Wirtualna szminka

ALEKSANDRA WALCZYBOK

Grudzień 2024

Problem

Problem polegał na segmentacji ust w czasie rzeczywistym, a następnie nakładaniu wybranego koloru szminki przez użytkownika.

START Research podobnych projektów i technologii.

1 Wybór zbioru danych uczących oraz preprocessing danych.

2 Implementacja oraz uczenie modelu segmentacyjnego.

3 Testowanie modelu w czasie rzeczywistym, wstępny skrypt z kamerką.

4 Przygotowanie interfejsu graficznego.

Projekty produkcyjne

MAYBELLINE NEW YORK

Virtual Makeup Try On - Makeup Tool - Makeup Tips

Get a virtual makeover with the virtual makeup tool by Maybelline. Instantly try on eye, face & lip...

Maybelline New York

Virtual Makeup Try On – L'Oréal Paris

Wypróbuj symulator makijażu od L'Oréal Paris, przetestuj najnowsze trendy w makijażu ust, oczu i...

lorealparis.pl

ŹRÓDŁA “STARTOWE”

Lip Segmentation with Attention UNet

Explore and run machine learning code with Kaggle Notebooks | Using data from multiple data sources

Kaggle / Sep 20, 2022

Priyansi/**eyes-lips-segmentation**

PyTorch code for binary segmentation on CelebAMask-HQ dataset via both a UNet written from scratch and a pretrained DeepLabv3 model.

1 Contributor · 0 Issues · 6 Stars · 0 Forks

Priyansi/eyes-lips-segmentation: PyTorch code for binary segmentation on CelebAMask-HQ dataset via both a UN...

PyTorch code for binary segmentation on CelebAMask-HQ dataset via both a UNet written from scratch and a pretrained DeepLabv3 model. - Priyansi/eyes-lips-segmentation

GitHub

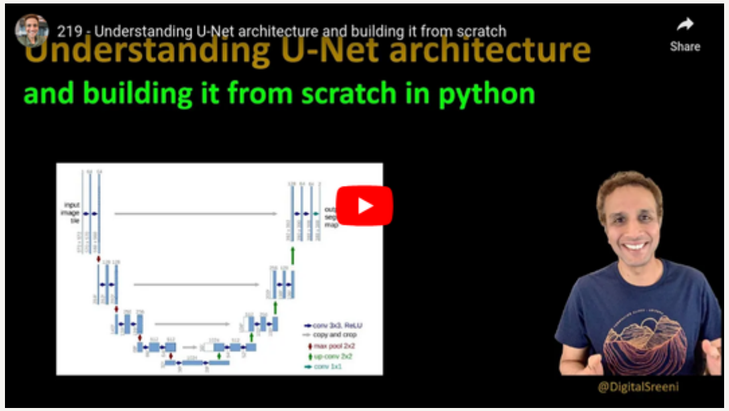
raahatg21/**Eyes-Lips-Segmentation**

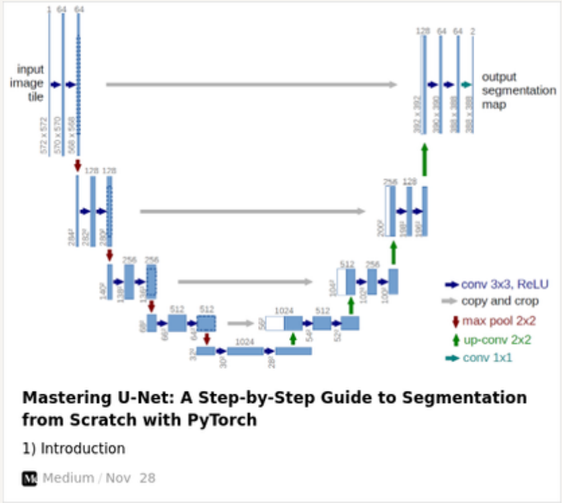
1 Contributor · 1 Issue · 18 Stars · 6 Forks

raahatg21/Eyes-Lips-Segmentation

Contribute to raahatg21/Eyes-Lips-Segmentation development by creating an account on GitHub.

