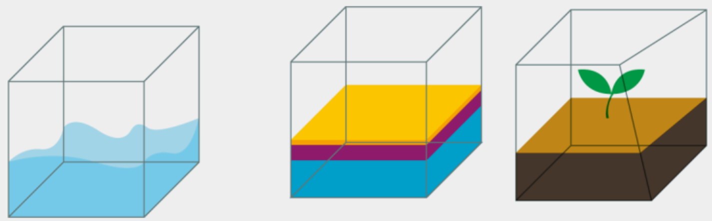
AAI. Seminar

국민청원 분류 - TextCNN



2021.11.03

김 민 준

CONTENTS

CHAPTER 1

- CNN

CHAPTER 2

- Task

CHAPTER 3

- TextCNN

CHAPTER 4

- CODE

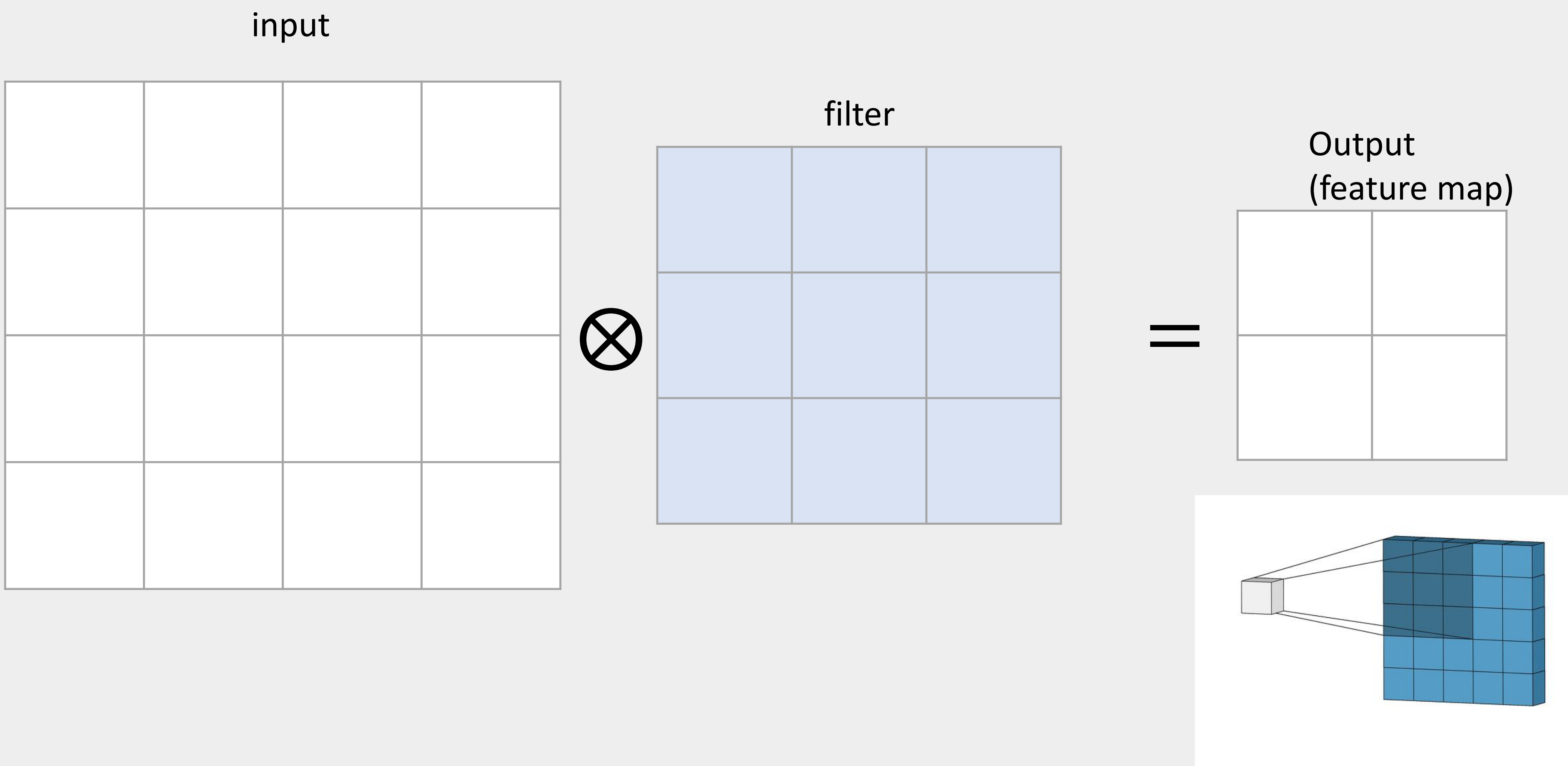
➢What is CNN(Convolution Neural Network)?

- 이미지 인식 – 패턴 학습에 특화된 신경망
- Convolution 연산을 통한 연산
- **부분**을 보는 것이 핵심 아이디어
(부분 : Filter)
- Fully-Connected Layer에 비해 매우 빠르고 적은 파라미터를 가짐.

➢What is Convolution ?

$*$: *Convolution symbol*

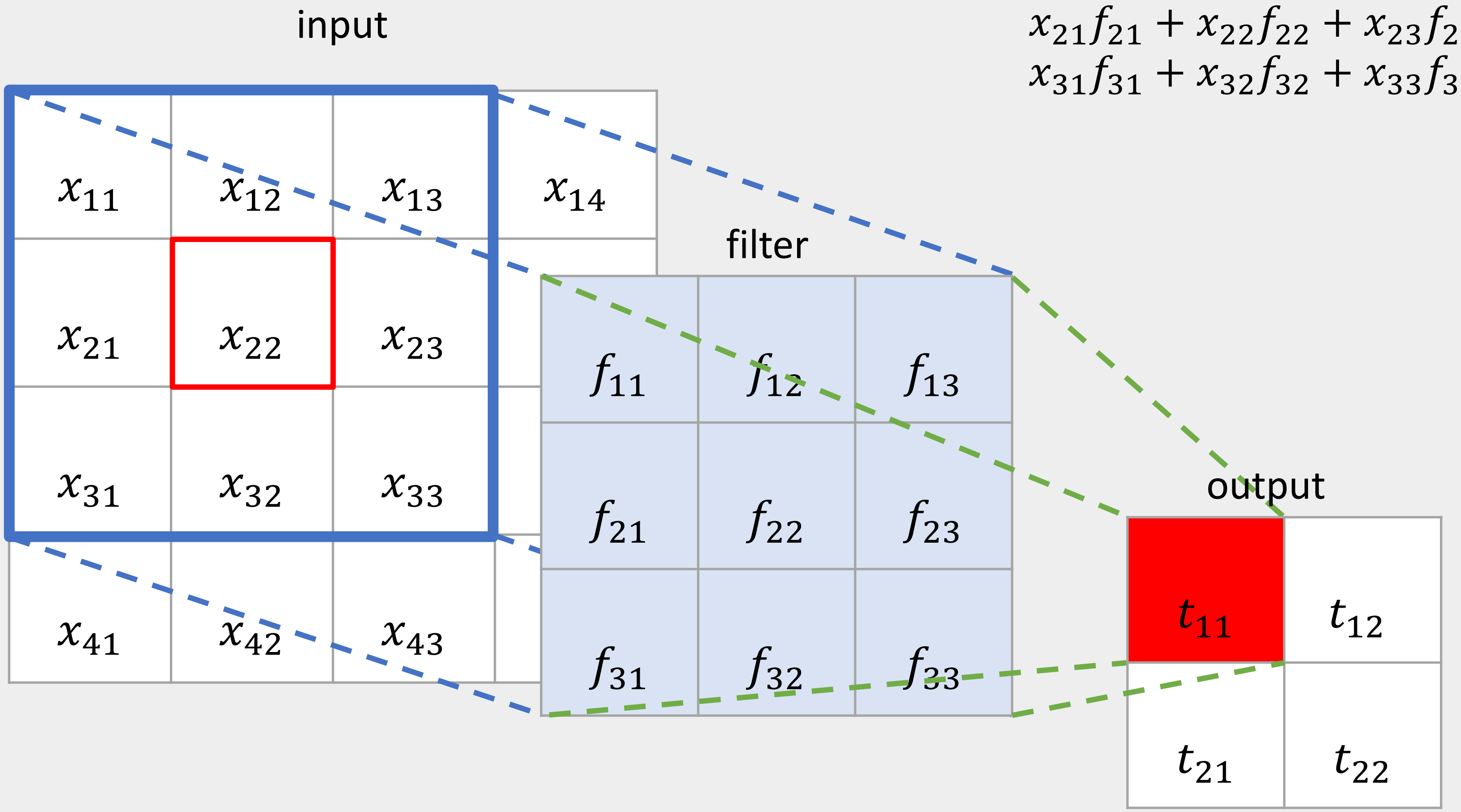
$t[n] = x[n] * h[n] = \Sigma_{k=0}^{N} x[k]h[n-k]$



➢Convolution Operation

$*$: *Convolution symbol*
$t[n] = x[n] * h[n] = \Sigma_{k=0}^{N} x[k]h[n-k]$

$t_{11} = x_{11}f_{11} + x_{12}f_{12} + x_{13}f_{13} +$
$x_{21}f_{21} + x_{22}f_{22} + x_{23}f_{23} +$
$x_{31}f_{31} + x_{32}f_{32} + x_{33}f_{33}$

