

Politechnika Wrocławskas

Wydział Elektroniki, Fotoniki i Mikrosystemów

KIERUNEK: Automatyka i Robotyka (AIR)

PRACA DYPLOMOWA INŻYNIERSKA

TYTUŁ PRACY:
Aplikacja webowa zwiększająca
rozdzielncość obrazów

AUTOR:
Eryk Wójcik

PROMOTOR:
dr hab. inż. Andrzej Rusiecki,
Katedra Informatyki Technicznej

0

TODO: dodać do bibliografii framework i biblioteki użyte w apce TODO: Comic.png jest z Set14

Spis treści

1 Wstęp	3
2 Podstawy teoretyczne	5
2.1 Definicja super-rozdzielczości	5
2.2 Przegląd metod powiększania obrazów	6
2.3 Głębokie uczenie w przetwarzaniu obrazów	9
2.4 Wstęp do transformacji falkowej	14
3 Deep Wavelet Super Resolution	23
3.1 Architektura DWSR	24
3.2 Proces treningu	25
3.3 Przykłady zastosowań i rezultaty	27
4 Enhanced Super-Resolution Generative Adversarial Network	29
4.1 Architektura ESRGAN	30
4.2 Proces treningu	32
4.3 Przykłady zastosowań i rezultaty	32
5 Aplikacja webowa do powiększania rozdzielczości obrazów	35
5.1 Projektowanie aplikacji	37
5.2 Wybór narzędzi i technologii	39
5.3 Vue.js: Frontend	40
5.4 Django: Backend	41
5.5 Implementacja aplikacji	42
5.6 Integracja algorytmów super-rozdzielczości	44
5.7 Implementacja interfejsu użytkownika	47
5.8 Plany na przyszłość	53
6 Porównanie algorytmów ESRGAN i DWSR	57
6.1 Jakość odtwarzania obrazów	57
6.2 Analiza wydajności	61
6.3 Ograniczenia i wyzwania	62
7 Podsumowanie i wnioski	63
7.1 Dyskusja wyników	63
7.2 Rekomendacje i kierunki dalszych badań	63
Bibliografia	64

Rozdział 1

Wstęp

Cel pracy

Opis celu badań, czyli stworzenia aplikacji webowej służącej do zwiększania rozdzielczości obrazów z użyciem algorytmów ESRGAN i DWSR oraz analiza i porównanie tych algorytmów.

Zakres pracy

Przedstawienie koncepcji i zagadnień, które zostaną omówione w pracy, w tym wybrane metody i technologie.

Rozdział 2

Podstawy teoretyczne

Celem rozdziału jest przedstawienie podstawowych definicji, wytłumaczenie aparatu matematycznego oraz metod wykorzystywanych w algorytmach na których skupia się praca. Dodatkowo ma on na celu ułatwienie dalszego czytania poprzez zapoznanie czytelnika z przyjętymi konwencjami, oznaczeniami oraz symbolami, które mogą pojawić się w kolejnych rozdziałach.

2.1 Definicja super-rozdzielczości

Super-rozdzielcość (ang. Super-Resolution) odnosi się do procesu poprawy rozdzielczości obrazu lub sekwencji obrazów. W kontekście cyfrowym, super-rozdzielcość jest często realizowana za pomocą algorytmów komputerowych, które mają na celu odtworzenie wysokiej rozdzielczości obrazu [Rys 2, 3] z jednego lub wielu obrazów o niskiej rozdzielcość [Rys 1].



Rys 1. Obraz oryginalny



Rys 2. Obraz powiększony czterokrotnie



Rys 3. Obraz powiększony szesnastokrotnie

Przykłady zastosowań super-rozdzielczości

W kontekście praktycznym, techniki super-rozdzielczości znalazły zastosowanie w wielu dziedzinach. Przykładowo, w branży rozrywkowej, technika ta umożliwia remastering sta-

rych gier, filmów czy materiałów wideo do standardów HD czy 4K.

W ostatnich miesiącach popularne stało się generowanie obrazów z użyciem sztucznej inteligencji na podstawie podanego opisu tekstowego. Generowane w ten sposób obrazy są określonej wielkości i popularne modele takie jak **DALL-E** czy **Midjourney** nie pozwalają na zwiększenie jej. W tym miejscu z pomocą przychodzi super-rozdzielczość, która pozwala na powiększenie takich obrazów.

2.2 Przegląd metod powiększania obrazów

Istnieje wiele metod powiększania rozdzielczości obrazów. Najprostszą z nich jest **interpolacja najbliższego sąsiada (Nearest Neighbor, NN)**, która polega na powielaniu pobliskich pikseli w celu zwiększenia rozdzielczości obrazu.

Metoda ta jest bardzo prosta w implementacji, jednakże nie daje ona zadowalających rezultatów. Obraz powiększony w ten sposób wygląda jak obraz o niskiej rozdzielczości z większymi pikselami [Rys 5].



Rys 4. Obraz oryginalny



Rys 5. Obraz powiększony metodą NN

Aby poprawić jakość obrazu, można zastosować **interpolację dwuliniową**. Metoda ta rozszerza interpolację liniową na interpolację funkcji dwóch zmiennych [Rys 6].

W efekcie polega to na wyznaczeniu średniej ważonej pikseli sąsiadujących z pikselem, który chcemy powiększyć. Współczynniki wag są wyznaczane na podstawie odległości od piksela, który chcemy powiększyć. Kroki algorytmu:

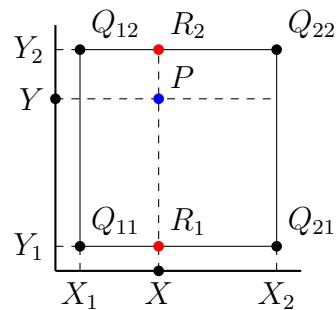
- Przeprowadzana jest interpolacja liniowa wzduż osi OX :

$$f(R_1) \approx \frac{x_2-x}{x_2-x_1} f(Q_{11}) + \frac{x-x_1}{x_2-x_1} f(Q_{21}) \quad \text{gdzie} \quad R_1 = (x, y_1),$$

$$f(R_2) \approx \frac{x_2-x}{x_2-x_1} f(Q_{12}) + \frac{x-x_1}{x_2-x_1} f(Q_{22}) \quad \text{gdzie} \quad R_2 = (x, y_2).$$

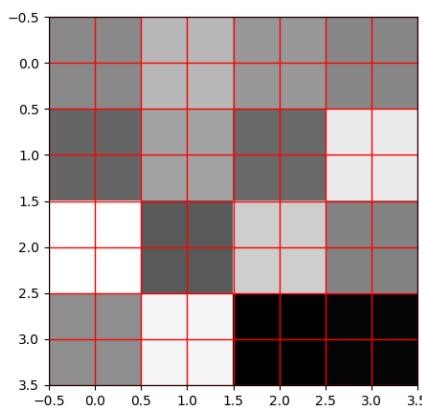
- Następnie przeprowadzana jest interpolacja wzduż osi OY :

$$f(P) \approx \frac{y_2-y}{y_2-y_1} f(R_1) + \frac{y-y_1}{y_2-y_1} f(R_2).$$

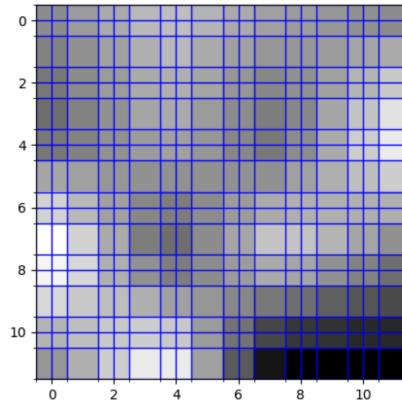


Rys 6. Wizualizacja interpolacji dwuliniowej

W efekcie otrzymujemy obraz wyglądający następująco [Rys 8, 10].



Rys 7. Obraz wejściowy



Rys 8. Obraz powiększony przez interpolację dwuliniową



Rys 9. Obraz wejściowy



Rys 10. Obraz powiększony przez interpolację dwuliniową

Metoda ta daje lepsze rezultaty niż interpolacja najbliższego sąsiada, jednakże wprowadziła ona duże rozmycie, które jest szczególnie widoczne na krawędziach obiektów i obszarach wysokiej częstotliwości.

Kolejnym algorytmem, jaki można zastosować w zadaniu Super-Rozdzielczości jest **interpolacja dwusześcienna (bicubic)**. Ta metoda interpolacji polega na wykorzystaniu funkcji sklejanych trzeciego stopnia w celu wygładzenia przejść między pikselami. W przeciwieństwie do interpolacji dwuliniowej, która bierze pod uwagę tylko 4 najbliższe piksele, interpolacja dwusześcienna bierze pod uwagę 16 pikseli sąsiadujących z piksem docelowym, co pozwala na bardziej płynne przejścia.

Kroki algorytmu interpolacji dwusześciennej:

1. Dla każdej osi (X i Y) obliczana jest wartość interpolowana przy użyciu funkcji sklejanych trzeciego stopnia:

$$f(x, y) \approx \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j$$

gdzie współczynniki a_{ij} są obliczane na podstawie wartości i pochodnych funkcji w punktach sąsiednich.

2. Interpolacja jest przeprowadzana osobno dla każdego kanału koloru, co pozwala na uzyskanie wysokiej jakości wyników, zwłaszcza w przypadku obrazów o dużym nasyceniu barw.

Uwaga: Interpolacja dwusześcienna jest bardziej obliczeniowo intensywna niż dwuliniowa, ale oferuje znacznie lepszą jakość, szczególnie przy znacznym powiększaniu obrazów.

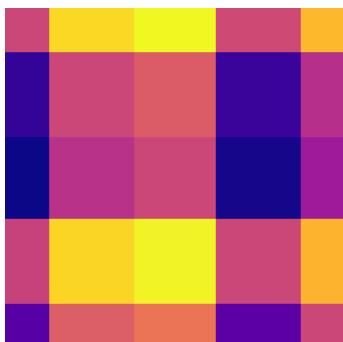


Rys 11. Obraz wejściowy

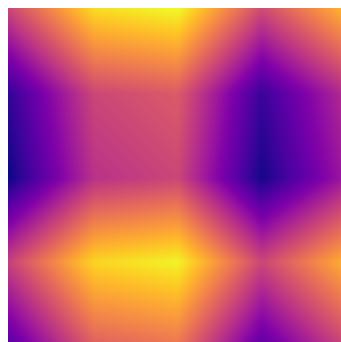


Rys 12. Obraz powiększony przez interpolację dwusześcienną

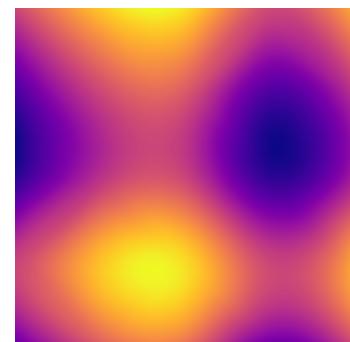
Na pierwszy rzut oka na obrazie [Rys 12] widać, że jest on bardziej wygładzony niż obraz powiększony metodą dwuliniową [Rys 10]. Różnicę między omówionymi algorytmami widać lepiej na przykładzie obrazu bardzo małego [Rys 13, 14, 15].



Rys 13. Obraz powiększo-ny przez NN



Rys 14. Obraz powiększo-ny przez Bilinear



Rys 15. Obraz powiększo-ny przez Bicubic

Na przykładzie powyższych wyników możemy stwierdzić, że każda kolejna metoda daje lepiej rozmyty obraz, wartości są coraz bardziej wygładzone. Jednakże wraz z wygładzeniem obrazu, tracimy na szczegółach. W przypadku obrazów o niskiej rozdzielczości, szczegóły są bardzo ważne, ponieważ są one jedynymi informacjami, które możemy wykorzystać do odtworzenia obrazu o wysokiej rozdzielczości.

Teoria informacji

W teorii informacji istnieje koncepcja zwana **nierównością przetwarzania danych**. Zgodnie z nią niezależnie od sposobu przetwarzania, **nie można dodać informacji, której nie ma w oryginalnej serii danych** [Wzór 2.1].

$$\begin{aligned} X \rightarrow Y \rightarrow Z \\ I(X; Y) \geq I(X; Z) \end{aligned} \tag{2.1}$$

Oznacza to, że brakujących danych nie można odzyskać poprzez dalsze przetwarzanie. Czy to oznacza, że super-rozdzielcość odwzorowująca szczegóły obrazów jest teoretycznie niemożliwa? Nie, jeśli mamy dodatkowe źródło informacji.

2.3 Głębokie uczenie w przetwarzaniu obrazów

Głębokie uczenie rewolucjonizuje przetwarzanie obrazów, wprowadzając modele zdolne do uczenia się cech z serii danych. W przetwarzaniu obrazów, głębokie sieci neuronowe są wykorzystywane do zadań takich jak detekcja obiektów, segmentacja, klasyfikacja obrazów, czy właśnie super-rozdzielcość [Rys 17, 18].

Sieć neuronowa może nauczyć się odtwarzać szczegóły obrazów na podstawie pewnych informacji, które zbiera z dużego zbioru obrazów.

Szczegóły dodawane do powiększanego obrazu w przy użyciu modelu uczenia maszynowego nie naruszają nierówności przetwarzania danych, ponieważ wykorzystywane informacje są w zbiorze treningowym, nawet jeśli nie ma ich na obrazie wejściowym.

Podstawy Głębokiego Uczenia

Głębokie uczenie, będące zaawansowaną formą uczenia maszynowego, wykorzystuje wielowarstwowe sieci neuronowe do analizy i interpretacji dużych zbiorów danych. Te sieci składają się z warstw skomplikowanych struktur algorytmicznych, które naśladują sposób, w jaki ludzki mózg przetwarza informacje.



Rys 16. Obraz wejściowy



Rys 17. Obraz powiększony algorytmem DWSR



Rys 18. Obraz powiększony algorytmem ESRGAN

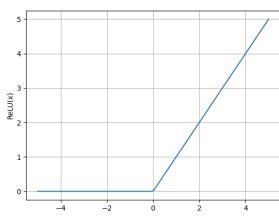
Architektura Sieci Neuronowych

Architektura sieci neuronowych w głębokim uczeniu charakteryzuje się wieloma ukrytymi warstwami, które pozwalają na przetwarzanie danych na różnych poziomach abstrakcji. Każda warstwa składa się z wielu neuronów, z których każdy otrzymuje dane wejściowe, przetwarza je i przekazuje dalej. Istnieją różne rodzaje warstw, w tym między innymi warstwy konwolucyjne (Convolutional Layers), które są fundamentem sieci konwolucyjnych. Są szeroko stosowane w przetwarzaniu obrazów, stosując filtr konwolucyjny do danych wejściowych wydobywając lokalne cechy takie jak krawędzie, kształty czy tekstury.

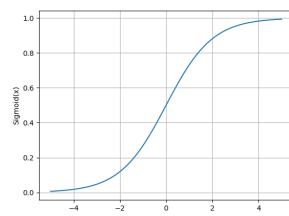
Funkcje Aktywacji

Funkcje aktywacji w sieciach neuronowych to nieliniowe transformacje stosowane do wyjść neuronów. Pozwalają one na modelowanie złożonych zależności między danymi wejściowymi a wyjściowymi. W głębokim uczeniu stosuje się różne funkcje aktywacji, w tym:

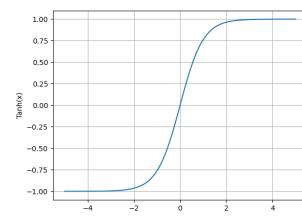
- **ReLU (Rectified Linear Unit)**: Przekształca wszystkie ujemne wartości na zero, podczas gdy wartości dodatnie pozostają niezmienione [Rys 19].
- **Sigmoid**: Konwertuje wartości wejściowe na te z zakresu od 0 do 1 [Rys 20].
- **Tanh (Hyperbolic Tangent)**: Podobnie jak sigmoid, ale konwertuje wartości na zakres od -1 do 1 [Rys 21].



Rys 19. ReLU



Rys 20. Sigmoid



Rys 21. Tanh

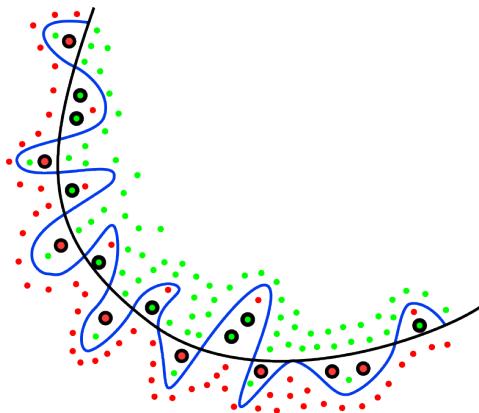
Strategie Uczenia w Głębokim Uczniu

Głębokie uczenie obejmuje różne strategie uczenia, które są stosowane w zależności od rodzaju i charakteru danych oraz oczekiwanych wyników. Do głównych strategii należą:

- **Uczenie nadzorowane (Supervised Learning):** W tym podejściu model uczy się na podstawie zestawu danych, które zawierają zarówno dane wejściowe, jak i odpowiednie etykiety. Jest to szeroko stosowane w zadaniach takich jak np. klasyfikacja.
- **Uczenie nienadzorowane (Unsupervised Learning):** Model próbuje znaleźć wzorce w danych bez etykiet, stosowane głównie w grupowaniu i redukcji wymiarowości.
- **Uczenie ze wzmacnianiem (Reinforcement Learning):** W tej strategii agent uczy się podejmować decyzje poprzez interakcję z otoczeniem, dążąc do maksymalizacji sumy nagród.

Przeuczenie i Generalizacja

Przeuczenie (Overfitting) występuje, gdy model zbyt dokładnie dopasowuje się do danych treningowych, tracąc zdolność do efektywnego działania na nowych danych [Rys 22]. Niebieska linia reprezentuje przeuczony model, zaś czarna dopasowany. Widać, że niebieska linia podąża za danymi treningowymi, jest zbyt zależna od nich, co sprawia, że model przeuczony będzie miał wyższy poziom błędu dla nowych danych. Jest to problem szczególnie w przypadku zbyt skomplikowanych modeli w stosunku do danych.



Rys 22. Wizualizacja przeuczenia

Celem jest stworzenie modelu, który efektywnie działa na nowych, nieznanych danych, co oznacza, że model jest dobrze dostosowany do rzeczywistych scenariuszy.

Przykładowe sposoby na zapobieganie przeuczeniu:

- **Wczesne zatrzymywanie (early stopping):** Polega na monitorowaniu wydajności modelu na zestawie walidacyjnym i zatrzymaniu treningu, gdy wydajność przestaje się poprawiać, co zapobiega przeuczeniu.
- **Walidacja krzyżowa (cross-validation):** Metoda oceny modelu, w której zestaw danych dzieli się na kilka części. Model jest następnie trenowany na jednej części

(zwanej zestawem treningowym) i walidowany na innej (zwanej zestawem walidacyjnym), co jest powtarzane na różnych kombinacjach części danych. Pozwala to na lepszą ocenę zdolności modelu do generalizacji na nieznanych danych.s

- **Augmentacja danych (data augmentation):** Zwiększa różnorodność danych treningowych poprzez wprowadzenie niewielkich losowych zmian, co pomaga modelowi lepiej uogólnić i zmniejszyć przeuczenie.

Generatywne Sieci Przestawne

Generatywne sieci przestawne (ang: *Generative Adversarial Networks*) to rodzaj architektury sieci neuronowej, która składa się z dwóch sieci: generatora i dyskryminatora. Generator próbuje wygenerować dane, które są podobne do danych treningowych, podczas gdy dyskryminator próbuje stwierdzić prawdopodobieństwo, że dane pochodzą z danych treningowych, a nie z generatora. Sposób ten wywodzi się z teorii gier, gdzie agenci grają przeciw sobie. W tym przypadku rolą generatora jest oszukiwanie dyskryminatora, podczas gdy rolą dyskryminatora jest nie dać się oszukać.

Te dwie sieci są trenowane naprzemiennie, aż do osiągnięcia równowagi, w której dyskryminator nie jest w stanie odróżnić danych wygenerowanych przez generator od danych treningowych, w wyniku czego dyskryminator zawsze zwraca prawdopodobieństwo równe 0.5. Ciekawą rzeczą w generatywnych sieciach jest to, że model musi pracować bardziej, gdy zawodzi. Dyskryminator szuka słabości generatora i zmusza go do poprawy, zaś generator adresuje te problemy. Gdy generator naprawi swoje błędy, dyskryminator szuka kolejnych słabości, co prowadzi do ciągłego rozwoju obu sieci.

Algorytmy Głębokiego Uczenia w Super-Rozdzielczości

Aby stworzyć algorytm głębokiego uczenia do zastosowania w problemie Super-Rozdzielczości potrzebujemy zestawu danych z obrazami o wysokiej rozdzielczości i tych samych obrazów ze zmniejszoną rozdzielczością.

Następnie trenujemy model, który jako wejście przyjmuje obraz o niskiej rozdzielczości i zwraca obraz o wysokiej rozdzielczości, który jest najbliższym oryginałem.

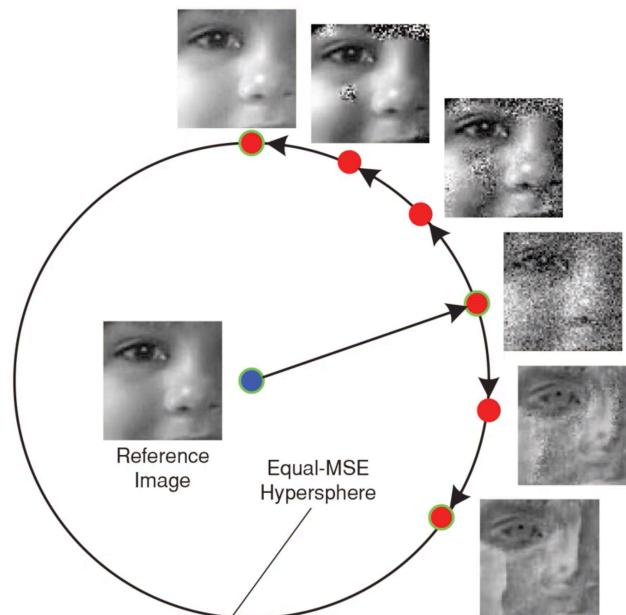


Rys 23. Przykład sieci neuronowej do super-rozdzielczości

Do porównania obrazu wyjściowego z obrazem oryginalnym można wykorzystać funkcję błędu średniokwadratowego *MSE* - *Mean Squared Error*, która jest zdefiniowana jako średnia arytmetyczna kwadratów różnic między wartościami pikseli w obrazie wyjściowym $\hat{\theta}$ i oryginalnym θ [Równanie 2.2].

$$MSE(\hat{\theta}) = \sum (\hat{\theta} - \theta)^2 \quad (2.2)$$

Ale czy metoda błędu średniokwadratowego jest odpowiednia do optymalizacji? Niestety błąd średniokwadratowy nie wyraża dobrze ludzkiego postrzegania wierności obrazu [12].



