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

Zijin CAI
Student ID – 224040002
email: 224040002@link.cuhk.edu.cn

Abstract

Task 1 aims to implement a simple image affine transform pipeline, considering rotation, scaling, and translation. The programming result achieves that transformation can be made upon image according to the given parameters (i.e., scale factor, rotation angle and translation distance). Task 2 use the pinhole camera model which performs a perspective projection of 3D cube onto a 2D image plane. The coding results implement projection of 3D cubes, with customizing the camera intrinsic and extrinsic parameters (i.e., rotation angles, center position and focal length). Task 3 reproduces the PortraitNet[1], a highly accurate and efficient model for semantic image segmentation. The model is initially trained and evaluated on EG1800 DataSet, performing tests by visualizing the segmentation results. Further efforts are made to implement face detection and image preprocessing, demonstrating an improved performance on non-centered portrait image. Moreover, the model is investigated on the Matting Human Dataset and EasyPortrait Dataset with robust performances, presenting comparable results and extensions to model segmentation.

1. Task 1 - Image Affine Transformation

Affine Transformation Implementation Details The affine transformation is a combination of linear transformation and translation, such that lines maps to lines, and parallel lines remain parallel, while the ratios are preserved

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{1}$$

$$= \left(\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{2}$$

where s_x and s_y are the scaling factors, θ is the rotation angle, and t_x, t_y are the translation distances[2].

Results & Analysis this method transform the given angle θ to radian $(\pi \times \theta/180)$, then sample (a triangle) points from the given image, and compute corresponding coordinates in the target.

given the scale factor $s = 0.5$, rotation angle $\theta = 45$ and translation distance $(t_x, t_y) = (50, -50)$, the affine transformation is applied to the [pearl.jpeg], and the results are shown in Figure 1.

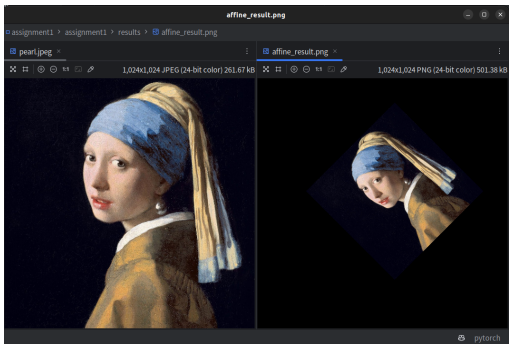

Figure 1. The results of image affine transformation.

Alternatively, the affine transformation can be implemented by using the $cv2.getRotationMatrix2D()$ in-OpenCV, which provides an equivalent but more efficient and accurate transformation.

2. Task 2 - Project 3D Points to Retina Plane

Perspective Projection Implementation Details The pinhole camera model performs a perspective projection of 3D points onto a 2D image plane, with the camera intrinsic and extrinsic parameters.

$$P' = K[R \quad T]P = MP \tag{3}$$

where P is the 3D point in the world coordinate, K is the camera intrinsic matrix, R and T are the rotation matrix and translation vector (extrinsic parameters) of the camera, and P' is the 2D point in the image plane[3].

$$= \begin{bmatrix} \alpha r_1^T - \alpha \cot\theta r_2^T + c_x r_3^T & \alpha t_x - \alpha \cot\theta t_y + c_x t_z \\ \frac{\beta}{\sin\theta} r_2^T + c_y r_3^T & \frac{\beta}{\sin\theta} t_y + c_y t_z \\ r_3^T & t_z \end{bmatrix} \tag{4}$$

where α and β are the focal lengths, (c_x, c_y) is the principal point, r_1, r_2, r_3 are the rotation matrix, t_x, t_y, t_z are the translation vector, and θ is the angle between the optical axis and the image plane[3].

