드론 원격센싱 기반의 에너지 소모 최적화를 위한 딥러닝 연산 오프로딩 연구

Deep Learning Computation Offloading Experiment for Energy Consumption Optimization Based on Drone Remote Sensing

\*Seon Min Kim, \*DuHyeok Jang,

\*Hyun Woong Choi, \*Jung Soo Lee, \*Jun Young Heo

ISYS Lab - Hansung University

# Abstract

소형 무인 비행장치인 드론은 최근 다양한 산업과 연계 되어 다양하게 활용 되어 지고 있다. 이에 따라 드론이 처리하도록 요구 되어 지는 데이터의 규모도 늘어났고 보다 효율적인 작업을 위해서 드론이 소모하는 에너지를 최적화하여 그에 맞게 환경을 구성할 필요가 증가되었다.

재난 재해 수색 등 다양한 방면으로 활용 되어지면서

수집이 되는 데이터 또한 매우 증가하였다. 하지만, 드론의 제한된 하드웨어 사양으로 인해, 데이터 처리에 대한 제약이 발생되어 졌고, 이에 대한 에너지 소모량, 시간 등을 가장 효율적으로 이용할 수 있게 하는 것에 대한 중요성이 대두되었다..

이 논문에서는 이를 극복하기 위해 드론과 유사한 저전력 환경의 여러 장비에 딥러닝 기반의 모델을 탑재하여 이를 통해 추론하고, 특정 레이어 까지의 소모 에너지를 파악하는 실험을 진행하였다. 해당 실험을 통해 드론 컴퓨팅 디바이스의 최적 에너지 소모점 계산 알고리즘과 특정 네트워크 환경에서 데이터를 서버로 전송하는 동안 소모되는 에너지의 양을 계산하여 네트워크 처리 성능과 에너지 간에 최적의 모델을 찾는 기법에 대해 제안한다.

또한, VGG16 기반 모델을 레이어 단위로 나누어 실험한 결과, Raspberry Pi 3 B+과 Jetson Nano 보다 Coral USB Accelerator 를 사용한 실험이 더 적은 에너지 소모를 보여주었고, 네트워크 전송 에너지를 계산한 결과, 전반적으로 TCP 프로토콜이 UDP프로토콜 보다 짧은 전송 시간과 낮은 전력 소모량을 보이는 결과를 보여주었다. 이를 통해 최적 에너지 소모 알고리즘을 설계하고, 레이어와 에너지 간의 관계 모델을 도출함으로서 해당 모델의 오프로딩 최적점을 제안할 수 있었다.

1. Introduction

드론은 무인 비행장치로 항공∙통신∙센서∙소프트웨어 등의 산업에서 중요성은 커지고 있고 공공분야에서의 활용, 또한 환경조사 및 감시, 해양감시∙관리, 시설물 점검 및 관리, 지형정보 조사, 재난 재해 수색 등 다양한 방면으로 활용 되어 지고 있다. 그에 따라, 드론이 처리해야 할 데이터의 양 또한 급격하게 증가하고 있고, 대용량의 데이터를 처리해야 하는 경우도 발생한다. 하지만, 드론의 제한된 하드웨어 환경으로 인해 실시간 분석 및 데이터 전송이 어려운 경우가 발생할 수 있고. 또한, 데이터 처리에 과도한 전력을 소모하여, 다른 기능들을 효과적으로 장기간 수행 할 수 없는 문제점 등이 있다.

따라서, 드론을 통해 추론을 하거나, 네트워크로 데이터를 보내 처리할 시 에너지를 적절하게 분배하여 효율적으로 사용할 수 있는 최적화된 오프로딩 방법이 필요하다.

본 논문에서는 이를 극복하기 위해 드론과 유사한 저전력 컴퓨팅 디바이스들에 대해 딥러닝 모델을 탑재 하고, 직접 설계한 소모 에너지 최적점 계산 알고리즘에 입각한, 딥러닝 모델의 각 레이어마다 소모되는 에너지를 측정하는 실험을 진행하여, 특정 모델과 네트워크 환경에서의 최적의 성능 및 에너지를 갖는 레이어를 구하는 Computation Offloading 기법에 대해서 제안하게 되었다.

