Image Processing and Computer Vision (CSC6051/MDS6004) Assignment 1

Bryan Delton Tawarikh Sibarani
121040028
121040028@link.cuhk.edu.cn

Abstract

This report presents the implementation and analysis of multiple tasks in image processing and computer vision. In Task 1, affine transformations, including scaling, rotation, and translation, are applied to images, and their effects are analyzed. Task 2 involves projecting a 3D cube into 2D screen space using camera intrinsic matrices, showcasing the effect of perspective. Task 3 implements a modified version of the PortraitNet architecture for real-time portrait segmentation, using cross-entropy loss and consistency constraint loss for training. Various data augmentation techniques were applied to enhance performance. The model was trained on the EG1800 and Mattinghuman datasets, achieving high accuracy in semantic segmentation tasks. The face detection task is also briefly discussed, using pre-trained models to crop and resize face images. Despite computational constraints, the models demonstrated competitive results, particularly in boundary segmentation and overall accuracy, with mIOU reaching 95.31% on the EG1800 test dataset using EG1800 pretrained and 93.83% using Mattinghuman pretrained. Furthermore, the face parsing implemented using multi-class classification reach 94.38% overall mIOU metrics over all classes.

1. Task 1

Implementation Details Affine transformation is a linear mapping method that preserves points, straight lines, and planes. In this task, we are specifically concerned with scaling, rotation, and translation. The transformation is governed by the following equation:

$$\mathbf{x}' = A\mathbf{x} + \mathbf{t} \tag{1}$$

where:

- $\mathbf{x}$ is the original coordinate vector,
- $\mathbf{x}'$ is the transformed coordinate vector,
- A is the transformation matrix,
- $\mathbf{t}$ is the translation vector.

Scaling is applied by modifying the coordinates of the image through the scaling matrix:

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \tag{2}$$

The rotation of an image by an angle θ is performed around a central point, which is calculated as the center of the image. The rotation matrix is given by:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3}$$

The combined affine matrix for scaling and rotation is computed using the OpenCV function `cv2.getRotationMatrix2D`, which internally combines the rotation and scaling matrices. The translation is then applied to the affine matrix by modifying the third column.
The translation vector $\mathbf{t}$ is applied as:

$$T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{4}$$

Results & Analysis Here is the affine result of specified parameters that is the same result corresponding with the image in the problem set.

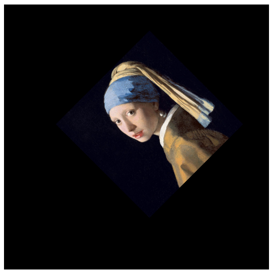
Figure 1. $s = 0.5$, $\theta = 45$, $t_x = 50$, $t_y = -50$

2. Task 2

Implementation Details The camera intrinsic matrix K is fundamental in converting 3D coordinates in camera space to 2D pixel coordinates on the image plane. It is represented by:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where:

- f_x and f_y are the focal lengths along the x and y axes, respectively.
- c_x and c_y are the coordinates of the principal point (usually the center of the image).

The focal lengths are typically specified in pixels, and the principal point is where the optical axis intersects the image plane.
To project a 3D point $\mathbf{x}_c = [X_c, Y_c, Z_c]^T$ from camera space to screen space, we use the following process:

1. **Convert to Homogeneous Coordinates**: A 3D point $\mathbf{x}_c$ is extended to 4D homogeneous coordinates by appending a 1:

$$\mathbf{x}_c^{\text{homogeneous}} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

2. **Apply the Camera Intrinsic Matrix**: We then apply the intrinsic matrix K to transform the point from 3D to 2D coordinates in screen space:

$$\mathbf{x}_s^{\text{homogeneous}} = K\mathbf{x}_c$$

This results in a 3D vector $\mathbf{x}_s^{\text{homogeneous}} = [X_s, Y_s, Z_s]^T$, where Z_s is used to normalize the coordinates.

3. **Normalization**: Finally, we convert back from homogeneous coordinates to 2D pixel coordinates by dividing by the third component Z_s:

$$\mathbf{x}_s = \begin{bmatrix} \frac{X_s}{Z_s} \\ \frac{Y_s}{Z_s} \end{bmatrix}$$

This yields the pixel coordinates on the 2D plane.

