

201635924 김민석

201635922 김동주

201636042 이지윤

CNN을 통한 악성코드 분석

목차

[프로젝트 목표 - 2 -](#_Toc74931324)

[프로젝트 계획 - 2 -](#_Toc74931325)

[프로젝트 과정 및 결과 - 3 -](#_Toc74931326)

[악성코드 탐지 - 3 -](#_Toc74931327)

[악성코드 탐지 준비 - 3 -](#_Toc74931328)

[악성코드 탐지 (1차시도) - 3 -](#_Toc74931329)

[악성코드 탐지 (2차시도) - 4 -](#_Toc74931330)

[악성코드 탐지 (최종) - 5 -](#_Toc74931331)

[악성코드 분류 - 6 -](#_Toc74931332)

[분류 종류 - 6 -](#_Toc74931333)

[악성코드 분류 (1차시도) - 7 -](#_Toc74931334)

[악성코드 분류 (2차시도) - 7 -](#_Toc74931335)

[악성코드 분류 (3차시도) - 8 -](#_Toc74931336)

[악성코드 분류 (최종) - 8 -](#_Toc74931337)

[결론 - 10 -](#_Toc74931338)

[참고문서 - 11 -](#_Toc74931339)

# 프로젝트 목표

악성코드 분석에는 크게 두 종류가 있다. 정적 분석과 동적 분석이 그것이다. 이중 우리 팀은 정적 분석을 도전해보기로 했다. 정적 분석의 방법중에서도 바이너리 데이터를 통해 분석 목표로 했다.

해당 목표로 프로젝트를 기획하던 중 인공지능을 활용하여 악성코드를 탐지 및 분류를 해보면 어떨까 싶었다. 바이너리 데이터를 바이너리 이미지 파일로 변환하고, CNN을 통해서 학습을 시키는 방식이 가능한지, 가능할 경우 탐지와 분류에서 어느 정도의 정확도를 보여줄지를 확인하는 것이 이 프로젝트의 목표이다.

# 프로젝트 계획

프로젝트는 다음과 같은 방식으로 진행한다.

1. 악성코드 파일과 일반 파일들을 바이너리 이미지로 변환한다.
2. 변환된 이미지들을 다음과 같이 두가지로 분류한다.
   1. 악성코드와 정상파일
   2. 각 악성코드 종류별
3. 분류된 악성코드와 정상파일을 통해 학습을 진행하고 실제 탐지가 가능한지 확인한다.
4. 분류된 악성코드 종류별 파일들을 통해 학습을 진행하고 실제 분류가 가능한지 확인한다.

# 프로젝트 과정 및 결과

## 악성코드 탐지

### 악성코드 탐지 준비

우선적으로 악성코드 파일과 일반 파일들을 바이너리 이미지로 변환하였다. 그림1은 각각의 악성코드 파일과 일반 파일들의 대표적인 바이너리 이미지 이다.

|  |  |
| --- | --- |
|  |  |
| 악성코드 바이너리 이미지 | 일반 파일 바이너리 이미지 |

그림 악성코드와 일반파일 바이너리 이미지

바이너리 이미지를 모두 만든 이후에는 직접적으로 CNN모델에 넣어 학습을 시킨다. 해당 과정에서는 여러 시행착오가 있었으며, 아래에는 그 과정들과 결과적으로 최종 결과물을 기술한다.

### 악성코드 탐지 (1차시도)

첫번째 시도에 사용된 데이터의 개수는 5000개였다. 학습에 활용한 데이터가 너무 적어서 인지 Overfitting이 너무 쉽게 발생하였고, 실제 예측 결과도 좋지 못했다. 그림2는 1차시도에 대한 Model Accuracy 및 Model Loss이다.

|  |  |
| --- | --- |
|  |  |

그림 악성코드 탐지(1차시도) 결과

학습 결과 Train Accuracy는 90% 후반, Validation Accuracy는 80% 후반대의 성능을 보여준다. 하지만 절대적인 데이터가 부족하여 약 60 epoch에서부터 Overfitting이 발생하는 것을 확인할 수 있다.

### 악성코드 탐지 (2차시도)

훈련 데이터셋 부족을 해결하기 위해서 데이터 증식 기법을 활용하기로 결정하기로 했다. 5000개의 데이터를 데이터 증식하여 13000개의 데이터를 확보하였다. 이후 이 데이터들을 모델이 넣어 훈련시켰으며, 그림3은 그 결과이다.

|  |  |
| --- | --- |
|  |  |

그림 악성코드 탐지(2차시도) 결과

학습 결과 Overfitting 현상은 적어진 것이 확인된다. 하지만 Train Accuracy는 90% 초반, Validation Accuracy는 80% 초반대의 성능을 보여주며 오히려 더 낮아진 성능을 보여준다.

