전이학습을 활용한 Image CAPTCHA 해독프로그램 구현

컴퓨터공학과 2017104024 정민혁

지도 교수: 최진우 교수님

요약

CAPTCHA(Completely Automated Public Turing test to tell Computers and Human Apart)는 사용자를 사람인지 컴퓨터 프로그램인지 구분하기 위해 사용되는 방법의 통칭으로, 다양한 방면으로 활용되지만 주로 로그인이나 회원가입 등에 하나의 보안 인증 절차로 사용된다. 그 중 한 종류인 Image CAPTCHA는 그리드로 나뉜 하나의 사진 혹은 여러 장의 사진에서 특정 대상이 존재하는 사진을 모두 고르도록 요구하는 CAPTCHA로 Google이 제공하는 reCAPTCHA가 대표적 예시이다. 이에 본 프로젝트에서는 전이학습을 통해 Image CAPCHA에 주로 등장하는 물체들을 학습시킨 모델을 사용하여 Image CAPCHAT를 해독하는 어플리케이션을 설계하고자 한다.

1. 서론

1.1. 연구배경

CAPTCHA(Completely Automated Public Turing test to tell Computers and Human Apart)는 튜링 테스트의 한 종류로, 사용자와 컴퓨터 프로그램을 구분하기 위한 용도로 사용되는 방법의 통칭이다. 튜링 테스트는 1950년 앨런 튜링(Alan Turing)에 의해 제시된 개념으로, 인간이 컴퓨터와 인간을 구분하는데 있어서 기계가 인간이 할 수 있는 것을 할 수 있는지 그 능력을 검증하는 테스트이다. 튜링 테스트의 주체는 인간 평가자인 것에 반해, CAPTCHA는 기계가 컴퓨터와 인간 사용자를 구별한다는 점에서 CAPTCHA는 역 튜링 테스트(Reverse Turing Test)라고 부르기도 한다.

컴퓨터 프로그램을 구분하기 위한 CAPTCHA를 되려 컴퓨터 프로그램으로 해독하는 것은 역설적으로 들릴 수 있으나, AI를 이용하여 CAPTCHA를 풀려는 시도는 과거부터 존재해왔다. Ahn, Blum and Langford의 논문에서 저자들은 AI로 CAPTCHA를 풀지 못하면 컴퓨터와 사람을 구분할 방법이 존재한다는 것이며, 풀었을 경우에는 다른 복잡한 인식 문제에 적용 가능한 AI의 발전을 의미하는 윈-윈 상황을 가져온다고 주장한바 있다[1].

가장 많이 볼 수 있는 CAPTCHA의 형태로는 이미지에 적힌 텍스트를 읽고 그대로 입력하거나 해당 질문의 답을 기입하는 text CAPTCHA나, 다양한 물체나 풍경 사진을 활용한 Image CAPTCHA가 있다.

1.2. 연구목표

시간이 흐를수록 머신 러닝과 OCR 프로그램의 발전으로 기계들도 높은 수준의 판독률을 보이게 되면서 낮은 복잡도의 CAPTCHA는 개선이나 다른 보안 방식으로 대체됨이 요구되고 있다. Text CAPTCHA를 해독하는 연구는 많은 사례가 있는 반면, Image CAPTCHA를 해독하려는 연구는 그 수가 많지 않다. 따라서 본 프로젝트에서는 기계학습을 이용하여 가장 많이 사용되는 Image CAPTCHA인 reCAPTCHA V2의 해독 모델을 설계하여 사용자들에게 해독 자동화의 가능성과 기존 Image CAPTCHA가 안전하지 않음을 보이고자 한다.

reCAPTCHA V2에서는 3x3 또는 4x4 그리드 형식의 이미지에서 찾아야 하는 대상을 알려주고. 그리드 내에서 대상(이하 타겟)이 포함된 셀(또는 타일)을 모두 선택할 것을 요구한다. 그리드에 포함되는 이미지는 보통 유사한 카테고리에 속한 물체들인 경우가 많으며, 때로는 외관적으로 비슷한 형태를 띄는 물체들을 배치하기도 한다. 여기서 착안하여 구현하고자 하는 프로그램의 task를 그리드의 각 셀에 대한 Image Classification으로 재해석하였다.

각 셀을 훈련된 Image classification 모델의 input으로 넣어, output으로 그 셀이 어느 class에 속하는지 예측한다. 각 셀을 예측 결과를 모아 최종적으로 CAPTCHA의 타겟이 타일들에 존재하는지 표시하도록 한다. reCAPTCHA에서 많이 볼 수 있는 문제 유형으로 교통과 관련된 이미지에서 버스, 트럭 등 흔히 볼 수 있는 대상을 찾도록 하는 유형이 있다. Task의 범위를 제한하기 위해 본 프로젝트에서는 훈련시키는 모델을 교통과 관련된 몇 class로 한정하여 훈련을 진행한다.

모델을 훈련시키는 방법으로는 전이학습을 통해 학습을 진행한다. 효과적인 모델을 구축하기 위해선 수많은 데이터와 수많은 시행착오가 필요하기 때문에 기존에 다수의 class를 대상으로 학습이 진행된 모델을 기반으로 미세조정을 거쳐 필요한 형태로 새로운 모델을 학습시킨다.

2. 관련 기술 및 연구

2.1 reCAPTCHA

CAPTCHA의 한 종류인 reCAPTCHA는 Luis von Ahn, David Abraham, Manuel Blum, Michael Crawford, Ben Maurer, Colin McMillen, and Edison Tan에 의해 개발되었고, 2009년부터 구글에 소유되어 무료로 사용가능한 서비스이다. V2는 Image CAPTCHA의 형태를 띄며, 그리드 상에서 문제에서 제시하는 물체를 포함한 이미지를 찾아 클릭하는 방식이다. 인증과정에서 생기는 사용자의 불편함을 없애기 위해 사용자의 행동에 점수를 매기고 이를 근거로 작동하는 reCAPTCHA V3까지 출시된 상태이나, V2가 여전히 많이 사용되고 있다.

