KIPS Trans. Softw. and Data Eng.
Vol.9, No.6 pp.177~186 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2020.9.6.177

An Efficient Deep Learning Based Image Recognition Service System Using AWS Lambda Serverless Computing Technology

Hyunchul Lee† · Sungmin Lee†† · Kangseok Kim†††

ABSTRACT

Recent advances in deep learning technology have improved image recognition performance in the field of computer vision, and serverless computing is emerging as the next generation cloud computing technology for event-based cloud application development and services. Attempts to use deep learning and serverless computing technology to increase the number of real-world image recognition services are increasing. Therefore, this paper describes how to develop an efficient deep learning based image recognition service system using serverless computing technology. The proposed system suggests a method that can serve large neural network model to users at low cost by using AWS Lambda Server based on serverless computing. We also show that we can effectively build a serverless computing system that uses a large neural network model by addressing the shortcomings of AWS Lambda Server, cold start time and capacity limitation. Through experiments, we confirmed that the proposed system, using AWS Lambda Serverless Computing technology, is efficient for servicing large neural network models by solving processing time and capacity limitations as well as cost reduction.

Keywords : Deep Learning, Serverless Computing, AWS Lambda Server, Cold Start Time, Capacity Limitation

AWS Lambda Serverless Computing 기술을 활용한 효율적인 딥러닝 기반 이미지 인식 서비스 시스템

이 현 철† · 이 성 민†† · 김 강 석†††

요 약

최근 딥러닝(Deep Learning) 기술의 발전에 따라 컴퓨터 비전(Computer Vision) 분야의 이미지 인식 성능이 향상되고 있으며, 또한 Serverless Computing이 이벤트 기반의 클라우드 애플리케이션 개발 및 서비스를 위한 차세대 클라우드 컴퓨팅 기술로 각광받고 있어 딥러닝과 Serverless Computing 기술을 접목하여 실생활에 이미지 인식 서비스를 사용하고자 하는 시도가 증가하고 있다. 따라서 본 논문에서는 Serverless Computing 기술을 활용하여 효율적인 딥러닝 기반 이미지 인식 서비스 시스템 개발 방법을 기술한다. 제안하는 시스템은 Serverless Computing 기반 AWS Lambda Server를 이용하여 적은 비용으로 대형 신경망 모델을 사용자에게 서비스할 수 있는 방법을 제안한다. 또한 AWS Lambda Server의 단점인 Cold Start Time 문제와 용량제한 문제를 해결하여 효과적으로 대형 신경망 모델을 사용하는 Serverless Computing 시스템을 구축할 수 있음을 보인다. 실험을 통해 AWS Lambda Serverless Computing 기술을 활용하여 본 논문에서 제안한 시스템이 비용 절감뿐만 아니라 처리 시간 및 용량제한 문제를 해결하여 대형 신경망 모델을 서비스하기에 효율적인 성능을 보임을 확인하였다.

키워드 : 딥러닝, 서버리스 컴퓨팅, AWS 람다 서버, Cold Start Time, 용량제한

1. 서 론

최근 지능화되고 차별화된 서비스를 위해 딥러닝 기술을 이용한 다양한 애플리케이션의 개발 및 연구가 활발히 진행되고 있다. 진화하는 딥러닝 기술을 통해 사람 얼굴을 인식하여 다양하게 변화하는 모습을 보여주거나, 이미지를 인식하여 검색하는 등, 지능화된 서비스를 제공하는 방법들에 대한 관심이 증가되고 있다[1]. 딥러닝을 이용한 이미지 인식 기술은 GPU(Graphics Processing Units)와 같은 그래픽스 연산 전용 하드웨어를 필요로 하는 시간 소모적인 작업이다. 최근 이미지 인식 서비스는 모바일 단말에서 많이 사용되고 있다. 그러나 모바일 환경은 딥러닝을 위한 학습 및 추론에는 적합하지 않다. 그 이유는 모바일이라는 제한적인 공간으로 인해 GPU를 사용할 수 없고 학습을 위해 많은 양의 데이터를 읽고 쓰기에는 메모리 또한 부족하기 때문이다. 또한 추론

† 준 회 원 : (주)한화시스템 연구원
†† 비 회 원 : QURAM 연구소 선임연구원
††† 정 회 원 : 아주대학교 사이버보안학과 부교수
Manuscript Received : November 8, 2019
First Revision : January 31, 2020
Accepted : February 27, 2020
* Corresponding Author : Kangseok Kim(kangskim@ajou.ac.kr)

을 위해 학습된 신경망을 모바일에서 사용하는 것은 서비스 유지, 보수 측면에서 모델 및 기능 업데이트에 어려움이 따른다. 따라서 클라우드 시스템을 이용하여 고성능의 컴퓨팅 환경을 구축하고 딥러닝을 이용한 이미지 인식 애플리케이션을 실행 후, 결과를 모바일 사용자에게 서비스할 수 있는 환경이 필요하다. 그러나 클라우드 시스템을 사용하기 위해서는 고사양의 서버와 서버 관리 및 구축비용이 추가로 필요하다. 이 문제를 해결하기 위해 Serverless Computing[2, 3]이 등장하였고 Serverless Computing 기술을 사용하여 서버 관리 및 구축을 적은 비용으로 사용할 수 있는 환경이 형성되었다. Serverless Computing은 CPU 환경에서 지원하기 때문에 이미지 인식을 위한 신경망 모델을 학습하는 것은 어렵지만, 추론을 위해 학습된 신경망을 사용하는 것은 가능하다. 추론은 학습 데이터가 필요 없고 GPU를 사용하지 않아도 서비스가 가능하기 때문이다. 본 논문에서는 이미 학습된 신경망을 사용하여 이미지 인식 서비스 시스템을 구축한다. 이때 서버의 관리 및 비용을 줄이기 위해 Serverless Computing 기술을 사용하고 Serverless Computing이 가지고 있는 문제점을 해결하여 효과적인 신경망 서비스를 구축하는 방법을 제시한다. Serverless Computing 환경을 위해 AWS(Amazon Web Service)[4]에서 제공하는 Lambda Server[5]를 사용하며, 학습된 모델을 Lambda Server에서 실행 시 발생할 수 있는 비용 및 처리 속도에 대한 효율성을 평가할 것이다. 이때 효율성은 여러 사용자가 동시 접속 할 경우와 요청에 대한 응답까지의 처리 시간으로 평가할 것이다. 또한 Lambda Server의 단점[6]인 용량제한(Capacity Limitation) 문제와 초기에 Lambda Server를 생성하는데 소요되는 Cold Start Time 문제를 해결하여 클라우드 환경에서 Serverless Computing을 이용한 딥러닝 기반 이미지 인식 서비스를 효과적으로 제공할 수 있음을 보일 것이다. Fig. 1은 이미지 인식 서비스 시스템의 진화 과정을 나타낸다. 초기에는 신경망과 같은 고성능의 CPU를 필요로 하는 인식기를 사용하지 않고 간단한 인식 솔루션을 사용하여 단말에서 사용되었으며 이후 클라우드 서버를 이용하여 신경망 모델을 사용하게 되었다. 최근에는 클라우드 서버의 단점을 개선한 Serverless Computing Server의 등장으로 저렴한 비용으로 신경망 서비스를 지원할 수 있도록 진화하고 있다. 본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 설명한다. 3장에서는 AWS Lambda Server를 활용한 효율적인 딥러닝 기반 이미지 인식 서비스 시스템 개발 방법을 기술하며, 4장에서는 실험을 통하여 제안한 시스템의 효율성을 평가하고, 5장에서 결론을 맺는다.

