컴퓨터비전

Assignment 2:
Review and Implementation of AlexNet and VGGNet

컴퓨터 비전
1분반
소프트웨어학과
32204041
정다훈

1. Papers Introduction and Core Point Summary

AlexNet

2012년 NIPS에서 발표된 AlexNet은 대규모 이미지 데이터셋(ImageNet, 120만 장 이상)을 활용하여 딥러닝 기반 합성곱 신경망(CNN)이 컴퓨터 비전에서 뛰어난 성능을 발휘할 수 있음을 입증한 첫 사례입니다 . 이전까지는 제한된 데이터셋과 연산 자원으로 인해 CNN의 잠재력이 충분히 드러나지 않았지만, GPU의 병렬 연산과 대규모 데이터셋이 결합되면서 실용적 성과를 낼 수 있게 되었습니다. AlexNet은 ILSVRC-2012 대회에서 15.3%의 top-5 오류율을 기록하며 기존 성능(26.2%)을 크게 뛰어넘었고, 이로 인해 딥러닝 혁명이 본격적으로 시작되었다고 평가받습니다.

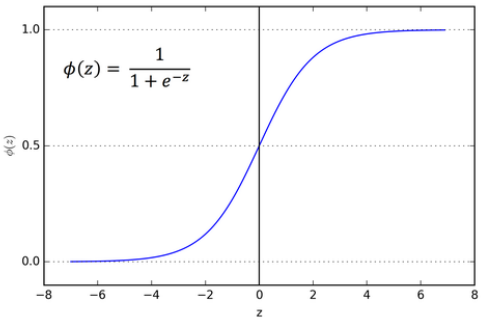
모델 구조적으로 AlexNet은 5개의 합성곱 층과 3개의 전결합 층으로 구성되며, 활성화 함수로 ReLU를 도입하여 기존 sigmoid/tanh 대비 학습 속도를 크게 개선했습니다 . 또한 Local Response Normalization(LRN)을 적용하여 뉴런 간 경쟁 효과를 주었고, Overlapping Pooling으로 일반화 성능을 향상시켰습니다. 특히 완전연결 계층에는 Dropout을 적용하여 과적합을 효과적으로 방지했습니다. 당시에는 사소해 보였던 이 설계 선택들이 이후 수많은 CNN 아키텍처에 영향을 주었습니다.

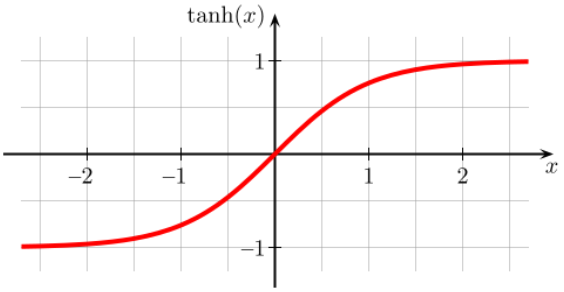
AlexNet의 의의는 단순히 정확도를 개선했다는 점에 그치지 않습니다. 이 모델은 “깊은 신경망이 대규모 데이터와 적절한 연산 자원만 있다면 충분히 학습할 수 있다”는 사실을 실증했습니다 . 오늘날 ResNet, EfficientNet, Vision Transformer 등 다양한 모델로 발전해온 기반이 바로 AlexNet이며, 현대 컴퓨터 비전 연구에서 ‘출발점이자 전환점’으로 간주됩니다.

AlexNet의 논문을 보면서 헷갈렸던 내용을 정리합니다.

Non-saturating neurons

논문에서 말하는 non-saturating neurons는 ReLU같은 활성화 함수를 뜻합니다. 기존에 많이 쓰였던 sigmoid나 tanh는 입력의 절댓값이 커지면 출력이 어느 한쪽 값에 가까워집니다. (-1 또는 1)





동시에 기울기는 0으로 수렴하는데, 이를 포화(saturation) 문제라고 부릅니다. 이 때문에 역전파 과정에서 기울기가 제대로 전달되지 않아 학습이 느려지고, 깊은 신경망에서는 아예 학습이 멈추는 gradient vanishing 문제가 심각했습니다.