The 3D cube consists of multiple points defining its vertices. Each point is projected onto the camera plane using the aforementioned projection process. The resulting 2D coordinates form the cube's projection in screen space, which is visualized as a series of connected faces.

Results & Analysis Using the default camera settings, including focal lengths $f_x = f_y = 70$ and principal point at the image center, the cube is projected onto the 2D plane. The result is a typical perspective projection where the nearer vertices of the cube appear larger, while the farther vertices appear smaller due to the foreshortening effect of perspective. By rotating the cube along its axes, the shape of the projected cube changes. This demonstrates how orientation impacts the appearance of 3D objects in 2D space. For instance, a 30-degree rotation around the x-axis and a 50-degree rotation around the y-axis result in a distorted view of the cube, as shown in Figure 2 below.

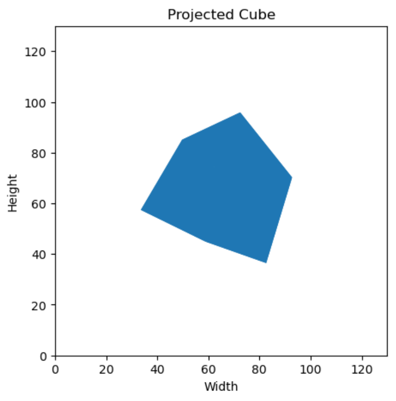

Figure 2. Projected cube (30° around x-axis, 50° around y-axis).

3. Task 3

Model, Loss Function, Evaluation Metrics The encoder-decoder structure of the PotraitNet is implemented adapting the original code of the PotraitNet[1].

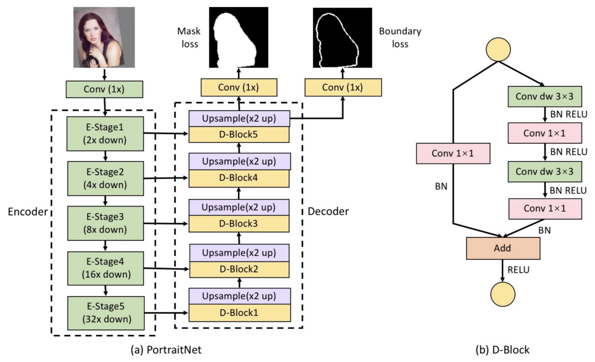

Figure 3. Overview of Original PotraitNet [2]

However, in this project implementation, the decoder structure does not use the *boundary loss* part to reduce the complexity of training. However, as PotraitNet is a lightweight segmentation model, the precision declines compared with the state-of-the-art model [2]. Therefore, even without using the boundary loss, we will still add a *consistency constraint loss*. Therefore, we have two loss function, which are:

1. **Mask Loss**

   Here, we simply using Cross Entropy Loss to measures the difference between two probability distributions: the true distribution (the ground truth labels) and the predicted distribution (the output probabilities from the model).

$$L_m = -\sum_{i=1}^{n} (y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) \quad (5)$$

   where y_i represents the ground truth label of pixel i and p_i represents the predicted probability of pixel i. The predicted probability p_i is computed as:

$$p_i(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad (6)$$

   where z is the original output of the decoder and K is the number of ground truth classes (which in this masking problem, $K = 2$).

2. **Consistency Constraint Loss**

   Using the ground truth semantic segmentation as the supervision signal is simple, with portrait pixels in the image manually marked as 1 and all other pixels as 0. These labels are commonly referred to as hard labels since they consist of only two categories. However, it has been demonstrated that soft labels, which contain additional information, can enhance the model training process [2]. The method here will use data augmentation from the original data to generate soft-label training process.

   For a given original image, we first apply deformation enhancements to generate image A, and subsequently perform texture enhancement on image A to produce A'. Since texture enhancement does not alter the shape of the images, the segmentations of images A and A' remain identical. Let us denote the network output for image A as heatmap B and for image A' as heatmap B'. Theoretically, heatmaps B and B' should be the same. However, due to the texture augmentation techniques, the quality of image A' is inferior to that of A, resulting in the generated heatmap B' being of lower quality than B. Therefore, we utilize the higher quality heatmap B as the soft labels for heatmap B'. Specifically, we introduce a consistency constraint loss between heatmaps B and B', which is formulated as a KL divergence:

$$L_{m'} = -\sum_{i=1}^{n} (y_i \log(p'_i) + (1 - y_i) \log(1 - p'_i)) \quad (7)$$

