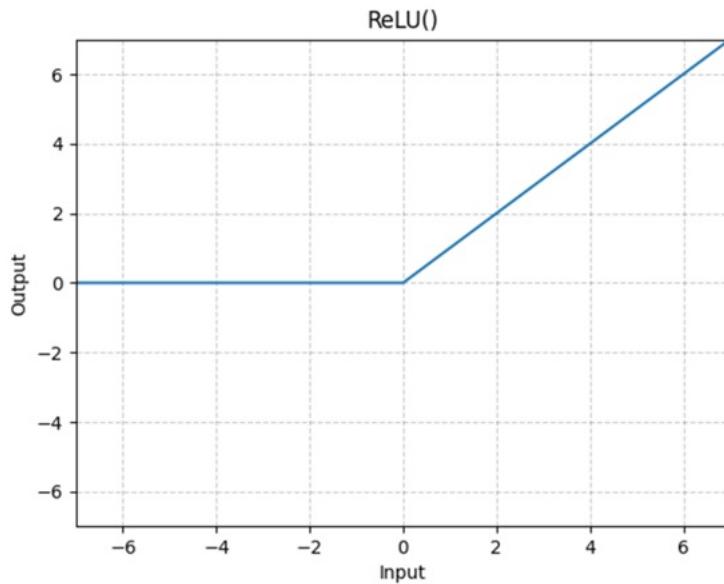


Rysunek 1. Schemat neuronu McCullocha-Pittsa. Opracowano na podstawie: [14]

Widoczne po lewej stronie sygnały wejściowe x_1, x_2, \dots, x_n są za pomocą sumatora sumowane z określonymi wagami i porównywane z wartością progową w_0 . Wynik tej sumy trafia do bloku funkcji aktywacji, który określa, jaka wartość pojawi się na wyjściu. W pierwszych modelach neuronów za funkcję aktywacji przyjmowano funkcję skokową

$$f(\sum) = \begin{cases} 1, & \sum > 0 \\ 0, & \sum \leq 0 \end{cases}$$

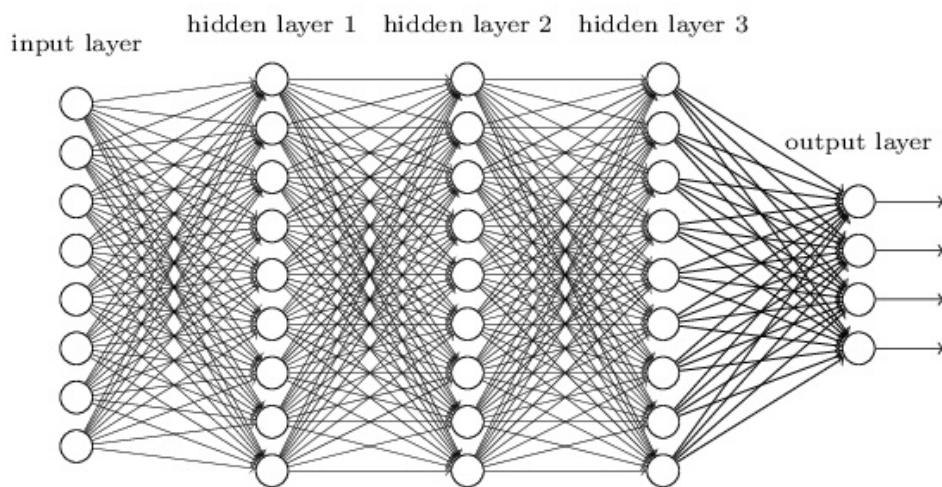
W miarę rozwoju tej dziedziny coraz częściej decydowano się jednak na bardziej złożone funkcje aktywacji. Jedną z obecnie popularniejszych jest funkcja ReLU, która jest definiowana jako nieujemna część argumentu. Wprowadza ona nieliniowość do modelu sieci neuronowej i jest szczególnie popularna w głębokim nauczaniu. Wykres funkcji ReLU został przedstawiony na rysunku 2.



Rysunek 2. Wykres funkcji aktywacji ReLU. Źródło: [23]

2.2 G  ebokie nauczanie

Nauczanie g  ebokich sieci neuronowych, składających się z wielu warstw neuronów, określa się mianem g  ebokiego nauczania. Nazwa pochodzi od liczby tych warstw, która często jest bardzo duża w porównaniu z innymi rodzajami sieci. W g  ebokiej sieci neuronowej wyróżniamy warstwę wejściową, warstwy ukryte i warstwę wyjściową. Przykładowy schemat budowy sieci neuronowej przedstawiono na rysunku 3.



Rysunek 3. Struktura g  ebokiej sieci neuronowej. Źródło: [9]

Głębokie nauczanie jest obszerną podkategorią uczenia maszynowego. W trakcie procesu nauczania sieć sama wyodrębnia cechy charakterystyczne obiektów i decyduje o optymalnym rozmieszczeniu zadań identyfikacji tych cech pomiędzy ukryte warstwy sieci. Dzięki temu model sieci może skuteczniej nauczyć się rozpoznawać określone wzorce. Mimo wszystko w procesie głębokiego nauczania do opracowywania ilości warstw, ich rozmiarów i innych hiperparametrów jest potrzebna ludzka ręka.

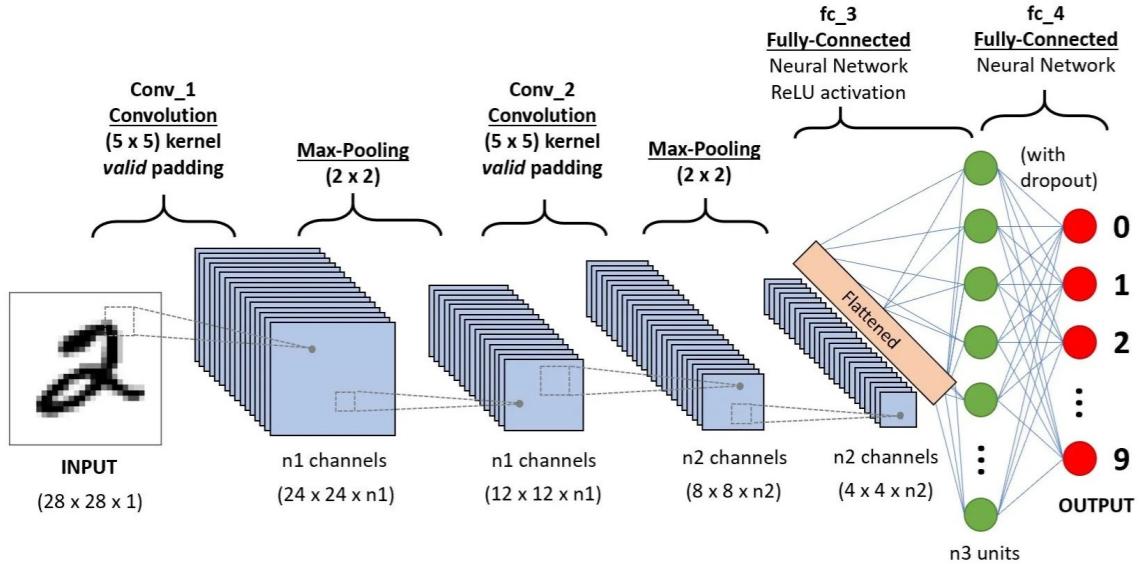
2.3 Splotowe sieci neuronowe

Jest to podkategoria sieci głębokich, która posiada szczególne znaczenie w klasyfikacji obrazów do określonych kategorii. Jako głęboka sieć neuronowa jest w stanie automatycznie wykrywać pożądane cechy charakterystyczne obiektów i na ich podstawie decydować o przypisaniu do klasy. Warstwy splotowe zapewniają sieci cechy niezmienne względem translacji, co umożliwia identyfikację wzorców bez względu na zmianę orientacji, położenia czy skali obrazów wejściowych. Jest to cecha szczególnie przydatna przy powiększaniu zbioru danych wejściowych.

Na budowę splotowych sieci neuronowych składają się cztery główne rodzaje bloków:

1. **Warstwa splotowa**, która pełni najważniejszą rolę w całej strukturze sieci. W pierwszej warstwie ukrytej na zdjęcie wejściowe, które najczęściej jest w postaci tensora pikseli, nakładane są filtry (maski filtrujące) w postaci macierzy o określonej szerokości i długości, które wędrują z określonym krokiem po tensorze pikseli obrazu wejściowego. Wykorzystując wiele masek filtrujących o różnych wartościach, powstaje wiele obrazów wyjściowych, rejestrujących różne cechy charakterystyczne obrazu wejściowego, są one w postaci macierzy i noszą nazwę macierzy splotowych. Wartości masek filtrujących są automatycznie dostosowywane do problemu podczas procesu nauczania [11].
2. **Funkcja aktywacji ReLU**, jest stosowana po każdej operacji splotu, pomaga sieci uczyć się nielinowych relacji pomiędzy cechami obrazu, co zwiększa odporność sieci na identyfikację różnych wzorców. Pomaga również łagodzić problem zanikającego gradientu [6].
3. **Warstwa łącząca**, której głównym celem jest wyodrębnienie najważniejszych atrybutów z map cech (macierzy splotowych), poprzez zastosowanie określonych operacji agregacji. Jedną z popularnych operacji jest przepisanie jedynie największej wartości z (określonego przez filtr) obszaru macierzy splotowej, nosi ona nazwę *max pooling*.
4. **Warstwa w pełni połączona**, jest ostatnią warstwą w splotowej sieci neuronowej. Za jej wejścia służą dane macierzy jednowymiarowej, spłaszczonej przez ostatnią warstwę łączącą. Warstwa predykcji softmax generuje wartości prawdopodobieństwa dla każdej z otrzymanych etykiet. Jako ostateczna predykcja jest określana etykieta z największym prawdopodobieństwem.

Na rysunku 4 została przedstawiona przykładowa architektura splotowej sieci neuronowej do klasyfikowania odreźnie napisanych cyfr.

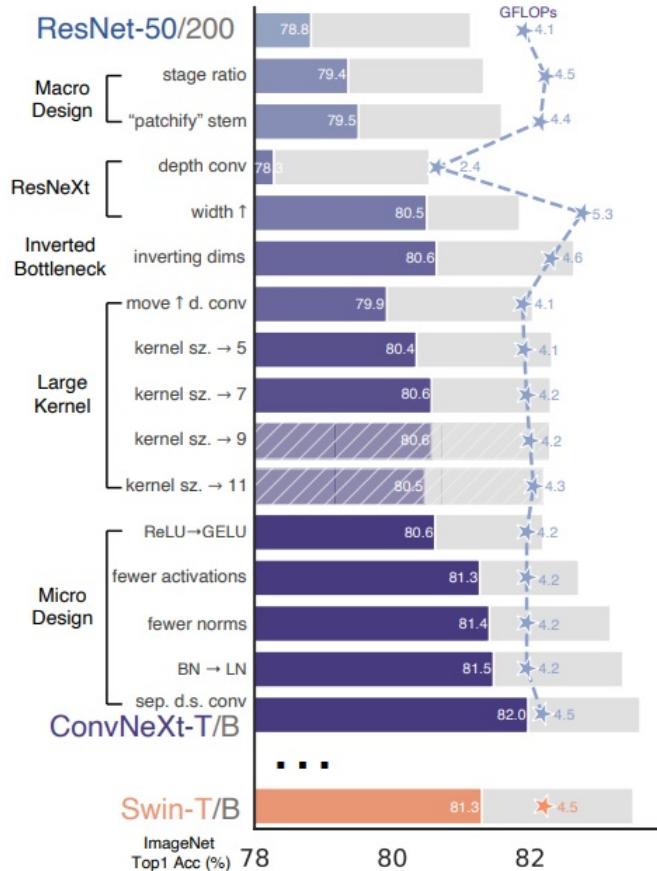


Rysunek 4. Przykładowa architektura splotowej sieci neuronowej. Źródło: [15]

2.4 ConvNeXt

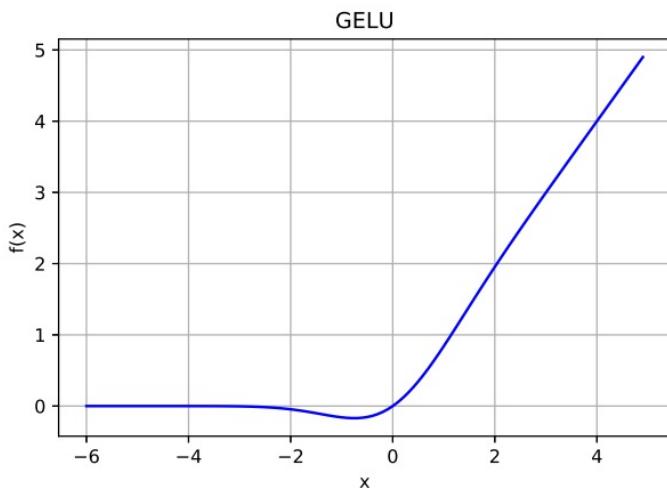
W latach 20. XXI wieku zostały rozpoczęte prace nad udoskonaleniem splotowych sieci neuronowych. Polegały one na przeanalizowaniu architektury sieci aktualnie popularnych w problemie klasyfikacji obrazów i próbie implementacji niektórych z wybranych rozwiązań w nowopowstającej architekturze splotowej sieci neuronowej, którą nazwano ConvNeXt.

Prace rozpoczęto stawiając architekturę sieci ResNet jako bazę do późniejszej modernizacji. Na rysunku 5 przedstawiono, jakim zmianom poddano klasyczną budowę sieci ResNet w celu uzyskania lepszych rezultatów nie tylko w problemie klasyfikacji obrazów, ale również w przypadku segmentacji semantycznej i detekcji obiektów na obrazie. Wynik zestawiono z rezultatem uzyskanym w momencie publikacji przez popularną sieć neuronową Swin-Transformer, która jest jedną z sieci typu Vision-Transformer [13] i która cały czas konkuuuje z sieciami CNN w wyżej wymienionych problemach. Architektura sieci ConvNeXt dziedziczy wiele aspektów budowy sieci typu Vision-Transformer.



Rysunek 5. Prace nad projektowaniem sieci ConvNeXt. Źródło: [7]

Przeprowadzane zmiany miały na uwadze poprawę dokładności oceny i możliwe jak najmniejsze GFLOPs. Ważnym krokiem podczas procesu projektowania architektury sieci było zwiększenie rozmiaru maski filtrującej, co miało na celu zniwelowanie lokalnej samoświadomości sieci i wymuszenie percepkcji większej części obrazu. Kolejną interesującą zmianą była zamiana funkcji aktywacji ReLU na funkcję GELU, na wzór budowy bloku sieci typu Vision-Transformer. Na rysunku 6 pokazano wykres funkcji GELU.



Rysunek 6. Wykres funkcji aktywacji GELU. Źródło: [2]

2.5 Transfer Learning

Użycie gotowego modelu sieci neuronowej i dostrojenie go do podobnego problemu, dla którego sieć została wytrenowana, jest znacznie szybsze i łatwiejsze od zaprojektowania nowej sieci. W praktyce ta metoda jest bardzo popularna i nosi nazwę *transfer learning*. Aby dostroić sieć, potrzeba również znacznie mniejszej liczby danych uczących. Do nauczania modeli „od zera”, używa się gotowych baz danych. Popularną bazą zdjęć jest ImageNet, który składa się z ponad 14 milionów obrazów [12].

Proces transfer learningu polega zwykle na zmianie ostatnich warstw sieci neuronowej i dostosowaniu ich do rozwiązywanego problemu. Można w ten sposób zmienić liczbę klas wyjściowych, do których sieć będzie klasyfikować zdjęcia dostarczane podczas trenowania.

2.6 Sieci dyfuzyjne

Modele sieci dyfuzyjnych wykorzystywane głównie do generowania obrazów, są trenowane poprzez głębokie uczenie, aby z losowo wygenerowanego szumu po procesie odszumiania otrzymać nowe zdjęcie o pożądanych atrybutach. Proces nauczania sieci dyfuzyjnych można podzielić na kilka etapów [1]:

1. **Proces dyfuzji do przodu**, w którym obraz treningowy jest krok po kroku zaszumiany i przekształcany w czysty szum (zwykle szum Gaussa).
2. W **procesie dyfuzji odwrotnej** sieć uczy się przekształcać szum w gotowy obraz, na podstawie informacji o każdym poprzednim kroku zaszumiania w oryginalnym procesie dyfuzji do przodu.
3. **Generowanie nowego obrazu**, gdzie wyuczony model pobiera próbki losowego szumu i przekształca go w wysokiej jakości obraz. Element losowości w podawanym na wejściu szumie jest konieczny do uzyskania nowych obrazów, różniących się od tych w zbiorze treningowym.

