Proceedings of the 2019 36th International Conference on Machine Learning(ICML) (Vol. 97, pp. 812–821)

Low Latency Privacy Preserving Inference

Alon Brutzkus (Microsoft Research and Tel Aviv University, Israel)
Oren Elisha (Microsoft, Israel [3]Microsoft Research, Israel)
Correspondence to: Ran Gilad- Bachrach

1. Introduction

- **Private Neural-Networks Inference**
  - **머신러닝에서 프라이버시 문제 중요성 증가**
    - Multi-Party Computation (MPC)
    - HW enclaves
    - Intel Software Guard Extension (SGX)
    - **Homomorphic Encryption (HE)**
  - **데이터의 프라이버시를 보호하기 위한 접근 방식**
    - 데이터를 prediction service에 보내기 전에 **암호화**
    - 원본 데이터에 접근하지 않고도 암호화된 데이터를 기반으로 예측 수행

1. Introduction

- **기존 방법의 한계 및 제안**

CryptoNets

- 장점) MNIST를 이용한 예측 시 99%의 정확도

- 단점1) **Latency** - 205초
- 단점2) **Network Width** 제한
- 단점3) **Deep Network** 적용 불가

1. Introduction

- **기존 방법의 한계 및 제안**

CryptoNets

- 장점) MNIST를 이용한 예측 시 99%의 정확도
- **단점1) Latency - 205초**
- **단점2) Network Width 제한**
- 단점3) Deep Network 적용 불가

LoLa (Low-Latency CryptoNets)

- MNIST 신경망을 **2.2초** 만에 평가 가능
- 연산 시 **다양한 데이터 표현 방식** 도입
- 암호화된 메시지 처리 시 필요한 **메모리 사용량 감소**
  - CryptoNets: CIFAR-10 처리에 100GB 이상 필요
    → LoLa는 단 **몇 GB**만 사용

1. Introduction

● 기존 방법의 한계 및 제안

CryptoNets

- 장점) MNIST를 이용한 예측 시 99%의 정확도
- **단점1) Latency - 205초**
- **단점2) Network Width 제한**
- **단점3) Deep Network 적용 불가**

LoLa (Low-Latency CryptoNets)

- MNIST 신경망을 **2.2초** 만에 평가 가능
- 연산 시 **다양한 데이터 표현 방식** 도입
- 암호화된 메시지 처리 시 필요한 **메모리 사용량 감소**
  - CryptoNets: CIFAR-10 처리에 100GB 이상 필요
    → LoLa는 단 **몇 GB**만 사용

Transfer Learning - Deep Feature

- 데이터를 사전 처리해 **중요한 특징**만 추출
- CalTech-101 데이터셋 실험 결과
  - 정확도 81.6% / 0.16s

2. Related Work

- **Private Predictions 연구 동향**

① **CryptoNets (Dowlin et al., 2016)**: 동형 암호 기반의 신경망 예측을 제안.

- 장점: 높은 정확도(98.95%)

- 단점: **205초의 latency**

② **Bourse et al. (2017)**: 빠른 부트스트래핑을 지원하는 HE 스킴 사용을 사용했으나, 정확도가 떨어짐.

- 레이어 추가 시 선형 비용만 발생하지만, **연산 속도가 느리고 정확도가 낮음**.

③ **Boemer et al. (2018)**: Intel nGraph **컴파일러 기반 HE 확장**을 제안했지만, CryptoNets보다 느린 latency을 보여줌.

④ **Badawi et al. (2018)**: **GPU**를 활용하여 CryptoNets 가속화.

- 그러나 CPU만 사용하는 LoLa가 6배 이상 빠름

⑤ **Sanyal et al. (2018)**: 암호화 매개변수의 데이터 유출 가능성을 줄였으나, 솔루션 속도는 상대적으로 **느림**.

⑥ **기타 연구**: Chameleon 시스템(**MPC** 사용), **하드웨어 기반 접근법**(Tramer & Boneh, 2018) 제안. 속도는 빠르지만 보안 수준이 낮음

2. Related Work

- **기존 연구 대비 LoLa의 성능**

[표 1] LoLa 실험 결과

	MNIST	CIFAR-10
Speed	2.2s	730s
Accuracy	98.95%	74.1%

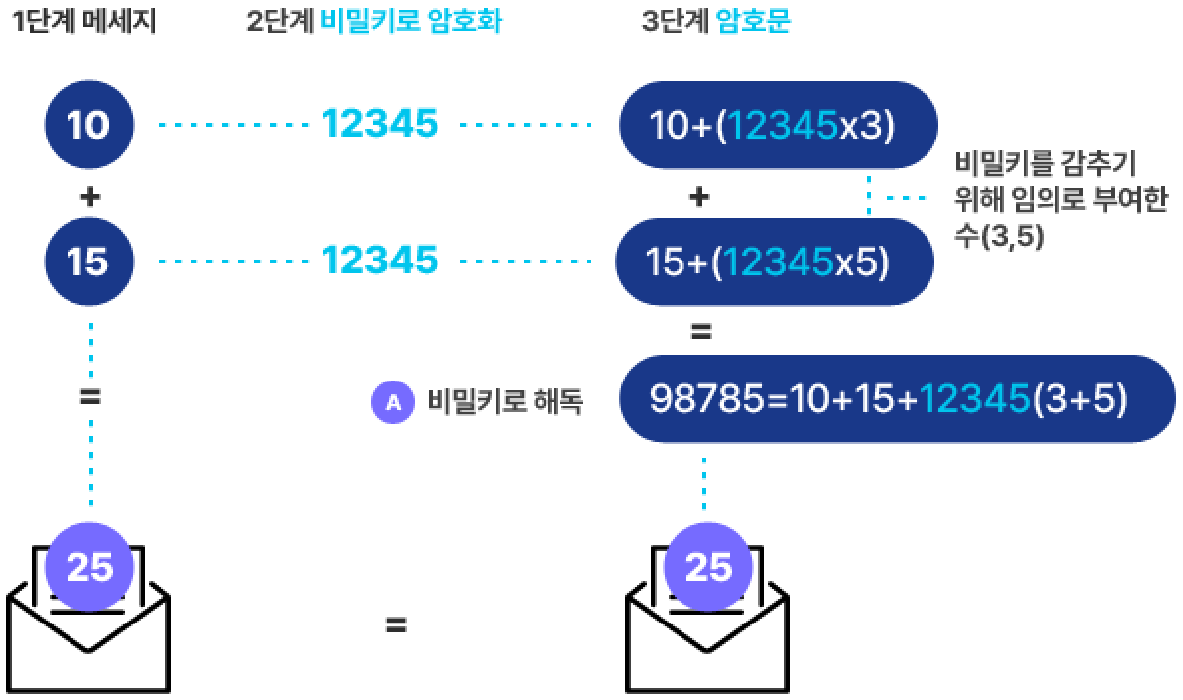
[표 2] 성능 비교

	LoLa	CryptoNets	Faster-CryptoNets	HCNN
Speed	2.2s	730s	39.1s	**14.1s**
Accuracy	98.95%	74.1%	98.7%	**99%**

3.1 Background - Cryptography

- **Homomorphic Encryption (HE)**
  - 데이터의 복호화 없이 암호문간 연산이 가능한 암호화 기법
    - **주요 연산**
      - 덧셈: $\mathbb{D}(\mathbb{E}(x_1) \oplus \mathbb{E}(x_2)) = x_1 + x_2$
      - 곱셈: $\mathbb{D}(\mathbb{E}(x_1) \otimes \mathbb{E}(x_2)) = x_1 \times x_2$
        - $\mathbb{E}$: 암호화 함수(Encryption)

          $\mathbb{D}$: 복호화 함수(Decryption)



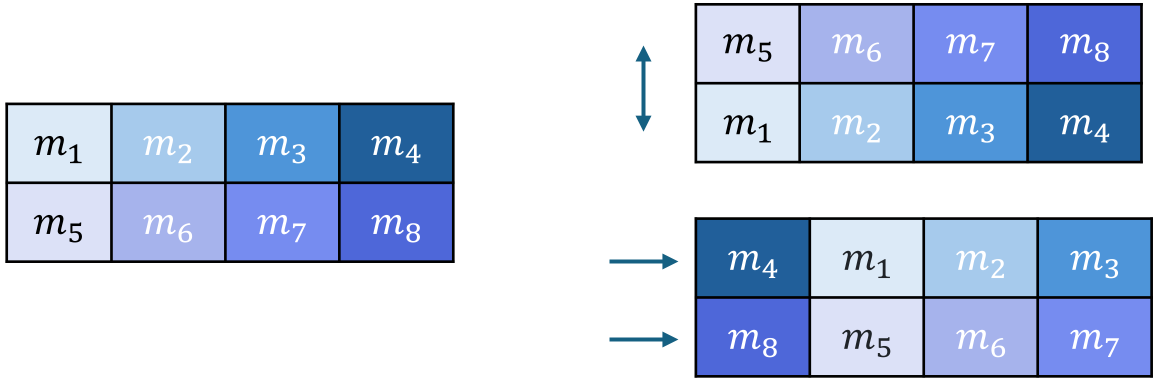
[그림 1] 동형암호의 원리

3.1 Background - Cryptography

- **BFV scheme (Brakerski/Fan-Vercauteren)**
  - 동형암호 스킴 중 하나로 정수 연산을 지원함
    - **다항식 링** $\boldsymbol{\mathcal{R}} = \mathbb{Z}_{\boldsymbol{p}}[\boldsymbol{x}]\,/\,(\boldsymbol{x}^{\boldsymbol{n}} + \mathbf{1})$ **기반으로 동작**
      - $\mathbb{Z}_p$: 정수 집합 $\mathbb{Z}$ 를 소수 p 로 나눈 나머지들의 집합
        - *ex)* $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ 즉, $\mathbb{Z}_p = 0, 1, \dots, \mathrm{p}-1 \pmod{\mathrm{p}}$
      - $\mathbb{Z}_p[x]$: 계수가 $\mathbb{Z}_p$ 에서 정의된 다항식 집합
        - ex) p=5일 때 $2x^2 + 3x + 1 \in \mathbb{Z}_5[x]$
      - $(x^n + 1)$: $\mathbb{Z}_p[x]$ 의 차수를 $(n-1)$로 제한
        - n : 다항식의 최대 차수를 결정하는 값으로, 보통 2의 거듭제곱으로 설정됨

