

บทเล่นเกมอัตโนมัติ One Hit Kill โดยใช้ image classification

โดย ญาณหัย ไชยธรัช 6410500211

บทนำ

เกม One Hit Kill เป็นเกมในเว็บไซต์ itch.io, ที่เราต้องสู้กับศัตรูไม่จำกัดในขณะที่เลือดของเราอยู่ที่ 1HP เท่านั้น. เราต้องจอมตีล้มพากมันภายใน การโจมตีเดียวเพื่อชนะและไปต่อ

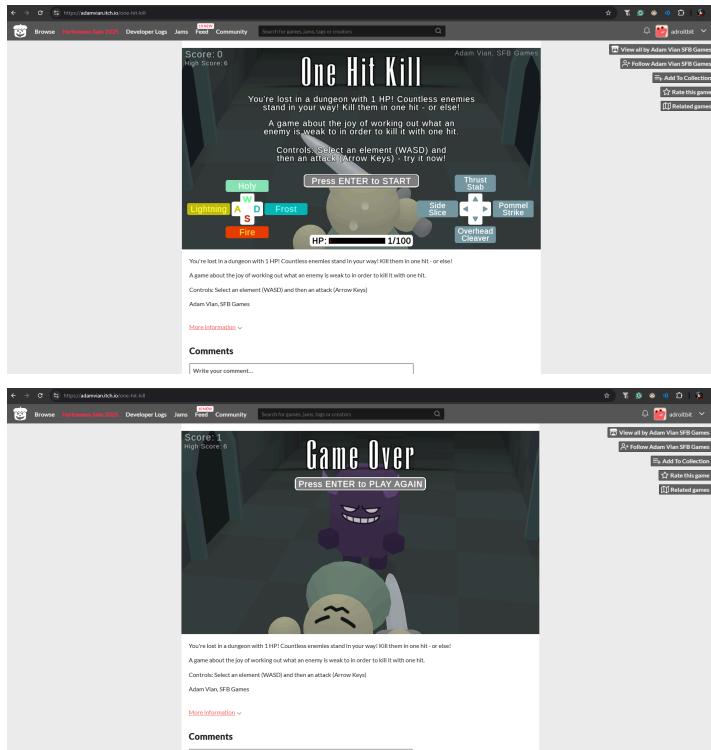
ศัตรุแต่ละตัวจะมีจุดอ่อนของมันเอง.

- กด WASD เพื่อเลือกธาตุที่จะใช้โจมตี
- กดปุ่มลูกศร เพื่อเลือกการทำการโจมตี

ด้วยความง่ายๆ กดแล้วรัน ใจง่ายของเกม ผมจึงอยากลอง ทำบอตเล่นเกมนี้อัตโนมัติ, เพื่อเล่นเกมนี้แทนเราดู และดูว่าจะได้ คะแนนสูงสุดอยู่ที่เท่าไหร

โปรเจคนี้ผมเคยทำขึ้นมาแล้วครั้งนึงแต่ใช้ YoloV11 ที่เป็น object detection เพื่อตรวจจับศัตรู

➡️ [Automatic bot for \[One Hit Kill\]\(itch.io\)](#) แต่ครั้งนี้จะลองใช้วิชาของ MIT Deep Learning เพื่อทำ image classification เพื่อตรวจจับศัตรูแทน



ตัวเกม: <https://adamvian.itch.io/one-hit-kill>

สาเหตุที่ใช้ Deep Learning

วิธีที่จะไม่ใช้ deep learning หรือ machine learning เลยในการคัดลักษณะทางกายภาพของศัตรูนั้นก็คือใช้ อัลกอริทึม SIFT หรือ ORB, นั้นคือการใช้ OpenCV, แต่จากการณ์ที่ผ่านมาของผมวิธีเหล่านั้น มันไม่สามารถเร่งความเร็วได้ และการจะ detect ศัตรูต้องใช้หลาย sample แบบ real time ทำให้ข้าเป็นพิเศษได้เสมอ

การใช้ YoloV11 ก็มีความเร็วเป็นพิเศษและจากที่ผมเคยทำมาโดยใช้วิธีที่ทำให้ได้คะแนนสูงสุดไปถึง 38 แต่อยากได้คะแนนสูงกว่านี้อยู่ดีและตอนรันโน้มเดล Yolo แบบ polling ตึงรูปเรื่อยๆ ก็รู้สึกได้ถึง latency นิดนึง, การเปลี่ยนมาใช้ image classification อาจจะทำให้บอททำงานเร็วขึ้น