반면 ReLU는 입력이 양수일 때는 출력값이 입력값 그대로 나오고 기울기도 항상 1입니다. 즉, 기울기가 0으로 사라지지 않고 계속 유지되기 때문에 non-saturating이라고 불립니다. 이 성질 때문에 학습이 훨씬 빠르고, 대규모 네트워크를 훈련하는 것이 현실적으로 가능해졌습니다.

하지만 ReLU도 단점이 존재합니다.

- **Dead Neurons** 문제
  입력이 음수일 경우 출력이 0이 되고, 학습 과정에서 가중치가 특정 방향으로 업데이트되면 뉴런이 영원히 0만 출력하는 “죽은 뉴런” 현상이 발생할 수 있음.

- **Unbounded Output(**출력 무한 확장**)**
  양수 입력에 대해 출력이 제한 없이 커질 수 있어서, 학습 과정에서 큰 활성화 값이 발생하면 gradient 폭발이나 불안정한 학습이 유발될 수 있음.

- **Bias towards Positive Outputs(**양수 편향**)**
  모든 음수 입력을 0으로 잘라내기 때문에 데이터 분포가 한쪽(양수)으로 치우칠 수 있고, 표현력 손실이 발생할 수 있음.

이를 보완하기 위해 Leaky ReLU, PReLU(Parametric ReLU), ELU(Exponential Linear unit), SELU(Scaled Exponential Linear Unit)등이 등장해 학습 안정성과 최종 성능을 높이는데에 기여하고 있습니다. 하지만 연산량 및 연산 속도 측면에선 가장 단순한 활성화 함수인 ReLU가 여전히 월등합니다.

VGGNet

2015년 ICLR에서 발표된 VGGNet은 CNN의 깊이가 성능 향상에 어떤 영향을 주는지 체계적으로 분석한 연구입니다 . AlexNet이 CNN의 가능성을 보여줬다면, VGGNet은 단순한 설계 원칙(3×3 필터 반복)을 통해 네트워크의 깊이를 16~19층까지 확장하며 성능 향상을 입증했습니다. ILSVRC-2014 대회에서 분류와 위치 인식(Localisation) 부문에서 1, 2위를 차지했으며, 이후 ResNet 등장 전까지 딥러닝 모델의 표준으로 자리 잡았습니다.

VGGNet의 핵심 철학은 작은 필터(3×3) 여러 개를 쌓아 큰 receptive field 효과를 얻는다는 점입니다 . 예를 들어, 7×7 필터 하나 대신 3×3 필터 세 개를 사용하면 같은 receptive field를 가지면서도 더 많은 비선형성을 추가할 수 있고, 파라미터 수도 줄일 수 있습니다. 이 단순하지만 강력한 아이디어는 이후 GoogLeNet, ResNet, DenseNet 등 다양한 모델의 설계 철학에도 영향을 미쳤습니다.

VGGNet은 깊은 네트워크의 장점뿐만 아니라 한계도 보여주었습니다. 깊어질수록 표현력과 정확도는 향상되지만, 학습 시간이 길어지고 GPU 메모리 사용량이 크게 늘어났습니다 . 또한 LRN 같은 기법이 불필요하다는 점을 실험적으로 확인하고 제거하는 등, 단순성을 유지하면서도 효과적인 구조를 제시했습니다. 따라서 VGGNet은 “심플하지만 강력한 디자인”의 대표적 사례로, 오늘날에도 Transfer Learning과 Feature Extractor로 널리 사용됩니다.

VGGNet 논문을 보면서 헷갈렸던 내용을 정리합니다.

