

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka (INF)
SPECJALNOŚĆ: Grafika i systemy multimedialne (IGM)

**PRACA DYPLOMOWA
INŻYNIERSKA**

Detekcja uszkodzeń elewacji budynków za pomocą
algorytmów przetwarzania obrazu i sztucznej
inteligencji

Detection of defects of building elevation using
image processing and artificial intelligence
algorithms

AUTOR:
Adrianna Markowska

PROWADZĄCY PRACE:
dr Marek Bazan, K30W04D03

Spis treści

1	Wprowadzenie – problematyka, geneza problemu	1
2	Cel i założenia projektu	2
3	Wstęp teoretyczny	3
3.1	Sieci CNN	3
3.2	Transfer learning	4
3.3	Opis sieci MobileNetV2	5
4	Implementacja	9
4.1	Zastosowane technologie	9
4.2	Użyta baza zdjęć	9
4.3	Opis implementacji	12
5	Testy i analiza wyników	16
6	Podsumowanie i możliwy rozwój	19
	Literatura	21
A	Spis zawartości załączonej płyty CD/DVD	22

Rozdział 1

Wprowadzenie – problematyka, geneza problemu

Posiadacze nieruchomości muszą liczyć się z koniecznością regularnego sprawdzania stanu swoich budynków oraz renowacji, jeżeli chcą aby ich własność nie traciła na wartości na przestrzeni lat. Zewnętrzne powierzchnie budynków są szczególnie narażone na oddziaływanie warunków atmosferycznych. Deszcz, śnieg, wiatr oraz nagłe zmiany temperatury wpływają na elewacje budynków powodując odbarwienia, niszczenie farby, pęknięcia, dziury oraz sprzyjając rozwojowy grzybów. W przypadku wielopiętrowych budynków lub całych ich kompleksów sprawdzenie stanu elewacji na zewnątrz w celu zaplanowania ewentualnych prac remontowych może być dla zarządcy sporym problemem. Takie zadanie jest pracochłonne, czasochłonne oraz może wiązać się z zagrożeniem zdrowia i bezpieczeństwa sprawdzających, szczególnie na trudno dostępnych wysokościach. Raport z takiego badania służyć będzie do oszacowania doraźnych i prognozowanych długoterminowych kosztów odnowy, naprawy i utrzymania budynku. Dzięki odpowiednio wcześnieemu wykryciu wad możliwe jest przeciwdziałaniu ich rozwoju do poziomu, gdy stają się niebezpieczne lub zbyt kosztowne do naprawy.

Alternatywą do przeprowadzenia kontroli na miejscu są techniki analizy obrazu do wykrywania wad. Metody te nie należą jednak do najprostszych a ich wyniki są uwarunkowane jakością zdjęć, których robienie zależne jest od rzeczywistych warunków takich jak światło pod złym kątem, cień czy przeszkody fizyczne w ustawieniu aparatu (np. rozległe drzewa w bliskim sąsiedztwie budynku). W ciągu ostatnich parunastu lat opublikowane zostało wiele badań o technikach wykrywania defektów na różnych powierzchniach opartych na obliczeniach miękkich oraz uczeniu maszynowym [1] [2] [3].

Warto wspomnieć o gotowym produkcie z zakresu wykrywania defektów stworzonym dla rosyjskiej grupy badawczej do analizy stanu dróg [4]. Materiał wideo dostarczony z kamier umieszczonych na samochodzie badawczym jest analizowany z rozłożeniem problemu na dwa mniejsze: pęknięcia na drodze oraz powstałe już dziury. Dostarczenie tego produktu wymagało od twórców przygotowania modelu, który będzie w stanie poradzić sobie ze wszystkimi możliwymi do spotkania drodze obiektyami, a więc samochodami, łatami na drogach czy różnymi powierzchniami dróg.

Rozdział 2

Cel i założenia projektu

Po analizie zmiennych problemu oraz dostępnych materiałów, w szczególności możliwych do uzyskania zdjęć zdecydowano o uproszczeniu problemu detekcji uszkodzeń elewacji budynków do rozpoznawania czy na zdjęciu znajduje się pęknięcie - utrata spójności, dziura - ubytek lub elewacja bez uszkodzeń. Docelowym pomysłem na dostarczanie zdjęć do analizy było użycie drona z kamerą, który przesyłałby zdjęcia na urządzenia mobile. Zamiast tworzyć nową sieć konwolencyjną zdecydowano o użyciu gotowej architektury sieci MobileNetV2, dedykowanej właśnie na urządzenia mobilne.

Rozdział 3

Wstęp teoretyczny

3.1 Sieci CNN

Konwolucyjna sieć neuronowa (Convolutional Neural Network CCN) jest algorytmem głębskiego uczenia maszynowego, który dla zadanego na wejście obrazu przypisuje znaczenie znajdującym się na nim obiektom lub cechom, dzięki czemu jest w stanie rozróżnić je miedzy sobą. Sieci tego rodzaju są używane do rozwiązywania problemów z zakresów klasyfikacji obrazów, detekcji obrazów, lokalizacji i segmentacji. Wraz z rozwojem zdolności obliczeniowej sprzętu, potrzebnej w celu ich trenowania, w pierwszej dekadzie dwudziestego pierwszego wieku sieci CNN znalazły się w obszarze zainteresowań badaczy zajmujących się tematyką computer vision.

Sieci konwolucyjne podobnie do sieci neuronowych składają się z połączonych warstw neuronów. Perceptrony, najprostsze z sieci neuronowych składające się z pojedynczego neuronu, klasyfikują dane pojawiające się na wejściu i używając wytrenowanych na przykładach wartości wag wejść oraz odchyłeń ustawiają odpowiednia wartość wyjścia. Są one w stanie klasyfikować zbiory, które dają się rozdzielić od siebie poprzez prostą na \mathbb{R}^2 , płaszczyznę na \mathbb{R}^3 oraz hiperplaszczyznę w wyższych wymiarach [5]. Aby zwiększyć możliwości klasyfikacji sieci perceptrony są łączone tworząc wielowarstwową sieć, której dane pojawiające się na wejściu są analizowane przez wewnętrzne warstwy, tak zwane warstwy ukryte, aby zostać na wyjściu przypisane do danej kategorii.

