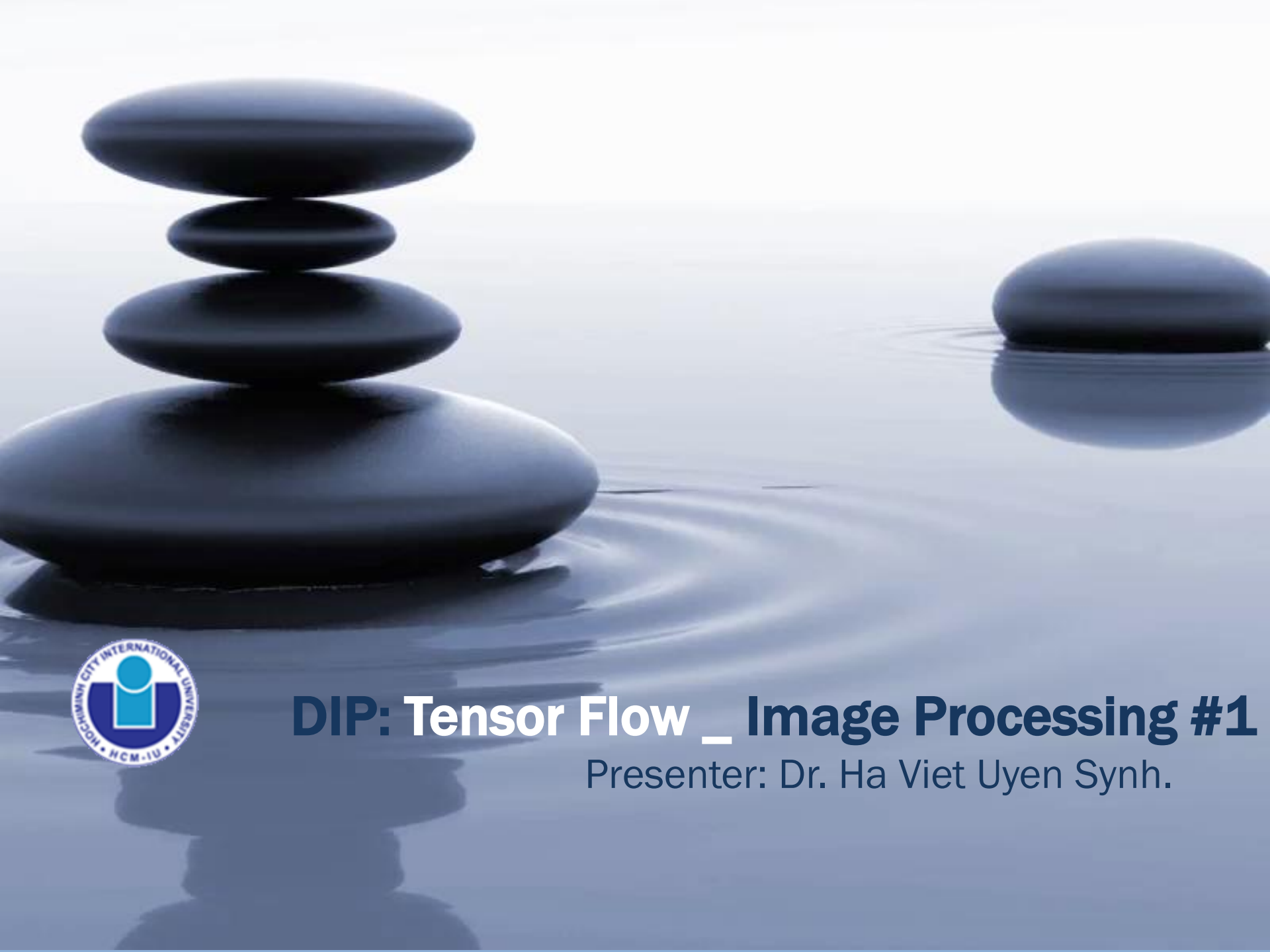


DIP: Tensor Flow _ Image Processing #1

Presenter: Dr. Ha Viet Uyen Synh.