Rys 24. Przykład sieci neuronowej do super-rozdzielczości

Przykładowo wszystkie obrazy na Rysunku 24 mają ten sam błąd średniokwadratowy, ale różnią się znacząco wizualnie. W tym przypadku błąd średniokwadratowy nie jest w stanie wykryć różnic w jakości obrazu.

Metoda błędu średniokwadratowego nie bierze pod uwagę różnic strukturalnych między obrazami, z tego wynikają różnice przedstawione wyżej.

Rozwiązaniem tego może być inna miara błędu, która bierze pod uwagę różnice strukturalne między obrazami. Jedną z takich miar jest **błąd strukturalny SSIM** - *Structural Similarity Index Measure* [11], który jest zdefiniowany jako funkcja zależna od obrazu, która mierzy podobieństwo między dwoma obrazami. SSIM jest zdefiniowany jako iloczyn trzech funkcji: podobieństwa jasności, kontrastu i struktury [Równanie 2.3].

$$SSIM(x, y) = [l(x, y)][c(x, y)][s(x, y)] \quad (2.3)$$

Algorytm k-średnich

Algorytm k-średnich jest jednym z najprostszych algorytmów grupowania. Polega na podziale zbioru danych na k grup, w którym każdy punkt danych należy do grupy z najbliższym środkiem. Środki grup są obliczane jako średnia wszystkich punktów danych należących do grupy.

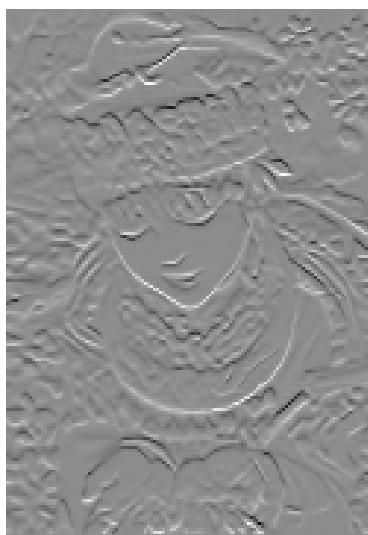
Algorytm k-średnich jest algorytmem iteracyjnym, który działa w następujący sposób:

1. Wybierz k punktów jako środki grup.
2. Przypisz każdy punkt danych do najbliższego środka grupy.
3. Dla każdej grupy oblicz nowy środek grupy, biorąc średnią wszystkich punktów danych należących do tej grupy.
4. Powtarzaj kroki 2 i 3, aż do momentu, gdy środki grup nie będą się już zmieniać lub zmiany będą poniżej pewnego progu.

Algorytm ten jest stosowany w aplikacji do wyznaczania dominujących kolorów występujących na obrazie. Więcej na ten temat opisane w części traktującej o aplikacji [Rozdział 5].

2.4 Wstęp do transformacji falkowej

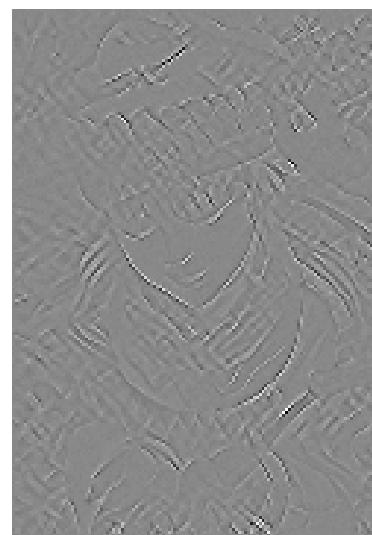
Transformacja falkowa (wavelet transform) stanowi istotne narzędzie w analizie sygnałów i przetwarzaniu obrazów, zwłaszcza w kontekście zastosowań takich jak super rozdzielczość. Charakterystyczna dla funkcji falkowych jest ich zdolność do dostosowania się do wymagań analizy sygnału. W przeciwieństwie do transformacji Fouriera, która skupia się wyłącznie na częstotliwości, falki pozwalają na efektywną lokalizację zjawisk w sygnale, uwzględniając zmiany zarówno w czasie, jak i częstotliwości.



Rys 25. Detale poziome



Rys 26. Detale pionowe

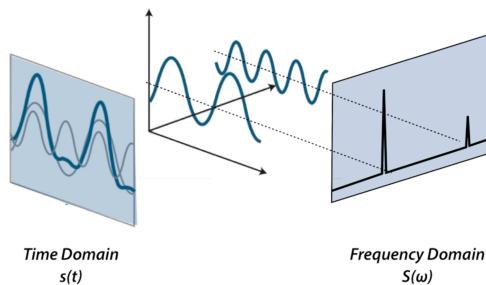


Rys 27. Detale diagonalne

W kontekście super rozdzielczości, funkcje falkowe są wykorzystywane do zwiększenia jakości obrazów i sygnałów poprzez umożliwienie dokładniejszej analizy i rekonstrukcji ich składowych. Mamy informacje o kierunkach i wielkościach częstotliwości w analizowanym obrazie [Rys 25, 26 27].

Dziedzina czasu i częstotliwości

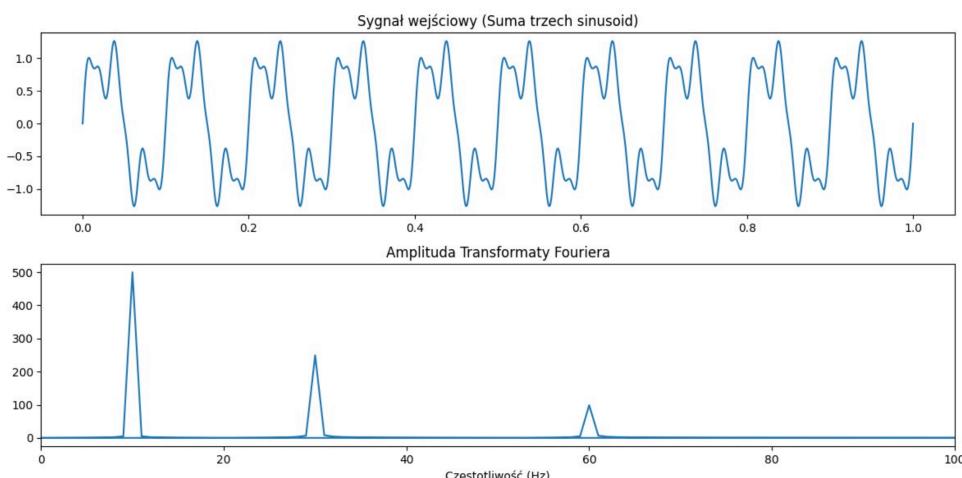
Dziedzina czasu odnosi się do analizy sygnału w zakresie czasu, co oznacza, że sygnał jest przedstawiany i analizowany w kontekście jego zmian w czasie. Jest to intuicyjna forma reprezentacji sygnałów, niemniej jednak nie daje ona nam pełnej informacji o sygnale. Z drugiej strony sygnał możemy określić w dziedzinie częstotliwości, która koncentruje się na analizie częstotliwościowych składników sygnału.



Rys 28. Wizualizacja dziedziny czasu i częstotliwości

Transformata Fouriera

Transformacja Fouriera jest algorytmem używanym do konwersji sygnału z dziedziny czasu do dziedziny częstotliwości. Pozwala ona na uzyskanie widma amplitudowego i fazowego, które prezentują częstotliwości występujące w sygnale i ich amplitudy.



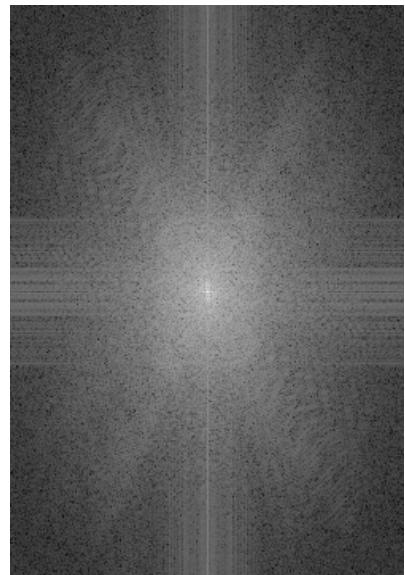
Rys 29. Wynik transformacji Fouriera dla przykładowego sygnału

W kontekście przetwarzania obrazów, transformata Fouriera jest stosowana przekształcenia obrazu do dziedziny częstotliwościowej. Wynik transformacji Fouriera jest liczbą zespoloną, która może być reprezentowana jako wektor złożony z dwóch części: rzeczywistej i urojonej, co być przedstawione w postaci dwóch wykresów: amplitudowego i fazowego [Rys 31, 32].

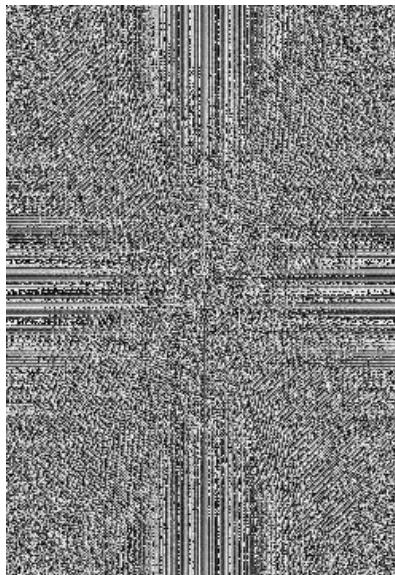
Transformacja Fouriera jest operacją odwracalną, co oznacza, że można ją wykorzystać do rekonstrukcji oryginalnego sygnału z częstotliwościowej reprezentacji stosując odwrotną transformację Fouriera.



Rys 30. Obraz wejściowy



Rys 31. Wykres amplitudowy



Rys 32. Wykres fazowy

Ograniczenia Transformacji Fouriera

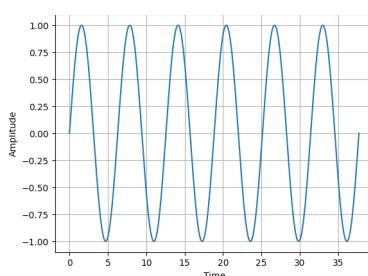
Zgodnie z zasadą nieoznaczoności Heisenberga nie można jednocześnie precyzyjnie określić i lokalizować zjawisk zarówno w dziedzinie czasu, jak i częstotliwości, zawsze mamy do czynienia z kompromisem. Możemy albo dokładnie określić wartość sygnału w czasie, albo dokładnie określić jego częstotliwość, ale nie możemy zrobić obu jednocześnie.

W rezultacie, podczas gdy Transformata Fouriera oferuje doskonałą rozdzielcość częstotliwościową, traci na zdolności do lokalizacji zjawisk w czasie. Oznacza to, że badając sygnał wyłącznie w dziedzinie częstotliwości, wiemy jakie częstotliwości występują, ale nie jesteśmy w stanie określić kiedy występuje dana częstotliwość.

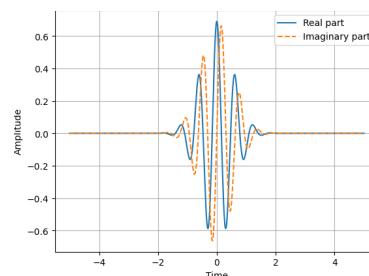
Czy istnieje narzędzie pozwalające nam na dokładniejszą analizę sygnału? Kompromis pomiędzy czasem a częstotliwością jest nieunikniony, ale istnieje sposób na poprawę lokalizacji w czasie kosztem rozdzielcości częstotliwościowej? Odpowiedzią na te pytania jest transformacja falkowa.

Funkcje Falkowe

Gdy wykonujemy transformatę Fouriera rozdzielimy sygnał na sumę sinusoid i cosinusoid o różnych częstotliwościach. Funkcje falkowe są podobne do funkcji sinusoidalnych, ale różnią się od nich tym, że mają skońzoną długość i są ograniczone do określonego obszaru [Rys 33, 34].



Rys 33. Sinusoida



Rys 34. Falka Morlet

Funkcje falkowe są rodziną funkcji. Aby funkcja $\Psi(t)$ była falką, musi spełniać następujące warunki:

- Funkcja musi mieć zerową średnią, czyli całka z funkcji musi być równa 0 (powierzchnia poniżej krzywej musi być równa powierzchni powyżej krzywej):

$$\int_{-\infty}^{+\infty} \Psi(t) dt = 0 \quad (2.4)$$

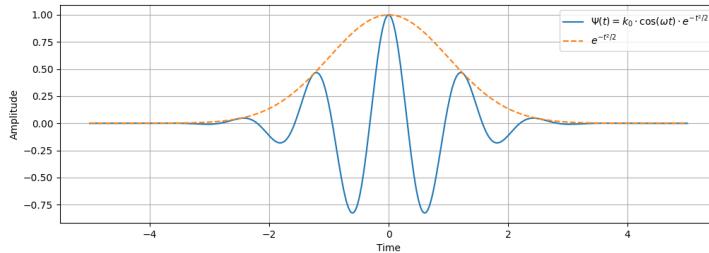
- Funkcja musi mieć skończoną energię (to ograniczenie sprawia, że funkcja jest ograniczona do określonego obszaru):

$$\int_{-\infty}^{+\infty} |\Psi(t)|^2 dt < \infty \quad (2.5)$$

Przykładem może być falka Morlet, która jest jedną z najczęściej stosowanych funkcji falkowych. Falka Morlet jest funkcją sinusoidalną, która jest przemnożona przez funkcję Gaussa. Rzeczywista część falki Morlet jest zdefiniowana następująco:

$$\Psi(t) = k_0 \cdot \cos(\omega t) \cdot e^{-\frac{t^2}{2}} \quad (2.6)$$

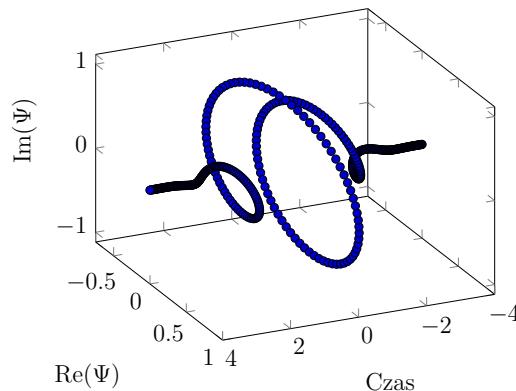
Część rzeczywista Morlet składa się funkcji cosinus (określająca częstotliwość falki) i funkcji Gaussa, która jest odpowiedzialna za ograniczenie falki do określonego obszaru.



Rys 35. Falka Morlet (część rzeczywista)

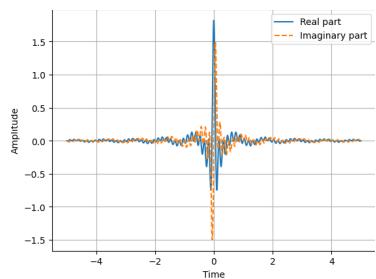
Falka Morlet jest funkcją zespoloną, co oznacza, że składa się z części rzeczywistej i urojonej. Całość jest zdefiniowana następująco [Rys 36]:

$$\Psi(t) = k e^{i \omega_0 t} \cdot e^{-\frac{t^2}{2}} \quad (2.7)$$

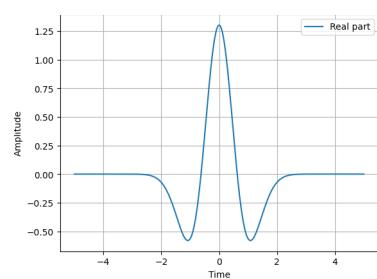


Rys 36. Falka Morlet w przestrzeni 3D

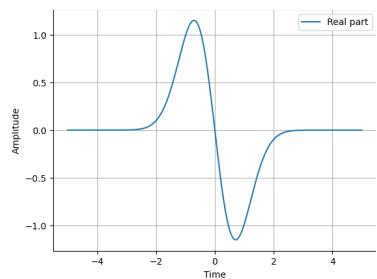
Inne funkcje falkowe:



Rys 37. Falka fbsp



Rys 38. Falka Mexican Hat



Rys 39. Falka Gaussa

Transformacja Falkowa

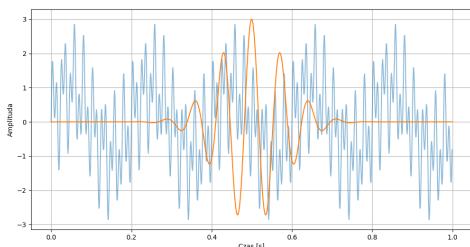
Transformacja falkowa to proces dekompozycji sygnału na zestaw funkcji falkowych. W przeciwnieństwie do transformacji Fouriera, która analizuje sygnał w kontekście czystych częstotliwości, transformacja falkowa rozkłada sygnał na serię "falek", które są przesuwane w czasie i częstotliwości, aby zbadać charakterystykę sygnału.

Transformację falkową przedstawiamy jako:

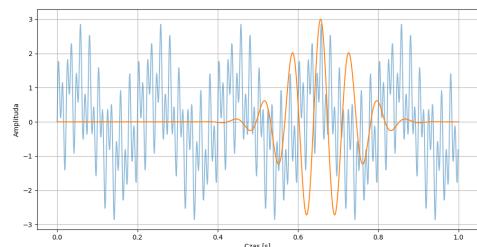
$$\tilde{s}_\Psi(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} s(t) \Psi\left(\frac{t-b}{a}\right) dt, \quad (2.8)$$

gdzie:

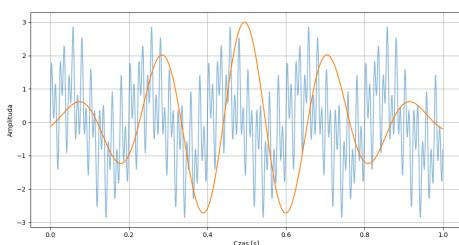
- $s(t)$ - sygnał wejściowy,
- $\Psi\left(\frac{t-b}{a}\right)$ - funkcja falkowa,
- b - parametr przesunięcia (w dziedzinie czasu) [Rys 41].
- a - parametr skali (przesunięcie w dziedzinie częstotliwości) [Rys 42],



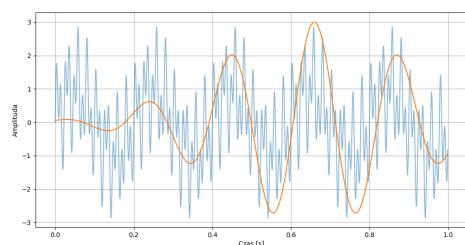
Rys 40. Brak przesunięcia



Rys 41. Przesunięcie w czasie

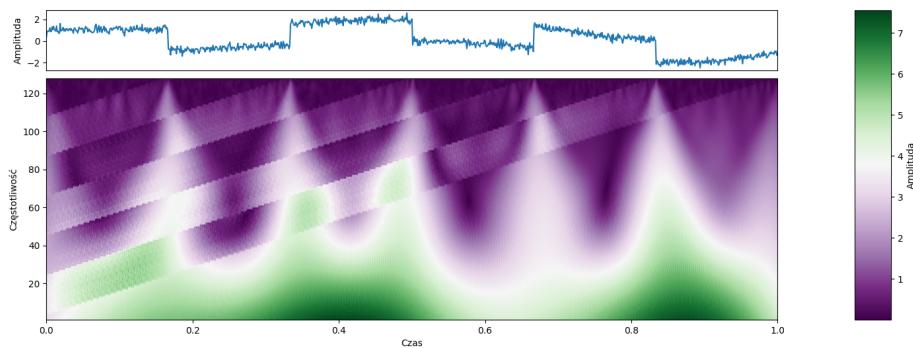


Rys 42. Przesunięcie w częstotliwości

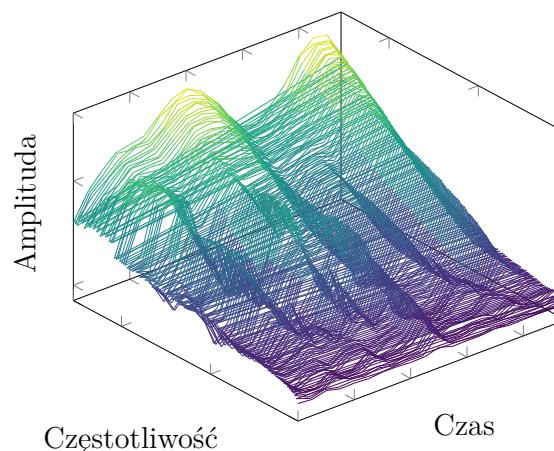
Rys 43. Przesunięcie w t i f

Wynikiem transformacji falkowej jest mapa współczynników, które są zależne od parametrów a i b , wybranej funkcji falkowej oraz sygnału wejściowego. Dla danych wartości a i b współczynnik jest miarą podobieństwa pomiędzy falką a fragmentem sygnału. Im większa wartość współczynnika, tym większe podobieństwo.

W rezultacie funkcja $\tilde{s}_\Psi(a, b)$ składa się na mapę współczynników, która jest reprezentacją sygnału w dziedzinie czasu i częstotliwości. Mapę taką nazywamy skalogramem, skalogram przykładowego sygnału przedstawiono na Rys 44 i 45.