Sieci CNN specjalizujące się w przetwarzaniu obrazów jako wejście przyjmują obrazy w postaci cyfrowej a więc dla zdjęcia o rozdzielcości $w \times h$ pikseli używającego przestrzeni barw RGB są to 3 macierze o rozmiarach $w \times h$ reprezentujące 3 kanały kolorów RGB. Celem sieci konwolucyjnych jest redukcja zdjęć do formy, która jest łatwiejsza do przetworzenia, bez utraty cech potrzebnych do przeprowadzenia dobrej prognozy. Zawdzięczają one swoją nazwę matematycznemu działaniu - konwolucji czyli mnożeniu splotowemu dla dwóch funkcji. W kontekście macierzy obrazu polega na przykładaniu centralnie do każdego piksela obrazu filtru - macierzy konwolucji, czego wynikiem jest powstanie obrazu gdzie każdy piksel jest zależny od odpowiadającemu mu piksela w obrazie wejściowym i jego sąsiedztwa. Filtry o różnych wartościach i rozmiarach pełnią różne funkcje np. wyodrębniają krawędzie lub struktury liniowe, eliminują szum. Następujące po sobie warstwy konwolucyjne są w stanie wyciągnąć informacje o coraz bardziej skomplikowanych zależnościach przestrzennych na obrazie aplikując odpowiednie filtry.

Sieci CNN mogą mieć najróżniejsze architektury jednak większość stosuje się do pewnych ogólnych zasad, wykorzystując odpowiednie rodzaje warstw.

- INPUT- neurony przechowują wartości pikseli zdjęcia.

- CONV layer - warstwa konwolucyjna - każdy neuron połączony do $n = w * h * d$ neuronów z poprzedniej warstwy gdzie h, w, d to wysokość, szerokość i głębokość filtrów. Wylicza nową wartość piksela obrazów pochodnych.
- ReLU layer - rektyfikowane jednostki liniowe - funkcja aktywacji neuronu powyżej pewnego poziomu znaczenia, zazwyczaj $f(x) = \max(0, x)$. Przeciwdziała problemowi znikającego gradientu oraz wykładniczemu wzrostowi obliczeń wymaganych do obsługi sieci neuronowej.
- POOL layer - warstwa łącząca - operacja downsamplingu, służy do progresywnej redukcji rozmiaru przestrzennego do zredukowania ilości cech i złożoności obliczeniowej.
- DENSE layer - warstwa gęsta, w pełni połączona - każdy neuron warstwy jest połączony z każdym neuronem warstwy poprzedniej. Oblicza wynik klasyfikacji, na zasadzie prawdopodobieństwa przynależności do danej klasy, stąd jej rozmiar jest równy ilości klas. Do tej warstwy dołączona jest zazwyczaj funkcja aktywacji Softmax, przydzielająca obraz do jednej z klas określając jej etykietę np. 0 lub 1 dla klasyfikacji binarnej.

Typowe wzory powtarzane w architekturach to CONV - POOL oraz CONV - ReLU - POOL. Warto wspomnieć że nie wszystkie warstwy posiadają parametry do wyuczenia przez sieć. Warstwy POOL i ReLU implementują określone funkcje, natomiast dla warstw CONV i DENSE sieć w procesie uczenia nadzorowanego metodą gradientową (zazwyczaj propagacją wsteczną błędów), aktualizuje wagi i odchylenia danych neuronów tak, aby klasy obliczane przez sieć były zgodne z etykietami w zestawie szkoleniowym każdego obiektu.

3.2 Transfer learning

Ogromnym atutem ludzkiego mózgu jest jego zdolność do przenoszenia wiedzy zdobytej w jednej dziedzinie na inną, dalszy rozwój na znanych podstawach. Osoby, które nauczyły się języka włoskiego, nauczą się języka hiszpańskiego w krótkim czasie, ponieważ podstawa gramatyczna jest podobna. Algorytmy uczenia maszynowego były tradycyjnie projektowane dla odosobnionych zadań. Wymagają w tym celu ogromnej ilości danych aby sieć mogła zrozumieć wzór łączący dane. Zdobywanie oraz opis danych do uczenia bywa trudne, czasochłonne, a w niektórych przypadkach również drogie. Transfer learning (uczenie metodą transferu) jest metodą przewyciężenia izolowanego paradygmatu uczenia i wykorzystania wiedzy zdobytej w ramach jednego zadania do rozwiązywania tych z nim powiązanych. Dzięki temu podejściu możliwe jest użycie wiedzy (cech, wag) zdobytej na trenowaniu poprzedniego modelu do nowego modelu, potrzebując przy tym mniejszej ilości danych i/lub czasu. Dla problemów z zakresu widzenia komputerowego proste cechy, takie jak krawędzie, kształty, narożniki i intensywność, mogą być ponownie użyte dla różnych zadań umożliwiając transfer wiedzy między nimi [6] [7].

W pracy wprowadzającej [8] aby przedstawić ramy dla zrozumienia transfer learning używane są pojęcia domeny, zadania i prawdopodobieństwa. Domena D składa się z dwóch komponentów: przestrzeni cech χ oraz rozkładu prawdopodobieństwa $P(X)$ gdzie $X = \{x_1, \dots, x_n\} \in \chi$. Dla danej domeny $D = \{\chi, P(X)\}$, zadanie T składa się z dwóch komponentów: przestrzeni etykiet $Y = \{y_1, \dots, y_n\} \in \gamma$ oraz obiektywnej funkcji predykcyjnej η , która może być wykorzystana do przewidzenia odpowiadającej jej etykiety.

$T = \{\gamma, \eta\}$ ze statystycznego punktu widzenia może być zapisana jako $P(y|x)$. Przy użyciu zadanych pojęć podaje się definicje: Transfer Learning - biorąc pod uwagę określona domenę źródłową D_S i zadanie uczenia T_S , domenę docelową D_T i zadanie uczenia T_T , transfer learning ma na celu pomóc w poprawie uczenia się docelowej funkcji predykcyjnej γ_T w D_T używając wiedzy w D_S i T_S gdzie $D_S \neq D_T$, $T_S \neq T_T$. Opierając się o powyższa definicje istnieją 4 scenariusze transfer learningu, opisane z przykładami:

1. $\chi_S \neq \chi_T$ Przestrzenie cech domeny źródłowej i docelowej są różne, np. dokumenty są napisane w dwóch różnych językach. W kontekście przetwarzania języka naturalnego nazywa się to ogólnie adaptacją między językową.
2. $P(X_S) \neq P(X_T)$ Krańcowe rozkłady prawdopodobieństwa domeny źródłowej i docelowej są różne, np. dokumenty omawiają różne tematy. Ten scenariusz nazywany jest adaptacją domeny.
3. $\gamma_S \neq \gamma_T$ Przestrzenie etykiet między dwoma zadaniami są różne. Występuje razem ze scenariuszem 4, ponieważ niezwykle rzadko zdarza się aby dwa różne zadania miały różne przestrzenie etykiet ale dokładnie takie same rozkłady prawdopodobieństwa warunkowego.
4. $P(Y_S|X_S) \neq P(Y_T|X_T)$ Warunkowe rozkłady prawdopodobieństwa zadań źródłowych są różne np. dokumenty źródłowe i docelowe są niezrównoważone pod względem ich klas.