GitHub





Mastering U-Net: A Step-by-Step Guide to Segmentation from Scratch with PyTorch

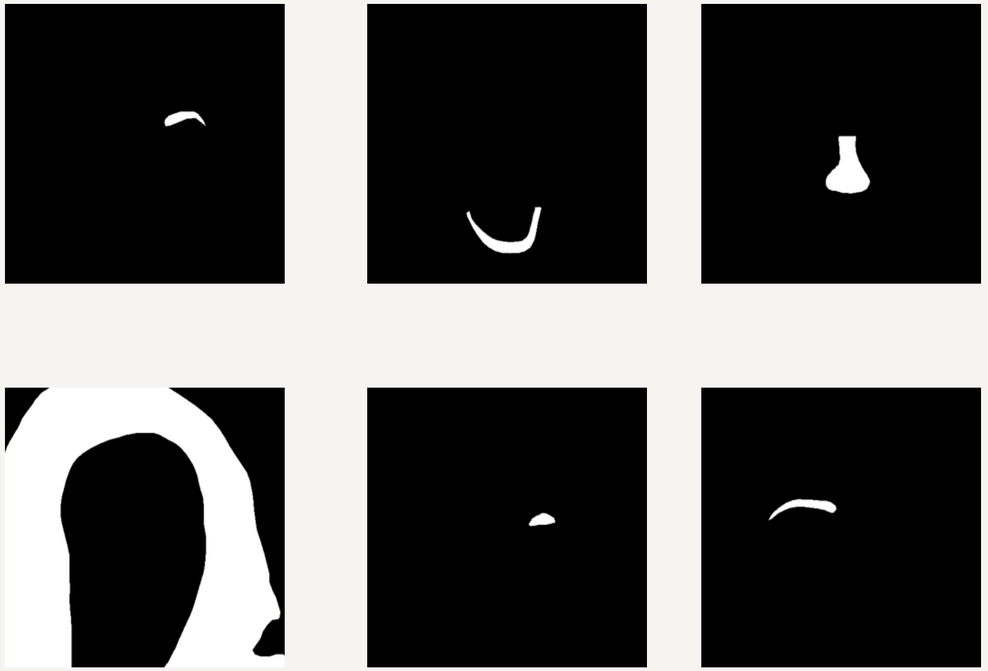
1) Introduction

Medium / Nov 28

1

Zbiór danych

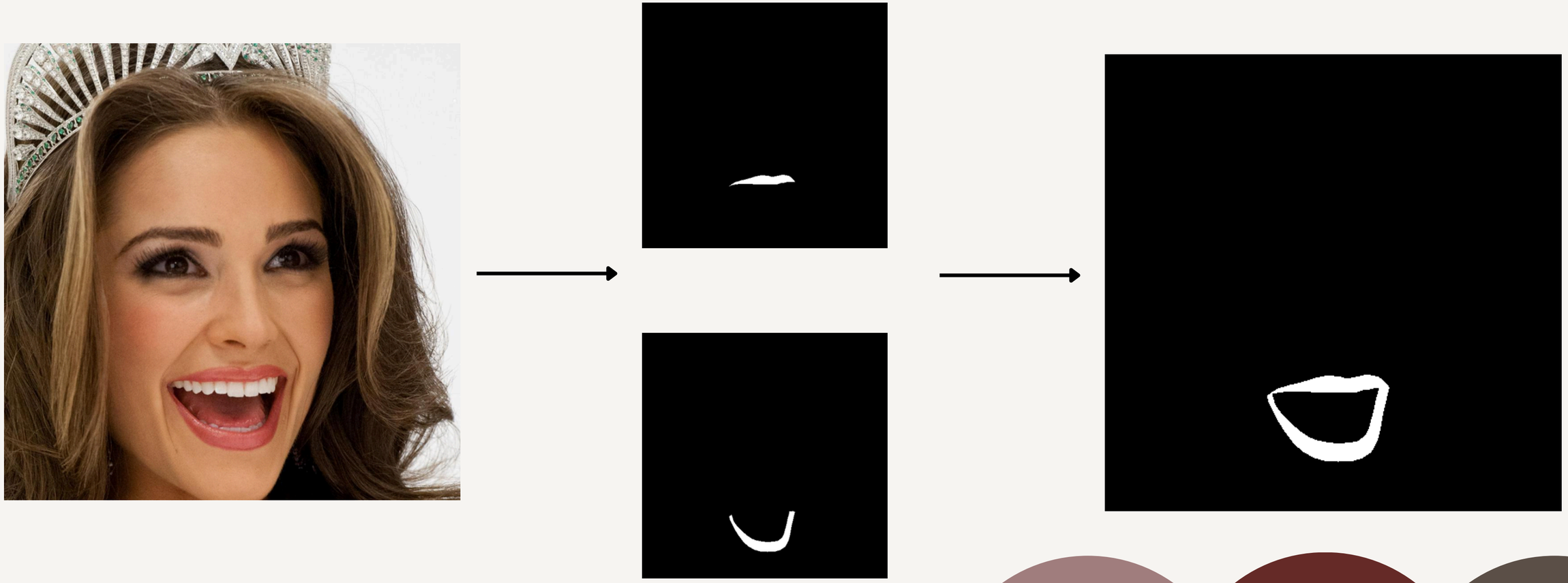
CelebAMask-HQ Dataset



30 000 ⟶ 3000

1 Zbiór danych

CelebAMask-HQ Dataset



TRAIN 80%

VAL 10%

TEST 10%

2

Model

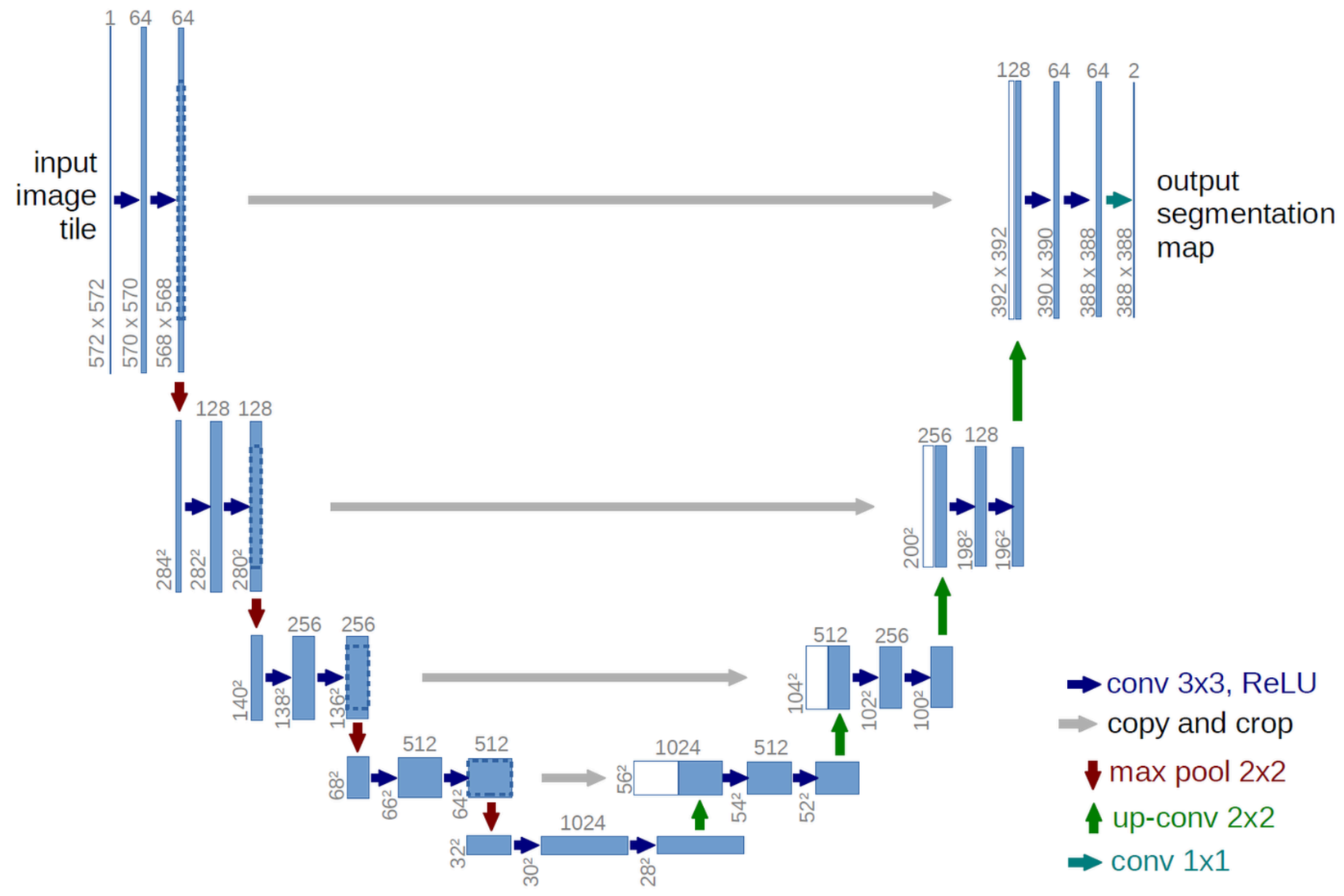
PyTorch

UNet

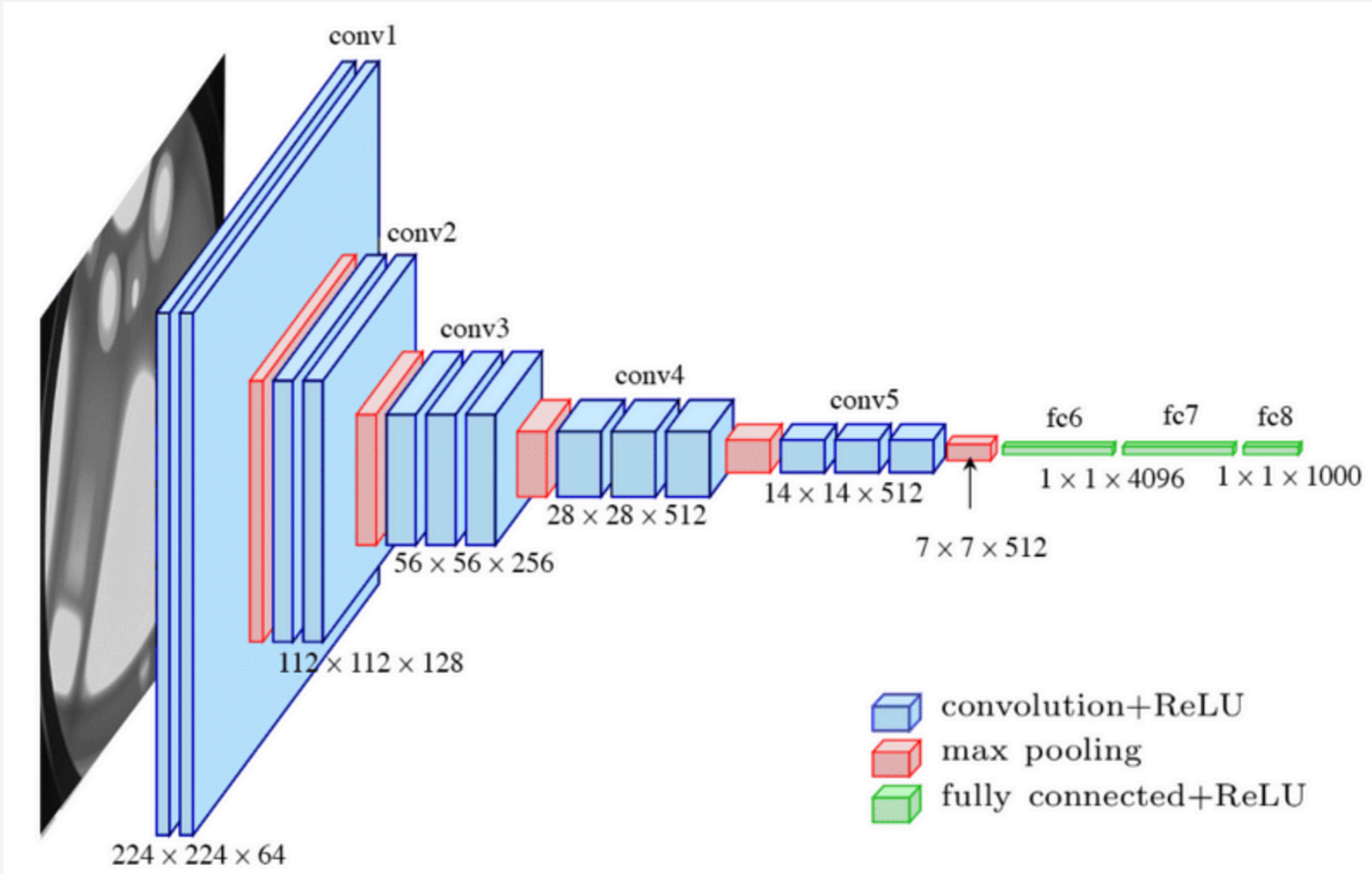
segmentation models pytorch + vgg
(Fine Tuning)

UNet “from scratch”

UNet


input image tile
