Linear Algebra for Computer Vision Operations (A Review)

January 9th, 2019

Presented by Dr. Sander Ali Khowaja

Scalar	Vector	Matrix	Tensor
1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$

All the images in this slide are with courtesy to Google Images

Vectors

- A column vector $\mathbf{v} \in \mathbb{R}^{n \times 1}$ where

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

- A row vector $\mathbf{v}^T \in \mathbb{R}^{1 \times n}$ where

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

T denotes the transpose operation

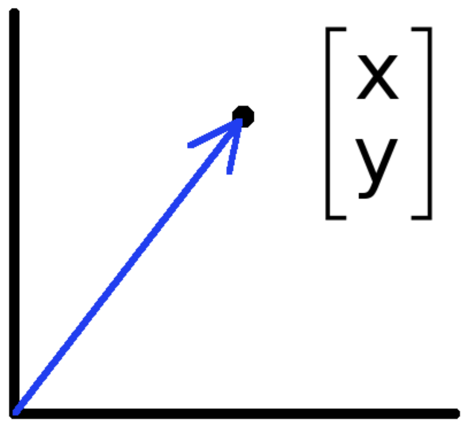
Vectors

- We'll default to column vectors in this class

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

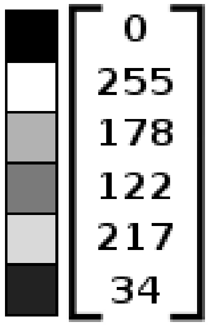
- You'll want to keep track of the orientation of your vectors when programming in python/MATLAB
- You can transpose a vector *V* in python by writing **V'** (But in class materials, we will **always** use V^T to indicate transpose

Vectors have two main uses



- Vectors can represent an offset in 2D or 3D space.
- Points are just vectors from the origin.

- Data (pixels, gradients at an image key point, etc.) can also be treated as a vector.
- Such vectors don't have a geometric interpretation, but calculations like "distance" can still have value.



Matrix

- A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is an array of numbers with size *m* x *n*, i.e. m rows and n columns.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & & & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

A

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

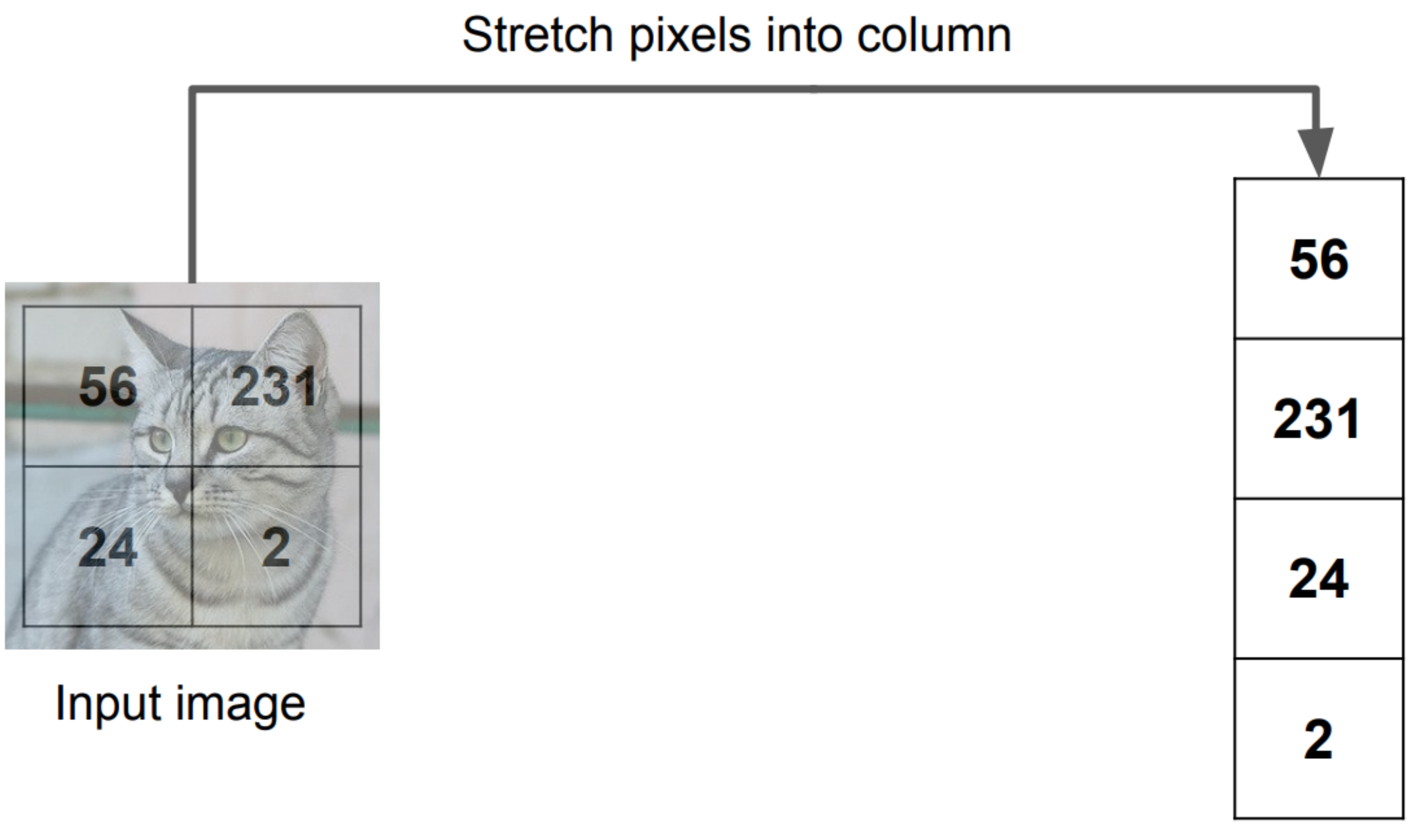
- If $m = n$, we say that $\mathbf{A}$ is square.

