

Assignment 1:

# Scale Invariant Feature Detection and Image Filtering

Computer Vision  
NTU, Spring 2023

Announced: 111/03/10 (Fri.)

**Due: 111/03/30 (Thu.) 23:59**

# Outline

## Part 1: Scale Invariant Feature Detection

- Implement Difference of Gaussian (DoG)

## Part 2: Image Filtering

- Advanced color-to-gray conversion
- Implement bilateral filter

## Rules

- Environment
- Report
- Submission
- Grading
- Late Policy

# Outline

## Part 1: Scale Invariant Feature Detection

- Implement Difference of Gaussian (DoG)

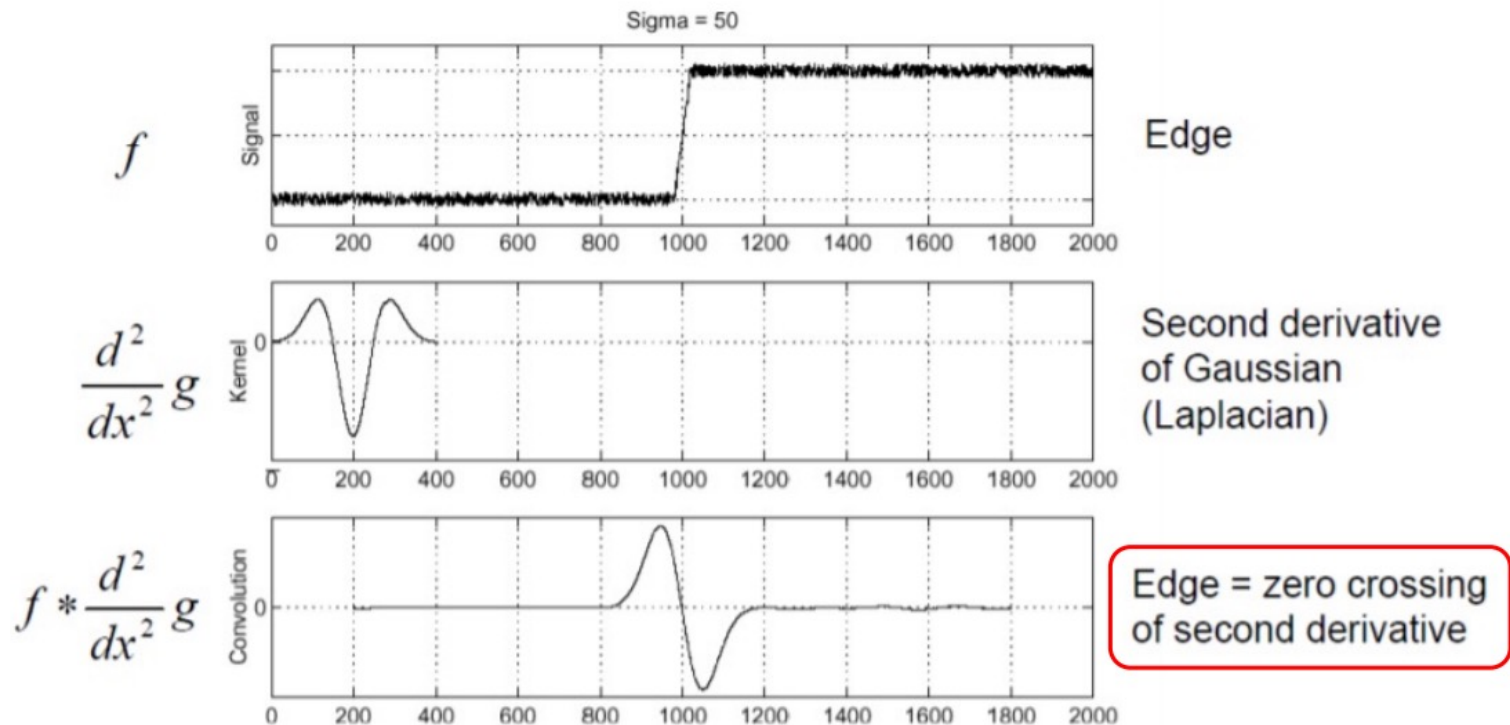
## Part 2: Image Filtering

- Advanced color-to-gray conversion
- Implement bilateral filter

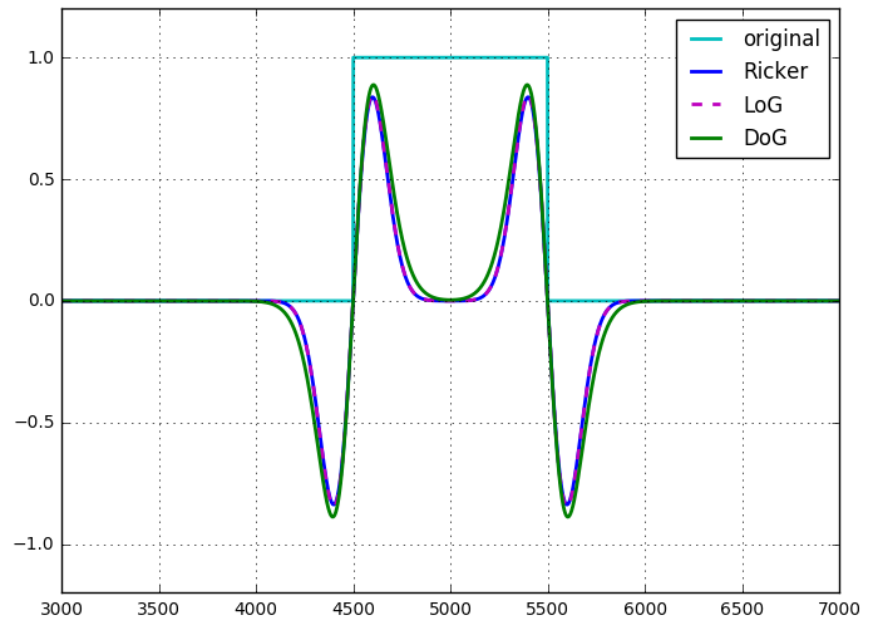
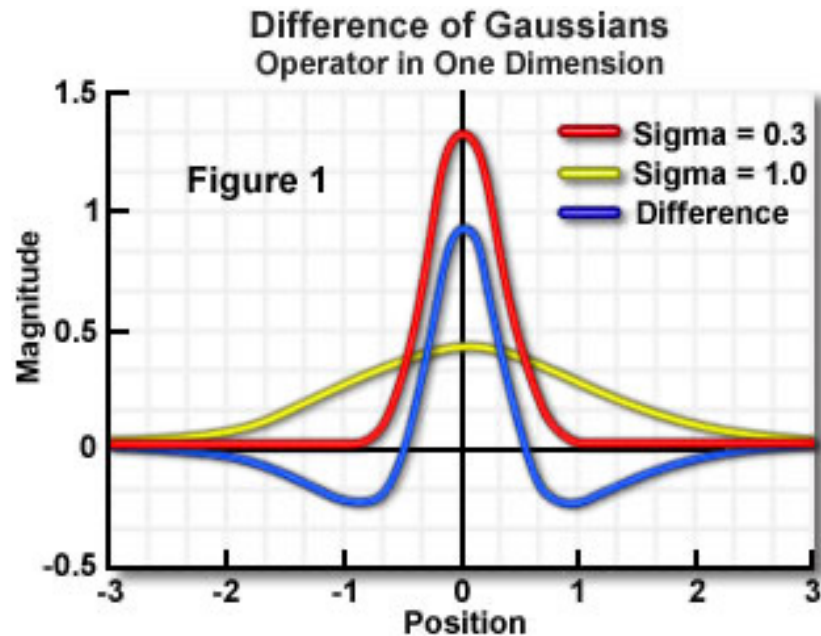
## Rules

