

**AGH**

**AKADEMIA GÓRNICZO-HUTNICZA  
IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI  
INSTYTUT ELEKTRONIKI**

**PRACA DYPLOMOWA INŻYNIERSKA**

**System analizy pęknięć powierzchni budowlanych na podstawie  
analizy obrazów**

**Image-based construction surface crack analysis system**

Autor:

Mateusz Szczęśniak

Kierunek studiów:

Elektronika i Telekomunikacja

Opiekun pracy:

prof. dr hab. inż. Bogusław Cyganek

Kraków, 2026

# Spis treści

Spis stosowanych symboli i skrótów . . . . .	4
Wstęp . . . . .	5
Cel pracy . . . . .	5
Powiązane prace . . . . .	6
<b>1 Część teoretyczna</b>	<b>7</b>
1.1 Sieci segmentujące . . . . .	7
1.1.1 U-Net . . . . .	7
1.1.2 SegFormer . . . . .	8
1.1.3 YOLO . . . . .	9
1.1.4 Metryki opisujące sieci segmentujące . . . . .	10
1.2 Sieci klasyfikujące . . . . .	11
1.2.1 EfficientNet . . . . .	11
1.2.2 ConvNeXt . . . . .	12
1.2.3 Metryki opisujące sieci klasyfikujące . . . . .	12
1.3 Uczenie zespołowe ( <i>ensemble</i> ) . . . . .	13
<b>2 Metodyka oraz projekt systemu</b>	<b>14</b>
2.1 Schemat działania systemu . . . . .	14
2.2 Uzasadnienie doboru architektur sieci neuronowych . . . . .	15
2.2.1 Dobór modeli segmentacyjnych . . . . .	15
2.2.2 Dobór modeli klasyfikacyjnych . . . . .	15
2.3 Kontroler domeny . . . . .	16
2.4 Zbiór danych modeli segmentacyjnych . . . . .	17
2.5 Zbiór danych kontrolera domeny . . . . .	18
2.6 Zbiór danych dla modeli klasyfikacyjnych . . . . .	19
2.7 Analiza geometryczna . . . . .	21
2.8 Interfejs użytkownika . . . . .	21
<b>3 Implementacja oraz środowisko badawcze</b>	<b>24</b>
3.1 Środowisko badawcze . . . . .	24
3.2 Proces uczenia modeli . . . . .	24
3.2.1 Wybrane parametry treningowe . . . . .	24
3.2.2 Potok przesyłania kodu oraz danych . . . . .	26
3.2.3 Ograniczenia . . . . .	26
3.2.4 Struktura kodu źródłowego . . . . .	26
3.3 Implementacja analizy geometrycznej . . . . .	26
3.4 Serwer HTTP i Interfejs Użytkownika . . . . .	27
3.4.1 Komunikacja i przesył danych . . . . .	27
3.4.2 Interfejs użytkownika . . . . .	27
<b>4 Analiza i porównanie wyników</b>	<b>28</b>

4.1	Weryfikacja skuteczności kontrolera domeny . . . . .	28
4.1.1	Macierz pomyłek i metryki binarne . . . . .	28
4.1.2	Analiza wydajnościowa . . . . .	29
4.2	Analiza porównawcza modeli klasyfikacyjnych . . . . .	29
4.2.1	Przebieg procesu uczenia . . . . .	29
4.2.2	Skuteczność klasyfikacji . . . . .	32
4.2.3	Analiza błędów . . . . .	33
4.2.4	Analiza wydajnościowa . . . . .	33
4.3	Analiza porównawcza modeli segmentacyjnych . . . . .	34
4.3.1	Przebieg procesu uczenia . . . . .	34
4.3.2	Analiza jakościowa . . . . .	37
4.4	Analiza wpływu różnych funkcji straty . . . . .	38
4.5	Analiza wpływu danych treningowych . . . . .	40
4.6	Porównanie z modelami SOTA ( <i>State of the Art</i> ) . . . . .	41
4.7	Analiza wydajnościowa i zasobowa . . . . .	42
4.8	Porównanie systemu analizy . . . . .	43
4.8.1	Porównanie z nowoczesnymi metodami <i>Instance Segmentation</i> . . . . .	44
4.9	Wnioski . . . . .	44
4.10	Podsumowanie . . . . .	45
	<b>Bibliografia</b>	<b>47</b>
	<b>Spis ilustracji</b>	<b>50</b>
	<b>Spis tabel</b>	<b>51</b>
	<b>A Konfiguracja środowiska</b>	<b>53</b>

# Spis stosowanych symboli i skrótów

ACC - Accuracy (Dokładność klasyfikacji)  
AP - Average Precision (Średnia precyzja)  
BCE - Binary Cross Entropy (Binarna entropia krzyżowa)  
CE - Cross Entropy (Entropia krzyżowa)  
CNN - Convolutional Neural Network (Konwolucyjna sieć neuronowa)  
CPU - Central Processing Unit (Centralna jednostka obliczeniowa)  
CUDA - Compute Unified Device Architecture (Architektura obliczeń równoległych)  
Dataset - Zbiór danych  
EDT - Euclidean Distance Transform (Euklidesowa transformata odległościowa)  
Ensemble - Uczenie zespołowe  
FFN - Feed-Forward Network (Sieć jednokierunkowa)  
FN - False Negative (Fałszywie negatywny - brak detekcji)  
FP - False Positive (Fałszywie pozytywny - fałszywy alarm)  
FPS - Frames Per Second (Liczba klatek na sekundę)  
GAP - Global Average Pooling (Globalne łączenie średnie)  
GPU - Graphics Processing Unit (Jednostka przetwarzania graficznego)  
HTTP - Hypertext Transfer Protocol (Protokół przesyłania dokumentów hipertekstowych)  
IoU - Intersection over Union (Wskaźnik pokrycia - stosunek części wspólnej do sumy)  
JSON - JavaScript Object Notation (Lekki format wymiany danych)  
LR - Learning Rate (Współczynnik uczenia)  
mAP - Mean Average Precision (Średnia precyzja dla wszystkich klas)  
MLP - Multi-Layer Perceptron (Wielowarstwowy perceptron)  
ReLU - Rectified Linear Unit (Funkcja aktywacji)  
RGB - Red, Green, Blue (Model przestrzeni barw)  
RT-DETR - Real-Time DEtection Transformer (Transformator detekcji w czasie rzeczywistym)  
SOTA - State of the Art (Najnowszy stan techniki)  
TinyImage - Zbiór danych Tiny ImageNet  
TN - True Negative (Prawdziwie negatywny)  
TP - True Positive (Prawdziwie pozytywny)  
ViT - Visual Transformer (Transformator wizyjny)  
VRAM - Video Random Access Memory (Pamięć wideo)  
YOLO - You Only Look Once (Model detekcji obiektów)

# Wstęp

Monitorowanie stanu technicznego infrastruktury budowlanej jest kluczowym procesem zapewniającym bezpieczeństwo eksploatacji obiektów. Jednym z podstawowych wskaźników degradacji struktur betonowych i murowanych jest występowanie pęknięć powierzchniowych. Tradycyjne metody inspekcji wizualnej, oparte na manualnych pomiarach i subiektywnej ocenie ekspertów, charakteryzują się niską powtarzalnością wyników oraz wysoką czasochłonnością. W związku z tym, w inżynierii lądowej oraz diagnostyce obrazowej dąży się do automatyzacji procesu detekcji uszkodzeń z wykorzystaniem metod komputerowego przetwarzania obrazu oraz uczenia maszynowego.

Niniejsza praca dyplomowa przedstawia projekt i implementację zintegrowanego systemu analizy pęknięć powierzchni budowlanych na podstawie analizy obrazów cyfrowych. Głównym założeniem technicznym systemu jest wyeliminowanie błędów pojedynczych estymatorów poprzez zastosowanie technik uczenia zespołowego (*Ensemble learning*), łączącego predykcje zróżnicowanych architektur sieci neuronowych.

System realizuje proces analizy w czterech etapach przetwarzania potokowego:

1. **Wstępna weryfikacja (Kontroler Domeny):** Wykorzystanie konwolucyjnej sieci neuronowej do binarnej klasyfikacji obrazu w celu odfiltrowania próbek niezawierających uszkodzeń
2. **Segmentacja:** Wyodrębnienie maski pęknięcia z tła przy użyciu zespołu modeli: U-Net, SegFormer oraz YOLOv8-seg. Łączenie wyników jest metodą głosowania *soft-voting*.
3. **Klasyfikacja:** Ocena stopnia degradacji poprzez przypisanie do kategorii (włosowe, małe, średnie, duże) w oparciu o zespół klasyfikatorów EfficientNet oraz ConvNeXt.
4. **Analiza geometryczna:** Wyznaczenie parametrycznych cech uszkodzenia, takich jak długość, szerokość, pole powierzchni oraz krętość, z wykorzystaniem algorytmów szkieletyzacji oraz Euklidesowej Transformaty Odległościowej.

## Cel pracy

Celem pracy jest zapewnienie systemu oceny pęknięć powierzchni budowlanych za pomocą sygnałów wizualnych. Jest to realizowane za pośrednictwem pracujących wspólnie modułów sieci neuronowej, których zadaniem jest odpowiednie przygotowanie oraz rozróżnienie ze zdjęcia kluczowych fragmentów, oznaczających ubytek strukturalny, odpowiednie sklasyfikowanie relatywnej powagi usterki oraz analizę geometryczną w oparciu o liczbowe parametry opisujące pęknięcia.

Efektem wykorzystania pracy jest przedstawienie na zadanym zdjęciu wejściowym miejsca, w którym występuje pęknięcie lub pęknięcie, przedstawienie w czytelny sposób na zdjęciu najważniejszych parametrów takie jak szerokość pęknięcia, długość, czy pole, natomiast parametry mniej oczywiste zostaną przedstawione w interfejsie poza samym zdjęciem.

Implementacja rozwiązania została zrealizowana w języku Python z wykorzystaniem biblioteki PyTorch oraz środowiska CUDA do akceleracji obliczeń na GPU. Interfejs użytkownika opracowano w architekturze klient-serwer przy użyciu frameworka Flask, co umożliwia

zdalną analizę materiału zdjęciowego. Praca zawiera szczegółową analizę porównawczą skuteczności modeli, weryfikację metryk oraz ocenę wydajności czasowej poszczególnych modułów w kontekście zastosowań praktycznych.

## Powiązane prace

Większość systemów do wykrywania pęknięć opiera się na architekturze typu Encoder-Decoder. Dobrym punktem odniesienia jest tutaj praca *DeepCrack* [1]. Tam metoda polega na łączeniu cech obrazu z różnych poziomów sieci, co pozwala uzyskać bardzo dokładne wyniki. Mimo to, klasyczne sieci konwolucyjne (takie jak U-Net) analizują obraz przez małe wycinki. Przez to często przerywają ciągłość długich i cienkich pęknięć, traktując je jako osobne plamy, a nie jedną całość.

Aby naprawić problem z ciągłością pęknięć, nowsze badania skupiają się na mechanizmie atencji (*Attention*). Pozwala on sieci analizować cały obraz naraz, a nie tylko jego fragmenty. Przykładem jest *CrackFormer* [2], który lepiej radzi sobie z zachowaniem kształtu pęknięcia. W tym projekcie wykorzystano architekturę SegFormer, ponieważ jest ona hybrydą łączącą szybkość klasycznych sieci CNN z dokładnością Transformerów. Dzięki temu model potrafi wykryć nawet bardzo drobne pęknięcia, których starsze sieci mogłyby nie zauważyc.

Kiedy najważniejszy jest czas działania, standardem są modele z rodziny YOLO (You-Only-Look-Once). Prace takie jak *Object Detection Analysis Study in Images based on Deep Learning Algorithm* [3] pokazują, że YOLO świetnie sprawdza się w inspekcji pęknięć przy drogach. W przeciwieństwie do starszych wersji, które zaznaczały tylko prostokąt wokół uszkodzenia, w tej pracy użyto nowszej wersji YOLOv8-seg. Ten model potrafi jednocześnie znaleźć pęknięcie i stworzyć maskę.

# Rozdział 1

## Część teoretyczna

Konwolucyjne Sieci Neuronowe (*CNN*) to klasa sieci dedykowana do analizy i przetwarzania obrazów. Ich kluczowym elementem jest warstwa splotowa (konwolucyjna).

*CNN* analizuje obraz fragmentami, poszukując lokalnych cech, takich jak krawędzie, linie czy tekstury. Dzięki temu sieć rozpoznaje relacje przestrzenne między pikselami i potrafi rozpoznać poszukiwany obiekt niezależnie od tego, w którym miejscu kadru się on znajduje.

### 1.1 Sieci segmentujące

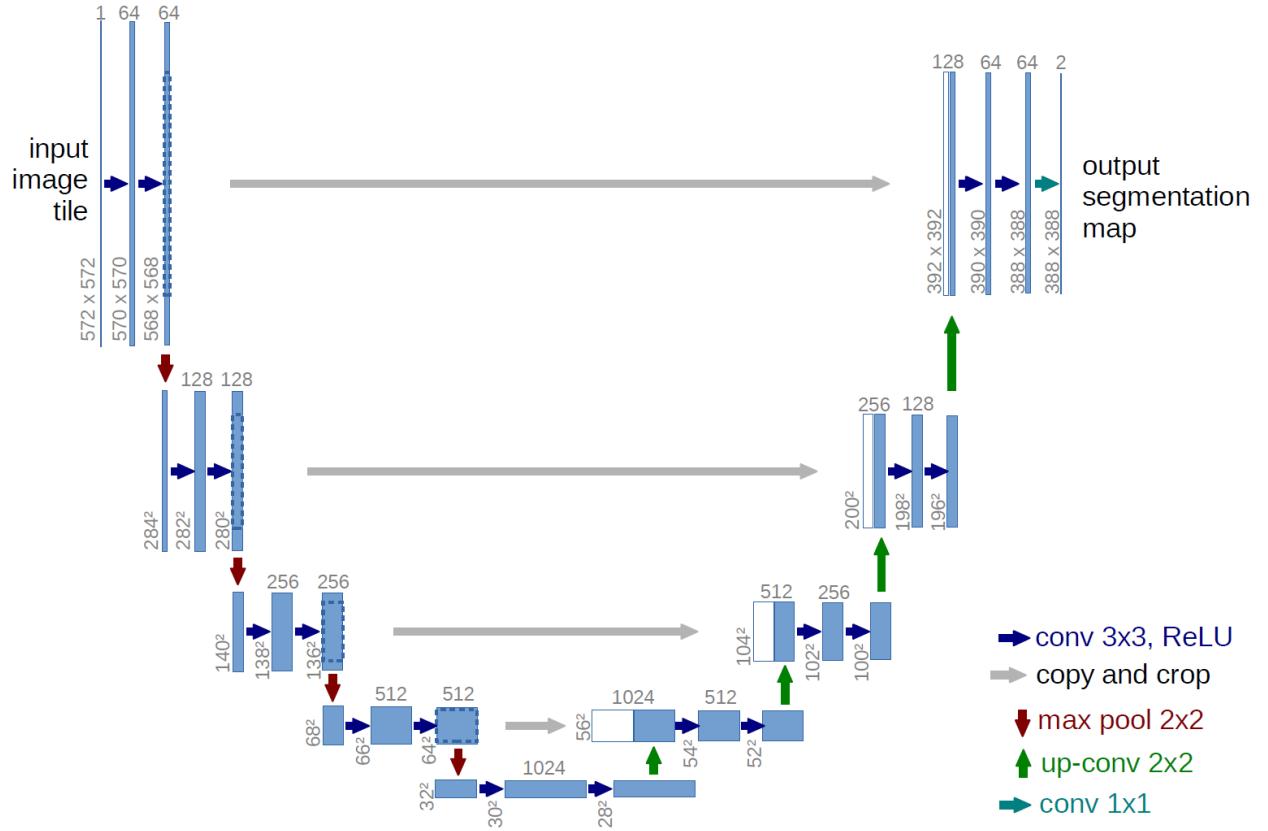
Sieci segmentacyjne to modele, które rozpoznają obecność obiektu na zdjęciu oraz wyznaczają jego granice. Sieć ta analizuje obraz punkt po punkcie, tworząc dokładną maskę oddzielającą obiekt od tła.

#### 1.1.1 U-Net

U-Net to model sieci neuronowej powstały w 2015 roku, którego głównym zadaniem była segmentacja obrazów w dziedzinie medycznej. Jego nazwa pochodzi od jego symetrycznej budowy. Podstawowymi cechami działania modelu sieci U-Net jest wprowadzenie zamiast operacji łączących (pooling operators) operatorów nadpróbkowywania, dzięki którym warstwy zwiększają swoją rozdzielczość, następująca po nich warstwa splotowa może dzięki temu nauczyć się składać bardziej precyzyjny wynik w oparciu o te informacje [4].

Głównymi zadaniami realizowanymi za pomocą U-Netu są segmentacje, zwiększanie rozdzielczości zdjęć oraz dyfuzja, pozwalająca na tworzenie obrazów przy zadanej komendzie tekstowej. Obecnie modele U-Net są wykorzystywane najczęściej do modeli dyfuzyjnych, takich jak *DALL-E*, czy *Midjourney*, których zadaniem jest tworzenie obrazów na podstawie zadanej przez użytkownika instrukcji[5].

W tej pracy wykorzystywany U-Net spełnia zadanie segmentacji, czyli tworzenia zdjęcia z wyróżnionymi elementami na podstawie wyuczonych masek zdjęć (masks, Ground Truths). Dane wyjściowe stanowią wyodrębnione pęknienia w powierzchniach z odfiltrowanymi danymi tła.



**Rysunek 1.1:** Architektura sieci U-Net (przykład dla rozdzielczości  $32 \times 32$  piksele na najniższym poziomie). Każdy niebieski blok odpowiada wielokanałowej mapie cech. Liczba kanałów jest podana nad blokiem. Rozmiar  $x-y$  jest podany przy lewej dolnej krawędzi bloku. Białe bloki reprezentują skopiowane mapy cech. Strzałki oznaczają poszczególne operacje. Źródło: [4, Fig. 1]

Model wykorzystuje podział na enkoder oraz dekoder, co zilustrowano na Rysunku 1.1 (lewa strona to enkoder, prawa to dekoder). Enkoder wykonuje sekwencję operacji splotowych przy użyciu jąder (*kernel*) o wymiarach  $3 \times 3$ , po których następuje nieliniowa funkcja aktywacji ReLU [6]. Kolejnym etapem jest operacja podpróbkowania metodą „*max pooling*” z filtrem  $2 \times 2$ . Proces ten polega na redukcji wymiarów mapy cech (*feature map*) poprzez wybranie wartości maksymalnej z każdego okna sąsiednich czterech wartości. Implementacja ta pozwala na generowanie map cech o niższej rozdzielczości względem danych wejściowych, co umożliwia ekstrakcję cech o rosnącym stopniu abstrakcji (wykrycie różnych kształtów).

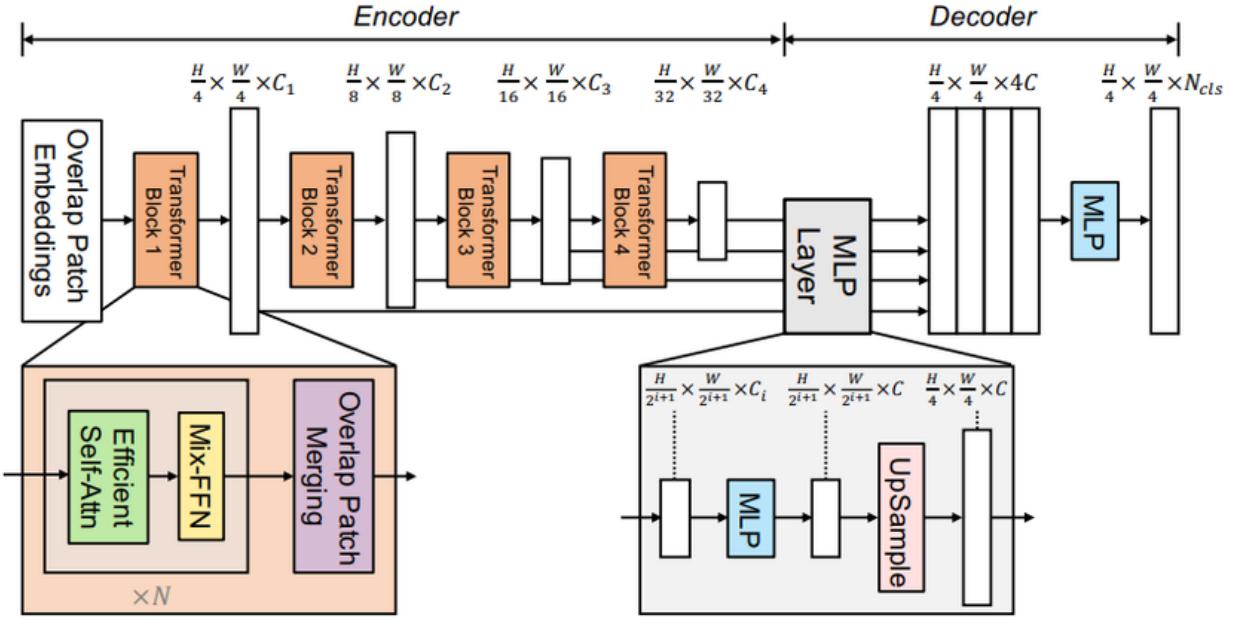
W tej pracy zastosowano model z wgranymi już początkowymi wagami, potrafiącymi rozróżniać podstawowe kształty dzięki enkoderowi *Resnet34*.

### 1.1.2 SegFormer

SegFormer (Segmentation Transformer) to specyficzny model typu Visual Transformer zaprojektowany konkretnie do segmentacji semantycznej, czyli przypisywania klasy każdemu pikselowi obrazu [7] .

Ze względu na swoją szybkość i precyzję, SegFormer jest stosowany w aplikacjach wymagających wysokiej dokładności: inspekcja infrastruktury, pojazdy autonomiczne czy też analiza

zdjęć satelitarnych [8].