2. 관련 연구

2.1 딥러닝(Deep Learning)

딥러닝(혹은 심층학습)은 이미지 인식, 음성 인식 및 자연어 처리 같은 비정형적인 데이터 영역을 이해하고 추론하기 위한 기계 학습(Machine Learning)의 한 분야이다[7]. 대형

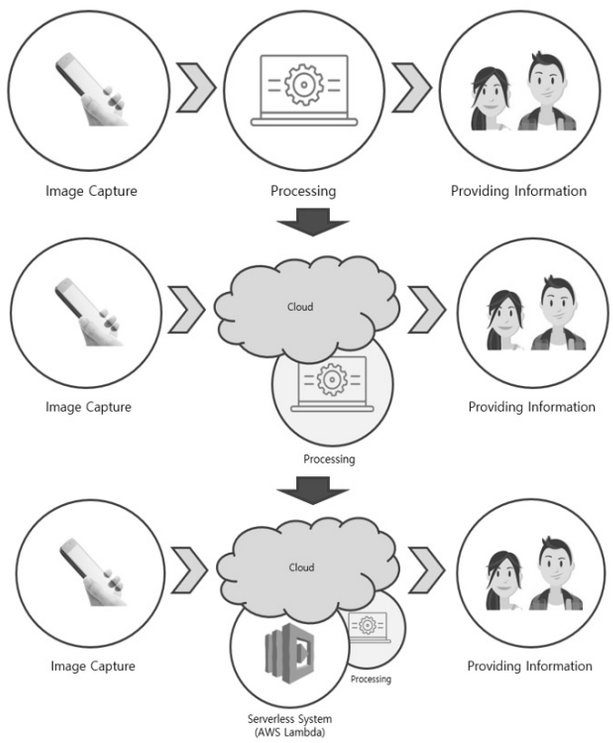


Fig. 1. Evolution of Image Recognition Service System

신경망 모델을 훈련시키기 위해서는 상당한 시간이 소요되며, 신경망의 여러 특성 변수에 적합한 가중치 매개변수를 찾기 위해서는 많은 양의 훈련 데이터와 고성능 하드웨어 장비가 필요하다. 본 논문에서는 이미지 분류에서 가장 많이 사용되는 ResNet50[8] 모델을 사용하였으며, 50개 층(50 layers)의 가중치를 저장하고 있는 신경망을 Serverless Computing 기술 기반 클라우드 서버에 사용할 수 있도록 제안한 시스템에 적용하였다. Table 1은 ImageNet[9] 데이터셋으로 미리 학습한 ResNet Model들의 정확도를 측정한 결과이다.

Table 1. ResNet Models Performance

Model Structure	Top-1 error	Top-5 error	Size(MB)
resnet18	30.24	10.92	44.6
resnet34	26.7	8.58	83.2
resnet50	23.85	7.13	97.7
resnet101	22.63	6.44	170
resnet152	21.69	5.94	230

사전에 학습된 모델을 사용하기 위해서는 인식하고자 하는 서비스의 데이터와 사전에 학습된 데이터가 유사해야 한다. 유사하지 않을 경우 사전에 학습된 모델을 그대로 사용할 수는 없으며 추가로 데이터를 수집하여 재학습을 수행해야 한다. 본 논문에서는 사전에 학습된 ImageNet 데이터와 서비스하고자 하는 데이터가 다르다고 가정하여 재학습을 수행하

였다. 이때 재학습을 하는 방법은 두 가지가 있다. 첫 번째는 사전에 학습된 모델의 마지막 Layer에 Fully Connected Layer(완전 연결 계층)를 추가하여 마지막 Layer만 재학습을 진행하는 것이다. 이 방법은 새로 수집한 데이터의 양이 적거나 사전에 학습한 데이터와 수집한 데이터가 유사할 경우에 사용할 수 있다. 두 번째는 학습된 모델의 마지막 Layer에 Fully Connected Layer를 추가하고 전체 모델을 재학습하는 것이다. 이 방법은 새로 수집한 데이터의 양이 많을 경우 사용할 수 있다. 두 가지 방법 모두 마지막 Layer에 Fully Connected Layer를 연결하는 이유는 사전에 학습된 모델의 클래스 수와 새로 학습한 모델의 클래스 수가 같지 않을 수 있기 때문이다. 마지막에 Fully Connected Layer를 추가함으로써 클래스의 수를 변경하여 모델을 재정의 할 수 있다.

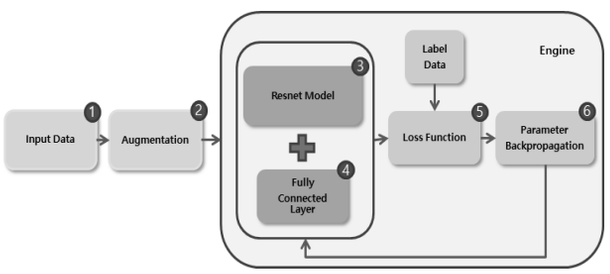


Fig. 2. Process of Deep Learning

Fig. 2는 사전에 학습한 모델을 사용하여 재학습을 수행하는 모델의 구조이다. 이러한 학습 방법을 Fine Tuning[10] 기법이라고 부르며 일반적으로 사전에 학습된 모델을 사용할 때 많이 사용되는 방법이다. 추가로 수집된 데이터의 양이 적을 경우 데이터의 양을 늘리기 위해 Augmentation[11]이라는 기법을 사용한다. Augmentation은 수집된 데이터에 크기나 회전, 조명 변화를 주어 데이터의 양을 늘리는 기법을 말한다.