Images

$$= \begin{bmatrix} 193 & 180 & 210 & 112 & 125 \\ 189 & 8 & 177 & 97 & 114 \\ 100 & 71 & 81 & 195 & 165 \\ 167 & 12 & 242 & 203 & 181 \\ 44 & 25 & 9 & 48 & 192 \end{bmatrix}$$

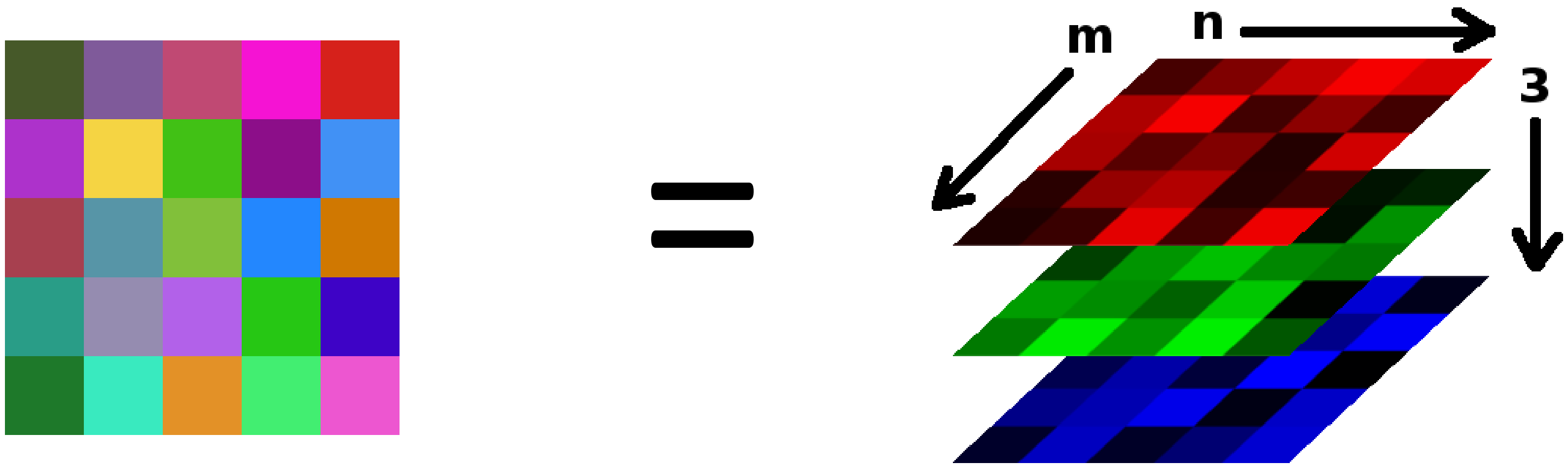
- Python/MATLAB represents an image as a matrix of pixel brightness
- Note that the upper left corner is [y,x] = (0,0) in python and [x,y] = (1,1) in MATLAB

Images as both a matrix as well as vector



Color Images

- Grayscale images have one number per pixel, and are stored as an m × n matrix.
- Color images have 3 numbers per pixel – red, green, and blue brightness (RGB)
- Stored as an m × n × 3 matrix



Basic Matrix Operations

- We will discuss:
  - Addition
  - Scaling
  - Dot product
  - Multiplication
  - Transpose
  - Inverse / pseudoinverse
  - Determinant / trace

Matrix Operations

- Addition $$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a+1 & b+2 \\ c+3 & d+4 \end{bmatrix}$$
  - Can only add a matrix with matching dimensions, or a scalar.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + 7 = \begin{bmatrix} a+7 & b+7 \\ c+7 & d+7 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times 3 = \begin{bmatrix} 3a & 3b \\ 3c & 3d \end{bmatrix}$$