Results & Analysis $get\_camera\_intrinsic()$ is implemented to generate the camera intrinsic matrix K, with the focal length (α, β) and principal point (c_x, c_y). Together with the function $get\_perspective\_projection()$, the 3D cube can thus be projected onto the pixel coordinates, given the 3D points x_c in camera space and the camera intrinsic parameters K.

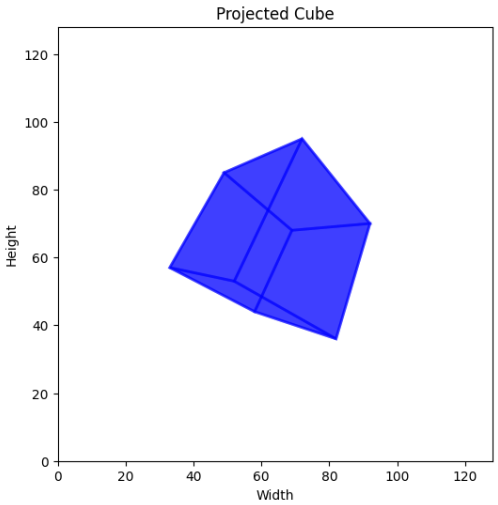

Figure 2. The results of 3D cube projection.

the projection result for the given 3D cube are shown in Figure 2, with the camera intrinsic parameters $(\alpha, \beta) = (70, 70)$, principal point $(c_x, c_y) = (64, 64)$, and the rotation angles $\theta = (30, 50, 0)$ with cube center at $(0, 0, 2)$.

Further experiments are conducted on cube reorientation, center coordinates adjustment and focal length modification, demonstrating the flexibility and robustness of the perspective projection model. (please refer to $[task2.ipynb]$ for details)

3. Task 3 - PortraitNet

PortraitNet is a real-time segmentation model, consisting of some convolutional networks. The model is based on a lightweight U-shape architecture, with two auxiliary losses for training, while no extra cost is required for portrait inference[1].

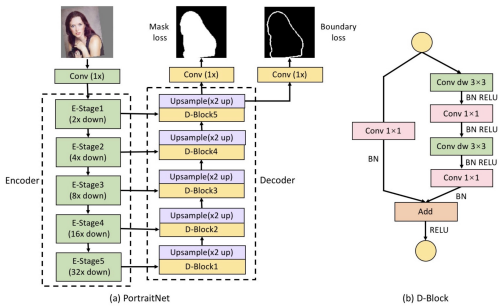

Figure 3. The architecture of PortraitNet.

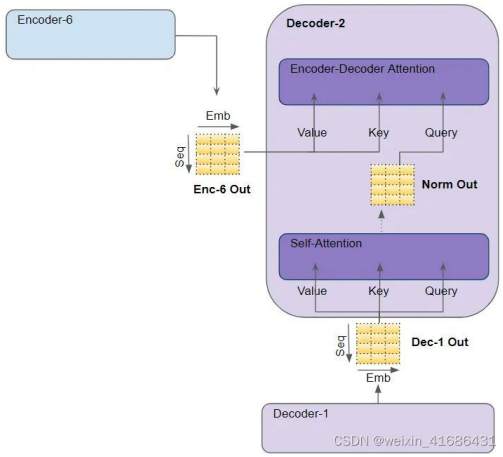

Figure 4. The design of Encoder-Decoder architecture.

A detailed illustration of the PortraitNet architecture is shown in Figure 3.

PortraitNet Implementation - Encoder-Decoder Architecture The Encoder is responsible for extracting features from the raw RGB image, and the PortraitNet-Encoder uses $32x$ down-sampling to obtain large receptive fields and high inference efficiency. Usually, the Encoder employs $MobileNet - v2$ as the backbone network, utilizing depthwise separable convolutions instead of traditional convolutions to reduce the computational cost[4].

The Decoder consists refined residual blocks and upsampling layers, transiting the feature maps by a factor. Upsampling performs de-convolution to the feature maps, and the transition module is modified to reduce the computational cost and improve the segmentation performance

An example diagram (Figure 4) from CSDN illustrates such architecture design [5].