	ข้อดี (ในเชิงทฤษฎี)	ข้อเสีย (ในเชิงทฤษฎี)
ใช้ SIFT เพื่อคัดประเภทของศัตรู	Train ตัว detector ง่าย ไม่ต้องใช้ GPU	Detect ช้าเกินไปสำหรับ real time กินทรัพยากร CPU ค่อนข้างเยอะ
ใช้ Yolov11 เพื่อคัดประเภทของศัตรู	เร็ว, เนื่องจากถูก optimize มา หลายอย่าง แม่นยำ เนื่องจากตัว framework ผสมเทคนิค augmentation และ อื่นๆ เข้าไป	ใช้ GPU อาจจะมี overhead ของ Yolo เอง
ทำ image classification เอง เพื่อคัดประเภทศัตรู	เร็วที่สุด, และอาจจะเร็วกว่า Yolo เนื่องจาก Yolo อาจจะมี overhead งานที่เราต้องการ ก็แค่จำแนก ศัตรูเท่านั้น การใช้วิธีนี้จึงจะง่าย กว่า	ใช้ GPU อาจจะเทียบกับการใช้ Yolo ไม่ ได้ในเรื่องความแม่นยำ เนื่องจาก Yolo ถูกพัฒนาให้รวมเทคโนโลยี หลายอย่างเข้ามาอยู่แล้ว เทคนิคการทำ image classification ต้อง implement เอง

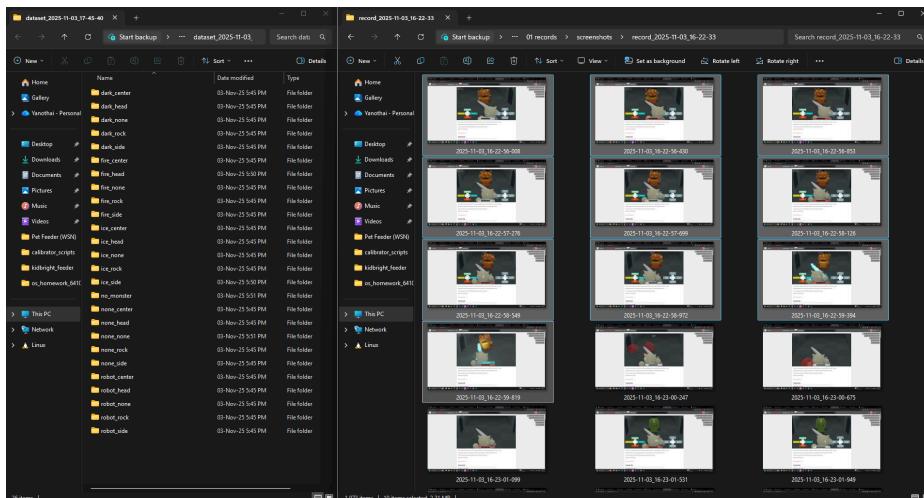
ขั้นตอนการทำงาน

เก็บข้อมูล

```
01 records > recorder.py > ...
1 import pyautogui
2 import os
3 import time
4 from datetime import datetime
5
6
7
8 def create_record_folder(base_dir="screenshots"):
9     """Create a new timestamped folder for this recording session."""
10    timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
11    folder_name = f'record_{timestamp}'
12    path = os.path.join(base_dir, folder_name)
13    os.makedirs(path, exist_ok=True)
14    return path
15
16 def record_screen(interval=1.0):
17     """Continuously capture screenshots and save them with timestamps."""
18     save_dir = create_record_folder()
19     print(f"[INFO] Saving screenshots to: {save_dir}")
20     print("[INFO] Press Ctrl + C to stop recording.")
21
22     try:
23         while True:
24             # Create a timestamped filename
25             filename = datetime.now().strftime("%Y-%m-%d_%H-%M-%S-%f")[-3:] + ".png"
26             filepath = os.path.join(save_dir, filename)
27
28             # Capture and save
29             screenshot = pyautogui.screenshot()
30             screenshot.save(filepath)
31
32             print(f"[CAPTURED] {filename}")
33             time.sleep(interval)
34
35     except KeyboardInterrupt:
36         print("\n[INFO] Recording stopped by user.")
37         print(f"[INFO] Screenshots saved in: {save_dir}")
38
39 if __name__ == "__main__":
40     os.chdir(os.path.dirname(os.path.abspath(__file__)))
41     print(os.getcwd())
42     # record_screen(interval=1.0) # 1 screenshot per second
43     record_screen(interval=1/3) # 1 screenshot per second
```

