**REPORT**



|  |  |
| --- | --- |
| **과목명** | **딥러닝** |
| **학과** | **소프트웨어 전공** |
| **학번** | **202020933** |
| **이름** | **송주훈** |
| **제출일자** | **2024-12-14** |

**목표**

CIFAR-10 데이터셋을 기반으로 PyTorch를 활용해 Convolutional Neural Network(CNN)를 구현하고, 데이터 전처리, 증강, 드롭아웃 등 다양한 기법을 사용하여 모델을 학습 및 평가한다.

1. **모델 아키텍처**

#### ****CNN 구조****

구현한 CNN은 다음과 같은 구조를 가진다:

1. **컨볼루션 레이어**:

1차 컨볼루션 레이어: 입력 이미지에서 3채널(RGB) 데이터를 처리하며, 필터 개수는 32개, 필터 크기는 3×33 \times 33×3, 패딩은 1로 설정.

2차 컨볼루션 레이어: 첫 번째 레이어의 출력(32채널)을 입력받아 필터 개수를 64로 증가.

3차 컨볼루션 레이어: 두 번째 레이어의 출력(64채널)을 입력받아 필터 개수를 128로 증가.

각 컨볼루션 레이어는 활성화 함수로 ReLU를 사용하여 비선형성을 추가.

1. **맥스 풀링 레이어**:

각 컨볼루션 레이어 뒤에 2×2 커널과 스트라이드 2로 구성된 맥스 풀링을 적용하여 공간 차원을 절반으로 줄임.

1. **완전 연결(Fully Connected) 레이어**:

128채널의 4×4 특성 맵을 평탄화하여 입력 크기를 128×4×4=2048로 변환.

첫 번째 Fully Connected 레이어는 256개의 뉴런을 사용.

두 번째 Fully Connected 레이어는 최종 출력 클래스(10개)를 분류.

1. **드롭아웃 (Dropout)**:

첫 번째 Fully Connected 레이어 뒤에 드롭아웃(비율 0.5)을 추가하여 과적합을 방지.

드롭아웃은 학습 과정에서 일부 뉴런을 임의로 비활성화함으로써 과적합을 방지하고 모델의 일반화 성능을 향상시킴.

1. **데이터 전처리 및 증강**

**데이터셋 로딩**

PyTorch의 torchvision.datasets를 사용하여 CIFAR-10 데이터셋을 로드:

train\_dataset: 학습 데이터.

test\_dataset: 테스트 데이터.

**데이터 증강 기법**

다양한 데이터 증강 기법을 사용하여 모델의 일반화 성능을 향상

* **랜덤 크롭(Random Crop)**: 이미지를 임의로 자르고 4픽셀 패딩을 추가.
* **좌우 반전(Horizontal Flip)**: 데이터를 좌우로 뒤집어 데이터 다양성 증가.
* **정규화(Normalization)**: 평균 0, 표준편차 1로 정규화.

**증강 강화 실험에서 회전과 색상 조정을 추가했으나, 정확도가 감소하여 최종적으로 제외했다**

#### ****학습 과정****

#### ****손실 함수 및 최적화 기법****

* **손실 함수**: 크로스 엔트로피 손실을 사용. 다중 클래스 분류 문제에서 널리 사용되며, 확률 기반 출력의 손실을 계산.
* **최적화 기법**: Adam 옵티마이저를 사용. 학습률은 0.001로 설정. Adam은 SGD의 단점을 보완하며, 학습 속도와 안정성을 동시에 제공.

#### ****학습 설정****

* **에폭(Epoch)**: 초기 10에서 50까지 실험. 최적 에폭은 42로 설정.
* **배치 크기**: 64로 설정. 128로 증가시킨 경우 성능 저하 발생하여 최종 64로 설정.