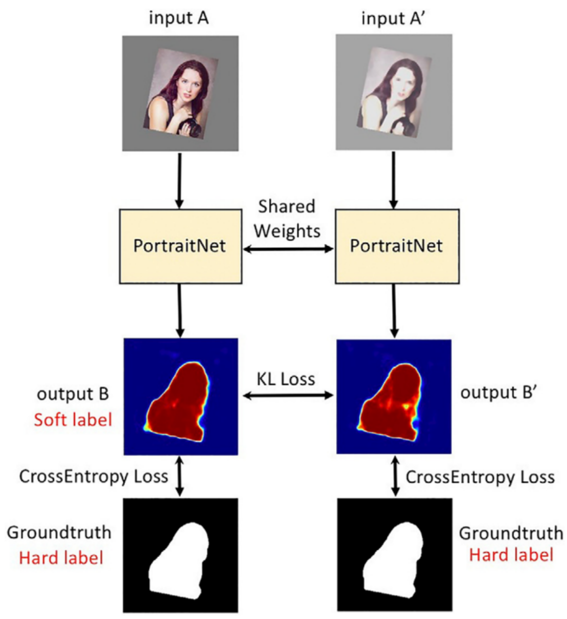

Figure 4. Illustration of Consistency Constraint Loss [2]

$$L_c = \frac{1}{n} \sum_{i=1}^{n} q_i \times \log \frac{q_i}{q'_i} \times T^2 \quad (8)$$

The total loss become:

$$L = L_m + L_{m'} + \alpha \times L_c \quad (9)$$

Here, α is used to balance the two losses. T is used to smooth the outputs. p'_i is defined as follows:

$$p'_i(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad (10)$$

And q_i and q'_i in Eq. (8) are defined similarly:

$$q_i(z_j) = \frac{e^{\frac{z_j}{T}}}{\sum_{k=1}^{K} e^{\frac{z_k}{T}}}, \quad q'_i(z_j) = \frac{e^{\frac{z_j}{T}}}{\sum_{k=1}^{K} e^{\frac{z_k}{T}}} \quad (11)$$

Furthermore, the evaluation process is conducted through mean Interaction-over-Union (IOU) as follows:

$$mIOU = \frac{1}{N} \sum_{i=1}^{N} \frac{maskPD_i \cap maskGT_i}{maskPD_i \cup maskGT_i} \quad (12)$$

where $maskPD_i$ and $maskGT_i$ represent segmentation result and ground truth label of i-th image of test dataset, respectively.

Data Loading and Preprocessing The data loading and processing part is also adapted from the original code of PotraitNet [1]. To enhance the generalizability of the trained model, we employ various data augmentation techniques to supplement the original training dataset, resulting in improved segmentation outcomes. These augmentation methods can be categorized into two types: deformation augmentation and texture augmentation. Deformation augmentation alters the position or size of the target without modifying the texture. Conversely, texture augmentation enriches the texture information of the target while preserving its position and size. The deformation augmentation techniques utilized in our experiments comprise [2]:

- random horizontal flip
- random rotation {-45° to 45°}
- random resizing {0.5 to 1.5}
- random translation {-0.25 to 0.25}

The texture augmentation methods applied include [2]:

- random noise {Gaussian noise, $\sigma = 10$}
- image blur {kernel sizes of 3 and 5 randomly}
- random color change {0.4 to 1.7}
- random brightness change {0.4 to 1.7}
- random contrast change {0.6 to 1.5}
- random sharpness change {0.8 to 1.3}

Each operation in both deformation and texture augmentation was applied with a probability of 0.5 during training. After performing data augmentation, we normalize the input images using the image mean {[103.94, 116.78, 123.68], BGR order} and image val (0.017). The normalization formula is given by (*image* - *mean*) × *val*.

Training Process The experiment runs in Google Colab Free utilizing T4 GPU with batch-size 64, initial learning rate 0.001, weight decay $5e^{-4}$ with initial number of epochs 2000 and image size reduced to 224×224 to speed up training process. However, due to free-tier computation limit, it can only reach 247 epochs with the result as follows.

