

Table of Contents

NEW_AI	1.1
Part1. Object Detection	1.2
1. Template Matching	1.2.1
2. Corner Detection	1.2.2
3. Edge Detection	1.2.3
4. Contour Detection	1.2.4
6. Integrated Project - Basketball	1.2.5
7. Yolo-V5	1.2.6
internGPT	1.2.7
Housing Price	1.2.8
avocado_price	1.2.9
intro	1.3
chatgpt	1.4
problem1	1.4.1

Python Artificial Intelligent Projects

Tutorial

How to read the code?

- Click the links listed in catalog. Each folder has a README.md file which will tell you concepts, theories and codes of the module.

How to contact me?

- You can send me email directly
- Contact me on my personal website: <http://www.andiogame.com>

Catalog

:muscle:Introduction

- [Introduction English Version] Sorry, I need time to prepare English Tutorial
- [人工智能普通話版 #1. 課程簡介](#)
- [人工智能粵語版 #1. 課程簡介](#)

:muscle:OpenCV

Object Detection

- [Template Matching](#)
 - [English Youtube Video]
 - 普通話 Youtube 影片
 - 廣東話 Youtube 影片
- [Corner Detection](#)
 - [English Youtube Video]
 - 普通話 Youtube 影片
 - 廣東話 Youtube 影片
- [Edge Detection](#)

:muscle:Tensorflow Supervised Learning

- Deep Binary Learning
- Deep Classification
- Deep Regression

:muscle:Reinforcement Learning

- Q-Learning
- Deep Q-Learning

:muscle:Integrated Projects:

Web+AI

- How to deploy AI solutions with webpage or API?
 - Django + TensorFlow

Game Bots

- How to solve puzzle and dragon with limited steps?
 - Reinforcement Learnign

Robotic AI

- How to use Rasberry PI to build your first smart robot?
 - Electronics + Reinforcement Learning + OpenCV + Simulations

Template Matching

A. Usage

- Detect a copped image inside a image
- Text Recognition in early age (Fixed size)
- Detection of 2D game



B. Limitation

- Fixed size of target (If target is blend or modified, hardly to detect the image)
- The copped image should exited in the image

C. Time complexity

- matchTemplate: $O(n^3)$
- minMaxLoc: $O(n^2)$
- Total: $O(n^3)$

D. Code highlight

Simple matching

```
full_copy = img.copy()

method = eval('cv2.TM_CCOEFF')
res = cv2.matchTemplate(full_copy,target1,method)

height, width,channels = target1.shape
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

# Assign the Bottom Right of the rectangle
top_left = max_loc
bottom_right = (top_left[0] + width, top_left[1] + height)

# Draw the Red Rectangle
cv2.rectangle(full_copy,top_left, bottom_right, 255, 10)
plt.imshow(full_copy)
plt.show()
```

Trick code

```
if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:  
    top_left = min_loc  
else:  
    top_left = max_loc
```

E. Other concepts

- Calculate shape of result = $W-w+1, H-h+1$
- [Original Documentation](#)
- [Equations of matching](#)

Template Matching

A. Outcome



B. Usage

- Calculate differentiation of regions to identify the corner of image
- Check boards, Grids (A solution to detect edge but not very useful)

C. Limitation

- Depend on color intensity (greatly affected by surrounding environment)
- Model isn't flexible enough

D. Code highlight

Harris Corner Detection

```

@widgets.interact(x=(0, 1.0, 0.01), blockSize=(1,10), ksize=(3,30,2))

def f(x=0, blockSize=1, ksize=3):
    dst = cv2.cornerHarris(src=img2_gray,blockSize=blockSize,ksize=ksize,k=0.04)
    dst = cv2.dilate(dst,None)
    img2_copy = img2.copy()
    img2_copy[dst>x*dst.max()]=[255,0,0]

    plt.imshow(img2_copy)

```

Parameters

- double x: Ajust threshold (0.0, 1.0)
- int blockSize: Neighborhood size (It is the size of neighbourhood considered for corner detection.)
- int ksize: Aperture parameter for the Sobel operator (Size of sober kernel, should be odd number) size increases, more pixels are part of each convolution process and the edges will get more blurry.
- double k: Harris detector free parameter (0.0, 1.0)