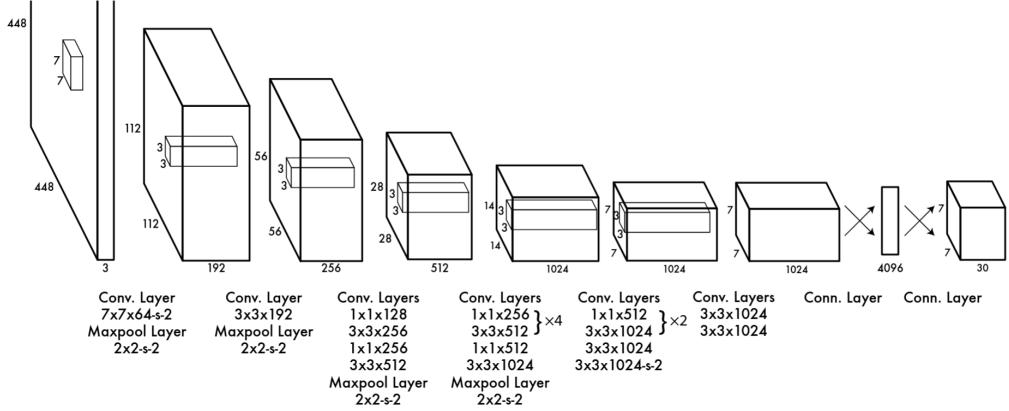
**Rysunek 1.2:** Architektura SegFormer składa się z dwóch głównych modułów: hierarchicznego encodera typu Transformer, służącego do ekstrakcji cech zgrubnych (coarse) i szczegółowych (fine), oraz lekkiego dekodera opartego wyłącznie na warstwach MLP (All-MLP decoder), którego zadaniem jest bezpośrednia fuzja tych wielopoziomowych cech i predykcja maski segmentacji semantycznej. Skrót „FFN” oznacza sieć typu feed-forward (sieć jednokierunkową)[7, Fig. 2]

Zasada działania SegFormera oparta jest o Visual Transformer (ViT). Jest to adaptacja architektury Transformer (pierwotnie stworzonej do przetwarzania tekstu) do zadań związanych z obrazem. W przeciwieństwie do CNN, które analizują obraz lokalnie, Transformery skupiają się na globalnych zależnościach między fragmentami obrazu, wykorzystując mechanizm atencji.

### 1.1.3 YOLO

Głównym celem modeli YOLO (You Only Look Once) jest wykrycie obiektu oraz opisania go w zadanych punktach stanowiących granicę występowania obiektu (bounding box) [9]. Ich głównym atutem jest ich szybkość działania. Występują one w systemach do detekcji obiektów w czasie rzeczywistym, pozwalając na błyskawiczne działanie.

Modele YOLO działają poprzez nałożenie na zdjęcie wejściowe siatki o rozmiarach  $S \times S$ , które następnie tworzą dwie struktury: jedną, stanowiącą siatki przynależności obiektu, jego ramkę graniczną (*bounding box*) z określona pewnoścą wykrycia obiektu; drugą, dla każdej komórki w siatce  $S \times S$  prawdopodobieństwo przynależenia do pewnej klasy obiektu. Ostatecznie łączy obie struktury w jeden, finalny wynik.



**Rysunek 1.3:** Architektura. Nasza sieć detekcyjna składa się z 24 warstw splotowych, po których następują 2 warstwy w pełni połączone (fully connected layers). Naprzemienne stosowane warstwy splotowe  $1 \times 1$  redukują przestrzeń cech z warstw poprzednich. Warstwy splotowe poddajemy wstępнемu trenowaniu (pretraining) na zadaniu klasyfikacji ImageNet przy połowie docelowej rozdzielczości (obraz wejściowy  $224 \times 224$ ), a następnie podwajamy rozdzielczość na potrzeby detekcji [9, Fig. 3] .

## YOLOv8

Wersja *YOLOv8*, wydana w 2023 roku przez *Ultralytics*. W niniejszej pracy wykorzystano wariant *YOLOv8-seg*, który rozszerza standardową detekcję o moduł segmentacji instancji, umożliwiając generowanie dokładnej maski kształtu pęknięcia w czasie rzeczywistym [10].

### 1.1.4 Metryki opisujące sieci segmentujące

#### IoU - Intersect over Union

Podstawowa metryka stosowana w zadaniach segmentacji semantycznej oraz detekcji obiektów. Określa stosunek powierzchni części wspólnej maski predykcji i maski wzorcowej do powierzchni ich sumy. Wartość 1 oznacza idealne dopasowanie, a 0 brak jakiegokolwiek nakładania się obszarów.

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP + FP + FN}, \quad (1.1)$$

gdzie A oznacza zbiór pikseli maski wzorcowej (Ground Truth), a B zbiór pikseli maski predykcji. W ujęciu macierzy pomyłek: TP (True Positive) to obszar poprawnie wykrytego pęknięcia, FP (False Positive) to tło błędnie uznane za pęknięcie, a FN (False Negative) to niewykryty fragment pęknięcia.

#### mIoU

Mean IoU jest to odmiana metryki IoU opisująca średnią wartość IoU dla każdej klasy obiektów do detekcji.

$$\frac{1}{C} \sum_n^C IoU(n), \quad (1.2)$$

gdzie C oznacza liczbę wykrywanych klas obiektów.

## mAP50

mAP50 (Mean Average Precision @ IoU=0.5) Standardowa metryka dla modeli detekcji obiektów (takich jak YOLO). Oblicza średnią precyzję (Average Precision - AP) dla wszystkich klas obiektów, 50 oznacza, że detekcja jest uznawana za poprawną (True Positive), tylko jeśli jej  $IoU$  z ramką wzorcową wynosi co najmniej 0.5 (50%).

$$AP_{50} = \int_0^1 p(r) dr \quad (1.3)$$

$$mAP_{50} = \frac{1}{N} \sum_{i=1}^N AP_{50}^{(i)}, \quad (1.4)$$

gdzie  $p(r)$  oznacza funkcję precyzji w zależności od czułości (krzywa Precision-Recall). Indeks 50 wskazuje, że detekcja jest uznana za popawną tylko wtedy, gdy IoU predykcji z ramką wzorcową wynosi 0.5. N to liczba klas obiektów (w tym projekcie N=1 dla klasy 'crack').

## mAP50-95

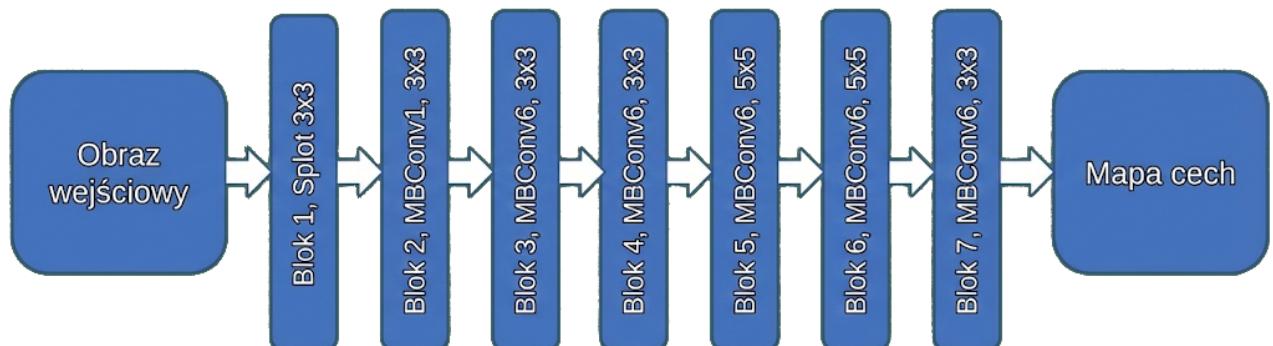
mAP50-95 (mean Average Precision @ IoU=0.5:0.95) to bardziej rygorystyczna wersja metryki mAP, - stanowi średnią arytmetyczną z wartości mAP obliczonych dla dziesięciu różnych progów IoU: od 0.50 do 0.95 z krokiem 0.05. Metryka ta nagradza modele, które nie tylko wykrywają obiekt, ale bardzo precyzyjnie dopasowują do niego ramkę lub maskę.

$$mAP_{50-95} = \frac{1}{10} \sum_{k=0}^9 mAP_{0.5+0.05k} \quad (1.5)$$

## 1.2 Sieci klasyfikujące

Sieci klasyfikujące to algorytmy, które przypisują klasę do badanego obiektu. Analizując obraz jako całość przypisują prawdopodobieństwo przynależności do danej kategorii.

### 1.2.1 EfficientNet



Rysunek 1.4: Architektura modelu EfficientNet-b0

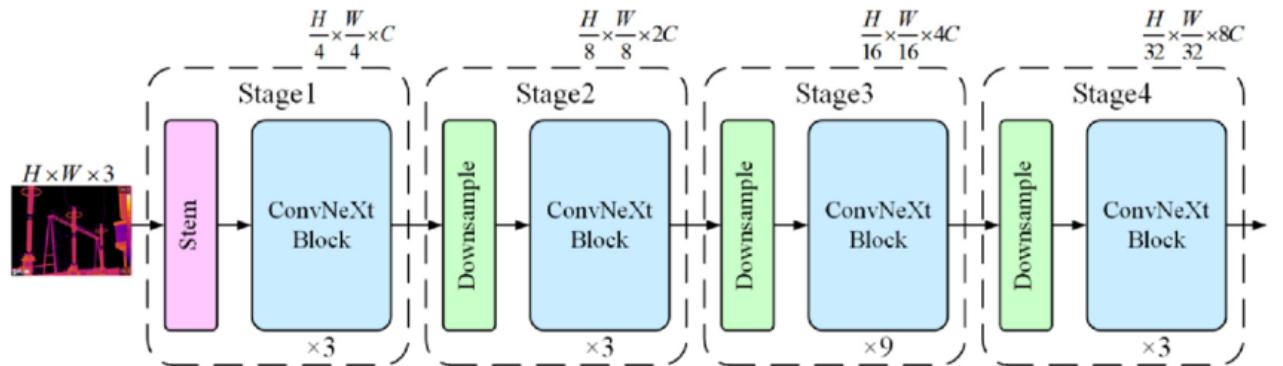
EfficientNet to CNN, której celem jest osiągnięcie wysokiej wydajności przy wykorzystaniu mniejszych zasobów obliczeniowych w porównaniu do wcześniejszych architektur. Została

wprowadzona przez Google Research w artykule z 2019 roku [11].

Główna idea stojąca za EfficientNet jest metoda skalowania, która równomiernie skaluje wszystkie wymiary sieci: głębokość, szerokość oraz rozdzielcość, przy użyciu tzw. współczynnika złożonego (compound coefficient) [12]. W niniejszej pracy wykorzystano wariant EfficientNet-B0, stanowiący model bazowy tej rodziny.

### 1.2.2 ConvNeXt

ConvNeXt [13] to typ modelu uczenia głębokiego używany do celów ekstrakcji cech. Działa on szczególnie dobrze na wizualnych zbiorach danych i może być również stosowany na urządzeniach o mniejszej mocy obliczeniowej, takich jak urządzenia mobilne [14].



Rysunek 1.5: Architektura modelu ConvNeXt-Tiny [15]

Kluczową cechą ConvNeXt jest zastosowanie dużych filtrów splotowych o wymiarze  $7 \times 7$  (w przeciwieństwie do standardowych  $3 \times 3$ ), co znacząco zwiększa pole recepcji sieci i pozwala na lepsze modelowanie kontekstu globalnego. Architektura wykorzystuje bloki z odwróconym przewężeniem (inverted bottleneck), gdzie wymiarowość kanałów jest czterokrotnie zwiększana wewnętrz bloku, a następnie redukowana.

W niniejszym projekcie ConvNeXt wykorzystywany jest jako element układu klasyfikacyjnego (ensemble), wprowadzając dywersyfikację strukturalną względem modelu EfficientNet, co zwiększa ogólną odporność systemu na błędy predykcji.

### 1.2.3 Metryki opisujące sieci klasyfikujące

#### Macierz pomyłek

Jest to tabela wizualizująca skuteczność modelu klasyfikacyjnego. Wiersze reprezentują zazwyczaj rzeczywiste klasy (Ground Truth), a kolumny klasy przewidziane przez model (Predicted). Umożliwia szybką identyfikację, czy model częściej myli pęknięcia z tłem (FN), czy generuje fałszywe alarmy (FP).

Tabela 1.1: Tablica pomyłek dla klasyfikacji binarnej

	Preidykcja: Pęknięcie	Preidykcja: Tło
Rzeczywistość: Pęknięcie	TP (True Positive)	FN (False Negative)
Rzeczywistość: Tło	FP (False Positive)	TN (True Negative)

## Dokładność (Accuracy)

Metryka, określająca stosunek wszystkich poprawnych predykcji (zarówno pozytywnych, jak i negatywnych) do całkowitej liczby badanych próbek. Jest użyteczna tylko wtedy, gdy zbiór danych jest zbalansowany (liczba zdjęć z pęknięciami i bez jest zbliżona).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1.6)$$

## Precyzja (Precision)

Wysoka precyzja oznacza, że model rzadko generuje fałszywe alarmy.

$$Precision = \frac{TP}{TP + FP} \quad (1.7)$$

## Czułość (Recall)

W systemach inspekcyjnych jest to często najważniejsza metryka, ponieważ pominięcie pęknięcia (wysokie FN) może prowadzić do zagrożenia bezpieczeństwa konstrukcji.

$$Recall = \frac{TP}{TP + FN} \quad (1.8)$$

## 1.3 Uczenie zespołowe (*ensemble*)

W celu zwiększenia stabilności predykcji oraz poprawy zdolności generalizacji systemu, zastosowano technikę uczenia zespołowego (Ensemble Learning). Podejście to polega na agregacji wyników kilku niezależnych modeli w celu uzyskania jednego, finalnego rezultatu o wyższej wiarygodności niż w przypadku pojedynczego estymatora [16].

Główną przesłanką stosowania metod zespołowych w segmentacji obrazów jest redukcja wariancji błędu. Różne architektury sieci (w tym przypadku: U-Net, SegFormer, YOLO) charakteryzują się odmienną specyfiką błędów. Ich połączenie pozwala na wzajemną kompensację niedoskonałości, w szczególności, że modele działają w oparciu o różne metody.

Realizowane podejście w tej pracy jest oparte na miękkim głosowaniu (*soft-voting*). Metoda ta operuje na mapach prawdopodobieństwa (przed binaryzacją). Każdy model zwraca wartość z przedziału  $[0,1]$  dla każdego piksela, reprezentującą stopień pewności przynależności do klasy pęknięcia.

Finalny wynik jest średnią arytmetyczną prawdopodobieństw zwróconych przez wszystkie modele w zespole:

$$P_{final} = \frac{1}{N} \sum_{i=1}^N P_i \quad (1.9)$$

Gdzie  $P_i$  to mapa prawdopodobieństwa wygenerowana przez  $i$ -ty model. Otrzymana uśredniona mapa  $P_{final}$  jest następnie poddawana progowaniu (thresholding) w celu uzyskania binarnej maski wynikowej.

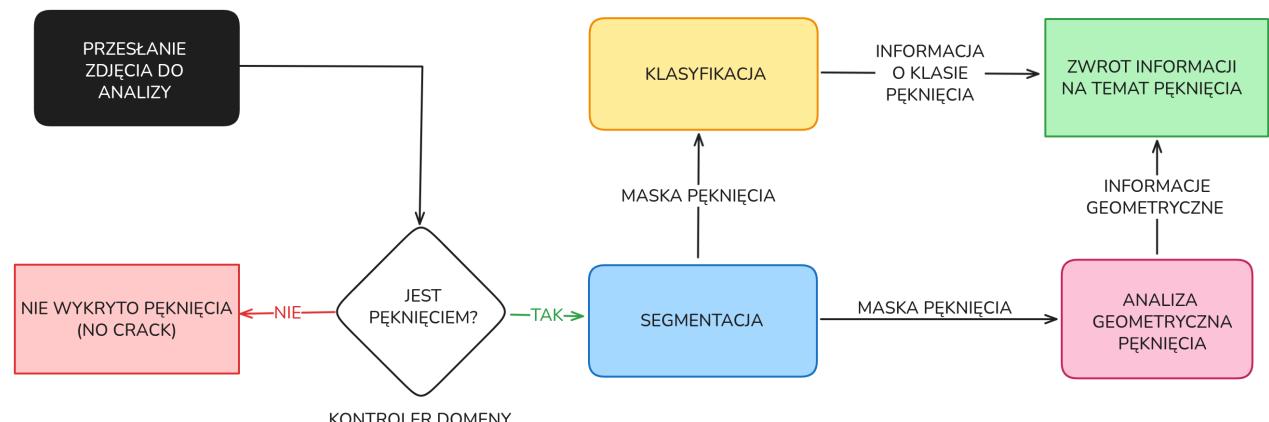
Analogiczne działanie zastosowano dla modeli klasyfikacyjnych, gdzie uśredniane są prawdopodobieństwa zwarcane przez dwa modele klasyfikacyjne, w celu wybrania najbardziej prawdopodobnej klasy pęknięcia.

# Rozdział 2

## Metodyka oraz projekt systemu

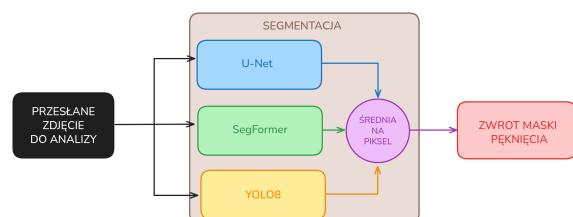
### 2.1 Schemat działania systemu

Logiczną strukturę zaprojektowano w sposób pozwalający na zastosowanie uczenia zespołowego - moduły segmentacyjne oraz klasyfikujące są podzielone na pary sieci kolejno *U-Net* - *SegFormer* - *YOLOv8* oraz *EfficientNet* - *ConvNeXt*.

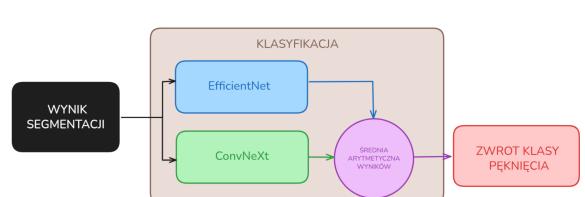


Rysunek 2.1: Schemat ideowy działania programu

Uczenie zespołowe pozwalające na osiągnięcie kompromisu dwóch różnie działających modeli, analizujących dane wejściowe w różny sposób, pozwalając na polepszenie działania sieci. Realizacja została przeprowadzona dla sieci segmentacyjnych w postaci średniej wartości każdego piksela wyniku obu modeli, natomiast dla sieci klasyfikujących obliczane jest średnie prawdopodobieństwo przynależności do każdej z kategorii.



Rysunek 2.2: Schemat ideowy uczenia zespołowego dla sieci segmentacyjnej. Realizacja poprzez uśrednienie predykcji każdego piksela zwróconego z wyjść modeli segmentacyjnych



Rysunek 2.3: Schemat ideowy uczenia zespołowego dla sieci klasyfikującej. Realizacja poprzez uśrednienie prawdopodobieństw klas otrzymanych z modeli klasyfikacyjnych

## 2.2 Uzasadnienie doboru architektur sieci neuronowych

Dobór modeli wchodzących w skład systemu jest kluczowy dla osiągnięcia lepszej dokładności działania modułów uczenia zespołowego. Każda z wybranych architektur przetwarza informacje wizualne w odmienny sposób, co pozwala na minimalizację skorelowanych błędów.

### 2.2.1 Dobór modeli segmentacyjnych

W zadaniu segmentacji pęknięć istotne jest odnalezienie lokalizacji krawędzi drobnych pęknięć oraz wykrycie globalnego trendu topologii uszkodzenia. W tym celu wybrano trzy architektury o fundamentalnie różnych paradymatach działania.

U-Net stanowi model klasycznej sieci CNN o symetrycznej budowie, która pozwala na ekstrakcję różnych cech w oparciu o lokalne okna splotu. Przenoszenie map cech o wysokiej rozdzielczości bezpośrednio z enkodera do dekodera pozwala tej sieci na wykrycie cienkich, włosowatych pęknięć, o których informacja w innych sieciach mogłaby zostać utracona.

W przeciwnieństwie do sieci U-Net, SegFormer opiera się na architekturze Visual Transformer (ViT). Wykorzystuje on mechanizm atencji, który oblicza relacje między lokalnymi fragmentami obrazu, a wszystkimi innymi. Pozwala to na wykrycie globalnej topologii pęknięć, które w przypadku architektury U-Net mogłyby pozostać niewykryte.

Model YOLOv8-seg to detektor jednostopniowy, który traktuje segmentację jako rozszerzenie zadania detekcji obiektów. Jego metoda działania opiera się na generowaniu masek prototypowych, które są następnie przycinane przez wykryte ramki graniczne. Ten model został wybrany ze względu na jego zdolność do wykrywania obiektów, co za tym idzie, dyskryminację tła. YOLO skutecznie eliminuje obszary nie będące pęknięciami, efektywnie filtrując fałszywe alarmy (False Positives) generowane przez bardziej czułe modele U-Net oraz SegFormer.

Takie zestawienie różnie działających modeli powoduje, że ostateczna predykcja wymaga "zgody" co najmniej dwóch modeli względem wykrytego pęknięcia, ostatecznie osiągając lepszą filtrację błędów.

### 2.2.2 Dobór modeli klasyfikacyjnych

W zadaniu oceny powagi uszkodzenia zastosowano modele, które wykorzystują różną głębokość ekstrakcji cech, tworząc razem zbalansowany układ.

Architektura EfficientNet-B0 została zaprojektowana z wykorzystaniem metody skalowania złożonego, która równomiernie optymalizuje głębokość, szerokość i rozdzielczość sieci. Pełni on rolę lekkiego, szybkiego estymatora. Jego zadaniem jest zapewnienie bazowej klasyfikacji przy minimalnym narzucie obliczeniowym.