Dla sieci CCN transfer learning polega na użyciu wag dobrze wytrenowanej sieci jako podstawy dla nowego modelu. Użyta w pracy sieć MobileNetV2 została wytrenowana na bazie ImageNet [9] - ponad 14 milionów obrazów podzielonych na ponad 20 000 kategorii, obejmujące takie obiekty jak puzzle, balon czy różne rasy psów. Daje to solidną podstawę do dalszej nauki wybranego zagadnienia defektów elewacji. Jak było to już po przednio wspomniane, niższe warstwy konwolucyjne rozpoznają nieskomplikowane cechy obrazów, takie jak krawędzie, podczas gdy wyższe warstwy rozpoznają coraz to bardziej skomplikowane wzorce, jak ludzkie twarze, natomiast uznaję się, że dopiero warstwy w pełni połączone wyłapują informacje istotne do klasyfikacji obrazu do jednej z kategorii. Stąd o ile umiejętności wyłapania twarzy na obrazie nie będzie istotna dla problemu na przykład rozpoznawania kotów, to informacje na temat tego, jak obraz jest zbudowany, jaka kombinacja krawędzi się na nim znajduje, są już wiedząwartą przeniesienia na nowy model. Strategię opierającą się na tym założeniu nazywa się użyciem wytrenowanej sieci jako ekstraktora cech. Gotowy model pozbawia się ostatnich warstw, których wiedza jest specyficzna dla zagadnienia w którym był wykorzystywany i dokłada się nowe warstwy do wytrenowania na docelowym zestawie danych. Wagi modelu bazowego zostają zazwyczaj zamrożone podczas trenowania nowej sieci, to znaczy nie są one aktualizowane w wyniku propagacji wstecznej. Drugą stosowaną techniką jest Fine Tuning gdzie ponownie trenuje się wybrane warstwy modelu bazowego. Odmrażając wyższe warstwy modelu bazowego pozwala się im na dostosowanie do specyfiki nowego zadania.

3.3 Opis sieci MobileNetV2

Architektura MobileNetV2 opiera się głównie na pomyśle używania Depthwise Separable Convolutions (głębokich rozdzielnych konwolucji) [10] [11] oraz ideach Inverted Residual

Block (odwróconych bloków rezydualnych) [12] i Linear Bottlenecks (liniowych zwężeń) [12].

Depthwise Separable Convolutions różnią się od normalnych konwolucji sposobem w jaki zajmują się kanałami kolorów RGB, głębokością macierzy obrazu [13]. Dla klasycznych konwolucji dla każdego z kanałów RGB używana jest ta sama funkcja jądrową i jako wynik otrzymuje się macierz obrazu o głębokości 1. Depthwise Separable Convolutions można podzielić na dwa kroki. Pierwszy to Depthwise Convolutions, które aplikują 3 różne funkcje jądrowe (ang. kernels) o tej samej wielkości do każdego z kanałów obrazu z osobna czego wynikiem jest macierz obrazu o głębokości takiej jak na wejściu konwolucji. Drugi to Pointwise Convolution zwiększający ilość kanałów obrazu, iterując funkcję jądrową o wielkości $1 \times 1 \times N$ przez każdy piksel zdjęcia, gdzie N to ilość głębokość macierzy wyjściowej, ilości kanałów obrazu. Aby zrozumieć prawdziwą siłę Depthwise Separable Convolutions trzeba policzyć ilość mnożeń wykonywanych przez komputer. Dla przykładowego zdjęcia o rozdzielczości 12×12 i funkcji jądrowej o rozmiarze 5×5 obraz wyjściowy będzie mieć rozdzielcość 8×8 .

Dla normalnych konwolucji, jeżeli chcielibyśmy zwiększyć ilość kanałów należałoby ją powtórzyć N razy:

$$5 * 5 * 3 * 8 * 8 * N = 4800N$$

Dla Depthwise Convolutions:

$$5 * 5 * 3 * 8 * 8 * N = 4800$$

Dla Pointwise Convolutions:

$$1 * 1 * 3 * 8 * 8 * N = 192N$$

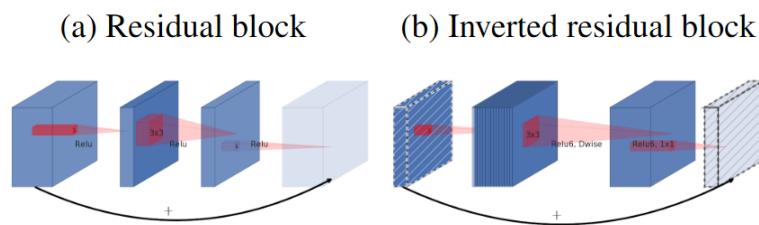
A więc dla $N > 1$:

$$4800N > 192N + 4800$$

Residual Block łączy początek i koniec bloku konwolucji z pominięciem warstw pomiędzy. Połączenie tych dwóch stanów daje sieci dostęp do wcześniejszych funkcji aktywacji, nie zmodyfikowanych przez blok konwolucji. Bloki tego typu zastosowane w architekturach ResNet działają zmieniając głębokość macierzy obrazu według wzorca szeroka \rightarrow wąska \rightarrow szeroka. Ilość kanałów w obrazie wchodząącym do bloku zostaje zmniejszona funkcją jądrową 1×1 o niskim koszcie obliczeniowym, przez co następująca po tym konwolucja ma mniejszą ilość parametrów. Z kolei dla sieci MobileNetV2 pierwszym krokiem w bloku jest zwiększenie głębokości macierzy obrazu, ponieważ głęboka konwolucja posiada mniej parametrów od klasycznej, a następnie jej zmniejszenie funkcją jądrową 1×1 do wejściowej ilości kanałów. Z powodu tych odwróconych działań w stosunku do oryginału blok nosi nazwę Inverted Residual Block. Ich przewaga polega na mniejszej ilości parametrów w stosunku do oryginału.

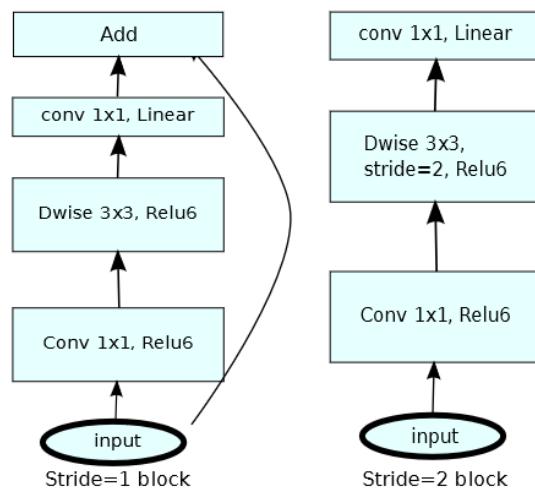
Użycie funkcji aktywacji ReLU odrzuca wartości mniejsze niż 0, co powoduje utratę informacji rozwijalnej przez zwiększenie głębokości macierzy obrazu aby zwiększyć przepustowość sieci. Jako że przy użyciu Inverted Residual Blocks zmniejszamy głębokość macierzy obrazu, czym ograniczamy skuteczność działania sieci, autorzy przedstawili pomysł Linear Bottlenecks gdzie ostatnia warstwa konwolucyjna bloku pozbawiona jest filtrowania funkcją ReLU.

