# COMP2211 Exploring Artificial Intelligence

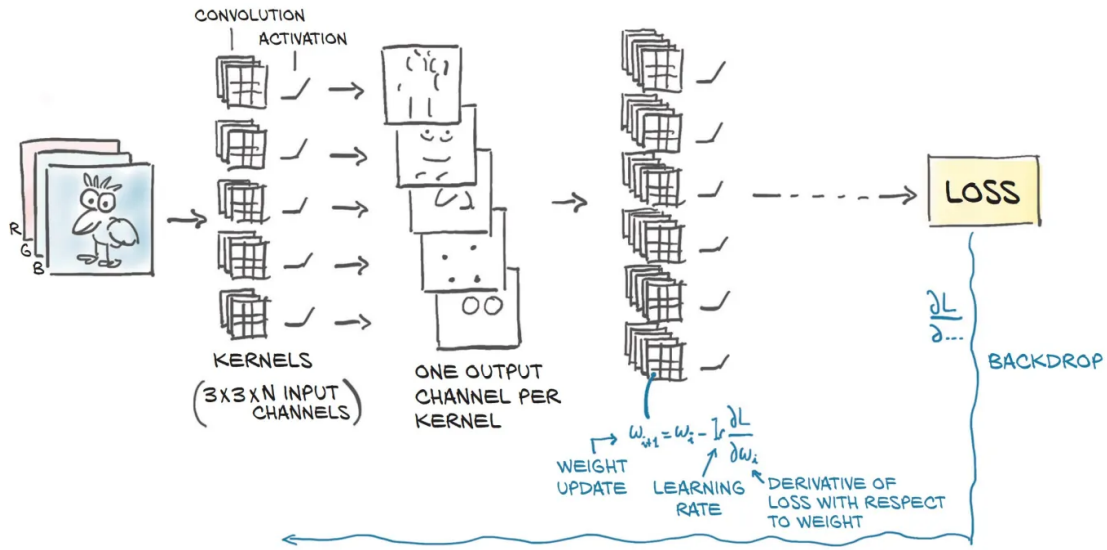
## Digital Image Processing Fundamentals

Huiru Xiao

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

# Convolutional Neural Network

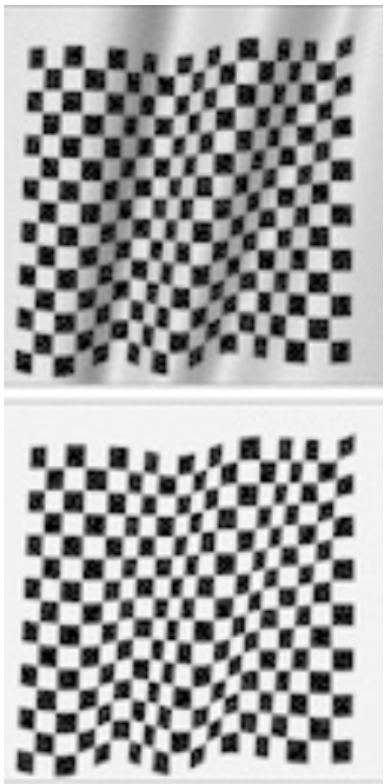
- Convolutional Neural Network (CNN or ConvNet) is a type of Artificial Neural Networks applied to analyze visual imagery.
- Research has demonstrated that preprocessing images before feeding them to CNN would significantly improve the classification/recognition accuracy.
- To learn how to preprocess images and use this powerful tool to tackle Computer Vision tasks, you will first be introduced to digital image processing fundamentals.



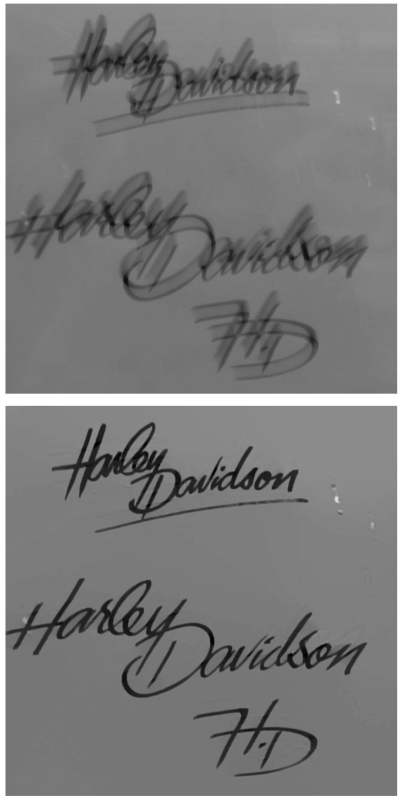
# Image Preprocessing

- Image preprocessing refers to processing an image so that the resulting image is more suitable than the original for a specific application.
- A preprocessing method that works well for one application may not be the best method for another application.
- Some preprocessing tasks:
  - Shading correction: to adjust image inconsistency, generally seen on the edges, due to illumination and background inhomogeneity.
  - De-blurring: to remove the blurring artifacts from images to restore the original, sharp content.
  - De-noising: to remove or reduce noise from an image and to recover the original image, which is considered to be noise-free, from a noisy observation.
  - Contrast enhancement: to touch up the contrast of the image, or the difference in color and light between parts of it, to improve its perception by human eye.