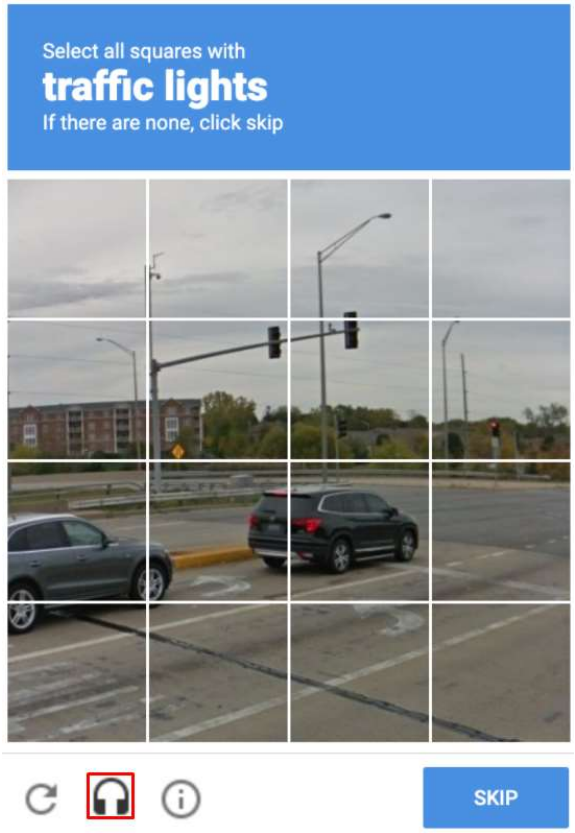

그림 1 4x4 reCAPTCHA V2의 예

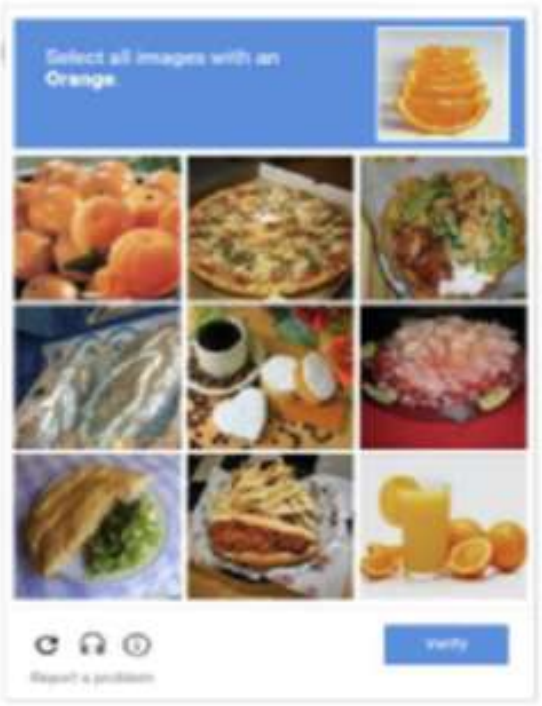

그림 2 3x3 reCAPTCHA V2의 예

2.2. 기계학습

기계학습(머신 러닝)은 컴퓨터과학의 한 분야로 경험을 통해 학습 또는 발전하는 알고리즘을 연구한다. AI연구의 한 분야로 간주되며, 모델을 구축하여 샘플 데이터를 통해 결과를 얻고, 이 결과를 이용하여 모델을 다시 수정하는 과정을 반복하면서 입력 값에 대해 원하는 결과를 출력하도록 수정하는 과정을 거친다. Tom M. Mitchell은 "어떤 경험이 작업의 평가를 더 좋게 만들면 그 경험을 통해 프로그램이 학습했다고 한다" 라고 컴퓨터의 학습을 정의하였다. ("A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.") [2]

기계학습은 가진 데이터를 이용하여 통계학적 모델을 학습시켜 인공지능을 구현하는 것으로, 이에 사용되는 모델은 사람의 신경망을 모방하는 형태로 구축된다. 모델을 이루는 네트워크 레이어의 개수가 많은 경우 이 방법론을 딥 러닝이라고 부른다. 이미지나 동영상을 활용하는 기계학습에 자주 사용되는 네트워크 구조로 CNN이 있고, 본 논문에서도 CNN을 활용하여 모델을 설계한다.

2.2.1 CNN[3]

CNN은 딥 러닝에 주로 사용되는 네트워크 구조의 한 종류로, 커널의 weight를 통한 convolution 연산을 거쳐, 모델에 있는 모든 노드들이 완전 연결되어 있는 것이 아닌 부분연결 구조를 가지는 모델의 복잡도가 낮은 모델이다. 기본적인 구조로 입력 계층과 컨볼루션 은닉 계층, 완전 연결 은닉 계층, 출력 계층의 형태로 네트워크가 구성되어 있다.

Convolution연산은 과거부터 영상처리나 신호처리 분야에서 많이 사용된 기술로 그 효과가 입증 되어있고, 따라서 CNN역시 영상과 관련된 작업에 강점을 보이며, Image Classification, Image Segmentation, Video Recognition등과 같은 연구 분야에 많이 사용된다.

2.2.2 전이학습

전이학습(Transfer learning)은 어떤 문제를 해결하기 위해 학습하면서 배운 지식을 다른 문제에도 적용시킬 수 있을 것이라는 생각에서 비롯된 학습 방법의 한 갈래이다. 처음부터 하나하나 모든 무작위로 초기화한 상태에서 시작하여 학습하는 것이 아니라 이전에 잘 학습된 모델을 가져와 초기 값으로 시작해 새로 적용하고자 하는 대상에 대해 학습을 진행하는 것으로 이 과정을 미세조정(fine-tuning)이라고 한다. 이는 데이터가 충분히 많은 양이 갖추어지지 않은 경우에도 효과적으로 학습을 할 수 있게 해주기 때문에 본 설계에서도 ImageNet 데이터셋으로 미리 학습된 신경망 중 하나를 사용하여 미세조정을 거쳐서 모델을 구축한다.