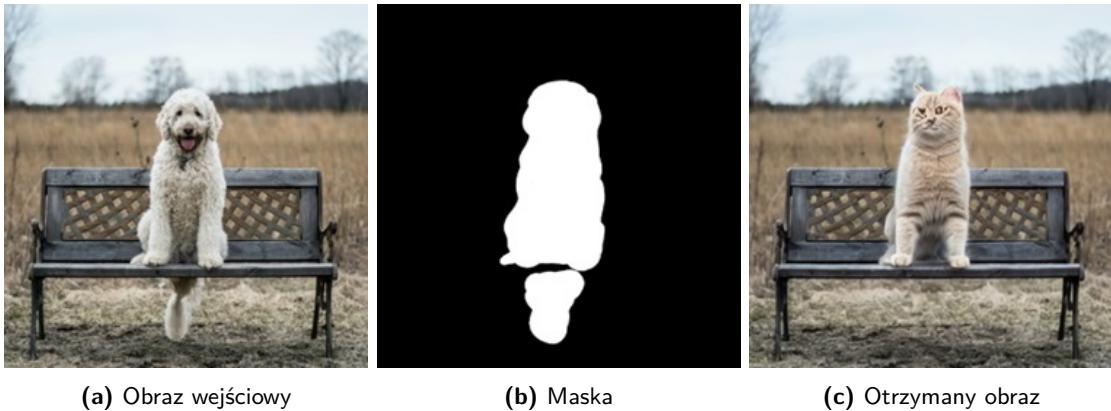
Wytrenowany model dyfuzyjny jest w stanie wyprodukować wysokiej jakości wariacje zdjęć użytych podczas trenowania, jednak w jego praktycznym zastosowaniu, często pożąda się wyniku spoza tematyki zdjęć w bazie trenującej. Zostały więc wprowadzone zmiany w celu nadania użytkownikowi kontroli nad wynikiem generacji. Najbardziej popularną metodą kierowania wynikiem jest wprowadzenie tekstu pomocniczego, który krótko opisuje pożądany obraz. To rozwiązanie wymaga korelacji sieci dyfuzyjnej z siecią LLM. W wielu opracowanych modelach jest również funkcja dodania zdjęcia referencyjnego, które sieć traktuje jako punkt startowy. Proces generacji nowego obrazu jest wówczas bardziej złożony, sieć nakłada na wprowadzone zdjęcie szum, a następnie generuje nowy obraz w procesie odszumiania. Na rysunku 7 przedstawiono wyniki generacji trzech zdjęć, po różnej liczbie kroków odszumiania.



(a) 3 kroki odszumiania (b) 20 kroków odszumiania (c) 150 kroków odszumiania

Rysunek 7. Wyniki generacji obrazów z różnicą liczbą kroków odszumiania. Źródło: własne

Jeśli celem generacji jest jedynie zmiana określonej części obrazu referencyjnego i zachowanie pozostałej części w stanie przed generacji, można zastosować metodę *inpainting*. Polega ona na dostarczeniu obrazu wejściowego, maski i tekstu pomocniczego, w którym należy opisać, jakie zmiany mają zajść w miejscu obrazu referencyjnego wyznaczanego przez maskę. Na rysunku 8 pokazano przykładowe zastosowanie tej metody. Został wprowadzony tekst pomocniczy o treści „cat sitting on the bench”, a do generacji wybrano model stable-diffusion-inpainting.



(a) Obraz wejściowy

(b) Maska

(c) Otrzymany obraz

Rysunek 8. Przykładowy wynik generacji nowego obrazu za pomocą metody inpainting. Opracowano na podstawie: [18]

2.7 System oceny czystości chodników

W 2020 roku pojawiła się praca o podobnej tematyce, związana z oceną zabrudzenia liśćmi chodników miejskich [19]. W pierwszym etapie prac autor przeprowadził nauczanie sieci ResNet pod kątem badanego problemu, a następnie otrzymane wyniki zestawił z wynikami kolejnego etapu badań. Tym razem zastosował generatywne sieci przeciwwstawne GAN, których idea polega na przekształceniu zdjęcia wejściowego w zdjęcie wyjściowe o pożądanych atrybutach. Podczas nauczania sieci konieczne było ponowne „zabrudzenie chodnika” w celu uzyskania możliwości przemiennej zmienności, co zostało przedstawione na rysunku 9. W ten sposób zdjęcie zabrudzonego chodnika były przekształcane w zdjęcie chodnika czystego, a następnie porównywane ze zdjęciem wejściowym. Na podstawie różnic pomiędzy zdjęciem oryginalnym a wygenerowanym można było określić przynależność zdjęcia wejściowego do konkretnej klasy zabrudzenia.



(a) Zdjęcie zabrudzonego chodnika

(b) Zdjęcie po wyczyszczeniu

(c) Zdjęcie powtórnie zabrudzone

Rysunek 9. Zmiany zdjęcia wejściowego w procesie nauczania sieci GAN. Źródło: [19]

Wykorzystując sieć ResNet i powiększając zbiór danych o losową rotację, autorowi udało się uzyskać dokładność 85% na zbiorze testowym podzielonym na 3 klasy zabrudzeń. Wykorzystując sieć GAN i ten sam zbiór danych, uzyskano dokładność 78% na zbiorze testowym.

2.8 Porównanie modeli oceniających czystość nawierzchni

W pracy inżynierskiej z 2022 roku przeprowadzono porównanie kilku popularnych w tamtym okresie modeli sieci stosowanych do klasyfikacji obrazów. Wśród nich badane były sieci TResNet, ShuffleNet V2 oraz PMG. W pracy tej przystosowano je do oceny poziomu zanieczyszczenia liśćmi nawierzchni chodnikowej różnego rodzaju. Badanie przeprowadzono dla zbioru danych uczących podzielnego na 3 i 6 klas zabrudzenia. Dla problemu trzyklasowego osiągnięto dokładność zbioru testowego rzędu 90%, natomiast dla problemu sześcioklasowego dokładność ta oscylowała w granicach 80% [8].

Rozdział 3

Wykorzystane narzędzia i biblioteki

3.1 Python

Ten imperatywno-obiektowy [16] język programowania wysokiego poziomu jest 3. rokiem z rzędu uznany za najpopularniejszy według wysoko cenionego rankingu TIOBE. Wyniki górnej części rankingu zostały umieszczone w tabeli 1. Rosnąca popularność Python z pewnością zawdzięcza intensywnemu w ostatnich latach rozwojowi sztucznej inteligencji. Oprócz obiektowego i imperatywnego, wspierany jest również w mniejszym stopniu funkcyjny paradymat programowania. Decyzja o skorzystaniu z tego języka w niniejszej pracy była związana z dużą różnorodnością bibliotek wspieranych przez Python, przygotowanych do problemu głębokiego nauczania.

Tabela 1. Pierwsze 8 miejsc rankingu TIOBE w 2024 roku. Źródło: [5]

Nov 2024	Nov 2023	Change	Programming Language	Ratings	Change
1	1		Python	22.85%	+8.69%
2	3		C++	10.64%	+0.29%
3	4		Java	9.60%	+1.26%
4	2		C	9.01%	-2.76%
5	5		C#	4.98%	-2.67%
6	6		JavaScript	3.71%	+0.50%
7	13		Go	2.35%	+1.16%
8	12		Fortran	1.97%	+0.67%

3.2 PyCharm

Jest to jeden z popularniejszych, zwłaszcza wśród początkujących programistów IDE do języka Python. Jego powodzenie wynika między innymi z wielu korzystnych funkcji, takich jak autouzu-

pełnianie i kolorowanie tekstu. Również bardzo dużo książek i poradników dla osób początkujących wykorzystuje środowisko PyCharm. Jego instalacja jest bardzo prosta i intuicyjna, wspiera wszystkie oficjalne wersje języka Python i przy tworzeniu nowych projektów daje możliwość wybrania określonej wersji języka. Jest to szczególnie przydatne w pracy nad projektami obejmującymi szybko rozwijające się dziedziny, gdyż często zdarza się, że określone modele lub biblioteki są wspierane jedynie w starszej wersji języka. Z tych względów zdecydowano się przy pisaniu tej pracy na skorzystanie z tego IDE.

3.3 Pytorch

Jest to dynamicznie rozwijający się framework do uczenia maszynowego. Sposób instalacji jest bardzo prosty i intuicyjny. Na głównej stronie PyTorch jest opcja pobrania modułu, należy wybrać build aplikacji, system operacyjny komputera, na którym będzie ona używana, następnie rodzaj paczki instalacyjnej, dla języka Python szczególnie popularny jest pip. Potem należy wybrać język programowania, gdzie jest również opcja „C++/Java”, ale framework jest znacznie lepiej wspierany przez język Python. Następnie trzeba wybrać platformę obliczeniową, w tej pracy korzystano z CUDA, czyli specjalnej architektury opracowanej przez firmę NVIDIA [10], w tym celu wymagane jest posiadanie karty graficznej z serii NVIDIA GeForce. Obliczenia z wykorzystaniem biblioteki PyTorch będą wówczas wykonywane z użyciem GPU. Możliwe jest w tym miejscu wybranie CPU, wtedy wszystkie obliczenia będą wykonywane na głównym procesorze komputera, jednak należy tutaj wziąć pod uwagę jego większe ograniczenia obliczeniowe niż w przypadku procesorów GPU.

Na potrzeby tej pracy wykorzystane zostały dwa duże pakiety, będące częścią biblioteki PyTorch:

1. Pakiet **torch** zawiera wiele przydatnych w głębokim nauczaniu narzędzi operujących na wielowymiarowych tensorach. Posiada także obszerny moduł *torch.nn*, który zapewnia wiele możliwości podczas pracy z sieciami neuronowymi [22].
2. Pakiet **torchvision** zawiera bibliotekę modeli sieci neuronowych, funkcje operacji na zbiorach danych i bibliotekę transformacji obrazów, przydatną podczas przygotowywania zdjęć do „przejścia” przez sieć neuronową podczas jej nauczania i testowania.

Główną strukturą danych wykorzystywaną w pakietach biblioteki jest *tensor*, bardzo podobny do struktury *array* z popularnej biblioteki *numpy* wspieranej przez język Python. Główna różnica polega na tym, że tensorzy mogą być uruchamiane na GPU, które zapewniają większą moc obliczeniową [21]. Tensorzy są używane do zapisywania danych wejściowych i wyjściowych modeli głębokiego nauczania oraz do przedstawiania za ich pomocą parametrów modeli.

Jest wiele sposobów na zainicjalizowanie tensora. Jedną z nich jest bezpośrednia inicjalizacja, podając dane do tensora:

```
data = [[1, 2], [3, 4]]  
x_data = torch.tensor(data)
```

gdzie typ danych jest przypisywany automatycznie. Jeden tensor nie może zawierać danych różnego typu. Tensorzy można również inicjalizować na podstawie istniejących tensorów:

```

data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)
x_ones = torch.ones_like(x_data)
print(x_ones)

```

Powstały w ten sposób tensor `x_ones` będzie zawierał same jedynki i odziedziczy kształt po tensorze `x_data`, więc jako wynik powyższego fragmentu kodu otrzymamy:

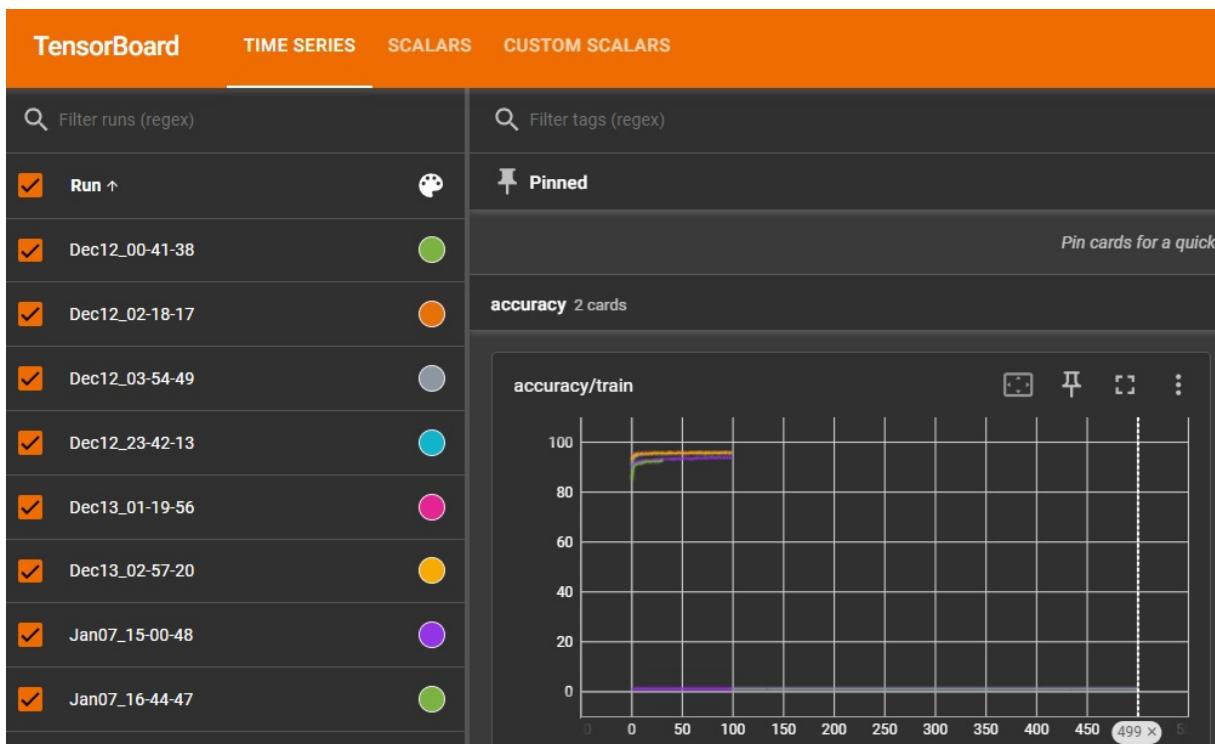
```

tensor([[1, 1],
       [1, 1]])

```

3.4 TensorBoard

To narzędzie do wizualizacji efektów głębokiego uczenia jest powszechnie używane z TensorFlow, ale może być również wykorzystywane z innymi bibliotekami do uczenia sieci, takimi jak na przykład PyTorch. Pozwala ono na kontrolowanie przebiegu nauczania i po odpowiedniej implementacji w kodzie programu jest w stanie aktualizować wyniki na lokalnym serwerze po każdej epoce uczącej, więc możliwe jest otrzymywanie bieżącego stanu naczania sieci. TensorBoard zapewnia klasę `SummaryWriter()`, która pozwala na zapisywanie rezultatów wszystkich przeprowadzonych nauczeń w folderze, z którego potem dane są przekazywane do lokalnego serwera. Na rysunku 10 ten folder o nazwie `Run` oraz kilka baz danych zebranych podczas nauczania widać po lewej stronie.



Rysunek 10. Fragment okna programu TensorBoard. Źródło: własne

3.5 Git LFS

Git Large Files Storage jest otwarto-źródłowym rozszerzeniem systemu Git, które pozwala na kopiowanie dużych plików na przykład z repozytorium GitHuba. W tej pracy użyto tego rozszerzenia do pobierania modeli dyfuzyjnych o wielkości nawet do 50 GB. Instalacja jest bardzo prosta, intuicyjna i można ją wykonać na oficjalnej stronie git-scm.com w zakładce *Downloads*.

3.6 Hugging Face

Jest to platforma zrzeszająca zarówno deweloperów, jak i entuzjastów zajmujących się szeroko pojętą sztuczną inteligencją. Służy ona do ogólnoświatowej kolaboracji w celu tworzenia nowatorskich rozwiązań dla problemów uczenia maszynowego. Można tam znaleźć gotowe modele sieci neuronowych, udostępniane przez twórców do samodzielnego eksperymentowania. W tej pracy zostały wykorzystane gotowe modele sieci dyfuzyjnych do generowania obrazów, pobrane z platformy Hugging Face. Uruchamianie modeli w środowisku PyCharm było możliwe dzięki bibliotece *diffusers*, stworzonej na potrzeby testowania wytrenowanych modeli dyfuzyjnych.

