

#IIE project_03

Rolling Planet

- The Combination of Augmented Reality & Deep Learning

109034545 蘇沛強

Our Journey

- Introduction
- 5W1H
- Recognize the Dice
- Handling input
- Augmented Reality game
- Conclusion

Introduction

What are we trying to do?

Train a model to train a human.

Background

Combination AR & AI to help kids:

- Recognize digit
- Learning Arithmetic
- Handwrite digit



5W1H

5W1H

What

Enhance the efficiency of learning arithmetic and recognize digit.

Why

Help child get better at learning math.

When

The period children are learning arithmetic and recognize digit.

Where

School, Home, Elementary education departments

Who

Kids and student.

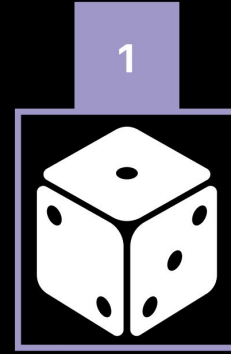
How

Through the Augmented reality game combines with AI.

Recognize the Dice

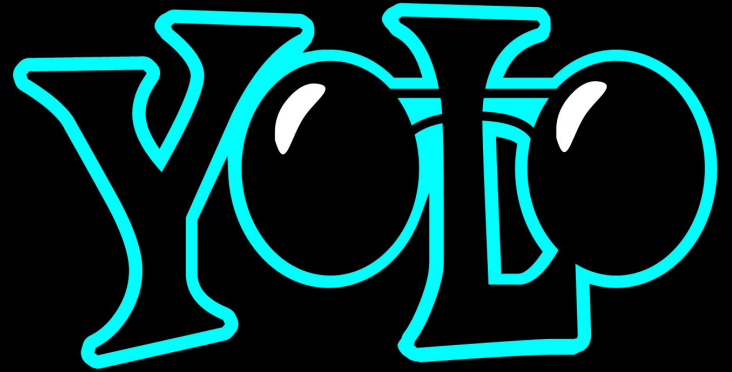
What we need?

- Recognize the number of dice on the table.
- Recognizing the values on the dice
- Real time detection and classification.



Model_YOLO 9000

- Popular objected detection framework.
- One stage
- superb speed
- Better, Faster, Stronger than YOLO



Why not YOLO v3, v4, v5?

Big model will result in long latency

| Type | Filters | Size/Stride | Output |
|---------------|---------|----------------|------------------|
| Convolutional | 32 | 3×3 | 224×224 |
| Maxpool | | $2 \times 2/2$ | 112×112 |
| Convolutional | 64 | 3×3 | 112×112 |
| Maxpool | | $2 \times 2/2$ | 56×56 |
| Convolutional | 128 | 3×3 | 56×56 |
| Convolutional | 64 | 1×1 | 56×56 |
| Convolutional | 128 | 3×3 | 56×56 |
| Maxpool | | $2 \times 2/2$ | 28×28 |
| Convolutional | 256 | 3×3 | 28×28 |
| Convolutional | 128 | 1×1 | 28×28 |
| Convolutional | 256 | 3×3 | 28×28 |
| Maxpool | | $2 \times 2/2$ | 14×14 |
| Convolutional | 512 | 3×3 | 14×14 |
| Convolutional | 256 | 1×1 | 14×14 |
| Convolutional | 512 | 3×3 | 14×14 |
| Convolutional | 256 | 1×1 | 14×14 |
| Convolutional | 512 | 3×3 | 14×14 |
| Maxpool | | $2 \times 2/2$ | 7×7 |
| Convolutional | 1024 | 3×3 | 7×7 |
| Convolutional | 512 | 1×1 | 7×7 |
| Convolutional | 1024 | 3×3 | 7×7 |
| Convolutional | 512 | 1×1 | 7×7 |
| Convolutional | 1024 | 3×3 | 7×7 |
| Convolutional | 1000 | 1×1 | 7×7 |
| Avgpool | | Global | 1000 |
| Softmax | | | |

Table 6: Darknet-19.

Why not YOLO v3, v4, v5?