Shi-Tomasi Corner Detector

```

@widgets.interact(maxCorners=(1,30), qualityLevel=(1,100), minDistance=(1,100))
def f(maxCorners=1, qualityLevel=1, minDistance=1):
    img2_copy = img2.copy()

    corners = cv2.goodFeaturesToTrack(image=img2_gray, maxCorners=maxCorners, quality=qualityLevel)
    corners = np.int0(corners)

    for i in corners:
        x,y = i.ravel()
        cv2.circle(img2_copy,(x,y),3,255,-1)

    plt.imshow(img2_copy)

```

Parameters

- int maxCorners: Maximum point to detect
- double qualityLevel: Parameter characterizing the minimal accepted quality of image. (A threshold to reject some scores, similar to x)
- double minDistance: Set up minimum Euclidean distance

E. Reference

- [Use slide bar in Jupyter notebook](#)
- [Theory](#)

Template Matching

:+1:Outcome



:+1:Usage

- Find Edge of an image
- This is an image processing skill

:+1:Limitation

- None

:+1:Code highlight

Canny with Manual Justification

```
threshold1 = 17
threshold2 = 65

edges = cv2.Canny(image=blurred_img, threshold1=threshold1, threshold2=threshold2)

plt.subplot(1,2,1)
plt.imshow(edges, cmap='gray_r')
plt.subplot(1,2,2)
plt.imshow(img)
plt.show()
```

Parameters

- int threshold1: [0, 255]
- int threshold2: [0, 255]

Canny with Auto Median Detection

```
``` blurred_img = cv2.blur(img,ksize=(5,5)) median_val = np.median(img)

@widgets.interact(const1=(0, 1, 0.1), const2=(0, 1, 0.1)) def f(const1=0,
const2=1):
```

```
lower = int(max(0, const1* median_val))
upper = int(min(255,const2 * median_val))

edges = cv2.Canny(image=blurred_img, threshold1=lower, threshold2=upper)
plt.imshow(edges, cmap='gray_r')
```

...

## :+1:Problem Lists

- Why blur the image?
  - Reduce noise and unnecessary details in image

## :+1:Reference

- [Canny](#)

# Contour Detection

## A. Outcomes



## B. Usage

- Calculate differentiation of two region to identify the corner of image
- Check boards, Grids (A solution to detect edge but not very useful)

## C. Limitation

- Depend on color intensity (greatly affected by surrounding environment)
- Model isn't flexible enough

## D. Code highlight

### Harris Corner Detection

```
@widgets.interact(x=(0, 1.00000), blockSize=(1,10), ksize=(3,30))

def f(x=0, blockSize=1, ksize=3):
 if ksize%2==0:
 ksize+=1
 dst = cv2.cornerHarris(src=img2_gray,blockSize=blockSize,ksize=ksize,k=0.04)
 dst = cv2.dilate(dst,None)
 img2_copy = img2.copy()
 img2_copy[dst>x*dst.max()]=[255,0,0]

 plt.imshow(img2_copy)
```

### Parameters

- double x: Ajust threshold (0.0, 1.0)
- int blockSize: Neighborhood size (It is the size of neighbourhood considered for corner detection.)
- int ksize: Aperture parameter for the Sobel operator (Size of sober kernel, should be odd number) size increases, more pixels are part of each convolution process and the edges will get more blurry.
- double k: Harris detector free parameter (0.0, 1.0)

### Shi-Tomasi Corner Detector

```
@widgets.interact(maxCorners=(1,30), qualityLevel=(1,100), minDistance=(1,100))
def f(maxCorners=1, qualityLevel=1, minDistance=1):
 img2_copy = img2.copy()

 corners = cv2.goodFeaturesToTrack(image=img2_gray, maxCorners=maxCorners, qualityLevel=qualityLevel, minDistance=minDistance)
 corners = np.int0(corners)

 for i in corners:
 x,y = i.ravel()
 cv2.circle(img2_copy,(x,y),3,255,-1)