3.1 Background - Cryptography

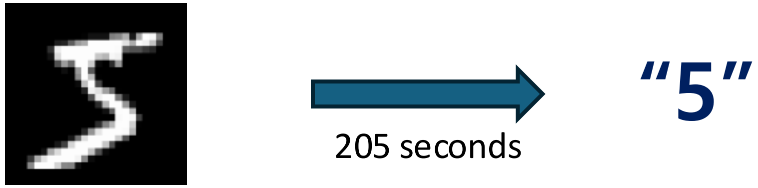
- **BFV 스킴에서의 Rotation Operation**
  - **Rotation**
    - **데이터 간의 순서가 특정 방향으로 cyclic 하게 밀리는 구조**
      - $m = [m_1, m_2, \dots, m_n]$ 의 k 칸 회전: 벡터의 i번째 요소 m_i 를 $((i+k)\ mod\ n)$ 으로 이동
        - ex) $2 \times n/2$ 크기의 행렬로 표현된 메시지의 rotation 예시
          - 행 회전(Row Rotation): 행 간 순서를 교환
          - 열 회전(Column Rotation): 열을 순환 이동



[그림 2] rotation 예시

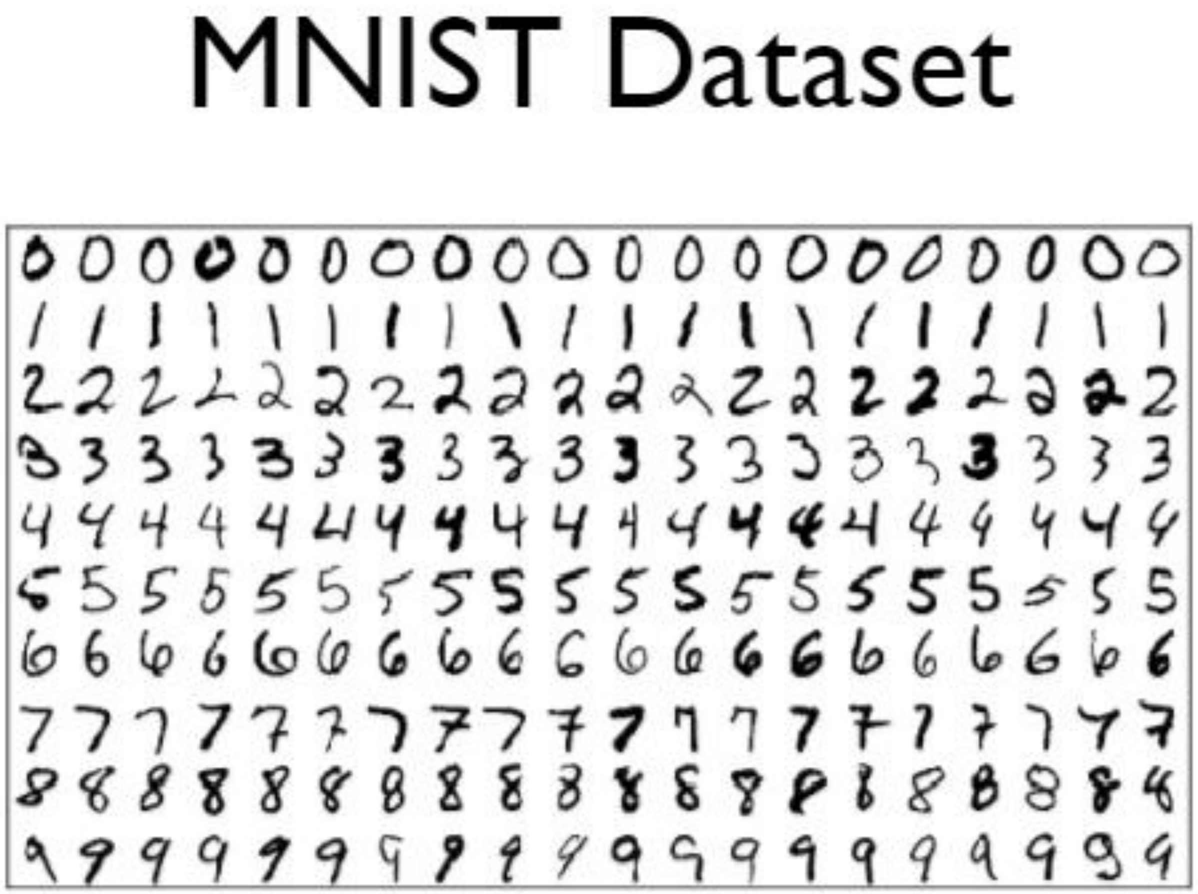
3. 2Background - CryptoNets

- **CryptoNets (Dowlin, 2016)**
  - **개요**
    - **동형암호**를 활용해 **암호화된 데이터로 신경망 예측**을 하는 프라이버시 보호 서비스
    - MNIST 데이터셋에서 **99%** 정확도, 단일 예측에 **205초** 소요
      - 시간당 약 **59,000개** 처리 가능

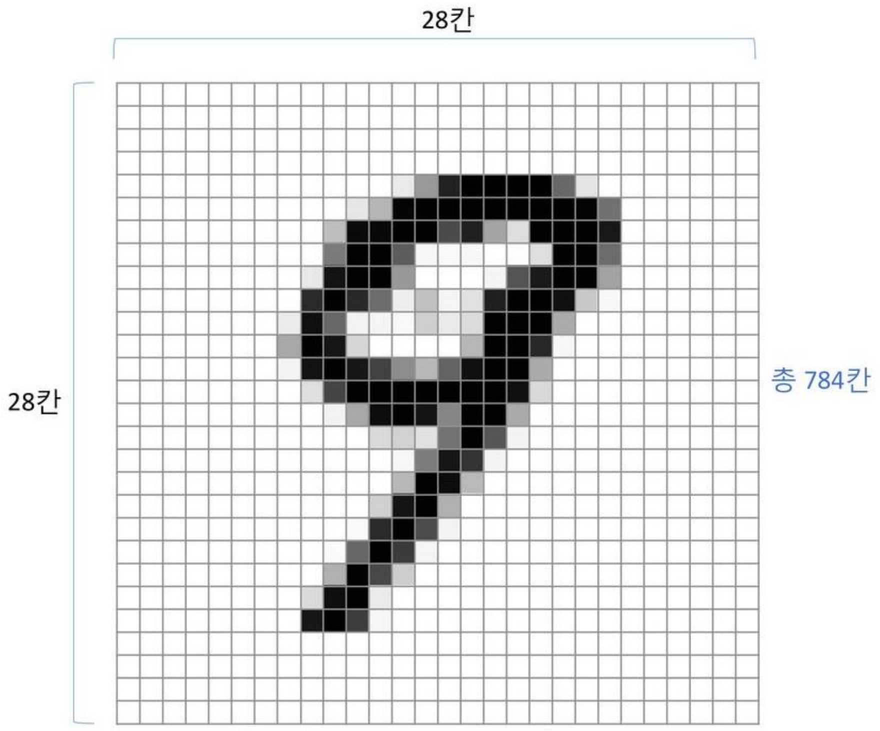


[그림 3] Single prediction example

3.2 Background – MNIST



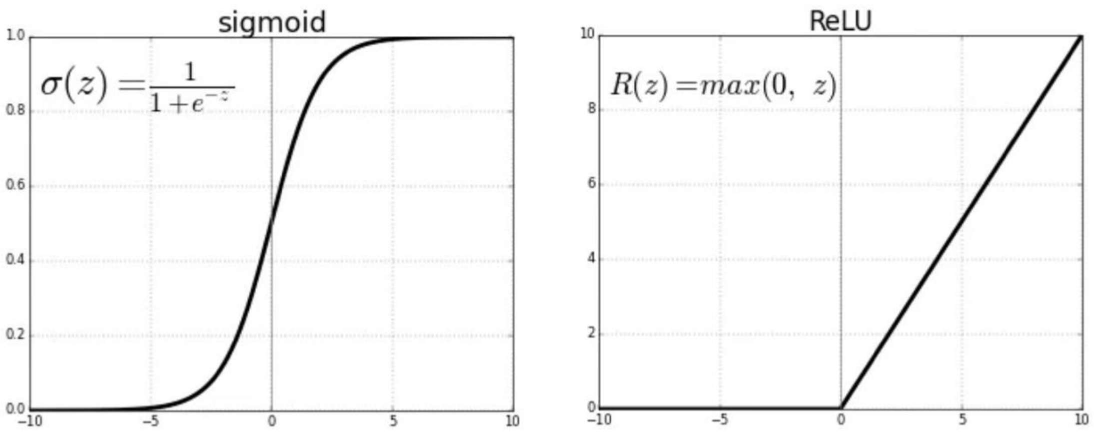
[그림 4] MNIST Dataset



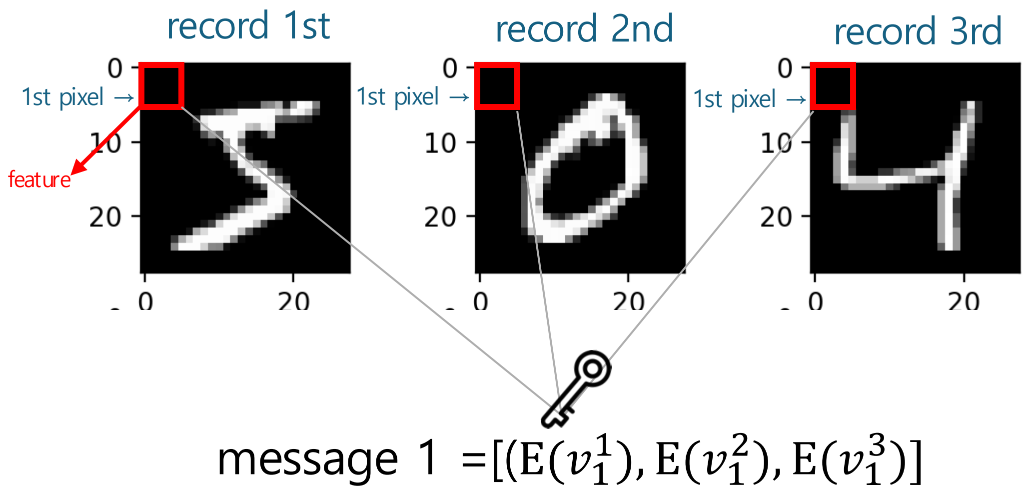
[그림 5] 개별 MNIST 이미지 예시

3.2 Background – CryptoNets

- **CryptoNets (Dowlin, 2016)**
  - **특징**
    - 비선형 활성화 함수를 제공하지 않아, 대신 **제곱 활성화(Square Activation)** 함수 사용
      - **비선형 활성화 함수**: ReLU, Sigmoid
    - 암호화된 **메시지를 벡터 구조로 사용**하여 여러 입력을 동시에 처리: v_j^i → **SIMD** (Single Instruction Multiple Data)



