

엣지/클라우드 컴퓨팅 기반 AWS DeepLens 를 이용한 모션 인식 및 실시간 스트리밍

경희대학교 최지우 이민규

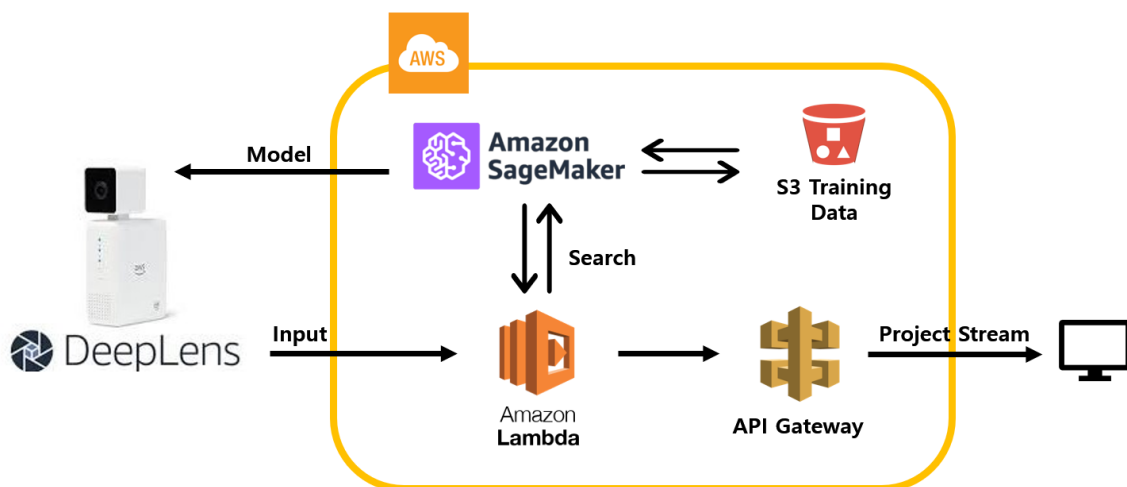
개발 기간	2020.09 ~2020.11
개발 환경	Python / AWS DeepLens, SageMaker, S3, Lambda, Kinesis
개발 인원	2 명
기능	1) 요가 자세 인식 및 정확도 출력 2) 실시간 영상 스트리밍 3) Edge/Cloud Computing 을 이용하여 AWS DeepLens 단말기만으로 딥러닝 학습 및 추론, 실시간 스트리밍이 전부 가능하도록 구현

A. Abstract

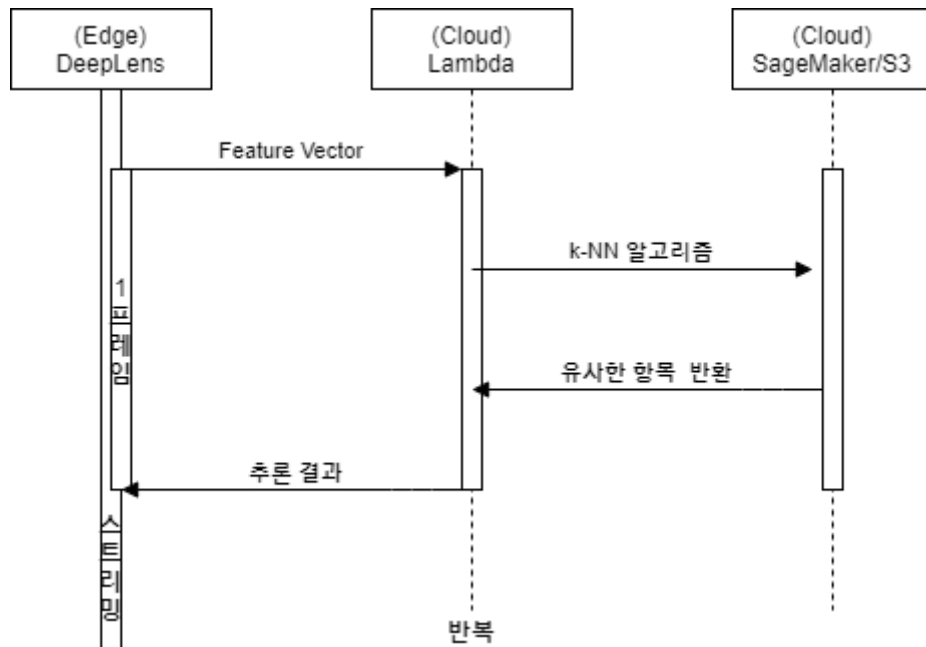
최근 세계적으로 다양한 분야에서 비대면 커뮤니케이션으로의 전환이 이루어지고 있다. 하지만 직접 몸을 움직이거나 자세 교정이 필요한 실내 운동의 강좌는 비대면으로의 전환이 어려운 상황이다. 따라서 이 연구는 Edge 노드로 활용되고 있는 AWS DeepLens 를 기반으로 실내 운동 모션의 인식과 실시간 스트리밍을 통해 강 의자와 수강생의 커뮤니케이션이 가능하도록 하는 서비스를 제안하고 구현한다.

