ResNet을 이용한 전이 학습 프로젝트

202004030 김준호

목차

Ⅰ. 서론

1. 개요

대한민국 육군은 과학화 시스템의 도입으로 최전방 철책에서 초병 근무 인원을 줄이고 각종 감시 장치와 보안 장치를 통해 적을 감시하고 군사분계선을 넘어 남쪽으로 침입하는 적을 막고 있다. 3년간의 근무 동안 이러한 시스템 안에서 다양한 문제점과 한계점을 느꼈고 특히 감시에 있어서 감시에 대한 범위가 넓어졌지만, 넓어진 만큼 사람의 눈도 몇 초마다 주기적으로 돌아가는 수많은 카메라와 이를 보여주는 2대 이상의 모니터를 통해 최소 인원으로 감시하는 문제점이 있었고 인원을 추가하거나 카메라 감시 구역 조정하는 등 해결하려 했지만, 감시의 공백은 여전히 한계점으로 남아있었다. 또한 특이 사항 발생 시 감시자와 중개자 그리고 지휘관의 서로 다른 판단이 통일된 작전 판단으로 이어지지 못하는 어려움으로 작용했다. 따라서 이번 인공지능 프로젝트를 통해 이러한 감시 시스템에 도움을 줄 수 있게 진행해 보고자 한다. 이미지 분류 모델에 기반인 CNN 구조에서 발전된 형태인 ResNet을 이용하여 인공지능을 설계하며 감시 체계에 있어 가장 중요한 사람에 대한 감시, 그리고 이와 혼동을 주는 동물과 그 외 사물들을 구분할 수 있는 것이 현재 군의 감시 체계에서는 중요한 부분이기에 최종적으로 사람・동물・사물 이미지를 분류하는 ResNet 모델로 프로젝트 방향을 정하였다. 또한 PyToch의 사전 학습된 ResNet을 가져와 전이 학습으로 진행, 프로젝트 목적에 맞는 형태로 파인 튜닝 하여 인공지능 구축 시간을 줄이고 성능 목표를 달성할 수 있게 계획하였다. 보고서의 본론은 프로젝트의 진행 과정과 그 결과를 보여주고 있다.

2. 목표 및 기대효과

PyTorch에서 제공하는 ResNet 18의 경우 공식 문서상 정확도(상위 1등) 69.8%를[1] 보이고 있고 실제 감시 체계에 적용하기에는 예측 신뢰성이 부족하다. 따라서 프로젝트 성능 목표는 정확도 90% 이상 그리고 F1-SCORE 0.9 이상을 목표로 설정하였다. 군과 같은 특수 목적 기관에서는 정보의 신뢰성이 중요하다. 만약 모델의 예측 정확도가 신뢰할 수 있는 기준보다 낮다면 모델의 예측이 작전 실패로 이어지기에 사용할 수 없게 된다. 만약 90% 이상의 정확도를 보인다면 도입의 가능성은 커지고 10%의 오차가 있더라도 사람에 대한 예측 재현율이 높다면 적어도 작전 실패에는 지장이 없을 것이다. 프로젝트는 이러한 점을 고려해 정확도 및 정밀도와 재현율의 조화 평균인 F1-SCORE에 대한 평가를 통해 목표 성능 이상으로 모델이 구현될 수 있게 진행하였다. 프로젝트와 같은 인공지능 모델이 실제 군 감시 체계에 도입된다면 직접 감시하며 생기는 공백 문제를 각종 센서를 통해 감지한 특이 사항에 대한 모델의 정확한 예측으로 빠른 작전 상황 인지가 가능하고 직접 검증이 함께 작용하여 판단의 도움을 줌과 동시에 혼란을 최소화하여 지휘관의 작전 하달의 신뢰성을 높이고 성공적인 작전 수행으로 이어지는 효과를 제공할 수 있다. 또한 프로젝트 목적이 세부적인 분류가 아닌 3개의 큰 범주로 분류하기 때문에 목표 달성에 대한 기대효과 또한 매우 높다.

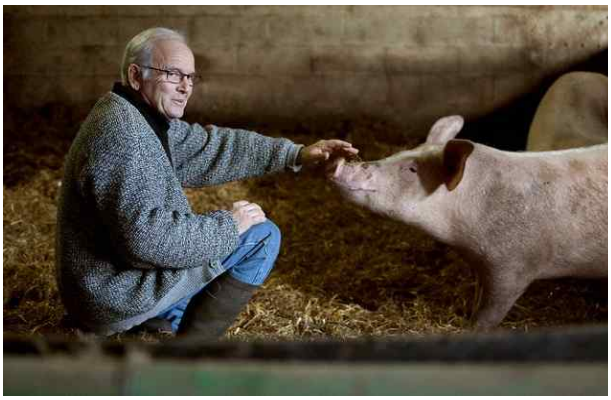
---

1) PyTorch, resnet18 - Torchvision main document,
https://docs.pytorch.org/vision/stable/models/generated/torchvision.models.resnet18.html