เก็บข้อมูลจากภาพหน้าจอเกมด้วยการรันสคริปต์ที่ผ่านด้วย pyautogui และ save ลงโฟลเดอร์

กำกับข้อมูล



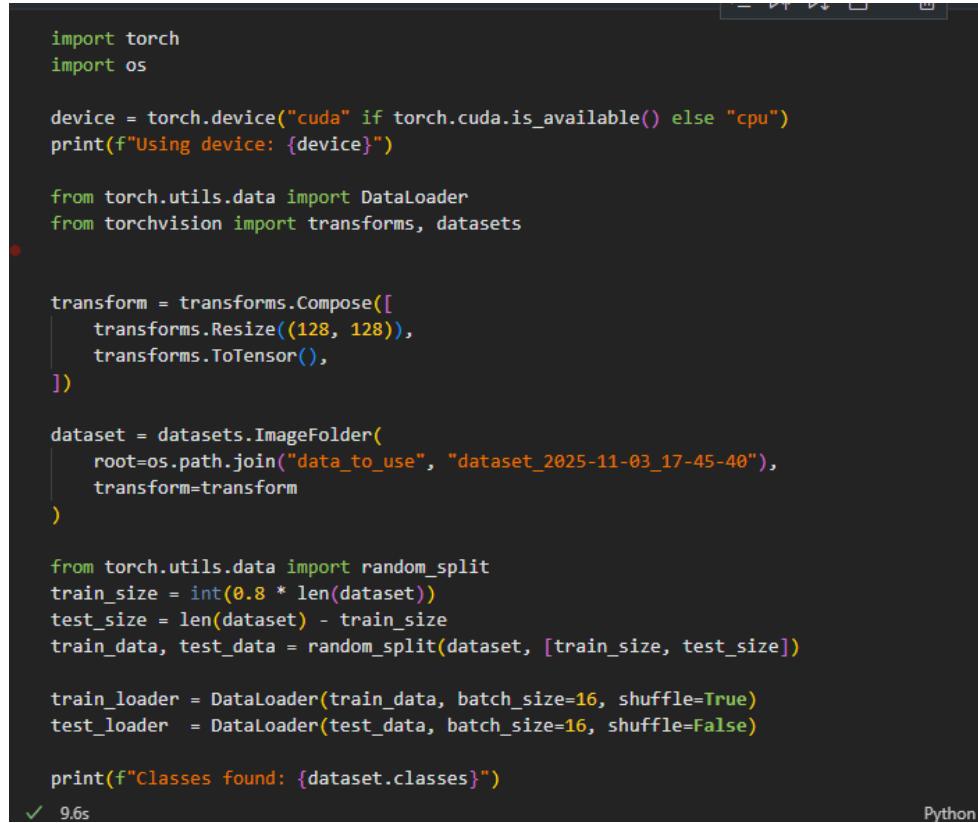
ทำ dataset โดยใช้รูปแบบ Folder-based labels ของ Pytorch

โดย format ของ class คือ “ฐาน_จุดอ่อน” “element_weakness” ซึ่งถูกมาใช้เป็นชื่อไฟล์เดอร์ แสดง class

โดย label ไปแล้ว 643 รูป เป็น dataset

เทรน์โนเมเดล

เริ่มจากการ initialize สำหรับ data loader สำหรับตัวโนเมเดล



```
import torch
import os

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

from torch.utils.data import DataLoader
from torchvision import transforms, datasets

transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

dataset = datasets.ImageFolder(
    root=os.path.join("data_to_use", "dataset_2025-11-03_17-45-40"),
    transform=transform
)

from torch.utils.data import random_split
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_data, test_data = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_data, batch_size=16, shuffle=True)
test_loader = DataLoader(test_data, batch_size=16, shuffle=False)

print(f"Classes found: {dataset.classes}")
```

