Projektpraktikum Human Activity Understanding

Perfect Pancake

Group B: Student Presentation

Presenter: Siyi Dai, Shuang Wang, Zheng Tao, Chi Wang

Contents

- Motivation
- Project Introduction
- Story Line
- Data Collection Setup
- Pipeline Overview
  - → Object Detection
  - → Noun Detection
  - → Verb Detection
  - → Visualizer
- Conclusion
- Demo
- Futurework
- Reference

Motivation

- Human Activity Understanding is the area of recognizing and analyzing human activities
- The goal of this project is to learn and implement the key components required for a **vision-based** Human Activity Understanding pipeline.

Goal:

- Input: Images
- Output:
  - **Objects + Positions**
  - **Verb + Noun**

Figure 1: EPIC-KITCHENS Dataset

I. Damen, Dima, et al. "Rescaling egocentric vision: collection, pipeline and challenges for epic-kitchens-100."

Project Introduction

- **Ego-Perspective:**
  - Head mounted RGB-D camera (Realsense D435)
  - Ego-Perspective data
- **Problem Domain**
  - **Kitchen:** To make a "perfect" pancake 🥞 from zero

Figure 2: The "Perfect" Pancake

Story Line

- **Materials:**
  - Pan, honey in bottle, cranberries in box, blueberries in box, batter in bottle, Spatula and plate

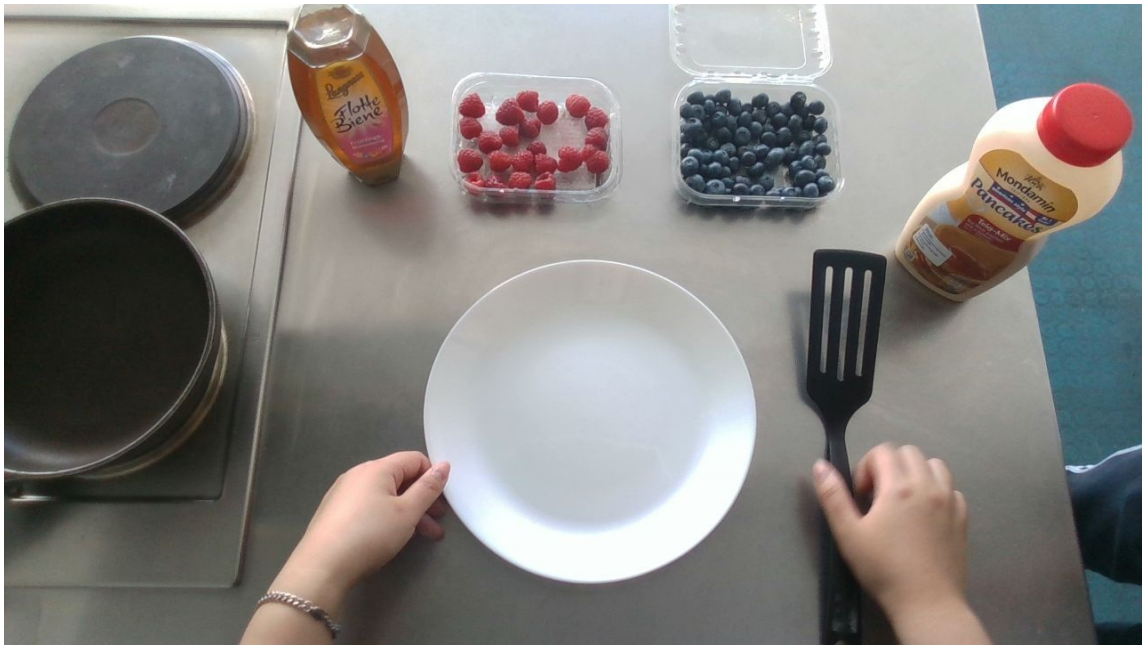


Figure 3: Materials for Making a Pancake

Story Line

- **Section 1: Pour the Batter into the Pan**

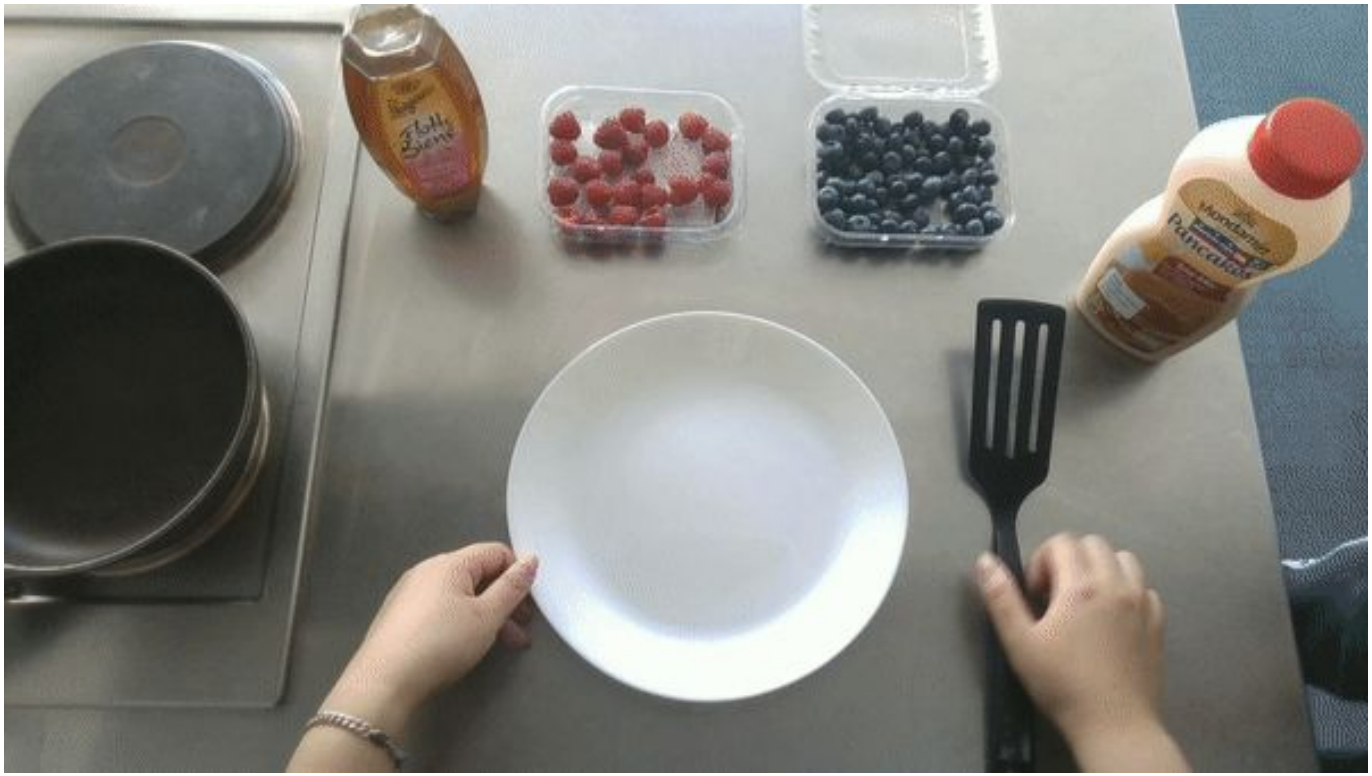


Figure 4: Perfect Pancake Story Line: Section 1

Story Line

- **Section 2: Waiting… (speed x5)**

Figure 5: Perfect Pancake Story Line: Section 2

Story Line

- **Section 3: Shovel and Flip**

Figure 6: Perfect Pancake Story Line: Section 3

Story Line

- **Section 4: Another waiting… (speed x5)**

Figure 7: Perfect Pancake Story Line: Section 4

Story Line

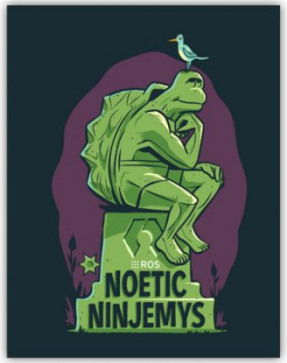
- **Section 5: Decorate** 🌟

Figure 8: Perfect Pancake Story Line: Section 5

Data Collection Setup

- **Ego Perspective:**
  - Helmet
  - Realsense D435
  - ROS

ROS



Logo of ROS Noetic Ninjemys

Figure 9: Data Collection Setup

Pipeline Overview

- Data **Collection** with Randomizations
  - **20** rosbags with **4** subjects, recorded **3** times
- Semantic Information **Extraction**
  - Rosbags → Images
- Recognize **Activities**
  - **Object** Detection → Shuang Wang
  - **Noun** Detection → Zheng Tao
  - **Verb** Detection → Siyi Dai
- Update Models
  - **YOLO, Efficient Net, LSTM**
- Visualization
  - **Visualizer** → Chi Wang

Object Detection
Group B

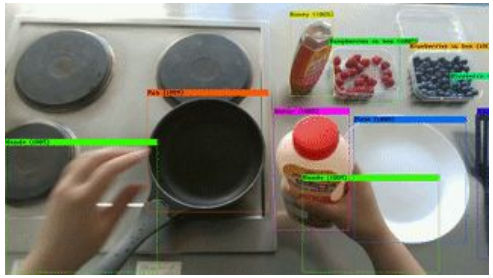
Shuang Wang

Motivation: Why YOLO?

- **Goal:**
  - Object Detection in Video/Real-time → **Environment Understanding**
- **Requirement:**
  - Detection Speed
  - Accuracy of Object Name Detection
  - Accuracy of Object Position Detection

Original Video

YOLO



Mask R-CNN

FPS	30	≈9.5	≈1.25
Detection	-	Box bounding objects	Segmented instances

Introduction: How YOLO works?

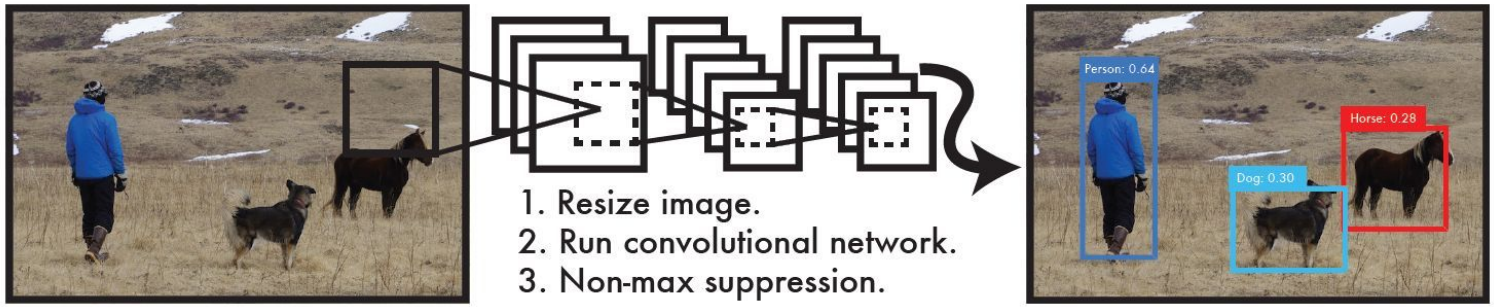


Figure 10: YOLO workflow

Advantages:

1) Give b**ox bounding every detected objects** on the frame (bbox)
2) Relatively **high detection efficiency**
3) **End-to-end** train and test

Disadvantages:

1) High hardware requirement and long training time
2) Sometimes the information of bbox are not quite precise for some potential further processing tasks

Introduction: Model I&O

Model input: Image(s)

Model output:
1) Center of bbox(x,y)
2) Width and height of bbox(w,h)
3) Probability of the corresponding class of an object and the corresponding confidence score