3.6.1 Stable Diffusion v1.5

Wersja 1.5 została pierwotnie umieszczona na platformie przez Runway, czyli firmę zajmującą się SI, jednak po pewnym czasie model usunięto. Istnieją jednak na platformie lustrzane odbicia repozytorium do modelu SD v1.5, które nie są już w żaden sposób powiązane z firmą Runway i właśnie to repozytorium zostało użyte. Biblioteka *diffusers* pozwala na skorzystanie z kilku możliwych sposobów kontroli wyników generacji:

1. **Text-to-image**, czyli generowanie obrazów na podstawie wprowadzonego tekstu pomocniczego, używany za pomocą modułu *StableDiffusionPipeline* importowanego z biblioteki.
2. **Image-to-image**, czyli generacja na podstawie wprowadzonego zdjęcia referencyjnego. Jest również możliwość wprowadzenia tekstu pomocniczego jako dodatkowe naprowadzenie sieci neuronowej na oczekiwany wynik. Do generacji używany jest moduł *StableDiffusionImg2ImgPipeline*.
3. **Inpainting**, czyli generacja jedynie zaznaczonego fragmentu zdjęcia wejściowego na podstawie podanej maski i wprowadzonego tekstu pomocniczego. Konieczny jest w tym przypadku moduł *StableDiffusionInpaintPipeline*.

3.6.2 Stable Diffusion v2.1

Nowsza wersja modelu dyfuzyjnego została przedstawiona przez firmę Stability AI. Przy korzystaniu z tego modelu generowano obrazy, wykorzystując jedynie metody *text-to-image* i *image-to-image*. Wykorzystano te same moduły co dla modelu w wersji 1.5.

3.6.3 Stable Diffusion Inpainting

Jest to model specjalnie dostrojony do problemu generowania zdjęć metodą inpainting. W niniejszej pracy testowano dwie wersje tego modelu:

- stable-diffusion-inpainting
- stable-diffusion-2-inpainting

Do generacji zdjęć przy wykorzystaniu powyższych modeli użyto modułu *DiffusionPipeline* z biblioteki *diffusers*.

3.6.4 Kluczowe argumenty

Przy implementacji modelu dyfuzyjnego w celu generacji obrazów niezbędne jest przekazanie do modelu kluczowych argumentów, z których najważniejsze to:

- **prompt**, w którym przekazany zostaje tekst pomocniczy, z którego korzysta model w procesie generacji,
- **width** oraz **height**, za pomocą których należy określić (w pikselach) rozmiary obrazu wyjściowego,
- **guidance_scale**, im wyższa jest podana w tym miejscu wartość, tym bardziej model stosuje się do wytycznych wpisanych w tekście pomocniczym, czego kosztem jest słabsza jakość wygenerowanego obrazu,
- **num_inference_steps** określa liczbę kroków w procesie odszumiania.

Przy generowaniu obrazów metodą image-to-image, należy podać również poniższe parametry:

- **image**, gdzie przekazać trzeba obraz referencyjny, od którego model ma zacząć proces generacji,
- **strength** w przedziale od 0 do 1, określa ilość wstępnie dodanego szumu i to jak w jakim stopniu model ma korzystać z dodanego zdjęcia referencyjnego, gdzie wartość 1 oznacza zignorowanie tego obrazu.

Podczas generacji metodą inpainting dochodzi jeszcze jeden parametr o nazwie **mask_image**, w którym przekazujemy maskę określającą, które miejsca na obrazie wejściowym mają ulec zmianie.

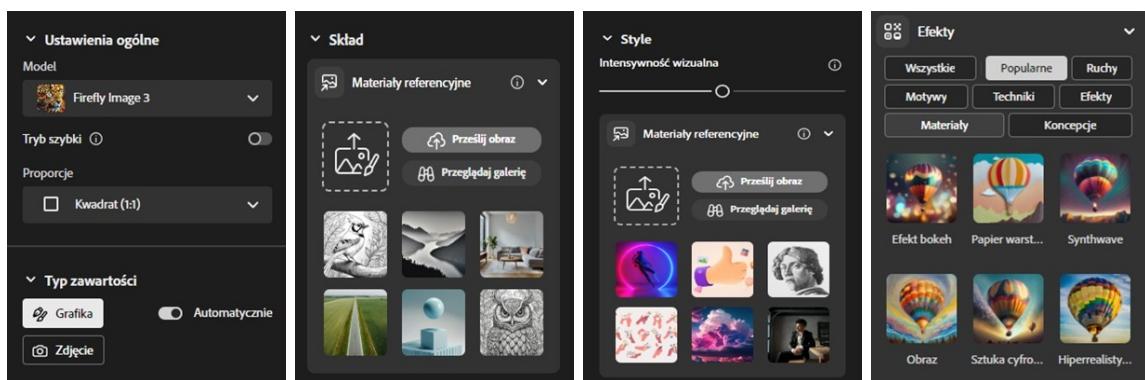
3.7 Adobe Firefly

Firma Adobe zajmująca się oprogramowaniem komputerowym zaprojektowała model *Adobe Firefly Image*. Aktualnie na stronie dostępne są dwie wersje modelu: *Firefly Image 2* i *Firefly Image 3*¹. Na potrzeby tej pracy wykorzystana została 3. wersja modelu. Głównym założeniem projektu było stworzenie generatywnej sztucznej inteligencji wspierającej kreatywność i rozwijającej zmysły artystyczne korzystających z niej osób. Do trenowania modeli firma Adobe używa między innymi zbioru danych z usługi Adobe Stock, który w styczniu 2025 roku liczy ponad 318 milionów zdjęć.

¹Stan na: 03.12.2024r.

Dostęp do generowania obrazów za pomocą modeli Adobe Firefly jest darmowy, lecz ograniczony. Bezpłatny plan pozwala na wygenerowanie 100 zdjęć na miesiąc. W przypadku potrzeby generacji większej liczby zdjęć można skorzystać z płatnej subskrypcji. Subskrypcja poziomu pierwszego zapewnia 400, a poziomu drugiego 1000 generacji na miesiąc.

Aplikacja Adobe Firefly zapewnia szereg narzędzi ułatwiających generację obrazów odpowiadających oczekiwaniom użytkownika. Można zarówno wybrać kształt generowanego obrazu, jak i określić, czy obraz wyjściowy ma mieć charakter zdjęcia albo grafiki. Istnieje również opcja dodania zdjęć referencyjnych, które mogą być przydatne w kierowaniu wynikiem generacji. Zdjęcia te mogą pomóc w określeniu stylu i/lub zawartości obrazu wyjściowego. Jest również dostępna zakładka *Efekty*, która pozwala na dostosowywanie koloru, odcieni, światła, a nawet kąta aparatu generowanych zdjęć. Model przy każdej generacji dodatkowo opiera się na tekście pomocniczym, który dostarcza użytkownik. Wygląd wyboru niektórych narzędzi w programie pokazano na rysunku 11.



Rysunek 11. Fragmenty UI programu Adobe Firefly. Źródło: własne

3.8 Scikit-learn

Jest to darmowa, otwarto-źródłowa biblioteka do języka Python. Znacząco ułatwia wszelkie prace i eksperymenty z nauczaniem maszynowym, w tym z głębokim nauczaniem. Umożliwia tworzenie macierzy pomyłek, które wyznaczone z danych testowych pozwalają na łatwą ocenę jakości i poprawności modeli klasyfikacyjnych. Na rysunku 12 przedstawiono przykładową macierz pomyłek.

True\predicted	c_1	c_2	c_3
c_1	5	0	2
c_2	1	7	2
c_3	1	4	3

Rysunek 12. Macierz pomyłek klasyfikatora trzech klas. Źródło: [20]

Na podstawie macierzy pomyłek można również wyliczyć dokładność (ang. *accuracy*), która określa procent dobrze sklasyfikowanych obrazów w stosunku do wszystkich ocenianych zdjęć. Aby to zrobić, należy sumę liczb na diagonali macierzy pomyłek podzielić przez sumę wszystkich danych w macierzy. Jeśli wynik ma być w procentach, otrzymany iloraz trzeba pomnożyć przez 100%. Dla macierzy pomyłek z rysunku 12 dokładność klasyfikacji wynosi:

$$\text{accuracy} = \frac{\sum_{i=1}^3 c[i,i]}{\sum_{i=1}^3 \sum_{j=1}^3 c[i,j]} \times 100\% = \frac{5 + 7 + 3}{5 + 0 + 2 + 1 + 7 + 2 + 1 + 4 + 3} \times 100\% = 60\%$$

Dla problemów klasyfikacji, w których liczba danych uczących jest równa we wszystkich klasach, dokładność jest wystarczającą metryką na określenie poprawności klasyfikacji [17].

3.9 Pillow

Ta biblioteka dodaje funkcje przetwarzania obrazów do lokalnego interpretera języka Python. Wspiera formaty plików takie jak JPEG i PNG, które można załadować do programu z lokalnych katalogów na komputerze za pomocą funkcji *open* modułu *Image*. Na tak zainicjalizowanym obrazie, teraz klasy *Image*, można stosować metody tej klasy. Wśród nich przydatną jest *convert*, która umożliwia konwersję obrazu wejściowego pomiędzy trybami *L*, *RGB* i *CMYK* [4]. Dostępne są również metody *resize* oraz *crop*, pozwalające odpowiednio na zmianę rozmiarów zdjęcia i wycięcie fragmentu obrazu.

Rozdział 4

Przygotowanie bazy zdjęć

Na bazę zdjęć wykorzystanych w tej pracy składają się zdjęcia utworzone przez modele generatywne, zostaną one użyte w trenowaniu sieci neuronowej. W bazie znajdują się również realne zdjęcia zrobione aparatem fotograficznym, które zostaną wykorzystane do testowania nauczonej sieci. Zbieranie bazy danych obejmuje więc zarówno generację sztucznych zdjęć, jak i gromadzenie prawdziwych fotografii chodników.

4.1 Generowanie obrazów za pomocą Adobe Firefly

Jako pierwsze narzędzie do generowania zdjęć wykorzystano oferowany przez firmę Adobe model Firefly Image dostępny na oficjalnej stronie Adobe. Program jest dostępny w formie aplikacji przeglądarkowej, więc dużą zaletą korzystania z niego jest łatwy dostęp, który wymaga jedynie posiadania konta Adobe.

Na początku generowano zdjęcia metodą text-to-image jedynie wpisując tekst pomocniczy w wyznaczone do tego miejsce. Jednak wyniki odbiegały kompozycją i ujęciem kamery od tych pożądanych, więc zdecydowano się użyć dostępnej w programie metody image-to-image. Obraz referencyjny został umieszczony jako materiał wizualny wpływający na skład zdjęcia wyjściowego, ponieważ zadaniem dodanego zdjęcia w tym przypadku było kierowanie kompozycją i ujęciem obrazu wyjściowego. Wymagało to użycia jednego realnego zdjęcia zabrudzonego chodnika, którego intensywność wpływu na obraz wyjściowy ustalono w aplikacji na 50%. Wyniki tym razem były znacznie lepsze i bardziej realistyczne. Po eksperymentowaniu z dodatkowymi funkcjami programu zdecydowano się dodać do zdjęć wyjściowych efekt stonowanego koloru, ponieważauważono tendencję modelu do generowania obrazów o bardzo jaskrawych kolorach. Stonowanie barw miało na celu upodobnienie wyników do zdjęć rzeczywistych. Na rysunku 13 przedstawiono wyniki generacji dla trzech wyżej opisanych metod przy użyciu tego samego tekstu pomocniczego: „weak quality camera photo of park pavement covered in leaves, cloudy day”. Ustawiono „Zdjęcie” jako typ zawartości wygenerowanego obrazu.

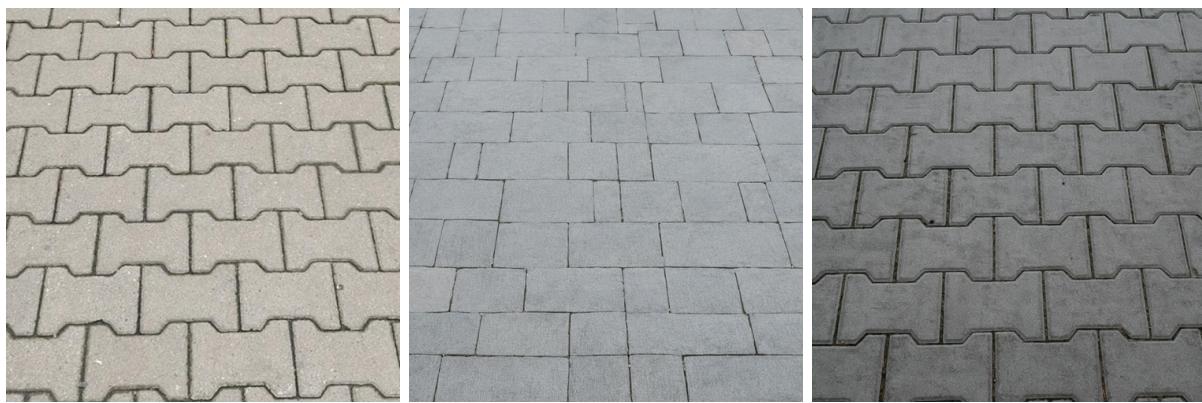


(a) Generacja metodą text-to-image (b) Generacja metodą image-to-image (c) Generacja metodą text-to-image z efektem stonowanego koloru

Rysunek 13. Generacje zdjęć zabrudzonych chodników za pomocą programu Adobe Firefly. Źródło: własne

Z otrzymanych rezultatów bez wątpienia zastosowanie metody image-to-image i stonowania koloru przyniosło najlepszy rezultat.

W ten sposób generowano również ujęcia czystych chodników, zmieniając zdjęcie referencyjne i tekst pomocniczy na: „clean gray pavement with equal sized tiles, cloudy day”. Wspomnienie o różnych rozmiarach kafelków było spowodowane tendencją modelu do generowania chodników z kafelkami różnych rozmiarów, co często odbiega od rzeczywistości. Stosowano wiele zdjęć referencyjnych czystych chodników o różnych kształtach kafelków w celu wzbogacenia wyników. Dodatkowo podczas generacji zmieniano intensywność wpływu zdjęcia pomocniczego między 0%, 50% i 100%, co również zwiększało różnorodność generacji. Pojawia się tutaj jednak kolejne ograniczenie, można wprowadzać jedynie trzy wymienione wartości intensywności. Na rysunku 14 przedstawiono różnice między obrazami wygenerowanymi przy użyciu różnej intensywności zdjęcia referencyjnego w stosunku do tego zdjęcia.



(a) Zdjęcie rzeczywiste, referencyjne (b) Generacja z intensywnością 50% (c) Generacja z intensywnością 100%

Rysunek 14. Wyniki generacji czystych chodników z różną intensywnością zdjęcia referencyjnego z użyciem modelu Firefly Image. Źródło: własne

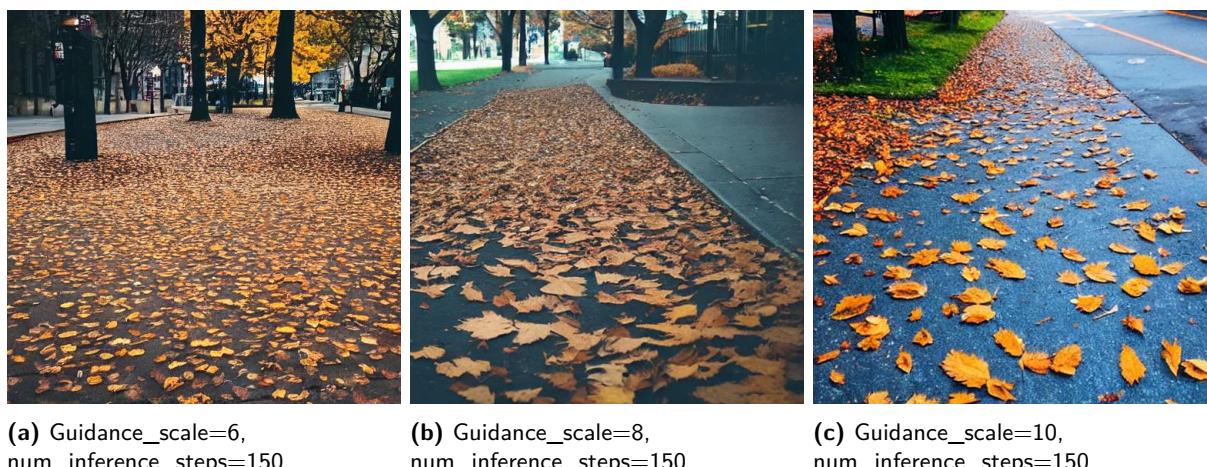
Korzystając z wybranego sposobu wygenerowano taką liczbę zdjęć, na jaką pozwalała bezpłatna usługa w aplikacji Adobe Firefly. Ograniczenia ze strony programu nie są jedynym minusem tego sposobu generowania zdjęć. Jest nim również brak możliwości automatyzacji tego procesu, co oznacza, że każda generacja wiąże się z potrzebą fizycznego kliknięcia w programie. Jednorazowo można wygenerować do czterech zdjęć. Jest to szczególnie niekorzystne w przypadku potrzeby zebrania bardzo dużego zbioru obrazów. Ograniczenia te narzucały wykorzystanie jedynie niewielkiej liczby zdjęć wygenerowanych za pomocą Adobe Firefly w zgromadzonej bazie obrazów. Były one jednak również wykorzystane podczas generacji zdjęć innymi metodami.