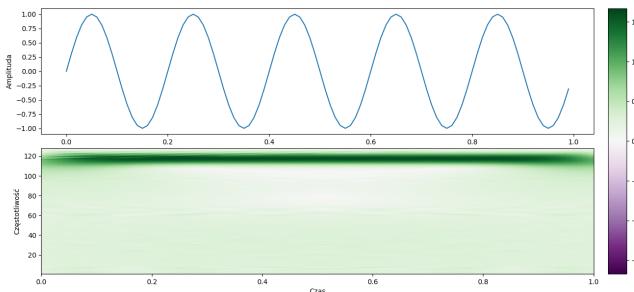


Rys 44. Skalogram falkowy w zestawieniu z sygnałem wejściowym



Rys 45. Skalogram falkowy w przestrzeni 3D

Rozdzielcość czasu i rozdzielcości w transformacji falkowej nie jest idealna, widać że mamy do czynienia z kompromisem pomiędzy tymi wartościami, szczególnie przy analizy zwykłej funkcji sinus. Skalogram dla sinusa [Rys 46] nie przedstawia jednoznacznie częstotliwości sygnału, jest to jedynie przybliżenie co widać na krawędziach skalogramu.



Rys 46. Niedokładność danych na skalogramie

Dyskretna Transformacja Falkowa

Dyskretna transformacja falkowa (DWT) jest dyskretną wersją transformacji falkowej, czyli działa na dyskretnych danych wejściowych. Przykładem takich danych może być cyfrowy sygnał taki jak obraz lub dźwięk.

Dla sygnałów dyskretnych, transformacja falkowa jest zdefiniowana jako:

$$D(m, a^j) = \frac{1}{\sqrt{a^j}} \sum_{n=0}^{N-1} x[n] \varphi^*(n/a^j), \quad (2.9)$$

gdzie:

- $x[n]$ - sygnał dyskretny,
- $\varphi[n]$ - dyskretna funkcja falkowa,
- a - parametr skali,
- j - poziom dekompozycji,
- N - liczba próbek sygnału,
- m - indeks próbki.

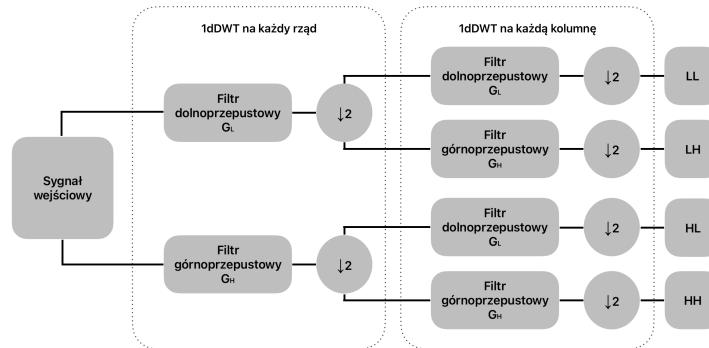
Aby wykonać dyskretną transformację falkową sygnał $x[n] \in \mathbb{R}^N$ jest przepuszczany przez filtr górnoprzepustowy $G_H[n]$ i filtr dolnoprzepustowy $G_L[n]$, które są definiowane jako (w przypadku falki Haara):

$$G_H[n] = \begin{cases} 1, & n = 0 \\ -1, & n = 1 \\ 0, & \text{w przeciwnym razie} \end{cases}, \quad G_L[n] = \begin{cases} 1, & n = 0, 1 \\ 0, & \text{w przeciwnym razie} \end{cases} \quad (2.10)$$

Po filtrowaniu połowa próbek może zostać wyeliminowana zgodnie z regułą Nyquista, ponieważ sygnał ma teraz pasmo częstotliwości $\frac{\pi}{2}$ radianów zamiast $/pi$.

Obraz x jest reprezentowany jako sygnał 2D o indeksach $[n, m]$, gdzie $x[n, m]$ jest wartością piksela w n -tej kolumnie i m -tym wierszu.

Sygnał dwuwymiarowy $x[n, m]$ może być traktowany jako dwa sygnały jednej zmiennej: $x[n, :]$ w n -tej kolumnie i wśród kolumn $x[:, m]$ w m -tym wierszu. Transformacja falkowa o pierwszym stopniu dekompozycji może być wykonywana w sposób przedstawiony na Rys 47.



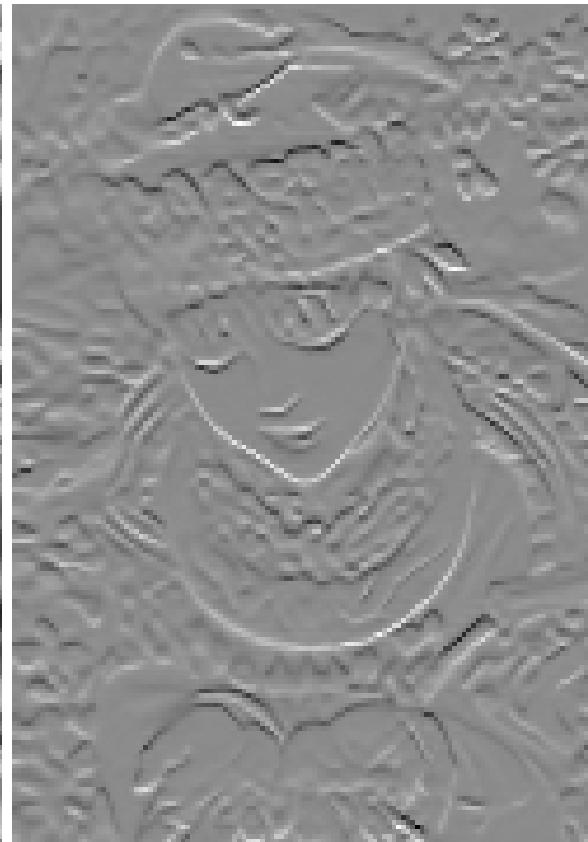
Rys 47. Dekompozycja DWT 1-poziomu

Przykładowa dekompozycja pierwszego stopnia wykonana na obrazie **comic.png** widoczna jest na Rys 48 - niskie częstotliwości na obrazie, 49 - poziome detale, 50 - pionowe detale, 51 - detale diagonalne. Obrazy po dekompozycji są tej samej wielkości co obraz wejściowy.

Jeśli chcemy wykonać odwrotną transformację falkową na obrazie, wystarczy wykonać kroki z Rys 47.



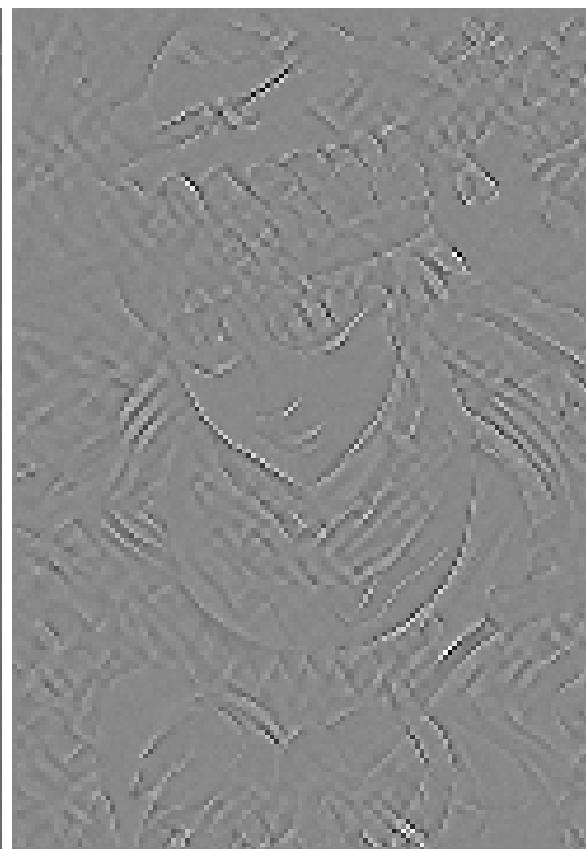
Rys 48. LL (niskie częstotliwości)



Rys 49. LH (poziome detale)



Rys 50. HL (pionowe detale)



Rys 51. HH (diagonalne detale)

Rozdział 3

Deep Wavelet Super Resolution

Pierwszy z omawianych algorytmów proponuje rozwiązanie problemu super-rozdziałości obrazów przy użyciu transformacji falkowej i głębszej sieci neuronowej. Algorytm został opracowany przez: Tiantong Guo, Hojjat Seyed Mousavi, Tiep Huu Vu, Vishal Monga, dla: School of Electrical Engineering and Computer Science, The Pennsylvania State University, State College, PA, 16803 [4].

Ten algorytm skupia się na wykorzystaniu transformacji falkowej do rekonstrukcji szczegółów obrazu o niskiej rozdziałości. W tej pracy badane są zalety wykorzystywania danych domen transformacji falkowej w zadaniu SR, zwłaszcza w celu uchwycenia większej ilości informacji strukturalnych w obrazach aby uniknąć artefaktów.



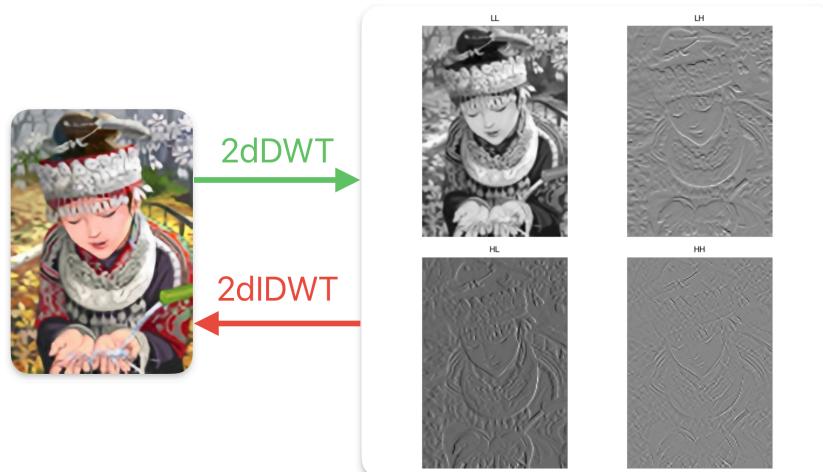
Rys 1. Obraz wejściowy



Rys 2. Obraz powiększony algorytmem **DWSR** czterokrotnie

3.1 Architektura DWSR

Na problem super-rozdrożliwości możemy spojrzeć jak na problem rekonstrukcji detali obrazu wejściowego o niskiej rozdrożliwości. Takie podejście świetnie współgra z dekompozycją transformacji falkowej. Jak widać na rysunku 3, jeśli potraktujemy obraz wejściowy jako wyjście LL poziomu 2dDWT, przewidywanie podpasm HL, LH i HH 2dDWT da nam brakujące szczegóły obrazu LL. Następnie można użyć 2dIDWT, aby zebrać przewidywane szczegóły i wygenerować wyniki SR.



Rys 3. Proces dekompozycji Dyskretnej Transformacji Falkowej

Korzystając z falki Haar, współczynniki 2dDWT można zapisać jako:

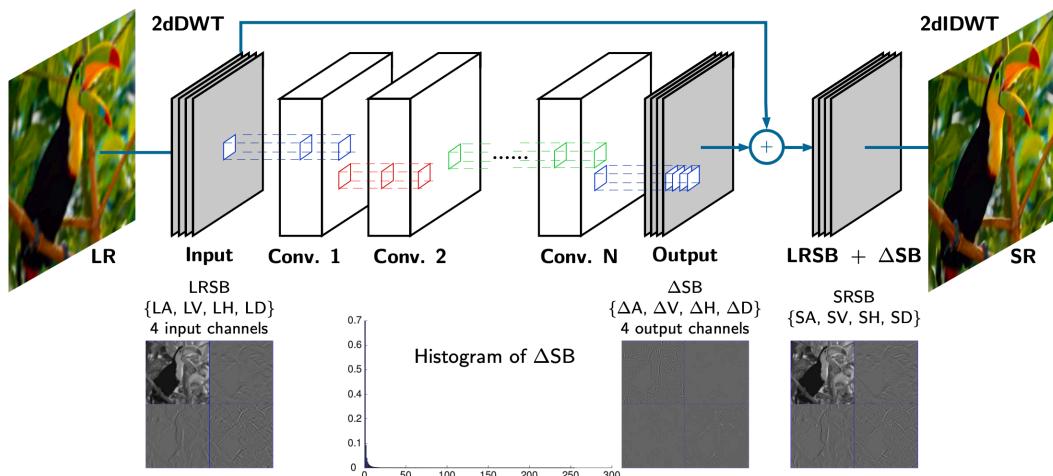
$$\begin{cases} A = a + b + c + d \\ B = a - b + c - d \\ C = a + b - c - d \\ D = a - b - c + d \end{cases} \quad (3.1)$$

gdzie:

- A, B, C, D są pikselami w siatce 2×2 w lewym górnym rogu obrazu HR,
- a jest pikselem w lewym górnym rogu obrazu LL,
- b jest pikselem w lewym górnym rogu obrazu HL,
- c jest pikselem w lewym górnym rogu obrazu LH,
- d jest pikselem w lewym górnym rogu obrazu HH.

Dlatego przy pomocy transformacji falkowej, problem SR staje się predykcją współczynników falkowych.

Struktura sieci



Rys 4. Struktura sieci DWSR [[4]]

Strukturę omawianej sieci [4] zilustrowano na Rys 4. Proponowana sieć ma głęboką strukturę z dwiema warstwami wejściowymi i wyjściowymi z 4 kanałami. Podczas gdy większość metod SR opartych na głębokim uczeniu ma tylko jeden kanał wejściowy i wyjściowy, przedstawiana sieć bierze pod uwagę cztery kanały wejściowe i generuje cztery odpowiadające im kanały na wyjściu.

W pierwszej warstwie znajdują się 64 filtry o rozmiarze $4 \times 3 \times 3$, a w ostatniej 4 filtry o rozmiarze $64 \times 3 \times 3$. W środkowej części sieci znajduje się N warstw ukrytych o tym samym rozmiarze z $64 \times 3 \times 3 \times 64$ filtrami każda. Dane wyjściowe każdej warstwy, z wyjątkiem warstwy wyjściowej, są podawane do funkcji aktywacji ReLU w celu wygenerowania nielinowej mapy aktywacji.

3.2 Proces treningu

Obrazy treningowe o niskiej rozdzielczości są powiększane przez interpolację dwusześcienne (omawianą w poprzednim rozdziale). Następnie powiększone obrazy LR są przetwarzane przez 2dDWT z falką Haara w celu uzyskania czterech podpasm LR (LRSB), które są oznaczone jako:

$$LRSB = \{LA, LV, LH, LD\} := 2dDWT \{LR\}, \quad (3.2)$$

gdzie:

- LA - współczynniki LL,
- LV - współczynniki HL,
- LH - współczynniki LH,
- LD - współczynniki HH.

Transformacja jest aplikowana w podobny sposób do obrazów o wysokiej rozdzielczości, aby uzyskać cztery podpasma HR (*HRSB*), które są oznaczone jako:

$$HRSB = \{HA, HV, HH, HD\} := 2dDWT\{HR\}, \quad (3.3)$$

gdzie:

- *HA* - współczynniki LL,
- *HV* - współczynniki HL,
- *HH* - współczynniki LH,
- *HD* - współczynniki HH.

Następnie różnicę ΔSB między *LRSB* i *HRSB* można zapisać jako:

$$\begin{aligned} \Delta SB &= HRSB - LRSB \\ &= \{HA - LA, HV - LV, HH - LH, HD - LD\} \\ &= \{\Delta A, \Delta V, \Delta H, \Delta D\} \end{aligned} \quad (3.4)$$

Naszym celem jest wygenerowanie ΔSB z *LRSB*. Procedura podawania dalej jest oznaczana jako $f(LRSB)$. Koszt wyjścia sieci jest zdefiniowany jako:

$$\text{cost} = \frac{1}{2} \|\Delta SB - f(LRSB)\|_2^2, \quad (3.5)$$

Wagi i odchylenia zostały oznaczone jako (Θ, b) , następnie problem optymalizacji jest zdefiniowany jako:

$$(\Theta, b) = \arg \min_{\Theta, b} \frac{1}{2} \|\Delta SB - f(LRSB)\|_2^2 + \lambda \|\Theta\|_2^2, \quad (3.6)$$

gdzie $\|\Theta\|_2^2$ jest standardową regulacją rozkładu wag parametru λ .

Ogólnie rzecz biorąc, sieć jest przygotowana do nauki różnic pomiędzy podpasmami obrazów o wysokiej i niskiej rozdzielczości. W ten sposób sieć uczy się rekonstruować szczegóły obrazu o wysokiej rozdzielczości na podstawie szczegółów obrazu o niskiej rozdzielczości.

Generowanie wyników SR

Aby uzyskać wyniki SR, obrazy wejściowe LR w powiększeniu przez interpolację dwuszczytnią są przekształcane przez 2dDWT w celu uzyskania *LRSB* jako Równanie 3.2. Następnie *LRSB* jest przekazywane dalej przez wytrenowaną sieć w celu uzyskania ΔSB . Dodanie *LRSB* i ΔSB razem generuje cztery SR wavelet Sub-Bands (*SRSB*) oznaczone jako:

$$\begin{aligned} SRSB &= \{SA, SV, SH, SD\} \\ &= LRSB + \Delta SB \\ &= \{LA + \Delta A, LV + \Delta V, LH + \Delta H, LD + \Delta D\} \end{aligned} \quad (3.7)$$

W ostatnim etapie 2dIDWT generuje SR obraz jako:

$$SR = 2dIDWT\{SRSB\} \quad (3.8)$$

Przygotowanie danych treningowych

Podczas fazy szkoleniowej wykorzystywanych jest 800 obrazów szkoleniowych **NTIRE** [9] bez rozszerzenia. Obrazy NTIRE HR $\{Y_i\}_{i=1}^{800}$ są próbkowane w dół o współczynnik c . Obrazy te są powiększane za pomocą interpolacji dwusześciennej o ten sam współczynnik c , aby utworzyć obrazy treningowe $LR \{X_i\}_{i=1}^{800}$. Należy zauważyć, że obraz Y_i jest przycinany tak, aby jego szerokość i wysokość były wielokrotnością c . Dlatego X_i i Y_i mają ten sam rozmiar, gdzie Y_i reprezentuje obraz treningowy HR , zaś X_i reprezentuje odpowiadający mu obraz treningowy LR . X_i i Y_i są następnie przycinane do pod-obrazów 41×41 pikseli z nakładającymi się 10 pikselami do treningu.

Dla każdego pod-obrazu z X_i $LRSB$ jest obliczane jako równanie 3.2. Dla każdego pod-obrazu z Y_i , $HRSB$ jest obliczany zgodnie z równaniem 3.3. Następnie resztę ΔSB jest obliczana zgodnie z równaniem 3.4.

Zarówno fazy uczenia, jak i testowania **DWSR** [4] wykorzystują tylko informacje o kanale luminancji. W przypadku obrazów kolorowych, kanały C_r i C_b są bezpośrednio powiększane przez interpolację dwusześcienną z obrazów LR . Te powiększone kanały chrominancji są łączone z kanałem luminancji SR w celu uzyskania kolorowych wyników SR .

Ustawienie parametrów treningu

Sieć została wytrenowana przez [4] w następujący sposób.

Podczas procesu uczenia wykorzystywanych było kilka technik. Gradienty są obcinane do 0,01 za pomocą opcji obcinania normy w pakiecie szkoleniowym. Użyty został optymalizator Adam. Początkowa szybkość uczenia wynosi 0,01 i zmniejsza się o 25% co 20 epok.

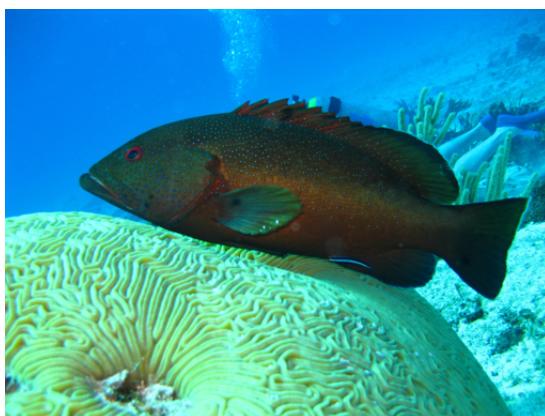
Regulator wagi jest ustawiony na 1×10^{-3} , aby zapobiec przeuczeniu. Oprócz warstw wejściowych i wyjściowych, **DWSR** ma $N = 10$ ukrytych warstw konwolucyjnych o tym samym rozmiarze z filtrem o rozmiarze $64 \times 3 \times 3 \times 64$. Ta konfiguracja skutkuje siecią o relatywnie niewielkiej liczbie parametrów. Schemat uczenia został zaimplementowany za pomocą pakietu TensorFlow z interfejsem interakcji Python 2.7. Użyty został jeden procesor graficzny GTX TITAN X 12 GB zarówno do uczenia, jak i testowania.

3.3 Przykłady zastosowań i rezultaty

Przykładowe wyniki działania algorytmu **DWSR** przedstawiono na Rys: 6 8, 10. Obrazy pochodzą z repozytorium GitHub, gdzie umieszczony został również algorytm **DWSR** [4].

Ilustracja praktycznych zastosowań DWSR oraz ocena i interpretacja osiągniętych dzięki niemu wyników. Omawiany algorytm świetnie sobie radzi z rekonstrukcją detali z powtarzających się wzorów, jak na przykład pasy zebry, czy trawa na obrazie 10, krawędzie okien na obrazie 8, czy tekstura łusek ryby na obrazie 6.

Główną zaletą algorytmu **DWSR** jest szybkość działania, co wynika z niewielkiej ilości parametrów. W porównaniu z innymi metodami SR, **DWSR** osiąga konkurencyjne lub lepsze wyniki, a jednocześnie jest znacznie oszczędniejsza pod względem liczby parametrów. Dzieje się tak ponieważ falki zapewniają reprezentację obrazu, która naturalnie upraszcza mapowanie, którego należy się nauczyć. Dokładniejsza analiza działania algorytmu **DWSR**, oraz analiza porównawcza z algorytmem **ESRGAN** została przedstawiona w Rozdziale 6.



Rys 5. Obraz wejściowy



Rys 6. Obraz powiększony algorytmem **DWSR** czterokrotnie



Rys 7. Obraz wejściowy



Rys 8. Obraz powiększony algorytmem **DWSR** czterokrotnie



Rys 9. Obraz wejściowy



Rys 10. Obraz powiększony algorytmem **DWSR** czterokrotnie

Rozdział 4

Enhanced Super-Resolution Generative Adversarial Network

Kolejny omawiany algorytm rozwiązuje problem super-rozdzielczości obrazów przy użyciu Generatywnych Sieci Przestawnych. Algorytm został opracowany przez: Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, Xiaou Tang [10].