B. Project Architecture

A. Architecture Diagram



B. Scenario



C. 구현

A. 이미지 크롤링 및 데이터 세트 가공

딥러닝 모델 학습을 위해 가장 먼저 필요한 것은 이미지 데이터 세트이다. 이미지 수집에는 google_images_download 라이브러리를 사용하였으며, Python 스크립트를 작성하여 이미지 크롤링을 진행하였다. 분류하고자 하는 각 라벨마다 약 200 장의 이미지를 수집하였다.

정교한 모델 학습을 위해 데이터 세트가 부족한 경우에는 이미지 변형을 통한 가공이 필요하다. 아래 코드와 같은 MXNet 을 이용한 Keras 라이브러리를 통해 이미지 데이터 세트 가공을 진행하였으며, 이미지 반전, 회전 등을 통해 각 라벨 별 약 650 장의 Train set 와 약 200 장의 Validation set 를 생성하여 대략 75:25 의 비율로 데이터 세트를 마련하였다.

이로써 모델 학습을 위한 이미지 데이터 세트가 마련되었으며, 데이터는 Train/Validation set 유형과 Label 분류에 맞춰 적절한 저장 경로를 설정해준다.

```

import numpy as np
import os
np.random.seed(3)
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array,
load_img
optInputPath = '~/yoga/'
optOutputPath = '~/preview/'

# 이미지 크기 조정 비율
optRescale = 1./255
# 이미지 회전
optRotationRange=10
# 이미지 수평 이동
optWidthShiftRange=0.2
# 이미지 수직 이동
optHeightShiftRange=0.2
# 이미지 밀림 강도
optShearRange=0.5
# 이미지 확대/축소
optZoomRange=[0.9,2.2]
# 이미지 수평 뒤집기
optHorizontalFlip = True
# 이미지 수직 뒤집기
optVerticalFlip = False
optFillMode='nearest'
# 이미지당 늘리는 갯수
optNbrOfIncreasePerPic = 3
# 배치 수
optNbrOfBatchPerPic = 2
train_datagen = ImageDataGenerator(rescale=optRescale,
                                    rotation_range=optRotationRange,
                                    width_shift_range=optWidthShiftRange,
                                    height_shift_range=optHeightShiftRange,
                                    shear_range=optShearRange,
                                    zoom_range=optZoomRange,
                                    horizontal_flip=optHorizontalFlip,
                                    vertical_flip=optVerticalFlip,
                                    fill_mode=optFillMode)

def checkFolder(path):
    try:
        if not(os.path.isdir(path)):
            os.makedirs(os.path.join(path))
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise

def increaseImage(path , folder):

```

```

for index in range(0, optNbrOfIncreasePerPic):
    img = load_img(path)
    x = img_to_array(img)
    x = x.reshape((1,) + x.shape)
    i = 0
    checkFoler(optOutputPath+folder)
    print('index : ' + str(index))
    for batch in train_datagen.flow(x, batch_size=1,
save_to_dir=optOutputPath+folder, save_prefix='tri', save_format='jpg'):
        i += 1
        print(folder + " " + str(i))
        if i >= optNbrOfBatchPerPic:
            break

# 이미지 생성기
def generator(dirName):
    checkFoler(optOutputPath)
    try:
        fileNames = os.listdir(dirName)
        for fileName in fileNames:
            fullFileName = os.path.join(dirName, fileName)
            if os.path.isdir(fullFileName):
                generator(fullFileName)
            else:
                ext = os.path.splitext(fullFileName)[-1]
                folderName = os.path.splitext(fullFileName)[0].split('/')[ -2]
                if(ext == '.jpg'):
                    increaseImage(fullFileName, folderName)

    except PermissionError:
        pass
if __name__ == "__main__":
    generator(optInputPath)

```

3.2. Amazon SageMaker 를 통한 모델 학습

Amazon SageMaker 의 다양한 예제 코드 중 Image-Classification-FullTraining 을 사용한다. 이는 ResNet 네트워크를 사용하여 훈련되며, 객체를 식별할 수 있는 객체 분류 모델을 빌드한다.