[그림 6] 활성화 함수 Sigmoid, ReLU



[그림 7] CryptoNets 메시지 구조 예시

3.2 Background – CryptoNets

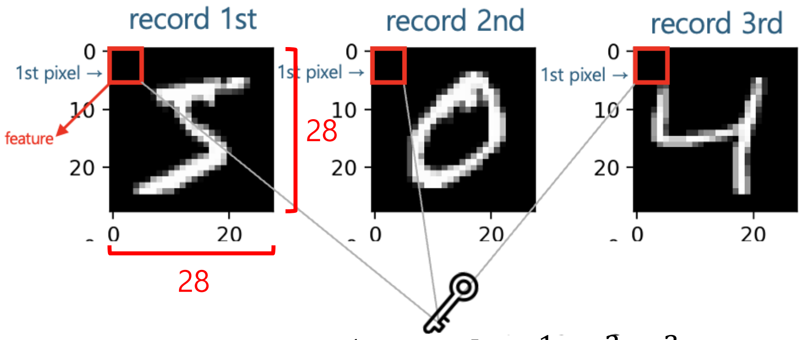
- **CryptoNets (Dowlin, 2016)**
  - **한계점**
    - **High latency**: 단일 예측에 많은 연산 필요
      - 각 feature이 별도의 메시지로 표현되므로 한 이미지 예측에 많은 연산이 필요
    - **메모리 병목 현상**
      - 다수의 메시지를 처리해야 하므로 메모리 사용량이 높아지고 병목 현상 발생
    - **Deep network 에서의 비효율성**: 곱셈 연산 시 노이즈 증가

4.1 LoLa

- **LoLa (Low-Latency CryptoNets)**
  - **주요 특징**
    - **다양한 표현 방식 사용**
      - 암호화된 메시지를 처리할 때, 다양한 형태로 데이터를 표현하고 계산
        - 이는 기존 CryptoNets가 각 픽셀(feature)을 별도의 메시지로 처리했던 방식을 개선하여, 메시지 표현 방식의 효율성을 높인 것
    - **Latency 및 메모리 사용량 개선**
      - 메시지를 더 효율적으로 인코딩하여 지연 시간을 단축하고, 메모리 사용량을 줄임

4.1 Encoding Message

- **Linear Classifier Example**
  - **CryptoNets의 연산 : d**
    - 입력 벡터 $v = [m_1, m_2, \dots, m_{784}]$와 가중치 벡터 $w = [w_1, w_2, \dots, w_{784}]$ 간의 **내적** 계산: $w \cdot v = w_1 \cdot m_1 + w_2 \cdot m_1 + \cdots + w_{784} \cdot m_{784}$
      - **d번**의 곱셈 및 덧셈 연산이 필요



message 1 : $m_1 = [\mathbb{E}(v_1^1,\ v_1^2,\ v_1^3)]$

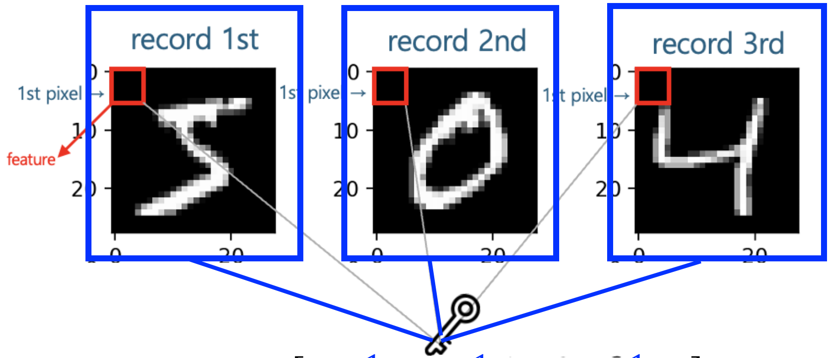
$\vdots$

message 784 : $m_{784} = [\mathbb{E}(v_{784}^1, v_{784}^2, v_{784}^3)]$

[그림 8] CryptoNets 메시지 수 예시

4.1 Encoding Message

- **Linear Classifier Example**
  - **LoLa의 연산 : log d**
    - 각 n번째 이미지의 전체 벡터를 하나의 메시지로 암호화: $m_n = [E(v_1^n), E(v_2^n), \ldots, E(v_{784}^n)]$



$$m_1 = [E(v_1^1), E(v_2^1), \ldots, E(v_{784}^1)]$$
$$\vdots$$
$$m_3 = [E(v_1^3), E(v_2^3), \ldots, E(v_{784}^3)]$$

[그림 9] LoLa 메시지 수 예시

4.1 Encoding Message

- **Linear Classifier Example**
  - **LoLa의 연산 : log d**
    - 가중치 벡터 $w = [w_1, w_2, \dots, w_{784}]$ 와 메시지 m_n 간의 내적 계산
      - $m_1' \Rightarrow m_1 \cdot w = [E(v_1^1 \cdot w_1) + E(v_2^1 \cdot w_2) + \cdots + E(v_{784}^1 \cdot w_{784})]$

        $m_2' \Rightarrow m_2 \cdot w = [E(v_1^2 \cdot w_1) + E(v_2^2 \cdot w_2) + \cdots + E(v_{784}^2 \cdot w_{784})]$

        $m_3' \Rightarrow m_3 \cdot w = [E(v_1^3 \cdot w_1) + E(v_2^3 \cdot w_2) + \cdots + E(v_{784}^3 \cdot w_{784})]$

4.1 Encoding Message

- **Linear Classifier Example**
  - **LoLa의 연산 : log d**
    - 가중치 벡터 $w = [w_1, w_2, \dots, w_{784}]$ 와 메시지 m_n 간의 내적 계산
      - $m'_1 \Rightarrow m_1 \cdot w = [E(v_1^1 \cdot w_1) + E(v_2^1 \cdot w_2) + \cdots + E(v_{784}^1 \cdot w_{784})]$

        $m'_2 \Rightarrow m_2 \cdot w = [E(v_1^2 \cdot w_1) + E(v_2^2 \cdot w_2) + \cdots + E(v_{784}^2 \cdot w_{784})]$

        $m'_3 \Rightarrow m_3 \cdot w = [E(v_1^3 \cdot w_1) + E(v_2^3 \cdot w_2) + \cdots + E(v_{784}^3 \cdot w_{784})]$
    - **Log d 번의 rotation 을 통한 덧셈 연산 필요**: log d = log 784 ≒ 10
      - ex) 벡터 $x = [x_1, x_2, x_3, x_4]$ 에서 네 벡터의 합을 구하는 경우
        - **1 칸 회전 후 원래 벡터와 더함**: $[x_2, x_3, x_4, x_1] \rightarrow [x_1 + x_2,\ x_2 + x_3,\ x_3 + x_4,\ x_4 + x_1]$
        - **2 칸 회전 후 원래 벡터와 더함**:

          $[x_3 + x_4,\ x_4 + x_1, x_1 + x_2,\ x_2 + x_3] \rightarrow [x_1 + x_2 + x_3 + x_4,\ x_2 + x_3 + x_4 + x_1,\ x_3 + x_4 + x_1 + x_2,\ x_4 + x_1 + x_2 + x_3]$
          - **최종 결과**: $x_1 + x_2 + x_3 + x_4$

4.2 LoLa

- **LoLa 메시지 표현 방식**
    - **데이터 표현 방식**이 연산의 효율성에 중요한 영향을 미침
    - **LoLa의 접근법**
        - 네트워크 각 **Layer**에 맞는 최적의 메시지 표현 방식 사용

            1) Dense representation

            2) Sparse representation

            3) Stacked representation

            4) Interleaved representation

            5) Convolution representation

4.3 Vector Representation

- **Vector Representation**
  - **벡터 표현 개요**
    - **입력 데이터(메시지)**가 연산에 사용되기 위해 **구조화**되는 방식을 정의

1) Dense Representation

- 벡터 v의 모든 요소를 **단일 메시지 m**으로 표현
  - $m = [v_1, v_2, v_3, v_4]$

2) Sparse Representation

- 각 벡터 항목을 **개별 메시지**로 저장
  - 길이가 k=4 인 벡터 $v = [v_1, v_2, v_3, v_4]$
    - $m_1 = [v_1, v_1, v_1, v_1]$, $m_2 = [v_2, v_2, v_2, v_2]$, $m_3 = [v_3, v_3, v_3, v_3]$, $m_4 = [v_4, v_4, v_4, v_4]$

4.3 Vector Representation

- **Vector Representation**

3) Stacked Representation

- 벡터 v를 하나의 메시지 m에 **순환 구조**로 여러 번 **복사**하여 포함하는 방식
  - $v = [v_1, v_2, v_3] \longrightarrow \lceil \log(3) \rceil = 2 \longrightarrow \mathrm{d} = 2^2$
    - $m = [v_1, v_2, v_3, v_1]$