 plt.imshow(img2_copy)
```

## Parameters

- int maxCorners: Maximum point to detect
- double qualityLevel: Parameter characterizing the minimal accepted quality of image. (A threshold to reject some scores, similar to x)
- double minDistance: Set up minimum Euclidean distance

## E. Reference

- [Use slide bar in Jupyter notebook](#)
- [Theory](#)

# Object Detection, Tracking of Basketballs

## Failure Experience:

1. cv2.HoughCircles only find object with ball shape, but there are many similar objects in the real world can be classified as ball object

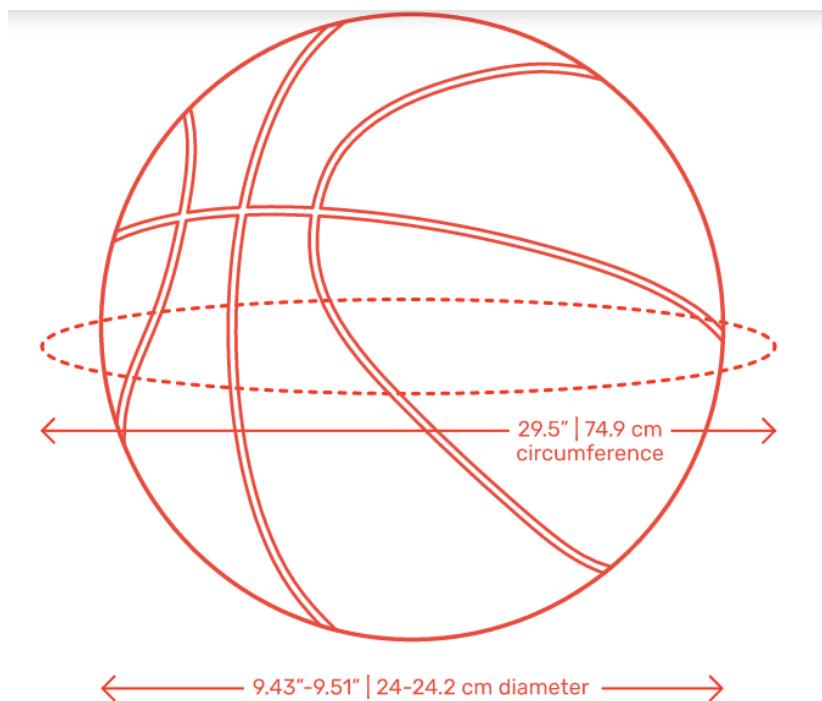
## A. Basket ball features

### A1. General feautures

- shape: ball / circle
- color: organge

### A2. Minor features

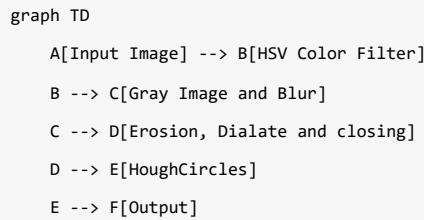
- line: black\_line
- size in real world: 12-12.1 cm radius



## B. Solutions

1. HSV Color Filter
2. Gray Image and Blur
3. Erosion, Dialate and closing: [Reference link](#)
4. HoughCircles

## C. Diagram



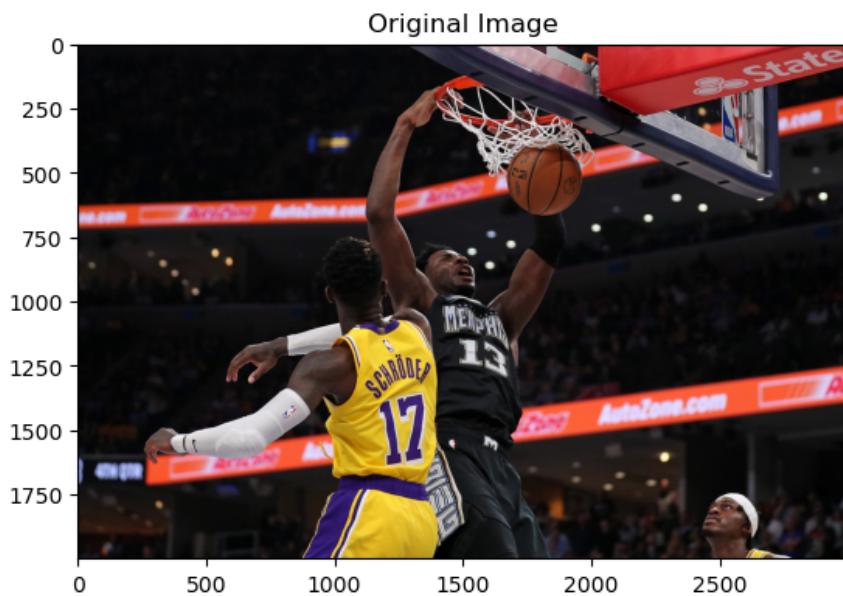
## C. Implementation

### Import required libraries

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

### C1. Read Original Image

