기계학습개론 프로젝트 보고서

2315028 김성현

---

1. 서론 및 배경

CIFAR-10 데이터셋은 10 개의 서로 다른 클래스에 속하는 60,000 개의 32x32 컬러 이미지를 포함하고 있습니다. 이 프로젝트의 목표는 이러한 이미지를 정확하게 분류할 수 있는 강력한 합성곱 신경망(CNN) 모델을 구축하는 것입니다. 저희 팀은 다양한 실험을 통해 최적의 모델 구조와 하이퍼파라미터를 찾아내었고, 이를 기반으로 E_Net 이라는 모델을 구현하였습니다.

2. 개념 설명

1) 데이터 전처리

기존의 데이터 전처리 되었던 이미지크기 조정, pytorch 텐서로 변환, 정규화하는 부분은 그대로 하되 다른 데이터 증강방식을 더 추가해서 데이터 전처리를 해주었습니다.

transforms.RandomCrop(32, padding=4) : 이미지의 각 변에 4 픽셀의 패딩을 추가한 후 임의의 32x32 크기의 패치를 잘라내도록 하였습니다. 패딩을 추가한 후 크롭하는 이유는 모델이 이미지의 다양한 부분적 특징을 학습할 수 있도록 하였습니다.

transforms.RandomHorizontalFlip(p=0.5) : 이미지를 50% 확률로 좌우 반전하도록 하였습니다. 좌우 반전은 이미지의 의미를 크게 변경하지 않으면서도 데이터의 다양성을 크게 증가시킬 수 있는 간단하면서 효과적인 방법입니다. 이 확률 값인 0.5 는 학습 데이터의 절반을 반전시켜 모델이 다양한 방향에서 학습할 수 있도록 하였습니다.

transforms.RandomVerticalFlip(p=0.5) : 이미지를 50% 확률로 상하 반전하도록 하였습니다. 상하 반전도 좌우 반전과 마찬가지로 데이터 다양성을 증가시키며, 모델이 보다 일반적인 특징을 학습할 수 있도록 합니다. 0.5 라는 확률 값은 동일하게 사용되며, 이는 데이터의 절반을 반전시켜 충분한 변화를 주도록 하였습니다.

transforms.RandomRotation(degrees=15) : 이미지를 -15 도에서 15 도 사이의 임의의 각도로 회전시키도록 하였습니다. 회전 각도를 15 도로 설정한 이유는 이미지가 너무 많이 회전되면 원래의 의미를 상실할 수 있기 때문에, 적당한 범위 내에서 회전시켜 모델이 다양한 각도의 이미지를 학습할 수 있도록 하였습니다.

transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2) : 이미지의 밝기, 대비, 채도, 색조를 변경하도록 하였습니다. 각 매개변수 값을 0.2 로 설정한 이유는 이미지의 색상 변화를 너무 극단적이지 않게 하여 자연스러운 변화를 주려고 하였습니다. 모델이 다양한 조명 조건과 색상 변화를 학습할 수 있도록 하였습니다.

2) 네트워크

저희 팀은 E_Net 이라는 CNN 모델을 구현하였습니다. E_Net 모델은 두 가지 주요 구성 요소인 BasicBlock 클래스와 E_Net 클래스로 구성하였습니다.

BasicBlock 클래스

BasicBlock 클래스는 두 개의 합성곱 층과 배치 정규화, 드롭아웃을 포함하는 기본 블록입니다. 이는 모델의 기본 구성 요소로 사용하였습니다. 먼저, __init__ 메서드를 통해 블록의 기본 구조를 정의하였습니다. 첫 번째 합성곱 층(self.conv1)은 입력 채널에서 출력 채널로 변환하도록 하였습니다. 여기서 커널 크기는 3x3 으로 설정했으며, 이는 이미지의 세부 정보를 추출하는 데 매우 적합한 표준 크기이기에 값을 설정하였습니다. 스트라이드는 기본값으로 1 을 사용하고, 패딩은 1 로 설정하여 입력 이미지와 출력 이미지의 크기를 동일하게 유지하도록하였습니다. 두 번째 합성곱 층(self.conv2)도 3x3 커널 크기, 스트라이드 1, 패딩 1 을 사용하여 출력 채널을

유지하도록 하였습니다. 각 합성곱 층 뒤에는 배치 정규화 층을 넣어주었습니다. 배치 정규화(self.bn1, self.bn2)는 학습을 안정화하고 속도를 높이기에 사용하였습니다. 또한, 드롭아웃 층(self.dropout)은 과적합을 방지하기 위해 사용하였습니다. 40% 확률로 값을 설정하여 뉴런을 무작위로 드롭하도록 하였습니다.

