Sketch Image Data Classification WrapUP Report

Haneol Kim* , Bohyun Kim† , Sungjoo Kim‡, Suhyun Jung§, Namgyu Yun¶, Minseok Heo‖ [1]

[1] *Naver Boostcamp AI Tech, CV 21조*

1 프로젝트 개요

스케치 기반 이미지 분류 대회는 주어진 스케치 이미지 데이터를 통해 해당 이미지가 어떤 객체를 나타내는지 분류하는 것을 목표로 한다. 스케치 데이터는 일반적인 사진 이미지와는 달리, 색상, 질감, 세부 정보가 부족하고 단순화된 형태로 되어 있어, 기존의 이미지 분류 모델과는 다른 접근이 필요하다.

이번 대회의 목적은 스케치 이미지 데이터의 특성을 이해하고, 이를 효과적으로 학습하여 높은 정확도로 객체를 분류하는 것이다. 스케치 이미지는 단순한 형태로 이루어져 있지만, 객체의 본질적인 특징을 간결하게 표현하므로, 이러한 데이터를 학습하는 모델은 추상적인 데이터를 처리하는 능력을 향상시킬 수 있다.

본 보고서에서는 대회의 목표와 문제 정의를 명확히 하고, 주어진 데이터를 분석하여 최적의 데이터 전처리 및 학습 방법을 다룬다. 또한, 사전 학습된 모델을 활용하여 스케치 이미지를 분류하는 다양한 실험을 수행하였으며, 앙상블 기법을 통해 모델의 성능을 극대화하는 방법도 다루었다. 이러한 과정에서 얻은 실험 결과와 통찰력을 바탕으로, 스케치 기반 이미지 분류 대회 진행 과정을 기록했다.

2 프로젝트 팀 구성 및 역할

이름	역할
김한얼	Construct pipeline, Code refactoring, Schedule management, Workload management, Model search
김보현	Model search, Dataset curation, Hyperparameter tuning, Induce efficient augmentation
김성주	Data feature analysis, Enhanced evaluation accuracy, Implemention data augmentation technique, Hyperparameter tuning
윤남규	Data preprocessing management, Model search, Data augmentation
정수현	EDA, data generate, model search, feature engineering
허민석	data curation, dataset gSeneration model search, code refactoring

Table 1: 팀 역할

*haneol.kijm@gmail.com
†bhkim4550@gmail.com
‡comaoz1@naver.com
§mandudu6363@gmail.com
¶yynk2012@gmail.com
‖tig0601537@gmail.com

3 프로젝트 수행 절차 및 방법

3.1 Image EDA

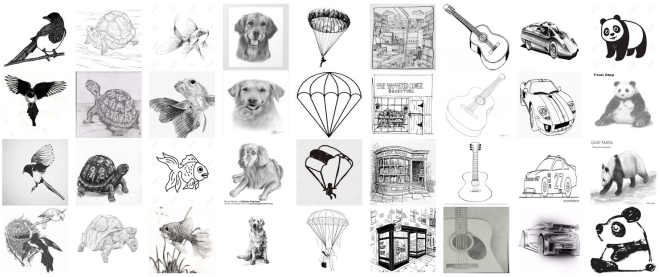


Figure 1: ImageNet Sketch 데이터셋

원본 ImageNet Sketch 데이터셋은 50,889개의 이미지 데이터로 구성되어 있으며 1,000개의 객체에 대해 각각 대략 50개씩의 이미지를 가지고 있다. 일반적인 객체들의 핸드 드로잉 이미지로 구성되어 있으며, 실제 객체를 대표하는 다양한 스타일과 특징을 보여준다. 이번 경진대회에서 제공되는 데이터셋은 1,000개의 클래스에서 정제 후 이미지 수량이 많은 상위 500개의 객체를 선정했으며 총 25,035개의 이미지 데이터가 활용되었다. 해당 이미지 데이터는 15,021개의 학습데이터와 10,014개의 Private와 Public 평가데이터로 나누어 구성된다.

3.2 Directory Descriptions

- **config**
  모델 학습에 필요한 설정 파일들을 관리한다. 다양한 모델의 설정 정보를 담고 있으며, 새로운 설정을 추가하거나 수정할 수 있다.

- **dataset**
  데이터셋과 관련된 코드를 포함하며, 데이터 로딩 및 전처리 과정을 처리한다.

- **engine**
  학습 프로세스와 관련된 코드를 포함하고 있으며, 모델 학습 및 튜닝을 담당한다.

- **model**
  다양한 모델을 정의하는 파일들이 포함된 폴더로, 딥러닝 모델의 구조를 정의한다.

- **utils**
  프로젝트에서 반복적으로 사용되는 유틸리티 함수들을 포함한다.

- **train.py**
  모델을 학습시키는 메인 실행 스크립트로, 전체 파이프라인을 실행한다. 학습, 평가, 데이터 로딩 등을 포함한 전체 과정을 관리한다.

```
|-- config
|   |-- __init__.py
|   |-- config.py
|   |-- config_factory.py
|   |-- custom_nn_config.py
|   |-- deit3_config.py
|   |-- deit3_large_config.py
|   `-- vit_config.py
|-- dataset
|   |-- __init__.py
|   |-- dataloader.py
|   |-- dataset.py
|   `-- transforms.py
|-- engine
|   |-- __init__.py
|   |-- callbacks.py
|   |-- test_runner.py
|   `-- tuner.py
|-- model
|   |-- __init__.py
|   |-- DeiT3.py
|   |-- DeiT3Large.py
|   |-- ResNet18.py
|   |-- ViT.py
|   |-- CoAtNet.py
|   |-- lightning_module.py
|   `-- model_factory.py
|-- utils
|   |-- __init__.py
|   |-- ensemble.py
|   |-- logger.py
|   |-- soup_ensemble.py
|   `-- wandb_example.py
|-- train.py
```

3.3 Pretrained Models

사전학습된 모델을 사용했다. timm 라이브러리를 사용하여 사전 학습된 가중치를 불러와 초기화하여 사용했다. 클래스는 사전 학습된 DeiT3 Vision Transformer 모델을 불러오기 위해 timm 라이브러리를 사용한다.

```python
import timm
import torch.nn as nn
class DeiT3Large(nn.Module):
    def __init__(self, num_classes: Optional[int] = 500,
         pretrained: bool = True, **kwargs):
        super(DeiT3Large, self).__init__()

        self.model = timm.create_model
        ('deit3_large_patch16_224.fb_in22k_ft_in1k',
        pretrained=pretrained,
        num_classes=num_classes,
        **kwargs)
```

사전 학습된 가중치(pretrained=True)와 함께 모델을 불러온다. 22k 클래스에 대해 사전 학습된 후 1k 클래스에 대해 파인 튜닝된 모델이다.