```
im = cv2.imread("data\\png_ball1.png")
im_original = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
plt.title("Original Image")
plt.imshow(im_original)
plt.show()
```



### C2. HSV Color Filter

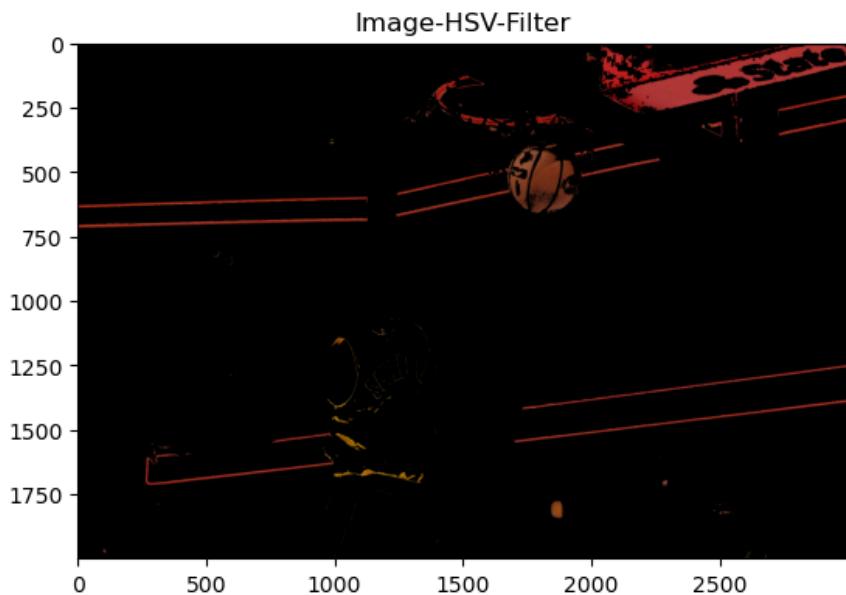
```

im_hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)

take only the orange, highly saturated, and bright parts
im_hsv = cv2.inRange(im_hsv, (7, 180, 180), (11, 255, 255))
im_hsv = cv2.inRange(im_hsv, (0, 150, 100), (20, 255, 200))

To show the detected orange parts:
im_orange = im.copy()
im_orange[im_hsv == 0] = 0
plt.title("Image-HSV-Filter")
plt.imshow(cv2.cvtColor(im_orange, cv2.COLOR_BGR2RGB))
plt.plot()

```

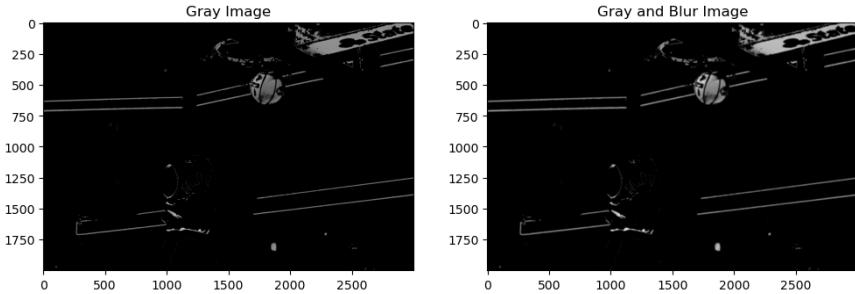


### C3. Gray Image and Gaussian Blur

```

im_gray = cv2.cvtColor(im_orange, cv2.COLOR_BGR2GRAY)
ksize = (10, 10)
im_gray.blur = cv2.blur(im_gray, ksize)
plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.title("Gray Image")
plt.imshow(im_gray, cmap="gray")
plt.subplot(122)
plt.title("Gray and Blur Image")
plt.imshow(im_gray.blur, cmap="gray")
plt.show()

```



## C4. Erosion, Dialate and closing

Erosion:

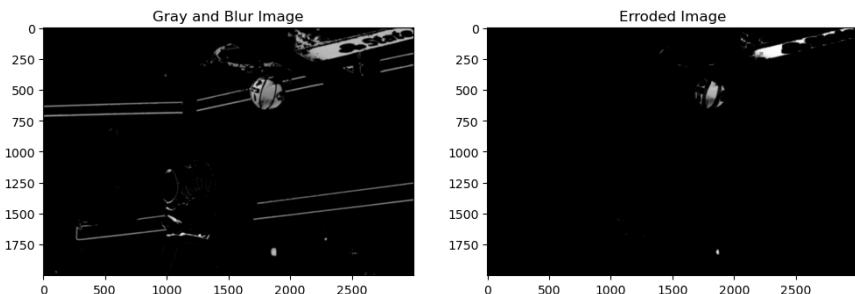
```