3. 프로젝트 진행 환경

3.1. 개발 환경

프로젝트에 사용된 소스코드는 파이썬 3.10.6버전에서 주피터 노트북을 통해 작성되었다. 기계학습은 Pytorch 라이브러리 1.12.1+cu113 버전에서 RTX3060 GPU를 사용하여 진행하였다. 동일한 환경에서 학습을 진행하기 위해서는 CUDA 및 CUDA toolkit 11.7 버전, cudnn 8.5.0 버전이 필요하다. 기타 소스코드 별 필요한 라이브러리는 requirements.txt 파일에 기재하였다.

3.2. 데이터셋

분류 모델의 학습을 위해 사용한 데이터셋은 Google에서 제공하는 Open Images Dataset V6에서 FiftyOne 라이브러리를 사용하여 필요한 클래스만을 다운로드 받아서 구성한 데이터셋을 사용하였다. 본 프로젝트에서 학습시키고자 한 클래스는 총 13개로, 다음과 같다:

```
'Airplane', 'Ambulance', 'Bicycle', 'Bus', 'Car', 'Fire hydrant',
'Helicopter', 'Motorcycle', 'Parking meter', 'Stop sign', 'Taxi',
'Traffic sign', 'Truck'
```

초기에 목표한 클래스의 수는 더욱 많았으나, 학습 정확도가 떨어지거나, 학습의 유용한 데이터의 수량이 타 클래스보다 부족한 경우를 제외하였다.

FiftyOne 라이브러리를 통해 train set, validation set에서 해당 클래스에 속하는 이미지들을 추출해 왔으며, 효과적인 학습을 위해 object centric하지 않은 이미지들이나 클래스가 잘못 분류된 이미지들을 솎아 냈다.

reCAPTCHA 이미지를 직접적으로 제공하는 데이터셋은 공개된 것을 찾지 못하여 임의로 reCAPTCHA 이미지를 생성해야 했다. 이미지를 NxN 그리드 형태로 자르거나 NxN개의 이미지를 붙여서 생성하는 방법이 있다. 이미지를 잘라서 reCAPTCHA를 생성하는 방법은 아무리 좋은 이미지를 찾아도 클래스에 속하는 물체의 종류와 개수가 너무 적어지는 경우가 많아 이미지를 붙여서 생성하는 방법을 채택하였다.

reCAPTCHA 생성을 위해 본래 Open Images 데이터셋의 test set을 사용하여 하였다. 그러나 Open Images에 있는 데이터의 경우 대부분 사람이 찍은 사진이기 때문에 여러 object가 섞여 있거나 상기한 경우와 같이 label이 잘못 달린 경우가 있었다. 따라서 Selenium 라이브러리를 사용하여 Google images에서 각 클래스에 속하는 이미지들을 크롤링하여 수집한 뒤, 이들을 랜덤하게 붙여 NxN 그리드 형태의 reCAPTCHA로 만들었다.

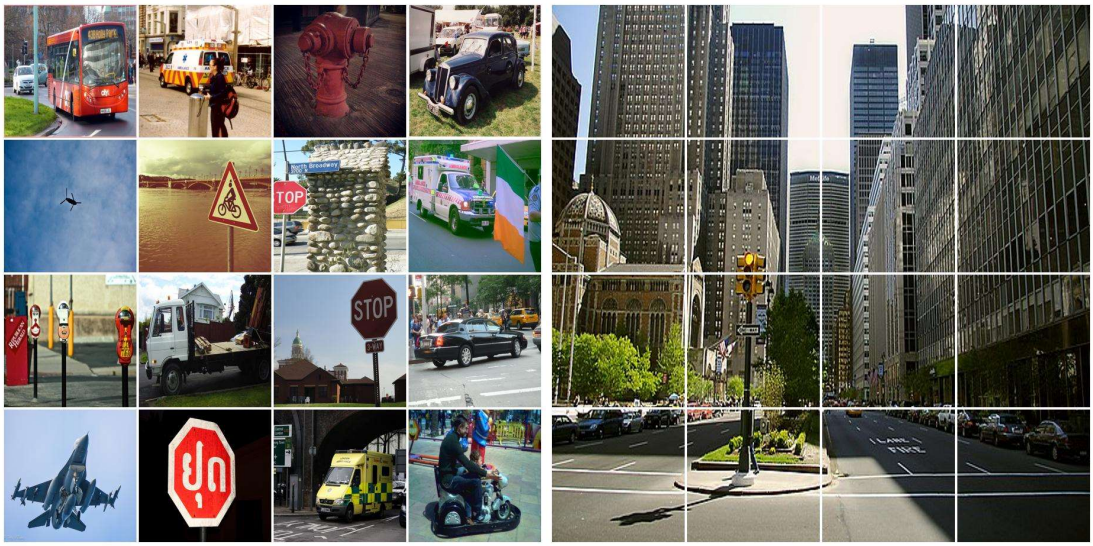

그림 4. 이미지를 붙여 임의로 생성한 4x4 reCAPTCHA와 이미지를 잘라 임의로 생성한 4x4 reCAPTCHA.

3.3. 모델

전이학습은 ImageNet을 대상으로 학습된 CNN네트워크들 중 높은 성능을 가진 것으로 알려진 ResNet152을 사용하여 초기 학습을 진행하였다. 이후 가능한 더 좋은 모델을 얻기 위해 ResNext101과 VGG16 모델 또한 사용하여 학습을 시켜 training/validation accuracy를 비교하였다. ResNet류의 네트워크와 VGG류 네트워크는 ILSVCR(ImageNet Large Scale Visual Recognition Challenge) 대회에서 우승/준우승을 거둔 모델로 다른 모델의 베이스로 종종 활용된다.