4 Data Preprocessing and Augmentation

4.1 Data Direct Editing

데이터셋에서 직접적으로 불필요한 데이터를 제거하거나 수정하는 방식으로 전처리를 수행했다.

4.2 AutoAugment & Trivial Augment

데이터 증강 기법으로 AutoAugment 및 Trivial Augment를 적용하여 학습 데이터의 다양성을 확보했다. Trivial Augment는 단순한 설정으로도 강력한 성능을 발휘하는 자동화된 데이터 증강 방법이다. Trivial Augmentation (TA)는 이미지 데이터 증강 기법 중 하나로, 간단한 랜덤 변환을 반복적으로 적용하는 방식이다. 이 기법은 이미지 데이터를 모델에 학습시키기 전에 회전, 잘림, 밝기 조정, 대비 조정 등의 간단한 랜덤 변환을 통해 데이터의 다양성을 높이고, 모델이 다양한 상황에 대응할 수 있도록 훈련하는 데 도움을 줍니다. Trivial Augmentation의 주된 목적은 과적합을 방지하고, 모델이 새로운 데이터에 대해 일반화 성능을 향상시키는 것입니다.

TA의 핵심 특징 중 첫 번째는 매우 간단한 방법들을 사용한다는 점입니다. 복잡한 증강 기법이 아닌 가장 단순한 증강 기법들로 성능을 끌어올리는 것이 TA의 중요한 원칙 중 하나입니다. 두 번째 특징은 무작위성(randomness)입니다. TA에서는 증강 강도나 방식이 무작위로 적용되기 때문에, 모델이 특정 패턴에 지나치게 의존하지 않도록 하여 일반화 성능을 향상시키는 데 기여합니다. 세 번째로, Trivial Augmentation은 복잡한 증강 방법들에 비해 매우 가볍고 빠르게 적용할 수 있으며, 성능 면에서 비교적 좋은 결과를 보여줍니다.

따라서, Trivial Augmentation은 복잡하지 않으면서도 효과적으로 이미지 증강을 수행할 수 있는 기본적인 기법으로, 다양한 상황에서 성능 개선을 위한 중요한 도구로 활용됩니다.

4.3 Mixup and Cutmix

두 가지 데이터 증강 기법인 Mixup과 Cutmix를 적용하여 모델의 일반화 성능을 향상시켰습니다. Mixup은 두 이미지의 픽셀 값을 섞는 방식이고, Cutmix는 이미지 일부를 잘라 다른 이미지와 합치는 방식입니다.

```
self.mixup = 0.8
self.cutmix = 1.0
self.mixup_prob = 1.0
self.mixup_switch_prob = 0.5
```

Mixup과 CutMix를 사용하여 학습 데이터를 보다 다양하게 만들어 모델의 일반화 성능을 높이고자 했습니다.

4.4 Label Smoothing

라벨의 불확실성을 반영하기 위해 Label Smoothing 기법을 적용하여 모델이 지나치게 학습되지 않도록 방지했습니다.

```
self.smoothing = 0.1
```

Label Smoothing을 사용하면 레이블의 값이 원핫인코딩이 아닌 예를 들어[0.9,0.05,0.05]처럼 약간 부드럽게 만들어서 모델이 지나치게 한 클래스에 확신을 갖지 않도록 하고, 다른 클래스에 대한 예측 확률도 약간 고려하도록 만들었습니다.

5 Optimization Techniques

5.1 AdamW & Weight Decay

최적화 기법으로 AdamW를 사용하고, Weight Decay를 적용하여 과적합을 방지하며 일반화 성능을 높였습니다.

5.2 Learning Rate Warmup & Cosine Scheduler

초기 학습률을 천천히 증가시키는 Learning Rate Warmup 기법과 Cosine Scheduler를 함께 사용하여 학습 초기 안정성을 높였습니다. Cosine Scheduler는 학습이 진행됨에 따라 학습률을 점차 감소시킵니다.

```
self.sched = 'cosine' # cosine, step
self.warmup_lr = 1e-6
self.warmup_epochs = 5
```

cosine 함수를 기반 학습률을 감소시켰습니다. 학습이 진행됨에 따라 학습률을 점차 줄여가면서, 특히 후반부에 아주 작은 학습률을 사용해 학습을 마무리해서 학습 초기에는 상대적으로 큰 학습률을 사용하고, 중간부터는 학습률이 급격히 줄어들면서 수렴하게 했습니다. 또한 Warmup을 통해서 학습 초기 단계에서는 모델의 가중치가 랜덤하게 초기화되어 있기 때문에, 너무 큰 학습률을 사용하면 모델이 불안정해지는 것을 방지했습니다. 작은 학습률로 시작해 가중치가 어느 정도 안정된 후 학습률을 증가시켜서 학습 안정성을 높였습니다. 초반에 높은 학습률을 사용하면 모델이 너무 빨리 수렴하여 로컬 미니멈에 빠질 수 있습니다. 이를 방지하기 위해 초기에는 학습률을 낮게 유지하고, 점차 원하는 값까지 올리면서 빠른 수렴을 방지하게 했습니다.

