CSC6051 / MDS6004 – Image Processing and Computer Vision – Assignment 1

1 Programming Environment and Hardware Requirements

All tasks in this assignment must be completed using Python. Python is widely used in machine learning and deep learning, with rich library and framework support. You may choose either PyTorch or TensorFlow as your deep learning framework. While the example code in this document is primarily based on PyTorch, you may choose the framework according to your preference. **Note that if you opt for PyTorch, you must use version 2.0 or higher**.

Model training for this assignment requires GPU acceleration. All experiments are configured for a single GPU. You can choose to use either a personal computer equipped with an NVIDIA GPU or the free GPU resources provided by Google Colab.

Google Colab (`https://colab.research.google.com`) is a free cloud-based Jupyter notebook environment that provides GPU support. It offers several advantages, including no need for a local GPU, pre-installed deep learning libraries and frameworks, and the ability to write and run code directly in the browser.

When using Google Colab, please be aware that GPU usage time is limited for each session, a Google account is required for login, and it's recommended to save your code and data to Google Drive for easy management and persistence.

To optimize your use of Google Colab for this assignment, follow these steps:

1. **Prepare your Google environment:**
   - Create a Google account if you don't already have one
   - The required datasets (Mattinghuman and EasyPortrait) are available directly from Google Drive. You don't need to upload them yourself. Use the following links to access them:
     - Mattinghuman, EasyPortrait
   - To use these datasets, you'll need to add a shortcut to them in your own Google Drive
2. **Set up Google Colab:**
   - Mount your Google Drive by executing the following code:

     ```
     from google.colab import drive
     drive.mount('/content/drive')
     ```

   - Copy the dataset to the Colab environment using this command (approximately 4 minutes):

     ```
     ! cp /content/drive/MyDrive/ML_Datasets/matting_human_sample.zip
     /content/sample_data
     ```

   - Use the following command to extract the dataset (around 3 minutes):

     ```
     ! unzip /content/sample_data/matting_human_sample.zip /path/to/data
     ```

   - **Important:** Due to resource limitations on Colab, keep input images to a reasonable size.

If you choose to use a personal computer, ensure that your machine is equipped with an NVIDIA GPU, CUDA and cuDNN are correctly installed, and you have installed a GPU version of PyTorch or TensorFlow compatible with your CUDA version.

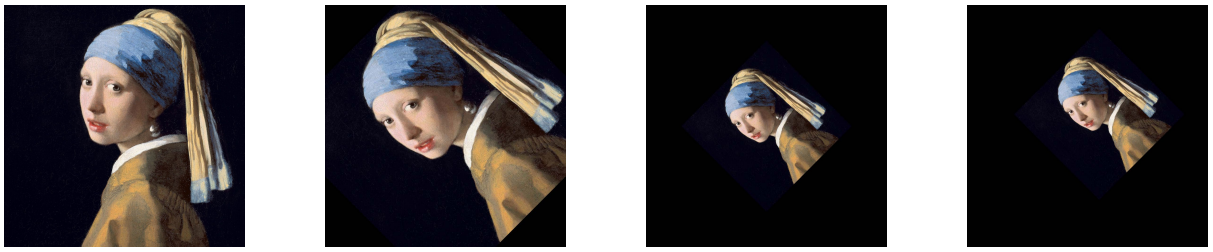
2 Task 1: Image Affine Transformation (10 Points)

2.1 Background

This section aims to implement a simple image affine transformation, only considering scale, rotation, and translation. Given an image I, a scale factor s, a rotation angle θ, and a pair of translation distance t_x and t_y, the implemented function means to output the transformed image $\hat{I}$ according to s, θ, t_x and t_y.

2.2 Todo List:

Implement the image transformation function with the parameters scale, rotation, and translation. The scale ranges from 0.0 to 2.0, the rotation angle ranges from 0 to 360 (anticlockwise), and the translation distance from -100 to 100. Some examples are shown in Figure 1.



Input | $s = 1, \theta = 45, t_x = 0, t_y = 0$ | $s = 0.5, \theta = 45, t_x = 0, t_y = 0$ | $s = 0.5, \theta = 45, t_x = 50, t_y = -50$

Figure 1: The visualization of our input and outputs.

2.3 Hints

There exist multiple solutions to realize the proposed target. Here we would like to recommend one approach which computes the affine transformation matrix first. Specifically, it consists of the following steps:

- Transform the given angle θ to radian ($\pi * \theta / 180$).
- Sample three points (a triangle) in the given image, and compute corresponding point coordinates in the target.
- Calculate transformation matrix using six points with function `cv2.getAffineTransform()`.
- Obtain the target image by using transform matrix with function `cv2.warpAffine()`.

Important! If you follow the given approach to implement your program, you need to note the coordinate system used in the image rotation. When we process images by using cv2 or other packages, the original point is located in the left-up corner of the image. Meanwhile, we usually conduct the rotation operation referring to the center of the image as the original point like Figure 2.

3 Task2: Project 3D Points to Retina Plane (20 points)

3.1 Background

This section aims to use the pinhole camera model which performs a perspective projection. We will project a 3D cube onto a 2D camera plane using this model.