Ⅱ. 본론

1. 데이터 가공 및 준비

프로젝트 모델을 학습하기 위해서는 이미지 데이터를 준비해야 한다. 사람, 동물, 사물 이미지가 많이 필요하면서도 다양한 패턴이 중요하다고 판단했다. PyTorch에서 제공하는 ResNet의 경우 ImageNet-1K를 사용하여 학습한 모델이고[2] 기존의 모델이 학습한 데이터를 그대로 가져와 전이 학습하는 것은 성능 향상에 큰 도움을 주지 못할 것으로 생각했다. 따라서 다른 데이터 셋을 이용하여 데이터를 준비했으며 최종적으로 구글에서 제공하는 Open Images Dataset V7을 이용하였다. Open Images Dataset V7은 V1부터 V7을 거치면서 총 66,400,000개의 이미지 데이터와 5,827개의 클래스로 이루어져 있고 이미지 분류 모델, 물체 감지 모델 등 다양한 신경망 모델에서 이미지 데이터를 사용하여 학습할 때 가로, 세로 500px 이상의 고해상도 이미지를 제공하고 있다.[3] 다만 ResNet의 경우 224X224px 의 이미지를 입력 데이터로 사용하기 때문에, 프로젝트 모델이 Open images의 데이터를 사이즈를 줄이지 않고 사용하기에는 적합하지 않다. 또한 이미지의 형태가 <그림 1>과 같이 하나의 클래스에 대한 사진만 있는 것이 아닌 다양한 클래스가 복합적으로 섞여 있는 형태의 이미지가 다수 존재했다. Open Images Dataset은 객체 탐지 모델의 학습을 고려해 이러한 형태의 이미지를 많이 보유하고 있다. (사실상 데이터의 절반 이상이 이러한 형태이다.)



<그림 1>

이러한 이미지에서 원하는 클래스 객체만을 추출하는 방법이 필요했고 detections.csv 파일에 접근해 이를 해결하였다. 그 전에 먼저 데이터를 가져오는 과정을 설명하려 한다. 앞으로 본론에서 작성할 프로젝트 소스 코드 일부는 프로젝트 설명을 위해 작성되었으며 코드의 자세한 동작 및 설명은 원본 소스 코드에 주석으로 작성되어 있다.

---

2) 위의 링크

3) googleapis.com, Open Images Dataset V7 and Extensions - Googleapis.com, https://storage.googleapis.com/openimages/web/index.html

데이터는 Open Images에서 설명하는 다운로드 방식 중 FiftyOne 라이브러리를 이용하여 다운로드를 진행하였다. 프로젝트의 data_find.py에 작성되어 있고 사람, 동물, 사물의 이미지 데이터를 가져왔다. 프로젝트의 목적에 맞게 모델 실사용 환경을 대한민국으로 설정했고 이에 따라 일부 데이터는 대한민국 환경에 맞게 클래스를 선정하여 가져왔다. 클래스에 대한 분류 및 이름은 Open Images에서 제공하고 있고 소스 코드를 통해 어떤 클래스의 데이터를 가져왔는지 확인할 수 있다. 예를 들어 대한민국에는 야생 원숭이는 존재하지 않기 때문에 다운로드 이미지 클래스에 포함하지 않았으며 사물 또한 감시 체계에서 위험한 사물을 위주로 구성하였다. 또한 모델이 사람, 동물, 사물의 특징 구분에 대한 학습이 잘 이루어지도록 클래스를 선정하였다. 이미지는 약 60,000개를 최종 데이터로 확보할 수 있도록 총 62,000개를 가져왔다.

<그림 2>

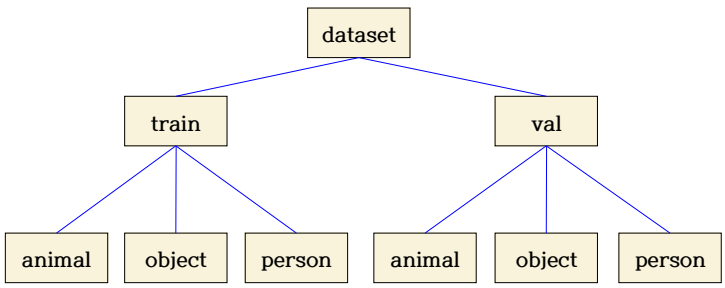
다운로드한 이미지 데이터는 detections.csv 파일에 객체 클래스 ID와 바운딩 박스의 위치 좌표가 명시되어 있다. <그림 2>와 같이 이미지에 존재하는 모든 객체는 csv 파일을 통해 라벨링이 되어 있다. 또한 다운로드한 데이터 셋에는 클래스 이름과 ID에 대한 csv 파일이 있어 detections.csv에서 원하는 클래스에 대한 이미지 바운딩 박스 좌표를 추출할 수 있다. 이러한 원리를 통해 프로젝트에 data_fix.py에서 해당 추출 방식을 적용했다. pandas 라이브러리를 이용하여 detections.csv를 읽어와 추출할 클래스와 다운로드한 이미지만 남기고 이미지 ID를 기준으로 정렬하여 원본 이미지에서 원하는 클래스의 이미지만을 추출할 수 있게 준비하였다. 이후 OpenCV를 통해 pandas로 만든 데이터 프레임에서 각종 정보를 가져와 실제 다운로드한 이미지를 읽어와 해당 부분을 추출하는 방식을 적용하였다. 추출할 이미지의 박스 크기가 너무 작으면 객체 식별이 어렵기 때문에 작다면 추출하지 않게 구현했고 또한 추출한 크기의 비율은 유지하면서 224px로 사이즈로 만들기 위해 224를 추출한 이미지의 가로, 세로로 나눈 최소값을 cv2.resize(cut, (int(cw * ratio), int(ch * ratio)))를 통해 리사이징 했다. 다만 완전한 224x224px이 아닌 일부 비어 있는 공간이 만들어지기 때문에 tuple(np.median(xy, axis=0).astype(int))을 통해 이미지 각 꼭짓점 좌표의 색깔을 가져와 평