Norm

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

- More formally, a norm is any function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that satisfies 4 properties:

$$x \in \mathbb{R}^n,\ f(x) \geq 0$$

- **Non-negativity:** For all
- **Definiteness**: f(x $x \in \mathbb{R}^n,\ t \in \mathbb{R},\ f(tx) = |t|\, f(x)$
- **Homogeneity**: For al $x, y \in \mathbb{R}^n,\ f(x+y) \leq f(x) + f(y)$
- **Triangle inequality**: For all

Norms

- **Example Norms**

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \qquad \|x\|_\infty = \max_i |x_i|.$$

- General ℓ_p norms:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

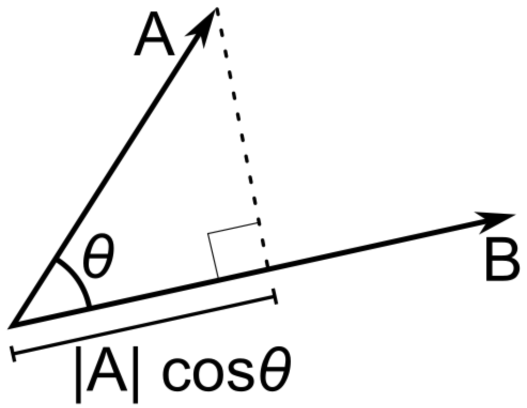
- **Matrix norms:** Norms can also be defined for matrices, such

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2} = \sqrt{\mathrm{tr}(A^T A)}.$$

Product

- Inner product (dot product) of vectors
  - Multiply corresponding entries of two vectors and add up the result
  - x·y is also |x||y|Cos(*the angle between x and y*)
  - If B is a unit vector, then A·B gives the length of A which lies in the direction of B

$$\mathbf{x}^T\mathbf{y} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^n x_i y_i \qquad \text{(scalar)}$$



Product

- The product of two matrices

$$A \in \mathbb{R}^{m\times n}$$

$$B \in \mathbb{R}^{n\times p}$$

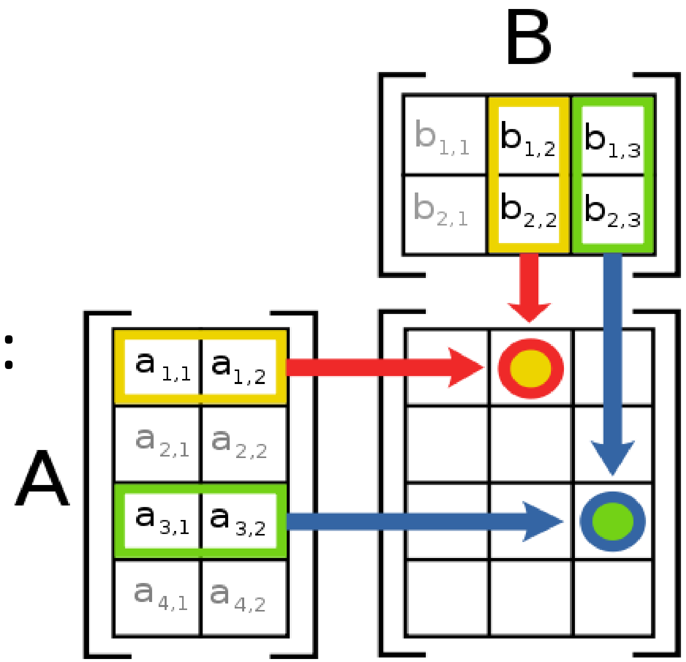
$$C = AB \in \mathbb{R}^{m\times p}$$

$$C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$$

$$C = AB = \begin{bmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \cdots & b_p \\ | & | & & | \end{bmatrix} = \begin{bmatrix} a_1^T b_1 & a_1^T b_2 & \cdots & a_1^T b_p \\ a_2^T b_1 & a_2^T b_2 & \cdots & a_2^T b_p \\ \vdots & \vdots & \ddots & \vdots \\ a_m^T b_1 & a_m^T b_2 & \cdots & a_m^T b_p \end{bmatrix}.$$

Product

- Multiplication
- The product AB is:



- Each entry in the result is (that row of A) dot product with (that column of B)
- Many uses, which will be covered later

Product

- Multiplication example:

A x B

$$\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 2 \\ 4 & 6 \end{bmatrix} \quad \begin{bmatrix} 10 & 14 \\ 34 & 54 \end{bmatrix}$$

$$0 \cdot 3 + 2 \cdot 7 = 14$$

– Each entry of the matrix product is made by taking the dot product of the corresponding row in the left matrix, with the corresponding column in the right one.

Matrix multiplication is associative: $(AB)C = A(BC)$.

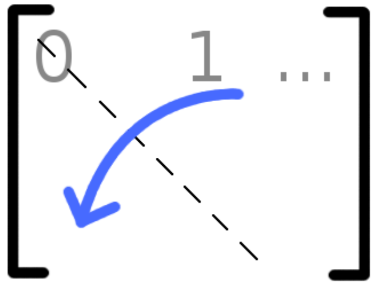
Matrix multiplication is distributive: $A(B + C) = AB + AC$.