4.2 Generacja zabrudzonych chodników za pomocą modeli Stable Diffusion

W tym podrozdziale został przetestowany szereg modeli dyfuzyjnych pod kątem jakości i naturalności wygenerowanych zdjęć chodników zarówno czystych, jak i zabrudzonych. Na początku generowano obrazy jedynie metodami text-to-image oraz image-to-image, jednak później zdecydowano również o wykorzystaniu metody inpainting. Wartości kluczowych argumentów dobierano metodą eksperymentalną.

4.2.1 Metoda text-to-image

Prace rozpoczęto od modelu Stable Diffusion w wersji 1.5, który jest bardzo popularny wśród użytkowników platformy Hugging Face. Na początku generowano obrazy, wprowadzając do modelu jedynie tekst pomocniczy. Na rysunku 15 przedstawiono wyniki generacji z różnymi wartościami parametru *guidance_scale* dla stałej wartości parametru *num_inference_steps* równego 150. We wszystkich generacjach użyto tego samego tekstu pomocniczego: „weak quality camera photo of park pavement covered in leaves”.



(a) Guidance_scale=6,
num_inference_steps=150

(b) Guidance_scale=8,
num_inference_steps=150

(c) Guidance_scale=10,
num_inference_steps=150

Rysunek 15. Wyniki generacji metodą text-to-image dla różnych wartości parametru *guidance_scale*, model SD w wersji 1.5. Źródło: własne

Rozdział 4. Przygotowanie bazy zdjęć

Wartość równa 10 dla tego parametru przyniosła najlepsze rezultaty. Następnie sprawdzono, jak zmiana parametru *num_inference_steps* wpłynie na wyniki generacji. Dla wszystkich przypadków ustawiono parametr *guidance_scale* równy 10. Rezultaty przedstawiono na rysunku 16.



(a) Num_inference_steps=50,
guidance_scale=10

(b) Num_inference_steps=100,
guidance_scale=10

(c) Num_inference_steps=200,
guidance_scale=10

Rysunek 16. Wyniki generacji metodą text-to-image dla różnych wartości parametru *num_inference_steps*, model SD w wersji 1.5. Źródło: własne

Mniejsza wartość parametru *num_inference_steps* negatywnie wpływa na wynik generacji, ale im większa jest ta wartość, tym bardziej czasochłonny jest proces generacji.

Następnie użyto wersji 2.1 modelu Stable Diffusion do generacji obrazów metodą text-to-image. Na rysunku 17 przedstawiono wyniki trzech generacji z różnymi wartościami parametru *guidance_scale* i z liczbą kroków odszumiania równą 150.



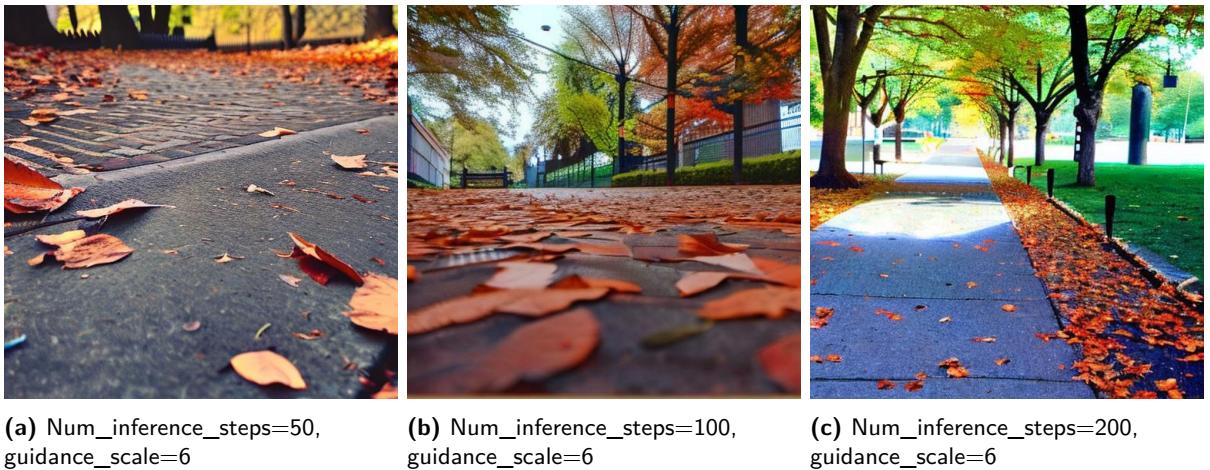
(a) Guidance_scale=6,
num_inference_steps=150

(b) Guidance_scale=8,
num_inference_steps=150

(c) Guidance_scale=10,
num_inference_steps=150

Rysunek 17. Wyniki generacji metodą text-to-image dla różnych wartości parametru *guidance_scale*, model SD w wersji 2.1. Źródło: własne

Tym razem najlepszy rezultat otrzymano dla współczynnika *guidance_scale* równego 6. Zastosowano więc tę wartość przy zmiennej liczbie kroków odszumiania. Wyniki generacji przedstawiono na rysunku 18.



Rysunek 18. Wyniki generacji metodą text-to-image dla różnych wartości parametru *num_inference_steps*, model SD w wersji 2.1. Źródło: własne

Wyniki generacji modelu 2.1 są wyraźnie gorsze od wyników starszego modelu, są mniej naturalne, posiadają więcej atrybutów sztucznej generacji.

Najlepsze rezultaty metodą text-to-image uzyskano stosując wersję 1.5 modelu Stable Diffusion z następującymi wartościami parametrów: *guidance_scale*=10, *num_inference_steps*=200.

4.2.2 Metoda image-to-image

Przekazując zarówno tekst pomocniczy, jak i obraz referencyjny, można osiągnąć rezultaty bardziej zbliżone do oczekiwanych. Metoda image-to-image jest w stanie zapewnić odpowiednie ujęcie kamery generowanych zdjęć, co wpłynie pozytywnie na proces nauczania. Za pomocą modułu *StableDiffusionImg2ImgPipeline* można zaprogramować modele Stable Diffusion do generowania w trybie image-to-image.

Jako pierwszy testowano model w wersji 1.5. Tym razem określany był również trzeci parametr *strength* i jako zdjęcie referencyjne użyto jednego ze zdjęć wygenerowanych za pomocą Adobe Firefly. To samo zdjęcie zostało podane na wejście w każdym z testowanych przypadków. Pierwszym dostosowywanym parametrem był *strength*, czego wyniki przedstawiono na rysunku 19. Wartości pozostałych parametrów przyjęto następująco: *guidance_scale*=8, *num_inference_steps*=150.



(a) Strength=0,55,
guidance_scale=8,
num_inference_steps=150

(b) Strength=0,65,
guidance_scale=8,
num_inference_steps=150

(c) Strength=0,8,
guidance_scale=8,
num_inference_steps=150

Rysunek 19. Wyniki generacji metodą image-to-image dla różnych wartości parametru *strength*, model SD w wersji 1.5. Źródło: własne

Po analizie wyników zdecydowano na zastosowanie wartości 0,55 parametru *strength*. Następnie, podobnie jak w przypadku metody text-to-image, dobierano parametry *num_inference_steps* i *guidance_scale*. Wyniki przeprowadzonych generacji przedstawiono na rysunkach 20 i 21.



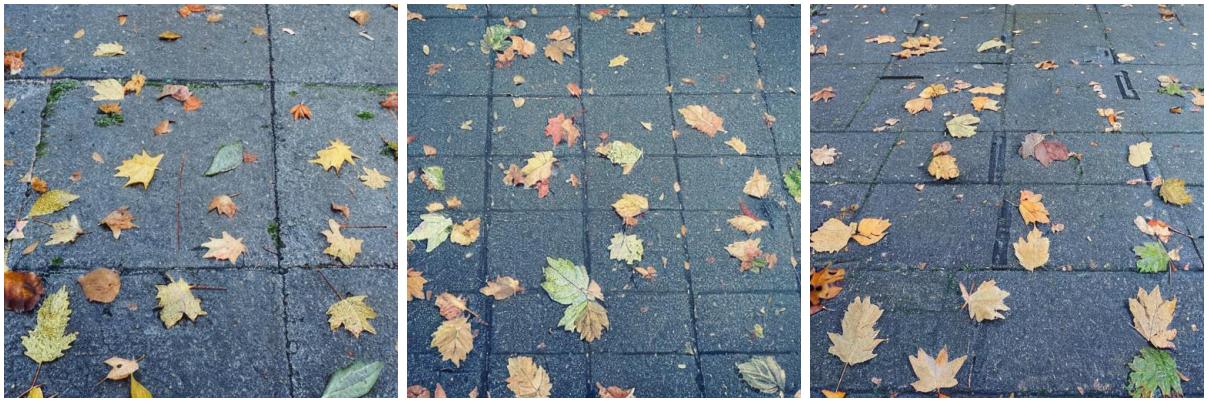
(a) Guidance_scale=4,
strength=0,55,
num_inference_steps=150

(b) Guidance_scale=6,
strength=0,55,
num_inference_steps=150

(c) Guidance_scale=10,
strength=0,55,
num_inference_steps=150

Rysunek 20. Wyniki generacji metodą image-to-image dla różnych wartości parametru *guidance_scale*, model SD w wersji 1.5. Źródło: własne

4.2. Generacja zabrudzonych chodników za pomocą modeli Stable Diffusion



(a) Num_inference_steps=50,
strength=0,55,
guidance_scale=8
(b) Num_inference_steps=100,
strength=0,55,
guidance_scale=8
(c) Num_inference_steps=200,
strength=0,55,
guidance_scale=8

Rysunek 21. Wyniki generacji metodą image-to-image dla różnych wartości parametru *num_inference_steps*, model SD w wersji 1.5. Źródło: własne

Do testowania wpływu liczby kroków wybrano współczynnik *guidance_scale* równy 8, ponieważ jego zastosowanie powodowało generację najbardziej pożądanych wyników. Liczba kroków odszumiania równa 100 okazała się najbardziej optymalna.

Następnie do generowania obrazów metodą image-to-image zastosowano nowszy model Stable Diffusion w wersji 2.1. Dostosowywanie parametrów przeprowadzono identycznie jak w przypadku starszego modelu. Na rysunku 22 przedstawiono wyniki generacji dla różnych wartości parametru *strength*, przy niezmiennych pozostałych parametrach. Użyto tego samego obrazu referencyjnego co przy generacji za pomocą starszej wersji modelu.



(a) Strength=0,55,
guidance_scale=8,
num_inference_steps=150
(b) Strength=0,65,
guidance_scale=8,
num_inference_steps=150
(c) Strength=0,8,
guidance_scale=8,
num_inference_steps=150

Rysunek 22. Wyniki generacji metodą image-to-image dla różnych wartości parametru *strength*, model SD w wersji 2.1. Źródło: własne

Różnica w naturalności i realizmie generowanych zdjęć między dwiema wersjami modelu była natywnie duża, że zaniechano dalszego testowania wartości parametrów wersji 2.1 modelu SD.

Podczas generacji obrazów metodą image-to-image najlepsze rezultaty otrzymano stosując model SD w wersji 1.5 o następujących wartościach parametrów: *guidance_scale*=8, *num_inference_steps*=100, *strength*=0,55.

4.2.3 Metoda inpainting

Zdjęcia zabrudzonych chodników można również generować poprzez wmalowywanie zabrudzeń na zdjęciach czystych chodników, stosując metodę inpainting. Użyto bazy zdjęć czystych chodników jako zdjęcia referencyjne. Pojawił się w tym miejscu problem ze zdjęciem-maską, ponieważ niepożądane jest zadanie jednej maski dla wszystkich generowanych w ten sposób zdjęć, gdyż wtedy zabrudzenia pojawiałyby się tylko w jednym określonym miejscu. Rozwiążaniem mogłoby być na przykład automatyczne generowanie maski dla każdej generacji, jednak na drodze wielu prób i eksperymentów odkryto, że zadowalające rezultaty są otrzymywane po zastosowaniu zdjęcia referencyjnego również jako maski. Przykłady takiej generacji przedstawiono na rysunku 23. Powstałe w ten sposób rezultaty wyglądają naturalnie, a wyniki są różnorodne.



(a) Zdjęcie referencyjne użyte również jako zdjęcie-maska

(b) Pierwszy wynik generacji

(c) Drugi wynik generacji

Rysunek 23. Wyniki generacji metodą inpainting po zastosowaniu zdjęcia referencyjnego jako zdjęcie-maska.
Źródło: własne

Podobnie jak w poprzednich przypadkach, przeprowadzono dobór parametrów dla kilku modeli w celu znalezienia modelu generującego zdjęcie możliwie jak najbardziej naturalne i rzeczywiste. We wszystkich generacjach testowych zastosowano to samo zdjęcie referencyjne, a co za tym idzie, również to samo zdjęcie-maskę. Jako tekst pomocniczy wprowadzono: „park pavement covered in leaves”.

Jako pierwszy testowano standardowy model Stable Diffusion w wersji 1.5, na generację metodą inpainting z jego użyciem pozwala moduł *StableDiffusionInpaintPipeline*. Na początku

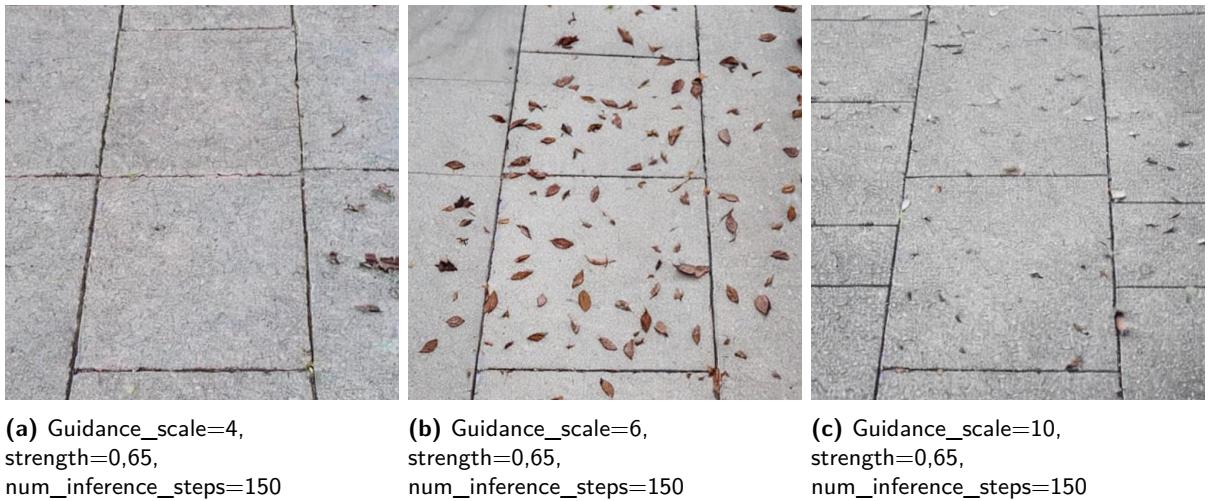
4.2. Generacja zabrudzonych chodników za pomocą modeli Stable Diffusion

dobierany był parametr *strength*. Na rysunku 24 pokazano wyniki generacji z różnymi wartościami tego parametru. Wartości pozostałych argumentów ustawiono następująco: *guidance_scale*=8, *num_inference_steps*=150.