2. Related Work

Computation Offloading이란, 특정 장치의 하드웨어 제한을 극복하기 위해 주로 사용되는 기법으로, 과도한 계산이 필요한 연산에 대해서 특정 프로세서나 외부 플랫폼으로 전송하여 계산을 하는 방법이다. 특히, 드론의 활용도가 점점 높아지고, 딥러닝의 비약적인 발전을 거듭하며, Computation Offloading 분야 또한, 많은 연구가 진행되고 있다.

Kang, Yiping 등 7명은AlexNet 모델을 기반으로 다양한 네트워크 환경과 CPU 및 GPU를 이용한 추론 방식을 이용하여, 각 환경 간의 가장 효과적인 레이어를 추출하는 방법에 대해 제안하였다[1].

Qinglin Yang 등 5명은 모바일 디바이스 환경에서의 CNN(Convolution Neural Network) 추론 연산의 일부를 에지 컴퓨팅 사이트로 오프로드 하는 것을 제안하였다. 이 부분에서 효과적인 데이터 전달을 위해 배치 단위로 데이터를 보내는 알고리즘과 에지 사이트의 여러 개의 배치 데이터를 처리하려고 할 때, 각 배치당 예상된 작업 시간을 계산하여, 가장 최소로 소요되는 배치를 선택하여 먼저 처리하는 on-line 알고리즘을 제안하였다[2].

Mach, Pavel 등 2명은 실시간 어플리케이션 및 대용량 데이터 처리를 위한Mobile Edge Computing(MED) 기법의 특징 및 문제점을 설명하며, MEC 표준화 방법에 대해 제안하였다[3].

Yu, Shuai 등 3명은 전체적인 실행비용을 최소화 하기 위해 다중 라벨 분류 딥러닝 프레임워크를 제안하였다. 여기서 라벨은 오프로드를 하지 않음(NOS), 무작위 오프로드(ROS), 다중 레이블 선형 분류기 기반 오프로드(MOS), 총 오프로드 계획(TOS) 등의 여러 가지 라벨 값이 존재하고, 실험 결과 각 라벨 마다 실행 비용을 절감할 수 있음을 보여주었다[4].

앞서 말했듯이, 일반적으로 드론은 배터리로 동작하기 때문에, 효율성 고려와 무선을 통한 데이터 송수신의 경우 대역폭이 제한되는 문제점 등 여러가지가 제한적이다. 민홍 외 3명의 제안방법에서 는 공동의 목표를 공유하는 다수의 드론을 활용하여 클라우드에서 데이터를 처리하는 시스템에서 다수의 드론들이 협업했을 때의 비용과 클라우드에서 수행했을 때의 비용을 비교를 한다. 만약 데이터를 수집하면, 각 드론에서 작업 시 수행 완료시간이 동일한 성능을 가진다는 가정 하에 전체 작업량에서 드론 개수(n)으로 나누어 처리하여 수행 시간을 통계적으로 분석하여, 드론 자체내에서 데이터를 처리할 것인지 아니면 클라우드로 데이터를 전송하여 처리를 할 것인지 대한 결정을 하게 된다. [5]

이런 점에서, 본 논문은 단일 드론시스템이기 때문에 오프로딩 방식이더라도, 효율성 차이가 크게 차이가 날 것이다. 만약에 드론을 개수를 증가 시켜 이와 같은 제안방법으로, 탑재된 딥러닝 모델에서 컴퓨팅 작업의 최적화된 지점를 찾는다면, 최적점이 다를 것이다.