먼저 데이터셋 가공을 통해 얻은 이미지 데이터 세트를 호스팅하는 버킷을 AWS S3 에서 정의한다. 버킷 이름에는 'deeplens'가 포함되어야 한다. 이 예제 코드에서는

Amazon SageMaker 의 기본 이미지 분류모델인 ResNet 을 사용하기 때문에, 이미지 데이터 세트를 .rec 형태의 확장자로의 변경이 필요하다. 이미지 파일을 rec 파일로 변환하기 위해서는 MXNet tool 함수 중 im2rec.py 를 사용한다. 이후 rec 파일을 경로에 맞춰 버킷에 업로드한다.

아래의 코드는 이미지 분류 모델 학습 프로세스 제어에 사용되는 하이퍼 파라미터 목록이다. num_layers 를 통해 네트워크 깊이를 정의할 수 있다. ResNet 은 18, 34, 50 등 여러 네트워크 깊이를 지원하며, 초기값인 18 을 사용한다. image_shape 는 입력 이미지의 크기이며, 224*224 크기의 3 가지 색상(RGB)을 사용한다. num_training_samples 는 훈련할 데이터의 샘플 수로, 약 800 장의 이미지를 사용하였다. num_classes 는 모델의 출력 클래스 수로, 분류할 두 가지 자세와 '그 외의 이미지'를 분류하기 위해 3으로 설정한다. mini_batch_size 는 한 번 반복에서 사용되는 학습 예제의 수이며, epochs 는 전체 데이터 세트가 네트워크에서 반복 처리될 횟수이다.

이후 코드에서 Amazon SageMaker 에서 모델을 훈련하기 위해 훈련 작업(JOB)이 생성된다. JOB 에는 이미지 데이터를 저장한 Amazon S3 버킷의 URL 과, 모델 교육에 사용할 컴퓨팅 리소스, 출력을 저장할 S3 버킷의 URL 정보가 포함된다. 모델 교육에 사용할 컴퓨팅 리소스의 기본 설정이 클라우드에서 가장 큰 GPU 지원 가상 머신인 p2.xlarge 이기 때문에, 일반 사용자는 문의를 통해 인스턴스 제한 변경을 요청하여야 한다. 작업 생성이 완료되면 CloudWatch 에서 로그 기록을 통해 학습 결과를 볼 수 있다. 이를 통해 AWS DeepLens 에 직접 가져올 수 있는 모델이 생성된다.

```
# 알고리즘에 사용되는 네트워크 깊이
num_layers = "18"
# 이미지 크기
image_shape = "3,224,224"
# 학습 이미지 샘플 수
num_training_samples = "1100"
# 분류할 라벨 수
num_classes = "3"
# 배치 사이즈
mini_batch_size = "64"
# 반복 수
epochs = "3"
# learning rate
learning_rate = "0.01"
```