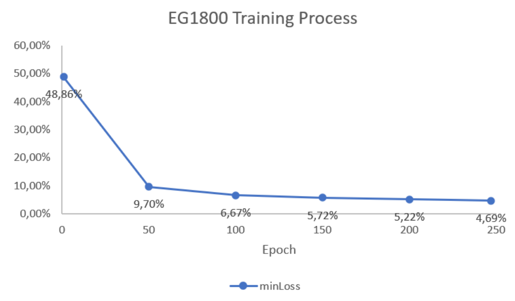

Figure 5. Minimum Loss vs Epoch for EG1800 training

After the model has been trained, the evaluation using $mIOU$ metrics resulting in 95.31% on the EG1800 test dataset.

When performed on custom image, it perform really well, even if the color of the background similar to skin color.

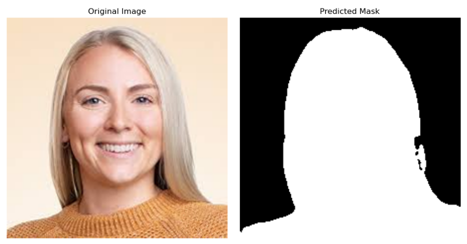

Figure 6. Image from Internet

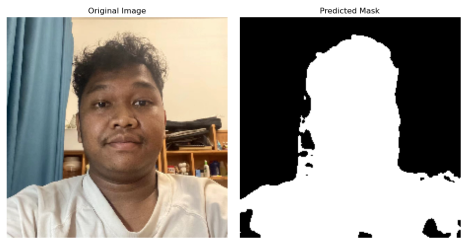

Figure 7. Personal Photo

However, when the model is tested for personal image, some small mistake in boundary segmentation is found. It may occur due to noisy background as well as the model may not be good enough to classify the boundaries as we are not training it with the boundary loss.

Face Detection For the face detection, pre-trained face detection model like OpenCV's Haar Cascades can be used to detect faces in images. Since the task suggests it does not need a full object detection network, a simpler classification or feature-based approach like Haar cascades should suffice. The model will output 4 variables which are x, y, w, h. After detecting faces based on the 4 variables above, crop the image around the detected face(s). Then, the cropped face will be resized to a uniform size (e.g., 224x224 pixels) to ensure consistency for further model input or analysis. By cropping images based on detected faces, you will focus only on the region of interest, which will improve performance on non-centered portrait images.

Figure 8. Face Detection

Large Scale Dataset (Mattinghuman) Pretraining In this task, the data is 10x larger than the previous data, meaning that it requires more computing power to reach the same epoch training. However, as data becomes larger, the model can learn the general pattern better with much smaller epochs. Experimenting with similar setup as EG1800, the training process run as follows:

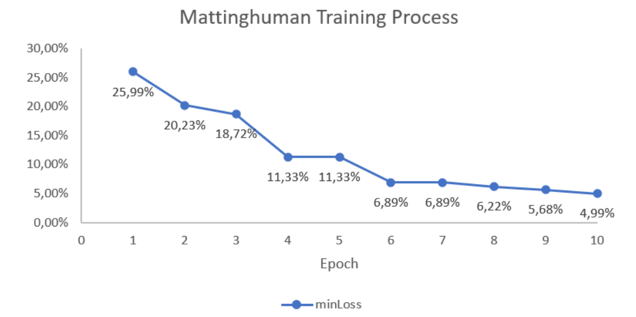

Figure 9. Minimum Loss vs Epoch for Mattinghuman training

We can see at the first epoch, the total loss is already much smaller compared to the first epoch in EG1800. Due to computational limit of free-tier Colab, the model training could only reach 10 epoch. However, it is noticable that the objective loss function at this small number of epochs already achieve similar performance with EG1800 training that runs at 250 epochs. After running evaluation of Mattinghuman pre-trained model using $mIOU$ metrics on the EG1800 test dataset, it achieve in 93.83% which is outstanding for very small number of epochs. If we trained it to the same epochs number as EG1800 training process, it is expected that the accuracy would be astonishing as the model learn a lot of information from larger dataset.

When tested on the personal photo, this model is even better in classifying the boundary compared to the EG1800 pre-trained. The model size is still the same as it contains the same structure and number of weights. Therefore, training on this dataset would achieve a better performance (with comparable number of epochs) as the model can be more general.