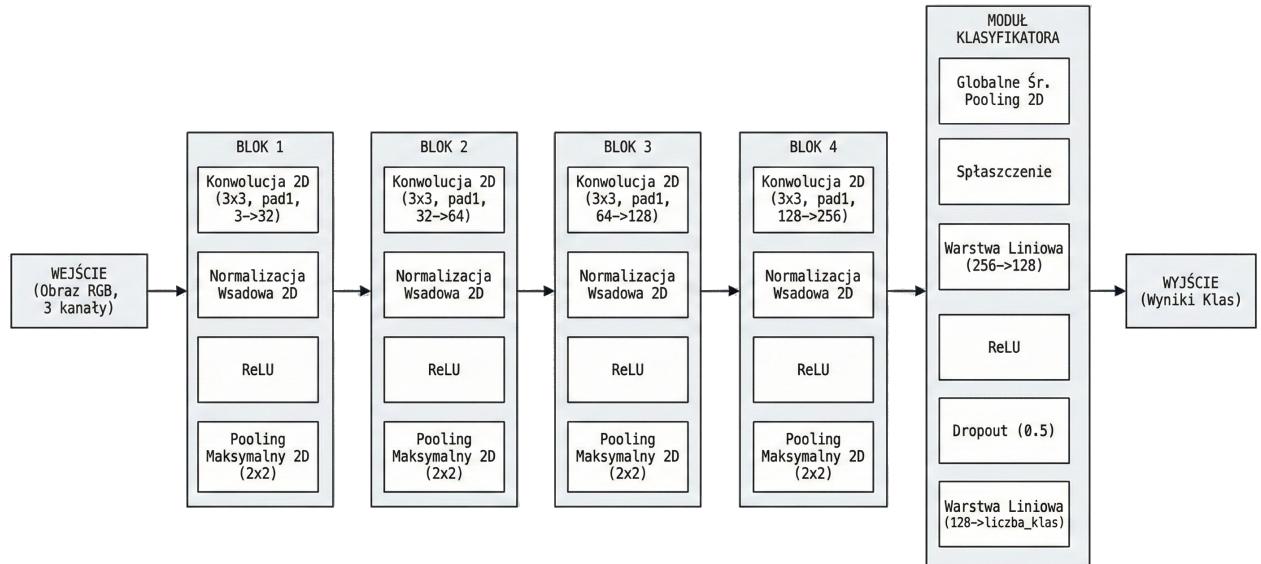
ConvNeXt używa architektury CNN, która adaptuje rozwiązania strukturalne znane z Transformerów, ale zachowuje indukcyjny bias operacji splotowych. Poprzez zastosowanie większych filtrów splotowych zwiększa efektywne pole widzenia sieci zbliżając ją do działania Transformerów. Służy on jako ekstraktor cech zdolny do wychwycenia subtelnych różnic w szerokości pęknięcia które mogą umknąć modelowi EfficientNet. W przypadku tego projektu jest to kluczowa różnica ze względu, iż dane klasyfikacyjne (2.6 Zbiór danych dla modeli

klasyfikacyjnych) zostały podzielone szerokościami, które w miejscach granicznych różnią się nieznacznie od sąsiednich klas.

## 2.3 Kontroler domeny

W celu wstępnej filtracji danych wprowadzono moduł *Kontrolera Domeny*, bazujący na autorskiej architekturze *SimpleClassifier*. Jest to lekka Konwolucyjna Sieć Neuronowa (CNN), której zadaniem jest błyskawiczna klasyfikacja binarna obrazu wejściowego.

Moduł ten pełni funkcję detekcji domeny, odróżnia zdjęcia przedstawiające pęknięcie (klasa pozytywna) od zdjęć nieistotnych, takich jak tło, szum czy inne obiekty (klasa negatywna). Pozwala to na odrzucenie błędnych danych na wczesnym etapie, co oszczędza zasoby obliczeniowe zarezerwowane dla bardziej złożonych modeli segmentacyjnych i klasyfikacyjnych.



**Rysunek 2.4:** Schemat architektury Kontrolera Domeny.

Struktura sieci nawiązuje do klasycznych architektur typu VGG [17], opierając się na sekwencyjnym przetwarzaniu obrazu w blokach ekstrakcji cech. Architektura składa się z czterech głównych bloków konwolucyjnych. Każdy z nich zawiera:

- Warstwę splotową (*Convolution*) z filtrem  $3 \times 3$ , która odpowiada za ekstrakcję lokalnych cech obrazu przy zachowaniu jego wymiarów przestrzennych.
- Normalizację wsadową (*Batch Modeling*), stabilizującą proces uczenia poprzez normalizację aktywacji.
- Nieliniową funkcję aktywacji ReLU (Rectified Linear Unit).
- Warstwę łączenia (*MaxPooling*) z oknem  $2 \times 2$ , redukującą wymiary przestrzenne o połowę, co zwiększa pole widzenia kolejnych warstw.

Liczba filtrów (kanałów) w kolejnych blokach rośnie sekwencyjnie w układzie. Pozwala to sieci na tworzenie coraz bardziej abstrakcyjnych reprezentacji danych.

Istotnym elementem optymalizacyjnym jest zastosowanie na końcu części konwolucyjnej warstwy *Global Average Pooling* (GAP), która redukuje mapy cech o wymiarach ( $C \times H \times W$ ) do wektora o wymiarze ( $C$ ). Rozwiążanie to, w przeciwieństwie do klasycznego spłaszczenia, drastycznie zmniejsza liczbę parametrów modelu, minimalizując ryzyko przeuczenia (*overfitting*).

Ostatnim etapem jest klasyfikator, składający się z dwóch warstw. Pierwsza warstwa redukuje wektor cech z 256 do 128 elementów oraz finalna projekcja do 2 klas wyjściowych.

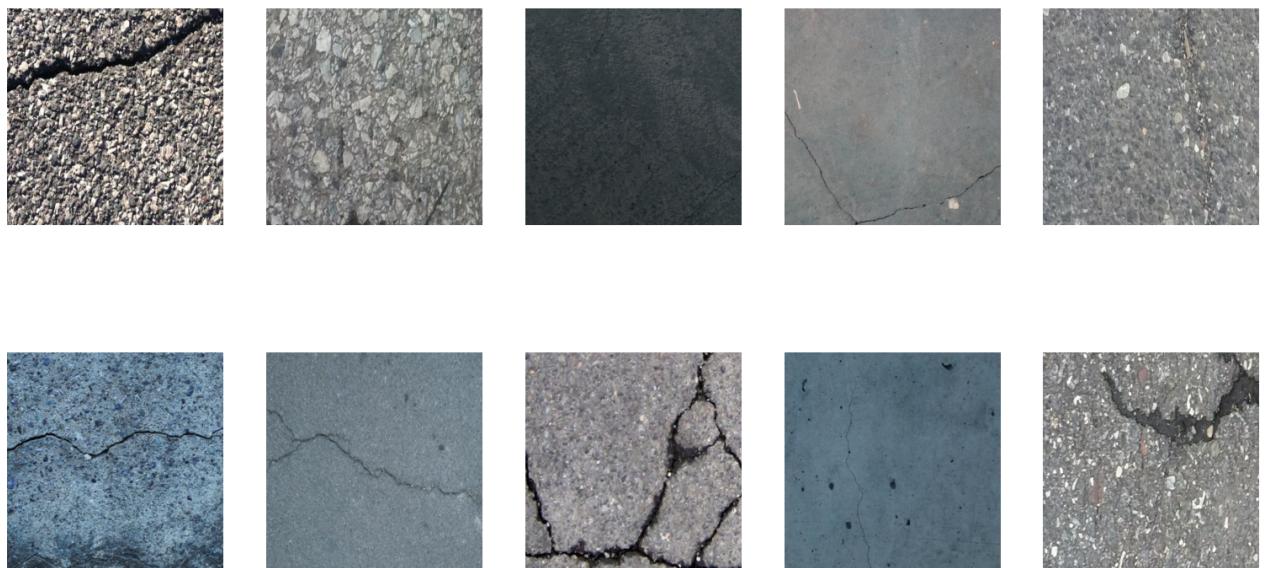
## 2.4 Zbiór danych modeli segmentacyjnych

Dane do uczenia modeli składają się z trzech głównych zbiorów danych:

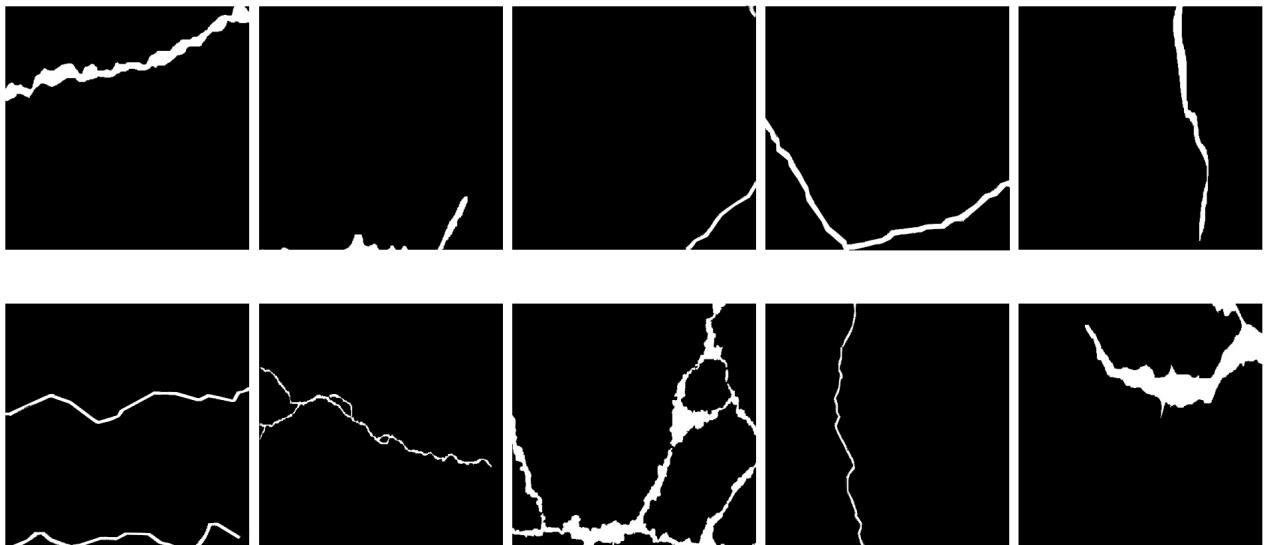
- *DeepCrack* [18],
- *Concrete Crack Images for Classification* [19]
- *Crack Segmentation Dataset* [20],
- *Concrete Crack Images for Classification* [21].

Zostały one wybrane ze względu na ich wysoką jakość oraz różnorodność pęknięć. Zbiory danych zostały połączone w jeden duży, dalej nazywanym *całym zbiorem danych*. Cały zbiór danych został podzielony na zestaw treningowy, zestaw walidacyjny oraz testowy. Pary zdjecie-maska zostały rozdzielone folderami o nazwach `x_img/` oraz `x_lab/`, gdzie `x` oznacza zbiór treningowy, walidacyjny bądź testowy, natomiast "img" oznacza zbiór zdjęć, a "lab" oznacza zbiór mask pęknięć.

Ilość danych (ilość par zdjecie-maska) została podzielona na: zbiór treningowych: **9916**, zbiór walidacyjny: **1710** oraz zbiór testowy: **1710**. Dane treningowe zostały augmentowane, wprowadzając transformacje obrazu takie jak losowe rotacje o kąt, losowe ucinanie, obroty osi x i y. Zrealizowano to w celu minimalizacji *overfittingu*.



**Rysunek 2.5:** Losowo wybrane zdjęcia z całego zestawu danych treningowych

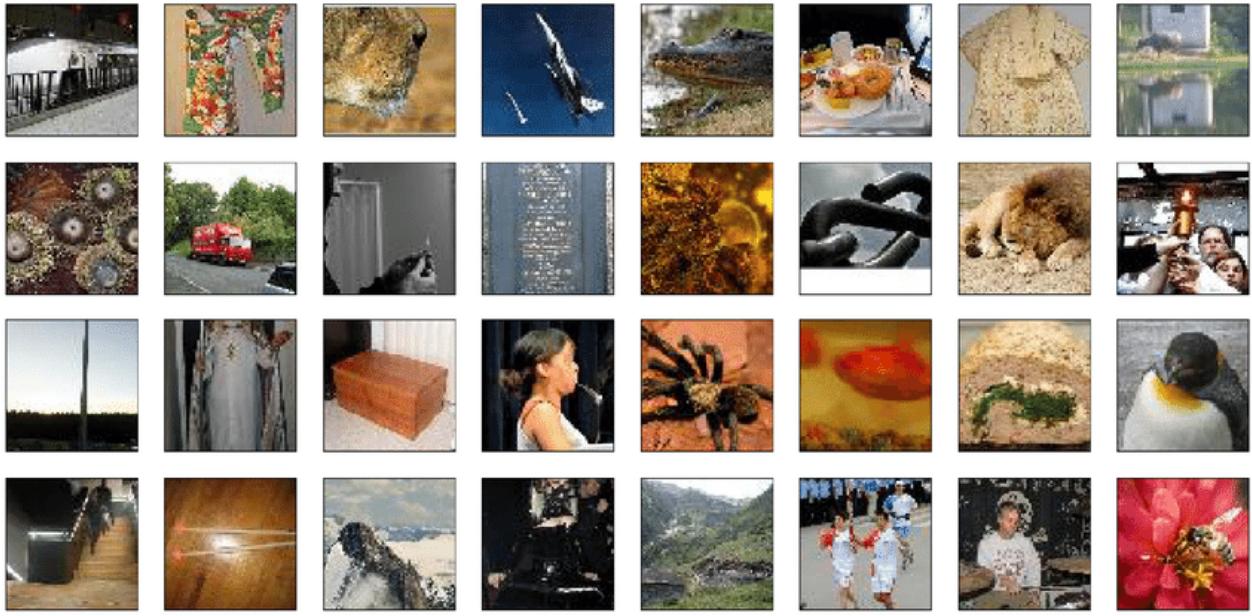


**Rysunek 2.6:** Maski zdjęć z powyższych danych treningowych

## 2.5 Zbiór danych kontrolera domeny

Zbiorem danych kontrolera domeny wybrano połączenie zdjęć pęknięć całego zestawu danych segmentacyjnych (Rysunki 2.5-2.6) oraz danych *Tiny-Image-200*[22]. Dzięki prostemu podziałowi struktury danych na powierzchnię z pęknięciem oraz na zbiór różnorodnych zdjęć zapewnionych w zbiorze danych *Tiny-Image-200* możliwe jest osiągnięcie kontrolera domeny.

Dane treningowe zostały augmentowane, podobnie jak w przypadku danych treningowych segmentacji.

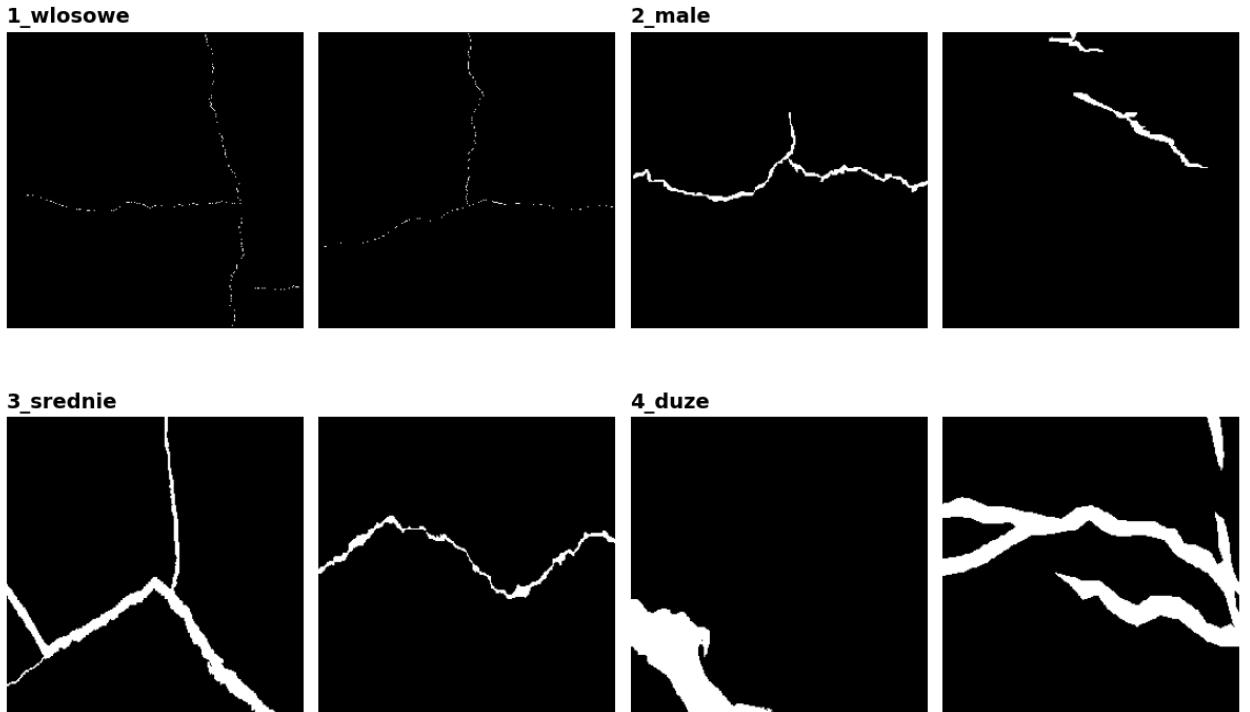


Rysunek 2.7: Losowo wybrane próbki danych zbioru Tiny-Image-200.

## 2.6 Zbiór danych dla modeli klasyfikacyjnych

Kategorie klasyfikacyjne zdefiniowano w oparciu o analizę histogramu szerokości pęknięć w całym zbiorze danych, co pozwoliło na wyodrębnienie czterech klas decyzyjnych.

W celu uzyskania danych opracowano skrypt sprawdzający szerokości pęknięć oraz kopiący te dane do odpowiednich folderów. Dla każdego rodzaju danych (treningowe, walidacyjne oraz testowe) uruchomiono skrypt. Algorytm opisujący wykrycie szerokości opisano w sekcji 2.7 Analiza geometryczna. Podział tych klas jest widoczny na Rysunku 2.8, natomiast w Tabeli 2.1 widoczne są kategorie względem szerokości pęknięcia.

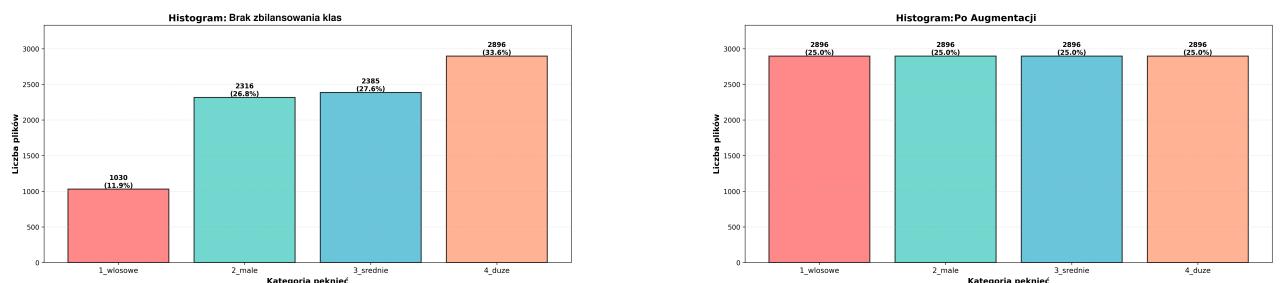


**Rysunek 2.8:** Przedstawienie losowo wybranych danych treningowych dla modeli klasyfikacyjnych

**Tabela 2.1:** Przedstawienie klas pęknięć względem ich zakresów szerokości

KATEGORIA	ZAKRES [px]
WŁOSOWE	[ 1 – 8 ]
MAŁE	[ 9 – 14 ]
ŚREDNIE	[ 15 – 26 ]
DUŻE	> 26

Pierwotny podział danych przedstawiał duże niezbalansowanie klas, stąd postanowiono dodać augmentacje do danych. Ostateczna ilość danych została przedstawiona na Rysunku 2.10. Łączna ilość danych walidacyjnych oraz testowych wynosi tyle samo, jak w przypadku całego zbioru danych segmentacyjnych, natomiast danych treningowych jest **11584**.



**Rysunek 2.9:** Histogram danych treningowych - widoczne niezbalansowanie klas

**Rysunek 2.10:** Histogram danych treningowych po augmentacji danych

## 2.7 Analiza geometryczna

Surowe maski segmentacyjne poddano procesowi post-processingu w celu redukcji szumu i zapewnienia ciągłości topologicznej. Zastosowano sekwencję operacji morfologicznych: filtrację małych obiektów (usuwanie artefaktów tła), wypełnianie otworów oraz domknięcie morfologiczne elementem  $3 \times 3$ , które łączy rozerwane fragmenty pęknięcia.

Podstawowe parametry geometryczne wyznaczono na podstawie analizy momentów obrazu:

- **Powierzchnia ( $A$ ):** Suma pikseli maski
- **Orientacja ( $\theta$ ):** Wyznaczona poprzez dopasowanie ekwiwalentnej elipsy bezwładności.