ESRGAN jest rozwinięciem algorytmu **SRGAN** [7]; wprowadza kilka istotnych zmian, które znacznie poprawiają jakość obrazów wygenerowanych przez sieć. W tej pracy badane są zalety zastosowanych zmian w architekturze sieci generatora i dyskryminatora, oraz funkcji straty.



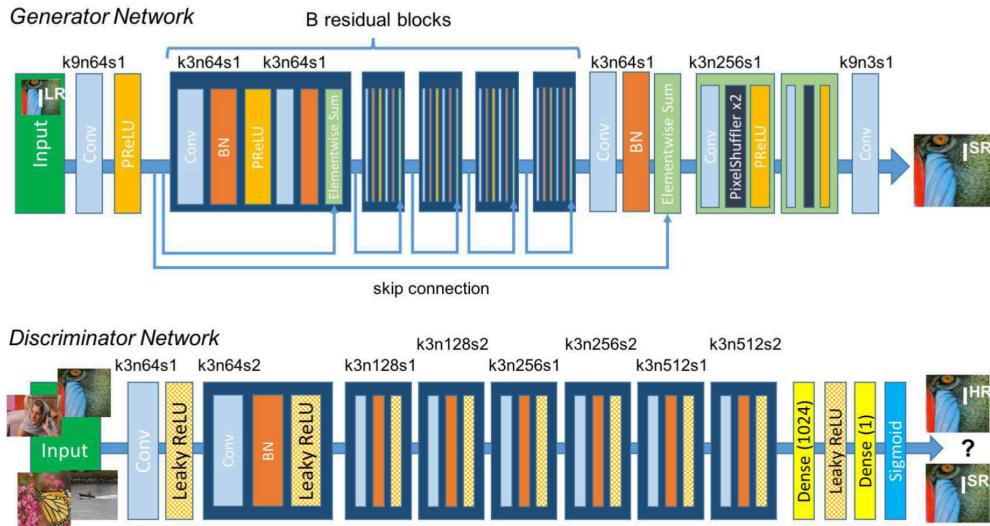
Rys 1. Obraz wejściowy



Rys 2. Obraz powiększony algorytmem **ESRGAN** czterokrotnie

4.1 Architektura ESRGAN

Algorytm **ESRGAN** bazuje na założeniach **SRGAN** [7], jednak wprowadza istotne zmiany w architekturze sieci, przeciwnych strat i strat percepcyjnych.



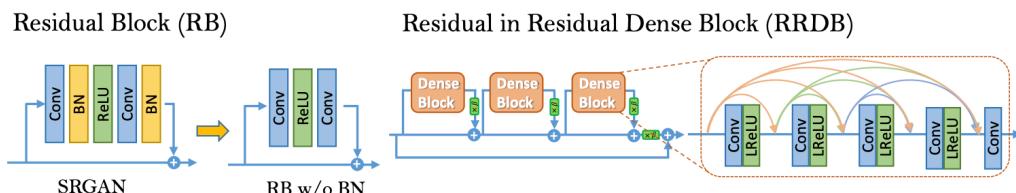
Rys 3. Architektura SRGAN [10]

Architektura **SRGAN** składa się z dwóch sieci: generatora i dyskryminatora [Rys 3]. Generator przyjmuje na wejściu obraz o niskiej rozdzielcości i zwraca obraz o wysokiej rozdzielcości. Dyskryminator przyjmuje na wejściu obraz o wysokiej rozdzielcości i zwraca prawdopodobieństwo, że obraz jest rzeczywisty, a nie wygenerowany przez sieć.

SRGAN wykorzystuje VGG loss, czyli funkcję straty opartą na głębszej sieci neuronowej VGG, używaną do oceny podobieństwa percepcyjnego między obrazami. Zamiast mierzyć różnicę pomiędzy pikselami, jak w przypadku błędu średniokwadratowego (MSE), VGG loss porównuje cechy reprezentacyjne (np. tekstury, kształty) wyekstrahowane przez sieć VGG z obu obrazów. Poprzez użycie tej funkcji straty, SRGAN skupia się na generowaniu obrazów, które są percepcyjnie bardziej podobne do obrazów wysokiej rozdzielcości, zamiast jedynie minimalizować błąd średniokwadratowy. Dzięki temu, generowane obrazy charakteryzują się lepszą jakością percepcyjną, bardziej zbliżoną do naturalnych zdjęć.

ESRGAN wprowadza kilka istotnych zmian w architekturze sieci. Zmiany te mają na celu poprawę jakości generowanych obrazów i eliminację artefaktów na obrazach wyjściowych.

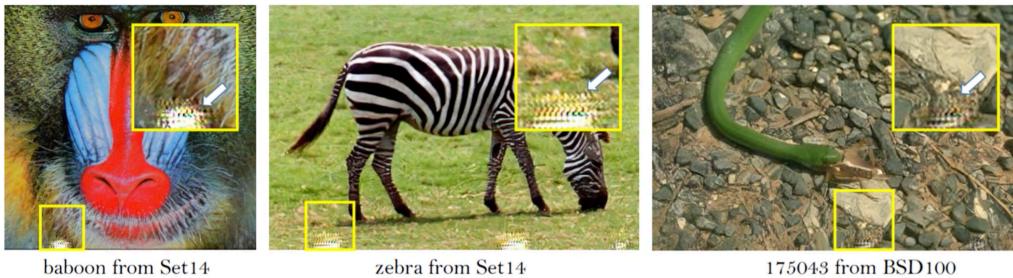
Zmiany w architekturze



Rys 4. Zmiany zastosowane w architekturze ESRGAN [10]

Pierwszą zmianą było usunięcie warstw normalizacji wsadowej z generatora (lewa strona Rys 4). Normalizacja wsadowa jest techniką stosowaną w procesie uczenia sieci neuronowych, która ma na celu poprawę stabilności i szybkości uczenia poprzez normalizację danych wejściowych w każdej warstwie na podstawie mini-zestawów (batchy). Technika ta polega na dostosowaniu średniej i wariancji danych w każdym mini-zestawie, co pomaga w redukcji problemu zwanego "wewnętrzny przesunięciem kowariancji" (internal covariance shift) i sprzyja szybszemu i bardziej stabilnemu uczeniu sieci.

Warstwy te powodowały, że obrazy wyjściowe posiadały artefakty [Rys 5] - powtarzające się tekstury.

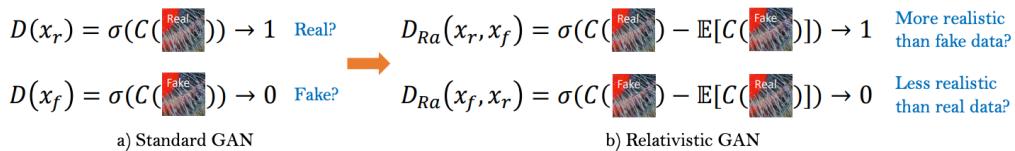


Rys 5. Przykłady artefaktów powstały przez normalizację wsadową [10]

Kolejną zmianą było powiększenie ilości warstw w generatorze (prawa strona Rys 4). W **SRGAN** generator składał się z 16 warstw, natomiast w **ESRGAN** z 23 warstw. Zwiększenie ilości warstw pozwoliło na zwiększenie złożoności sieci, co przełożyło się na lepszą jakość generowanych obrazów.

Zmiany w dyskryminatorze

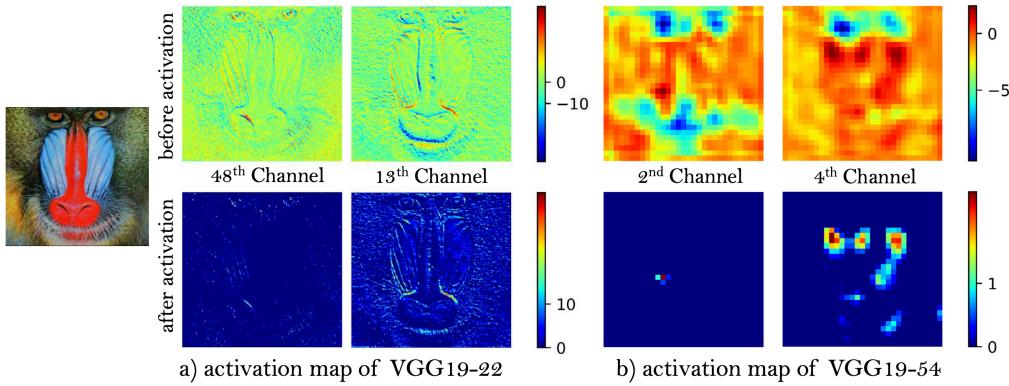
W **ESRGAN** zastosowano zmiany nie tylko w generatorze, ale również w dyskryminatorze stosując dyskryminator relatywistyczny [6]. Dyskryminator stosowany w **SRGAN** zwracał prawdopodobieństwo, że obraz jest rzeczywisty, a nie wygenerowany przez sieć. Dyskryminator relatywistyczny przewiduje prawdopodobieństwo czy obraz prawdziwy jest relatywnie bardziej realistyczny niż obraz wygenerowany przez generator [Rys 6].



Rys 6. Różnica między dyskryminatorem standardowym a relatywistycznym [10]

Zmiany w funkcji straty

Zastosowano również zmianę w funkcji straty. Zastosowano funkcję strat VGG, ale mapy cech były porównywane przed aktywacją, a nie po. Porównanie cech przed aktywacją pozwala wykorzystanie większej ilości informacji, gdyż mapy aktywacji są gęstsze i zawierają więcej informacji o obrazie przed analizą przez funkcję aktywacji [Rys 7].



Rys 7. Mapa cech przed aktywacją i po aktywacji [10]

4.2 Proces treningu

Podobnie jak w **SRGAN** [7], w **ESRGAN** [10] obrazy powiększane są ze skalą 4 pomiędzy obrazami *LR* a *HR*.

Do treningu **ESRGAN** wykorzystano bazę danych DIV2K [1], która zawiera 800 zdjęć o wysokiej rozdzielczości (2k), oraz zbioru danych Flickr2K, który zawiera 2650 zdjęć o takiej samej rozdzielczości. Dodatkowo wzbogacono zestaw treningowy o losowe odbicia lustrzane i rotacje o 90 stopni.

Obrazy *HR* przeskalowano z użyciem Matlaba. Używano mini-zestawów (mini-batch) o wielkości 16 i kadrowano fragmenty HR o rozmiarze 128×128 pikseli. Większy rozmiar fragmentów pomagał w przechwytywaniu większej ilości informacji semantycznej, co było korzystne dla głębszych sieci, ale zwiększało czas treningu i zapotrzebowanie na zasoby obliczeniowe.

Do optymalizacji użyto metody Adam z parametrami $\beta_1 = 0.9$ i $\beta_2 = 0.999$, aktualizując na przemian sieci generatora i dyskryminatora do momentu zbieżności modelu.

Jako zestaw testowy wykorzystano obrazy z zestawów danych Set5 [2], Set14 [13], BSD100 [8] i Urban100 [5].

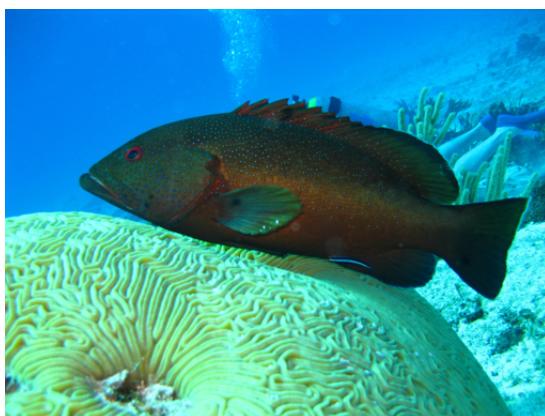
4.3 Przykłady zastosowań i rezultaty

Przykładowe wyniki działania algorytmu **ESRGAN** przedstawiono na Rys: 9 11, 13. Obrazy pochodzą z repozytorium GitHub, gdzie umieszczony został również algorytm **DWSR** [4].

Omawiany algorytm radzi sobie jeszcze lepiej niż opisywany w poprzednim rozdziale **DWSR**, zwłaszcza z teksturami takimi jak włosy, czy futro. **ESRGAN** świetnie radzi sobie z krawędziami obiektów na obrazach, widać to na przykładzie Rys. 11, gdzie ramy okien są bardzo wyraźnie zarysowane.

Jako, że **ESRGAN** jest dużym i skomplikowanym modelem jego czas egzekucji jest znacznie dłuższy niż w przypadku **DWSR**.

Dokładniejsza analiza działania algorytmu **ESRGAN**, oraz analiza porównawcza z algorytmem **DWSR** została przedstawiona w Rozdziale 6.



Rys 8. Obraz wejściowy

Rys 9. Obraz powiększony algorytmem
ESRGAN czterokrotnie

Rys 10. Obraz wejściowy

Rys 11. Obraz powiększony algorytmem
ESRGAN czterokrotnie

Rys 12. Obraz wejściowy

Rys 13. Obraz powiększony algorytmem
ESRGAN czterokrotnie

Rozdział 5

Aplikacja webowa do powiększania rozdzielczości obrazów

Podczas korzystania z Internetu miałem kilka sytuacji w których potrzebowałem narzędzia, które pozwoli mi na powiększenie rozdzielczości obrazów. Stron internetowych tego typu jest wiele, sporo aplikacji do edycji zdjęć umożliwia powiększenie rozdzielczości obrazów, przykładem może być **Adobe Photoshop**.

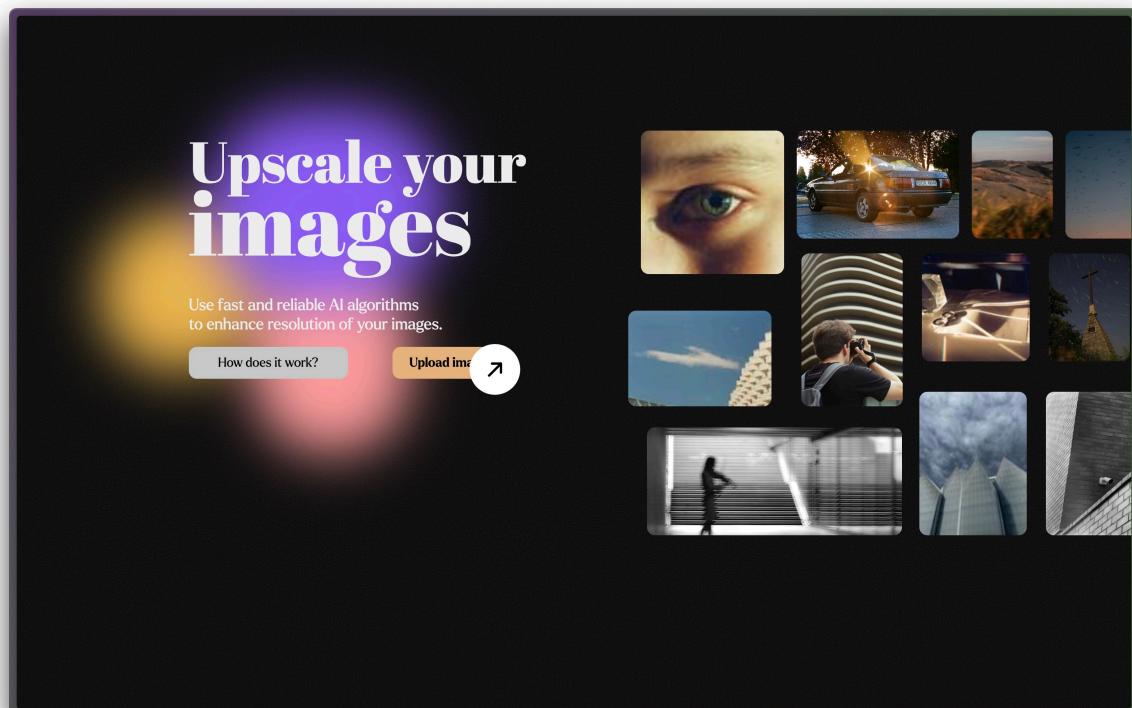
Korzystając z rozwiązań ogólnodostępnych zauważylem, że darmowe aplikacje nie gwarantują wysokiej jakości obrazów wyjściowych, zaś bardziej rozbudowane rozwiązania są płatne lub mają dostępne tylko jedno powiększenie obrazu na dobę.

Postanowiłem więc wykorzystać ogólnodostępne algorytmy rozwiązuające problem superrozdzielczości, następnie użyć ich do aplikacji która w domyśle będzie darmowa i będzie dawała możliwość porównania wyników działania kilku algorytmów. Nie zawsze jedna z wdrożonych metod będzie dawać najlepsze wyniki, więc chciałbym żeby użytkownik miał wybór a pro po tego którą metodę chce wykorzystać.

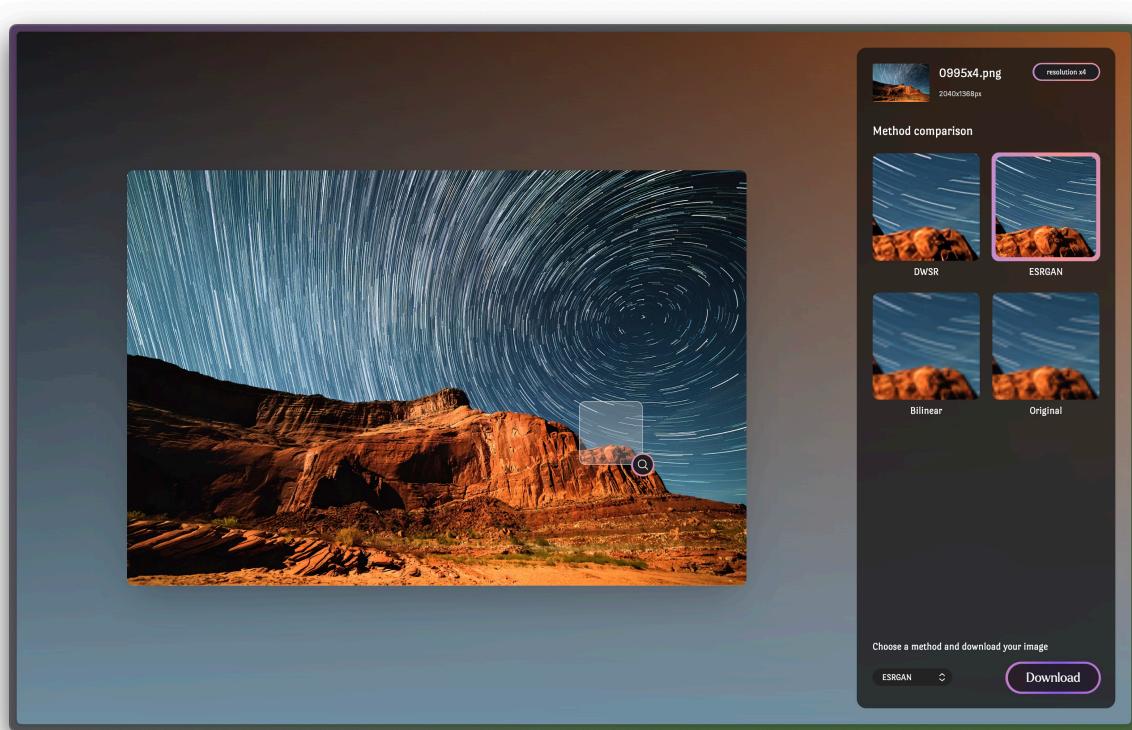
Głównym założeniem aplikacji jest intuicyjność i minimalizacja interakcji potrzebnych do powiększenia obrazu; zauważylem, że rozwiązania konkurencji wymagają akceptacji regulaminów, lub dopiero po przejściu kilku ekranów możemy wysłać obraz, którego rozdzielcość chcemy powiększyć. Uznałem to za bardzo ważne, gdyż użytkownicy chcą szybko dostać wynik, użytkownik nie będzie czekał nie wiadomo jak długo na wynik.

Nie spotkałem się z tym, żeby aplikacje innych firm pozwalały na porównanie wyników różnych algorytmów ze sobą. Uważam że to jest istotne gdy chcemy uzyskać wynik najwyższej jakości, bo jak wspomniałem nie ma rozwiązań idealnych i nie zawsze jedna z wdrożonych metod będzie dawać najlepsze wyniki.

Ważne jest dla mnie, żeby aplikacja była estetyczna i przyjemna dla oka. Dużo częściej korzystamy z narzędzi czy urządzeń które lepiej wyglądają, bardziej się nam podobają i chciałbym żeby tak było w tym przypadku. W planie również jest dbanie o animacje i o to żeby interakcje z narzędziem były przyjemne i satysfakcjonujące.



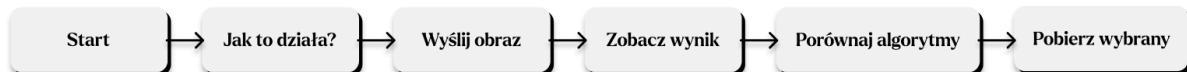
Rys 1. Widok strony głównej aplikacji



Rys 2. Widok prezentacji wyników

5.1 Projektowanie aplikacji

Pierwszym etapem tworzenia aplikacji było stworzenie diagramu przepływu użytkownika (user flow diagram) [Rys 3], który ilustruje kolejność interakcji użytkownika z aplikacją. Potencjalni użytkownicy mają już pewne oczekiwania i potrzeby, spodziewają się gdzie na ekranie znajdą się konkretne elementy. Dlatego ważne jest, żeby zrozumieć jak użytkownik będzie korzystał z aplikacji, jakie akcje będzie wykonywał i w jaki sposób będzie się poruszał po stronie.