BasicBlock 의 forward 메서드는 입력 데이터를 받아서 첫 번째 합성곱 층을 통과시킨 후 배치 정규화와 ReLU 활성화 함수를 적용하도록 하였습니다. 그 다음, 드롭아웃을 적용하여 과적합을 방지하도록 하였습니다. 마지막으로 두 번째 합성곱 층과 배치 정규화를 수행하고 다시 ReLU 활성화를 적용하여 출력을 반환하도록 하였습니다.

E_Net 클래스

E_Net 클래스는 여러 BasicBlock 을 사용하여 이미지 데이터를 처리하고 최종적으로 10 개의 클래스에 대한 예측을 수행하도록 하였습니다. 첫 번째 합성곱 층(self.conv1)은 입력 이미지 (RGB 채널, 3 채널)에서 4 채널로 변환하도록 하였습니다. 3 채널(RGB)을 4 채널로 확장함으로써, 모델은 더 다양한 필터를 사용하여 다양한 기본 특징을 추출할 수 있도록 하였습니다. 3x3 커널 크기와 1 픽셀의 패딩을 사용하여 출력 이미지의 크기를 입력 이미지와 동일하게 유지하도록 하였습니다. 배치 정규화 층(self.bn1)은 첫 번째 합성곱 층의 출력을 정규화하도록 넣어주었습니다.

그 후, 네 개의 BasicBlock 레이어가 순차적으로 배치되도록 하였습니다. 각 레이어는 점진적으로 채널 수를 증가시키고 이미지 크기를 줄입니다. self.layer1 은 4 채널 입력에서 4 채널 출력을 생성하며, self.layer2 는 4 채널 입력에서 16 채널 출력을 생성합니다. 이와 같은 방식으로 self.layer3 과 self.layer4 는 각각 16 채널 입력에서 32 채널 출력, 32 채널 입력에서 64 채널 출력을 생성하도록 하였습니다. 각 레이어의 스트라이드는 2 로 설정되어 이미지 크기를 절반으로 줄이도록 하였습니다. 이는 점진적으로 특성 맵의 크기를 줄이고 채널 수를 늘려서 특징을 더 추상화하는데 도움을 줍니다.

마지막으로, 평균 풀링(F.avg_pool2d)을 통해 공간 차원을 줄이도록 하였습니다. 4x4 커널을 사용하여 최종 특성 맵의 크기를 (1, 1)로 줄이도록 하였습니다. 이는 모든 공간 정보를 하나의 값으로 압축합니다. 그 후, 출력 텐서를 평탄화하여 Linear Layer 에 입력할 수 있도록 변환하였습니다. 완전 연결 층(self.linear)은 최종 출력을 10 개의 클래스로 변환하도록 하였습니다. CIFAR-10 데이터셋은 10 개의 클래스로 구성되어 있으므로, 출력 노드 수를 10 으로 설정하였습니다.