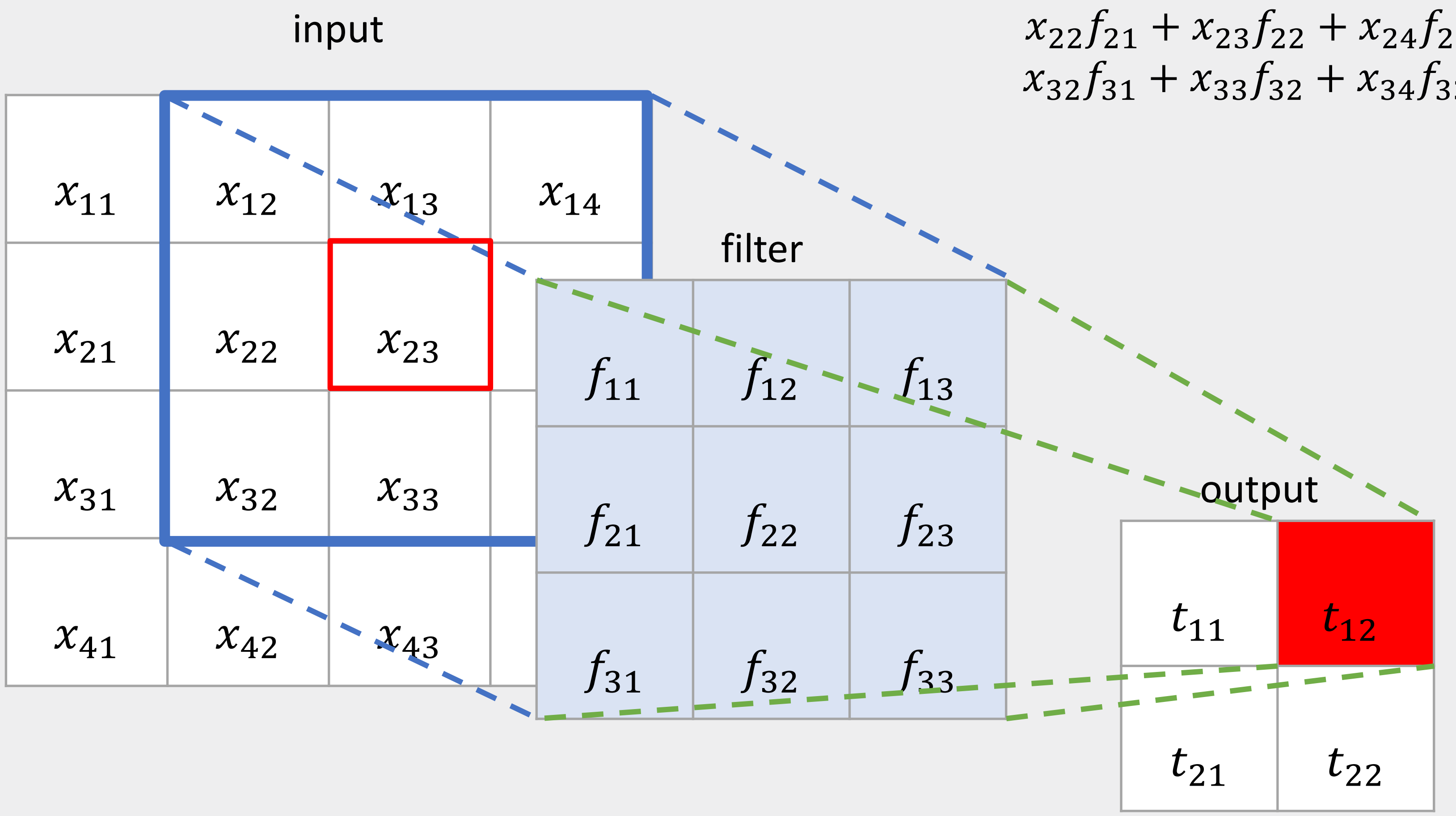


➢Convolution Operation

$*$: *Convolution symbol*

$t[n] = x[n] * h[n] = \Sigma_{k=0}^{N} x[k]h[n-k]$

$$t_{12} = x_{12}f_{11} + x_{13}f_{12} + x_{14}f_{13} + x_{22}f_{21} + x_{23}f_{22} + x_{24}f_{23} + x_{32}f_{31} + x_{33}f_{32} + x_{34}f_{33}$$

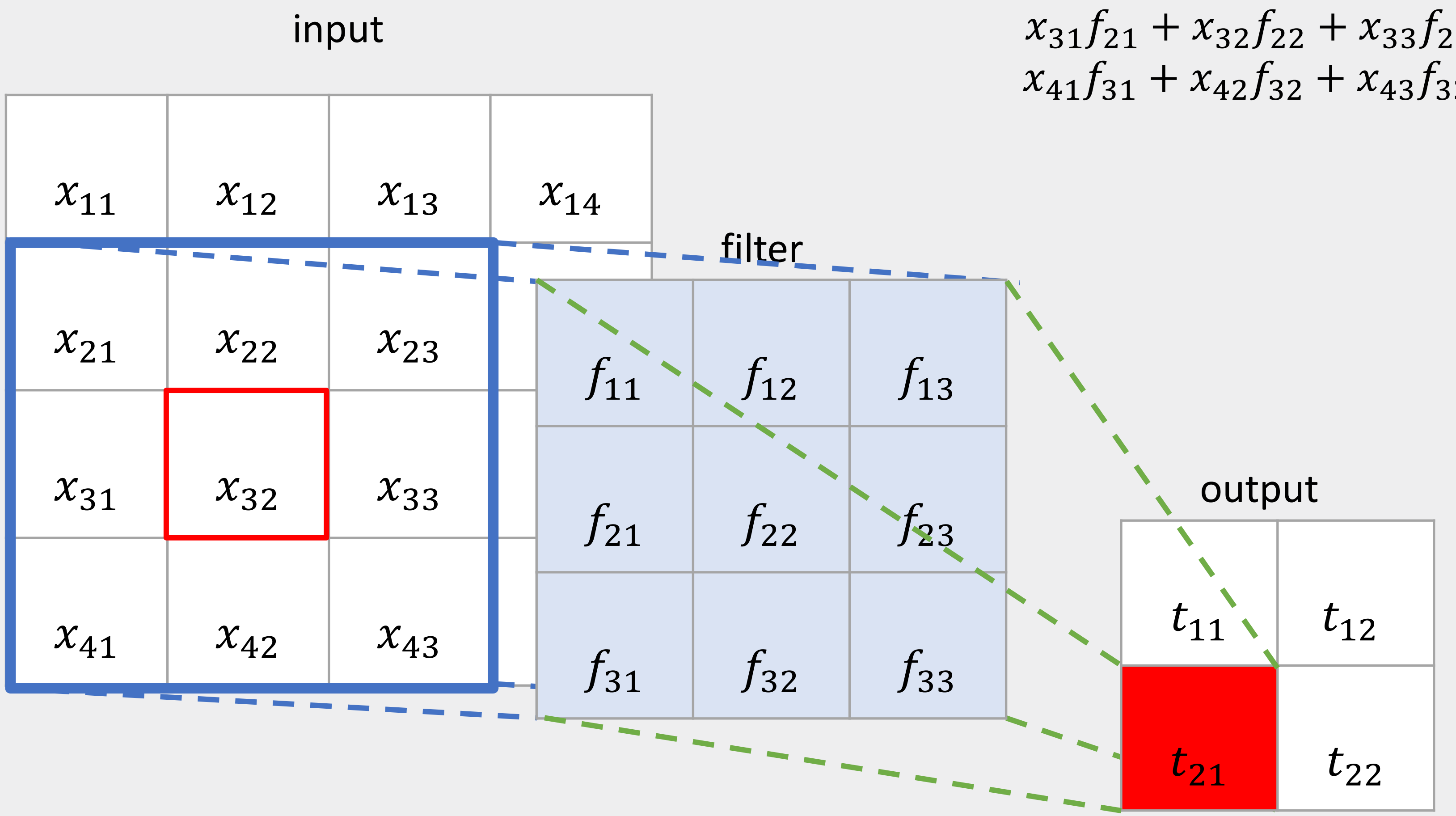


➢Convolution Operation

$*$: *Convolution symbol*

$$t[n] = x[n] * h[n] = \Sigma_{k=0}^{N} x[k]h[n-k]$$

$$t_{21} = x_{21}f_{11} + x_{22}f_{12} + x_{23}f_{13} + x_{31}f_{21} + x_{32}f_{22} + x_{33}f_{23} + x_{41}f_{31} + x_{42}f_{32} + x_{43}f_{33}$$

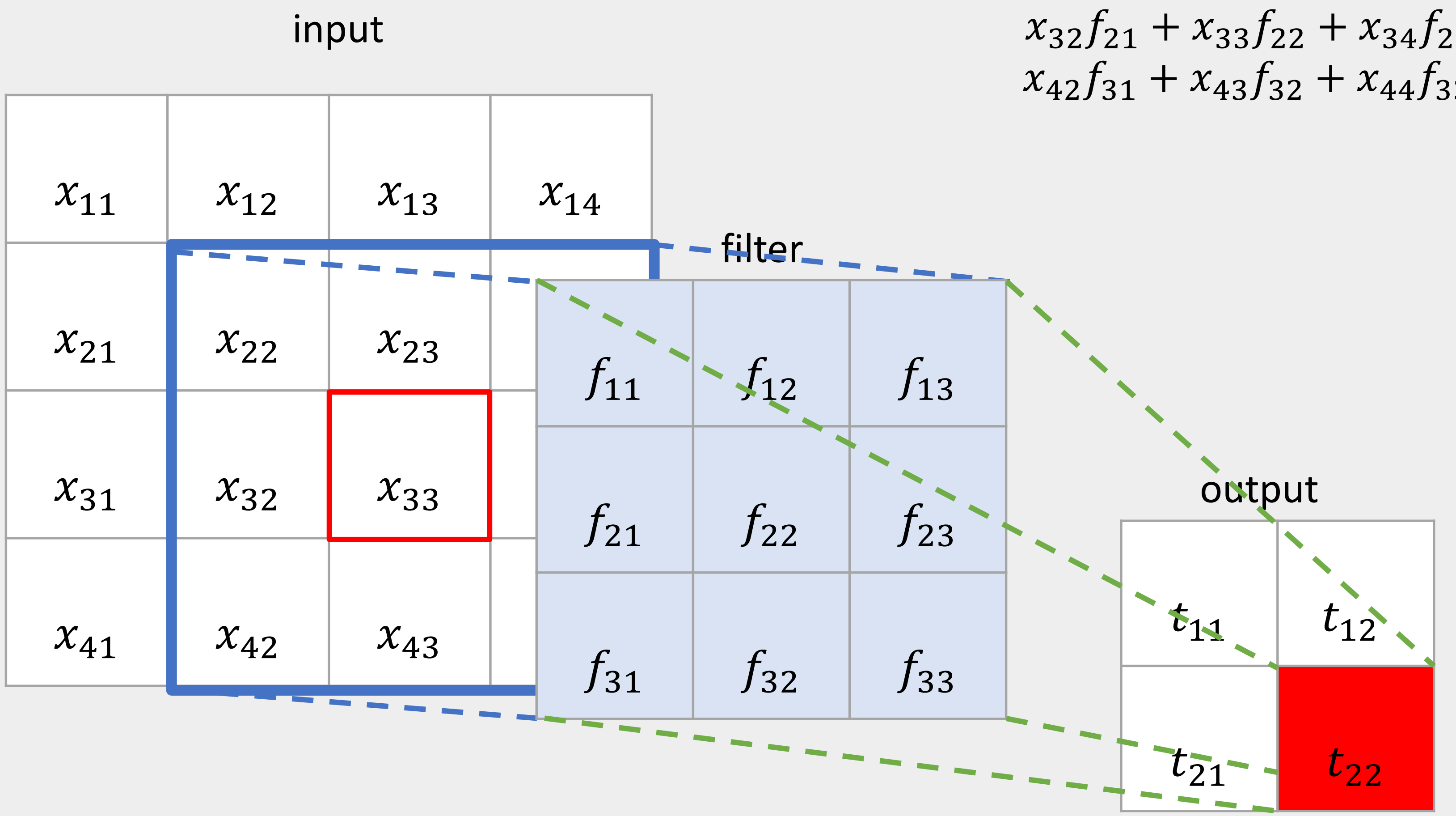


➢Convolution Operation

$*$: *Convolution symbol*

$t[n] = x[n] * h[n] = \Sigma_{k=0}^{N} x[k]h[n-k]$

$t_{22} = x_{22}f_{11} + x_{23}f_{12} + x_{24}f_{13} +$
$x_{32}f_{21} + x_{33}f_{22} + x_{34}f_{23} +$
$x_{42}f_{31} + x_{43}f_{32} + x_{44}f_{33}$