572 x 572
570 x 570
568 x 568
1 64 64
284²
282²
280²
128 128
140²
138²
136²
256 256
68²
66²
64²
512 512
32²
30²
28²
1024
56²
54²
52²
1024 512
104²
102²
100²
512 256
200²
198²
196²
256 128
392 x 392
390 x 390
388 x 388
388 x 388
128 64 64 2
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

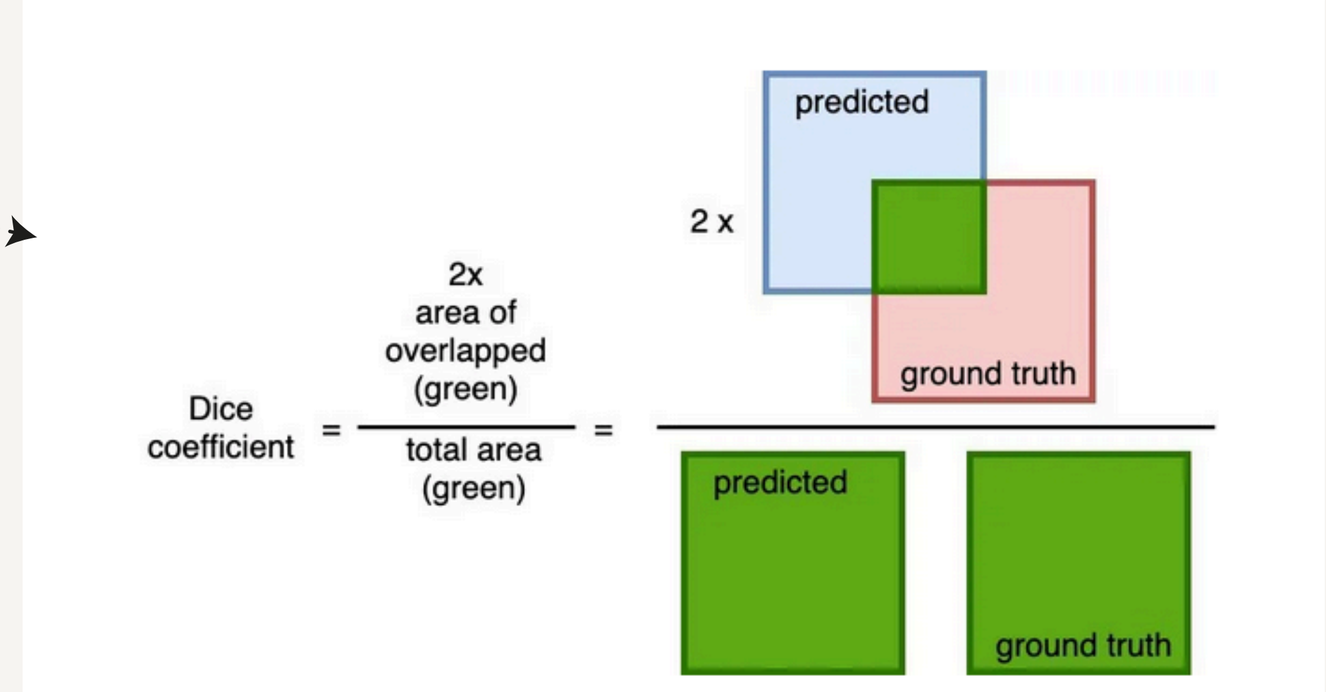
VGG


conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU

Funkcje kosztu

$$\text{BCE Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(\sigma(p_i)) + (1 - y_i) \cdot \log(1 - \sigma(p_i))\right]$$

