Busan science high school
2023 Ocean ICT Festival
2023 BOIF
B
37

QR 코드 영역
QR 삽입 후
테두리 삭제

Youtube 영상 QR

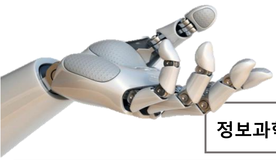
부수어획 방지를 위한 선택적 어획 기술 개발

물고기를 살려조 : 2512 윤제후, 2516 이현진

Introduction

육지 생물의 멸종 위기에 대한 심각성은 쉽게 접해볼 수 있다. 이러한 문제는 해양 생물의 경우에도 예외가 아니다. 특히 해양 생태계에서 일어나는 **부수어획**에 관한 문제점은 비교적 최근에서야 주목을 받고 있다. 부수 어획은 어업활동 과정에서, 의도치 않은 어종이 잡히며 발생하는 문제이다. 부수어획으로 잡힌 물고기의 80퍼센트 이상은 제기 불능 상태가 되기에, 방생 후에도 수일내에 사망한다. 큰 문제점은, 부수어획으로 사망하는 바다생물의 수는 25만마리로 추정되며, 이는 플라스틱을 삼켜 죽는 바다생물의 수 1천 마리에 비해 **250배** 높은 수치이다. 전문가들은 위 수치가 공식적인 통계로 비롯되며 불법 어선들로 인해 집계되지 않는 수는 훨씬 높다고 지적한다. 부수어획은 어종 감소, 해양 생태계 파괴에 직접적으로 영향을 주며, 버려지는 해양 생물들로 인한 비경제적 손실을 초래한다. 이외에도 해양 생물은 식량문제 와도 직결되어진다. 우리는 이러한 문제를 해결할 수 있는 대응책을 제시하고자 이번 작품을 제작하게 되었다.

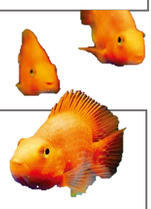
Combining field



정보과학 ◀ ▶ 생명과학 ◀ ▶ 기계공학

이번 작품에서는 기계공학 분야와 생명과학 분야, 정보과학 분야를 융합시키고자 한다. 해양 생태계 구성 및 해양 생물의 이동 경로, 서식지와 관련한 것에 있어서 생명과학 분야의 지식이 요구될 것으로 예측하고 있다.
해양 생물별 움직임의 특성 및 특징적인 부분에서도 생명과학 지식을 이용하고자 한다.
우리는 아두이노를 이용해 실제로 물고기 시뮬을 처리할 수 있는 장치를 기계적으로 구성해보고자 한다.
물고기를 식별하고, 선택적 어획 기술을 구현할 수 있는 모델을 제작하는 것을 목표로 두고 있다. 위 과정에서 해양 생물들의 이동 경로 및, 외형적 특징을 기반으로 분류, 분석을 처리하는 과정에서 회귀 분석을 진행하고자 한다.

Literature review



선형 회귀란?
통계학 및 머신 러닝에서 사용되는 기법 중 하나로, 두 변수 간의 관계를 모델링하는 데 사용된다.
주로 종속 변수(y)와 하나 이상의 독립 변수(x) 간의 선형적인 관계를 찾는 데 활용된다.

1) 단순 선형 회귀 : 하나의 독립 변수(x)와 하나의 종속 변수(y) 간의 관계를 모델링한다.
데이터를 가장 잘 설명하는 직선을 찾아내는 것을 목적으로 한다.

2) 다중 선형 회귀 : 둘 이상의 독립 변수(x1, x2, ...)와 하나의 종속 변수(y) 간의 관계를 모델링한다.
고차원 공간에서 데이터를 설명하는 평면을 찾아내는 것을 목적으로 한다.

적합도를 나타내는 지표로 평균 제곱 오차(Mean Squared Error) 등을 사용한다.
회귀 분석을 통해 얻은 선형 모델을 사용하면, 독립 변수의 값에 따른 종속 변수의 예측값을 쉽게 계산할 수 있다.

로지스틱 회귀란?
통계학 및 머신 러닝에서 사용되는 분류 기법으로, 종속 변수가 이항 혹은 다항 분류인 경우에 사용된다.
머신 러닝 분야에서 기본적이면서도 확실한 분류 알고리즘 중 하나로 이용된다.

Methodology

Linear Regression

- 선형 회귀를 통한
- **이미지 정확도 강화학습 모델**

```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_dir = 'path/to/training/dataset'
test_dir = 'path/to/test/dataset'
class_names = sorted(os.listdir(train_dir))

train_dataset = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)

test_dataset = ImageDataGenerator(rescale=1./255.0)

train_generator = train_dataset.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_dataset.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    keras.layers.MaxPooling2D(2, 2),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D(2, 2),
    keras.layers.Conv2D(128, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D(2, 2),
    keras.layers.Flatten(),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(len(class_names), activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_generator, epochs=10)

test_loss, test_accuracy = model.evaluate(test_generator)
```

- 데이터 경로 설정
- 데이터 전처리
- 학습 데이터 불러오기
- 테스트 데이터 불러오기
- CNN 모델 제작
- 모델 학습 및 평가

Arduino Code

- 초음파 센서를 이용한
- 거리 인식 알고리즘

```
const int trigPin = 9;
const int echoPin = 10;

void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);

  float distance = (duration / 2) * 0.0343;

  Serial.print("거리: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(1000);
}
```