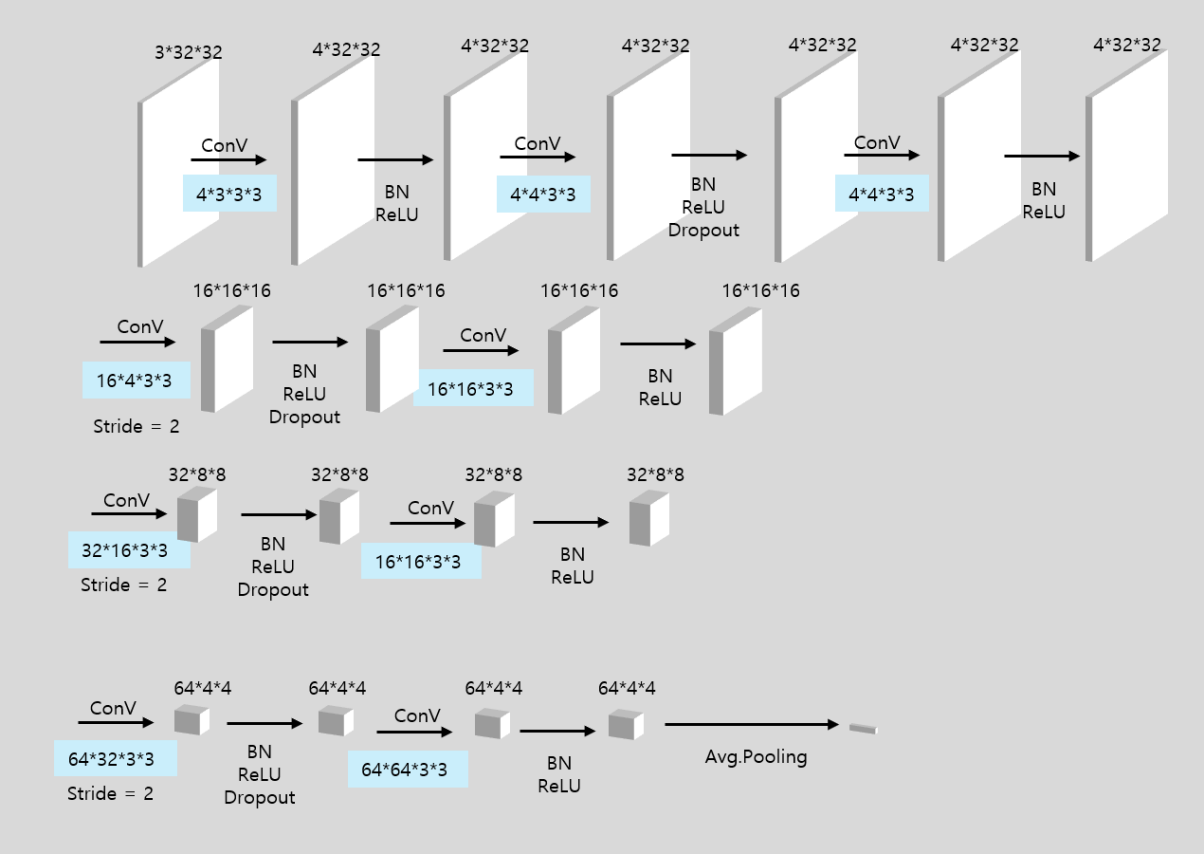

그림_1) 네트워크 구조도

3) 손실함수, 옵티마이저, 학습률 스케줄러

Cross Entropy Loss

손실 함수로 CrossEntropyLoss 를 사용하였습니다. 이는 다중 클래스 분류 문제에서 가장 많이 사용되는 손실 함수로, 모델의 출력(예측값)과 실제 라벨 간의 차이를 계산합니다. CIFAR-10 데이터셋은 10 개의 클래스가 있는 다중 클래스 분류 문제이므로, 기존 코드에 있었던 그대로 본 함수를 사용하게되었습니다.

Adam

옵티마이저로 Adam 을 선택하였습니다. Adam 옵티마이저는 모멘텀과 RMSProp 의 장점을 결합한 방법으로, 학습 속도가 빠르고, 비교적 적은 튜닝으로도 좋은 성능을 발휘하는 특징이 있습니다. 학습률 lr 을 0.001 로 설정한 이유는 Adam 옵티마이저가 기본적으로 학습이 안정적으로 진행되도록 하기에 이 값을 권장하고 있어서기도 하고 실험도 진행해본 결과 본 값으로 결정하게되었습니다.

Step LR

학습률 스케줄러로 StepLR 을 사용하였습니다. StepLR 스케줄러는 일정한 에폭마다 학습률을 감소시킵니다. 여기서 step_size=10 은 매 10 에폭마다 학습률을 감소시키도록 설정하였습니다. gamma=0.1 학습률 감소 비율을 나타냅니다. 즉, 10 에폭마다 학습률을 현재 값의 10%로 감소시키도록 하였습니다. 이는 초기 학습 단계에서는 큰 학습률로 빠르게 수렴하도록 하고, 이후에는 학습률을 점차 낮춰서 더 세밀하게 최적화되도록 유도하는 전략을 사용하였습니다.

3. 방법

Epoch

Epoch : 20 20 10	Epoch : 50
92%	86%

저희 팀은 에폭을 한번에 돌리는 것보다 나누어서 돌리는 것이 더 좋은 성능이 나왔기에 에폭을 20-20-10 으로 나누어서 모델의 성능을 측정하는 방법을 택하였습니다. 같은 에폭수임에도 다른 결과가 나오는 이유를 찾아보게되었습니다. 에폭을 나누어서 학습하면 각각의 학습률에 해당하는 에폭에서 모델 파라미터가 초기화되기 때문에, 이전 학습 과정의 영향을 받지 않고 새로운 학습을 시작할 수 있습니다. 이는 각 학습률에 맞게 모델이 적절하게 초기화되고, 학습 과정이 더 안정적으로 이루어지며, 과적합을 방지할 수 있는 장점을 제공합니다. 따라서 초기화된 모델을 사용하여 각 학습률에 해당하는 에폭을 독립적으로 진행함으로써 더 효율적인 학습이 가능하게 됩니다.

4. 실험

팀원들은 각자 다양한 모델을 구성하고 실험하였으며, 가장 성능이 좋았던 모델을 선택하여 E_Net 을 최종 모델로 결정하였습니다. 이후 여러 하이퍼파라미터와 네트워크 구조를 조정하며 성능을 최적화하였습니다. 각자 실험 결과를 공유하며 성능을 개선할 수 있는 방법을 모색하였습니다.