# Image Preprocessing

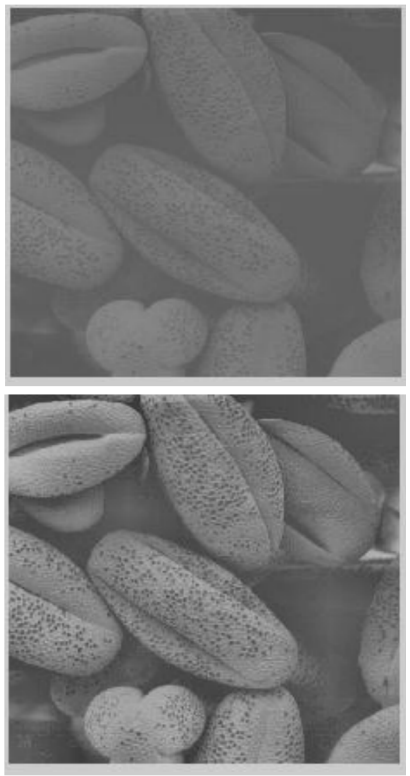


Shading Correction



De-blurring

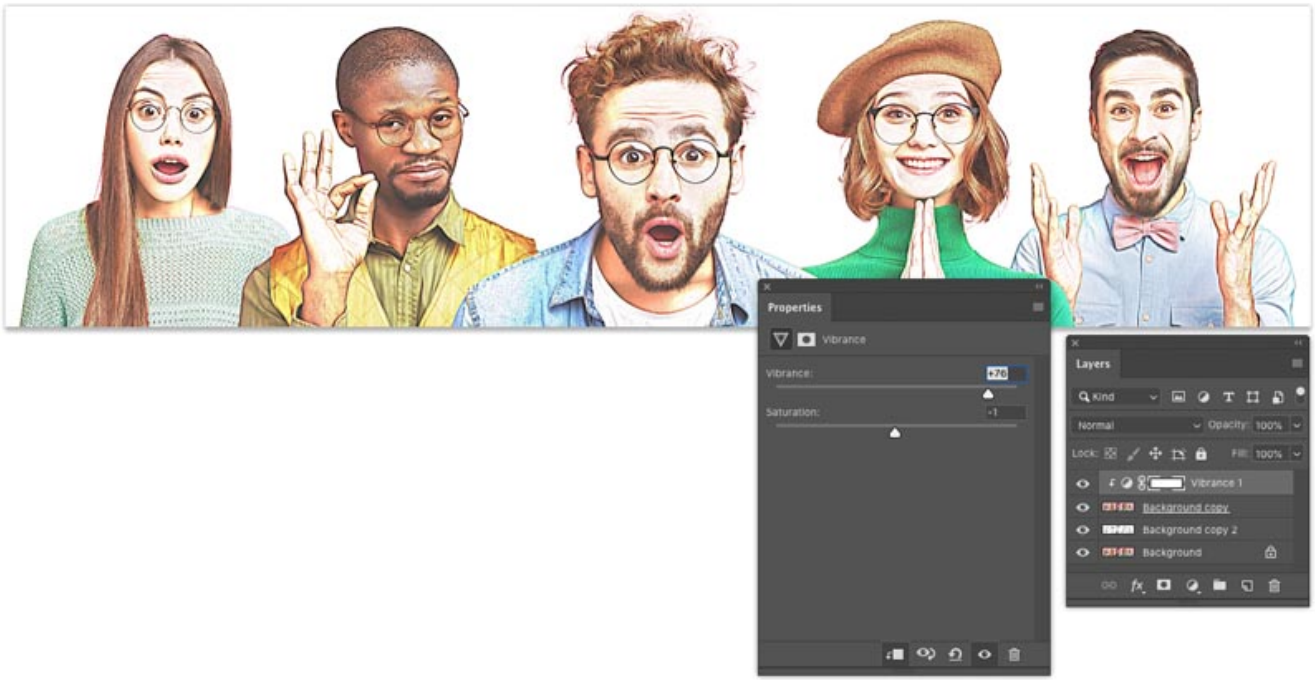
De-noising



Contrast Enhancement

# Digital Image Processing

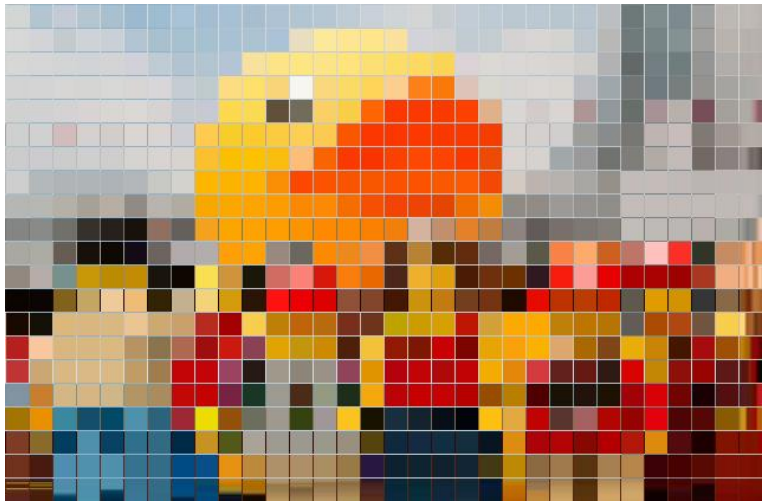
- Digital image processing (DIP) is the method to manipulate a digital image to either enhance the quality or extract relevant information.
- These methods provide the foundation to preprocess images that are more suitable for specific applications.



# Digital Image Fundamentals

# Digital Image

- A digital image is a two-dimensional grid of intensity values, represented by I(x,y), where x and y are coordinates, and the value of I at coordinates (x,y) is called intensity.
- Pixels: Short for Picture Element. A pixel is a single point (dot) in an image.
- Dimensions: Specified by the width and height of the image.
  - Image width is the number of columns in the image.
  - Image height is the number of rows in the image.



An image of dimensions $32 \times 21$ (i.e., image width $= 32$ pixels, image height $= 21$ pixels)

# Image Coordinate System

- A specific pixel is specified by its coordinates (x,y) where x is increasing from left to right, and y is increasing from top to bottom.
- The origin (0,0) is in the top-left corner.
- The following shows the coordinate system of digital images:

| | | | | |
|---|---|---|---|---|
| (0,0) | (1,0) | (2,0) | $\cdots$ | (width-1,0) |
| (0,1) | (1,1) | (2,1) | $\cdots$ | (width-1,1) |
| $\vdots$ | | | | |
| (0, height-1) | (1, height-1) | (2, height-1) | ... | (width-1, height-1) |

where width and height are the image width and image height, respectively.

- The legal range of x-coordinate is between 0 to width-1
- The legal range of y-coordinate is between 0 to height-1

# Grayscale Images

- Grayscale is a range of gray shades from black to white.
- Grayscale images are most commonly used in image processing because the data are smaller and allow us to process the images in a short time.
- A grayscale digital image is an image in which the value of each pixel carries only intensity information.
- A grayscale image contains 8-bit/pixel data, which has $2^8 = 256$ different gray levels (0 for black, 127 for gray, and 255 for white).

| | | |
|---|---|---|
| Black | Gray-level $= 0$ | |
| Dark gray | Gray-level $= 64$ | |
| Medium gray | Gray-level $= 127$ | |
| Light gray | Gray-level $= 190$ | |
| White | Gray-level $= 255$ | |

# Color Images

- Color images have intensity from the darkest and lightest of 3 different colors, Red, Green, and Blue (RGB).
- The mixtures of these color intensities produce a color image.
- Since RGB images contain 3×8-bit intensities, they are also referred to as 24-bit color images.
- An 8-bit intensity range has 256 possible values, 0 to 255.
- Common colors represented in RGB:

| | | |
|---|---|---|
| Black | RGB = (0, 0, 0) | |
| White | RGB = (255, 255, 255) | |
| Red | RGB = (255, 0, 0) | |
| Green | RGB = (0, 255, 0) | |
| Blue | RGB = (0, 0, 255) | |
| Yellow | RGB = (255, 255, 0) | |
| Gray | RGB = (128, 128, 128) | |
| Dark Gray | RGB = (50, 50, 50) | |

24-bit grayscale images are subset of RGB images where the RGB intensity are all equal (e.g., Gray and Dark Gray shown above).

# Access Images in Google Drive

- To access files/images in Google drive, you need to mount your Google Drive to Colab using the following code:

```
# Import drive from google.colab package
from google.colab import drive
# Import os and sys modules
import os, sys

# Mount Google Drive
drive.mount('/content/drive')
# Assume a folder "images" has been created, go to the folder "images"
os.chdir('/content/drive/My Drive/images')
# Add the path for interpreter to search
sys.path.append('/content/drive/My Drive/images')
```

# Read Images in Colab

- To read an image, you need to first import matplotlib.image:

```
import matplotlib.image as mpimg
```

- Then use imread() method of the matplotlib.image module.

## Syntax

```
mpimg.imread(fname, format=none)
```

Parameters:

- fname: The image file to read: a filename or a file-like object opened in read-binary mode.
- format: The image file format assumed for reading the data. If format is not given, the format is deduced from the filename. If nothing can be deduced, PNG is tried.
- Return value: numpy.array.
  - (M,N) for grayscale images
  - (M,N,3) for RGB images
  - (M,N,4) for RGBA images: red, green, blue, alpha. A three-channel RGB color model supplemented with a fourth alpha channel. Alpha indicates how opaque each pixel is, i.e., transparency.

PNG images are returned as float arrays (0-1). All other formats are returned as int arrays, with a bit depth determined by the file's contents.

URL: `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imread.html`

# Show Images in Colab

- To show an image, you need to first import matplotlib.pyplot:

```
import matplotlib.pyplot as plt
```

- Then use imshow() method of the matplotlib.pyplot module.

## Syntax

```
plt.imshow(X, cmap, vmin, vmax)
```

Parameters:

- X: The image data. Supported array shapes are
  - (M,N): an image with scalar data. The values are mapped to colors using normalization and a colormap.
  - (M,N,3): an image with RGB values (0-1 float or 0-255 int)
  - (M,N,4): an image with RGBA values (0-1 float or 0-255 int), i.e., including transparency.

  The first two dimensions (M,N) define the rows and columns of the image.

## Syntax

Parameters:

- cmap: str (e.g., 'gray') or Colormap. The Colormap instance or registered colormap name used to map scalar data to colors. This parameter is ignored for RGB(A) data.
- vmin, vmax:
  - By default, imshow scales elements of the numpy array so that the smallest element becomes 0, the largest becomes 1, and intermediate values are mapped to the interval [0,1] by a linear function.
  - Optionally, imshow can be called with arguments, vmin and vmax. In such case all elements of the array smaller or equal to vmin are mapped to 0, all elements greater or equal to vmax are sent to 1, and the elements between vmin and vmax are mapped in a linear fashion into the interval [0,1].
- Returns AxesImage
  AxesImage is an image attached to an Axes.

URL: `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html`

# Save Images in Colab

- To save an image, you need to first import matplotlib.pyplot:

```
import matplotlib.pyplot as plt
```

- Then use imsave() method of the matplotlib.pyplot module.