- Environment
- Report
- Submission
- Grading
- Late Policy

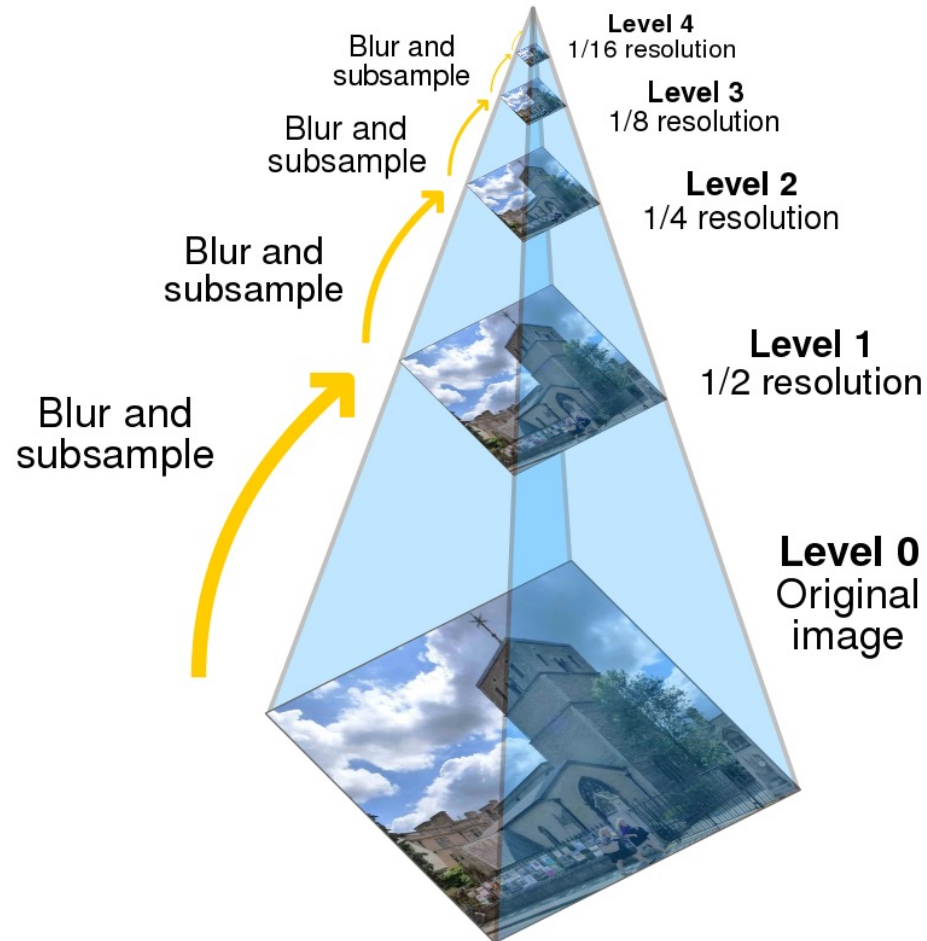
# Laplacian of Gaussian



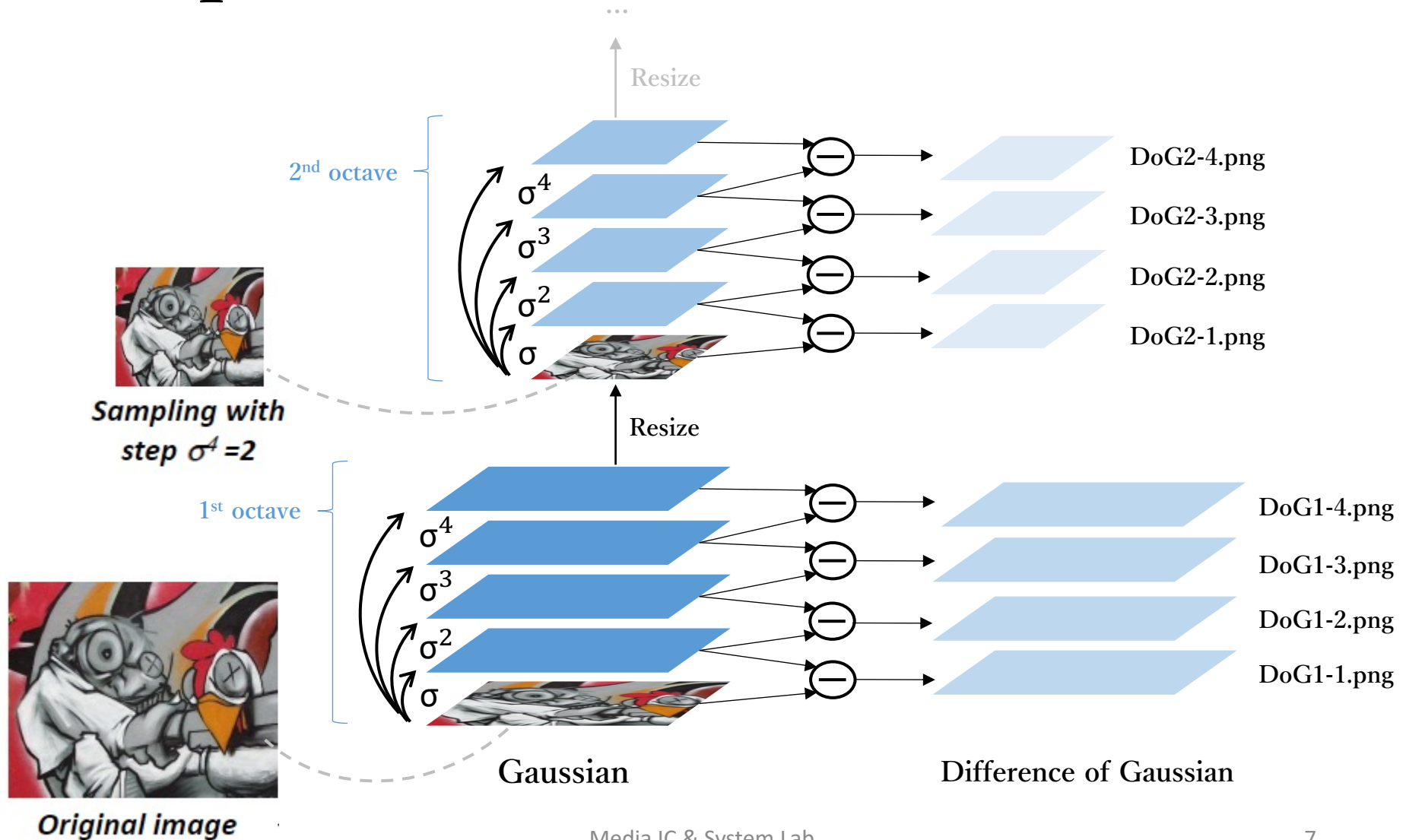
# Difference of Gaussian Filter



# Gaussian Pyramid



# Implementation

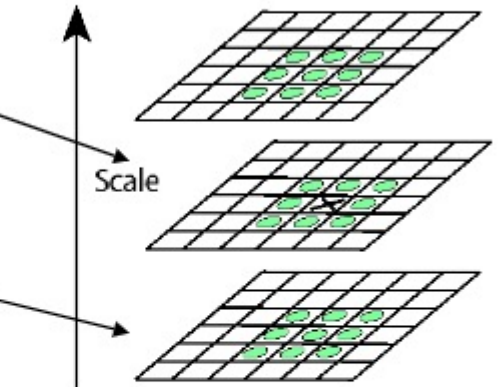
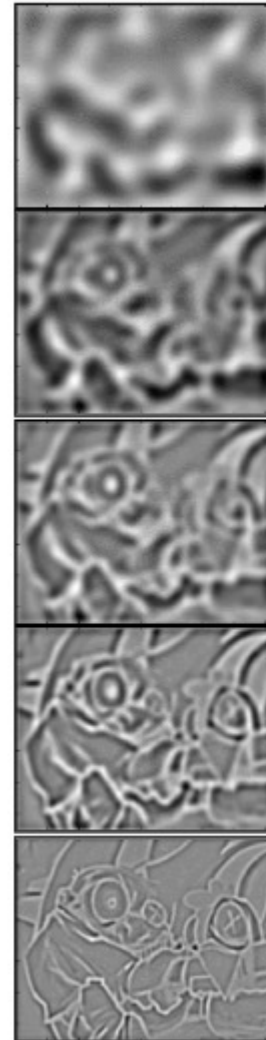


# Implementation

- Find local extremum (maximum and minimum)
- Padding is not needed



Gaussian Blur  
+  
Difference



List of (x, y)  
or  
(2x, 2y) if resize to 1/2



# Implementation

## In DoG.py

```
# Step 1: Filter images with different sigma values (5 images per octave, 2 octave in total)
# - Function: cv2.GaussianBlur (kernel = (0, 0), sigma = self.sigma**__)
```

- Apply gaussian blur with corresponding sigma value on the input.
- The base image of the 1<sup>st</sup> octave is the input image.
- Down sample the last blurred image in the 1<sup>st</sup> octave as the base image of the 2<sup>nd</sup> octave.

```
# Step 2: Subtract 2 neighbor images to get DoG images (4 images per octave, 2 octave in total)
# - Function: cv2.subtract(second_image, first_image)
```

- You should subtract the second image (more blurred one) to the first image (less blurred one) to get DoG.

```
# Step 3: Thresholding the value and Find local extremum (local maximum and local minimum)
#         Keep local extremum as a keypoint
```

- Find the local extremum and threshold the pixel value.

```
# Step 4: Delete duplicate keypoints
# - Function: np.unique
```

- Delete duplicate keypoints from 2 octaves.