균을 구한 색깔을 비어 있는 공간의 배경색으로 채우면서 리사이징한 이미지는 224px 안에서 np.random.randint를 통해 랜덤 위치에 배치하여 최종적으로 모델에 사용될 데이터가 가공되게 된다. <그림 3>은 <그림 1>을 가공하여 돼지만 추출한 예시이다.

<그림 3>

<그림 3>과 같이 데이터를 가공한 이유는 모델이 원본 이미지의 비율을 (정보 왜곡 없음) 학습하고 224px로 리사이징하면서 생기는 비어 있는 공간을 모든 데이터에 대해서 이미지 중앙 배치 및 같은 배경색을 적용한다면 모델이 이러한 반복된 특징을 학습할 우려가 있기에 랜덤하게 이미지를 배치하고 배경색은 최대한 이미지의 배경과 어울리게 유사색(꼭짓점 평균 색)으로 구성하여 데이터의 다양성을 제공하였다. 이 외에도 multiprocessing 라이브러리를 통해 10,000장이 넘는 이미지 가공을 병렬로 처리하여 처리 속도를 높였다.

최종적으로 62,000장을 가공하여 총 41,000장을 리사이징 하였다. 추출하지 못한 데이터는 전부 객체의 박스 크기가 원본 비율에 비해 너무 작은 경우였다. 50,000장으로 구성해 보기 위해 나머지 9,000장을 추가해 볼 계획이었으나 겹치는 이미지에 대한 처리가 어려웠고 일부 클래스의 경우 Open Images에 데이터 수가 자체적으로 적었기에 41,000장으로 마무리하였다. 준비한 데이터는 torchvision 라이브러리의 ImageFolder 포맷으로 사용될 수 있게 <그림 4>와 같이 폴더 구조를 구성했고 8:2로 학습 데이터와 평가 데이터를 나누어 각각의 이미지를 레이블(person, animal, object)에 맞게 위치시켰다.

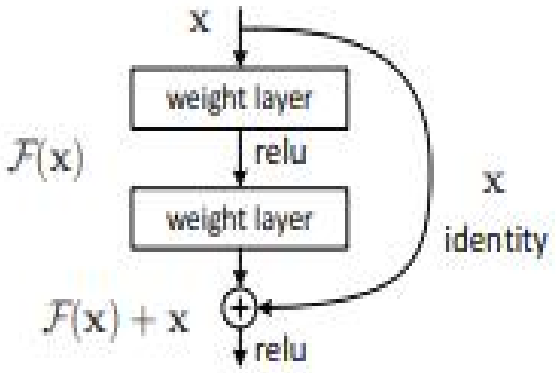


<그림 4>

RandomHorizontalFlip()과 ColorJitter(0.2, 0.2, 0.2, 0.1)를 통해 이미지 증강을 진행했고 Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])를 통해 이미지 정규화를 진행했다. 이미지 정규화 값은 일관된 모델의 학습을 위해 Pytorch에서 제공하는 ResNet 학습에 사용된 정규화 값을 사용했다.[4] 학습 데이터에 대해서만 증강을 적용했고 최종적으로 앞선 과정을 통해 모델을 학습하고 평가할 데이터를 준비하였다.

2. 모델 선정 및 준비

프로젝트에 사용할 모델은 Pytorch의 ResNet 18로 선정하였다. 모델의 층이 깊어질수록 성능이 향상되지만,[5] 단순히 층의 수를 늘리기만 한다면 역전파 시 학습 기울기가 점점 0에 가까워지면서 가중치가 업데이트되지 못하고 학습이 멈추게 되는 기울기 소실의 문제가 발생한다. ResNet은 이러한 기울기 소실의 문제를 해결하고자 잔차 학습이라는 개념을 적용했다.[6] 기존 신경망의 구조인 입력값 x에 대해 출력을 만드는 함수 H(x)를 학습하는 것이 아닌 H(x) - x인 잔차를 학습하여 모델이 잔차를 최소화하는 방향으로 학습하는 것을 목표로 하고 있다. 또한 잔차를 구하는 H(x) - x를 f(x)라고 했을 때 H(x) = f(x) + x가 되고, 이는 기존의 신경망 구조를 유지하면서 각 블록은 복잡한 함수 H(x)가 아닌 간단한 잔차 f(x)에 대해 학습하며 출력은 입력값을 그대로 더하는 shortcut connections이 추가되게 된다. 이는 순전파를 진행하면서 모델은 잔차에 대해서만 학습하며 H(x) = x가 되는 방향으로 학습을 진행하게 되고 잔차가 작아질수록 출력이 입력값과 비슷해지면서 안정적이면서 빠른 학습이 가능하고 shortcut connections를 통해 역전파를 진행하면서 모델의 층이 많아지더라도 기울기가 모든 층에 소실 없이 전달된다. <그림 5>는 이러한 ResNet의 구조를 보여주고 있다.[7]



