2022 년도 전자공학부 캡스톤디자인2 보고서

팀명	아무거나			
과제명	딥러닝을 이용한 캠퍼스 안내 애플리케이션			
구성원	학과(전공)	학번	이름	
	전자공학부	2017006126	김민상	
	전자공학부	2017007001	윤지환	
과제유형	HW ( ) SW (V) HW/SW ( )	지도교수	최명렬 (인)	
과제개요	직관적이고 흥미로운 캠퍼스 안내 애플리케이션을 제작하고자 했습니다. 딥러닝 기반의 Object Detection 모델을 사용하여 캠퍼스 내의 건물들을 인식하며, 가벼운 게임 컨셉트를 적용하여 사용자가 효과적으로 캠퍼스를 체험할 수 있도록 하고자 했습니다.			

1. 과제의 목표 및 개요

[Fig 1. 애플리케이션 초기 구상]

직관적이고 흥미를 유발하는 캠퍼스 안내 애플리케이션을 제작하는 것이 저희 팀의 목표입니다. 딥러닝 기반의 Object Detection 모델을 사용하여 캠퍼스 내의 건물들을 인식하며, 가벼운 게임 컨셉트를 적용하여 사용자가 효과적으로 캠퍼스를 체험할 수 있도록 하고자 했습니다. 별도로 추가 하드웨어 없이 스마트폰만으로 간단하게 사용

가능합니다. 본 애플리케이션을 통해 신입생이 학교 지리를 익히거나, 가이드가 없는 외국인 유학생들이나 첫 방문자들에게 직관적이고 흥미로운 가이드북이 되어줄 수 있습니다.

2. 과제의 필요성 및 독창성

[Fig 2. 국립중앙박물관 신라관 'AR 도슨트 서비스']

지금까지 AR 기반의 관광용 애플리케이션은 별도의 하드웨어가 각 오브젝트에 별도로 설치되거나 실내에서만 동작하는 문제점이 있었습니다. 본 과제에서는 검출하고자 하는 7개의 오브젝트들에 대해 별도로 라벨링 작업을 시행하고 딥러닝 모델을 학습하였습니다. 따라서 별도의 하드웨어 설치 없이 실외에서도 휴대용으로 동작하는 관광용 모바일 애플리케이션이 완성되었습니다.

또한 지금까지 신입생이나 재학생, 유학생, 외부인에게 캠퍼스 내부를 안내하는 오프라인 프로그램의 다양성과 안내 지도의 한계, 관련 애플리케이션들의 확장성 문제들을 해결할 수 있습니다.

[Fig 3. 에버랜드 어트랙션 스탬프 미션]

저희 팀은 기존의 캠퍼스를 소개하는 방식이 아닌 놀이동산의 어트랙션을 돌아다니며 스탬프를 모으는 것에서 아이디어를 얻어 캠퍼스를 돌아다니며 스탬프를 얻는 게임 방식으로 다른 캠퍼스 안내 프로그램과의 차별성을 두었습니다. 이에 사용자가 더 효과적으로 체험에 임할 수 있을 것이라 기대합니다.