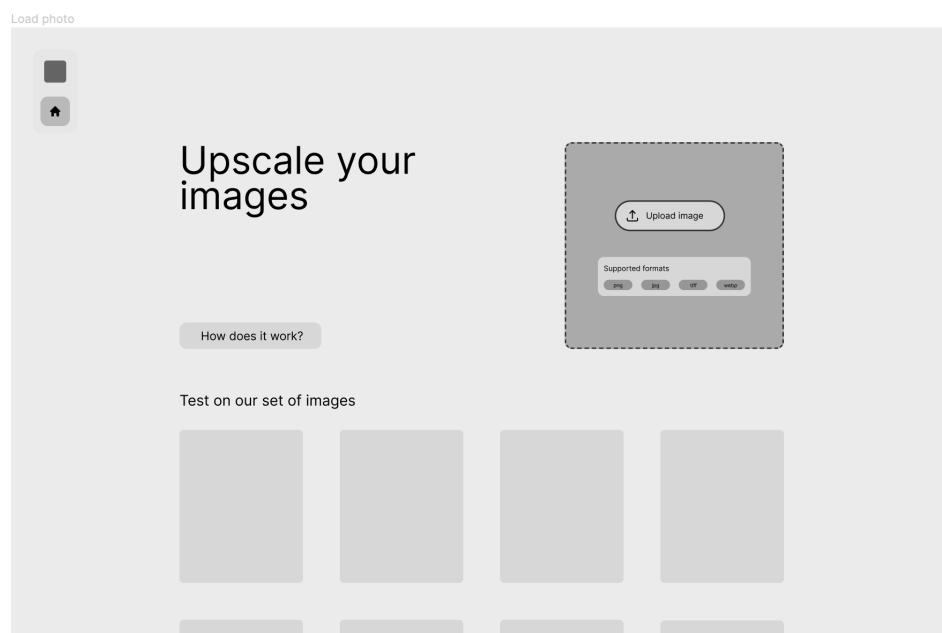


Rys 3. Diagram przepływu użytkownika

Diagram ten przedstawia wszystkie interakcje, które użytkownik może wykonać w aplikacji. Te interakcje nie muszą być wykorzystane, ale są dostępne dla użytkownika.

Kolejnym krokiem był projekt interfejsu użytkownika. Zdecydowałem, że narzędzie będzie składać się z dwóch ekranów: ekranu głównego 1, oraz widoku prezentacji obrazu wynikowego 2.

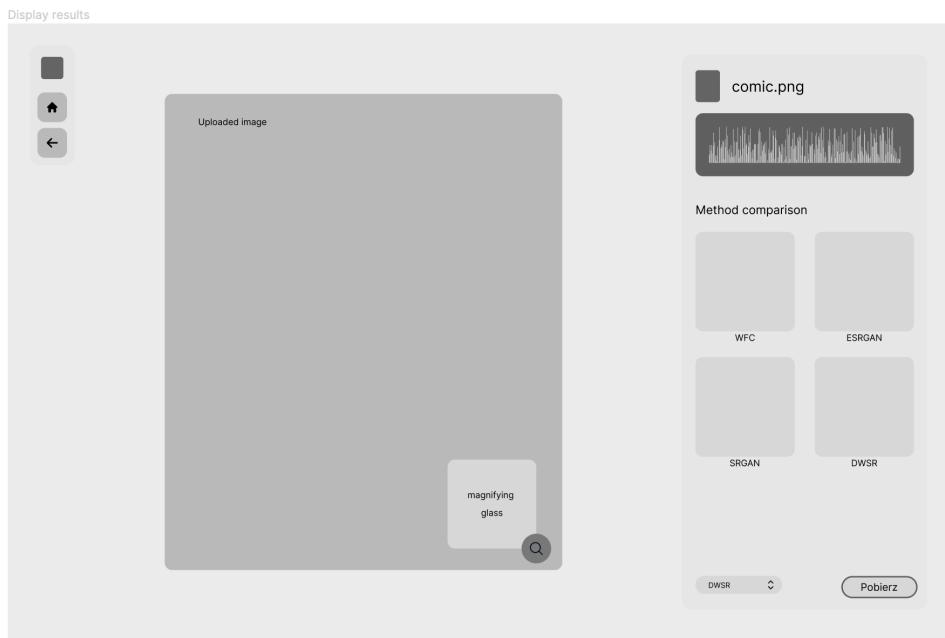
Projekt wyglądu aplikacji rozpocząłem od rozrysowania wireframe'ów [Rys 4] z użyciem narzędzia **Figma**, służącym do projektów graficznych między innymi aplikacji. Wireframe'y to proste szkice, które pozwalają na zobrazowanie układu elementów na stronie. Elementy te odpowiadają za funkcjonalność, a nie wygląd aplikacji i są ścisłe powiązane z diagramem przepływu użytkownika.



Rys 4. Pierwsza wersja UX aplikacji (ekran główny)

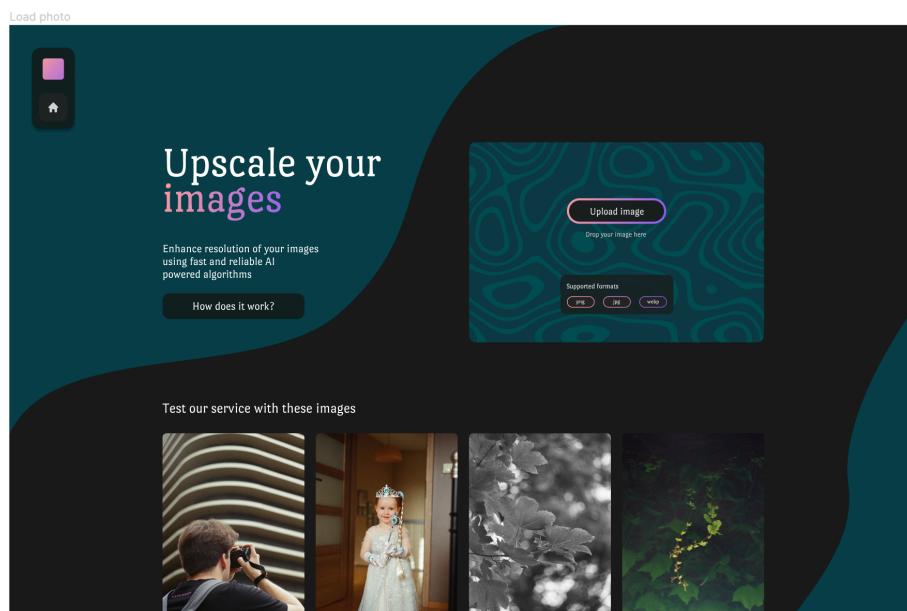
Pierwotny rozkład elementów na stronie różni się od tego jak to wygląda teraz. Wraz z rozwojem aplikacji zmieniałem układ elementów, usprawniałem interakcję z użytkownikiem i poprawiałem rozkład elementów na ekranach aplikacji.

Kolejnym etapem pracy był projekt graficzny aplikacji. Zdecydowałem, że aplikacja będzie w ciemnym motywie, gdyż taki styl pomaga nam skupić się na tym co jest na ekranie,



Rys 5. Pierwsza wersja UX aplikacji (ekran prezentacji wyników)

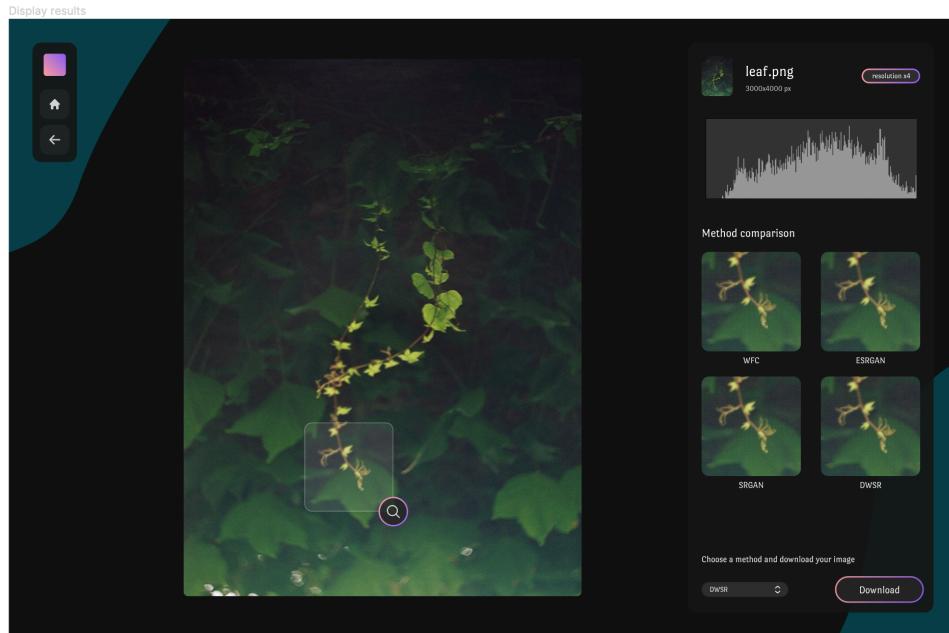
zwłaszcza w kontekście edycji zdjęć. Zależało mi na tym, żeby aplikacja nie była jednowymiarowa i żeby wyglądała nowocześnie. Początkowo eksperymentowałem z grafikami wektorowymi w tle [Rys 6], lecz ten wygląd nie przekonywał mnie. Eksperymentując z wyglądem pomyślałem, że w nawiązaniu do szumu na zdjęciach analogowych, tło aplikacji może mieć szum, zaś jako że mamy do czynienia z nowoczesnym narzędziem to kurSOR i elementy UI będą przejrzyste i czyste. W ten sposób aplikacja zyskuje na głębi i tak wygląda aktualna wersja, którą postanowiłem zaimplementować [Rys 1].



Rys 6. Pierwsza wersja UI strony głównej

Podobnie wyglądał proces projektowania ekranu prezentacji wyników [Rys 7]. Zdecydowałem, żeby na ekranie prezentacji wyników były tylko istotne elementy interfejsu, żeby

użytkownik nie pogubił się w nadmiarze informacji. Zależało mi na tym, żeby użytkownik mógł łatwo porównać wyniki różnych algorytmów, dlatego zdecydowałem się na układ kafelków, które można wybrać i porównać z sobą. Dodatkowo w tych kafelkach uznałem, że świetnie sprawdzi się widok z bliska, który pozwoli na dokładne przyjrzenie się szczegółom obrazu, tak powstała lupka, która podąża za kursorem gdy wskaźnik znajduje się nad obrazem.



Rys 7. Pierwsza wersja UI ekranu wyświetlania wyników

W kolejnej wersji widoku prezentacji wyników postanowiłem zmienić wygląd tła. Zainspirowany aplikacjami typu **Apple Music** czy **Spotify** zdecydowałem się na zmianę tła na gradient z kolorów występujących na obrazie. W ten sposób aplikacja zyskuje na głębi i wygląda bardziej nowocześnie [Rys 2]. W tym miejscu projekt aplikacji uznałem za gotowy do implementacji, gdzieś trzeba się zatrzymać, żeby sprawdzić jak działają mechanizmy w praktyce. O planach rozbudowy aplikacji traktuję rozdział 5.8.

5.2 Wybór narzędzi i technologii

Kolejnym krokiem w tworzeniu aplikacji jest decyzja odnośnie używanych technologii i narzędzi. W tym rozdziale opiszę jakie technologie wybrałem do stworzenia aplikacji webowej.

Aplikacja dzieli się na Frontend i Backend, gdyż zadanie Super-Rozdzielczości, podobnie jak inne zadania z dziedziny uczenia maszynowego, wymaga dużej mocy obliczeniowej. Z tego powodu implementowane algorytmy będą działać po stronie Backendu, a Frontend będzie odpowiedzialny za interfejs użytkownika, wyświetlenie wyników i komunikację z Backendem. Dodatkowo Backend pozwala na przechowywanie obrazów w bazie danych, co jest istotne gdy chcemy porównać wyniki różnych algorytmów ze sobą od strony administratora.

5.3 Vue.js: Frontend

Do implementacji Frontendu zdecydowałem się na użycie frameworku **Vue.js**

Vue.js to progresywny framework JavaScript służący do budowania interfejsów użytkownika. Został stworzony przez Evaną You i jest utrzymywany przez niezależnych współtwórców z całego świata. Jego elastyczność pozwala na łatwą integrację z innymi bibliotekami lub istniejącymi projektami, a także jest świetny do tworzenia zaawansowanych aplikacji jednostronnicowych (SPA - Single Page Applications). Swietnie nadaje się do tego projektu, gdyż jest to niewielka aplikacja, która będzie korzystać z wielu bibliotek i narzędzi. W kolejnych podrozdziałach opiszę dlaczego Vue.js jest dobrym wyborem do tego projektu.

Architektura i Komponenty

Architektura Vue opiera się na systemie komponentów. Komponenty w Vue są blokami wielokrotnego użytku z własnym stanem, metodami i szablonami. Dzięki temu łatwo jest tworzyć interfejsy składające się z mniejszych, niezależnych części, co znacznie ułatwia zarządzanie, utrzymanie i czytelność kodu.

Reaktywność i Dwukierunkowe Wiązanie Danych

Jedną z kluczowych cech Vue.js jest jego reaktywny system, który zapewnia automatyczną aktualizację interfejsu użytkownika w odpowiedzi na zmiany stanu aplikacji. Framework ten używa systemu dwukierunkowego wiązania danych (two-way data binding), co oznacza, że zmiany w modelu danych od razu są odzwierciedlane w widoku i na odwrót.

Deklaratywne Renderowanie

Vue.js wykorzystuje deklaratywne renderowanie. Oznacza to, że developer określa, jakie dane powinny być wyświetlane, a framework zajmuje się ich aktualizacją w widoku. To sprawia, że kod jest bardziej zrozumiały i łatwiejszy w utrzymaniu.

Virtual DOM

Vue korzysta z koncepcji Virtual DOM, co pozwala na efektywną aktualizację widoku bez konieczności odświeżania całej strony. Jest to znacznie szybsze niż tradycyjne manipulowanie DOM, ponieważ zmiany są najpierw aplikowane do Virtual DOM, a następnie, w optymalny sposób, przekazywane do rzeczywistego DOM.

Ekosystem i Społeczność

Vue ma rozbudowany ekosystem, w skład którego wchodzą takie narzędzia jak Vue Router (do zarządzania nawigacją w aplikacji) i Vuex (do zarządzania stanem aplikacji). Dodatkowo, wsparcie społeczności i dostępność zasobów edukacyjnych, takich jak dokumentacja, poradniki i fora dyskusyjne, sprawiają, że nauka i praca z Vue.js jest dostępna i przyjemna.

5.4 Django: Backend

Do implementacji Backendu zdecydowałem się na użycie frameworku **Django**.

Django to wysokopoziomowy framework webowy napisany w Pythonie, który umożliwia szybkie tworzenie bezpiecznych i łatwych w utrzymaniu stron internetowych. Został zaprojektowany z myślą o uproszczeniu zadań związanych z tworzeniem aplikacji internetowych, dzięki czemu deweloperzy mogą skupić się po prostu na pisaniu aplikacji. W kolejnych podrozdziałach opiszę dlaczego Django jest dobrym wyborem do tego projektu.

Python

Django jest napisane w języku Python, który jest jednym z najpopularniejszych i najbardziej lubianych języków programowania. W języku tym zostały zaimplementowane algorytmy DWSR i ESRGAN, więc wykorzystanie Django pozwoli na łatwą integrację tych algorytmów z aplikacją.

Architektura Wzorca Projektowego MTV

Django wykorzystuje wzorzec projektowy "Model-Template-View" (MTV), który jest podobny do popularnego wzorca MVC. W tym podejściu:

- **Model** odpowiada za strukturę danych oraz ich walidację.
- **Template** odpowiada za prezentację danych.
- **View** odpowiada za logikę aplikacji, odbierając żądania od użytkownika i zwracając odpowiednie odpowiedzi.

W tego typu projektach warto stosować wzorce projektowe, ponieważ ułatwiają one zarządzanie kodem i zwiększały jego czytelność. Ponadto, stosowanie wzorców projektowych jest dobrym zwyczajem, który pozwala na łatwiejsze utrzymanie aplikacji w przyszłości.

ORM (Object-Relational Mapping)

Django zawiera wbudowany ORM, który pozwala na interakcję z bazami danych za pomocą kodu Python, zamiast pisać surowe zapytania SQL. ORM przekształca tabele bazy danych w klasy Pythona, co ułatwia manipulowanie danymi i sprawia, że kod jest bardziej czytelny, prostszy w zarządzaniu i łatwiejszy w utrzymaniu.

Wbudowane Funkcje

Django oferuje wiele wbudowanych funkcji, takich jak system uwierzytelniania użytkowników, mapowanie URL na widoki, mechanizm szablonów, system administrowania itd. Dzięki temu można szybko rozpocząć pracę nad projektem, mając już na starcie zestaw potrzebnych narzędzi.

Bezpieczeństwo

Bezpieczeństwo jest jedną z głównych zalet Django. Framework ten automatycznie chroni aplikacje przed wieloma powszechnymi zagrożeniami, takimi jak SQL Injection, Cross-site Scripting (XSS), Cross-site Request Forgery (CSRF) i Clickjacking, dzięki czemu deweloperzy mogą skupić się na budowaniu aplikacji, nie martwiąc się o podstawowe kwestie bezpieczeństwa.

Skalowalność

Django jest skalowalny i może obsługiwać zarówno małe, jak i duże projekty. Ponadto, framework ten wspiera koncepcję wersjonowania API, co jest kluczowe przy rozwijaniu i utrzymywaniu dużych aplikacji.

Wsparcie Społeczności i Dokumentacji

Podobnie jak w przypadku Vue.js, Django ma silną i aktywną społeczność. Dzięki temu deweloperzy mają dostęp do bogatej dokumentacji, licznych zasobów edukacyjnych i gotowych rozwiązań, co ułatwia naukę i rozwiązywanie problemów.

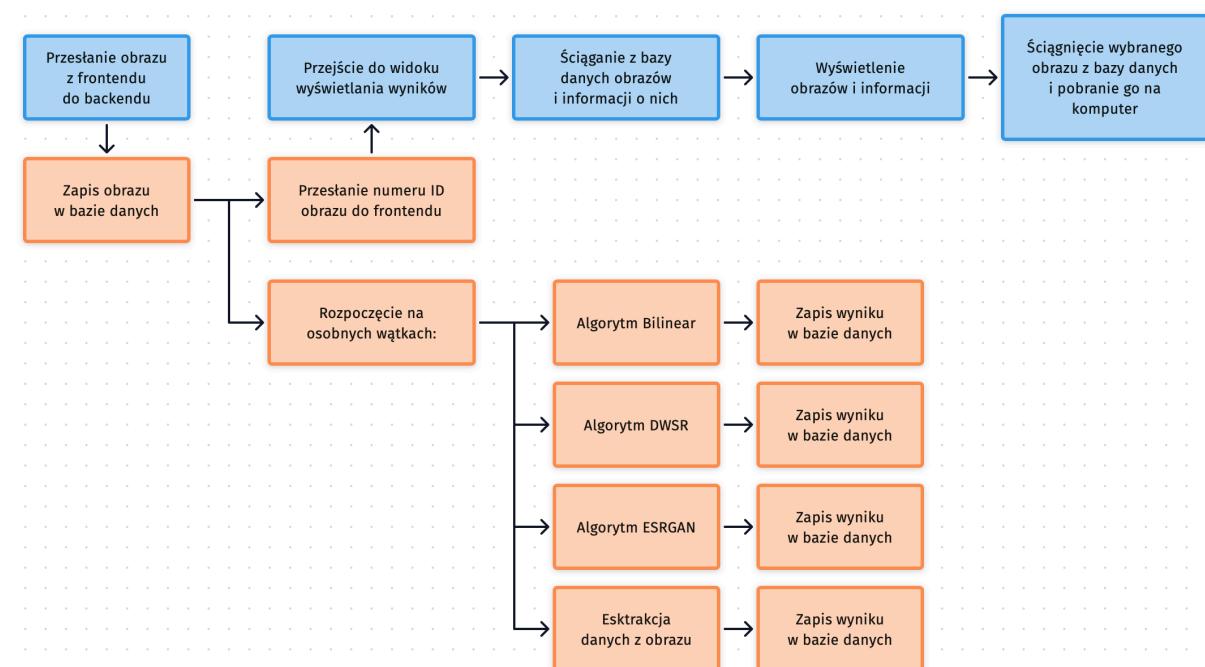
5.5 Implementacja aplikacji

Po wyborze streszczenia technologicznego kolejnym krokiem jest skupienie się na implementacji rozwiązań. W tym rozdziale opiszę jakie decyzje podjąłem przy pisaniu kodu aplikacji, jak wygląda jej struktura i jakie problemy napotkałem podczas implementacji.

Struktura aplikacji

Aplikacja składa się z dwóch części - Frontendu i Backendu. Przy tworzeniu takiego projektu warto zadbać o to, żeby każda część była od siebie niezależna i żeby komunikacja między nimi była jak najmniej skomplikowana.

W tym miejscu wracamy do diagramu przepływu użytkownika [Rys 3], jak na nim widać użytkownik nie może wykonać zbyt wiele akcji, struktura aplikacji jest liniowa. Na podstawie diagramu przepływu użytkownika można stworzyć schemat blokowy aplikacji [Rys 8], który pozwoli zrozumieć zachowanie programu.



Rys 8. Schemat blokowy aplikacji (kolor niebieski - Frontend, pomarańczowy - Backend)

W pierwszej kolejności użytkownik wysyła obraz do serwera Backend, który zapisuje go w bazie danych. Następnie serwer zleca wykonanie algorytmów na osobnych wątkach, o czym opowiem w dalszej części rozdziału [5.6]. Gdy algorytmy rozpoczną pracę, serwer zwraca do Frontendu informację o tym że operacja zapisu się powiodła i podaje numer ID obrazu.

Frontend zmienia widok na ten z wynikami i wysyła zapytanie do Backenu o obraz oryginalny i przetworzone. Następnie jeśli serwer zwróci obrazy, Frontend je wyświetla. W przeciwnym wypadku próbuje je pozyskać ponownie aż do skutku. Dzieje się tak, dlatego że zadanie super-rozdzielczości jest czasochłonne i czasem może zająć kilka sekund a w innych wypadkach nawet kilka minut, wszystko w zależności od rozdzielczości obrazów. W tym czasie użytkownik może porównywać uzyskane wyniki i wybrać najlepszy.

Gdy użytkownik wybierze obraz, może go pobrać na swój komputer. Wtedy Frontend wysyła zapytanie do serwera o obraz w pełnej rozdzielczości, a serwer zwraca obraz, który przeglądarka automatycznie pobiera.