또한 데이터를 오프로딩하는 방법 중에 어느 지점에서 무선으로 통한 데이터 통신이 이루어지 는가에 따라 효율성이 달라질 수 도 있다. 최종석 외 2명이 제안한 방법은 망 이동에 따른 위치를 측정하고, UE(User equipment)의 셀 단위 이동 좌표를 경로를 모델에 적용하여 맵을 만든 뒤 이를 기반한 학습을 통하여 사용자 이동 패턴과 체류 구역에 따라 오프로딩을 사용하기 효율적인 위치를 찾는다. [6]

즉 드론으로 분석을 한 뒤에 무선통신을 하기 위해서, 데이터수집과 전처리 과정을 하면서, 드론의 이동패턴과 체류구역에 분석을 통해 효율적인 무선통신의 위치를 찾는 과정을 거친다면, 배터리사용과 무선통신 효율성은 더욱 올라 갈 것이다.

3. Mobile Edge Computing

3.1 Environment Set up

저전력 컴퓨팅 디바이스들에 대해 RasberryPi B+, NVIDIA Jetson Nano, RasberryPi3 B+ & Google Coral USB Acceleator, 총 3가지의 경우에 대한 실험을 진행하였고, 프로그램 실행 간 전력 측정은 외부 전력 공급 방식을 이용하여MONSOON사의 Power Monitor와 UM25C 장비를 이용하였으며, Power Monitor는 1초에 최대 5000번, UM25C는 초당 1번 값을 측정 할 수 있다. 관련 하드웨어에 대한 세부 사항들은 아래와 같다.

Table 1: Mobile Platform Specifications - Raspberry Pi 3 B+

|  |  |
| --- | --- |
| Hardware | Specifications |
| System | Raspberry Pi 3 B+ |
| CPU | 1.4-GHz Quad Core Broadcam BCM2837B0 (Cortex A53) |
| Memory | 1 GB LPDDR2 SDRAM |
| GPU | 400 MHz VideoCore IV |

Table 2: Mobile Platform Specifications - NVIDIA Jetson Nano

|  |  |
| --- | --- |
| Hardware | Specifications |
| System | NVIDIA Jetson Nano |
| AI Accelerator | 128 Cuda Cores |
| CPU | 1.4-GHz 64-bit Quad Core ARM  Cortex-A57 MPCore |
| Memory | 4 GB LPDDR4 |
| GPU | 128-Core Nvidia Maxwell |

Table 3: Mobile Platform Specifications – Google Coral USB Accelerator

|  |  |
| --- | --- |
| Hardware | Specifications |
| System | Coral USB Accelerator |
| ML accelerator | Google Edge TPU coprocessor |
| Connector | USB Type-C |
| Dimensions | 65mm x 30mm |

Table 4: Hardware Specifications – Monsoon Power Monitor AAA10F

|  |  |  |
| --- | --- | --- |
| Hardware | Specifications | |
| System | Monsoon Power Monitor AAA10F | |
| Software Version | 5.0.0.25 | |
| Component | Minimum | Maximum |
| Input Voltage range  (Main Channel) | 0.8 V | 13.5 V |
| Input Voltage range  (USB Channel) | 2.1 V | 5.4 V |
| Input Voltage range  (Auxiliary Channel) | 0.5 V | 5.5 V |

Table 5: Hardware Specifications – UM25C

|  |  |
| --- | --- |
| Hardware | Specifications |
| System | UM25C |
| Screen | 144 inch Color LCD Display |
| Voltage range | 4 – 24.000 V |
| Current range | 0 – 5.0000A |

3.2 Model Setting

실험을 위해, 사용한 모델은 CNN(Convolution Neural Network)기반의 VGG16 유사 모델로 진행하였고, 모델에 대한 세부 사항은 다음 Figure 1과 같다.

|  |
| --- |
| ConvNet Configuration |
| 17 weight layers |
| Input (112 x 112 RGB image) |
| Conv3-64  Conv3-64 |
| Maxpool |
| Conv3-128  Conv3-128 |
| Maxpool |
| Conv3-256  Conv3-256  Conv3-256 |
| Maxpool |
| Conv3-512  Conv3-512  Conv3-512 |
| Maxpool |
| FC-4096 |
| FC-2048 |
| FC-2 |

