# 중고차 이미지 분류

## 1. 프로젝트 개요

### 1.1. 프로젝트 주제

최근 자동차 산업의 디지털 전환과 더불어, 다양한 차종을 빠르고 정확하게 인식하는 기술의 중요성이 커지고 있습니다. 특히 중고차 거래 플랫폼, 차량 관리 시스템, 자동 주차 및 보안 시스템 등 실생활에 밀접한 분야에서는 정확한 차종 분류가 핵심 기술로 떠오르고 있습니다.

이미지 기반 차종 인식 기술은 기존의 수작업 방식에 비해 높은 정확도와 효율성을 제공하며, 인공지능(AI)을 활용한 자동화 기술이 빠르게 발전하고 있습니다. 그중에서도 다양한 차종을 세밀하게 구분할 수 있는 능력은 실제 서비스 도입 시 차별화된 경쟁력을 좌우하는 요소로 작용합니다.

이에 따라 저는 실제 중고차 차량 이미지를 기반으로 한 차종 분류 AI 모델 개발하고자 합니다.

### 1.2.학습 환경 구성

- **GPU:** NVIDIA GeFprce RTX 4070 Ti SUPER
- **CPU**: Intel(R) Core(TM) i5-14400F @ 2.50GHz
- **운영체제**: Windows 11 Home

## 2. 프로젝트 수행 결과

### 2.1. Dataset Info

▼ 전체 데이터셋 통계

- Train: 396 클래스, 총 33,137장 (클래스당 평균 84장)
- Test: 총 8,258장
- Test.csv: `ID` , `img_path`
- 제출 형식: `sample_submission.csv` → 396개의 class 확률값 필요 (Softmax output)

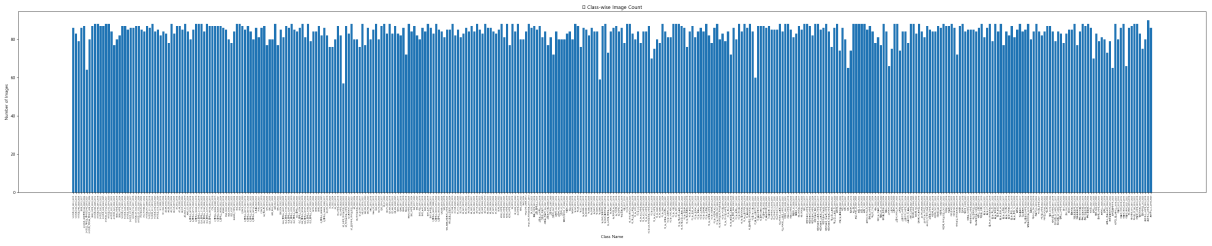
- **평가 방법**

  ▼ Log Loss

$$\text{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j} \cdot \log(p_{i,j})$$

### 2.2. EDA 및 가설 설정

2.2.1. 클래스 분포 분석



전체 데이터셋의 클래스별 이미지 수를 분석한 결과, 클래스 간 분포는 대체로 균일한 편으로 확인되었습니다. 대부분의 클래스는 약 60~80장의 이미지를 포함하고 있어, 통계적으로 유의미한 수준의 클래스 불균형은 존재하지 않았습니다. 이에 따라 클래스 밸런싱 기법은 우선순위에서 제외하였습니다.

2.2.2. 이미지 해상도 및 밝기 분포 분석

```
             width       height    brightness
count  33137.000000  33137.000000  33137.000000
mean     690.290732    436.583819    108.586380
std      150.817513     74.954611     20.269282
min       91.000000     61.000000     54.512823
25%      560.000000    395.000000     93.649401
50%      688.000000    430.000000    106.635201
75%      830.000000    470.000000    121.886312
max     1881.000000   1310.000000    225.306517
```

이미지 해상도는 주로 500~800 픽셀 구간에 집중되어 있었으며, 최대 해상도는 1881×1310으로 다소 큰 편에 속했습니다. 이에 따라 모델 입력 크기를 상위 분포에 맞춰 조정하거나 고해상도 이미지를 활용한 학습 실험을 고려하였습니다.