<그림 5>

4) PyTorch, 앞의 링크
5) Simonyan, Karen, Zisserman, Andrew, 「Very Deep Convolutional Networks for Large-Scale Image Recognition」, 『ICLR 2015』, International Conference on Learning Representations, 2015
6) Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 「Deep Residual Learning for Image Recognition」, 『CVPR 2016』, IEEE Conference on Computer Vision and Pattern Recognition, 2016
7) 위의 논문, p.2

ResNet 18에서 18은 층의 개수를 의미하고 18개 외에도 34, 50, 101등 다양한 ResNet이 있고 층이 늘어날수록 성능이 좋으며 앞서 설명한 구조를 통해 이를 구현하였다. 다만 프로젝트의 모델로 ResNet 18을 선정한 이유는 실제 환경 운영 및 처리 속도에 있다. 성능만을 본다면 ResNet 101 이상의 모델을 채택하는 것이 성능 보장이 되지만 층이 많다는 것은 그만큼 모델이 무겁다는 것이고 입력 후 완전 연결 층을 통해 결과를 도출하기까지 연산량이 많아 처리 속도가 층 적은 모델보다 오래 걸리는 점은 무시할 수 없는 조건이다. 실제 군 감시 체계에 도입한다고 가정해 보았을 때 무거운 모델을 사용하여 안정적으로 운영하기 위해서는 그만큼의 컴퓨팅 자원과 관리 시스템이 필요하고 감시 체계를 운용하는 모든 기지에 대해서 이러한 시스템이 적용되어야 하기에 실현 가능성을 본다면 가벼운 모델을 사용하여 운용하는 것이 좋다고 판단했다. 또한 실제 환경에서는 모델을 사용하기 위해서는 각종 센서를 통해 감지된 상황을 모델에 사용할 수 있는 이미지로 변환하는 과정부터 출력을 보는 과정까지에 시간이 소요되기에 모델의 처리 속도는 작전 수행에 있어 중요한 요소이다. 따라서 프로젝트는 PyTorch에서 사전 학습된 모델 중 가장 작은 모델인 ResNet 18을 사용하여 성능을 최대한으로 올려보려 한다.

torchvision 라이브러리를 통해 models.resnet18(weights=models.ResNet18_Weights.DEFAULT)와 같이 모델을 불러와 준비했다. 가중치는 pretrained된 가중치를 가져와 설정했으며 ResNet18_Weights.DEFAULT는 ImageNet-1K을 사용해 학습된 가중치이다. 프로젝트는 사람, 동물, 사물로 구분하는 모델이기 때문에 완전 연결 층의 출력을 3개로 수정했고 드롭아웃을 30% 적용하여 모델의 과적합을 방지했다. 또한 사람, 동물, 사물의 특징을 잘 구별하는 것이 필요했고 층이 깊어질수록 세부적인 특징을 추출하여 완전 연결 층에 출력을 만들기 때문에 총 4개의 블록에서 마지막 블록을 제외하고 나머지는 층은 Freeze를 적용하여 사전 학습된 가중치를 그대로 유지 시키고 마지막 4번 블록과 완전 연결 층에 대해서만 학습이 진행될 수 있게 설정하였다.

모델 학습을 위한 하이퍼파라미터의 경우 에포크는 50 에포크로 설정했고 배치 사이즈는 128로 선정했다. 학습률은 0.00001로 설정하여 모델이 천천히 학습하면서 안정적으로 수렴할 수 있게 했다. 그 외 학습 조기 종료를 위한 파라미터를 3으로 두어 조기 종료 시점을 설정하였다. (모델 구현에서 설명)