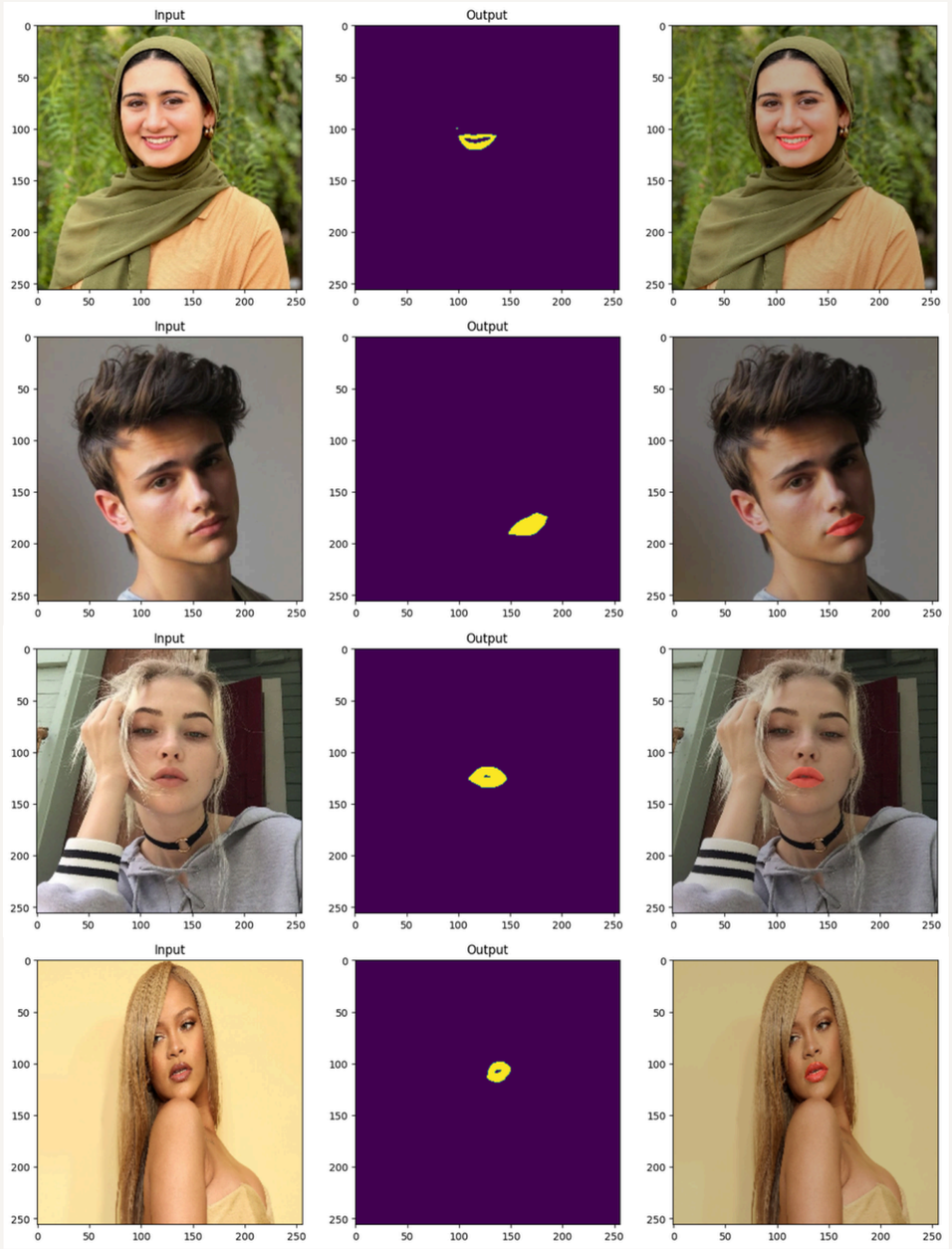
$$\text{DSC} = \frac{2 \cdot \sum_{i=1}^{N} p_i y_i}{\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} y_i^2}$$



- BCE skupia się na pikselach indywidualnie,
- Dice Loss patrzy na całkowite dopasowanie obrazów.