Architektura bazy danych

Baza danych w aplikacji jest bardzo prosta, przy przesłaniu każdego zdjęcia w bazie tworzone jest pole `Image`, które przechowuje informacje o obrazie oraz jego przetworzonych wersjach. W tabeli 5.1 przedstawiam jedyną tabelę w bazie danych, która przechowuje informacje o obrazach.

Image		
Pole	Typ	Opis
image	ImageField	Przesłany obraz.
bilinear_image	ImageField	Obraz powiększony algorymem Bilinear.
dwsr_image	ImageField	Obraz powiększony algorymem DWSR.
esrgan_image	ImageField	Obraz powiększony algorymem ESRGAN.
original_height	PositiveIntegerField	Wysokość oryginalnego obrazu.
original_width	PositiveIntegerField	Szerokość oryginalnego obrazu.
dominant_colors	TextField	Pole tekstowe z listą dominujących kolorów.

Tabela 5.1: Struktura bazy danych - `Image`.

Jak widać w bazie danych przechowywane są obrazy w formacie `ImageField`, który jest dostarczany przez bibliotekę Django. Jest to pole, które przechowuje ścieżkę do pliku na dysku serwera. Zapisujemy również informacje o oryginalnych wymiarach obrazu, które są wyświetlane użytkownikowi przez Frontend.

Dodatkowo w bazie danych przechowujemy listę dominujących kolorów, które są wykrywane przez algorytm K-średnie *K-means*, o którym opowiem w kolejnym rozdziale [5.6]. Jest to lista kolorów w formacie HEX, które wykorzystuje Frontend do wyświetlenia kolorowych gradientów tła.

5.6 Integracja algorytmów super-rozdzielczości

Gdy obraz jest już zapisany w bazie danych, serwer Backend zleca wykonanie algorytmów super-rozdzielczości na osobnych wątkach [5.1]. W tym rozdziale opiszę w jaki sposób zostały zaimplementowane algorytmy w aplikacji.

```

1 def upload_image(request):
2     try:
3         form = Image(image=request.FILES['image'])
4         form.save()
5
6         input_image_path = form.image.path
7
8         thread1 = threading.Thread(target = run_bilinear,
9                                     args = (input_image_path, 4, form))
10        thread2 = threading.Thread(target = run_dwsr,
11                                   args = (input_image_path, 4, form))
12        thread3 = threading.Thread(target = run_esrgan,
13                                   args = (input_image_path, form))
14        thread4 = threading.Thread(target = extract_image_info,
15                                   args = (input_image_path, form))
16
17        thread1.start()
18        thread2.start()
19        thread3.start()
20        thread4.start()
21
22        image = Image.objects.latest('id') # Gets the latest entry
23
24        return JsonResponse({'message': 'Image uploaded, processing started',
25                             'image_id': image.id})
26    except Exception as e:
27        return JsonResponse({'error': str(e)}, status=400)

```

Listing 5.1: Obsługa zapisu i przetwarzania obrazów.

Stworzone zostały cztery funkcje - *run_bilinear*, *run_dwsr*, *run_esrgan* oraz *extract_image_info*. Pierwsze trzy z nich odpowiadają za uruchomienie algorytmów super-rozdzielczości, a ostatnia za wydobycie informacji o obrazie.

Algorytm Interpolacji Dwuliniowej

Pierwszy z algorytmów, który został zaimplementowany to algorytm Interpolacji Dwuliniowej omawiany w rozdziale Podstawy Teoretyczne w sekcji 2.2. Jest to najmniej skomplikowany algorytm, więc implementacja jego nie była trudna [5.2].

```

1 def run_bilinear(input_image_path, scale, image_instance):
2     img = cv2.imread(input_image_path, cv2.IMREAD_COLOR)
3
4     height, width = img.shape[:2]
5     new_width, new_height = width * scale, height * scale
6
7     # Resize the image using bilinear interpolation
8     output = cv2.resize(img, (new_width, new_height), interpolation=cv2.
9                         INTER_LINEAR)
10
11    save_output_image(output, input_image_path, image_instance)

```

Listing 5.2: Implementacja algorytmu Bilinear.

W pierwszej kolejności wczytujemy obraz do powiększenia, następnie pobieramy jego wyymiary i mnożymy je przez skalę powiększenia otrzymując nową wielkość obrazu. W kolejnym kroku wykorzystujemy funkcję `cv2.resize` z biblioteki OpenCV2, która pozwala na zmianę rozmiaru obrazu. W tym miejscu wykorzystujemy parametr `interpolation=cv2.INTER_LINEAR`, który odpowiada za wybór algorytmu interpolacji. W tym wypadku wybraliśmy algorytm Interpolacji Dwuliniowej. Na koniec zapisujemy obraz w bazie danych.

Algorytm DWSR

Kolejnym algorymem jest algorytm DWSR, który został opisany w rozdziale ???. Implementacja tego algorytmu jest bardziej skomplikowana, ponieważ wymaga on zainstalowania modelu, który jest wykorzystywany do przetwarzania obrazów, oraz wymagała przepisania skryptów napisanych w Matlabie na język Python [5.3].

```

1 def run_dwsr(input_image_path, scale, image_instance):
2     enlarged_lr_dir, sr_lum_dir, output_dir = create_output_dir()
3
4     fileName = os.path.basename(input_image_path)
5
6     # 1. Generate enlarged LR images
7     generate_enlarged_lr(input_image_path, enlarged_lr_dir, scale)
8
9     # 2. Process image with DWSR
10    process_image(enlarged_lr_dir + '/' + fileName, sr_lum_dir, scale)
11
12    # 3. Generate color SR
13    final_img_path = generate_color_sr(input_image_path, sr_lum_dir,
14                                         output_dir, scale)
15
16    save_output_image(final_img_path, input_image_path, image_instance)
```

Listing 5.3: Implementacja algorytmu DWSR.

Algorytm DWSR składa się z trzech etapów, najpierw obraz jest powiększany metodą Bicubic o podaną skalę analogicznie do algorytmu Bilinear, lecz tym razem z parametrem `interpolation=cv2.INTER_CUBIC`. Obraz konwertowany jest do skali szarości. Następnie obraz jest przetwarzany przez model DWSR [5.4], który zwraca obraz w skali szarości. Na koniec obraz jest konwertowany do RGB i zapisywany w bazie danych.

```

1 def process_image(input_image_path, output_dir, scale):
2     session = initialize_session()
3
4     coeffs = dwt2_image(input_image_path)
5     model_input_data = construct_model_input(coeffs)
6
7     model_output_data = session.run([model_output],
8                                     feed_dict={model_input_data})
9
10    super_res_image = idwt2_image(model_output_data)
11
12    output_image_path = os.path.join(output_dir,
13                                    ntpath.basename(input_image_path))
14    cv2.imwrite(output_image_path, super_res_image)
15    session.close()
```

Listing 5.4: Przetwarzanie przez model DWSR.

Przetwarzanie obrazu przez model DWSR rozpoczyna się od załadowania wytrenowanego modelu i obrazu powiększonego przez Bicubic. Następnie obraz jest dekomponowany na współczynniki falkowe (z użyciem biblioteki PyWavelets), które są przekształcane do formatu odpowiedniego dla modelu. Model prognozuje ulepszone detale tych współczynników. Te przewidziane detale są następnie łączone z oryginalnymi współczynnikami, tworząc obraz o wyższej rozdzielczości. Wynikowy obraz jest zapisywany w tymczasowej lokalizacji a w kolejnym kroku jest on konwertowany ze skali szarości do RGB i zapisywany w bazie danych.

Algorytm ESRGAN

Ostatnim algorytmem Super-Rzdzielczości implementowanym w aplikacji jest algorytm ESRGAN, który został opisany w rozdziale ???. Implementacja tego algorytmu była prosta, ponieważ wymagała jedynie zapisu modelu, który jest wykorzystywany do przetwarzania obrazów, oraz jako że model jest już napisany w języku Python, nie było potrzeby przepisywania go na inny język [5.5]. Do tej funkcji nie podajemy skali powiększenia, ponieważ model ESRGAN jest w stanie powiększyć obraz czterokrotnie.

```
1 def run_esrgan(input_image_path, image_instance):
2     model, device = initialize_esrgan_model()
3     output = process_image_with_esrgan(model, device, input_image_path)
4     save_output_image(output, input_image_path, image_instance)
```

Listing 5.5: Implementacja algorytmu ESRGAN.

W pierwszej kolejności wczytujemy model ESRGAN, następnie przetwarzamy obraz przez model [5.6] i zapisujemy go w bazie danych.

```
1 def process_image_with_esrgan(model, device, input_image_path):
2     img_LR = read_img(input_image_path, device)
3
4     with torch.no_grad():
5         output = model(img_LR).data.squeeze().float().cpu().clamp_(0, 1).numpy()
6
7     output = np.transpose(output[[2, 1, 0], :, :], (1, 2, 0))
8     output = (output * 255.0).round()
9
10    return output
```

Listing 5.6: Przetwarzanie przez model ESRGAN.

Algorytm K-średnich

Algorytm K-średnich został opisany w rozdziale Podstawy Teoretyczne w sekcji 2.2. Jest to algorytm, który wykorzystujemy do wydobycia informacji o obrazie, a dokładniej do wydobycia listy dominujących kolorów. Implementacja tego algorytmu jest bardzo prosta, gdyż wykorzystałem bibliotekę *scikit-learn*, która zawiera w sobie algorytm [5.7].

```
1 def extract_dominant_colors(input_image_path, image_instance):
2     with PILImage.open(input_image_path) as img:
3         pixels = np.array(img.getdata())      # Reshape the image data for k-means
4         pixels = pixels.reshape(-1, 3)
5
```

```
6      kmeans = KMeans(n_clusters=5)      # 5 dominant colors
7      kmeans.fit(pixels)
8      dominant_colors = kmeans.cluster_centers_
9
10     # Convert dominant colors to HEX values
11     dominant_colors_list = [rgb_to_hex(tuple(map(int, color)))
12                           for color in dominant_colors]
13
14     image_instance.set_dominant_colors(dominant_colors_list)
15     image_instance.save()
```

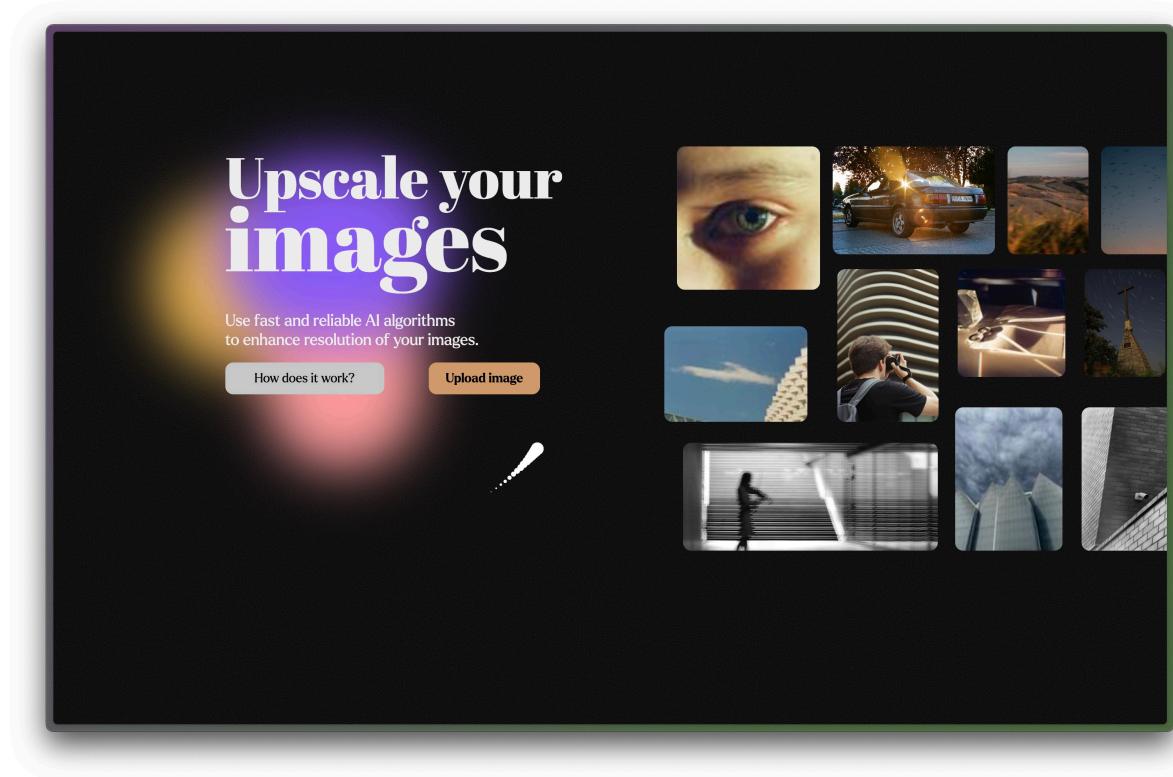
Listing 5.7: Implementacja algorytmu K-srednich.

W pierwszej kolejności wczytujemy obraz, następnie przekształcamy go do formatu RGB i przekształcamy do formatu, który może być wykorzystany przez algorytm K-srednich. Następnie wykorzystujemy funkcję *KMeans* z biblioteki *scikit-learn*, która zwraca nam listę dominujących kolorów. Na koniec konwertujemy kolory do formatu HEX i zapisujemy je w bazie danych.

5.7 Implementacja interfejsu użytkownika

Interfejs użytkownika zawiera dwa widoki - widok ze stroną główną, przez który można wysłać zdjęcie do powiększenia rozdzielczości oraz widok z wynikami, w którym można porównać obraz oryginalny z przetworzonymi wersjami. W tym rozdziale opiszę jak zostały zaimplementowane te funkcjonalności.

Widok ze stroną główną



Rys 9. Widok strony głównej aplikacji

Widok ten wydaje się drobny i prosty, lecz w rzeczywistości jest rozbudowany. W idei skupić się ma na tym co najważniejsze, czyli na przesłaniu obrazu do powiększenia. Mam sporo pomysłów na ot jak go rozbudować na bazie istniejących interakcji i jakie dodatkowe funkcjonalności można by dodać, ale o tym w rozdziale 5.8.

Widok ten składa się z dwóch części - z tytułu aplikacji oraz przycisków i z obrazów przykładowych [Rys 9].

W aplikacji znajdziemy dwie czcionki: *Abrial Fatface* z Google Fonts i *Larken* z Adobe Fonts. Pierwsza z nich jest używana do tytułu aplikacji, a druga do reszty tekstu, przycisków itp. Niżej na stronie znajdują się dwa przyciski, jeden do przesłania obrazu, a drugi do wyświetlenia informacji o aplikacji. Przycisk do przesłania obrazu jest wyróżniony kolorem, więc od razu zwraca na siebie uwagę, zaś przycisk z informacjami jest w kolorze szarym, więc jest mniej widoczny. Póki co przycisk z informacjami nie ma żadnej funkcjonalności, ale o tym w rozdziale 5.8.

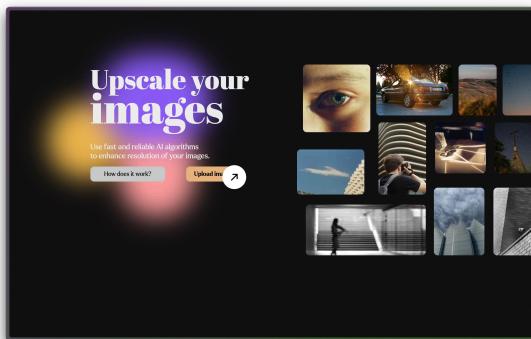
Prawa część ekranu składa się z obrazów przykładowych, które mają zachęcić użytkownika do przesłania obrazu. Część z nich jest wysokiej rozdzielczości, a część niższej, aby pokazać różnicę między obrazem oryginalnym a przetworzonym przez aplikację. Obrazy te wyłamują się poza siatkę strony, aby nadać stronie dynamiki.

Tło tego widoku, tak jak wspomniałem wcześniej, nawiązuje do szumu na zdjęciach analogowych, dodatkowo składa się z animowanych kształtów które znajdują się za warstwą szumu. Animacja tła jest subtelna i składają się na nią trzy kształty, które poruszają się pomiędzy wyszczególnionymi za pomocą *@keyframes* punktami. W ten sposób uzyskujemy efekt, że kształty poruszają się w sposób płynny i nie przewidywalny.

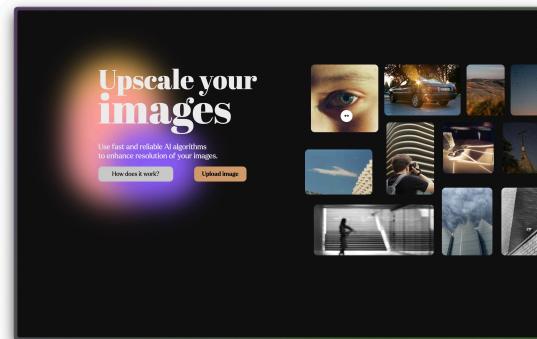
Kolejnym istotnym elementem na stronie jest kurSOR. Ma on wbudowane funkcjonalności, które dają szerokie pole do popisu w kwestii interakcji z użytkownikiem. KurSOR zmienia swój wygląd w zależności od tego nad jakim elementem znajduje się wskaźnik myszy. Na przykład gdy wskaźnik znajduje się nad przyciskiem, kurSOR powiększa się i wyświetla się wewnątrz niego ikona. KurSOR ma kilka stanów do interakcji nad obiektami oznaczonymi klasą *interactable*:

- **Init** - stan początkowy, gdy wskaźnik znajduje się na ekranie, skaluje się od 0 do 1.2 i po osiągnięciu 1.2 zmienia się na stan **Default**.
- **Default** - stan domyślny, gdy wskaźnik nie znajduje się nad żadnym specjalnym elementem. Warto w tym miejscu dodać, że kurSOR tak naprawdę składa się z piętnastu elementów, które ciągną cię za miejscem w którym znajduje się wskaźnik. Dzięki temu za każdym ruchem pozostaje ślad, który uprzyjemnia interakcje z aplikacją [Rys 9].
- **Hover** - stan nad elementem, gdy wskaźnik znajduje się nad elementem oznaczonym klasą *interactable*, kurSOR powiększa się i wyświetla się wewnątrz niego ikona w zależności od tego nad czym się znajduje [Rys 10, 11, 12].
- **DragOver** - stan gdy do aplikacji chcemy przeciągnąć obraz, kurSOR zmienia swój kolor, skaluje się na dużo większy i wyświetla się wewnątrz niego ikona [Rys 13].

Ikony z których korzystam są z bibliotek *Font Awesome*, oraz *Heroicons* i są one wykorzystywane w zależności od tego nad jakim elementem znajduje się wskaźnik myszy. Stan **DragOver** bardzo uprzyjemnia korzystanie z aplikacji, gdyż kiedy użytkownik tylko przeciągnie obraz nad okno przeglądarki, kurSOR podąża za nim, więc użytkownik może upuścić obraz w dowolnym miejscu na stronie a obraz się prześle.



Rys 10. Kursor w stanie Hover (link)



Rys 11. Kursor w stanie Hover (obraz)



Rys 12. Kursor w stanie Hover (przycisk)



Rys 13. Kursor w stanie DragOver

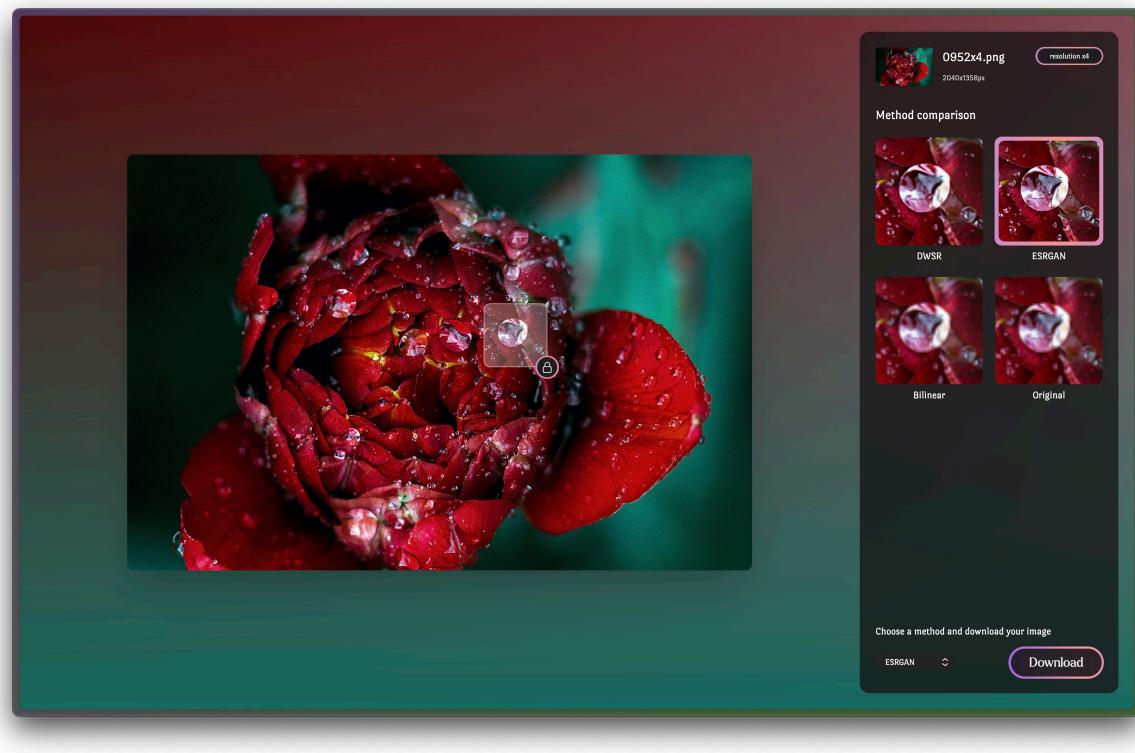
Gdy obraz zostaje upuszczony, uruchamiana jest funkcja, która odpowiada za sprawdzenie ile plików zostało przesłanych, czy jest to obraz jest wspieranego typu i jeśli wszystko się zgadza wysyła obraz do serwera Backend. Wtedy oczekuje odpowiedzi z Backendu, jeśli wszystko się powiedzie, czyli obraz zostanie zapisany w bazie danych, to Frontend otrzymuje ID obrazka i router zmienia adres na "upload-image/:id".

