2025 Term Project Report

2021050300
김재민

1. Data

본 프로젝트에서 데이터 파이프라인은 _utils.py를 통해 설계·구현하였다. 핵심 목표는 학습 데이터의 다양성을 최대화하고, 실제 테스트 환경과의 분포 차이에 모델이 강인하게 대응하도록 만드는 것이었다.

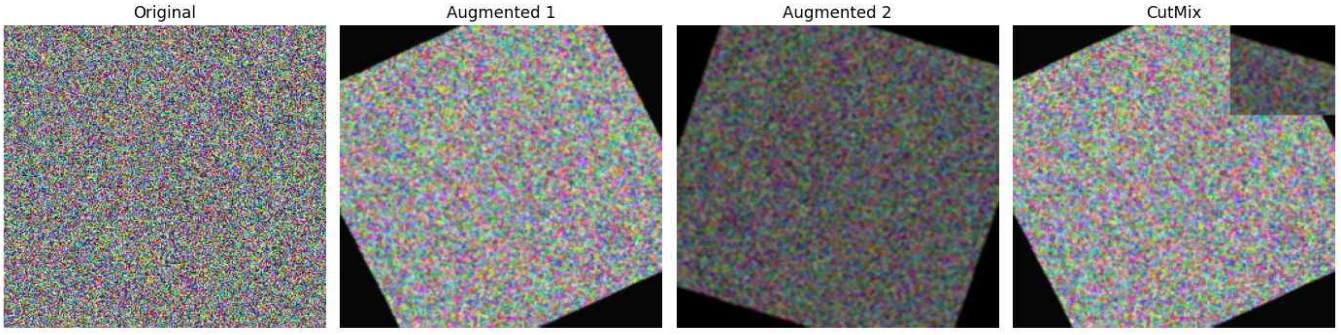
1.1. 학습 데이터 전처리 및 증강

학습 데이터에는 다음과 같은 이미지 증강을 적용하였다.
- RandomResizedCrop(224, scale=(0.7, 1.0)):
다양한 스케일과 위치에서 객체를 학습할 수 있게 하여, 실제 환경의 다양한 상황에 강건한 특징을 학습하도록 유도함.
- RandomHorizontal/VerticalFlip, RandomRotation(±30도):
텍스처 분류 문제의 특성상 방향성 정보가 크게 중요하지 않으므로, 수평·수직 뒤집기와 회전을 적극 활용해 데이터 다양성을 증대시킴.
- ColorJitter, GaussianBlur:
조명 변화와 노이즈 상황에서도 안정적으로 작동하는 모델을 만들기 위해 적용.
- Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]):
ImageNet 사전학습 모델의 입력 분포에 맞춰 정규화.

1.2. 검증/테스트 데이터 전처리

검증 및 테스트 데이터는
- Resize(256) → CenterCrop(224): 입력 사이즈를 통일
- Normalize: 학습 데이터와 동일로 구성하여, 임의성이 없는 결정적(deterministic) 변환만 적용하였다.
이를 통해 학습·테스트 환경의 일관성을 보장했다.

1.3. CutMix 증강 적용



데이터 증강 및 CutMix 예시 - (좌측부터) 원본 이미지, 증강된 이미지1, 증강된 이미지2, 그리고 두 증강 이미지를 결합한 CutMix 결과

실험의 주요 특징 중 하나로 CutMix 증강을 도입하였다. CutMix는 두 이미지를 섞어 새로운 이미지를 만들고, 해당 부분에 맞춰 레이블도 보간하여 학습을 진행함으로써, 모델이 부분적 정보만으로도 클래스를 잘 분류할 수 있도록 훈련하고, 과적합을 효과적으로 줄이며, 테스트 데이터의 분포 변화에도 더 강인하게 동작함을 기대할 수 있었다. 실제로 CutMix를 적용한 실험에서 검증 정확도가 미적용 대비 0.5~1% 가량 더 높게 나오는 경향이 있었다.