✓ 9.6s

Python

นิยามสถาปัตยกรรมโมเดล

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    """
    Simple 3-layer CNN for 128x128 RGB images.
    Automatically builds correct FC layer size.
    """

    def __init__(self, num_classes):
        super().__init__()

        # Input: (3, 128, 128)
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        # Output: (16, 128, 128)
        self.pool1 = nn.MaxPool2d(2, 2)
        # Output after pool1: (16, 64, 64)

        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        # Output: (32, 64, 64)
        self.pool2 = nn.MaxPool2d(2, 2)
        # Output after pool2: (32, 32, 32)

        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        # Output: (64, 32, 32)
        self.pool3 = nn.MaxPool2d(2, 2)
        # Output after pool3: (64, 16, 16)

        self.dropout = nn.Dropout(0.3)

        # Dynamically determine fc input size

        # Precompute FC input size (for 128x128 images)
        self.fc_input_size = 64 * 16 * 16
        self.fc1 = nn.Linear(self.fc_input_size, 128)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = F.relu(self.conv3(x))
        x = self.pool3(x)
        x = x.view(x.size(0), -1)
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.fc2(x)
        return x

model = SimpleCNN(num_classes=26).to("cuda" if torch.cuda.is_available() else "cpu")
dummy = torch.randn(1, 3, 128, 128).to(next(model.parameters()).device)
out = model(dummy)
print("Final output:", out.shape)

with torch.no_grad():
    x = torch.randn(1, 3, 128, 128).to(device)
    for layer in [model.conv1, model.pool1, model.conv2, model.pool2, model.conv3, model.pool3]:
        x = layer(x)
        print(x.shape)
```

✓ 0.9s

```
from torchsummary import summary

model = SimpleCNN(num_classes=26).to(device)
summary(model, input_size=(3, 128, 128))

✓ 0.2s



| Layer (type)                     | Output Shape      | Param #   |
|----------------------------------|-------------------|-----------|
| =====                            |                   |           |
| Conv2d-1                         | [1, 16, 128, 128] | 448       |
| MaxPool2d-2                      | [1, 16, 64, 64]   | 0         |
| Conv2d-3                         | [1, 32, 64, 64]   | 4,640     |
| MaxPool2d-4                      | [1, 32, 32, 32]   | 0         |
| Conv2d-5                         | [1, 64, 32, 32]   | 18,496    |
| MaxPool2d-6                      | [1, 64, 16, 16]   | 0         |
| Linear-7                         | [1, 128]          | 2,097,280 |
| Dropout-8                        | [1, 128]          | 0         |
| Linear-9                         | [1, 26]           | 3,354     |
| =====                            |                   |           |
| Total params:                    | 2,124,218         |           |
| Trainable params:                | 2,124,218         |           |
| Non-trainable params:            | 0                 |           |
| -----                            |                   |           |
| Input size (MB):                 | 0.19              |           |
| Forward/backward pass size (MB): | 4.38              |           |
| Params size (MB):                | 8.10              |           |
| Estimated Total Size (MB):       | 12.67             |           |
| -----                            |                   |           |


```

```

import torch.optim as optim

model = SimpleCNN(num_classes=len(dataset.classes)).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# optimizer = optim.Adam(model.parameters(), lr=0.01)

epochs = 10
for epoch in range(epochs):
    model.train()

    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss+=loss.item()
    avg_loss = running_loss / len(train_loader)
    print(f"Epoch [{epoch+1}/{epochs}] Loss: {avg_loss:.4f}")

```

✓ 3m 7.6s

```

Epoch [1/10] Loss: 2.7244
Epoch [2/10] Loss: 2.6331
Epoch [3/10] Loss: 2.5133
Epoch [4/10] Loss: 2.4138
Epoch [5/10] Loss: 2.3325
Epoch [6/10] Loss: 2.1979
Epoch [7/10] Loss: 1.9840
Epoch [8/10] Loss: 1.9533
Epoch [9/10] Loss: 1.8642
Epoch [10/10] Loss: 1.7115

```

```

model.eval()
correct, total = 0, 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"Accuracy: {accuracy:.2f}%")

```

✓ 4.6s

```
Accuracy: 52.71%
```

จากนั้น save weight ของ model เพื่อนำไปใช้งานต่อ

Save weight and reuse weight