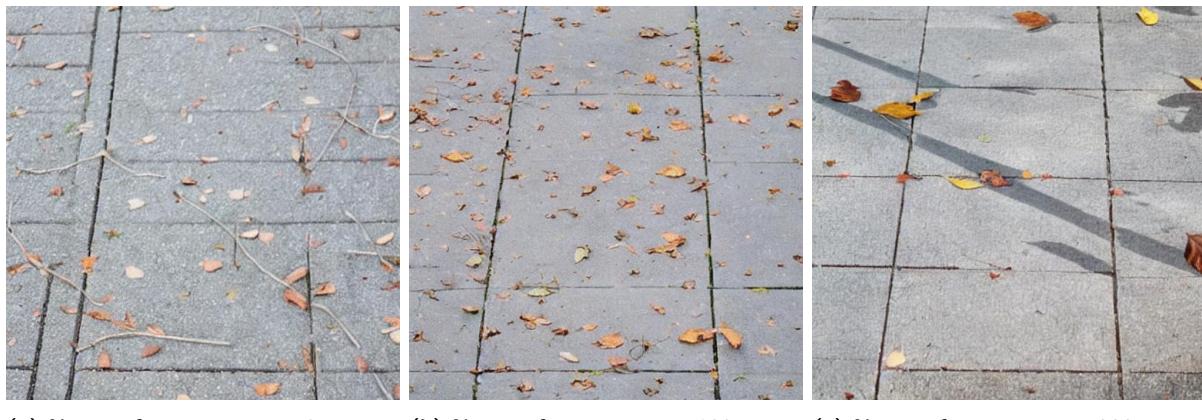


Rysunek 24. Wyniki generacji metodą inpainting dla różnych wartości parametru *strength*, model SD w wersji 1.5. Źródło: własne

Do dalszych testów wybrano 0,65 jako wartość parametru *strength*. Na rysunkach 25 i 26 przedstawiono wyniki generacji z odpowiednio zmiennymi parametrami *guidance_scale* i *num_inference_steps*.



Rysunek 25. Wyniki generacji metodą inpainting dla różnych wartości parametru *guidance_scale*, model SD w wersji 1.5. Źródło: własne

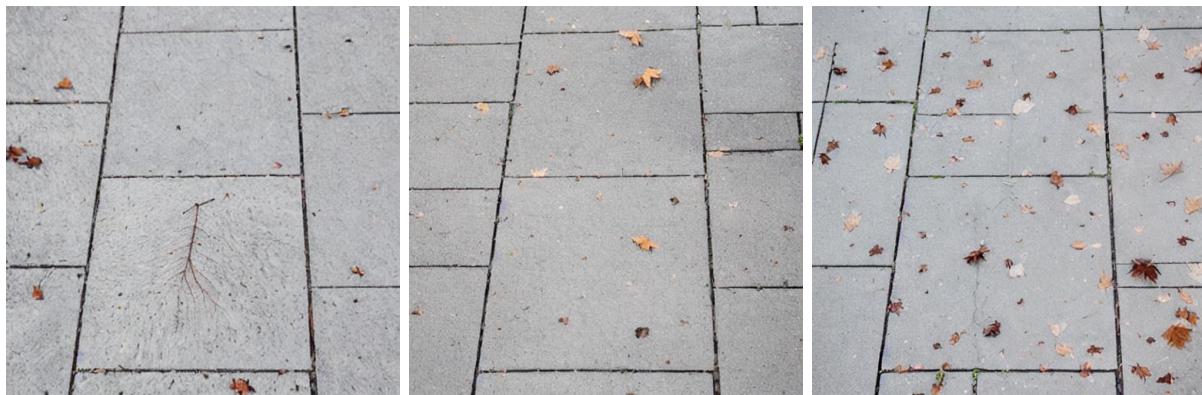


(a) Num_inference_steps=50,
strength=0,65,
guidance_scale=6
(b) Num_inference_steps=100,
strength=0,65,
guidance_scale=6
(c) Num_inference_steps=200,
strength=0,65,
guidance_scale=6

Rysunek 26. Wyniki generacji metodą inpainting dla różnych wartości parametru *num_inference_steps*, model SD w wersji 1.5. Źródło: własne

Dla generacji przy różnych wartościach parametru *num_inference_steps* wybrano *guidance_scale*=6, natomiast spośród wartości parametrów określających liczbę kroków odszumiania wybrano 100.

Następnym testowanym modelem był *stable-diffusion-inpainting*, który jest specjalnie dostrojony do generowania obrazów metodą inpainting. Dobieranie najoptymalniejszych wartości parametrów zostało przeprowadzone w ten sam sposób, co w poprzednim modelu. W każdej generacji użyto tego samego zdjęcia referencyjnego i tekstu pomocniczego, co przy testowaniu modelu Stable Diffusion w wersji 1.5. Jako obraz-maskę w tym przypadku również zastosowano obraz referencyjny. Na rysunku 27 przedstawiono wybrane wyniki generacji z różnymi parametrami podanymi do modelu.

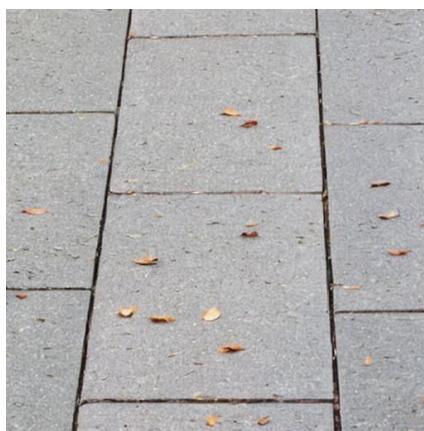


(a) Guidance_scale=6,
num_inference_steps=150,
strength=0,65
(b) Guidance_scale=8,
num_inference_steps=150,
strength=0,65
(c) Guidance_scale=10,
num_inference_steps=150,
strength=0,65

Rysunek 27. Wyniki generacji metodą inpainting dla różnych wartości parametrów, model stable-diffusion-inpainting. Źródło: własne

Najlepsze wyniki generował model z parametrami: $guidance_scale=8$, $num_inference_steps=150$, $strength=0,65$.

Ostatnim testowanym modelem generującym obrazy metodą inpainting był stable-diffusion-2-inpainting, który jest nowszą wersją wcześniej testowanego modelu. Parametry dobrano, stosując niezmiennie to samo zdjęcie referencyjne i zdjęcie-maskę oraz ten sam tekst pomocniczy: „park pavement covered in leaves”. Spośród testowanych wartości parametrów najlepsze rezultaty otrzymano po zastosowaniu $strength=0,55$, $guidance_step=8$ i $num_inference_steps=150$. Wynik generacji z wymienionymi wartościami parametrów przedstawiono na rysunku 28



Rysunek 28. Obraz wyjściowy uzyskany po zastosowaniu optymalnych wartości parametrów, model stable-diffusion-2-inpainting. Źródło: własne

Generując obrazy metodą inpainting, najlepsze i najbardziej różnorodne wyniki uzyskano po zastosowaniu wersji 1.5 modelu Stable Diffusion o następujących wartościach parametrów: $guidance_scale=6$, $num_inference_steps=100$, $strength=0,65$.

4.3 Generacja zdjęć większych grup liści

Podczas generacji zdjęć zabrudzonych chodników wymienionymi we wcześniejszych podrozdziałach metodami, zauważono problem z generacją skupisk liści. Używane wcześniej zdjęcia referencyjne i teksty pomocnicze nie pozwalały na generowanie większych grup zebranych liści, które często pojawiają się na chodnikach i też powinny być klasyfikowane jako zabrudzenie. Do masowej generacji tych zdjęć użyto nowych zdjęć referencyjnych wygenerowanych za pomocą programu Adobe Firefly i nowego tekstu pomocniczego „park pavement totally covered in leaves, cloudy day”. Generowanie odbyło się stosując jedynie model SD w wersji 1.5 i metodę image-to-image, ponieważ zarówno modele generujące metodą text-to-image, jak i te stosujące do generacji metodę inpainting, nie radziły sobie z tym problemem. Na rysunku 29 przedstawiono przykładowe wyniki generacji większych skupisk liści wraz z zastosowanymi w ich generacji parametrami.



(a) Guidance_scale=4,
num_inference_steps=150,
strength=0,45

(b) Guidance_scale=6,
num_inference_steps=100,
strength=0,45

(c) Guidance_scale=6,
num_inference_steps=150,
strength=0,65

Rysunek 29. Wyniki generacji skupisk liści metodą image-to-image, model SD w wersji 1.5. Źródło: własne

Najlepsze i najbardziej różnorodne wyniki generował model z parametrami: *guidance_scale*=6, *num_inference_steps*=100, *strength*=0,45.

4.4 Generacja zdjęć czystych chodników

Do generowania chodników niezabrudzonych na pewno nie można wykorzystać metody inpainting. Metoda text-to-image nie przynosiła zadowalających rezultatów. Czyste chodniki były generowane metodą image-to-image wykorzystując model SD w wersji 1.5, który stosując tę metodę w przypadku zabrudzonych chodników, generował najlepsze wyniki. Do generacji wykorzystano bazę zdjęć chodników wygenerowanych za pomocą programu Adobe Firefly jako zdjęcia referencyjne. Wprowadzono tekst pomocniczy: „clean gray pavement with equal sized tiles, cloudy day”. Na rysunku 30 przedstawiono wyniki generacji z użyciem różnych parametrów.



Rysunek 30. Wyniki generacji czystych chodników metodą image-to-image, model SD w wersji 1.5. Źródło: własne

Najlepsze wyniki, model SD 1.5 generował z parametrami: *guidance_scale*=6, *num_inference_steps*=150, *strength*=0,55.

4.5 Podsumowanie generacji zdjęć chodników

Zdjęcia ostatecznie użyte do bazy danych trenujących, generowano za pomocą programu Adobe Firefly metodą image-to-image, oraz używając wersji 1.5 modelu SD metodami image-to-image i inpainting, odpowiednio dzięki modułom *StableDiffusionImg2ImgPipeline* i *StableDiffusionInpaintPipeline*, stosując najbardziej optymalne parametry uzyskane dzięki testom opisany w poprzednich podrozdziałach. Nie wszystkie wygenerowane obrazy były gotowe do umieszczenia w bazie danych, część trzeba było wykluczyć, ponieważ posiadały zbyt dużo artefaktów sztucznej generacji, część natomiast należało przyciąć. Łącznie jako bazę danych do trenowania sieci neuronowej użyto 15 tysięcy zdjęć. W tabeli 2 umieszczono liczbę zdjęć, których uzyto do trenowania sieci, wygenerowanych określoną metodą.

Tabela 2. Liczba zdjęć chodników wygenerowanych przez poszczególne modele generatywne. Źródło: własne

Zabrudzenie/Model, metoda	Adobe Firefly, image-to-image	SD 1.5, image-to-image	SD 1.5, inpainting
Chodniki czyste	202	4798	0
Chodniki zabrudzone	859	5930	3211
Suma	1061	10728	3211

W bazie danych zdjęć trenujących zdecydowaną większość stanowiły zdjęcia wygenerowane używając modelu SD 1.5 i metody image-to-image. Ten sposób generacji zdjęć okazał się najlepszy ze względu na możliwość generowania zabrudzeń różnego stopnia, na dużą skalę.

4.6 Problemy podczas generacji zdjęć

Podczas generacji zdjęć za pomocą modeli dyfuzyjnych, metodą image-to-image, pojawił się problem zwalniania procesu generacji, który był mierzony w iteracjach na sekundę. Okazało się, że rozmiary zdjęć referencyjnych nie zostały sprawdzone. Niektóre zdjęcia były formatu PNG i często miały rozmiar większy niż 1 MB, co było powodem zwalniania programu. Zmiana formatu wszystkich zdjęć referencyjnych na JPEG okazała się rozwiązaniem.

4.7 Zbiór rzeczywistych zdjęć chodników

Zbiór danych testujących nauczoną sieć neuronową zgodnie z założeniami pracy ma składać się ze zdjęć rzeczywistych, których przynależność do określonej klasy będzie oceniać nauczona sieć neuronowa. Zdjęcia robiono aparatem fotograficznym wbudowanym w telefon komórkowy firmy Samsung, model Galaxy S10 Lite. Zdjęcia chodników robione były na terenie Warszawy, w pobliżu centrum. Łącznie zebrano prawie 3 tysiące zdjęć chodników zabrudzonych w różnym stopniu. Na rysunkach 31, 32 i 33 przedstawiono przykładowe fotografie warszawskich chodników, które w rozdziale 5. zostały przydzielone odpowiednio do klasy 0, 1 i 2.



Rysunek 31. Rzeczywiste zdjęcia chodników ze zbioru pierwszego, przydzielone do klasy 0. Źródło: własne



Rysunek 32. Rzeczywiste zdjęcia chodników ze zbioru pierwszego, przydzielone do klasy 1. Źródło: własne

4.7. Zbiór rzeczywistych zdjęć chodników



Rysunek 33. Rzeczywiste zdjęcia chodników ze zbioru pierwszego, przydzielone do klasy 2. Źródło: własne

Do testowania nauczonych modeli wykorzystano również drugi zbiór zdjęć rzeczywistych, który został zgromadzony i wykorzystany w pracy [19]. Niestety, wiele zdjęć z tego zbioru zostało zniszczonych przez liczne kompresje. Pomimo tej niedogodności, zdecydowano się użyć tej bazy zdjęć do testów w celu możliwości lepszego porównania wyników tej pracy z wynikami prac poprzednich. Ponieważ do trenowania sieci w tej pracy użyto wygenerowanego zbioru danych, można było wykorzystać cały zbiór zdjęć ze wspomnianej pracy do testowania wytrenowanych sieci. Na rysunkach 34, 35 i 36 przedstawiono przykładowe zdjęcia chodników ze wspomnianej bazy, które w rozdziale 5. zostały przydzielone odpowiednio do klasy 0, 1 i 2.



Rysunek 34. Rzeczywiste zdjęcia chodników ze zbioru drugiego, przydzielone do klasy 0. Źródło: [19]

Rozdział 4. Przygotowanie bazy zdjęć



Rysunek 35. Rzeczywiste zdjęcia chodników ze zbioru drugiego, przydzielone do klasy 1. Źródło: [19]



Rysunek 36. Rzeczywiste zdjęcia chodników ze zbioru drugiego, przydzielone do klasy 2. Źródło: [19]

Rozdział 5

Implementacja rozwiązania

W tym rozdziale opisane zostały prace nad realizacją procesu uczenia sieci neuronowych metodą transfer learning oraz późniejsze testowanie doczonych sieci, wykorzystując oba zbiory danych zdjęć rzeczywistych. Przeprowadzono również analizę otrzymanych wyników.

5.1 Przygotowanie bazy danych do nauczania

5.1.1 Klasyfikacja danych

W tej pracy zbiory wygenerowanych zdjęć i zdjęć rzeczywistych zostały podzielone na 3 klasy zabrudzenia. Aby ułatwić kategoryzację, podczas podziału posługiwano się pewnymi przyjętymi kryteriami:

Klasa 0: chodnik czysty, mogą pojawić się pojedyncze, drobne liście.

Klasa 1: powierzchnia zajmowana przez liście zdecydowanie mniejsza od powierzchni widzianego czystego chodnika, widoczne odstępy pomiędzy leżącymi liśćmi, nie ma większych grup liści (maksymalnie po kilka liści na grupę).

Klasa 2: widoczna powierzchnia zajmowana przez liście bliska lub większa od połowy powierzchni widzianego czystego chodnika, mogą występować większe grupy liści.

Łącznie w bazie danych biorących udział w nauczaniu umieszczono po 5000 zdjęć na klasę, natomiast z bazy danych testujących składającej się ze zdjęć rzeczywistych wybrano po 917 zdjęć na klasę. Drugi zbiór testowy, zgromadzony przez mgr inż. Adama Szewczyka, również został podzielony na 3 klasy zgodnie z powyższymi wytycznymi. W tym przypadku w każdej klasie znalazło się po 1400 zdjęć. Na rysunku 37 przedstawiono przykładowe wygenerowane zdjęcia chodników należących do trzech różnych klas zabrudzenia.



Rysunek 37. Zdjęcia chodników należących do trzech różnych klas zabrudzenia. Źródło: własne

5.1.2 Podział i transformacje danych

Zgromadzona baza wygenerowanych zdjęć musi przejść odpowiednie podziały i transformacje, zanim znajdzie się na wejściu sieci neuronowej podczas procesu nauczania.

Zbiór danych został podzielony na dane treningowe, które mają wpływ na zmianę parametrów sieci podczas uczenia i dane walidacyjne, na podstawie których można ocenić poziom nauczenia sieci i zadecydować o przerwaniu procesu dostrajania parametrów sieci. Podział danych na opisane wcześniej dwa podzbiory był przeprowadzony w sposób losowy z zachowaniem równowagi w klasach, tzn. zarówno zbiór danych treningowych, jak i danych walidacyjnych posiadał równą liczbę zdjęć chodników z każdej klasy zabrudzenia. Proporcje liczby zdjęć w obu grupach wynosiły: 80% danych w zbiorze treningowym i 20% danych w zbiorze walidacyjnym.