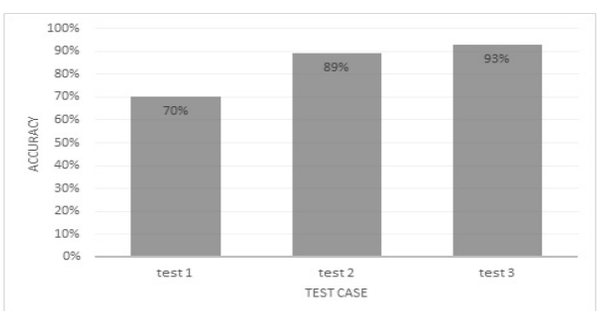


Fig. 3. Model Recognition Performance

본 논문에서는 모델의 성능을 측정하기 위해 3가지 종류의 모델을 만들어 비교하였다. Fig. 3은 3가지 측정 결과를 나타낸다. 학습에 사용된 데이터는 인터넷에서 수집한 특정 제품 이미지 이며 30 제품을 분류하기 위해 제품 당 100장 내외로 수집하였다. 1차는 직접 합성곱 신경망(CNN: Convolution Neural Network)을 생성한 모델로 테스트를 하였고 2차는 사전에 학습된 ResNet50 모델로 테스트를 진행한 결과이다. 3차는 ResNet50 모델을 사용하고 수집된 데이터에 Augmentation 기법을 적용하여 데이터를 증가시킨 후 테스트한 결과이다. 일반적으로 알려진 기법을 사용할수록 성능이 올라가는 것을 확인하였으며 본 논문에서는 이와 같은 기법들을 사용하였다.

2.2 AWS Lambda Server

차세대 클라우드 컴퓨팅 기술로 각광받고 있는 Serverless Computing 기술이 다양한 클라우드 애플리케이션 개발을 위해 활용되고 있다[12, 13]. 이장에서는 Serverless Computing 기술을 활용한 AWS Lambda Server에 대한 기술과 AWS Lambda Server의 단점인 Cold Start Time 문제와 용량제한(Capacity Limitation) 문제를 간략히 기술한다.

1) Serverless Computing 기반 AWS Lambda Server

AWS Lambda는 서버를 프로비저닝(Provisioning)하거나 관리하지 않고도 코드를 실행할 수 있게 해주며, 사용한 컴퓨팅 시간에 대해서만 요금을 지불하면 되는 클라우드 컴퓨팅 서비스이다[5]. AWS Lambda Server는 이벤트에 대한 응답으로 코드를 실행하고 자동으로 기본 컴퓨팅 리소스를 관리하는 Serverless Computing 서비스로서[5] 일반적으로 Fig. 4와 같이 동작하며 Lambda Instance는 Amazon API Gateway(Rest API)[14]를 통해 Lambda Function을 호출할 때 생성된다. Lambda Instance가 생성되면 초기화 과정이 진행되고 설정해 두었던 환경 설정을 하게 된다. 그 후 기능 수행을 위한 초기화 작업을 수행하고 생성이 완료된다. 이때 AWS Lambda Instance는 초기화 시간(Cold Start Time)이 필요하며 Serverless Function이 호출될 때마다 지연시간이 추가된다. 지연시간을 최소화하기 위해 플랫폼은 Serverless Function을 계속해서 호출하여 인스턴스(Instance)를 재사용 한다. 이때 인스턴스를 재사용 할 경우 지연되는 시간을 Warm Start Time이라고 한다[6]. 본 논문에서는 적은 비용으로 딥러닝 기반 이미지 인식 서비스를 사용자에게 제공하기 위하여 Serverless Computing 기술을 활용한 AWS Lambda Server를 사용하였다. 또한 AWS Lambda Server

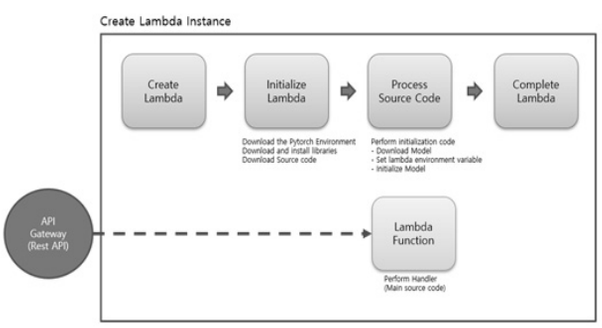


Fig. 4. Creating Process of Lambda Instance [13]

를 활용하여 사전에 학습된 신경망을 사용하여 실시간 이미지 인식 서비스 시스템을 구축 시 발생할 수 있는 용량제한 문제와 Cold Start Time 문제를 해결하여 클라우드 서비스를 이용한 대규모 신경망 실시간 서비스 시스템 개발 방법을 제안한다.

2) AWS Lambda Server의 문제점

AWS Lambda Server 시스템은 코드를 원활하게 배포하고 모든 관리 및 보안 패치를 수행하며 Amazon CloudWatch를 통해 내장된 로깅 및 모니터링 기능이 가능하며 이벤트가 발생한 후 1초 이내에 코드를 실행한다[5]. 또한, Lambda는 자동으로 확장되어 이벤트 발생 빈도가 증가해도 성능이 일관되게 유지된다. 코드 상태가 저장되지 않으므로 Lambda는 배포와 구성에 대한 지연 없이 필요한 만큼 많은 인스턴스를 시작할 수 있는 큰 장점을 가지고 있다. 그러나 AWS Lambda Server는 One-Time Instance와 용량제한 문제를 야기할 수 있다. One-Time Instance 문제는 일정시간 동안 서버의 사용이 없으면 자동적으로 서버에서 사용되고 있던 설정 값들이 초기화되는 문제이다. 이것은 대규모 신경망을 서비스하기 위한 각종 라이브러리와 구동 환경 시스템들을 매번 다시 설정해야 하는 경우가 발생할 수 있다. 용량제한 문제의 경우 AWS Lambda Server는 제한적인 저장 공간을 가지고 있기 때문에 큰 용량의 라이브러리 및 신경망 모델을 사용하는데 어려움이 있다. AWS Lambda Server의 초기화 방법은 3가지 경우(3MB 이하, 50MB, 250MB)로 나눌 수 있는데, 대부분 신경망 모델의 크기는 50MB를 초과하기 때문에 용량제한 문제를 효과적으로 해결하는 것이 필요하다.

3. AWS Lambda Server를 활용한 효율적인 딥러닝 기반 이미지 인식 서비스 시스템

본 논문에서 제안하는 AWS Lambda Server를 활용한 딥러닝 기반 이미지 인식 서비스 시스템의 전체적인 아키텍처를 간략히 기술하며, AWS Lambda Server를 사용 시 발생할 수 있는 Cold Start Time 과 용량제한 문제를 해결하기 위한 방법을 기술한다.

