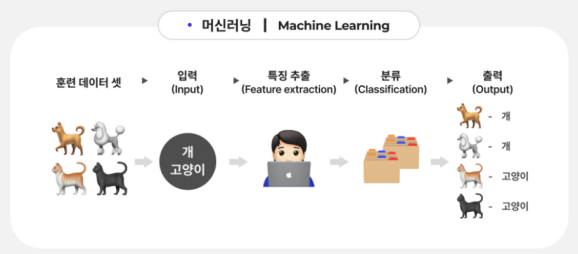

# SEMES - 볼트 이상 진단 AI 모델 개발

## 이미지 분류 AI 모델 (정상 / 파손 / 유실 분류용)

## 머신러닝 vs 딥러닝

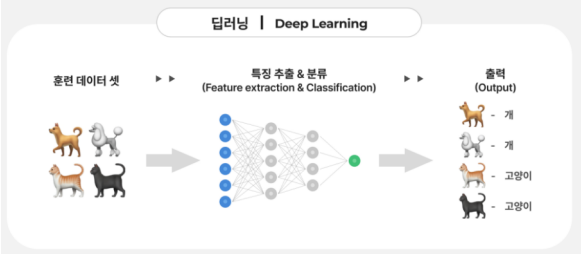
### 머신러닝




- **머신러닝**은 주어진 데이터를 인간이 먼저 처리

1. 이미지 데이터의 클래스를 정의하여 데이터 셋을 만들어 준다
2. 사람이 먼저 컴퓨터에 특정 패턴을 추출하는 방법을 지시
3. 그 이후 컴퓨터가 스스로 데이터의 특징을 분석하고 축적
4. 이미지를 특징에 맞게 평가하여 분류된 결과를 얻는다

### 딥러닝




- **딥러닝**은 머신러닝에서 사람이 하던 패턴 추출 작업이 생략

1. 이미지 데이터의 클래스를 정의하여 데이터 셋을 만들어 준다
2. 컴퓨터가 스스로 데이터를 기반으로 학습할 수 있도록 정해진 신경망을 컴퓨터에게 줌
3. 어린아이가 학습하는 것처럼 경험 중심으로 학습을 수행
4. 이미지를 인공 신경망에 평가하여 분류된 결과를 얻는다

⇒ **딥러닝 방식** 선정

⇒ 정답지가 있는 학습 방법인 **지도 학습**과 정답지가 없는 학습 방법인 **비지도 학습**

⇒ 시간 단축을 하고, 작은 데이터셋으로도 성능 향상을 위해 **지도학습 방식** 선정

- 모델 선정 : Pytorch / Tensorflow에서 지원하는 모델 중 선정
  - 모델의 정의 : 여러 가지 방법론의 구현체

## Pytorch

▼ 지원모델

- **AlexNet**
- **ConvNeXt**
- **DenseNet**

## Tensorflow

▼ 지원모델

- EfficientNetB7
- EfficientNetB6
- EfficientNetB5

- **EfficientNet**
- **EfficientNetV2**
- **GoogLeNet**
- **Inception V3**
- **MaxVit**
- **MNASNet**
- **MobileNet V2**
- **MobileNet V3**
- **RegNet**
- **ResNet**
- **ResNeXt**
- **ShuffleNet V2**
- **SqueezeNet**
- **SwinTransformer**
- **VGG**
- **VisionTransformer**
- **Wide ResNet**

▼ 정확도 - Pytorch

| Weight | Acc@1 | Acc@5 | Params | GFLOPS |
|---|---|---|---|---|
| ViT_H_14_Weights.IMAGENET1K_SWAG_E2E_V1 | 88.552 | 98.694 | 633.5M | 1016.72 |
| RegNet_Y_128GF_Weights.IMAGENET1K_SWAG_E2E_V1 | 88.228 | 98.682 | 644.8M | 374.57 |
| ViT_L_16_Weights.IMAGENET1K_SWAG_E2E_V1 | 88.064 | 98.512 | 305.2M | 361.99 |
| RegNet_Y_32GF_Weights.IMAGENET1K_SWAG_E2E_V1 | 86.838 | 98.362 | 145.0M | 94.83 |
| RegNet_Y_128GF_Weights.IMAGENET1K_SWAG_LINEAR_V1 | 86.068 | 97.844 | 644.8M | 127.52 |
| RegNet_Y_16GF_Weights.IMAGENET1K_SWAG_E2E_V1 | 86.012 | 98.054 | 83.6M | 46.73 |
| EfficientNet_V2_L_Weights.IMAGENET1K_V1 | 85.808 | 97.788 | 118.5M | 56.08 |
| ViT_H_14_Weights.IMAGENET1K_SWAG_LINEAR_V1 | 85.708 | 97.73 | 632.0M | 167.29 |
| ViT_B_16_Weights.IMAGENET1K_SWAG_E2E_V1 | 85.304 | 97.65 | 86.9M | 55.48 |
| ViT_L_16_Weights.IMAGENET1K_SWAG_LINEAR_V1 | 85.146 | 97.422 | 304.3M | 61.55 |
| EfficientNet_V2_M_Weights.IMAGENET1K_V1 | 85.112 | 97.156 | 54.1M | 24.58 |
| RegNet_Y_32GF_Weights.IMAGENET1K_SWAG_LINEAR_V1 | 84.622 | 97.48 | 145.0M | 32.28 |
| ConvNeXt_Large_Weights.IMAGENET1K_V1 | 84.414 | 96.976 | 197.8M | 34.36 |
| EfficientNet_V2_S_Weights.IMAGENET1K_V1 | 84.228 | 96.878 | 21.5M | 8.37 |
| EfficientNet_B7_Weights.IMAGENET1K_V1 | 84.122 | 96.908 | 66.3M | 37.75 |
| Swin_V2_B_Weights.IMAGENET1K_V1 | 84.112 | 96.864 | 87.9M | 20.32 |
| ConvNeXt_Base_Weights.IMAGENET1K_V1 | 84.062 | 96.87 | 88.6M | 15.36 |
| EfficientNet_B6_Weights.IMAGENET1K_V1 | 84.008 | 96.916 | 43.0M | 19.07 |
| RegNet_Y_16GF_Weights.IMAGENET1K_SWAG_LINEAR_V1 | 83.976 | 97.244 | 83.6M | 15.91 |