# Assignment Description

- `part1/eval.py`
  - TA will run this code to evaluation your result.
  - **DO NOT Modify!**
- `part1/main.py`
  - Execute DoG, visualize results for report, ... etc.
- `part1/DoG.py`
  - Follow the instructions and implement Difference of Gaussian.
    - The output format should be a numpy array with shape  $(x, 2)$
    - $x$  is the number of feature points

# Assignment Description

- Recommended steps

1. Implement Difference of Gaussian in DoG.py
2. Use eval.py to evaluate your DoG.py

```
$ python3 eval.py --threshold 3.0 --image_path ./testdata/1.png --gt_path ./testdata/1_gt.npy
```

- Your Result needs to match Ground truth

```
[Info] All keypoints match.
```

3. Finish remaining code in main.py if needed
  - e.g. visualize DoG, plot feature points

# Outline

## Part 1: Scale Invariant Feature Detection

- Implement Difference of Gaussian (DoG)

## Part 2: Image Filtering

- Advanced color-to-gray conversion
- Implement bilateral filter

## Rules

- Environment
- Report
- Submission
- Grading
- Late Policy

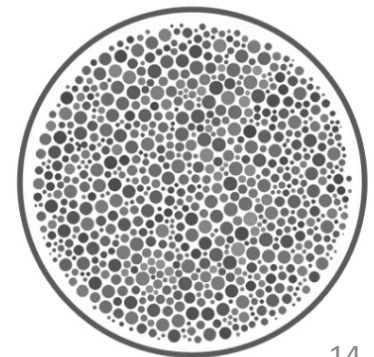
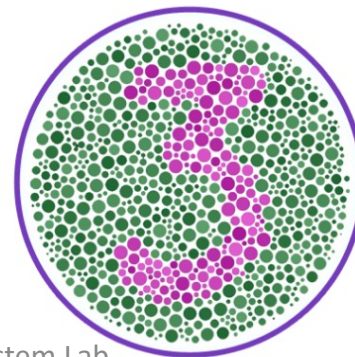
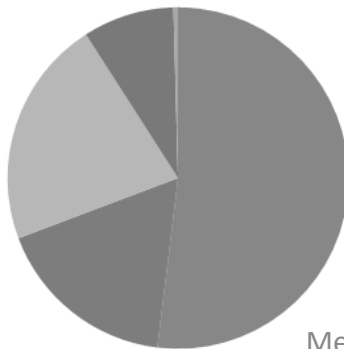
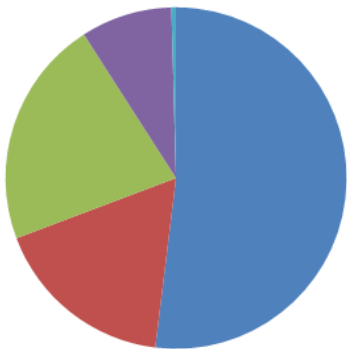
# Color Conversion

- RGB2YUV
  - Read <https://en.wikipedia.org/wiki/YUV> for more details

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix},$$
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}.$$

- Many vision systems only take the Y channel (luminance) as input to reduce computations

# Problems

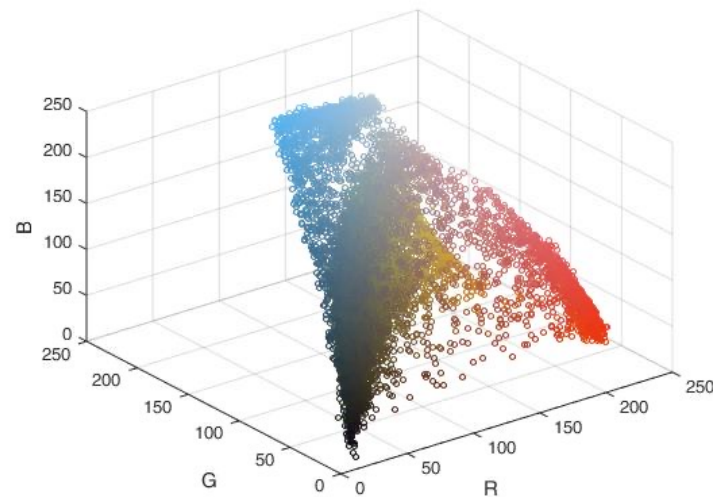
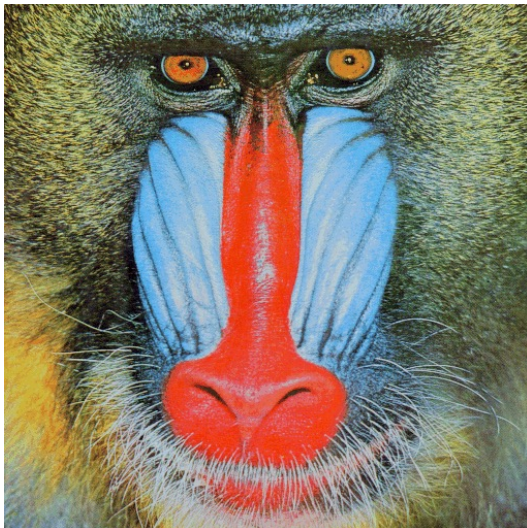


# What happened?

- Dimensionality reduction

$$Y = 0.299R + 0.587G + 0.114B$$

- Another view:
  - The conversion is actually a plane equation! All colors on the same plane are converted to the same grayscale value.