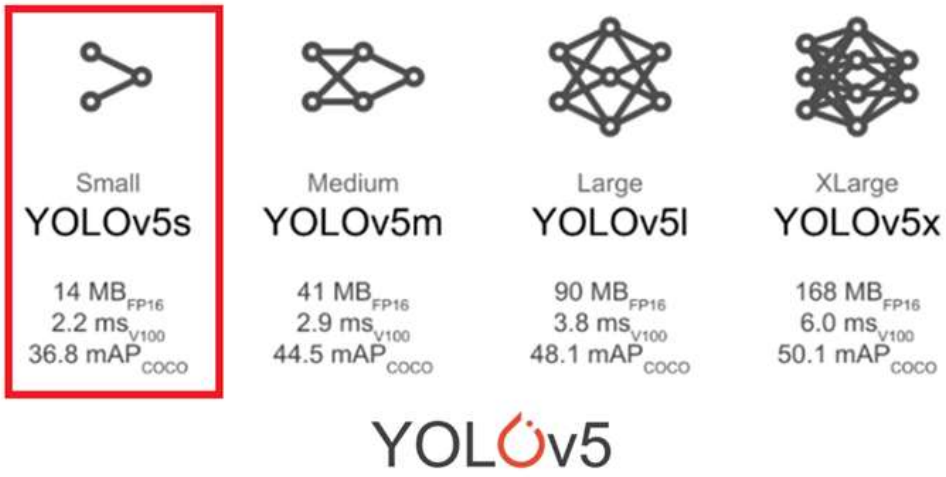
3. 캡스톤 디자인1에서의 변경점

지난 학기의 구상안에서는 딥러닝 모델 관련하여 'Unity+OpenCV' 패키지를 통해 영상처리 작업을 시행할 계획이었습니다. 하지만 모바일에서는 동작하지 않는다는 문제점을 확인하였고 대신 관련 알고리즘을 직접 제작하여 사용하는 방안으로 변경되었습니다. 학습 단계에서 사용될 'YOLOv5' 모델을 분석한 결과, 출력되는 값이 바로 사용 가능한 정제된 값이 아니라는 것을 알게 되었습니다. Python 개발 환경에서는 YOLO팀이 제공하는 NMS 알고리즘이 있지만, 최종 개발 환경인 Unity의 C# 환경에서는 NMS 관련 알고리즘이 제공되지 않아서 직접 알고리즘을 제작할 필요가 있었습니다. 또한 'Unity Barracuda' 패키지에서 모바일의 특성상 아직은 GPU보다는 CPU가 더 좋은 성능을 낼 수 있으므로 추론 단계에서는 CPU를 사용하게 되었습니다.

애플리케이션 개발 측면에서는 자세 인식, 평면 인식 등을 이용한 AR 컨텐츠를 제작하려 했지만, 개발 과정에서 예상했던 만큼의 성과가 나오지 않았고 간단한 게임 컨셉트를 주제로 확정을 지음으로써 애플리케이션의 볼륨을 줄이게 되었습니다. 개발 도중 온라인 플레이보다 개인의 사용성에 중요도를 두어 Node.js를 이용한 웹 서버 구축은 진행하지 않고 애플리케이션의 완성도를 높이는 것에 집중했습니다.

4. 진행 과정

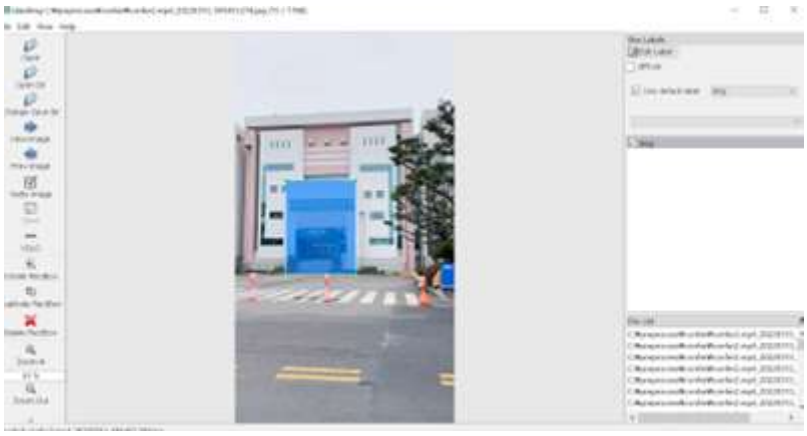
1) 딥러닝 모델 선정



[Fig 4. YOLOv5 모델 종류]

빠른 처리 속도로 실시간 검출이 가능한 One-Stage Detection 방식의 'YOLOv5s' 딥러닝 모델을 선정했습니다. YOLO 모델의 버전 중에서도 가벼우며 타 플랫폼으로의 확장성이 보장됩니다. 별도의 기기 대신 개인의 스마트폰을 이용하는 본 애플리케이션과 적합하다고 생각했습니다.

2) 데이터 셋 구성



[Fig 5. 'LabelImg' 프로그램을 이용한 라벨링 작업]

'LabelImg' 프로그램을 사용하여 해당 이미지에서 학습해야 하는 물체가 있는 좌표를 지정하는 라벨링 작업을 수행했습니다. 본 애플리케이션에 사용되는 Class는 하냥이 조각상, 본관, 컨퍼런스 홀, 셔틀콕, 아고라, 학술정보관, 복지관으로 7개입니다. Class마다 직접 촬영한 2000개 가량의 사진과 라벨링 데이터를 종합한 데이터 셋을 구성했습니다.