5.3 StepLR Scheduler

또 다른 스케줄러로 StepLR을 사용하여 일정 에폭마다 학습률을 감소시키는 방식도 실험했습니다. PyTorch에서 제공하는 StepLR 학습률 스케줄러는 일정한 주기(step_size)에 따라 학습률(learning rate)을 감소시키는 역할을 합니다. 주어진 step_size 에포크마다 학습률이 gamma 값만큼 곱해져 감소합니다. 이는 학습 중에 과적합을 방지하고, 학습이 안정화되도록 학습률을 조절하는 데 효과적입니다.

- optimizer: 학습률을 조정할 옵티마이저
- step_size: 학습률을 감소시키는 주기 (에포크 단위)
- gamma: 학습률 감소 비율 (기본값: 0.1)
- last_epoch: 마지막 에포크 인덱스 (기본값: -1)
- verbose: True로 설정 시 학습률 변경 시마다 메시지가 출력됨 (기본값: False)

StepLR을 사용하는 예시입니다. 학습이 진행됨에 따라 매 30 에포크마다 학습률이 0.1배로 감소하는 방식으로 설정하였습니다.

```
import torch
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR

# 옵티마이저설정
optimizer = Adam(model.parameters(), lr=0.05)

# StepLR 스케줄러설정
scheduler = StepLR(optimizer, step_size=30, gamma=0.1)

# 학습루프
for epoch in range(100):
    train(...)  # 학습함수
    validate(...)  # 검증함수
    scheduler.step()  # 스케줄러업데이트
```

위 예시에서 lr은 다음과 같이 조정됩니다:

- 에포크 0 29: 학습률 0.05
- 에포크 30 59: 학습률 0.005
- 에포크 60 89: 학습률 0.0005
- 에포크 90 100: 학습률 0.00005

6 Model Freezing & Fine-tuning

6.1 Head-Only Model Freeze

모델의 헤드 부분을 제외하고 나머지 파라미터는 고정한 상태에서 학습을 진행하여 전이 학습 효과를 극대화했습니다.

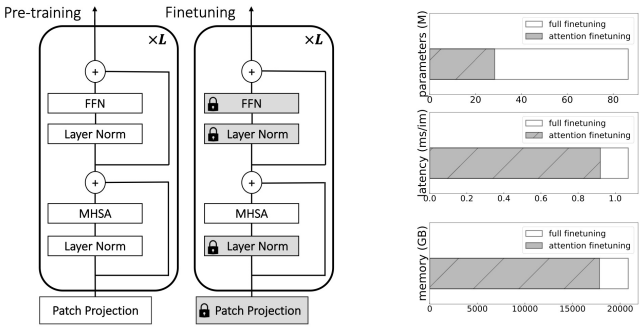


Figure 2: finetune only the attentions (flag –attn-only)

6.2 Attn-Only Model Freeze

DeiT 모델 개발진들의 가이드를 참고하여 Attention 파라미터만 학습하도록 설정하는 방법을 적용했습니다. DeiT (Data-Efficient Image Transformers)는 ICML 2021에서 소개된 모델로, 이미지 트랜스포머의 데이터 효율성을 향상시키는 것을 목표로 합니다. Attention-Only Fine-Tuning은 모델을 더 높은 해상도로 적응시키거나 전이 학습을 수행하기 위해 Attention 파라미터만을 미세 조정하는 방법입니다. MLP 패치 프로젝션은 선형 패치 프로젝션을 MLP 패치 프로젝션으로 대체하는 것을 제안합니다. 이 전처리 방식은 BeiT와 같은 마스크 기반 자가 지도 학습에 호환되며, 성능 향상에 기여합니다.

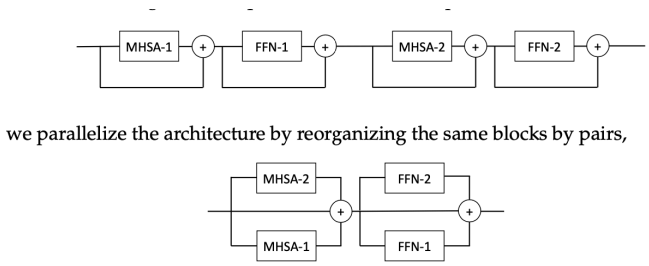


Figure 3: MLP patch projection

7 Ensemble and Model Soup

7.1 Ensemble Method (Mean, Vote)

다양한 모델의 예측 결과를 평균 내거나, 투표 방식을 적용하여 앙상블을 수행했습니다.

```
def ensemble_predict(self, models, dataloader):
    ensemble_predictions = []
    for i, model in enumerate(models):
        trainer = Trainer(devices=1)
        predictions = trainer.predict(model, dataloaders=
            dataloader)
        prediction_tensor = torch.cat(predictions, dim=0)
        print(f"{i}th model prediction output shape is {
            prediction_tensor.shape}")
        ensemble_predictions.append(prediction_tensor)
    ensemble_output = torch.stack(ensemble_predictions).mean
        (dim=0)
    return ensemble_output
```

각 모델이 동일한 입력에 대해 예측한 값들이 ensemble_predictions에 담겨 있습니다. 이 값들은 모두 동일한 크기(shape)를 가지고 있어야 하며, 예측값은 보통 모델의 출력 (logits 또는 확률) 형태로 저장됩니다. torch.stack 함수는 ensemble_predictions 리스트에 있는 텐서들을 새로운 차원을 추가하면서 이어 붙이는 역할을 합니다. 예를 들어, 각 예측 텐서의 크기가 (batch_size, num_classes)라고 가정하면, torch.stack(ensemble_predictions)의 결과는 (num_ensembles, batch_size, num_classes)가 됩니다. 여기서 num_ensembles는 예측에 참여한 모델 또는 예측의 개수입니다. 여러 개의 예측값을 하나의 3차원 텐서로 만듭니다. mean(dim=0)은 앙상블을 위한 연산입니다. dim=0으로 지정된 차원을 기준으로 평균을 구합니다. dim=0은 앙상블된 모델이나 예측값의 차원이므로, 여러 예측값의 평균을 구해 최종 앙상블 예측값을 계산하게 됩니다. 이렇게 하면 여러 예측 결과를 평균내어 단일 예측값을 도출할 수 있습니다. 이 방법은 앙상블의 평균 방식을 사용한 것입니다.