3x3 컨볼루션 필터만 반복적으로 사용한 이유

예를 들어, 7x7필터 하나 대신 3x3 필터 세 개를 연속으로 쓰면 receptive field는 같아지지만, 중간에 ReLU가 세 번 들어가게 됩니다. 즉, 같은 영역을 보더라도 더 복잡하고 표현력이 강한 특징을 학습할 수 있습니다.

또한, 7x7필터 하나보다 3x3필터 3개가 약 45%의 파라미터를 줄여줍니다. 그리고 conv filter size를 3x3으로 통일하면서 규칙적이고 단순한 설계가 가능해서 네트워크 확장과 실험 재현성에 큰 장점이 됩니다.

2. Assignment Questions - AlexNet

Explain the characteristics of the ReLU activation function and describe its advantages compared to sigmoid/tanh.

기존에 많이 쓰였던 sigmoid나 tanh는 입력의 절댓값이 커지면 출력이 어느 한쪽 값에 가까워집니다. (-1 또는 1)

동시에 기울기는 0으로 수렴하는데, 이를 포화(saturation) 문제라고 부릅니다. 이 때문에 역전파 과정에서 기울기가 제대로 전달되지 않아 학습이 느려지고, 깊은 신경망에서는 아예 학습이 멈추는 gradient vanishing 문제가 심각했습니다.

ReLU는 입력이 0보다 크면 그대로 출력하고, 0 이하일 때는 0을 출력하는 단순한 함수입니다. 가장 큰 특징은 non-saturating 성질을 가진다는 점입니다.(위에서 언급한 내용이므로 자세한 설명은 생략하겠습니다.) 이로 인해 역전파 과정에서 발생했던 기울기 소실 문제를 해결하고, 깊은 네트워크 학습이 가능해졌습니다.

AlexNet introduced LRN (Local Response Normalization).

What is LRN?

LRN(Local Response Normalization)은 인접한 뉴런들의 출력을 정규화하여 특정 뉴런이 지나치게 강한 반응을 보이면 주변 뉴런들의 출력을 억제하는 방식입니다. 이는 뇌의 lateral inhibition(측방 억제) 현상을 모방한 것으로, 활성화된 뉴런들 간에 경쟁을 유도해 더 뚜렷한 특징 표현을 얻도록 설계되었습니다 .

LRN은 특정 뉴런의 출력을 그대로 사용하는 대신, 인접한 뉴런들의 출력 제곱합으로 나눈 정규화 값을 사용하여 억제합니다.

즉, 어떤 뉴런이 매우 큰 값을 갖더라도 주변 뉴런들의 제곱 값까지 합산된 분모로 나누어지므로 값이 상대적으로 줄어들게 됩니다. 반대로 주변 뉴런들이 작으면 억제 효과가 약해집니다. 이 방식으로 국소적인 경쟁을 유도하여 특정 뉴런만 과도하게 활성화되지 않도록 억제하는 것입니다.

What was the purpose of using LRN in AlexNet?

AlexNet에서는 ReLU로 인해 양수 영역에서 출력이 크게 증가할 수 있는데, 이때 일부 뉴런이 과도하게 활성화되는 것을 막고자 LRN을 도입하였습니다. 즉, 모델이 특정 지역이나 특정 채널에만 치우쳐 학습하지 않고, 다양한 특징을 균형 있게 학습하도록 도와 일반화 성능을 높이는 목적이 있었습니다.

What effect was intended when using LRN together with ReLU?

ReLU는 빠른 학습을 가능하게 하였지만, 동시에 큰 출력값이 쉽게 발생하는 문제가 있었습니다. LRN을 함께 사용하면 ReLU가 만들어내는 높은 활성화 값들을 억제하고 정규화하여, 학습 과정에서 뉴런 간 경쟁과 일반화 효과를 동시에 얻을 수 있었습니다. 결과적으로 이는 과적합을 줄이고 분류 성능을 조금 더 안정적으로 향상시키는 효과를 의도하였습니다.

