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

WROCŁAW 2023

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 Wdrożenie i utrzymanie 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

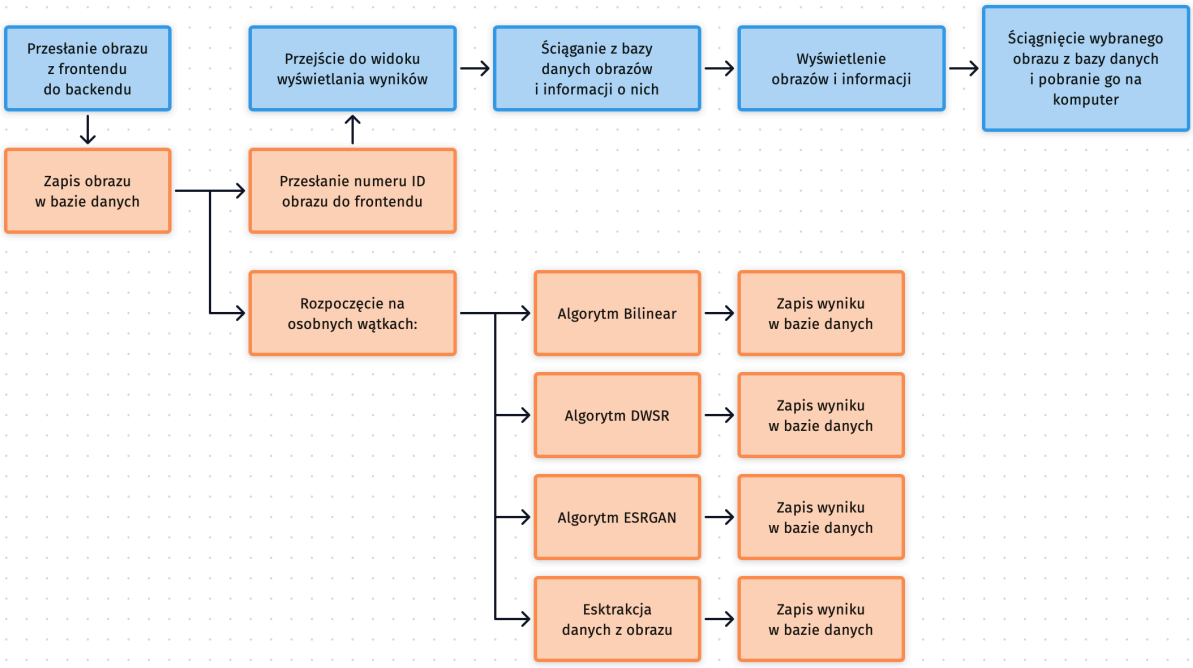
TODO: Usuń diagram przepływu

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 ESRGAN.
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.

```
def upload_image(request):
try:
    form = Image(image=request.FILES['image'])
    form.save()

    input_image_path = form.image.path

    thread1 = threading.Thread(target = run_bilinear,
                               args = (input_image_path, 4, form))
    thread2 = threading.Thread(target = run_dwsr,
                               args = (input_image_path, 4, form))
    thread3 = threading.Thread(target = run_esrgan,
                               args = (input_image_path, form))
    thread4 = threading.Thread(target = extract_image_info,
                               args = (input_image_path, form))
    thread1.start()
    thread2.start()
    thread3.start()
    thread4.start()

    image = Image.objects.latest('id')  # Gets the latest entry

    return JsonResponse({'message': 'Image uploaded, processing started',
                         'image_id': image.id})
except Exception as e:
    return JsonResponse({'error': str(e)}, status=400)
```

Listing 1.1: Obsługa zapisu i przetwarzania obrazów (Django).

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].

```
def run_bilinear(input_image_path, scale, image_instance):
    img = cv2.imread(input_image_path, cv2.IMREAD_COLOR)

    height, width = img.shape[:2]
    new_width, new_height = width * scale, height * scale

    # Resize the image using bilinear interpolation
    output = cv2.resize(img, (new_width, new_height), interpolation=cv2.
    INTER_LINEAR)

    save_output_image(output, input_image_path, image_instance)
```

Listing 1.2: Implementacja algorytmu Bilinear (Django).

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].

```
def run_dwsr(input_image_path, scale, image_instance):
    enlarged_lr_dir, sr_lum_dir, output_dir = create_output_dir()

    fileName = os.path.basename(input_image_path)

    # 1. Generate enlarged LR images
    generate_enlarged_lr(input_image_path, enlarged_lr_dir, scale)

    # 2. Process image with DWSR
    process_image(enlarged_lr_dir + '/' + fileName, sr_lum_dir, scale)

    # 3. Generate color SR
    final_img_path = generate_color_sr(input_image_path, sr_lum_dir, output_dir
    , scale)

    save_output_image(final_img_path, input_image_path, image_instance)
```

Listing 1.3: Implementacja algorytmu DWSR (Django).

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.

```
def process_image(input_image_path, output_dir, scale):

    test_input = tf.placeholder(np.float32)
    # Feeding Forward
    test_output, _, _ = model(test_input)
    with tf.Session(config=tf.ConfigProto()) as sess:
        tf.initialize_all_variables().run()
        saver = tf.train.Saver(tf.all_variables())

        saver.restore(sess, MODEL_PATH)

        print('Processing Image %s' % ntpath.basename(input_image_path))
        testBBImg = cv2.imread(input_image_path, 0)
        tcoeffs = pw.dwt2(testBBImg, WV)
        tcA, (tcH, tcV, tcD) = tcoeffs
```

```
        tcA = tcA.astype(np.float32) / 255
        tcH = tcH.astype(np.float32) / 255
        tcV = tcV.astype(np.float32) / 255
        tcD = tcD.astype(np.float32) / 255
        test_temp = np.array([tcA, tcH, tcV, tcD])
        test_elem = np.rollaxis(test_temp, 0, 3)
        test_data = test_elem[np.newaxis, ...]
        start_time = time.time()
        output_data = sess.run([test_output], feed_dict={test_input: test_data
    })
        duration = time.time() - start_time
        dcA = output_data[0][0, :, :, 0]
        dcH = output_data[0][0, :, :, 1]
        dcV = output_data[0][0, :, :, 2]
        dcD = output_data[0][0, :, :, 3]
        srcoeffs = (dcA * 255 + tcA * 255,
                    (dcH * 255 + tcH * 255,
                     dcV * 255 + tcV * 255,
                     dcD * 255 + tcD * 255))
        sr_img = pw.idwt2(srcoeffs, WV)

        output_image_path = os.path.join(output_dir, ntpath.basename(
    input_image_path))
        cv2.imwrite(output_image_path, sr_img)
        print('Processed image saved to %s, processing time: %s' % (
    output_image_path, str(duration)))

        sess.close()
```

Listing 1.4: Przetwarzanie przez model DWSR (Django).

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.

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

Listing 1.5: Implementacja algorytmu ESRGAN (Django).

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

```
def process_image_with_esrgan(model, device, input_image_path):
    img_LR = read_img(input_image_path, device)

    with torch.no_grad():
        output=model(img_LR).data.squeeze().float().cpu().clamp_(0, 1).numpy()

```

```
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 (Django).

Algorytm K-średnich

1.3 Implementacja interfejsu użytkownika

1.4 Wdrożenie i utrzymanie aplikacji

Omówienie procesu wdrożenia gotowej aplikacji oraz planów dotyczących jej przyszłego utrzymania i aktualizacji.

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.