### 악성코드 탐지 (최종)

1차시도에서 발생한 데이터 부족으로 인한 Overfitting, 2차시도에서 발생한 낮은 정확도를 해결하기 위한 방법으로 원본 데이터 추가 수집이라는 방법을 택했다. KISA의 악성코드 파일들을 조교님의 도움을 받아 구할 수 있었으며, 이를 통해서 최종적으로 22500개의 데이터를 이용해 훈련을 진행하였다. 그림4는 최종 모델의 구조이며 그림5는 각각 데이터 개수가 15000개로 훈련했을 때와 22500개로 훈련했을 때의 결과이다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 악성코드 탐지(최종) 모델 요약

|  |  |
| --- | --- |
|  |  |
| 15000개 | 22500개 |

그림 악성코드 탐지(최종) 데이터 개수 별 결과

학습 결과 Train Accuracy는 90% 후반, Validation Accuracy는 90% 초반대의 성능을 보여주며 데이터셋이 많아질 수록 그 정확도가 증가하는 것을 볼 수 있다. 최종적으로 모델을 통해 예측을 수행하였을 경우의 정확도는 표1과 같다.

|  |  |
| --- | --- |
| 일반 파일에 대한 예측 정확도 | 89% |
| 악성코드에 대한 예측 정확도 | 95% |
| 복합 파일에 대한 예측 정확도 | 91% |

표 악성코드 탐지 최종 모델 성능

## 악성코드 분류

### 분류 종류

CNN을 통한 악성코드 분류의 경우 데이터는 약 230,000개를 활용하였으며 이는 다음과 같은 16개의 클래스로 이루어져 있다. 최종적으로는 굵게 표시한 10개의 클래스로 분류가 되도록 학습하였다.

* **Backdoor**
* **Constructor**
* **Exploit**
* **Flooder**
* **HackTool**
* **Hoax**
* **RootKit**
* **Trojan**
* **Virus**
* **Worm**
* DoS
* Sniffer
* Spoofer
* SpamTool
* Virtool
* Packed

### 악성코드 분류 (1차시도)

악성코드 분류에 활용한 16개의 클래스는 불균형하게 이루어져있었다. Trojan과 Backdoor 2개의 클래스에 비정상적으로 많은 데이터들이 몰려 있었다. 이는 학습에 영향을 미쳤으며, 그 결과는 그림6과 같다.