Overview

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

import tensorflow as tf
from PIL import Image
```

NUMPY LIBRARY

Numpy Library

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

```
pip install numpy
```

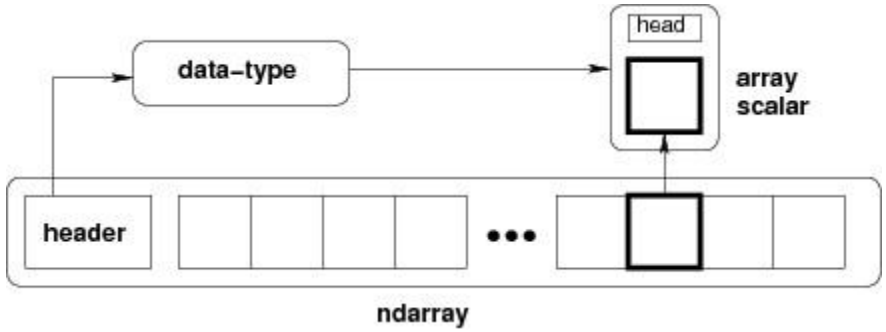
```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"
b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Ndarray Object

The most important object defined in NumPy is an N-dimensional array type called ndarray. ndarray describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.

Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called dtype).

Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types.



Ndarray Object _ Syntax

numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array, or any (nested) sequence.
2	**dtype** Desired data type of array, optional
3	**copy** Optional. By default (true), the object is copied
4	**order** C (row major) or F (column major) or A (any) (default)
5	**subok** By default, returned array forced to be a base class array. If true, sub-classes passed through
6	**ndmin** Specifies minimum dimensions of resultant array

Example

```
import numpy as np

a = np.array([1,2,3])
print (a) #[1, 2, 3]

a = np.array([[1, 2], [3, 4]])
print (a) #[[1, 2] ,  [3, 4]]

a = np.array([1, 2, 3,4,5], ndmin = 2)
print (a) #[[1, 2, 3, 4, 5]]

a = np.array([1, 2, 3], dtype = complex)
print (a) #[ 1.+0.j,  2.+0.j,  3.+0.j]
```

Data Types

```
import numpy as np

# using array-scalar type
dt = np.dtype(np.int32)
print (dt)  #int32

#int8, int16, int32, int64 can be replaced by equivalent string 'i1', 'i2','i4', etc.
dt = np.dtype('i4')
print (dt) # int32

# first create structured data type
dt = np.dtype([('age',np.int8)])
print (dt) #[('age', 'i1')]
a = np.array([(10,),(20,),(30,)], dtype = dt)
print (a) #[(10,) (20,) (30,)]

student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype = student)
print (a) #[('abc', 21, 50.0), ('xyz', 18, 75.0)]
```

Array Attributes

- ndarray.shape

```
a = np.array([[1,2,3],[4,5,6]])
print (a.shape) # (2,3)
a.shape = (3,2) # a.reshape(3,2)
print (a) #[[1, 2], [3, 4], [5, 6]]
```

- ndarray.ndim

```
a = np.arange(24)
# [0 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 17 18 19 20 21 22 23]
b = a.reshape(2,4,3)
print (b)
```

- numpy.itemsize

```
x = np.array([1,2,3,4,5], dtype = np.float32)
print x.itemsize # 4
```

```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"

d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #              [ 0.68744134  0.87236687]]"
```

Array From Existing Data

```
numpy.asarray(a, dtype = None, order = None)
x = [1,2,3]
a = np.asarray(x, dtype = float)
print (a)

numpy.frombuffer(buffer, dtype = float, count = -1, offset = 0)
s = 'Hello World'
a = np.frombuffer(s, dtype = 'S1')
print (a) #['H'  'e'  'l'  'l'  'o'  ' '  'W'  'o'  'r'  'l'  'd']
```