손실함수와 옵티마이저의 경우 Focal loss와 AdamW를 적용했다. Focal loss는 크로스엔트로피 손실함수의 개선된 형태이며 -α・(1－pt)^γ・log(pt)를 통해 기존의 크로스엔트로피 손실함수의 값에 α와 γ를 사용하여 특정 클래스(α)에 대해 더 중요하게(γ) 계산하는 것을 의미한다. 앞서 설명했듯이 Open Images의 이미지 데이터는 객체 탐지 모델을 위한 데이터로 대부분 구성되어 있고, 이는 이미지에 여러 클래스 객체가 있다는 것을 의미한다. 그리고 이미지를 가공하는 과정을 통해 원하는 클래스만 추출했지만, 일부 이미지는 다른 클래스도 포함이 될 수도 있다. Focal loss는 이러한 쉬운 샘플은 (사람, 동물, 사물이 이미지에 단독으로 존재) 무시하면서 어려운 샘플의 (이미지에 여러 클래스가 존재) 손실률에 집중하는 함수이다. 다만 α값은 객체 탐지 모델에서 클래스의 불균형 문제를 위해 클래스별 가중치를 설정하는 값이지만 프로젝트에서는 불균형 문제는 없으므로 1로 설정하여 동일하게 주었고 γ값의 경우 기여도의 정도를 의미하며 너무 높은 γ값은 모델이 편향될 수 있기에 2로 설정하였다.[8)]

8) Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, 「Focal Loss for Dense Object Detection」, 『ICCV 2017』, IEEE International Conference on Computer Vision, 2017

AdamW는 Adam 옵티마이저와 비슷하지만 Adam의 L2 정규화 문제를 해결한 옵티마이저이다. SGD 옵티마이저의 경우 모든 파라미터에 대해 같은 스케일로 업데이트하여 L2 정규화가 가중치 감쇠의 역할로 작동하지만, Adam의 경우 adaptive scaling으로 SGD와 다르게 L2 정규화가 스케일링의 영향으로 가중치 감쇠의 의미가 왜곡되기 때문에 AdamW는 기울기에서 L2 정규화를 분리하고 파라미터에 적용하여 adaptive scaling으로도 가중치 감쇠가 SGD와 같이 정확한 의미의 정규화로 작동할 수 있게 설계되었다.[9] AdamW는 Adam의 장점과 SGD의 장점을 챙겨 일반화 성능을 올리고 과적합을 방지할 수 있어 프로젝트에 채택하였다. 또한 AdamW의 성능을 높이기 위해 가장 성능이 좋은 조합인 cosine annealing 스케줄러를 도입하였다.[10] 매 에포크마다 코사인 함수의 형태로 학습률이 부드럽게 감소하면서 안정적인 학습이 진행될 수 있게 설정하였다. 싸이클 주기는 에포크의 절반으로 설정해서 학습 초기에는 모델이 천천히 안정적으로 학습하고 중반 이후부터 다시 빠르게 학습하여 최적점을 찾을 수 있게 설정하였다.

3. 모델 구현

앞서 설명한 준비 과정을 통해 모델이 학습할 준비를 마쳤으며 프로젝트의 ResNet.py에 최종 구현된 모델이 작성되어 있다. 학습은 총 50회를 진행하였고 매 학습 이후 평가를 진행하여 손실률과 정확도를 "[01] 학습: [Loss: 0.2034, Acc: 0.8464] | 평가: [Loss: 0.0744, Acc: 0.9568]" 와 같이 터미널에 기록하여 진행 상황 및 실시간 평가가 가능하도록 설계했다. 학습마다 손실함수를 통해 손실률을 계산하고 역전파를 통해 기울기를 계산하여 옵티마이저가 업데이트할 수 있게 하였으며 모델 평가에 대한 손실률을 통해 전 에포크보다 손실률이 개선되었으면 해당 모델을 저장하고 앞서 설명한 학습 조기 종료 시점은 3은 손실률 개선이 3번의 학습 동안 이루어지지 않으면 과적합 징후라 판단하여 학습을 조기 종료하여 학습 시간을 줄였다. 스케줄러를 업데이트하여 학습률을 조정하는 과정을 마지막으로 모델 구현을 마쳤다. 학습이 종료된 후 프로젝트 모델의 성능을 시각적으로 확인할 수 있게 학습곡선을 추가하였고 저장된 최고 성능 모델을 불러와 혼동행렬을 통해 모델의 약점을 파악할 수 있게 추가하였다. 또한 프로젝트 목표 달성을 확인하기 위해 scikit-learn 라이브러리의 classification_report 메소드를 사용하여 각종 평가지표 및 정확도를 확인하였다.