Figure 1: Specification of Model based on VGG16

해당 모델을 Raspberry pi 3 B+ 와 NVIDIA Jetson Nano 는 일반적인32-bit Float model로 실험하였고, Google Coral USB Accelerator는 기기 특성 상 TPU환경에 맞추기 위해 양자화(Quantized) 과정을 거쳐 8-bit Integer model로 변환한 모델을 통해 실험하였다. 또한 세부적인 레이어 별 측정을 위해 각 해당 레이어까지 별로 분할되어 저장된 모델들을 훈련이미지와 동일한 크기의 테스트 이미지에 반복해서 추론하고 소모 에너지 및 시간의 평균 값을 측정하였다.

3.3 Energy Consumption Optimal Point Algorithm

다음 Figure 2. 와 같은 알고리즘을 통해 에너지 소모 최적점을 산출하도록 한다.

|  |  |
| --- | --- |
| 1 | Ec(x) |
| 2 | Return “energy used to Layer x” |
| 3 | En(x) |
| 4 | Return “energy used to transmit the output data of Layer x” |
| 5 | FindMiniumCutLayer() |
| 6 | Lm = 0 |
| 7 | E\_l(Lm) = MAX\_FLOAT |
| 8 | For L in range(1, MAX\_LAYERS) |
| 9 | E\_l(L) = Ec(L) + En(L) |
| 10 | If E\_l(L) < E\_l(Lm) then |
| 11 | Lm = L |
| 12 | Return Lm |

Figure 2. 에너지 소모 최적점 계산 알고리즘

3.4 Examing the mobile edge - Raspberry pi 3 B+

해당 실험의 환경은 3.2에서 언급한 바와 같이, 각 레이어를 분할하여 저장한 모델을 Raspberry Pi 3 B+ 에 탑재하고 추론 간 소모되는 시간과 에너지를 측정 하였다.

Raspberry Pi 3 B+는 다른 기기의 도움 없이는 자체적으로 CPU 연산만 가능하며, 해당 실험도 CPU 추론을 진행하였다.

Table 6: Raspberry pi 3 B+ Edge Computing

|  |  |  |  |
| --- | --- | --- | --- |
|  | 17 layers | 실행시간(Sec) | 소모전력(W) |
| 1 | Conv3-64 | 0.0377 | 4.084 |
| 2 | Conv3-64 | 0.1466 | 4.644 |
| 3 | Maxpool | 0.1707 | 4.887 |
| 4 | Conv3-128 | 0.2331 | 4.943 |
| 5 | Conv3-128 | 0.3362 | 5.098 |
| 6 | Maxpool | 0.3533 | 5.111 |
| 7 | Conv3-256 | 0.4122 | 5.262 |
| 8 | Conv3-256 | 0.5199 | 5.312 |
| 9 | Conv3-256 | 0.6321 | 5.412 |
| 10 | Maxpool | 0.6396 | 5.432 |
| 11 | Conv3-512 | 0.7743 | 5.553 |
| 12 | Conv3-512 | 0.8442 | 5.572 |
| 13 | Conv3-512 | 0.9713 | 5.607 |
| 14 | Maxpool | 0.9778 | 5.625 |
| 15 | FC-4096 | M.S | M.S |
| 16 | FC-2048 | M.S | M.S |
| 17 | FC-2 | M.S | M.S |

Table 6. 을 살펴보면, 다른 기기에 비해 이미지 한 개당 추론 시간이 높은 것을 알 수 있다. 또한, 15Layer에서부터는 Fully Connected(FC) Layer에 해당하는 부분으로, Convolution Filter에서 추출한 특징 값들을 FC Layer에 연결하려고 할 때, Raspberry Pi 3 B+에서는 메모리 부족으로 인해 더 이상 실행할 수 없었다.