단 본 실험 결과들은 모델 확정전에 실험한 것들이고, 다른 팀원분들의 결과를 공유한 것도 있기에 상대적인 것만 비교가능하고, 절대적인 수치적으로 비교하면 안됩니다.

- **전처리** : 데이터 전처리 및 증강의 여부를 실험해 최적의 조합을 찾아내었습니다.

Flip, color jitter, random crop	Filp, color jitter	전처리 x
49%	47%	40%

- **배치 사이즈**: 16, 32, 64 를 실험하여 최적의 값을 찾아내었습니다.

Batchsize : 16	Batchsize : 32	Batchsize : 64
67%	87%	204%

- **드롭아웃 확률**: 0.3, 0.4, 0.5 를 실험하여 최적의 값을 찾았습니다.

Drop out : 0.3	Drop out : 0.4	Drop out : 0.5
105%	96%	72%

- **블록 크기**: 다양한 블록 크기를 실험하여 최적의 조합을 찾았습니다.

4-8-12-16	4-8-12-32	4-16-16-32	4-16-32-64	16-32-64-128
75%	80%	80%	96%	132%

- **최적화 알고리즘**: Adam, SGD with momentum, SGD, Adagrad 등을 실험하였으며, 학습률도 실험하여 설정하였습니다.

adam	Sgd with momentum	sgd	adagrad
92%	82%	81%	54%

Learning rate : 0.0005	Learning rate : 0.001
67%, Epoch : 30	71%, Epoch : 20
69%, Epoch : 40	75%, epoch : 40
70%, Epoch : 50	75%, Epoch : 50

- **스케줄러**: StepLR 과 Cosine Annealing 을 사용하여 학습률 조정을 실험하였습니다. 특히 StepLR 의 스텝 크기를 조정하여 최적의 성능을 찾았습니다.

Step LR	Cosine annealing
92%	75%

Step size : 2	Step size : 3	Step size : 5	Step size : 10
54%	59%	66%	73%

5. 결과 및 결론

실험 결과를 통해 모델의 성능을 평가하고 최적화하는 데 기여한 다양한 요인들을 확인했습니다. 특히, 에폭을 나누어 학습하는 방법이 모델의 성능 향상에 큰 도움이 되었습니다. 앞선 실험 결과를 바탕으로 최적의 조합을 구성하여 결국 최종 결과로는 92%의 정확도를 달성하게되었습니다.

```
  return Variable._execution_engine.run_backwar
[Epoch - 1] Loss: 1.956
[Epoch - 2] Loss: 1.782
[Epoch - 3] Loss: 1.718
[Epoch - 4] Loss: 1.673
[Epoch - 5] Loss: 1.642
[Epoch - 6] Loss: 1.615
[Epoch - 7] Loss: 1.600
[Epoch - 8] Loss: 1.584
[Epoch - 9] Loss: 1.571
[Epoch - 10] Loss: 1.559
[Epoch - 11] Loss: 1.526
[Epoch - 12] Loss: 1.514
[Epoch - 13] Loss: 1.515
[Epoch - 14] Loss: 1.507
[Epoch - 15] Loss: 1.507
[Epoch - 16] Loss: 1.497
[Epoch - 17] Loss: 1.498
[Epoch - 18] Loss: 1.501
[Epoch - 19] Loss: 1.498
[Epoch - 20] Loss: 1.495
Finished Training for current learning rate
[Epoch - 1] Loss: 1.496
[Epoch - 2] Loss: 1.491
[Epoch - 3] Loss: 1.490
[Epoch - 4] Loss: 1.491
...
[Epoch - 9] Loss: 1.486
[Epoch - 10] Loss: 1.492
Finished Training for current learning rate
Finished Training
```

[9]

```python
# Test the trained model with overall test dataset

correct = 0
total = 0

net.eval()
for data in testloader:
    # Load the data
    inputs_test, labels_test = data
    inputs_test = inputs_test.to(device)
    labels_test = labels_test.to(device)

    # Estimate the output using the trained network
    outputs_test = net(inputs_test)
    _, predicted = torch.max(outputs_test.data, 1)

    # Calculate the accuracy
    total += labels.size(0)
    correct += (predicted == labels_test).sum()

# Final accuracy
print('Accuracy of the network on the 10,000 test images: %d %%' % (100

## [SimpleNet / Training 5 epochs] Accuracy of the network on the 10,000
## [VGG11 / Training 5 epochs] Accuracy of the network on the 10,000 tes
```

Pytho

```
Accuracy of the network on the 10,000 test images: 92 %
```