

Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

KIERUNEK: Automatyka i Robotyka (AIR)

PRACA DYPLOMOWA INŻYNIERSKA

TYTUŁ PRACY:
Aplikacja webowa zwiększająca
rozdzielczość obrazów

AUTOR:
Eryk Wójcik

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

Spis treści

1	Wstęp	3
1.1	Implementacja aplikacji	4
1.2	Integracja algorytmów super-rozdzielczości	6
1.3	Implementacja interfejsu użytkownika	9
1.4	Testowanie aplikacji	9
1.5	Plany na przyszłość	9
2	Porównanie algorytmów ESRGAN i DWSR	11
2.1	Kryteria porównawcze	11
2.2	Analiza wydajności	11
2.3	Jakość odtwarzania obrazów	11
2.4	Ograniczenia i wyzwania	11
3	Podsumowanie i wnioski	13
3.1	Dyskusja wyników	13
3.2	Rekomendacje i kierunki dalszych badań	13
	Bibliografia	14

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.



Rys 1. Diagram przepływu użytkownika

TODO: Usun diagram przepływu

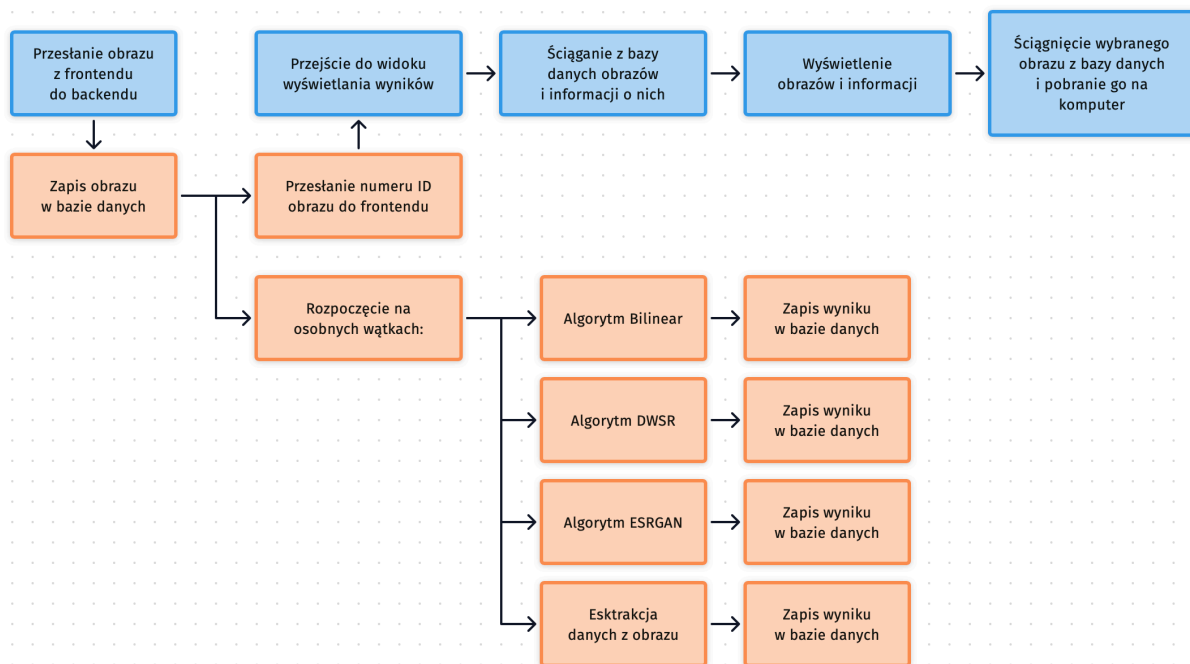
1.1 Implementacja aplikacji

Po wyborze stosu 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 1], 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 2], który pozwoli zrozumieć zachowanie programu.



Rys 2. 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 [1.2]. 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 Backendu 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 1.1 przedstawiam jedyną tabelę w bazie danych, która przechowuje informacje o obrazach.

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

Tabela 1.1: 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 [1.2]. Jest to lista kolorów w formacie HEX, które wykorzystuje Frontend do wyświetlenia kolorowych gradientów tła.

1.2 Integracja algorytmów super-rozdzielczości

Gdy obraz jest już zapisany w bazie danych, serwer Backend zleca wykonanie algorytmów super-rozdzielczości na osobnych wątkach [1.1]. 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 1.1: 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 TODO: Końcowe: Daj ref do podstaw teoretycznych Bilinear. Jest to najmniej skomplikowany algorytm, więc implementacja jego nie była trudna [1.2].

```

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 1.2: Implementacja algorytmu Bilinear.

W pierwszej kolejności wczytujemy obraz do powiększenia, następnie pobieramy jego wymiary 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 algorytmem jest algorytm DWSR, który został opisany w rozdziale TODO: Końcowe: Daj ref do DWSR. 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 [1.3].

```
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 1.3: 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 [1.4], 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 1.4: 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-Rozdzielczości implementowanym w aplikacji jest algorytm ESRGAN, który został opisany w rozdziale TODO: Końcowe: Daj ref do ESRGAN. 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 [1.5]. 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 1.5: Implementacja algorytmu ESRGAN.

W pierwszej kolejności wczytujemy model ESRGAN, następnie przetwarzamy obraz przez model [1.6] 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     return output
```

Listing 1.6: Przetwarzanie przez model ESRGAN.

Algorytm K-średnich

Algorytm K-średnich został opisany w rozdziale TODO: Końcowe: Daj ref do K-średnich. 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 [1.7].

```
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)
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    image_instance.set_dominant_colors(dominant_colors_list)
14    image_instance.save()
```

Listing 1.7: Implementacja algorytmu K-średnich.

W pierwszej kolejności wczytujemy obraz, następnie przekształcamy go do formatu RGB i przekształcamy do formatu, który może być wykorzystany przez algorytm K-średnich. 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.

1.3 Implementacja interfejsu użytkownika

1.4 Testowanie aplikacji

1.5 Plany na przyszłość

Opis błędów, rzeczy do poprawy w aplikacji. Omówienie jakie są plany rozbudowy aplikacji.

Przyciski nawigacji z widoku do widoku

Rozdział 2

Porównanie algorytmów ESRGAN i DWSR

[1]

2.1 Kryteria porównawcze

Ustalenie kryteriów, które będą stosowane do oceny i porównania skuteczności i efektywności algorytmów super rozdzielczości.

2.2 Analiza wydajności

Bezpośrednie porównanie wydajności obu metod w różnych warunkach, bazujące na ustalonych kryteriach.

2.3 Jakość odtwarzania obrazów

Ocena jakości obrazów generowanych przez oba algorytmy, uwzględniając różne aspekty jakości wizualnej.

2.4 Ograniczenia i wyzwania

Dyskusja na temat ograniczeń obu metod i potencjalnych wyzwań w ich stosowaniu.

Rozdział 3

Podsumowanie i wnioski

3.1 Dyskusja wyników

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

3.2 Rekomendacje i kierunki dalszych badań

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

TODO: Odkomentuj TODO: dodaj ref do wzorów

Literatura

- [1] 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.