Array From Numerical Ranges

numpy.arange(start, stop, step, dtype)
numpy.linspace(start, stop, num, endpoint, retstep, dtype)
numpy.logspace(start, stop, num, endpoint, base, dtype)

```
x = np.arange(5, dtype = float)
print (x) #[0.  1.  2.  3.  4.]
x = np.arange(10,20,2)
print (x) #[10  12  14  16  18]

x = np.linspace(10,20, 5, endpoint = False)
print (x) #[10.   12.   14.   16.   18.]
x = np.linspace(1,2,5, retstep = True)
print (x) # (array([ 1.  ,  1.25,  1.5 ,  1.75,  2.  ]), 0.25)
# retstep here is 0.25

a = np.logspace(1,10,num = 10, base = 2)
print (a) #[ 2.    4.    8.   16.   32.   64.  128.  256.   512.  1024.]
```

```
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])

print 'Our array is:'
print a
print '\n'

# this returns array of items in the second column
print 'The items in the second column are:'
print a[...,1]
print '\n'

# Now we will slice all items from the second row
print 'The items in the second row are:'
print a[1,...]
print '\n'

# Now we will slice all items from column 1 onwards
print 'The items column 1 onwards are:'
print a[...,1:]
```

```
Our array is:
[[1 2 3]
 [3 4 5]
 [4 5 6]]

The items in the second column are:
[2 4 5]

The items in the second row are:
[3 4 5]

The items column 1 onwards are:
[[2 3]
 [4 5]
 [5 6]]
```

Slicing (start:stop:step)

```
import numpy as np

a = np.arange(10)

s = slice(2,7,2)
print (a[s]) # [2 4 6]

b = a[2:7:2]
print (b) # [2 4 6]

print (a[2:]) # [2  3  4  5  6  7  8  9]

a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print (a)
# slice items starting from index
print (a[1:])
[[3 4 5]
 [4 5 6]]
```

Integer array indexing

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]“

a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print(a)  # prints "array([[ 1,  2,  3],
          #               [ 4,  5,  6],
          #               [ 7,  8,  9],
          #               [10, 11, 12]])"
b = np.array([0, 2, 0, 1])
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"
a[np.arange(4), b] += 10
print(a)  # prints "array([[11,  2,  3],
          #               [ 4,  5, 16],
          #               [17,  8,  9],
          #               [10, 21, 12]])
```

Boolean array indexing

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)
print(bool_idx)      # Prints "[[False False]
                     #         [ True  True]
                     #         [ True  True]]"

print(a[bool_idx])  # Prints "[3 4 5 6]"
print(a[a > 2])      # Prints "[3 4 5 6]“

x = np.array([1, 2])   # Let numpy choose the datatype
print(x.dtype)         # Prints "int64"

x = np.array([1.0, 2.0])   # Let numpy choose the datatype
print(x.dtype)             # Prints "float64"

x = np.array([1, 2], dtype=np.int64)   # Force a particular datatype
print(x.dtype)                         # Prints "int64"
```

Array math

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
```

Array math

```
# Elementwise division; both produce the array
# [[ 0.2        0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))
# Elementwise square root; produces the array
# [[ 1.         1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
v = np.array([9,10])
w = np.array([11, 12])
# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))
# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

```
import numpy as np

x = np.array([[1,2],[3,4]])

print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]“

print(x)    # Prints "[[1 2]
            #         [3 4]]"
print(x.T)  # Prints "[[1 3]
            #         [2 4]]"

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print(v)    # Prints "[1 2 3]"
print(v.T)  # Prints "[1 2 3]"
```

Broadcasting

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x
# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

# Now y is the following
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
print(y)
```

Broadcasting

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))   # Stack 4 copies of v on top of each other
print(vv)                 # Prints "[[1 0 1]
                          #         [1 0 1]
                          #         [1 0 1]
                          #         [1 0 1]]"
y = x + vv  # Add x and vv elementwise
print(y)  # Prints "[[ 2  2  4
          #         [ 5  5  7]
          #         [ 8  8 10]
          #         [11 11 13]]“

y = x + v  # Add v to each row of x using broadcasting
print(y)  # Prints "[[ 2  2  4]
          #         [ 5  5  7]
          #         [ 8  8 10]
          #         [11 11 13]]"
```

