딥러닝을 통한 S-PRESENT 알려진 평문 공격 개선

임세진

https://youtu.be/adFDIwLUbuw

HSU 한성대학교 HANSUNG UNIVERSITY

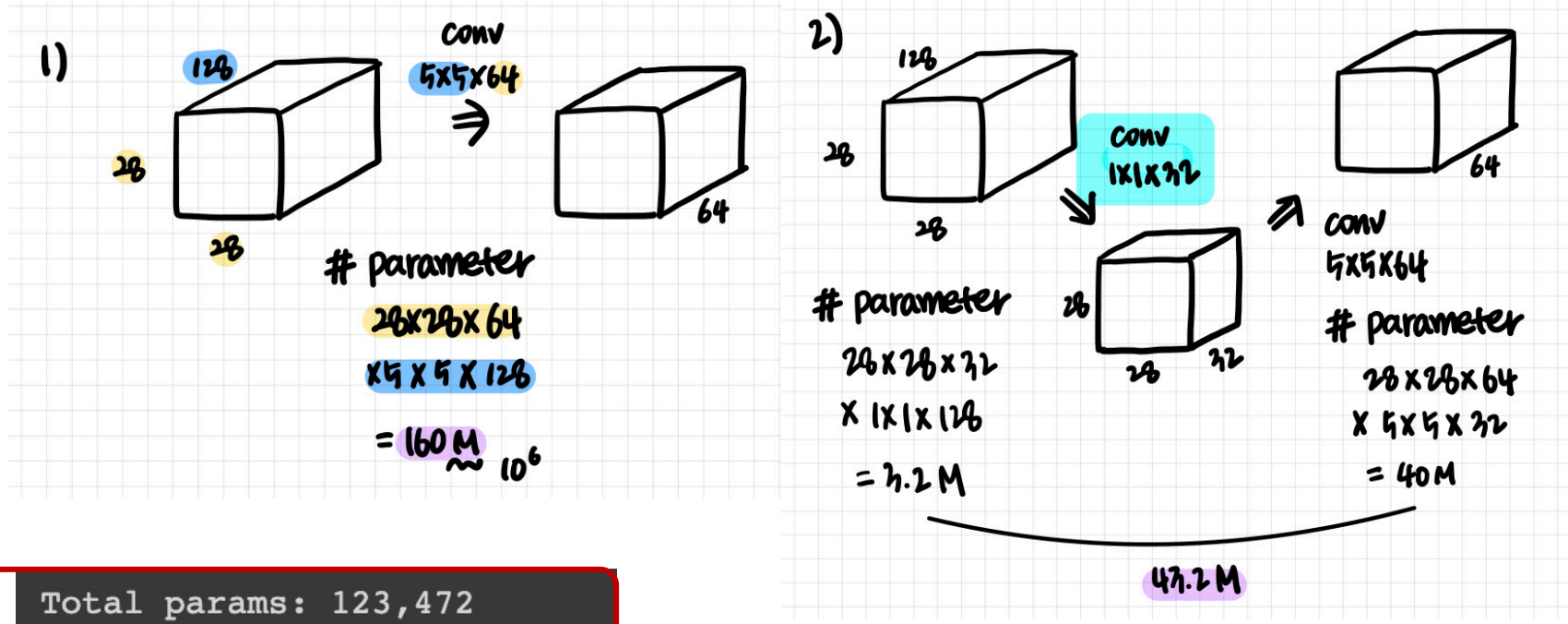
CryptoCraft LAB

Contents

CryptoCraft LAB

01. GoogLeNet

- GoogLeNet 논문에서 제안한 1x1 컨볼루션(kernel = 1)을 개선 모델에 적용함
- Parameter 수 = (입력데이터의 너비 x 높이 x 채널 수) x (필터의 너비 x 높이 x 채널 수)
- 1x1 컨볼루션
  - 네트워크에 비선형성을 더해줌
  - 채널 수 조절 가능
  - 연산량 감소




```
Total params: 1,057,360
Trainable params: 1,055,536
Non-trainable params: 1,824
```