3.1 AWS Lambda Server 기반 제안 시스템

AWS Lambda Server 기반으로 개발된 제안 시스템은 기본적으로 4개의 컴포넌트(User Application, WAS, S3 Storage Server, Lambda Classification Server)로 구성되며 Fig. 5는 전체 시스템 아키텍처이다.

1) User Application Component

User Application은 서비스를 요청하는 역할을 하며 다양한 디바이스에서 동시에 수행될 수 있다. 본 논문에서는 이미지 인식 신경망을 사용하기 때문에 스마트폰의 Camera API를 사용하여 이미지를 획득하고 그 이미지를 WAS (Web Application Server)로 전송하여, 서비스를 제공받는 역할을 한다.

2) WAS(Web Application Server) Component

WAS는 스마트폰 애플리케이션으로부터 서비스 요청을 처리해주는 역할을 하는 서버이다. 다른 컴포넌트(User Application, S3 Storage Server, Lambda Classification Server)를 제어하며, 이미지 분류(Classification)에 대한 응답을 스마트폰 애플리케이션에게 전달해 주는 웹 애플리케이션 서버이다.

3) Amazon S3 Storage Server Component

S3는 사용자가 요청한 데이터들을 저장하며 필요할 때 사용할 수 있게 한다. 또한 AWS Lambda Server가 생성될 때 필요한 데이터들을 저장한다. WAS와 S3의 연동을 위해 아마존에서 제공하는 Eclipse AWS Toolkit을 사용하였다.

4) AWS Lambda Classification Server Component

대형 신경망 모델을 수행 할 수 있는 환경을 갖추고 있으며, 요청 사항을 해결하여 WAS에 반환해 준다. AWS Lambda Classification Server는 제안하는 시스템 환경에서 딥러닝 기반 이미지 인식 서비스를 위해 핵심적인 역할을 하는 엔진을 포함하고 있다.

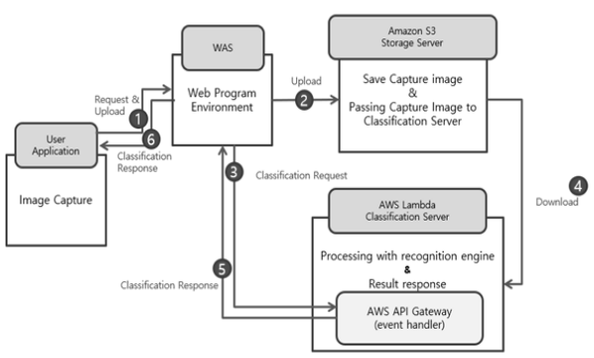


Fig. 5. Proposed Overall System Architecture

전체적인 시스템의 처리 과정은 다음과 같이 6단계로 나눌 수 있다.

- 1 단계 : User Application에서 이미지를 WAS에 전달하는 과정이다. 전달되는 이미지는 모바일 카메라를 통해 정지 영상을 얻거나 저장 장치에 저장된 이미지를 사용할 수 있다.
- 2 단계 : WAS가 전달받은 이미지를 S3 Storage Server로 전달하는 과정이다. Storage Server에 전달하는 이유는 서비스에 사용되는 데이터를 수집하는 기능과 Lambda Server의 용량제한 문제를 해결하기 위해 추가되었다.
- 3 단계 : WAS가 Lambda Server로 인식 서비스를 요청

하는 과정이다. 요청 시 API Name과 Storage Server에 저장되어 있는 이미지의 경로를 함께 포함하여 요청한다. 요청은 POST 방식과 JSON 형태의 파라미터로 전달한다. Table 2는 JSON 파라미터 형식을 나타낸다.

Table 2. JSON Parameter Format for Request

Parameter	Type	Description
Method	String	Rest API Name
File path	String	File Path

- 4 단계 : Lambda Server가 요청받은 서비스를 실행하면서 필요한 데이터를 Storage Server로 부터 다운로드 받는 과정이다. 다운로드 받는 데이터는 처리할 이미지와 Lambda Instance를 생성하고 서비스를 수행할 수 있도록 하는 초기 데이터를 포함하고 있다.
- 5 단계 : Lambda Server에서 서비스 결과를 WAS에게 전달해 주는 과정이다. 응답은 JSON 형태로 반환된다. 응답 데이터는 API 이름, 인식된 제품의 Softmax 값이 높은 순으로 Top 5 까지 반환하며 Top 5 제품의 각각의 ID코드와 함수 수행시간을 초단위로 반환 한다. Table 3은 Lambda Server에서 WAS로 응답하는 데이터를 보여준다.
- 6 단계 : 서비스 결과를 User Application에게 전달해 주는 과정이다.

Table 3. An Example of Classification Response

Return Data
API Name, Top1~Top5(ID, Softmax), Process Time(ms)

Table 4. An Example of Classification Request

POST Method
Ex) HTTP URL = https://grzejeam12.execute-api.ap-northeast-2.amazonaws.com/classification

본 논문에서는 제품 이미지 인식 솔루션과 웹 서비스를 연동시키기 위해 Rest API를 정의하여 인식 서비스를 요청하는데 활용하였다. Rest API의 종류는 “classification”과 “wakeup”으로 구성되며, Table 4는 WAS에서 Lambda Server로 이미지 인식 서비스를 요청하는 Post Method의 한 예이며 “domain/APIName” 형태로 사용한다. Fig. 6은 classification API 사용의 한 예로써 입력된 제품 이미지를 분류해주는 역할을 한다. 클라이언트가 제품 이미지 파일을 Storage Server에 업로드 후 해당 파일의 경로를 포함하여 classification API를 호출하게 된다. 해당 API를 받은 Lambda Server는 파일 경로를 통해 Storage Server로부터 제품 이미지를 다운로드 받아 처리한다.

```
Request (JSON Type) {
        "method": "classification",
        "file_path": "test/20200118jpg"
}
Response (JSON Type) {
      "method": "classification",
      "TOP1_ID": "0121",
      "TOP1_softmax": 0.13,
      "TOP2_ID ": "1201",
      "TOP2_softmax": 0.06,
      "TOP3_ID ": "00011",
      "TOP3_softmax": 0.05,
      "TOP4_ID ": "00060",
      "TOP4_softmax": 0.04,
      "TOP5_ID ": "10001",
      "TOP5_softmax": 0.04,
      "time": 0.23528790473937988
}
```