Broadcasting

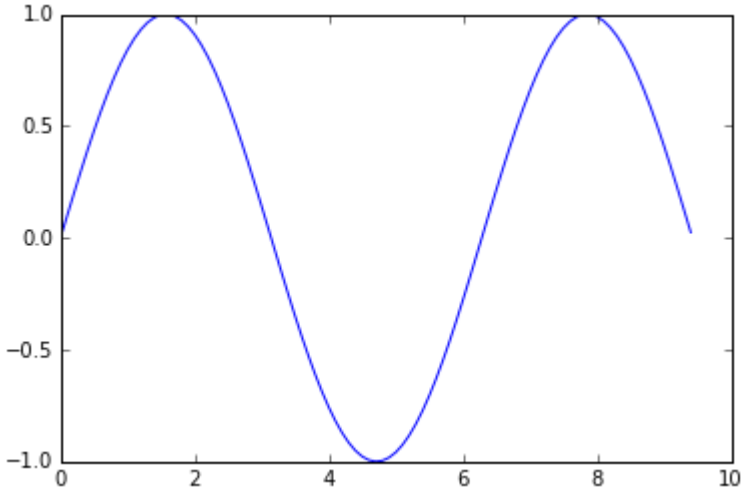
```
import numpy as np

# Compute outer product of vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)
print(np.reshape(v, (3, 1)) * w)
# [[ 4  5]
#  [ 8 10]
#  [12 15]]

# Add a vector to each row of a matrix
x = np.array([[1,2,3], [4,5,6]])
print(x + v)
# [[2 4 6]
#  [5 7 9]]
print((x.T + w).T)
# [[ 5  6  7]
#  [ 9 10 11]]
print(x + np.reshape(w, (2, 1)))
print(x * 2)
# [[ 2  4  6]
#  [ 8 10 12]]
```

MATPLOTLIB LIBRARY

matplotlib

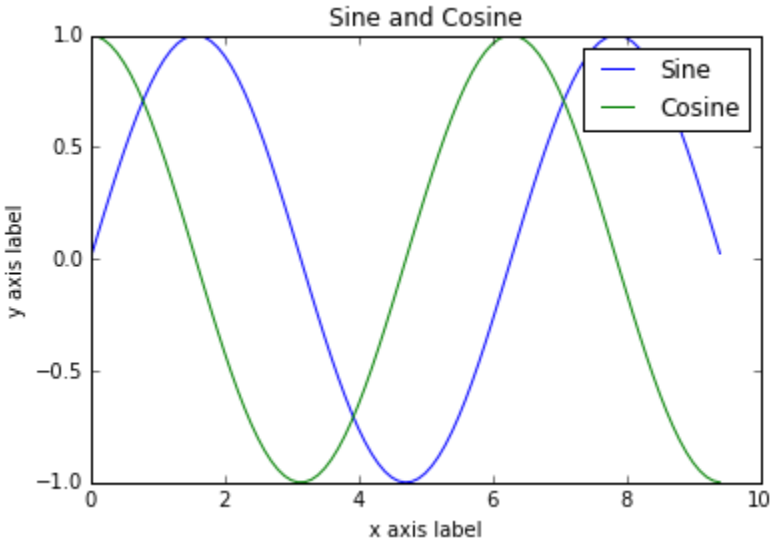
Matplotlib is a plotting library. In this section give a brief introduction to the matplotlib.pyplot module, which provides a plotting system.

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```



matplotlib

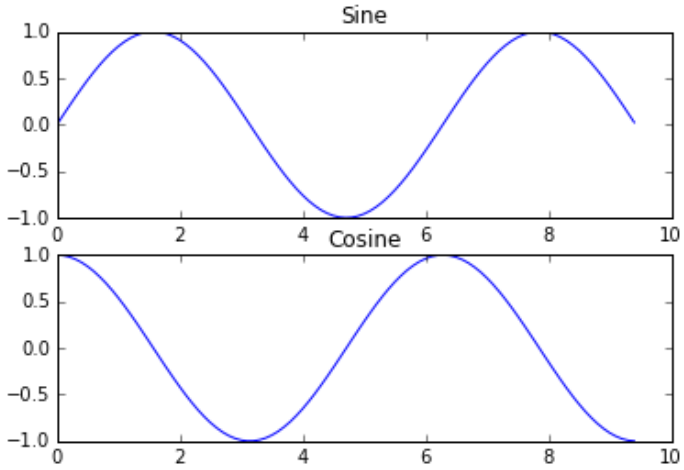
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



