

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI



Wirtualna Szminka

Sprawozdanie z projektu

AUTOR

Aleksandra Walczybok

nr albumu: **272454**

kierunek: **Inżynieria systemów**

2 styczeń 2025

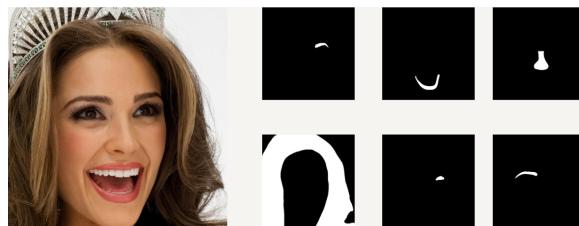
1 Wstęp – sformułowanie problemu

Jako temat realizacji projektu wybrano stworzenie aplikacji wirtualnej szminki. Celem aplikacji jest umożliwienie użytkownikowi wyboru odpowiedniego koloru szminki oraz nałożenie jej na usta, co pozwoli na przetestowanie efektu przed dokonaniem ostatecznego wyboru. Aplikacja ma działać w czasie rzeczywistym.

W ramach implementacji przyjęto następujące podejście: proces nakładania szminki na usta rozpoczyna się od segmentacji ust na obrazie, a następnie na wyodrębnioną powierzchnię ust nakłada się wybrany kolor w formacie BGR. Zostaną uwzględnione zarówno wagę maski, jak i dane wejściowe zdjęcia, aby zapewnić jak najbardziej naturalny i precyzyjny efekt wizualny.

2 Zbiór danych

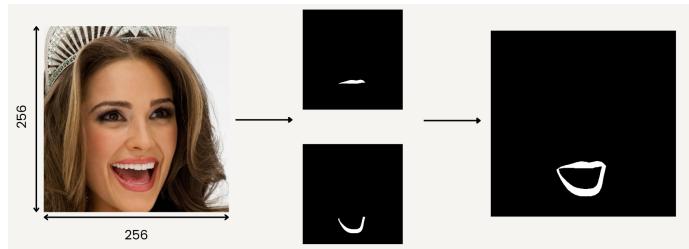
Jako zbiór danych użyto popularnego datasetu - CelebAMask-HQ Dataset. Zawiera on 30 000 zdjęć twarzy celebrytów o różnych płciach i rasach, co jest dodatkowym atutem w kwestii różnorodności zbioru danych. Do każdego zdjęcia załączone są również maski konkretnych części twarzy tj.: lewej brwi, prawego oka, włosów, szyi, górnej wargi itp.



Rysunek 1: Przykład zdjęcia i odpowiadających masek

W przypadku projektu istotne były jedynie maski odpowiadające za fragmenty ust. Aby otrzymywać na wyjściu jedną maskę, zastosowano skrypt, który łączył odpowiednio zdjęcia dwóch masek (górnej i dolnej wargi).

Tworząc model, zaczęto uczyć go od małej ilości danych, aby potwierdzić ogólne działanie. Następnie powiększano zbiór do takiego momentu, w którym wyniki były zadowalające, a czas obliczeń nie był zbyt długi. Ostatecznie użyto 3000 zdjęć. Ze względu na charakter użytych w późniejszej fazie architektur, zmieniono rozmiar zdjęć oraz masek na 256x256.



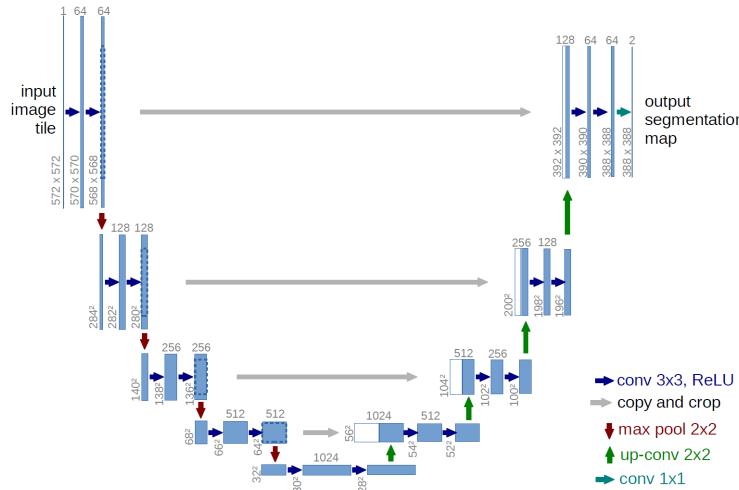
Rysunek 2: Przykład połączonych masek ust

Zbiór danych podzielono na zbiór treningowy (80%), walidacyjny (10%) oraz testowy (10%).