# Finding a better conversion

- The general form of linear conversion:

$$Y = w_r \cdot R + w_g \cdot G + w_b \cdot B$$

$$w_r, w_g, w_b \geq 0$$

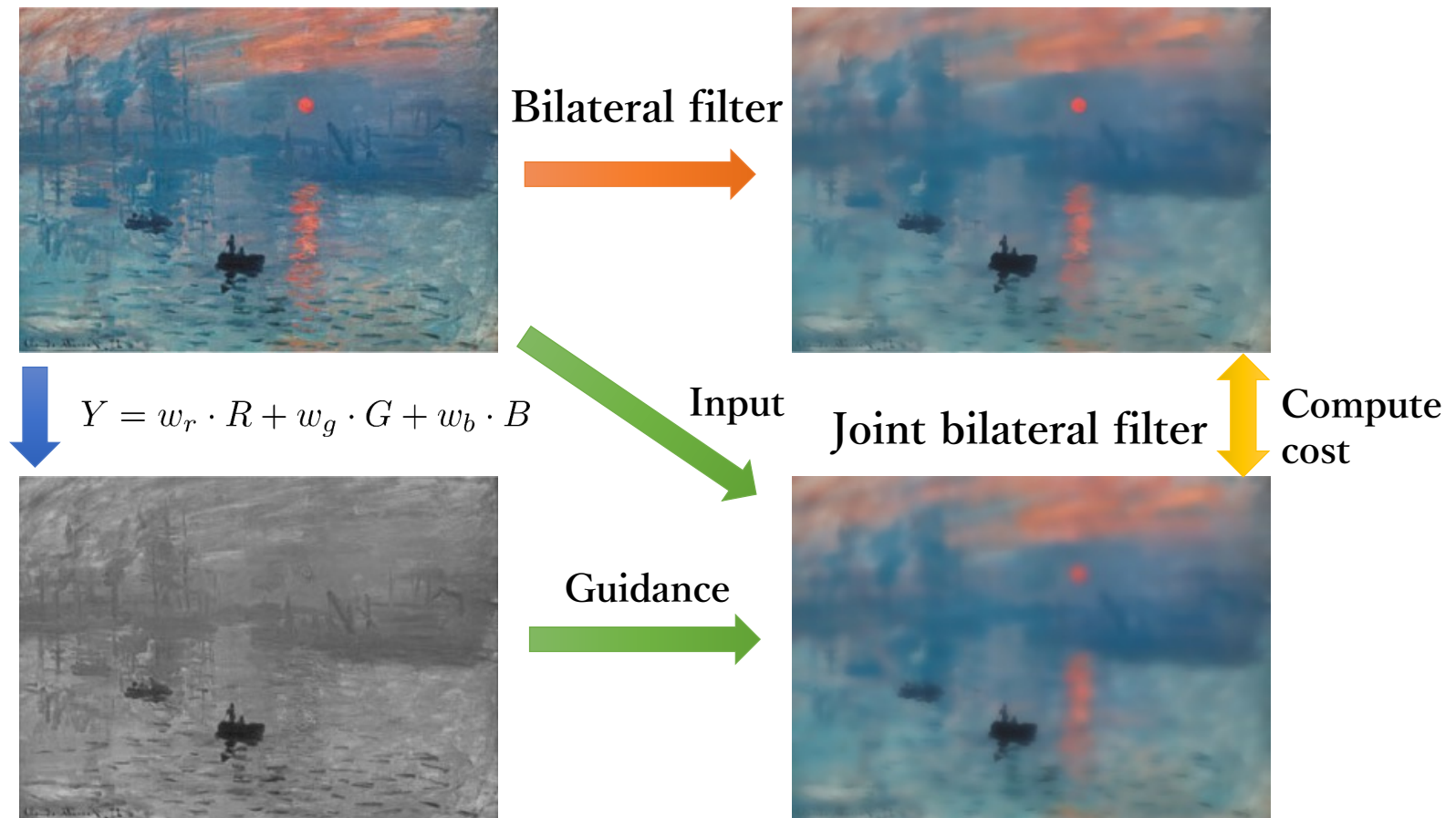
$$w_r + w_g + w_b = 1$$

- Let's consider the quantized weight space  $w \in \{0, 0.1, 0.2, \dots, 1\}$ 
  - For example:  $(w_r, w_g, w_b) = (0, 0, 1)$   
 $(w_r, w_g, w_b) = (0, 0.1, 0.9)$
  - Given a color image, a set of weight combination corresponds to a grayscale image candidate.
  - We are going to identify which candidate is better!