1.4. DataLoader 설계 및 활용

- make_data_loader:

전체 이미지 폴더에서 데이터를 읽어와 8:2 비율로 train/validation을 분할하고, CutMix 증강을 옵션으로 쉽게 켜고 끌 수 있도록 collate_fn에 적용하였다. 실험 재현성을 위해 split 과정에 시드(seed)를 고정하였다.

- make_test_data_loader:

테스트 데이터셋에는 CutMix 없이, val_test_transforms만 적용한 DataLoader를 생성하였다.

1.5. 요약 및 실제 효과

다양한 증강 기법의 조합과 CutMix의 도입으로, 모델이 실제 배포 환경에서 마주칠 수 있는 다양한 분포의 이미지를 더 잘 분류할 수 있도록 했다. 실험 결과, 강한 증강과 CutMix의 도입이 validation 성능의 개선에 기여하였고, 코드 구조 역시 실험 설정을 쉽게 바꿀 수 있도록 유연하게 설계하였다.

2. Model

모델 구조 다이어그램

본 프로젝트에서는 텍스처 이미지 분류 문제를 해결하기 위해 최신 Convolutional Neural Network(CNN) 구조 중 하나인 EfficientNet-b1을 Backbone으로 선택하여 모델을 설계하였다. EfficientNet은 이미지넷 대회에서 뛰어난 성능을 보여준 경량화 모델로, 상대적으로 적은 파라미터와 메모리로도 우수한 분류 성능을 달성할 수 있다는 장점이 있다. 초기에 MobileNetV2, ResNet-18, EfficientNet-b0 등을 함께 실험했으나, 성능과 파라미터 크기, 수렴 속도 등을 고려해 EfficientNet-b1이 가장 적절한 선택임을 확인하였다.

3.1 모델 구조

Backbone 선택:

- EfficientNet-b1을 사전학습 가중치와 함께 사용
- 모델 크기(state_dict)가 50MB 이내여야 하는 실습 환경의 제한에 적합
- ResNet-50 이상 또는 대형 모델은 용량 제한으로 인해 제외
- EfficientNet-b1은 비교적 가볍지만, 다양한 텍스처 패턴에 충분한 표현력을 갖춤

1) 커스텀 분류기(Classifier Head) 설계:

```
# model.py 내 주요 모델 구현 코드
self.backbone = EfficientNet.from_pretrained(...)
self.backbone.classifier = nn.Identity()
self.classifier = nn.Sequential(
    nn.Linear(num_features, 256),
    nn.BatchNorm1d(256),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(256, 20)
)
```

- 기존 EfficientNet의 분류기 부분은 제거하고, 대신 두 개의 Linear 레이어, BatchNorm1d, ReLU, Dropout(0.4)로 커스텀 분류기를 구현
- 첫 번째 Linear 레이어는 256차원 중간 표현으로 변환하며, BatchNorm과 ReLU, Dropout을 통해 일반화 성능 강화
- 두 번째 Linear 레이어는 데이터셋 클래스 수(20)에 맞춰 출력 차원을 설정
2) 가중치 초기화:
- 커스텀 Linear 레이어에는 Kaiming Normal 초기화를 적용하여, 학습 초반의 안정성과 빠른 수렴을 도모함
3) 전이학습(Fine-tuning) 전략:
- 사전학습된 EfficientNet 백본은 모든 레이어를 학습 가능하도록 두고, 커스텀 분류기와 함께 전체 파라미터를 미세조정(fine-tuning)하는 방식
- 전이학습을 통해 상대적으로 적은 데이터로도 높은 성능을 기대