Kolejną charakterystyką sieci MobileNetV2 jest stosowanie po każdej warstwie konwolucyjnej warstwy Batch Normalization. Aplikuje ona transformacje, która utrzymuje średnią wartość wyjściową bliską 0, a odchylenie standardowe wartości wyjściowej bliskie 1.



Rysunek 3.1: Residual Blocks, klasyczny i odwrócony. Obraz pochodzi z [12]

Głównymi składowymi architektury MobileNetV2 są dwa następujące po sobie bloki, których architektura pokazana jest na Rys. 3.2 a występujące w nich warstwy na Rys. 3.3



Rysunek 3.2: Architektury bloków wchodzących występujących w MobileNetV2 Obraz pochodzi z pracy [12]

block_15_expand (Conv2D)	(None, 7, 7, 960)	153600	block_14_add[0][0]
block_15_expand_BN (BatchNormal)	(None, 7, 7, 960)	3840	block_15_expand[0][0]
block_15_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_15_expand_BN[0][0]
block_15_depthwise (DepthwiseCo)	(None, 7, 7, 960)	8640	block_15_expand_relu[0][0]
block_15_depthwise_BN (BatchNor)	(None, 7, 7, 960)	3840	block_15_depthwise[0][0]
block_15_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_15_depthwise_BN[0][0]
block_15_project (Conv2D)	(None, 7, 7, 160)	153600	block_15_depthwise_relu[0][0]
block_15_project_BN (BatchNorma)	(None, 7, 7, 160)	640	block_15_project[0][0]
block_15_add (Add)	(None, 7, 7, 160)	0	block_14_add[0][0] block_15_project_BN[0][0]
block_16_expand (Conv2D)	(None, 7, 7, 960)	153600	block_15_add[0][0]
block_16_expand_BN (BatchNormal)	(None, 7, 7, 960)	3840	block_16_expand[0][0]
block_16_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_16_expand_BN[0][0]
block_16_depthwise (DepthwiseCo)	(None, 7, 7, 960)	8640	block_16_expand_relu[0][0]
block_16_depthwise_BN (BatchNor)	(None, 7, 7, 960)	3840	block_16_depthwise[0][0]
block_16_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_16_depthwise_BN[0][0]
block_16_project (Conv2D)	(None, 7, 7, 320)	307200	block_16_depthwise_relu[0][0]
block_16_project_BN (BatchNorma)	(None, 7, 7, 320)	1280	block_16_project[0][0]
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409600	block_16_project_BN[0][0]
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]
<hr/>			
Total params: 2,257,984			
Trainable params: 0			
Non-trainable params: 2,257,984			

Rysunek 3.3: Wycinek z podsumowania modelu bazowego w projekcie, pokazujący ostatnie warstwy. Blok 15 obrazuje jak wyglądają wszystkie bloki o numerach parzystych, a blok 16 wszystkie bloki o numerach nieparzystych

Rozdział 4

Implementacja

4.1 Zastosowane technologie

Projekt został stworzony w Google Colab - środowisku interaktywnym języka Python opartym na Jupyter Notebook, działającym w chmurze Google. Google Colab zapewnia dostęp do GPU, niezbędny do trenowania głębszych sieci neuronowych. Dodatkowo została użyta biblioteka TensorFlow [14] dla uczenia maszynowego oraz Karas API [15], ułatwiająca jej użycie. Keras pozwala projektować, trenować, testować oraz używać modeli głębszego uczenia maszynowego w stosunkowo prosty sposób, przez co jest popularnie używana przez deweloperów.

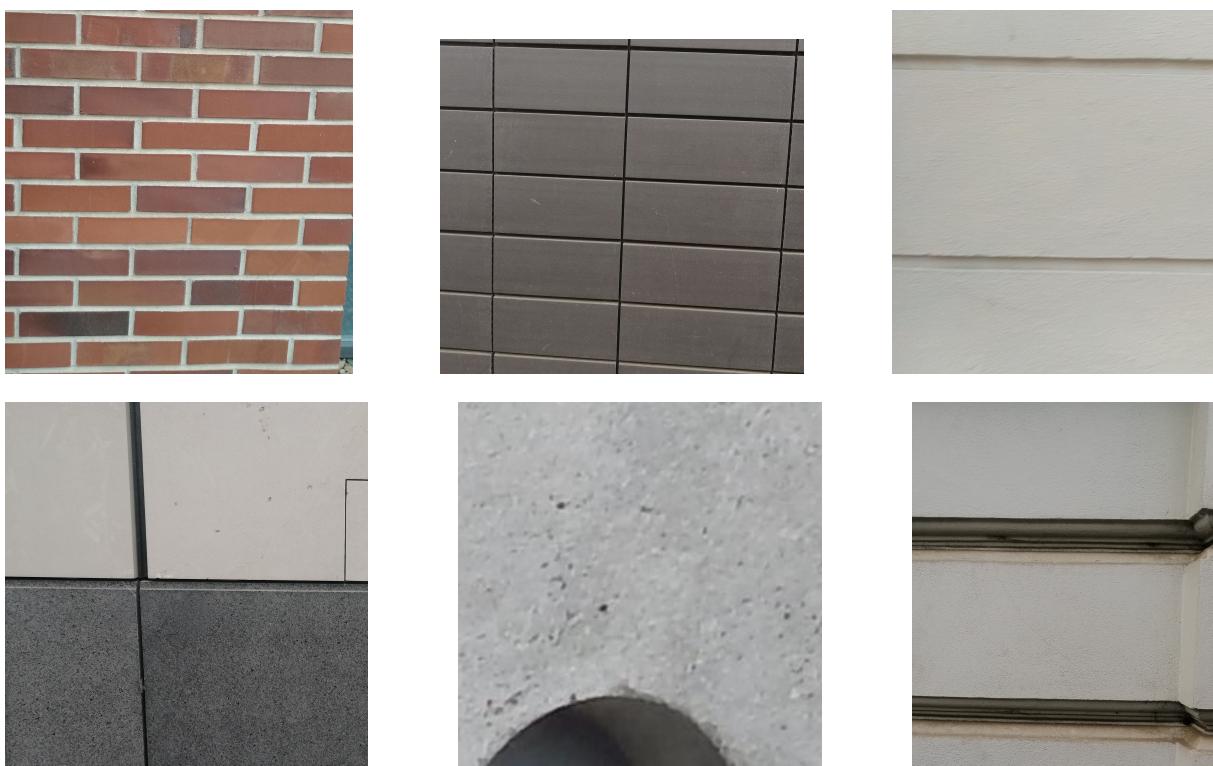
4.2 Użyta baza zdjęć

Zdjęcia użyte w pracy zostały wykonane kamerą telefonu oraz pozyskane z bazy "Structural Defects Network (SDNET) 2018" udostępnionej na serwisie Kaggle [16]