Do analizy cech liniowych wykorzystano algorytm szkieletyzacji (zachowujący topologię algorytm Zhang-Suena), redukujący pęknięcie do osi środkowej o szerokości 1 piksela. Pozwoliło to na wyznaczenie:

- **Długości ( $L_{geo}$ ):** Suma pikseli szkieletu.
- **Szerokości ( $W$ ):** Zastosowano Euklidesową Transformatę Odległościową (EDT). Algorytm ten przypisuje każdemu pikselowi pęknięcia wartość równą jego odległości do najbliższego piksela tła (krawędzi). Wartość szerokości w danym punkcie szkieletu  $p$  zdefiniowano jako podwojoną wartość tej transformaty (średnicę koła wpisanego):

$$W(p) = 2 \cdot EDT(p) \quad (2.1)$$

- **Krótości ( $\tau$ ):** Stosunek długości rzeczywistej pęknięcia (mierzonej wzduż krzywizny szkieletu) do najkrótszej odległości w linii prostej między jego końcami ( $d(p_{start}, p_{end})$ ):

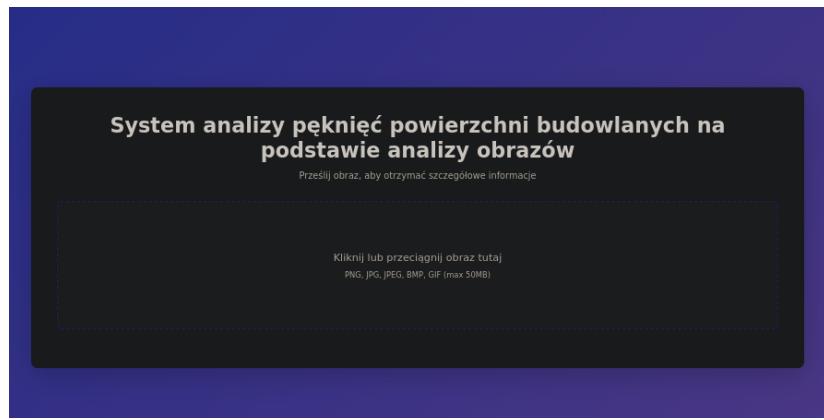
$$\tau = \frac{L_{szkieletu}}{d(p_{start}, p_{end})} \quad (2.2)$$

## Ograniczenia metody (Skrzyżowania i rozgałęzienia)

W przypadku pęknięć o złożonej topologii, takich jak skrzyżowania ("X") lub rozgałęzienia ("Y") występują problemy z jednoznaczną definicją parametrów geometrycznych. Dla pęknięć krzyżujących się, algorytm szkieletyzacji generuje punkty węzłowe, w których spotykają się trzy lub więcej segmentów szkieletu. W takiej konfiguracji wyznaczenie pary "punktów końcowych" ( $p_{start}, p_{end}$ ) staje się niejednoznaczne, co uniemożliwia poprawne obliczenie odległości w linii prostej, a tym samym zaburza wynik współczynnika krętości  $\tau$ .

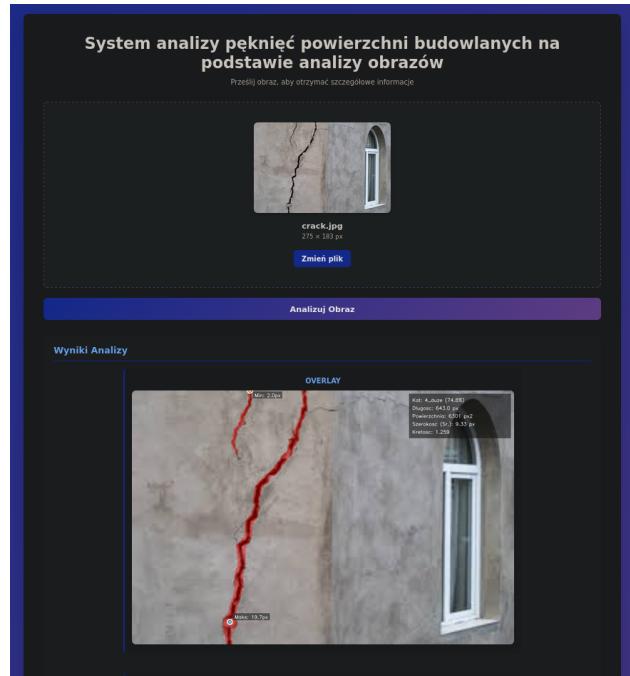
## 2.8 Interfejs użytkownika

Interfejs użytkownika zrealizowano w formie lekkiej aplikacji przeglądarkowej, działającej w architekturze klient-serwer, pozwalającej na dodanie obrazu w celu jego analizy. Poprzez kliknięcie przycisku "Analizuj Obraz" (Rysunek 2.11) przeprowadzona jest weryfikacja danych wejściowych, która określa czy obraz jest powierzchnią z pęknięciem (Rysunek 2.12) lub nią nie jest (Rysunek 2.13). Jeżeli wykryto pęknięcie, system przechodzi do szczegółowej analizy (Rysunek 2.14). System zapewnia tylko rozróżnienie plików graficznych, natomiast nie sprawdza ich pod kątem możliwych błędów, zakładając że dodane zdjęcia są nieuszkodzone.

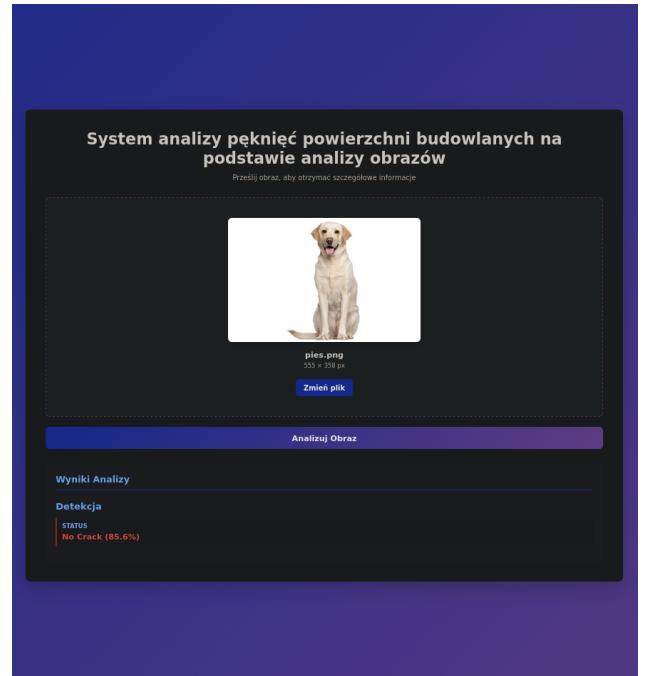


Rysunek 2.11: Interfejs użytkownika - widok powitalny z opcją dodania zdjęcia do analizy

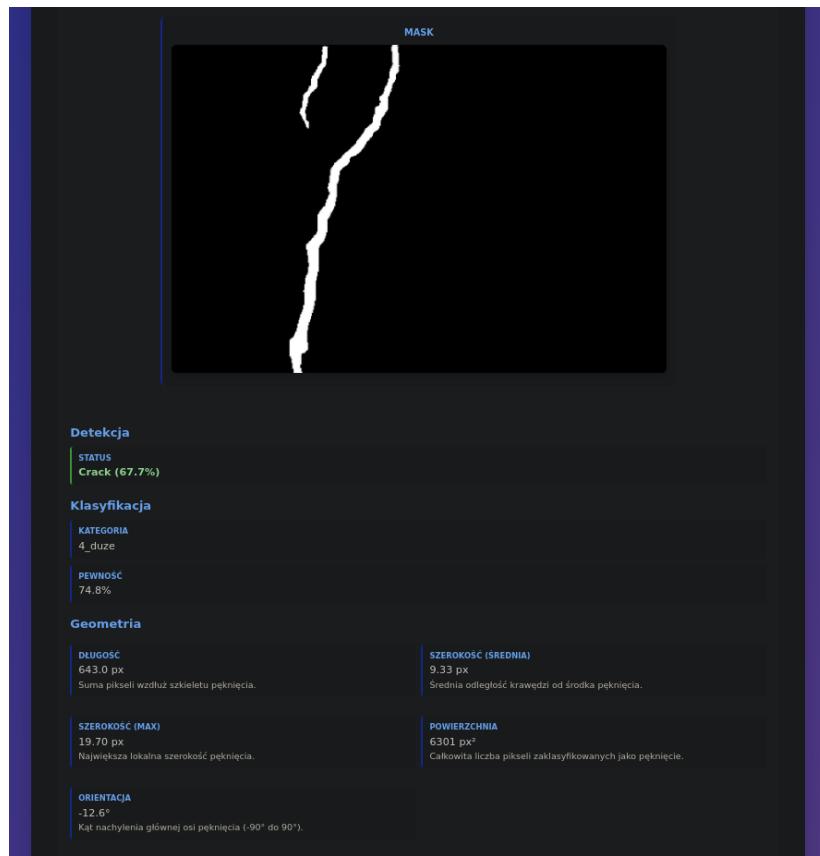
Analiza nakłada na zdjęcie warstwę z wykrytym i wyodrębnionym pęknięciem, Parametry podstawowe prezentowane są w formie nakładki na obraz, zaś szczegółowe metryki w panelu analitycznym (Rysunek 2.14). Obliczane parametry przedstawione poniżej przeanalizowanego zdjęcia są opisane w 2.7 Analiza geometryczna.



Rysunek 2.12: Interfejs użytkownika - przykład przeanalizowanego zdjęcia z wykrytą, pękniętą powierzchnią



Rysunek 2.13: Interfejs użytkownika - brak wykrycia pęknięcia oraz brak dalszej analizy



Rysunek 2.14: Interfejs użytkownika - przedstawienie części wyników analizy pęknięcia

# Rozdział 3

## Implementacja oraz środowisko badawcze

### 3.1 Środowisko badawcze

- Środowisko: *Visual Studio Code* (wersja 1.107).
- Infrastruktura pracy: *PyTorch* 2.5.1, *CUDA* 12.4
- System operacyjny: *Ubuntu* 24.04.3 LTS
- Kontrola wersji: *Git*
- Synchronizacja kodu: *GitHub*
- Środowisko treningowe: *Google Colab* (T4)
- Logowanie procesu uczenia: *TensorBoard*
- Serwer strony z interfejsem użytkownika: *Flask* 3.1.2

Wykorzystanie usługi *Google Colab* jest spowodowane brakiem dostępu do mocy obliczeniowych potrzebnych do wyszkolenia modeli uczenia maszynowego w racjonalnym czasie. Oferowana w darmowym trybie sesji karta graficzna *NVidia T4* zapewnia 16GB pamięci VRAM, pozwalającej na naukę na większej ilości danych niż zasoby lokalne.

### 3.2 Proces uczenia modeli

Nauka prezentowanych modeli odbywała się poprzez pierwotne przetestowanie różnych funkcji strat oraz dostrajanie hiperparametrów metodą siatki (*grid search*), testując dla ograniczonej ilości epok (do 10). Dostarczenie zbiorów danych na sesję *Google Colab* odbywało się poprzez zamieszczenie całego zbioru danych na platformie *Google Drive* oraz dołączenie go bezpośrednio do sesji. Rozpoczęcie treningu modeli zaczynało się w momencie wywołania konkretnego skryptu `train.py`.

#### 3.2.1 Wybrane parametry treningowe

Wybór funkcji strat wymagał, by model poprawnie uczył się mimo nierównowagi klas spowodowanej niską zawartością pęknięć względem tła.

- **CrossEntropy (CE):** Standardowa funkcja straty stosowana w zadaniach klasyfikacji (SimpleClassifier, EfficientNet, ConvNeXt). Mierzy ona rozbieżność między rozkładem prawdopodobieństwa predykcji a etykietą rzeczywistą.

$$L_{CE} = - \sum_c y_c \cdot \log(p_c) \quad (3.1)$$

gdzie  $y_c$  oznacza maskę binarną, a  $p_c$  to przewidziane prawdopodobieństwo.

- **DiceBCE (Dice Loss + Binary Cross Entropy):** Hybrydowa funkcja straty zastosowana w modelu U-Net. Łączy ona zalety dwóch podejść:

- *BCE* zapewnia stabilne gradienty i analizuje każdy piksel niezależnie. Zastosowano parametr `pos_weight=5.0`, który pięciokrotnie zwiększa karę za błędy na pikselach pęknięcia, kompensując przewagę liczebną tła.

$$L_{BCE} = - \frac{1}{N} \sum_{i=1}^N [w_{pos} \cdot y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)] \quad (3.2)$$

gdzie  $N$  to liczba pikseli,  $y_i$  to wartość etykiety (0 lub 1),  $p_i$  to predykcja modelu, a  $w_{pos}$  odpowiada parametrowi `pos_weight`.

- *Dice Loss* jest funkcją opartą na regionach, która bezpośrednio optymalizuje metrykę IoU, kluczową dla jakości segmentacji.

$$L_{Dice} = 1 - \frac{2 \sum_{i=1}^N y_i p_i + \epsilon}{\sum_{i=1}^N y_i + \sum_{i=1}^N p_i + \epsilon} \quad (3.3)$$

gdzie  $\epsilon$  jest małą stałą zapobiegającą dzieleniu przez zero (stabilizacja numeryczna).

- **Tversky Loss:** Zastosowana w modelu SegFormer, będąca uogólnieniem funkcji Dice. Pozwala na sterowanie wagami przypisywanymi błędom typu False Positive (FP) i False Negative (FN).

$$L_{Tversky} = 1 - \frac{\sum_{i=1}^N p_i y_i}{\sum_{i=1}^N p_i y_i + \alpha \sum_{i=1}^N p_i (1 - y_i) + \beta \sum_{i=1}^N (1 - p_i) y_i + \epsilon} \quad (3.4)$$

gdzie  $\alpha$  i  $\beta$  to hiperparametry kontrolujące karę odpowiednio za FP i FN. W projekcie przyjęto wartości  $\alpha = 0.3$  oraz  $\beta = 0.7$ , co kładzie większy nacisk na minimalizację błędów typu False Negative (niewykrycie pęknięcia).

**Tabela 3.1:** Zestawienie parametrów modeli wykorzystanych w projekcie. LR - współczynnik nauki (*learning rate*)

Model	Architektura	Batch Size	LR	Funkcja Straty	Epoki
SimpleClassifier	CNN	32	$1 \cdot 10^{-4}$	CrossEntropy	50
EfficientNet-B0	ImageNet	16	$2 \cdot 10^{-5}$	CrossEntropy	25
ConvNeXt-Tiny	ImageNet	16	$2 \cdot 10^{-5}$	CrossEntropy	30
U-Net	ResNet34	16	$1 \cdot 10^{-4}$	DiceBCE ( <code>pos_weight=5.0</code> )	35
SegFormer	MiT-B0	16	$6 \cdot 10^{-5}$	TverskyLoss ( $\alpha = 0.3, \beta = 0.7$ )	35
YOLOv8	YOLOv8m-seg	16	$1 \cdot 10^{-3}$	Box + Seg + Cls	100

Liczba epok została wybrana empirycznie, przestano trenować modele w momencie osiągnięcia zbieżności dokładności.

### 3.2.2 Potok przesyłania kodu oraz danych

W celu wykorzystania zasobów *Google Colab* należało stworzyć system dostarczania kodu źródłowego, danych oraz modeli do wznawiania nauki. Kod źródłowy pracy został umieszczony na platformie *GithHub*, z której następnie otwierano sesję *Google Colab* za pomocą pliku notatnika *Jupyter* (.ipynb), w którym zaimplementowano pobieranie reszty kodu źródłowego oraz nawigację kodu sposób pozwalający go uruchomić z poziomu głównego interfejsu *Google Colab*.

Po zakończeniu procesu uczenia modeli zapisywano model końcowy oraz logi w postaci pliku *TensorBoard*. Ze względu na to, że nie da się bezpośrednio edytować kodu (z plików .py) w sesji *Google Colab*, każdorazowa zmiana kodu wymagała restartu sesji oraz ponownego pobrania repozytorium *GithHub* z kodem źródłowym.

### 3.2.3 Ograniczenia

Głównym ograniczeniem działania projektu jest dostęp do zasobów obliczeniowych na szkolenie modeli segmentacyjnych i klasyfikacyjnych. Ze względu na korzystanie z zasobów udostępnionych przez *Google Colab* o darmowym trybie dostępowym, wymagało wielokrotnego zatrzymywania i wznawiania procesu nauki. Brak informacji o pozostałym, dostępnym czasie sesji powodował rozłączenie oraz bezpowrotną utratę wyszkolonych modeli zdobytych podczas trwania sesji, skutkując w częstym procesie ponawiania nauki utraconych modeli.

### 3.2.4 Struktura kodu źródłowego

Kod źródłowy został podzielony ze względu na zadanie na podfoldery odpowiadające za: klasyfikację, segmentację, kontroler domeny, analizę geometryczną, potoku ostatecznej predykcji, serwer oraz funkcje pomocnicze.

Struktura każdego z modeli została podzielona w ten sam sposób:

- hparams.py - plik ze stałymi parametrami oraz ścieżkami do plików
- model.py - implementacja modelu, sposobu ładowania, zapisywania do pamięci itp.
- fine\_tuning.py - szukanie dogodnych hyperparametrów dla szkolonych modeli
- train.py - pętla treningowa uruchamiana z Pythona
- inference.py - podgląd działania modelu

W przypadku modeli klasyfikacji oraz segmentacji wyodrębniono części wspólne kodu oraz dodano je do folderu `common/`, a każdy konkretny wariant modelu miał tylko inne pętle treningowe, dostrajanie (`fine_tuning.py`) oraz inferencję.

Najważniejszym fragmentem jest `colab_training/` zawierający główny notatnik *Jupyter*, który jest obsługiwany przez *Google Colab*.

## 3.3 Implementacja analizy geometrycznej

Moduł zaimplementowano wykorzystując biblioteki *NumPy*, *scikit-image* oraz *OpenCV*. Architektura rozwiązania opiera się na sekwencyjnym przetwarzaniu maski binarnej w podejściu hybrydowym, łączącym analizę konturów i regionów (momenty obrazu, szkieletyzacja) w

ramach jednego przebiegu algorytmu. W celu optymalizacji zasobów, blok analityczny uruchamiany jest warunkowo, wyłącznie po wykryciu uszkodzenia (kontroler domeny). Wyznaczone parametry są zwracane w formie struktury danych (json) oraz wizualizowane na obrazie źródłowym w formie nałożenia maski o niskiej przezroczystości.

## 3.4 Serwer HTTP i Interfejs Użytkownika

System został zbudowany w architekturze klient-serwer, wykorzystując framework *Flask* (Python). Serwer pełni rolę pośrednika, odbierając dane od użytkownika, przekazuje je do modułu analizy, a następnie odsyła wyniki. Dzięki temu użytkownik korzysta z systemu przez przeglądarkę, bez konieczności instalowania oprogramowania na własnym komputerze.

### 3.4.1 Komunikacja i przesył danych

Wymiana danych odbywa się za pośrednictwem protokołu HTTP z wykorzystaniem metody *POST*. Proces ten przebiega następująco:

1. **Żądanie (Request):** Gdy użytkownik zatwierdza wybór zdjęcia, przeglądarka wysyła je na adres serwera.
2. **Przetwarzanie:** Serwer odbiera plik, weryfikuje jego format i przekazuje go do potoku analitycznego. W tym czasie serwer oczekuje na zakończenie obliczeń.
3. **Odpowiedź (Response):** Po zakończeniu analizy serwer generuje odpowiedź w formacie *JSON*. Zastosowano kodowanie *Base64*, pozwalające zamienić wynikowe obrazy (maski pęknięć) na ciąg znaków tekstowych. Zastosowano takie podejście ze względu na relatywnie mały rozmiar tych obrazów.

### 3.4.2 Interfejs użytkownika

Warstwa prezentacji została zrealizowana w oparciu o standardowe technologie webowe, gdzie *HTML* i *CSS* odpowiadają za strukturę oraz responsywny wygląd interfejsu. Interaktywność zapewnia skrypt *JavaScript*, który realizuje asynchroniczną komunikację z serwerem i dynamicznie aktualizuje wyniki analizy bez konieczności przeładowywania strony. Dodatkowo zapewniono funkcjonalność dla mniejszych ekranów, odpowiednio zmieniając marginesy elementów stanowiących strukturę strony.

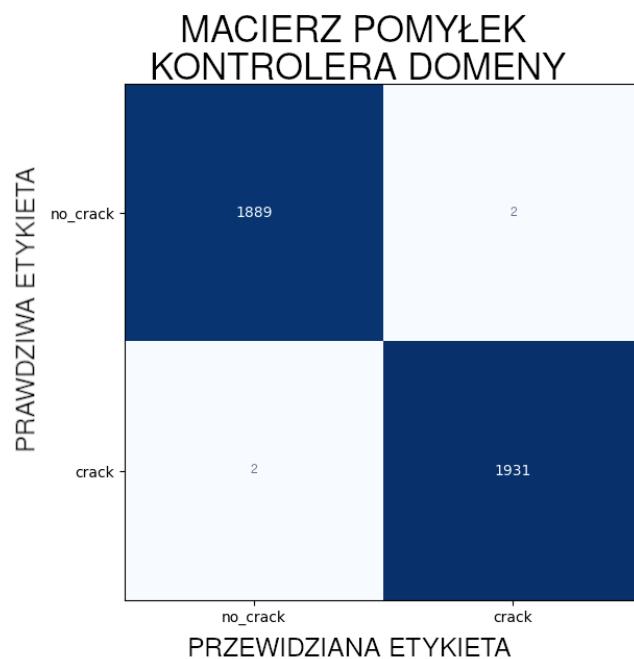
# Rozdział 4

## Analiza i porównanie wyników

### 4.1 Weryfikacja skuteczności kontrolera domeny

Przeprowadzono weryfikację działania kontrolera domeny stanowiącego binarną klasyfikację obrazu wejściowego. Weryfikację przeprowadzono na zbiorze testowym liczącym 3824 próbki.

#### 4.1.1 Macierz pomyłek i metryki binarne



Rysunek 4.1: Macierz pomyłek kontrolera domeny dla danych testowych

Kluczowym parametrem dla bezpieczeństwa systemu jest minimalizacja błędu False Negative. Wartość FN=2 oznacza, że ryzyko pominięcia uszkodzenia powierzchni jest marginalne. Wystąpienie FP=2 natomiast oznacza, że model przekazuje do modeli segmentacyjnych zdjęcia, które nie stanowią pęknięcia. Skutkuje to przeprowadzeniem pełnej analizy dla zdjęcia bez pęknięcia, otrzymując bezsensowne wyniki. Stosunek poprawnie sklasyfikowanych zdjęć względem tego błędu natomiast jest mała, ale nie pomijalna.