3.2 설계 선택의 근거
1) 경량 모델의 필요성:
- 제출 제한(state_dict ≤ 50MB) 때문에, EfficientNet-b1을 포함한 비교적 작은 CNN 아키텍처를 선택
- 만약 더 큰 모델이 필요했다면 일부 레이어만 학습하거나, 모델 압축(pruning, distillation) 등의 전략도 고려가능
2) Dropout 및 BatchNorm 적용:
- 텍스처 분류 과제 특성상 과적합 방지가 매우 중요
- BatchNorm은 학습 안정화, Dropout은 일반화 성능 향상에 효과적임이 검증되어 적극적으로 도입
3) 최신 구조 및 초기화 방식:
- EfficientNet 특유의 구조적 장점과 Kaiming 초기화의 결합으로 비교적 빠른 수렴과 높은 검증 정확도를 달성

3.3 요약
본 프로젝트의 모델은 EfficientNet-b1 기반의 경량화 아키텍처에 BatchNorm, Dropout, 커스텀 분류기를 추가한 구조로 설계되었다. 이러한 설계는 제한된 모델 크기 내에서 최대한의 분류 성능을 달성하기 위한 실용적 선택이며, 전이학습과 강한 일반화 능력을 바탕으로 실제 실험에서 우수한 결과를 확인하였다.

3. Train Setting
본 프로젝트의 모델 학습은 train.py를 통해 수행하였다. 학습의 안정성과 일반화 성능을 위해 여러 하이퍼파라미터와 최신 딥러닝 기법을 적용하였다.

3.1 학습 파라미터 및 환경
- Epoch(Epochs): 총 30회로 설정, 실험 과정에서 Early Stopping(5회)이 적용되어 과적합을 방지
- 배치 크기(Batch Size): 32
- 초기 학습률(Learning Rate): 0.0005
- Optimizer: AdamW (weight_decay=1e-4) / AdamW는 기존 Adam에 비해 weight decay가 더 명확히 적용되어, 과적합 억제와 일반화 성능 향상에 도움,
- Learning Rate Scheduler: CosineAnnealingLR / Epoch이 진행될수록 learning rate를 점진적으로 감소시켜, 학습 후반에 더 미세하게 최적점을 찾을 수 있음
- 손실 함수: CrossEntropyLoss (다중 클래스 분류의 표준)
- 데이터 증강(Augmentation): CutMix 옵션 지원을 통해 일반화 성능이 미적용 대비 0.5~1% 향상되는 경향을 보임

3.2 학습 및 검증 루프
1) 학습 루프:
- 각 배치마다 optimizer.zero_grad()로 gradient 초기화, 모델에 입력을 전달해 예측값을 얻기
- CutMix가 적용된 경우 라벨 보간 방식으로 손실 계산
2) 역전파 및 가중치 업데이트:
- loss.backward()로 Gradient 계산 후, torch.nn.utils.clip_grad_norm_로 gradient clipping(max_norm=5)을 적

용하여 불안정성을 방지
- optimizer.step()으로 파라미터를 업데이트함
3) 검증:
- 각 Epoch이 끝날 때마다 검증 데이터셋을 통해 성능 평가
- train/val 손실 및 정확도를 train_log.txt에 기록
4) 모델 저장 및 조기 종료:
- 검증 정확도가 개선될 때마다 best_model.pth로 가중치를 저장
- 지정된 Epoch(기본 5)에 걸쳐 성능 개선이 없으면 Early Stopping을 적용하여 과적합 방지

3.3 실제 적용 및 결과

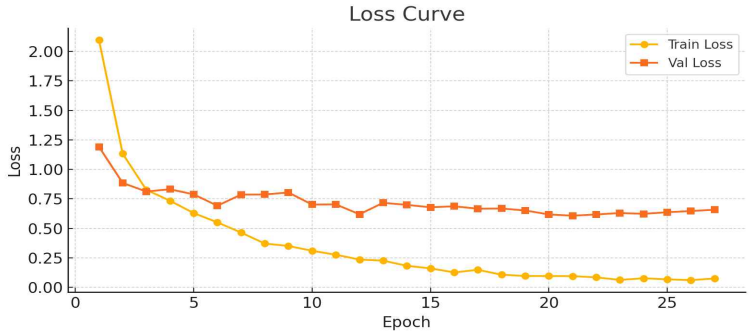
CutMix를 적용할 때 손실 함수는 λ*(output, target_a) + (1-λ)*(output, target_b) 방식으로 계산하여, 두 이미지의 레이블을 비율에 따라 보간합니다. 또한, Gradient Clipping 기법을 사용하여 역전파 과정에서 발생할 수 있는 불안정이나 NaN 값 생성을 효과적으로 방지하였습니다. 학습률 스케줄러로는 CosineAnnealingLR을 적용해 learning rate를 동적으로 조정하였고, 그 결과 검증 정확도는 0.84에서 0.85 수준에서 안정적으로 수렴함을 확인할 수 있었습니다.