kernel = np.ones((21, 21), np.uint8)
erosion = cv2.erode(im_gray_blur, kernel, iterations=1)

plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.title("Gray and Blur Image")
plt.imshow(im_gray_blur, cmap="gray")

plt.subplot(122)
plt.title("Eroded Image")
plt.imshow(erosion, cmap="gray")
plt.show()

```



Dilate:

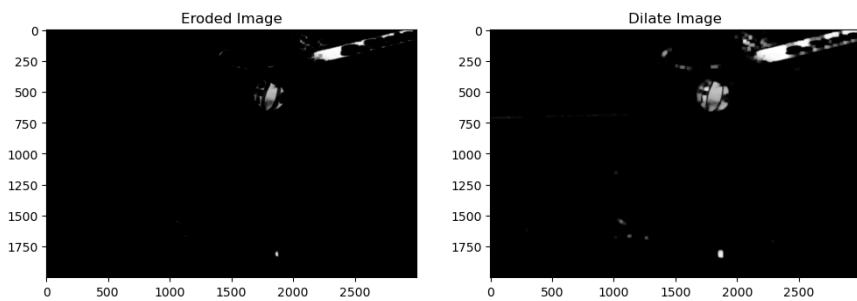
```

kernel = np.ones((11, 11), np.uint8)
dilate = cv2.dilate(erosion, kernel, iterations=2)

plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.title("Eroded Image")
plt.imshow(erosion, cmap="gray")

plt.subplot(122)
plt.title("Dilate Image")
plt.imshow(dilate, cmap="gray")
plt.show()

```



**Closing:**

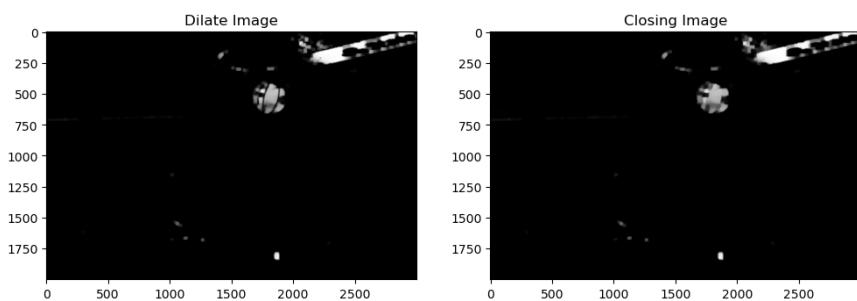
```

kernel = np.ones((21, 21), np.uint8)
closing = cv2.morphologyEx(dilate, cv2.MORPH_CLOSE, kernel)

plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.title("Dilate Image")
plt.imshow(dilate, cmap="gray")

plt.subplot(122)
plt.title("Closing Image")
plt.imshow(closing, cmap="gray")
plt.show()

```



## C5. Detection: HoughCircles

```

im_copy = im.copy()
circles = cv2.HoughCircles(closing, cv2.HOUGH_GRADIENT, 1, 50,
 param1=50, param2=30, minRadius=10, maxRadius=300)

if circles is not None:
 circles = np.round(circles[0, :]).astype("int")
 for (x, y, r) in circles:

 cv2.circle(im_copy, (x, y), r, (0, 255, 0), 2)

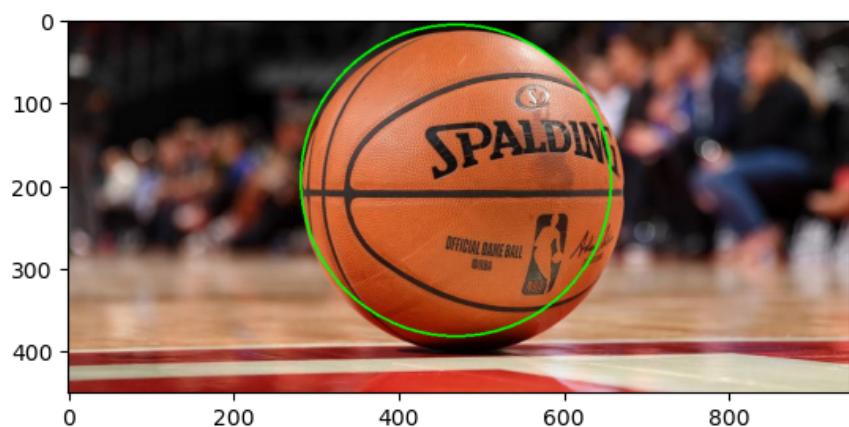
Display the processed frame
cv2.imshow('Ball Detection', img)
plt.imshow(cv2.cvtColor(im_copy, cv2.COLOR_BGR2RGB))

```



## D. Testing and Evaluation

### D1. Detect a ball in a image



## D2. Detect a ball in a complex image



## A. Installation

1. Clone the Yolo-v5 repository from github