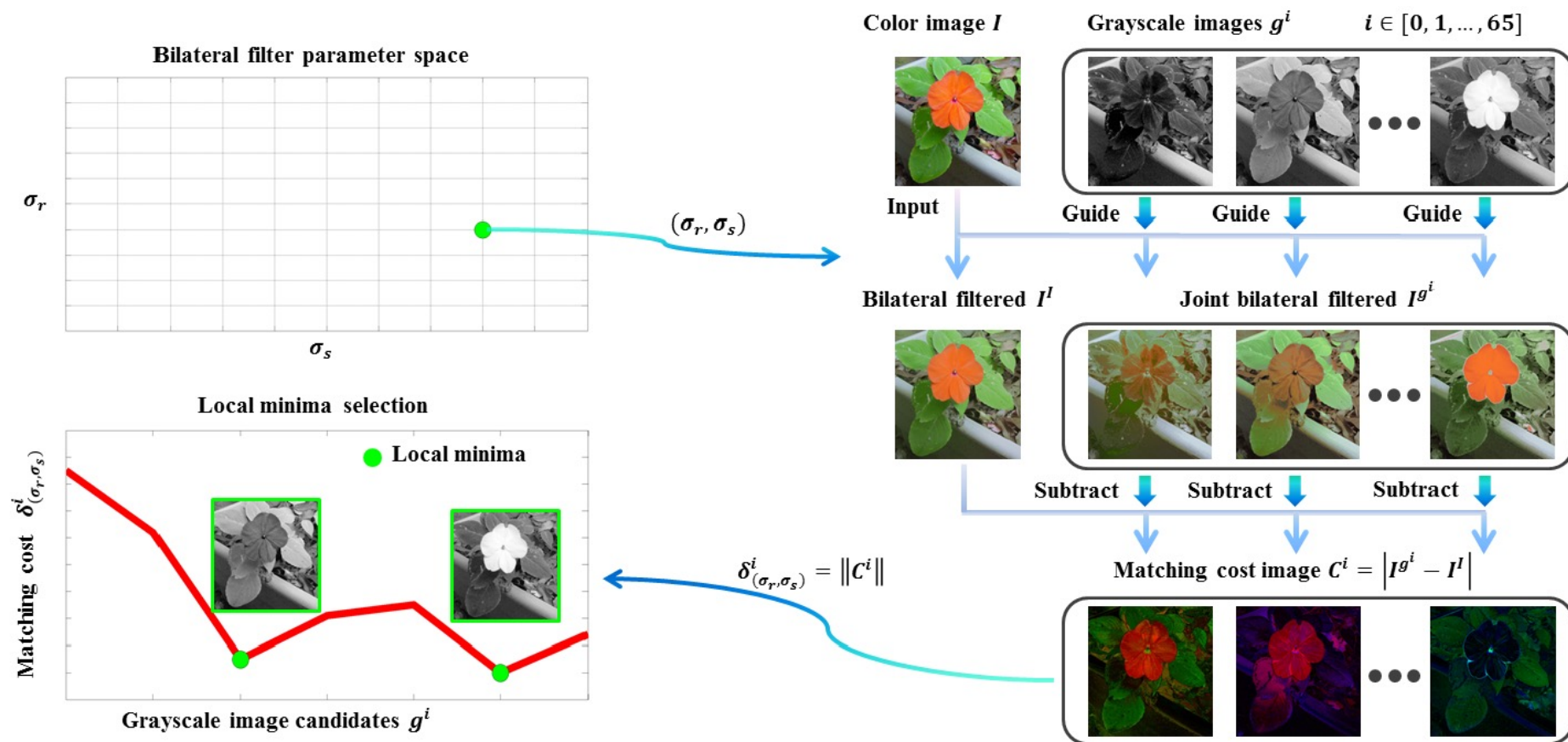
# Measuring the perceptual similarity

- Joint bilateral filter (JBF) as the similarity measurement



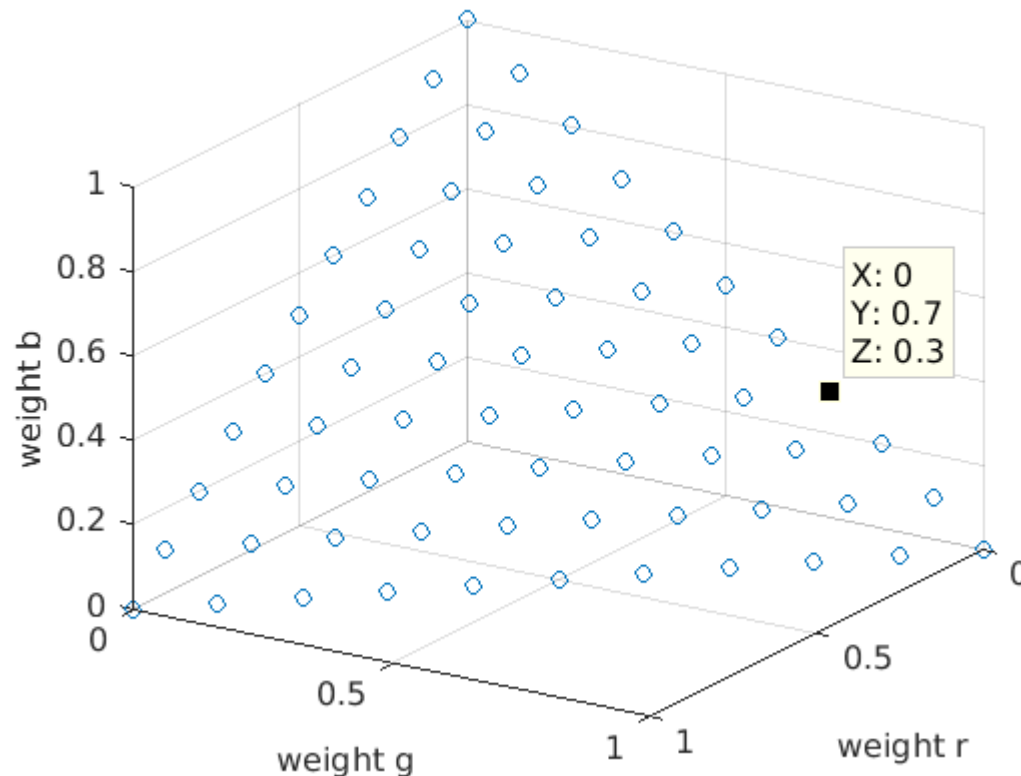
# Measuring the perceptual similarity

- Joint bilateral filter (JBF) as the similarity measurement



# Measuring the perceptual similarity

- Find local minimum
  - The actual weight space looks like this:



$$w_r, w_g, w_b \geq 0$$
$$w_r + w_g + w_b = 1$$

# Outline

## Part 1: Scale Invariant Feature Detection

- Implement Difference of Gaussian (DoG)