3) 딥러닝 모델 학습

[Fig 6. 학습 환경 플랫폼 Google Colab]

딥러닝 모델을 학습하기 위해 Google에서 제공하는 플랫폼인 'Colab'을 사용했습니다. 192 Epoch, 64 Batch size 등의 하이퍼 파라미터들을 사용하여 학습 커맨드를 작성했습니다.

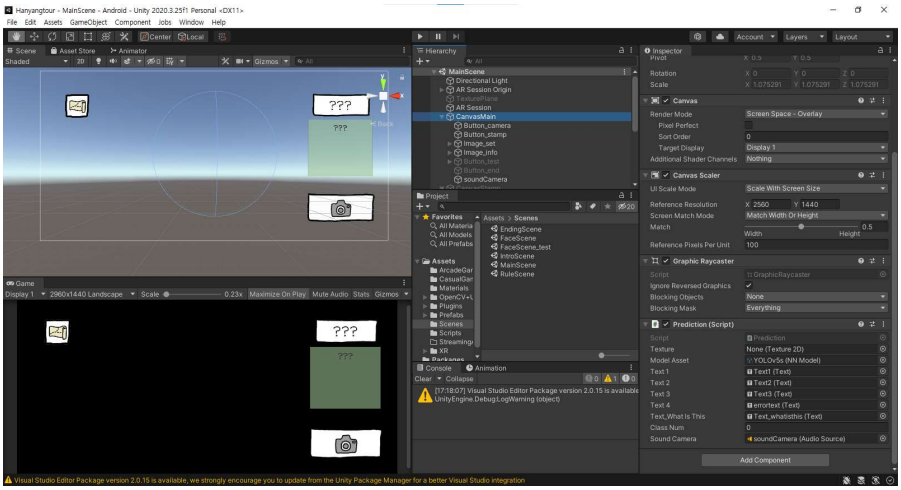
```
python /home/yunjihwan/Ahagi/Cap/yolov5/train.py --batch 64--epochs 250--data '/home/yunjihwan/Ahagi/Cap/dataset/customdata.
yaml' --cfg '/home/yunjihwan/Ahagi/Cap/yolov5/models/yolov5s.yaml' --weights yolov5s.pt --name sevenlable
```

[Fig 7. 학습 커맨드 설정]

```
                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16           [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     90880  models.common.C3                        [256, 128, 1, False]
 18                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  1    296448  models.common.C3                        [256, 256, 1, False]
 21                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  1   1182720  models.common.C3                        [512, 512, 1, False]
 24  [17, 20, 23]  1     32364  models.yolo.Detect                      [7, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 32
6, 512]]
Model Summary: 270 layers, 7038508 parameters, 7038508 gradients, 15.9 GFLOPs
```

[Fig 8. Model 구조]

4) 개발

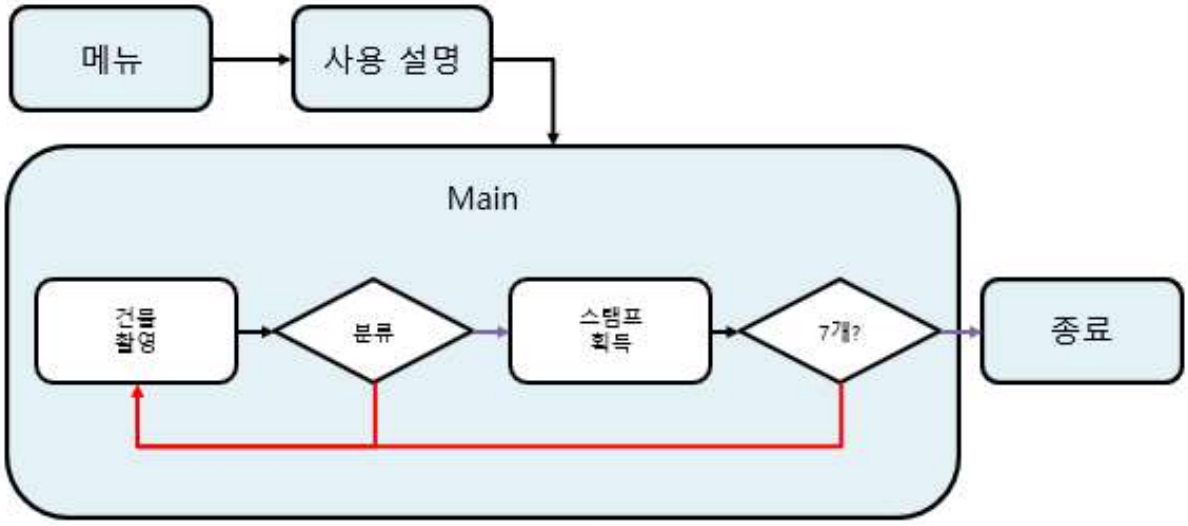


[Fig 9. 'Unity'를 이용한 애플리케이션 개발]

애플리케이션 개발 플랫폼으로 C# 언어를 사용하는 'Unity'를 채택하여 개발을 진행했습니다.