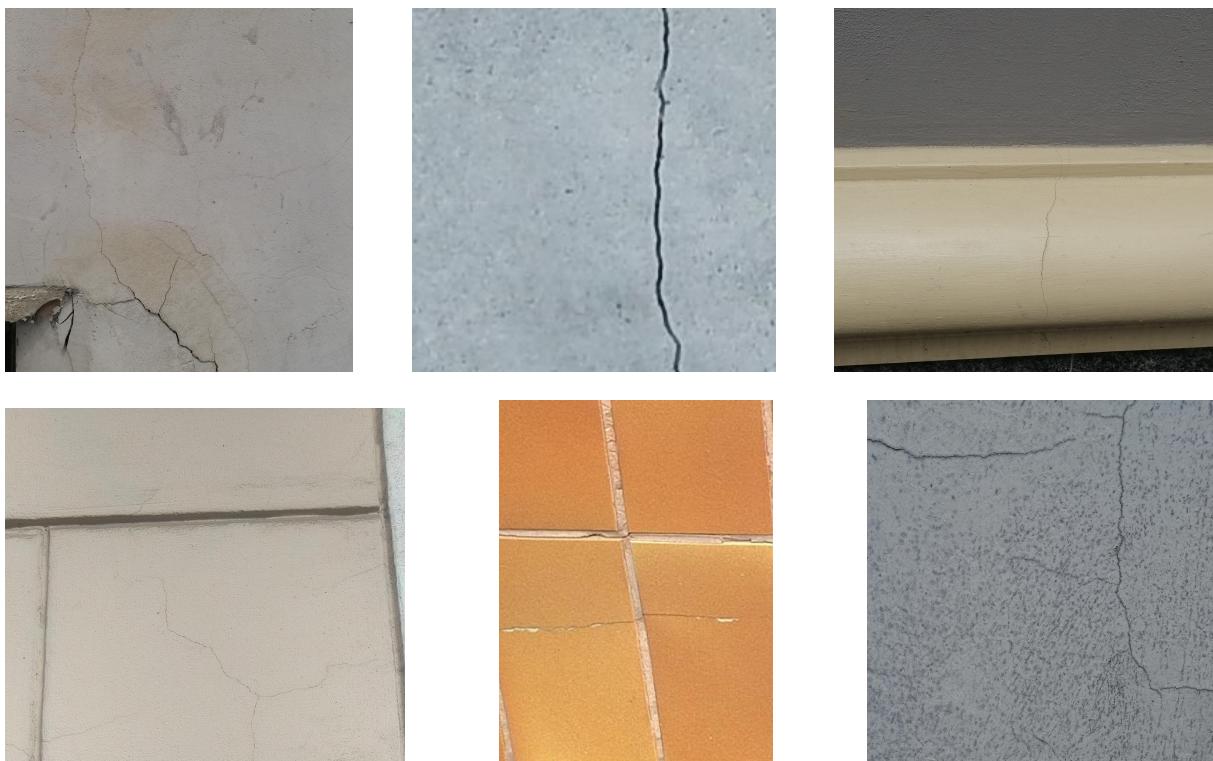
802 obrazy w zbiorze to zdjęcia zrobione we Wrocławskiej dzielnicy Ołbin. Zawiera dzięki temu zarówno zdjęcia fasad starych wrocławskich kamienic wraz często spotykanymi złobieniami, jak i nowszych budynków, wyłożonych płytami z betonu architektonicznego.

Baza "Structural Defects Network (SDNET) 2018" składa się z ponad 56 000 zdjęć betonowych powierzchni mostów, chodników oraz ścian, podzielonych właśnie ze względu na pochodzenie. W każdej z tych kategorii istnieje podział na obrazy zawierające pęknięcia oraz nieuszkodzone powierzchnie. W celu powiększenia istniejącej bazy ze zbioru zdjęć ścian zostały wybrane 433 zdjęcia.

Cała baza zdjęć obejmuje 1242 obrazy, podzielone na trzy kategorie: elewacje bez naruszeń, pęknięcia oraz dziury. Obrazy o dużo wyższej niż wymagana rozdzielczości zostały ręcznie oraz z pomocą skryptów w języku Python podzielone na mniejsze. Pozwoliło to uniknąć utraty znacznej ilości danych w procesie zmniejszenia rozdzielczości z np. 2448×5120 do 224×224 pikseli, które przyjmuje na wejście sieć neuronowa. Istniała obawa że drobne pęknięcia zostałyby w procesie scalania pikseli pominięte. Trzeba zauważać że na nieuszkodzonych elewacjach zdarzają się przypadki odbarwień, zmiany koloru oraz zabrudzeń, więc strata zbyt dużej ilości danych mogłaby skutkować przekazaniem sieci danych, z których nie można by wyciągnąć mających sens wzorców.



Rysunek 4.1: Elewacje bez uszkodzeń

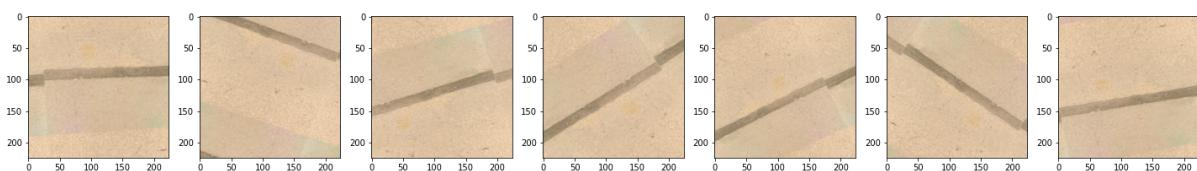


Rysunek 4.2: Elewacje z pęknięciami



Rysunek 4.3: Elewacje z dziurami

W związku z niewielkim zbiorem zdjęć oraz występującym problemem przetrenowania sieci [17] zastosowano augmentacje danych przy użyciu transformacji geometrycznych, takich jak: odbicie w poziomie, odbicie w pionie, powiększenie, rotacja do 40 stopni, przesunięcie w poziomie, przesunięcie w pionie oraz pochylenie. Dla transformacji mogących skutkować utratą części obrazu przez wysunięcie poza granice zdjęcia, zastosowano dodatkową opcję użycia tych fragmentów w miejsce pustych pikseli. Obrazy zostały zmniejszone do rozmiaru 224x224 pikseli i a wartości kanałów RGB znormalizowane.