➢padding?

$$
\underset{\text{input}}{\begin{array}{|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 & 0 \\ \hline
0 & x_{11} & x_{12} & x_{13} & x_{14} & 0 \\ \hline
0 & x_{21} & x_{22} & x_{23} & x_{24} & 0 \\ \hline
0 & x_{31} & x_{32} & x_{33} & x_{34} & 0 \\ \hline
0 & x_{41} & x_{42} & x_{43} & x_{44} & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}}
\otimes
\underset{\text{filter}}{\begin{array}{|c|c|c|}
\hline
f_{11} & f_{12} & f_{13} \\ \hline
f_{21} & f_{22} & f_{23} \\ \hline
f_{31} & f_{32} & f_{33} \\ \hline
\end{array}}
=
\underset{\text{output}}{\begin{array}{|c|c|c|c|}
\hline
t_{11} & t_{12} & t_{13} & t_{14} \\ \hline
t_{21} & t_{22} & t_{23} & t_{24} \\ \hline
t_{31} & t_{32} & t_{33} & t_{34} \\ \hline
t_{41} & t_{42} & t_{43} & t_{44} \\ \hline
\end{array}}
$$

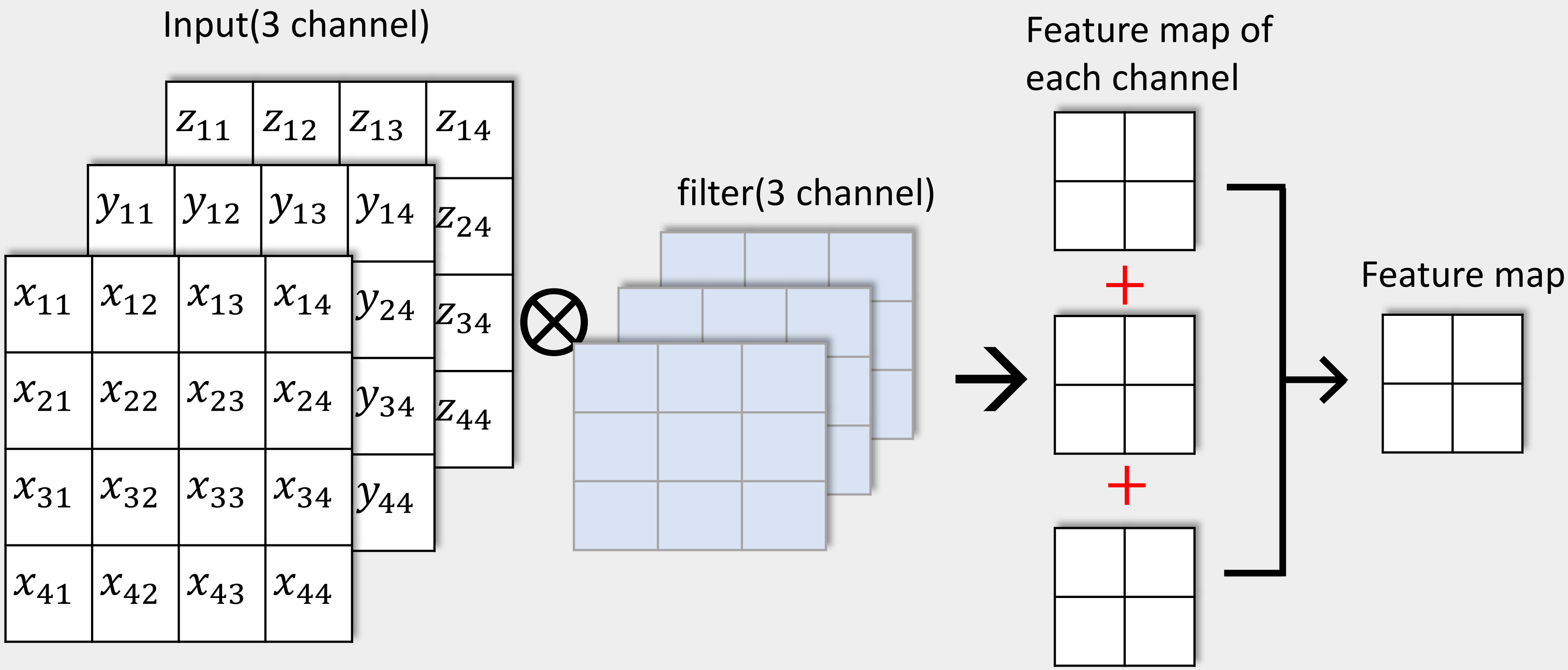
➢Pooling? Max Pooling?

Down sampling

1	2	0	1	**3**	2
3	1	**2**	1	0	1
0	**4**	1	0	2	**5**
1	3	2	**4**	2	1
1	**2**	1	4	1	2
0	2	**5**	4	**3**	1

3	2	3
4	4	5
2	5	3

➢Channel?


Input(3 channel)
filter(3 channel)
Feature map of each channel
Feature map

➢Text 분석을 통해 청원 인원 1000명 up, down예측

→ 1000명을 넘는 청원, 넘지 않는 청원을 class 로 생각

→ 각 class를 결정하는 Text의 어떠한 패턴이 있을 것

→ 비슷한 의미를 갖는 embedding vector의 패턴 인식

➢TextCNN?

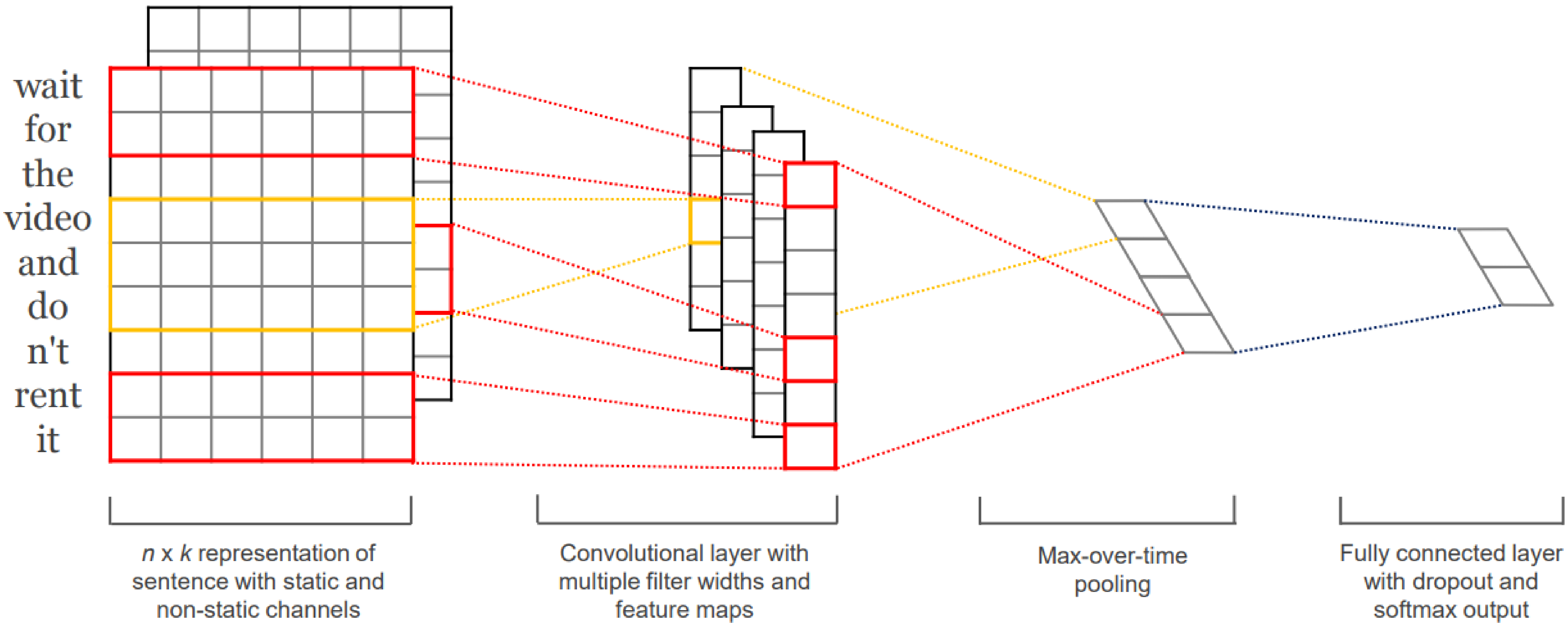
- Filter가 문장을 스캔하며 문맥적 의미 파악

1. Word embedding
2. Feature Map 생성
3. Max Pooling
4. Classification

장점

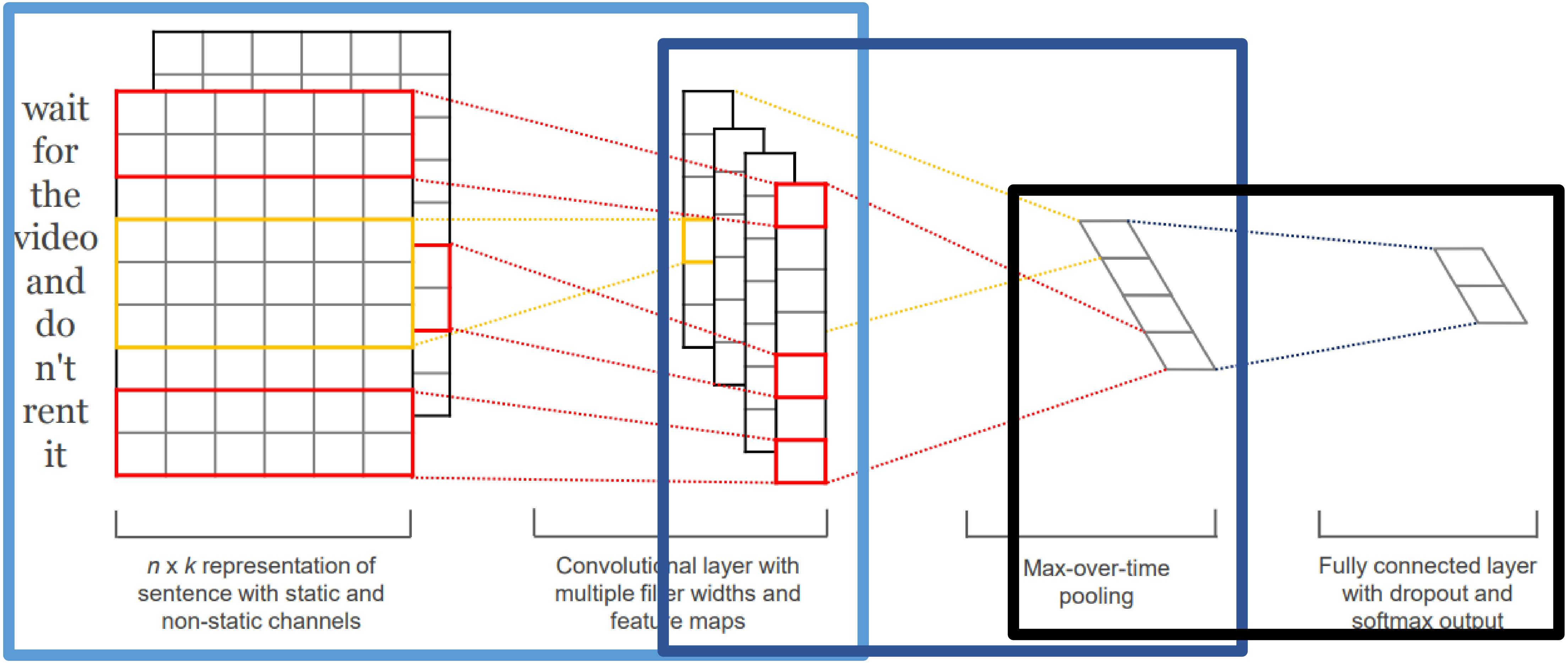
- 문장의 문맥적 의미를 파악하는 과정에서 **정보를 집약 → 연산 속도** ↑
- **분류 문제에서** RNN보다 좋은 성능