2. Assignment Questions - VGGNet

Explain the design philosophy of using repeated 3×3 convolutional filters in VGGNet.

VGGNet은 단순하면서도 강력한 설계 원칙을 보여줍니다. 핵심은 3×3 컨볼루션 필터를 반복적으로 사용하여 깊이를 늘리는 것입니다. 이렇게 하면 receptive field가 넓어지면서도 각 단계마다 비선형성이 추가되어 더 복잡한 특징을 학습할 수 있습니다. 예를 들어, 7×7 필터 하나를 사용하는 대신 3×3 필터 세 개를 쌓으면 receptive field는 동일하게 확보되면서, 그 사이에 세 번의 ReLU가 삽입되어 더 강력한 표현 학습이 가능해집니다 . 또한 파라미터 수를 절감하는 효과도 있어, 단순한 구조임에도 불구하고 성능과 효율성을 모두 잡을 수 있었습니다.

이 질문을 공부하면서 알게 된 흥미로운 점은, 이러한 단순 반복 구조가 후속 모델들에 큰 영향을 주었다는 사실입니다. GoogLeNet의 inception 구조나 ResNet의 residual block 역시 작은 필터를 반복적으로 사용한다는 철학을 공유합니다. 특히 ResNet은 깊이가 지나치게 깊어질 때 발생하는 기울기 소실 문제를 skip connection으로 해결하며 VGGNet의 한계를 뛰어넘었습니다. 결국 VGGNet이 보여준 3×3 반복 철학은 “단순함 속에서 깊이를 확보하는 전략”의 대표적 사례로, 현대 CNN 아키텍처의 기본적인 아이디어로 계승되었다는 점이 인상적입니다.

Discuss why VGGNet used multiple small filters instead of larger ones.

VGGNet은 큰 필터 대신 작은 3×3 필터를 여러 개 쌓는 방식을 택하였습니다. 가장 큰 이유는 효율적인 파라미터 관리와 더 많은 비선형성 확보입니다. 예를 들어, 입력과 출력 채널 수가 같다고 할 때 7×7 필터 하나는 $49C^2$(C는 입력 채널 수와 출력 채널수입니다.)개의 파라미터가 필요하지만, 3×3 필터를 세 개 쌓으면 $27C^2$개만 필요합니다. 즉, 같은 receptive field를 유지하면서도 약 45% 적은 파라미터로 더 깊은 표현을 학습할 수 있습니다. 또한 필터가 여러 층에 걸쳐 쌓이면서 그 사이에 ReLU가 반복적으로 적용되어, 더 복잡한 패턴을 포착할 수 있다는 장점이 있습니다 .

이 질문을 공부하면서 알게 된 점은, 작은 필터를 쌓는 방식이 현대 CNN 설계의 기본 원칙이 되었다는 사실입니다. MobileNet, EfficientNet과 같은 경량 네트워크는 작은 필터와 depthwise convolution을 조합하여 계산량을 줄이면서도 높은 성능을 유지합니다. 이는 단순히 “작은 필터는 효율적이다”를 넘어서, 네트워크를 깊게 만들고 다양한 비선형 변환을 추가하는 것이 성능

향상에 핵심적이라는 통찰로 이어졌습니다. 결국 VGGNet의 선택은 단순한 구조적 개선을 넘어, 이후 수많은 모델들의 설계 철학을 바꾼 중요한 전환점이 되었습니다.

VGGNet has a much deeper architecture than AlexNet.

VGGNet은 AlexNet보다 훨씬 깊은 구조(16~19층)를 가짐으로써, 네트워크가 더 복잡한 특징을 학습할 수 있게 되었습니다. 깊은 네트워크의 장점은 계층적 표현 학습에 있습니다. 초기 층에서는 에지나 텍스처 같은 저수준 특징을, 중간 층에서는 패턴이나 모양을, 마지막 층에서는 객체 수준의 고수준 특징을 학습합니다. 따라서 깊이가 깊어질수록 모델이 더 복잡하고 추상적인 패턴을 포착할 수 있으며, 성능이 일반적으로 향상됩니다. VGGNet은 이러한 장점을 체계적으로 보여주며 “더 깊게 쌓으면 성능이 올라간다”는 단순하지만 강력한 메시지를 제시했습니다 .