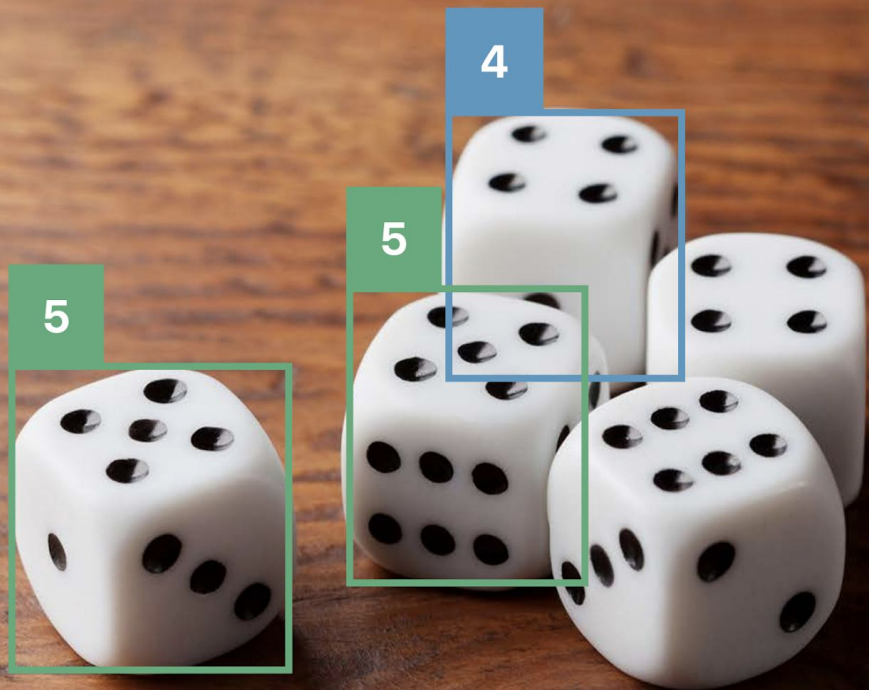
Big model will result in long latency

| | Type | Filters | Size | Output |
|----|---------------|---------|------------------|------------------|
| | Convolutional | 32 | 3×3 | 256×256 |
| | Convolutional | 64 | $3 \times 3 / 2$ | 128×128 |
| 1x | Convolutional | 32 | 1×1 | |
| | Convolutional | 64 | 3×3 | |
| | Residual | | | 128×128 |
| | Convolutional | 128 | $3 \times 3 / 2$ | 64×64 |
| 2x | Convolutional | 64 | 1×1 | |
| | Convolutional | 128 | 3×3 | |
| | Residual | | | 64×64 |
| | Convolutional | 256 | $3 \times 3 / 2$ | 32×32 |
| 8x | Convolutional | 128 | 1×1 | |
| | Convolutional | 256 | 3×3 | |
| | Residual | | | 32×32 |
| | Convolutional | 512 | $3 \times 3 / 2$ | 16×16 |
| 8x | Convolutional | 256 | 1×1 | |
| | Convolutional | 512 | 3×3 | |
| | Residual | | | 16×16 |
| | Convolutional | 1024 | $3 \times 3 / 2$ | 8×8 |
| 4x | Convolutional | 512 | 1×1 | |
| | Convolutional | 1024 | 3×3 | |
| | Residual | | | 8×8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |

Table 1. Darknet-53.

Training input

Data Annotation - Whole dice

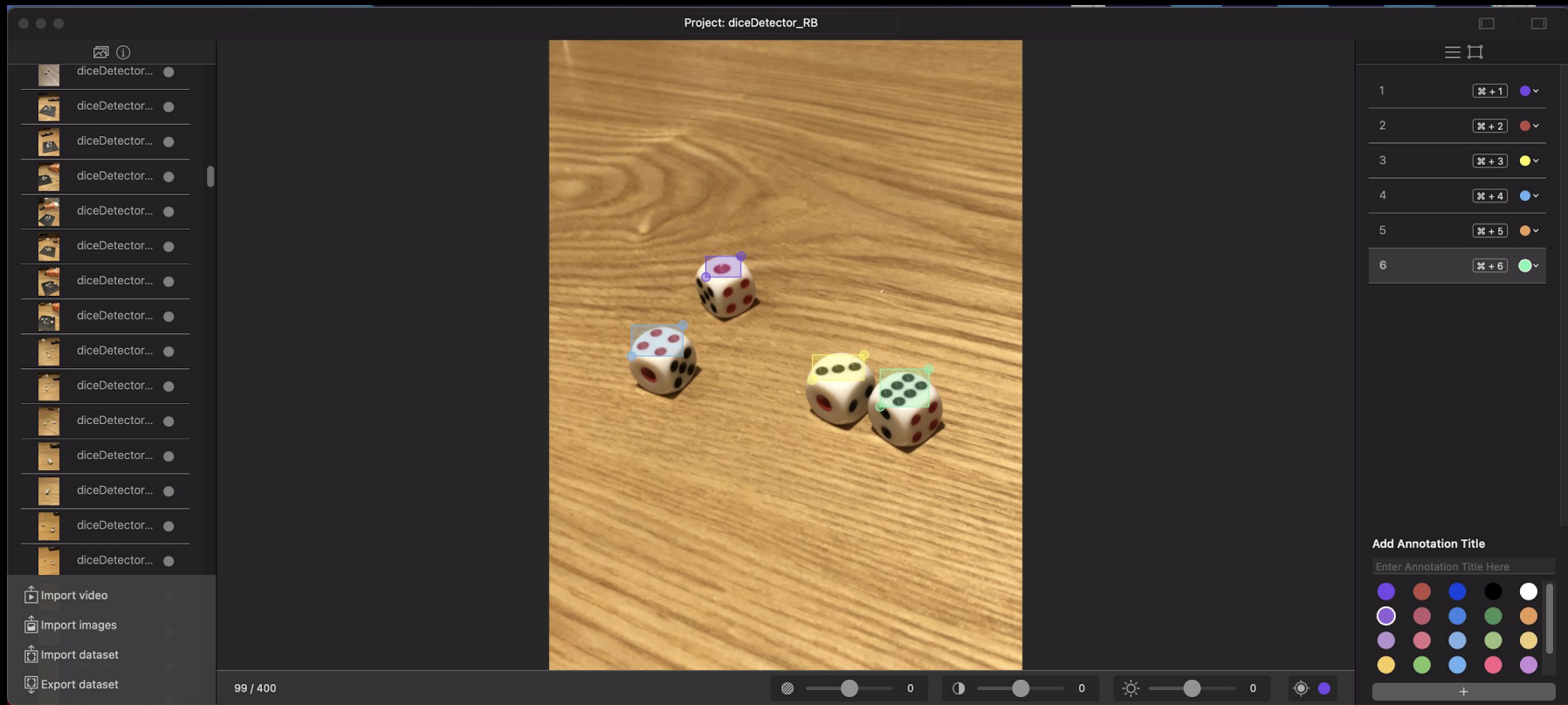


Data Annotation - Focusing on top



Data Annotation

Project: diceDetector_RB



The screenshot displays a data annotation application interface. The central area shows a video frame of four dice on a wooden surface. Each die is enclosed in a colored bounding box: a purple box on the top die, a blue box on the left die, a yellow box on the middle die, and a green box on the right die. The interface includes a left sidebar with a list of 15 'diceDetector...' items and options for 'Import video', 'Import images', 'Import dataset', and 'Export dataset'. The bottom status bar shows '99 / 400' and various control sliders. On the right, a legend lists six categories with corresponding colors and labels: 1 (purple, +1), 2 (red, +2), 3 (yellow, +3), 4 (blue, +4), 5 (orange, +5), and 6 (green, +6). Below the legend is an 'Add Annotation Title' section with a text input field and a color selection palette.