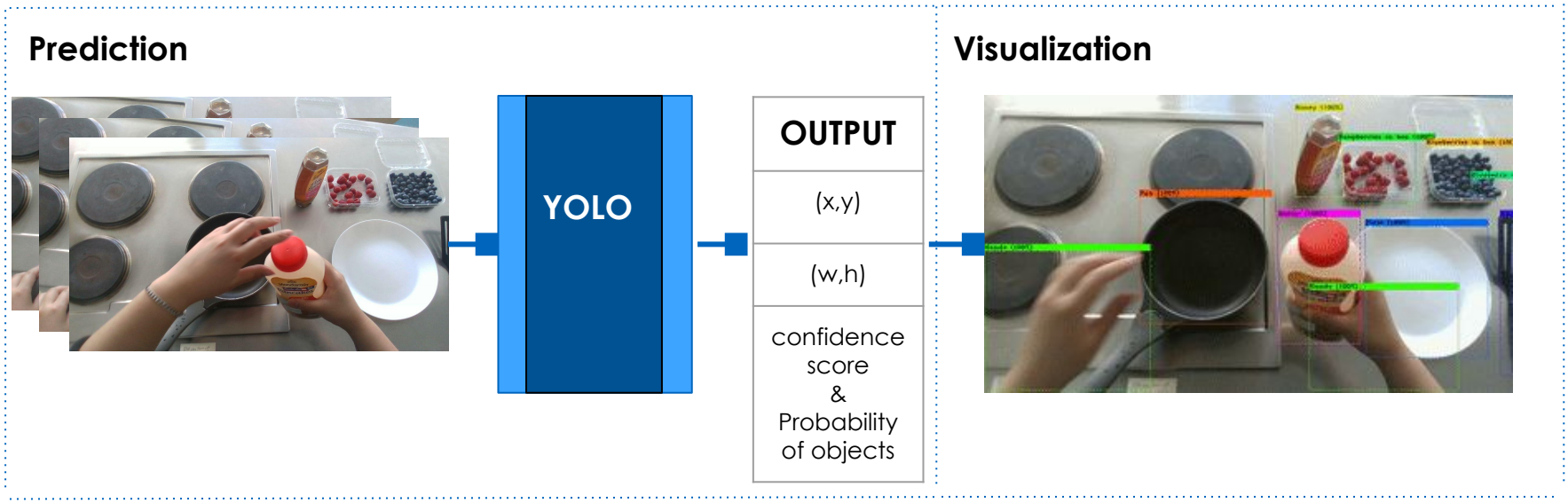


Figure 11: Model I&O

Tasks

Annotation

- CVAT
- MiVOS

Training Configuration

Training Phase

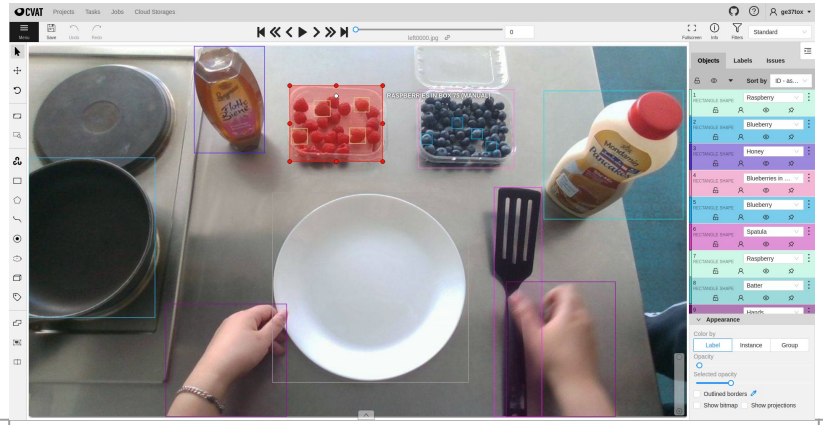
Test Phase

- Pancake Status Transformation
- Visualization & Output

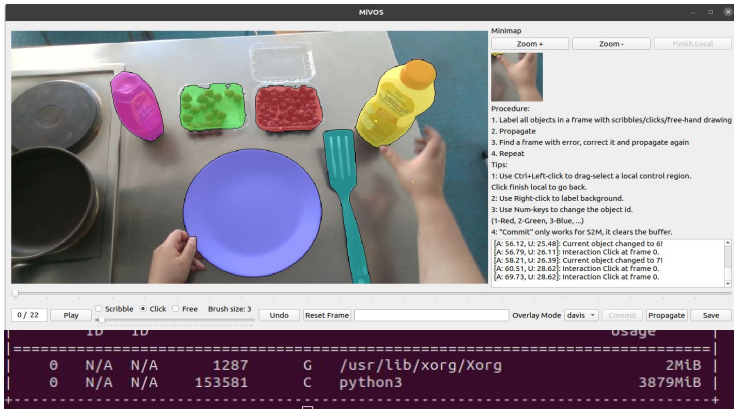
Annotation

Annotation Tool Comparison

CVAT (☺)



MiVOS



+	❑ Annotation by hand (precise) ❑ Independent to temporal relations ❑ No requirement on hardware	❑ Efficiency
-	❑ Efficiency	❑ Dependent on temporal relations ❑ Annotation with DNN model's help ❑ High requirement on hardware

Annotation

- **Object Classes**

1. Batter
2. Pancake(0% finished)
3. Pancake(50% finished)
4. Pancake(100% finished)
5. Pancake(80% finished)
6. Honey
7. Spatula
8. Blueberry
9. Blueberries in box
10. Raspberry
11. Raspberries in box
12. Pan
13. Plate
14. Hands

Pancake Status Transformation

1. Bubbles popping up (0% →50 %)

2. Flipping(50% → 80%)

Training Configuration

- **Input size**
  - Input image size
- **Batch size**
  - Updating Parameters after batch_size samples
- **Subdivision**
  - Subdivide batch into several small batches
- **Max_batches**
  - Classes*2000
- **Steps**
  - 80% to 90% of max_batches
- **Filters** in the last convolutional layer before each yolo layer:
  - (num_classes+5)*3
- **Number of classes** in each yolo layer

Training Phase

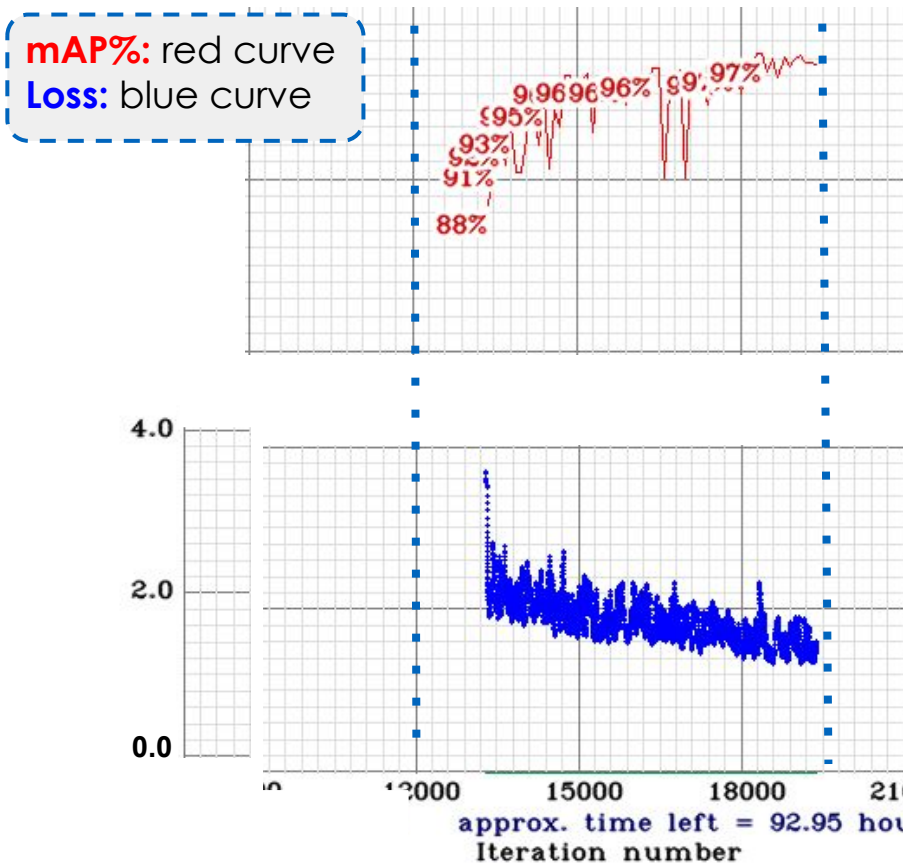


Figure 12: mAP% and training loss

- **Data Recording (Ego Perspective)**
  - **20** rosbags from **4** subjects;
  - Over **6000** frames in each **rosbag**
- **Annotation (Training set)**
  - **Downsampled** frames from **4 rosbags** recorded by **2 subjects** (130 frames in total)
- **Test**
  - Videos from **3 randomly selected** rosbag

Device for training:

Geforce GTX 960M 4GB

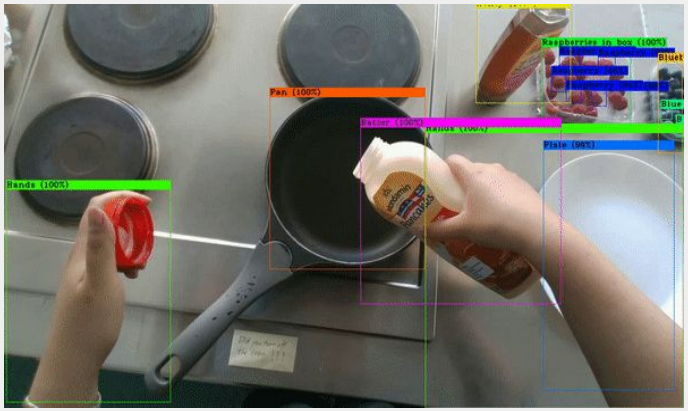
Training time:

about a week for 20000 iterations

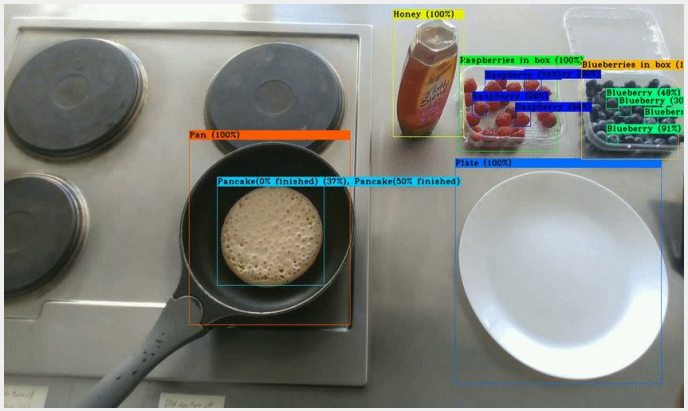
Test Phase

Pancake Status Transformation

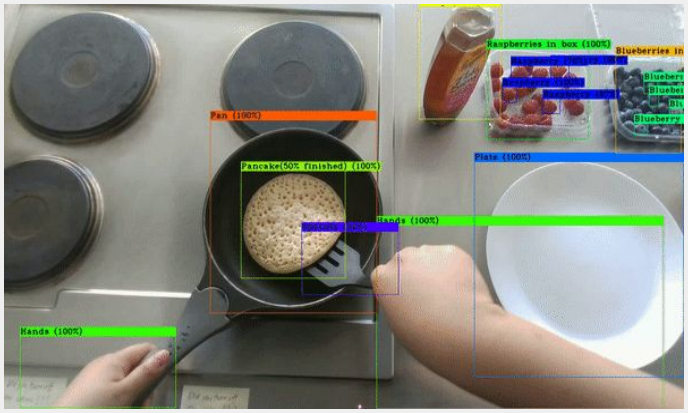
1. Pour the batter



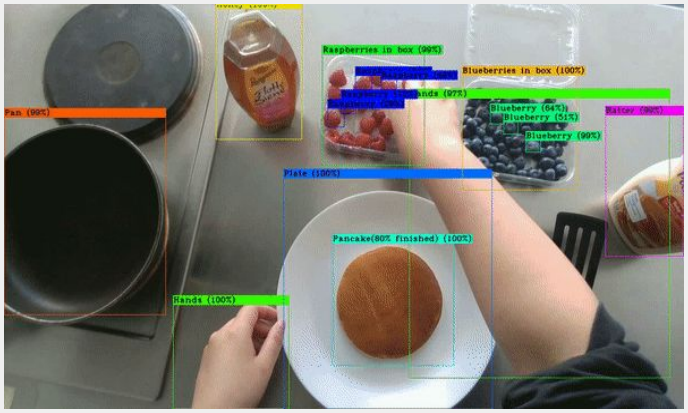
2. Bubbles popping up (0% →50 %)