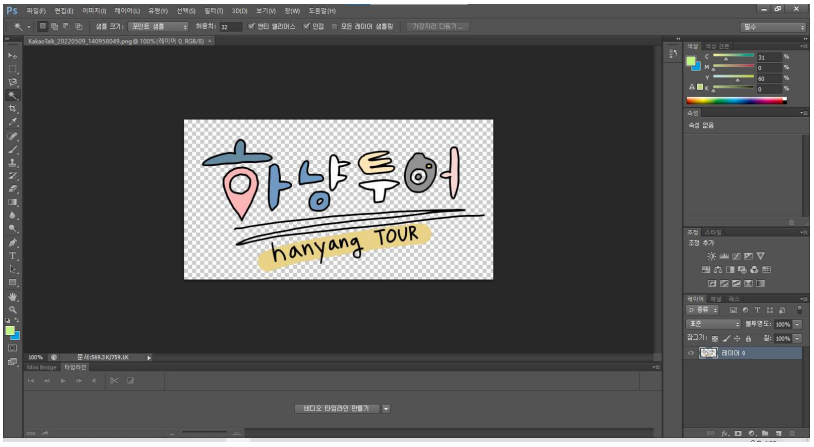
Unity에서 학습 모델을 활용하기 위해 적합한 파일 형식인 '.ONNX' 확장자로 변환하여 사용했습니다. 'ONNX'는 서로 다른 환경에서 만들어진 모델을 서로 호환하여 사용

할 수 있게 도와주는 인공지능 오픈소스 플랫폼입니다. Unity에서 onnx 파일을 사용하기 위해서는 추론을 보조하는 'Unity Barracuda' 라이브러리를 사용했습니다.



[Fig 10. 애플리케이션 동작 플로우 차트]

구체적인 애플리케이션 동작 플로우 차트입니다. 앞서 말씀드린 것처럼 캠퍼스 내의 건물들을 돌아다니며 스탬프를 얻으며 진행하는 것이 애플리케이션의 목적입니다. 각각의 상태마다 Scene 단위로 나누어 구성했으며, Scene 간의 이동은 Unity 버튼의 OnClick 기능으로 구현했으며, 사용자는 스마트폰 터치로 상호작용합니다. 먼저 애플리케이션을 시작하면 사용 방법에 대해 익히고 사용 가능합니다. 스마트폰의 카메라로 캠퍼스 내의 학습된 건물을 촬영하고 모델로 건물을 인식 후, 성공한다면 스탬프를 획득합니다. 이후, 7개의 스탬프를 모두 모을 때까지 반복한 후에 애플리케이션이 마무리됩니다. 모든 스탬프를 모으면 AR 하냥이 페이스 필터가 사용 가능합니다. AR 하냥이 페이스 필터에 대해서는 과제 결과 부분에서 다시 설명하겠습니다.