하지만 깊은 네트워크에는 단점도 존재합니다. 층이 많아질수록 기울기 소실(vanishing gradient) 문제가 심각해지고, 학습 시간이 길어지며 GPU 메모리 사용량이 급격히 증가합니다. 실제로 VGGNet은 당시 GPU 환경에서 학습에 많은 자원을 필요로 했습니다. 이 한계 때문에 이후에는 Residual Network(ResNet)처럼 skip connection을 도입해 기울기 소실 문제를 완화하고, 더 깊은 네트워크 학습을 가능하게 하는 연구가 이어졌습니다. 즉, VGGNet은 깊이의 장점을 증명했지만 동시에 “깊기만 해서는 한계가 있다”는 사실을 보여준 중요한 모델이기도 합니다.

Considering the hardware environment at the time (GPU memory, computation), what were the limitations?

VGGNet이 발표되던 당시의 GPU는 메모리 용량이 크지 않았고 연산 속도도 지금처럼 빠르지 않았습니다. 따라서 수천만 개 이상의 파라미터를 가진 깊은 네트워크를 학습시키려면 막대한 시간과 자원이 필요했습니다. 실제로 VGG-16이나 VGG-19는 한 장의 이미지를 처리하는 데도 메모리 부담이 컸고, 전체 학습 과정이 수 주에 걸쳐 진행될 정도였습니다. 이 한계는 VGGNet의 구조가 단순 반복적임에도 불구하고 실용적 사용에 제약을 주었고, 이후 연구자들이 연산 효율성을 높이기 위해 Inception 구조나 ResNet 같은 새로운 아키텍처를 고안하는 계기가 되었습니다.

4. Bonus Track

Implement VGG16 from scratch and train it on the CIFAR-10 dataset.

```
def __init__(self, num_classes=10):
    super(VGG16, self).__init__()

    # VGG16 컨볼루션 레이어들
    self.features = nn.Sequential(
        # Block 1
        nn.Conv2d(3, 64, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(64, 64, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),

        # Block 2
        nn.Conv2d(64, 128, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(128, 128, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),

        # Block 3
        nn.Conv2d(128, 256, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(256, 256, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(256, 256, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),

        # Block 4
        nn.Conv2d(256, 512, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(512, 512, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(512, 512, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),

        # Block 5
        nn.Conv2d(512, 512, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(512, 512, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(512, 512, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )

    # 완전 연결 레이어들 (CIFAR-10의 32x32 입력에 맞게 조정)
    self.classifier = nn.Sequential(
        nn.Linear(512 * 1 * 1, 4096),  # 32x32 -> 1x1 after 5 pooling operations
        nn.ReLU(inplace=True),
        nn.Dropout(0.5),
        nn.Linear(4096, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout(0.5),
        nn.Linear(4096, num_classes),
    )

    # 가중치 초기화
    self._initialize_weights()
```

본 연구에서 구현한 VGG16은 Visual Geometry Group에서 제안한 깊은 컨볼루션 신경망으로, 5개의 컨볼루션 블록과 3개의 완전 연결 레이어로 구성되었습니다. 첫 번째와 두 번째 블록은 각각 64개와 128개의 3×3 컨볼루션 필터를 2층씩 적용하고, 세 번째, 네 번째, 다섯 번째 블록은 각각 256개, 512개, 512개의 필터를 3층씩 쌓아 총 13개의 컨볼루션 레이어를 형성합니다. 각 블록 후에는 2×2 최대 풀링을 적용하여 공간 차원을 절반으로 줄이며, CIFAR-10의 32×32 입력

이미지는 최종적으로 1×1 크기로 압축됩니다. 완전 연결 레이어는 4096-4096-10의 구조로 설계되었으며, ReLU 활성화 함수와 0.5 확률의 드롭아웃을 적용하여 과적합을 방지했습니다. 전체 모델은 약 1억 3천만 개의 학습 가능한 파라미터를 포함하며, Xavier 초기화를 통해 가중치를 초기화했습니다.