7.2 Model Soup (Uniform Soup & Greedy Soup)

여러 체크포인트를 결합하는 Model Soup 방법론을 실험했으나, 여러 모델의 체크포인트를 앙상블하는 상황에서 이 방법은 적용하지 못했습니다.

```
def ensemble(self):
    models = self.load_models()
    if self.method == 'uniform_soup':
        model = self.uniform_soup(models)
    elif self.method == 'greedy_soup':
        model = self.greedy_soup(models)
    elif self.method == 'ensemble_predict':
        pass
    else:
        raise ValueError("Invalid ensemble method")
    if self.method == 'ensemble_predict':
            predictions = self.ensemble_predict(models, self
                .test_loader)
    else:
        predictions = model.predict(self.test_loader)

    self.save_to_csv(predictions=predictions)
```

self.uniform_soup(models)을 호출하여 Uniform Model Soup 방식을 적용합니다. 여러 모델의 파라미터를 단순 평균하여 하나의 모델로 결합하는 방식을 사용하려고 했습니다. self.greedy_soup(models)을 호출하여 Greedy Model Soup 방식을 적용해서 성능이 좋은 모델들만 선택적으로 결합하여 하나의 모델로 만들어보고 싶었습니다.

8 Advanced Techniques

8.1 Test-Time Augmentation (TTA)

추론 과정에서 데이터 증강 기법을 적용하여 여러 번 추론한 결과를 평균 내어 최종 예측 성능을 향상시켰습니다. Test Time Augmentation (TTA)는 테스트 시간에 데이터를 증강하여 모델의 성능을 개선하는 방법입니다. TTA의 목적은 훈련 중에 사용하는 데이터 증강 방식과 유사하게, 테스트 이미지에 다양한 변형을 가한 후 여러 번 모델에 입력해 예측값을 얻고, 이를 결합하여 최종 예측값을 도출하는 것입니다. 예를 들어, 깨끗한 이미지(원본)를 한 번만 모델에 입력하는 대신, 여러 가지 증강된 이미지를 모델에 입력하고 그 결과를 평균, 최댓값, 기하평균 등으로 결합하여 최종 예측값을 얻는 방식입니다.

```
import ttach as tta
tta_model = tta.SegmentationTTAWrapper(model, tta.aliases.
    d4_transform(), merge_mode='mean')
```

이 코드는 분할 모델에서 TTA를 사용하는 방법입니다. d4_transform()을 사용하여 4가지 변형(수평 및 수직 반전과 90도 회전)을 적용하고, 최종 예측값을 평균(mean)으로 결합했습니다.

```
tta_model = tta.ClassificationTTAWrapper(model, tta.aliases.
    five_crop_transform())
```

이 코드는 분류 모델에서 TTA를 사용하는 방법입니다. 여기서는 FiveCrop 변형을 사용하여 5개의 작은 이미지를 잘라내어 예측을 수행했습니다.

8.2 AMP Mixed Precision

AMP를 사용하여 혼합 정밀도 학습을 진행하여 메모리 사용량을 절약하고 학습 속도를 높였습니다.

```
self.use_amp = True
```

일반적으로 모델은 32비트 부동소수점 연산(Full Precision)을 사용하여 학습을 진행하지만, AMP를 사용하면 일부 연산에서 16비트 부동소수점(Half Precision)을 사용하여 메모리 사용량을 줄이고 학습 속도를 향상시켰습니다. 참고문헌에서는 3배정도 빨라질 수 있다고 되어있었지만 본 저자들이 썼을 땐 6배 정도 빨라지는 효과를 경험했습니다.

9 Additional Techniques

9.1 Drop Path Rate

Drop Path 기법을 사용하여 모델의 레이어별로 경로를 무작위로 드롭하는 방식으로 과적합을 방지하고 일반화 성능을 높였습니다.

```
self.drop_path_rate = 0.0
```

Drop Path를 도입햇거 학습 단계에서 특정 레이어나 블록을 확률적으로 건너뛰도록해서 특정 경로를 따르는 대신 다른 경로를

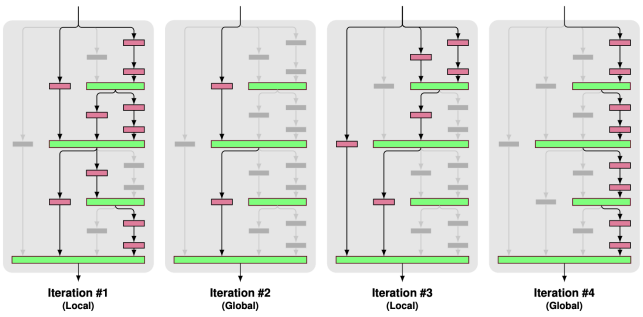


Figure 2: **Drop-path.** A fractal network block functions with some connections between layers disabled, provided some path from input to output is still available. Drop-path guarantees at least one such path, while sampling a subnetwork with many other paths disabled. During training, presenting a different active subnetwork to each mini-batch prevents co-adaptation of parallel paths. A global sampling strategy returns a single column as a subnetwork. Alternating it with local sampling encourages the development of individual columns as performant stand-alone subnetworks.

Figure 4: drop_path

택하게 되며, 이로 인해 모델의 각 레이어가 서로 다른 데이터를 처리하게 되어 과적합을 줄이고 일반화 성능을 향상시켰습니다.