#### ****학습 진행 과정****

각 에폭마다 학습 및 검증 손실을 기록:

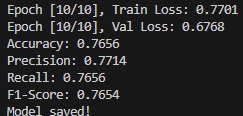
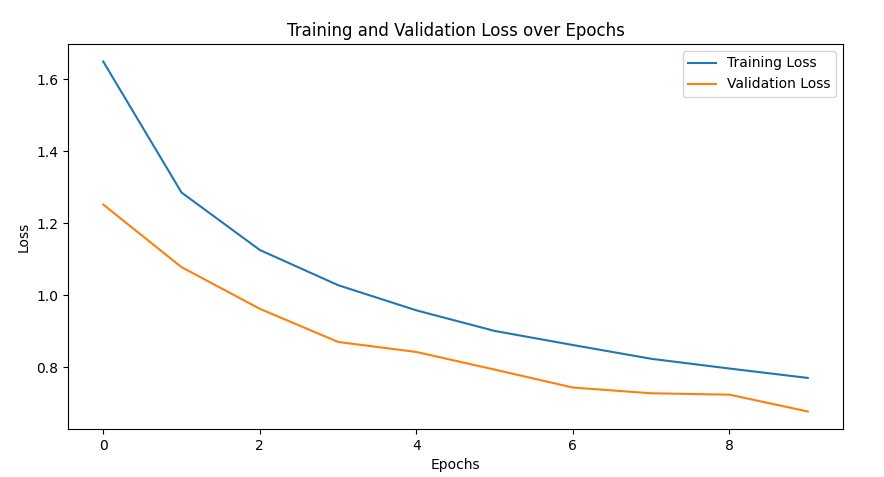
* **Training Loss**와 **Validation Loss** 그래프를 통해 모델 수렴 여부 확인.

학습 중 주요 이슈:

1. **배치 사이즈 증가**: 학습 안정성이 증가했지만 성능은 감소.
2. **드롭아웃 비율 변경**: 0.3으로 감소시킨 경우 정확도가 감소

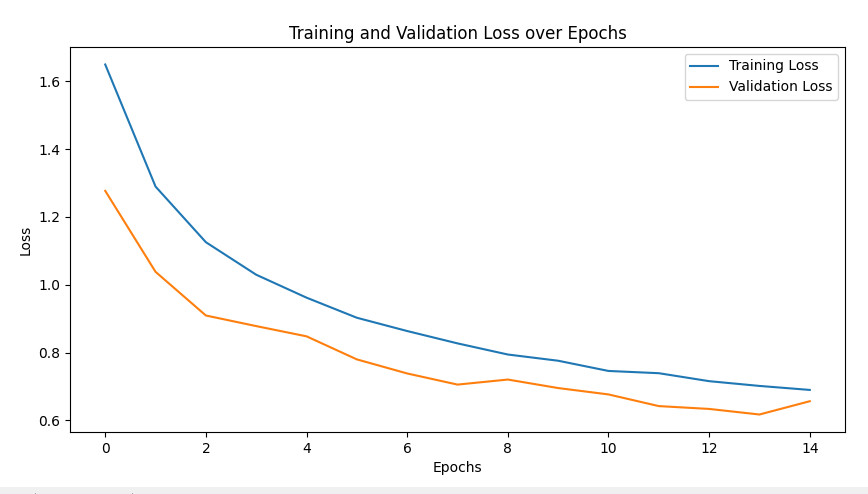
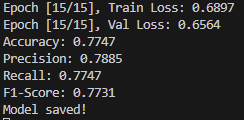
**1. 에폭 크기 변경**