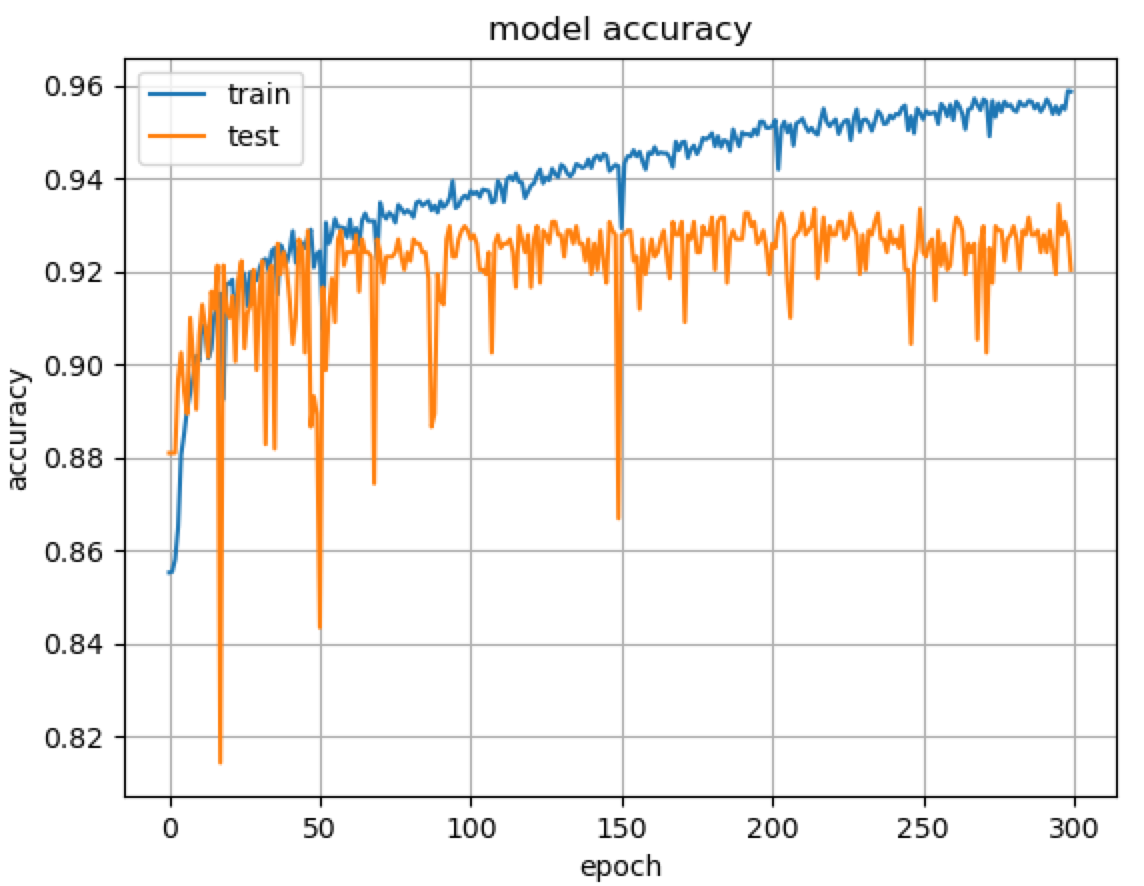


그림 악성코드 분류(1차시도) 결과

학습 결과 Train Accuracy는 90% 후반, Validation Accuracy는 90% 초반대의 성능을 보여준다. 매우 좋은 성능을 내는 것으로 보이지만 실제적으로 데이터가 많은 Trojan과 Backdoor에서만 준수한 성능을 낸다. 다른 클래스는 예측하였을 때는 그 정확도가 좋지 못했다.

### 악성코드 분류 (2차시도)

데이터 편향을 해결하기 위해서 2차시도에는 각 클래스의 데이터 수에 따라서 가중치를 조절하는 Class Weight를 적용하였다. 그림7은 그 결과이다.