4) Interleaved Representation

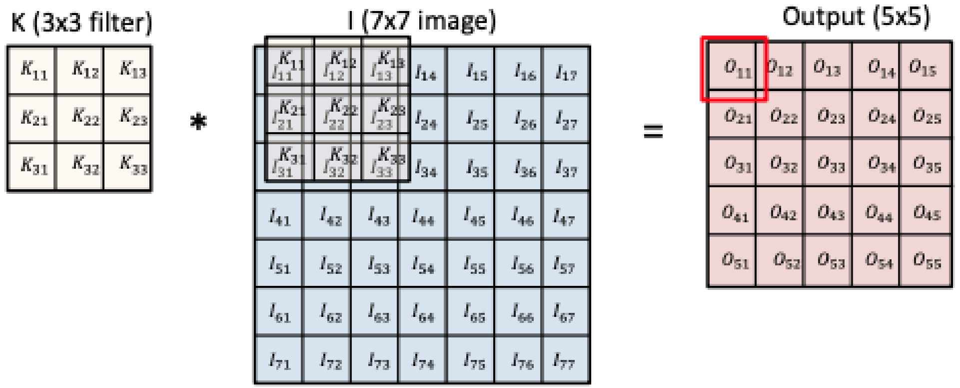
- **순열**을 사용해 벡터 요소들을 특정 순서로 **재배치**한 메시지 생성
- $[1, \ldots, n]$ 의 순열 σ 이용해 $m_{\sigma(i)} = v_i$로 설정
  - $v = [v_1, v_2, v_3, v_4], \sigma = [3, 1, 4, 2]$
    - $m_{\sigma(1)} = v_1 \longrightarrow m_3 = v_1$ / $m_{\sigma(2)} = v_3 \longrightarrow m_4 = v_3$ / $m_{\sigma(3)} = v_2 \longrightarrow m_1 = v_2$ / $m_{\sigma(4)} = v_4 \longrightarrow m_2 = v_4$
      - $\therefore m = [v_2, v_4, v_1, v_3]$

4.3 Vector Representation

Vector Representation

5) Convolution Representation

- **Convolution 연산**을 효율적으로 수행하기 위한 벡터 표현 방식
- Convolution의 선형 변환: $\sum_j \boldsymbol{w_j v_{\sigma_i}(j)}$
  - 가중치벡터 w · 입력 벡터 v
  - 입력 벡터 v 는 **윈도우 크기 r** 에 따라 r 개의 메시지 $m^1, m^2, \ldots, m^r$ 로 나눠 표현됨



[그림 10] Convolution 과정 예시

4.4 Matrix-Vector Multiplications

- **Matrix-Vector Multiplications**
  - **개요**
    - Matrix-Vector Multiplication은 신경망에서 중요한 연산으로, matrix는 학습된 가중치를, vector는 노드의 값을 나타냄
    - LoLa 방식은 **각 layer에 적합한 벡터 표현 방식**을 선택하여 효율적으로 연산 수행

$$\begin{pmatrix}1&1&2\\2&1&3\\1&4&2\end{pmatrix}\begin{pmatrix}3\\1\\2\end{pmatrix}=\begin{pmatrix}1\cdot3+1\cdot1+1\cdot2\\ \\ \\ \end{pmatrix}$$

First row,

$$\begin{pmatrix}1&1&2\\2&1&3\\1&4&2\end{pmatrix}\begin{pmatrix}3\\1\\2\end{pmatrix}=\begin{pmatrix}1\cdot3+1\cdot1+1\cdot2\\2\cdot3+1\cdot1+3\cdot2\\ \\ \end{pmatrix}$$

next row,

$$\begin{pmatrix}1&1&2\\2&1&3\\1&4&2\end{pmatrix}\begin{pmatrix}3\\1\\2\end{pmatrix}=\begin{pmatrix}1\cdot3+1\cdot1+1\cdot2\\2\cdot3+1\cdot1+3\cdot2\\1\cdot3+4\cdot1+2\cdot2\end{pmatrix}=\begin{pmatrix}6\\13\\11\end{pmatrix}$$

last row,
then do the addition.

[그림 11] Matrix-Vector Multiplication 예시

4.4 Matrix-Vector Multiplications

- **1) Dense Vector - Row Major**
  - 행렬-벡터 곱을 dot-product 연산으로 계산
    - 최종 출력 형태: Sparse vector

    - $W = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}$

    - $$\begin{matrix} r_1 = [1, 2, 3] \\ r_2 = [4, 5, 6] \\ r_3 = [7, 8, 9] \end{matrix} \cdot v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \Rightarrow \begin{matrix} V_1 = [14,14,14] \\ V_2 = [32,32,32] \\ V_3 = [50,50,50] \end{matrix}$$

4.4 Matrix-Vector Multiplications

- **2) Sparse Vector - Column Major**
  - 희소 벡터 v를 사용하면 각 성분을 **독립적**으로 계산할 수 있어 불필요한 계산을 줄이고 빠르게 행렬-벡터 곱셈 가능
    - 최종 출력 형태: Dense vector
    - $\boldsymbol{Wv} = \sum \boldsymbol{v_i c^i}$
    - $W = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$
    - $v_1 = [1, 1, 1]$

      $v_2 = [2, 2, 2]$

      $v_3 = [3, 3, 3]$

      $c^1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} c^2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} c^3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \Rightarrow \quad Wv = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix}$

4.4 Matrix-Vector Multiplications

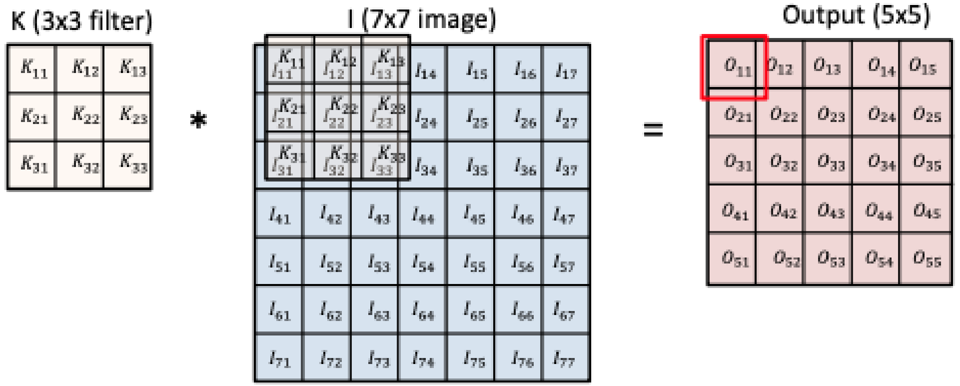
- **3) Stacked Vector - Row Major**
  - 벡터 v 를 복사하여 하나의 메시지 m 으로 저장해 데이터를 스택하고 회전 및 덧셈을 활용하는 방법
    - 최종 출력 형태: Interleaved vector
    - $W = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot v = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$
      - **1단계**: 벡터 복사 $m = (5, 6, 5, 6)$ 및 행렬 평탄화 $W' = (1, 2, 3, 4)$
      - **2단계**: 요소별 곱셈 $W'm = (1 \cdot 5,\ \ 2 \cdot 6,\ \ 3 \cdot 5,\ \ 4 \cdot 6)$
      - **3단계**: rotation 및 덧셈
        - $(1 \cdot 5,\ \ 2 \cdot 6,\ \ 3 \cdot 5,\ \ 4 \cdot 6) \rightarrow (2 \cdot 6,\ 3 \cdot 5,\ 4 \cdot 6,\ \ 1 \cdot 5)$
        - $(1 \cdot 5 + 2 \cdot 6,\ \ 2 \cdot 6 + 3 \cdot 5,\ \ 3 \cdot 5 + 4 \cdot 6,\ \ 4 \cdot 6 + 1 \cdot 5)$
      - **4단계**: 최종 내적 결과 추출 : $\begin{bmatrix} 1 \cdot 5 + \ 2 \cdot 6 \\ 3 \cdot 5 + 4 \cdot 6 \end{bmatrix}$
        - 첫 번째 행의 내적 결과: $1 \cdot 5 + \ 2 \cdot 6$ 두 번째 행의 내적 결과: $3 \cdot 5 + 4 \cdot 6$

4.4 Matrix-Vector Multiplications

- **4) Interleaved Vector - Row Major**
  - 기존 Dense 방식에서 행렬의 열을 재배열한 방식
    - 최종 출력 형태: Sparse vector
      - $W = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow W' = \begin{bmatrix} 1 & 3 & 2 \\ 4 & 6 & 5 \\ 7 & 9 & 8 \end{bmatrix}$
      - $$\begin{array}{l} r_1 = [1, 3, 2] \\ r_2 = [4, 6, 5] \\ r_3 = [7, 9, 8] \end{array} \cdot v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \Rightarrow \begin{array}{l} m_1 = [13,13,13] \\ m_2 = [31, 31, 31] \\ m_3 = [49, 49, 49] \end{array}$$

4.4 Matrix-Vector Multiplications

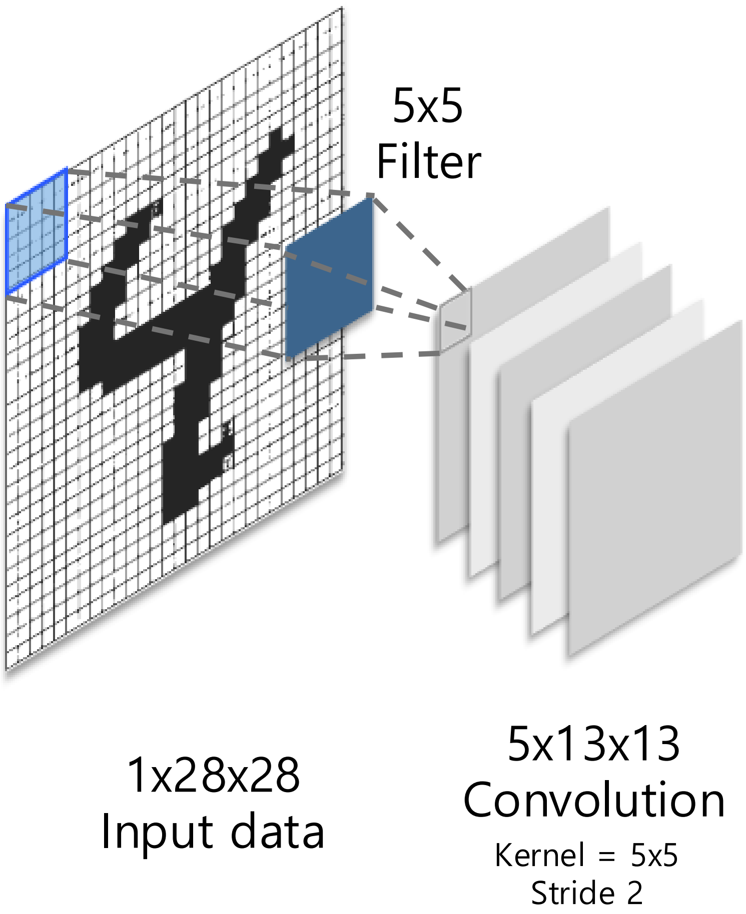
- **5) Convolution Vector - Row Major**
  - 데이터 벡터 v에 컨볼루션 필터를 적용해, 원하는 위치마다 선형 변환 결과를 계산 ⟶ r개의 메시지 생성
    - 최종 출력 형태: Dense vector