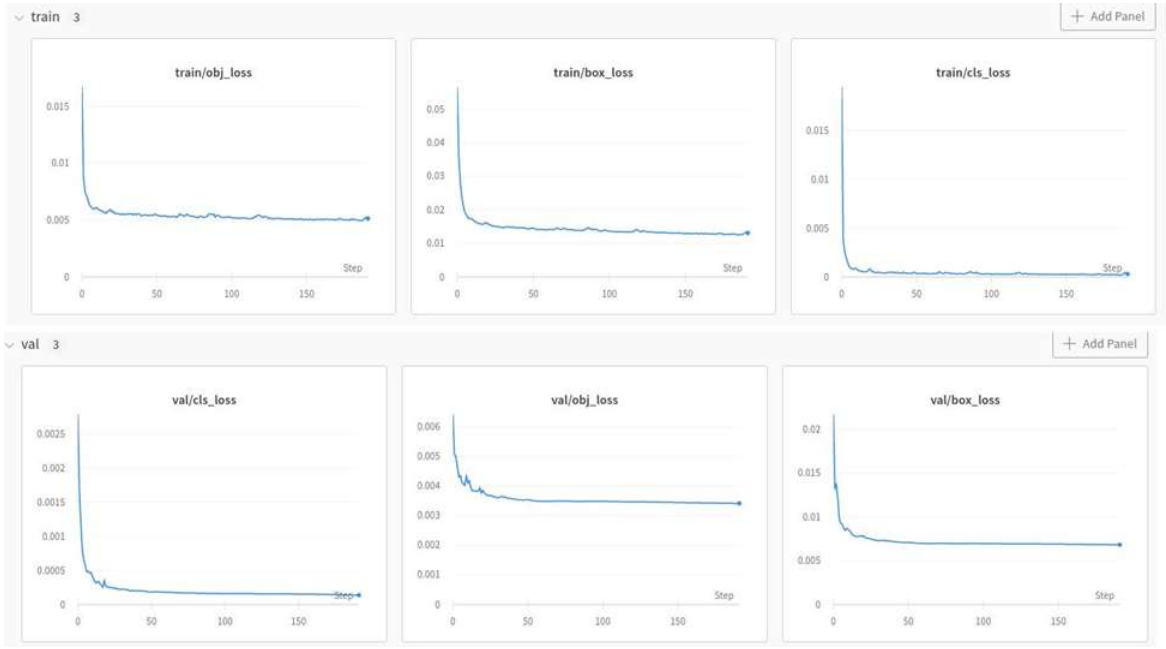


[Fig 11. 애플리케이션 UI 스케치]

애플리케이션에 사용되는 UI는 게임 컨셉트에 맞춰 직접 스케치한 것을 사용했고, 효과음과 배경 음악은 Unity Asset Store에서 오픈 소스 무료 에셋을 사용했습니다.

5. 과제 결과

1) YOLOv5s 모델 구축 성공



[Fig 12. YOLOv5s 모델 학습 결과: Loss]

[Fig 13. YOLOv5s 모델 학습 결과: Test]

설정한 7개의 class의 14000여 개의 데이터 셋을 학습시켜 테스트해 본 결과 분류한 class에 대해 준수한 성능을 보임을 확인했습니다. 구축한 모델을 .onnx 확장자로 변환

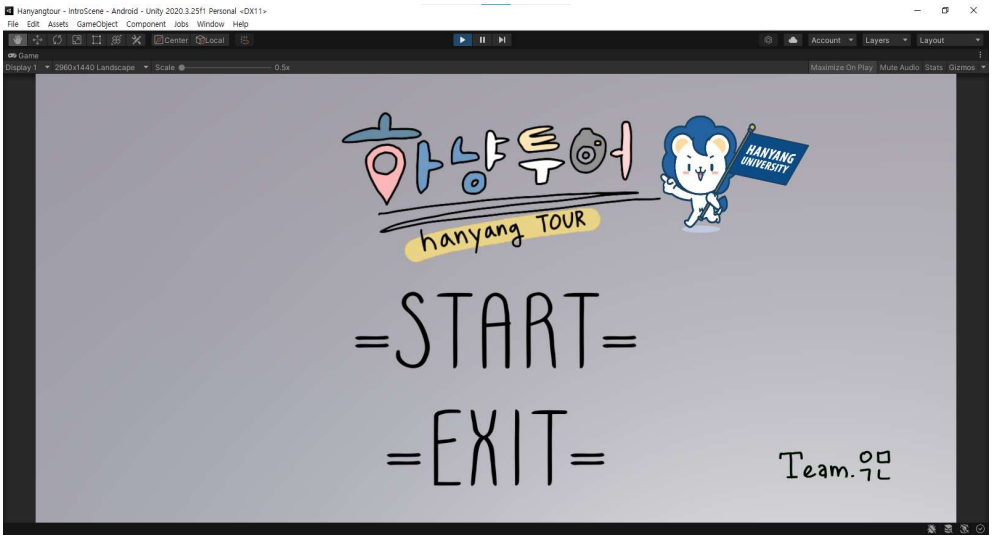
후 Unity에서 활용했습니다.