3.2 Todo List:

- Complete the function `get_camera_intrinsics` which returns a 3x3 camera matrix for provided focal lengths and the principal point. (5 points)
- Develop the function `get_perspective_projection`, which takes in a 3D point in camera space x_c and the camera matrix K and returns the point in screen space (i.e. the pixel coordinates) x_s. (5 points)

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2, 0)	(2, 1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

cv2.imread

(-2,-2)	(-2, -1)	(-2,0)	(-2, 1)	(-2,2)
(-1,-2)	(-1, -1)	(-1, 0)	(-1, 1)	(-1,2)
(0, -2)	(0, -1)	(0,0)	(0,1)	(0,2)
(1,-2)	(1, -1)	(1, 0)	(1, 1)	(1,2)
(2,-2)	(2, -1)	(2, 0)	(2, 1)	(2,2)

In Fact

Figure 2: The coordinate systems of the pixel location used in cv2 (left) and image rotation. For better illustration, we assume the image size is 5×5.

- Plot the projected cube, using the default settings as illustrated in Figure 3. (5 points)
- Experiment with different settings. (5 points)
  - Cube orientation: Rotate the cube to observe changes in its projected shape
  - Cube position: Adjust the cube's center coordinates relative to the camera
  - Focal length: Modify the camera's focal length to simulate different fields of view

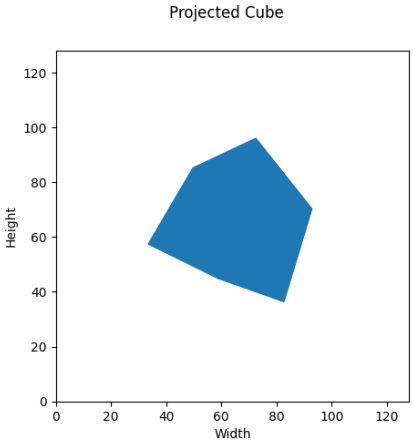


Figure 3: Perspective projection of a 3D cube. Parameters: cube center at (0, 0, 2), rotation angles [30°, 50°, 0°], scale=1.0, camera focal lengths $\alpha = \beta = 70$ pixels and cx = cy = 64 pixels.

4 Task3: PortraitNet (70 points)

4.1 Background

In this section, you need to reproduce PortraitNet by yourself which is a semantic image segmentation network with high accuracy and efficiency. PortraitNet consists of some convolution network, hence you can use any deep learning library(e.g. Pytorch and TensorFlow).

4.2 Todo List:

- Implement the Encoder-Decoder structure of PortraitNet. (10 points)
- Implement the loss functions and evaluation metrics. (10 points)

- Train and evaluate the model. (15 points)
  - Use the EG1800 dataset for training and evaluation
  - Implement data loading and preprocessing pipelines. Then train the model, monitoring loss and performance metrics
  - Perform validation and testing, reporting final performance metrics
- Demonstrate model performance on custom cases. (5 points)
  - Apply the trained model to images from the internet or test the model on self-portraits or personally collected images
  - Visualize and analyze the segmentation results
  - Discuss any limitations or interesting observations from these custom cases
- Face Detection and Image Preprocessing (10 points)
  - Implement face detection to locate faces in images
  - Develop a preprocessing pipeline to crop and resize images based on detected faces
  - Demonstrate improved performance on non-centered portrait images
  - Hints: A full object detection network is not necessary as there is only one target; a classification approach may suffice.
- Large-scale Dataset Pretraining. (10 points)
  - Train your model on the larger Mattinghuman dataset (10x larger than EG1800)
  - Analyze performance changes on the EG1800 validation set and real-world images
  - Compare and discuss the impact of increased training data on model performance
  - Investigate the effect of model size when using larger datasets
- Face Parsing Extension. (10 points)
  - Extend the model to segment facial features (e.g., teeth, eyes) using the EasyPortrait dataset
  - Describe necessary changes to the output layer and loss functions of PortraitNet
  - Implement and evaluate the extended model (mIOU)

4.3 Reference

1. PyTorch. (n.d.). Learn the Basics. PyTorch Tutorials. `https://pytorch.org/tutorials/beginner/basics/intro.html`
2. PyTorch. (2024). PyTorch Documentation. `https://pytorch.org/docs/stable/index.html`
3. PortraitNet repository. `https://github.com/dong-x16/PortraitNet`
4. Mattinghuman dataset.
   `https://www.kaggle.com/datasets/laurentmih/aisegmentcom-matting-human-datasets`
5. EasyPortrait dataset.`https://www.kaggle.com/datasets/kapitanov/easyportrait?resource=download`

5 Submission Guidelines

Your submission must include:

`report.pdf`: A comprehensive project report (maximum 6 pages including references) detailing your work. Only the provided LaTeX template (CVPR style) is allowed. The report should include: Detailed procedures with explanatory images (if needed), final pseudo-code, relevant mathematical theory, your thought process, thorough experiments and results analysis, and **a brief introduction to your code usage (Important!)**.

`code/`: A directory containing all your code for this assignment.

`checkpoints/`: Required checkpoints. For large files, provide a properly shared OneDrive link. Inaccessible links count as non-submission.

Grading: 60% report, 40% code. **Plagiarism is strictly prohibited and will be checked. Ensure your code runs without errors.**

Submission Process: Rename your folder as `<your student ID>-<your name>-Asgn1`, compress it to `<your student ID>-<your name>-Asgn1.zip`, and upload to the blackboard system.

Please read these guidelines carefully. Late submissions will incur penalties: 1 day late: -20 marks; 2 days: -40 marks; 3 days: -100 marks.