99 / 400

Add Annotation Title
Enter Annotation Title Here

1 +1
2 +2
3 +3
4 +4
5 +5
6 +6

Data Augmentation - Random crop



Data Augmentation - Color augmentation



Model Training

Training set: 440 (Raw data) → 20000 (Data Augmentation)

Class number: 6

Iterations: 15000

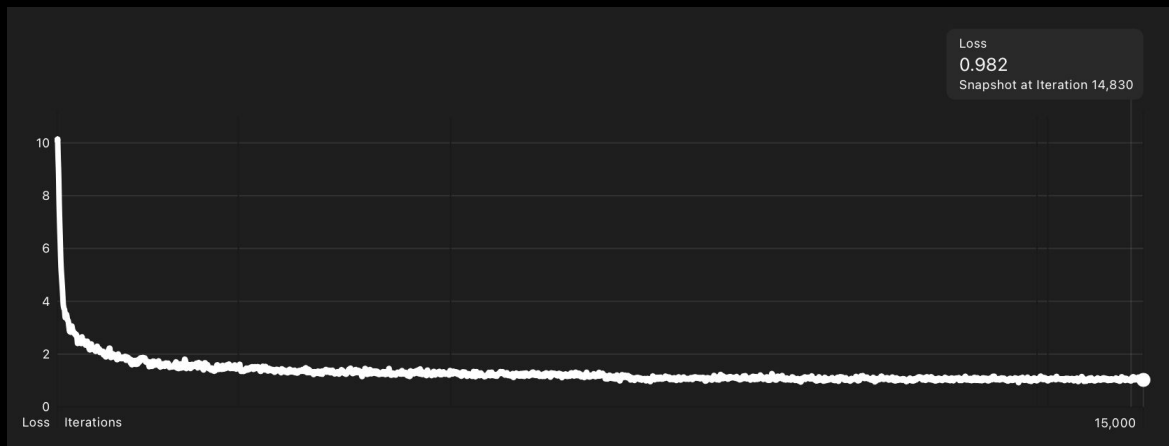
Batch size: 32

Validation Set: 1150 Items

Loss: 0.982

Training_Acc: 94%

Validation_Acc: 92%



Real input data preprocessing

camera frame input \neq Train input

Training input: 416 x 416 x 3

Camera input: **NEED RESCALE** to 416x416 pixels



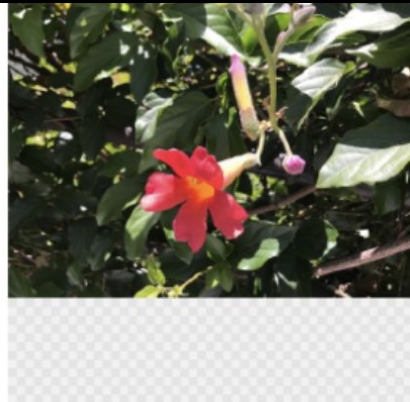
Original



Center crop



Scale fill



Scale fit

When Does a Roll End?

- We need a unequivocal signal to perform the game.



When Does a Roll End?

When we observe:

- The number of dice change **between two consecutive frames.**
- If the prediction of **two consecutive frame are different && the bounding box don't overlap over 0.85.**

Interpreting model output

```
func hasRollEnded(observations: [VNRecognizedObjectObservation]) -> Bool {  
    // First check if same number of dice were detected  
    if lastObservations.count != observations.count {  
        lastObservations = observations  
        return false  
    }  
    var matches = 0  
    for newObservation in observations {  
        for oldObservation in lastObservations {  
            // If the labels don't match, skip it  
            // Or if the IOU is less than 85%, consider this box different  
            // Either it's a different die or the same die has moved  
            if newObservation.labels.first?.identifier ==  
                oldObservation.labels.first?.identifier &&  
                intersectionOverUnion(oldObservation.boundingBox,  
                    newObservation.boundingBox) > 0.85 {  
                matches += 1  
            }  
        }  
    }  
    lastObservations = observations  
    return matches == observations.count  
}
```

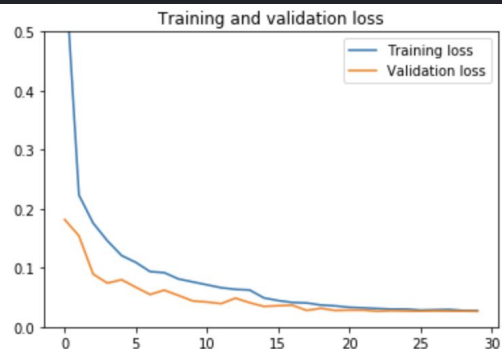
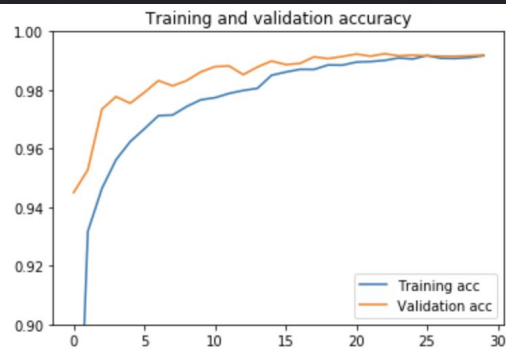
Model - LeNet

- Model Size: 0.2MB
- Training set: 60000
- Class number: 10
- Iterations: 30
- Batch size: 32
- Loss: 0.1441
- Training_Acc: 99.17 %
- Validation_Acc: 99.19 %
- Platform: Colab

Model: "sequential_1"

| Layer (type) | Output Shape | Param # |
|---------------------------------------|--------------------|---------|
| conv2d_1 (Conv2D) | (None, 28, 28, 6) | 156 |
| average_pooling2d_1 (Average Pooling) | (None, 14, 14, 6) | 0 |
| conv2d_2 (Conv2D) | (None, 10, 10, 16) | 2416 |
| average_pooling2d_2 (Average Pooling) | (None, 5, 5, 16) | 0 |
| flatten_1 (Flatten) | (None, 400) | 0 |
| dense_1 (Dense) | (None, 120) | 48120 |
| dense_2 (Dense) | (None, 84) | 10164 |
| dense_3 (Dense) | (None, 10) | 850 |