Matrix multiplication is, in general, *not* commutative; that is, it can be the case that $AB \neq BA$. (For example, if $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times q}$, the matrix product BA does not even exist if m and q are not equal!)

- Powers
  - By convention, we can refer to the matrix product AA as A^2, and AAA as A^3, etc.
  - Obviously only square matrices can be multiplied that way

Transpose

- Transpose – flip matrix, so row 1 becomes column



$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

- A useful identity:

$$(ABC)^T = C^T B^T A^T$$

Determinant

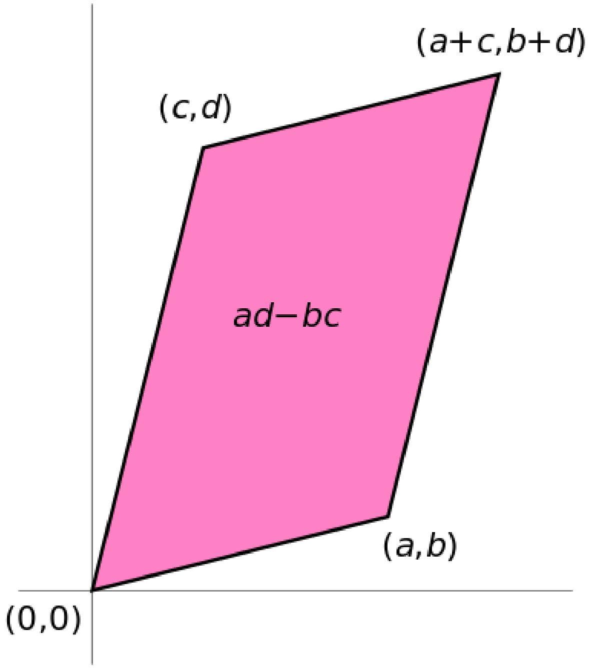
- **Determinant**
  - $\det(\mathbf{A})$ returns a scalar
  - Represents area (or volume) of the parallelogram described by the vectors in the rows of the matrix
  - For $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\det(\mathbf{A}) = ad - bc$
  - Properties:

$$\det(\mathbf{AB}) = \det(\mathbf{BA})$$

$$\det(\mathbf{A}^{-1}) = \frac{1}{\det(\mathbf{A})}$$

$$\det(\mathbf{A}^T) = \det(\mathbf{A})$$

$$\det(\mathbf{A}) = 0 \Leftrightarrow \mathbf{A} \text{ is singular}$$



Trace

$$\text{tr}(\mathbf{A}) = \text{sum of diagonal elements}$$

$$\mathbf{tr}(\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}) = 1 + 7 = 8$$

- Invariant to a lot of transformations, so it's used sometimes in proofs. (Rarely in this class though.)

- Properties:

$$\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA})$$

$$\text{tr}(\mathbf{A} + \mathbf{B}) = \text{tr}(\mathbf{A}) + \text{tr}(\mathbf{B})$$

Special Matrices

- Identity matrix **I**
  - Square matrix, 1's along diagonal, 0's elsewhere
  - **I** · [another matrix] = [that matrix]

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Diagonal matrix
  - Square matrix with numbers along diagonal, 0's elsewhere
  - A diagonal · [another matrix] scales the rows of that matrix

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2.5 \end{bmatrix}$$

Special Matrices

- Symmetric **matrix**

$$\mathbf{A}^T = \mathbf{A}$$

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 7 \\ 5 & 7 & 1 \end{bmatrix}$$