또한 밝기의 분포가 평균 108 수준이며, 최소 54에서 최대 225까지 폭넓게 분포함에 따라  밝기 보정이나 색상 관련 데이터 증강 기법 (Color Jitter, Histogram Equalization 등)이 모델 성능 향상에 기여할 수 있을 것으로 판단하였습니다. 해당 요소를 기반으로 실험 가설을 설정하였습니다.

2.2.3. 중복 이미지 탐지

MD5 해시 기반의 중복 이미지 탐지 기법을 적용한 결과, 전체 이미지(33,137개) 중 1,637개의 중복 이미지가 존재하는 것으로 확인되었습니다. 이는 전체의 약 4.9%에 해당하는 수치이기 때문에 데이터 다양성 확보 및 일반화 성능에 영향을 줄 수 있는 요소로 판단되어 **중복 제거 전/후 모델 성능 비교 실험**을 고려하였습니다.

2.2.4. 클래스 통합 및 라벨링 전략

| No | 클래스 쌍(Class pair) | |
|---|---|---|
| 1 | K5_3세대_하이브리드_2020_2022 | K5_하이브리드_3세대_2020_2023 |
| 2 | 디_올뉴니로_2022_2025 | 디_올_뉴_니로_2022_2025 |
| 3 | 718_박스터_2017_2024 | 박스터_718_2017_2024 |
| 4 | RAV4_2016_2018 | 라브4_4세대_2013_2018 |
| 5 | RAV4_5세대_2019_2024 | 라브4_5세대_2019_2024 |

주최 측에서는 분류 난이도 완화를 목적으로 시각적으로 유사한 5쌍의 클래스에 대해 복수 정답을 허용한다고 공지하였습니다. 이에 따라 해당 클래스 쌍을 통합하여 그에 따른 모델 성능 변화를 실험적으로 검증하고자 하였습니다.

## 2.3. Data preprocessing

2.3.1. 클래스 통합 실험

주최 측이 제시한 시각적으로 유사한 클래스 쌍에 대해 클래스를 통합하여 분류 문제를 단순화하는 전략을 실험하였습니다.

| Model | 클래스 통합 | Val LogLoss | Leaderboard |
|---|---|---|---|
| ResNet18 | X | **0.2097** | **0.3687458099** |
| ResNet18 | O | 0.2380 | 0.4100389977 |

그 결과, 클래스를 통합한 경우 오히려 성능이 하락하였습니다. 이는 클래스 간 시각적 유사성은 존재하지만 오히려 정보 손실이 성능 저하로 이어질 수 있음을 시사합니다. 따라서 **클래스를 통합하지 않고 원래의 세분화된 레이블 체계를 유지하는 것이 성능 면에서 더 유리하다**는 결론을 도출하였습니다.

2.3.2. 중복 이미지 제거 실험

MD5 해시를 기반으로 총 **1,810장의 중복 이미지**를 식별하고 제거한 후 동일한 모델 구조(ResNet18)를 기반으로 성능 비교 실험을 수행하였습니다.

| Model | 중복 이미지 제거 | Val LogLoss | Leaderboard |
|---|---|---|---|
| ResNet18 | X | 0.2097 | 0.3687458099 |
| ResNet18 | O | 0.2387 | 0.3877283049 |

중복 이미지를 제거한 경우, Val LogLoss와 리더보드 점수 모두 하락하는 경향을 보였습니다. 이는 원본 데이터셋에서는 클래스 간 분포가 비교적 균형적이었으나 중복 이미지를 제거함으로써 일부 클래스의 샘플 수가 급감하여 불균형이 유발된 것으로 해석됩니다.

따라서 **중복 이미지 제거 시 오히려 분포 왜곡으로 인한 성능 저하가 발생할 수 있어 원본 데이터셋을 그대로 유지하는 것이 더 적절한 접근**이라는 결론을 내렸습니다.