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0



Handling input

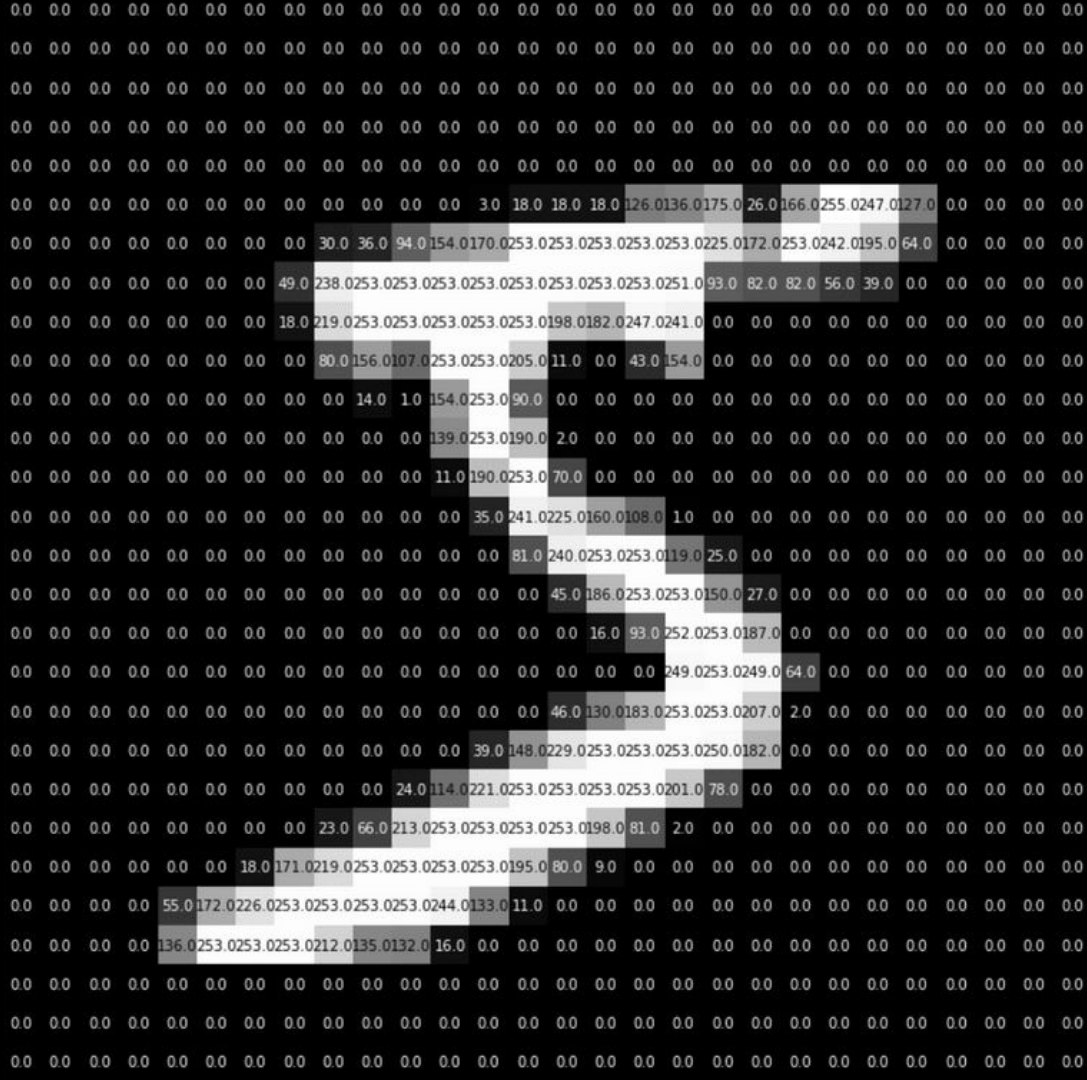
Handwriting digit classifier in real-time

- Let user input handwriting stroke to make UX consistent.



MNIST dataset

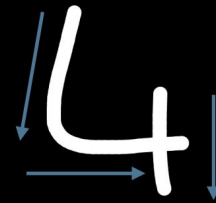
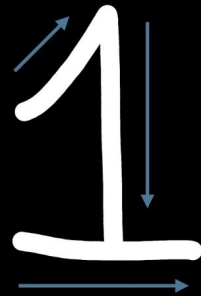
- 60000 Training set
- 10000 Testing set
- 10 Classes