The detailed implementation of such PortraitNet can be found in the $[./task3/portraitnet.py]$ file,

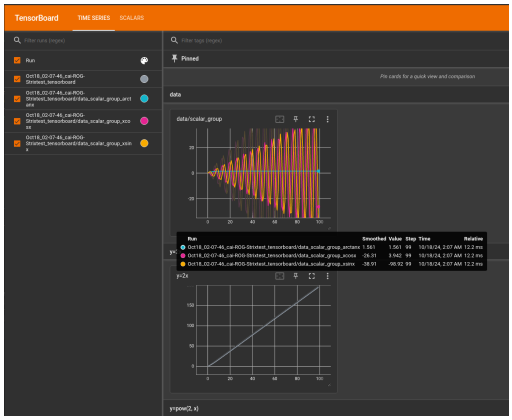

Figure 5. The TensorBoard visualization of training process.

Loss Function and Evaluation Metrics The implementation of the loss function and evaluation metrics are defined in the corresponding $[./task3/loss.py]$.

The $FocalLoss()$ function, widely applied in object detection tasks, is designed to address the class imbalance issue, where some classes have significantly more pixels.

The $calcIOU()$ function, commonly used for image segmentation evaluation, calculates the Intersection over Union (IoU) metric, which is the ratio of the intersection area to the union area of the predicted and ground-truth masks.

The $test()$ function, is thus then test the performance of a model given the dataset

Training and Evaluation on EG1800 Dataset For the specified dataset, the designed python flow [6] retrieves the identifier of the image by stripping the leading/trailing whitespace, and then constructs the path to the image file using $ImageRoot$ and image identifier.

After loading the image and the corresponding mask, the image is resized to grayscale and normalized to the range $[0, 1]$, while retrieving the height, width, and channel.

A bounding box is then generated to crop the image and mask, and the image is augmented by random rotation, scaling, and flipping, while the mask is augmented accordingly.

The TensorBoard helps to visualize the training process as in Figure 5. Assessment of the hyperparameters for better performance then can be conducted by comparing different runs, where lower loss and higher accuracy are generally desirable.

The loss and performance of training are shown in the following log (around 300 epochs). The loss of training is at 0.0451 and the testing loss around 0.0430, Together with the accuracy of 0.9999 (approx. 1) and $6.78e-08$ (approx. 0), this indicates a good convergence and generalization of the model.

(please refer to $[./task3/output.log]$ for details of monitoring the loss and performance metrics)

```
===========>   training    <===========
Epoch: [299][0/46]    Lr-deconv: [0.0]
Lr-other: [0.00048767497911552955]
Loss 0.1190 (0.1190)
0.99989605 6.144205e-08
===========>   testing    <===========
Epoch: [299][0/289]    Lr-deconv: [0.0]
Lr-other: [0.00048767497911552955]
Loss 0.0491 (0.0491)
0.9999552 6.7805765e-08
Epoch: [299][100/289]    Lr-deconv: [0.0]
Lr-other: [0.00048767497911552955]
Loss 0.0741 (0.0628)
0.99985397 1.2288675e-07
Epoch: [299][200/289]    Lr-deconv: [0.0]
Lr-other: [0.00048767497911552955]
Loss 0.0731 (0.0587)
0.9999676 6.894597e-08
loss:  0.04506452709205533 0.043041068380329284
```

The model performance is then evaluated calling the test data loader, the model and experiment arguments. The IoU score measures the model's accuracy in segmenting images, and the results are shown as following:

(more details can be found in $[./task3/test.ipynb]$)

```
===========> loading data <===========
datasetlist:  ['EG1800']
<enumerate object at 0x722791c1dc40>
289
finish load dataset ...
===========> loading model <===========
minLoss:  0.043041068380329284 283
=> loaded checkpoint '/path/to/
model_best.pth.tar' (epoch 283)

mean iou:  0.9569678524998915
```

The model achieves a mean IoU score of 0.9569678524998915 on the EG1800 dataset, indicating a high accuracy and robustness of the model, while likely capturing the nuances needed to segment the images correctly.