SageMaker 예 JOB 생성

```
sagemaker = boto3.client(service_name='sagemaker')
sagemaker.create_training_job(**training_params)

status =
sagemaker.describe_training_job(TrainingJobName=job_name) ['TrainingJobStatus']
print('Training job current status: {}'.format(status))

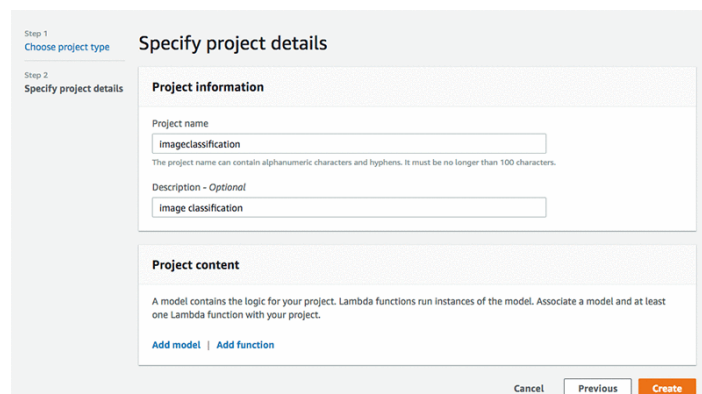
try:
sagemaker.get_waiter('training_job_completed_or_stopped').wait(TrainingJobName=job_
name)
    training_info = sagemaker.describe_training_job(TrainingJobName=job_name)
    status = training_info['TrainingJobStatus']
    print("Training job ended with status: " + status)
except:
    print('Training failed to start')
    message =
sagemaker.describe_training_job(TrainingJobName=job_name) ['FailureReason']
    print('Training failed with the following error: {}'.format(message))

training_info = sagemaker.describe_training_job(TrainingJobName=job_name)
status = training_info['TrainingJobStatus']
print("Training job ended with status: " + status)
```

3.3. DeepLens 모델 배포 및 Lambda 함수 설정

AWS DeepLens 콘솔에서 Amazon SageMaker 를 사용하여 훈련된 모델을 직접 가져오는 기능을 사용한다. DeepLens 버전이 1.2.3 이상 및 MXNet 버전 0.12 로 업데이트가 필요하다.

AWS DeepLens 기기를 콘솔에서 등록하는 작업이 필요하다. AWS 콘솔에서 설명하는 절차를 따라 등록이 완료되면, 새로운 빈 프로젝트를 생성한다.



[그림 5] AWS DeepLens Project 생성

위 그림과 같이 프로젝트 이름을 설정하고 Project content 에서 Add model 을 클릭하여 Amazon SageMaker trained model 을 선택하고, 이전의 과정에서 학습한 모델을 선택하여 가져온다. 모델이 추가되면 Add Function 를 선택하여 Lambda 함수를 추가하여야 한다. 아래는 AWS Lambda 콘솔을 사용하여 추론 Lambda 함수를 생성하는 과정이다.

딥러닝 모델에 대한 추론을 실행하는 Lambda 함수를 생성하기 위해 Create Function 을 진행하고, 블루프린트 사용을 선택하여 샘플 코드를 이용한다. greengrass-hello-world 샘플 코드를 사용하였으며, Lambda 함수에 모델과 동일한 이름을 지정한다. 함수에 대한 권한을 정의하는 역할 지정에서 AWSDeepLensLambdaRole 기본 IAM 역할을 선택한다. 함수 생성이 완료되면 핸들러가 greengrassHelloWorld.function_handler 인지 확인하여야 한다. 이후 GreengrassHello 파일에서 모든 코드를 제거하고 추론 Lambda 함수에 대한 코드를 작성한다. Lambda 함수 작성이 완료되면 저장하여 배포, 게시한다. 이 Lambda 함수를 DeepLens 프로젝트에서 선택하여 추가하면 프로젝트 생성이 완료된다.