### 4.1.2 Analiza wydajnościowa

Weryfikacja narzutu czasowego została przeprowadzona w środowisku CUDA. Zgodnie z logami procesu validacji:

- Całkowity czas przetwarzania zbioru walidacyjnego (3820 próbek): 18.24 s.
- Średni czas inferencji na jedną próbke  $t_p = 4.77$  ms.

Uzyskany wynik  $t_p$  przedstawia szybki czas predykcji. W kontekście całego systemu, gdzie czas inferencji najszybszego modelu (Ensemble) 51.59 ms (4.7 Analiza wydajnościowa i zasobowa), narzut kontrolera domeny jest akceptowalny. Pozwala to na uproszczenie pracy modeli klasyfikacji oraz segmentacji, które nie muszą rozróżniać stanu "braku pęknienia".

## 4.2 Analiza porównawcza modeli klasyfikacyjnych

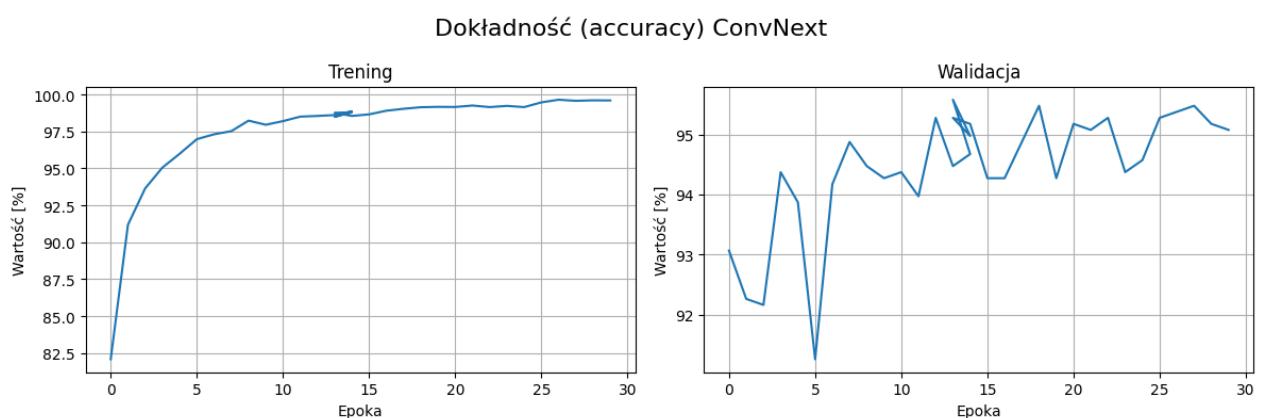
Proces uczenia modeli klasyfikacji został przeanalizowany wykorzystując jednostkę obliczeniową CPU oraz CUDA.

### 4.2.1 Przebieg procesu uczenia

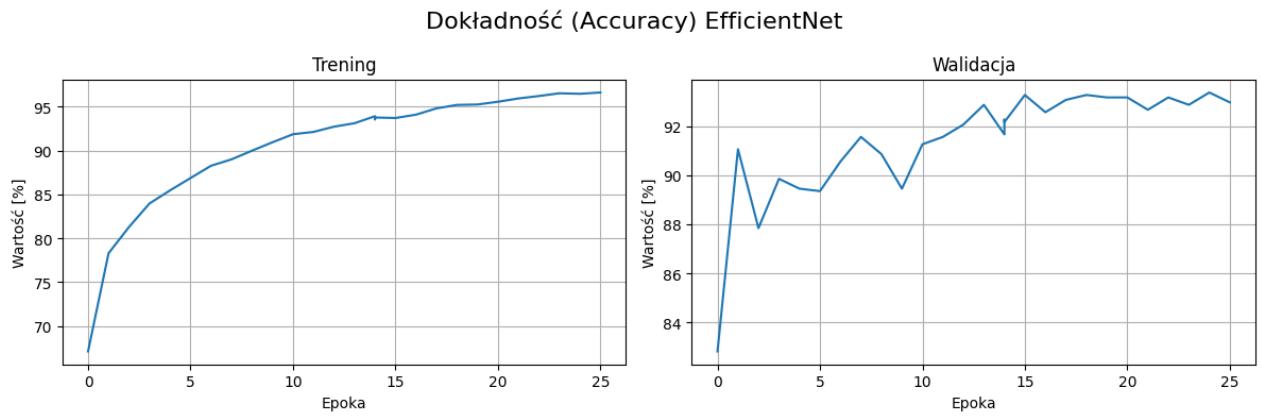
Poniżej przedstawiono wykresy procesów nauki modeli klasyfikacyjnych oraz zinterpretowano osiągnięte wyniki.

#### Porównanie tempa wzrostu dokładności (*Accuracy*)

Analiza krzywych dokładności (Rysunki 4.2-4.3) wykazuje prawidłową zbieżność obu architektur, jednak o różnej dynamice. Model ConvNeXt osiąga stabilizację dokładności na poziomie powyżej 95% już w okolicach 15 epoki. W przypadku modelu EfficientNet (Rysunek 4.3) obserwuje się większą wariancję metryk walidacyjnych w początkowej fazie (epoki 0–10), co następuje w drugiej połowie procesu treningowego, osiągając finalnie pułap 93%.



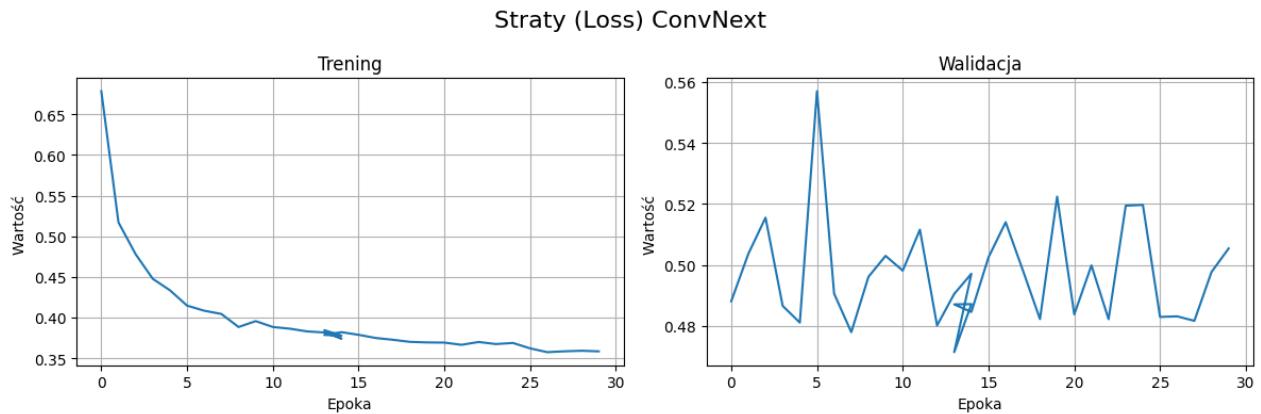
Rysunek 4.2: Wykres dokładności sieci ConvNeXt trwającej 30 epok



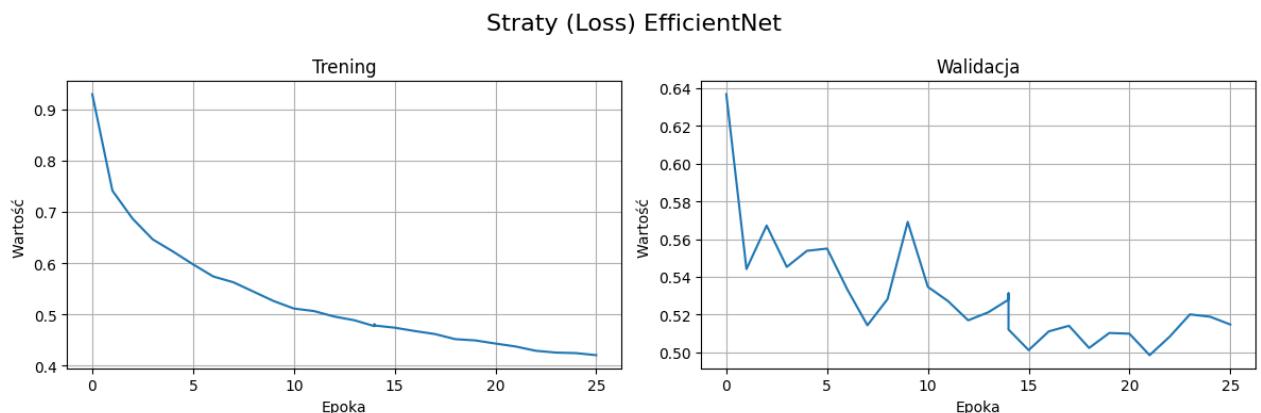
Rysunek 4.3: Wykres dokładności sieci EfficientNet trwającej 30 epok

#### Porównanie stabilności procesu nauki (*loss*)

Przebieg funkcji straty dla zbioru walidacyjnego (Rysunki 4.4-4.5) ujawnia istotne różnice w stabilności uczenia. Dla modelu ConvNeXt krzywa walidacyjna podąża blisko krzywej treningowej, co sugeruje optymalne dopasowanie pojemności modelu do zbioru danych. W przypadku EfficientNet widoczne są lokalne oscylacje amplitudy straty, mimo monotonicznego spadku błędu na zbiorze treningowym.



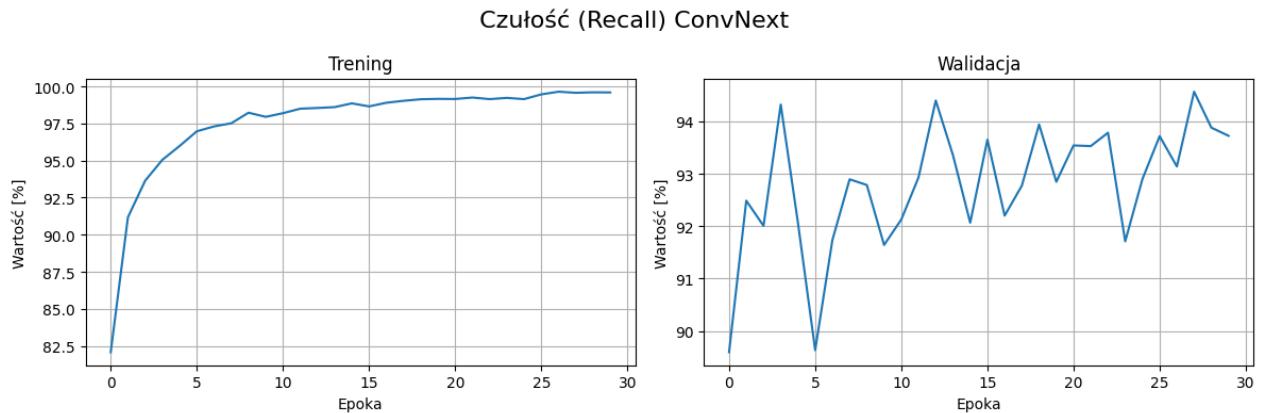
Rysunek 4.4: Wykres funkcji strat sieci ConvNeXt trwającej 30 epok



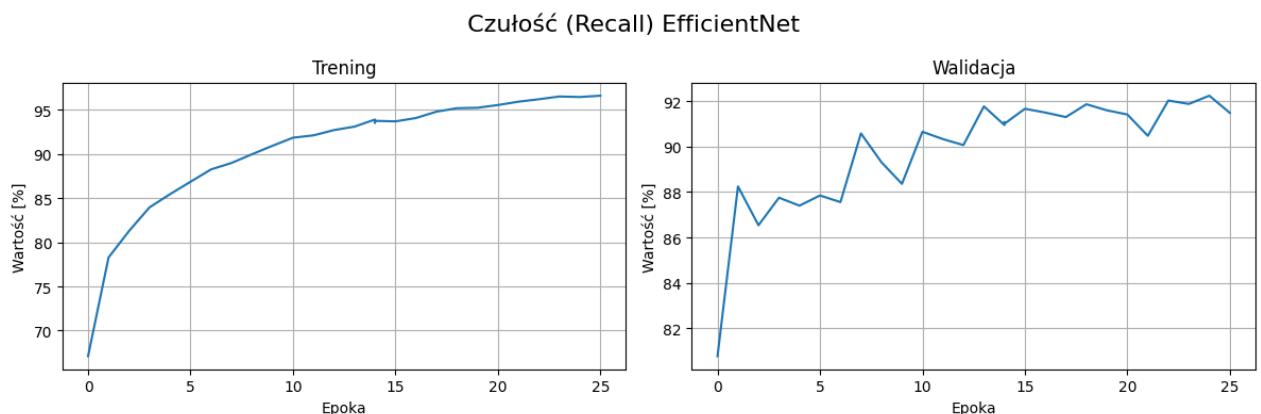
Rysunek 4.5: Wykres strat sieci EfficientNet trwającej 30 epok

## Porównanie czułości (*recall*)

W systemach detekcji uszkodzeń parametr czułości jest kluczowy, gdyż minimalizuje ryzyko niewykrycia pęknięcia (błąd False Negative). Jak przedstawiono na Rysunku 4.6, model ConvNeXt utrzymuje czułość walidacyjną w stabilnym przedziale 93–95%, wykazując wzrost monotoniczny. EfficientNet (Rysunek 4.7) charakteryzuje się znacznie większą zmiennością wartości Recall (zakres 82 – 92%) w początkowych epokach. Stabilizacja następuje dopiero po 20 epoce, co potwierdza konieczność dłuższego procesu treningowego dla tej architektury w celu poprawnej generalizacji cech granicznych.



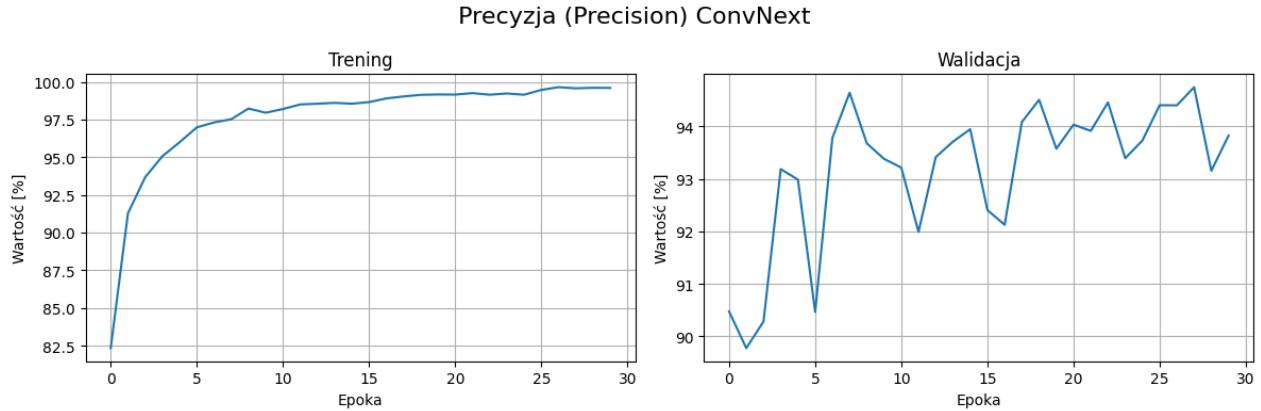
Rysunek 4.6: Wykres czułości sieci ConvNeXt trwającej 30 epok



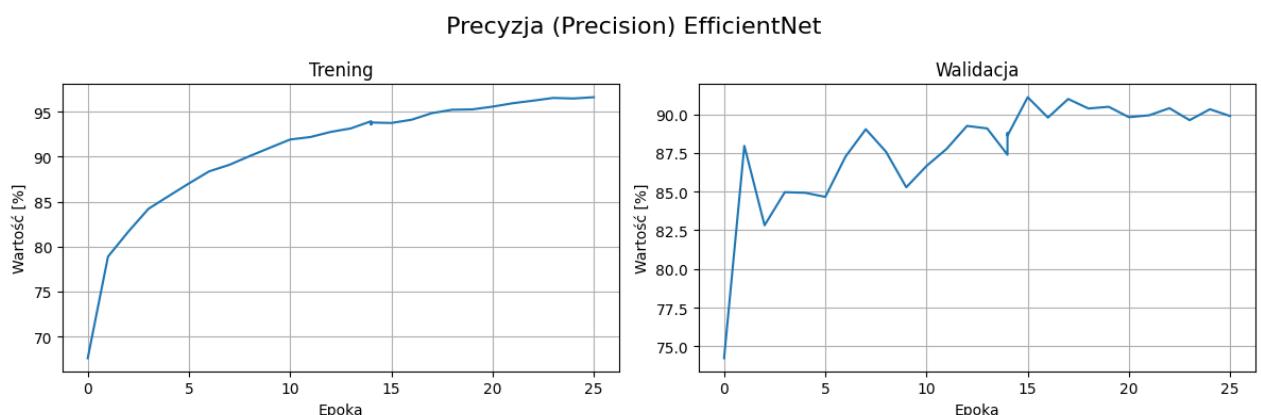
Rysunek 4.7: Wykres czułości sieci EfficientNet trwającej 30 epok

## Porównanie precyzji (*precision*)

Analiza metryki precyzji (Rysunki 4.8-4.9) wskazuje na zdolność modeli do minimalizacji fałszywych alarmów (False Positives). Model ConvNeXt osiąga nasycenie metryki już w 5. epoce, utrzymując ją na poziomie zbliżonym do 95%. Oznacza to, że model szybko uczy się dyskryminować tło od pęknięcia. EfficientNet wykazuje tendencję rosnącą przez cały okres trwania treningu, osiągając maksimum w ostatniej epoce, co sugeruje, że dalsze wydłużenie procesu uczenia mogłoby przynieść marginalną poprawę tego parametru.



Rysunek 4.8: Wykres precyzji modelu ConvNeXt



Rysunek 4.9: Wykres precyzji sieci EfficientNet trwającej 30 epok

Proces optymalizacji modeli klasyfikacyjnych wykazuje zróżnicowaną dynamikę zbieżności. Architektura ConvNeXt osiąga nasycenie metryk treningowych szybciej (po 20. epoce) w porównaniu do modelu EfficientNet, który wymaga około 15 epok do osiągnięcia wstępnej zbieżności. Analiza przebiegu funkcji straty dla zbioru walidacyjnego (Rysunki 4.4-4.5) wykazuje lokalne oscylacje o amplitudzie dochodzącej do 4 punktów procentowych, co przy jednoczesnym monotonicznym spadku straty treningowej wskazuje na tendencję do przeuczenia (*overfitting*). Ostateczne metryki walidacyjne ( $\text{Accuracy} > 93\%$ ) potwierdzają jednak zdolność modeli do poprawnej ekstrakcji cech semantycznych mimo obserwowanej niestabilności walidacyjnej.

#### 4.2.2 Skuteczność klasyfikacji

Wysoka wartość dokładności dla wszystkich modeli świadczy o poprawnym zamodelowaniu granic decyzyjnych, mimo że są one umowne i wynikają z ciągłego charakteru cechy szerokości. Czułość, oznaczająca zdolność systemu do wykrywania wszystkich klas uszkodzeń wynosi powyżej 95%, oznaczając minimalizację błędu *False Negative*. W połączeniu z wysoką wartością parametru precyzji, oznaczając minimalizację błędu *False Positive*, grube błędy są minimalizowane.

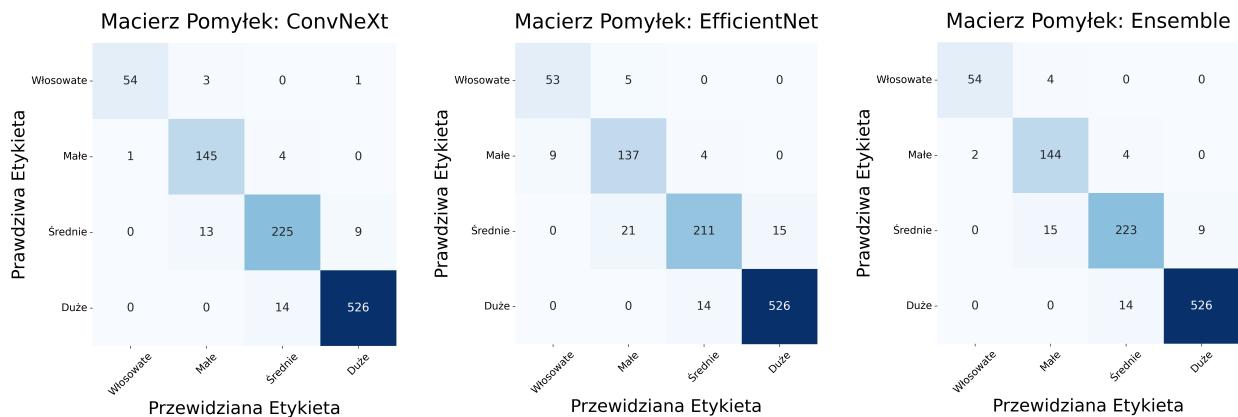
**Tabela 4.1:** Zestawienie metryk skuteczności (F1, Accuracy, Recall, Precision) dla badanych architektur na zbiorze testowym.

MODEL	F1 [%]	ACC [%]	RECALL [%]	PRECISION [%]
CONVNEXT	95.49	95.48	95.48	95.54
EFFICIENTNET	93.17	93.17	93.17	93.29
ENSEMBLE	95.19	95.18	95.18	95.27

### 4.2.3 Analiza błędów

Analiza macierzy pomyłek z Rysunku 4.10 przedstawia dla wszystkich badanych modeli wykazuje wyraźną tendencję do popełniania błędów niemal wyłącznie pomiędzy klasami sąsiednimi (np. pęknienia "Włosowane" mylone z "Małymi"). Jest to zjawisko wynikające z ciągłego charakteru cechy, jaką jest szerokość pęknienia. Granica decyzyjna między klasami jest umowna, co sprawia, że próbki graniczne są trudne do jednoznacznej oceny nawet dla eksperta.

Kluczowym osiągnięciem jest fakt, że modele nie popełniają tzw. błędów grubych. Jak wynika z macierzy, liczba pomyłek między klasami skrajnymi (np. sklasyfikowanie pęknienia "Włosowatego" jako "Duże") wynosi zero lub jest marginalna. Oznacza to, że system poprawnie interpretuje semantykę obrazu, a błędy wynikają jedynie z niepewności na granicach przedziałów szerokości.