추가로 Unity C# 환경에서 인식률을 높이기 위해 C# 스크립트에서 예측 후 출력에 대한 후처리로 기존의 'NMS(Non-Maximum Suppression)' 대신에 간단하게 사용할 수 있는 Max Object Confidence 알고리즘을 설계하여 구현하였습니다.

```
for (int i = 0; i < 25200; i++)
{
    float temp1 = (float)OutputY[0, 0, 4, i];
    if (temp0 < temp1)
    {
        temp0 = temp1;

        indexofob2 = indexofob1;    //전전 존재확률의 최댓값의 인덱스 저장
        indexofob1 = indexofob;     // 직전 존재확률의 최댓값의 인덱스 저장
        indexofob = i;              //현재 존재확률의 최댓값의 인덱스 저장
    }
}

x0 = OutputY[0, 0, 0, indexofob];
x1 = OutputY[0, 0, 1, indexofob];
x2 = OutputY[0, 0, 2, indexofob];
x3 = OutputY[0, 0, 3, indexofob];

//NMS 구현
x4 = (OutputY[0, 0, 4, indexofob] + OutputY[0, 0, 4, indexofob1] + OutputY[0, 0, 4, indexofob2]) / 3;
x5 = (OutputY[0, 0, 5, indexofob] + OutputY[0, 0, 5, indexofob1] + OutputY[0, 0, 5, indexofob2]) / 3;
x6 = (OutputY[0, 0, 6, indexofob] + OutputY[0, 0, 6, indexofob1] + OutputY[0, 0, 6, indexofob2]) / 3;
x7 = (OutputY[0, 0, 7, indexofob] + OutputY[0, 0, 7, indexofob1] + OutputY[0, 0, 7, indexofob2]) / 3;
x8 = (OutputY[0, 0, 8, indexofob] + OutputY[0, 0, 8, indexofob1] + OutputY[0, 0, 8, indexofob2]) / 3;
x9 = (OutputY[0, 0, 9, indexofob] + OutputY[0, 0, 9, indexofob1] + OutputY[0, 0, 9, indexofob2]) / 3;
x10 = (OutputY[0, 0, 10, indexofob] + OutputY[0, 0, 10, indexofob1] + OutputY[0, 0, 10, indexofob2]) / 3;
x11 = (OutputY[0, 0, 11, indexofob] + OutputY[0, 0, 11, indexofob1] + OutputY[0, 0, 11, indexofob2]) / 3;
//['hanayang','mainhall','conference','shuttle','agora','library','welfare']
```

[Fig 14. 'Prediction.c'의 NMS 알고리즘]

2) Unity 애플리케이션 개발

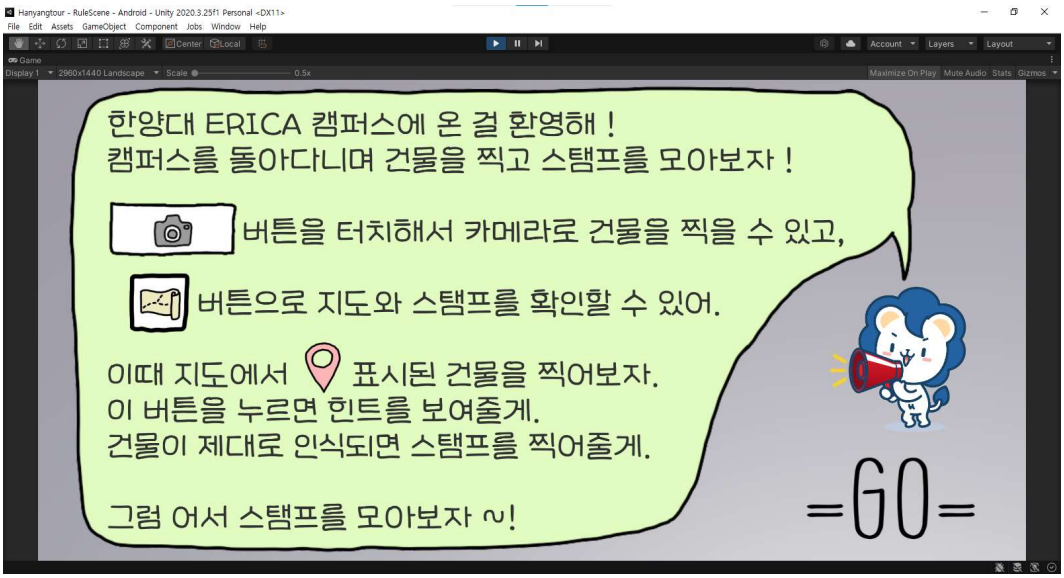


[Fig 15. 'IntroScene']