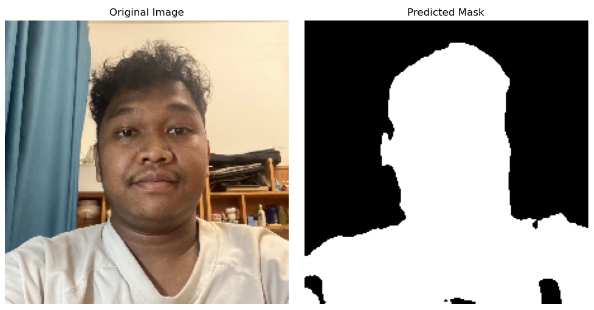

Figure 10. Personal Photo on Mattinghuman

Face Parsing Extension The original PortraitNet model segments only two classes (background and portrait). For facial feature segmentation, it need to be extended to the number of classes in the output layer to accommodate the different facial features. Based on EasyPotrait dataset, it has 8 output class (investigating on annotation, it only has 0-7 labels despite the Kaggle documentation state it has 0-8 labels); therefore, the output number of classes would be changed from 2 to 8. Furthermore, for the loss function, it should be changed to cross entropy loss and not binary cross entropy. However, as the code implementation is for general K number of classes and already using cross entropy loss; there is no need to change the loss implementation at the code. The mathematical definition of mask loss for multi-class cross-entropy is as follows:

$$L_{m_{multi}} = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_i^k \log(p_i^k) \tag{13}$$

where K is the number of classes (in our case is 8), y_i^k is 1 if the true class for pixel i is class k, and 0 otherwise (one-hot encoding), and p_i^k is the predicted probability that pixel i belongs to class k.
Experimenting with similar setup as EG1800, the training process run as follows:

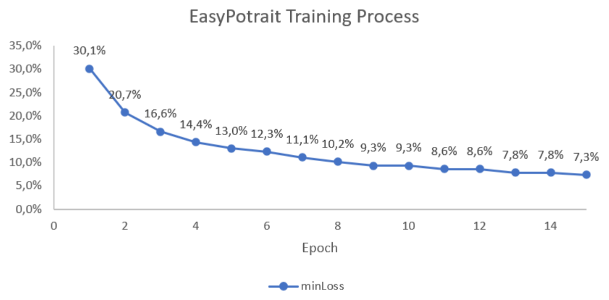

Figure 11. Minimum Loss vs Epoch for EasyPotrait training

Due to computational limit of free-tier, the model training could only reach 15 epochs. After running evaluation of EasyPotrait pre-trained model using mean mIOU metrics over all 8 classes on the EasyPotrait test dataset (different with validation dataset), it achieve in 94.38% which also outstanding for very small number of epochs in multi-class classification.

How to run the code To run the training for binary segmentation (background and potrait), please run through the following command

```
python train.py
```

The default config path is *config.yaml* and it is required to modify the file according to objective and the dataset used.

To run the test, please run through the following command

```
python test.py
```

The default config path is *config_test.yaml* and it is required to modify the file according to objective and the dataset used.

For the multi-class segmentation (EasyPotrait), please run through the following command

```
python train_easypotrait.py
```

The default config path is *config_easypotrait.yaml* and it is required to modify the file according to dataset location.

To run the test, please run through the following command

```
python test_easypotrait.py
```

The default config path is *config_test_easypotrait.yaml* and it is required to modify the file according to objective and the dataset used.

The detail for the custom image is located at *custom_case.ipynb*, while for face detection is located at *face_detection.ipynb*

4. Conclusion

In this assignment, multiple image processing and computer vision tasks were successfully implemented, including affine transformations, 3D-to-2D projection, and semantic segmentation using a deep learning model. The modified PortraitNet achieved promising results with both binary and multi-class segmentation tasks. The experiments on larger datasets, such as Mattinghuman, showed significant improvements in performance even with fewer training epochs due to model learn more information. Although computational limitations restricted further experimentation, the models produced high accuracy and reliable segmentation results, confirming the robustness of the approach.

References

[1] Xin Dong. Potraitnet. https://github.com/dong-x16/PortraitNet, 2019. 2, 3

[2] Song-Hai Zhang, Xin Dong, Hui Li, Ruilong Li, and Yong-Liang Yang. Portraitnet: Real-time portrait segmentation network for mobile device. *Computers Graphics*, 80:104–113, 2019. 2, 3, 4