**Rysunek 4.10:** Tabele pomyłek kolejno ConvNeXt, EfficientNet oraz Ensemble obu modeli

### 4.2.4 Analiza wydajnościowa

Analiza wydajnościowa została przeprowadzona w celu sprawdzenia narzutu czasowego oraz obliczeniowego. Kluczowymi parametrami są czas predykcji ( $t_p$ ) oraz przepustowość układu wyrażona w klatkach na sekundę ( $FPS$ ).

**Tabela 4.2:** Porównanie działania modeli klasyfikacyjnych z uwzględnieniem parametru FPS.

Zasoby	PARAMETR	ConvNeXt	EFFICIENTNET	ENSEMBLE
<i>CPU</i>	<b>RAM [MB]</b>	143.44	143.44	358.22
	$t_p$ [ms]	84.73	30.15	113.97
	<b>FPS [1/s]</b>	11.80	33.17	8.77
	<b>DOKŁADNOŚĆ (ACCURACY) [%]</b>	95.48	93.07	95.18
<i>CUDA</i>	<b>VRAM [MB]</b>	143.44	143.44	143.44
	$t_p$ [ms]	36.75	30.03	67.02
	<b>FPS [1/s]</b>	27.21	33.30	14.92
	<b>DOKŁADNOŚĆ (ACCURACY) [%]</b>	95.48	93.07	95.18

EfficientNet wykazał najmniejszy narzut obliczeniowy będąc uruchomionym na CPU, osiągając średni czas predykcji  $t_p = 30.15$  ms. Jest to wynik niemal trzykrotnie lepszy od architektury ConvNeXt oraz blisko czterokrotnie lepszy od układu Ensemble. Wskazuje to na EfficientNet jako optymalny wybór dla implementacji na jednostkach centralnych bez akceleracji sprzętowej, kosztem nieznacznie niższej dokładności.

Zastosowanie akceleracji GPU zniwelowało znaczną część różnic w czasie inferencji pomiędzy pojedynczymi modelami. ConvNeXt osiągnął porównywalną szybkość działania, zbliżając się do wyniku EfficientNet. Układ Ensemble, mimo wykorzystania GPU, charakteryzuje się najniższą wydajnością, co wynika z konieczności sekwencyjnego przetwarzania przez oba modele składowe oraz narzutu operacji agregacji wyników (soft-voting).

## 4.3 Analiza porównawcza modeli segmentacyjnych

Proces uczenia modeli segmentacyjnych został przeanalizowany w oparciu o środowisko CUDA, ze względu na wysokie zapotrzebowanie na pamięć operacyjną.

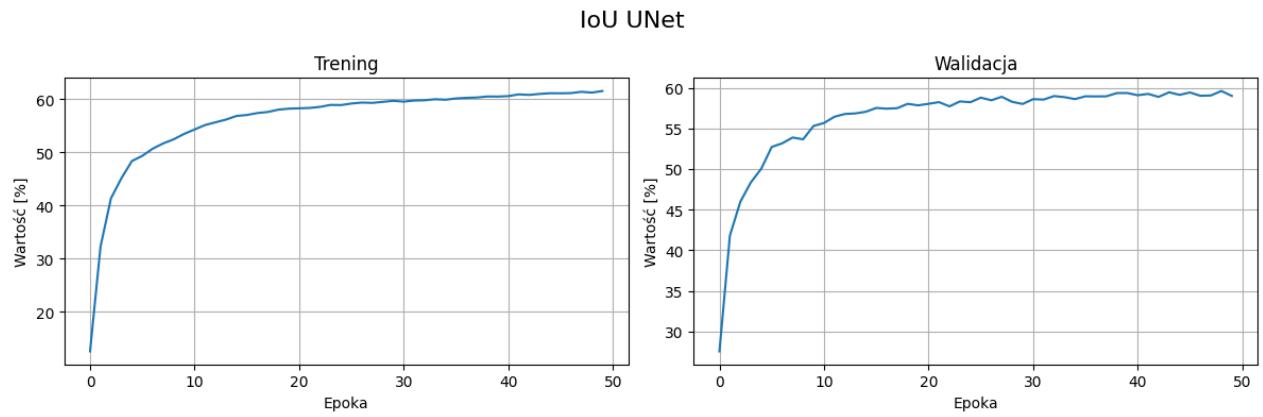
### 4.3.1 Przebieg procesu uczenia

Poniżej przedstawiono wykresy procesów nauki oraz interpretację otrzymanych wyników.

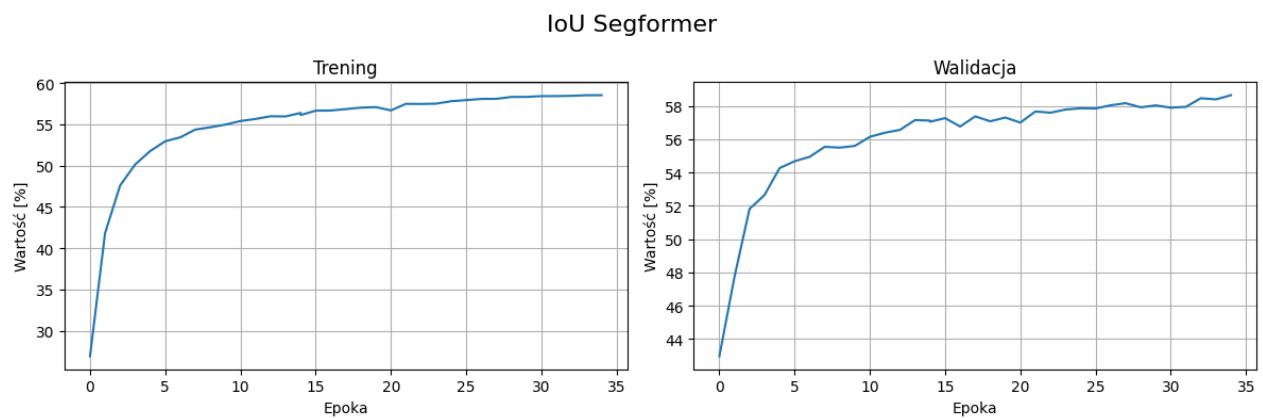
#### Porównanie metryk działania modeli (*IoU* i *mAP*)

Wykresy wartości metryki IoU dla modeli semantycznych (Rysunki 4.11-4.12) wykazują charakterystyczne dla segmentacji nasycenie logarytmiczne. Model U-Net osiąga stabilizację w okolicach 30 epoki na poziomie ok. 60% IoU. SegFormer wykazuje szybszą zbieżność początkową (strome nachylenie krzywej do 10 epoki), co wynika z zastosowania mechanizmu atencji, który szybciej globalizuje kontekst sceny niż operacje splotowe.

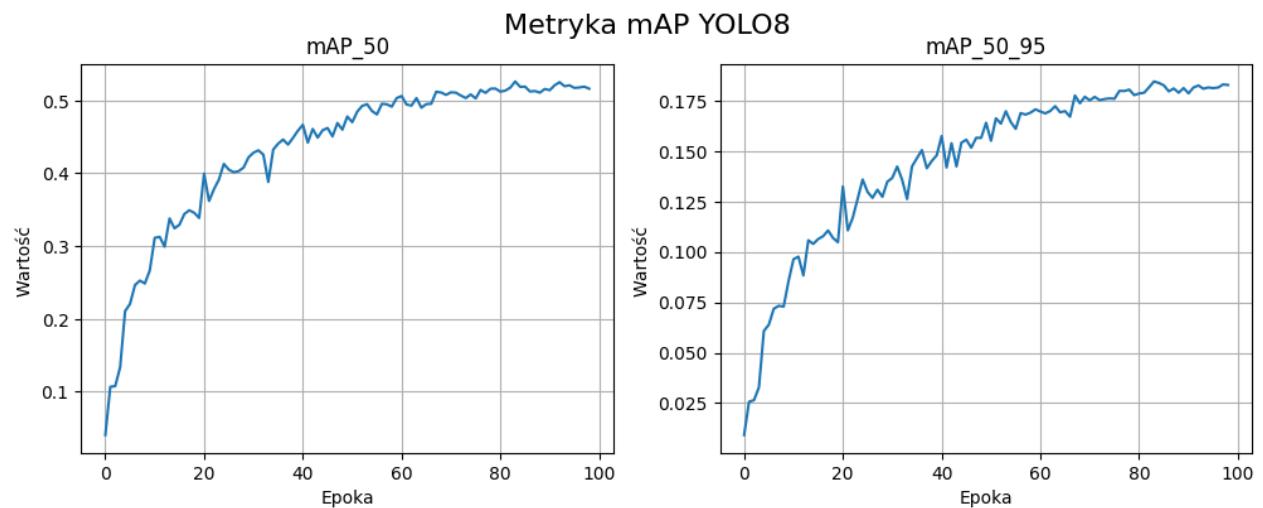
W przypadku modelu YOLOv8 (Rysunek 4.13), metryki mAP50 oraz mAP50-95 rosną wolniej, osiągając stabilizację dopiero w okolicach 80 epoki (trening trwał 100 epok). Jest to typowe zachowanie dla detektorów jednostopniowych, które muszą symultanicznie optymalizować regresję ramki oraz maskę segmentacji.



**Rysunek 4.11:** Wykres funkcji IoU dla modelu U-Net



**Rysunek 4.12:** Wykres funkcji IoU dla modelu SegFormer



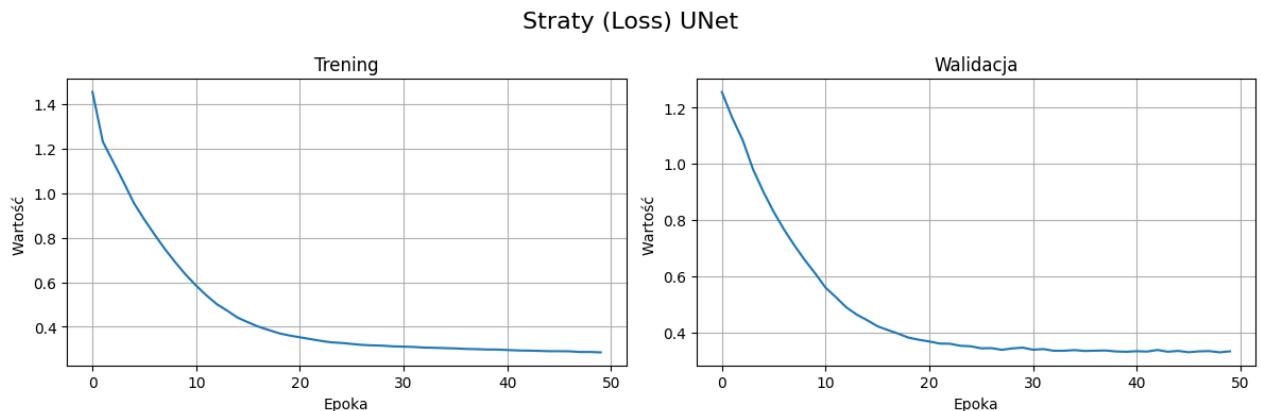
**Rysunek 4.13:** Wykres funkcji mAP dla modelu YOLOv8

### Porównanie stabilności procesu nauki (*loss*)

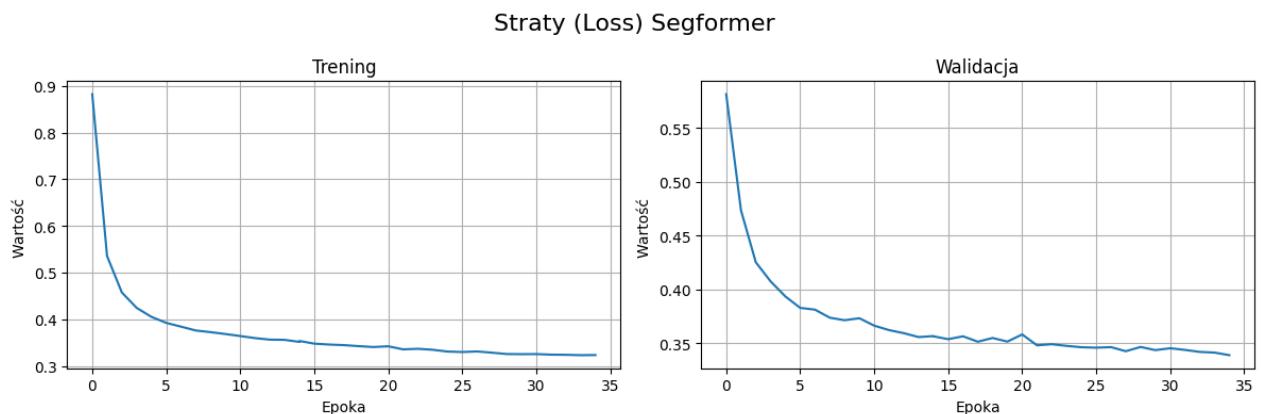
Porównanie przebiegu funkcji kosztu (Loss) uwidacznia fundamentalne różnice w architekturach. Modele U-Net i SegFormer (Rysunki 4.15-4.15) charakteryzują się gładkim, monotonijnym spadkiem wartości straty walidacyjnej, co świadczy o stabilnych gradientach i braku

przeuczenia.

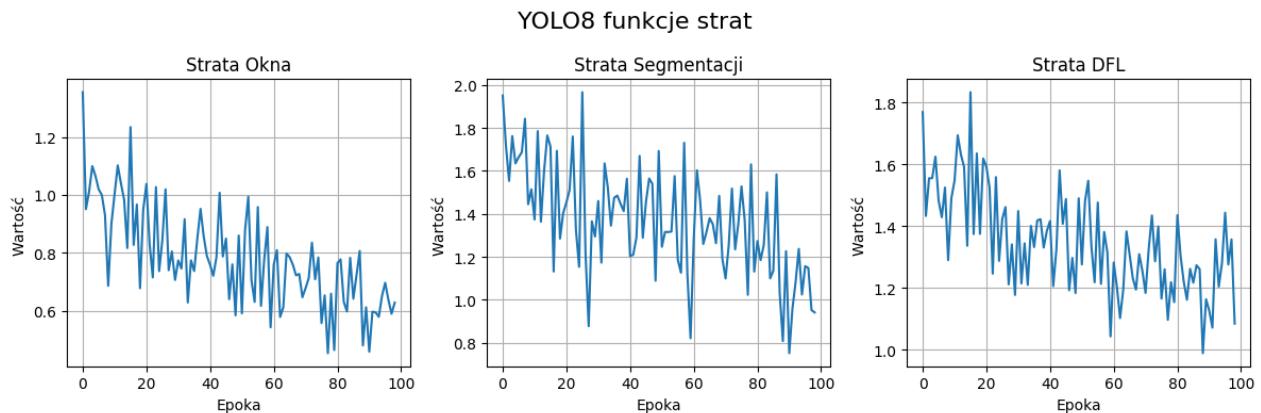
W przeciwnieństwie do nich, funkcje strat składowych modelu YOLOv8 (Box Loss, Seg Loss, DFL Loss – Rysunek 4.16) wykazują duże oscylacje między sąsiednimi epokami. Jest to zjawisko naturalne dla tej architektury, wynikające z mechanizmu anchor-free oraz dynamicznego przydzielania próbek pozytywnych. Mimo lokalnych fluktuacji, globalny trend pozostaje spadkowy, co potwierdza poprawność procesu optymalizacji.



**Rysunek 4.14:** Wykres funkcji strat dla modelu U-Net



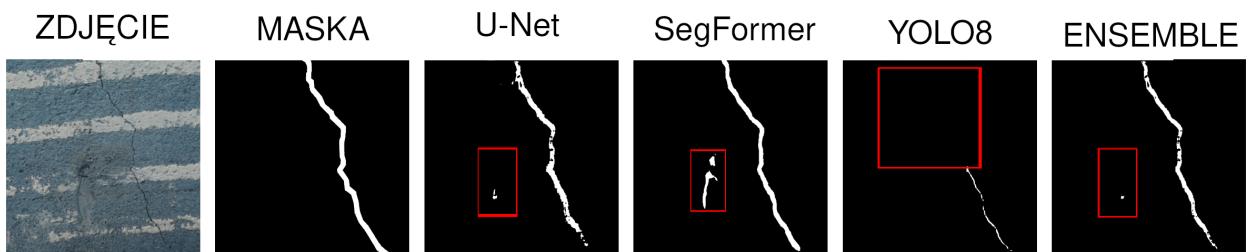
**Rysunek 4.15:** Wykres funkcji strat dla modelu SegFormer



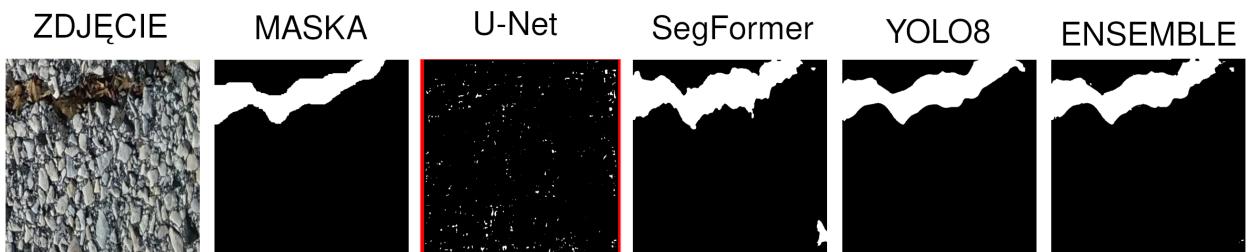
**Rysunek 4.16:** Wykres funkcji strat dla modelu YOLOv8

Przebieg funkcji strat dla modeli semantycznych (U-Net, SegFormer) charakteryzuje się wysoką stabilnością i monotonicznym spadkiem wartości, co świadczy o poprawnej generalizacji bez wyraźnych oznak przeuczenia. Metryka IoU wykazuje nasycenie w okolicach 30 epoki dla modelu U-Net oraz szybszą zbieżność (20-25 epoki) dla architektury SegFormer, co wynika z efektywności mechanizmów atencji. W przypadku modelu YOLOv8 (Rysunek 4.16) funkcje strat składowych (Box, Seg, DFL) wykazują charakterystykę stochastyczną z wysokim poziomem szumu, co jest typowe dla detektorów jednostopniowych trenowanych na mniejszych *batch size*, jednak globalny trend metryk mAP50 oraz mAP50-95 pozostaje logarytmicznie rosnący aż do osiągnięcia plateau w okolicach 60. epoki.

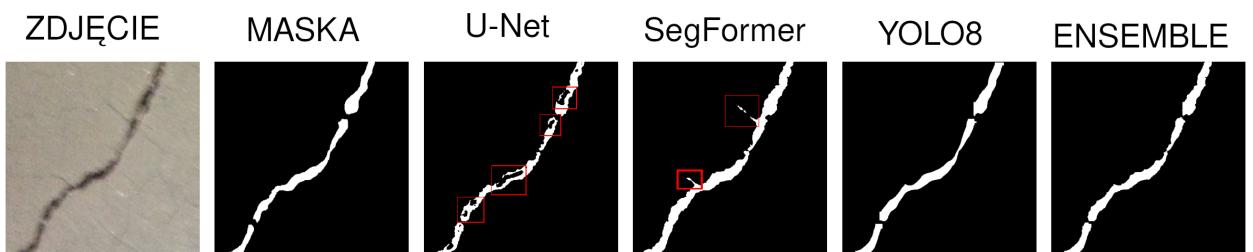
#### 4.3.2 Analiza jakościowa



Rysunek 4.17: Przedstawienie działania modeli segmentacyjnych oraz ich ensemble



Rysunek 4.18: Przedstawienie działania modeli segmentacyjnych oraz ich ensemble



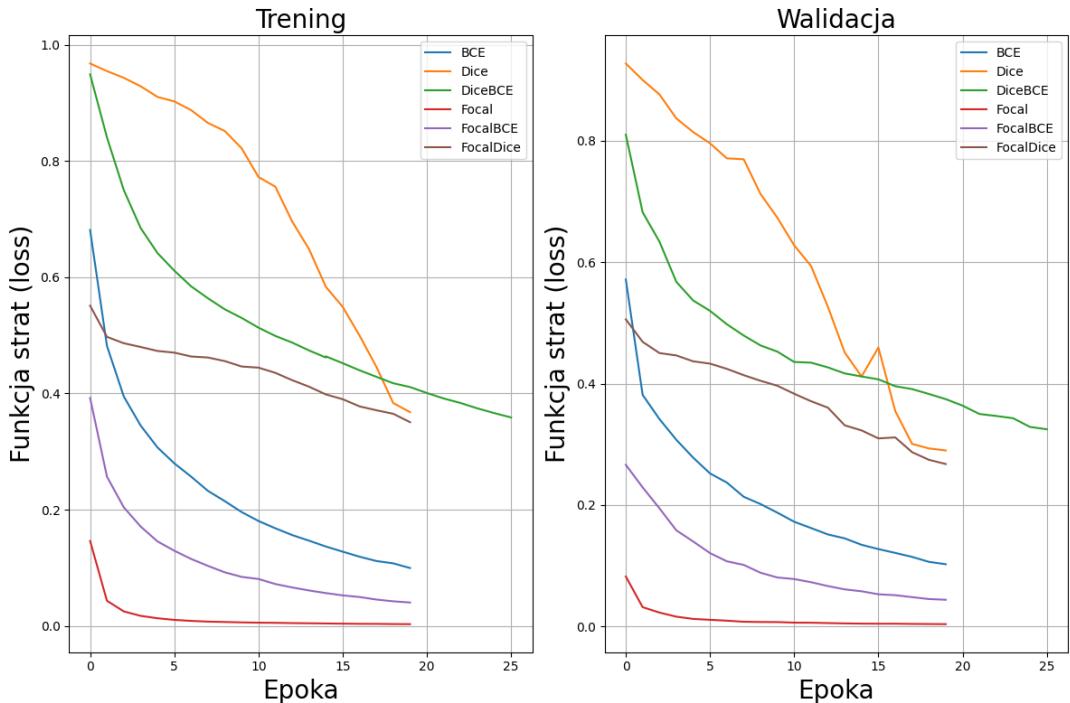
Rysunek 4.19: Przedstawienie działania modeli segmentacyjnych oraz ich ensemble

Model U-Net wykazuje tendencję do fragmentacji pęknięć o niewielkiej szerokości oraz o nieregularnym przebiegu. Jest to wynik ograniczonego pola recepcji operacji splotowych, które analizują obraz lokalnie. Szczególnie jest to widoczne na Rysunku 4.18, gdzie chropowata powierzchnia pęknięć powoduje błędy. W przeciwieństwie do niego, modele SegFormer i YOLOv8, wykorzystujące mechanizmy atencji oraz wieloskalową ekstrakcję cech, dzięki czemu skuteczniej zachowują ciągłość struktur liniowych.