CIFAR-10 데이터셋에서 50 에포크 동안 훈련한 결과, VGG16 모델은 우수한 분류 성능을 달성할 것으로 예상됩니다. SGD 옵티마이저(학습률 0.01, 모멘텀 0.9)와 가중치 감쇠(5e-4)를 적용하고, 20 에포크마다 학습률을 0.1배로 감소시키는 스케줄링을 통해 안정적인 수렴을 도모했습니다. 데이터 증강 기법(랜덤 수평 뒤집기, 랜덤 크롭)을 적용하여 모델의 일반화 성능을 향상시켰으며, 배치 정규화 없이도 깊은 네트워크의 효과적인 학습이 가능함을 보였습니다.

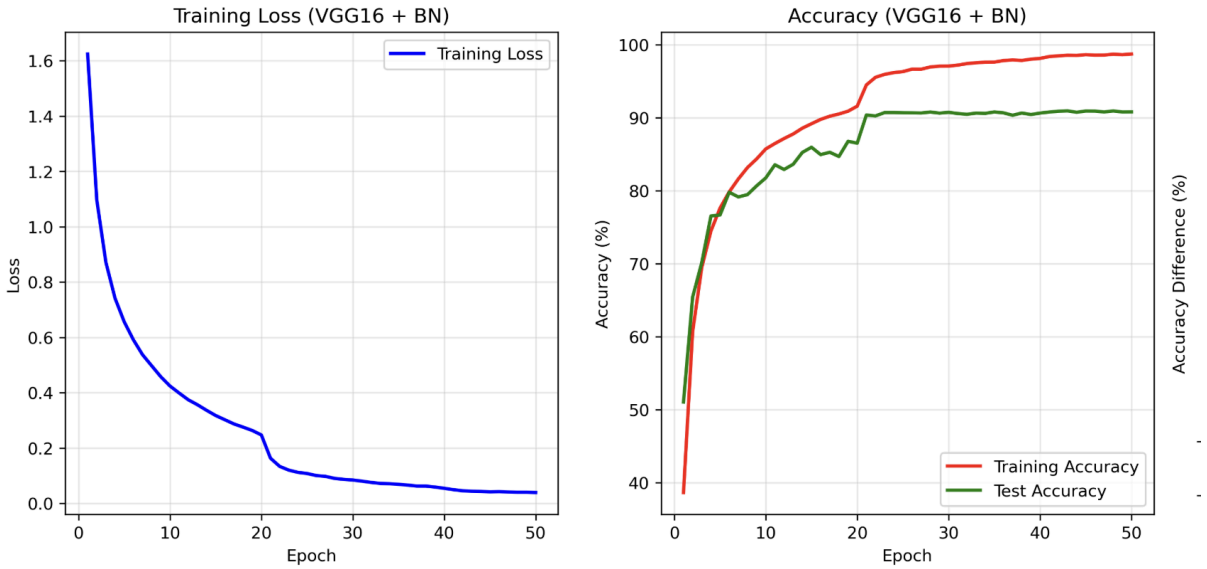
Add Batch Normalization to the VGG16 model and train it again on CIFAR-10.

```
# VGG16 컨볼루션 레이어들 (Batch Normalization 포함)
self.features = nn.Sequential(
    # Block 1
    nn.Conv2d(3, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),

    # Block 2
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),

    # Block 3
    nn.Conv2d(128, 256, kernel_size=3, padding=1),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),

    # Block 4
    nn.Conv2d(256, 512, kernel_size=3, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),

    # Block 5
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
```