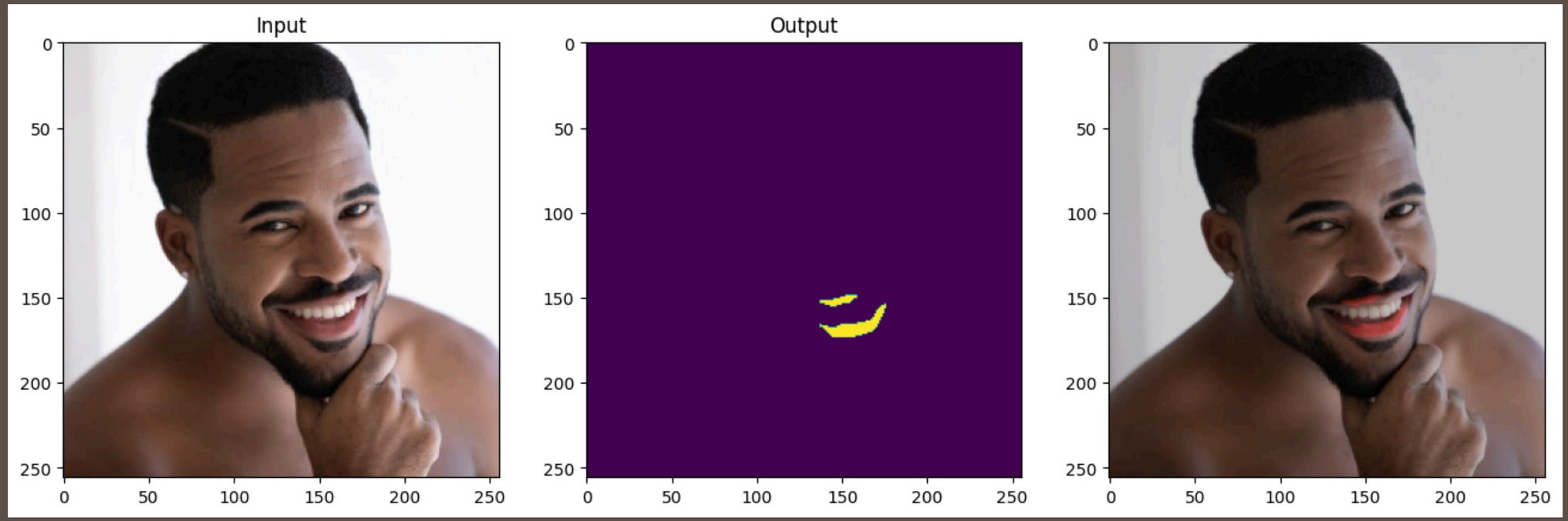
Testowanie i porównanie

UNet + vgg



własny UNet

Testowanie i porównanie



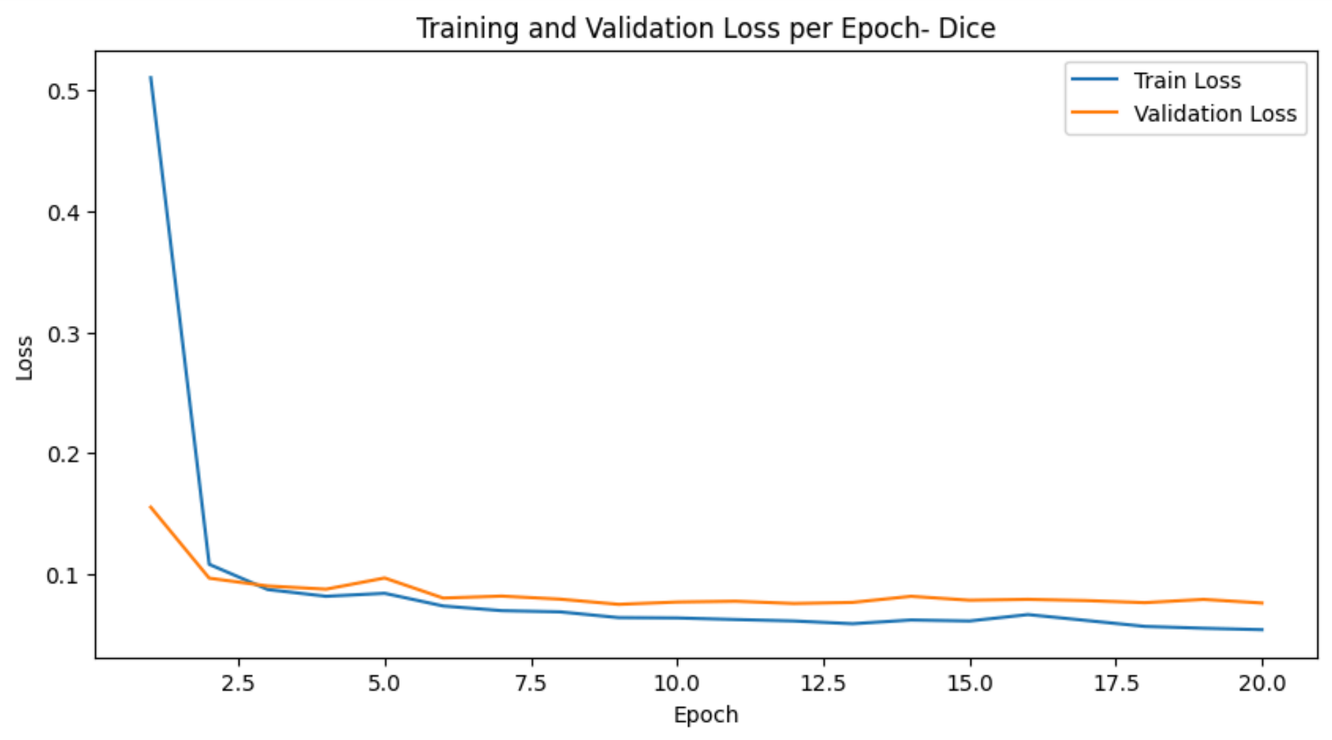
Testowanie i porównanie

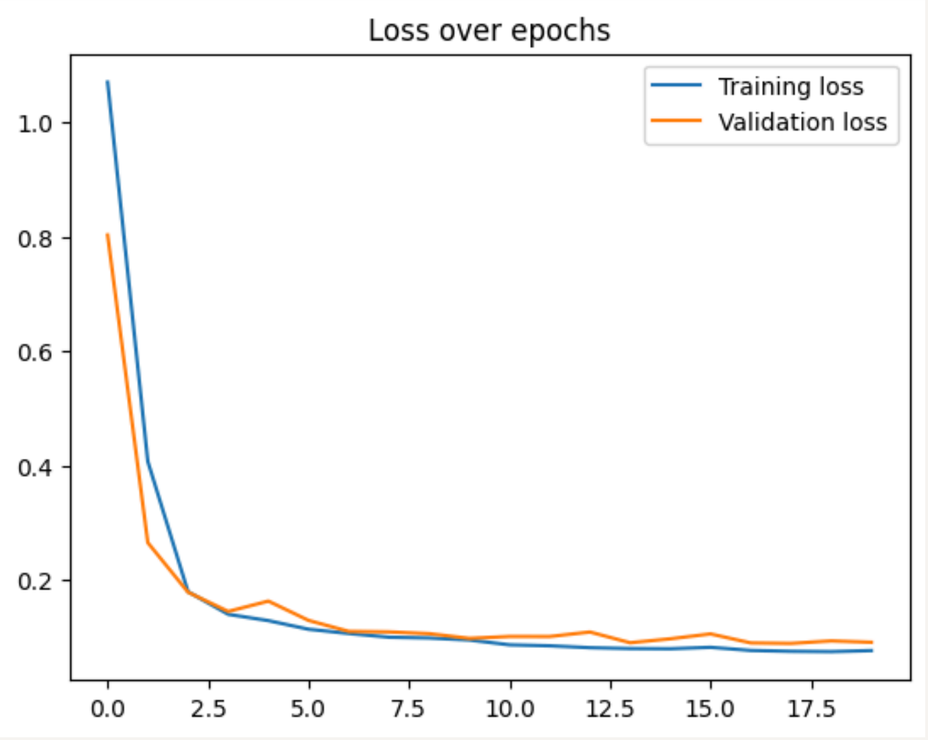
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