## Part 2: Image Filtering

- Advanced color-to-gray conversion
- Implement bilateral filter

## Rules

- Environment
- Report
- Submission
- Grading
- Late Policy

# Bilateral Filter

- Given input image  $I$  and guidance  $T$ , the bilateral filter is written as:

$$I'_p = \frac{\sum_{q \in \Omega_p} G_s(p, q) \cdot G_r(T_p, T_q) \cdot I_q}{\sum_{q \in \Omega_p} G_s(p, q) \cdot G_r(T_p, T_q)}$$

- $I_p$ : Intensity of pixel  $p$  of original image  $I$
- $I'_p$ : Intensity of pixel  $p$  of filtered image  $I'$
- $T_p$ : Intensity of pixel  $p$  of guidance image  $T$
- $\Omega_p$ : Window centered in pixel  $p$
- $G_s$ : Spatial kernel
- $G_r$ : Range kernel

# Bilateral Filter

- For the spatial kernel  $G_s$ :

$$G_s(p, q) = e^{-\frac{(x_p - x_q)^2 + (y_p - y_q)^2}{2\sigma_s^2}}$$

- For the range kernel  $G_r$ :
  - If  $T$  is a single-channel image:

$$G_r(T_p, T_q) = e^{-\frac{(T_p - T_q)^2}{2\sigma_r^2}}$$

- If  $T$  is a color image:

$$G_r(T_p, T_q) = e^{-\frac{(T_p^r - T_q^r)^2 + (T_p^g - T_q^g)^2 + (T_p^b - T_q^b)^2}{2\sigma_r^2}}$$

- Pixel values should be **normalized to [0, 1]** (divided by 255) to construct range kernel.

# Assignment Description

- part2/main.py
  - Read image, execute joint bilateral filter, read setting file, select the best grayscale conversion... etc.
- part2/JBF.py
  - Implement joint bilateral filter

```
class Joint_bilateral_filter(object):  
    def __init__(self, sigma_s, sigma_r):  
        self.sigma_r = sigma_r  
        self.sigma_s = sigma_s  
        self.wndw_size = 6*sigma_s+1  
        self.pad_w = 3*sigma_s  
  
    def joint_bilateral_filter(self, img, guidance):  
        BORDER_TYPE = cv2.BORDER_REFLECT  
        padded_img = cv2.copyMakeBorder(img, self.pad_w, self.pad_w, self.pad_w, self.pad_w, BORDER_TYPE)  
        padded_guidance = cv2.copyMakeBorder(guidance, self.pad_w, self.pad_w, self.pad_w, self.pad_w, BORDER_TYPE)  
  
        ### TODO ###  
  
        return np.clip(output, 0, 255).astype(np.uint8)
```

Define window size

Pad the input and guidance image

Output image should be in format of uint8

# Assignment Description

- part2/eval.py (**DO NOT Modify!**)
  - Evaluate the correctness of the output of joint bilateral filter

```
def main():
    parser = argparse.ArgumentParser(description='evaluation function of joint bilateral filter')
    parser.add_argument('--sigma_s', default=3, type=int, help='sigma of spatial kernel')
    parser.add_argument('--sigma_r', default=0.1, type=float, help='sigma of range kernel')
    parser.add_argument('--image_path', default='./testdata/ex.png', help='path to input image')
    parser.add_argument('--gt_bf_path', default='./testdata/ex_gt_bf.png', help='path to ground truth bf image')
    parser.add_argument('--gt_jbf_path', default='./testdata/ex_gt_jbf.png', help='path to ground truth jbf image')
    args = parser.parse_args()

    img = cv2.imread(args.image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # create JBF class
    JBF = Joint_bilateral_filter(args.sigma_s, args.sigma_r)

    bf_out = JBF.joint_bilateral_filter(img_rgb, img_rgb).astype(np.uint8)
    t0 = time.time()
    jbf_out = JBF.joint_bilateral_filter(img_rgb, img_gray).astype(np.uint8)
    print('[Time] %.4f sec'%(time.time()-t0))
```

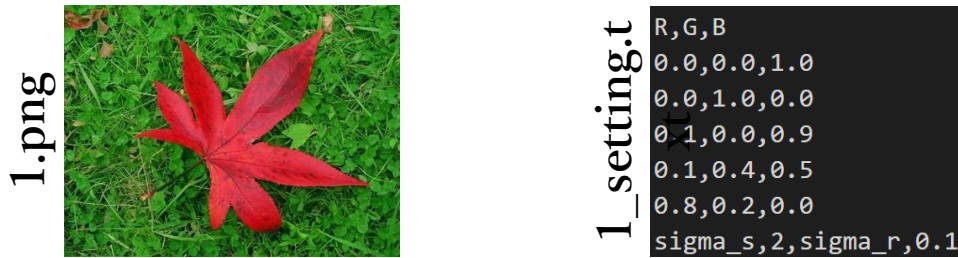
We will test your inference duration of joint bilateral filter.

- TAs will run this file to score your code.
- When testing your code, different arguments, e.g.  $\sigma_s$  and  $\sigma_r$ , and corresponding ground truth file will be applied.