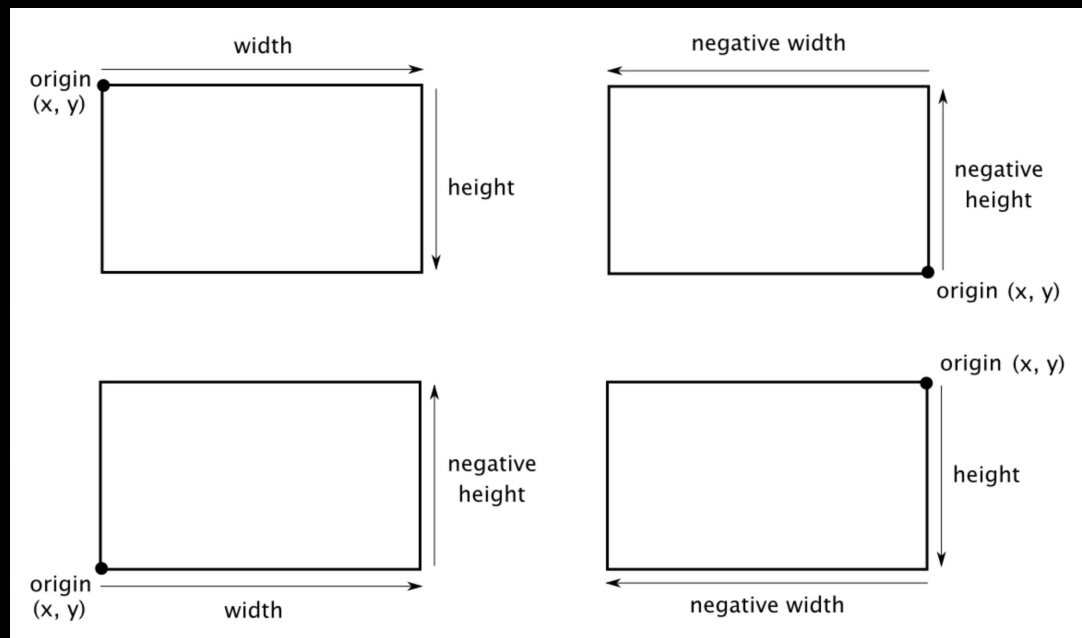
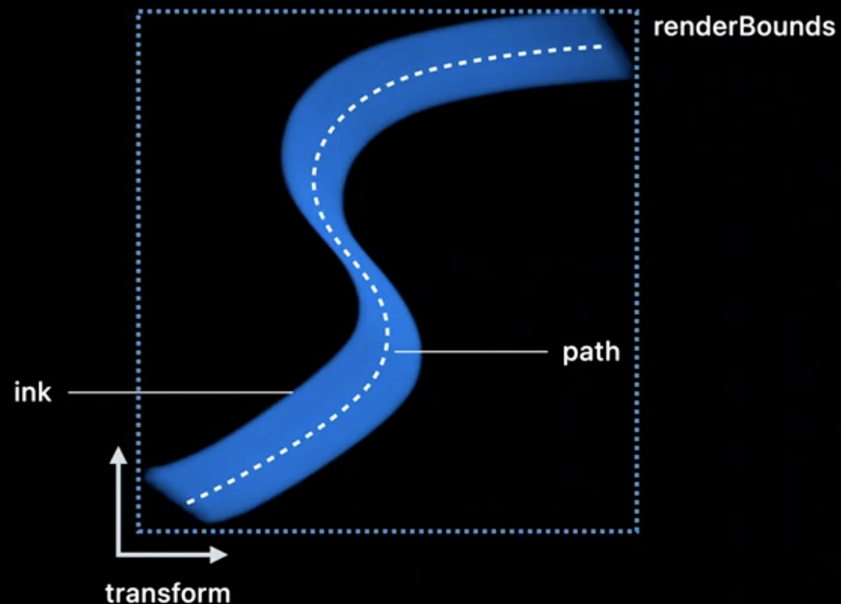
Demo



How to dealing with multiple digit and multiple stroke?



Understanding the strokes



Understanding the strokes

While there exists any stroke:

1. Keep update the position of the final stroke
2. if **minX** of final stroke $>$ **MaxX** in previous stroke:

New digit detected!

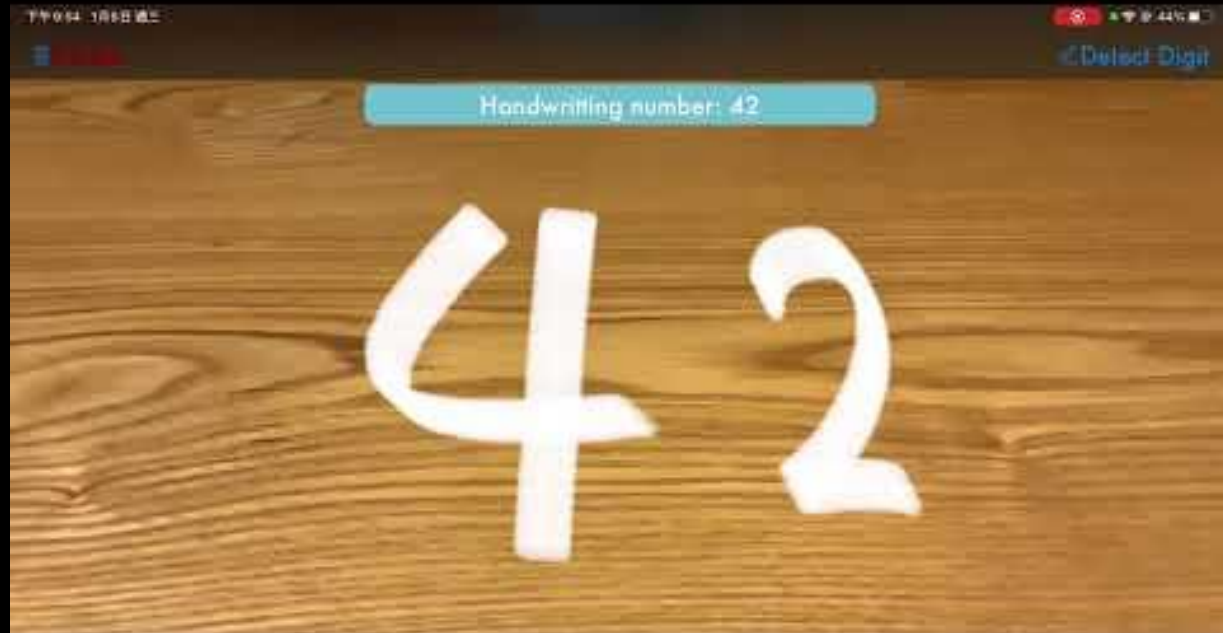
Inference old digit and push into **queue**.

Comparision each digit with the result of objected detection.