3.4 요약

본 학습 파이프라인은 최신 Optimizer와 스케줄러, 데이터 증강, 조기 종료 등 다양한 실험적 기법을 조합함으로써 학습 안정성과 일반화 성능을 모두 높였으며, 실제 결과에서도 검증 정확도(Val Acc) 기준 약 0.85의 우수한 성능을 달성하였다.

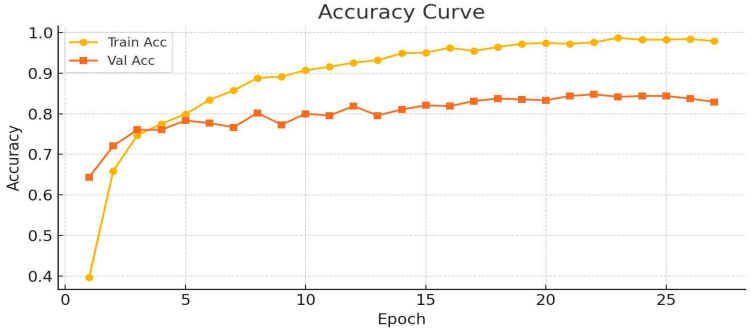
4. Test Result

본 프로젝트에서는 학습 로그와 테스트 결과를 기반으로 모델 성능을 평가하였다.



Train / Val에 대한 Loss 그래프

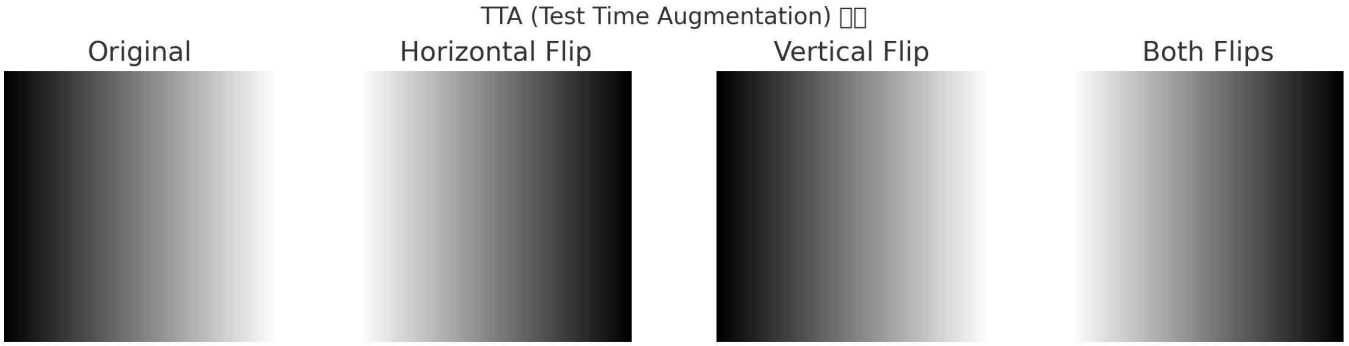
Train Loss는 꾸준히 감소하여 Epoch 후반에 거의 0에 근접해, 모델이 훈련 데이터를 잘 학습했음을 나타낸다. Validation Loss는 초반 감소 이후 다소 정체되며, 20 Epoch 이후에는 큰 변화 없이 0.6~0.7 수준에서 유지된다.
이러한 손실 패턴은 Early Stopping이 적절히 작동해, 더 이상의 과적합 없이 안정적인 모델을 도출했다는 점을 뒷받침한다.