3.5 Examing the mobile edge – NVIDIA Jetson Nano

해당 실험의 환경은 위에서 레이어를 분할하여 저장한 모델을 동일하게 사용하였고, NVIDIA Jetson Nano는 기기에 내장되어 있는 GPU인 128-Core NVIDIA Maxwell을 추론에 사용하였다.

Table 7: Jetson Nano Edge Computing

|  |  |  |  |
| --- | --- | --- | --- |
|  | 17 layers | 실행시간(Sec) | 소모전력(W) |
| 1 | Conv3-64 | 0.017 | 4.896 |
| 2 | Conv3-64 | 0.019 | 6.027 |
| 3 | Maxpool | 0.02 | 7.133 |
| 4 | Conv3-128 | 0.021 | 7.316 |
| 5 | Conv3-128 | 0.025 | 7.675 |
| 6 | Maxpool | 0.027 | 7.929 |
| 7 | Conv3-256 | 0.032 | 8.047 |
| 8 | Conv3-256 | 0.038 | 8.283 |
| 9 | Conv3-256 | 0.046 | 8.368 |
| 10 | Maxpool | 0.046 | 8.474 |
| 11 | Conv3-512 | 0.057 | 8.502 |
| 12 | Conv3-512 | 0.077 | 8.504 |
| 13 | Conv3-512 | 0.096 | 8.612 |
| 14 | Maxpool | 0.096 | 8.682 |
| 15 | FC-4096 | M.S | M.S |
| 16 | FC-2048 | M.S | M.S |
| 17 | FC-2 | M.S | M.S |

Table 7. 을 살펴보면, Jetson Nano로 진행한 실험은 CPU연산을 사용한 Raspberry Pi 3 B+보다 수 배 빠른 실행시간을 보였으나, 소모 전력에서는 더 높은 소모량을 보였다. 3.4와 똑같이FC Layer가 포함된15번째 Layer 부터는 동일하게 메모리 부족으로 인해 실행이 불가능하였다.

3.6 Examing the mobile edge – Coral USB Accelerator

임베디드 보드에 탑재한 머신러닝 모델에 대해 추론을 가속화시키기 위해 Google사의 Coral USB Accelerator를 장착하였다. 이 장치는 TPU연산을 제공하며 보조 프로세서 역할을 수행하는 USB장치이다. Coral USB Accelerator를 사용하기 위해서는 설계한 모델를 Tfite확장자에 가중치를 8비트 양자화로 변환해 주어야 수행이 가능하다. 변환 과정은 다음과 같은 순서로 바뀌게 된다.

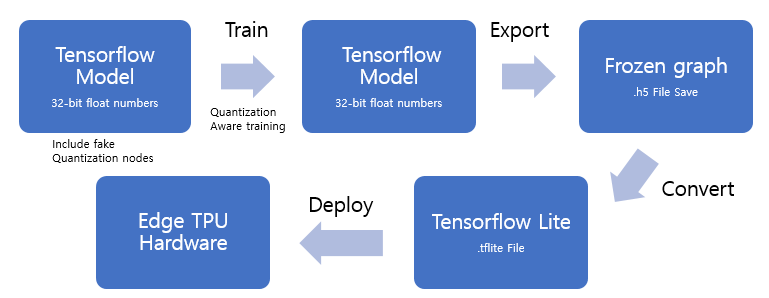


Figure 3: Process of Tflite convert and 8bit Quantification

머신 러닝에 사용되는 가중치 값은 대체로 32비트 부동 소수점 값이다. 즉, 각 층의 최대, 최소 값을 저장하고, 부동 소수점 값 256개 범위 이내에서 가장 가까운 실수 값 8비트 정수형으로 압축시킨다. 그러면 8비트의 값을 읽어오는 것은 부동 소수점을 읽어오는 것의 25%에 해당되는 메모리 대역폭만 요구하기 때문에 정확도가 거의 떨어지지않고, 모델의 크기가 줄어듦 으로서 모델들을 임베디드 보드에 수행하는 것보다, Coral USB Accelerator를 사용하여 빠르고, 저전력으로 동작시켜 추론 속도가 빨라질 수 있다.