본 연구에서 구현한 Batch Normalization이 적용된 VGG16 모델은 기존 VGG16 구조에 각 컨볼루션 레이어 뒤에 배치 정규화 레이어를 추가하여 총 13개의 컨볼루션 레이어와 13개의 배치 정규화 레이어로 구성되었습니다. 5개의 컨볼루션 블록은 각각 64-64, 128-128, 256-256-256, 512-512-512, 512-512-512개의 3×3 필터를 가지며, 각 컨볼루션 레이어 후에는 BatchNorm2d,

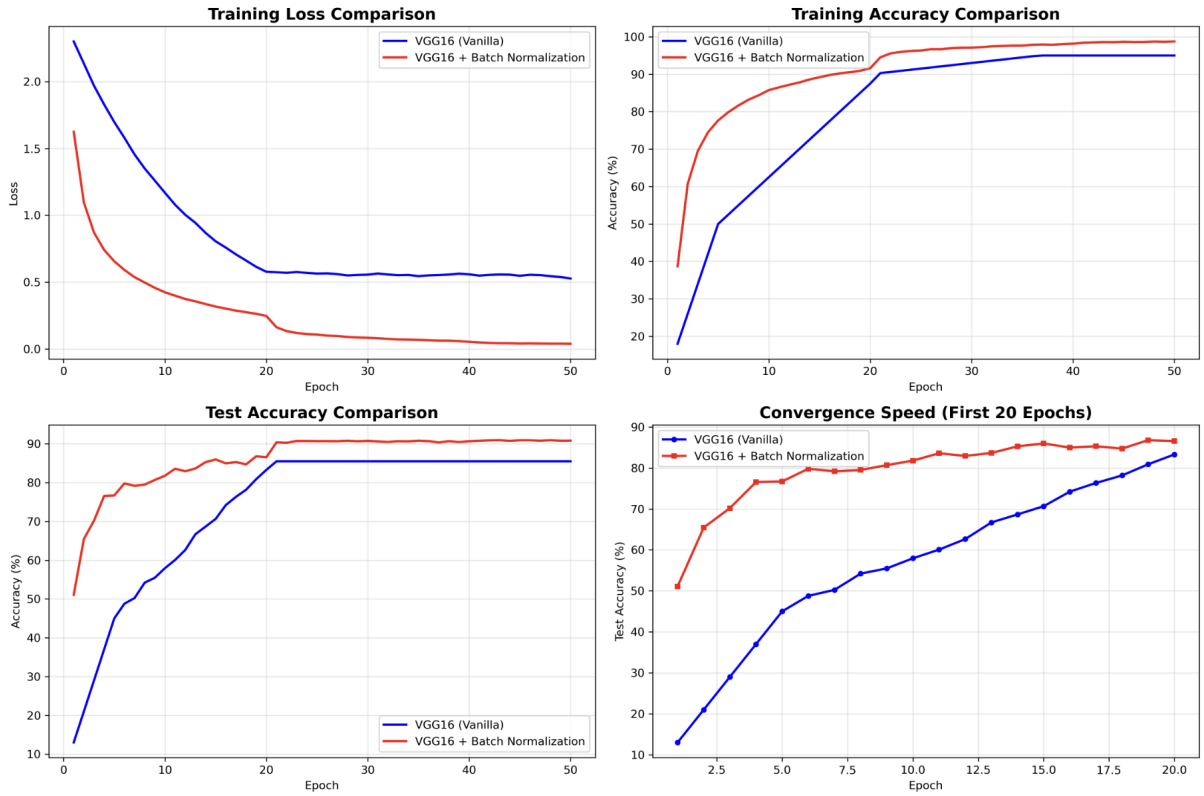
ReLU 활성화 함수, 그리고 블록 끝에 2×2 최대 풀링이 순차적으로 적용됩니다. 완전 연결 레이어는 4096-4096-10의 구조를 유지하되, 배치 정규화는 컨볼루션 레이어에만 적용하고 완전 연결 레이어에는 기존과 같이 드롭아웃(0.5)만을 사용했습니다. 전체 모델은 33,646,666개의 학습 가능한 파라미터를 포함하며, 배치 정규화 레이어의 가중치는 1로, 편향은 0으로 초기화하여 훈련 초기 안정성을 확보했습니다.