3 Segmentacja

Początkowym i głównym zadaniem była segmentacja ust. Aby tego dokonać, użyto dwóch podejść, stosując architekturę UNet.

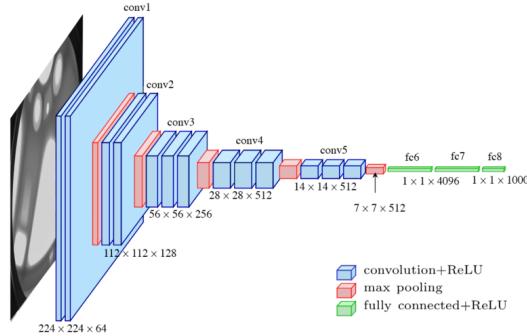
U-Net to architektura sieci neuronowej zaprojektowana do segmentacji obrazów, która wykorzystuje strukturę z symetrycznym układem kodera i dekodera. Dzięki połączeniom skip (pomijającym warstwy) między warstwami kodera i dekodera, U-Net pozwala na precyzyjne odzyskiwanie szczegółów z obrazu. Poniżej przedstawiony został schemat tej architektury:



Rysunek 3: Architektura UNet

3.1 Początkowe podejście - Użycie biblioteki SMP

W pierwszym podejściu wykorzystano bibliotekę *segmentation-models-pytorch* oraz model UNet z enkoderem *vgg19*. VGG19 jest wybierany jako encoder w modelu U-Net ze względu na swoją głęboką architekturę i zdolność do efektywnej ekstrakcji cech z obrazów dzięki warstwom konwolucyjnym o małych filtrach (3x3). Enkoder ten został wstępnie wytrenowany na zbiorze danych ImageNet, co pozwala na przyspieszenie procesu uczenia dzięki wykorzystaniu transferu wiedzy z wcześniejszego treningu. Następnie model został zaimplementowany i przetrenowany na nowo przy użyciu opisywanego wcześniej zbioru danych, dostosowując wagę do specyfiki nowego zadania segmentacji.



Rysunek 4: Schemat VGG

Wybranym kryterium była funkcja straty Dice Loss, która lepiej uwzględnia niewielkie obszary, takie jak usta, w porównaniu do Binary Cross-Entropy. Początkowo rozważano zastosowanie Binary Cross-Entropy, jednak ze względu na dysproporcję między obszarem ust a resztą twarzy, funkcja ta prowadziła do wysokich wyników nawet w przypadku, gdy model całkowicie pomijał segmentację ust. Dice Loss została wybrana, ponieważ skuteczniej penalizuje takie błędy, optymalizując model pod kątem dokładniejszej segmentacji małych obszarów.

$$DiceLoss = 1 - \frac{2 \sum_{i=1}^N p_i g_i}{\sum_{i=1}^N p_i^2 + \sum_{i=1}^N g_i^2 + \epsilon}$$

$$BCE = -\frac{1}{N} \sum_{i=1}^N (g_i \log(p_i) + (1 - g_i) \log(1 - p_i))$$

3.2 Drugie podejście - własny UNet

Po pozytywnym działaniu pierwszego podejścia postanowiono samemu napisać architekturę UNet, używając jedynie dziedzicznej klasy nn.Module.

Architerura została tworzona zgodnie z rysunkiem 3.

W zaimplementowanej architekturze UNet ścieżka kontrakcji (down-sampling) została zrealizowana poprzez iteracyjne stosowanie modułu ‘DoubleConv’, który składa się z dwóch warstw konwolucyjnych, normalizacji BatchNorm oraz aktywacji ReLU, a następnie redukcje wymiarów obrazu za pomocą operacji ‘MaxPool2d’. Ścieżka ekspansji (up-sampling) wykorzystuje warstwy transponowanych konwolucji (‘ConvTranspose2d’) do zwiększania wymiarów obrazu, a następnie łączy (concatenate) wyjścia z odpowiadającymi im warstwami ze ścieżki kontrakcji za pomocą połączeń skip, co umożliwia precyzyjne odzyskanie szczegółów, zanim zostanie zastosowany końcowy moduł ‘DoubleConv’ i warstwa konwolucji wyjściowej (‘final conv’).