**Syntax**

```
plt.imsave(fname, arr)
```

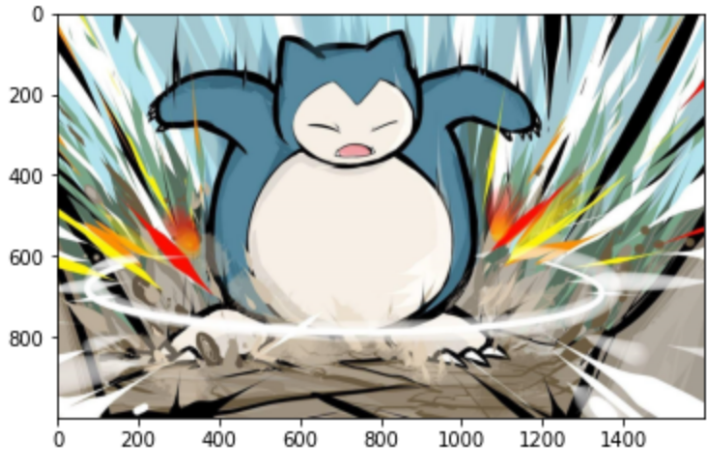
Parameters:

- fname: a path or a file-like object to store the image in.
- arr: The image data. The shape can be one of M×N (luminance), $M \times N \times 3$ (RGB) or $M \times N \times 4$ (RGBA).
  The first two dimensions (M,N) define the rows and columns of the image.

URL: `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imsave.html`

```
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np

# Read and show the image
img = mpimg.imread('snorlax.png')
plt.imshow(img)

# Find the shape of input image
[height, width, layers] = np.shape(img)

# Print all the required information
print('Dimensions: {}x{}x{}'.format(height, width, layers))
print('Total number of pixels:', width*height)

plt.imsave('snorlax2.png', img) # Save the image
```

```
Dimensions: 1000x1600x4
Total number of pixels: 1600000
```



# Image Processing

# Image Processing using OpenCV

- OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.
- OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.
- The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.
- OpenCV supports a wide variety of programming languages such as Python, C++, Java, etc.

# Convert Color Images to Grayscale

- In certain problem, you will find it useful to lose unnecessary information from your images to reduce space and computational complexity.
- Converting colored images to grayscale images is an example. This is done, as color is not necessary to recognize and interpret an image.
- Grayscale can be good enough for recognizing certain objects, because color images contain more information than black and white images, they can add unnecessary complexity and take up more space in memory.

# Convert Color Images to Grayscale

- One way to convert color images to grayscale is to apply the following formula:

$$V = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

- To perform the above using OpenCV, you need to first import cv2

```
import cv2
```

- Then use cvtColor() method of the cv2 module.

**Syntax**

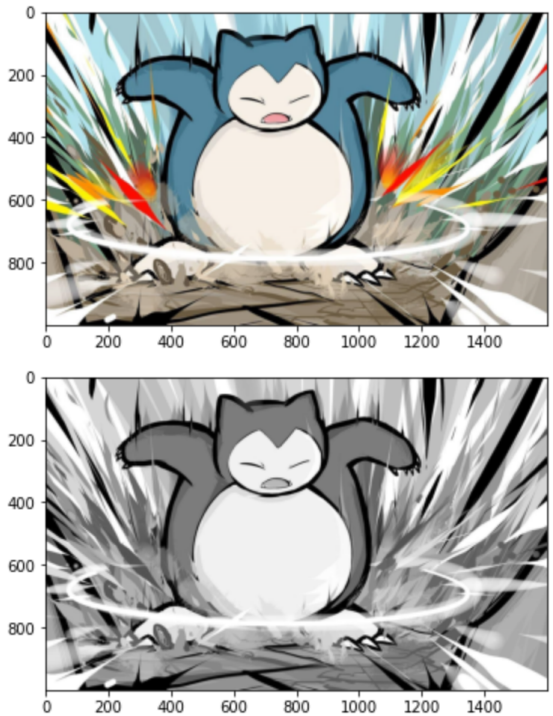
```
cv2.cvtColor(image, code)
```

Parameters:

- image: Image to be processed in n-dimensional array
- code: Conversion code for colorspace. For converting RGB to grayscale, we use cv2.COLOR_RGB2GRAY
- Return value: Converted image.

URL: `https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab`

```
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Read and show the image
img = mpimg.imread('snorlax.png')
plt.figure()
plt.imshow(img)

# Convert color image to gray
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

# Show the image
plt.figure()
plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Save the image
plt.imsave('snorlax-gray.png', grayImg)
```



# Image Affine Transformation

- An affine transformation is any transformation that preserves collinearity, parallelism as well as the ratio of distances between the points (e.g., midpoint of a line remains the midpoint after transformation).
- It does not necessarily preserve distances and angles.
- Geometric transformations, such as translation, rotation, scaling, shearing, etc., are all affine transformations.
- In general, the affine transformation can be expressed in the form of a linear transformation followed by a vector addition as follows:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_{00} \\ b_{10} \end{bmatrix} = \begin{bmatrix} a_{00}x + a_{01}y + b_{00} \\ a_{10}x + a_{11}y + b_{10} \end{bmatrix}$$

$$= \begin{bmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{10} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- $M = \begin{bmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{10} \end{bmatrix}$ is a transformation matrix. To define what transformation you want to do, you need to define M.

# Image Translation

- To perform the image affine transformation using OpenCV, you need to first import cv2

```
import cv2
```

- Then use warpAffine() method of the cv2 module.

**Syntax**

```
cv2.warpAffine(src, M, dsize, flags, borderMode, borderValue)
```

Parameters:

- src: input image
- M: $2 \times 3$ transformation matrix
- dsize: size of the output image
- flags: combination of interpolation methods
- borderMode: pixel extrapolation method
- borderValue: value used in case of a constant border; by default, it is 0
- Return value: output image that has the size dsize and the same type as src

URL: `https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#ga0203d9ee5fcd28d40dbc4a1ea4451983`

# Image Translation

- Translation is simply the shifting of object location.
- Suppose we have a point $P(x, y)$ which is translated by $(t_x, t_y)$, then the coordinates after translation denoted by $P'(x', y')$ are given by

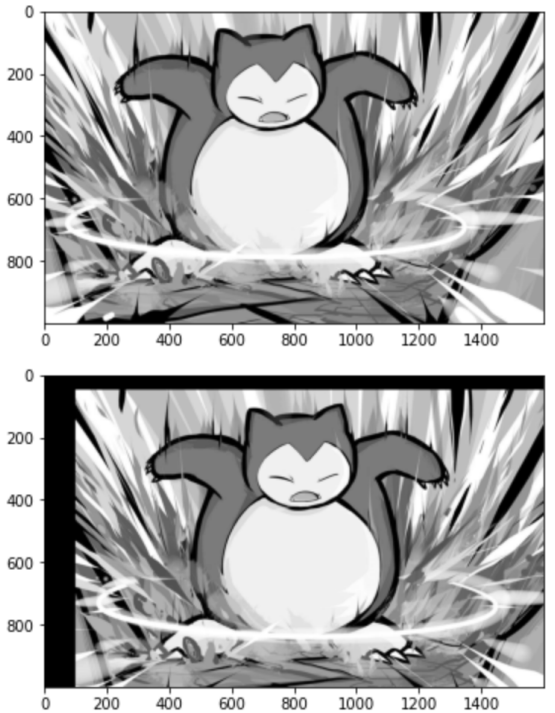
$$x' = x + t_x$$
$$y' = y + t_y$$

- In matrix form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

```python
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the requird libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np

# Read the image
img = mpimg.imread('snorlax.png')

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure();
plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Find the shape of the gray image
rows, cols = grayImg.shape

# Form the transformation matrix of translation
M = np.float32([[1, 0, 100],[0, 1, 50]])
# Perform the transformation
translatedImg = cv2.warpAffine(grayImg, M, (cols,rows))

# Show the image
plt.figure(); plt.imshow(translatedImg, cmap='gray', vmin=0, vmax=1)
```

```
<matplotlib.image.AxesImage at 0x7fc1460d1850>
```