Kernel = 3

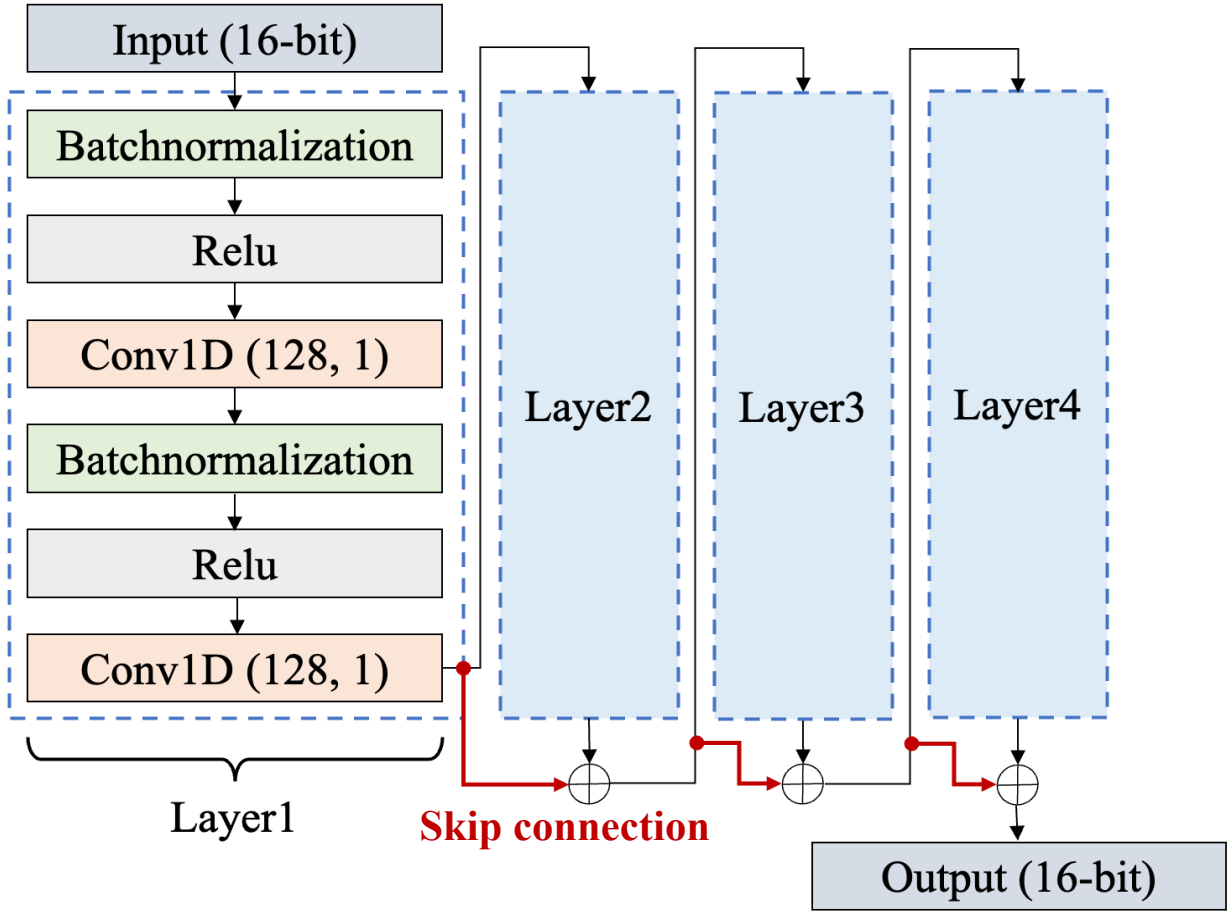
```
Total params: 123,472
Trainable params: 121,648
Non-trainable params: 1,824
```

Kernel = 1

02. 모델 개선

개선사항

- MLP 모델은 그대로, 추가된 모델도 구조는 이전과 동일
- Batch size 32 -> 128로 변경
- Conv1D는 최적의 Kernel size 탐색 → 변경
- Conv2D 추가
  - 최적의 Kernel size
  - parameter의 수가 적은 경우



02. 모델 개선

CNN1D / CNN2D 모델

- Conv1D (128, kernel, padding='same') → Conv2D (128, kernel, padding='same')로 바꾸면 2D 모델 구조임
- 입력 데이터 차원 변경 필요

```
kernel = 11

x = tf.keras.layers.BatchNormalization()(inputs)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Conv1D(128,kernel,padding='same')(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Conv1D(128,kernel,padding='same')(x)

skip_connection = x
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Conv1D(128,kernel,padding='same')(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Conv1D(128,kernel,padding='same')(x)
x = tf.keras.layers.add([x, skip_connection])
```

```
skip_connection = x
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Conv1D(128,kernel,padding='same')(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Conv1D(128,kernel,padding='same')(x)
x = tf.keras.layers.add([x, skip_connection])

skip_connection = x
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Conv1D(128,kernel,padding='same')(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Conv1D(128,kernel,padding='same')(x)
x = tf.keras.layers.add([x, skip_connection])

outputs = tf.keras.layers.Dense(16)(x)
```

03. 성능 평가 및 분석

- MLP보다 CNN 모델의 정확도가 평균적으로 높음
- Key 공간을 8-9bit로 설정했을 때는 CNN1D(k=11)과 CNN2D(k=3)이 좋은 성능을 보임