Understanding the stroke

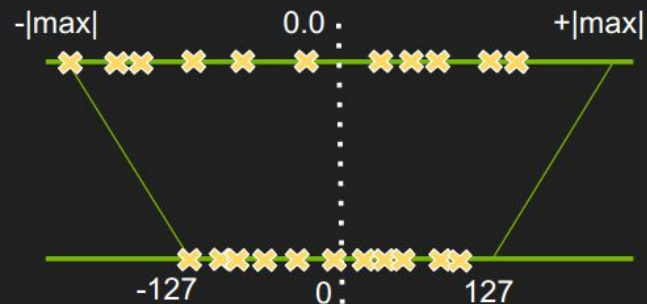
```
if self.stableFlag == 1 {
  if self.userInput[0] == -1
  {
    if self.wrongCount == 0 {
      self.handwriteDigit.text = "Write down your number."
    }
    else {
      self.handwriteDigit.text = "Try again!"
    }
  }
  else if self.userInput[0] != -1 && self.userInput[1] == -1
  {
    self.handwriteDigit.text = "Handwritting number: \(self.userInput[0])"
  }
  else if self.userInput[0] != -1 && self.userInput[1] != -1
  {
    self.handwriteDigit.text = "Handwritting number: \(self.userInput[0])\(self.userInput[1])"
  }
}
else {
  self.handwriteDigit.text = "\(self.playerLabel.text!) rolling the dice!"
  self.stableFlag = 0
}
```


Demo

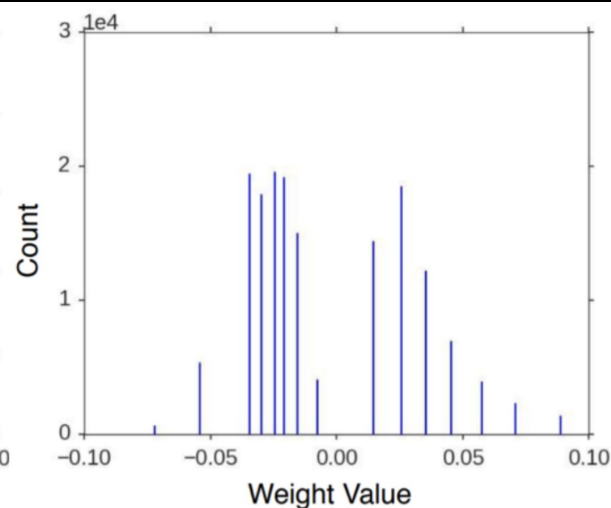
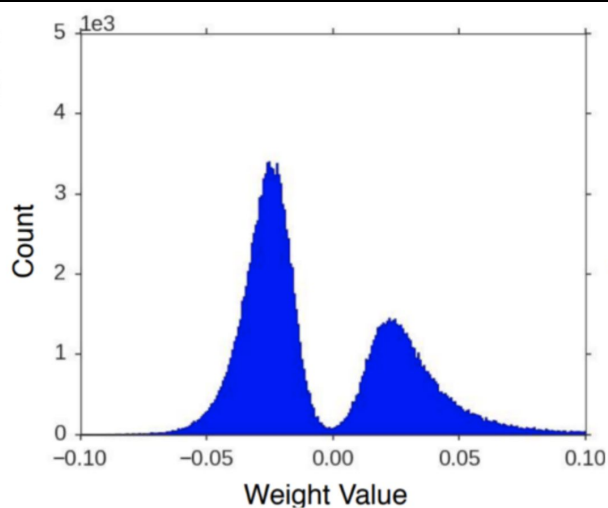


Quantize the model and integrate to app.

- No saturation: map $|\max|$ to 127



- Significant accuracy loss, in general



Augmented Reality game

Rules of the Game

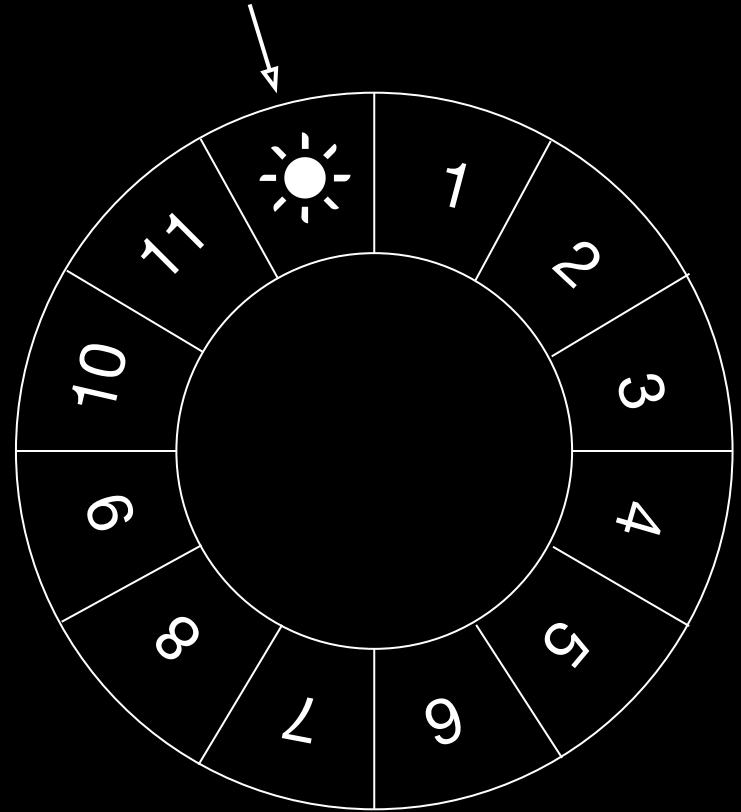
Roll one or two dice

Move the planet by:

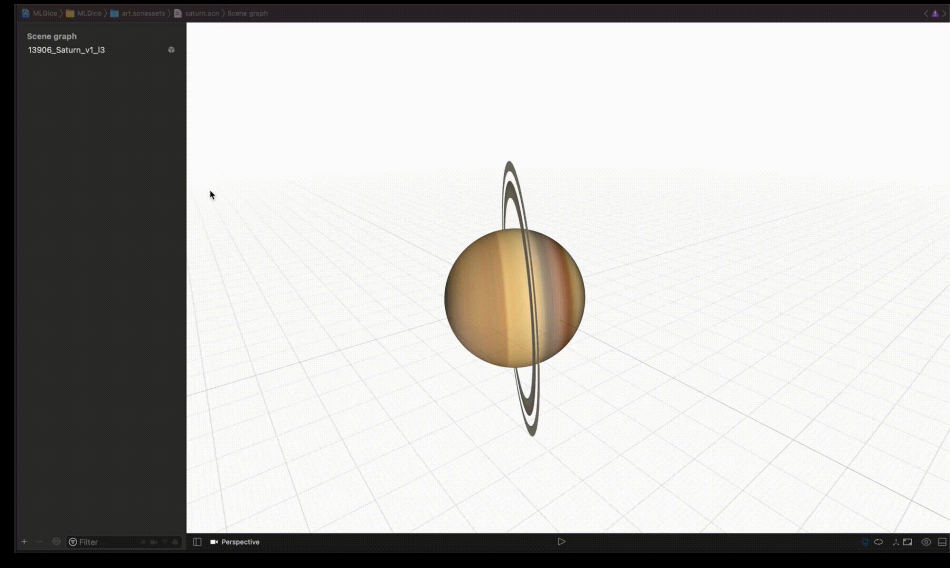
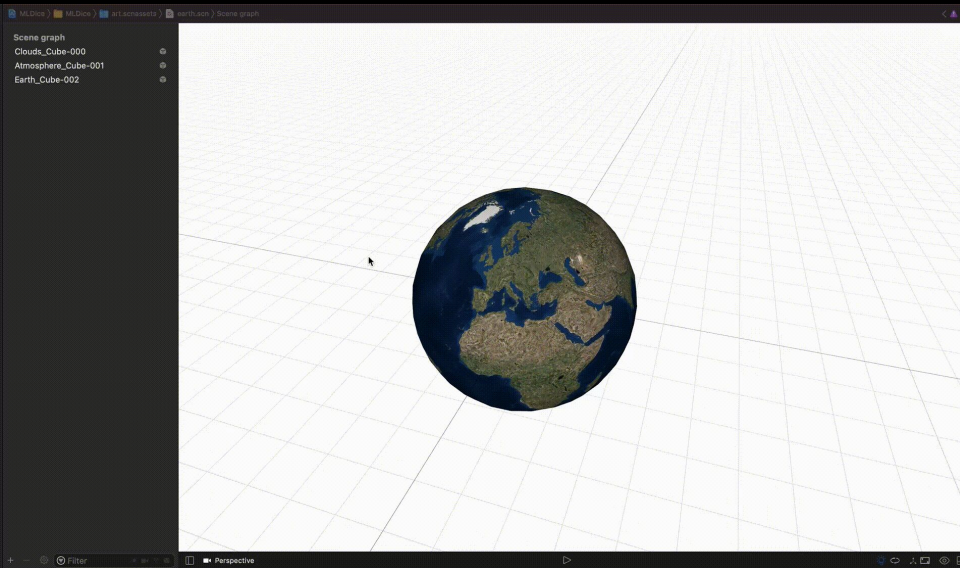
1. Addition
2. Substraction
3. Multiplication

Winner Condition:

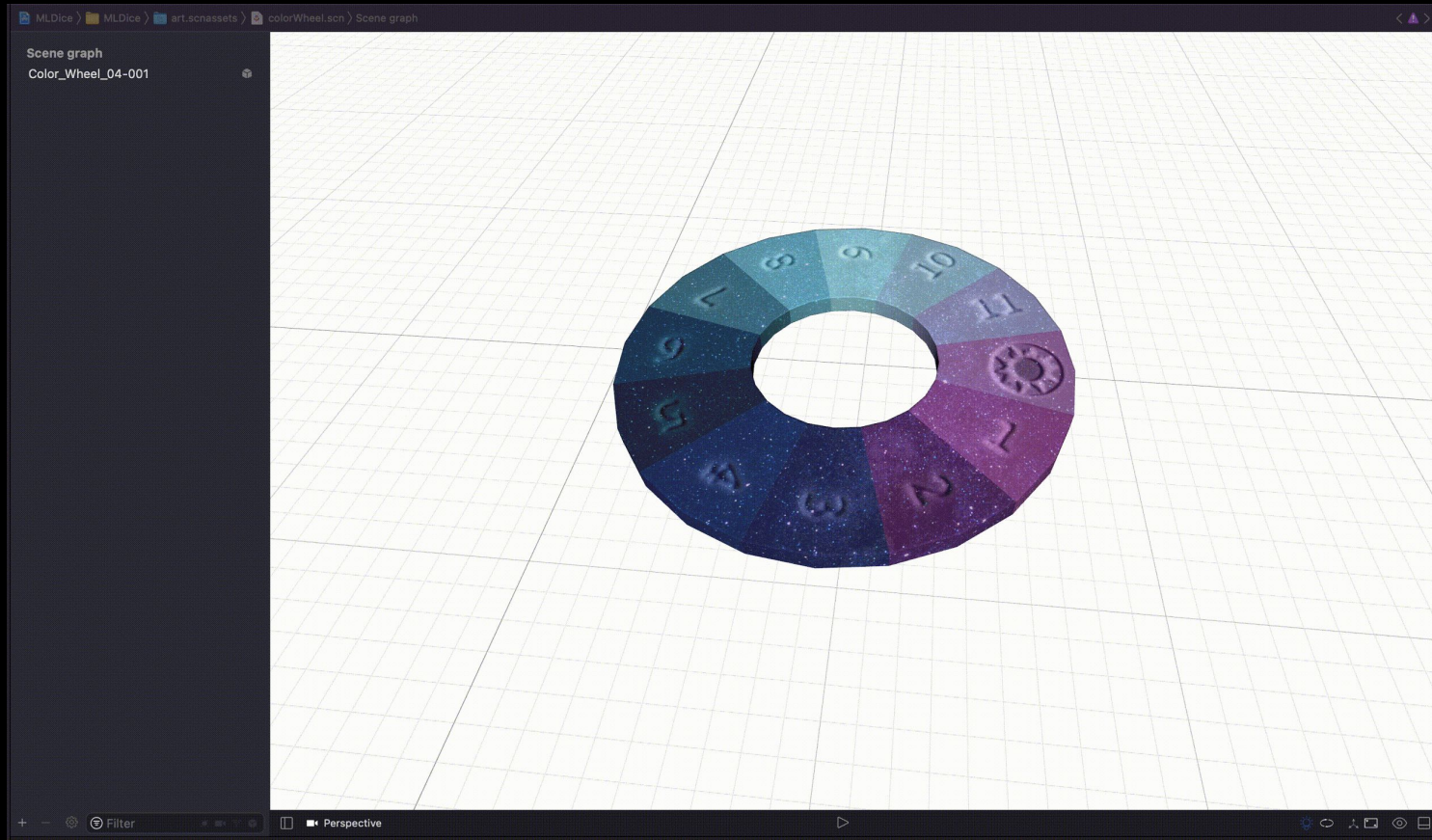
First player went back and right on the sun.