3. Flipping (50% → 80%)



4. Decoration (80% → 100%)



Label Colors:
0% finished
50% finished
80% finished
100% finished

Test Phase

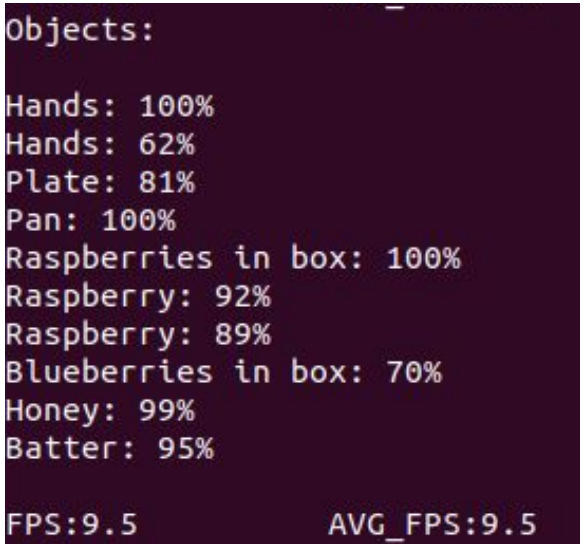


Figure 13: Video on test sequence (By **AlexeyAB**, **left**) and objects, confidence score and fps (**right**)

Output in **python** for further task (**Noun Detection**):

- Format: **dict** → { "**object_name**":
  {
  "**bbox**": bbox,
  "**confidence**": confidence score,
  }
  }

Noun Detection
Group B

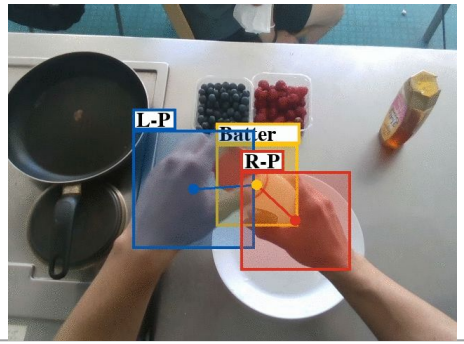
Zheng Tao

Motivation

- **Known:** Objects + Positions
- **Goal:** Find which object interacting with hands in video/real time
- **Requirement:**
  - Localization speed
  - Accuracy of noun (object name) detection

Input

Output



FPS	15	5
Noun Detection	-	Both hands BBoxes Noun label and BBoxes

Tasks

Hand Object Interactive Algorithm

- Hand Object Detector

Noun Detector with YOLO

- YOLO

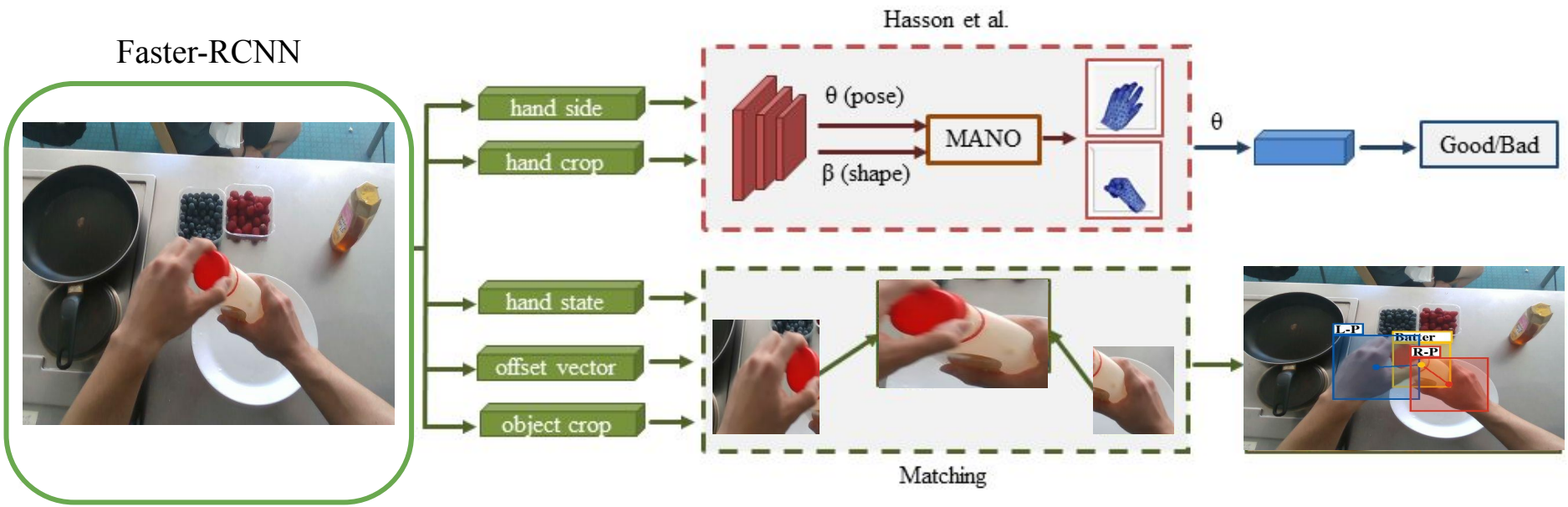
Noun Detector Further Exploration

- Dataset Preparation
- Dataset Generation
- YOLO + VGG
- Efficient Net

Result

- Model comparison
- Noun detection Display

Hand Object interactive Algorithm



Model Input

- Images

Model Output

- Noun BBox
- Both Hands BBox

Figure 14: Hand Object Detector

Noun Detector with YOLO

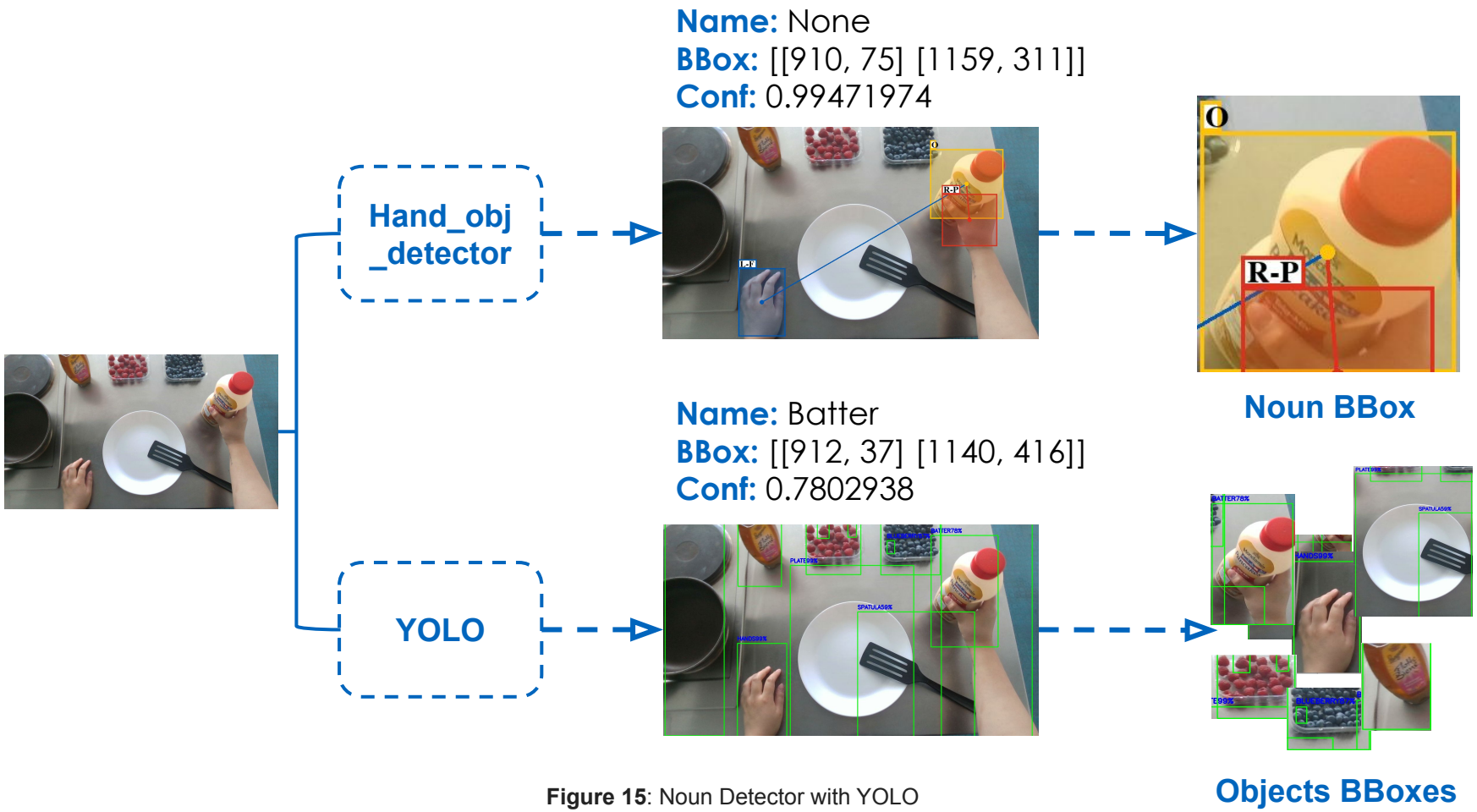


Figure 15: Noun Detector with YOLO

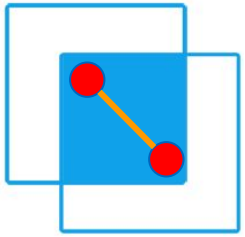
Noun Detector with YOLO

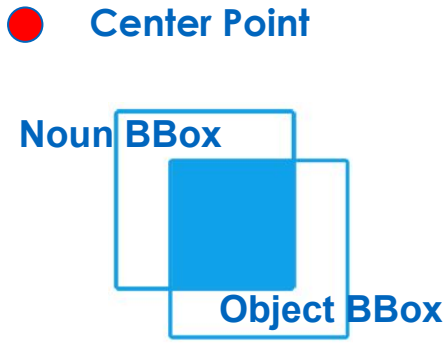
➔ Get the name of Noun

- **IoU**: calculate the **IoU** between **Noun BBox** and **Object BBoxes**
  - **IoU threshold: 0.5**

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

- **Distance**: calculate the **distances** of **Center points** of BBoxes
  - **Distance threshold: 150**



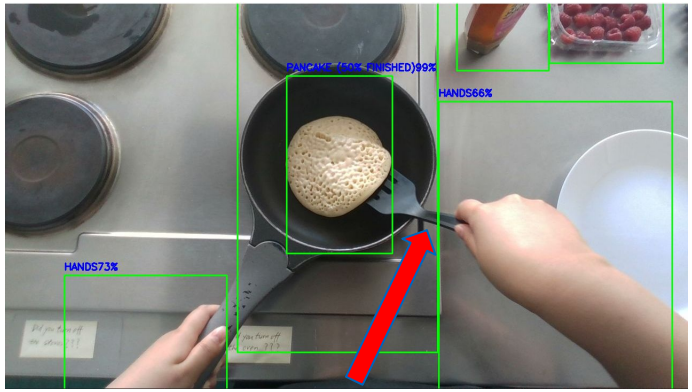


Noun Detector with YOLO

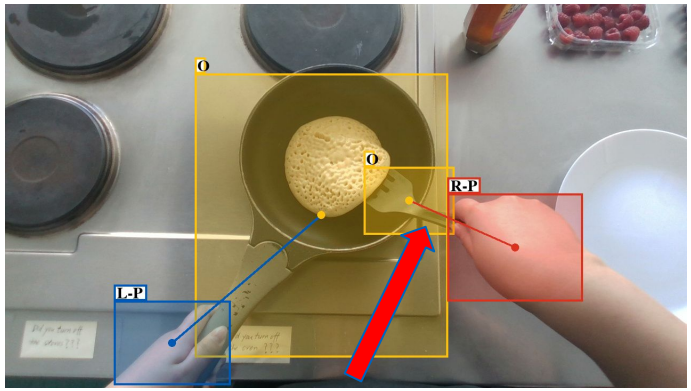
PROS	CONS
• Nearly real time processing • Output Bounding Boxes • High accuracy in Static object detection	• Bad performance in dynamic object detection, like "Spatula"