9.2 ASHA

ASHA는 자원을 절약하면서도 효율적인 하이퍼파라미터 튜닝을 가능하게 하는 알고리즘으로, 최적의 하이퍼파라미터를 찾기 위해 실험을 관리하는 데 사용되었습니다. 병렬 처리를 효과적으로 활용하고, 초기 중단 전략을 사용하여 시간과 자원을 절약하고 싶어서 도입하게 되었습니다. 먼저, 많은 수의 하이퍼파라미터 조합을 동시에 학습시켰습니다. 각 하이퍼파라미터 조합의 성능을 비교한 후, 성능이 좋지 않은 조합들은 일정 단계에서 초기 중단을 시킵니다. 성능이 좋은 조합들만 계속 학습을 진행하여 더 많은 자원을 할당합니다.

9.3 Population-Based Training (PBT) & PB2

PBT는 여러 모델을 동시에 학습시키며 성능이 좋은 모델의 하이퍼파라미터를 공유하는 방식으로 실험을 관리하는 방법입니다. PB2는 PBT의 확장 버전으로, 이 방법을 통해 다양한 실험을 시도했습니다.

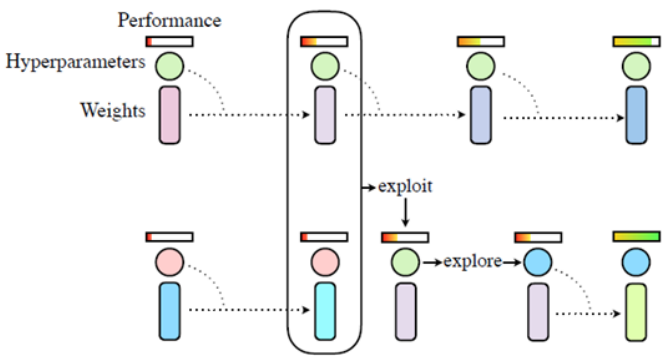


Figure 5: PBT

9.4 Pre-commit for Code Improvement

코드 품질 향상을 위해 Pre-commit을 적용하여 실시간 코드 스타일을 개선하고, 협업을 원활히 진행했습니다.

10 Challenges

10.1 EMA & Gradient Clipping

EMA와 Gradient Clipping은 적용을 시도했으나 구현에 어려움을 겪어 적용하지 못했습니다. 다음 프로젝트에서는 PyTorch에서 지원하는 torch.optim.swa_utils 패키지를 통해 EMA와 유사한 Stochastic Weight Averaging(SWA)을 활용해보면 어떨까 싶습니다. 또 구현하지 못했던 코드를 수정해서 모델의 매번 업데이트 때마다 EMA 값을 갱신하여 저장하는 방법을 성공하면 좋겠습니다. PyTorch에서 모델 파라미터를 직접 추적하여 EMA 업데이트를 수행하는 코드를 작성했는데 다음 프로젝트에서 구현할 수 있으면 좋겠습니다.

10.2 Object Detection for Preprocessing

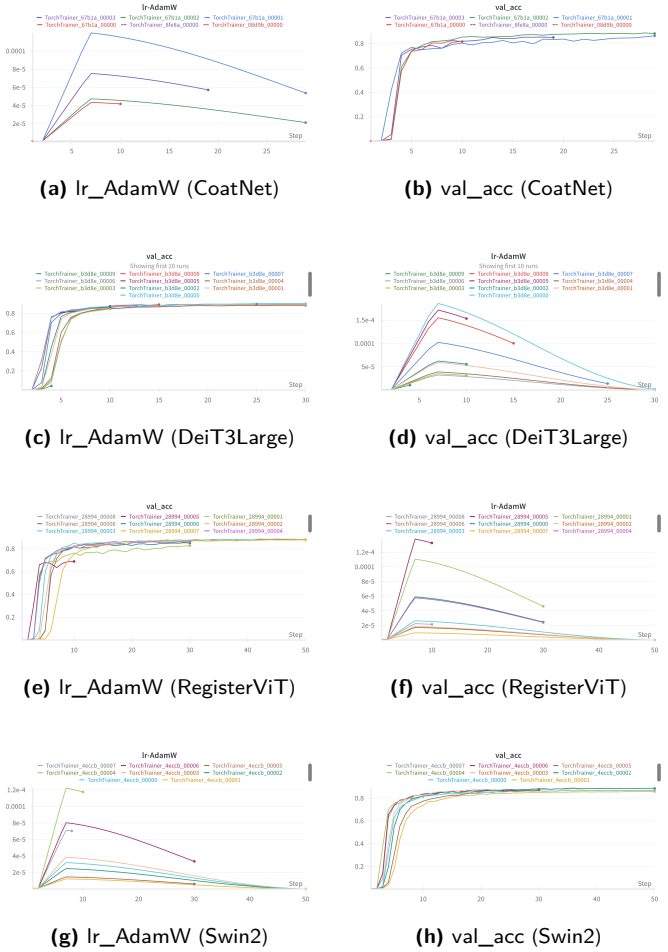
객체 검출을 통해 이미지의 불필요한 부분을 제거하는 방식으로 전처리를 진행하고자 했습니다.