Table 8: with Coral USB Accelerator Edge Computing

|  |  |  |  |
| --- | --- | --- | --- |
| layer | 17 layers | 실행시간(Sec) | 소모전력(W) |
| 1 | Conv3-64 | 0.027 | 1.1459 |
| 2 | Conv3-64 | 0.314 | 1.2464 |
| 3 | Maxpool | 0.370 | 1.2272 |
| 4 | Conv3-128 | 0.394 | 1.2356 |
| 5 | Conv3-128 | 0.473 | 1.2599 |
| 6 | Maxpool | 0.481 | 1.2619 |
| 7 | Conv3-256 | 0.625 | 1.3008 |
| 8 | Conv3-256 | 0.702 | 1.3121 |
| 9 | Conv3-256 | 0.887 | 1.3715 |
| 10 | Maxpool | 0.911 | 1.3787 |
| 11 | Conv3-512 | 0.935 | 1.3879 |
| 12 | Conv3-512 | 1.130 | 1.4902 |
| 13 | Conv3-512 | 1.573 | 1.5737 |
| 14 | Maxpool | 1.612 | 1.5632 |
| 15 | FC-4096 | 1.628 | 3.0036 |
| 16 | FC-2048 | 1.636 | 3.1770 |
| 17 | FC-2 | 1.646 | 4.7831 |

VGG16모델을 17개의layer으로 잘라, 모델들을 추론하여, Table 6과 비교했을 때, 실행시간도 빠르며, 소모전력도 2배차이로 저전력 소모인 것을 확인 할 수 있으며, Raspberry pi 3 B+ , Jetson Nano보다 전반적으로 향상 됨을 알 수 있다.

4. Cloud Computing

4.1 Environment Setting

네트워크 전송 소모 에너지 측정을 위한 환경 설정은 먼저, 동일 Ethernet으로 연결된 서버와 클라이언트 간의 통신 환경을 구축하였다. 클라이언트는 앞서 실험 대상인 임베디드 기기가 대상이 된다. 여기서 세부적인 비교를 위해 TCP/IP와 UDP 프로토콜을 나누어서 소모 에너지를 측정하도록 한다.

실험 방법은 Raspberry PI 3+와 Jetson Nano의 각 환경에서 10Mb~100Mb로 10Mb씩 용량을 나누어 파일을 전송하는데 사용되는 시간과 에너지를 여러 번 반복 실험하여 산출된 평균값을 측정하도록 한다.

4.2 Examing the Network in TCP/IP

다음 Table 9. 는 TCP/IP 네트워크 상에서 전송 소모 에너지를 측정한 것이다. 실험 결과, 파일 크기가 증가함에 따라 소모 에너지도 선형적으로 증가한다.

또한 평균적으로 Raspberry PI 3+ 의 소모 에너지가 Jetson Nano에 비해 약 4배정도 높은 것으로 확인되었다.

Table 9: Network Consumption – TCP/IP

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20MB | 40MB | 60MB | 80MB | 100MB |
| Raspberry PI 3+ | 0.666 | 1.401 | 2.100 | 2.791 | 3.507 |
| Jetson Nano | 0.178 | 0.357 | 0.546 | 0.732 | 0.952 |

단위 : J

4.3 Examing the Network in UDP

다음 Table 10. 는 UDP 네트워크 상에서 전송 소모 에너지를 측정한 것이다. 실험 결과, 소모 에너지 추이는TCP/IP와 동일하며 에너지 소모량은TCP/IP 에 비해 비교적 높게 측정되었다.