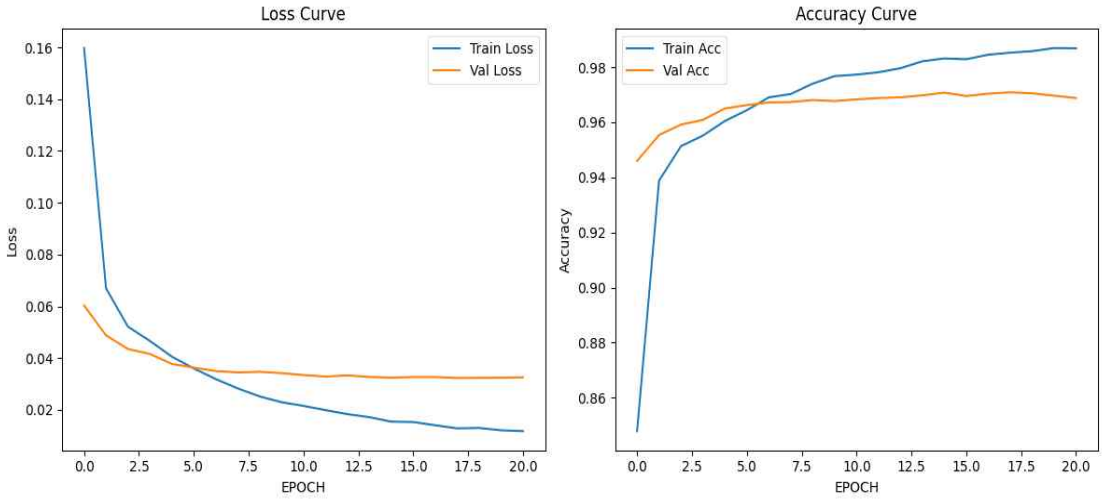
4. 모델 평가

프로젝트 모델은 21 에포크를 마지막으로 조기 종료되었다. <그림 6>의 모델 학습곡선을 보면 5 에포크까지는 학습 및 평가의 손실률이 급격하게 감소하였으며 이후부터는 스케줄러에 의해 완만하게 안정적으로 학습하였고 조금씩 손실률이 줄어들다가 15 에포크부터 큰 차이 없이 진행되다가 20 에포크를 기점으로 학습에 진전이 없자 21 에포크에 조기 종료되었다. 학습은 계속해서 손실률이 0으로 수렴하려 하지만 손실률의 그에 비해 평가에 대해서는 손실률이 크게 변하지 않고 유지되었으며 학습이 더 진행되었다면 모델의 과적합이 발생

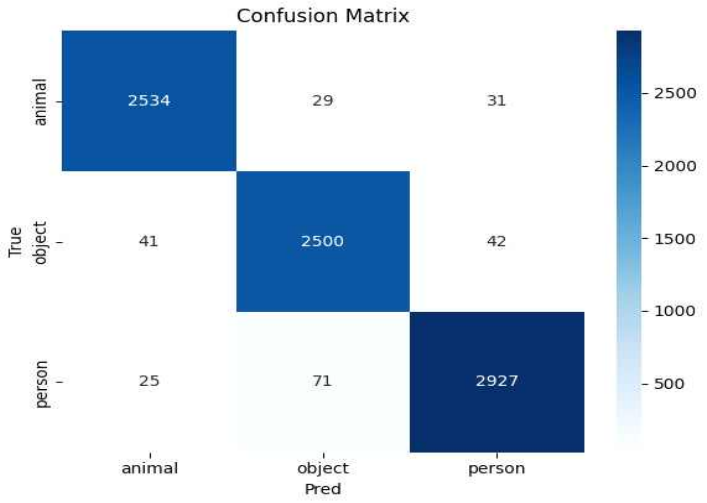
9) Ilya Loshchilov, Frank Hutter, 「Decoupled Weight Decay Regularization」, 『ICLR 2019』, International Conference on Learning Representations, 2019

10) 위의 논문

했을 것으로 예상된다. 정확도 또한 학습 정확도는 계속해서 증가하지만, 평가 정확도는 17 에포크를 기준으로 조금씩 감소하고 있는 것으로 확인되었고 학습을 더 진행하더라도 손실률과 정확도를 평가해 보았을 때 과적합이 발생해서 학습에 의미가 없을 것으로 예상된다.



<그림 6>



<그림 7>

다음은 학습을 통해 만든 최고 성능 모델에 대한 혼동행렬이다. <그림 7>과 같이 전체적으로 잘 예측했으나 가장 많은 오류는 사람을 사물로 예측한 경우였고 두 번째로 많은 오류는 사물을 사람으로 예측한 경우와 사물을 동물로 예측한 경우였다. 이 중 눈에 띄는 약점은 사람과 사물 간의 관계였다.

<표 1>과 같이 평가지표를 보면 정확도는 최종적으로 97%를 기록했고 F1-SCORE는 각각 0.98, 0.96, 0.97을 기록하였다. 평가지표를 통해 프로젝트 목표는 성공적으로 달성했다.

	precision	recall	f1-score	support
animal	0.97	0.98	0.98	2594
object	0.96	0.97	0.96	2584
person	0.98	0.97	0.97	3023
accuracy		0.97		8200
macro avg	0.97	0.97	0.97	8200
weighted avg	0.97	0.97	0.97	8200