# Image Reflection

- Reflection refers to mirroring (or flipping) images along x-axis or y-axis.
- To make sure the resulting image fits the image coordinate, after flipping image along x-axis, we need to translate it by the amount of number of rows in y-axis. Similarly, after flipping image along y-axis, we need to translate it by the amount of columns in x-axis.
- Suppose we have a point $P(x, y)$ which is reflected along the x-axis, then the coordinates after reflection denoted by $P'(x', y')$ are given by

$$x' = x$$
$$y' = -y + (rows - 1)$$

In matrix form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & (rows - 1) \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Similarly, a point $P(x, y)$ which is reflected along the y-axis, the coordinates after reflection denoted by $P'(x', y')$ are given by

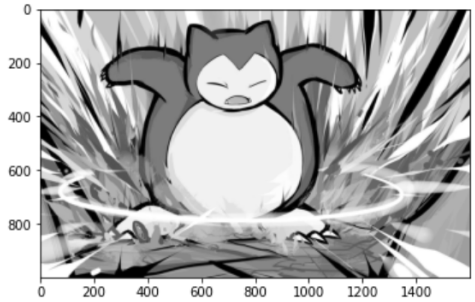
$$x' = -x + (cols - 1)$$
$$y' = y$$

In matrix form

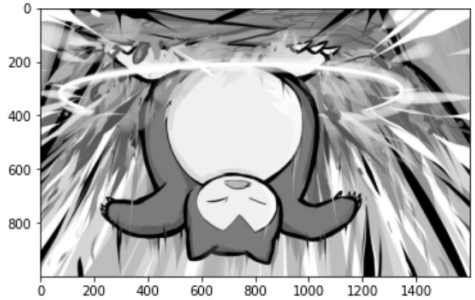
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 & (cols - 1) \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

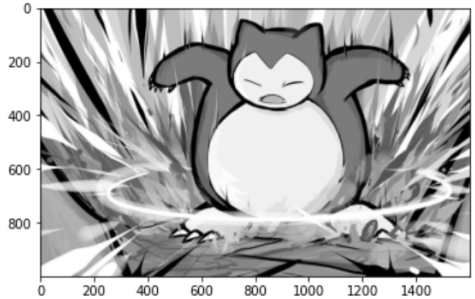
```
# Assume Google Drive has been mounted, the path has been added for inte
```

```
<matplotlib.image.AxesImage at 0x7fc1460a5850>
```




```
# Import all the required libraries
import cv2; import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

img = mpimg.imread('snorlax.png')  # Read the image
# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure(); plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

rows, cols = grayImg.shape  # Find the shape of the gray image
# Form the transformation matrix of x-axis reflection
M = np.float32([[1, 0, 0], [0, -1, rows-1]])
# Perform the transformation
xaxisreflection = cv2.warpAffine(grayImg, M, (cols,rows))
plt.figure(); plt.imshow(xaxisreflection, cmap='gray', vmin=0, vmax=1)

# Form the transformation matrix of y-axis reflection
M = np.float32([[-1, 0, cols-1], [0, 1, 0]])
# Perform the transformation
yaxisreflection = cv2.warpAffine(grayImg, M, (cols,rows))
plt.figure(); plt.imshow(yaxisreflection, cmap='gray', vmin=0, vmax=1)
```



# Image Rotation

- Image rotation refers to rotating an image $\theta$ degree along $(x_0, y_0)$.
- Suppose we have a point $P(x, y)$ which is rotated along the center of the image, then the coordinates after rotation denoted $P'(x', y')$ are given by

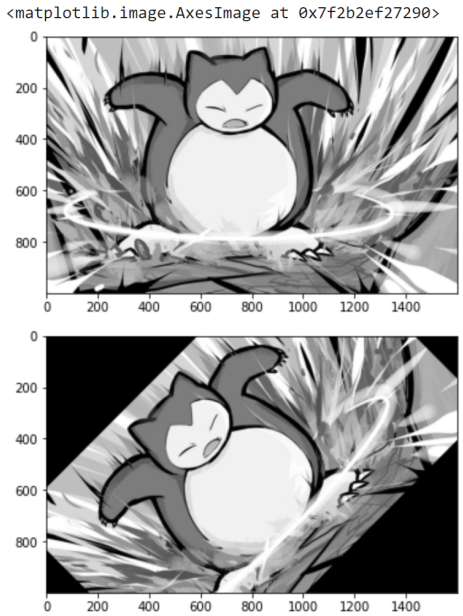
$$x' = (x - x_0)cos\theta + (y - y_0)sin\theta + x_0$$
$$y' = -(x - x_0)sin\theta + (y - y_0)cos\theta + y_0$$

In matrix form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & -x_0cos\theta - y_0sin\theta + x_0 \\ -sin\theta & cos\theta & x_0sin\theta - y_0cos\theta + y_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

URL: `https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#gafbbc470ce83812914a70abfb604f4326`

```
# Assume Google Drive has been mounted, the path has been added for interpreter to search

import cv2, math; import numpy as np  # Import all the required libraries
import matplotlib.image as mpimg; import matplotlib.pyplot as plt

img = mpimg.imread('snorlax.png')  # Read the image
# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure(); plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

rows, cols = grayImg.shape  # Find the shape of image

# Form the transformation matrix of rotation
# The angle for math.sin and math.cos should be in radian.
# 45 degree = pi/4 radian
angle = math.pi/4
M = np.float32([[math.cos(angle), math.sin(angle),
  -(cols//2)*math.cos(angle)-(rows//2)*math.sin(angle) + (cols//2)],
                [-math.sin(angle), math.cos(angle),
  (cols//2)*math.sin(angle)-(rows//2)*math.cos(angle) + (rows//2)]])

# Another way to generate the required transformation matrix
# The angle for getRotationMatrix2D should be in degree
# M = cv2.getRotationMatrix2D((cols//2,rows//2), 45, 1.0)

# Perform the transformation
rotate45 = cv2.warpAffine(grayImg, M, (cols,rows))
plt.figure(); plt.imshow(rotate45, cmap='gray', vmin=0, vmax=1)
```



# Image Resizing/Scaling

- Resizing/Scaling images is a critical preprocessing step in computer vision.
- Machine learning models train faster on smaller images. Moreover, many deep learning model architectures require that our images are the same size, and our raw collected images may vary in size.
- Resizing is a common approach to make the input images the same size, and it works well unless you have a very different aspect ratio from the expected input shape.

# Image Resizing/Scaling

- To perform the image resizing/scaling using OpenCV, you need to first import cv2

```
import cv2
```

- Then use resize() method of the cv2 module.

**Syntax**

```
cv2.resize(src, dsize, dst, fx = 0, fy = 0, interpolation = INTER_LINEAR)
```

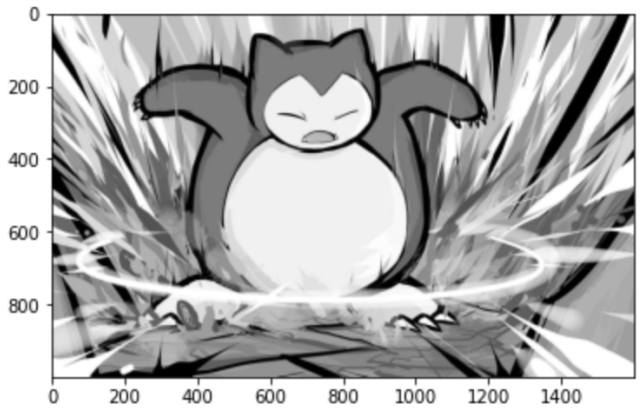
Parameters:

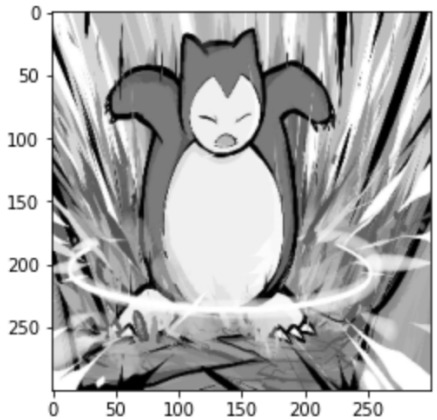
- src: Input image
- dsize: The size for the output image
- dst (optional): The output image with size dsize
- fx (optional): The scale factor along the horizontal axis
- fy (optional): The scale factor along the vertical axis
- interpolation: The algorithm used to reconstruct the new pixels
  - cv2.INTER_NEARNEST (nearest neighbor interpolation)
  - cv2.INTER_LINEAR (bilinear interpolation)
  - cv2.INTER_CUBIC (bicubic interpolation)
- Returns AxesImage