➢TextCNN architecture



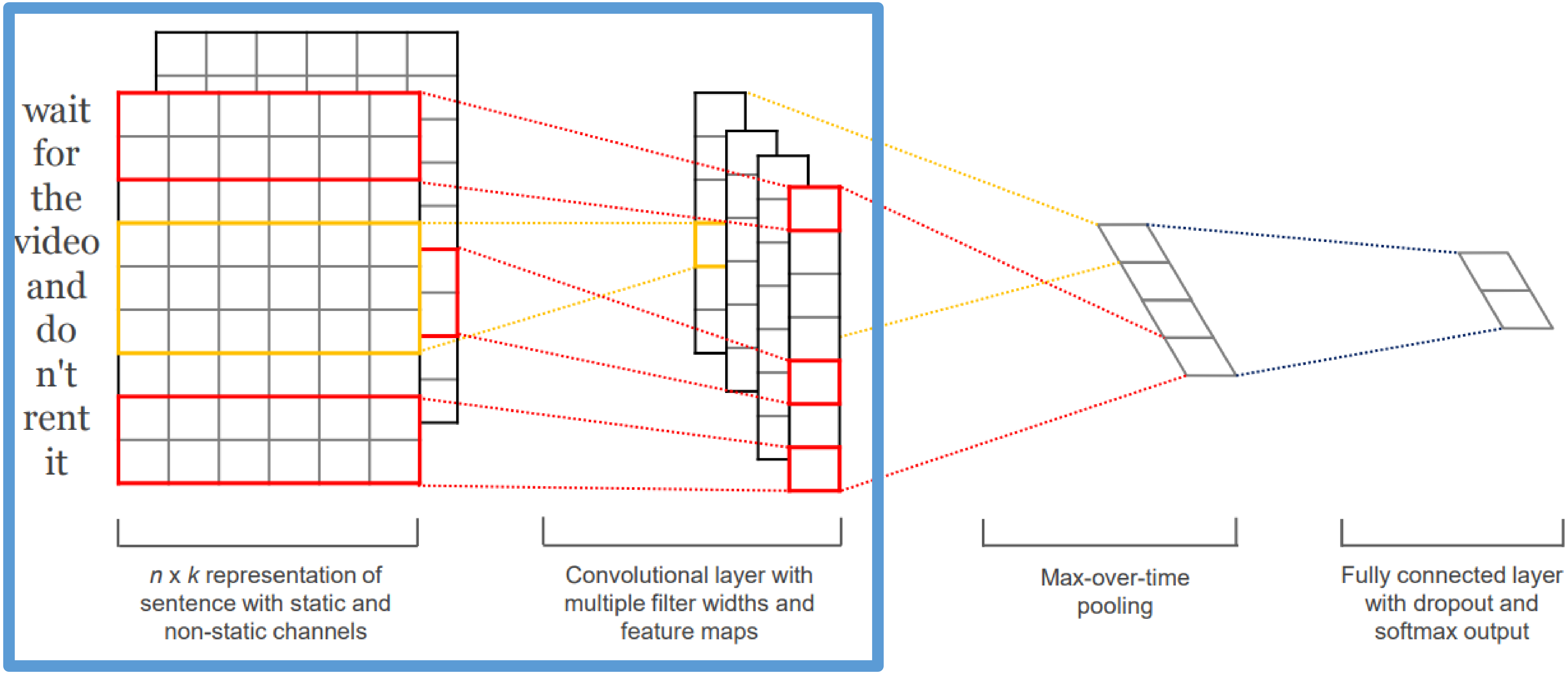
[Y. Kim] Convolutional Neural Networks for Sentence Classification (2014)

➢TextCNN architecture



[Y. Kim] Convolutional Neural Networks for Sentence Classification (2014)

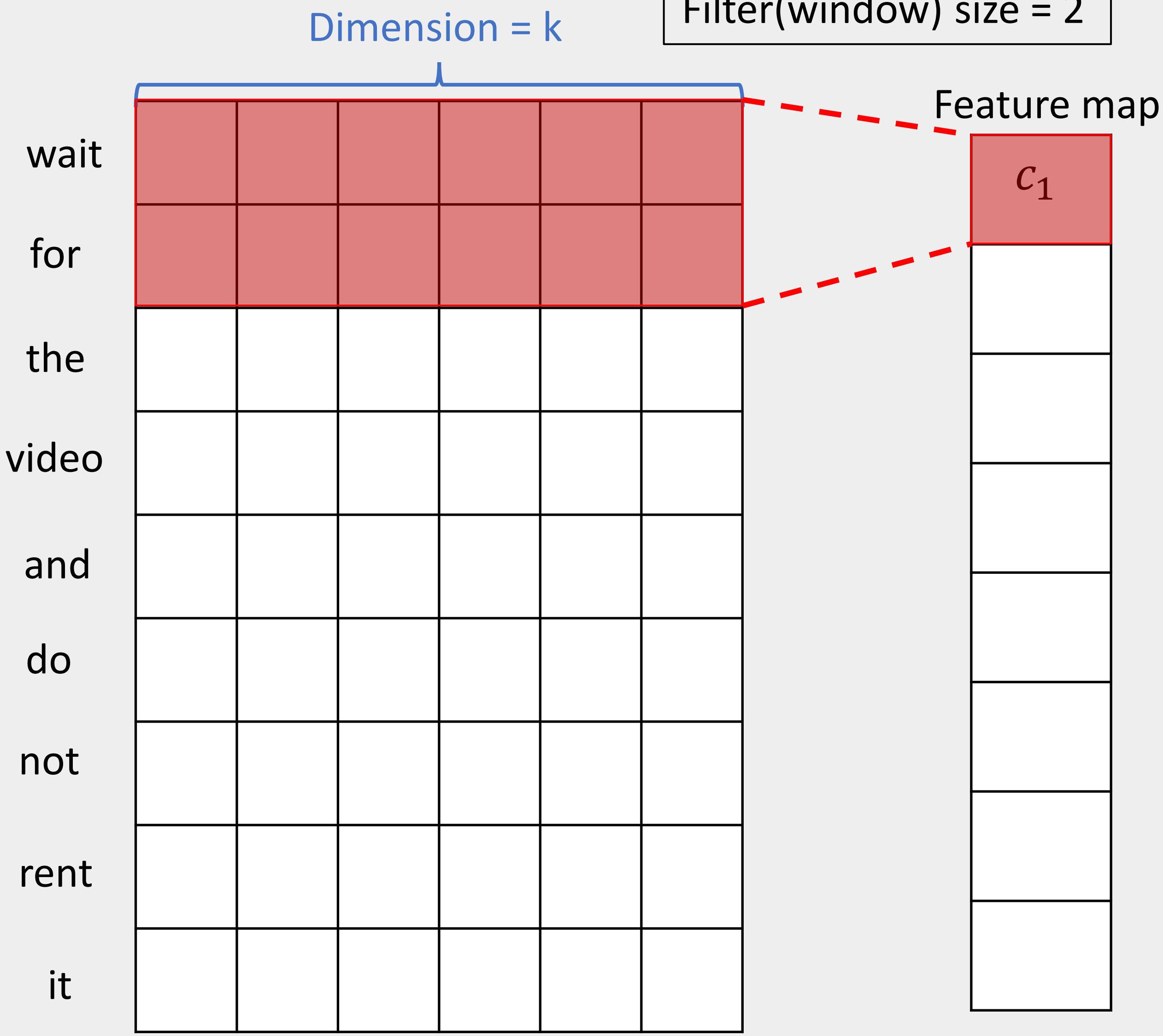
➢TextCNN architecture

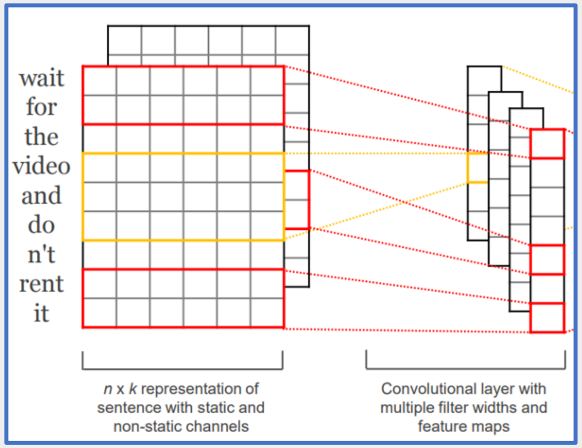


[Y. Kim] Convolutional Neural Networks for Sentence Classification (2014)