Rysunek 4.4: Przykład wykonanych transformacji obrazu z bazy

4.3 Opis implementacji

Pierwszym krokiem implementacji było stworzenie obiektu klasy ImageDataGenerator generującego partie tensorów obrazów przy użyciu augmentacji danych w czasie rzeczywistym. Jako argumenty zostały przekazane parametry transformacji do wykonania na obrazach oraz jaki ułamek zbioru który ma zostać użyty jako zestaw walidacyjny przy trenowaniu. Przy wywołaniu funkcji `flow_from_directory()` przekazywane są parametry określające format oraz rozmiar przekazywanych do sieci zdjęć, ale również katalog gdzie znajduje się baza oraz kolejność występowania zdjęć. Kolejność występowania obrazów w kolejnych partiach jest zmieniana (parametr '`shuffle`') w celu uniknięcia wyuczenia przez sieć powtarzanej sekwencji. Wygenerowane obrazy zostały zapisane ze względu na piszącą pracę.

```
train_data_gen= ImageDataGenerator( rescale=1./255,
                                    rotation_range = 40,
                                    width_shift_range = 0.3,
                                    height_shift_range = 0.3,
                                    shear_range = 0.2,
                                    zoom_range = 0.1,
                                    horizontal_flip = True,
                                    vertical_flip = True,
                                    fill_mode = 'wrap',
                                    validation_split = 0.2
                                   )

train_generator = train_data_gen.flow_from_directory(
                    directory = data_dir,
                    batch_size = BATCH_SIZE,
                    target_size = (IMAGE_RES,IMAGE_RES),
                    shuffle = True,
                    seed=16,
                    save_to_dir = saves_dir,
                    save_format = "jpg",
                    subset = "training",
                    class_mode = 'binary')
```

>>Found 995 images belonging to 3 classes.

Dane używane do walidacji są pobierane ze pomocą narzędzia Keras Dataset Preprocessing, a następnie normalizowane.

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
            data_dir,
            validation_split = 0.2,
            subset = "validation",
            seed = 16,
            image_size = (IMAGE_RES, IMAGE_RES),
            batch_size = BATCH_SIZE )

>>Found 1242 files belonging to 3 classes.
>>Using 248 files for validation.
```

```

def normalize(images, labels):
    images = tf.cast(images, tf.float32)
    images /= 255
    return images, labels

val_dataset = val_ds.map(normalize)

```

Keras Applications zawiera architektury modeli wytrenowanych na dużych zbiorach danych, między innymi użytą MobileNetV2. Jako bazę tworzonego modelu pobrano instancje architektury MobileNetV2, bez wierzchniej warstwy klasyfikującej. Model został zainicjalizowany wagami wytrenowanymi na zbiorze 'ImageNet', a następnie jego parametry zamrożone. Na zamrożone warstwy pochodzące z MobileNetV2 dołożone zostały następujące warstwy:

- GlobalAveragePooling2D - redukcja, przez wyciągnięcie średniej, tensora o wielkości (7, 7, 1280) do wektora (1280)
- Dropout - mechanizm walczenia z przetrenowaniem, losowe opuszczanie węzłów w warstwie poprzedzającej.
- Dense - warstwa w pełni połączona, oblicza wynik klasyfikacji.

```

base_model = tf.keras.applications.MobileNetV2(
    input_shape = (IMAGE_RES, IMAGE_RES, 3),
    include_top = False,
    weights = 'imagenet')

base_model.trainable = False

inputs = tf.keras.Input(shape=(IMAGE_RES, IMAGE_RES, 3))
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = tf.keras.layers.Dense(3)(x)
model = tf.keras.Model(inputs, outputs)

```

Jako konfiguracji użyto optymalizatora Adam, ze zmniejszonym wskaźnikiem uczenia od 15 epoki, wg wzoru $lr * exp(-0.1)$ [18] oraz SparseCategoricalCrossentropy jako funkcji strat.

Po 30 epokach trenowania osiągnięto następujące parametry sieci:

```
loss: 0.3015 - accuracy: 0.8673 - val_loss: 0.2041 - val_accuracy: 0.9274
```

W celu dostrojenia(fine-tuning) sieci zostały odmrożone ostatnie 40 warstw MobileNet, a po kolejnych 15 epokach trenowania osiągnięto następujące parametry:

```
loss: 0.1969 - accuracy: 0.9256 - val_loss: 0.1440 - val_accuracy: 0.9556
```