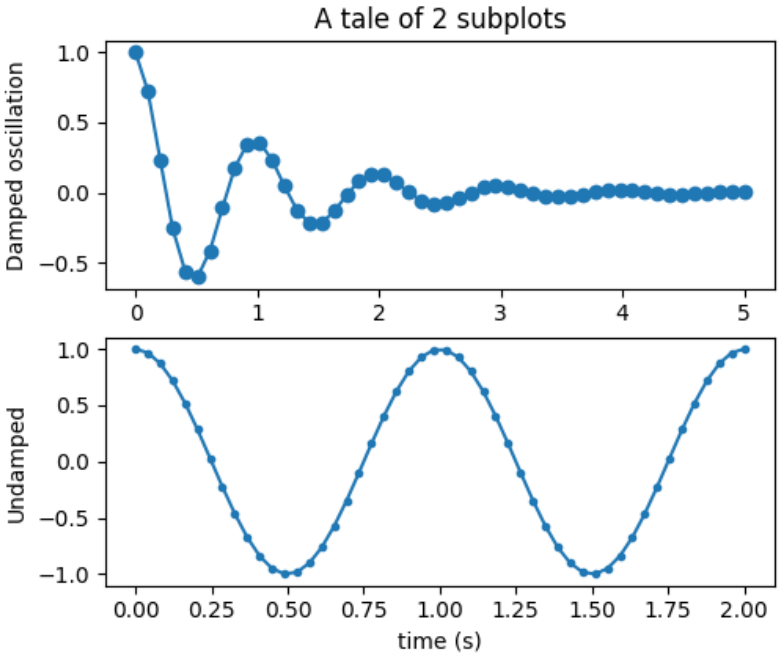
Subplots

subplot(nrows, ncols, plot_number)

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

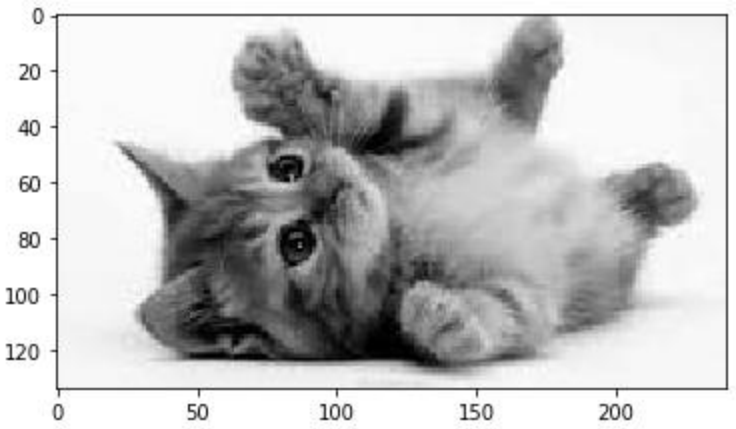


```
import numpy as np
import matplotlib.pyplot as plt


x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)

plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'o-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')

plt.subplot(2, 1, 2)
plt.plot(x2, y2, '.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')

plt.show()
```