➔ **Problem**: some object that Faster RCNN can detect but YOLO cannot

➔ **YOLO**



➔ **Faster RCNN**



Noun Detector Further Exploration

- **Dataset Preparation**
  - **Data Extraction**
    - **4** rosbags, **4** subjects in **Ego perspective**
    - Republishing compressed images
  - **Data Preprocessing**
    - Getting noun BBoxes by running Faster RCNN
    - Cropping image according to these BBoxes
      - nearly: 2500 images per rosbag
    - image saved as *.jpg

Noun Detector Further Exploration

- **Dataset Generation**

Label cropped images into 12 classes:

1. Batter
2. Batter cap
3. Blueberry
4. Honey
5. Raspberry
6. Spatula
7. Spatula with Pancake (100% finished)
8. Spatula with Pancake (80% finished)
9. Spatula with Pancake (50% finished)
10. Pancake (100% finished)
11. Pan with Pancake (0% finished)

➔ Split all these images into **training dataset** and **testing dataset**

◆ **Ratio: 7 vs 3**

Noun Detector Further Exploration

- **YOLO + VGG**
  - ➢ VGG helps to classify the undetected object in YOLO

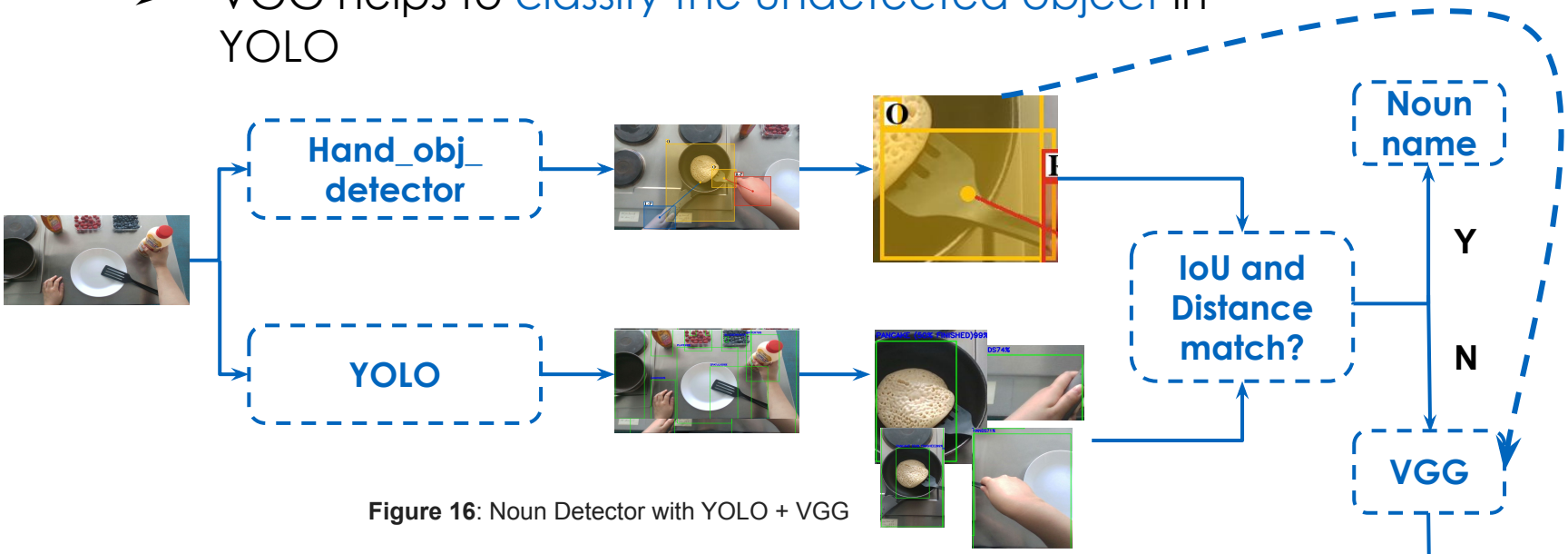


Figure 16: Noun Detector with YOLO + VGG

➔ Problem: Long processing time

YOLO (FPS)	0.70901
VGG (FPS)	0.87656

Noun Detector Further Exploration

- **Efficient Net**
  - Hand_obj_detector + VGG ≈ YOLO
    - Get rid of YOLO
    - Replace VGG with Efficient Net

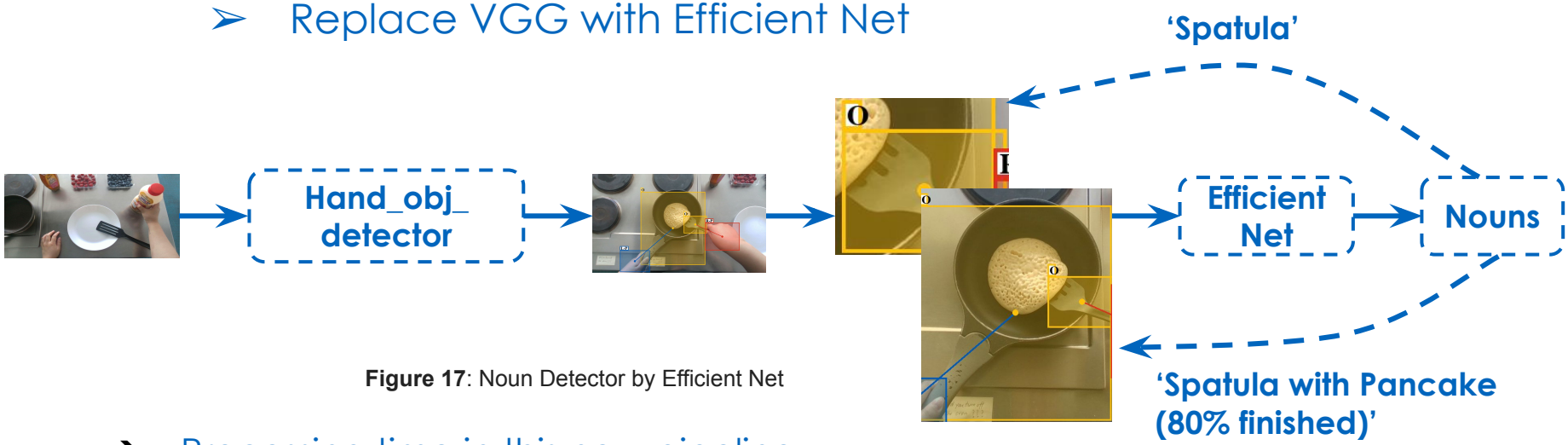


Figure 17: Noun Detector by Efficient Net

➔ Processing time in this new pipeline

Hand_obj_detector	0.371621
Efficient (FPS)	0.302238

Result

- **Model Comparison**
  - **YOLO**

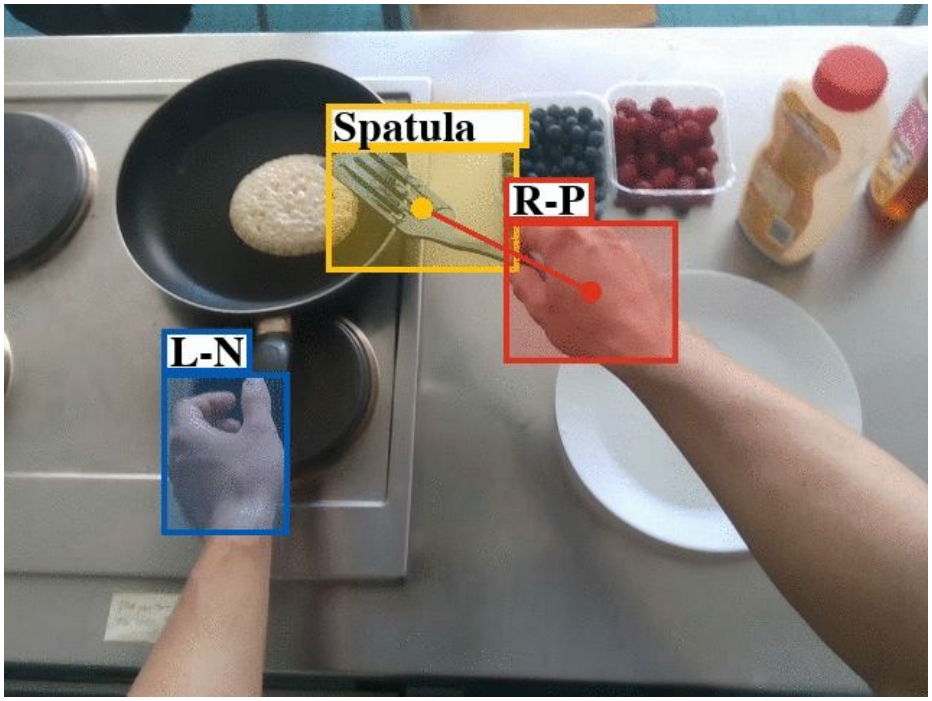
    ❌ Missing target object
  - **YOLO + VGG**

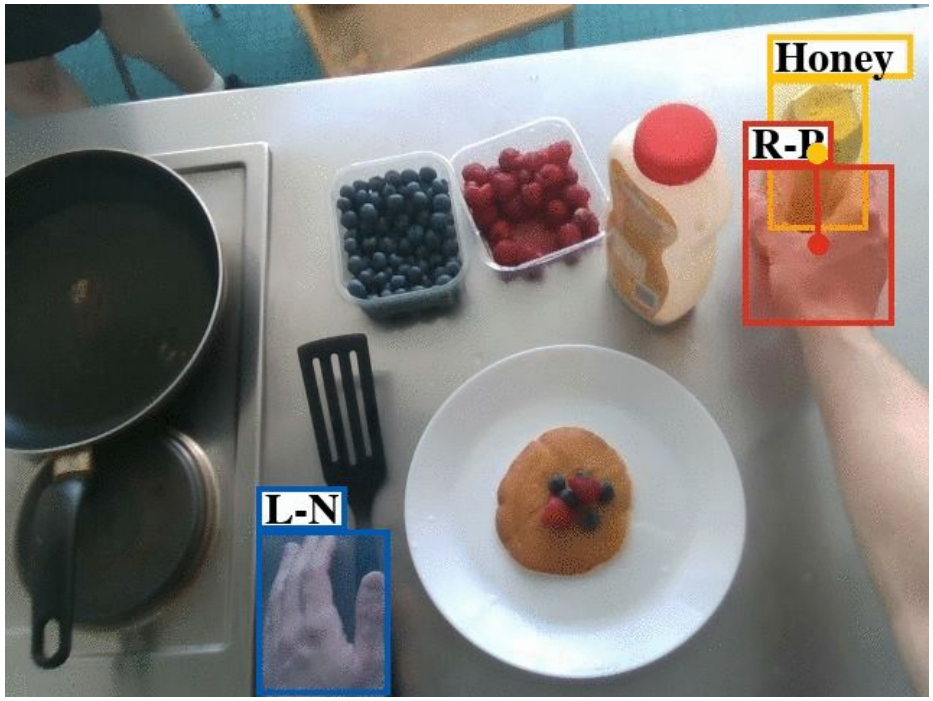
    ❌ Processing time too long
  - **Efficient Net**