4. 결과

4.1 구현 소스코드

모든 소스 코드는 주피터 노트북 파일로 작성되어 있다.

- **CAPTCHA generator:** 클래스별 이미지들을 랜덤하게 조합해서 NxN 형태의 reCAPTCHA를 만드는 기능이 작성되어 있다. 클래스와 이미지는 중복되어 뽑힐 수 있다. 이미지를 붙여서 만드는 방식은 셀마다 답이 정해져 있으므로 reCAPTCHA를 생성하면서 답안을 기록한 텍스트 파일도 같이 저장한다.
- **CAPTCHA solver:** reCAPTCHA를 답을 예측하는 기능과 관련된 내용을 구현한 파일이다. 디렉토리 내 모든 reCAPTCHA를 예측하는 함수와 단일 reCAPTCHA 이미지의 답을 예측하는 기능이 주요 기능이다. 높지 않은 정확도 문제를 해결하기 위해 Softmax 함수를 거쳐서 나온 예측 확률 값에 thresholding을 적용하는 방법을 시도해 보았었고, 이 때의 최적의 threshold를 찾기 위한 함수와 코드 또한 작성되어 있다.
- **Dataset_downloader:** FiftyOne 라이브러리를 활용하여 각종 데이터셋을 쉽게 필요한 부분만 골라서 원하는 장수만큼 다운 받는 것이 가능하다. Open Images V6 데이터셋을 다운 받는데 필요한 내용이 작성되어 있다.
- **Image_augmentation:** Pillow, Albumentation 라이브러리를 활용하여 기존에 부족한 데이터를 보충하기 위해 기존 이미지를 각종 transformation을 적용시켜 새로운 이미지를 만드는 data augmentation을 위한 코드가 작성되어 있다.
- **Image_Scraper:** Selenium 라이브러리를 활용하여 구글 이미지로부터 reCAPTCHA generation에 필요한 이미지를 Scraping 해오는 코드가 작성되어 있다.
- **Model_train:** Image classification task를 위한 모델을 훈련시키는 코드가 작성되어 있다. 데이터의 수량과 훈련에 드는 시간을 고려하여 전이학습은 convolution layer는 동결시키고, fully-connected/classifier layer의 가중치만 갱신되도록 미세조정(fine-tuning)하였다. Pytorch 공식 튜토리얼을 참고하여 작성되었다.

4.2.1 Output

"CAPTCHA solver" 파일의 test함수를 통해 디렉토리 내 모든 reCAPTCHA 이미지에 대한 예측을 진행할 수 있고, test_single_captcha함수를 통해 경로의 reCAPTCHA 이미지 한 장에 대한 예측을 하고 이에 대한 결과를 볼 수 있다.

test함수를 통해 디렉토리 전체에 대해 예측을 시행할 경우 출력 예시는 다음과 같다.

```
test(model=model, dataloader=data_loader_merge, device=DEVICE, class_names=class_names, mode="merge")
```

```
1) target: Traffic sign, prediction: [6], in-top3: [5, 6, 10, 13], answer: [5]
2) target: Ambulance, prediction: [], in-top3: [], answer: []
3) target: Traffic sign, prediction: [7], in-top3: [7, 12, 13], answer: [15]
4) target: Helicopter, prediction: [6], in-top3: [6], answer: []
5) target: Motorcycle, prediction: [9, 11, 13], in-top3: [4, 7, 9, 11, 13, 15], answer: [14]
```

…

```
195) target: Motorcycle, prediction: [1, 11, 13], in-top3: [1, 5, 8, 9, 10, 11, 13, 14], answer: [8]
196) target: Bus, prediction: [2, 9, 14], in-top3: [2, 9, 10, 14], answer: [1, 4, 14]
197) target: Truck, prediction: [10], in-top3: [3, 10], answer: [5, 11]
198) target: Fire hydrant, prediction: [5], in-top3: [5], answer: [0]
199) target: Helicopter, prediction: [1, 2, 11, 13], in-top3: [1, 2, 5, 6, 10, 11, 13, 15], answer: []
200) target: Bicycle, prediction: [11], in-top3: [0, 1, 9, 11], answer: [0, 12]
Test accuracy: 0.1
```

test함수의 parameter들의 설명은 다음과 같다.

- model: 사전에 image classification을 위해 학습된 모델. 초기 테스트에는 resnet152 모델을 기반으로 전이학습을 진행한 모델이 사용되었다.
- dataloader: 디렉토리를 돌면서 이미지를 분할하여 모델의 입력으로 넣기 위한 pytorch 라이브러리를 상속받아 구현된 커스텀 dataloader.
- device: cpu와 gpu 중 어느 쪽을 사용하여 연산할 것인지 결정. torch.device 매소드를 통해 사전에 미리 선언한다.
- class_names: 이미지를 분류할 클래스 이름들을 담은 리스트
- save_dir(추가): 예측 결과 이미지를 저장할 디렉토리. 디폴트로 "../temp/" 로 되어있다.

실행 시에 CAPTCHA 이미지에서 찾을 타겟 클래스가 이미지 별로 무작위로 정해지며, dataloader에서 해당 이미지와 이미지의 각 셀의 정답을 불러온다. 이미지의 각 셀 별로 훈련된 모델을 통과하여 image classification을 하며, 각 셀이 클래스 별로 속할 확률을 얻게 된다. 이 때 확률이 가장 높은 클래스가 찾는 타겟 클래스와 같을 경우 'prediction' 항목에 해당 셀의 인덱스를 저장하게 된다. Prediction과 Answer항목이 완전히 일치할 경우 온전한 정답으로 간주하여 Accuracy를 측정하였다. 찾고자 하는 타겟 클래스가 랜덤하게 정해지므로 성능과 정확도가 실행마다 다르게 나오며, 5%-10% 정확도를 보였다. In-top3 항목은 부족한 성능을 보완하기 위해 추가된 항목으로 이후 절에서 후술한다.