먼저 메뉴 장면이 되는 'IntroScene'입니다. 프로젝트 이름과 팀 로고가 주가 되게 구성했으며, 시작 버튼과 종료 버튼을 추가하여 사용자가 게임을 시작할 수 있습니다. 모든 Scene에서 Scene 간의 이동은 'UnityEngine.SceneManagement' 라이브러리를 사용하여 버튼으로 Scene 전환을 할 수 있도록 'SceneController.c'를 구성했습니다.

```
3     using UnityEngine;
4     using UnityEngine.SceneManagement;
5
6     public class SceneController : MonoBehaviour
7     {
8         public AudioSource SoundButton;
9
10        public void ButtonToIntro()
11        {
12            SceneManager.LoadScene("IntroScene");
13            SoundButton.Play();
14        }
15
16        public void ButtonToMain()
17        {
18            SceneManager.LoadScene("MainScene");
19            SoundButton.Play();
20        }
```

[Fig 16. 'SceneController.c'의 SceneManagement 라이브러리 사용]



[Fig 17. 'RuleScene']

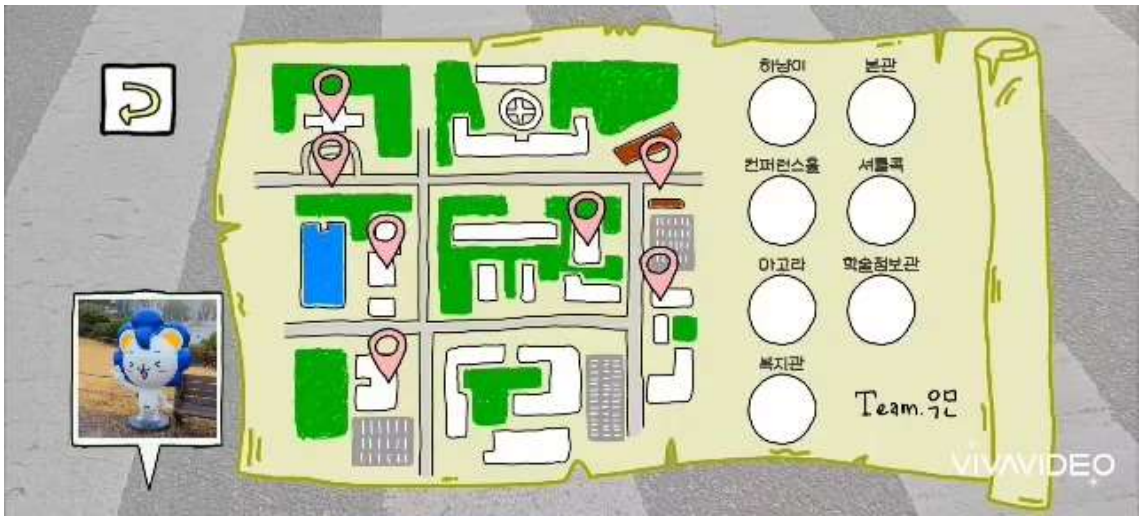
게임을 시작하기 전에 본 게임의 규칙을 설명해주는 장면인 'RuleScene'입니다. 게임의 전체적인 규칙을 설명합니다.

[Fig 18. 'MainScene' : 카메라 - 아고라 인식]

[Fig 19. 'MainScene' : 카메라 – 본관 인식]

다음은 저희 프로그램의 메인이 되는 'MainScene'입니다. 스마트폰의 카메라를 이용해 촬영할 수 있고, 촬영할 때마다 사전에 구축한 모델로 객체를 추론하여 인식합니다. 객체 인식이 성공적으로 진행된다면, 객체의 이름과 입력한 정보가 우측에 나타납니다. 객체 인식에 대한 정보를 나타내는 과정은 'InfoController.c'를 통해 구성했습니다. 그리고 성공한 객체에 대한 스탬프가 지도에 찍힙니다. 이와 같은 과정을 반복하여 7개의 건물에 대한 스탬프를 얻으면 게임 종료를 진행할 수 있습니다.

사전에 구축한 YOLOv5s 모델을 활용하여 'Prediction.c' 스크립트를 프로그래밍한 결과 준수한 성능으로 애플리케이션에서도 사용 가능함을 확인하였습니다.