#### 2.3.3. 데이터 클리닝

전체 학습 데이터 중 중고차 이미지와 무관한 이미지 5장을 제거하였습니다. 제거된 이미지들은 명백히 라벨링 오류로 판단되는 사례였으며 노이즈 제거의 일환으로 처리하였습니다.

### 2.4. 모델 선정 및 분석

#### 2.4.1. Torchvision 기반 모델 실험

기초 모델 탐색을 위해 PyTorch의 `torchvision` 라이브러리에 포함된 다양한 백본(backbone)을 실험하였습니다. 실험 조건은 입력 이미지 크기 224, 배치 크기 64, 학습률 0.0001로 고정하였고, 손실 함수는 CrossEntropy, 옵티마이저는 Adam을 사용하였습니다.

| Model | IMG_SIZE | BATCH_SIZE | LEARNING_RATE | Loss | Optimizer | Val LogLoss |
|---|---|---|---|---|---|---|
| resnet18 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.20974 |
| resnet50 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.26089 |
| convnext_tiny | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.22337 |
| swinT_tiny | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.21221 |
| efficient-b0 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.22065 |
| efficient-b1 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.20961 |
| **efficient-b2** | **224** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.19627** |
| **efficient-b2** | **288** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1750** |
| efficient-b3 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.20864 |
| efficient-b4 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.26609 |
| **densenet121** | **224** | 64 | **0.0001** | **CrossEntropy** | **Adam** | **0.18704** |
| **densenet121** | **256** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1627** |
| **densenet169** | **224** | 64 | **0.0001** | **CrossEntropy** | **Adam** | **0.16091** |
| **densenet169** | **256** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1626** |

해당 실험을 통해, param 수 약 8M~14M의 모델이 안정적인 성능을 보임을 확인하였습니다. 이에 따라 이후 실험에서도 이와 유사한 파라미터 범위의 모델을 중심으로 탐색하였습니다.

#### 2.4.2. Timm 기반 모델 실험

모델의 다양성과 성능 개선 가능성을 탐색하기 위해 `timm` 라이브러리의 다양한 구조를 실험하였습니다. 특히, parameter 수가 8M~15M 수준인 모델들을 우선적으로 선택하여 실험을 진행하였습니다.

| Model | IMG_SIZE | BATCH_SIZE | LEARNING_RATE | Loss | Optimizer | Val LogLoss |
|---|---|---|---|---|---|---|
| tf_efficientnetv2_b3 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.2672 |
| **xcit_tiny_24_p8_384** | **224** | **32** | **0.0001** | **CrossEntropy** | **Adam** | **0.1878** |
| **tiny_vit_11m_224** | **224** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1720** |
| mobilevitv2_175 | 224 | 32 | 0.0001 | CrossEntropy | Adam | 0.2488 |
| regnetz_c16 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.2737 |
| **hgnet_tiny** | **224** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1899** |
| hgnet_tiny | 288 | 64 | 0.0001 | CrossEntropy | Adam | 0.2274 |
| efficientformerv2_s2 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.2607 |
| coatnext_nano_rw_224 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.3062 |
| fastvit_sa12 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.2697 |
| **coat_mini** | **224** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1963** |
| sebotnet33ts_256 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.2719 |
| convnextv2_pico | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.6484 |
| sehalonet33ts | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.2218 |

| Model | IMG_SIZE | BATCH_SIZE | LEARNING_RATE | Loss | Optimizer | Val LogLoss |
|---|---|---|---|---|---|---|
| cait_xxs24_384 | 384 | 16 | 0.0001 | CrossEntropy | Adam | 0.2734 |
| rexnet_150 | 224 | 64 | 0.0001 | CrossEntropy | Adam | 0.2272 |

#### 2.4.3. 중간 규모 모델 성능 비교 및 최종 후보 선정

모든 실험 결과를 종합하여 Val LogLoss 기준 상위 5개 모델을 선정하였습니다.