- 정확도가 높은 모델

  ▼ VIT(Vision Transformer)

  - 특징: Transformer 기반 비전 모델, CNN 대신에 self-attention 기술 사용, 이미지 패치를 벡터로 변환한 후 Transformer 구조에 입력

▼ 정확도 - Tensorflow

| Model | Top-1 Accuracy | Top-5 Accuracy | Size |
|---|---|---|---|
| EfficientNetB7 | 84.30% | 97.00% | 256 |
| EfficientNetB6 | 84.00% | 96.80% | 166 |
| EfficientNetB5 | 83.60% | 96.70% | 118 |
| EfficientNetB4 | 82.90% | 96.40% | 75 |
| NASNetLarge | 82.50% | 96.00% | 343 |
| EfficientNetB3 | 81.60% | 95.70% | 48 |
| InceptionResNetV2 | 80.30% | 95.30% | 215 |
| EfficientNetB2 | 80.10% | 94.90% | 36 |
| EfficientNetB1 | 79.10% | 94.40% | 31 |
| Xception | 79.00% | 94.50% | 88 |
| ResNet152V2 | 78.00% | 94.20% | 232 |
| InceptionV3 | 77.90% | 93.70% | 92 |
| DenseNet201 | 77.30% | 93.60% | 80 |
| ResNet101V2 | 77.20% | 93.80% | 171 |
| EfficientNetB0 | 77.10% | 93.30% | 29 |
| ResNet152 | 76.60% | 93.10% | 232 |
| ResNet101 | 76.40% | 92.80% | 171 |
| DenseNet169 | 76.20% | 93.20% | 57 |
| ResNet50V2 | 76.00% | 93.00% | 98 |
| DenseNet121 | 75.00% | 92.30% | 33 |
| ResNet50 | 74.90% | 92.10% | 98 |

- 정확도가 높은 모델

  ▼ EfficientNet

- 장점: 기존 CNN보다 높은 성능을 보이며, input size가 큰 이미지에도 적용 가능, 대규모 데이터셋에서 다른 모델보다 우수한 성능을 보임
- 단점: 많은 데이터와 연산량이 필요, 속도가 느리고 복잡한 구조로 인해 모델 크기가 큼

▼ RegNet(Regularization Network)

- 특징: 네트워크 구조를 효율적으로 구성하는 알고리즘을 개발한 ResNet의 발전형 모델, 모듈(Module) 구조를 조합하여 생성
- 장점: 모듈 구조의 조합으로 인해 모델 크기를 줄이면서도 성능을 유지, 작은 모델에서도 큰 모델과 비슷한 성능을 보임
- 단점: 다른 모델과 마찬가지로 학습 데이터가 적을 때 성능이 떨어짐

▼ EfficientNet

- 특징: 네트워크 깊이, 너비, 해상도를 조정하여 모델의 성능과 효율성을 극대화한 모델
- 장점: 더 적은 연산량과 파라미터로도 높은 성능을 보이며, 다양한 해상도 입력 이미지에 대해 일관된 성능을 보임
- 단점: 다른 모델보다 뛰어난 성능을 보이는 만큼, 모델의 구조가 복잡하고 학습에 시간이 많이 걸림. 특정 문제에 최적화된 모델보다는 일반화 성능이 더 뛰어나기 때문에 특정 문제에 대한 최적화가 필요할 수 있음.