Następnie zbiór danych treningowych został poddany transformacjom w celu powiększenia zbioru danych (ang. *data augmentation*). Wprowadzanie do modelu w każdej epoce uczącej różnych wariacji tych samych zdjęć treningowych ma pozytywny wpływ na proces nauczania i otrzymywane wyniki. Celem jest również przekształcenie wprowadzanych do modelu danych do formatu, który sieć jest w stanie przetworzyć. Implementacja transformacji w języku Python wygląda następująco:

```
'train': transforms.Compose([
    transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),
    transforms.RandomRotation(degrees=15),
    transforms.RandomHorizontalFlip(),
    transforms.CenterCrop(size=224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                      [0.229, 0.224, 0.225])]
```

Użyto w tym miejscu modułu *transforms* z pakietu *torchvision*. Transformacja *RandomResizedCrop* przycina obraz wejściowy o losowy rozmiar w podanym zakresie skali względem oryginalnego obrazu. Rozmiar przyciętego obrazu jest następnie zmieniany do podanych w tym przypadku 256×256 pikseli. Operacja *RandomRotation* rotuje obraz o losowy kąt w podanym zakresie (w tym przypadku od -15 do

15 stopni). *RandomHorizontalFlip* wykonuje przerzut obrazu w poziomie z szansą 50%. Transformacja *CenterCrop* wycina środkową część zdjęcia, jako kwadrat o podanym boku, w tym przypadku bok ma rozmiar 224 piksele, ponieważ taki był rozmiar zdjęć, na których wstępnie wytrenowane zostały modele sieci neuronowych wykorzystane w tej pracy. Następnie obraz wejściowy jest konwertowany do formatu tensora zmiennoprzecinkowego, który następnie jest normalizowany przez średnią i odchylenie standardowe obliczone na zestawie danych ImageNet, który jest powszechnie używany do trenowania modeli głębokiego uczenia. Na rysunku 38 przedstawiono kilka transformacji tego samego zdjęcia.



Rysunek 38. Cztery losowe transformacje zdjęcia zabrudzonego chodnika. Źródło: własne

Zbiór danych walidacyjnych nie musi być zwiększany i został poddany jedynie zmianie rozmiaru do oczekiwanych 224×224 pikseli, oraz niezbędnym transformacjom *ToTensor* i *Normalize*.

5.2 Przygotowanie modelu sieci neuronowej do nauczania

5.2.1 ConvNeXt

Do nauczania metodą transfer learning został wybrany model ConvNeXt. Pakiet *torchvision.models* pozwala na korzystanie ze wszystkich modeli znajdujących się na oficjalnej stronie PyTorch. Model ConvNeXt jest dostępny w czterech rozmiarach: *Tiny*, *Small*, *Base* i *Large*, różnią się one między sobą ilością parametrów i precyzją klasyfikacji na zbiorze testowym. W tej pracy został wykorzystany model w wersji *Small*, ponieważ precyzja klasyfikacji nie różni się bardzo od większych modeli (o 0,8 p.p. mniejsza niż w wersji *Large*), natomiast jest dużo mniejszy, przez co posiada mniejsze wymagania sprzętowe. Jednak oryginalny model pobrany z dostępnej biblioteki modeli na wyjściu ma zaimplementowaną klasyfikację obrazów na 1000 klas, czego przyczyną jest wstępne uczenie tego modelu na zbiorze danych ImageNet. W badanym problemie zbiór danych został podzielony na 3 klasy. Aby przystosować model do nowych warunków klasyfikacji, należy wprowadzić zmiany w warstwie wyjściowej modelu. W tym celu został wprowadzony następujący kod:

```
model_conv.classifier[2] = torch.nn.Linear(
    in_features=model_conv.classifier[2].in_features,
    out_features=3
)
```

Liczba klas, które ma rozróżniać klasyfikator, została w tym przypadku ustawiona na 3.

Aby rozpocząć transfer learning należy jeszcze określić optymalizator i funkcję kosztu, które pełnią kluczowe role w procesie uczenia. Po przeanalizowaniu podobnych problemów nauczania, jako funkcję kosztu wybrano *CrossEntropyLoss*, a za optymalizator przyjęto algorytm SGD z określonym współczynnikiem uczenia (ang. *learning rate*).

5.2.2 ResNet

W celu zestawienia wyników nauczania została wytrenowana również sieć ResNet z 2015 roku, która została użyta w podobnym problemie w pracy [19]. W bibliotece torchvision dostępne jest pięć modeli sieci ResNet, które różniły się między sobą liczbą warstw w architekturze sieci. Wybrano najbardziej złożony model ResNet152, składający się ze 152 warstw [3], ponieważ posiada najwyższą dokładność na zbiorze testowym wśród dostępnych modeli, wynoszącą 78,3%. Wybrano model wstępnie wytrenowany na zbiorze "IMAGENET1K_V1", tym samym, na którym został wytrenowany model ConvNeXt. Minusem zastosowania tego modelu był jego rozmiar. Liczba parametrów w tym przypadku przewyższała liczbę parametrów użytej sieci ConvNeXt w wersji Small o 10M. Funkcję kosztu i optymalizator zastosowano taki sam jak w przypadku sieci z 2022 roku. Aby dostosować model do problemu z tej pracy, zmieniono liczbę wyjść w ostatniej warstwie sieci (warstwie w pełni połączonej) z wartością wstępnie ustalonej na 1000 (na potrzeby zbioru ImageNet), na wartość 3, równą liczbie klas w tym problemie.

5.3 Optymalizacja hiperparametrów nauczania

Na wyniki nauczania ma wpływ wiele hiperparametrów, ten podrozdział został przeznaczony na opis doboru ich najbardziej optymalnych wartości dla obu trenowanych sieci. Wyznacznikiem podczas doboru była precyzaja klasyfikacji na zbiorze walidacyjnym.

5.3.1 Dobór rozmiaru paczki uczącej

Paczka ucząca (ang. *batchsize*), określa liczbę danych trenujących przesyłanych jednorazowo na wejście sieci neuronowej w procesie nauczania. Wyniki uczenia z czterema rozmiarami paczki uczącej umieszczone w tabeli 3.

Tabela 3. Precyzaja klasyfikacji na zbiorze walidacyjnym w stosunku do różnych rozmiarów paczki uczącej.
Źródło: własne

Rozmiar paczki uczącej	Dokładność na zbiorze walidacyjnym sieć ConvNeXt	Dokładność na zbiorze walidacyjnym sieć ResNet152
32	91,43%	91,43%
16	92,63%	91,80%
8	93,70%	90,67%
4	94,21%	88,10%

W przypadku sieci ConvNeXt, wybrano rozmiar paczki uczącej równy 4, ponieważ zapewniał najwyższą dokładność klasyfikacji. Sieć ResNet152 najlepszy wynik osiągnęła dla rozmiaru paczki równej 16. Powyższe testy przedstawiają dokładność po 5 epokach uczących i z wstępnie ustawionym współczynnikiem uczenia równym 0,0005.

5.3.2 Dobór współczynnika uczenia

Wartość współczynnika uczenia jest hiperparametrem ustawianym w algorytmie optymalizatora. W jego doborze istotne jest, aby nie wprowadzić zbyt dużej wartości, gdyż wtedy może dojść do pominięcia najbardziej optymalnej drogi do minimalizacji funkcji kosztu. Jednak nie może on być również zbyt mały, ponieważ znacznie wydłuży to czas ustalania optymalnego minimum funkcji kosztu, co w tym przypadku jest niepożądane. Podczas dobierania kierowano się osiągniętą dokładnością na zbiorze walidacyjnym w pierwszych 100 epokach uczących. Wyniki uczenia obu modeli z trzema różnymi wartościami współczynnika uczenia, umieszczone w tabeli 4.

Tabela 4. Dokładność klasyfikacji na zbiorze walidacyjnym w stosunku do różnych wartości współczynnika uczenia. Źródło: własne

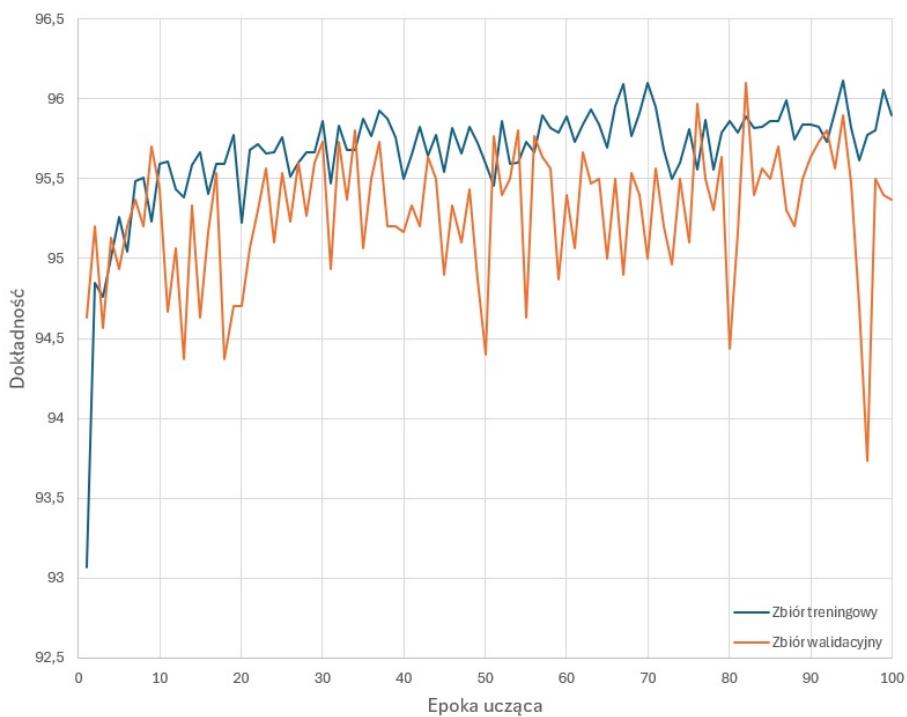
Współczynnik uczenia	Dokładność na zbiorze walidacyjnym sieć ConvNeXt	Dokładność na zbiorze walidacyjnym sieć ResNet152
0,0005	95,67%	93,97%
0,001	95,76%	94,3%
0,005	96,1%	94,73%

Zarówno w przypadku sieci ConvNeXt, jak i ResNet wybrano wartość współczynnika uczenia równą 0,005, ponieważ po jej zastosowaniu dokładność na zbiorze testowym osiągnęła najwyższą wartość na przestrzeni 100 epok uczących.

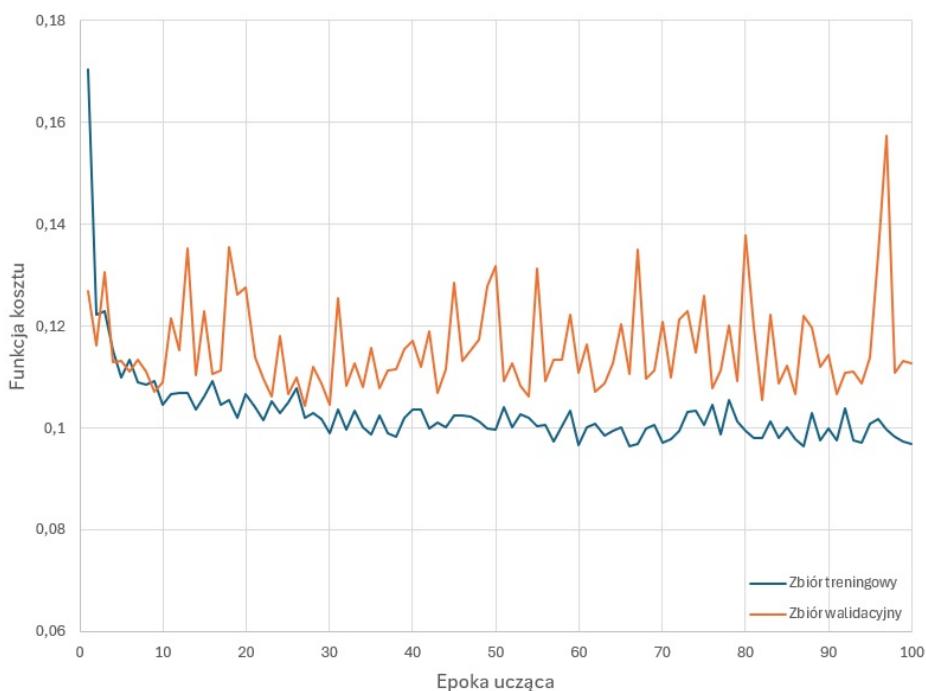
5.4 Wyniki nauczania sieci neuronowych

5.4.1 Sieć ConvNeXt

Na rysunkach 39 i 40 przedstawiono przebiegi nauczania sieci ConvNeXt trwającego 100 epok uczących z ustalonymi następującymi hiperparametrami sieci: rozmiar paczki uczącej = 4, współczynnik uczenia = 0,005.



Rysunek 39. Wartość dokładności wyrażonej w procentach, w zależności od numeru epoki uczącej. Przebieg nauczania sieci ConvNeXt. Źródło: własne



Rysunek 40. Wartość funkcji kosztu, w zależności od numeru epoki uczącej. Przebieg nauczania sieci ConvNeXt. Źródło: własne

Zbiory danych z każdej klasy były równe, więc wartość dokładności jest wystarczającą metryką określającą poziom nauczenia sieci. Dokładność zbioru walidacyjnego była największa w epoce 82. i wyniosła 96,1%. Zaobserwowano, że ustaliła się ona bardzo szybko blisko 95%, co mogło być spowodowane bardzo dużym zbiorem treningowym, dzięki któremu sieć szybko uczyła się w początkowych epokach. Model z 82. epoki wykorzystano w klasyfikacji danych testujących. Wyniki przedstawiono na rysunku 41 w formie macierzy pomyłek. Jako bazy danych testujących użyto zebranej bazy zdjęć rzeczywistych chodników.



Rysunek 41. Macierz pomyłek dla zbioru testowego nr 1. Źródło: własne

Wyniki oceny na zbiorze testowym są bardzo dobre. Udało się osiągnąć dokładność 92,8%. Model nie pomylił się o więcej niż o jedną klasę, co znaczy, że nie oceniał bardzo zabrudzonych chodników jako czystych i na odwrót. Ta cecha klasyfikatora jest bardzo pożądana. Najwięcej niepoprawnych ocen padło w przypadku odróżnienia bardzo-zabrudzonych od średnio-zabrudzonych chodników, co nie jest oczywiste również w przypadku oceny podejmowanej przez człowieka, ponieważ nie istnieje jednoznaczna granica oddzielająca jedną klasę od drugiej. Jednak nawet mimo błędów w ocenie, stanowiły one jedynie niewielki ułamek w stosunku do decyzji podjętych poprawnie. Co ciekawe, model ani razu nie zakwalifikował średnio-zabrudzonego chodnika jako czystego, natomiast zdarzało się mu ocenić zdjęcie czystego chodnika jako chodnik średnio-zabrudzony. Mogło być to spowodowane tym, że podczas ręcznego podziału zdjęcia testowe z niewielkimi zabrudzeniami w formie gałęzi, czy drobnych śmieci, które kształtem przypominały małe liście, były często przydzielane do zbioru czystych chodników, ponieważ liści na nich nie było.

Na rysunku 42 przedstawiono przykładowe błędne klasyfikacje zdjęć ze zbioru testowego nr 1.



Rysunek 42. Błędne klasyfikacje otrzymane stosując wyuczoną sieć ConvNeXt i zbiór testowy nr 1. Źródło: własne

W pierwszym przypadku można wskazać błąd klasyfikatora, ponieważ chodnik jest czysty, a występujące jedynie drobne liście w przerwach pomiędzy płytami są ledwie dostrzegalne. Jednak w pozostałych dwóch przypadkach ciężko jest jednoznacznie określić, do jakiej klasy zabrudzenia powinny one należeć. Pomimo tego, że w tej pracy starano się to jak najdokładniej określić, zawsze znajdują się zdjęcia na granicy dwóch klas i w powyższych przypadkach właśnie taka sytuacja ma miejsce.

Przeprowadzono również testy na drugim zbiorze zdjęć rzeczywistych liści, który został wykorzystany w pracy [19]. Wyniki w postaci macierzy pomyłek przedstawiono na rysunku 43.