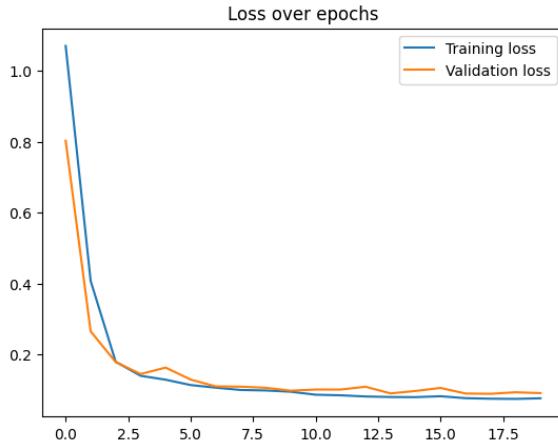
Za kryterium przyjęto tutaj sumę DiceLoss i BinaryCrossEntropy.

3.3 Trening dwóch modeli oraz porównanie

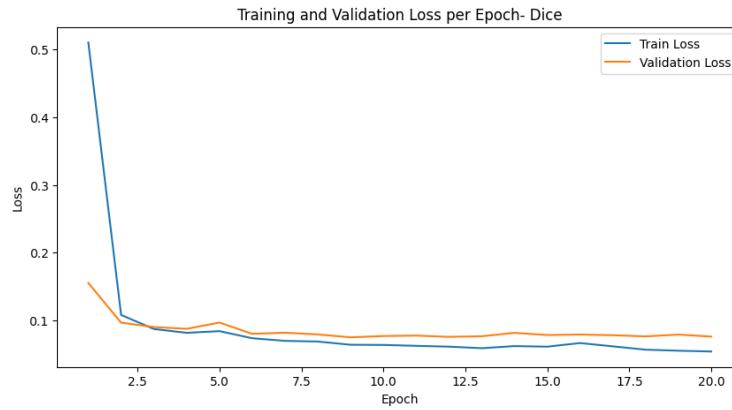
Oba modele wykorzystywały optymalizator Adam, który łączy zalety adaptacyjnego współczynnika uczenia oraz momentu, co pozwala na szybką i bardziej stabilną konwergencję.

W dwóch podejściach zapisywano wagi modelu, jeśli w danej epoce wynik kryterium na zbiorze walidacyjnym był wyższy niż dotychczasowy najlepszy wynik. Zapobiega to nadmiernemu dopasowaniu modelu do danych treningowych (overfitting) oraz pozwala zaoszczędzić czas i zasoby obliczeniowe.

Oba modele uczone podczas 20 epoch. Czas uczenia wynosił około 9 h. Jak widać na poniższych wykresach, po 10 epokach koszt nie malał już znacząco. Analizując wykresy, okazuje się, że oba podejścia osiągają podobne wyniki na koniec uczenia.



Rysunek 5: Własny UNet



Rysunek 6: UNet z vgg

3.4 Testowanie

Do oceny modeli użyto 3 metryk: IoU, accuracy oraz recall. Ostatnią metrykę – recall – wybrano z założeniem, że lepiej jest, gdy model nie wykryje wszystkich pikseli ust, niż gdyby błędnie rozpoznał

piksele znajdujące się poza nimi, na przykład na nosie. Takie błędne detekcje mogłyby znacząco wpływać na komfort użytkowania aplikacji, dlatego preferowano minimalizowanie tego typu pomyłek.

1. Intersection over Union (IoU):

$$IoU = \frac{\sum_{i=1}^N (p_i \cdot g_i)}{\sum_{i=1}^N (p_i + g_i - p_i \cdot g_i)}$$

gdzie p_i to przewidywana wartość (predykcja) dla piksela i , g_i to wartość rzeczywista (ground truth) dla piksela i , a N to liczba pikseli w obrazie.