Model Performance on Custom Cases The model is then tested on custom cases (for instance, a certain image from EG1800), including non-centered portrait images, which handles background blurring for a single portrait image (Figure 6).

In this example, the segmentation efficiently identify the portrait and background, as we can see such $alphaRGB$

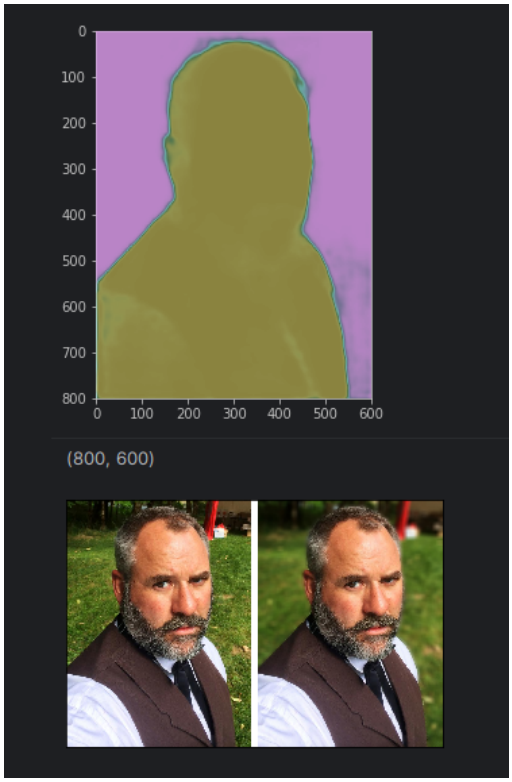

Figure 6. The results of model performance on custom cases.

highlighted portrait (colored). This leads to the satisfactory output with the blurred background and clear portrait, demonstrating the model's robustness and generalization.

However, there are limitations for such segmentation model, such as the inability to handle complex backgrounds or multiple portraits in a single image. We still not guarantee the model can perform well in cases for instance, the resolution of the image is too low or the portrait is too small, or the contrast between the portrait and background is challenging to distinguish.

Generally, the model can be further improved by training on more diverse datasets, or by fine-tuning the model on specific cases to enhance the performance.

Face Detection and Image Preprocessing There are two main methods for face detection, we can either use $DlibHOG$ and $MTCNN$ for pre-trained model, or, employ a simple CNN for binary classification (OpenCV).

Once the face is detected, we can crop the face region using bounding box and paddings, and resize it to a fixed size with augmentation, then feed it into the PortraitNet model for segmentation.

The face detection can thus be implemented by using the $cv2.CascadeClassifier()$ in OpenCV, which provides an efficient and accurate detection of the face region.

Figure 7. The results of face detection and image preprocessing.

The face detection and image preprocessing are then applied to the non-centered portrait image, and the results are shown in Figure 7.

Model Performance on Matting Human Dataset I tried to tackle this task, yet I faced obstacles with the Mattinghuman dataset. The main issues were the complex project hierarchy, and the format discrepancies between the clip and matting images.

Nonetheless, it is generally understood that employing a larger dataset can greatly benefit model training [7]. Larger dataset provides a broader range of samples, capturing nuances that smaller datasets might overlook. These nuances, such as changes in angles or lighting, are essential for enhancing model efficacy in practical scenarios. Greater image exposure to such variations allows the model to generalize better results, which in turn leads to a higher mean Intersection over Union (mIOU) score.

Furthermore, larger datasets increase the model's robustness. By training on a more diverse set of scenarios, the model is less likely to overfit and can maintain consistent performance across various settings. However, this comes at the cost of more computational resources and longer training times for processing extensive data. Effective data management and streamlined training methods are crucial to mitigate these challenges.