4X4 reCAPTCHA를 기준으로 그리드 내 셀의 인덱스는 표와 같다.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

표1. reCAPTCHA 그리드 인덱스

4.2.2 결과 분석

200개의 생성된 reCAPTCHA이미지에 대해 예측을 시행한 결과, 보통 5-10% 사이의 정확도를 보였으며, reCAPTCHA solver의 저조한 성능에 대한 이유로 다음과 같은 원인들을 찾을 수 있었다.

1. Image classification 모델의 정확도가 충분히 높지 않으며 훈련 데이터셋에 대해 과적합이 의심된다.
2. 정답을 맞추기 위해선 타겟이 포함된 셀 만을 찾아서는 안 되고, 다른 셀을 타겟으로 오판해서도 안 된다.
3. CAPTCHA solving task의 난이도가 기본적으로 높다.

첫번째 원인은 전이학습을 시킨 모델의 정확도가 크게 높지 않은 점이다. 초기 훈련을 통해 얻은 모델은 training set에 대해 약 85%의 정확도, validation set에 대해 약 75%의 정확도를 보였다. 이는 우수하다고 말하기는 어려운 정확도이다. 다음은 모델 확인을 위해 training set, validation set, test set 이미지를 전부 훈련시킨 모델에 통과시켰을 때의 정확도이다.

```
Class Airplane accuracy: 379/425 -> 0.8917647058823529
Class Ambulance accuracy: 230/252 -> 0.9126984126984127
Class Bicycle accuracy: 301/322 -> 0.9347826086956522
Class Bus accuracy: 311/335 -> 0.9283582089552239
Class Car accuracy: 232/307 -> 0.755700325732899
Class Fire hydrant accuracy: 325/329 -> 0.9878419452887538
Class Helicopter accuracy: 349/380 -> 0.9184210526315789
Class Motorcycle accuracy: 329/374 -> 0.8796791443850267
Class Parking meter accuracy: 176/177 -> 0.9943502824858758
Class Stop sign accuracy: 272/302 -> 0.9006622516556292
Class Taxi accuracy: 222/239 -> 0.9288702928870293
Class Traffic sign accuracy: 291/315 -> 0.9238095238095239
Class Truck accuracy: 255/291 -> 0.8762886597938144
Total accuracy: 3672/4048 -> 0.9071146245059288
```

```
Class Airplane accuracy: 85/100 -> 0.85
Class Ambulance accuracy: 73/88 -> 0.8295454545454546
Class Bicycle accuracy: 70/80 -> 0.875
Class Bus accuracy: 128/158 -> 0.810126582278481
Class Car accuracy: 103/140 -> 0.7357142857142858
Class Fire hydrant accuracy: 89/90 -> 0.9888888888888889
Class Helicopter accuracy: 126/140 -> 0.9
Class Motorcycle accuracy: 117/140 -> 0.8357142857142857
Class Parking meter accuracy: 60/60 -> 1.0
Class Stop sign accuracy: 64/79 -> 0.810126582278481
Class Taxi accuracy: 111/140 -> 0.7928571428571428
Class Traffic sign accuracy: 107/130 -> 0.823076923076923
Class Truck accuracy: 98/140 -> 0.7
Total accuracy: 1231/1485 -> 0.828956228956229
```

그림5,6. Training set, Validation set을 모델에 통과시켰을 때의 정확도

```
Class Ambulance accuracy: 85/95 -> 0.8947368421052632
Class Bicycle accuracy: 94/99 -> 0.9494949494949495
Class Bus accuracy: 39/41 -> 0.9512195121951219
Class Car accuracy: 37/44 -> 0.8409090909090909
Class Fire hydrant accuracy: 87/92 -> 0.9456521739130435
Class Helicopter accuracy: 89/94 -> 0.9468085106382979
Class Motorcycle accuracy: 94/94 -> 1.0
Class Parking meter accuracy: 89/91 -> 0.978021978021978
Class Stop sign accuracy: 84/92 -> 0.9130434782608695
Class Taxi accuracy: 61/90 -> 0.6777777777777778
Class Traffic sign accuracy: 64/89 -> 0.7191011235955056
Class Truck accuracy: 81/93 -> 0.8709677419354839
Total accuracy: 984/1107 -> 0.8888888888888888
```

그림7. Test set을 모델에 통과시켰을 때의 정확도

그림5,6,7을 종합하여 보았을 때 Car, Taxi, Traffic sign, Truck 클래스가 비교적 낮은 정확도를 보였으며, 그 외에도 Bus, Stop sign이 Validation set에서는 위의 4클래스에 준하게 낮은 정확도를 보였다. 그럼에도 불구하고 전체 데이터 셋 정확도는 82~90% 범위에 속하는 것을 보아 가지고 있는 데이터 셋에 모델이 overfitting 되었고, Car와 Taxi, Traffic sign과 Stop sign 같은 일부 유사 클래스에 대해서 제대로 된 학습이 이루어지지 않은 것으로 보인다.

두번째 원인으로 오판에 의해 CAPTCHA 해독 정확도가 저하되는 원인이 있다. 타겟 클래스가 있는 셀을 모두 잘 찾아낸 경우에도 만약 다른 셀에서도 타겟 클래스가 있다고 판독할 경우 최종적으론 오답이 된다. 예를 들어 예측이 [0,1,2,7] 인데 정답이 [0,2,7]인 경우도 오답이 된다.

세번째 원인으로 이 task자체의 난이도가 예상했던 것보다 어렵다는 것이다. NxN 그리드 형태의 reCAPTCHA를 각 셀마다 image classification을 진행하면 모델이 p의 정확도를 가질 때 NxN개의 셀 모두 올바르게 예측할 확률은 $P^{(N\times N)}$ 이 된다. 4x4개의 셀에 대해 90% 정확도를 가진 모델로 예측을 한다고 해도 0.9^16 = 0. 1853...로 매우 낮다. 실제 task는 몇몇 셀에서 오판을 하게 되어도 타겟 클래스로 오판하지만 않으면 최종 예측엔 포함시키지 않기 때문에 실제로 최종 예측 성공 확률은 이것보다는 높지만 image classification을 통한 reCAPTCHA의 해독자체가 매우 높은 정밀도를 요구하는 난이도 높은 작업인 것은 분명하다.