[그림 12] Convolution 과정 예시

5.1 LoLa - MNIST

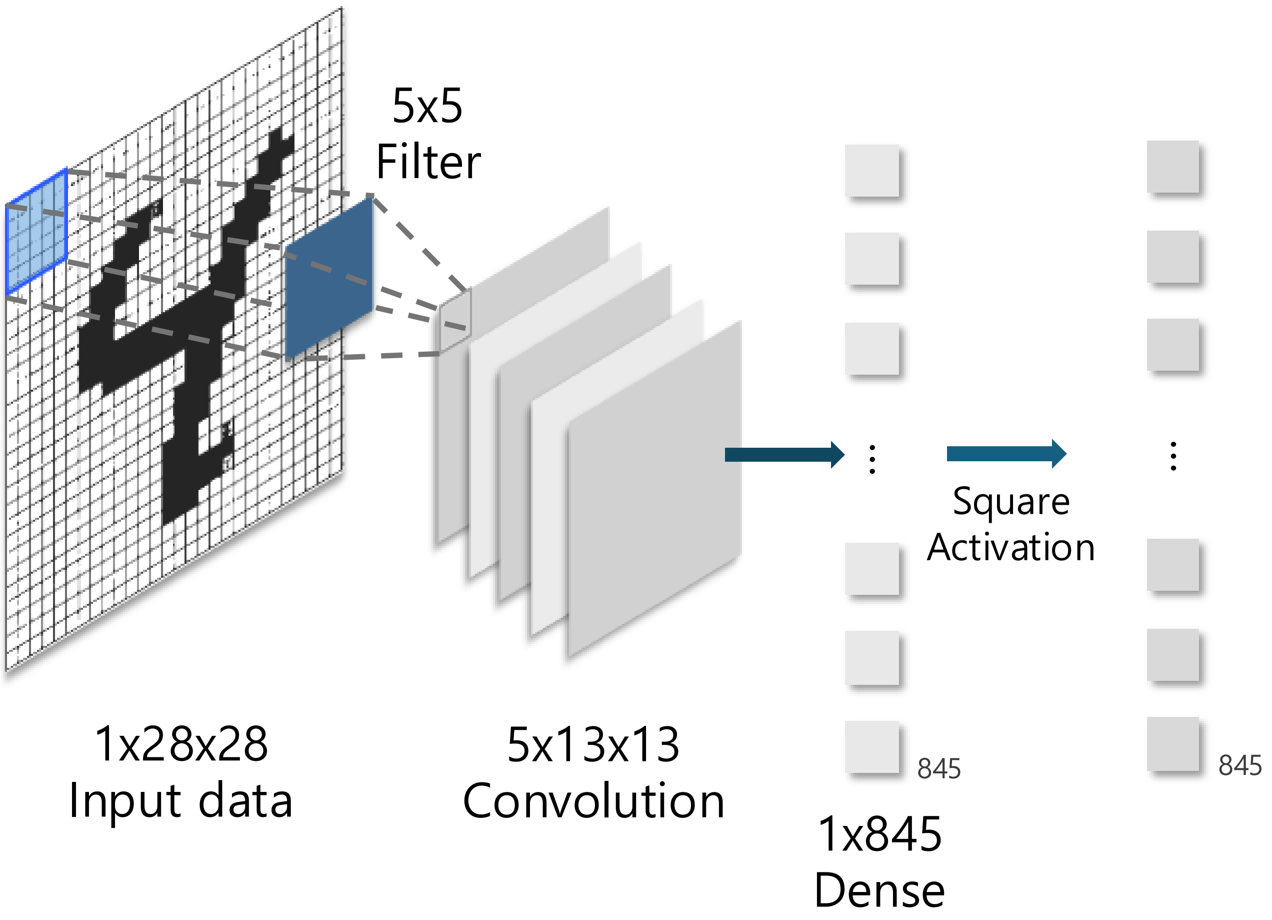
① Convolution vector – Row major multiplication



[그림 13] LoLa 아키텍처의 MNIST 분류 과정

5.1 LoLa - MNIST

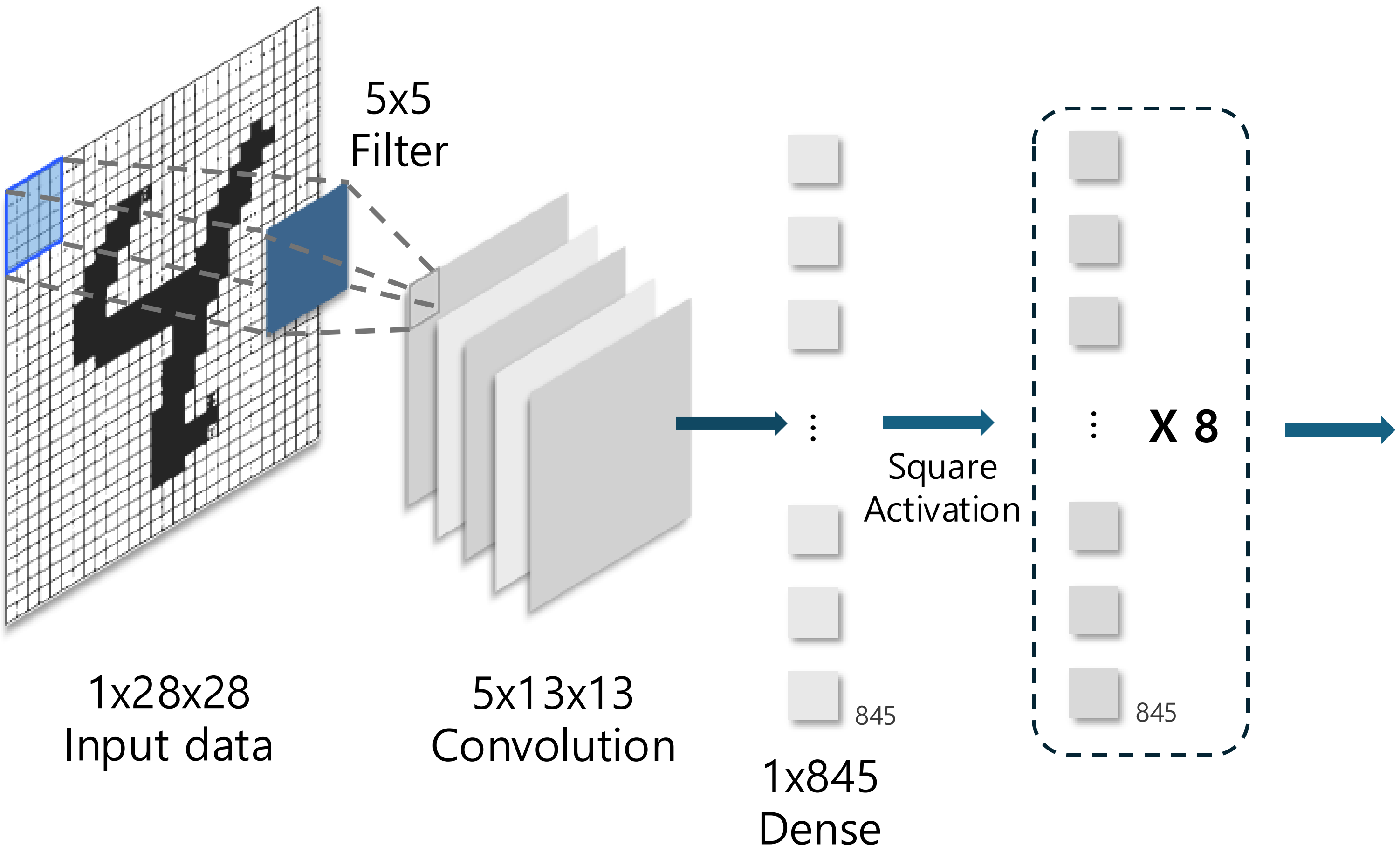
① Convolution vector – Row major multiplication