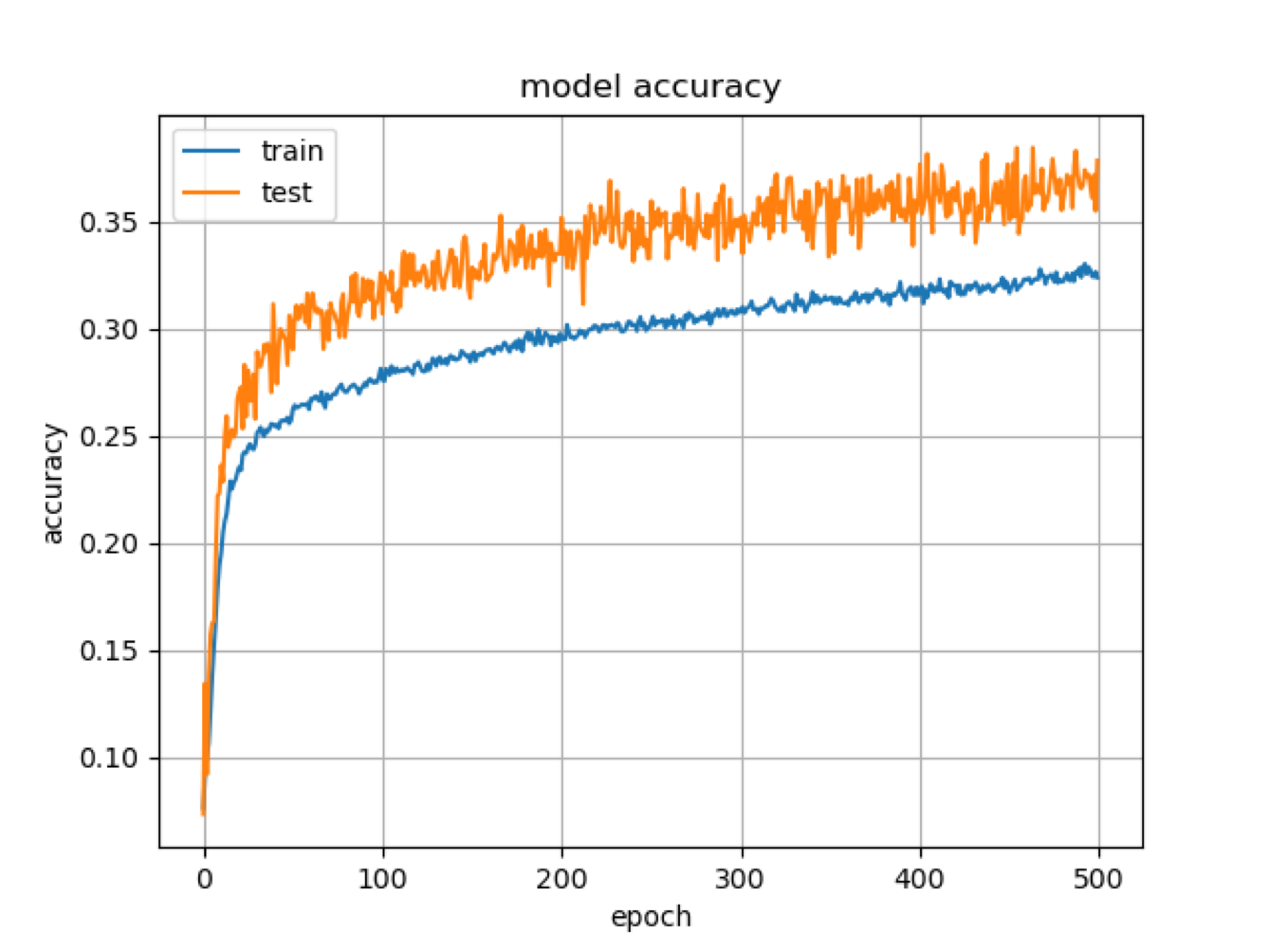


그림 악성코드 분류(2차시도) 결과

학습 결과 Train Accuracy는 35%대, Validation Accuracy는 30% 초반대의 성능을 보여준다. 예상과는 다르게 매우 좋지 않은 성능을 보여주었으며 데이터가 너무 편향 되어있기에 발생한 결과로 추측하였다.

### 악성코드 분류 (3차시도)

3번째 시도에는 색다른 방법으로 접근해 보기로 했다. Incremental Coordinate 이미지화 기법을 활용하기로 했다. 이 방식은 악성코드를 2byte씩 읽어와 1byte씩 (0~255, 0~255)의 2차원 배열 x, y축에 맵핑하는 방식이다. 이후에 2차원 배열의 가장 큰 값으로 정규화를 진행한다. 최종적으로 만들어진 2차원 배열을 256 \* 256크기의 그레이스케일 이미지로 변환하여 분석한다. 그림은 Incremental Coordinate 기법을 활용하여 만들어진 이미지 샘플들이다.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Backdoor | Trojan | Worm |

그림 Incremental Coordinate 기법 샘플 이미지

학습 결과 Train Accuracy는 70%대, Validation Accuracy는 70% 초반대의 성능을 보여준다. 이 방식도 목표로 하는 성능을 내지는 못했다.

### 악성코드 분류 (최종)

악성코드 분류 1차, 2차시도의 클래스 내 데이터 편향 문제를 해결하기 위해서 자체적인 전처리 작업을 수행하기로 하였다. 우선 17개의 클래스 중 데이터가 적은 클래스를 제거하였다. 이후에 남은 10개의 클래스내에서 잡음이 많은 데이터들을 걸러내는 등 각각의 클래스가 비슷한 개수의 데이터를 가질 수 있도록 데이터를 균등화하였다.

이후에 전처리 작업으로 만들어진 데이터들을 잘 학습시키도록 모델의 가중치 및 레이어를 변경 및 추가하며 학습을 진행하였다. 그림9은 최종 모델의 구조이며, 그림10은 최종 모델의 정확도 및 예측 수행 결과이다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 악성코드 탐지 (최종) 모델 요약

|  |  |
| --- | --- |
|  | 텍스트이(가) 표시된 사진  자동 생성된 설명 |
| 모델 정확도 | 예측 수행 결과 |

그림 악성코드 탐지(최종) 결과

학습 결과 Train Accuracy는 87.5%대, Validation Accuracy는 82.5% 초반대의 성능을 보여준다. 예측을 수행하였을 때는 약 85%의 정확도를 보여준다.

# 결론

프로젝트 수행을 통해 나온 결론은 다음과 같다. CNN을 통한 인공지능 악성코드 정적 분석은 가능하다. 탐지와 분류 모두 가능 하였으며 두 경우 모두 85%이상의 준수한 성능을 보여주었다.

다만, 목표로 하였던 90% 후반대의 성능을 기록하지는 못하였으며 이를 달성하기 위해서는 더 충분한 데이터셋, 편향되지 않은 데이터셋 그리고 더 확실한 전처리 과정이 필요할 것으로 보인다.

다양한 문서를 참고하여 모델을 수정하였지만, 해당 방식들을 따라하는 것만으로는 준수한 성능을 내지 못했다. 이를 해결 및 차별화된 프로젝트를 수행하기 위해서 팀원들이 모여서 자체적인 전처리 작업을 수행하였다. 데이터들을 세분화, 통합, 제거하는 등 다양한 전처리 방식을 시도하는 과정에서 모델의 성능을 좋아지는 것을 확인 할 수 있었다.

# 참고문서

<https://lsjsj92.tistory.com/355>

<https://epicarts.tistory.com/81>

<https://ccurity.tistory.com/369>

[https://towardsdatascience.com/악성코드-classification-using-convolutional-neural-networks-step-by-step-tutorial-a3e8d97122f](https://towardsdatascience.com/malware-classification-using-convolutional-neural-networks-step-by-step-tutorial-a3e8d97122f)

https://scienceon.kisti.re.kr/srch/selectPORSrchArticle.do?cn=JAKO201835372349734&dbt=NART