# Assignment Description

- part2/testdata/
  - One example image with bf and jbf ground truth
  - Two images with respective setting files



- There are  $\sigma_s$ ,  $\sigma_r$  and 5 groups of gray-conversion parameters in the setting files. Generate **6 gray-scale images** by those parameters and `cv2.cvtColor()`.
- Use those gray-scale images as guidance to run joint bilateral filter and compute the perceptual similarity.
  - Refer p.21 and p.22 for details (use L1-norm as our cost function).
  - Note: casting the image into `np.int32` is needed to avoid overflow when subtraction.

# Assignment Description

## Recommended steps:

1. Implement joint bilateral filter in JBF.py
2. Use eval.py to evaluate your JBF.py

```
$ python3 eval.py --image_path ./testdata/ex.png --gt_bf_path ./testdata/ex_gt_bf.png  
--gt_jbf_path ./testdata/ex_gt_jbf.png
```

- The error of bilateral and joint bilateral filter should be **both 0**

```
[Error] Bilateral: 0  
[Error] Joint bilateral: 0
```

3. Finish remaining code in main.py if needed
4. Improve the inference speed of joint bilateral filter

# Assignment Description

- About the speed test of JBF...
  - Some useful tips
    - Build look-up-table for both spatial and range gaussian kernels
    - Reduce the usage of for-loop to enhance parallel processing
  - Reference time of TA code on ex.png
    - MacBook Pro 2019 (Intel Core i5 CPU) + 8 GB RAM  $\Rightarrow$   $\sim$ 1.09 sec
    - Intel Core i7-9700K CPU + 8 GB RAM  $\Rightarrow$   $\sim$ 0.93 sec
    - Intel Core i9-9900K CPU + 128 GB RAM  $\Rightarrow$   $\sim$ 0.57 sec
    - You can run TA's code on your own Linux platform. (TA/JBF.so)
  - For fair comparison, you can only use basic functions (e.g. **cannot** use cv2.filter2D, cv2.GaussianBlur) in JBF.py
  - Cython, multi-thread and GPU acceleration is **forbidden**.

# Outline

## Part 1: Scale Invariant Feature Detection

- Implement Difference of Gaussian (DoG)

## Part 2: Image Filtering

- Advanced color-to-gray conversion
- Implement bilateral filter

## Rules

- Environment
- Report
- Submission
- Grading
- Late Policy

# Environment

- Package Limitation
  - Python  $\geq 3.6$
  - Numpy  $\geq 1.21.1$
  - Opencv-python  $== 4.5$
  - Python standard library
    - <https://docs.python.org/3.7/library/>
- The following APIs are **forbidden** in part2/JBF.py
  - OpenCV: cv2.filter2D, cv2.GaussianBlur
  - Cython, multi-thread and GPU acceleration
- You will **NOT** get corresponding points if you violate rules above.

# Report (30%)

- Follow the template we provide and submit pdf file
- Part1: Difference of Gaussian (9%)
  - Plot 8 DoG images described in page.7 with threshold 3.0 on 1.png (4%)
  - Use **three thresholds (1.0, 2.0, 3.0)** on 2.png, plot DoG feature, and describe the difference. (5%)

# Report (30%)

- Part2: Joint bilateral filter (21%)
  - For 1.png and 2.png:
    - Report the cost for each filtered image (by using 6 grayscale images as guidance) (1%+1%)
    - Show **original RGB image, two filtered RGB images and two grayscale images** with highest and lowest cost (five images in total for each input image) (2%+2%)
    - Describe the difference between those two grayscale images. (5%+5%)
  - Describe how you speed up the implementation of bilateral filter. (5%)

# Submission

- Directory architecture:
  - + R11234567/
    - DoG.py
    - JBF.py
- Put above files in a directory (named **StudentID**) and compress the directory into zip file (named **StudentID.zip**)
  - e.g. After TAs run “unzip R111234567.zip”, **it should create one directory named “R11234567”**
- Do NOT copy homework (code and report) from others
- Do NOT show or write images in files you hand in.



# Submission

- Please submit those two files, i.e. your **StudentID.zip** and **report.pdf**, separately to **NTU COOL**
- Deadline: **111/03/30 (Thu.) 23:59**

文件上傳

上傳檔案，或者選擇已上傳的檔案。

⬆ 上傳文件

 使用網路攝影機

+ 新增另一個檔案

按一下此處，以找到已上傳的檔案

# Grading (Total 15%)

- Part 1 Code: 30%
  - 30%, runs within 5 mins and no error (TA will check your answer on 2.png)
  - 0%, others
- Part 2 Code: 30%
  - 30%, runs within 5 mins and no error (both bf and jbf error == 0)
  - 0%, others
- Report : 30%
- Part 2 Inference time: 10%
  - 10%, Top ~ 20%
  - 6%, 20% ~ 50%
  - 3%, 50% ~ 80%
  - 0%, 80% ~

# Late Policy

- Each student has in total 3 days delay quota.
  - Specifically, we take **12 hr. (half day)** as one unit.
  - Each student has **6 units**.
- All four homework assignments share the same 3-day delay.
  - For example, if you delay 1 day in HW1, then you will have 2 day delay quota left for HW2, HW3 and HW4.
- If you have spent all 6 delay units, your late submission will not be accepted.
  - That is, you are required to submit your homework in time to get points.
- If you have questions about the policy, you can contact TAs.

# TA information

- Hsin-Yu Chen (陳欣妤)

E-mail: [hychen@media.ee.ntu.edu.tw](mailto:hychen@media.ee.ntu.edu.tw)

TA hour: Mon. 14:00 - 16:00

Location: 博理 421

- Yu-Kai Chen(陳昱愷)

E-mail: [chenyukai@media.ee.ntu.edu.tw](mailto:chenyukai@media.ee.ntu.edu.tw)

Location: 博理 421