- Skew-symmetric matrix

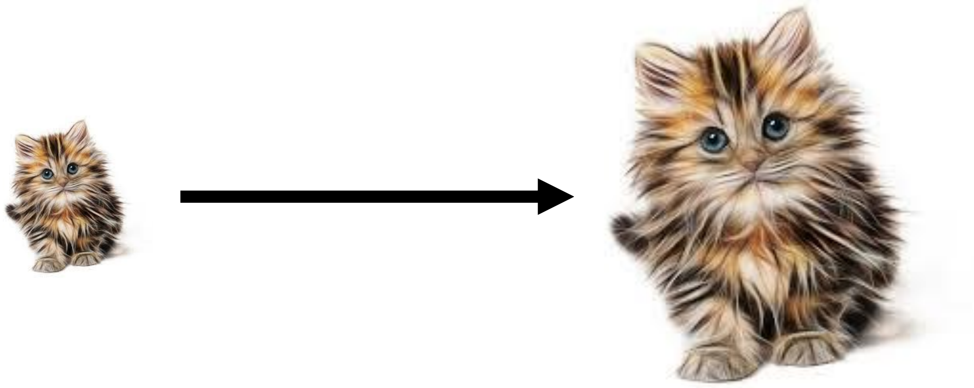
$$\mathbf{A}^T = -\mathbf{A}$$

$$\begin{bmatrix} 0 & -2 & -5 \\ 2 & 0 & -7 \\ 5 & 7 & 0 \end{bmatrix}$$

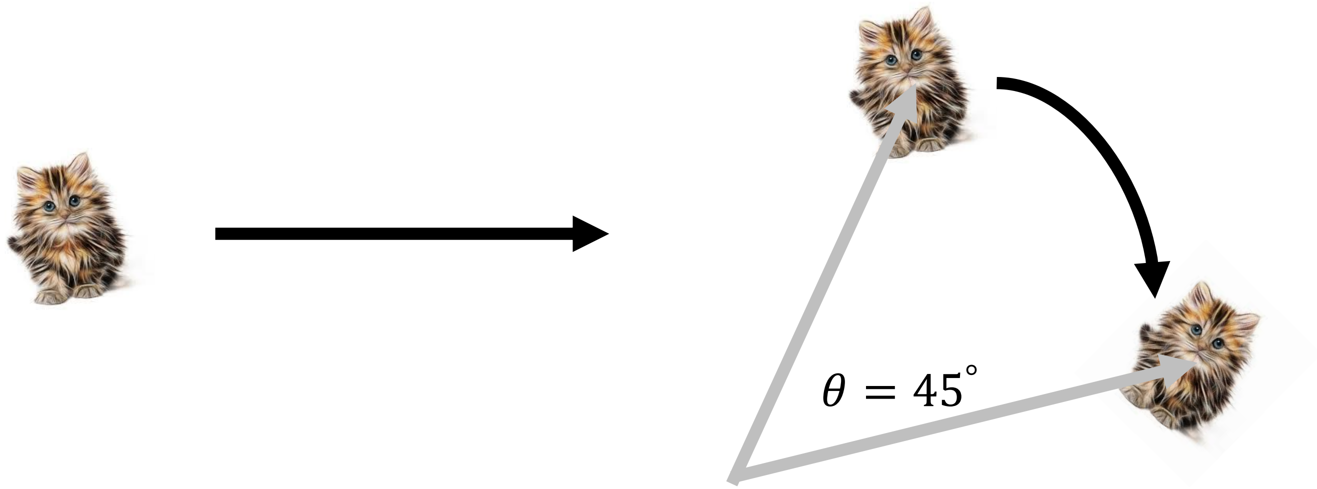
Transformations (Scaling)

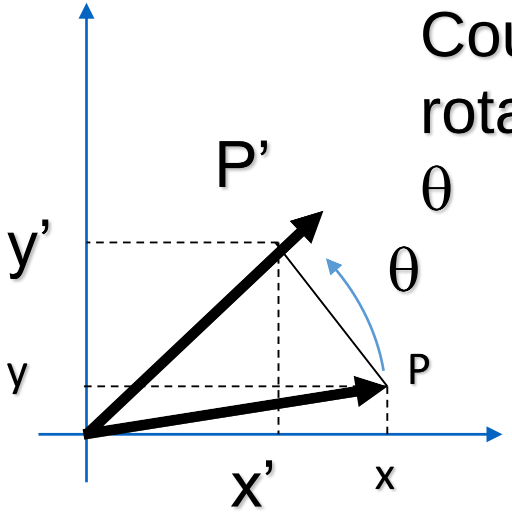
- Matrices can be used to transform vectors in useful ways, through multiplication: x'= Ax
- Simplest is scaling: $$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$

(Verify to yourself that the matrix multiplication works out this way)



Transformations (Rotation)





Counter-clockwise rotation by an angle θ

$$x' = \cos\theta\, x - \sin\theta\, y$$

$$y' = \cos\theta\, y + \sin\theta\, x$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P'} = \mathbf{R}\,\mathbf{P}$$

Homogeneous systems (multiple transformations)

- In general, a matrix multiplication lets us linearly combine components of a vector

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

  - This is sufficient for scale, rotate, skew transformations.
  - But notice, we can't add a constant! ☹

  - The (somewhat hacky) solution? Stick a "1" at the end of every vector:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

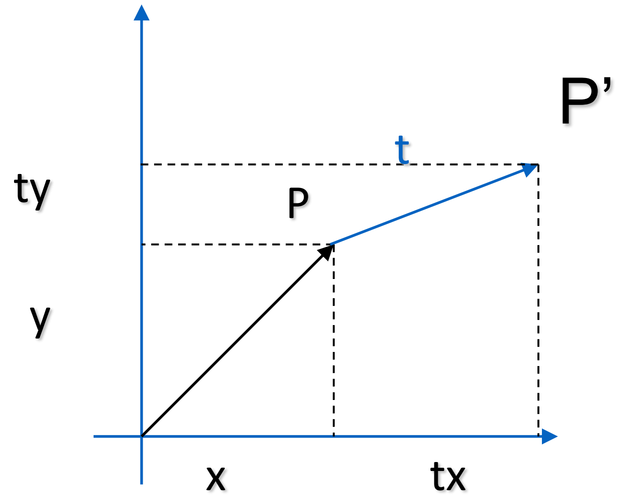
  - Now we can rotate, scale, and skew like before, AND translate (note how the multiplication works out, above)
  - This is called "homogeneous coordinates"