URL: `https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#ga47a974309e9102f5f08231edc7e7529d`

```python
# Assume Google Drive has been mounted, the path
# has been added for interpreter to search

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

img = mpimg.imread('snorlax.png')  # Read the image

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure()
plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Perform the transformation
resizedImg = cv2.resize(grayImg, (300, 300),
                        interpolation=cv2.INTER_LINEAR)

# Show the image
plt.figure()
plt.imshow(resizedImg, cmap='gray', vmin=0, vmax=1)
```

```
<matplotlib.image.AxesImage at 0x7f149d2057d0>
```

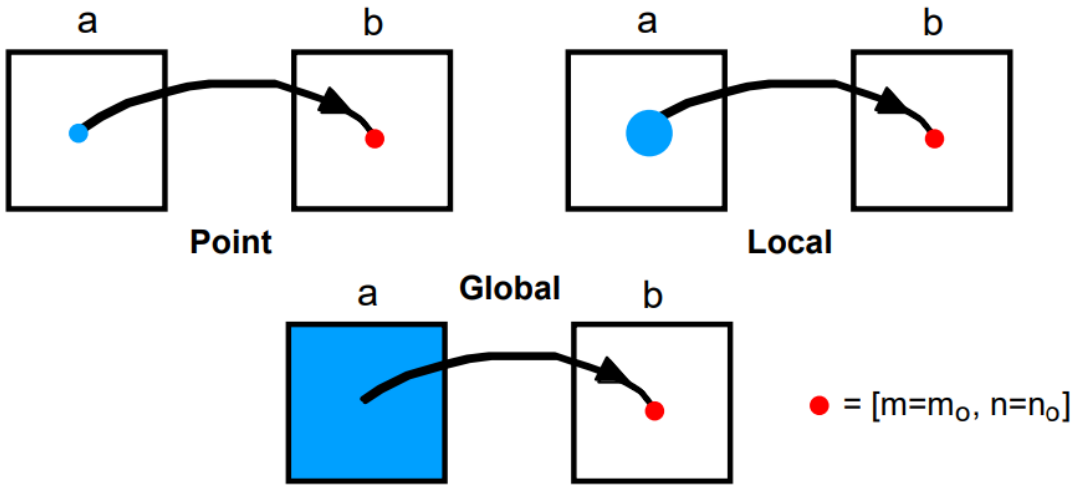


# Characteristics of Image Operations

# Characteristics of Image Operations

- There are many ways to classify image operations.
- One way for doing so is to be based on the "region" used to process the pixels.
  1. Point: The output value at a specific coordinate is dependent only on the input value at the same coordinate.
  2. Local: The output value at a specific coordinate is dependent on the input values in the neighborhood of that same coordinate.
  3. Global: The output value at a specific coordinate is dependent on all the values in the input image



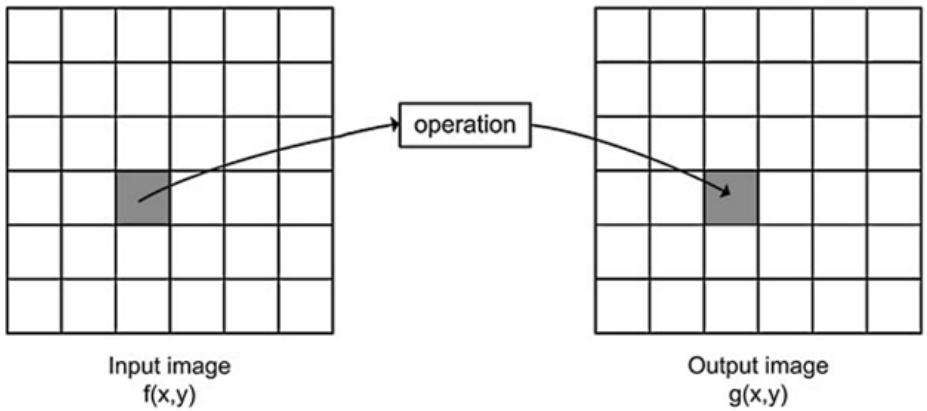
# Question

What type of image operation is color to grayscale conversion? :D

Answer:
Point operation! Since the output value at a specific coordinate of the grayscale image is dependent only on the input value at the same coordinate of the color image.

# Point-based Operations (Examples)

- Brightness adjustment: Make images brighter or dimmer (Optional)
- Contrast stretching: Adjust the contrast of images
- Gamma correction: Grayscale non-linear transformation (Optional)
- Grayscale threshold: Convert a grayscale image into a black and white binary image
- Histogram equalization: Transformation where an output image has approximately the same number of pixels at each gray level (Optional)



# Contrast Stretching

- Contrast stretching is an image enhancement method which attempts to improve an image by stretching the range of intensity values.
- One way to perform contrast stretching is to stretch minimum and maximum intensity values present to the possible minimum and maximum intensity values.
- Example:
  - Assume 0-255 taken as standard minimum and maximum intensity for 8-bit images.
  - If the minimum intensity value ($I_{min}$) present in the image is 100, then it is stretched to the possible minimum value 0.
  - Likewise, if the maximum intensity value ($I_{max}$) is less than the possible maximum intensity value 255, then it is stretched out to 255.
  - General formula for contrast stretching:

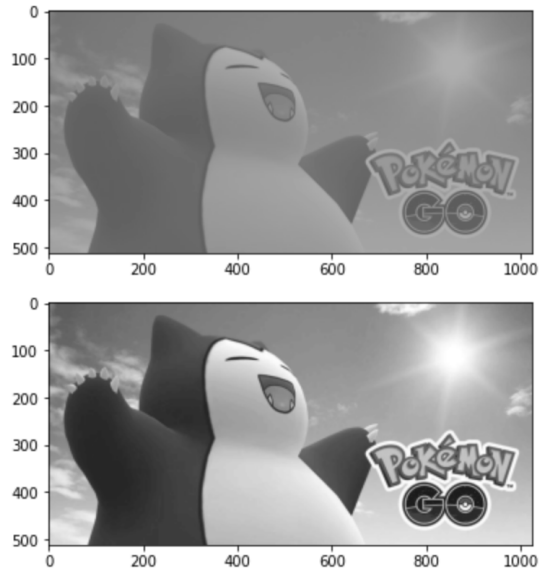
$$I_{new} = \frac{I - I_{min}}{I_{max} - I_{min}} \times 255$$

where

- $I$ is the current pixel intensity value
- $I_{min}$ is the minimum intensity value present in the whole image
- $I_{max}$ is the maximum intensity value present in the whole image
- $I_{new}$ is the output intensity value rounded up to the nearest integer value

```
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2; import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Read the image
img = mpimg.imread('snorlax-low-contrast.png')

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure(); plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Convert pixel values from [0,1] to [0,255]
grayImgUint = grayImg*255
grayImgUint = grayImgUint.astype(np.uint8)

# Find min and max pixel values and perform normalization
min = np.min(grayImgUint)
max = np.max(grayImgUint)
imageContrastEnhance = ((grayImgUint-min)/(max-min))*255

# Show the image
plt.figure(); plt.imshow(imageContrastEnhance.astype(np.uint8), cmap='gray', vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7f1351cc8850>
```



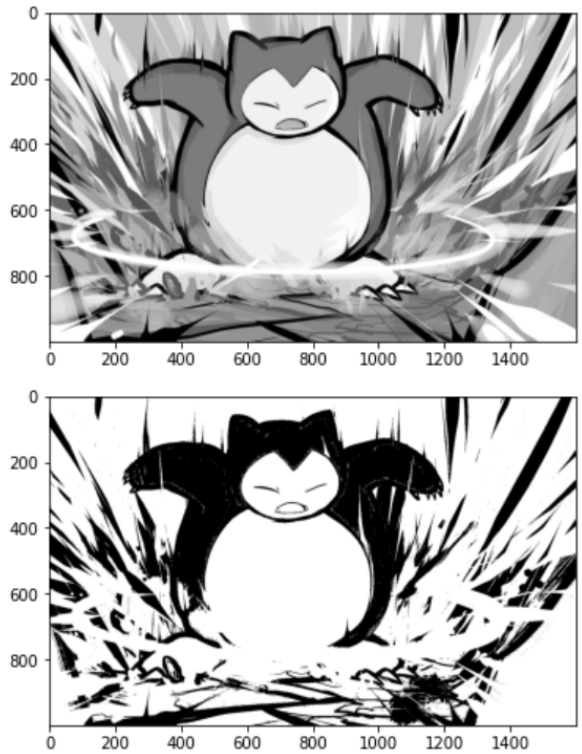
# Grayscale Thresholding

- Grayscale thresholding is a simple form of image segmentation.
- It is a way to create a binary image from a grayscale image or full-color image.
- This is typically done in order to separate "object" or foreground pixels from background pixels to aid in image processing.
- The formula of grayscale thresholding with threshold T is defined as

$$I_{new} = \begin{cases} 0 & I < T \\ 255 & otherwise \end{cases}$$

```python
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Read the image
img = mpimg.imread('snorlax.png')

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure(); plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Convert pixel values from [0,1] to [0,255]
grayImgUint = grayImg*255
grayImgUint = grayImgUint.astype(np.uint8)

# Perform thresholding
processedImg = grayImgUint > 128

# Show the image
plt.figure(); plt.imshow(processedImg, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7fc145092a50>
```