4.3 성능 개선을 위한 시도

4.3.1 그리드 축소

해독하고자 하는 reCAPTCHA 이미지가 크면 클수록 난이도가 비약적으로 상승하기 때문에 그리드를 축소해보라는 피드백을 받았었다. 따라서 2x2, 3x3 형태의 reCAPTCHA를 임의로 생성하여 테스트를 진행해보았고, 200개의 생성된 이미지를 대상으로 2x2에 대해서는 평균 60%, 3x3에 대해서는 평균 30%에 가까운 정확도를 보였다. (타겟 클래스는 test에서 랜덤하게 정해지고, 이에 따라 정확도가 시행마다 달라지게 된다.) Task의 난이도를 직접적으로 낮추었기 때문에 시도했던 방법들 중 가장 큰 효과를 보았다.

4.3.2 Thresholding

모델이 타겟 클래스가 있는 셀을 잘 찾아도 다른 셀에서 오판을 하면 오답을 도출하게 된다는 점에서 착안하여 softmax를 취한 이후 값에 threshold 값을 주어 일정 확신이 없으면 예측에 포함시키지 않는 전략을 떠올렸다. 하지만 이 방법은 시도는 해볼 수 있지만 precision은 높아져도 recall이 떨어질 수 있어서 정확도 향상에 큰 이득이 없을 것으로 보인다는 조언을 받았다.

Test set 이미지를 대상으로 1부터 0까지 0.5간격으로 threshold를 적용해보면서 True Positive(TP), False Positive(FP), False Negative(FN), True Negative(TN)의 개수를 세고, False Positive Rate(FPR, fall-out)과 True Positive Rate(TPR, recall) 값을 구해 ROC커브를 그려보고, 이에 따른 최적의 threshold를 구해보았다. 그 결과, TPR-FPR값이 최대가 되는 threshold=0.8을 최적의 threshold으로 얻었다.

```
Threshold: 1.0 | FPR: 0.0 | TPR: 0.0
Threshold: 0.975 | FPR: 0.024390243902439025 | TPR: 0.5487804878048781
Threshold: 0.9500000000000001 | FPR: 0.05690569105691054 | TPR: 0.6402439024390244
Threshold: 0.925 | FPR: 0.08130081300813008 | TPR: 0.7022357723577236
Threshold: 0.9 | FPR: 0.12195121951219512 | TPR: 0.7489837398373984
Threshold: 0.875 | FPR: 0.15447154471544716 | TPR: 0.7876016260162602
Threshold: 0.85 | FPR: 0.17073170731707318 | TPR: 0.801829268292683
Threshold: 0.8250000000000001 | FPR: 0.17886178861788618 | TPR: 0.823170731707317
Threshold: 0.8 | FPR: 0.1951219512195122 | TPR: 0.8414634146341463
Threshold: 0.775 | FPR: 0.21951219512195122 | TPR: 0.858739837398374
Threshold: 0.75 | FPR: 0.25203252032520324 | TPR: 0.8699186991869918
Threshold: 0.725 | FPR: 0.2764227642276423 | TPR: 0.8851626016260162
Threshold: 0.7000000000000001 | FPR: 0.3008130081300813 | TPR: 0.8943089430894309
Threshold: 0.675 | FPR: 0.3008130081300813 | TPR: 0.9024390243902439
Threshold: 0.65 | FPR: 0.3170731707317073 | TPR: 0.9186991869918699
Threshold: 0.625 | FPR: 0.3821138211382114 | TPR: 0.9278455284552846
Threshold: 0.6 | FPR: 0.43902439024390244 | TPR: 0.9349593495934959
Threshold: 0.5750000000000001 | FPR: 0.4878048780487805 | TPR: 0.9400406504065041
Threshold: 0.55 | FPR: 0.5447154471544715 | TPR: 0.948170731707317
Threshold: 0.525 | FPR: 0.6016260162601627 | TPR: 0.959349593495935
Threshold: 0.5 | FPR: 0.6585365853658537 | TPR: 0.9644308943089431
Threshold: 0.47500000000000003 | FPR: 0.6829268292682927 | TPR: 0.9684959349593496
Threshold: 0.45 | FPR: 0.7317073170731707 | TPR: 0.9715447154471545
Threshold: 0.425 | FPR: 0.7479674796747967 | TPR: 0.9806910569105691
Threshold: 0.4 | FPR: 0.7886178861788617 | TPR: 0.9857723577235772
Threshold: 0.375 | FPR: 0.8048780487804879 | TPR: 0.9878048780487805
Threshold: 0.35000000000000003 | FPR: 0.8455284552845529 | TPR: 0.991869918699187
Threshold: 0.325 | FPR: 0.8861788617886179 | TPR: 0.9949186991869918
Threshold: 0.3 | FPR: 0.9024390243902439 | TPR: 0.9979674796747967
Threshold: 0.275 | FPR: 0.9186991869918699 | TPR: 0.9989837398373984
Threshold: 0.25 | FPR: 0.9512195121951219 | TPR: 1.0
Threshold: 0.225 | FPR: 0.967479674796748 | TPR: 1.0
Threshold: 0.2 | FPR: 0.983739837398374 | TPR: 1.0
Threshold: 0.17500000000000002 | FPR: 1.0 | TPR: 1.0
Threshold: 0.15 | FPR: 1.0 | TPR: 1.0
Threshold: 0.125 | FPR: 1.0 | TPR: 1.0
Threshold: 0.1 | FPR: 1.0 | TPR: 1.0
Threshold: 0.075 | FPR: 1.0 | TPR: 1.0
Threshold: 0.05 | FPR: 1.0 | TPR: 1.0
Threshold: 0.025 | FPR: 1.0 | TPR: 1.0
Threshold: 0.0 | FPR: 1.0 | TPR: 1.0
Optimal threshold: 0.8
```

