보고서: 손글씨 알파벳 인식을 위한 딥러닝 모델 구현 및 성능 개선 전략

학습 관련 Loss 및 Optimizers의 파라미터와 선정 이유

Loss Function

- **선택한 Loss Function**: `nn.CrossEntropyLoss`
- **선정 이유**: 이 Loss Function은 LogSoftmax와 NLLLoss를 결합한 것으로, 다중 클래스 분류 문제에 적합합니다. 각 클래스에 대한 모델의 확률 추정치를 계산하여, 실제 레이블과의 차이를 효율적으로 측정합니다. 또한, 이 함수는 수치적 안정성이 높고, 모델의 로짓(logit, 모델 출력의 원시 점수)을 직접 처리할 수 있어 사용하기 쉽습니다.

Optimizers

- **선택한 Optimizers**: `Adam`과 `SGD` (momentum 포함)
- **선정 이유**: - **Adam**: 적응형 학습률을 사용하여, 다양한 데이터와 네트워크 구조에 대해 좋은 성능을 보이는 것으로 알려져 있습니다. 특히, 빠른 수렴 속도와 효율적인 계산이 장점입니다. - **SGD with Momentum**: 기본적인 SGD에 비해 모멘텀(momentum)이 추가되어, 학습 과정에서 방향성을 유지하며 더 빠르고 안정적으로 최적점에 도달합니다. 파벳 인식 딥러닝 모델 실험 결과 및 분석

실험 결과

실험 과정에서 CrossEntropyLoss와 두 가지 옵티마이저 (Adam, SGD)를 사용하여 모델의 성능을 평가하였습니다. 아래는 각 옵티마이저에 대한 에포크별 훈련 및 검증 손실을 요약한 표입니다.

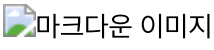
CrossEntropyLoss - Adam Optimizer 결과

에포크	훈련 손실	검증 손실
1	1.593	1.613
2	0.686	0.471
3	0.494	0.438
4	0.311	0.462
5	0.287	0.520
6	0.202	0.423
7	0.245	0.435
8	0.127	0.360
9	0.162	0.322
10	0.158	0.356

CrossEntropyLoss - SGD Optimizer 결과

에포크	훈련 손실	검증 손실
1	0.098	0.188
2	0.050	0.159
3	0.036	0.171
4	0.028	0.158
5	0.040	0.153
6	0.037	0.139
7	0.022	0.144
8	0.016	0.133
9	0.022	0.141
10	0.023	0.138

그래프:



최종 결론

- 실험 결과, 최적의 모델은 **CrossEntropyLoss와 SGD 옵티마이저**를 사용한 조합으로 확인되었습니다. 이 조합에서 모델은 에포크 8에서 가장 낮은 검증 손실(0.133)을 보여주었습니다.

성능 개선을 위해 시도한 방법

1. Pretrained Model (전이 학습) 사용

- **사전 학습된 ResNet-18 모델 적용**: 이미지넷에서 훈련된 가중치를 사용하여 기본 이미지 특성을 빠르게 인식하고 학습 시간을 단축함. 이를 통해 모델이 손글씨 인식에 필요한 특성을 더 효과적으로 학습할 수 있도록 함.

```
model = models.resnet18(pretrained=True)
```

2. 데이터 전처리 (Normalization) 적용

- **입력 데이터 정규화**: 코드에서

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

을 적용하므로서 모든 이미지 데이터를 0과 1 사이의 값으로 정규화하여 모델의 학습 효율성과 안정성을 높임. 이는 입력 데이터의 분포를 일정하게 유지하여 모델이 더 빠르고 안정적으로 학습할 수 있게 도움.

3. Validation Loss를 기준으로 최적의 모델 저장

- **검증 손실 기반 모델 저장**: 학습 중 검증 세트의 손실을 모니터링하고, 가장 낮은 손실을 보인 모델을 저장함. 이는 과적합을 방지하고, 테스트 시 실제 성능이 더 우수한 모델을 확보하는 데 중요. 코드 :

```
if avg_val_loss < best_val_loss:
    best_val_loss = avg_val_loss
    best_model = model.state_dict()
```

4. 미니 배치 사용

- **미니 배치 학습**: 미니 배치를 사용하여 학습 과정의 메모리 효율성을 높이고, 각 배치에서 다양한 데이터 샘플을 통해 모델의 일반화 능력을 개선함.

```
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
```

5. SGD에 관성 (Momentum) 적용

- **SGD 최적화기에 Momentum 추가**: 표준 SGD에 비해 Momentum을 적용하여, 학습 과정에서의 방향성을 유지하고, 최적점에 더 빠르고 안정적으로 도달함. 이는 지역 최적점(local minima)에 빠지는 것을 방지하고 전체적인 학습 속도를 향상시키는데 도움이 됨.

```
optimizers =  optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```