3D modeling - Planet



3D modeling - Galaxy Orbit



Correct input

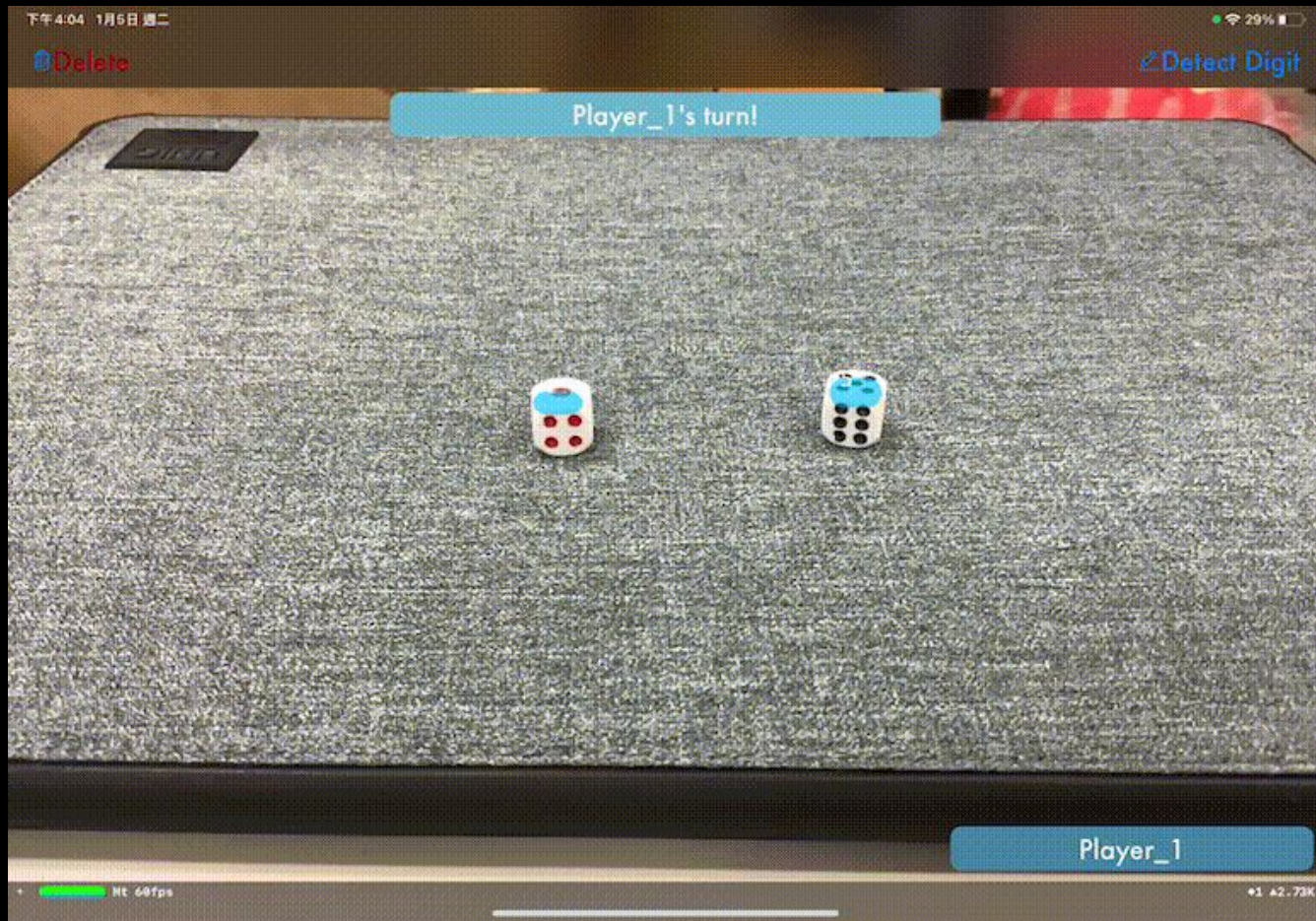


$$5 + 1 = 6$$

$$5 \times 1 = 5$$

$$5 - 1 = 4$$

Demo



Finalize the game.

Demo



Demo



Conclusion

- Real-world application **NOT** only focus on accuracy.
- Software architecture and algorithm are critical.
- Integration with different techniques is challenge but valuable.

Conclusion

Shortcoming

1. Improve the model by tuning more parameter.
2. Apply multiple precision quantization and pruning to different layer.
3. Try to add different I/O.
4. Try to deploy to more device and OS.

Prospect

1. Extend to much more application aspect.
2. Integrate with other computer vision, human interactive interface and artificial intelligence.
3. Develop more human-computer interaction services.

Thanks.