    ✅ 0.6716 FPS

Result

- **Noun detection display**





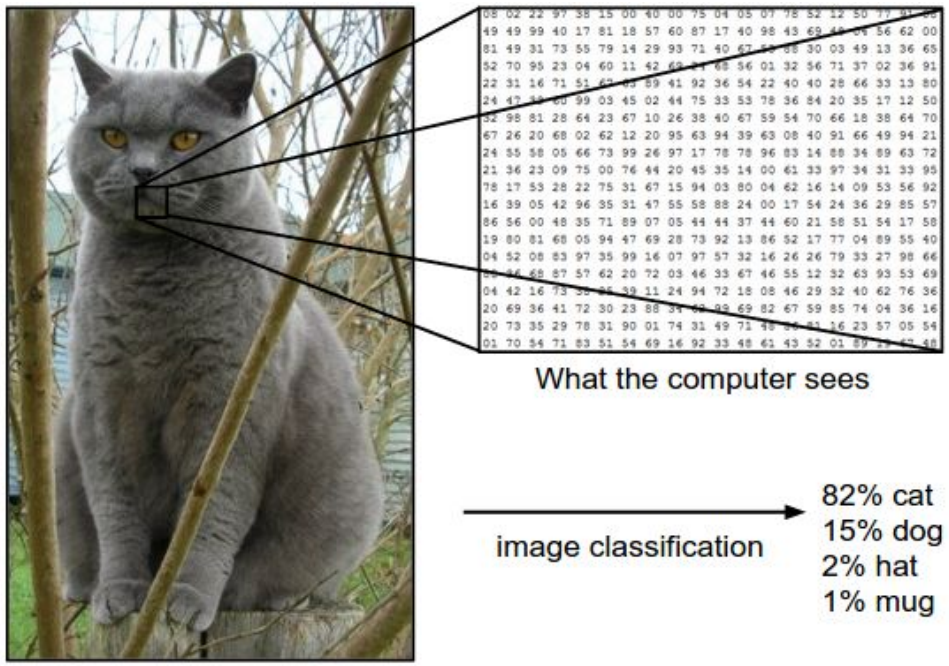
Verb Detection
Group B

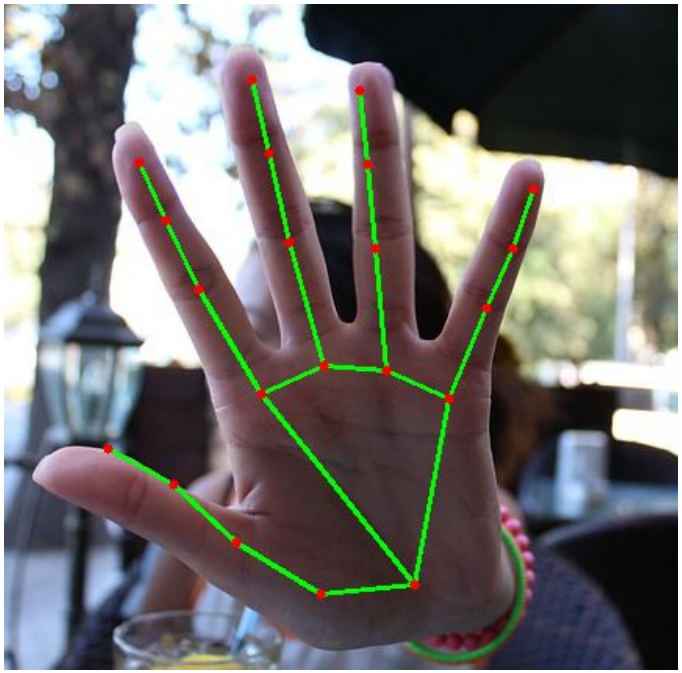
Siyi Dai

Motivation

- **Known:**
  - Objects + Positions
  - Noun (object interacting with hands)
- **Goal:**
  - Detect **Verb**
- **Requirement:**
  - **Accuracy** of verb detection
  - **Runtime speed** for real-time detection

Motivation





Classic Image Classification:
256x256x3 or even more!

Hand Landmarks Classification:
only **21** points per hand!

I. Lu, Dengsheng, and Qihao Weng. "A survey of image classification methods and techniques for improving classification performance."
II. Zhang, Fan, et al. "Mediapipe hands: On-device real-time hand tracking."

Tasks

Dataset Preparation

- Data Recording and Extraction
- Dataset Generation

LSTM Model

- Data Pre-process
- Data Loader
- Model Building
- Model Training

Results Testing

- Verb Pipeline

Dataset Preparation

- **Data Recording and Extraction**
  - Data **Recording**
    - **8** rosbags, **4** subjects, **Ego** perspective
    - Duration: **3-4mins**
      - **5** rosbags, **3** subjects → **training**
      - **3** rosbags, **3** subjects → **testing**
  - Data **Extraction**
    - Republish compressed images
    - Image saver
    - save in *.jpg

Dataset Preparation

- **Dataset Generation**
  - **Label** the **images** from each bags into **12** classes
    - "close": 0,
    - "decorate": 1,
    - "flip": 2,
    - "move_object": 3,
    - "open": 4,
    - "pick_up": 5,
    - "pour": 6,
    - "put_down": 7,
    - "screw": 8,
    - "shovel": 9,
    - "squeeze": 10,
    - "other": 11

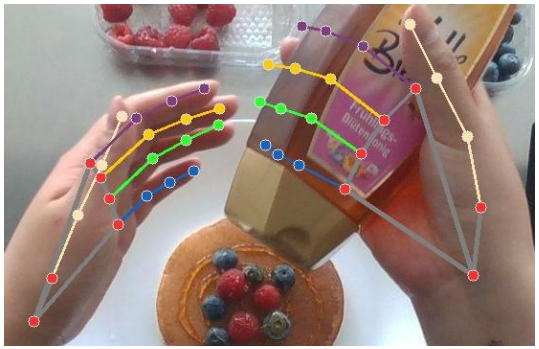
Consistency in Criteria →
Label all of the images:
37805 images

Dataset Preparation

- **Dataset Generation Tool**
  - **Folder** Structure for each recording/rosbag
    - sy0
    - ├── close
    - │ └── sy0_0
    - │ ├── images
    - │ └── close_sy0_0.csv

    ➡

    - Level 0: Recording name
    - Level 1: Verb name
    - Level 2: Recording_Sequence
    - Level 3: Image
  - Generate hand landmarks with **Mediapipe**
    - Detect 21 points per hand
    - landmark_pb2.NormalizedLandmarkList
    - Output as DataFrame
    - Save in .json/**.csv**



LSTM Model

- **Data Pre-process**
  - Drawbacks of Mediapipe: **no detection, no output**
  - **High** requirement in recordings! → we re-recorded for 3 times.

Figure 18: The "Perfectly Wrong" Example (speed x 0.5)

- Proper move speed
- Steady perspective
- Show complete hand
- No other subjects
- (show complete objects)

LSTM Model

- **Data Pre-process**
  - Drawbacks of mediapipe: **no detection, no output**
    - **Complete features:**
      - **zero-padding** when there's only one hand
    - **Continuous input:**
      - **zero-padding** when there's no detection at all
  - The number of features: **126 = 21 points x (x, y, z) x (left, right)**

Figure 19: The No Detection Examples

LSTM Model

- **Data Loader**
  - Temporal sequence → LSTM
    - Proper time steps and a stride for sliding window

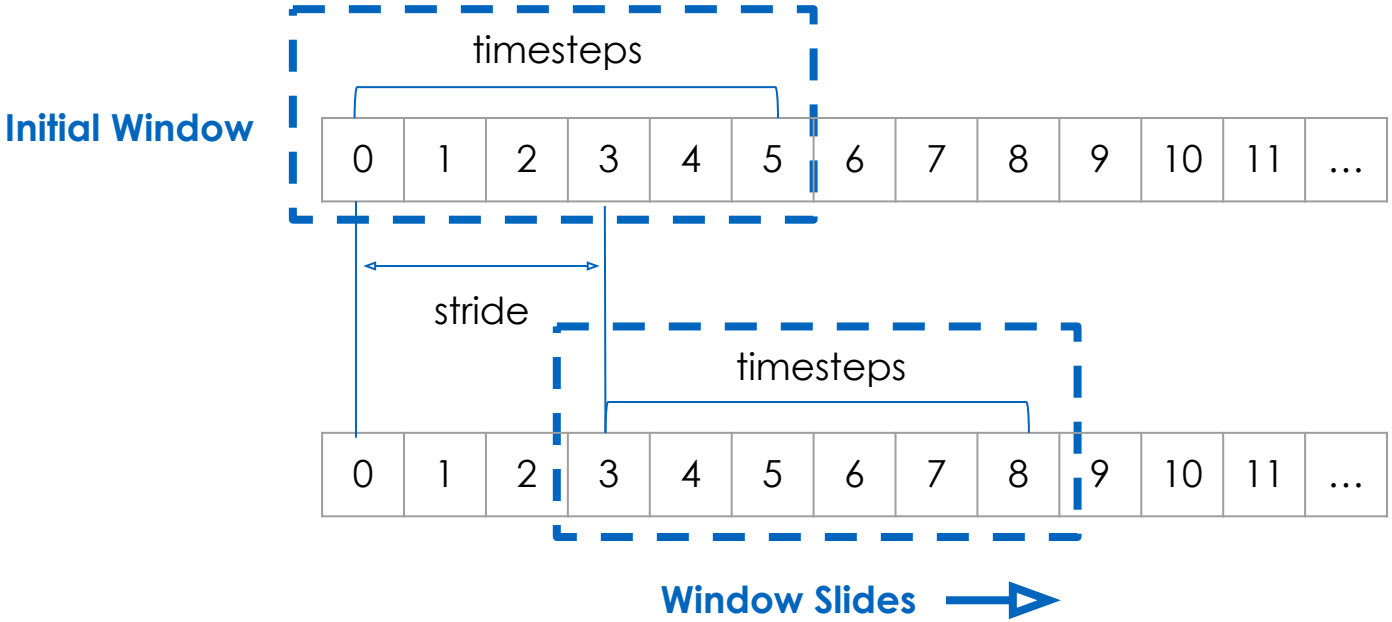


Figure 20: Sliding window demo

I. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory."

LSTM Model

Runtime =

Preprocess time + Predict time

- **Data Loader Parameters Tuning**

(Stride, Time_steps)	Accuracy ↑	Macro Avg f1-score ↑	Number of Classes Accuracy >= 0.85 ↑	Runtime (s) ↓	
				Mean ↓	Std ↓
(10, 100)	0.85	0.68	2/12	7.531	0.025
(5, 50)	0.78	0.60	1/12	3.981	0.014
(5, 20)	0.74	0.47	1/12	1.830	0.013
(1, 20)	**0.98**	0.93	11/12	1.837	0.020
(1, 10)	**0.98**	**0.94**	**12/12**	0.785	0.009
(1, 5)	0.97	0.89	8/12	**0.524**	**0.006**

Table 1: Stride and Timesteps Combination Comparison under 2 layers of LSTM with 128 units and 64 units. Runtime is calculated by 500 iterations with Google Colab GPU accelerator

LSTM Model

- **Model Building**

Model	Accuracy ↑	Macro Avg f1-score ↑	Number of Classes Accuracy >= 0.85 ↑	Inference Time (ms) ↓	
				Mean ↓	Std ↓
1 x LSTM, 64	0.96	0.88	8/12	4.837	0.752
1 x LSTM, 128	0.97	0.91	11/12	**4.180**	**0.341**
Encoder with 2 x LSTM, 128, 64	**0.98**	**0.94**	**12/12**	4.922	0.489
Autoencoder with 2 x LSTM, 128, 64 2 x LSTM, 64, 128	0.97	0.90	9/12	6.323	0.552

Table 2: Model Comparison with stride = 1 and timesteps = 10,
Inference time is calculated by 1000 iterations with Google Colab GPU accelerator

LSTM Model

- **Model Building**

Model: "**LSTM_128**"

Layer (type)	Output Shape	Param #
0_LSTM (LSTM)	(None, 10, **128**)	130560
1_Dropout (Dropout)	(None, 10, 128)	0
2_Flatten (Flatten)	(None, 1280)	0
3_Dense (Dense)	(None, 128)	163968
4_Dense (Dense)	(None, **12**)	1548

Total params: 296,076
Trainable params: 296,076
Non-trainable params: 0

Table 3: Model Summary

I. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory."