2. Accuracy:

$$Accuracy = \frac{\sum_{i=1}^N 1(p_i = g_i)}{N}$$

gdzie $1(p_i = g_i)$ to funkcja wskaźnikowa, która przyjmuje wartość 1, jeśli $p_i = g_i$ (piksel został poprawnie sklasyfikowany), w przeciwnym razie 0, a N to liczba pikseli.

3. Recall (czułość):

$$Recall = \frac{\sum_{i=1}^N (p_i \cdot g_i)}{\sum_{i=1}^N g_i}$$

gdzie p_i to przewidywana wartość (predykcja) dla piksela i , g_i to wartość rzeczywista (ground truth) dla piksela i , a N to liczba pikseli w obrazie.

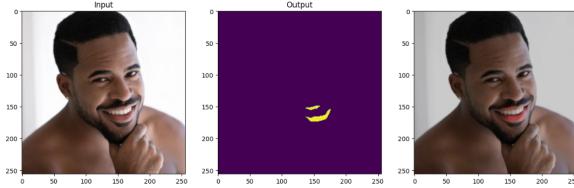
Metric	U-Net + VGG	Własny Model
Mean IoU	0.8596	0.8221
Mean Recall	0.9150	0.8925
Mean Accuracy	0.9982	0.9976

Tabela 1: Porównanie wyników modeli U-Net z vgg i własnego modelu

Oba modele uzyskują równie satysfakcjonujące wyniki. Widać nieznaczną przewagę dla modelu z vgg. W tym projekcie istotne jest jednak to, jak modele działają w praktyce, a więc poniżej zestawiono zdjęcia i segmentację dla kilku zdjęć testowych.



Rysunek 7: Porównanie kilku przypadków testowych

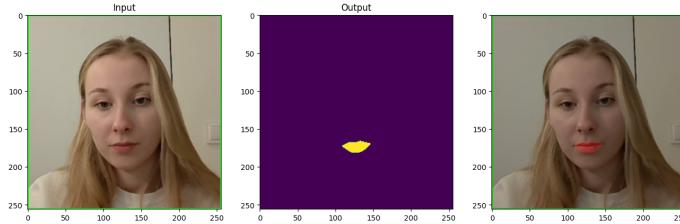


Rysunek 8: Przypadek testowy dla UNet + vgg

Z przypadków testowych wyniknęły dwa najważniejsze wnioski. Jeśli zdjęcie wejściowe jest zbyt szerokie, to zmiana rozmiaru może na tyle zwiększać krawędzie, iż ciężko będzie dla własnego UNetu wykryć usta. Ujęto tą kwestię w dalszej części projektu - przy tworzeniu części z kamerą.

Dodatkowo okazuje się, że dużym problemem dla obu modelów, ale w szczególności dla własnego UNetu, było wykrywanie ust u mężczyzn z zarostem. Sprawdzono te przypadki również na żywo - na osobach mających zarost, również modele nie działały tak dobrze jak na osobach bez zarostu. Problem ten nie jest jednak aż tak istotny, gdyż zakłada się, że takie osoby nie będą docelową grupą, do której skierowana jest aplikacja.

Następnie testowano również modele na tworzonych w skrypcie zdjęciach:



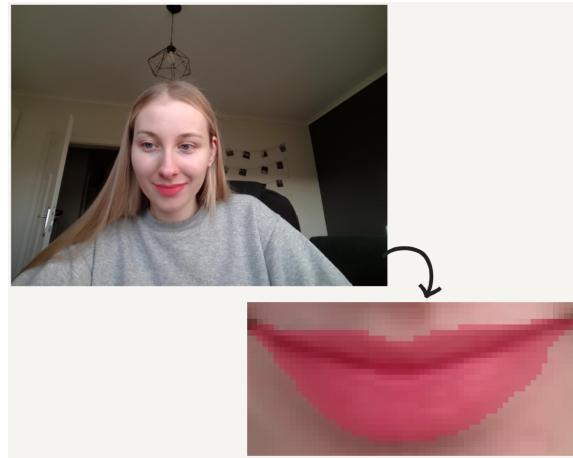
Rysunek 9: Maska na robionym zdjęciu

3.5 Nakładanie koloru ust

Kolejnym istotnym procesem było nałożenie koloru na usta na zdjęciu wejściowym. Ważne było to, aby kolor ust był naturalny.