**에폭의 크기가 10인 경우**



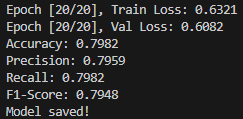
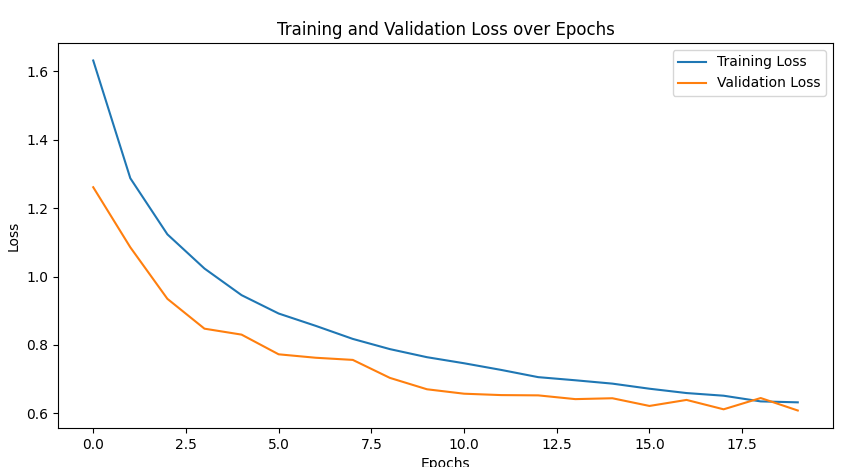
에폭의 크기가 10인 경우 위와 같이 0.76의 정확도를 확인할 수 있었고 추가적으로 정확도와 손실 값이 급격히 변화하는 모습을 확인할 수 있었다.

**에폭의 크기가 15인 경우**

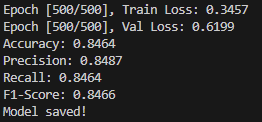
에폭의 크기가 15인 경우도 마찬가지로 모델이 계속해서 개선되고 있었으며, 정확도는 소폭 상승하고 loss 값도 낮아진 모습을 확인할 수 있었다.

**에폭의 크기가 20인 경우**



에폭의 크기가 20인 경우엔 정확도가 이전보다 큰 폭으로 상승한 것을 확인할 수 있었고 확실한 내용을 확인하기 위해 다음은 에폭의 크기를 더 크게 진행하도록 했다.

**에폭의 크기가 500인 경우**

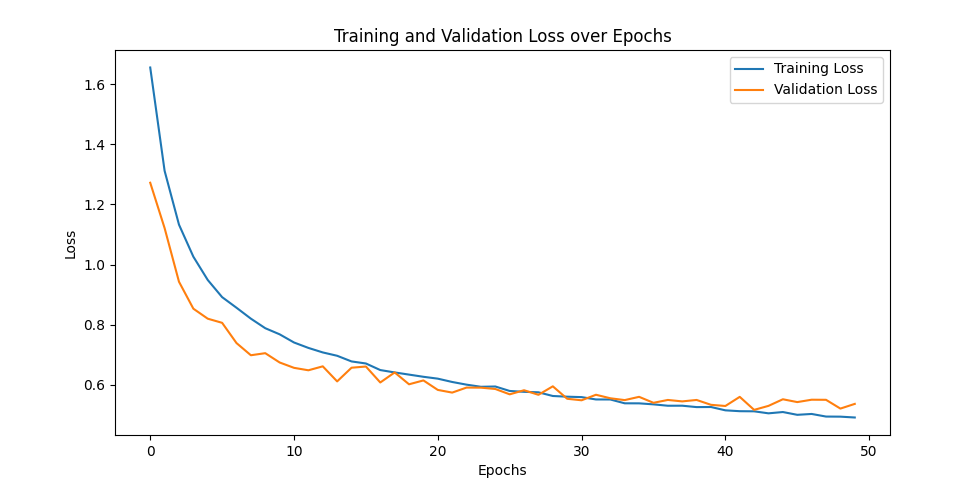
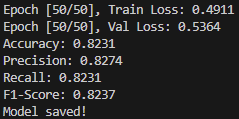


에폭의 크기가 500인 경우 아래 에폭의 크기가 50인 경우와 크게 차이가 발생하기 않았고 손실이 500 이후로는 크게 변하지 않는 것을 확인해 에폭을 50으로 지정한 후 다시 진행하도록 했다.

**에폭의 크기가 50인 경우**

에폭의 크기를 50으로 수정한 결과 약 40 초반에서 정확도와 손실 값이 더 이상 유의미하게 변화하지 않음을 확인했다.

이를 통해 어느정도 모델이 수렴한 것으로 해석해 앞으로의 과정에선 에폭의 크기를 42로 고정한 후 다른 방법으로 정확도를 높이고자 했다.