- 초음파 센서 및 보드 통신
- 펄스 길이 측정
- 음파 속도를 통한 거리 계산
- 도달거리 출력

Ver.1

- 카메라 센서를 이용한
- 이미지 인식 알고리즘

```
#include <Arduino.h>
#include "memorysaver.h"
#include <UTFT.h>

ArduCAM myCAM(OV7670, CS_PIN);

void setup() {
  Serial.begin(115200);
  Wire.begin();
  myCAM.InitCAM();

  myCAM.OV7670_set_JPEG_size(OV7670_320x240);

void loop() {
  myCAM.flush_fifo();
  myCAM.clear_fifo_flag();
  myCAM.start_capture();

  while (!myCAM.get_bit(ARDUCHIP_TRIG, CAP_DONE_MASK));

  int length = myCAM.read_fifo_length();

  byte *imageBuffer = (byte *)malloc(length);
  myCAM.CS_LOW();
  SPI.transfer(0);
  while (length--) {
    imageBuffer[length] = SPI.transfer(0x00);
  }
  myCAM.CS_HIGH();

  File imageFile = SD.open("image.jpg", FILE_WRITE);
  if (imageFile) {
    imageFile.write(imageBuffer, length);
    imageFile.close();
    Serial.println("이미지 저장");
  } else {
    Serial.println("이미지 저장 실패");
  }

  free(imageBuffer);
  delay(1000);
}
```

- Cam모듈 생성
- 통신시작

Ver.2

```
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <ArduCAM.h>
#include <OV7670.h>

ArduCAM myCAM(OV7670, CS_PIN);

const int chipSelect = 4;
void setup() {

  Serial.begin(115200);
  if (!SD.begin(chipSelect)) {
    Serial.println("SD 카드 초기화 실패");
    return;
  }

  myCAM.InitCAM();
  myCAM.OV7670_set_JPEG_size(OV7670_320x240);
}

void loop() {

  myCAM.flush_fifo();
  myCAM.clear_fifo_flag();
  myCAM.start_capture();

  while (!myCAM.get_bit(ARDUCHIP_TRIG, CAP_DONE_MASK));
  int length = myCAM.read_fifo_length();

  byte *imageBuffer = (byte *)malloc(length);
  myCAM.CS_LOW();
  SPI.transfer(0);
  for (int i = 0; i < length; i++) {
    imageBuffer[i] = SPI.transfer(0x00);
  }
  myCAM.CS_HIGH();

  File imageFile = SD.open("image.jpg", FILE_WRITE);
  if (imageFile) {
    imageFile.write(imageBuffer, length);
    imageFile.close();
    Serial.println("이미지 저장");
  } else {
    Serial.println("이미지 저장 실패");
  }

  free(imageBuffer);
  delay(1000);
}
```

- Cam 초기화
- 캡쳐 시작
- 데이터 읽기
- 이미지 데이터 확인
- 이미지 데이터 처리

```
import os
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from skimage.io import imread
from skimage.transform import resize


data_dir = 'path/to/dataset'
class_names = sorted(os.listdir(data_dir))

X = []
y = []

for class_idx, class_name in enumerate(class_names):
    class_dir = os.path.join(data_dir, class_name)
    for image_filename in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_filename)
        image = imread(image_path)
        image_resized = resize(image, (100, 100))
        X.append(image_resized.flatten())
        y.append(class_name)

X = np.array(X)
y = np.array(y)


model = LogisticRegression(max_iter=1000, multi_class='ovr')
model.fit(X, y)


user_input = input("물고기 종류를 입력하세요: ")


user_input_resized = resize(user_input_image, (100, 100)).flatten()
predicted_class = model.predict([user_input_resized])[0]

print(f"입력한 물고기 종류는 {predicted_class} 입니다.")
```

- 로지스틱 회귀를 통한
- **물고기 분류 모델**

- 데이터 경로 설정
- 데이터 라벨링
- **로지스틱 모델 제작**
- 물고기 종류 입력받기
- 입력받은 종류 분류

Conclusion

선형 회귀 분석 및 로지스틱 회귀 분석의 방법을 이용해 물고기를 이미지로 분류하는 방식의 프로그램을 제작했다. 동시에 실제 장치에서는 물고기의 거리를 인식하는 프로그램과 함께 해당 물고기에 대한 사진을 이미지로 저장시킬 수 있도록 알고리즘을 구성하였다. 이러한 두 단계가 순차적으로 동작 시, 화면에 잡힌 물고기를 이미지로 저장 후, 물고기의 종을 확인한 뒤 선택적 포획이 가능하게 된다. 하지만 해양생물의 빠른 속도에 비해, 이미지 처리 및 동작이 짧은 순간내에 이뤄지기 어렵다는 점에서 부족한 점을 느꼈다. 또한 아두이노 프로그램으로 이미지화 시키는 것이 원초적으로 구현해내기 어렵다는 점에서 해결하는 데에 어려움을 느꼈다. 장치의 방수 기능 또한 지원하지 않는 보드 및 재료가 많다는 점에서 해수 표면이 아닌 내부에서 동작을 위해선 도옥 고가의 비용이 예상된다.
이러한 단점들이 잔류했음에도, 현재 해양 생물의 부수 어획 개체수가 굉장히 많고, 직접적으로 해결할 수 있는 방안이 제시되지 않는 현 시점에서 새로운 해결방법의 방향성을 제공할 수 있었다.