➢TextCNN architecture

Filter(window) size = 2

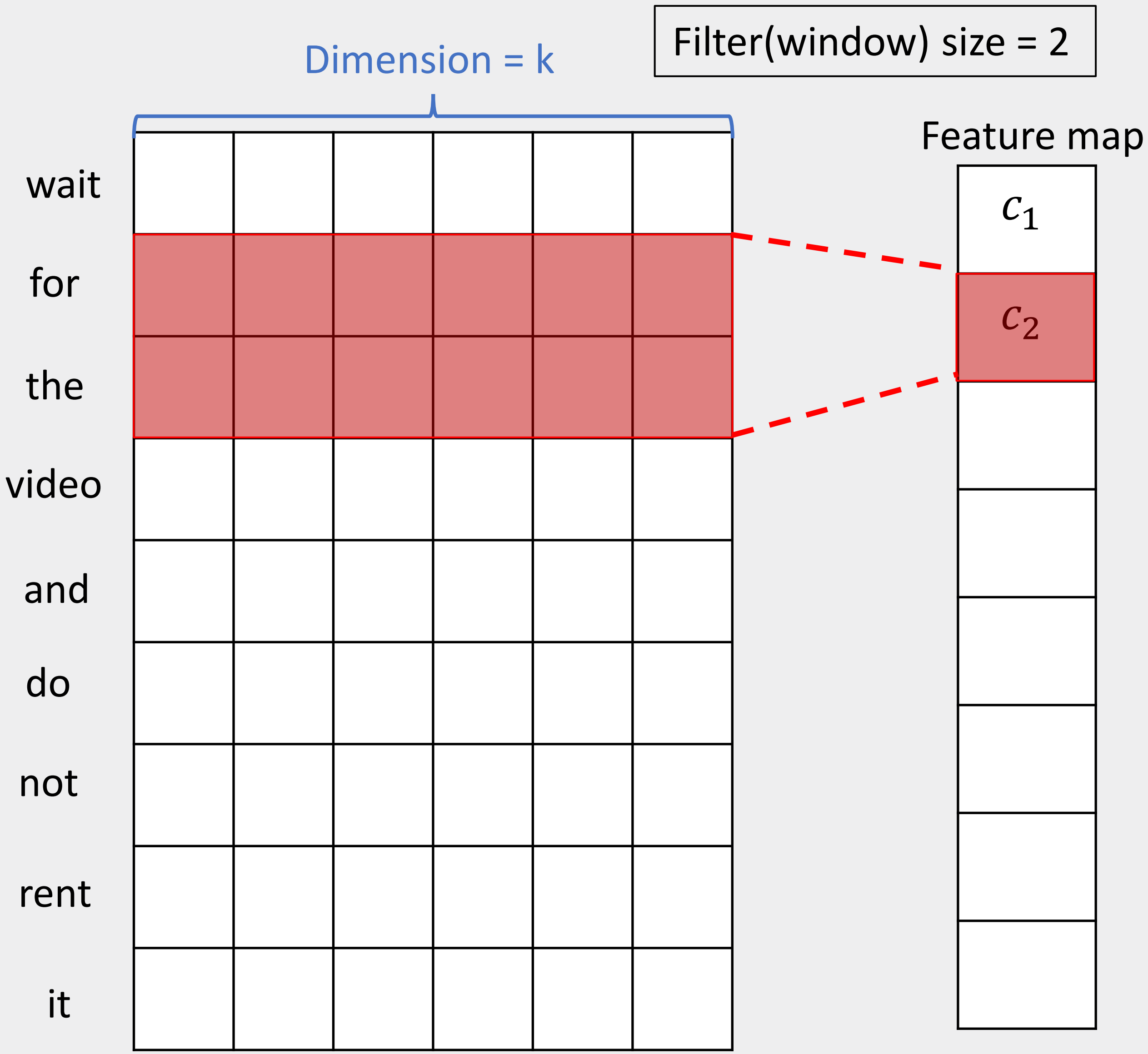


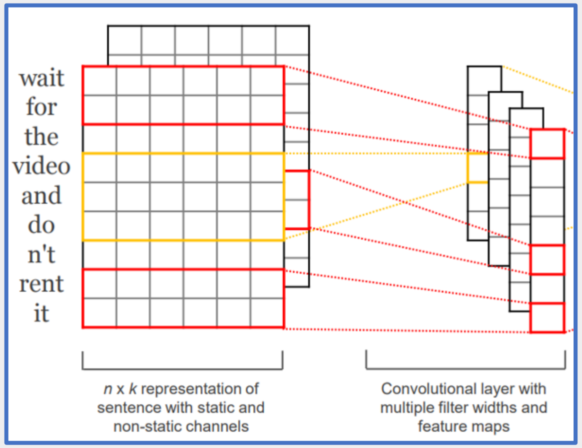


Filter size : (2, k)

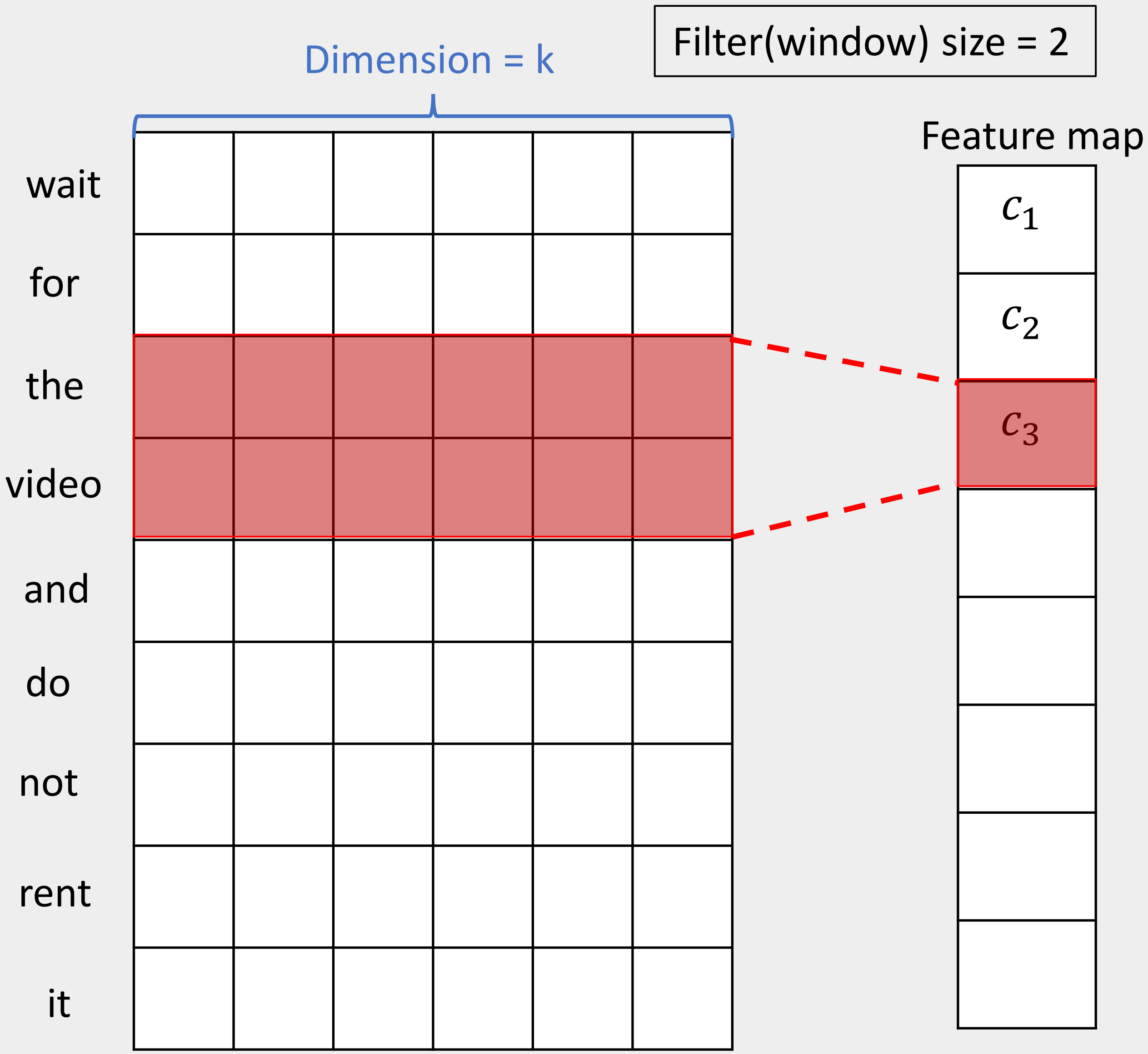
f_{11}	f_{12}	f_{13}	$\cdots$	$\cdots$	f_{1k}
f_{21}	f_{22}	f_{23}	$\cdots$	$\cdots$	f_{2k}