```
!git clone https://github.com/ultralytics/yolov5
%cd yolov5/
```

1. Install necessary libraries for Yolo-V5 training

2. Use requirements.txt to install all required libraries quickly with following command

```
!pip install -qr requirements.txt
```

- Common libraries for deep learning
  - torch: a standrad deep learning library lauched by Facebook
  - scipy: libray for scientific studies. Some tools such as Matrix calculation
  - pandas: DataFrame library. Package data to a frame and display information easily
  - numpy: sub-library in scipy.
  - matplotlib.pyplot: Display library with graph
- Check the setup

```
import torch

from IPython.display import Image, clear_output # to display images
from utils.downloads import attempt_download # to download models/datasets

clear_output()
print('Setup complete. Using torch %s %s' % (
 torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
```

It will output the cuda version of your computer. It is not important information for training but usually give you correct version if the current environment is available.

## B. Deploy training data to Roboflow

Roboflow works like Google Image but it is a platform to help you process and transfer images to the formated image of machine learning.

Alternatively, you can use below tools to reduce costs of training

- [CVAT](#)
- [LabelImg](#)

- [VOTT](#)

Although Roboflow will make your training easy, number of images / training data is limited. We may consider to use other tools to cut down the costs. However, we will use Roboflow to store training data at current stage because it greatly improve develop experience.

## Download your training data with python code

1. Install roboflow

```
!pip install -q roboflow
```

2. Direct to your data location

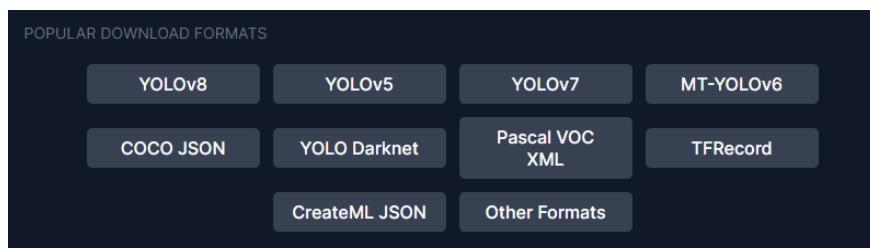
```
%cd /content/yolov5
```

3. Run the python code to download formated data

```
from roboflow import Roboflow
rf = Roboflow(api_key="{api_secret}")
project = rf.workspace("{workspace}").project("{project_name}")
This line will define what type of data you want to download from roboflow
dataset = project.version(1).download("yolov5")
```

In this example, you can use some official data provided by [Roboflow Data](#)

Different formats provided by Roboflow



1. Check your downloaded data

```
%cat {dataset.location}/data.yaml
```

"data.yaml" is a file to tell the system how to import images and labels from the directories

```
names:
- head
- helmet
- person
nc: 3
roboflow:
 license: Public Domain
 project: hard-hat-sample-zsg3b
 url: https://app.roboflow.com/test-zl4ft/hard-hat-sample-zsg3b/1
 version: 1
 workspace: test-zl4ft
 test: ../test/images
 train: Hard-Hat-Sample-1/train/images
 val: Hard-Hat-Sample-1/valid/images
```

Class names  
Number of classes  
Directories

Modify this template if you are using other tools to process training images