```
Model: "model"
```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
mobilenetv2_1.00_224 (Function)	(None, 7, 7, 1280)	2257984
global_average_pooling2d (Global)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 3)	3843
<hr/>		
Total params:	2,261,827	
Trainable params:	3,843	
Non-trainable params:	2,257,984	

Rysunek 4.5: Podsumowanie architektury stworzonego modelu

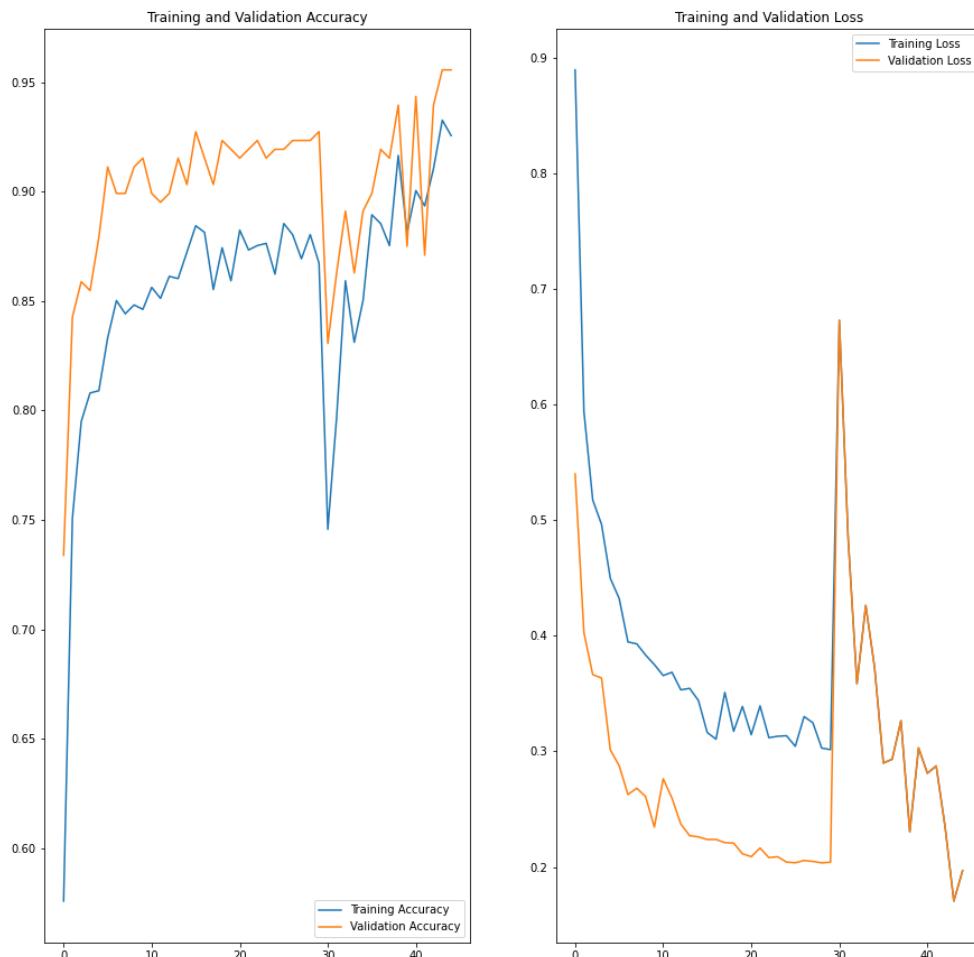
```
Model: "model"
```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
mobilenetv2_1.00_224 (Function)	(None, 7, 7, 1280)	2257984
global_average_pooling2d (Global)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 3)	3843
<hr/>		
Total params:	2,261,827	
Trainable params:	1,747,011	
Non-trainable params:	514,816	

Rysunek 4.6: Podsumowanie architektury stworzonego modelu po odmrożeniu 40 ostatnich warstw.

Lepsze wyniki dla zestawu walidacyjnego niż treningowego, choć zazwyczaj istnieje przeciwna zależność, mogą być związane z warstwą Dropout, przez którą część pikseli ze zdjęć nie była analizowana.

Na wykresach obrazujących zmianę dokładności oraz strat w trakcie trenowania widać że przed 30 epoką następuje wypłaszczenie wykresów co w próbach skutkowało niskimi zmianami w dalszym trenowaniu z zamrożonymi wagami. Mimo początkowego spadku wydajności sieci w procesie fine tuningu po paru epokach udało się dodatkowo poprawić parametry sieci. Dalsze próby dostrajania sieci mogłyby skutkować przetrenowaniem.

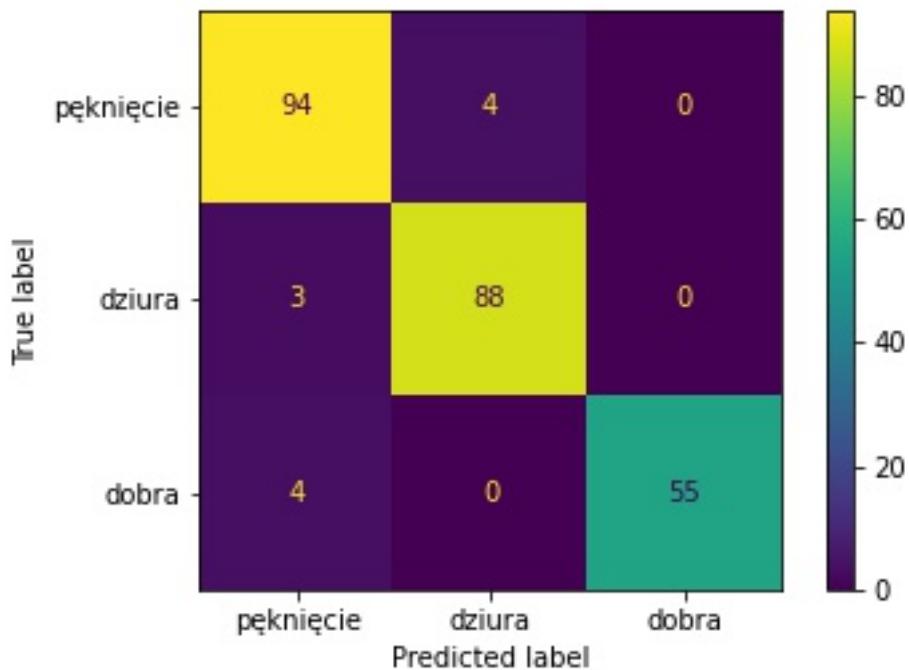


Rysunek 4.7: Wykresy zmiany dokładności oraz strat sieci zależne od epok

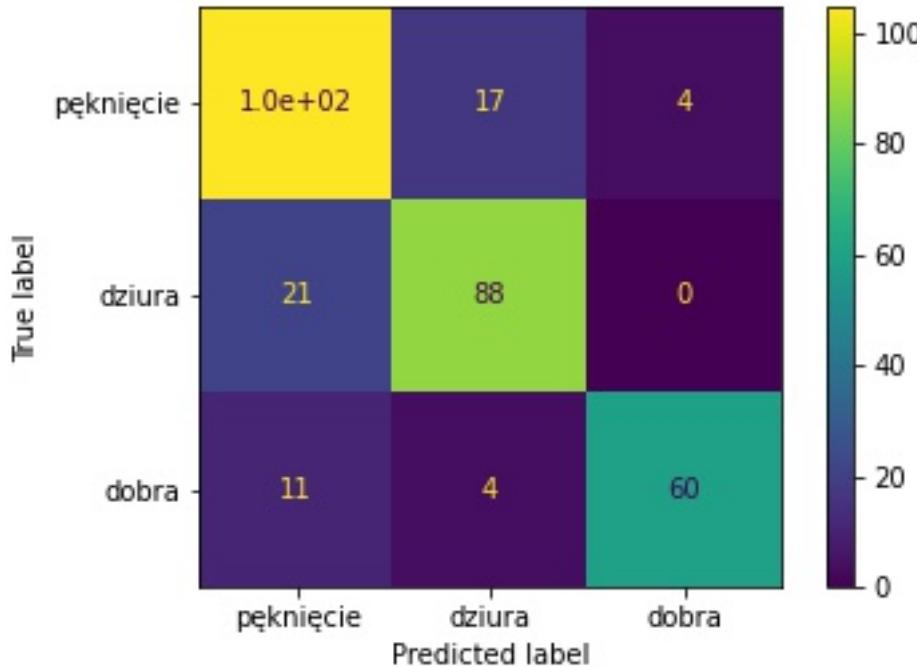
Rozdział 5

Testy i analiza wyników

W celach testowych, aby sprawdzić jak model radzi sobie ze zdjęciami, których nie widział z bazy danych wydzielono 310 obrazów. Dzięki takiemu zabiegowi istnieje możliwość sprawdzenia czy model faktycznie dobrze się sprawuje, czy dopasował się do zestawu walidacyjnego. Jako że baza zdjęć jest niebilansowana istnieje możliwość, że wytrenowany model jest stronniczy w kierunku klasy o większej ilości próbek, wyniku przesuniętego rozkładu prawdopodobieństwa [19]. Dlatego zamiast dokładności sprawdzono inne metryki oparte na przewidywaniach: prawdziwie pozytywnych(TP), fałszywie pozytywnych(FP), prawdziwie negatywnych(FN) i fałszywie negatywnych(FN) [19]. Przewidywania modelu dla zestawu walidacyjnego i testowego pokazują macierze pomyłek (ang. confusion matrix) Rys. 5.1 i Rys. 5.2



Rysunek 5.1: Macierz pomyłek dla zestawu walidacyjnego



Rysunek 5.2: Macierz pomyłek dla zestawu testowego

Użyte wskaźniki i ich znaczenie:

- Precison = $\left(\frac{TP}{FP+TP} \right)$ - określa ilość prawdziwych pozytywnych przewidywań we wszystkich pozytywnych przewidywaniach
- Recall = $\left(\frac{TP}{FN+TP} \right)$ - określa zdolność modelu do przewidywania prawdziwych pozytywów we wszystkich pozytywach.
- F1 = $\left(\frac{Precision*Recall}{Precision+Recall} \right)$ - średnia harmoniczna dwóch powyższych miar.

Dla zestawu testowego i walidacyjnego otrzymano następujące wyniki:

	Zestaw testowy	zestaw walidacyjny
Precision	0.84	0.96
Recall	0.81	0.95
F1 score	0.82	0.95

Można zauważyć że model lepiej poradził sobie z zestawem walidacyjnym, jednak wyniki otrzymane dla zestawu testowego są zadowalające. Przyglądając się Confusion Matrix można stwierdzić że prawdopodobieństwo nie wykrycia przez sieć uszkodzenia elewacji jest stosunkowo mała. Mogą zdarzyć się pomyłki w rozróżnieniu między pęknięciem a dziurą co jest akceptowalne, biorąc pod uwagę że wokół dziur często znajdują się pęknięcia, które mogą być wykrywane lub istnienie pęknięć na tyle gęstych że zostały zakwalifikowane jako dziury. Występujące w mniejszej ilości fałszywie pozytywne przewidywania dla dziur i pęknięć nie powinny stanowić przeciwwskazania do używania modelu.



Rysunek 5.3: Obrazy należące do kategorii fałszywych pozytywów/negatywów w zależności od liczonej klasy. Etykiety: 0 - pęknięcie, 1 - elewacja bez uszkodzeń, 2 - dziura

Rozdział 6

Podsumowanie i możliwy rozwój

Dla zagadnienia rozpoznawania defektów elewacji budynków stworzono model sieci konwolucyjnej zdolnej do wykrywania na zdjęciach pęknięć oraz dziur w elewacji budynków. Przy pomocy technik Transfer Learning stworzono sieć konwolucyjną, której modelem bazowym jest sieć MobileNetV2 wytrenowana na zbiorze ImageNet. W celu adaptacji jej do nowego zadania model został wytrenowany na nowej bazie obrazów, stworzonej na potrzeby pracy, ze zdjęć robionych we własnym zakresie oraz dostępnych w internecie.