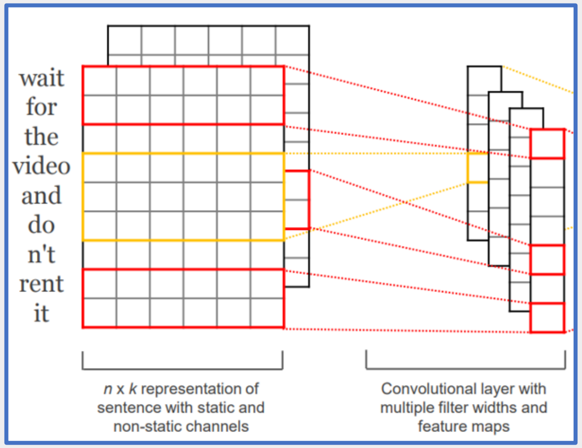
➢TextCNN architecture



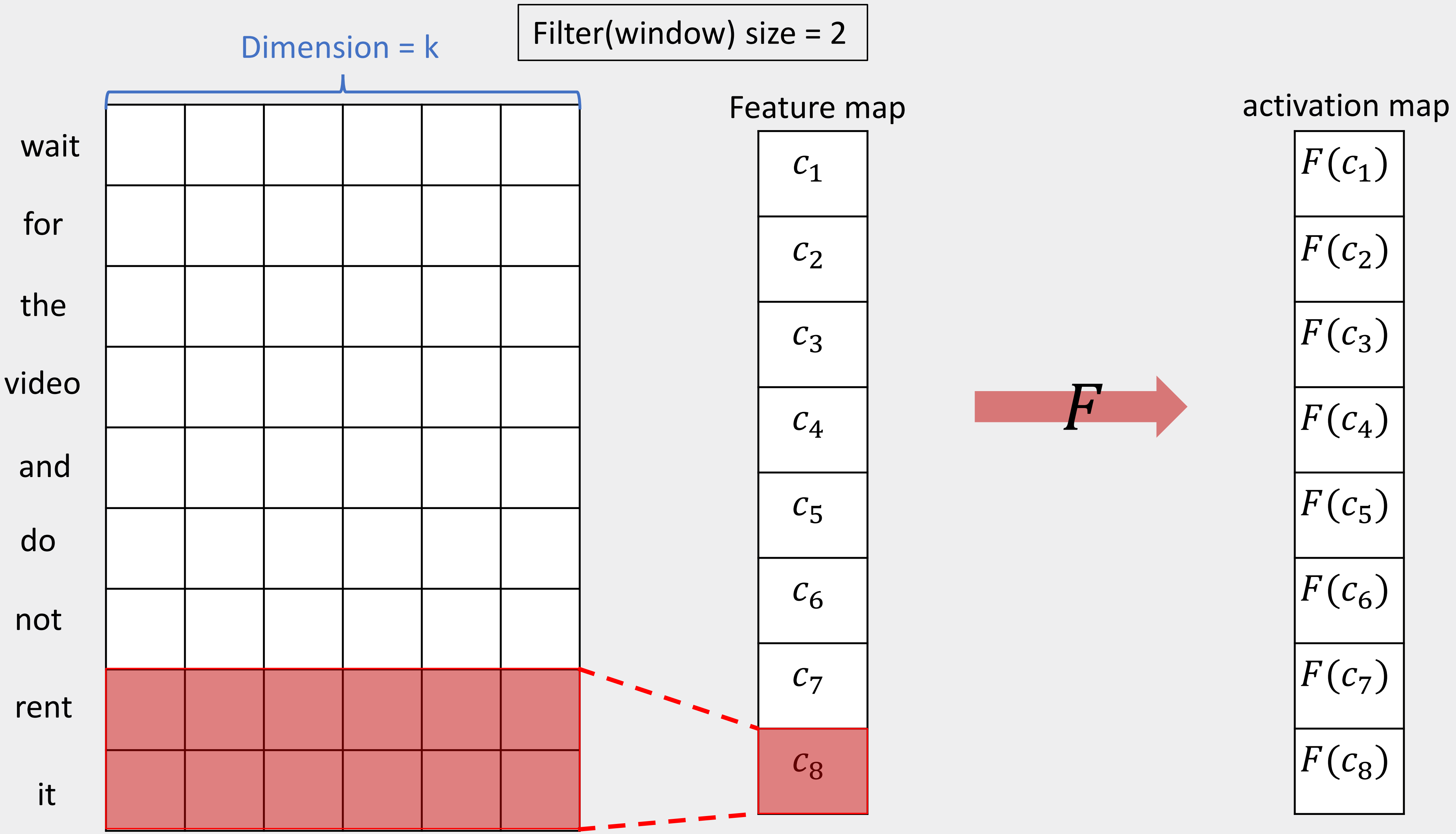


➢TextCNN architecture

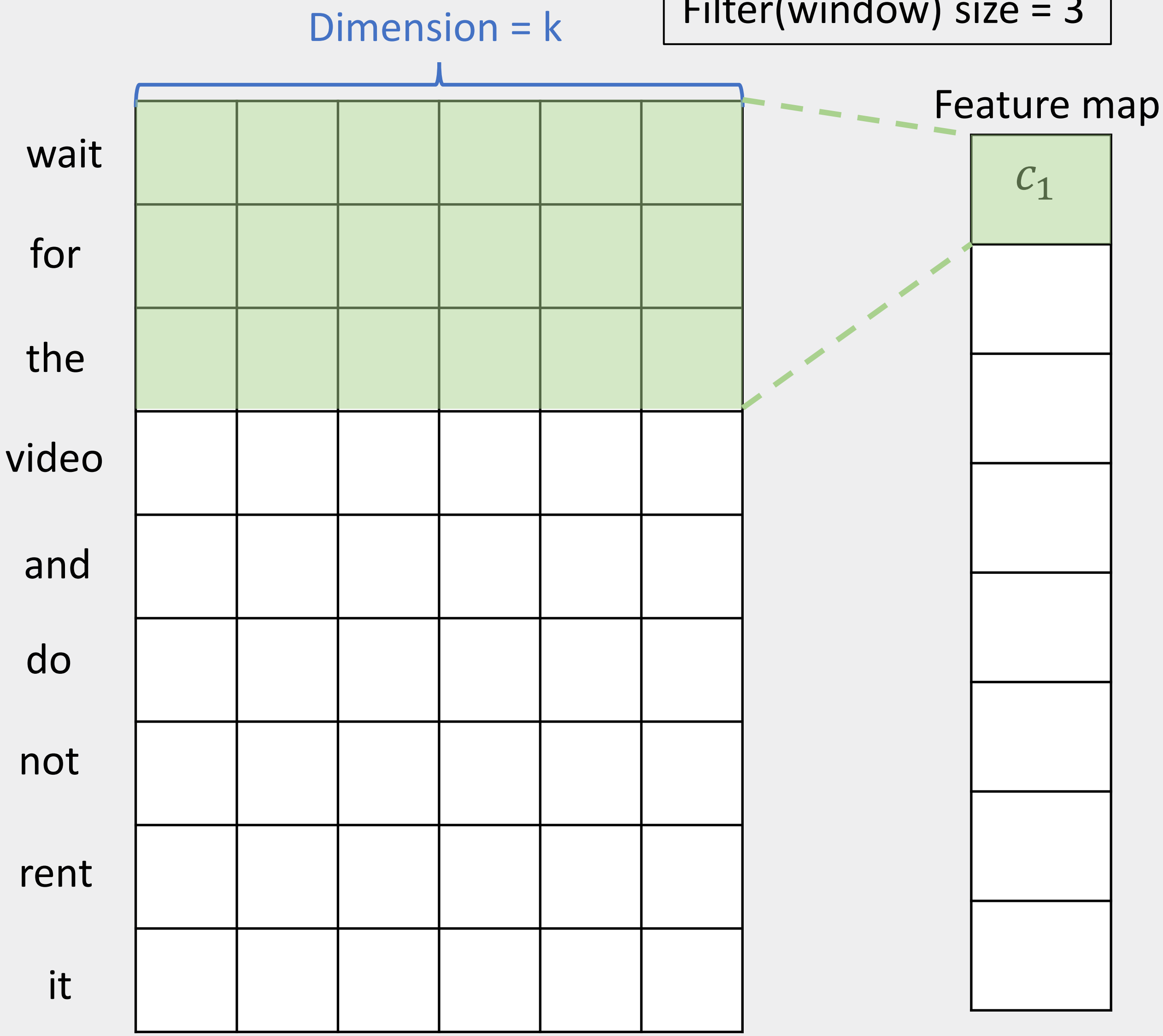


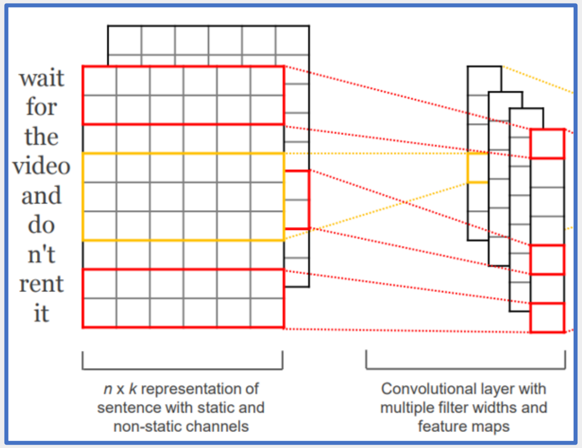


➢TextCNN architecture



➢TextCNN architecture

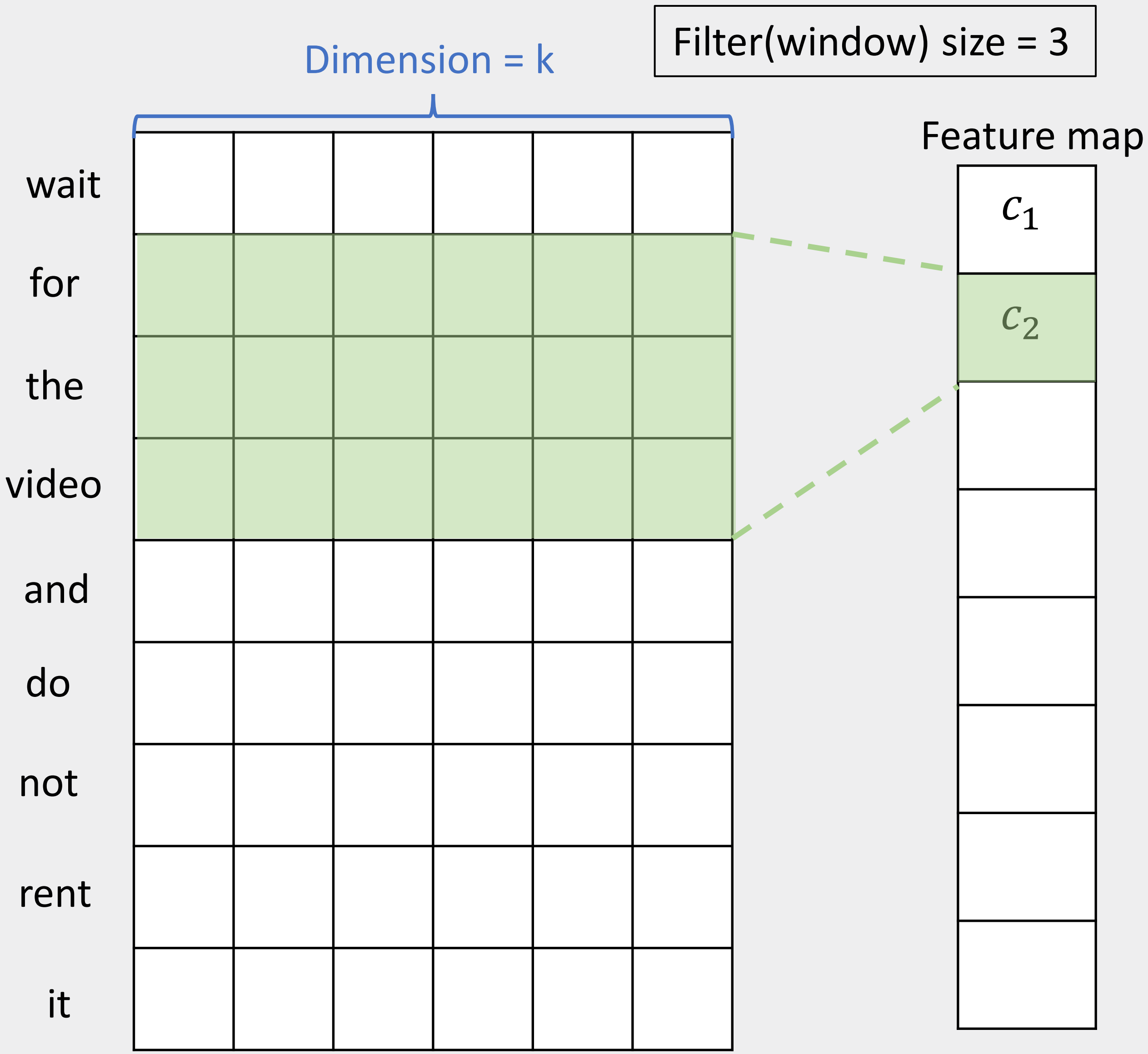


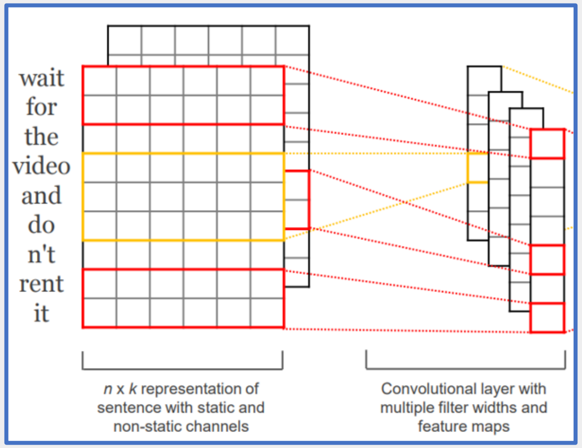


Filter size : (3, k)