| Model | IMG_SIZE | BATCH_SIZE | LEARNING_RATE | LOSS | Optimizer | Val LogLoss | Lea |
|---|---|---|---|---|---|---|---|
| **efficient-b2** | **288** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1750** | **0.3** |
| **densenet121** | **256** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1627** | **0.3** |
| densenet169 | 224 | 32 | 0.0001 | CrossEntropy | Adam | 0.16091 | 0.3 |
| xcit_tiny_24_p8_384 | 224 | 32 | 0.0001 | CrossEntropy | Adam | 0.1878 | 0.3 |
| **tiny_vit_11m_224** | **224** | **64** | **0.0001** | **CrossEntropy** | **Adam** | **0.1720** | **0.3** |

Val LogLoss 측면에서는 DenseNet169가 가장 낮았으나, 실제 리더보드 점수 기준에서는 `tiny_vit_11m_224`이 가장 우수하였습니다. 이에 따라 **최종 모델로는 tiny_vit을 선택**하였습니다.

#### 2.4.4. 대규모 모델 실험

실험 도중 당시 리더보드 1위를 기록한 팀이 ConvNeXt-Base (약 88M parameter) 모델 사용한 것으로 확인되었습니다. 이는 지금까지 실험한 모델들과의 성능 차이가 파라미터 규모에 기인할 수 있다는 점을 시사하였습니다. 이에 따라 **약 70M 이상 파라미터를 갖는 대규모 모델**들을 중심으로 후속 실험을 추가로 진행하였습니다.

| Model | PARAMS | IMG_SIZE | BATCH_SIZE | LEARNING_RATE | LOSS |
|---|---|---|---|---|---|
| **convnext_base** | **88.59** | **224** | **32** | **0.0001** | **Cros** |
| convnextv2_base.fcmae_ft_in22k_in1k | 88.72 | 224 | 16 | 0.0001 | Cros |
| **maxxvitv2_rmlp_base_rw_224.sw_in12k_ft_in1k** | **116.09** | **224** | **16** | **0.0001** | **Cros** |
| **coatnet_2_rw_224.sw_in12k_ft_in1k** | **73.87** | **224** | **16** | **0.0001** | **Cros** |
| beitv2_base_patch16_224.in1k_ft_in22k_in1k | 86.53 | 224 | 32 | 0.0001 | Cros |
| swinv2_base_window12to16_192to256.ms_in22k_ft_in1k | 87.92 | 256 | 16 | 0.0001 | Cros |

ConvNeXt-Base를 포함한 대규모 모델들은 기존 모델에 비해 확연한 성능 향상을 보였습니다. 특히 ConvNeXt-Base는 **Val LogLoss 0.1123**으로, 지금까지 실험한 모든 모델 중 가장 낮은 손실 값을 기록하였습니다.

### 2.5. Data Augmentation

초기에는 단일 증강 기법들을 조합하여 실험하였으나, 대부분의 경우 모델 성능이 오히려 하락하였습니다. 따라서 최종적으로는 대표적인 자동 증강 기법인 **AutoAugment**, **RandAugment**와 함께 **CutMix**, **MixUp** 등의 혼합 방식에 대하여 실험을 진행하였습니다. 모든 실험은 `tiny_vit_11m_224` 모델을 기준으로 동일한 조건 하에 수행하였습니다.

#### 2.5.1. AutoAugment / RandAugment

AutoAugment는 사전에 특정 데이터셋을 기반으로 설계된 고정된 증강 정책(policy)을 사용하는 방식으로, 라벨 수가 적은 상황에서도 일반화 성능 향상에 효과적인 것으로 알려져 있습니다.