```
names:
- head
- helmet
- person
nc: 3
test: ../test/images
train: Hard-Hat-Sample-1/train/images
val: Hard-Hat-Sample-1/valid/images
```

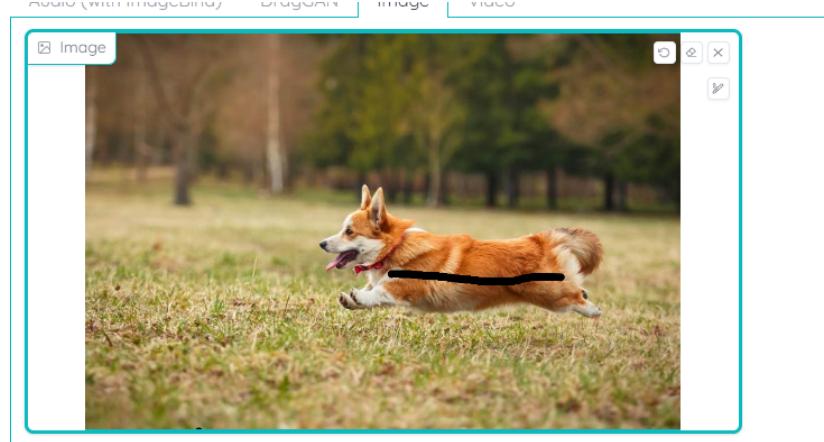
# Background

InternGPT is a pointing-language-driven visual interactive system which allows you to send images as input for GPT model. There are some basic functions should be supported by this project including click, drag and draw.

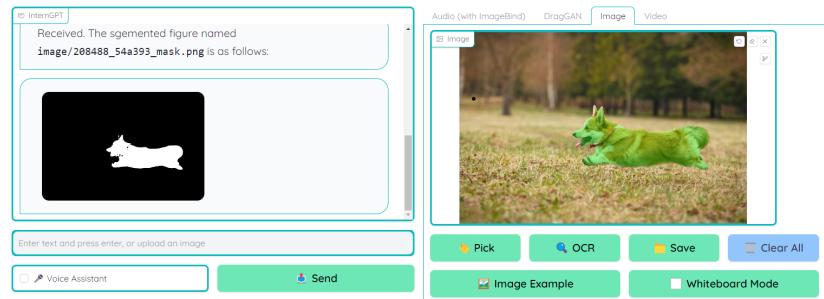
## Investigation

### A. Click

1. Draw a line to point some image objects in the photo

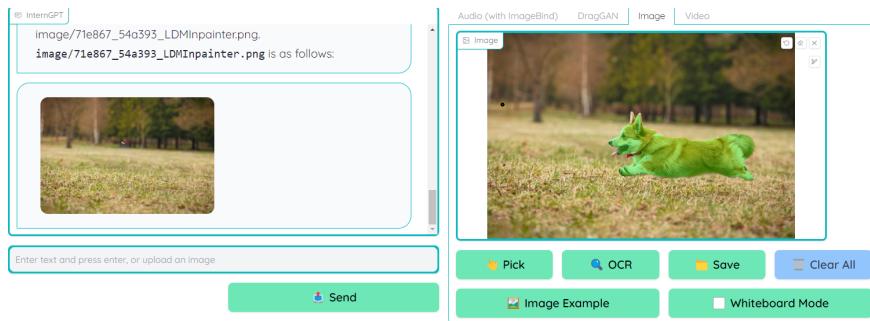


2. Click the pick button to select the object After processing, you are able to see the output mask image in the message box, suprsingly, the system can bound the image with high accuracy



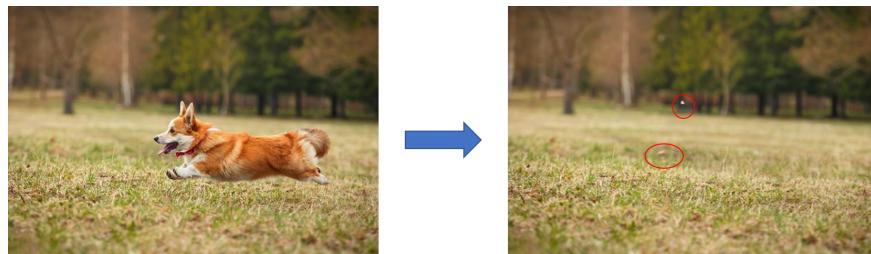
3. Send some commands (only natural language is required) to the message prompt and see the result

e.g. remove the dog in the image



## Evaluation

Some unwanted details are still remained in the output image, but it can be improved in the further model.



## B. Drag

### 1. Clone and Installation

```
git clone https://github.com/Zeqiang-Lai/DragGAN.git
cd DragGAN
conda create -n draggan python=3.7
conda activate draggan
pip install -r requirements.txt
```

### 1. Launch the server