# Grayscale Thresholding with Otsu's Method

- We need a way to automatically determine the threshold value T so that the result of thresholding is reproductible.
- A well-known approach is Otsu's method
  1. Select an initial estimate of the threshold T. A good initial value is the average intensity of the image.
  2. Partition the image into two groups, $R_1$, $R_2$, using the threshold T.
  3. Calculate the mean gray values $\mu_1$ and $\mu_2$ of the partitions, $R_1$, $R_2$.
  4. Compute a new threshold
  $$T = \frac{1}{2}(\mu_1 + \mu_2)$$
  5. Repeat steps 2-4 until the mean values $\mu_1$ and $\mu_2$ in successive iterations do not change.

# Grayscale Thresholding with Otsu's Method

- To perform Otsu's thresholding, you need to first import cv2

```
import cv2
```

- Then use threshold() method of the cv2 module.

## Syntax

```
cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)
```

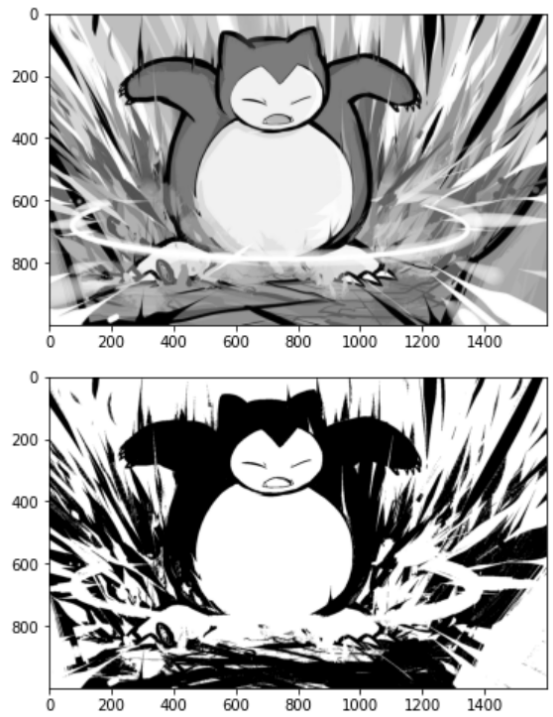
Parameters:

- source: input image array (must be grayscale)
- thresholdValue: value of threshold below and above which pixel values will change accordingly
- maxVal: Maximum value that can be assigned to a pixel
- thresholdingTechnique: The type of thresholding to be applied (For Otsu's, we put cv2.THRESH_BINARY + cv2.THRESH_OTSU)
- Return values:
  - The first is the threshold that was used.
  - The second is the thresholded image (i.e., the binary image).

URL: `https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html`

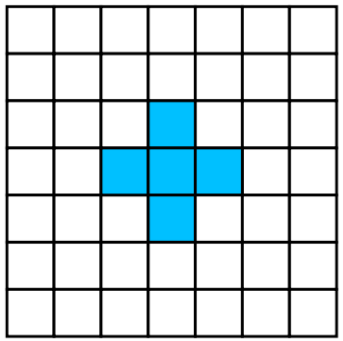
```python
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Read the image
img = mpimg.imread('snorlax.png')

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure(); plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Convert pixel values from [0,1] to [0,255]
grayImgUint = grayImg*255
grayImgUint = grayImgUint.astype(np.uint8)

# Perform thresholding using Otsu's method
thresh, processedImg = cv2.threshold(grayImgUint, 120, 255,
                                     cv2.THRESH_BINARY + cv2.THRESH_OTSU)
print('Optimal threshold:', thresh)

# Show the image
plt.figure(); plt.imshow(processedImg, cmap='gray')
```

```
Optimal threshold: 153.0
<matplotlib.image.AxesImage at 0x7fc14500ef10>
```

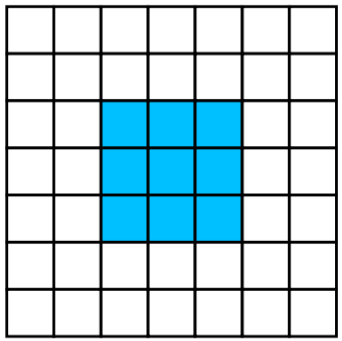


# Local Operations

- Recall, local operations refer to those the output value at a specific coordinate is dependent on the input values in the neighborhood of that same coordinate.
- Some of the most common neighborhoods are 4-connected neighborhood and the 8-connected neighborhood.



Rectangular sampling
4-connected



Rectangular sampling
8-connected

# Examples

- Image smoothing: It removes noise and softens edges and corners of the image. It is also called blurring.
- Image edge detection: It detects the boundaries (edges) of objects, or regions within an image.
- Image sharpening: It removes blur, enhances details, and dehazes.

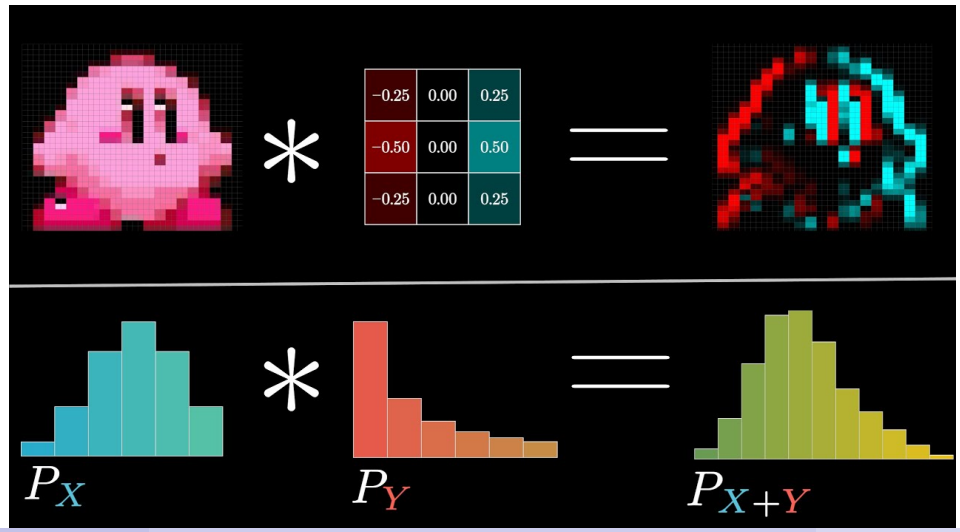
# Image Convolution

# Convolution

## Convolution

A convolution is an integral that expresses the amount of overlap of one function $g$ as it is shifted over another function $f$. It therefore "blends" one function with another.

- In mathematics, convolution is an operation performed on two functions ($f$ and $g$) to produce a third function.
- Convolution is one of the most important operations in signal and image processing. It could operate in 1D (e.g. speech processing), 2D (e.g. image processing) or 3D (video processing).



# Image Convolution

Image convolution is defined as

$$O(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} K(m,n) I(x-m, y-n)$$

where I is the input image, K is the image kernel.

- Assume the origin (i.e., (0,0)) of I is top-left corner, while
- the origin (i.e., (0,0)) of K is the center of the kernel.

If the image kernel is 3×3, then

$$O(x,y) = \sum_{m=-1}^{1} \sum_{n=-1}^{1} K(m,n) I(x-m, y-n)$$

Image kernel is also called image filter or image mask.