LSTM Model - **Classification Report**

	precision	recall	f1-score	support
close	0.91	0.87	0.89	60
decorate	0.96	0.95	0.96	288
flip	0.87	0.89	0.88	80
move_object	0.96	0.84	0.90	76
open	0.82	0.97	0.89	80
pick_up	0.94	0.94	0.94	431
pour	0.98	0.95	0.96	165
put_down	0.89	0.85	0.87	209
screw	0.96	0.95	0.95	333
shovel	0.87	0.77	0.81	209
squeeze	0.91	0.96	0.93	171
other	0.99	0.99	0.99	5423
accuracy			**0.97**	7525
macro avg	0.92	0.91	0.91	7525
weighted avg	0.97	0.97	0.97	7525

Table 4: Classification Report

```
"patience": 8,
"validation_split": 0.3,
"epochs": 100,
"batch_size": 32,
```

Results Testing - Verb pipeline

- **Data Pre-process**
  - Zero-paddings → **Content-fillings**
    - 6000 frames → small batch
      - What if the timestamp gap is not in the middle, but in the beginning or the end?
      - What if in this batch, there is no detection at all?
      - What if …?
  - The number of features: **126 = 21 points x (x, y, z) x (left, right)**
  - Data Window: **2D → 3D**
  - **Timestamp** Check
- **Visualization** for Hand Landmarks
  - Debug and visualizer
- Model **Load** and **Predict**

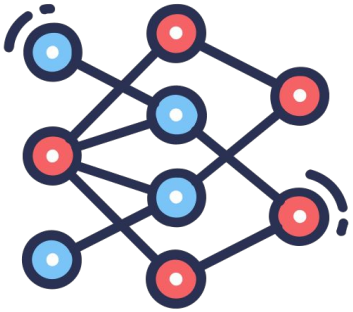
Results Testing - Verb pipeline

- **Runtime Optimization**
  - Load model while predict → integrated pipeline **2788.24ms**
  - Load model before predict → much faster
    - predict → 37.22ms (Predict time)
    - **predict_on_batch → 4.18 ms** (Predict time)
  - Load labels before predict
- **Runtime = Preprocess time + Predict time**
  - → mean = **804.66ms**, std = 8.31ms
    - calculated by 500 iterations with Google Colab GPU accelerator
  - **Original: 2788.24ms → Final: 804.66ms**
- **Runtime reduction: 71% ↓**

Results Testing - Verb pipeline

Original Image Flow → Image Batch = 10 → Mediapipe Process →

```
0_0_x,0_0_y,0_0_z,0_1_x,0_1_y,0_1_z,0_2_x,0_2_y,0_2_z
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.7134225368499756,0.5003337860107422,1.726602505414
0.712389349937439,0.49606287479400635,1.024372409119
0.7128814458847046,0.5019252896308899,2.348308356658
0.3508577048778534,0.575818657875061,3.245197035396
0.7106701135635376,0.5093257427215576,3.208752445971
0.7099168300628662,0.5119991302490234,3.026243575732
0.7075582146644592,0.5113846063613892,3.204249026111
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.7047097682952881,0.5115142464637756,3.009581348578
0.7087118625640869,0.5082047581672668,2.643965153343
```



```
The current action is: close
```

→ Fill in Content → Model Predict → Verb Detection

Figure 20: Verb Pipeline Workflow

Visualizer
Group B

Chi Wang

Tasks

Real Time Visualizer (Based on Intel Visualizer)

- Input: ROS bag or realsense camera.
- Outputs: 3D view, object detection results, hand landmarks, verb + noun phrase.

Merge All Components

- YOLO, noun detection, verb detection.
- Pytorch, Tensorflow, Mediapipe, etc…

Solve Package Version Conflicts

- CUDA Toolkit, pytorch and tensorflow version conflicts.

I. https://github.com/IntelRealSense/librealsense/blob/81d469db173dd682d3bada9bd7c7570db0f7cf76/wrappers/python/examples/opencv_pointcloud_viewer.py

Workflow

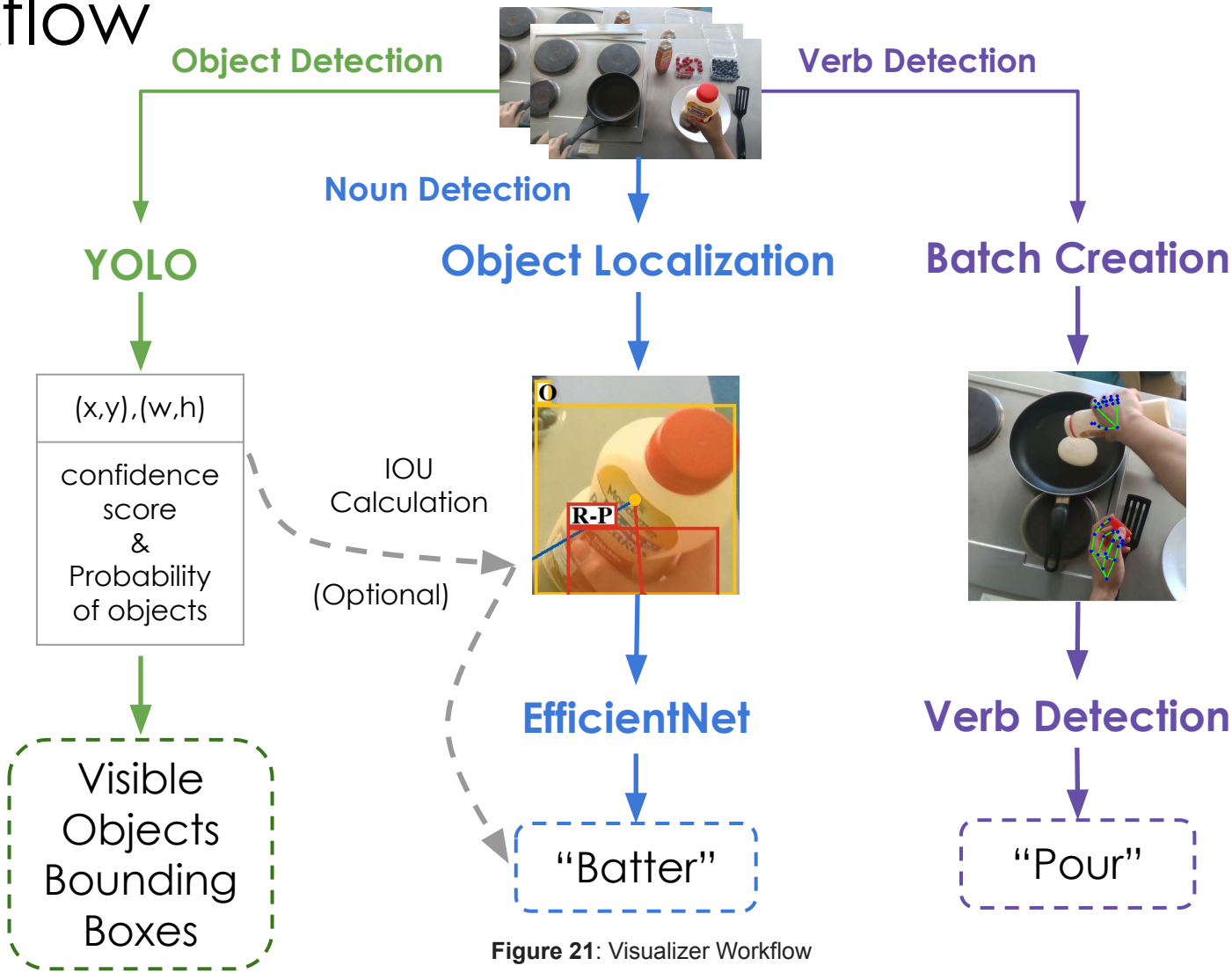


Figure 21: Visualizer Workflow

Read Rosbag

Ros.bag → No depth and camera info → Ros_new.bag → Need to rewrite subscriber nodes

Ros.bag --Topics--> Color_compressed

Ros_new.bag --Topics--> Color_compressed, Depth_compressed, Depth_camera_info

Before

```
self.color_subscriber = message_filters.Subscriber(color_topic, Image)
self.depth_subscriber = message_filters.Subscriber(depth_topic, Image)
self.depth_camera_info_subscriber = message_filters.Subscriber(depth_camera_info_topic, CameraInfo)
```

After

```
self.color_subscriber = message_filters.Subscriber(color_topic, CompressedImage)
self.depth_subscriber = message_filters.Subscriber(depth_topic, Image_new)
self.depth_camera_info_subscriber = message_filters.Subscriber(depth_camera_info_topic, CameraInfo)
```

Invalid Phrases

Verbs

- Decorate
- Close
- Flip
- move_object
- squeeze
- …

Nouns

- Batter
- Batter cap
- Blueberry
- Honey
- Raspberry
- Spatula
- Spatula with Pancake (100% finished)
- Spatula with Pancake (80% finished)
- Spatula with Pancake (50% finished)
- Pancake (100% finished)
- Pan with Pancake (0% finished)

There are 12 verbs and 11 nouns in our projects
Can we just simply combine them to get a phrase?

Invalid Phrases

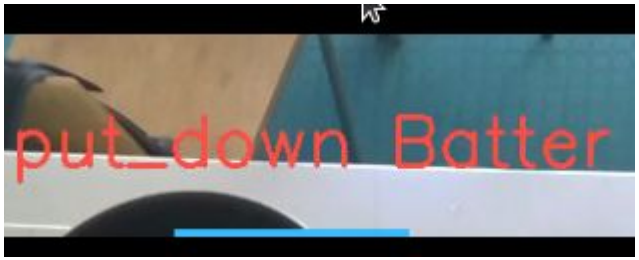


Figure 22: Valid and invalid phrases

Invalid Phrases

- **Define Valid Combinations**

Close
Open

Batter
Batter cap
Blueberry
Honey
Raspberry

squeeze

Batter
Honey

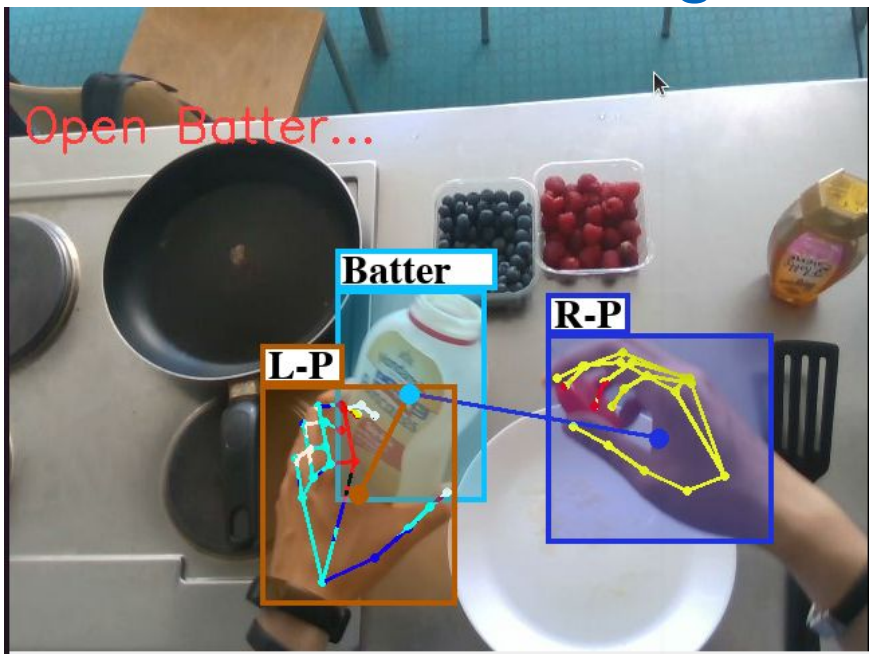
Flip

Spatula
Spatula with Pancake (100% finished)
Spatula with Pancake (80% finished)
Spatula with Pancake (50% finished)
Pancake (100% finished)
Pan with Pancake (0% finished)