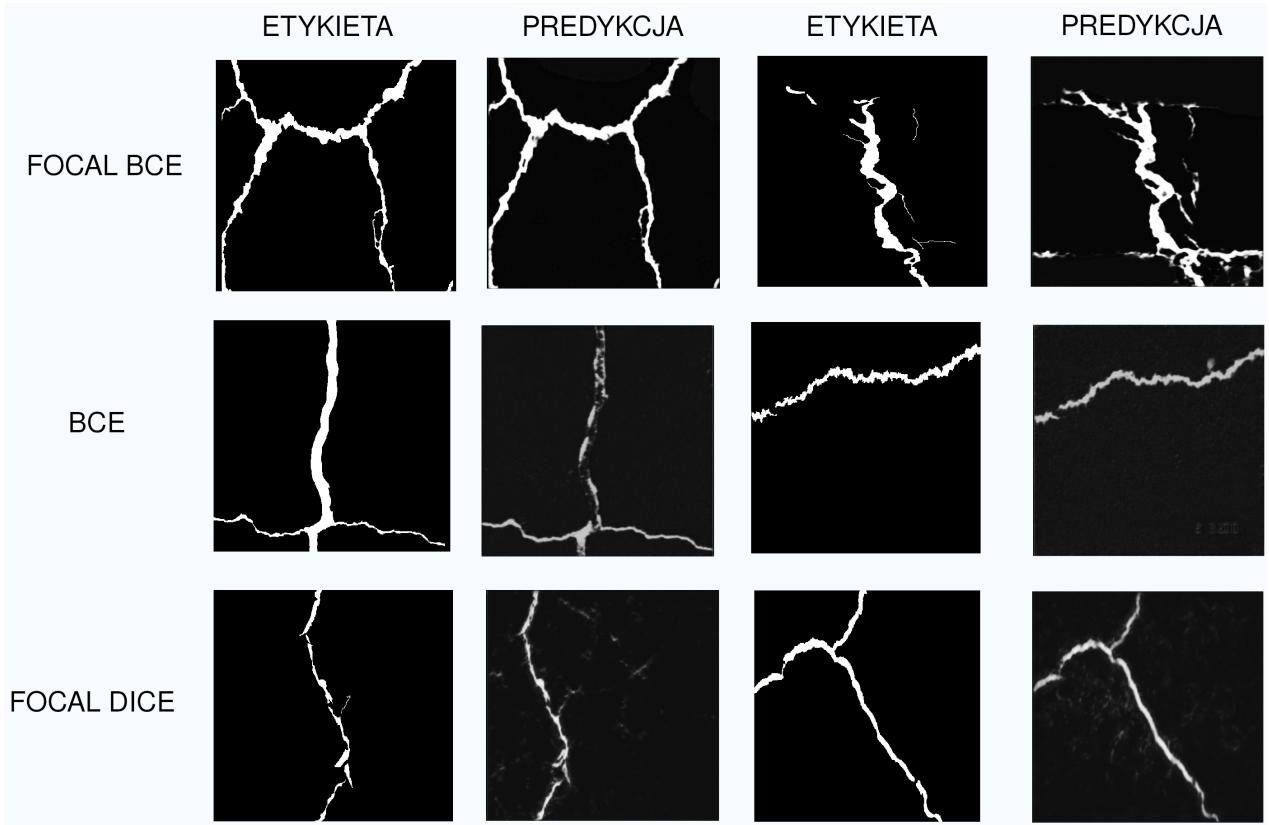
Zastosowanie metody Ensemble opartej na uśrednianiu map prawdopodobieństwa skutkuje widoczną poprawą wyniku. Układ działa eliminując losowe błędy pojedynczych modeli oraz wspólny szum, jednocześnie wzmacniając sygnał w obszarach, gdzie predykcje modeli składowych są zgodne, jak widać na Rysunkach 4.17-4.19.

## 4.4 Analiza wpływu różnych funkcji straty

Celem porównania była weryfikacja wpływu doboru funkcji straty na zbieżność modelu U-Net oraz jakość segmentacji w warunkach silnego niezbalansowania klas, stanowiącego stosunek ilości pikseli pęknień do ilości pikseli tła. Badanie przeprowadzono na zbiorze Deep-Crack przy ograniczonej liczbie epok (20 dla większości funkcji, 25 dla najlepszej konfiguracji), analizując metryki IoU, Dice oraz stabilność procesu uczenia.



Rysunek 4.20: Porównanie funkcji strat modelu U-Net



**Rysunek 4.21:** Porównanie predykcji niektórych funkcji strat modelu U-Net (FocalDice, FocalBCE, Dice)

**Tabela 4.3:** Porównanie różnych funkcji strat dla modelu U-Net trenowanym na zbiorze danych DeepCrack

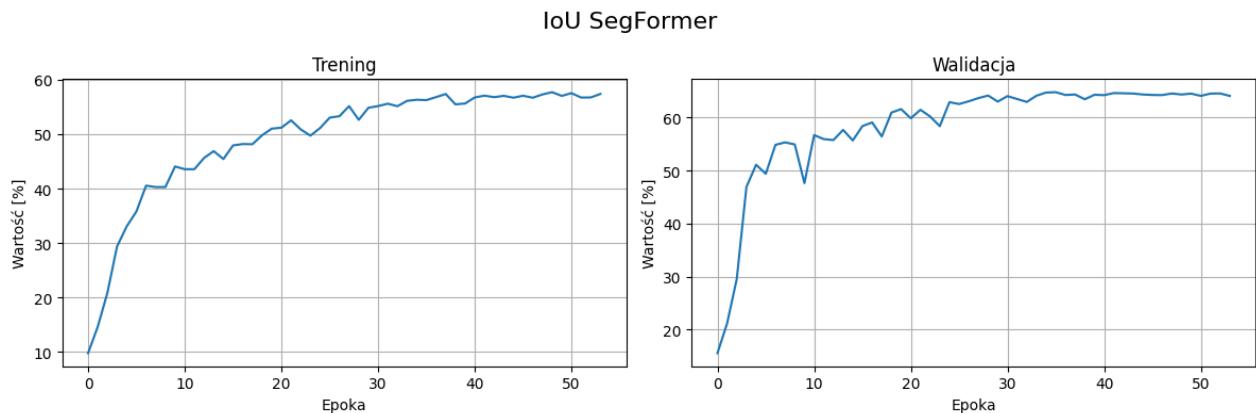
Funkcja straty	Epoki	IoU		Dice		Loss	
		train	val	train	val	train	val
Dice	20	71.03	67.24	82.86	80.35	0.37	0.29
Focal	20	53.32	49.52	71.37	68.45	0.08	0.08
BCE	20	64.80	64.20	77.86	78.47	0.11	0.11
Focal BCE	20	62.27	62.52	76.65	77.62	0.04	0.05
Focal Dice	20	66.71	65.80	80.24	79.25	0.35	0.30
Dice BCE	25	69.72	69.12	81.35	80.32	0.35	0.36

Do finalnego modelu U-Net nie wykorzystano funkcji straty Focal Dice ze względu na problemy w binaryzacji otrzymanego wyniku. Ze względu na występujące szумy oraz ze względu na lepsze parametry IoU, wybrano funkcję Dice BCE.

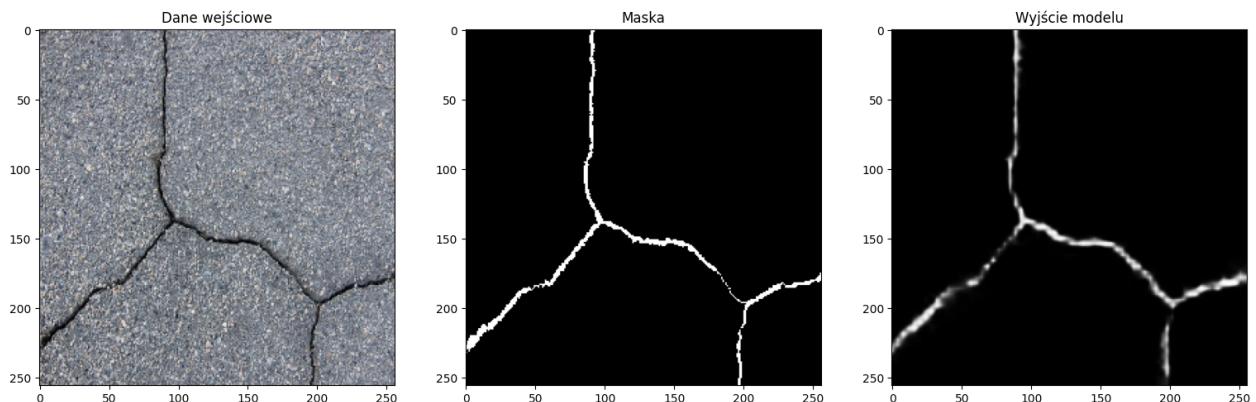
Na podstawie wyników zestawionych w Tabeli 4.3, najwyższą skuteczność osiągnęła hybrydowa funkcja straty DiceBCE, uzyskując  $IoU_{val} = 69.12\%$ . Wynik ten potwierdza założenie, że liniowa kombinacja BCE i Dice pozwala na jednoczesną optymalizację rozkładu prawdopodobieństwa na poziomie pikseli oraz globalnego dopasowania regionów.

## 4.5 Analiza wpływu danych treningowych

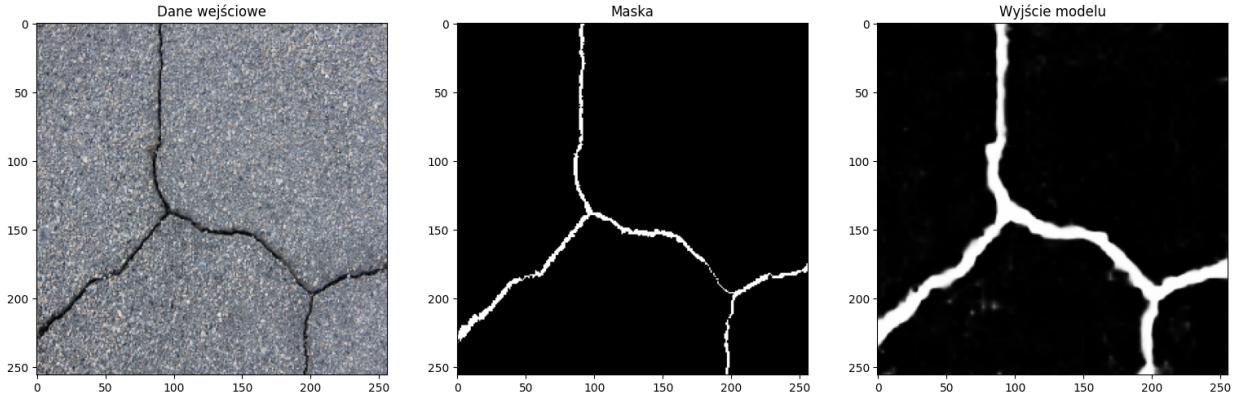
W celu demonstracji wyuczono dodatkowo dwa modele: U-Net oraz SegFormer, używając tylko jednego zbioru danych - DeepCrack, zamiast pełnej puli danych. W celu szybszego procesu nauki zmniejszono obrazy do rozmiaru  $256 \times 256$ , ponieważ zbiór DeepCrack ma wysoką rozdzielcość, nie wpływa drastycznie na pogorszenie "ilości" danych zawartych w zdjęciu oraz masce treningowej i walidacyjnej.



Rysunek 4.22: Funkcja IoU modelu SegFormer uczonego na ograniczonym zbiorze danych



Rysunek 4.23: Wyjście modelu SegFormer trenowanego na całym możliwym zbiorze treningowym



**Rysunek 4.24:** Wyjście modelu SegFormer trenowanego tylko na jednym zbiorze danych treningowych *DeepCrack*

Porównując modele trenowane na mniejszych zestawach danych widać, że zwiększenie danych wpływa na zdolność przewidywania modeli względem danych, które widzą pierwszy raz. Niedokładne wykrywanie pęknięć (zbyt duże wykrycie względem maski) oraz liczne zanieczyszczenia (kropki oddale od pęknięcia) wykazują, że więcej danych treningowych, tym wyższą skuteczność osiągają modele w generalizacji wykryć pomimo, że dla danych walidacyjnych metryka IoU pozostaje większa dla modelu trenowanego na tylko jednym zestawie danych.

Wynika z tego, że im więcej różnych danych, tym model będzie lepiej rozróżniał pęknięcia, więc aby system analizy pęknięć działał jeszcze lepiej, należy zwiększyć ilość danych treningowych.

Eksperyment z ograniczonym zbiorem danych dowodzi, że w przypadku detekcji pęknięć różnorodność tekstur tła jest ważniejsza niż sama liczba próbek. Model uczony na jednorodnym zbiorze DeepCrack generował dużą liczbę fałszywych detekcji (False Positives) na nieznanych fakturach, co dyskwalifikuje go z zastosowań uniwersalnych.

## 4.6 Porównanie z modelami SOTA (*State of the Art*)

W celu weryfikacji skuteczności opracowanego rozwiązania, wyniki modelu Ensemble zestawiono z najnowszymi osiągnięciami w dziedzinie segmentacji pęknięć, opublikowanymi w pracy Chen et al. (2025) [23]. By zestawić modele *SOTA* z modelami wykorzystywanymi w tej pracy, modele dodatkowo wyuczono tylko na zbiorze danych DeepCrack, zamiast na *całym zbiorze danych*. W celu ujednolicenia metryk przeprowadzono test wydajności obliczając metryki występujące w pracy porównawczej (np. mIoU, Precision).

**Tabela 4.4:** Porównanie modeli tej pracy z modelami SOTA trenowanymi na tym samym zbiorze danych DeepCrack. PR - precision, Acc - accuracy

Model	mIoU [%]	Dice [%]	PR [%]	Recall [%]	Acc [%]	$t_p$ [ms]
<b>Modele tej pracy</b>						
<b>SegFormer (MiT-B0)</b>	27.29	26.45	20.30	76.10	72.99	12.92
<b>YOLOv8 (v8m-seg)</b>	32.58	30.92	55.69	27.80	96.24	11.09
<b>U-Net (ResNet34)</b>	36.61	38.04	39.00	58.04	88.17	12.99
<b>Ensemble</b>	38.82	40.77	44.15	55.48	91.13	51.59
<b>Modele porównawcze SOTA</b>						
<b>PspNet</b>	77.80	72.90	61.40	89.70	-	83.10
<b>ResUNet</b>	79.70	76.10	80.90	71.80	-	149.60
<b>SegNet</b>	83.10	80.60	75.00	87.00	-	76.40
<b>Attention UNet</b>	85.40	83.70	78.90	89.10	-	51.90
<b>MSF-CrackNet</b>	87.90	86.80	84.30	89.40	-	31.80

W przeciwieństwie do modeli referencyjnych (SOTA), trenowanych powyżej 100 epok, w badaniach przyjęto limit 30 epok optymalizacyjnych. Decyzja ta podyktowana była analizą zbieżności funkcji strat oraz optymalizacją wykorzystania dostępnych zasobów obliczeniowych (GPU). Choć skrócony trening ograniczył precyzyję segmentacji semantycznej (niższe mIoU wynikające z charakterystyki uczenia dekodera), implementacja metody Ensemble pozwoliła na zredukowanie wariancji błędu i osiągnięcie wyników przewyższających skuteczność pojedynczych architektur.

Model MSF-CrackNet jest architekturą dedykowaną, wyposażoną w mechanizmy zaprojektowane stricte pod morfologię pęknięć, co przedstawiają wyniki z Tabeli 4.4. Opracowany model Ensemble bazuje na fuzji komplementarnych architektur segmentacyjnych.

Podczas gdy na ograniczonym zbiorze DeepCrack model osiąga niższy wskaźnik mIoU 38.00% w porównaniu do wyspecjalizowanych metod SOTA, jego skuteczność znacznie wzrasta po wytrenowaniu na *całym zbiorze danych*. Wskazuje to na wysoką zdolność architektury do generalizacji wiedzy zróżnicowanych danych i lepszą adaptację do zróżnicowanych warunków oświetleniowych i powierzchniowych przy równoczesnym zachowaniu czasów działania  $t_p$ .

## 4.7 Analiza wydajnościowa i zasobowa

Pojedyncze modele wykazują zbliżone zapotrzebowanie na pamięć karty graficznej, mieszczące się w przedziale 142–191 MB. Zastosowanie metody Ensemble skutkuje nieliniowym wzrostem zapotrzebowania na pamięć do poziomu 670.44 MB. Jest to wartość ponad 3.5-krotnie wyższa od najbardziej wymagającego modelu pojedynczego YOLOv8. Wynika to z potrzeby przechowywania wielu modeli w pamięci w tym samym czasie.

Model YOLOv8 osiągnął najkrótszy czas predykcji  $t_p = 11.09$  ms, co pozwala na przetwarzanie obrazu z prędkością ponad 90 FPS. Jest to wynik lepszy zarówno od modeli U-Net jak i SegFormer. Układ Ensemble, ze względu na złożoność obliczeniową, osiąga jedynie 19.38 FPS.

Ze względu na poprawę jakości oraz komplementarne działanie modeli, Ensemble osiąga najwyższy wskaźnik IoU (68.21%), co przewyższa każdy pojedynczy model. Dla takiego

rozwiązań wolniejszy czas przetwarzania jest akceptowalny.

**Tabela 4.5:** Porównanie działania modeli segmentacyjnych, gdzie  $t_p$  oznacza czas predykcji pojedynczej próbki danych, a FPS liczbę klatek na sekundę. Przetestowane na całym testowym zbiorze danych.

Zasoby	PARAMETR	U-Net	SEGFORMER	YOLOv8	ENSEMBLE
CUDA	<b>VRAM [MB]</b>	142.23	157.81	190.508	670.44
	$t_p$ [ms]	12.99	12.92	11.09	51.59
	<b>FPS [1/s]</b>	76.98	77.40	90.17	19.38
	<b>IoU [%]</b>	59.61	58.64	64.10	68.21

## 4.8 Porównanie systemu analizy

W literaturze przedmiotu dominuje podział na metody oparte wyłącznie na segmentacji semantycznej, detekcji obiektowej oraz klasycznych algorytmach przetwarzania obrazu. W pracach takich jak *DeepCrack* [1] czy *CrackFormer* [2], detekcja jest realizowana poprzez binarną segmentację pęknięcia od tła. Takie realizacje pozwalają na precyzyjną lokalizację uszkodzenia na poziomie piksela, generując maskę prawdopodobieństwa. Systemy te różnią się jednak fundamentalnie architekturą: w przypadku *DeepCrack* zastosowano model oparty na hierarchicznym uczeniu cech, który łączy mapy cech z różnych skal w strukturze koder-dekoder. Z kolei dla *CrackFormer* zaproponowano architekturę opartą na mechanizmie atencji, która skuteczniej modeluje długozasięgowe zależności, co pozwala na lepszą detekcję drobnych, nieciągłych pęknięć w złożonym tle.

W przypadku analizy obrazu w czasie rzeczywistym, priorytetem staje się czas inferencji. W pracach takich jak *An Improved Lightweight YOLOv8-Seg for Real-Time Pixel-Level Crack Detection of Dams and Bridges* [24], wykorzystuje się lekkie mechanizmy uwagi w celu przyspieszenia segmentacji oparte na specjalistycznych modelach YOLO. Tutaj również można wykorzystać różne modele, np. RT-DETR (Real-Time DEtection TRansformer) (praca *Road crack detection based on improved RT-DETR* [25], którego wykrywanie polega na identyfikacji instancji pęknięcia i oznaczeniu jej ramką graniczną (bounding box). W pracach porównawczych wykazano, że choć podejście to jest wysoce wydajne obliczeniowo, traci ono kluczową informację o geometrii pęknięcia. Ramka graniczna (bounding box) dla pęknięcia ukośnego znaczaco przeszacowuje jego powierzchnię i uniemożliwia dokładny pomiar szerokości rozwarcia.

W niniejszej pracy zaproponowano podejście hybrydowe, łączące zalety powyższych metod. Stanowi to rozwinięcie koncepcji przedstawionej w pracy *Automated Multi-Class Concrete Crack Detection and Severity Classification Using CNN-Based Deep Learning* [26], gdzie wykorzystano sieci CNN wyłącznie do klasyfikacji obrazów na 7 rozłącznych kategorii (w tym różne szerokości pęknięć oraz korozję), ale bez ich lokalizacji przestrzennej.

Analiza geometryczna pęknięć w starszych rozwiązaniach często realizowana jest za pomocą klasycznych algorytmów przetwarzania obrazów. W pracy *Crack Detection and Analysis of Concrete Structures Based on Neural Network and Clustering* [27] zrealizowano to poprzez binaryzację oraz filtrację w celu usunięcia szumu. Wykorzystuje również szkieletyzację (*skeletonization*) oraz analizę ortogonalną do wyznaczenia szerokości pęknięcia wzduż jego przebiegu.

#### 4.8.1 Porównanie z nowoczesnymi metodami *Instance Segmentation*

Współczesne modele SOTA, takie jak YOLOv11[28] oferują zintegrowane podejście *Instance Segmentation*[29], realizując w jednym przebiegu zadania, które w niniejszej pracy wykonuje wieloetapowy potok detekcji, segmentacji oraz klasyfikacji.