11 프로젝트 수행 결과

Model	Version
Resnet18	resnet18
CoAtNet	coatnet_bn_0_rw_224.sw_in1k
CoAtNet	coatnet_rmlp_2_rw_224.sw_in12k_ft_in1k
ConvNeXt	convnext_base
DeiT3	deit3_base_patch16_224.fb_in22k_ft_in1k
DeiT3	vit_base_patch16_224
ViT	vit_base_patch16_224
EfficientNet	efficientnet_b0
EVA-CLIP	eva02_base_patch14_448.mim_in22k_ft_in22k_in1
RegisterViT	vit_mediumd_patch16_reg4_gap_256.sbb2_e200_
Swin2	swinv2_small_window16_256.ms_in1k
EfficientNet	swinv2_small_window16_256.ms_in1k

Table 2: Model

이번 프로젝트를 통해 여러 최신 분류 모델 논문들을 참고하여, DeiT3의 fine-tuning 코드 방식과 CoAtNet 논문에서 다룬 hyper-parameter 조정 방법을 주로 활용했습니다. 특히 DeiT3의 fine-tuning 코드와 timm 라이브러리의 코드들을 참고하여, 학습의 정확도를 크게 향상시킬 수 있었습니다. 이러한 방법들을 전부 적용한 결과, 팀은 90% 달하는 높은 정확도를 달성할 수 있었습니다. 코드 한줄 차이로 50% 머무르던 정확도가 5일 전까지 해결되지 않아 큰 어려움을 겪었습니다. 이 경험을 통해 코드 검증이 반드시 필요한 절차임을 절감하게 되었으며, 팀원들과의 협업을 통해 코드 리뷰와 검증 과정을 더욱 철저히 해야 한다는 중요한 교훈을 배웠습니다. attention만 얼리고 나머지 부분을 fine-tuning하는 기법을 사용하여 메모리 소모를 약 20% 절감하면서도 모델 성능은 2% 향상시킬 수 있었습니다. 또한, mixed precision을 적용해 학습 속도를 4 6배까지 끌어올렸던 경험은 매우 인상적이었습니다. 이번 프로젝트에서 가장 큰 한계는 시작이 늦었다는 점이었습니다. 팀 프로젝트 경험이 부족했던 저는 프로젝트 초반에 일을 각자에게 할당하지 않았고, 앞으로 프로젝트를 미리 진행하고 역할분담을 제대로 해야겠다고 약속했습니다. 일을 체계적으로 할당한 이후, 팀원들은 엄청난 노력을 기울여 작업을 해주었지만, 결과를 뽑아내기에는 시간이 부족했습니다. 이러한 부분이 가장 아쉬웠던 점이었습니다. 이번 프로젝트의 교훈을 바탕으로, 다음 프로젝트에서는 더욱 체계적인 업무 분담과 코드 작성 전략을 세울 것입니다. 모델 학습 코드를 빠르게 완성하고, 가능한 한 일찍부터 체크포인트를 저장하여 다른 팀원들과 공유할 수 있는 시스템을 구축할 것입니다. 이를 통해 더 많은 방법론을 시도하고, 앙상블 등의 추가 기법을 적용할 수 있는 여유를 확보할 계획입니다. 이번 프로젝트에서 배운 교훈들을 활용해 더 나은 프로젝트 성과를 내기 위해 노력할 것입니다.



(a) lr_AdamW (CoatNet) **(b)** val_acc (CoatNet)

(c) lr_AdamW (DeiT3Large) **(d)** val_acc (DeiT3Large)

(e) lr_AdamW (RegisterViT) **(f)** val_acc (RegisterViT)

(g) lr_AdamW (Swin2) **(h)** val_acc (Swin2)

Figure 6: Training Results for Different Models

“‘latex article [utf8]inputenc

12 개인회고

12.1 김한얼

저는 기본적으로 이 팀에서 팀장 역할을 맡아서 진행했습니다.

12.1.1 나는 내 학습목표 달성을 위해 무엇을 어떻게 했는가?

대학원에 다닌 경험이 있고 코딩을 짜본 경험도 있었기에, 처음엔 추석 기간 동안 독단적으로 베이스 라인 코드를 전부 짜서 팀원들에게 제공하고자 했습니다. 그러나 추석 기간 동안 팀원들에게 딱히 일을 시키지 않았고, 보통 사람들은 일을 시키지 않으면 주도적으로 일을 하지 않는다는 사실을 이 때 깨달았습니다. 이러한 팀 분위기가 마음에 들지 않은 팀원 한 분이 이 점을 지적해주셨습니다. 그 이후론 모든 팀원에게 일을 각자 분배하고, 저는 해온 일을 검사, 피드백, 직접 수정 등의 역할을 하며 주도적으로 프로젝트 진행의 전반적인 부분을 담당했고, 이를 무사히 마무리할 수 있었습니다.

12.1.2 나는 어떤 방식으로 모델을 개선했는가?

저는 수많은 다른 최신 분류 모델의 논문들을 많은 부분 참고했습니다. 특히 DeiT3의 fine-tuning하는 코드 방식과 수업에서 다루었던 CoAtNet 논문의 fine tuning hyper-parameter 부분을 보고, 공통적으로 이들이 쓰는 방법들은 전부 적용해야 정확도를 끌어올릴 수 있다는 사실을 깨달았습니다. 그 이후 DeiT3 코드에서 쓰는 timm 코드를 참고하여 전부 적용하였고, 결과적으로 다른 팀들과 뒤쳐지지 않는 90% 정확도까지 얻는 것에 성공했습니다.

12.1.3 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떤 깨달음을 얻었는가?

제가 만든 코드 덕분에 저희 팀은 무사히 코드에 대한 걱정 없이 학습을 수행할 수 있었지만, 제가 일으킨 치명적인 코드 실수로 인해 5일 전까지도 50% 정확도라는 처참한 결과로 우울해 하고 있었습니다. 코드 검증은 반드시 필요한 절차이며, 코드를 독단적으로 짤 경우 생기는 무수한 오류를 혼자만의 힘으로 검증할 수 없음을 느꼈습니다.

12.1.4 전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

전에는 무식하게 모델이 무거워서 사용하기 힘들면 무식하게 batch size를 줄이는 방식만 사용했었습니다. 그러나 다양한 강의, 동료의 공부 및 조사한 내용 추천을 듣고 다양한 방식을 통해 모델 사이즈를 gpu에 끼워 맞추는 테크닉들을 깨달았고, 체감이 크게 되었습니다. 특히 attention만 얼리지 않고 fine tuning을 진행해서 20% 메모리 소모를 줄이면서 fine tuning 성능은 2% 정도 더 올리고, mixed precision을 통해 체감 학습 속도를 4-6배 올릴 수 있었던 점들이 인상 깊은 경험이 되었습니다.