```
if you have a Nvidia GPU
python gradio_app.py
if you use m1/m2 mac
python gradio_app.py --device mps
otherwise
python gradio_app.py --device cpu
```

## Result

Unfortunately, I am not able to drag the image in the system due to the lack of GPU. Eventhough I have tried to use the demo website provided by official develop team, it always gives me errors after running the model.

DraGAN is still in the development stage, it is not stable enough to use in the production environment. So, I will wait for a stable version to test this function.

## C. Draw

## References

1. [Original Paper](#)
2. [InternGPT Github](#)
3. [DraGAN Github](#)

**Reference:**

<https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/overview>

# Avocado Price

## Original Data & Challenge

- Resource: <https://www.kaggle.com/datasets/neuromusic/avocado-prices>

## Column Meanings

- Date - The date of the observation
- AveragePrice - the average price of a single avocado
- type - conventional or organic
- year - the year
- Region - the city or region of the observation
- Total Volume - Total number of avocados sold
- 4046 - Total number of avocados with PLU 4046 sold
- 4225 - Total number of avocados with PLU 4225 sold
- 4770 - Total number of avocados with PLU 4770 sold

## Inspiration // what we can do for this data set

1. In which cities can millenials have their avocado toast AND buy a home?
2. Was the Avocadopocalypse of 2017 real?

## Data Analysis

1. Use P-value
2. Use FFT to analyze pattern

## Modeling

### Part A. Model target

1. Find predicted price of avocado. If we can find the price of avocado, we can set a suitable price for avocado.
2. Find volume of avocado. If we can find the volume of avocado, we can decide how many avocados needed in our store.

### Part B. Solutions

1. Use current data to predict current data (easy)
2. Find potential problems
3. Use current data to predict one label in future (Medium)

4. Functional, but expecting bad result
5. Use time series prediction to find some outcomes (Hard)

## Part C. Time Series prediction

### C1. Pre-request

1. Convert discontinuous data into discrete data. Time series data should be sampled with a constant period
2. To handle missing data, we can use linear-equation to make the data more reasonable
3. Split data into train, val, test set. Look that the dataframe is continuous and discrete. We don't need random functions to make it bias.

```
train_df = data[0:int(n*0.7)]
val_df = data[int(n*0.7):int(n*0.9)]
test_df = data[int(n*0.9):]
```

4. Normalized the data. You can either use sklearn.preprocessing to normalized or input the formula by your self

```
train_df = (train_df - train_mean) / train_std
val_df = (val_df - train_mean) / train_std
test_df = (test_df - train_mean) / train_std
```

## Part C. Technical issues

### References

- [Kaggle Project link](#)
- [Explaination of p-value and t-test](#)
- [Time series forecasting with tensorflow](#)

### Interesting Discussion

- [Meaning of Data](#)
- [Prediction with Random Forest](#)



# **Abstract**

This is a testing session for chatGPT. The purpose of this dictorectories is to test accuracy and effieciency of using new AI technology to write codes

# **Problem 1**

How to detect basketball in a video using opencv?

To use OpenCV to detect basketballs in a video, you can follow these general steps:

Load the video: Use OpenCV's VideoCapture class to read the video frames.

Preprocess the frames: Convert the frames to grayscale, blur them to remove noise, and

Detect edges: Use an edge detection algorithm, such as Canny edge detection, to detect

Find contours: Use OpenCV's findContours function to identify the contours in the edge

Filter contours: Filter out any contours that don't have the expected size or shape of

Draw bounding boxes: Draw bounding boxes around the remaining contours to highlight th

Display the results: Display the processed frames with the detected basketballs.

Here's some sample code to get you started:

```
import cv2

Load the video
cap = cv2.VideoCapture('video.mp4')

Loop through the video frames
while cap.isOpened():

 # Read the frame
 ret, frame = cap.read()

 if ret:

 # Preprocess the frame
 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
 blurred = cv2.GaussianBlur(gray, (5, 5), 0)

 # Detect edges
 edges = cv2.Canny(blurred, 50, 150)

 # Find contours
 contours, hierarchy = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

 # Filter contours
 for contour in contours:
 if cv2.contourArea(contour) > 500 and cv2.contourArea(contour) < 5000:
 (x, y, w, h) = cv2.boundingRect(contour)
 if w / h > 0.7 and w / h < 1.3:
 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

```
Display the processed frame
cv2.imshow('Basketball Detection', frame)

Exit on key press
if cv2.waitKey(1) & 0xFF == ord('q'):
 break

else:
 break

Release the video capture and close the window
cap.release()
cv2.destroyAllWindows()

This code detects contours in each frame of the video, filters them based on area and
```