[그림 13] LoLa 아키텍처의 MNIST 분류 과정

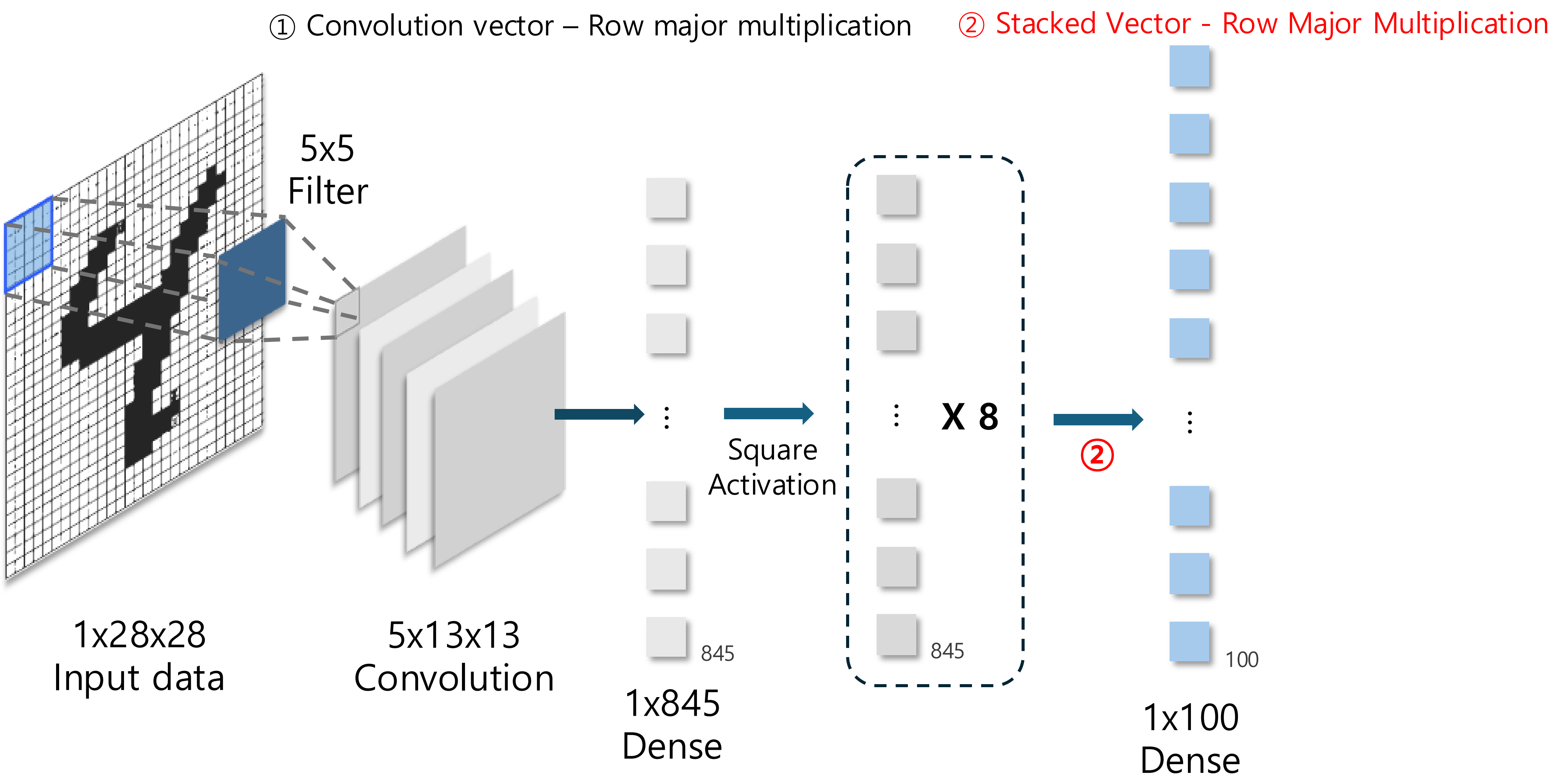
5.1 LoLa - MNIST

① Convolution vector – Row major multiplication



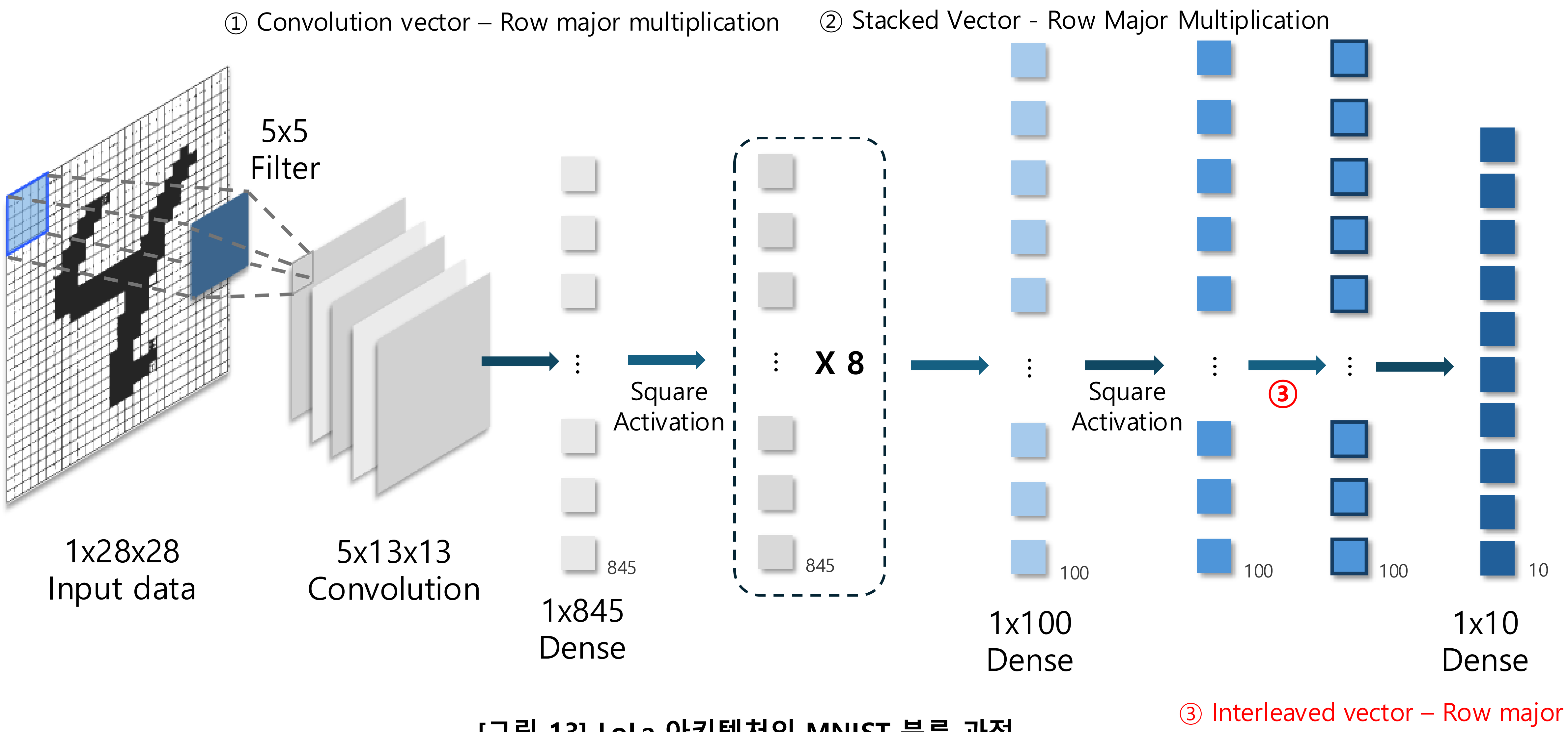
[그림 13] LoLa 아키텍처의 MNIST 분류 과정

4.5 LoLa - MNIST



[그림 13] LoLa 아키텍처의 MNIST 분류 과정

4.5 LoLa - MNIST



[그림 13] LoLa 아키텍처의 MNIST 분류 과정

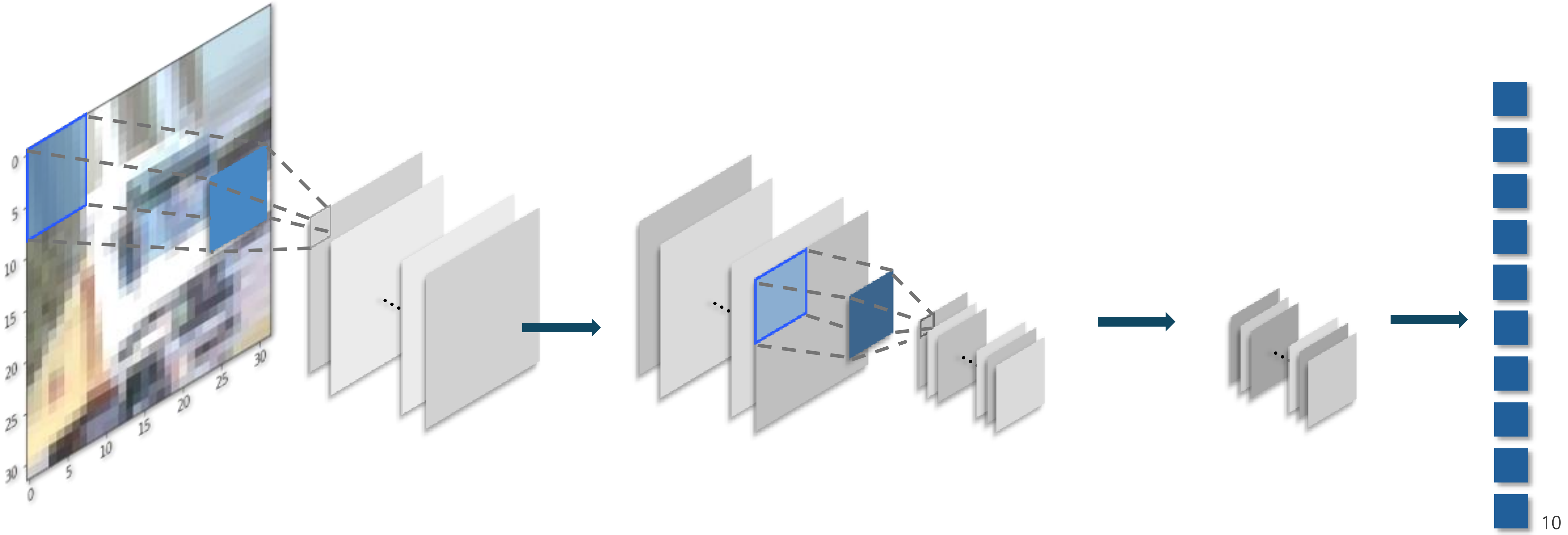
4.5 LoLa - MNIST

- **LoLa 추가 버전**
  - **LoLa-Dense,** LoLa-Small

[표 3] MNIST 연산 시 LoLa 버전 비교

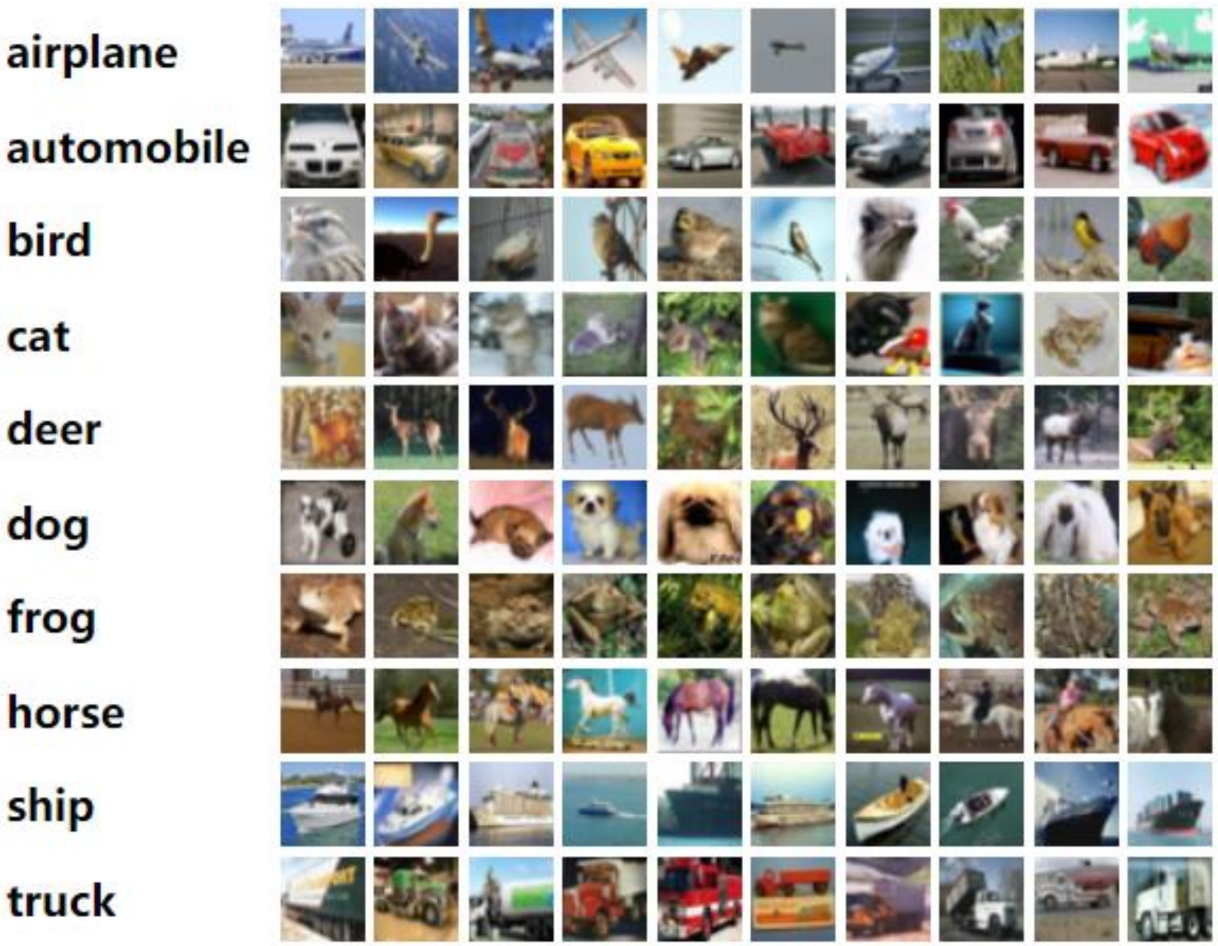
	LoLa	LoLa-Dense	LoLa-Small
Input data representation	Convolution	Dense → Convolution	Convolution
Speed	2.2s	7.2s	**0.29s**
Accuracy	**98.95%**	**98.95%**	96.92%