[Fig 20. 'MainScene' : 스탬프]

'MainScene'에서 좌측 상단의 지도 버튼을 클릭하면 위와 같은 스탬프 이미지를 확인할 수 있습니다. 대략적인 캠퍼스 지도와 인식 가능한 장소, 스탬프를 나타내었으며. 장소에 대한 인식 예시가 힌트로 좌측에 이미지로 나타납니다. 힌트를 나타내기 위해 'HintController.c'를 작성했습니다. 그리고 스탬프가 찍히는 것을 구현하기 위한 스크립트 'StampController.c'를 작성했고, 앞서 'Prediction.c'에서 인식이 완료된 객체에 대한 변수를 받아오는 방식으로 StampOn함수를 구성했습니다.

```
37      void Update()
38      {
39          Prediction pred = canvasMain.GetComponent<Prediction>();
40          stampNum = pred.classNum;
41
42          StampOn();
43      }
44
45      public void ButtonToStamp()...
51
52      public void ButtonToBack()...
58
59      private void StampOn()
60      {
61          if (stampNum == 1) { check01.SetActive(true); flag1 = true; }
62          else if (stampNum == 2) { check02.SetActive(true); flag2 = true; }
63          else if (stampNum == 3) { check03.SetActive(true); flag3 = true; }
64          else if (stampNum == 4) { check04.SetActive(true); flag4 = true; }
65          else if (stampNum == 5) { check05.SetActive(true); flag5 = true; }
66          else if (stampNum == 6) { check06.SetActive(true); flag6 = true; }
67          else if (stampNum == 7) { check07.SetActive(true); flag7 = true; }
```

[Fig 21. 'StampController.c'의 StampOn 함수]

[Fig 22. 'EndingScene']

7개의 스탬프를 모두 모으면 게임이 마무리 단계에 들어갑니다. 게임을 종료하거나 AR 하냥이 페이스 필터를 사용할 수 있습니다.

[Fig 23. 'AR 하냥이 페이스 필터']

AR 하냥이 페이스 필터는 'ARCore' 라이브러리를 이용하여 사용자의 얼굴을 인식합니다. 인식한 얼굴 위에 하냥이 얼굴을 올려 사진을 찍을 수 있는 기능입니다. 찍은 사진은 저장할 수 있습니다.

6. 토의 및 개선 방향

캡스톤 과제 수행 결과 딥러닝 모델에 대해 자세하게 분석하고 어떤 방식으로 모델이 학습하는지, 성능을 높이기 위해 어떻게 해야 하는지 등 학습 과정에서 필요한 통찰력을 얻게 되었습니다. 또한 모델을 지원하기 위해 사용되는 수학적 알고리즘을 직접 제작하여 프로그래밍에 대해 친숙해졌습니다.

본 애플리케이션을 통해 시사하는 점은 신입생이나 외부 방문객들을 위한 효과적인 가이드가 되어 캠퍼스를 소개하는 데에 이바지할 수 있다고 기대합니다.

향후 목적에 맞게 딥러닝 학습 클래스를 증가시켜 사용자가 캠퍼스 내에서 더 다양한 경험을 할 수 있도록 할 수 있습니다. 또한 한국어 이외에 영어, 일어, 중국어와 같이 다양한 언어를 지원하여 외국인 유학생과 같은 다양한 사용자들에게도 접근성을 높일 수 있습니다. 궁극적으로 캠퍼스 투어 목적으로만 사용하는 것이 아닌 박물관에서 문화재를 인식하는 용도, 관광지에서 랜드마크를 인식하는 용도 등등 여러 분야에서 다양한 방식으로 응용 가능하다고 기대합니다. 현재 본 애플리케이션은 안드로이드 환경에서만 구동할 수 있습니다. iOS 환경에서도 구축하여 다양한 스마트폰에서 사용 가능할 것으로 기대합니다.

7. 참고문헌 및 부록

YOLOv5 Github : github.com/ultralytics/yolov5
Unity Barracuda Gitgub : github.com/Unity-Technologies/barracuda-release
OpenCV plus Unity Asset:
https://assetstore.unity.com/packages/tools/integration/opencv-plus-unity-85928
효과음 Asset :
https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116
배경 음악 Asset :
https://assetstore.unity.com/packages/audio/music/arcade-game-bgm-17-210775
하냥이 사용 매뉴얼 :
https://blog.naver.com/hyerica4473/222498353516