그림8. Optimal threshold value를 찾기 위한 계산 결과

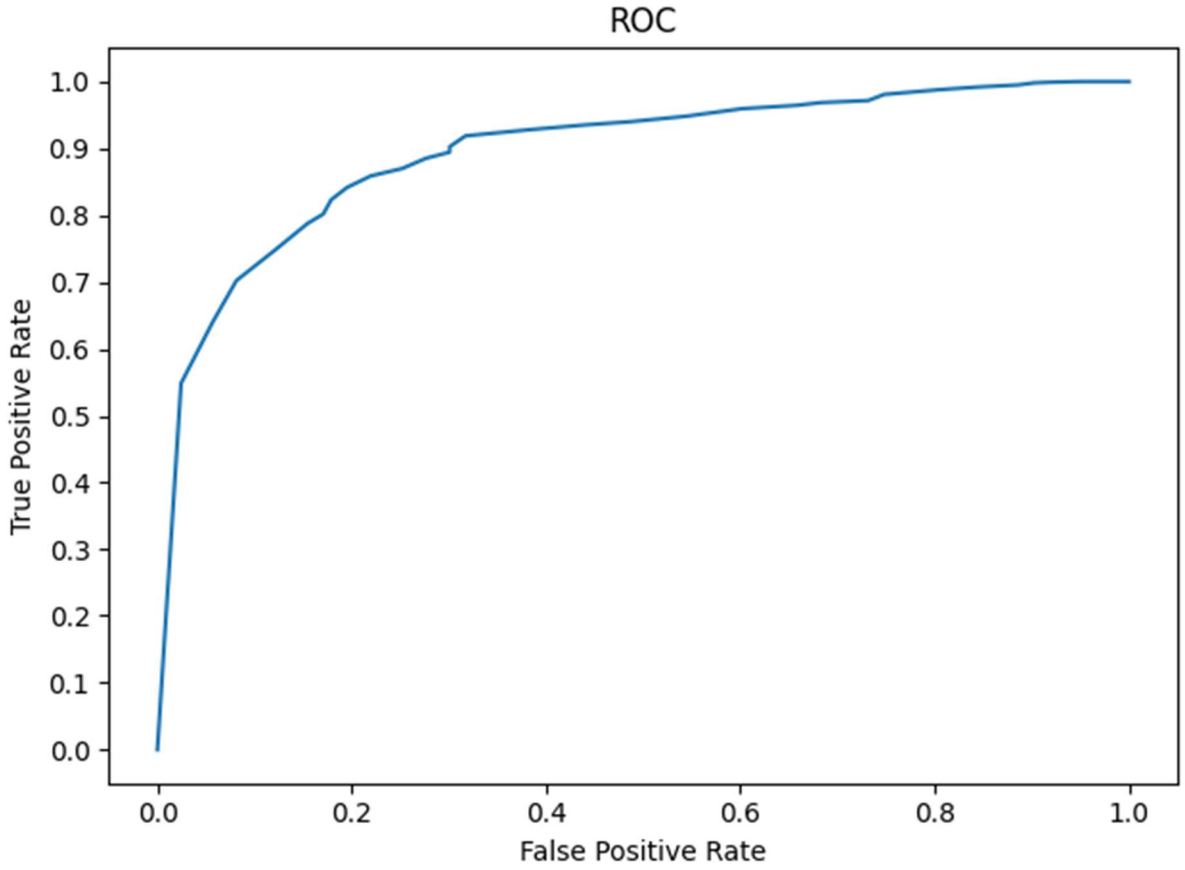

그림9. ROC Curve to find optimal threshold

```
Class Airplane accuracy: 76/93 -> 0.8172043010752689
Class Ambulance accuracy: 85/95 -> 0.8947368421052632
Class Bicycle accuracy: 89/99 -> 0.898989898989899
Class Bus accuracy: 36/41 -> 0.8780487804878049
Class Car accuracy: 37/44 -> 0.8409090909090909
Class Fire hydrant accuracy: 89/92 -> 0.967391304347826
Class Helicopter accuracy: 78/94 -> 0.8297872340425532
Class Motorcycle accuracy: 89/94 -> 0.9468085106382979
Class Parking meter accuracy: 86/91 -> 0.945054945054945
Class Stop sign accuracy: 72/92 -> 0.782608695652174
Class Taxi accuracy: 58/90 -> 0.6444444444444445
Class Traffic sign accuracy: 50/89 -> 0.5617977528089888
Class Truck accuracy: 82/93 -> 0.8817204301075269
Total accuracy: 927/1107 -> 0.8373983739837398
```

Threshold=0.8을 적용시켜 모델의 image classification 정확도를 재측정한 결과 조언 내용처럼 정확도면에서 소득은 없었고, 오히려 정확도가 떨어지는 모습을 보였다. 이렇게 된 원인을 찾기 위해 이미지를 모델에 넣었을 때 나오는 output tensor를 출력하여 확인한 결과, softmax를 거친 값들이 종종 가장 큰 값이 0.6 근처에 머무는 경우도 있었다. 따라서 thresholding이 이러한 경우도 걸러내 버리면서 정확도면에서 득보다 실이 커지게 되는 현상이 생긴 것으로 보인다.

4.3.3 Data augmentation and more train

효과가 높진 않을지 몰라도, 모델을 정확도를 높이는 것 또한 필요하기 때문에 조건을 바꾸어 가면서 모델을 여러 번 다시 훈련시켰다. 학습 Epoch 수, weight decay, learning rate 등의 파라미터를 바꿔보았고, data augmentation을 통해 훈련 데이터 수를 늘리기도 하였고, 전이학습에 사용하는 모델 종류를 바꿔 보기도 하였다.