Face Parsing Extension on EasyPortrait Dataset While trying to enhance the PortraitNet model to recognize more specific facial features such as eyes and teeth using the EasyPortrait dataset, I encountered the challenge that prevented me from completing the project.

Nevertheless, I was able to pinpoint some crucial modifications [8] needed for the output layer and loss functions:

- **Modification of the Output Layer:** The initial PortraitNet model could suggest a binary classification output layer, designed for distinguishing broad area segmentation like face and non-face. Thus, to identify more detailed features, here are some reasonable alteration of the output layer:

  - The last layer should be changed to a multi-class segmentation output, classifying each pixel into a particular facial feature category (e.g., eyes, teeth, skin).
  - The output tensor should be reshaped to [N, C, H, W], with N representing the batch size, C for the number of classes, and H × W for the image size.

- **Adjustment of the Loss Function:** Implement class weighting in conjunction with Focal Loss to address the imbalance between smaller features (like eyes and teeth) and larger ones (such as skin), ensuring the model accurately learns to segment the smaller features.

4. Conclusion

I really appreciate that the **MDS6004 - Image Processing and Computer Vision** course has provided me with a comprehensive understanding of the fundamental concepts and practical applications in the field. This course is certainly more valuable than any other courses I have taken.

However, the work of this assignment is really intensive and time-consuming (Figure 8), I would like to beg for your understanding and leniency for the incomplete parts.

(mercy marking will be greatly appreciated, thank you!) :)

References

[1] Dong, X. "PortraitNet GitHub Repository." (), [Online]. Available: https://github.com/dong-x16/PortraitNet.

[2] Li Jiang, "Lecture 5: Image Processing II," School of Data Science (SDS), The Chinese University of Hong Kong, Shenzhen, Shenzhen, China, Tech. Rep. CSC6051 / MDS6004, 2024.

[3] L. Jiang, "Camera model ii," *School of Data Science (SDS), The Chinese University of Hong Kong, Shenzhen*, 2014, Lecture Notes for CSC6051 / MDS6004: Image Processing and Computer Vision.

[4] S.-H. Zhang, X. Dong, H. Li, R. Li, and Y.-L. Yang, "Portraitnet: Real-time portrait segmentation network for mobile device," *Computers Graphics*, vol. 80, pp. 104–113, 2019, ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2019.03.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849319300305.

[5] weixin41686431, *Transformer-encoder-decoder*, Accessed on: 2024-03-18. [Online]. Available: https://blog.csdn.net/weixin_41686431/article/details/137041301.

[6] Yang, S. "PortraitNet$_{p}y3GitHubRepository$." (), [Online]. Available: https://github.com/SimonHanYANG/PortraitNet_py3.

[7] AISegment.com. "AISegment.com - Matting Human Datasets." (), [Online]. Available: https://www.kaggle.com/datasets/laurentmih/aisegmentcom-matting-human-datasets.

[8] Kapitanov, I. "EasyPortrait: Face Parsing Portrait Segmentation." (), [Online]. Available: https://www.kaggle.com/datasets/kapitanov/easyportrait.

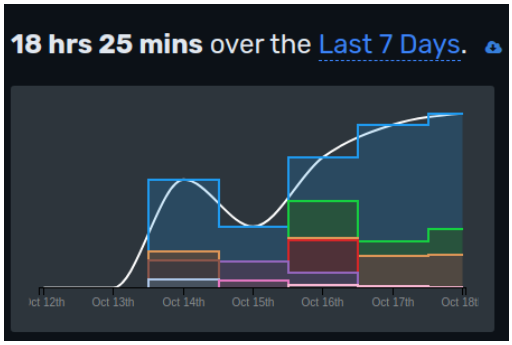

Figure 8. WakaTime - Coding Activity Report