**2. 이미지 회전 및 색상 변경 진행해보기**

# 데이터 증강 강화

transform\_train = transforms.Compose([

transforms.RandomCrop(32, padding=4),

transforms.RandomHorizontalFlip(),

**transforms.RandomRotation(15), # 15도 범위로 이미지를 회전**

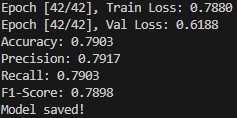
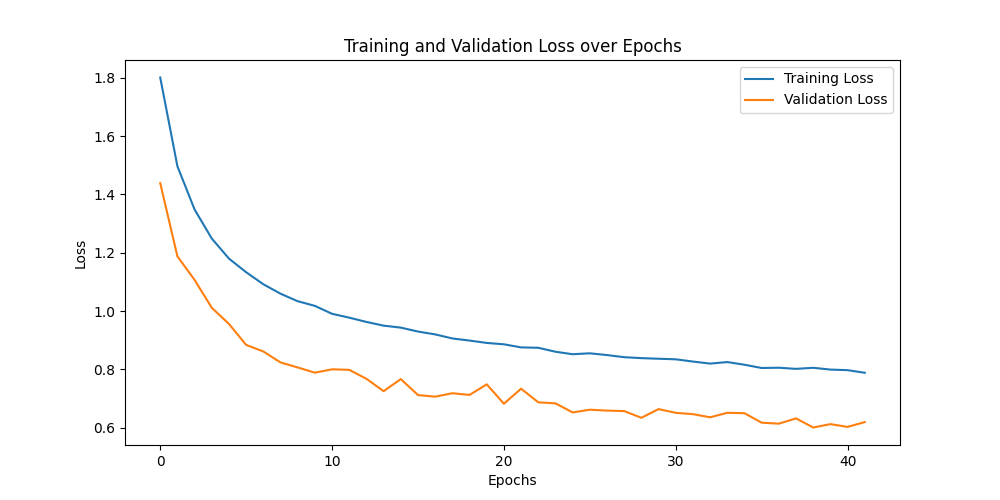
**transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2), # 색상 조정**

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

])

다음 방법으로는 위와 같이 데이터 증강 강화부분에서 이미지 회전과 색상을 조정하는 코드를 추가해 다시 진행해보았다.



진행 후 위와 같은 결과를 확인할 수 있었고 예상 외로 train loss가 줄지 않고 정확도 또한 크게 증가하지 않았음을 확인할 수 있었다.

따라서 이후의 과정에선 에폭의 크기는 42로 고정하되 이미지 회전과 색상 변경 과정은 제외한 후 다시 진행하도록 했다.

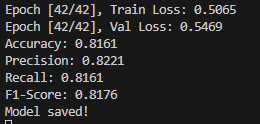
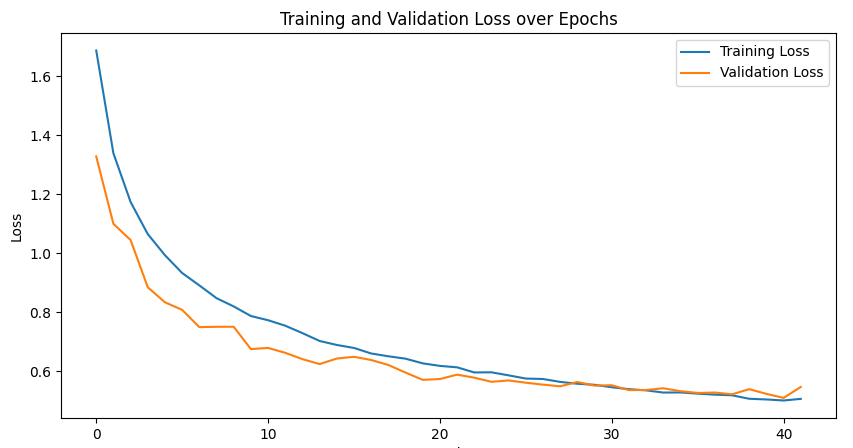
**3. 배치 사이즈 변경**

train\_loader = DataLoader(train\_dataset, batch\_size=128, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=128, shuffle=False)

다음 방법으로는 배치 사이즈의 크기를 64에서 128로 변경하여 진행하도록 했다.