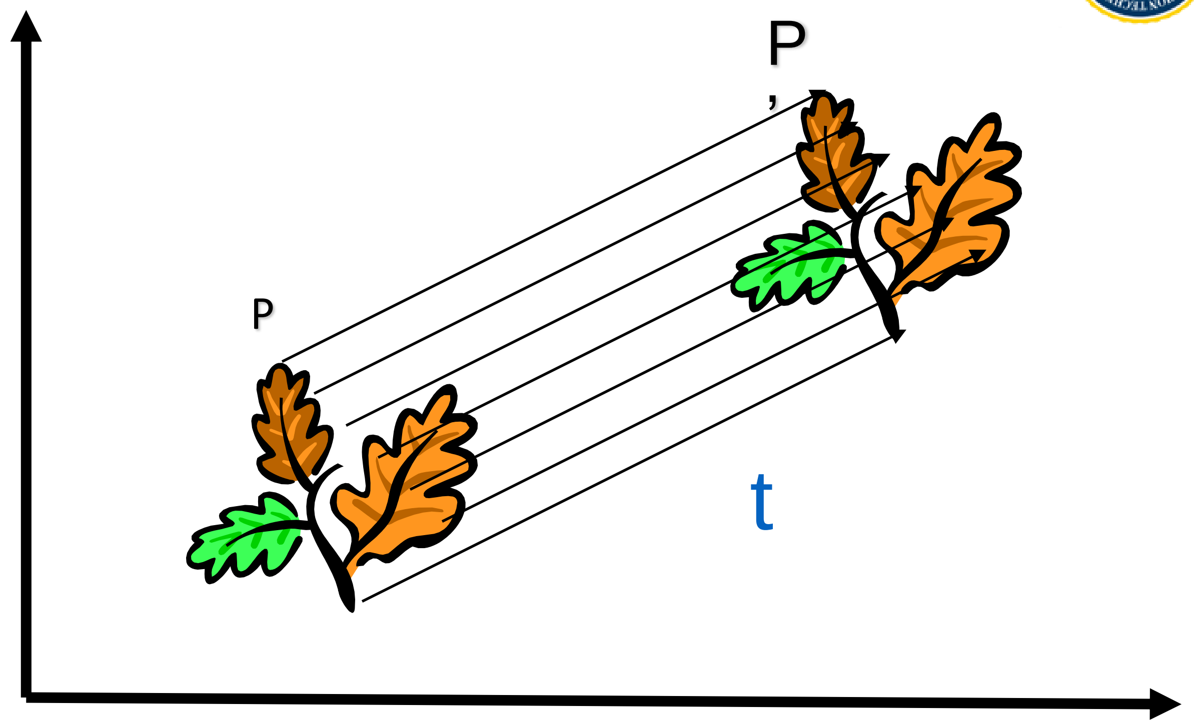
Homogeneous systems (division)

- One more thing we might want: to divide the result by something
  - For example, we may want to divide by a coordinate, to make things scale down as they get farther away in a camera image
  - Matrix multiplication can't actually divide
  - So, **by convention**, in homogeneous coordinates, we'll divide the result by its last coordinate after doing a matrix multiplication

$$\begin{bmatrix} x \\ y \\ 7 \end{bmatrix} \Rightarrow \begin{bmatrix} x/7 \\ y/7 \\ 1 \end{bmatrix}$$