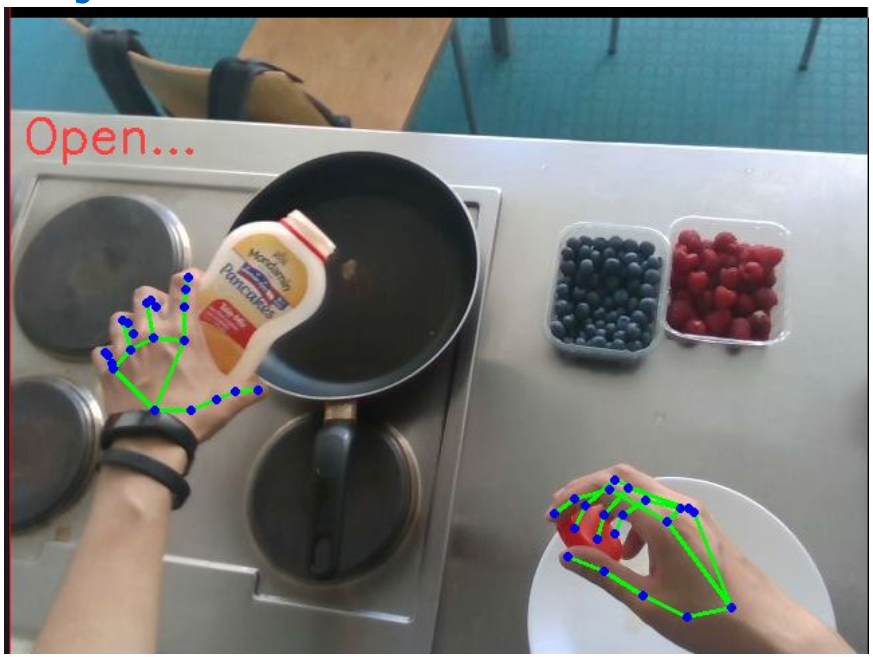
…

Miss-detection of Objects

- **Problem of missing hand object**



Frame 10
(with hand objects)



Frame 18
(no hand objects detected)

Figure 23: Illustration of missing hand objects

Miss-detection of Objects

- **Definition of Compensation**

Frame	0	1	2	3	4	5	6
Nouns	Honey	Batter	Batter	Batter	None	Batter	None

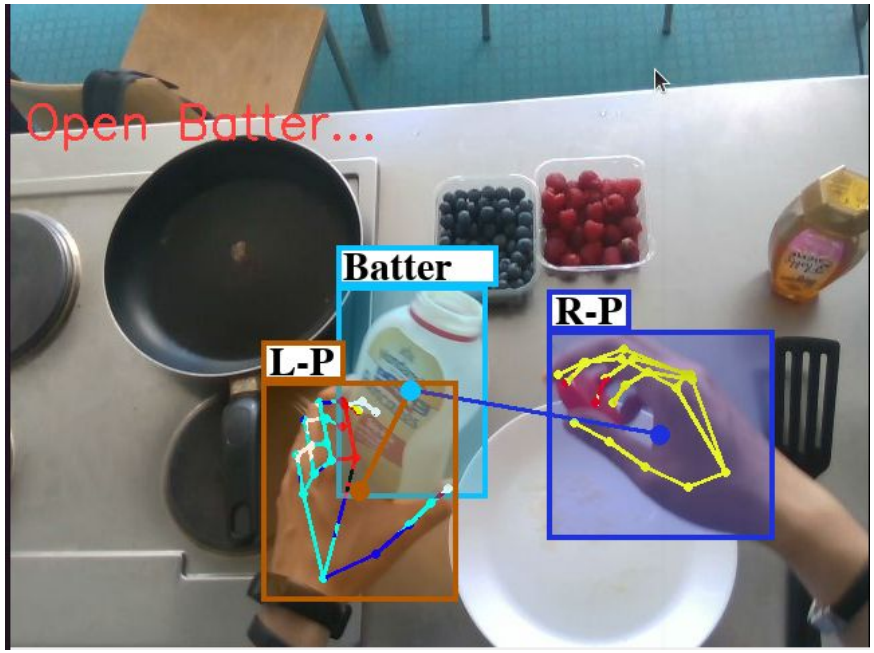
Sampling from previous k frames

Frame	0	1	2	3	4	5	6
Nouns	Honey	Batter	Batter	Batter	Batter	Batter	None

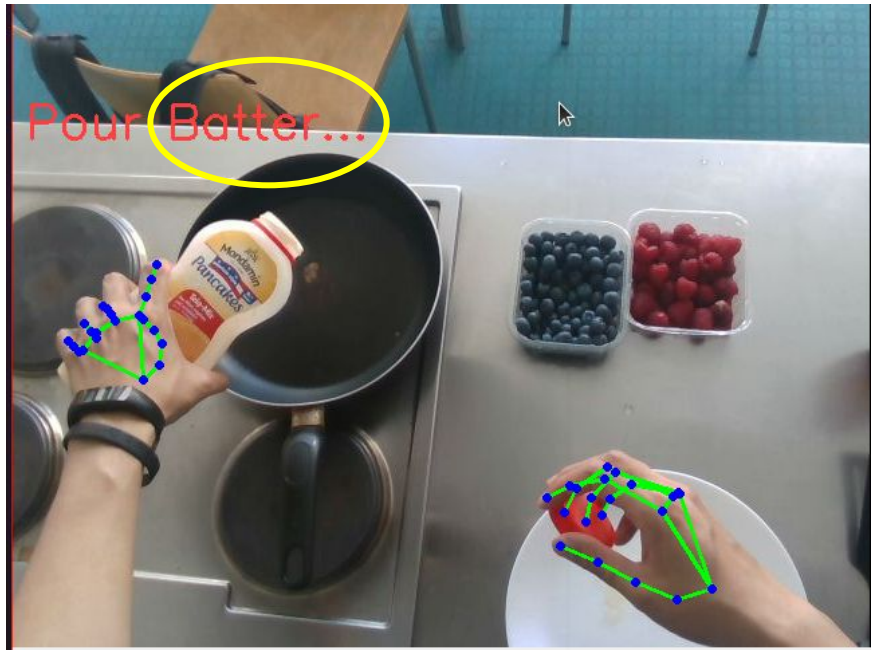
Sampling from previous k frames

Miss-detection of Objects

- **Compensate the frame**



Frame 10



Frame 18

Figure 24: Illustration after the compensation

Library Version Conflicts

Components	CUDA Toolkit	Pytorch	Tensorflow	CUDNN for Tensorflow
YOLO	-	-	-	-
Noun Detection	**10.0.130**	1.12.0	2.6.2	**7.3**
Verb Detection	11.3	-	2.6.2	8.1.0