이로써 Edge 기기(DeepLens)에서 생성된 데이터를 이용하여 Cloud 서비스에서의 Serverless 한 함수 실행을 통한 딥러닝 모델 추론이 가능해진다.

```
while True:
    # 스트리밍 마지막 프레임 가져오기
    ret, frame = awscam.getLastFrame()
    if not ret:
        raise Exception('Failed to get frame from the stream')
    # 인식 범위를 위한 프레임 크롭
    frame_crop =
frame[
                                int(frame.shape[0]/2-
region_size/2):int(frame.shape[0]/2+region_size/2), \ int(frame.shape[1]/2-
region_size/2):int(frame.shape[1]/2+region_size/2), :]

    # 학습 데이터에 맞춘 프레임 크기 변경
    frame_resize = cv2.resize(frame_crop, (input_height, input_width))
    frame_resize = cv2.cvtColor(frame_resize, cv2.COLOR_BGR2RGB)
    parsed_inference_results = model.parseResult(model_type,
model.doInference(frame_resize))

    # 일치도가 가장 높은 라벨
    top_k = parsed_inference_results[model_type][0:num_top_k]
```

```

# 프레임 복사
overlay = frame.copy()
cv2.rectangle(overlay, (0, 0),
               \ (int(label_region_width),int(label_region_height)), (211,211,211), -1)
opacity = 0.7
cv2.addWeighted(overlay, opacity, frame, 1 - opacity, 0, frame)

# 가장 높은 일치도를 가진 라벨에 따라 출력되는 메시지 변경
if top_k[0]['prob']*100 < 60 :
    cv2.putText(frame, 'Take your pose', (0,50), cv2.FONT_HERSHEY_SIMPLEX,
                1.5, (255, 0, 0), 3)
else :
    cv2.putText(frame, output_map[top_k[0]['label']] + ' ' +
str(round(top_k[0]['prob'], 3) * 100) + '%', \ (0, 150), cv2.FONT_HERSHEY_SIMPLEX, 1.5,
(255, 0, 0), 3)

# 인식 범위 화면에 출력
cv2.rectangle(frame, (int(frame.shape[1]/2-region_size/2), int(frame.shape[0]/2-
region_size/2)), \
(int(frame.shape[1]/2+region_size/2), int(frame.shape[0]/2+region_size/2)), (255,0,0),
5)
local_display.set_frame_data(frame)
# MQTT 를 통해 IOT 로그 남김
cloud_output = {}
for obj in top_k:
    cloud_output[output_map[obj['label']]] = obj['prob']
    client.publish(topic=iot_topic, payload=json.dumps(cloud_output))
except Exception as ex:
    client.publish(topic=iot_topic, payload='Error : {}'.format(ex))

```

3.4. Live Streaming

프로젝트가 생성되면, DeepLens 콘솔의 Device 목록에서 등록된 기기를 선택하여 프로젝트를 Deploy 할 수 있다. Review 와 Deploy 가 완료되면, DeepLens 콘솔에서 Streaming 화면을 확인할 수 있다. AWS DeepLens 에는 Amazon Kinesis 비디오 스트림 통합을 위한 DeepLens_Kinesis_Video 모듈이 포함된다. 이 Python 모듈을 사용하면 DeepLens 기기에서 Kinesis 로 비디오 피드를 보내고 스트리밍을 제어할 수 있다. 모듈은 DeepLens 프로젝트에서 배포된 Lambda 함수에서도 호출할 수 있으며, 이를 통해 웹에서 미디어 데이터 스트리밍 API 를 구현하여 인터넷 접속을 통해 실시간 스트리밍 영상을 볼 수 있도록 구현할 수 있다.

D. 결론

A. 연구 결과

구현했던 시스템의 초기 버전에는 문제점이 몇 가지 존재했다. 모델 추론의 정확도가 다소 떨어진다는 점과 스트리밍의 지연시간이 4~5 초정도로 상당한 시간의 지연이 생기는 것과 초당 프레임 수가 3~4 회로 매우 낮다는 점이다. 이러한 문제점을 바로잡기 위해 이미지 데이터와 코드를 조금씩 수정하여 수십번의 빌드를 거쳤다.

1) 모델 추론 정확도 향상

초기 버전에는 약 200 장의 이미지를 변형시켜 800 장 이상의 데이터 세트를 준비하였지만, 변형 과정에서 지나치게 회전하거나 이동된 이미지들로 인해 모델의 정확도가 크게 떨어졌다. 또한 선택한 자세 중 하나의 모양이 일반적으로 사람이 서 있는 모습과 비슷하여 높은 일치도를 출력하였다. 이를 해결하기 위해 이미지 변형률을 대폭 낮추어 진행하였으며, 새로운 라벨을 추가하고 더 나은 모델 학습을 위해 epochs 를 증가시켜 학습을 여러 번 진행하였다. 결론적으로 초기 버전보다는 나은 정확도 출력을 나타냈지만, 여전히 배경과 주위 환경에 따라 일치도 출력이 변화하는 모습을 보인다. 이는 추후 Background Substraction 등의 방법을 이용하여 개선이 필요하다.