Table 10: Network Consumption – UDP

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20MB | 40MB | 60MB | 80MB | 100MB |
| Raspberry PI 3+ | 0.720 | 1.429 | 2.147 | 2.879 | 3.656 |
| Jetson Nano | 0.201 | 0.368 | 0.568 | 0.749 | 0.911 |

단위 : J

5. 실험결과

다음 Figure 4. 는 모델에 대한 엣지 컴퓨팅시의 레이어별 에너지 소모량을 비교한 것으로 오프로딩 최적점은 14번째 layer이다. 즉 순수 추론을 통한 에너지 소모량 고려시14번쨰 Layer 까지 해당 기기에서 추론 진행을 하고 나머지 데이터는 네트워크 서버로 전송하여 나머지 작업을 하는 것이 최적의 효율을 보인다는 것이다.

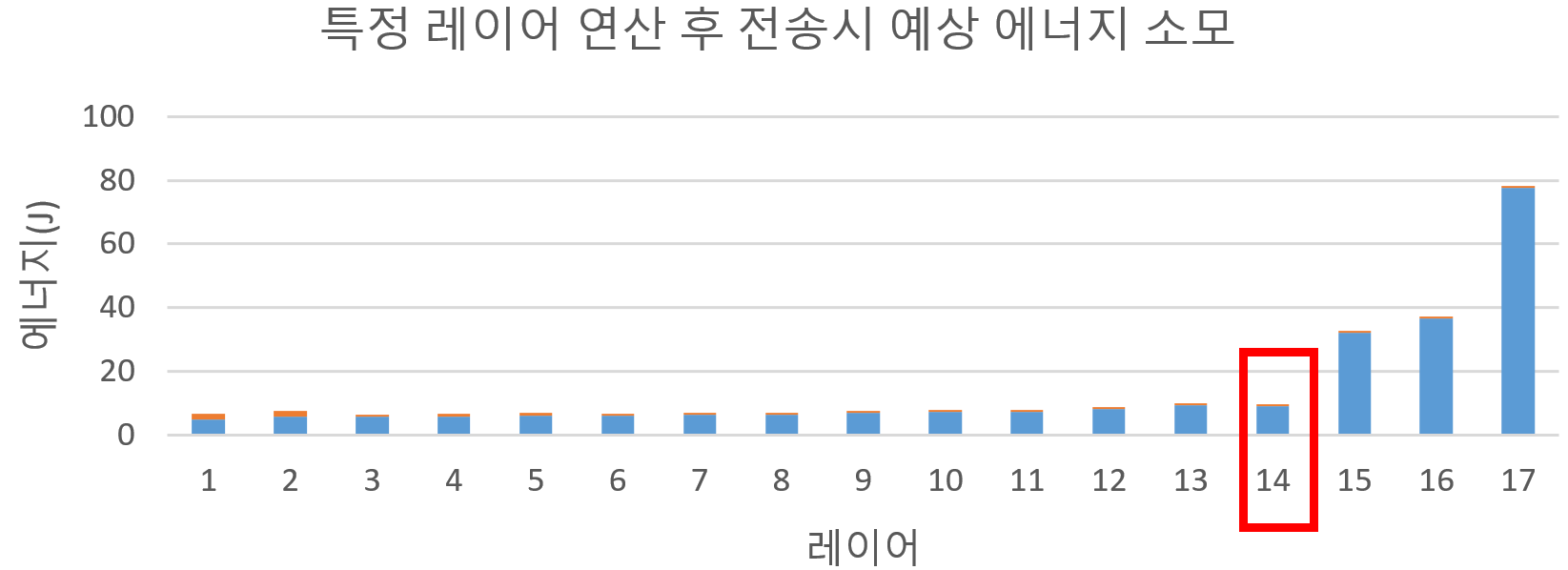


Figure 4. 특정 레이어 연산 후 전송 시 예상 에너지 소모량

다음 Table 11. 는 모델별 layer에 대한 Output size를 나타내며 네트워크 전송 간 데이터의 양을 나타낸다.