Table 5: Version Conflicts between Modules

This module is not compatible with others

This issue is not resolved, so we run all the module in CPU

Processing Time Optimization

- **Processing Time for One Frame in CPU**

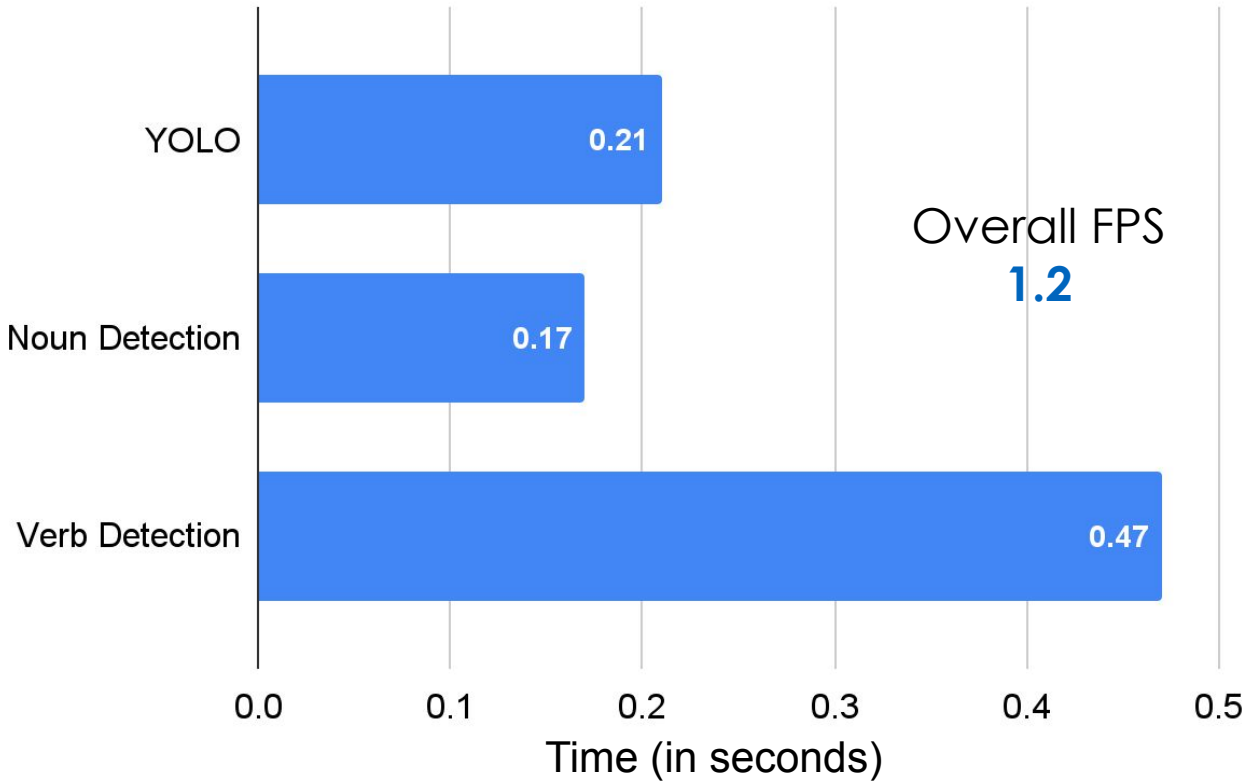


Figure 25: Processing time for one frame of each component in CPU

Modules Visualization

- **Object Detection**

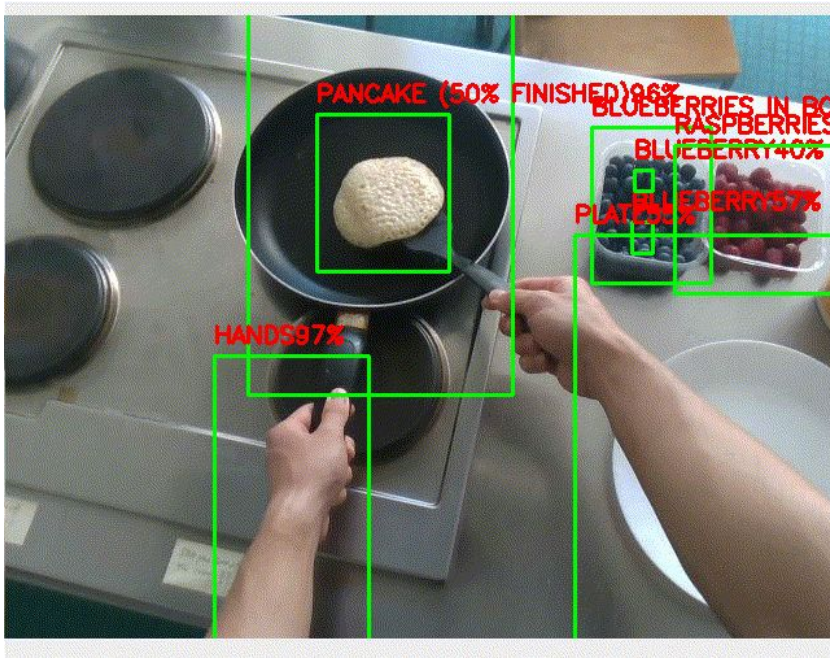
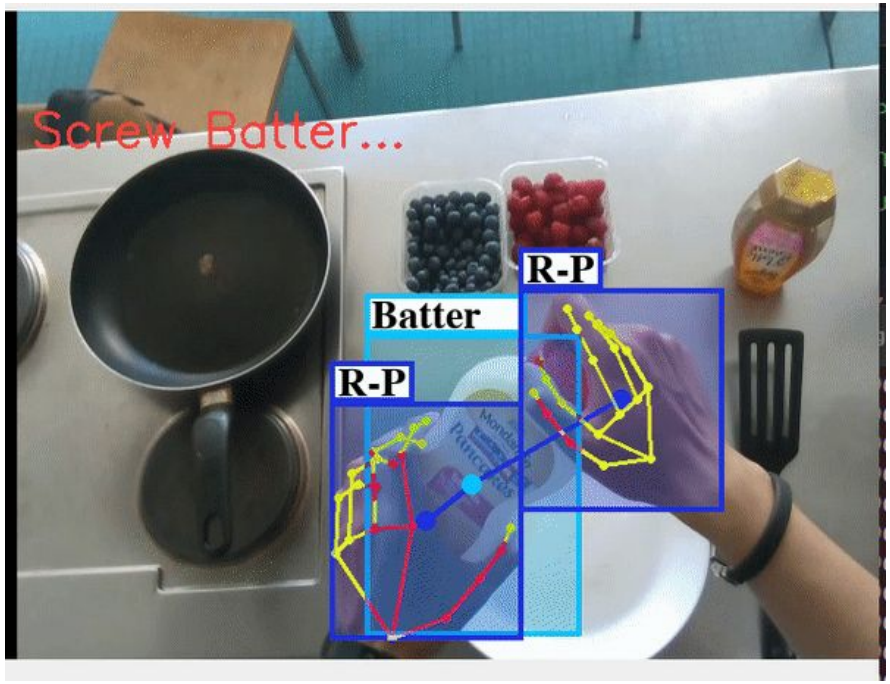


Figure 26: Visualization for yolo

Modules Visualization

- **Verb + Noun Detection**

Pick and open batter



Pour batter

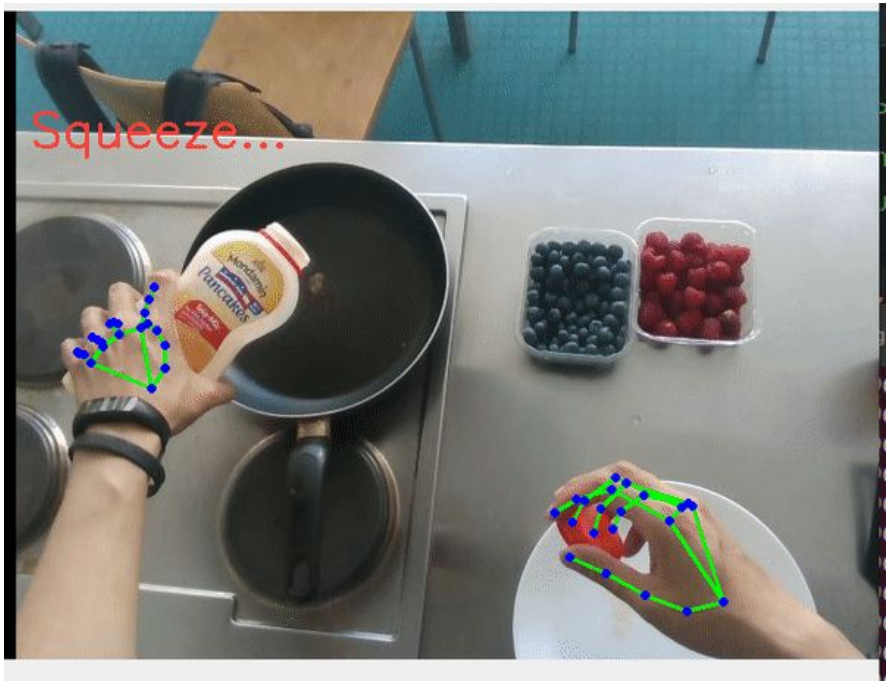
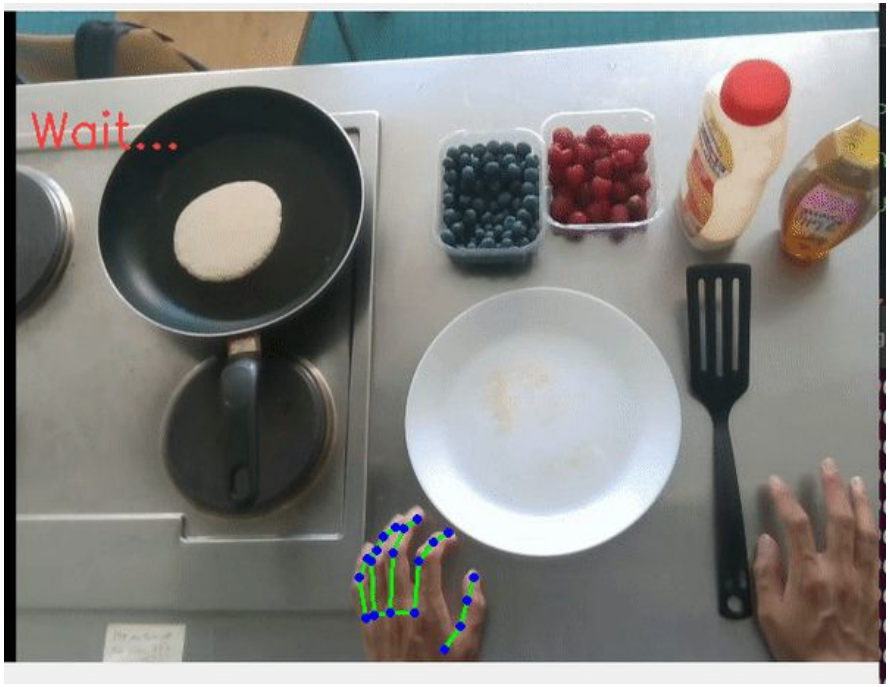


Figure 27: Visualization for pick. open and pour batter

Modules Visualization

- **Verb + Noun Detection**

Wait



Pick and shovel

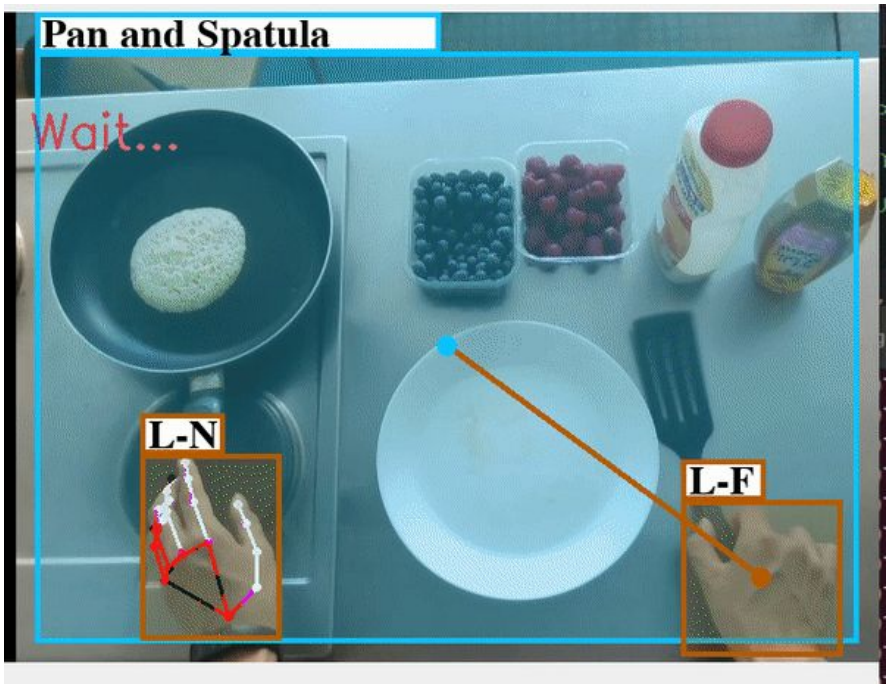
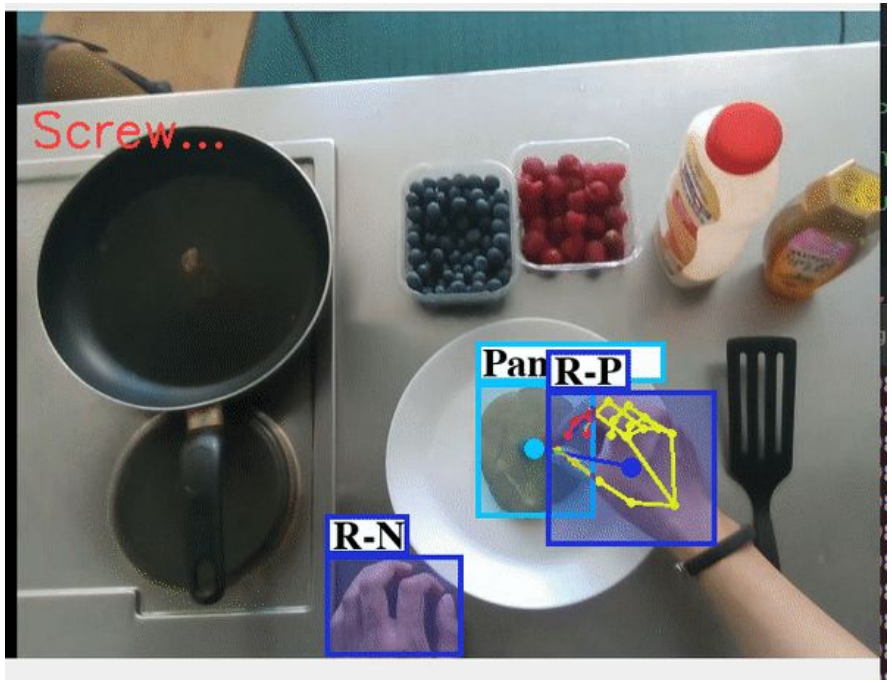


Figure 28: Visualization for wait, pick and shovel

Modules Visualization

- **Verb + Noun Detection**

Open, squeeze, decorate



A perfect pancake!

Figure 29: Visualization for open. squeeze and decorate

Conclusion

Human Activity Understanding

🥞 **Demo**

- **Ego-Perspective:**
  - Head mounted RGB-D camera (Realsense D435I)
  - Ego-Perspective data
- **Problem Domain**
  - **Kitchen:** To make a "perfect" pancake from zero

Input:
- **Images**

→

- Object Detection
- Noun Detection
- Verb Detection
- Visualizer

→

Output:
- **Objects + Positions**
- **Verb + Noun**

Figure 30: Project Pipeline Workflow

Future Work

- **Overfitting**
  - More subjects, more randomizations, more recordings
- **Depth Information Implementation**
  - Use depth for more **precise** hand landmarks verb detection
- **Problem Domain Exploration**
  - **Hand landmarks** verb detection can be adapted to other tasks
  - Noun detection with active objects
- **Task Feasibility for Robots**
  - **Comprehensive survey** on task feasibility for home robots
  - Create **human intervention indication** based on detections and survey
- **Detection → Prediction**
  - Shift labels to the next timestamp

Reference

I. Damen, Dima, et al. "Rescaling egocentric vision: collection, pipeline and challenges for epic-kitchens-100." International Journal of Computer Vision 130.1 (2022): 33-55.
II. Lu, Dengsheng, and Qihao Weng. "A survey of image classification methods and techniques for improving classification performance." International journal of Remote sensing 28.5 (2007): 823-870.
III. Zhang, Fan, et al. "Mediapipe hands: On-device real-time hand tracking." arXiv preprint arXiv:2006.10214 (2020).
IV. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
V. https://github.com/IntelRealSense/librealsense/blob/81d469db173dd682d3bada9bd7c7570db0f7cf76/wrappers/python/examples/opencv_pointcloud_viewer.py