Ze względu na ograniczone zasoby danych do trenowania istotnym było wykorzystanie Transfer Learning. Użycie MobileNetV2 jako ekstraktora cech umożliwiło otrzymanie zdowalających wyników przy niewielkiej ilości zdjęć w zestawie trenującym. MobileNetv2, jest siecią o małej ilości parametrów i dużej skuteczności dzięki wykorzystaniu Depthwise SeparableConvolutions, Inverted Residual Blocks i Linear Bottlenecks.

Dodatkowo użycie sieci MobileNetV2 pozwala pozwala na wykorzystanie modelu do implementacji rozwiązania dla inspekcji elewacji budynków do wykorzystania na miejscu badania i otrzymania wyników w krótkim czasie. W tym celu należałoby stworzyć aplikację mobilną używając TensorFlow Lite oraz wybranego języka programowania, która otrzymywałaby zdjęcia z drona i określała w którym obszarze zostało zrobione zdjęcie z wykrytą wadą. Przeprowadzone testy udowadniają że przewidywanie modelu z dużym prawdopodobieństwem wykryją defekty na elewacji co sprawia że może być używany jako narzędzie diagnostyczne.

Literatura

- [1] Perez, H.; Tah, J.H.M.; Mosavi, A. *Deep Learning for Detecting Building Defects Using Convolutional Neural Networks*, Sensors 2019, 19, 3556.
- [2] D. Weimera, H. Thamera, B. Scholz-Reiterb, *Learning defect classifiers for textured surfaces using neural networks and statistical feature representations* , 2013, Procedia CIRP 7 (2013) 347 – 352
- [3] Tamás Czimmermann, *Visual-Based Defect Detection and Classification Approaches for Industrial Applications—A SURVEY*, 2020,
- [4] Deep systems, *Road defects detection*, 2015, <https://deepsystems.ai/solutions/road-defects-detection>
- [5] Maja Czoków, Jarosław Piersa, Tomasz Schreiber, *Wstęp do Sieci Neuronowych*, 2013, <https://www-users.mat.umk.pl/~piersaj/www/contents/teaching/wsn2013/wsn-notatki.pdf>
- [6] Sebastian Ruder, *Transfer Learning - Machine Learning's Next Frontier*, 2017, <https://ruder.io/transfer-learning/>
- [7] Dipanjan Sarkar, *A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning*, 2018, <https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a>
- [8] S. J. Pan and Q. Yang, *A Survey on Transfer Learning* in IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345–1359, Oct. 2010, doi: 10.1109/TKDE.2009.191.
- [9] Deng, J. et al., *Imagenet: A large-scale hierarchical image database*. In 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255, 2009
- [10] Chollet, Francois, *Xception: Deep Learning With Depthwise Separable Convolutions*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017
- [11] Andrew G. Howard, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017
- [12] Paul-Louis Pröve, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, 2018, <https://towardsdatascience.com/mobilenetv2-inverted-residuals-and-linear-bottlenecks-8a4362f4ffd5>

- [13] Paul-Louis Pröve, *An Introduction to different Types of Convolutions in Deep Learning*, 2017, <https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d>
- [14] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.
- [15] François Chollet *Keras*, 2015, <https://github.com/fchollet/keras>
- [16] Leslie Lamport, *Structural defects network*, 2018, <https://www.kaggle.com/aniruddhsharma/structural-defects-network-concrete-crack-images/metadata>
- [17] Julien Despois, *Memorizing is not learning! — 6 tricks to prevent overfitting in machine learning.*, 2018, <https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42>
- [18] Jason Brownlee, *Understand the Impact of Learning Rate on Neural Network Performance*, 2020, <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>
- [19] Issa Memari, *Precision, Recall, Accuracy, and F1 Score for Multi-Label Classification*, 2021, <https://medium.com/synthesio-engineering/precision-accuracy-and-f1-score-for-multi-label-classification-34ac6bdfb404>

Dodatek A

Spis zawartości załączonej płyty CD/DVD

- Cyfrowa wersja pracy inżynierskiej "W04_233455_2021_praca_inżynierska.pdf"
- Kopia notatnika Colab z kodem źródłowym Implementacja.ipynb
- Folder "Model" z modelem sieci neuronowej wraz z parametrami.