Rysunek 43. Macierz pomyłek dla zbioru testowego nr 2. Źródło: własne

W tym przypadku wyniki są trochę gorsze niż w przypadku poprzedniego zbioru testowego. Dokładność na zbiorze testowym wynosi 85,2%. Wynik ten jest bardzo podobny do rezultatu osiągniętego w pracy [19], w której zdjęcia rzeczywiste z tej samej bazy danych brały udział nie tylko w testowaniu, ale również i w samym nauczaniu sieci neuronowej. Udało się wówczas osiągnąć dokładność na zbiorze testowym równą 85%. Mając na względzie to, że w niniejszej pracy zdjęcia w zbiorze uczącym sieć były wygenerowane, można wynik ten uznać za zadowalający.

Klasyfikator najczęściej mylił się w przypadku oceny chodnika średnio-zabrudzonego i często przyporządkowywał go do chodników czystych. Tym razem klasyfikator pomylił się o więcej niż 1 klasę, jednak liczba zdjęć błędnie sklasyfikowanych o dwie klasy stanowi zaledwie 1,9% całego zbioru testowego, zatem błąd ten można uznać za znikomy.

Na rysunku 44 przedstawiono przykładowe błędne klasyfikacje zdjęć ze zbioru testowego nr 2.



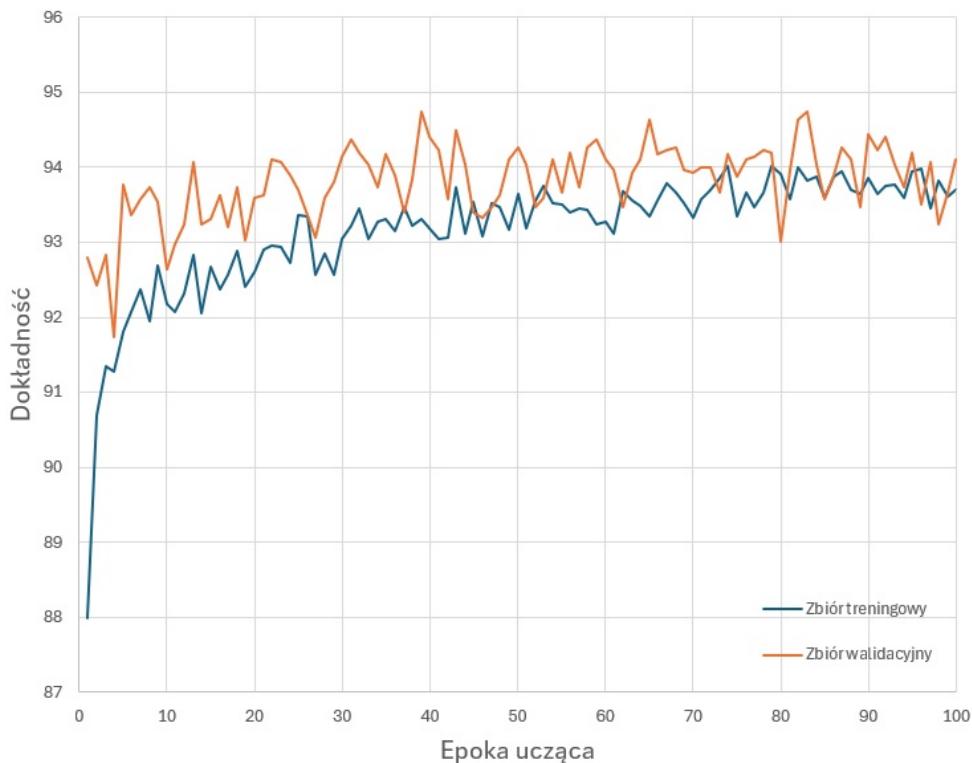
Rysunek 44. Błędne klasyfikacje otrzymane stosując wyuczoną sieć ConvNeXt i zbiór testowy nr 2. Źródło: własne

Warto w tym miejscu zwrócić uwagę na pierwszy przypadek. Klasyfikator pomylił się o dwie klasy i ocenił zdjęcie czystego kamiennego chodnika jako chodnik zabrudzony liśćmi. Powodem tej błędnej oceny może być fakt, że w zbiorze wygenerowanych chodników, którymi została nauczona sieć, nie było chodników kamiennych, w związku z czym sieć nie rozpoznała tej powierzchni chodnikowej i być może przez kolor kamieni sklasyfikowała je jako liście.

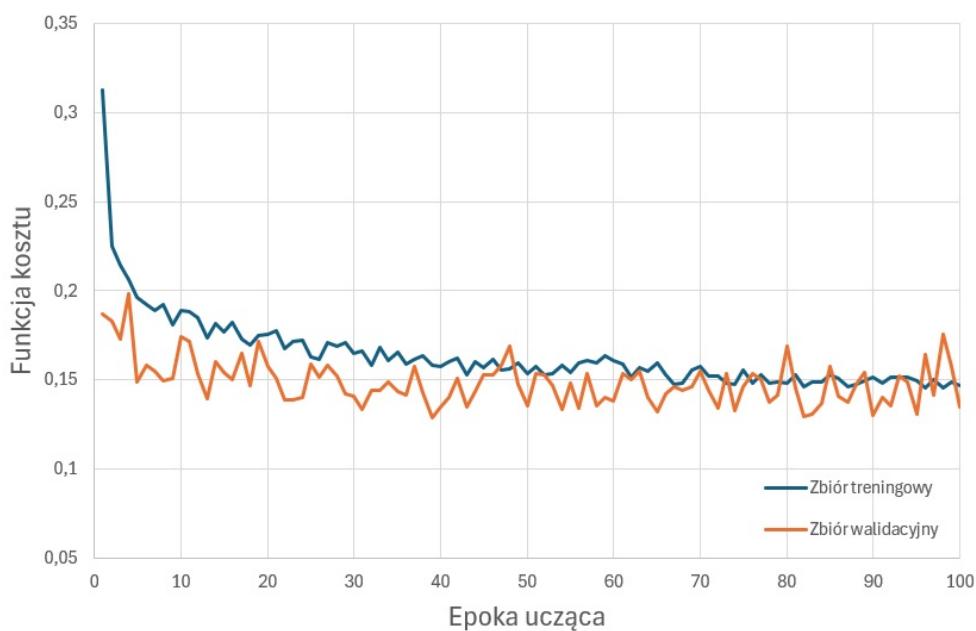
5.4.2 Sieć ResNet152

Na rysunkach 45 i 46 przedstawiono przebiegi nauczania sieci ResNet152 również trwającego 100 epok z ustalonymi następującymi hiperparametrami sieci: rozmiar paczki uczącej = 16, współczynnik uczenia = 0,005.

Rozdział 5. Implementacja rozwiązania



Rysunek 45. Wartość dokładności wyrażonej w procentach, w zależności od numeru epoki uczącej. Przebieg nauczania sieci ResNet152. Źródło: własne

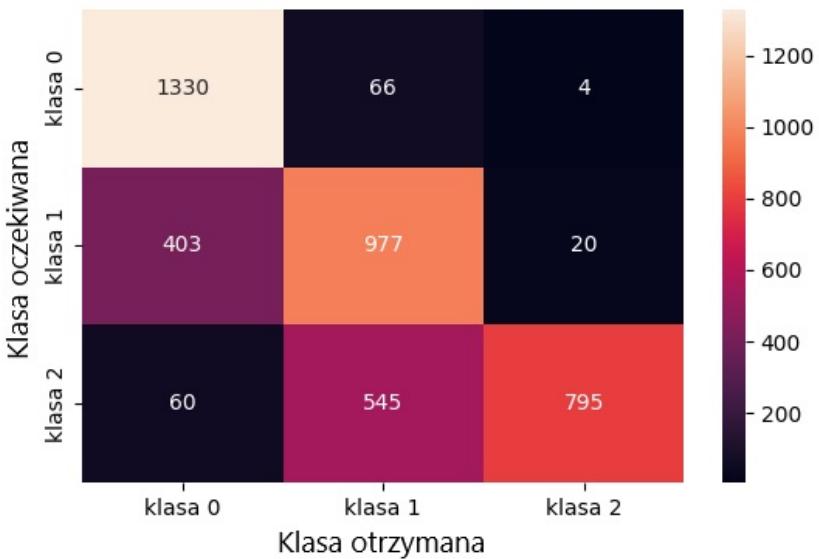


Rysunek 46. Wartość funkcji kosztu, w zależności od numeru epoki uczącej. Przebieg nauczania sieci ResNet152. Źródło: własne

Dokładność zbioru walidacyjnego była największa w epoce 83. i wyniosła 94,73%. W tym przypadku dokładność również ustalała się bardzo szybko blisko 93%. Model z 83. epoki wykorzystano w klasyfikacji danych testujących. Wyniki testów na zbiorze własnym, czyli zbiorze nr 1 oraz gotowym zbiorze nr 2 przedstawiono odpowiednio na rysunkach 47 i 48 w formie macierzy pomyłek.



Rysunek 47. Macierz pomyłek dla zbioru testowego nr 1. Źródło: własne



Rysunek 48. Macierz pomyłek dla zbioru testowego nr 2. Źródło: własne

Wytrenowana sieć ResNet osiągnęła dokładność 90,6% na zbiorze testowym nr 1. Wynik ten różni się od rezultatu osiągniętego za pomocą sieci ConvNeXt jedynie o 2,2 p.p. Rozstrzygającym

Rozdział 5. Implementacja rozwiązania

okazał się wynik na zbiorze testowym nr 2, na którym dokładność klasyfikacji wyniosła 73,8%. Wynik ten jednoznacznie uzasadnia zastosowanie nowszej architektury sieci ConvNeXt do tego problemu, ponieważ różnica w dokładności klasyfikacji między dwiema wytrenowanymi sieciami wynosi aż 11,4 p.p.

Na rysunkach 49 i 50 przedstawiono przykładowe błędne klasyfikacje zdjęć ze zbioru testowego odpowiednio nr 1 i 2, przeprowadzone przez sieć ResNet.



(a) Rzeczywista klasa: 0
Otrzymana klasa: 1

(b) Rzeczywista klasa: 1
Otrzymana klasa: 2

(c) Rzeczywista klasa: 2
Otrzymana klasa: 1

Rysunek 49. Błędne klasyfikacje otrzymane stosując wyuczoną sieć ResNet152 i zbiór testowy nr 1. Źródło: własne



(a) Rzeczywista klasa: 0
Otrzymana klasa: 1

(b) Rzeczywista klasa: 1
Otrzymana klasa: 0

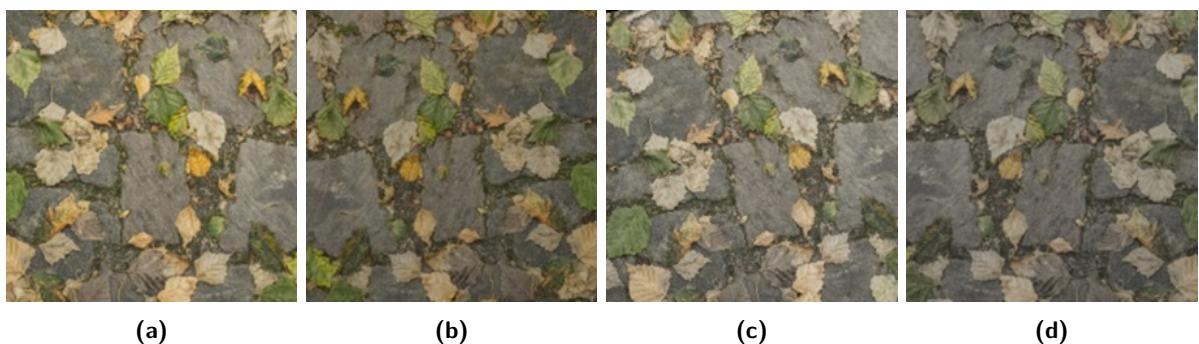
(c) Rzeczywista klasa: 2
Otrzymana klasa: 1

Rysunek 50. Błędne klasyfikacje otrzymane stosując wyuczoną sieć ResNet152 i zbiór testowy nr 2. Źródło: własne

Na powyższych zdjęciach można zauważyc, jak niewielkie bywają różnice pomiędzy dwiema klasami zabrudzenia. Można wskazać jednak również błędy popełnione przez klasyfikator, na przykład na rysunku 50 w podpunkcie (a), sieć prawdopodobnie potraktowała plamki na asfalcie jako liście, błędnie przypisując czystemu chodnikowi klasę „średnio-zabrudzony”.

5.5 Próba poprawy wyników

W celu poprawy rezultatów otrzymanych w poprzednim podrozdziale, wprowadzono dodatkowe transformacje zdjęć wejściowych w procesie uczenia. Wszelkie zmiany wprowadzono tym razem jedynie do sieci, która osiągnęła lepszy rezultat, czyli sieci ConvNeXt. Oprócz wcześniej zaimplementowanych losowych rotacji i przycięć, dodano losową zmianę jasności z zakresu od 0,6 do 1,2 jasności oryginalnej oraz losową zmianę saturacji barw w zakresie od 0,5 do 1. Zmiany te wprowadzono w celu powiększenia bazy zdjęć trenujących. Przykładowe transformacje obrazów wejściowych z dodatkowymi zmianami jasności i saturacji barw przedstawiono na rysunku 51.



Rysunek 51. Cztery transformacje zdjęcia zabrudzonego chodnika. a) Zdjęcie bez dodatkowych transformacji. b) Zdjęcie z dodaną losową zmianą jasności. c) Zdjęcie z dodaną losową zmianą saturacji barw. d) Zdjęcie z dodanymi losowymi zmianami jasności i saturacji barw. Źródło: własne

Przeprowadzono trzy nauczania: z dodatkową losową zmianą jasności, z dodatkową losową zmianą saturacji barw i z dodanymi obiema losowymi zmianami tych wartości w zdjęciach podawanych na wejście sieci w procesie nauczania. Skorzystano z ustalonych najbardziej optymalnych hiperparametrów nauczania. W tabeli 5 umieszczone zostały wyniki dokładności oceny na zbiorze walidacyjnym i obu zbiorach testowych najlepszych modeli z tych trzech procesów nauczania i porównanie ich z najlepszym otrzymanym dotychczas wynikiem.

Tabela 5. Porównanie wyników nauczania z wprowadzonymi dodatkowymi transformacjami obrazów wejściowych. Źródło: własne

Dodatkowa transformacja	Dokładność na zbiorze walidacyjnym	Dokładność na zbiorze testowym nr 1	Dokładność na zbiorze testowym nr 2
brak	96,10%	92,84%	85,24%
losowa zmiana jasności	95,93%	92,58%	84,29%
losowa zmiana saturacji	96,17%	91,64%	86,33%
losowa zmiana jasności i saturacji	95,83%	92,00%	85,93%

Najwyższą dokładność na zbiorze testowym nr 1 otrzymano stosując model bez losowych zmian jasności i saturacji barw zdjęć wejściowych. Zastosowanie dodatkowej transformacji w postaci losowej

Rozdział 5. Implementacja rozwiązania

zmiany saturacji barw zdjęć wejściowych w procesie nauczania spowodowało poprawę klasyfikacji zbioru testowego nr 2 o 1,1 p.p. Niekorzystnym rezultatem tej zmiany był spadek dokładności klasyfikacji na zbiorze testowym nr 1 o 1,2 p.p. W związku z tym ostatecznie wybrano model bez zastosowanych dodatkowych transformacji zdjęć wejściowych w procesie nauczania.

Rozdział 6

Podsumowanie i wnioski

W niniejszej pracy przeprowadzono nauczanie dwóch modeli sieci neuronowych w celu klasyfikacji stopnia zabrudzenia chodników liścimi. Zarówno zbiór danych trenujących sieć, jak i zbiór walidacyjny, który służył do oceny poziomu nauczenia sieci, zostały w pełni wygenerowane przez sztuczną inteligencję. W celu generacji obrazów przetestowano wiele modeli generatywnych i zastosowano wiele metod generacji, stosując różne formy wspomagania tego procesu. Badano zarówno naturalność generowanych zdjęć, jak i możliwość generacji na dużą skalę, w celu uzyskania jak największej liczby danych uczących. Najlepsze wyniki zostały uzyskane, stosując model Stable Diffusion w wersji 1.5 i metody image-to-image oraz inpainting. Do generacji tymi metodami użyto zdjęć referencyjnych wygenerowanych przez model Firefly Image, które również znalazły się w bazie danych uczących, jednak ich liczba była znacznie mniejsza od liczby zdjęć otrzymanych za pomocą modeli dyfuzyjnych. Łącznie do uczenia wykorzystano 15000 sztucznych zdjęć. Dodatkowo zebrano prawie 3000 fotografii rzeczywistych chodników, które zostały użyte do testowania klasyfikatora. Do testów wykorzystano również gotowy zbiór rzeczywistych zdjęć zgromadzony i wykorzystany w pracy magisterskiej na podobny temat. Na potrzeby tej pracy wykorzystano 4200 zdjęć z tego zbioru. Wszystkie zebrane zdjęcia podzielono na 3 klasy zabrudzenia: *czyste, średnio-zabrudzone i bardzo-zabrudzone*. W każdej klasie znalazła się identyczna liczba obrazów.