4.5 LoLa - CIFAR



[그림 14] LoLa 아키텍처의 CIFAR-10 분류 과정

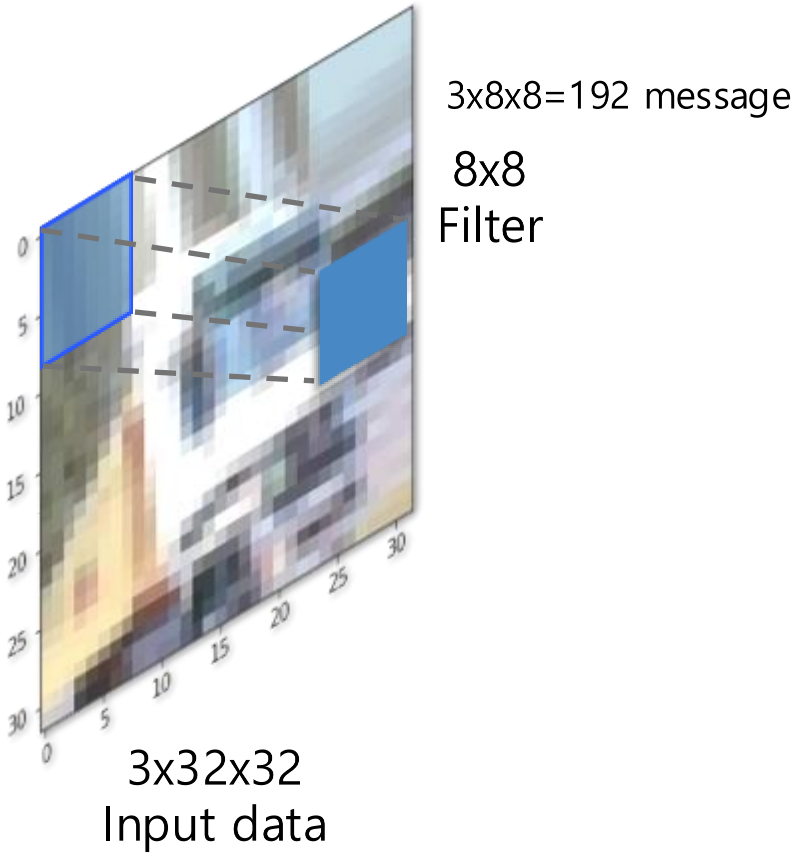
4.5 LoLa - CIFAR



[그림 15] CIFAR-10 Dataset

[그림 16] RGB 3 channels

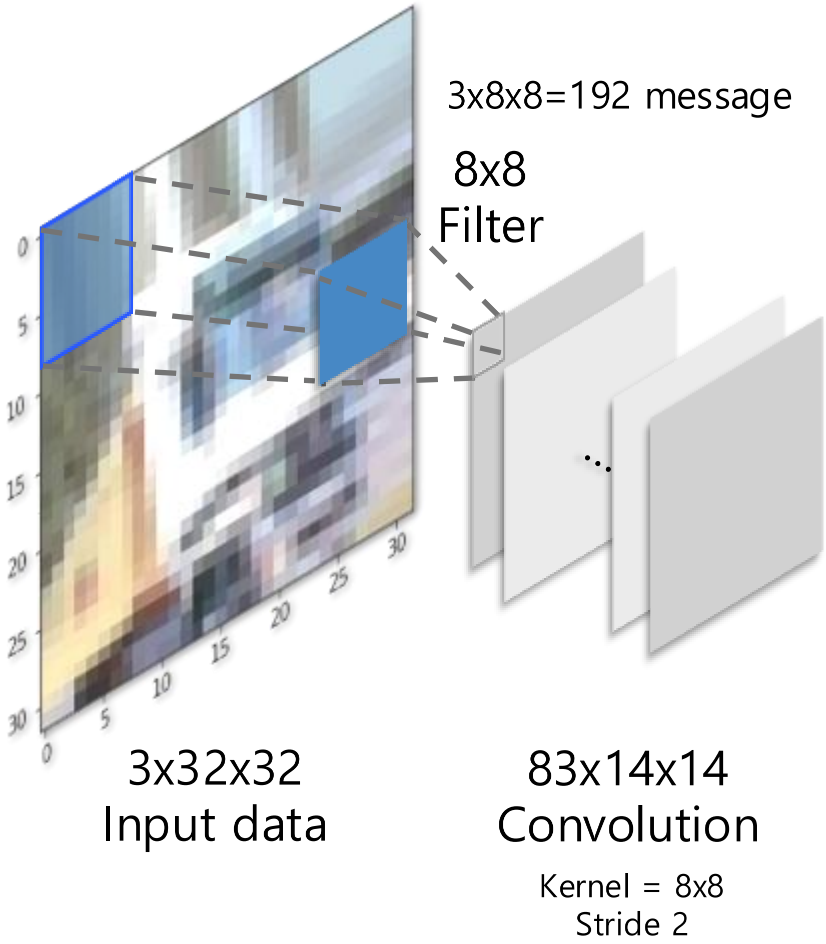
4.5 LoLa - CIFAR



[그림 17] LoLa 아키텍처의 CIFAR-10 분류 과정

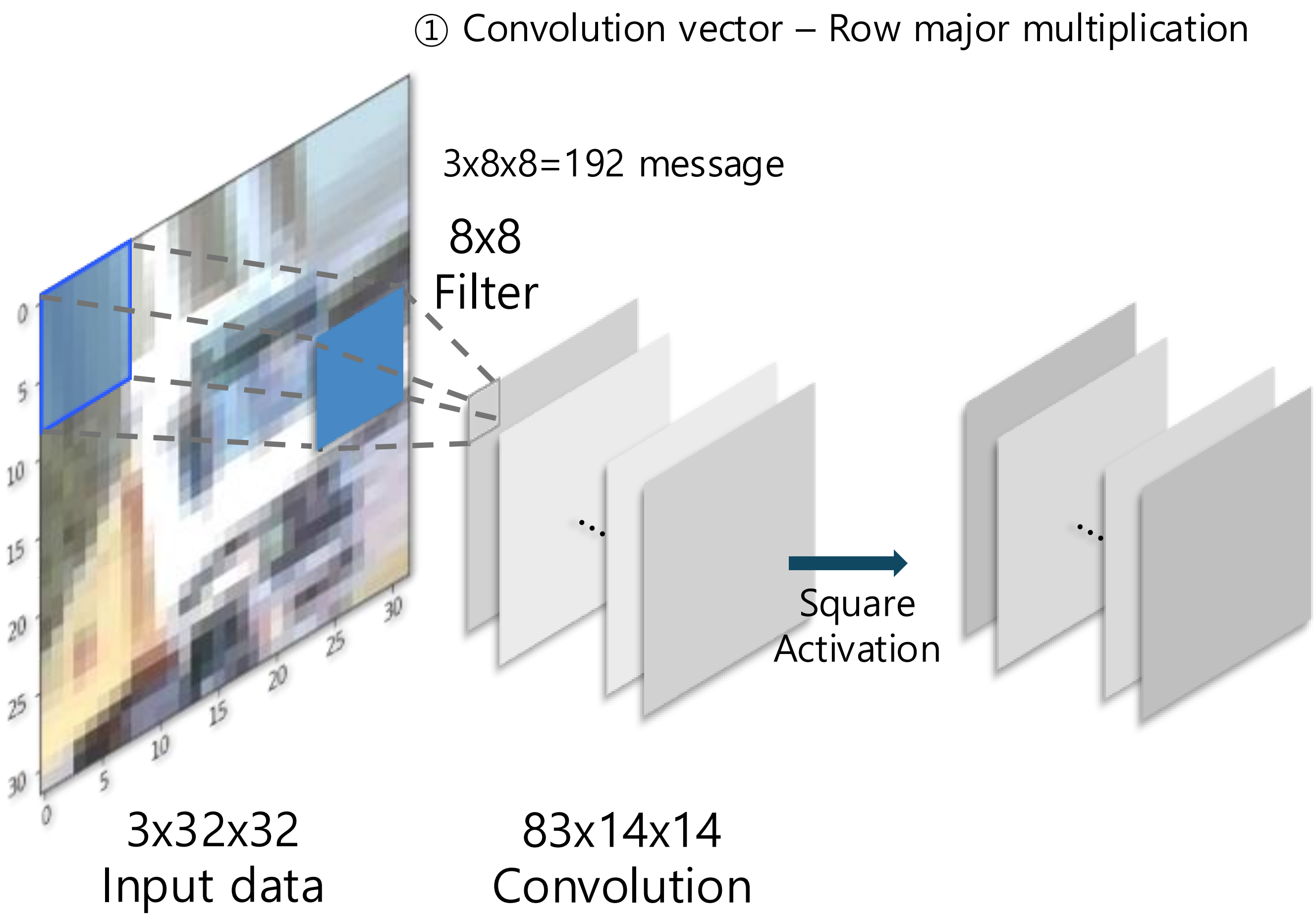
4.5 LoLa - CIFAR

① Convolution vector – Row major multiplication



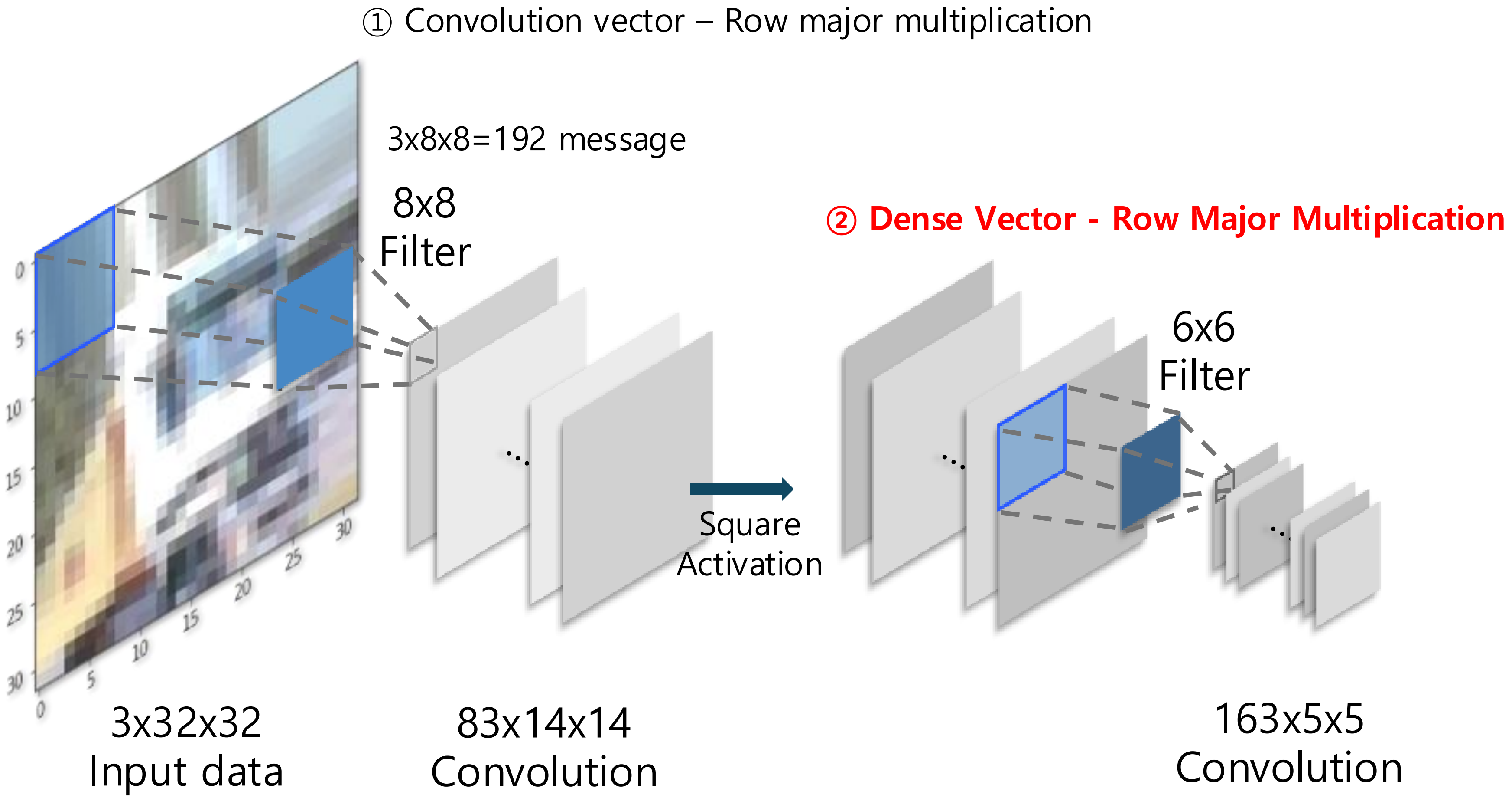
[그림 17] LoLa 아키텍처의 CIFAR-10 분류 과정

4.5 LoLa - CIFAR



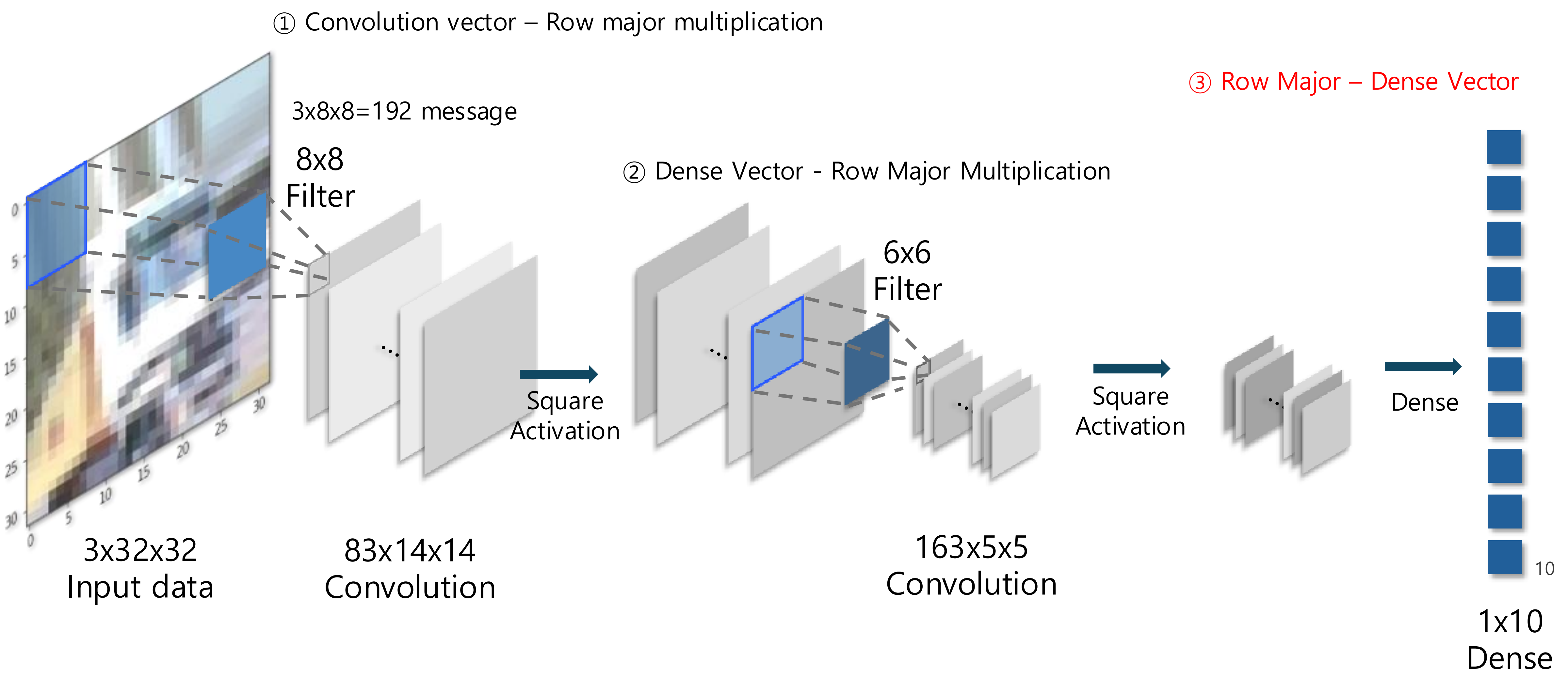
[그림 17] LoLa 아키텍처의 CIFAR-10 분류 과정

4.5 LoLa - CIFAR



[그림 17] LoLa 아키텍처의 CIFAR-10 분류 과정

4.5 LoLa - CIFAR



[그림 17] LoLa 아키텍처의 CIFAR-10 분류 과정

4.5 LoLa - Result

● 실험 결과

- 메모리 소모량: 12GB RAM
- 소모 시간: 730초
- 정확도: 74.1%

[표 4] 데이터셋 별 LoLa 결과

	MNIST	CIFAR-10
Speed	2.2s	**730s**
Accuracy	98.95%	**74.1%**