Widok z porównaniem wyników powiększenia obrazów

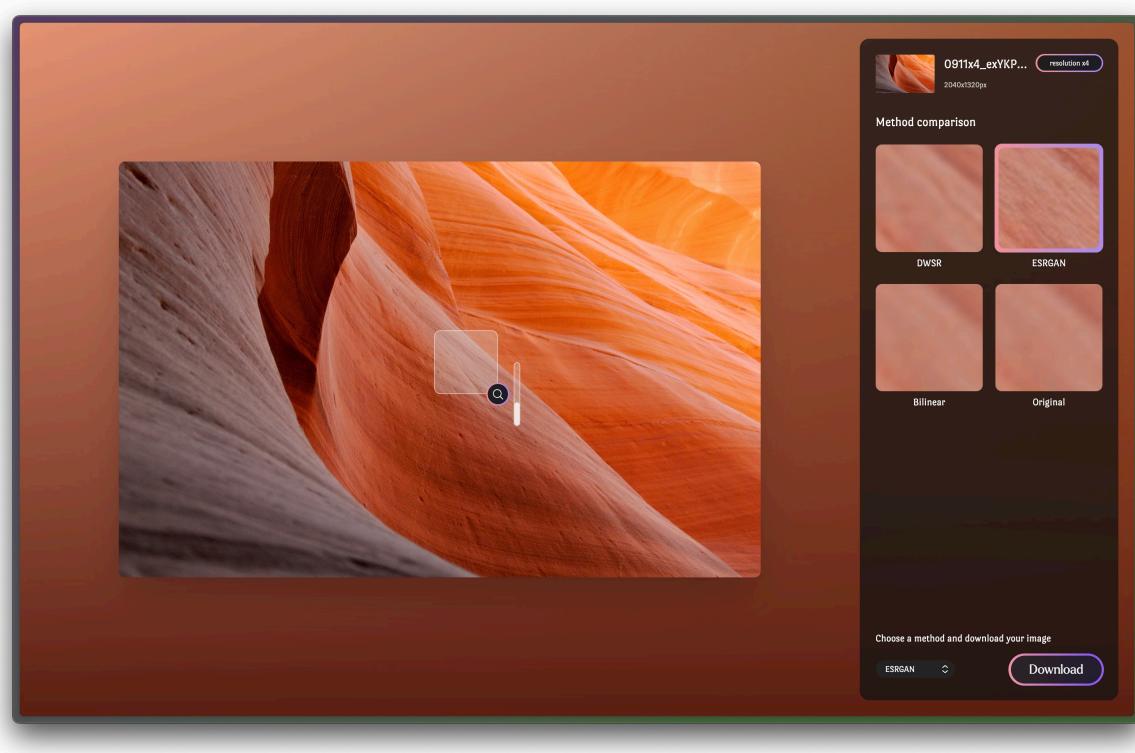
Drugi widok w aplikacji jest dużo ważniejszy i bardziej rozbudowany, gdyż odpowiada za porównanie wyników powiększenia obrazów. Widok ten składa się z dwóch części - z wybranego obrazu który jest wyświetlony, oraz z menu bocznego [Rys 14].

Największą część ekranu zajmuje aktualnie wyświetlany obraz wynikowy, ale tak naprawdę najważniejsze w tym widoku jest menu boczne. Widać na nim podstawowe informacje o obrazie takie jak miniatura, nazwa, nowa (powiększona) rozdzielcość, przycisk do pobrania obrazu oraz co najważniejsze - kafelki z przetworzonymi obrazami. Kafelki te są interaktywne, więc po kliknięciu w nie, obraz jest wyświetlany w głównym obszarze widoku, ale co jeszcze ważniejsze - przybliżony na kafelku fragment obrazu jest fragmentem, który jest pod kursorem myszy.

Kursor w tym widoku wizualnie jest bardziej złożony, zawiera ikonę sugerującą że działa jak lupa, która wyświetla się w stanie domyślnym. Oprócz tego w lewą górną stronę rozciąga się "szkło" lupy które jest animowane i porusza się wraz z kursem. Dodatkowo zaimplementowałem obsługę scrolla oraz przycisków "+" i "-", która pozwala na przybliżenie i oddalenie obrazu lupy, co sprawia że małe narzędzie jest intuicyjne [Rys 15]. Pasek postępu widoczny jest jedynie gdy użytkownik przybliża lub oddala obraz.



Rys 14. Widok z porównaniem wyników algorytmów



Rys 15. Widok z porównaniem wyników algorytmów

Kursor ma też opcję blokady w miejscu, która jest aktywowana po przytrzymaniu klawisza spacji lub po kliknięciu myszą. Sprawia ona że kursor nie porusza się wraz z myszą, więc użytkownik może swobodnie poruszać się po aplikacji. W stanie blokady na kursorze wyświetla się ikona z kłódką, która sugeruje blokadę kursora.

Tłem widoku jest gradient kolorów dominujących na obrazie. Kolory te są wyliczane przez algorytm K-średnich na Backendzie i zapisywane w bazie danych, dzięki temu Frontend może je wykorzystać.

```

1 const generateGradientStyles = (colors: string[]) => {
2   const angles = ['45deg', '135deg', '-45deg', '90deg', '0deg'];
3   return colors.map((color: any, index: number) => ({
4     backgroundImage: `linear-gradient(${angles[index]}, ${color} 0%, transparent 100%)`,
5     position: 'absolute', top: '0', right: '0', bottom: '0', left: '0',
6   }));
7 };

```

Listing 5.8: Wyznaczanie gradientów z kolorów dominujących.

```

1 <div v-for="(style, index) in gradientStyles" :key="index" :style="style"/>

```

Listing 5.9: Rysowanie kolorów dominujących (HTML).

Po odczytaniu kolorów wywoływana jest funkcja *generateGradientStyles*, która zwraca tablicę obiektów, które są wykorzystywane w szablonie HTML. W ten sposób uzyskujemy gradienty, które są wyświetlane w tle widoku.

Kolejną ważną rzeczą w interfejsie użytkownika jest działanie szkła powiększającego, czyli jeden z głównych elementów, które mają na celu ułatwić użytkownikowi porównanie wyników.

```

1 const updateMagnifyLocation = (e: MouseEvent) => {
2   // Clamp the cursor's coordinates within the image's bounds
3   const effectiveX:number = Math.min(Math.max(e.clientX, image.left
4                                             + magnifySize), image.right);
5   const effectiveY:number = Math.min(Math.max(e.clientY, image.top
6                                             + magnifySize), image.bottom);
7   cursorPosXInImg = effectiveX;
8   cursorPosYInImg = effectiveY;
9
10  // Calculate the position as a percentage of the image's dimensions
11  const percentX:number = (effectiveX-magnifySize/2-image.left)/image.width;
12  const percentY:number = (effectiveY-magnifySize/2-image.top)/image.height;
13
14  if (aspectRatio < 1) {
15    magnifyWindowX = (percentX * scaleValue * aspectRatio)
16                  -0.5 * scaleValue * aspectRatio;
17    magnifyWindowY = (percentY * scaleValue)
18                  -0.5 * scaleValue;
19  } else {
20    magnifyWindowX = (percentX * scaleValue)
21                  -0.5 * scaleValue;
22    magnifyWindowY = (percentY * scaleValue / aspectRatio)
23                  -0.5 * scaleValue / aspectRatio;
24  }
25}

```

Listing 5.10: Implementacja szkła powiększającego (TypeScript).

Jeśli kurSOR nie jest w stanie blokady przy ruchu myszy wywoływana jest funkcja *updateMagnifyLocation*, która odpowiada za wyliczenie pozycji szkła powiększającego. W pierwszej kolejności obliczamy pozycję kursora wewnątrz obrazu. Następnie przekształcamy tę wartość na procentową pozycję wewnątrz obrazu. Na koniec obliczamy pozycję szkła powiększającego w zależności od proporcji obrazu i skali powiększenia.

```

1 <div class="magnifying_glass">
2   <div v-for="type in imageTypes"
3     :key="type"
4     @click="updatePreview(type)">
5     <div class="w-full aspect-square rounded-lg z-[10] highlight"
6       :class="{ 'selected-image': type == selectedAlgorithm }">
7       <div class="miniature-view">
8         
12       </div>
13     </div>
14   </div>

```

Listing 5.11: Implementacja szkła powiększającego (HTML).

W następnej kolejności wartości *magnifyWindowX* i *magnifyWindowY* są używane przez szablon HTML, gdzie są wykorzystywane do wyświetlenia szkła powiększającego. W tym miejscu wykorzystujemy *v-for* do wyświetlenia wszystkich obrazów, które są przetworzone przez algorytmy. Każdy obraz jest wyświetlany w osobnym kafelku, który jest interaktywny. Po kliknięciu w kafelek, obraz jest wybierany jako algorytm do pobrania, wyświetlany w głównym obszarze widoku, a kafelek jest podświetlany.

Aby pobrać wybrany obraz wystarczy nacisnąć przycisk "Download" w menu bocznym, co wywołuje funkcję *downloadImage*, która pobiera obraz z serwera Backend.

```

1 async downloadImage() {
2   try {
3     const response = await fetch(this.imageUrls[this.selectedAlgorithm]);
4     if (!response.ok) throw new Error('Failed to fetch the image.');
5     const blob = await response.blob();
6     const url = window.URL.createObjectURL(blob);
7     const link = document.createElement('a');
8
9     link.href = url;
10    link.download = this.imageTitle;
11    document.body.appendChild(link);
12    link.click();
13
14    window.URL.revokeObjectURL(url);
15    document.body.removeChild(link);
16  } catch (error) {
17    console.error('Error downloading the image:', error);
18  }
19}

```

Listing 5.12: Pobieranie obrazu.

W pierwszej kolejności pobieramy obraz z serwera Backend, następnie tworzymy link do obrazu i nadajemy mu nazwę. Na koniec tworzymy element *<a>* i nadajemy mu atrybuty *href* i *download* oraz klikamy w ten link.

5.8 Plany na przyszłość

W stanie aktualnym aplikacja spełnia wszystkie założenia koncepcyjne. Użytkownik może przesłać obraz do powiększenia, a następnie porównać wyniki. W tym rozdziale opiszę jakie funkcjonalności mam w planie dodać do tego narzędzia w przyszłości.

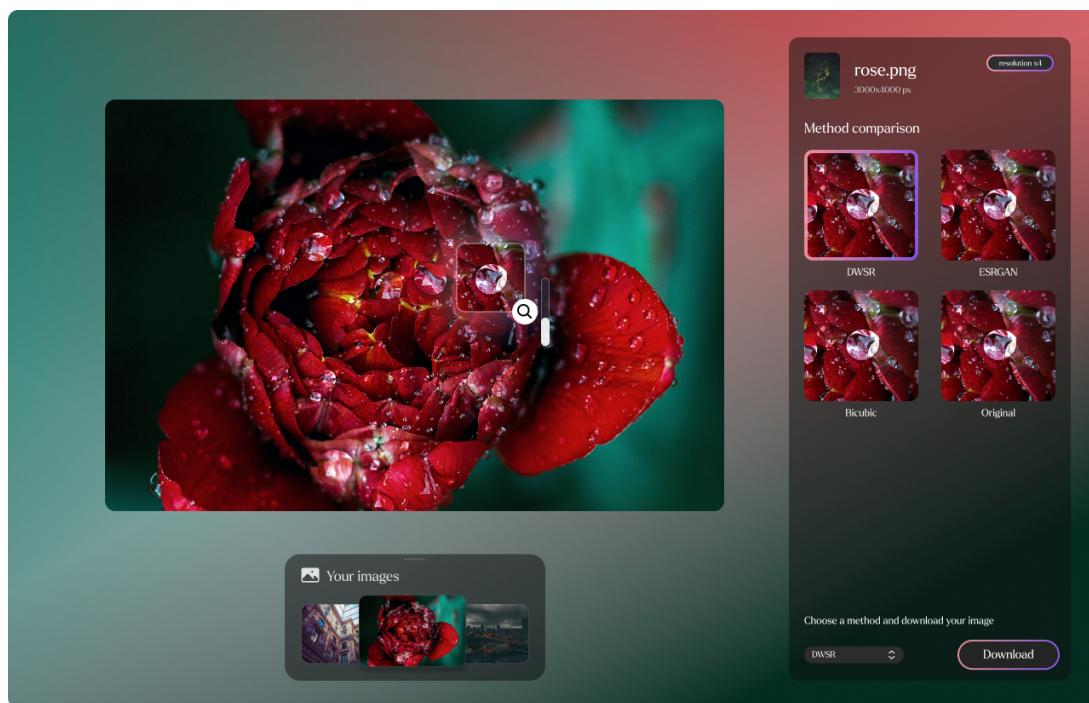
Usprawnienia widoku głównego

Do widoku głównego chciałbym dodać kilka funkcjonalności, które przedstawią działanie aplikacji i zachętą użytkownika do skorzystania z serwisu.

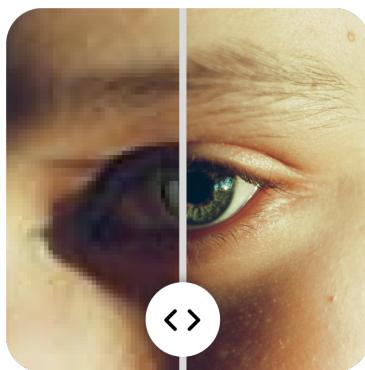
- Pierwszą i najważniejszą kwestią będzie dodanie do widoku informacji o tym jak działa aplikacja i zaimplementowane w niej algorytmy. Odnieść się do tego przez kogo zostały te metody opracowane, jakie są ich zalety i wady.
- Drugim usprawnieniem będzie dodanie interakcji z obrazami przykładowymi i możliwość sprawdzenia jak działa aplikacja na ich podstawie. Użytkownik będzie mógł przesunąć obraz myszką, aby zobaczyć jak działa szkło powiększające lub przesunąć suwakiem nad obrazem żeby porównać "przed i po" [Rys 17].
- Kolejną dużą zmianą będzie możliwość dodania do aplikacji kilku obrazów jednocześnie w celu powiększenia rozdzielczości. Ta zmiana wymaga modyfikacji w Backendzie, ponieważ obecnie aplikacja obsługuje przyjęcie i przechowanie tylko jednego obrazu na raz.

Usprawnienia widoku z wynikami

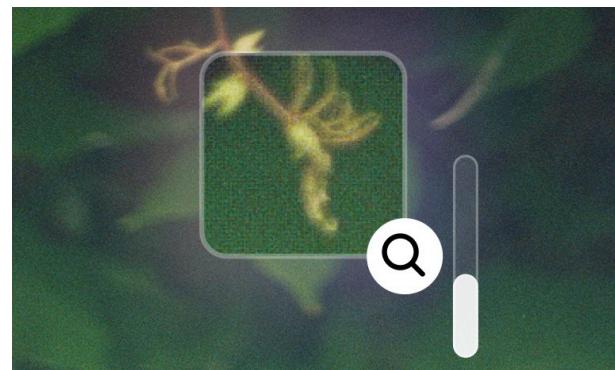
- Przede wszystkim chciałbym dodać możliwość wyświetlenia kilku obrazów po przetworzeniu. Obecnie widok wyświetla tylko jeden obraz, ale możliwość przedstawienia użytkownikowi wielu obrazów byłaby użyteczna i praktyczna. Możliwa implementacja jak na obrazie 16, lub integracja tej części interfejsu w pasku bocznym.
- Koleijną zmianą będzie lekka zmiana UI związanego z kursorem i szkłem powiększającym. Chciałbym żeby design całej aplikacji był spójny, więc dobrym pomysłem byłaby implementacja mechanizmów kurSORA z głównego ekranu do tego widoku. Dodatkowo myślę, że szkło powiększające powinno wyświetlać przybliżony fragment obrazu przy kurSORze a nie wyłącznie na pasku bocznym, żeby to bardziej intuicyjne dla użytkownika [Rys 18].
- Kolejną kwestią będzie optymalizacja wyświetlania obrazów. Obecnie obrazy są wyświetlane w pełnej rozdzielczości i w formacie oryginalnym, co jest niepotrzebne i powoduje spowolnienie działania aplikacji. W przyszłości chciałbym dodać mechanizm, który będzie wyświetlał obrazy w zależności od rozdzielczości ekranu użytkownika i w formacie webp, który jest mniej zasobozerny.
- Ostatnią zmianą będzie ulepszenie animacji ładowania tak żeby użytkownik wiedział, że aplikacja pracuje nad przetworzeniem obrazu. Obecnie animacja jest bardzo prosta i nie daje użytkownikowi żadnej informacji o tym co się dzieje.



Rys 16. Koncepcja widoku prezentacji wyników z obsługą wielu obrazów jednocześnie



Rys 17. Suwak pokazujący obraz przed i po



Rys 18. Koncepcja szkła powiększającego

Zmiany obejmujące całą aplikację

W tym dziale przedstawię jakie zmiany obejmujące cały projekt chciałbym wprowadzić i dlaczego.

- Pierwszą i najważniejszą zmianą ogólną w aplikacji będzie lepsza obsługa błędów. Póki co gdy coś jest nie tak przeglądarka wyświetla komunikat o błędzie, który jest czytelny dla mnie, ale nie dla użytkownika. W przyszłości chciałbym dodać własne komunikaty o błędach, które będą bardziej przyjazne dla użytkownika.
- Jedną z istotniejszych zmian w aplikacji będzie ukrycie id obrazka, gdyż w aktualnej wersji możemy przeglądać wszystkie obrazy w bazie danych na podstawie numeru ID w adresie.

- Kolejną rzeczą wymagającą sporo pracy będzie dostosowanie aplikacji do urządzeń mobilnych. Obecnie aplikacja wyświetla się poprawnie tylko na komputerach, ale w przyszłości chciałbym żeby interfejs użytkownika był dostosowany również do urządzeń mobilnych.
- Kolejną zmianą będzie modyfikacja backendu, tak żeby serwer obsługiwał przesyłanie więcej niż jednego obrazu jednocześnie.
- A propos Backendu, chciałbym żeby serwer nie przechowywał wszystkich obrazów w bazie danych, tylko po zamknięciu sesji żeby obrazy były usuwane.
- Następną zmianą będzie dodanie dwóch widoków: dokumentacji i o mnie. Widok dokumentacji wydaje mi się konieczny w tego typu projekcie, bo chciałbym w nim odnieść się do autorów wykorzystanych rozwiązań i podziękować im za ich pracę. Widok o mnie jest opcjonalny, ale chciałbym go dodać, aby użytkownik mógł dowiedzieć się więcej o mojej pracy i miał możliwość skontaktowania się ze mną.
- Ostatnią kwestią gdy już uda się zaimplementować wszystkie funkcjonalności będzie udostępnienie tej aplikacji w Internecie.

Rozdział 6

Porównanie algorytmów ESRGAN i DWSR

Skoro mamy narzędzie pozwalające korzystać algorytmów **DWSR** [4] i **ESRGAN** [10] do zadania super-rozdzielczości, to warto porównać te metody. W tym rozdziale zostanie przeprowadzona analiza porównawcza algorytmów, oceniona zostanie jakość rekonstruowanych obrazów, szybkość działania oraz złożoność implementacji algorytmów.

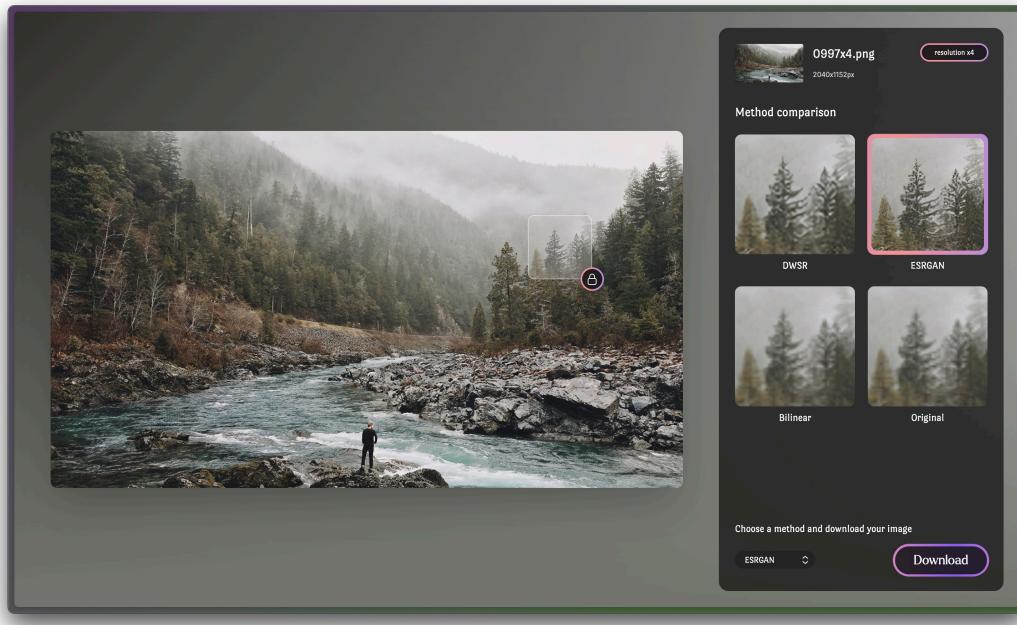
Do testów algorytmów wykorzystałem zestaw testowy z repozytorium **DWSR** [4]. Wybór padł na ten zestaw, gdyż obrazy te nie były wykorzystywane w treningu żadnej z implementowanych sieci. Obrazy w tym zestawie minimalnie różnią się od siebie rozdzielczością, co pozwala na dokładniejsze porównanie algorytmów zwłaszcza w kontekście analizy wydajności.

Na potrzebę wykonania testów wydajności algorytmów zostały wprowadzone zmiany w bazie danych w tabeli *Image*. Dodano pola *bilinear_time*, *dwsr_time* oraz *esrgan_time*, które przechowują czasy przetwarzania obrazu przez algorytmy. Czas przetwarzania został zmierzony również z etapami preprocessingu i postprocessingu obrazu, co ma znaczenie w przypadku algorytmu **DWSR**, który wymaga przeprowadzenia transformacji falkowej. Taki pomiar jest też bliższy rzeczywistości, gdyż w przypadku aplikacji webowej liczy się czas całego etapu przetwarzania obrazu, a nie samego powiększania.