Fig. 6 An Example of Classification API

3.2 Cold Start Time

AWS Lambda Server는 이벤트 발생 후 일정시간이 경과되면 인스턴스(Instance)를 순차적으로 제거한다. 인스턴스가 제거되면 서비스가 수행될 때마다 인스턴스 생성에 필요한 작업들이 다시 실행되어야 한다. Lambda Function은 일반적으로 인스턴스를 생성하고 초기화(bootstrapping)를 하는 수행 시간이 필요하다(Cold Start Time). 초기화 수행에 따른 지연시간은 사용자에게 신경망 서비스를 제공하기에 부적합한 환경 요소가 될 수 있다. Cold Start Time은 사용되는 언어 및 라이브러리에 따라 다를 수 있지만 본 논문에서 제안하는 환경에서는 8초 정도 소요되는 것을 확인하였다. 이벤트가 발생할 때마다 Cold Start Time이 발생하는 문제를 줄이기 위해서는 서비스를 제공하는 동안 Lambda Server가 유지될 수 있도록 처리하는 기법이 필요하다. Cold Start Time은 code를 다운로드 받고, container를 실행시키고, bootstrapping하는 시간을 포함한다. 실제 기능만 수행되는 code 실행 시간은 Warm Start Time이라고 부른다. Lambda Server는 Cold Start Time을 최소화하고 초기화 시간을 피하기 위해 Serverless Function을 주기적으로 호출하여 인스턴스를 재사용한다. 인스턴스를 재사용 할 경우 Cold Start Time이 아닌 Warm Start Time만 소요되기 때문에 사용자에게 최적의 신경망 서비스를 가능하게 해준다.

본 논문에서는 인스턴스를 유지하기 위해 Rest API를 주기적으로 호출하는 방식을 사용하였다. 그러나 유지되고 있는 인스턴스의 수 보다 많은 요청이 동시에 발생할 경우 같은 문제를 야기하였다. 따라서 JMeter[15]라는 웹 성능 테스트 툴을 이용하여 동시에 다수의 Lambda 인스턴스에 신호를 주는 방식을 사용하였다. 그 외의 방법인 AWS에서 제공하는 CloudWatch Events나 Kinesis를 통해서도 주기적으로 신호를 줄 수 있지만 다수의 인스턴스에 신호를 주지 못하여 Cold Start Time을 제거할 수는 없었다. JMeter를 사용하면 지속

적으로 일정 시간마다 다수의 Lambda Server에 신호를 줄 수 있었으며, 따라서 다수의 Lambda 인스턴스는 활성화 상태를 계속해서 유지할 수 있게 된다. 그러나 외부 툴을 사용하기 위해서는 별도의 서버가 필요한 문제가 있다. 본 논문에서는 외부 툴을 사용하여 다수의 Wakeup API를 요청하기 위하여 WAS Component를 활용하였다. WAS Component가 시스템의 모듈들을 제어하고 JMeter를 사용하여 Lambda Server에 이벤트를 발생시켜 Lambda Instance를 활성화 상태로 계속 유지시켜주는 역할을 한다. 시스템에서는 이렇게 지속적으로 일정 시간마다 Lambda Server에 신호를 보내는 Wakeup API를 생성하며 요청 및 응답 방법은 POST 방식과 JSON 구조를 사용한다. Table 5는 JSON 구조를 나타낸다. 파라미터 method는 wakeup API라는 것을 명시하고, 파라미터 sleep_time은 Lambda 지연시간을 나타낸다. Fig. 7은 Wakeup API의 사용 예를 보여준다. Wakeup API에서 sleep_time을 주는 이유는 많은 양의 Wakeup을 한 번에 요청할 경우 안정성이 떨어지는 이유 때문에 400ms의 sleep time을 주고 사용하였다. 이렇게 사용할 경우 안정적으로 Wakeup API의 역할을 하는 것으로 확인하였다.

Table 5. Wakeup JSON Parameter Format

Parameter	Type	Description
method	String	API Name
sleep_time	String	Sleep Time

```
Request (JSON Type) {

"method": "wakeup",
"sleep_time": "800"
}

Response (JSON Type) {

    "method": "wakeup",
    "time": 0.8
}
```

Fig. 7 An Example of Wakeup API

3.3 용량제한(Capacity Limitation)

대규모 신경망을 서비스하기 위해서는 환경 설정 및 모델과 라이브러리 등 많은 용량을 차지하는 패키지들을 필요로 한다. 제안한 신경망 모델은 PyTorch[16] 환경에서 학습한 모델을 사용하기 때문에 PyTorch 환경 패키지 및 모델 업로드가 필요한데, 용량제한 문제로 인해 Lambda 인스턴스에 업로드가 불가능 하다. 기존 논문[6]에서는 Warm Start Time과 Cold Start Time이 메모리 크기에 따라 어떻게 변하는지 확인하였다. 그 결과 Cold Start Time의 경우 메모리 크기에 따라 처리 시간은 감소하지만 Warm Start Time은 Cold Start Time과 유사한 패턴을 보이지는 않았다. 본 논문에서는 메모리 크기는 Cold Start Time에 대해서만 테스트를 진행하여 동시 접속의 양을 늘려가면서 메모리의 최적화된 값을 찾기 위한 실험을 하였다. 용량제한의 경우 Lambda direct upload 시간이 50MB로 제한되어 있기 때문에 본 논문에서 사용된 모델과 환경 설치 파일들을 업로드하기에는 부족하다[17]. 때문에 간단한 설치 파일의 경우 including layers에 포함시켜 설치하는 방식으로 진행하였고 모델이나 용량이 큰 설치 파일은 AWS S3 Storage Server에 업로드한 후 Lambda 인스턴스에서 불러오는 방식으로 사용하였다. 또한 신경망 서비스에서는 촬영된 데이터가 중요한 정보로 사용되기 때문에 사용자들의 데이터 또한 관리되어야 한다. 제안하는 시스템에서는 AWS S3 Storage Server Component를 활용하여 설치 패키지 및 학습 모델을 Storage Server에 미리 업로드 한 후 Lambda 인스턴스가 만들어 질 때 불러오는 방식으로 사용한다. 또한 사용자가 사용한 이미지 데이터의 경우 Storage Server에 저장하여 보관하고 Lambda Server가 필요 할 때 Storage Server로부터 이미지를 다운로드 받아 Lambda Function을 수행한 후 인식결과를 반환한다. 그러므로 AWS Lambda Server Component와 S3 Storage Server Component의 연동 서비스를 통해 용량제한 문제를 해결할 수 있었다.

4. 실험 및 결과

본 논문은 Serverless Computing 플랫폼을 이용하여 대형 신경망 추론에 활용할 수 있음을 보여주고 저렴한 비용으로 사용자에게 최적의 서비스를 제공할 수 있음을 보여주는 것이다. 실험은 비용 부분과 처리 속도 부분으로 나누어서 진행하였다.