5.1 Limitations of HE for Deep Learning

- **동형 암호화로 신경망 평가 시 한계점**

1) 노이즈 증가

- 네트워크의 계층이 깊어질수록, 연산이 반복되며 노이즈 증가
  → 노이즈가 임계값을 넘으면, 복호화가 불가능해져 정확한 결과를 얻지 못함
  - Bootstrapping

2) 메시지 크기 증가

- 암호화된 메시지는 연산 중에 크기가 점점 증가함
  → 네트워크가 깊어질수록 요구되는 메모리와 계산 리소스가 증가
  - HEAAN

5.1 Transfer Learning – Deep Representation

- **Deep Representations**
  - **원본 데이터를 직접 암호화하는 대신 Deep Representation 을 통해 변환된 데이터를 암호화**
    - 장점
      1) 원본 데이터보다 크기가 작아 암호화된 메시지 크기가 줄어듦
      2) Shallow network에서도 높은 정확도
  - **실험 결과**
    - AlexNet을 사용해 CalTech-101 데이터셋에서 단순한 선형 모델을 이용해 테스트
      - 정확도 **81.6%**
      - Latency **0.16s**

6. Conclusion

- 정리
  - 동형암호 기반 privacy inference 솔루션 제시
    - 연산 과정에서 여러 표현 방식 활용

      → **복잡한 네트워크**에서도 작동 가능

      → **Latency**를 낮춤

  - 향후 연구 방향
    - **암호화된 데이터 위**에서 머신러닝 모델 **학습**에 관한 연구

End

End