2-D Translation



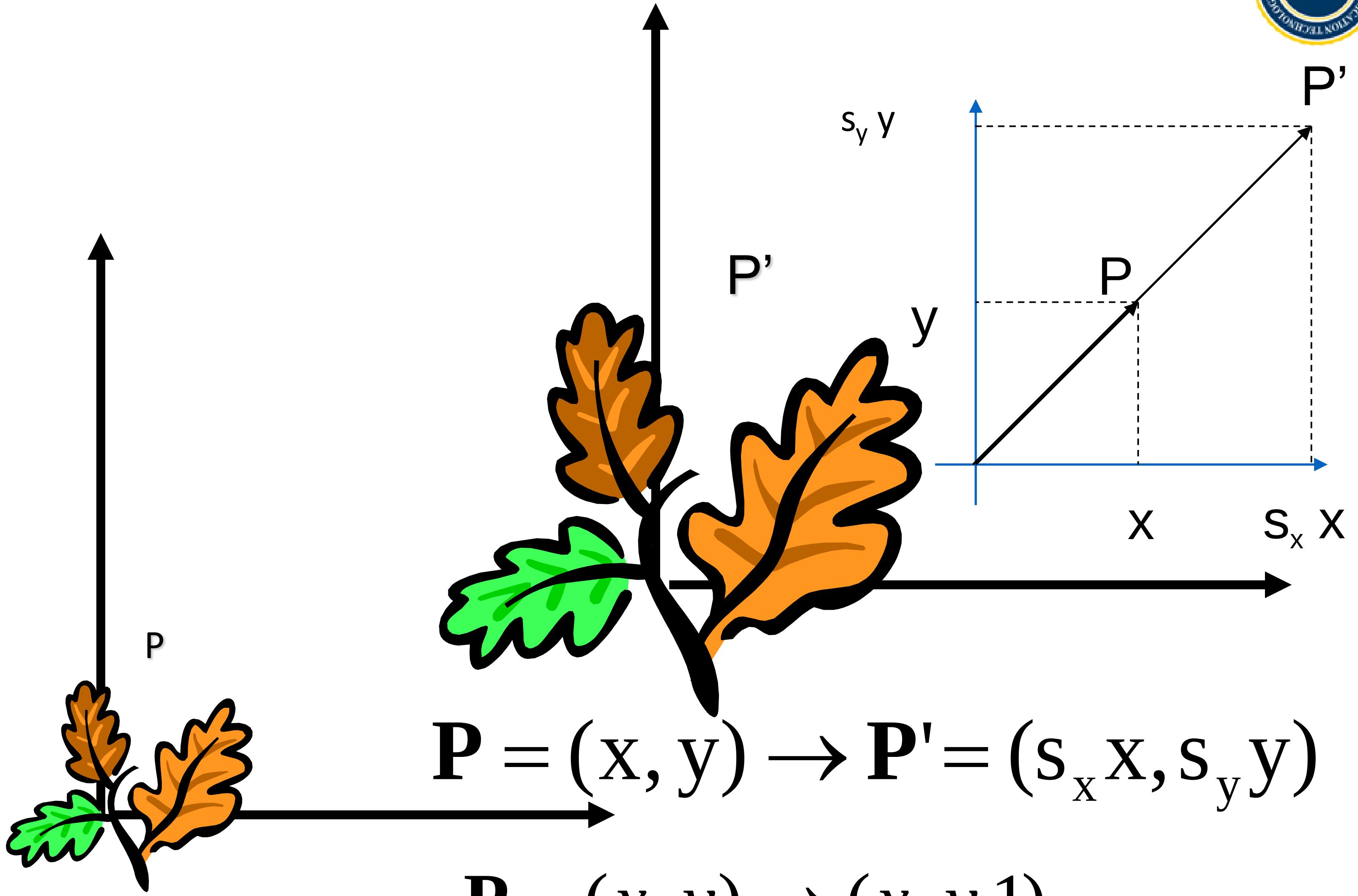


$$\mathbf{P}' \rightarrow \begin{bmatrix} x+t_x \\ y+t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P} = (x, y) \rightarrow (x, y, 1)$$

$$\mathbf{t} = (t_x, t_y) \rightarrow (t_x, t_y, 1)$$

Scaling

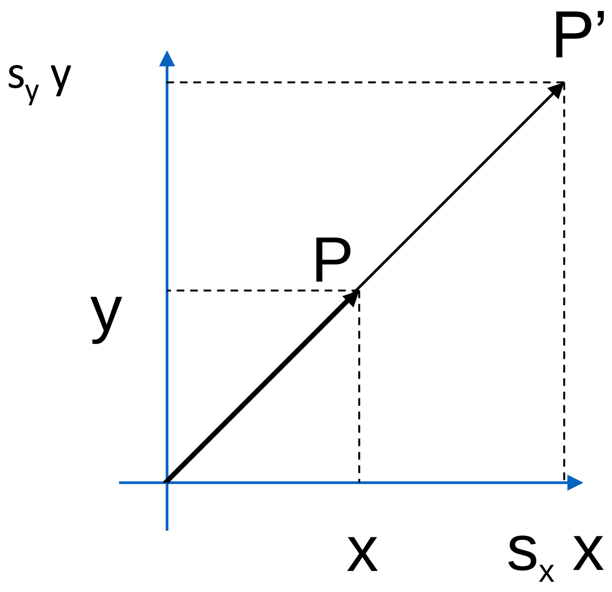


$$\mathbf{P} = (x, y) \rightarrow \mathbf{P'} = (s_x x, s_y y)$$

$$\mathbf{P} = (x, y) \rightarrow (x, y, 1)$$

$$\mathbf{P'} = (s_x x, s_y y) \rightarrow (s_x x, s_y y, 1)$$

Scaling



$$\mathbf{P} = (x, y) \rightarrow \mathbf{P}' = (s_x x, s_y y)$$

$$\mathbf{P} = (x, y) \rightarrow (x, y, 1)$$

$$\mathbf{P}' = (s_x x, s_y y) \rightarrow (s_x x, s_y y, 1)$$

$$\mathbf{P}' \rightarrow \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{S}} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}' & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \mathbf{P} = \mathbf{S} \cdot \mathbf{P}$$