Image

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

import tensorflow as tf
from PIL import Image

img = Image.open('gray_kitten.jpg')
plt.imshow(img)
plt.show()
```



PILLOW LIBRARY

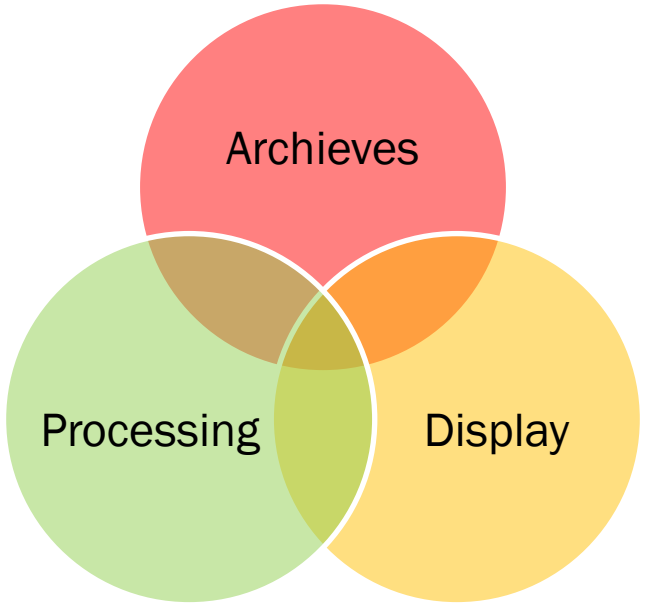
Pillow

The Python Imaging Library adds image processing capabilities to your Python interpreter.

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

python -m pip install Pillow



Using the Image class

To load an image from a file, use the open() function in the Image module:

```
from PIL import Image
im = Image.open("gray_kitten.jpg")

print(im.format, im.size, im.mode)
PPM (512, 512) RGB

print("Format: {0}\nSize: {1}\nMode: {2}".format(im.format, im.size, im.mode))
Format: JPEG
Size: (350, 232)
Mode: RGB
```

Reading and writing images

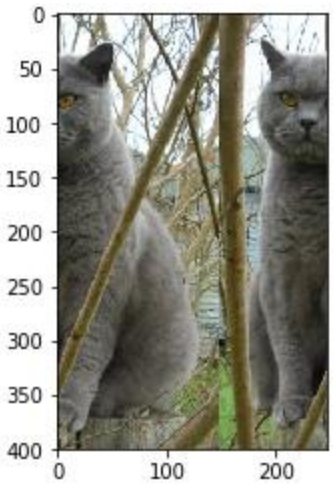
```
from PIL import Image

tatras = Image.open("tatras.jpg")

tatras.save('tatras.png', 'png')
```

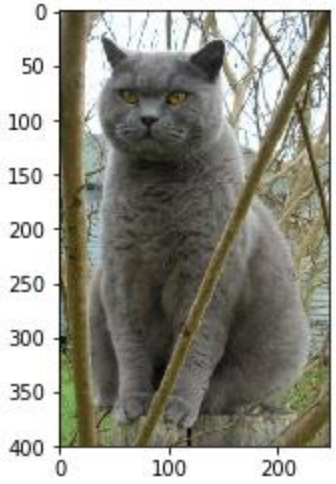
Cutting, pasting, and merging images

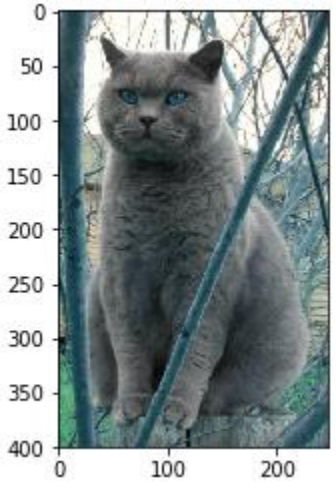
```
box = (100, 100, 400, 400)
region = im.crop(box)

region = region.transpose(Image.ROTATE_180)
im.paste(region, box)

def roll(image, delta):
    """Roll an image sideways."""
    xsize, ysize = image.size
    delta = delta % xsize
    if delta == 0:
        return image
    part1 = image.crop((0, 0, delta, ysize))
    part2 = image.crop((delta, 0, xsize, ysize))
    part1.load()
    part2.load()
    image.paste(part2, (0, 0, xsize-delta, ysize))
    image.paste(part1, (xsize-delta, 0, xsize, ysize))
    return image
```

Splitting and merging bands

```
img = Image.open('cat.jpg')
img_roll= roll(img,100)
plt.imshow(img_roll)
plt.show()
```








```
r, g, b = im.split()
im = Image.merge("RGB", (b, g, r))
```

Geometrical transforms