12.1.5 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

한계는 늦게 시작한 것이 저희 팀의 한계였습니다. 팀 프로젝트 경험이 부족했던 저는 1주일을 남기고 서야 각자에게 일을 할당하고, 일을 시켜야 한다는 것을 깨달았습니다. "각자 알아서 잘해" 같은 말은 실제 프로젝트에서는 통하지 않는 말인 것이죠. 실제로 일을 시키기 시작한 이후로 팀원들은 각자가 엄청난 노력으로 열심히 일을 해주었지만, 결과를 뽑는 방법을 늦게 깨달은 저희는 그걸 적용할 시간이 부족했습니다.

12.1.6 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

다음 프로젝트는 반드시 모델 학습 코드를 3일 안에 작성하여, 휴가 기간 동안 무거운 모델 위주로 체크포인트를 저장하고, 체크포인트를 이른 시기에 미리미리 뽑아내어 다른 사람의 서버에 전송하고 앙상블을 할 수 있도록 준비할 것입니다. 또한 이렇게 모델을 미리 선정하고 돌리면 가지는 장점이, 여러가지 다양한 방법론을 찾아보고 적용해볼 기회가 생긴다는 것입니다. 이번 프로젝트의 실패를 계기로 더 나은 업무 프로세스를 확보하도록 노력할 것입니다.

12.2 김보현

12.2.1 나는 내 학습목표 달성을 위해 무엇을 어떻게 했는가?

이번 프로젝트에서 저는 *Model search*, *Dataset curation*, *Hyper-parameter tuning*, 그리고 *Induce efficient augmentation*이라는 핵심 역할을 맡았습니다. 각 단계에서 목표를 설정하고 그에 맞는 방법론을 적용했습니다. 특히 모델 서치 과정에서 다양한 사전 학습된 모델을 비교하며 최적의 모델을 찾아내기 위해 많은 시간을 투자했습니다. 또한, 데이터셋을 구성하고 적절한 증강 기법을 적용하여 모델이 보다 다양한 상황에서도 강건하게 학습될 수 있도록 했습니다.

12.2.2 나는 어떤 방식으로 모델을 개선했는가?

모델 성능을 개선하기 위해 다양한 방법을 시도했습니다. 먼저 *Model search* 단계에서 여러 사전 학습된 모델을 탐색한 뒤, 가장 성능이 좋은 모델을 선택하여 Hyperparameter tuning을 진행했습니다. 또한, 학습 데이터를 효율적으로 구성하기 위해 *Dataset curation*을 통해 불필요한 데이터를 제거하고, 중요한 데이터만 남겨 모델의 학습 효율성을 극대화하려 했습니다. 마지막으로, *Induce efficient augmentation*에서는 효과적인 데이터 증강 기법을 적용하여 모델의 일반화 성능을 높였습니다.

12.2.3 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떤 깨달음을 얻었는가?

다양한 모델 서치와 하이퍼파라미터 튜닝을 통해 최적의 성능을 달성하는 것이 쉬운 일이 아님을 깨달았습니다. 특히, 데이터 증강 기법의 중요성을 다시 한 번 느꼈습니다. 잘 설계된 증강 기법을 적용했을 때, 모델의 성능이 크게 향상되었음을 확인할 수 있었습니다. 이러한 경험을 통해, 기본적인 하이퍼파라미터 조정이나 모델 변경만으로는 한계가 있으며, 데이터 처리와 증강이 모델 성능에 매우 중요한 요소라는 점을 배웠습니다.

12.2.4 전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

이전 프로젝트에서는 주로 기본적인 데이터 증강만 사용했지만, 이번 프로젝트에서는 효율적인 증강 기법을 연구하고 적용했습니다. 이를 통해 모델의 학습 속도와 성능이 개선되는 것을 경험했습니다. 또한, 하이퍼파라미터 튜닝에 더 많은 시간을 할애하여 모델의 성능을 세밀하게 조정하는 방법을 새롭게 시도했고, 그 결과로 모델의 성능을 극대화할 수 있었습니다.

12.2.5 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

데이터셋 구성 과정에서 더 세밀하게 데이터를 분석하고 분류할 시간이 부족했던 점이 아쉬웠습니다. 더 많은 시간을 투자하여 데이터를 더욱 정교하게 다듬었다면, 모델의 성능이 조금 더 향상될 수 있었을 것이라 생각합니다. 또한, 모델 서치 과정에서 최신의 다양한 모델을 충분히 탐색하지 못한 것도 아쉬운 부분이었습니다.

12.2.6 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

다음 프로젝트에서는 더 다양한 모델을 탐색하고, 그에 따른 하이퍼파라미터 튜닝을 보다 세밀하게 진행해보고 싶습니다. 또한, 데이터 증강 기법을 더욱 발전시키고, 증강된 데이터를 통해 모델의 성능을 최적화하는 방법론을 심도 있게 연구할 예정입니다.

12.3 허민석

12.3.1 나는 내 학습목표 달성을 위해 무엇을 어떻게 했는가?

CNN기반의 모델을을 돌려보기 위해서 CNN기반의 모델들을 찾아보았습니다.

12.3.2 나는 어떤 방식으로 모델을 개선했는가?

Train 데이터에서 불필요하거나 너무 noisy한 데이터가 많아 보여서 이를 정제하는 작업을 진행하였습니다. 정제하는 작업을 통해서 성능을 개선할 수 있다고 생각하였습니다.

12.3.3 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떤 깨달음을 얻었는가?

수작업을 통해서 데이터 정제를 시도하였습니다. 그 결과로 데이터 정제 작업을 통해 성능 향상을 기대했지만 정

"latex 정제된 데이터로 학습이 되면서 test 데이터는 정확하게 분류하지 못하는 문제가 발생하였습니다. 데이터를 깔끔하게 정리하는 것이 성능 향상에 있어서 무조건 좋을 것이라고 생각하였는데 직접 작업해보고 나니 정제된 데이터를 학습하니 일반화 성능이 떨어져서 오히려 성능이 제대로 나오지 않는 경우가 발생한다는 것을 깨달았습니다.