4.1 서비스 처리 비용 실험

1) AWS Lambda Server의 메모리 크기 별 처리 비용

Lambda Server는 메모리의 크기를 선택하기에 따라서 처리되는 비용이 다르다. Table 6은 AWS Lambda의 메모리 크기 별 사용 가격[6, 18]을 나타내며 제안하는 시스템에서 사용하는 768MB의 사양을 선택하여 사용 시 100ms 당 $0.00000125 비용이 들며, 요청 당 $0.0000002가 추가되어 최종 한번 요청 시 $0.00000145 가 부가된다. 100만 건의 요청이 발생할 경우 $1.45가 부가되며 처리 시간이 500ms라고 가정할 경우 $7.25가 부가된다.

Table 6. AWS Lambda Pricing [6,18]

Memory(MB)	Price per 100ms ($)
448	0.000000729
768	0.00000125
1536	0.000002501
3008	0.000004897

2) EC2 Server와 Lambda Server 처리 비용 비교

처리 비용 비교 시 Lambda Server는 메모리를 768MB 사용하고 하나의 요청을 처리하는데 500ms가 소요된다고 가정하였다. AWS Lambda는 필요시에만 코드를 실행하며, 하루에 몇 개의 요청에서 초당 수천 개의 요청까지 자동으로 확장이 가능하다. 또한 사용한 컴퓨팅 시간에 대해서만 요금을 지불하며 코드가 실행되지 않을 때는 요금이 부과되지 않는 장점이 있다. 때문에 Lambda Server는 Amazon EC2(Elastic Compute Cloud) Server의 처리 비용에 비해 하루 사용량에 따라 최대 약 10배 넘는 차이가 발생하게 된다. Fig. 8은 EC2 Server와 Lambda Server의 하루 처리비용을 비교한 그래프이다. 하루 10000건의 요청이 발생할 경우 EC2 Server는 3.3달러가 필요하지만 Lambda Server는 0.0725달러 밖에 필요하지 않으며, Lambda Server가 약 1,000,000건 정도 처리해야 EC2 Server와 동일한 비용을 사용하게 된다. Fig. 9는 EC2 Server와 Lambda Server가 동시에 처리량이 증가할 경우에 비용을 비교한 그래프이다. EC2 Server의 경우 동시 접속자가 증가할수록 Server를 증축해야 하는 문제 때문에 비용이 선형적으로 증가하지만 Lambda Server의 경우 동시 접속자가 증가하여도 처리되는 비용은 차이가 없다.

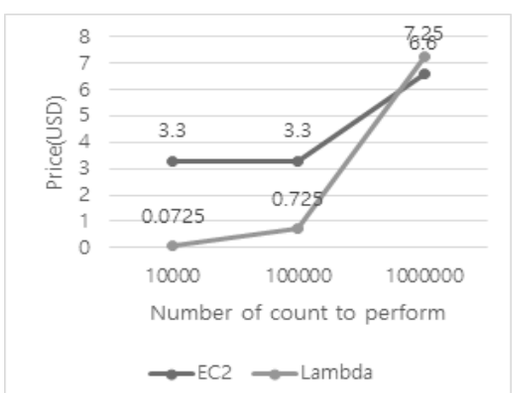


Fig. 8. Processing Costs for One Day Request (AWS EC2 Server vs. AWS Lambda Server)

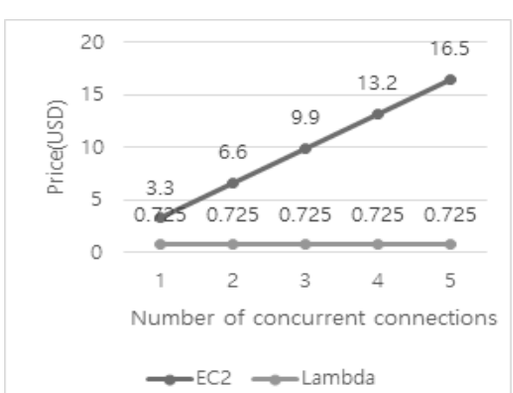


Fig. 9. Processing Costs for Concurrent Connections (AWS EC2 Server vs. AWS Lambda Server)

4.2 서비스 처리 속도 실험

1) 실험 환경

본 연구에서는 성능 측정을 위해 Table 7, Table 8, Table 9에 각각 기술된 개발 언어/라이브러리, 모바일 실험 환경, Lambda Server 실험 환경에서 테스트를 진행하였다.

Table 7. Development Language and Library

Category	Content
Language	Python
Usage Library	JSON, PyTorch, CUDA, cuDNN

Table 8. Mobile Experimental Environment (Galaxy note8)

Category	Content
GPU	ARM Mali-G71 MP20 546 MHz GPU
CPU	Samsung 2nd Custom CPU Architecture MP4 2.3 GHz CPU ARM Cortex-A53 MP4 1.7 GHz CPU
RAM	6GB LPDDR4X SDRAM
Network Environment	LTE

Table 9. Lambda Server Experimental Environment

Category	Content
Memory	768MB(128MB~3008MB)
CPU	Lambda allocates CPU according to memory size.

2) Lambda Server의 단계별 처리 속도

Fig. 10은 Lambda Server에서 수행되는 단계 별 처리 속도를 측정한 결과이다. 최초 수행 시 Cold Start Time으로 인해 10460ms 정도가 소요되지만(왼쪽 그림) 두 번째 수행부터는 460ms로 빠르게 수행되는 것을 확인할 수 있다(오른쪽 그림).

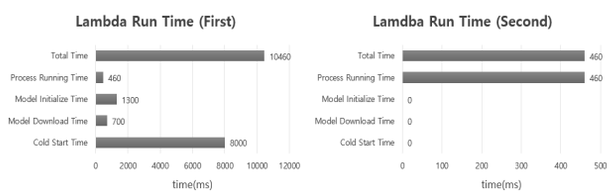


Fig. 10. Lambda Function Run Time

Fig. 11은 제안하는 시스템의 단계별 수행 시간을 나타낸다. 업로드나 다운로드의 경우 인터넷 망에 따라 성능이 달라질 수 있지만 LTE 망으로 가정하고 평가하였다. 측정 결과 최초 실행 시 발생하는 Cold Start Time과 Lambda Initialization (Model Initialization) 시간을 제외 할 경우 1초 이내의 시간으로 대형 신경망 서비스를 제공할 수 있다는 것을 확인하였다. 측정 시 요청(request)과 응답(response) 시간은 배제하였다.