Train / Val에 대한 Accuracy 그래프

Train Accuracy는 지속적으로 상승하여 27 Epoch 기준 약 98%까지 도달하며, 모델이 훈련 데이터에 잘 적응했음을 보여준다. Validation Accuracy는 초반 빠르게 상승한 뒤 22~25 Epoch 구간에서 84% 근처에서 수렴하며, 최고치인 0.8479를 기록한 것으로 보아 일반화 성능이 안정화되었음을 나타낸다.
두 곡선 간 간격이 점차 벌어지는 것으로 보아, 훈련 후반에는 일부 과적합 경향도 나타났을 가능성이 있다고 본다.

최종 모델의 테스트 정확도는 **0.96583(96.58%)**로, 모델이 새로운 데이터에 대해서도 높은 일반화 성능을 달성했음을 보여준다. 이는 학습 과정에서 적용된 데이터 증강 전략(CutMix), 안정적인 학습률 조정(CosineAnnealingLR), 그리고 정규화 및 드롭아웃 등의 과적합 방지 기법이 효과적으로 작용했음을 나타낸다.



추가적으로 Test Time Augmentation(TTA)이 적용되었다. TTA는 하나의 이미지를 기준으로 좌우 반전(Horizontal Flip), 상하 반전(Vertical Flip), 양방향 반전(Both Flips)을 적용하여, 총 4가지 변형된 이미지가 모델에 입력된다. 이렇게 다양한 관점에서 예측을 수행하고 결과를 평균내어 더 안정적인 최종 예측을 얻을 수 있다. 입력 이미지에 다양한 변형(좌우/상하 뒤집기 등)을 적용한 후, 다수의 예측 결과를 평균하여 최종 예측을 생성함으로써 모델의 강건성을 높였다.

추론은 run.py 스크립트를 통해 수행되며, 모델은 평가 모드(model.eval())에서 작동하고, 불필요한 gradient 계산을 방지하기 위해 torch.no_grad() 문맥에서 실행된다.

5. Conclusion

본 프로젝트에서는 EfficientNet-b1 기반 모델에 다양한 최신 기법을 적용하여, 제한된 모델 크기(≤50MB) 내에서 높은 정확도를 달성하였다. 향후 연구에서는 다음과 같은 방향으로 성능과 실용성을 더욱 향상시킬 수 있다.

1) 다양한 Backbone 아키텍처 비교 실험
EfficientNet-b0, ResNet-18, MobileNetV2 등 다양한 경량 Backbone 모델을 실험하여 정확도, 연산량, 파라미터 수 등을 비교함으로써 자원 제약 환경에 최적화된 구조를 탐색
2) 모델 해석 가능성 향상
CAM 또는 Grad-CAM 기법을 활용하여 모델이 특정 질감에 집중하는 방식과 오분류 원인을 시각적으로 분석하면, 해석 가능성과 신뢰도 향상
3) Few-Shot 및 Meta-Learning 적용
소량의 데이터만으로도 새로운 텍스처 클래스에 대응할 수 있도록, Few-Shot 또는 Meta-Learning 기반 접근을 고려
4) 증강 및 손실 함수 다양화
MixUp, AugMix, Focal Loss 등 다양한 증강 및 손실 함수 기법을 실험하여 클래스 간 오분류 문제를 개선
5) 모델 압축 및 경량화
Knowledge Distillation, Quantization, Pruning 등을 통해 모델의 추론 속도와 메모리 효율을 개선
6) 텍스처 특화 전처리 적용
Gabor 필터 등 텍스처 인식에 효과적인 전처리 기법을 도입하여, 미세한 패턴 차이를 보다 효과적으로 학습

이러한 확장은 본 프로젝트의 강점을 유지하면서 적용성과 확장 가능성을 더욱 강화할 수 있을 것으로 기대된다.