Wybrane podejście jest następujące: Na początku tworzona jest pusta maska o tym samym rozmiarze jak obraz wejściowy, której wartości są inicjalizowane na 0. Następnie, dla każdego piksela, który został wykryty jako część segmentacji (czyli dla wartości większych niż 0 w masce predykcji), przypisywane są wartości koloru z parametru ‘color’ (który jest wektorem BGR).

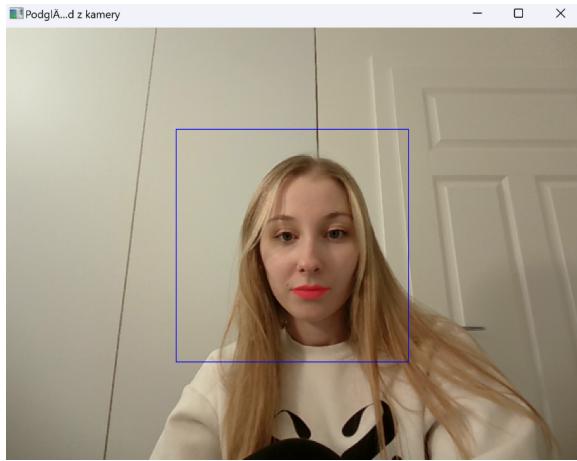
Kolejnym krokiem jest przygotowanie zmodyfikowanej wersji obrazu – kopii oryginalnego obrazu. Maski są nakładane tylko na te piksele, które zostały wykryte jako część ust, co jest określone przez maskę ‘pred’ (gdzie wartości większe niż 0 wskazują na obecność segmentowanych ust). W tych miejscach, z pomocą funkcji ‘cv2.addWeighted’, oryginalny obraz jest łączony z maską z określona przejrzystością, co pozwala na uzyskanie efektu przezroczystego nakładania koloru na segmentowane obszary. Dzięki temu uzyskuje się wizualizację segmentacji, gdzie obszary ust są pokolorowane, ale zachowują subtelne przejścia między oryginalnym obrazem a nałożoną maską.



Rysunek 10: Nałożony kolor ust na zdjęcie

3.6 Czas rzeczywisty

Następnym etapem była segmentacja i nakładanie koloru w czasie rzeczywistym. Ze względu na problemy ze zmianą rozmiarów zdjęcia użyto tutaj "wskaźnika" w postaci narysowanego obszaru (kwadratu), gdzie powinna znaleźć się twarz osoby. Miało to na celu łatwiejsze wycięcie pożądanego rozmiaru zdjęcia.



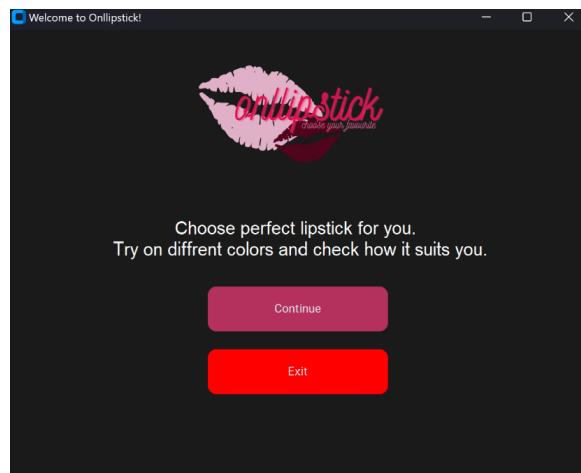
Rysunek 11: Zdjęcie z działania kamery - nakładania szminki w czasie rzeczywistym

Podejście do segmentacji ust w czasie rzeczywistym polega na użyciu kamery do przechwytywania obrazu, który jest następnie przetwarzany przez wytrenowany model segmentacji. W kodzie obraz z kamery jest najpierw przycinany do pożądanych rozmiarów, a następnie przetwarzany przez model U-Net, który przewiduje maskę segmentacyjną dla ust. Na podstawie tej maski nakładany jest kolorowy filtr na obszary wykryte jako usta, co umożliwia ich wizualizację na obrazie, a całość jest wyświetlana na żywo w oknie kamery.