배치 크기를 증가시키게 되면 학습이 더 안정적으로 진행되고 빠르게 학습이 가능하다는 장점이 있어 진행해 보았지만 다음과 같은 결과를 확인했다.



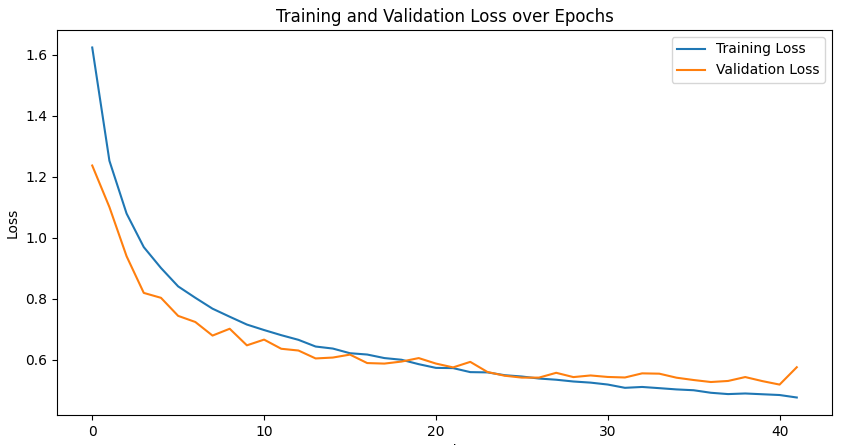
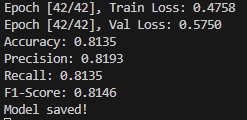
배치 사이즈를 증가시킨 것은 오히려 모델이 데이터에 과도하게 최적화하도록 만들어 성능이 떨어진 것을 확인했고 이후 진행할 땐 배치 사이즈 또한 증가시키지 않고 진행하도록 했다.

**4. 드롭아웃 비율 조정**

# 드롭아웃 비율을 변경

self.dropout = nn.Dropout(0.3) # 드롭아웃 비율을 0.3으로 변경

마지막으로 진행한 내용은 드롭아웃의 비율을 0.5에서 0.3으로 변경해 학습시킨 것으로 결과는 다음과 같다.

드롭 아웃을 0.3으로 수정한 결과 기존 0.5보다 0.01 정확도가 낮아진 것을 확인했고 이러한 결과로 처음에 작성한 코드에서 단순 에폭을 50정도로 증가시킨 결과가 가장 좋았다는 것을 확인할 수 있었다.

#### ****모델 평가****

**테스트 결과**

* **정확도(Accuracy)**: 0.8231
* **정밀도(Precision)**: 0.8274
* **재현율(Recall)**: 0.8231
* **F1-점수(F1-Score)**: 0.8237

#### ****결과 분석****

* 모델의 성능 향상을 위해 이미지 회전, 색상, 배치 크기 조절, 드롭아웃 비율 조절 등 다양한 시도를 진행했지만 에폭의 크기를 증가하는 방법 이외의 다른 파라미터 수정 혹은 기능 추가는 크게 영향을 미치지 않거나 정확성을 떨어뜨리는 것을 확인.

1. **모델 저장 및 재사용**

* 학습된 모델은 torch.save()를 사용해 저장 진행:

**torch.save(model.state\_dict(), "cnn\_cifar10.pth")**

* 재사용 시 state\_dict를 로드하여 모델 초기화 진행:

**model.load\_state\_dict(torch.load("cnn\_cifar10.pth"))**

1. **결론**

* 구현된 CNN은 기본적인 데이터 증강과 드롭아웃 기법을 활용해 CIFAR-10에서 82%의 정확도를 달성.
* 데이터 증강 강화 및 하이퍼파라미터 튜닝의 효과가 제한적이었으며, 정확성을 올리기 위해서는 이미지 회전과 색상과는 다른 기능 추가의 필요성을 느낌.

#### ****향후 개선 방향****

1. **사전 학습된 모델 활용**
2. **하이퍼파라미터 최적화**
3. **고급 데이터 증강 기법** 활용