|  |  |
| --- | --- |
| 17 weight layers | Output Size  (Transfer Data) |
| Conv3-64  Conv3-64 | (None, 112, 112, 64)  (None, 112, 112, 64) |
| Maxpool | (None , 56, 56, 64) |
| Conv3-128  Conv3-128 | (None , 56, 56, 128)  (None , 56, 56, 128) |
| Maxpool | (None , 28, 28, 128) |
| Conv3-256  Conv3-256  Conv3-256 | (None , 28, 28, 256)  (None , 28, 28, 256)  (None , 28, 28, 256) |
| Maxpool | (None, 14, 14, 256) |
| Conv3-512  Conv3-512  Conv3-512 | (None, 14, 14, 512)  (None, 14, 14, 512)  (None, 14, 14, 512) |
| Maxpool | (None, 7, 7, 512) |
| FC-4096 | (None, 4096) |
| FC-2048 | (None, 2048) |
| FC-2 | (None, 2) |

Table 11. Output Size per layer (Transmission)

다음 Figure 5. 는 네트워크로 데이터 전송 간 에너지 소모량이 합산 된 에너지 소모량을 비교한 것으로, 네트워크 에너지 소모량이 합산 되었을 때의 오프로딩 최적점은 3번째 layer이다. 이는 즉 네트워크 데이터 전송에너지 소모량 까지 고려하였을 때 3번째 layer까지 해당 기기에서 추론 진행을 하고 나머지 데이터는 네트워크 서버로 전송하여 나머지 작업을 하는 것이 최적의 효율을 보인다는 것이다.

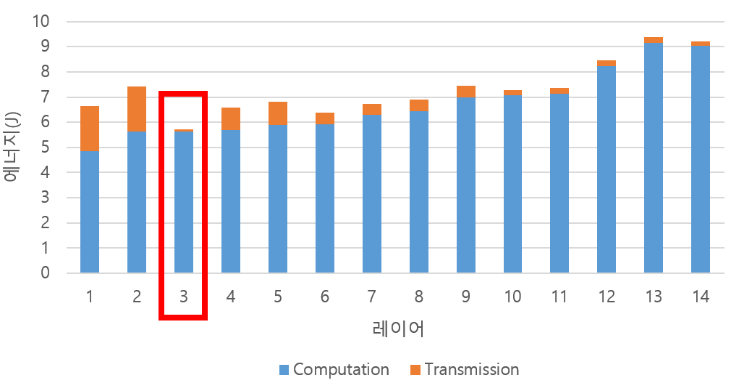


Figure 5. 네트워크 전송 소모량이 더해진 최종 그래프

6. 결론 및 향후 연구

- 본 연구에서는 드론 원격 센싱 기반의 에너지 소모 최적화를 위한 딥러닝 연산 오프로딩에 대해 연구 및 실험해 보았다. 최적 에너지 소모 알고리즘을 설계하고, 레이어와 에너지 간의 관계 모델을 도출함으로서 해당 모델의 오프로딩 최적점을 제안할 수 있었다.

이 최적점은 해당 모델에서와 비슷한 형태의 모델에서만 유의한 결과로 다른 형태의 모델 에서는 최적점 정보가 달라 질 수 있다.

Reference

[1] Kang, Yiping, et al. "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge." *ACM SIGARCH Computer Architecture News* 45.1 (2017): 615-629.

[2] Yang, Qinglin, et al. "Computation offloading for fast CNN inference in edge computing." *Proceedings of the Conference on Research in Adaptive and Convergent Systems*. 2019.

[3] Mach, Pavel, and Zdenek Becvar. "Mobile edge computing: A survey on architecture and computation offloading." *IEEE Communications Surveys & Tutorials* 19.3 (2017): 1628-1656.

[4] Yu, Shuai, Xin Wang, and Rami Langar. "Computation offloading for mobile edge computing: A deep learning approach." *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, 2017.

[5] 민홍, et al. "군집 드론의 안정적 데이터 처리를 위한 오프로딩 기법." *정보과학회논문지* 45.10 (2018): 990-995.

[6] 최종석, 송관호, and 신용태. "LTE 환경에서 망이동패턴 학습을 통한 효율적 Data-Offloading 방법연구." *한국통신학회 학술대회논문집* (2013): 33-34.