```
save_path = "simplecnn_weights.pth"
torch.save(model.state_dict(), save_path)
print(f"✓ Model weights saved at {save_path}")
✓ 0.1s
✓ Model weights saved at simplecnn_weights.pth

# Recreate model with same architecture
model = SimpleCNN(num_classes=len(dataset.classes)).to(device)

# Load saved weights
model.load_state_dict(torch.load(save_path, map_location=device))
model.eval() # set to eval mode for inference
print("✓ Model weights loaded successfully")
✓ 0.1s
✓ Model weights loaded successfully
```

ใช้งานโมเดลจิงกับบอท

```
class SimpleCNN(nn.Module):
    def forward(self, x):
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = F.relu(self.conv3(x))
        x = self.pool3(x)
        x = x.view(x.size(0), -1)
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.fc2(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN(num_classes=25).to(device)
model.load_state_dict(torch.load("simplecnn_weights.pth", map_location=device))
model.eval()

model_classes = ['dark_center', 'dark_head', 'dark_none', 'dark_rock',
                  'fire', 'ice', 'lightning', 'rock', 'water']

import pyautogui
from PIL import Image
import torchvision.transforms as transforms
import time
import keyboard as kb

transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

attack_on_element_pattern={
    "robot": "a",
    "fire": "d",
    "ice": "s",
    "dark": "w",
    "none": "z",
}
attack_on_weakness_pattern={
    "none": "right",
    "side": "left",
    "head": "down",
    "center": "up",
    "rock": "right",
}

while True:
    time.sleep(1/10)
    # Capture screen region
    screenshot = pyautogui.screenshot()
    image = transform(screenshot).unsqueeze(0).to(device)

    with torch.no_grad():
        output = model(image)
        _, predicted = torch.max(output, 1)
    class_name = model_classes[predicted.item()]
    print("Detected:", class_name)

    if class_name == "no_monster":
        continue
    element, weakness = class_name.split("_")
    # print(element, weakness)
    for k, v in attack_on_element_pattern.items():
        if k == element:
            kb.press(v)
            time.sleep(0.05)
            kb.release(v)
            break