RandAugment는 별도의 정책 검색 과정 없이 증강 강도(m)와 연산 수(n)라는 두 개의 하이퍼파라미터만으로 증강 조합을 구성하는 방식입니다. 실험에서는 보다 단순하지만 유연한 증강 조합을 통해 모델의 성능 개선 가능성을 평가하고자 하였습니다.

| | No Aug | AutoAugment | **RandAugment** |
|---|---|---|---|
| Val Log Loss | 0.1720 | 0.1046 | **0.0999** |
| Leaderboard | 0.3155701075 | 0.2237695701 | **0.1998684925** |

실험 결과, **RandAugment**가 가장 낮은 손실 값과 리더보드 점수를 기록하였으며 Log Loss 기준으로 기존 대비 약 0.11%p 감소하였습니다.

AutoAugment의 경우에도 성능이 향상되었으나 RandAugment보다 낮은 성능을 보였습니다. 이는 AutoAugment가 사전 정의된 증강 정책을 따르기 때문에 **도메인 특화된 중고차 이미지에는 적합하지 않은 증강 조합이 포함되었을 가능성** 때문으로 해석하였습니다.

#### 2.5.2. CutMix / MixUp

추가로, 두 개의 입력 이미지를 혼합하는 대표적인 증강 기법인 **CutMix** 및 **MixUp**을 실험하였습니다.

CutMix는 두 이미지를 공간적으로 섞어 일부 영역만 보여줌으로써 모델이 부분 정보로도 전체 클래스를 판단하도록 유도하는 방식입니다.

MixUp은 두 이미지를 선형 결합하여 새로운 샘플을 생성하며, 클래스 간 경계가 모호할 때 모델이 극단적인 학습 패턴에 빠지는 것을 방지하기 위해 사용했습니다.

| | No Aug | CutMix | MixUp |
|---|---|---|---|
| Val Log Loss | 0.1720 | 0.1009 | 0.1126 |
| Leaderboard | 0.3155701075 | 0.2240897204 | 0.2281253673 |

CutMix가 가장 우수한 성능을 보였으며 이는 중고차 이미지와 같이 **구조적으로 명확한 객체 중심 데이터**에서는 MixUp 방식이 시각적 정보 왜곡을 초래할 수 있기 때문으로 판단하였습니다. 실제로 MixUp 적용 시에는 차량의 형태가 모호해지며 모델이 학습에 혼란을 겪는 것으로 보였습니다.

#### 2.5.3. 증강 조합

최종적으로, 가장 우수한 성능을 보인 두 증강 기법인 **RandAugment와 CutMix를 조합하여 복합 증강 실험**을 수행하였습니다.

| | No Aug | RandAugment | CutMix | RandAugment+CutMix |
|---|---|---|---|---|
| Val Log Loss | 0.1720 | 0.0999 | 0.1009 | 0.1016 |
| Leaderboard | 0.3155701075 | 0.1998684925 | 0.2240897204 | 0.2018852602 |

복합 증강 실험에서는 성능이 소폭 하락하는 경향을 보였습니다. 이는 RandAugment로 충분한 이미지 변형이 이루어진 상태에서 추가적인 CutMix가 오히려 **과도한 정보 왜곡을 유발**했을 가능성 때문으로 해석했습니다.

따라서 최종 모델 학습 시에는 RandAugment 단독 증강 기법을 채택하였습니다.

### 2.6. Loss Function

모델의 학습 안정성과 일반화 성능 향상을 위해 총 세 가지 손실 함수에 대한 비교 실험을 수행하였습니다.

기본 베이스라인은 **Cross Entropy Loss**이며 추가로 적용한 손실 함수는 **Confidence Penalty Loss**와 **Poly1 Cross Entropy Loss** 입니다.

**Confidence Penalty Loss**는 모델이 특정 클래스에 과도한 확신을 가지지 않도록 제약을 가하는 방식으로, 불확실성이 존재하는 샘플에 대해 보다 유연하게 예측하기 위해 사용했습니다.

**Poly1 Cross Entropy Loss**는 기존의 Cross Entropy에 작은 다항식 항(polynomial term)을 추가한 형태로, 클래스 간 미세한 차이를 더 정밀하게 인식하기 위해 사용했습니다.