Resnet152, Resnext101, Vgg16 모델을 기반으로 동일한 다른 조건들은 고정하여 Epoch 50/100으로 기존 훈련 데이터셋과 data augmentation으로 늘린 데이터셋으로 바꿔가면서 훈련을 시켜 training accuracy와 validation accuracy를 벤치마킹하여 다음과 같은 결과를 얻었다.

dataset		Open Images		Open Images ext	
epoch		50	100	50	100
Resnet152	train acc	85.08	84.86	81.56	82.47
	val acc	88.65	87.97	88.43	88.65
Resnext101	train acc	86.56	86.41	84.45	84.36
	val acc	89.18	89.03	89.41	89.49
Vgg16	train acc	82.33	81.84	77.92	77.69
	val acc	83.59	84.12	84.04	84.72

표2. Resnet152, Resnext101, Vgg16 모델을 조건을 바꿔가며 훈련시켰을 때의 결과

Resnext101 모델을 기반으로 전이학습을 진행했을 때 가장 높은 정확도를 보이나, 다른 모델에 비해 눈에 띄게 큰 차이는 없어, 유의미한 시도는 아니었다고 생각된다. 이 방법이 효과를 거두기 위해서는 데이터를 훨씬 더 대용량으로 늘리고 학습 횟수 또한 늘리고 파라미터들을 조정해가며 많은 시행착오를 거쳐야 할 것으로 생각된다.

4.4 Top-K

모델에 입력 이미지를 넣으면 모델은 출력으로 각 클래스별로 입력 이미지가 속할 확률의 의미를 담은 수치를 가진 tensor를 반환하게 된다. Softmax 함수를 거치면 이 수치들은 합해서 1이 되는 확률 값으로 변환되고, Top-k란 여기서 가장 높은 k개의 값을 취하는 것을 뜻한다. 즉 입력 이미지가 속할 가능성이 높은 클래스 k개를 뽑는 것과 같다. 본 프로젝트에서는 출력 클래스의 개수가 13개로 적기 때문에 top-3값을 취하여, CAPTCHA의 셀마다 예측할 때 타겟 클래스가 top-3 결과 안에 있을 경우, 그 셀의 "in-top-3"라는 변수에 보관하여 최종적 결과에 '정답일수도 있는 셀'의 의미의 보조지표로 활용하기로 하였다. 이는 Thresholding을 시도할 때 발견한 예측 결과 값이 의외로 높지 않은 경우가 있다는 점에서 착안하였다. 최종적으로 모델의 예측 결과가 찾는 타겟 클래스일 경우 초록색 테두리로 표시하고, top-3안에 타겟 클래스가 있을 경우 노란색 테두리로 표시하고, 타겟 클래스가 발견되지 않은 경우 전체 이미지를 빨간 테두리로 표시하도록 구성하였다.

그림 10,11,12,13. CAPTCHA solver 출력 예시. 10: Ambulance, 11: Airplane, 12: Stop sign, 13: Ambulance를 타겟.

5. 결론 및 발전 가능성

본 프로젝트를 통해 일정 수준으로 학습된 image classification 모델을 통해 그리드 사이즈가 작은 image CAPTCHA의 해독이 가능함을 일부 보일 수 있었다. 모델의 성능에 크게 의존하는 이상 그리드의 사이즈가 커지면 정확도가 떨어져 image classification과 전이학습만으로는 예상했던 것보다 task의 난이도가 높다는 것 또한 알 수 있었다. 학습된 모델은 유사한 클래스간 구분이나 여러 물체가 등장하는 이미지에서 특히 낮은 정확도를 보였다. 더 정밀한 분류 모델을 위해서는 대용량의 학습데이터와 장시간의 학습이 필요할 것으로 판단된다.

프로젝트를 발전시킬 방법으로 image classification 대신 object detection으로 문제를 접근할 수 있을 것으로 보인다. 한 셀의 이미지가 다수의 물체를 포함시킨 경우, image classification으로는 정답을 놓칠 가능성이 있지만, object detection 모델로는 이미지를 검출하기만 한다면 정답 셀을 찾을 가능성이 더 높을 것으로 예상된다. 발견한 사례로 2020년 HOSSEN은 reCAPTCHA의

robustness 검증을 위해 Google, Microsoft, Amazon등에서 제공하는 Image Recognition Service와 Object Detection Service를 사용하여 분석을 진행하였다[4].

본 프로젝트를 통해 reCAPTCHA와 같은 Image CAPTCHA가 아직 많이 사용되고 있는지 알 수 있었다. 다만 온라인 상에서 Human resource를 통해 각종 CAPTCHA를 풀어주는 서비스나, 정확도를 알 수는 없지만 유료로 API key를 제공하여 CAPTCHA를 해독해주는 프로그램을 제공하는 서비스가 많이 보이는 만큼, CAPTCHA류 보안장치들의 강화 조치가 필요함은 여전히 부각된다.

6. 참고문헌

1. von Ahn, Luis; Blum, Manuel; Hopper, Nicholas J.; Langford, John (May 2003). "CAPTCHA: Using Hard AI Problems for Security". Advances in Cryptology — EUROCRYPT 2003. EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science. Vol. 2656. pp. 294–311.

2. Mitchell, T. M., & Mitchell, T. M. (1997). *Machine learning* (Vol. 1, No. 9). New York: McGraw-hill.

3. 오일석, (2017). 기계학습 pp. 23-25, 194-200, 225-229.

4. HOSSEN, Md Imran, et al. An Object Detection based Solver for {Google's} Image {reCAPTCHA} v2. In: *23rd international symposium on research in attacks, intrusions and defenses (RAID 2020)*. 2020. p. 269-284.

5. Pytorch reference: Pytorch documentation / tutorial

6. 김건우, 염상준 (2019), (펭귄브로의)3분 딥러닝 파이토치맛, 한빛미디어, ISBN: 9791162242278 93000