Testowanie i porównanie





Mean IoU: 0.8596

Mean Recall: 0.9150

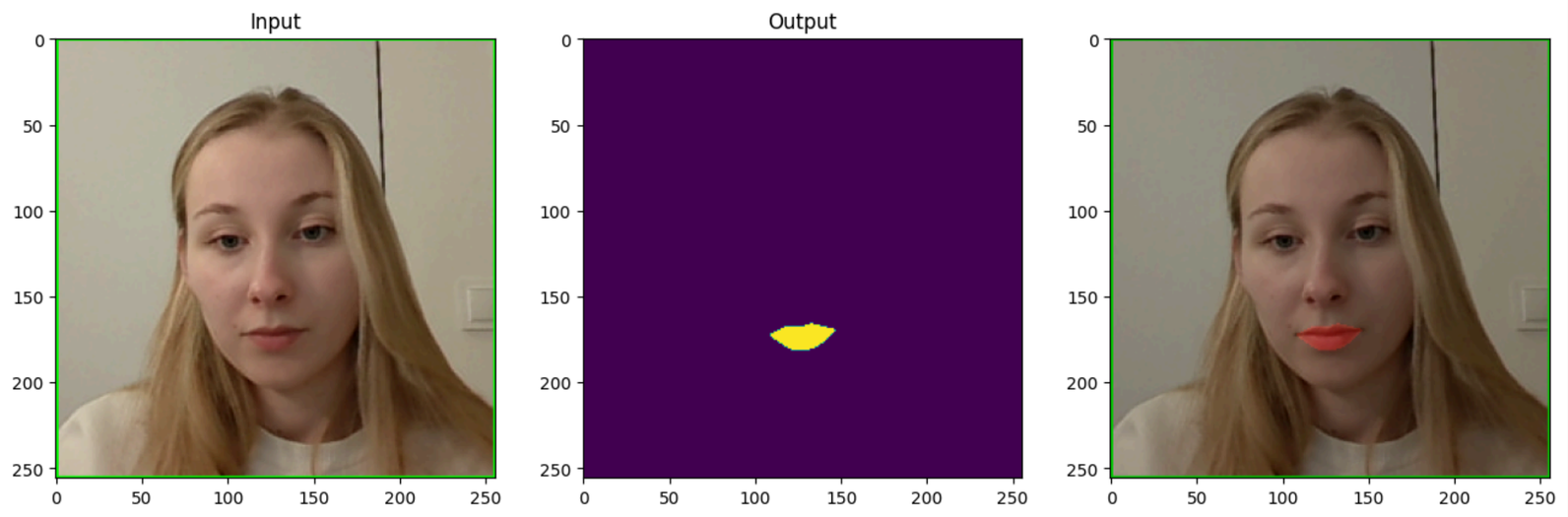
Mean Accuracy: 0.9982

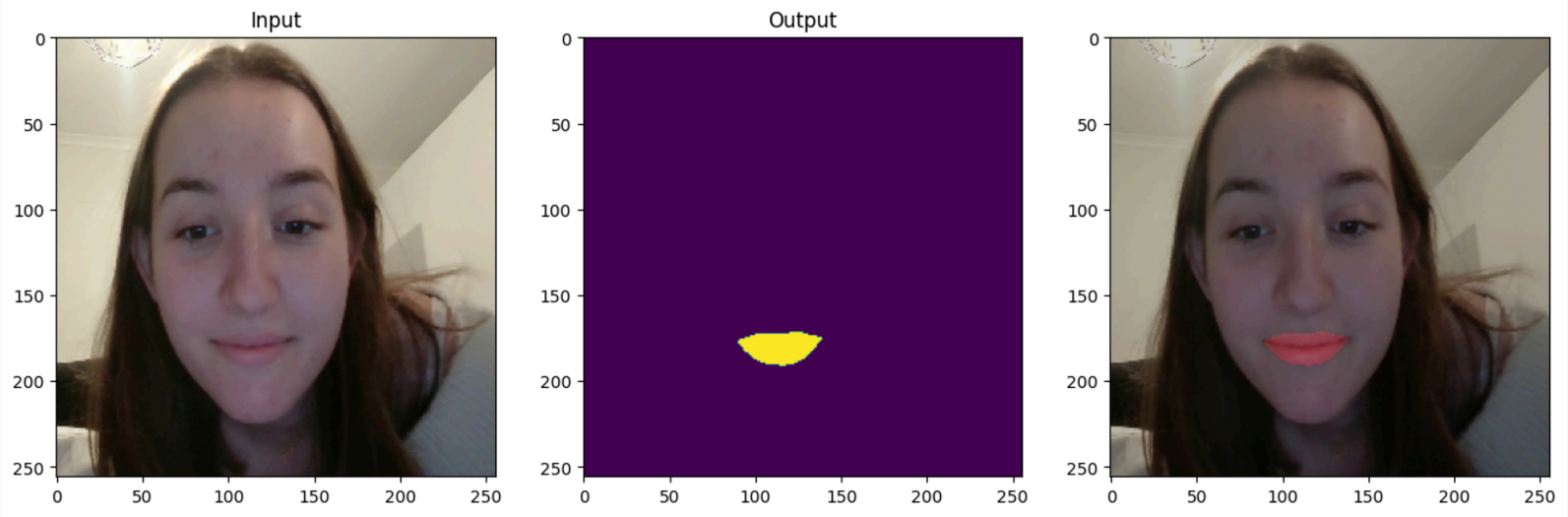
Mean IOU: 0.8221

Mean Recall: 0.8925

Mean Accuracy: 0.9976

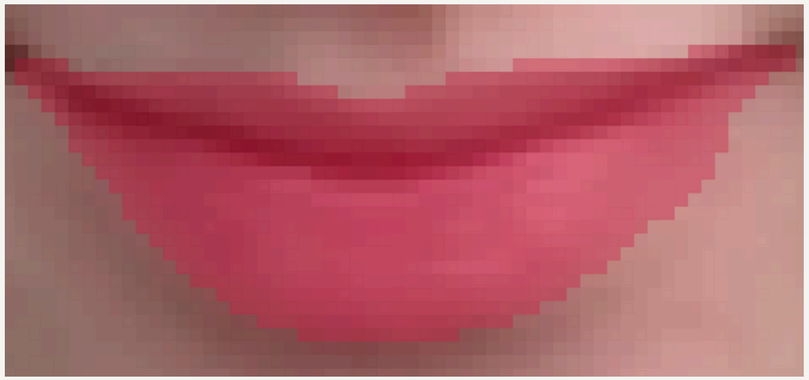
Testowanie i porównanie





3

Nakładanie koloru



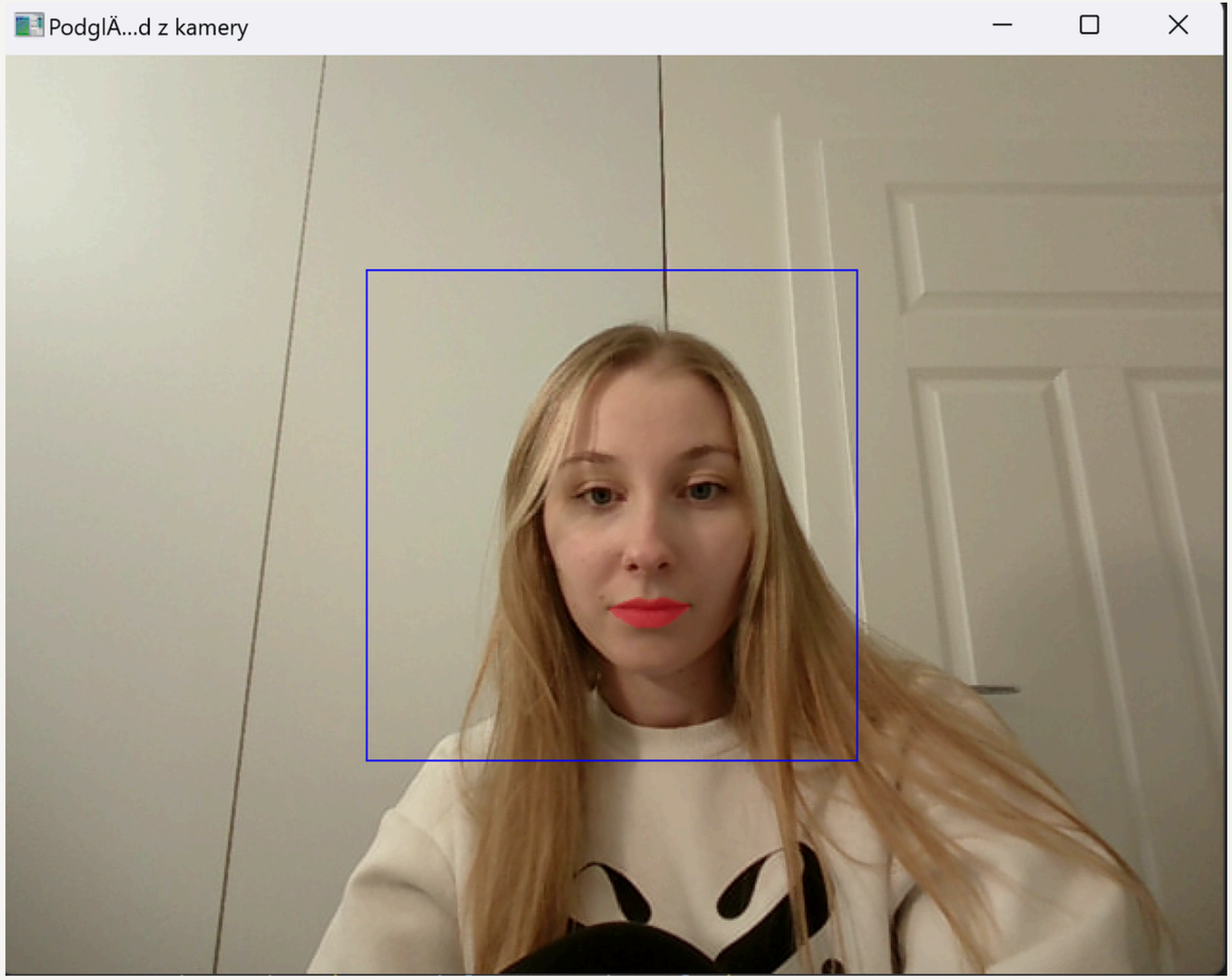
```
def get_mask_on_photo(img, pred, color, alpha=0.3):
    mask = np.zeros_like(img)
    mask[:, :, 0] = pred[:, :, 0] * color[0]    #B
    mask[:, :, 1] = pred[:, :, 0] * color[1]    #G
    mask[:, :, 2] = pred[:, :, 0] * color[2]    #R

    masked_image = img.copy()
    mask_indices = pred[:, :, 0] > 0

    if np.any(mask_indices):
        masked_image[mask_indices] = cv2.addWeighted(
            img[mask_indices].astype(np.float32), 1 - alpha,
            mask[mask_indices].astype(np.float32), alpha, 0
        ).astype(np.uint8)

    return masked_image
```