Scaling and Translating

$$\mathbf{P}'' = \mathbf{T} \cdot \mathbf{S} \cdot \mathbf{P} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} S & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translating & Scaling != Scaling & Translating

$$\mathbf{P'''} = \mathbf{T}\cdot\mathbf{S}\cdot\mathbf{P} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix}$$

$$\mathbf{P'''} = \mathbf{S}\cdot\mathbf{T}\cdot\mathbf{P} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} s_x & 0 & s_x t_x \\ 0 & s_y & s_y t_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + s_x t_x \\ s_y y + s_y t_y \\ 1 \end{bmatrix}$$

Scaling + Rotation + Translation

P'= (T R S) P

$$\mathbf{P}' = \mathbf{T}\cdot R \cdot \mathbf{S}\cdot \mathbf{P} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boxed{\begin{bmatrix} R\,S & t \\ 0 & 1 \end{bmatrix}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This is the form of the general-purpose transformation matrix

Inverse

- Given a matrix **A**, its inverse $\mathbf{A}^{-1}$ is a matrix such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
- E.g. $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$
- Inverse does not always exist. If $\mathbf{A}^{-1}$ exists, **A** is *invertible* or *non-singular*. Otherwise, it's *singular*.
- Useful identities, for matrices that are invertible:

$$(\mathbf{A}^{-1})^{-1} = \mathbf{A}$$

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$$

$$\mathbf{A}^{-T} \triangleq (\mathbf{A}^{T})^{-1} = (\mathbf{A}^{-1})^{T}$$

PseudoInverse

- Fortunately, there are workarounds to solve AX=B in these situations. And python can do them!
- Instead of taking an inverse, directly ask python to solve for X in AX=B, by typing **np.linalg.solve(A, B)**
- Python will try several appropriate numerical methods (including the pseudoinverse if the inverse doesn't exist)
- Python will return the value of X which solves the equation
  - If there is no exact solution, it will return the closest one
  - If there are many solutions, it will return the smallest one

Just for the sake of practicality

- Load any grayscale image
- I2=I/16;
- Imshow(I2,[]);
- Reduces the range from 0…255 to 0…16
- Or can do a logical AND operation
- I2=bitand(I,240);
- To reduce the spatial resolution sample every other row
- I3=I(1:2:end,1:2:end);
- Or use MATLAB’s imresize function
- I3=imresize(I,0.5);

Just for the information

- Install Python by visiting https://www.anaconda.com/distribution/
- The latest version is Python 3.7, I would recommend to search for python 3.6 instead if you want to use tensorflow in future.

- Once you have downloaded, install it.
- After the installation open Anaconda Prompt by typing the same in the windows search bar.
- At command prompt type pip install opencv-contrib-python
- Make sure the internet connection is stable.

Just for the information

- After the successful installation try the following
  - At Anaconda Prompt type python
  - Make sure the cursor changes to >>
  - Writing the following commands
  - Import cv2
  - Image = cv2.imread(“image path.image extension”)
  - Print(“width: {} pixels”.format(image.shape[1]))
  - Print(“height: {} pixels”.format(image.shape[0]))
  - Print(“channels: {}”.format(image.shape[2]))

  - Cv2.imshow(“Image”, image)
  - Cv2.waitKey(0)

Just for the information

- If you are using OpenCV with C++

```
//

#include "stdafx.h"

#include <opencv/cv.hpp>
#include <opencv/highgui.h>
//// properties, c/c++ C:\opencv\build\include
//// properties, linker general additional directories C:\opencv\build\x64\vc14\lib
//// properties, linker, input, additional dependencies, opencv_world320.lib (release) opencv_world320d.lib(debug)
//// copy opencv_world320.dll file from opencv folder to release exe file
//// copy opencv_world320d.dll file from opencv folder to debug exe file

int main()
{
        cv::Mat image;
        image = cv::imread("../testimage.jpg", CV_LOAD_IMAGE_COLOR );
        cv::imshow("original image", image);

        for (int h = 0; h < image.size().height; h++ ) {
                for (int w = 0; w < image.size().width; w++) {
                        for (int c = 0; c < image.channels(); c++ ) {
                                int v = image.data[h*(image.size().width) * image.channels() + w * image.channels() +c];
                                v *= 2;
                                if (v > 255) { v = 255; }
                                image.data[h*(image.size().width)*image.channels() + w*image.channels() + c] = v;
                        }
                }
        }

        cv::imshow("modified image", image);
        cv::imwrite("../newimage.bmp", image);

        cv::waitKey();
    return 0;
}
```

- *Questions*
  - *Feel Free to ask*