```
out = im.resize((128, 128))
out = im.rotate(45) # degrees counter-clockwise

out = im.transpose(Image.FLIP_LEFT_RIGHT)
out = im.transpose(Image.FLIP_TOP_BOTTOM)
out = im.transpose(Image.ROTATE_90)
out = im.transpose(Image.ROTATE_180)
out = im.transpose(Image.ROTATE_270)

im = Image.open("hopper.ppm").convert("L")
```

Modes

The mode of an image defines the type and depth of a pixel in the image. The current release supports the following standard modes:

1 (1-bit pixels, black and white, stored with one pixel per byte)
L (8-bit pixels, black and white)
P (8-bit pixels, mapped to any other mode using a color palette)
RGB (3x8-bit pixels, true color)
RGBA (4x8-bit pixels, true color with transparency mask)
CMYK (4x8-bit pixels, color separation)
YCbCr (3x8-bit pixels, color video format)
    Note that this refers to the JPEG, and not the ITU-R BT.2020, standard
LAB (3x8-bit pixels, the L*a*b color space)
HSV (3x8-bit pixels, Hue, Saturation, Value color space)
I (32-bit signed integer pixels)
F (32-bit floating point pixels)

Resize modes

NEAREST
Pick one nearest pixel from the input image. Ignore all other input pixels.

BOX
Each pixel of source image contributes to one pixel of the destination image with identical weights. For upscaling is equivalent of NEAREST. This filter can only be used with the **resize()** and **thumbnail()** methods.
New in version 3.4.0.

BILINEAR
For resize calculate the output pixel value using linear interpolation on all pixels that may contribute to the output value. For other transformations linear interpolation over a 2x2 environment in the input image is used.

HAMMING
Produces a sharper image than BILINEAR, doesn't have dislocations on local level like with BOX. This filter can only be used with the resize() and thumbnail() methods.
New in version 3.4.0.

BICUBIC
For resize calculate the output pixel value using cubic interpolation on all pixels that may contribute to the output value. For other transformations cubic interpolation over a 4x4 environment in the input image is used.

LANCZOS
Calculate the output pixel value using a high-quality Lanczos filter (a truncated sinc) on all pixels that may contribute to the output value. This filter can only be used with the resize() and thumbnail() methods.

Filters

out = im.filter(ImageFilter.mode)

Modes

BLUR

CONTOUR

DETAIL

EDGE_ENHANCE

EDGE_ENHANCE_MORE

EMBOSS

FIND_EDGES

SHARPEN

SMOOTH

SMOOTH_MORE

Filters

```
from PIL import Image, ImageFilter
img = Image.open("tatras.jpg")
blurred = img.filter(ImageFilter.BLUR)
blurred.save("blurred.png")
```

```
Color3DLUT(size, table, channels=3, target_mode=None, **kwargs)
BoxBlur(radius)
GaussianBlur(radius=2)
UnsharpMask(radius=2, percent=150, threshold=3)
Kernel(size, kernel, scale=None, offset=0)
RankFilter(size, rank)
MedianFilter(size=3)
MinFilter(size=3)
MaxFilter(size=3)
ModeFilter(size=3)
```

Point Operations

```
# multiply each pixel by 1.2
out = im.point(lambda i: i * 1.2)

# split the image into individual bands
source = im.split()
R, G, B = 0, 1, 2
# select regions where red is less than 100
mask = source[R].point(lambda i: i < 100 and 255)
# process the green band
out = source[G].point(lambda i: i * 0.7)
# paste the processed band back, but only where red was < 100
source[G].paste(out, None, mask)
# build a new multiband image
im = Image.merge(im.mode, source)

enh = Image.Contrast(im)
enh.enhance(1.3).show("30% more contrast")
```

Drawing to Pillow image

```
from PIL import Image, ImageDraw
img = Image.new('RGBA', (200, 200), 'white')
idraw = ImageDraw.Draw(img)
idraw.rectangle((10, 10, 100, 100), fill='blue')
img.save('rectangle.png')
```

```
from PIL import Image, ImageDraw, ImageFont
tatras = Image.open("tatras.jpg")
idraw = ImageDraw.Draw(tatras)
text = "High Tatras"
font = ImageFont.truetype("arial.ttf", size=18)
idraw.text((10, 10), text, font=font)
tatras.save('tatras_watermarked.png')
```