#### 2.6.1. 증강 미적용 상태에서의 손실 함수 비교

가장 공정한 비교를 위해 데이터 증강을 적용하지 않은 상태에서 실험을 먼저 수행하였으며, `tiny_vit_11m_224` 모델을 사용하였습니다.

| | Cross Entropy | Confidence Penalty | Poly1 |
|---|---|---|---|
| Augmentation | X | X | X |
| Val Log Loss | 0.1720 | 0.1007 | 0.1403 |

실험 결과, **Confidence Penalty Loss가 가장 낮은 Val LogLoss를 기록**하였으며, 이는 모델이 확신을 줄이면서 보다 일반화된 예측을 수행했기 때문으로 판단됩니다.

#### 2.6.2. 증강 적용 상태에서의 손실 함수 비교

이후 가장 성능이 우수했던 **RandAugment 기반 증강을 적용한 모델**에 대해서도 동일한 손실 함수 실험을 진행하였습니다.

| | Cross Entropy | Confidence Penalty |
|---|---|---|
| Augmentation | O | O |
| Val Log Loss | **0.0999** | 0.0984 |
| Leaderboard | **0.1998684925** | 0.2027363352 |

Val LogLoss 기준으로는 Confidence Penalty가 소폭 더 낮았으나, **리더보드 점수에서는 Cross Entropy Loss가 더 우수한 성능을 보였습니다.** 이는 Confidence Penalty가 검증 셋의 불확실한 샘플에 대해서는 더 유연한 대응을 가능하게 했지만 최종 리더보드 기준에서는 다소 **보수적인 예측을 한 것으로 추측했습니다.**

## 2.7. 모델 평가 및 개선

| Model | HEM | Val LogLoss | Leaderboard |
|---|---|---|---|
| tiny_vit_11m_224 | X | 0.0999 | 0.1998684925 |
| tiny_vit_11m_224 | O | 0.0850 | 0.2081149674 |

손실 값이 큰 어려운 샘플들을 선별하여 별도의 데이터셋을 구성하는 Hard Example Mining(HEM) 기법을 적용해 실험하였으나 성능 개선에는 실패하였습니다.

| Model | Classifier | Val LogLoss | Leaderboard |
|---|---|---|---|
| tiny_vit_11m_224 | Linear | 0.0999 | 0.1998684925 |
| convnext_base | Linear | 0.1037 | 0.2315700698 |
| **tiny_vit_11m_224** | **Cosine Similiarity** | **0.0959** | **0.1975553104** |
| convnext_base | Cosine Similiarity | 0.0950 | 0.2207785735 |

기존의 선형 분류기를 코사인 유사도 기반 분류기로 교체한 결과, 전반적으로 소폭 성능 향상을 확인하였습니다. 이에 따라 최종 분류기로 **Cosine Similarity Classifier를 채택**하였습니다.

| Model | Image Size | Augmentation | Batch Size | Val LogLoss | Leaderboard | 비고 |
|---|---|---|---|---|---|---|
| tiny_vit_11m_224 | 224 | RandAug | 64 | **0.0999** | **0.1998684925** | |
| tiny_vit_21m_224 | 224 | RandAug | 64 | 0.0995 | | 62 에폭 시점에서 11m 모델 성능 추월 |
| tiny_vit_21m_384 | 384 | RandAug | 32 | 0.0783 | 0.1690110131 | |
| tiny_vit_21m_512 | 512 | RandAug | 32 | | | 에폭당 약 20분 소요로 시간 부족 예상 |

파라미터 수 및 입력 이미지 크기에 따른 실험을 수행한 결과, 동일한 tiny_vit 아키텍처 내에서 약 10M 파라미터가 더 많은 `tiny_vit_21m_224` 모델이 중반부 에폭부터 `11m_224` 모델 대비 성능 우위를 보였습니다.

이후 입력 크기를 384로 확장하여 실험한 결과, LogLoss가 약 0.03%p 추가 감소하며 성능이 개선되었습니다.