12.3.4 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

데이터 정제 작업을 진행하면서 오랜 시간을 쏟게 되면서 프로젝트 진행 상황에 대해 놓치게 되었는데, 한번 흐름을 놓치니까 다시 잡기 힘들었습니다. 개인적인 목표를 달성하지 못한 것에 대해 아쉬움이 남지만 프로젝트를 진행하면서 얻게 된 지식들이 더 많은 것 같습니다.

12.3.5 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

다음 프로젝트에서는 강의를 미리 다 수강하고, 프로젝트에 도움이 되는 정보를 사전에 확보한 후 심화 과정을 진행해보고 싶습니다. 또한, 이번 수작업 방식이 많은 시간이 걸리고 결과가 좋지 않았던 만큼, 다음에는 더 나은 자동화된 방법을 찾아보겠습니다.

12.4 윤남규

12.4.1 나는 내 학습목표 달성을 위해 무엇을 어떻게 했는가?

프로젝트 초기부터 추석 연휴까지는 강의와 과제들에 집중하며 참고할 만한 논문들을 찾았습니다. 관련 주제를 명확하게 설정하고 배경 지식을 쌓는 데 집중했으나, 그 과정에서 방향을 잘못 설정했다는 점을 깨달았습니다. 또한, 프로젝트 경험이 부족했기에 도구들에 익숙해지려 노력했습니다.

12.4.2 나는 어떤 방식으로 모델을 개선했는가?

주로 데이터 전처리에 집중하였고, 모델 개선 측면에서는 다양한 기법을 적용해보는 데 미숙함이 있었습니다. 다음 프로젝트에서는 다양한 모델과 기법을 공부하고 적용하는 것을 목표로 할 것입니다.

12.4.3 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떤 깨달음을 얻었는가?

데이터 전처리 작업을 통해 모델 성능이 7-9% 정도 개선되었습니다. 이를 통해 데이터 전처리 과정이 성능 향상에 중요한 역할을 한다는 점을 깨달았습니다.

12.4.4 전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

데이터 전처리 과정에서 이미지 내의 여러 객체를 분리하고 이를 개별적으로 학습시킴으로써 더 정확한 결과를 얻을 수 있었습니다.

12.4.5 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

팀 프로젝트 경험이 없어서 협업 과정에서 미숙함을 느꼈습니다. 특히 이미지 프로젝트에 대한 경험 부족이 아쉬웠습니다.

12.4.6 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

다음 프로젝트에서는 보다 다양한 모델을 직접 구현하고 실험해보면서 팀원들과의 협업 경험을 쌓는 것이 목표입니다.

12.5 김성주

12.5.1 나는 내 학습목표 달성을 위해 무엇을 어떻게 했는가?

프로젝트 초기에는 부스트코스 강의를 빠르게 듣고 프로젝트에 적용할 수 있는 요인들을 찾고, 이를 적용했습니다. 또한 협업을 위해 GitHub와 원격 서버를 사용하는 방법을 익혔습니다.

12.5.2 나는 어떤 방식으로 모델을 개선했는가?

대회의 스케치 데이터의 특징을 분석하고, 이 데이터를 타겟으로 하는 논문들을 참고하여 전처리와 증강 방식을 개선했습니다. 특히 Test Time Augmentation(TTA)을 적용하여 성능을 향상시키려고 노력했습니다.

12.5.3 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떤 깨달음을 얻었는가?

스케치 데이터는 같은 클래스 내에서도 다양한 스타일을 가지며, 중복되는 이미지도 많다는 사실을 알게 되었습니다. 이를 해결하기 위해 smoothing을 적용해 모델이 클래스 정답에 너무 의존하지 않도록 학습했습니다.

12.5.4 전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

TTA를 통해 모델이 같은 이미지를 보더라도 약간의 변화에 더 잘 적응하도록 했습니다. 다양한 변화들을 통해 더 정확한 결과를 도출할 수 있었습니다.

12.5.5 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

초기에 더 많은 시간을 프로젝트에 투자하지 못해 충분한 실험을 진행하지 못했습니다. 또한 GitHub 사용 능력이 부족했던 점이 아쉬웠습니다.

12.5.6 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

다음 프로젝트에서는 EDA와 모델 학습을 동시에 진행하며 데이터를 적합하게 다듬는 과정을 시작할 것입니다. GitHub를 더 적극적으로 활용할 계획입니다.

12.6 정수현

12.6.1 나는 내 학습목표 달성을 위해 무엇을 어떻게 했는가?

데이터 전처리 과정에서 다중 객체를 포함한 이미지를 단일 객체 이미지로 분리하기 위해, 학습 데이터셋에 Yolov8s 모델을 이용하여 객체 탐지를 수행했습니다.

12.6.2 나는 어떤 방식으로 모델을 개선했는가?

객체 탐지로 전처리된 데이터셋을 통해 모델 성능을 개선하였으며, 하이퍼파라미터 튜닝을 통해 성능을 더욱 높였습니다.

12.6.3 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떤 깨달음을 얻었는가?

깨끗하게 전처리된 데이터셋보다는 기존 학습 데이터셋을 사용한 모델이 일반화 성능에서 더 좋다는 점을 발견했습니다.

12.6.4 전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

Weakly-supervised object detection에 대해 새롭게 배웠으며, 이를 통해 bbox annotation을 효율적으로 달 수 있다는 것을 알게 되었습니다.

12.6.5 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

시간을 계획적으로 사용하지 못한 점이 아쉬웠습니다. 이로 인해 실험 결과의 완성도가 떨어졌습니다.

12.6.6 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

구체적인 프로젝트 계획을 세우고, 시간을 효과적으로 관리하여 더 나은 결과를 얻도록 할 것입니다.