# Example

Input image I

| | | | | |
|---|---|---|---|---|
| 10 | 1 | 3 | 2 | 6 |
| 4 | 3 | 5 | 8 | 0 |
| 8 | 7 | 9 | 6 | 5 |

Image kernel K

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |

$$O(x,y) = \sum_{m=-1}^{1} \sum_{n=-1}^{1} K(m,n) I(x-m, y-n)$$

$$O(1,1) = \sum_{m=-1}^{1} (K(m,-1)I(1-m,1-(-1)) + K(m,0)I(1-m,1-0) + K(m,1)I(1-m,1-1))$$

$$\begin{aligned}
&= K(-1,-1)I(2,2) + K(-1,0)I(2,1) + K(-1,1)I(2,0) + \\
&\quad K(0,-1)I(1,2) + K(0,0)I(1,1) + K(0,1)I(1,0) + \\
&\quad K(1,-1)I(0,2) + K(1,0)I(0,1) + K(1,1)I(0,0) \\
&= (-1)(9) + (-1)(5) + (-1)(3) + (0)(7) + (0)(3) + (0)(1) + (1)(8) + (1)(4) + (1)(10) \\
&= -9 - 5 - 3 + 8 + 4 + 10 = 5
\end{aligned}$$

Can you calculate all the remaining output image values?

Answer:

| | | |
|---|---|---|
| 5 | -5 | 6 |

# Very Tedious :( Any Intuitive Way? YES!!!

- Steps
  1. Inverse the kernel, i.e., flipping the kernel in both horizontal and vertical directions about the center of kernel.

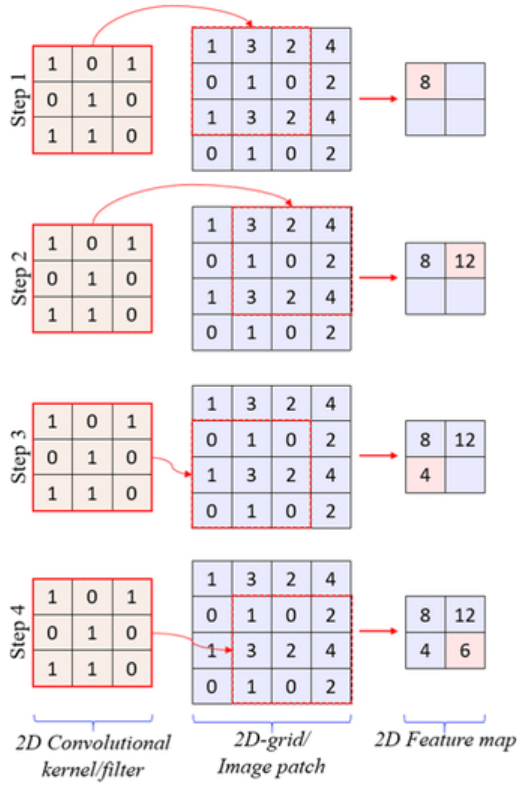
| | | |
|---|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

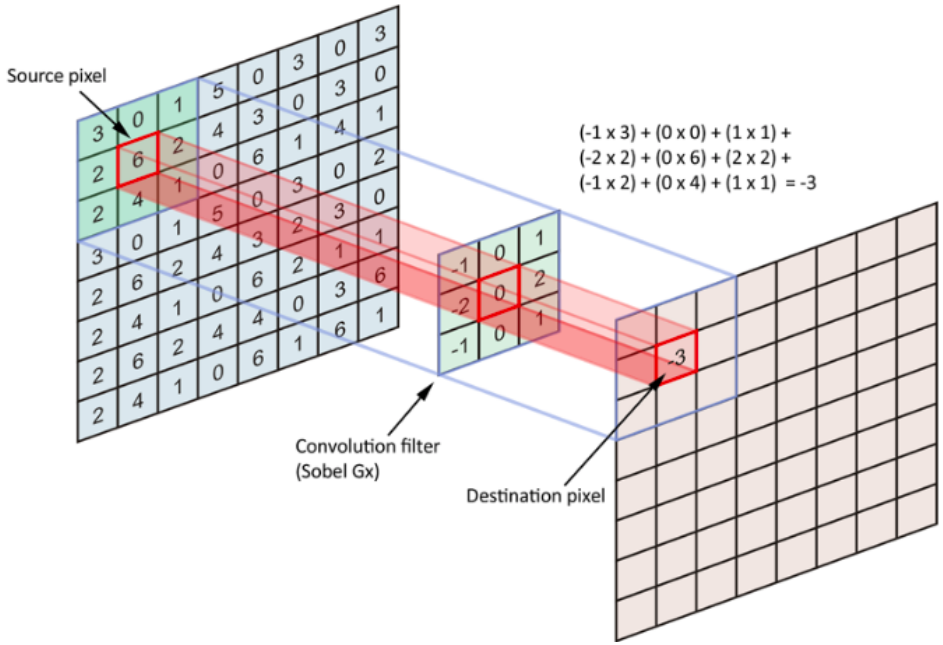
(Left) Original kernel, (Middle) Flipped horizontally, (Right) Flipped vertically

  2. Slide over the inversed kernel centered at interested point.
  3. Multiply inversed kernel data with the overlapped area.
  4. Sum and accumulate the output.




Step 2 to Step 4

# Image Convolution



# Problem

- When computing an output pixel at the boundary of an image, a portion of the convolution is usually off the edge of the image. How to deal with this?
- Solutions:
  1. Just ignore those boundary pixels. :P
  2. Do zero padding (i.e., add a border of pixels all with value zero around the edges of the input images.)

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 10 | 1 | 3 | 2 | 6 | 0 |
| 0 | 4 | 3 | 5 | 8 | 0 | 0 |
| 0 | 8 | 7 | 9 | 6 | 5 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

  3. Replicating boundary pixels

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 10 | 1 | 3 | 2 | 6 | 6 |
| 10 | 10 | 1 | 3 | 2 | 6 | 6 |
| 4 | 4 | 3 | 5 | 8 | 0 | 0 |
| 8 | 8 | 7 | 9 | 6 | 5 | 5 |
| 8 | 8 | 7 | 9 | 6 | 5 | 5 |

- Solutions:
  4. Reflecting boundary pixels

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 10 | 1 | 3 | 2 | 6 | 6 |
| 10 | 10 | 1 | 3 | 2 | 6 | 6 |
| 4 | 4 | 3 | 5 | 8 | 0 | 0 |
| 8 | 8 | 7 | 9 | 6 | 5 | 5 |
| 8 | 8 | 7 | 9 | 6 | 5 | 5 |

  5. Mirroring boundary pixels

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 4 | 3 | 5 | 8 | 0 | 8 |
| 1 | 10 | 1 | 3 | 2 | 6 | 2 |
| 3 | 4 | 3 | 5 | 8 | 0 | 8 |
| 7 | 8 | 7 | 9 | 6 | 5 | 6 |
| 3 | 4 | 3 | 5 | 8 | 0 | 8 |

# Image Convolution

- To do image convolution, you need to first import cv2
  `import cv2`
- Then use filter2D() method of the cv2 module.

## Syntax

`cv2.filter2D(src, ddepth, kernel, dst, anchor, delta, borderType=cv2.BORDER_DEFAULT)`

Parameters:

- src: input image that you want to convolve
- ddepth: desired depth of the destination image. If ddepth=-1, the output image will have the same depth as the src
- kernel: convolution kernel, a single-channel floating point matrix; if you want to apply different kernels to different channels, split the image into separate color planes using split and process them individually.
- dst: output image of the same size and the same number of channels as src.
- anchor: anchor of the kernel that indicates the relative position of a filtered point within the kernel; the anchor should lie within the kernel; default value (-1,-1) means that the anchor is at the kernel center.
- delta: optional value added to the filtered pixels before storing them in dst.
- borderType: pixel extrapolation method
  - cv2.BORDER_CONSTANT: iiiiii|abcdefgh|iiiiii with some specified i
  - cv2.BORDER_REPLICATE: aaaaaa|abcdefgh|hhhhhh
  - cv2.BORDER_REFLECT: fedcba|abcdefgh|hgfedc
  - cv2.BORDER_REFLECT_101: gfedcb|abcdefgh|gfedcba
  - cv2.BORDER_DEFAULT: Same as BORDER_REFLECT_101
- Return value: filtered image.