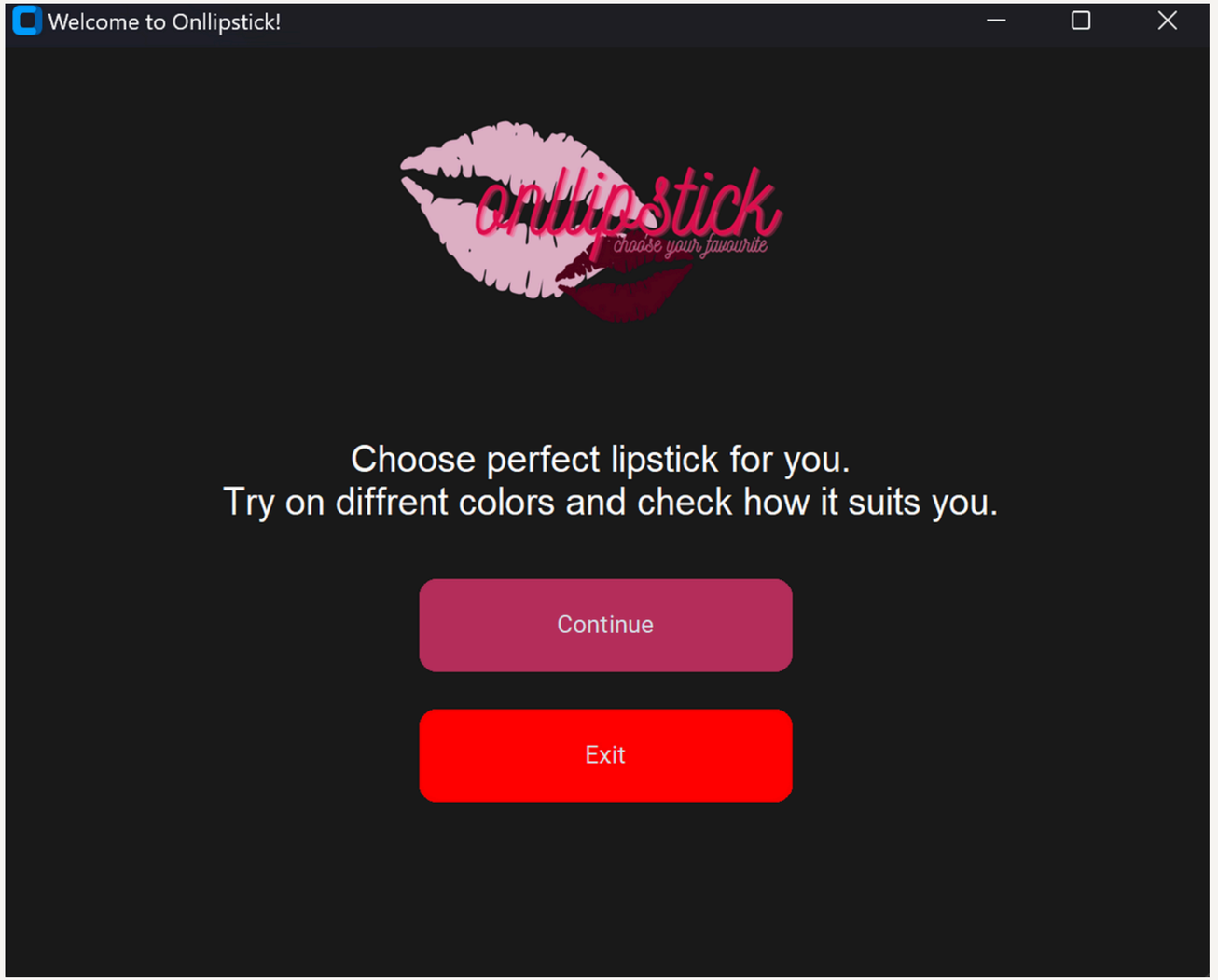
Kamerka - czas rzeczywisty



4

Interfejs graficzny

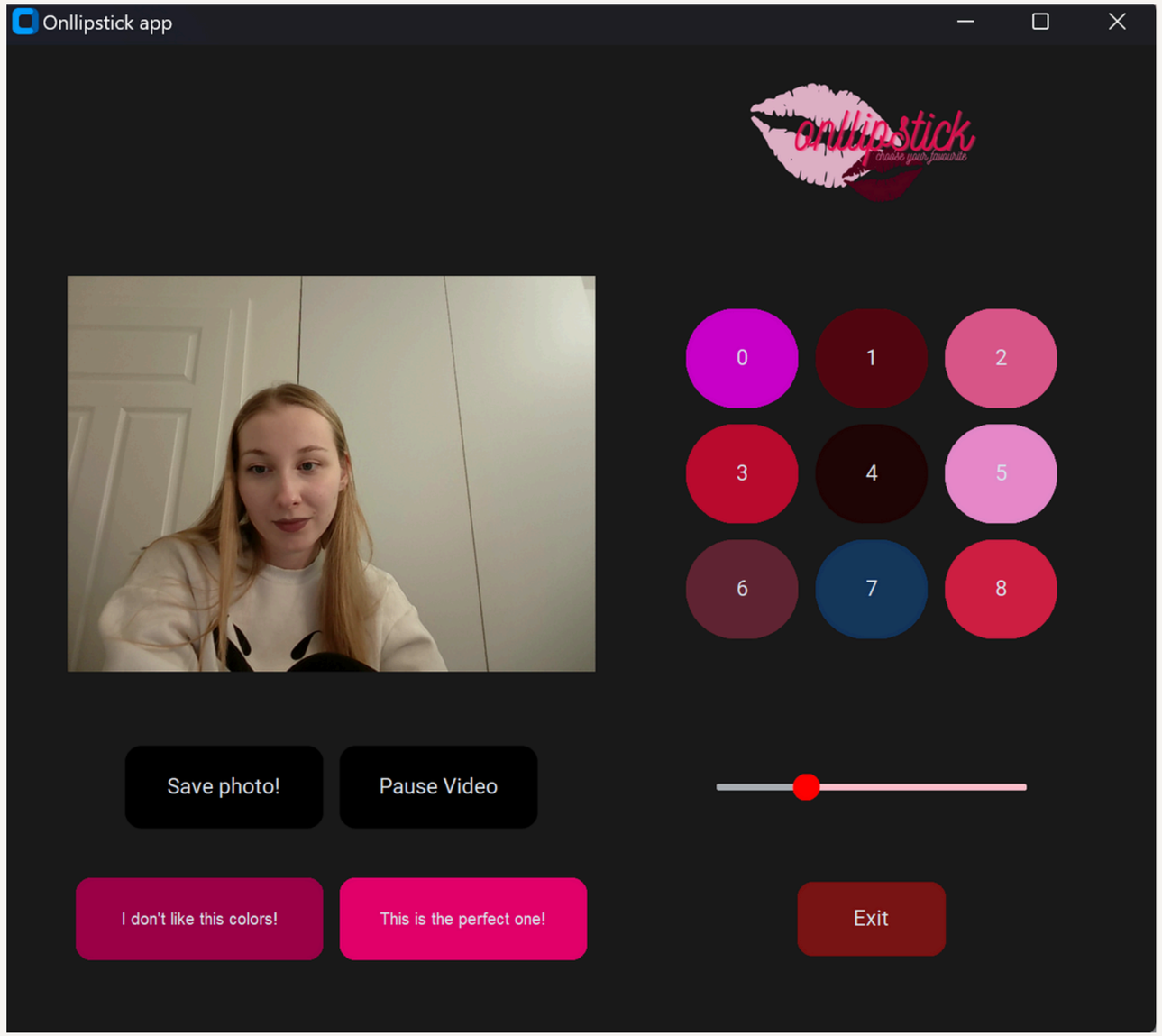
CustomTkinter