2) 스트리밍 지연시간과 초당 프레임 수와 관한 성능 향상

클라우드를 이용한 작업의 특성 상, 지연시간에 대한 문제를 완전하게 공략하기는 쉽지 않았다. 더군다나 DeepLens 가 서울 리전에 대한 서비스를 지원하지 않기 때문에, S3 스토리지와 Lambda 함수를 포함하여 모든 작업을 도쿄 리전에서 진행하였다. 초기 버전에서는 약 4~5 초의 지연시간과 초당 3~4 프레임을 나타내었다. 이를 개선하기 위해 코드를 최대한 간소화하고, 화질과 이미지 형식의 변경이 필요하였다. Lambda 함수에 화질을 설정하는 코드를 추가하여 480p 를 사용한 스트리밍을 출력하였고, 이미지 데이터 타입을 단정밀도(FP32)에서 반정밀도(FP16)로 변경하여 메모리 사용을 줄이고 데이터 전송을 빠르게 만들었다. 최종적으로 지연시간은

약 2 초 정도로 개선되었으며, 초당 프레임 수 또한 8 이상으로 증가하여 개선 효과를 얻어내었다.

3) 사용자와 상호작용위한 UI 변경

초기 모델의 인터페이스는 위 그림과 같이 아무 자세도 취하지 않으면 어떤 출력도 없고, 자세를 취하면 그 자세와 일치하는 라벨과 일치도만을 출력하는 방식을 사용했다. 최종적으로 빌드한 모델에서는 아래와 같이 사용자와 상호작용할 수 있도록 유저친화적인 메시지를 통해 아무 자세를 취하지 않았을 때는 “Take your pose”라는 메시지를 출력하며, 중앙에 자세를 인식할 수 있는 사각 범위를 설정하여 사용자가 자세를 취해야할 공간적 위치를 알려준다.



[그림] Tree 자세와 Plank 자세 시연

B. 성능 평가

마지막 빌드에서 사용한 모델과 Lambda 함수는 이전보다 개선된 결과를 출력하였다. 배경으로 인한 간섭으로 크게 변화하던 정확도는 자세를 취했을 때 높은 일치도를 출력하였으며, 정확한 라벨 추론이 가능해졌다. 스트리밍의 지연시간 또한 상당히 개선되어 약 2 초 미만 정도로 개선되었고, 초당 8~9 프레임의 영상을

출력하였다. 또한 사용자 인터페이스 환경을 개선하여 인식 가능한 사각 범위를 보여주고, 자세를 취하지 않았을 때는 자세를 취해달라는 메시지를 출력한다.

C. 결론

그림과 같이 자세를 취했을 때, 자세의 일치도가 상단에 표시된 화면이 실시간으로 스트리밍 된다. 따라서 자신이 취해야 할 자세를 바로 피드백 받을 수 있다. 이를 바탕으로, 프로젝트를 확장하여 COVID-19로 인해 비대면 수업 및 교육이 권장되고 있는 상황에서 비대면 수업에 제한사항이 있는 태권도, 요가와 같은 트레이닝 수업의 질을 향상시킬 수 있다. 수강생은 Edge 노드인 DeepLens를 이용한 스트리밍으로 강 의자와 실시간 소통이 가능하며, 학습되어진 운동 자세 데이터를 바탕으로 즉각적인 자세 피드백을 받을 수 있게 되는 것이다.

D. 추후 연구

시스템 초기 버전에 비해서는 모델 추론의 정확도와 스트리밍 지연 시간, 초당 프레임 수가 개선되었지만, 실시간 처리가 중요한 서비스인 만큼 더욱 개선해야 할 필요성이 존재한다. 모델 추론의 정확도는 Background Subtraction 알고리즘을 이용하여 개선시키는 것을 고려한다. 또한 스트리밍에 관해서는 로그를 남기는 코드와 같은 불필요한 부분을 제거하거나, DeepLens와 통신하는 SageMaker Job을 저장하는 S3 스토리지 혹은 실행되는 Lambda 함수를 서버 리전에서 실행시키는 방법을 고려하는 것이 필요하다.

이러한 시스템은 추후 다양한 서비스에서 활용될 수 있다. 요가, 태권도 강의 뿐만 아니라 실시간 처리가 중요한 모든 딥러닝/스트리밍 시스템에서 모델 추론의 정확도를 높이고, 스트리밍의 품질을 개선시킬 수 있는 방법을 제의할 수 있다.

E. 참고 문헌

- [1] AWS DeepLens : <https://aws.amazon.com/deeplens/>
- [2] DeepLensCommunity Project: <https://aws.amazon.com/deeplens/community-projects/>
- [3] AWS DeepLens 설명서 : <https://docs.aws.amazon.com/deeplens/index.html>
- [4] 강태욱, 인공지능 비전 장치 아마존 딥렌즈 (2019/7, AD&Graphics)