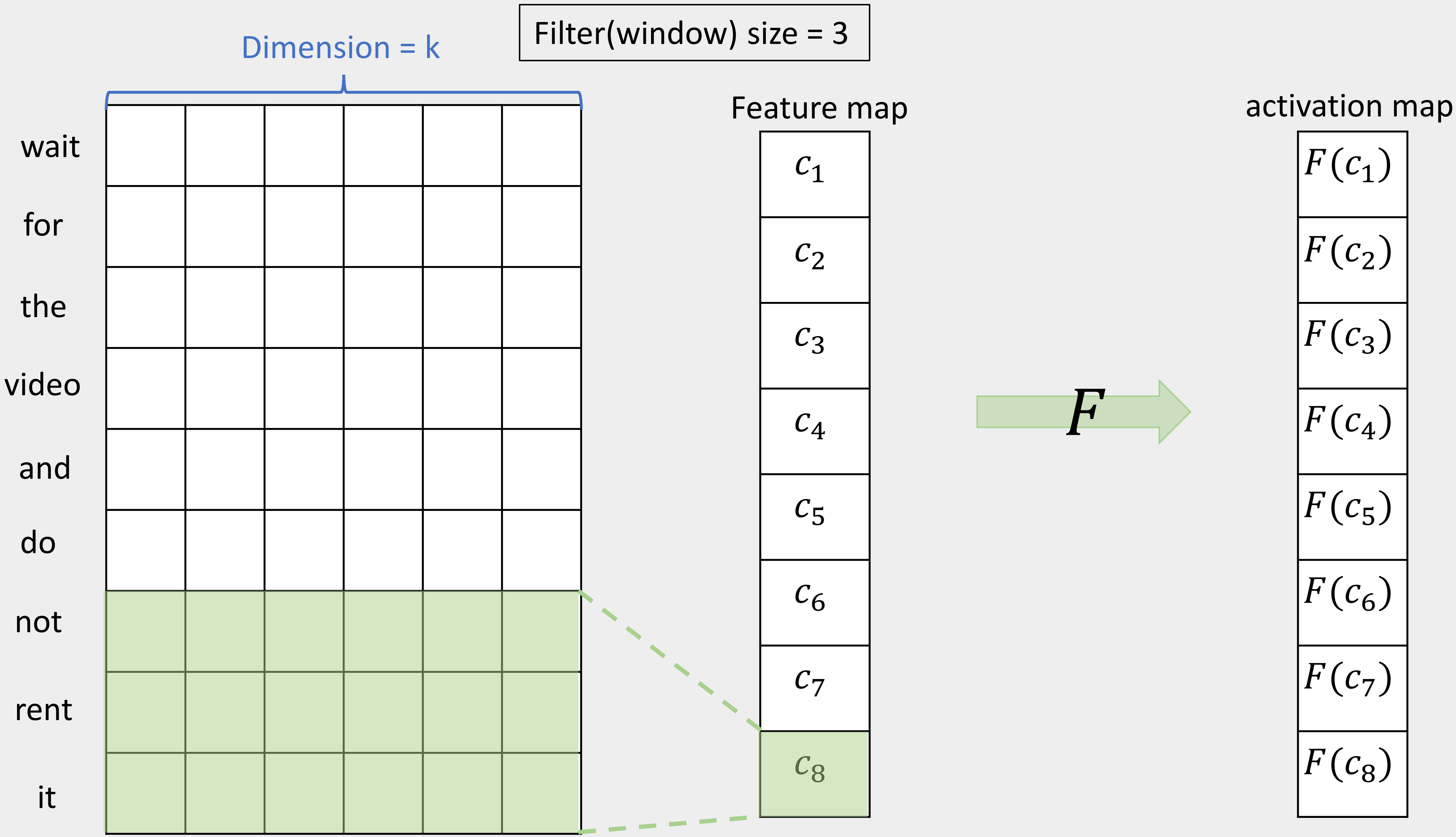
f_{11}	f_{12}	f_{13}	$\cdots$	$\cdots$	f_{1k}
f_{21}	f_{22}	f_{23}	$\cdots$	$\cdots$	f_{2k}
f_{31}	f_{32}	f_{33}	$\cdots$	$\cdots$	f_{3k}