4

Interfejs graficzny

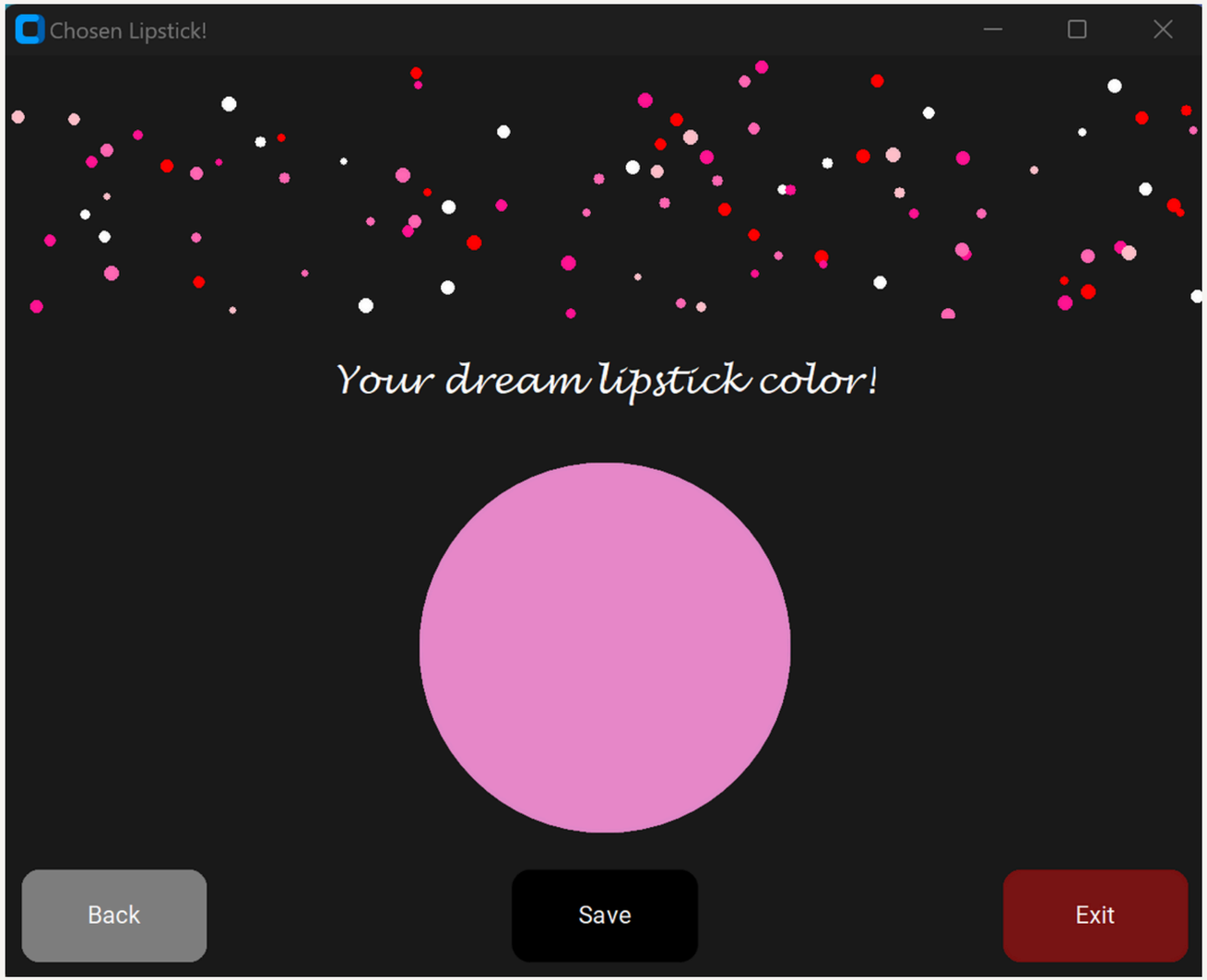
CustomTkinter



4

Interfejs graficzny

CustomTkinter



Dziękuję za uwagę!