Niewątpliwą przewagą YOLOv11 jest czas inferencji oraz krótszy proces treningowy, redukując go do optymalizacji jednej sieci neuronowej. System oparty na pojedynczym modelu działa w czasie rzeczywistym, podczas gdy proponowany system Ensemble posiada sumaryczny narzut czasowy wynikający z sekwencyjnego przetwarzania przez detektor domeny, trzy modele segmentacyjne oraz dwa modele klasyfikacji.

Porównując specyfikę działania modelu YOLOv11 oraz YOLOv8 należy jednak zauważać, że mimo optymalizacji architektury w nowszej wersji i poprawy metryki mAP[30], oba modele dzielą ten sam paradygmat przetwarzania obrazu, który stanowi ograniczenie w analizie małych uszkodzeń.

Modele klasy YOLO (v8, v11, v12) są optymalizowane pod kątem szybkości i zazwyczaj operują na zredukowanym rozdzielczościowo obrazie wejściowym (standardowo  $640 \times 640$  pikseli). W przypadku pęknięć włosowatych, które na oryginalnym zdjęciu wysokiej rozdzielczości mają szerokość 1-3 pikseli, proces skalowania w dół powoduje ich zatarcie lub całkowity zanik. Architektura U-Net, zastosowana w niniejszym systemie pozwala na precyzyjną rekonstrukcję krawędzi, której modele YOLO mogłyby nie zapewnić (Rysunek 4.17).

Choć YOLOv11 posiada głowicę klasyfikacyjną, decyzja o klasie podejmowana jest na podstawie cech wyekstrahowanych z pomniejszonego obrazu. W proponowanym systemie po wykryciu pęknięcia, system wycina jego fragment w oryginalnej rozdzielczości i przekazuje go do dedykowanych klasyfikatorów. Pozwala to na rozróżnienie subtelnych różnic w szerokości pęknięcia, które dla modelu YOLO mogłyby być nierozróżnialne.

### 4.9 Wnioski

Wprowadzenie wstępnej klasyfikacji binarnej (Kontroler Domeny) skutecznie eliminuje przetwarzanie obrazów niezawierających uszkodzeń. Przy minimalnym narzucie czasowym, moduł ten oszczędza zasoby obliczeniowe, nie generując znaczącego ryzyka błędu *False Negative* ( $FN=2$  dla zbioru testowego).

Analiza przebiegu procesu uczenia modeli klasyfikacyjnych (Rysunki 4.4, 4.5) wykazała tendencję do występowania zjawiska przeuczenia (*overfitting*). Obserwowane oscylacje funkcji straty dla zbioru walidacyjnego, przy jednoczesnej minimalizacji błędu treningowego, wskazują na ograniczoną zdolność generalizacji. W celu stabilizacji procesu uczenia i redukcji wariancji walidacyjnej, rekommendowane jest zwiększenie liczebności zbioru danych treningowych lub zastosowanie jeszcze bardziej agresywnych technik augmentacji danych.

Dla silnie niebalansowanych zbiorów danych (stosunek tła do pęknięcia), hybrydowa funkcja straty DiceBCE okazała się najskuteczniejsza, osiągając wyższe wartości metryki IoU w procesie walidacji w porównaniu do standardowych funkcji Focal Loss czy czystego Dice Loss.

Eksperymenty wykazały, że modele trenowane wyłącznie na jednorodnym zbiorze DeepCrack tracą zdolność generalizacji, generując liczne błędy False Positive na odmiennych

fakturach tła. Zwiększenie różnorodności danych treningowych jest kluczowe dla poprawności działania systemu w warunkach rzeczywistych.

Modele wykorzystujące mechanizm atencji oraz wieloskalową ekstrakcję cech wykazały wyższą zdolność do zachowania ciągłości cienkich struktur liniowych w porównaniu do klasycznej architektury splotowej U-Net, wykazuje tendencję do przerywania ciągłości pęknień o nieregularnym przebiegu.

W zadaniu klasyfikacji powagi pęknień model ConvNeXt osiągnął najwyższą dokładność, z kolei model EfficientNet-B0 okazał się optymalnym rozwiązaniem dla środowisk o ograniczonych zasobach, oferując trzykrotnie krótszy czas inferencji przy akceptowalnym spadku dokładności.

Analiza porównawcza z architekturami jednostopniowymi (typu YOLO) wykazała, że dominujący w nich paradygmat skalowania obrazu wejściowego stanowi istotną barierę w detekcji małych uszkodzeń. Wykazano, że podejście oparte na segmentacji w pełnej rozdzielcości (U-Net, SegFormer) jest niezbędne do zachowania ciągłości topologicznej pęknień włosowatych (o szerokości 1–3 pikseli), które ulegają zatarciu w procesie kompresji map cech typowym dla szybkich detektorów. Oznacza to, że precyza przestrzenna oferowana przez proponowany system wieloetapowy jest parametrem nadzorowanym względem czasu inferencji oferowanego przez modele Real-Time.

Potwierdzono również zasadność zastosowania podejścia Ensemble łączącego odmienne metody przetwarzania. Połączenie lokalnej precyzji operacji splotowych z globalnym modelowaniem kontekstu przez mechanizm atencji (Transformer) pozwoliła na redukcję błędów systematycznych pojedynczych modeli. Dodatkowo, rozdzielenie etapu detekcji od klasyfikacji umożliwiło ocenę powagi uszkodzenia na podstawie analizy tekstury w oryginalnej rozdzielcości radiometrycznej, co pozwoliło na precyzyjne rozróżnienie klas granicznych, nieosiągalne dla głowic klasyfikacyjnych operujących na zredukowanych cechach.

## 4.10 Podsumowanie

Celem niniejszej pracy była realizacja i weryfikacja systemu analizy pęknień powierzchni budowlanych, integrującego metody głębokiego uczenia z klasyczną analizą obrazu. Cel ten został osiągnięty poprzez zaprojektowanie i implementację wieloetapowego potoku przetwarzania, składającego się z: modułu filtracji wstępnej, opartej na autorskiej architekturze CNN, który z wysoką skutecznością odróżnia obrazy uszkodzone od tła, modułu segmentacji, wykorzystującego łączenie predykcji architektur U-Net, SegFormer oraz YOLOv8-seg, co pozwoliło na precyzyjną lokalizację pęknień na poziomie piksela oraz finalnie modułu klasyfikacji, szacującego stopień powagi pęknienia powierzchni w oparciu o modele ConvNeXt i EfficientNet oraz modułu analizy geometrycznej, wykorzystującej algorytmy szkieletyzacji i transformaty odległościowej do wyznaczenia parametrów fizycznych pęknienia.

Przeprowadzona analiza na zbiorze testowym potwierdziła skuteczność zaproponowanego rozwiązania. System klasyfikacyjny osiągnął dokładność przekraczającą 95%, a zastosowanie uczenia zespołowego w segmentacji pozwoliło na uzyskanie wyników IoU przewyższających osiągi pojedynczych modeli składowych.

Zrealizowany interfejs użytkownika w architekturze klient-serwer umożliwia praktyczne

wykorzystanie opracowanych algorytmów, prezentując wyniki detekcji oraz parametry geometryczne w czasie zbliżonym do rzeczywistego (przy wykorzystaniu serwera z odpowiednią kartą graficzną obsługującą *CUDA*).

Głównym ograniczeniem systemu pozostaje zależność jakości segmentacji od różnorodności danych treningowych oraz czas przetwarzania w trybie Ensemble. Dalsze prace rozwojowe powinny koncentrować się na rozbudowie bazy danych o przykłady zawierające zakłócenia (cienie, zabrudzenia) oraz optymalizacji kodu pod kątem inferencji na urządzeniach brzegowych.

# Bibliografia

- [1] Yahui Liu, Jian Yao, Xiaohu Lu, Renping Xie, and Li Li. DeepCrack: A deep hierarchical feature learning architecture for crack segmentation. *Neurocomputing*, 338:139–153, April 2019. Dostęp: 2026-01-04. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231219300566>, doi:10.1016/j.neucom.2019.01.036.
- [2] Huajun Liu, Xiangyu Miao, Christoph Mertz, Chengzhong Xu, and Hui Kong. CrackFormer: Transformer Network for Fine-Grained Crack Detection. pages 3783–3792, 2021. Dostęp: 2026-01-04. URL: [https://openaccess.thecvf.com/content/ICCV2021/html/Liu\\_CrackFormer\\_Transformer\\_Network\\_for\\_Fine-Grained\\_Crack\\_Detection\\_ICCV\\_2021\\_paper.html](https://openaccess.thecvf.com/content/ICCV2021/html/Liu_CrackFormer_Transformer_Network_for_Fine-Grained_Crack_Detection_ICCV_2021_paper.html).
- [3] Christian Hary and Satria Mandala. Object Detection Analysis Study in Images based on Deep Learning Algorithm. In *2022 International Conference on Data Science and Its Applications (ICoDSA)*, pages 226–231, July 2022. Dostęp: 2026-01-04. URL: <https://ieeexplore.ieee.org/document/9862922>, doi:10.1109/ICoDSA55874.2022.9862922.
- [4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/1505.04597>, doi:10.48550/arXiv.1505.04597.
- [5] Vidya Prasad, Chen Zhu-Tian, Anna Vilanova, Hanspeter Pfister, Nicola Pezzotti, and Hendrik Strobelt. Unraveling the Temporal Dynamics of the Unet in Diffusion Models, December 2023. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/2312.14965>, doi:10.48550/arXiv.2312.14965.
- [6] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU), February 2019. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/1803.08375>, doi:10.48550/arXiv.1803.08375.
- [7] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers, October 2021. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/2105.15203>, doi:10.48550/arXiv.2105.15203.
- [8] Yash Raj Suman. SegFormer Tutorial: Master Semantic Segmentation Fast, May 2025. Dostęp: 2026-01-04. URL: <https://www.labellerr.com/blog/segformer/>.
- [9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection, May 2016. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/1506.02640>, doi:10.48550/arXiv.1506.02640.
- [10] Ultralytics. Ultralytics YOLOv8, December 2023. Dostęp: 2026-01-04. URL: <https://docs.ultralytics.com/models/yolov8/>.

- [11] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, September 2020. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/1905.11946>, doi:10.48550/arXiv.1905.11946.
- [12] Efficientnet Architecture. Dostęp: 2026-01-04. URL: <https://www.geeksforgeeks.org/computer-vision/efficientnet-architecture/>.
- [13] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s, March 2022. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/2201.03545>, doi:10.48550/arXiv.2201.03545.
- [14] Atakan Erdogan. ConvNeXt — Next Generation of Convolutional Networks, April 2023. Dostęp: 2026-01-04. URL: <https://medium.com/@atakanerdogan305/convnext-next-generation-of-convolutional-networks-325607a08c46>.
- [15] Yiyi Zhang, Alu Xu, Danquan Lan, Xingtuo Zhang, Jie Yin, and Hui Hwang Goh. *ScienceDirect ConvNeXt-based anchor-free object detection model for infrared image of power equipment*. PhD thesis, December 2022. Dostęp: 2026-01-04.
- [16] Adil H. Khan, Dayang NurFatimah Awang Iskandar, Jawad F. Al-Asad, Hiren Mewada, and Muhammad Abid Sherazi. Ensemble learning of deep learning and traditional machine learning approaches for skin lesion segmentation and classification. *Concurrency and Computation: Practice and Experience*, 34(13):e6907, 2022. Dostęp: 2026-01-04. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6907>, doi:10.1002/cpe.6907.
- [17] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/1409.1556>, doi:10.48550/arXiv.1409.1556.
- [18] Yahui Liu, Jian Yao, Xiaohu Lu, Renping Xie, and Li Li. DeepCrack: A Deep Hierarchical Feature Learning Architecture for Crack Segmentation. *Neurocomputing*, 338:139–153, 2019. Dostęp: 2026-01-04. doi:10.1016/j.neucom.2019.01.036.
- [19] Çağlar Fırat Özgenel. Concrete Crack Images for Classification, 2019. Dostęp: 2026-01-04. URL: <https://data.mendeley.com/datasets/5y9wdsg2zt/1>.
- [20] Lakshay Middha. Crack Segmentation Dataset, 2021. Dostęp: 2026-01-04. URL: <https://www.kaggle.com/datasets/lakshaymiddha/crack-segmentation-dataset>.
- [21] Jakub Niemiec. Concrete Crack Images for Classification, 2019. Dostęp: 2026-01-04. URL: <https://www.kaggle.com/datasets/jakubniemiec/concrete-crack-images>.
- [22] Umang J. Tiny ImageNet 200, 2021. Dostęp: 2026-01-04. URL: <https://www.kaggle.com/datasets/umangjjw/tinyimagenet200>.
- [23] Yang Chen, Tao Yang, Shuai Dong, Like Wang, Bida Pei, and Yunlong Wang. Enhancing Crack Segmentation Network with Multiple Selective Fusion Mechanisms. *Buildings*, 15:1088, March 2025. Dostęp: 2026-01-04. doi:10.3390/buildings15071088.
- [24] Yang Wu, Qingbang Han, Qilin Jin, Jian Li, and Yujing Zhang. LCA-YOLOv8-Seg: An Improved Lightweight YOLOv8-Seg for Real-Time Pixel-Level Crack Detection of Dams and Bridges. *Applied Sciences*, 13(19):10583, January 2023. Dostęp: 2026-01-04. URL: <https://www.mdpi.com/2076-3417/13/19/10583>, doi:10.3390/app131910583.

- [25] Guangyuan Zhao, Weilin Zhang, Rui Sun, and Tong Wei. Road crack detection based on improved RT-DETR. *Signal, Image and Video Processing*, 19(7):583, May 2025. Dostęp: 2026-01-04. [doi:10.1007/s11760-025-04165-w](https://doi.org/10.1007/s11760-025-04165-w).
- [26] Wisam Bukaita, Kalyan Vankudothu, and Junaid Khan. Automated Multi-Class Concrete Crack Detection and Severity Classification Using CNN-Based Deep Learning. *American Journal of Civil Engineering*, 13:197–210, July 2025. Dostęp: 2026-01-04. [doi:10.11648/j.ajce.20251304.12](https://doi.org/10.11648/j.ajce.20251304.12).
- [27] Young Choi, Hee Won Park, Yirong Mi, and Sujeen Song. Crack Detection and Analysis of Concrete Structures Based on Neural Network and Clustering. *Sensors*, 24(6):1725, January 2024. Dostęp: 2026-01-04. URL: <https://www.mdpi.com/1424-8220/24/6/1725>, [doi:10.3390/s24061725](https://doi.org/10.3390/s24061725).
- [28] Ultralytics. Ultralytics YOLO11. Dostęp: 2026-01-04. URL: <https://docs.ultralytics.com/models/yolo11/>.
- [29] Ultralytics. Instance Segmentation. Dostęp: 2026-01-04. URL: <https://docs.ultralytics.com/tasks/segment/>.
- [30] Ranjan Sapkota and Manoj Karkee. Comparing YOLOv11 and YOLOv8 for instance segmentation of occluded and non-occluded immature green fruits in complex orchard environment, January 2025. Dostęp: 2026-01-04. URL: <http://arxiv.org/abs/2410.19869>, [doi:10.48550/arXiv.2410.19869](https://doi.org/10.48550/arXiv.2410.19869).
- [31] Mateusz Szcześniak. System analizy pęknięć powierzchni budowlanych na podstawie analizy obrazów, 2025. Dostęp: 2026-01-04. URL: <https://github.com/mszczesniak02/bachelor>.

# Spis ilustracji

1.1	Architektura sieci U-Net . . . . .	8
1.2	Architektura SegFormer składa się z dwóch głównych modułów: hierarchicznego encodera typu Transformer, służącego do ekstrakcji cech zgrubnych (coarse) i szczegółowych (fine), oraz lekkiego dekodera opartego wyłącznie na warstwach MLP (All-MLP decoder), którego zadaniem jest bezpośrednia fuzja tych wielopoziomowych cech i predykcja maski segmentacji semantycznej. Skrót „FFN” oznacza sieć typu feed-forward (sieć jednokierunkową)[7, Fig. 2] . . . . .	9
1.3	Architektura. Nasza sieć detekcyjna składa się z 24 warstw splotowych, po których następują 2 warstwy w pełni połączone (fully connected layers). Naprzemienne stosowane warstwy splotowe $1 \times 1$ redukują przestrzeń cech z warstw poprzednich. Warstwy splotowe poddajemy wstępemu trenowaniu (pretraining) na zadaniu klasyfikacji ImageNet przy połowie docelowej rozdzielczości (obraz wejściowy $224 \times 224$ ), a następnie podwajamy rozdzielcość na potrzeby detekcji [9, Fig. 3] . . . . .	10
1.4	Architektura modelu EfficientNet-b0 . . . . .	11
1.5	Architektura modelu ConvNeXt-Tiny [15] . . . . .	12
2.1	Schemat ideowy działania programu . . . . .	14
2.2	Schemat ideowy uczenia zespołowego dla sieci segmentacyjnej. Realizacja poprzez uśrednienie predykcji każdego piksela zwróconego z wyjść modeli segmentacyjnych . . . . .	14
2.3	Schemat ideowy uczenia zespołowego dla sieci klasyfikującej. Realizacja poprzez uśrednienie prawdopodobieństw klas otrzymanych z modeli klasifyjnych . . . . .	14
2.4	Schemat architektury Kontrolera Domeny. . . . .	16
2.5	Losowo wybrane zdjęcia z całego zestawu danych treningowych . . . . .	18
2.6	Maski zdjęć z powyższych danych treningowych . . . . .	18
2.7	Losowo wybrane próbki danych zbioru Tiny-Image-200. . . . .	19
2.8	Przedstawienie losowo wybranych danych treningowych dla modeli klasifyjnych . . . . .	20
2.9	Histogram danych treningowych - widoczne niezbalansowanie klas . . . . .	20
2.10	Histogram danych treningowych po augmentacji danych . . . . .	20
2.11	Interfejs użytkownika - widok powitalny z opcją dodania zdjęcia do analizy . .	22
2.12	Interfejs użytkownika - przykład przeanalizowanego zdjęcia z wykrytą, pęknietą powierzchnią . . . . .	22
2.13	Interfejs użytkownika - brak wykrycia pęknienia oraz brak dalszej analizy . . .	22
2.14	Interfejs użytkownika - przedstawienie części wyników analizy pęknienia . . . .	23
4.1	Macierz pomyłek kontrolera domeny dla danych testowych . . . . .	28
4.2	Wykres dokładności sieci ConvNeXt trwającej 30 epok . . . . .	29
4.3	Wykres dokładności sieci EfficientNet trwającej 30 epok . . . . .	30

4.4	Wykres funkcji strat sieci ConvNeXt trwającej 30 epok . . . . .	30
4.5	Wykres strat sieci EfficientNet trwającej 30 epok . . . . .	30
4.6	Wykres czułości sieci ConvNeXt trwającej 30 epok . . . . .	31
4.7	Wykres czułości sieci EfficientNet trwającej 30 epok . . . . .	31
4.8	Wykres precyzji modelu ConvNeXt . . . . .	32
4.9	Wykres precyzji sieci EfficientNet trwającej 30 epok . . . . .	32
4.10	Tabele pomyłek kolejno ConvNeXt, EfficientNet oraz Ensemble obu modeli . . . . .	33
4.11	Wykres funkcji IoU dla modelu U-Net . . . . .	35
4.12	Wykres funkcji IoU dla modelu SegFormer . . . . .	35
4.13	Wykres funkcji mAP dla modelu YOLOv8 . . . . .	35
4.14	Wykres funkcji strat dla modelu U-Net . . . . .	36
4.15	Wykres funkcji strat dla modelu SegFormer . . . . .	36
4.16	Wykres funkcji strat dla modelu YOLOv8 . . . . .	36
4.17	Przedstawienie działania modeli segmentacyjnych oraz ich ensemble . . . . .	37
4.18	Przedstawienie działania modeli segmentacyjnych oraz ich ensemble . . . . .	37
4.19	Przedstawienie działania modeli segmentacyjnych oraz ich ensemble . . . . .	37
4.20	Porównanie funkcji strat modelu U-Net . . . . .	38
4.21	Porównanie predykcji niektórych funkcji strat modelu U-Net (FocalDice, FocalBCE, Dice) . . . . .	39
4.22	Funkcja IoU modelu SegFormer uczonego na ograniczonym zbiorze danych . . . . .	40
4.23	Wyjście modelu SegFormer trenowanego na całym możliwym zbiorze treningowym . . . . .	40
4.24	Wyjście modelu SegFormer trenowanego tylko na jednym zbiorze danych treningowych <i>DeepCrack</i> . . . . .	41

## Spis tabel

1.1	Tablica pomyłek dla klasyfikacji binarnej . . . . .	12
2.1	Przedstawienie klas pęknień względem ich zakresów szerokości . . . . .	20
3.1	Zestawienie parametrów modeli wykorzystanych w projekcie. LR - współczynnik nauki ( <i>learning rate</i> ) . . . . .	25
4.1	Zestawienie metryk skuteczności (F1, Accuracy, Recall, Precision) dla badanych architektur na zbiorze testowym. . . . .	33
4.2	Porównanie działania modeli klasyfikacyjnych z uwzględnieniem parametru FPS. . . . .	34
4.3	Porównanie różnych funkcji strat dla modelu U-Net trenowanym na zbiorze danych DeepCrack . . . . .	39

4.4	Porównanie modeli tej pracy z modelami SOTA trenowanymi na tym samym zbiorze danych DeepCrack. PR - precision, Acc - accuracy . . . . .	42
4.5	Porównanie działania modeli segmentacyjnych, gdzie $t_p$ oznacza czas predykcji pojedynczej próbki danych, a FPS liczbę klatek na sekundę. Przetestowane na całym testowym zbiorze danych. . . . .	43

# Dodatek A

## Konfiguracja środowiska

W celu konfiguracji stanowiska należy pobrać kod z załącznika do platformy *GitHub* [31] oraz postępować zgodnie z instrukcją opisaną w pliku powitalnym repozytorium (`readme.md`).