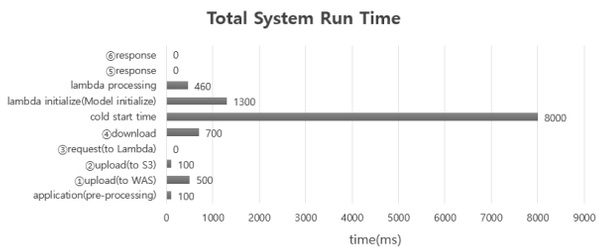


Fig. 11. Total System Run Time

Lambda Server는 특성상 일정시간이 지나면 인스턴스를 순차적으로 제거한다. 실험 결과 인스턴스가 다수 일 경우 5분 후 순차적으로 제거되며 인스턴스가 하나일 경우 30분 후 제거되는 것을 확인하였다. 인스턴스가 제거되면 인스턴스를 다시 생성하는 과정이 필요한데 이때 발생하는 Cold Start Time은 사용자에게 서비스를 제공하는데 문제를 야기할 수 있다. 때문에 Lambda 인스턴스를 유지할 수 있도록 신호를 주는 Rest API를 정기적으로 호출하는 루틴을 구현하였다. Rest API를 5분에 한 번씩 호출할 경우 한 달에 약 $0.3의 비용이 소요되므로 큰 무리 없이 사용할 수가 있다.

Lambda Server 인스턴스 개수만큼 서비스를 동시접속 사용 시 Cold Start Time을 피할 수 있다. 제안하는 시스템에서는 동시에 5명 이하 서비스를 사용할 수 있도록 5개의 인스턴스를 활성화하였다. 최초로 인스턴스를 생성할 때는 약 10초의 시간이 걸리지만 이후 Cold Start Time을 피하여 100ms 내의 시간만 걸린다. 100ms 안에 5명 이상이 동시에 접속하는 상황이 생길 경우 인스턴스의 개수를 늘리면 많은 사용자가 서비스를 이용해도 Cold Start Time이 발생하지 않는다. 주기적으로 5분마다 Lambda Server를 깨워주는(invoke) 해당 API를 호출해줌으로써 Lambda Cold Start Time을 피할 수 있었다.

4.3 동시 접속

본 논문에서는 동시에 많은 사용자가 접속할 경우에 Lambda Server의 인스턴스 확장이 자동으로 잘 이루어지는지 확인하였다. Fig. 12는 은 동시접속을 100회씩 늘려가면서 측정한 결과이며 1000회 미만이나 1000회 이상에서도 동시접속 수에 비례하여 응답시간이 증가하는 것을 확인 할 수 있었다. 이 테스트는 동시에 5명 이하에서 서비스할 수 있도록 5개의 인스턴스를 활성화한 상태이며 동시접속 분류마다 10번의 반복 테스트를 진행하였다. 결국 많은 양의 동시 접속자가 발생해도 서버가 과부하 되거나 오류가 발생하지 않는다는 것을 확인하였으며 일정한 비율로 동시접속량이 증가한 만큼 시간도 증가하는 것을 확인할 수 있다. 또한 Lambda Server에서 동시성 제한을 1000회로 설정하고 있지만 1000회 이상의 동시 접속이 발생하여도 응답시간에 영향을 미치지 않는 것으로 확인하였다. 결국 제안하는 시스템은 동시 접속자 수가 갑자기 증가하거나 감소한다고 해서 서비스에 큰 영향을 미치지 않음을 확인하였다.

Less than 1000 times				
cc	avg	min	max	stdev
100	663	542	845	62.7
200	974	689	1591	92.03
300	1233	585	1921	170.82
400	1410	668	2220	221.39
500	1656	777	2527	294.59

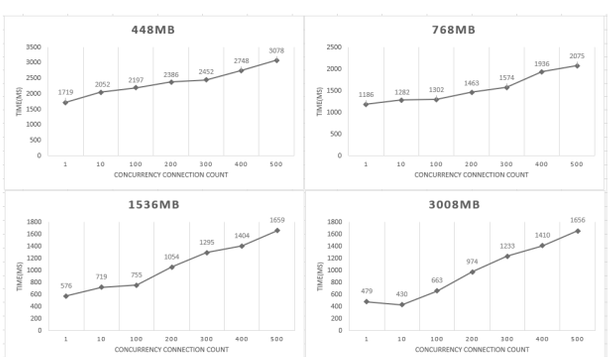
More than 1000 times				
cc	avg	min	max	stdev
1100	2530	552	6584	1035.77
1200	3097	407	7544	1154.99
1300	3182	377	6971	1245.67
1400	3517	380	7019	1276.65
1500	3835	409	7544	1325.64

*cc : concurrency connection

Fig. 12. Concurrency Connection Test

4.4 Lambda Server 메모리 별 성능 평가

Fig. 13은 Lambda Server의 메모리 별 처리 속도를 측정한 그래프이다. 속도 측정 시 메모리 사양에 따른 동시접속 성능도 평가하였다. 이 측정은 5개의 인스턴스를 활성화한 상태로 동시접속 분류마다 10번의 반복 테스트를 진행하였다. Lambda Server는 기본 설정에서 메모리를 128MB부터 3008MB까지 조절할 수가 있다. AWS Lambda Test Function을 이용하여 측정한 결과, 본 논문에서 Lambda Server가 사용하는 최대 메모리 크기는 422MB이다. 488MB에서 3998MB 사이의 메모리를 비교 한 결과 최대 3008MB와 1536MB 사양은 미세한 차이를 보여준 반면 768MB 사양 아래로 내려갈 경우 급격하게 성능이 떨어지는 것을 확인할 수 있었다. 결국 Lambda Server는 기본적으로 768MB 사양 아래로 내려갈 경우 성능이 급격하게 떨어진다고 볼 수 있다. 실험결과 ResNet50을 이용



minimum specification (448MB)				
cc	avg	min	max	stdev
1	1719	1719	1719	0
10	2052	1677	2322	275.52
100	2197	1406	2772	277.57
200	2386	1636	3084	210.45
300	2452	1587	3366	336.34
400	2748	1420	5139	417.35
500	3078	1634	4196	411.58

1/4 specification (768MB)				
cc	avg	min	max	stdev
1	1186	1186	1186	0
10	1282	1077	1391	91.64
100	1302	756	1695	180.79
200	1463	714	1987	207.12
300	1574	979	2137	242.67
400	1936	1068	2636	310.4
500	2075	932	3579	401

1/2 specification (1536MB)				
cc	avg	min	max	stdev
1	576	576	576	0
10	719	457	1116	88.18
100	755	556	950	87.41
200	1054	547	1338	161.95
300	1295	544	1939	235.55
400	1404	730	2062	279.43
500	1659	563	2498	367.01

maximum specification (3008MB)				
cc	avg	min	max	stdev
1	479	479	479	0
10	430	378	528	58.24
100	663	542	845	62.7
200	974	689	1591	92.03
300	1233	585	1921	170.82
400	1410	668	2220	221.39
500	1656	777	2527	294.59

*cc : concurrency connection

Fig. 13. Comparison by Memory Size for Lambda Server

한 신경망 서비스를 위해서는 768MB 이상의 메모리를 사용해야 최적의 서비스를 제공할 수 있다는 것을 확인하였다. 하지만 가격 대비 최적의 성능을 내기 위해서는 1536MB 이상의 사양은 비용 측면에서 768MB보다 2배 이상 높다. 또한 동시접속자 수가 증가하면 높은 사양으로도 가격대비 성능을 얻기 어렵다는 것을 확인하였다.