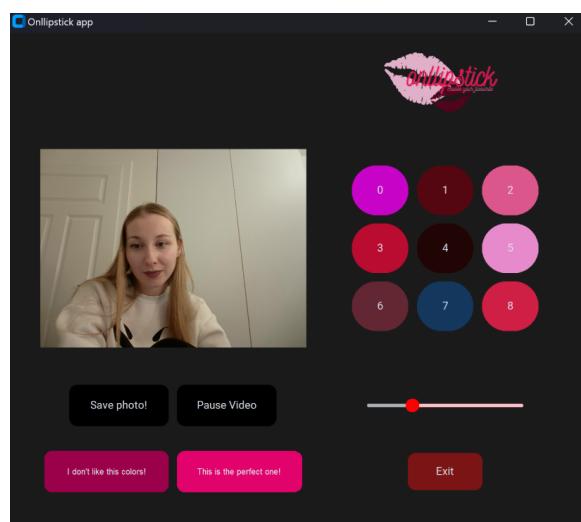
3.7 Interfejs graficzny

Ostatnim etapem było stworzenie interfejsu graficznego. Został on wykonany przy użyciu biblioteki customtkinter. W aplikacji użyto wcześniej wspomnianych metod.

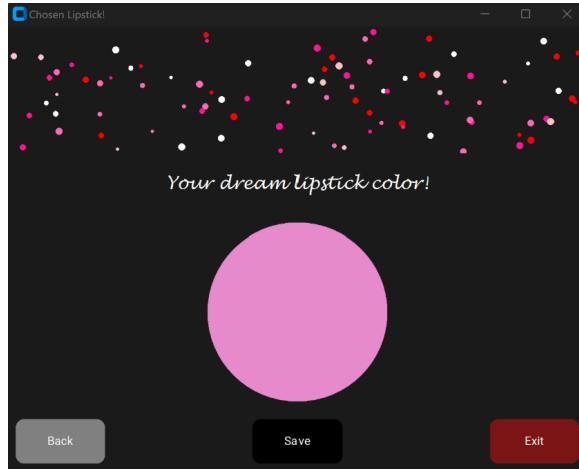
Aplikacja umożliwia „na żywo” przetestowanie różnych kolorów szminiek. Użytkownicy mogą zapisać zdjęcie, zatrzymać obraz z kamery oraz dostosować intensywność koloru, co pozwala na symulację nałożenia większej ilości produktu. Dodatkowo, dostępna jest opcja „I don’t like this color”, która po kliknięciu otwiera pełną paletę kolorów BGR. Po wybraniu koloru, w ostatnim oknie, użytkownik może zapisać wybrany odcień, wrócić do poprzedniego etapu lub zakończyć korzystanie z aplikacji.



Rysunek 12: Zdjęcie z aplikacji 1



Rysunek 13: Zdjęcie z aplikacji 2



Rysunek 14: Zdjęcie z aplikacji 3

4 Podsumowanie

W ramach projektu stworzono aplikację wirtualnej szminki, której celem jest umożliwienie użytkownikowi wypróbowania różnych kolorów szminki na swoich ustach w czasie rzeczywistym. Kluczowym zadaniem było przeprowadzenie segmentacji ust na zdjęciu przy użyciu architektury UNet, co pozwoliło na precyzyjne wyodrębnienie tej części twarzy. Do treningu modeli wykorzystano popularny zbiór danych CelebAMask-HQ, a dla oceny skuteczności zastosowano metryki takie jak IoU, accuracy i recall. W wyniku testów oba modele – UNet z enkoderem VGG oraz własny model UNet – osiągnęły zbliżone wyniki, z minimalną przewagą pierwszego podejścia. Dodatkowo, nałożenie koloru na usta zostało zrealizowane przy pomocy techniki maskowania i nakładania przezroczystych filtrów. W projekcie uwzględniono także segmentację w czasie rzeczywistym z użyciem kamery, co pozwoliło na natychmiastowe przetestowanie efektów. Całość została zamknięta w aplikacji z intuicyjnym interfejsem graficznym, umożliwiającym użytkownikom łatwe korzystanie z funkcji wirtualnej szminki.

A Dodatek

Kody źródłowe umieszczone zostały w repozytorium GitHub:

<https://github.com/aleksandra0014/lips-virtual-makeup>.