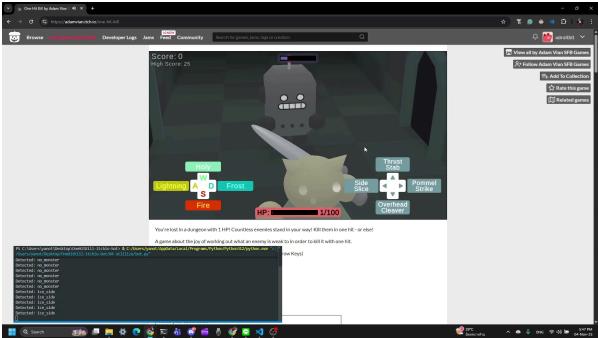
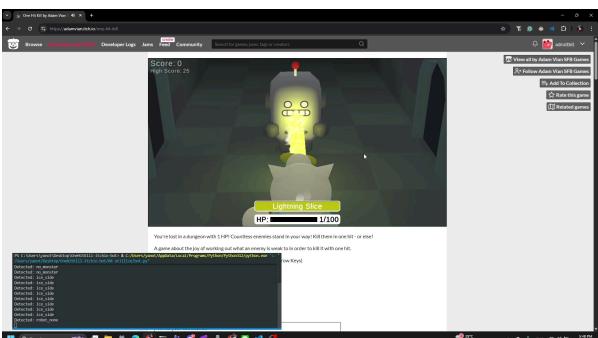
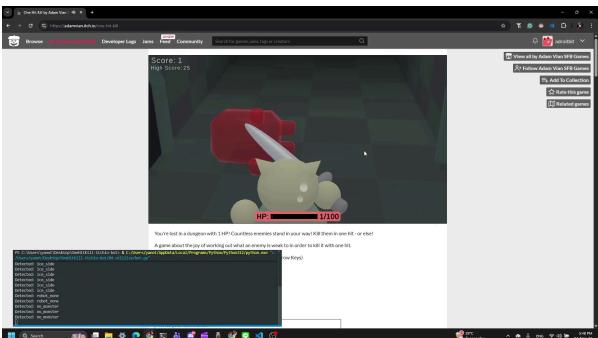
    for k, v in attack_on_weakness_pattern.items():
        if k == weakness:
            kb.press(v)
            time.sleep(0.05)
            kb.release(v)
            break
```

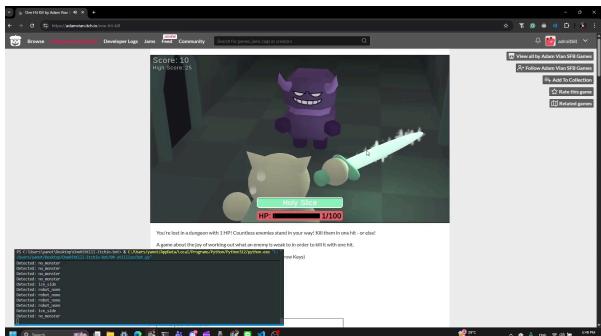
ใช้งานโมเดลจิงกับบอทด้วยการเขียนสคริปต์ของบอทให้กดปุ่มนคีย์บอร์ดตามลักษณะของศัตรูที่ โมเดลคำนึงได้มา

Accuracy ค่อนข้างต่ำเกินไป จึงไม่สามารถใช้งานกับบอทจริงได้ดีนัก

ผนวจังอัดวิดีโอเล่นด้วยตัวเอง และให้โมเดลทำนายไปพร้อมๆกัน เพื่อแสดงประสิทธิภาพของโมเดลตอนใช้งานจริง และจะแสดงผลลัพธ์บางส่วนในหน้ารายงานนี้แทน

<https://youtu.be/TVjJtoXOfsE?si=mUeMxYRaSaEonN8>

<h3>รุปจากคลิป</h3> 	<h3>คำอธิบาย</h3> <p>ศัตรู หุ่นยนต์ ไร้จดอ่อนโผล่มาในจาก แต่ทว่า โนเดลตรวจสอบได้เป็น ตัวน้ำแข็งที่มี จดอ่อนเป็นฟันข้าง</p>
	<p>(ต่อ) แต่ระหว่างที่ดำเนินกำลังฟันเข้าไป โนเดลกลับมองเห็นว่าศัตรูเป็นหุ่นยนต์พอดี</p>
	<p>หลังจากที่ศัตรูตายไป, โนเดลก์ทำนายได้เป็นว่า ไม่มีสัตว์ประหลาด ถือว่าถูกต้อง</p>
	<p>แซร์บันนิงที่หุ่นยนต์ไร้จดอ่อนโผล่เข้ามา โนเดลก์ได้ทำนายถูกกว่าเป็น robot_none (หุ่นยนต์ ไร้จดอ่อน)</p> <p>แต่สักพักนึงก็ทำนายได้เป็น ice_side ที่สื่อถึง ตัว น้ำแข็งที่มีจดอ่อนเป็นฟันข้าง</p>



ตัวที่โผลมานาเป็นธาตุมีด ที่มีจุดอ่อนพื้นห้าง

แต่ โนเดลเห็นสับไปมา ระหว่าง ice_side (ตัวน้ำแข็งที่มีจุดอ่อนด้านข้าง) และ robot_none (ตัวหุ่นยนต์ ที่เร江东อ่อน)



ตัวไฟรูปแบบหินโผลมานา

โนเดลก์ทำนายผิดเป็น fire_center (ตัวไฟที่มีจุดอ่อนตรงกลางหอง)

แต่แล้วโนเดลก์ทำนายถูกอีกที่เป็น fire_rock (ตัวไฟรูปแบบหิน)



ตัวไรธาตุที่มีจุดอ่อนพื้นห้างโผลมานา (none_side)

แต่โนเดลทำนายได้เป็น ice_side (ตัวน้ำแข็งที่มีจุดอ่อนพื้นห้าง)
เริ่มสังเกตได้ว่าโนเดลมีการทำนาย ice_side ค่อนข้างบ่อย แปลว่าโนเดลอารจะกำลังโง่ค่า loss ด้วยการทำนาย ice_side หรือไม่ก็ data ที่เรียนรู้ ice_side ค่อนข้างเบolareชั่งลงกับ dataset



ตัวน้ำแข็งรูปแบบหินโผลมานา

ปรากฏว่าโนเดลทำนายถูกต้อง ชี้งตรงกับ sample