<표 1>

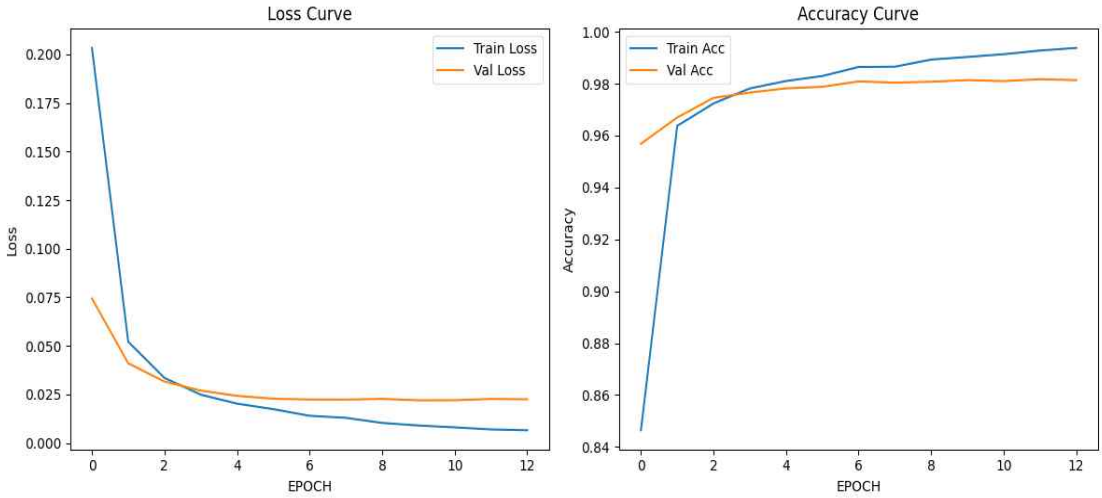
5. 모델 개선

앞서 확인한 모델의 문제점은 사람과 사물 간의 관계에 있어 오탐이 많다는 것이다. 프로젝트 모델을 테스트하면서 어떤 이미지에 대해서 오탐을 했는지 확인해 보았고 <그림 8>은 사람을 사물로 예측한 사례였다.

<그림 8>

해당 이미지는 총을 든 군인의 모습이다. 사물 데이터에는 총기에 대한 (클래스) 사진이 포함되어 있었고 해당 이미지는 두 개의 클래스가 (사람, 사물) 공존했고 분류에 있어 사람이 보통 생각하는 분류인 사람보다 사물로 예측한 것이다. 이러한 패턴이 (다수의 클래스가 공존) 프로젝트 모델의 성능을 떨어트린 요인이라 생각했고 준비된 데이터의 경우 이러한 패턴만 있는 것이 아니라 다양한 형태로 존재하다 보니 패턴에 대한 다양성은 높았으나 각각의 패턴에 대한 샘플이 부족하고 불균형하여 성능을 높이지 못한 것으로 예상된다. 또한 모델의 구조 자체가 특징을 잡아서 이미지를 분류하는 모델이고 여기서 더 나아간 형태인 객체 탐지 모델은 <그림 9>과 같은 이미지에서 추출된 특징을 바탕으로 이미지에서 클래스를 탐지하고 해당 클래스의 위치를 찾는 (이미지에서 어느 부분이 해당 클래스인지) 형태로 발전된 모델이다. 따라서 데이터의 패턴을 다양화하면서 각각의 데이터 수를 늘려서 모델이

어려운 패턴에 대해 잘 학습하게 만들거나 YOLO 모델과 같은 객체 탐지 모델로 프로젝트를 재구성하여 효과적으로 대응하는 방법이 프로젝트의 개선점이다. 다만 모델 재구성의 경우 고려해 볼 사항이 있었다. 손실함수를 Focal loss를 사용했기에 어려운 샘플에 대해 더 집중하며 이에 대한 손실률 개선의 기여도를 높게 설정하였고 패턴에 대한 다양한 학습이 부족한 것도 있지만 모델 자체가 이러한 패턴을 해석하기에는 합성곱 연산이 부족할 수 있다는 가정과 객체 탐지 모델의 경우 기존의 ResNet 18보다 연산량이 많아 처리 속도에 느리다는 점이다. 따라서 객체 탐지 모델로 개선하기보다는 ResNet의 신경망을 층을 늘리는 것이 해결책이 될 수 있다고 판단했다. 성능을 확인해 보기 위해 기존의 작성된 ResNet.py에서 ResNet 18을 50으로 바꾸고 기존의 설정값과 설계는 그대로 유지하면서 층만 추가했을 때 모델이 개선될 수 있는지를 확인해 보았다.

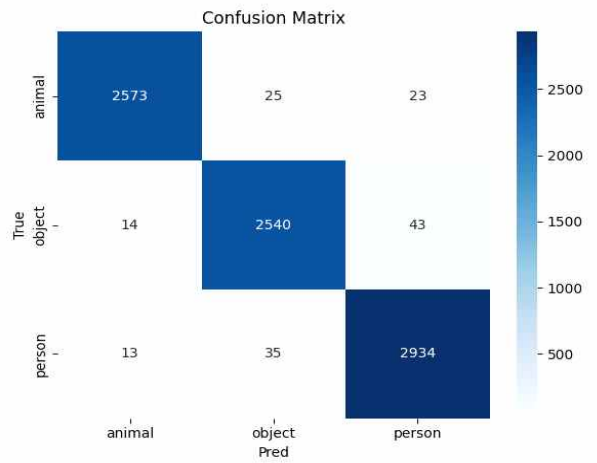


<그림 9>

<그림 9>는 ResNet 50을 사용했을 때 학습곡선이다. <그림 6>과 비교해 보았을 때 손실률도 많이 감소했고 무엇보다 학습 데이터와 평가 데이터에 대한 손실률 차이가 크게 좁혀진 것을 확인할 수 있었다.