5. 결론 및 향후 연구계획

본 논문에서는 클라우드 환경에서 딥러닝 기반 이미지 인식 서비스 시스템 개발을 위해 Amazon에서 개발한 Serverless Computing Lambda Server를 활용함으로써 일반적으로 사용되는 Amazon EC2 Server에 비해 적은 비용으로 대형 신경망 모델을 모바일 사용자에게 효율적으로 서비스할 수 있는 방법을 제안하였다. EC2 Server의 경우 시간 단위로 리소스를 대여해야 하는 반면, Lambda Server는 이벤트 요청에 따라서 사용될 수 있으며 또한 Lambda Server는 EC2 Server와 같이 인프라 설계를 고려할 필요가 없다. 또한 장애에 대한 고려가 필요 없고 요청량 증가에 따른 확장을 자동으로 처리해 준다. 따라서 Lambda Server를 사용함으로써 관리에 부담을 최소화할 수 있었다. 또한 Lambda Server를 사용하기 위해서 본 논문에서는 기존의 Lambda Server의 문제점인 Cold Start Time 문제와 용량제한 문제를 해결하기 위한 시스템 아키텍처를 제안하였고, 실험 결과 대형 신경망 서비스를 빠르고 저렴한 비용으로 서비스할 수 있음을 확인하였다. 그러므로 Serverless Computing 환경을 효과적으로 활용한다면 적은 비용으로 딥러닝 기반 이미지 인식 서비스뿐만 아니라 이미지 검색, 문자 인식(Optical Character Recognition), 객체 탐지 등 다양한 딥러닝 기반의 서비스를 제공할 수 있을 것이다. 향후 Serverless Computing 시스템의 Cold Start Time 프로세스를 인스턴스의 재사용이 아닌 직접적으로 해결할 수 있는 방법이 개발된다면 Serverless Computing 플랫폼은 앞으로 다양한 시스템과 애플리케이션 및 실시간 딥러닝 서비스에 사용될 차세대 플랫폼 기술이 될 것이라 생각한다.

References

[1] L. Feng, P. Kudva, D. D. Silva, and J. Hu, "Exploring serverless computing for neural network training," *IEEE 11th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, USA, Jul. 2-7, 2018, DOI: 10.1109/CLOUD.2018.00049

[2] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of serverless computing and Function-as-a-service (FaaS) in industry and research," *arXiv preprint arXiv:1708.08028*, Aug. 2017, DOI: 10.13140/RG.2.2.15007.87206.

[3] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski et al., "Serverless computing: Current trends and open problems," Research Advances in Cloud Computing. Springer, Singapore, pp.1-20, Dec. 2017, DOI: 10.1007/978-981-10-5026-8_1.

[4] AWS (Amazon Web Services) [Internet], https://aws.amazon.com/ko/.

[5] Lambda Server [Internet], https://aws.amazon.com/ko/lambda/.

[6] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," *2018 IEEE International Conference on Cloud Engineering (IC2E)*, Orlando, FL, USA, 17-20 April 2018, DOI: 10.1109/IC2E.2018.00052.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, Vol.521, No.7553, pp.436-444, May 2015, DOI: 10.1038/nature14539.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, pp.770-778, Jun. 2016, DOI: 10.1109/CVPR.2016.90.

[9] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, pp.248-255, 20-25 Jun. 2009, DOI: 10.1109/CVPR.2009.5206848.

[10] W. Ge and Y. Yu, "Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning," arXiv preprint, 2017, https://arxiv.org/abs/1702.08690.

[11] H. Inoue, "Data augmentation by pairing samples for images classification," arXiv preprint, 2018, https://arxiv.org/abs/1801.02929.

[12] R. Chard, K. Chard, J. Alt, D. Y. Parkinson, S. Tuecke, and I. Foster, "Ripple: Home automation for research data management," *IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp.389-394, Jun. 2017.

[13] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, Trento, Italy, Dec. 12-16, pp.1-4, 2016, DOI: 10.1145/3007203.3007217.

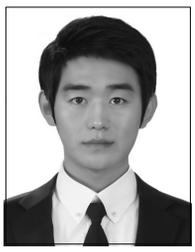
[14] Amazon API Gateway [Internet], https://aws.amazon.com/ko/api-gateway/.

[15] Apache JMeter [Internet], https://jmeter.apache.org/.

[16] PyTorch [Internet], https://pytorch.org/.

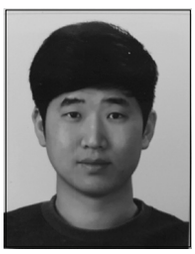
[17] AWS Lambda Limits [Internet], https://docs.aws.amazon.com/ko_kr/ lambda/latest/dg/limits.html.

[18] AWS Lambda Pricing, https://www.amazonaws.cn/en/lambda/pricing/.



이 현 철

https://orcid.org/0000-0003-2089-2010
e-mail : deletenim@naver.com
2012년 아주대학교 지식정보공학과(석사)
2012년 ~ 2019년 QURAM 연구소 선임연구원
2020년 ~ 현 재 ㈜한화시스템 연구원
관심분야 : 이미지 처리, 컴퓨터 비전, 기계학습, 딥러닝



이 성 민

https://orcid.org/0000-0001-9842-0334
e-mail : sm.lee@quramsoft.com
2013년 아주대학교 컴퓨터공학(학사)
2017년 아주대학교 컴퓨터공학과 (석사과정수료)
2013년 ~ 현 재 QURAM 연구소 선임연구원
관심분야 : 컴퓨터 비전, 컴퓨터 그래픽스

김 강 석

https://orcid.org/0000-0001-8950-7577
e-mail : kangskim@ajou.ac.kr
2007년 인디애나대학교 컴퓨터공학 (박사)
2010년 ~ 2016년 아주대학교 지식정보공학과 연구교수
2018년 ~ 현 재 아주대학교 인공지능·데이터사이언스학과 부교수
2016년 ~ 현 재 아주대학교 사이버보안학과 부교수
관심분야 : 클라우드컴퓨팅, IoT 협업시스템, 블록체인, 빅데이터 응용보안, 기계학습/딥러닝 기반 보안 분석