URL: `https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga27c049795ce870216ddfb366086b5a04`

# Image Convolution

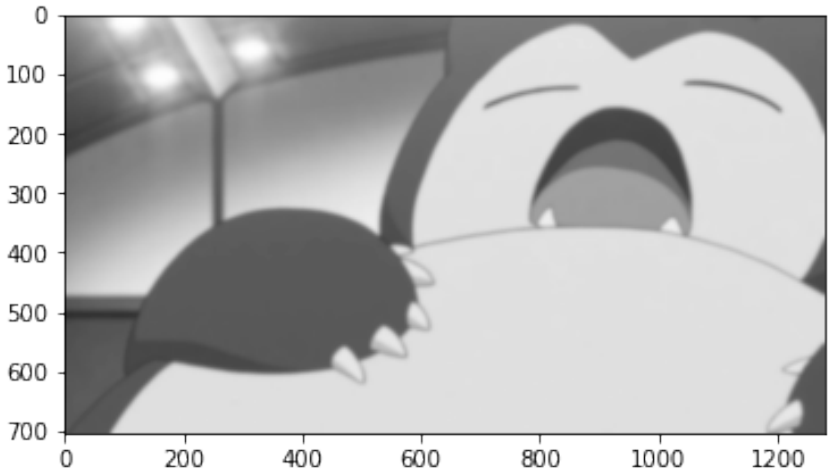
**Note**

filter2D does not mirror the kernel for you. You will need to flip the kernel before applying cv2.filter2D.

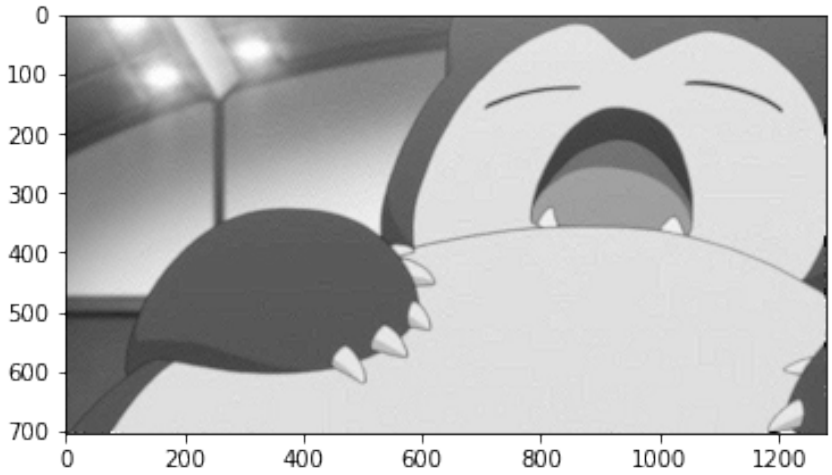
```
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2; import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Read the image
img = mpimg.imread('snorlax-sleep.png')

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure(); plt.imshow(grayImg, cmap='gray',
                         vmin=0, vmax=1)
```

```
# Prepare a kernel (a sharpening kernel here)
kernel_3x3 = np.array([ [0,-1,0],
                        [-1,5,-1],
                        [0,-1,0] ])

for i in range(5): # Perform filtering 5 times
    grayImg = cv2.filter2D(grayImg, -1,
                           kernel_3x3)

# Show the resulting image
plt.figure(); plt.imshow(grayImg, cmap="gray",
                         vmin=0, vmax=1)
```

# Before and After




Input image




Output image

# Smoothing (Averaging/Blurring) Kernel

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |


Original image

Effects of kernel in different size. (What do you observe?)


3×3 mask


5×5 mask


15×15 mask


25×25 mask

- Blurring an image can be done by averaging pixels
- Analogous to integration, related to sum of pixel intensity values

# Sharpening Kernel

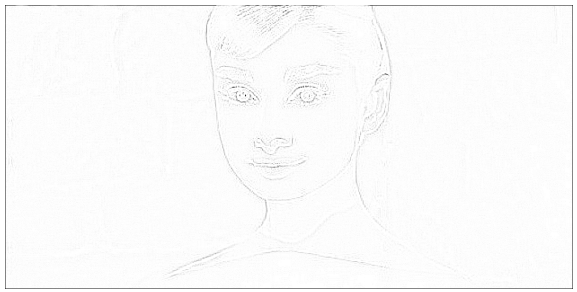
| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

Original image

+



Detail (Edge)
[Color flipped for clarity]

=

Sharpened image

- Sharpening has the opposite effect of blurring
- Analogous to differentiation, related to the difference of pixel intensity values

# Edge Kernel - Prewitt

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |

Kernel for detecting vertical edges

| | | |
|---|---|---|
| -1 | -1 | -1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Kernel for detecting horizontal edges


Edge image (vertical edges)
$|G_x|$

Edge image (horizontal edges)
$|G_y|$

Edge image (magnitude)
$\sqrt{G_x^2 + G_y^2}$

Pixels of the processed images are inverted (i.e. black to white, white to black) for making them more visible.

# Edge Kernel - Sobel

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

Kernel for detecting vertical edges

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

Kernel for detecting horizontal edges


Edge image (vertical edges)
$|G_x|$

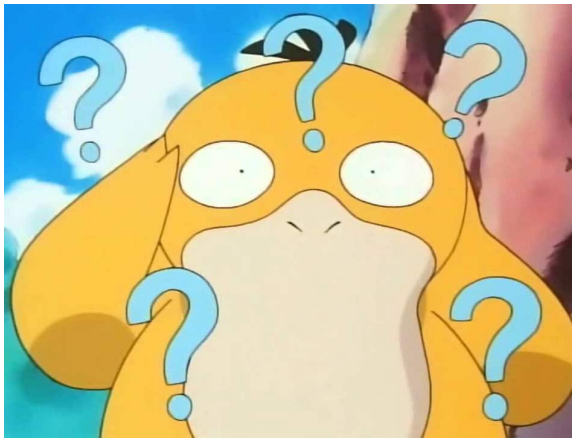
Edge image (horizontal edges)
$|G_y|$

Edge image (magnitude)
$\sqrt{G_x^2 + G_y^2}$

Pixels of the processed images are inverted (i.e. black to white, white to black) for making them more visible.

# Image Convolutions

- Clearly, image convolution is powerful in finding the features of an image if we already know the right kernel to use.
- Kernel design is an art and has been refined over the last few decades to do some pretty amazing things with images. But the important question is, what if we don't know the features we are looking for? Or what if we do know, but we don't know what kernel should look like?



# Practice Problem

A 3-bit/pixel (i.e. pixel value is in range 0 and $2^3$=8) of size $3 \times 3$ is given below.

| | | |
|---|---|---|
| 3 | 7 | 6 |
| 2 | 4 | 6 |
| 4 | 7 | 2 |

1. Find the output of a $3 \times 3$ averaging kernel at (1,1).
2. Find the edge magnitude at (1,1) using the Sobel masks shown below.

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

Sobel kernel for extracting horizontal edges

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

Sobel kernel for extracting vertical edges

Note: Edge magnitude $= \sqrt{horizontal\_edge\_value^2 + vertical\_edge\_value^2}$.

# Practice Problem

1. The output value of a $3 \times 3$ averaging kernel at (1,1) is

$$\frac{3+7+6+2+4+6+4+7+2}{9} = \frac{41}{9}$$

2. Horizontal edge value

$$=1 \times 3 + 2 \times 7 + 1 \times 6 + 0 \times 2 + 0 \times 4 + 0 \times 6 + (-1) \times 4 + (-2) \times 7 + (-1) \times 2$$
$$=3 + 14 + 6 - 4 - 14 - 2 = 3$$

Vertical edge value

$$=1 \times 3 + 0 \times 7 + (-1) \times 6 + 2 \times 2 + 0 \times 4 + (-2) \times 6 + 1 \times 4 + 0 \times 7 + (-1) \times 2)$$
$$=3 - 6 + 4 - 12 + 4 - 2 = -9$$

Edge magnitude $= \sqrt{(3)^2 + (-9)^2} = \sqrt{90} = 9.49$

That's all!

Any question?

**Welcome Back!**

# Acknowledgments


- The lecture notes are developed based on Dr. Desmond Tsoi's lecture slides.