<그림 10>은 ResNet 50의 최고 성능에 대한 혼동행렬이다. <그림 7>과 같이 가장 많은 오류를 만들었던 사람을 사물로 예측한 사례는 35개로 많이 감소한 모습을 보여주고 있고 다만 사물을 사람으로 예측한 사례는 1개 증가했지만, 거의 동일한 수준을 보이며 전체적으로 오류가 감소한 모습을 볼 수 있다. 또한 <표 2>를 보면 전체적인 성능 향상도 이루어졌는데 모델의 정확도는 98%로 1% 증가했으며 F1-SCORE가 전체적으로 개선되었음을 볼 수 있다.

향상된 성능의 모델을 <그림 8>을 사용하여 확인해 본 결과 사람으로 분류하는 것을 확인했다. 이로써 프로젝트가 기존의 사용한 방법인 ResNet 18의 성능을 더 높이면서도 약점을 개선하면서 테스트 결과 기존에는 잘못된 예측을 한 사례를 올바르게 예측하는 결과로 만들었다.



<그림 10>

	precision	recall	f1-score	support
animal	0.99	0.98	0.99	2621
object	0.98	0.98	0.98	2597
person	0.98	0.98	0.98	2982
accuracy		0.98		8200
macro avg	0.98	0.98	0.98	8200
weighted avg	0.98	0.98	0.98	8200

<표 2>

Ⅲ. 결론

ResNet 18을 사용하여 전이 학습한 프로젝트 결과는 정확도 97%, F1-SCORE 평균 0.97점으로 목표를 달성 했지만, 약점을 개선해 보고자 했고 모델의 설계를 최대한으로 가져가면서 성능을 높일 수 있는 층을 늘리는 방식을 테스트해 보았고 ResNet 50의 결과는 성공적이었다. 모델 선정 기준에 있어 ResNet 50은 ResNet 18보다는 우선이 될 수 없지만 성능이 목적이라면 실제 환경에서 충분히 채택될 여지가 있다. 객체 탐지 모델로 재구성하기보다 ResNet은 기존의 구조를 그대로 사용하면서 층만 늘려 사용할 수 있어 유동성 부분에서도 우위를 가질 수 있었다. 데이터에 대한 개선점도 적용이 된다면 혼동행렬에서 변화가 있을 것으로 예상되며 데이터 준비 과정에서 대한민국 환경에 맞는 데이터만을 추려서 사용했는데 프로젝트 모델의 범용성 부분에 있어 부족하다고 생각하며 데이터에 대한 개선점을 적용할 때 이러한 부분도 채워준다면 프로젝트 모델은 다양한 국가 환경의 감시 체계에서 큰 역할을 하는 모델로 성장할 것이라 예상한다.

Ⅳ. 참고 문헌

논문

Ilya Loshchilov, Frank Hutter, 「Decoupled Weight Decay Regularization」, 『ICLR 2019』, International Conference on Learning Representations, 2019

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 「Deep Residual Learning for Image Recognition」, 『CVPR 2016』, IEEE Conference on Computer Vision and Pattern Recognition, 2016

Simonyan, Karen, Zisserman, Andrew, 「Very Deep Convolutional Networks for Large-Scale Image Recognition」, 『ICLR 2015』, International Conference on Learning Representations, 2015

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, 「Focal Loss for Dense Object Detection」, 『ICCV 2017』, IEEE International Conference on Computer Vision, 2017

웹 사이트

abhinav-neil, resnet-transfer-learning, GitHub, 2023, https://github.com/abhinav-neil/resnet-transfer-learning

ckdgus1433, 파이토치 딥러닝 튜토리얼 예제 Transfer Learning(전이 학습), 네이버 블로그, 2018.11.27., https://blog.naver.com/ckdgus1433/221407059199

elevne, 딥러닝 파이토치 교과서 (4-2: CNN 전이학습(2)), 티스토리, 2023.03.12., https://elevne.tistory.com/entry/딥러닝-파이토치-교과서-4-2-CNN-전이학습2

googleapis.com, Open Images Dataset V7 and Extensions – Googleapis.com, https://storage.googleapis.com/openimages/web/index.html

PyTorch, AdamW – PyTorch 2.7 documentation, https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html

PyTorch, CosineAnnealingLR – PyTorch 2.7 documentation, https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html

PyTorch, resnet18 – Torchvision main document, https://docs.pytorch.org/vision/stable/models/generated/torchvision.models.resnet18.html

Solar the Nomadic Traveler, 딥러닝(6) - 전이 학습(1), 티스토리, 2023.03.18., https://solarthenomadictraveler.tistory.com/40

어쩌다보니 코딩하는 대학원생, [Pytorch] ResNet-18 코드 구현, 티스토리, 2022.12.22., https://haystar.tistory.com/94

정해서, Focal loss 설명, velog, 2021.10.20., https://velog.io/@heaseo/Focalloss-설명