6.1 Jakość odtwarzania obrazów

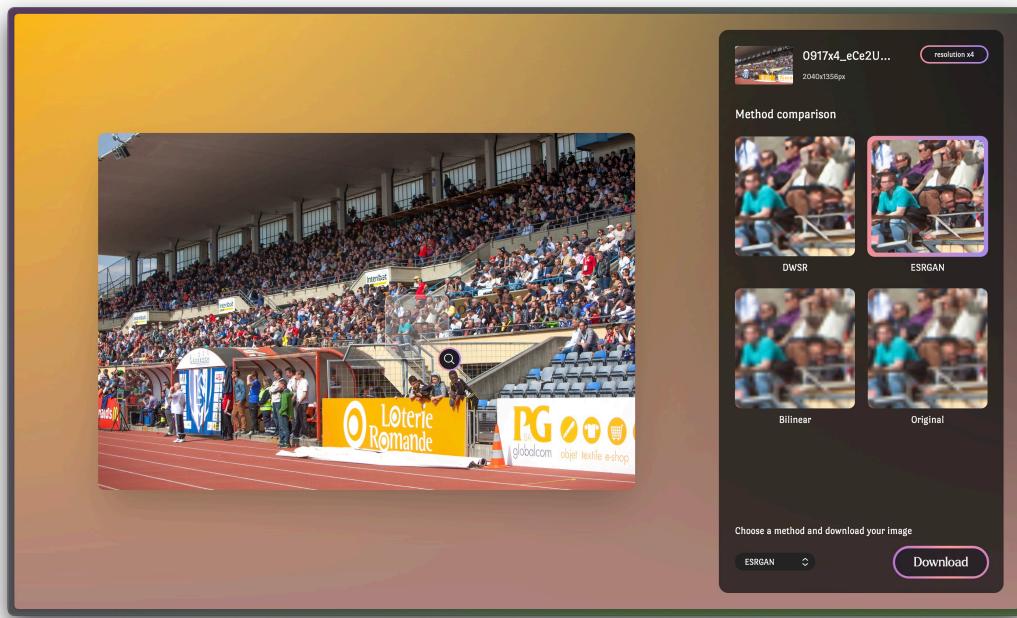
W tym rozdziale przedstawię wyniki testów jakości odtwarzania obrazów przez algorytmy **DWSR** i **ESRGAN**. Dokonałem selekcji obrazów - wybrałem te, które najlepiej obrazują różnice pomiędzy wynikami działania algorytmów.

Na obrazie 1. możemy sprawdzić jak algorytmy radzą sobie z rekonstrukcją tekstur drzew i liści. Są to detale wysokiej częstotliwości, które są trudne do odwzorowania, ale badane algorytmy potrafią sobie z nimi poradzić. Widać, że algorytm **DWSR** radzi sobie bardzo dobrze z zachowaniem kierunku tekstur, co zawdzięcza transformacji falkowej, lecz detale nie są odwzorowane idealnie. Algorytm **ESRGAN** dużo lepiej poradził sobie z odwzorowaniem detali i tekstur. Obrazy odwzorowane przez ten algorytm są bardzo ostre, miejscami aż nienaturalnie.



Rys 1. Obraz *0997x4.png* z zestawu testowego [4]

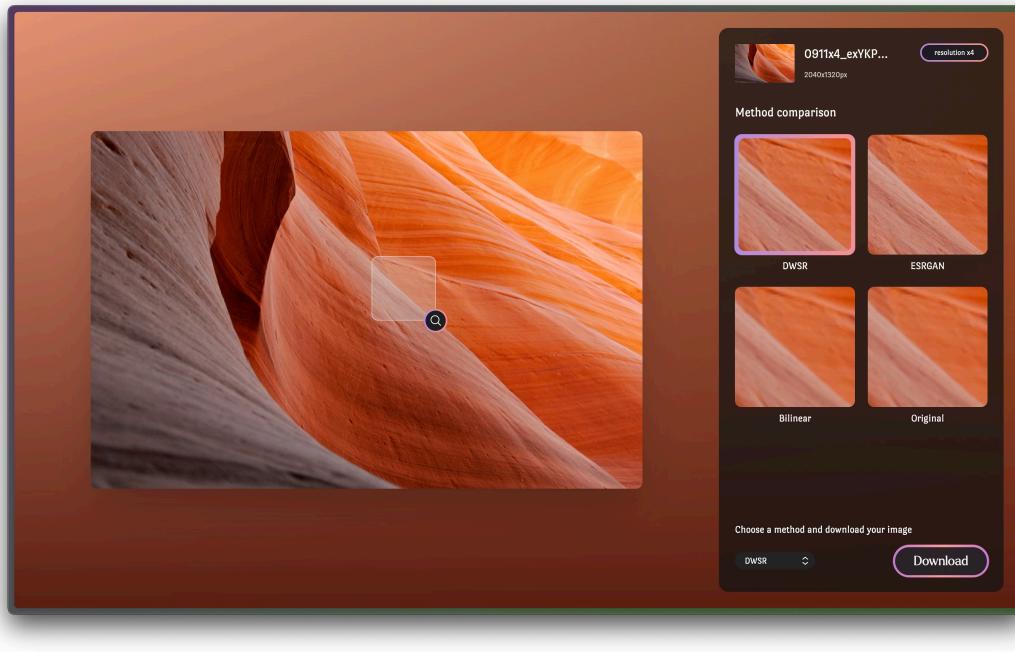
Na obrazie 2 możemy sprawdzić jak algorytmy radzą sobie z rekonstrukcją detali twarzy. Na tym przykładzie widać, że algorytm **ESRGAN** halucynuje detale twarzy i te wyglądają bardzo nienaturalnie, jednak brąz wygenerowany przez ten algorytm jest bardzo ostry. Algorytm **DWSR** radzi sobie dużo lepiej z zachowaniem naturalnego wyglądu twarzy, lecz nie odwzorowuje detali tak dobrze jak **ESRGAN**.



Rys 2. Obraz *0917x4.png* z zestawu testowego [4]

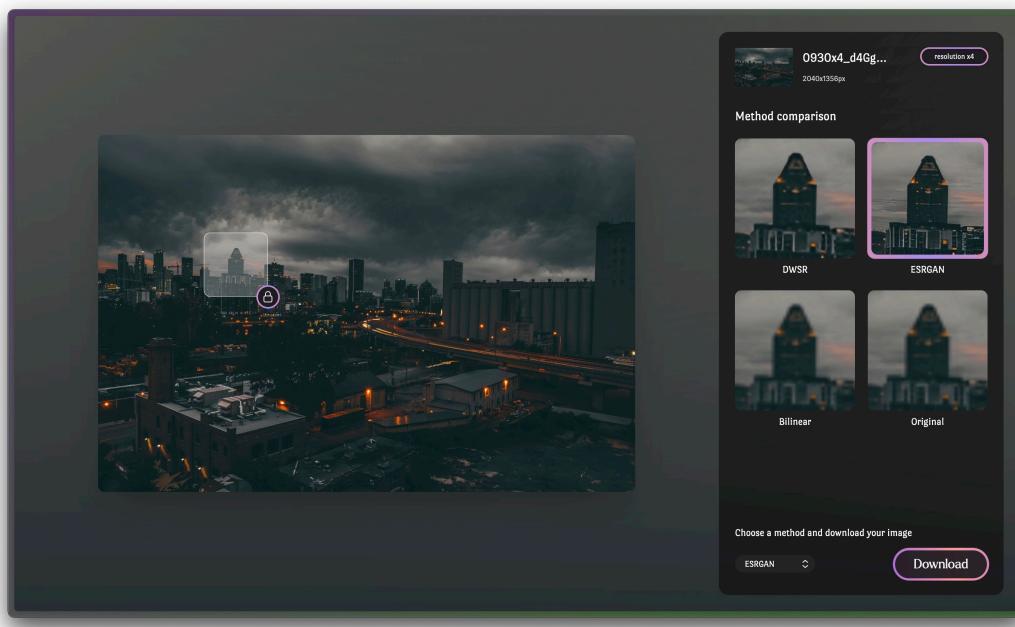
Na obrazie 3 możemy się przekonać jak algorytmy radzą sobie z powiększeniem obrazu, na którym zdecydowanie dominują niskie częstotliwości, ale występują też krzywe z ostrzejszymi krawędziami. Tutaj świetnie sprawdził się algorytm **DWSR**, który zachował

naturalny wygląd obrazu, dodatkowo zachowując ostrość krawędzi. Algorytm **ESRGAN** również poradził sobie bardzo dobrze. Na tego typu obrazach algorytm **Bilinear** (interpolacja dwuliniowa), również radzi sobie przyzwoicie, ale niestety krawędzie w obrazie wynikowym są bardzo rozmyte.



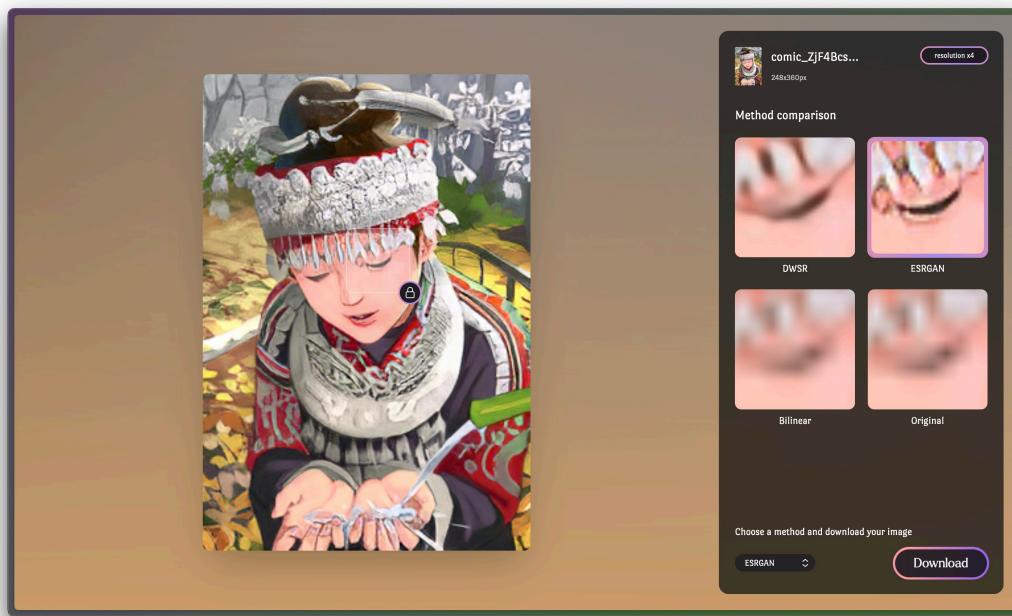
Rys 3. Obraz *0911x4.png* z zestawu testowego [4]

Obraz testowy 4 jest bardzo dobrym przykładem scenerii z dużą ilością detali i tekstur. Algorytm **ESRGAN** bezbłędnie poradził sobie z rekonstrukcją detali takich jak budynki, czy ulice. Algorytm **DWSR** również poradził sobie dobrze, jednak obraz nie jest nawet bliski ostrości jaką osiągnął algorytm **ESRGAN**.



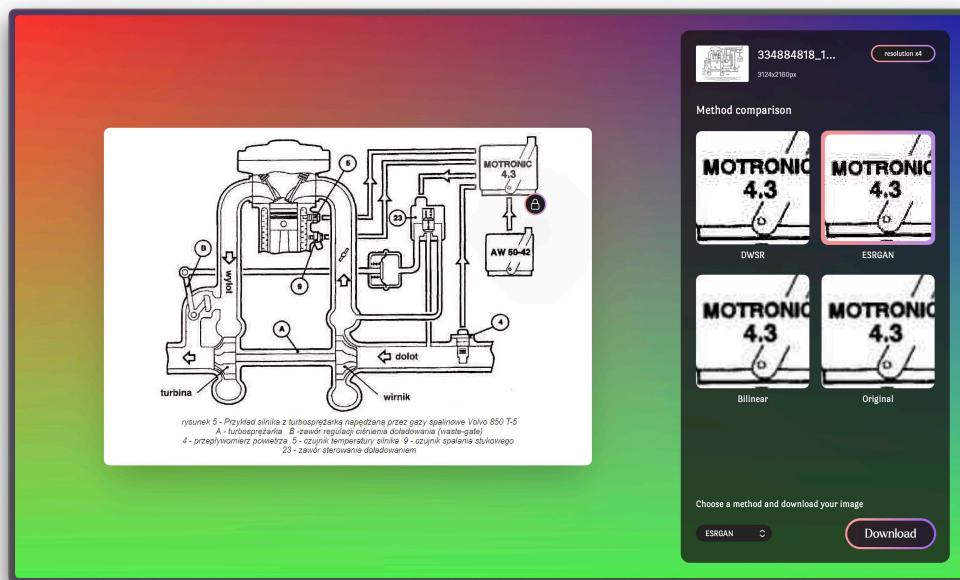
Rys 4. Obraz *0930x4.png* z zestawu testowego [4]

Obraz testowy 5 jest przykładem działania algorytmów, gdy mamy do czynienia z obrazem o bardzo niskiej rozdzielczości. Algorytm **ESRGAN** poradził sobie bardzo dobrze ze stworzeniem detali. Dokładnie zarysowane są wszystkie linie, ale obraz miejscami odbiega od oryginału, widać to między innymi na oku postaci. Algorytm **DWSR** poradził sobie dobrze, odtworzenie detali jest bliskie oryginałowi ale obraz jest bardzo rozmyty.



Rys 5. Obraz *comic.png* z zestawu testowego Set14 [13]

Ostatnim przykładem będzie rysunek techniczny z obrazu 6. Algorytm **ESRGAN** jest bardzo wrażliwy na kompresję jpg, w wyniku czego obraz wygląda nieprzyjemnie a czcionki są zbyt ostre. Algorytm **DWSR** poradził sobie dużo lepiej z rekonstrukcją obrazu, czcionki są wyraźne i czytelne, a obraz nie jest mocno zniekształcony.

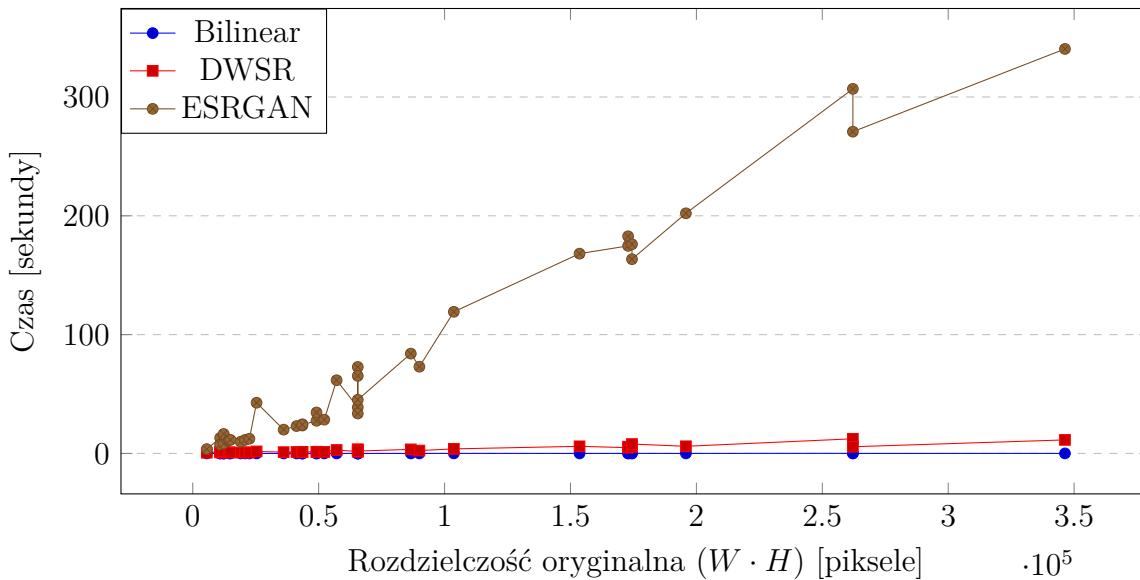


Rys 6. Schemat techniczny silnika z turbosprężarką ze strony [3]

6.2 Analiza wydajności

Na podstawie rozdziałów traktujących o strukturach algorytmów **DWSR** i **ESRGAN** można zauważać, że algorytm **ESRGAN** jest dużo bardziej skomplikowany. W związku z tym można przypuszczać, że algorytm **DWSR** będzie działał szybciej. W tym rozdziale sprawdzę jak wygląda wydajność obu algorytmów.

Do testów wydajności wykorzystałem obrazy z zestawu testowego [4] oraz [13]. Testy zostały przeprowadzone na komputerze z procesorem **Intel Core i7-9750H** z systemem **MacOS**, więc niestety nie mogłem wykorzystać GPU do przyspieszenia obliczeń, ponieważ biblioteka **PyTorch** nie wspiera kart graficznych **AMD**, w której wyposażony jest sprzęt.



Rys 7. Wykres czasu przetwarzania obrazów przez algorytmy w zależności od rozdzielczości

Jak widać na wynikach przedstawionych na wykresie 7, algorytm **Bilinear** działa najszybciej, ale nie jest to zaskoczeniem, ponieważ jest to najprostsza metoda. Algorytm **DWSR** działa zdecydowanie szybciej niż **ESRGAN** i moim zdaniem do w wielu przypadkach jest wystarczający. Widzimy tutaj kolejną zaletę korzystania z współczynników falkowych, których estymacja jest zdecydowanie szybsza niż rekonstrukcja całego obrazu przez sieć neuronową.

Warto zaznaczyć, że badania wydajnościowe na wykresie 7 zawierają czas działania nie tylko samego powiększania obrazów, ale również czas *preprocessingu* i *postprocessingu* obrazów, co ma znaczenie w przypadku algorytmu **DWSR**, który wymaga przeprowadzenia transformacji falkowej i odwróconej transformacji falkowej.

Algorytm **ESRGAN** natomiast był dużo łatwiejszy w implementacji, jedynym wymaganiem było zainstalowanie wspomnianych w rozdziale 4 bibliotek. Algorytm **DWSR** wymagał nie tylko implementacji biblioteki **PyWavelets** do przeprowadzenia transformacji falkowej, ale również przepisania programów do pre i postprocessingu z języka **MATLAB** do **Pythona**.

6.3 Ograniczenia i wyzwania

Obydwa algorytmy mają swoje wady i zalety. Algorytm **DWSR** jest dużo szybszy, ale nie jest w stanie odwzorować detali tak dobrze jak **ESRGAN**. **ESRGAN** jest dużo wolniejszy, zdecydowanie lepiej odwzorowuje szczegóły, ale często robi to w sposób nienaturalny, czego efektem mogą być obrazy wyglądające mniej przyjemnie niż **DWSR**.

Zauważałem również problem z aliasingiem i nienaturalną ostrością na obrazach wygenerowanych przez **ESRGAN**. Problemy nie występują nagminnie, ale gdy występują sprawiają, że obraz wygląda sztucznie.

Algorytmy równie dobrze radzą sobie z zaszumionymi obrazami, ale w przypadku obrazów mocno znieksztalconych przez kompresję, algorytm **ESRGAN** radzi sobie dużo gorzej, gdyż podbija te fragmenty. Algorytm **DWSR** radzi sobie dużo lepiej, gdyż nie podbija tak mocno znieksztalconych fragmentów, ale osłabia ich widoczność.

Rozdział 7

Podsumowanie i wnioski

7.1 Dyskusja wyników

Krytyczna analiza uzyskanych wyników w kontekście celów pracy oraz istniejących badań i literatury w dziedzinie.

7.2 Rekomendacje i kierunki dalszych badań

Sugestie dotyczące potencjalnych ulepszeń i obszarów, które wymagają dalszych badań, w oparciu o obserwacje i wyniki badań.

Literatura

- [1] E. Agustsson, R. Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.
- [2] M. Bevilacqua, A. Roumy, C. Guillemot, M. L. Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. 2012.
- [3] D. Bukowski. Doładowanie silników spalinowych. https://www.zssplus.pl/transport/referaty/referat_07.htm, 2004/2005.
- [4] T. Guo, H. S. Mousavi, T. H. Vu, V. Monga. Deep wavelet prediction for image super-resolution. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017.
- [5] J.-B. Huang, A. Singh, N. Ahuja. Single image super-resolution from transformed self-exemplars. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [6] A. Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan, 2018.
- [7] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [8] D. Martin, C. Fowlkes, D. Tal, J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, wolumen 2, strony 416–423. IEEE, 2001.
- [9] R. Timofte, E. Agustsson, L. V. Gool, M.-H. Yang, L. Zhang, B. Lim, S. Son, H. Kim, S. Nah, K. M. Lee, X. Wang, Y. Tian, K. Yu, Y. Zhang, S. Wu, C. Dong, L. Lin, Y. Qiao, C. C. Loy, W. Bae, J. Yoo, Y. Han, J. C. Ye, J.-S. Choi, M. Kim, Y. Fan, J. Yu, W. Han, D. Liu, H. Yu, Z. Wang, H. Shi, X. Wang, T. S. Huang, Y. Chen, K. Zhang, W. Zuo, Z. Tang, L. Luo, S. Li, M. Fu, L. Cao, W. Heng, G. Bui, T. Le, Y. Duan, D. Tao, R. Wang, X. Lin, J. Pang, J. Xu, Y. Zhao, X. Xu, J. Pan, D. Sun, Y. Zhang, X. Song, Y. Dai, X. Qin, X.-P. Huynh, T. Guo, H. S. Mousavi, T. H. Vu, V. Monga, C. Cruz, K. Egiazarian, V. Katkovnik, R. Mehta, A. K. Jain, A. Agarwalla, C. V. S. Praveen, R. Zhou, H. Wen, C. Zhu, Z. Xia, Z. Wang, Q. Guo. Ntire 2017 challenge on single image super-resolution: Methods and results. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, strony 1110–1121, 2017.

- [10] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, C. C. Loy. Esrgan: Enhanced super-resolution generative adversarial networks. *The European Conference on Computer Vision Workshops (ECCVW)*, September 2018.
- [11] Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
- [12] Z. Wang, A. C. Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 26(1):98–117, 2009.
- [13] R. Zeyde, M. Elad, M. Protter. On single image scale-up using sparse-representations. *International conference on curves and surfaces*, strony 711–730. Springer, 2010.