- 특징: 네트워크 깊이, 너비, 해상도를 조정하여 모델의 성능과 효율성을 극대화한 모델
- 장점: 더 적은 연산량과 파라미터로도 높은 성능을 보이며, 다양한 해상도 입력 이미지에 대해 일관된 성능을 보임
- 단점: 다른 모델보다 뛰어난 성능을 보이는 만큼, 모델의 구조가 복잡하고 학습에 시간이 많이 걸림. 특정 문제에 최적화된 모델보다는 일반화 성능이 더 뛰어나기 때문에 특정 문제에 대한 최적화가 필요할 수 있음.

▼ NASNet

- 특징: 자동으로 최적의 딥러닝 모델을 탐색하는 Neural Architecture Search 기술을 이용한 모델, 높은 성능과 일반화 능력을 가짐
- 장점: 자동으로 최적의 구조를 탐색하여 최적화된 모델을 생성, 대규모 데이터셋에서 높은 성능을 보임
- 단점: 모델의 구조가 복잡하여 속도가 느릴 수 있음, 모델 설계가 어려워 인간의 도움이 필요함

▼ InceptionResNet

- 특징: Inception 모듈과 ResNet 모듈을 결합하여 생성한 모델, 더 나은 성능과 효율성을 제공
- 장점: Inception 모듈의 다양한 필터 크기와 ResNet 모듈의 스킵 연결을 활용하여 성능을 향상, 작은 모델에서도 큰 모델과 비슷한 성능을 보임
- 단점: 모델의 구조가 복잡하여 속도가 느릴 수 있음

▼ Xception

- 특징: Inception 모델의 구조를 개선하여 생성한 모델, Depthwise Separable Convolution을 이용하여 더 효율적인 모델을 만듦
- 장점: Inception 모델보다 더 높은 성능과 효율성을 제공하며, 작은 모델에서도 큰 모델과 비슷한 성능을 보임
- 단점: 모델의 구조가 복잡하여 속도가 느릴 수 있음, 일부 문제에서 다른 모델보다 성능이 떨어질 수 있음.

▼ ResNet

- 특징: 신경망의 깊이가 증가하면서 발생하는 그레디언트 소실 문제(gradient vanishing problem)를 해결하기 위해 스킵 연결(skip connection)을 사용한 모델
- 장점: 깊은 모델에서도 높은 정확도를 보임, 스킵 연결을 통해 기울기가 손실되지 않고 정보가 흐를 수 있도록 함
- 단점: 깊은 모델에서는 성능이 좋지만, 모델의 깊이가 적을 때는 다른 모델과 성능 차이가 크지 않을 수 있음

- 추천 모델
  1. RegNet

- Pytorch만 지원
- 페이스북 AI 연구팀에서 발표한 모델
- 모델 구조를 효율적으로 설계하고, regularization 기법을 사용하여 overfitting을 방지하고 성능을 개선
- 모델의 구조가 단순하여 파라미터 수가 적어 상대적으로 모델의 크기가 작고, 학습 시간이 짧음

2. EfficientNet
   - Pytorch와 Tensorflow 모두 지원
   - google에서 발표한 모델
   - 31~256MB까지 다양한 크기에서 선택 가능(크기와 정확도는 비례)
   - 파라미터 수와 연산량이 효율적으로 조정되어 있어 더 높은 정확도와 더 적은 연산량
   - 네트워크의 깊이, 너비, 해상도(scale)를 동시에 최적화하여 높은 정확도
   - 모델의 특징 추출 부분인 Backbone Network(BN)과 분류기(Classifier)를 따로 학습할 수 있도록 모듈화

◦ 상대적 비교

| **Model** | **Frame work** | **Model Size** | **Learn time** |
|---|---|---|---|
| **RegNet** | Pytorch | small | short |
| **EfficientNet** | Pytorch, Tensorflow | large | long |

- 결론
  - 사용 프레임 워크 : **Pytorch**
    - Detection model을 Yolo.v5를 사용하는 상황
    - 유지/보수를 위해 같은 프레임워크 사용하는 것을 고려
  - 모델의 크기 및 신경망
    - 1차 분류 목표 (정상, 파손/유실) ⇒ 2개 클래스로 분류
    - 최종 분류 목표 (정상 / 유실 / 파손 / 풀림) ⇒ 4개 클래스로 분류
      - 분류할 클래스가 매우 적기 때문에 용량이 크고 깊은 신경망을 사용하지 않아도 될 것으로 판단됨
        - 위 자료의 정확도는 클래스가 997개의 데이터셋으로 학습한 결과의 정확도
      - 모델의 깊이와 크기가 커질수록 성능이 증가하는 것은 맞지만 분류할 클래스가 많지 않아 크기와 깊이가 작은 모델도 같은 성능을 보일 수 있음
        - 같은 정확도라면, 크기와 깊이가 작으면 속도가 빨라지는 장점을 살리면 좋음
  - 정확도와 시간
    - 짧은 기간 내에 높은 정확도를 목표
      - 학습 시간을 줄이는 것이 도움이 될 것으로 판단
      - 상대적으로 작은 데이터셋으로도 높은 정확도를 내는 것이 유리