Następnym etapem pracy było wytrenowanie modelu sieci neuronowej metodą transfer learning, przygotowując go specjalnie do problemu klasyfikacji stopnia zabrudzenia chodników. Do tego zagadnienia wybrano sieć ConvNeXt, której model został udostępniony w 2022 roku i wykorzystano zgromadzony zbiór danych uczących. Na tej samej bazie danych wytrenowano również model ResNet152 z 2015 roku w celu zarówno porównania wyników z poprzednią pracą, jak i uzasadnienia zastosowania nowszej architektury sieci. Podczas procesu nauczania skorzystano z metod powiększania zbioru danych poprzez losową rotację, losowy przerzut w poziomie oraz losowe wycięcie, które tworzyły różne wariacje zdjęć podawanych na wejście sieci w każdej epoce uczącej. Dobrano współczynnik uczenia równy 0,005 i rozmiar paczki uczącej równy 4 jako optymalne hiperparametry nauczania w przypadku sieci ConvNeXt. Natomiast do trenowania sieci ResNet152 wybrano współczynnik uczenia równy 0,005 i rozmiar paczki uczącej równy 16. Podczas uczenia

Rozdział 6. Podsumowanie i wnioski

kierowano się dokładnością na zbiorze walidacyjnym jako wyznacznikiem jakości klasyfikatora. Stosując sieć ConvNeXt udało się uzyskać dokładność 96,1% na zbiorze walidacyjnym, a na zbiorach testowych nr 1 i 2 osiągnięto dokładność odpowiednio 92,8% i 85,2%. Warto zaznaczyć, że klasyfikator podczas testowania na zbiorze nr 1 nie pomylił się ani razu o więcej niż 1 klasę, czyli nie mylił zdjęć bardzo zabrudzonych chodników z chodnikami czystymi. Wyniki osiągnięte przez wytrenowaną sieć ResNet152 to 90,6% i 73,8% odpowiednio na zbiorach testowych nr 1 i 2, co uzasadnia ostateczne zastosowanie sieci ConvNeXt w tym problemie. Próbowano następnie poprawić wynik uzyskany przez sieć z 2022 roku, stosując dodatkowe losowe zmiany saturacji barw i jasności zdjęć treningowych, co przyniosło pewną poprawę wyników na zbiorze testowym nr 2, ale niestety pogorszyło rezultat na zbiorze nr 1, w związku z czym zrezygnowano z tego rozwiązania.

Otrzymane wyniki są zadowalające i pokazują, że użycie wygenerowanych zdjęć do trenowania sieci neuronowej w problemie klasyfikacji jest jak najbardziej uzasadnione. Dodatkowym atutem takiego podejścia jest możliwość zgromadzenia dużej liczby danych uczących przy braku dostępu do zdjęć rzeczywistych spowodowanego na przykład warunkami atmosferycznymi. Zbiory danych biorące udział w nauczaniu i testowaniu sieci, pliki z kodem uczenia sieci ResNet, ConvNeXt i testowaniem oraz wytrenowany model sieci w formacie *pth* udostępniono pod adresem: https://drive.google.com/drive/folders/1eMiWXqwHQh8k8sGyG3Edo8iXgPjf_MdS?usp=sharing

Prace nad generacją sztucznych zdjęć w celu użycia ich do nauczania sieci mogą być rozwijane w wielu kierunkach. W dalszych badaniach można przetestować sieć wyuczoną do klasyfikacji innych rodzajów zabrudzeń, na przykład śniegu lub błota pośniegowego, czego nie udało się w tej pracy osiągnąć przez brak danych testowych, spowodowany niesprzyjającymi warunkami pogodowymi. Ciekawą ścieżką rozwoju byłoby również zaprojektowanie całego systemu generatywno-uczącego, który trenowałby model klasyfikujący określone przez użytkownika zabrudzenie, posługując się samodzielnie wygenerowaną bazą danych trenujących.

Bibliografia

- [1] Dave Bergmann, C. S., *What are Diffusion Models?*, <https://www.ibm.com/think/topics/diffusion-models>, sierp. 2024, data dostępu: 27.11.2024.
- [2] Halawa, K., „Sieci neuronowe i neurosterowniki - MLP, CNN”, <http://krzysztof.halawa.staff.iiar.pwr.wroc.pl/SieciNeuronowe2.pdf>, 2023, data dostępu: 07.01.2025.
- [3] He, K., Zhang, X., Ren, S. i Sun, J., „Deep Residual Learning for Image Recognition”, <https://arxiv.org/pdf/1512.03385>, 2015, data dostępu: 07.01.2025.
- [4] *Image Module*, <https://pillow.readthedocs.io/en/stable/reference/Image.html>, Pillow, data dostępu: 04.12.2024.
- [5] Jansen, P., „TIOBE Index for November 2024”, <https://www.tiobe.com/tiobe-index>.
- [6] Keita, Z., *An Introduction to Convolutional Neural Networks: A Comprehensive Guide to CNNs in Deep Learning*, <https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns>, data dostępu: 27.11.2024.
- [7] Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T. i Xie, S., „A ConvNet for the 2020s”, <https://arxiv.org/abs/2201.03545>, 2022, data dostępu: 27.11.2024.
- [8] Łukomski, J., „Sposób oceny czystości nawierzchni z wykorzystaniem sieci głębokiego uczenia”, Praca dyplomowa inżynierska, Politechnika Warszawska, Wydział Elektryczny, Instytut Sterowania i Elektroniki Przemysłowej, 2022.
- [9] Nielsen, M., „Neural Networks and Deep Learning”, <http://neuralnetworksanddeeplearning.com/chap5.html>, grud. 2019, data dostępu: 07.01.2025.
- [10] NVIDIA CUDA, <https://docs.nvidia.com/cuda/doc/index.html>, NVIDIA, data dostępu: 29.11.2024.
- [11] Osowski, S., *Sieci neuronowe do przetwarzania informacji*. Oficyna Wydawnicza Politechniki Warszawskiej, 2020.
- [12] paperswithcode, *Image Classification on ImageNet*, <https://paperswithcode.com/sota/image-classification-on-imagenet>, data dostępu: 02.12.2024.
- [13] paperswithcode, *Swin Transformer*, <https://paperswithcode.com/method/swin-transformer>, data dostępu: 27.11.2024.

Bibliografia

- [14] Sadhula, S. N., *The McCulloch-Pitts Neuron(MCP):A Foundation for Modern Artificial Neural Networks*, <https://medium.com/@nsail7644/the-mcculloch-pitts-neuron-mcp-a-foundation-for-modern-artificial-neural-networks-bad17a5c926b>, maj 2024, data dostępu: 07.01.2025.
- [15] Saha, S., *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way*, <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>, grud. 2018, data dostępu: 27.11.2024.
- [16] Siek, K., „Metody Bezpiecznego Programowania: Wprowadzenie”, <https://www.cs.put.poznan.pl/ksiek/fp/basics.html>, data dostępu: 07.01.2025.
- [17] Sopyła, K., „Precision, recall i F1 – miary oceny klasyfikatora”, <https://ksopyla.com/machine-learning/precision-recall-f1-miary-oceny-klasyfikatora/>, kw. 2024, data dostępu: 07.01.2025.
- [18] *Stable Diffusion v2 Model Card*, <https://huggingface.co/stabilityai/stable-diffusion-2-inpainting>, Stability AI, data dostępu: 02.12.2024.
- [19] Szewczyk, A., „System oceny czystości nawierzchni z wykorzystaniem głębokich sieci neuronowych”, prac. mag., Politechnika Warszawska, Wydział Elektryczny, Instytut Sterowania i Elektroniki Przemysłowej, 2020.
- [20] Szwed, P., „Metody eksploracji danych 4. Klasyfikacja”, <https://home.agh.edu.pl/~pszwed/wiki/lib/exe/fetch.php?media=med-w04.pdf>, "data dostępu: 04.12.2024".
- [21] *Tensors*, https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial, Pytorch, data dostępu: 29.11.2024.
- [22] *torch*, <https://pytorch.org/docs/stable/torch.html>, Pytorch, data dostępu: 29.11.2024.
- [23] *ReLU - PyTorch 2.5 documentation*, *ReLU*, <https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html>, data dostępu: 26.11.2024.

Wykaz skrótów i symboli

CCN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
GAN	Generative Adversarial Network
GELU	Gaussian Error Linear Unit
GFLOPS	Giga Floating Point Operations Per Second
GPU	Graphics Processing Unit
IDE	Integrated Development Environment
JPEG	Joint Photographic Experts Group
LLM	Large Language Model
PNG	Portable Network Graphics
RELU	Rectified Linear Unit
SD	Stable Diffusion
SGD	Stochastic Gradient Descent
SI	Sztuczna Inteligencja
UI	User Interface

Spis rysunków

1	Schemat neuronu McCullocha-Pittsa. Opracowano na podstawie: [14]	4
2	Wykres funkcji aktywacji ReLU. Źródło: [23]	5
3	Struktura głębokiej sieci neuronowej. Źródło: [9]	5
4	Przykładowa architektura splotowej sieci neuronowej. Źródło: [15]	7
5	Prace nad projektowaniem sieci ConvNeXt. Źródło: [7]	8
6	Wykres funkcji aktywacji GELU. Źródło: [2]	9
7	Wyniki generacji obrazów z różną liczbą kroków odszumiania. Źródło: własne	10
8	Przykładowy wynik generacji nowego obrazu za pomocą metody inpainting. Opracowano na podstawie: [18]	11
9	Zmiany zdjęcia wejściowego w procesie nauczania sieci GAN. Źródło: [19]	11
10	Fragment okna programu TensorBoard. Źródło: własne	15
11	Fragmenty UI programu Adobe Firefly. Źródło: własne	18
12	Macierz pomyłek klasyfikatora trzech klas. Źródło: [20]	18
13	Generacje zdjęć zabrudzonych chodników za pomocą programu Adobe Firefly. Źródło: własne	22
14	Wyniki generacji czystych chodników z różną intensywnością zdjęcia referencyjnego z użyciem modelu Firefly Image. Źródło: własne	22
15	Wyniki generacji metodą text-to-image dla różnych wartości parametru <i>guidance_scale</i> , model SD w wersji 1.5. Źródło: własne	23
16	Wyniki generacji metodą text-to-image dla różnych wartości parametru <i>num_inference_steps</i> , model SD w wersji 1.5. Źródło: własne	24
17	Wyniki generacji metodą text-to-image dla różnych wartości parametru <i>guidance_scale</i> , model SD w wersji 2.1. Źródło: własne	24
18	Wyniki generacji metodą text-to-image dla różnych wartości parametru <i>num_inference_steps</i> , model SD w wersji 2.1. Źródło: własne	25
19	Wyniki generacji metodą image-to-image dla różnych wartości parametru <i>strength</i> , model SD w wersji 1.5. Źródło: własne	26
20	Wyniki generacji metodą image-to-image dla różnych wartości parametru <i>guidance_scale</i> , model SD w wersji 1.5. Źródło: własne	26

21	Wyniki generacji metodą image-to-image dla różnych wartości parametru <i>num_inference_steps</i> , model SD w wersji 1.5. Źródło: własne	27
22	Wyniki generacji metodą image-to-image dla różnych wartości parametru <i>strength</i> , model SD w wersji 2.1. Źródło: własne	27
23	Wyniki generacji metodą inpainting po zastosowaniu zdjęcia referencyjnego jako zdjęcie-maskę. Źródło: własne	28
24	Wyniki generacji metodą inpainting dla różnych wartości parametru <i>strength</i> , model SD w wersji 1.5. Źródło: własne	29
25	Wyniki generacji metodą inpainting dla różnych wartości parametru <i>guidance_scale</i> , model SD w wersji 1.5. Źródło: własne	29
26	Wyniki generacji metodą inpainting dla różnych wartości parametru <i>num_inference_steps</i> , model SD w wersji 1.5. Źródło: własne	30
27	Wyniki generacji metodą inpainting dla różnych wartości parametrów, model stable-diffusion-inpainting. Źródło: własne	30
28	Obraz wyjściowy uzyskany po zastosowaniu optymalnych wartości parametrów, model stable-diffusion-2-inpainting. Źródło: własne	31
29	Wyniki generacji skupisk liści metodą image-to-image, model SD w wersji 1.5. Źródło: własne	32
30	Wyniki generacji czystych chodników metodą image-to-image, model SD w wersji 1.5. Źródło: własne	33
31	Rzeczywiste zdjęcia chodników ze zbioru pierwszego, przydzielone do klasy 0. Źródło: własne	34
32	Rzeczywiste zdjęcia chodników ze zbioru pierwszego, przydzielone do klasy 1. Źródło: własne	34
33	Rzeczywiste zdjęcia chodników ze zbioru pierwszego, przydzielone do klasy 2. Źródło: własne	35
34	Rzeczywiste zdjęcia chodników ze zbioru drugiego, przydzielone do klasy 0. Źródło: [19]	35
35	Rzeczywiste zdjęcia chodników ze zbioru drugiego, przydzielone do klasy 1. Źródło: [19]	36
36	Rzeczywiste zdjęcia chodników ze zbioru drugiego, przydzielone do klasy 2. Źródło: [19]	36
37	Zdjęcia chodników należących do trzech różnych klas zabrudzenia. Źródło: własne	38
38	Cztery losowe transformacje zdjęcia zabrudzonego chodnika. Źródło: własne	39
39	Wartość dokładności wyrażonej w procentach, w zależności od numeru epoki uczącej. Przebieg nauczania sieci ConvNeXt. Źródło: własne	42
40	Wartość funkcji kosztu, w zależności od numeru epoki uczącej. Przebieg nauczania sieci ConvNeXt. Źródło: własne	42
41	Macierz pomyłek dla zbioru testowego nr 1. Źródło: własne	43
42	Błędne klasyfikacje otrzymane stosując wyuczoną sieć ConvNeXt i zbiór testowy nr 1. Źródło: własne	44
43	Macierz pomyłek dla zbioru testowego nr 2. Źródło: własne	44

44	Błędne klasyfikacje otrzymane stosując wyuczoną sieć ConvNeXt i zbiór testowy nr 2. Źródło: własne	45
45	Wartość dokładności wyrażonej w procentach, w zależności od numeru epoki uczącej. Przebieg nauczania sieci ResNet152. Źródło: własne	46
46	Wartość funkcji kosztu, w zależności od numeru epoki uczącej. Przebieg nauczania sieci ResNet152. Źródło: własne	46
47	Macierz pomyłek dla zbioru testowego nr 1. Źródło: własne	47
48	Macierz pomyłek dla zbioru testowego nr 2. Źródło: własne	47
49	Błędne klasyfikacje otrzymane stosując wyuczoną sieć ResNet152 i zbiór testowy nr 1. Źródło: własne	48
50	Błędne klasyfikacje otrzymane stosując wyuczoną sieć ResNet152 i zbiór testowy nr 2. Źródło: własne	48
51	Cztery transformacje zdjęcia zabrudzonego chodnika. a) Zdjęcie bez dodatkowych transformacji. b) Zdjęcie z dodaną losową zmianą jasności. c) Zdjęcie z dodaną losową zmianą saturacji barw. d) Zdjęcie z dodanymi losowymi zmianami jasności i saturacji barw. Źródło: własne	49

Spis tabel

1	Pierwsze 8 miejsc rankingu TIOBE w 2024 roku. Źródło: [5]	13
2	Liczba zdjęć chodników wygenerowanych przez poszczególne modele generatywne. Źródło: własne	33
3	Precyza klasyfikacji na zbiorze walidacyjnym w stosunku do różnych rozmiarów paczki uczącej. Źródło: własne	40
4	Dokładność klasyfikacji na zbiorze walidacyjnym w stosunku do różnych wartości współczynnika uczenia. Źródło: własne	41
5	Porównanie wyników nauczania z wprowadzonymi dodatkowymi transformacjami obrazów wejściowych. Źródło: własne	49

Spis załączników

1 Specyfikacja komputera użytego w pracy	65
--	----

Załącznik 1

Specyfikacja komputera użytego w pracy

Komputer stacjonarny	
OS	Microsoft Windows 11 Home
OS typ	64-bit
CPU	13th Gen Intel Core i7-13700F 2.10 GHz, 16 rdzeni, 24 procesory logiczne
GPU	NVIDIA GeForce RTX 4060 Ti
RAM	32GiB