Model	Key	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th	15th	16th	AVG
CNN 1D (k=11)	8-bit	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.83	0.82	0.83	0.8	0.8	0.8	0.79	0.8	0.9
	9-bit	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.71	0.71	0.72	0.69	0.69	0.7	0.68	0.66	0.83
CNN 2D (k=3)	8-bit	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.83	0.82	0.83	0.79	0.8	0.81	0.79	0.79	0.9
	9-bit	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.71	0.71	0.72	0.72	0.7	0.7	0.7	0.68	0.68	0.83
CNN 2D (k=1)	8-bit	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	0.79	0.81	0.77	0.77	0.71	0.76	0.75	0.89
	9-bit	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.69	0.7	0.7	0.71	0.68	0.67	0.65	0.66	0.65	0.82
MLP	8-bit	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.79	0.78	0.79	0.74	0.58	0.76	0.74	0.73	0.87
	9-bit	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.59	0.6	0.62	0.61	0.53	0.54	0.59	0.55	0.54	0.76

03. 성능 평가 및 분석

- CNN 1D(kernel=1)를 사용했을 때 전체 16bit key 중 12bit까지 공격 성공

→ 커널 최적화를 수행한 CNN 1D, 2D 모델에 대해 16bit key 중 14bit까지 공격 성공

- 9, 10, 11bit는 취약, 13bit는 안전

→ 9, 10, 11bit도 취약한 편이지만, 7bit가 가장 취약 (14bit까지 공격에 성공하면서 7bit의 공격확률에 대한 데이터가 많이 생김)

→ 5bit가 가장 안전 (이전에는 MLP의 경우, 5bit의 공격확률을 확인할 수 없었음)

Model	Key	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th	15th	16th	AVG
CNN 1D (k=11)	10-bit	1.0	1.0	1.0	1.0	1.0	1.0	0.62	0.62	0.63	0.63	0.63	0.61	0.6	0.57	0.59	0.59	0.76
	11-bit	1.0	1.0	1.0	1.0	1.0	0.55	0.58	0.55	0.56	0.56	0.56	0.53	0.52	0.54	0.53	0.53	0.69
	12-bit	1.0	1.0	1.0	1.0	0.5	0.53	0.56	0.52	0.54	0.54	0.54	0.51	0.51	0.51	0.51	0.51	0.64
	13-bit	1.0	1.0	1.0	0.53	0.5	0.52	0.55	0.51	0.52	0.53	0.53	0.5	0.5	0.5	0.51	0.51	0.61
	14-bit	1.0	1.0	0.52	0.51	0.5	0.51	0.53	0.52	0.51	0.51	0.51	0.5	0.5	0.5	0.51	0.5	0.57
CNN 2D (k=3)	10-bit	1.0	1.0	1.0	1.0	1.0	1.0	0.62	0.61	0.63	0.63	0.63	0.61	0.59	0.55	0.59	0.58	0.75
	11-bit	1.0	1.0	1.0	1.0	1.0	0.55	0.57	0.54	0.55	0.55	0.55	0.52	0.52	0.52	0.53	0.53	0.68
	12-bit	1.0	1.0	1.0	1.0	0.5	0.5	0.56	0.53	0.53	0.53	0.54	0.51	0.51	0.51	0.51	0.51	0.64
	13-bit	1.0	1.0	1.0	0.53	0.5	0.52	0.55	0.52	0.52	0.52	0.53	0.5	0.51	0.51	0.51	0.51	0.61
	14-bit	1.0	1.0	0.52	0.52	0.5	0.51	0.53	0.51	0.51	0.51	0.51	0.5	0.5	0.5	0.51	0.51	0.57
CNN 2D (k=1)	10-bit	1.0	1.0	1.0	1.0	1.0	1.0	0.62	0.62	0.63	0.63	0.63	0.61	0.6	0.57	0.59	0.59	0.76
	11-bit	1.0	1.0	1.0	1.0	1.0	0.55	0.58	0.55	0.56	0.56	0.56	0.53	0.52	0.54	0.53	0.53	0.69
	12-bit	1.0	1.0	1.0	1.0	0.5	0.53	0.56	0.52	0.54	0.54	0.54	0.51	0.51	0.51	0.51	0.51	0.64
	13-bit	1.0	1.0	1.0	0.53	0.5	0.52	0.55	0.51	0.52	0.53	0.53	0.5	0.5	0.5	0.5	0.5	0.61
	14-bit	1.0	1.0	0.52	0.51	0.5	0.51	0.53	0.52	0.51	0.51	0.51	0.5	0.5	0.5	0.51	0.5	0.57
MLP	10-bit	1.0	1.0	1.0	1.0	1.0	1.0	0.57	0.54	0.56	0.57	0.56	0.51	0.52	0.52	0.52	0.52	0.71
	11-bit	1.0	1.0	1.0	1.0	1.0	0.52	0.56	0.51	0.53	0.52	0.53	0.51	0.5	0.5	0.5	0.5	0.67

감사합니다.