다만, 입력 크기를 512로 늘린 모델은 학습 시간이 에폭당 약 20분으로 긴 관계로 충분한 실험 시간을 확보하지 못했습니다.

## 2.8. Ensemble

다양한 모델의 예측 결과를 통합하는 방법으로, 각 모델의 예측 확류을 단순 평균하여 최종 예측을 산출하는 Soft Voting, 모델별 성능에 가중치를 부여하여 예측 확륭르 평균하는 Weighted Soft Voting, 모든 모델의 예측 확률을 곱한 뒤 루트를 취하여 모델 간 동의가 높은 에측에 더 큰 비중을 부여하는 Geometric Mean 세 가지 앙상블 기법을 비교 실험하였습니다.

| Model | Method | Leaderboard |
|---|---|---|
| tinyvit+densenet169+efficientb2 | soft voting | 0.1896440451 |
| tinyvit+densenet169+efficientb2 | Weighted Soft Voting | 0.189186104 |
| tinyvit+densenet169+efficientb2 | Geometric Mean | **0.1761756159** |

기존에 RandAugment가 적용된 상위 3개 모델을 대상으로 앙상블을 수행한 결과, **Geometric Mean 기법에서 가장 우수한 성능 향상**을 확인하였습니다.

| Model | Method | Leaderboard |
|---|---|---|
| convnext_base+densenet169+efficientnet+maxxvitv2+tiny_vit | Geometric Mean | **0.1540862313** |

이후 경량 모델뿐만 아니라, convnext_base, maxxvitv2_rmlp_base_rw_224 등 파라미터 수가 더 큰 모델을 추가하여 총 5개 모델로 Geometric Mean 앙상블을 수행한 결과, **최종 Log Loss 0.1541**을 달성하며 성능을 크게 향상시켰습니다.

## 5. 자체 평가 의견

- 잘한 점들
  - MMDetection3 라이브러리 설치로 최신 모델 적용을 통한 성능 개선
  - 모델 적용 결과에 대한 분석을 통해 추가 개선점을 발견하고, 이를 적용한 실험
  - 협업을 위해 매일 해야할 것과 진행상황 공유
  - 결과, 서버 사용 등 필요한 내용들을 노션에 기록하면서 체계적인 정리와 진행 사항 관리
- 시도했으나 잘 되지 않았던 것들
  - MMDetection3 라이브러리 적용이 늦어지면서 최신 모델을 더 적용하지 못함→ 다음 프로젝트에서는 매일 Slack 체크하면서 유용한 정보를 가져오는 것도 추가
  - 프로젝트 초기에 모델을 구현하는 과정에 많은 시간을 할애하면서 초반 실험 부족→ 학습과 프로젝트 사이에서 적절한 균형을 찾을 수 있는 접근 전략 수립
  - 후반부 실험 진행에 대한 모든 내용이 노션에 기록되진 않음→ 주기적으로 보고서에 업데이트해서 기록 누락 방지 및 방향성 확인
- 아쉬웠던 점들
  - 다음 프로젝트(Data-centric)에 대한 사전 조사 및 계획
  - 향후 최종 프로젝트 및 커리어를 위한 관심 분야를 탐구
  - 역할 분담 정의 및 분배를 통해 빠른 프로젝트 온보딩 (초기 서버 세팅 / 베이스라인 분석 및 프로젝트 구조화 / 노션 세팅 / 필요 기능 구현 - Streamlit 등)
- 프로젝트를 통해 배운 점 또는 시사점
  - MMDetection3 사용하면 최신 모델까지 적용하고, config 수정만으로 OD모델의 각 구성 요소(backbone, neck, head 등)들에 대한 세부적인 조정 가능
  - 모델 적용 결과에 대한 분석을 통해 추가적인 개선 사항 도출
  - Detectron 라이브러리를 사용하며 라이브러리의 사용법을 익힐 수 있는 계기가 됨