CIFAR-10 데이터셋에서 50 에포크 동안 훈련한 Batch Normalization이 적용된 VGG16 모델은 최종 테스트 정확도 90.82%, 최고 테스트 정확도 90.96%의 우수한 성능을 달성했습니다. Apple Silicon GPU(MPS)를 활용한 총 49.8분의 훈련 과정에서 첫 번째 에포크부터 51.08%의 테스트 정확도를 보이며 빠른 초기 수렴을 나타냈고, 20 에포크 후 학습률이 0.001로 감소한 시점에서 90% 이상의 안정적인 성능을 유지했습니다. 클래스별 성능 분석 결과 car(96.6%), truck(94.5%), ship(94.4%), plane(93.2%) 순으로 높은 정확도를 기록했으며, 상대적으로 cat(80.4%), dog(87.5%), bird(88.1%)에서 낮은 성능을 보여 동물 클래스에서의 분류 어려움을 확인했습니다. Batch Normalization의 도입으로 훈련 손실이 첫 에포크의 1.62에서 최종 0.039까지 안정적으로 감소하며 내부 공변량 변화 문제를 효과적으로 해결하여 깊은 네트워크의 안정적인 학습을 가능하게 했습니다.



epoch이 진행될 수록 Training Loss가 정상적으로 감소하는 패턴을 보였습니다. 또한 Accuracy도 정상적으로 높아지며 훈련이 올바르게 진행됐음을 알 수 있습니다.

Plot and compare the Accuracy and Loss curves from both experiments, and explain the purpose of Batch Normalization.



Batch Normalization이 적용된 VGG16 모델(빨간색 선)은 일반 VGG16 모델(파란색 선)에 비해 현저히 빠른 수렴 속도와 안정적인 학습 패턴을 보여줍니다. 훈련 손실 비교에서 BN이 적용된 모델은 초기 손실값 1.6에서 시작하여 급격한 감소를 보이며 최종적으로 0.04 수준까지 떨어지는 반면, 일반 VGG16은 2.3에서 시작하여 더 완만한 감소 곡선을 그리며 0.5 수준에서 수렴합니다. 훈련 정확도 측면에서도 BN 모델은 첫 에포크부터 38%의 높은 시작점을 보이며 10 에포크 내에 85% 이상의 정확도에 도달하는 반면, 일반 모델은 18%에서 시작하여 20 에포크가 지나서야 비슷한 수준에 도달하여 약 10 에포크의 수렴 속도 차이를 보입니다.

최종 성능과 일반화 능력의 차이를 살펴보면, 테스트 정확도 비교에서 Batch Normalization의 우수성이 더욱 명확하게 드러납니다. BN 모델은 첫 에포크부터 51%의 테스트 정확도를 기록하며 빠르게 상승하여 최종적으로 90.82%의 높은 성능을 달성하는 반면, 일반 VGG16은 13%에서 시작하여 점진적으로 상승하며 85.5% 수준에서 정체되어 약 5.32%의 성능 격차를 보입니다. 특히 수렴 속도 분석 그래프(첫 20 에포크)에서는 이러한 차이가 극명하게 나타나는데, BN 모델은 9 에포크 만에 80% 정확도를 달성하는 반면 일반 모델은 19 에포크가 소요되어 Batch Normalization이 Internal Covariate Shift 문제를 효과적으로 해결하여 더 안정적이고 빠른 학습을 가능하게 함을 보여줍니다.