➢TextCNN architecture

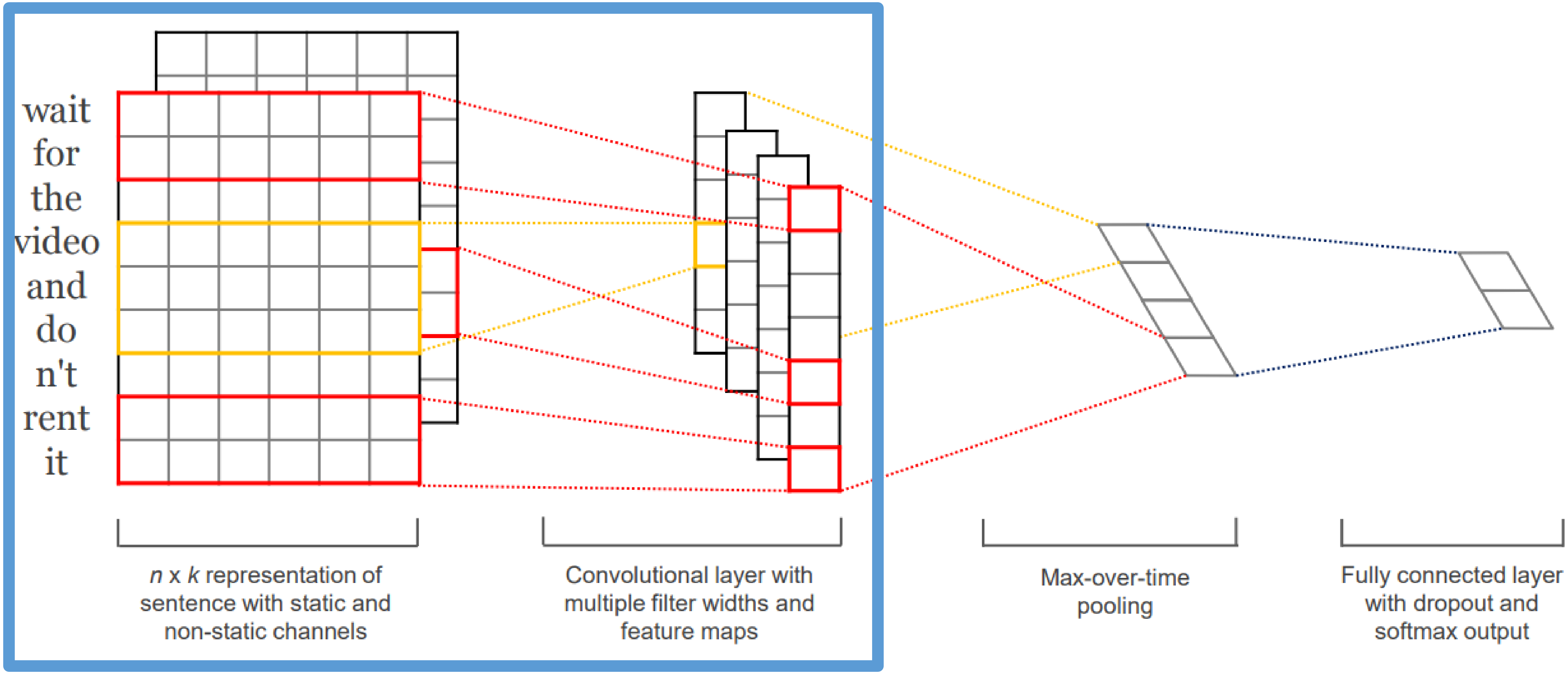




➢TextCNN architecture

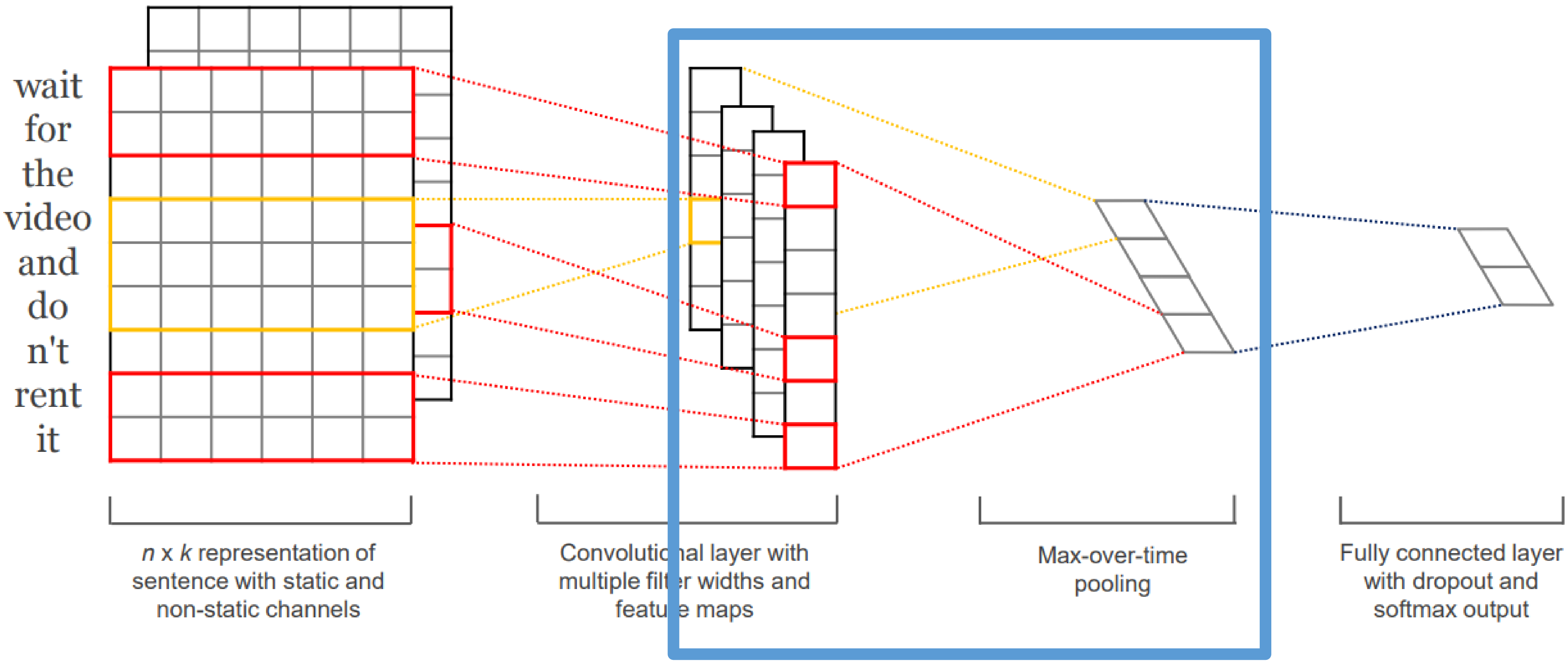


➢TextCNN architecture



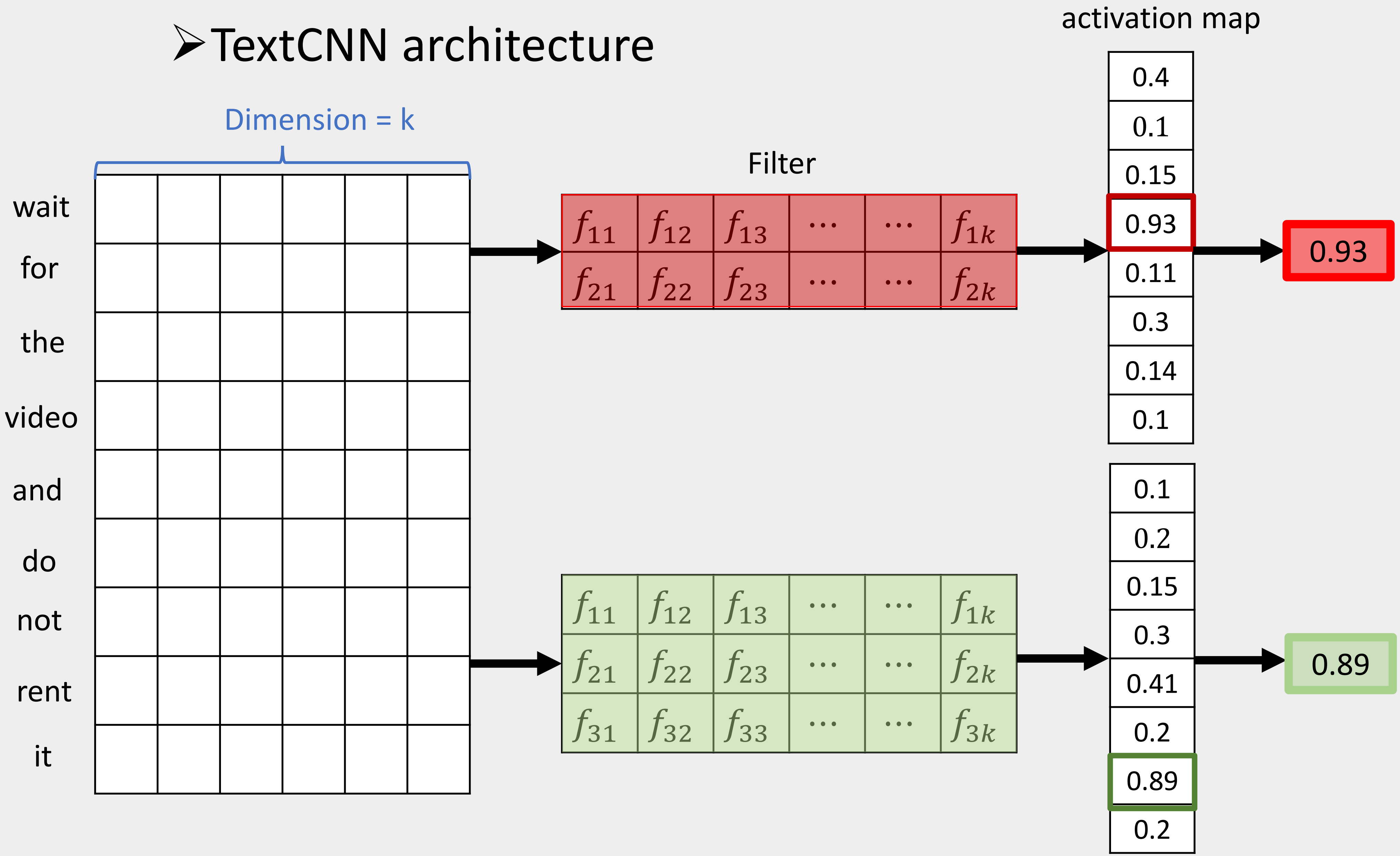
[Y. Kim] Convolutional Neural Networks for Sentence Classification (2014)

➢TextCNN architecture

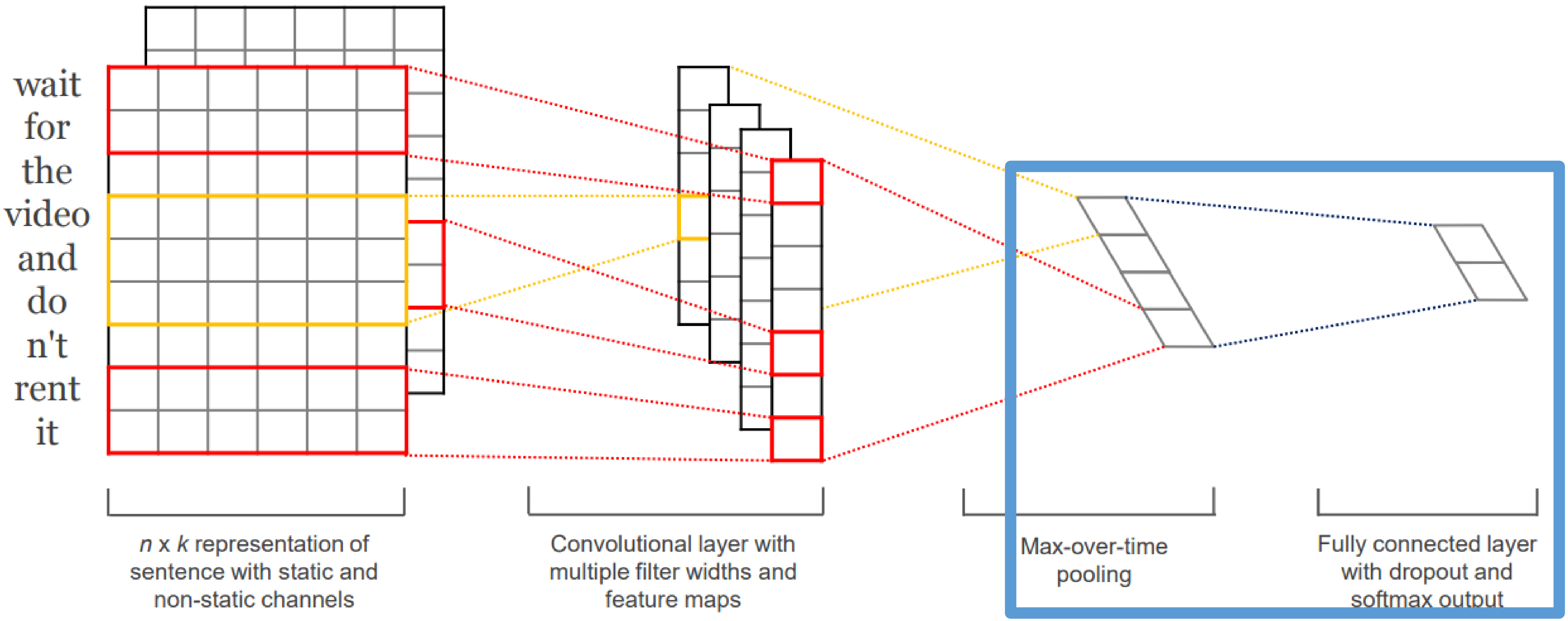


[Y. Kim] Convolutional Neural Networks for Sentence Classification (2014)

➢TextCNN architecture

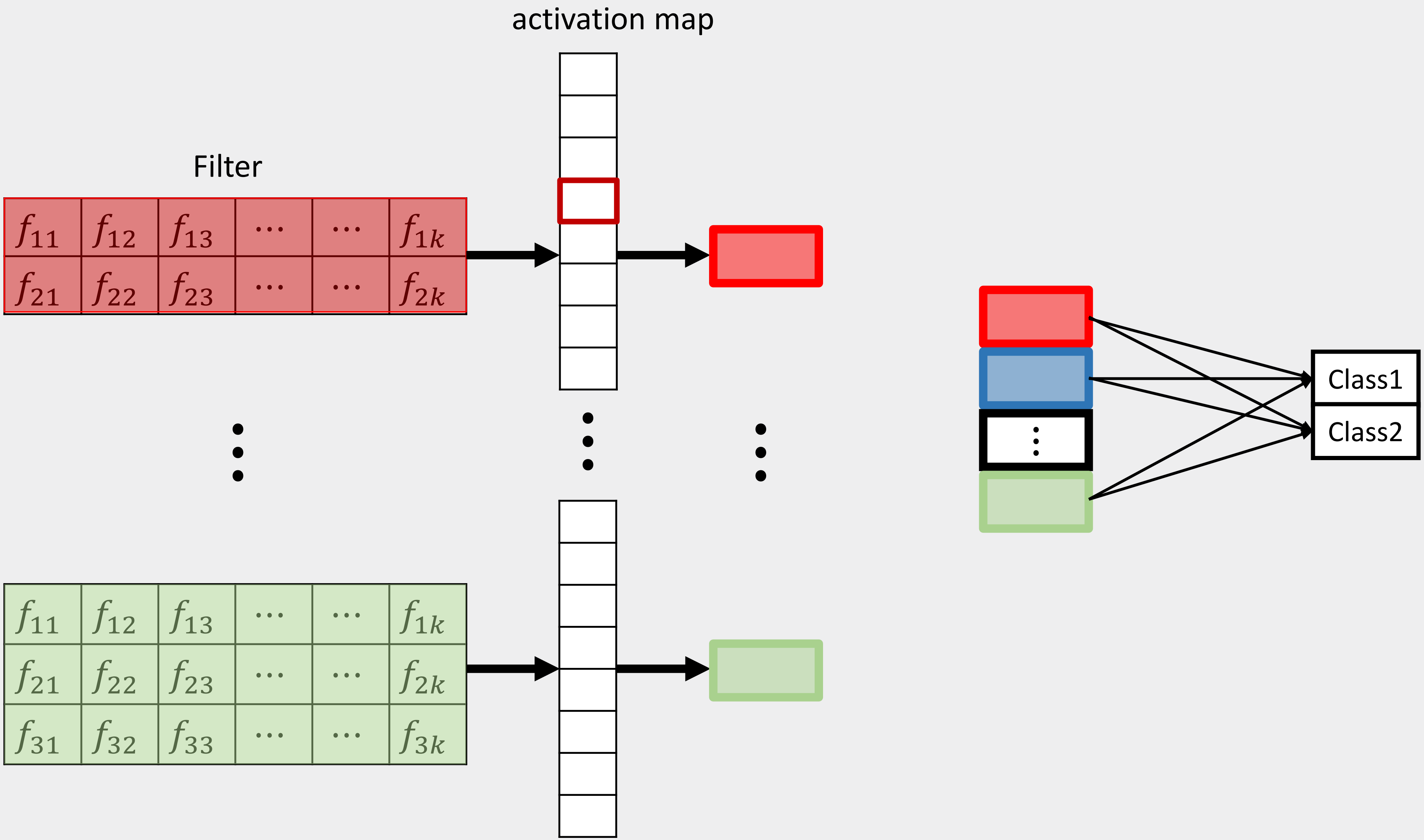


➢TextCNN architecture



[Y. Kim] Convolutional Neural Networks for Sentence Classification (2014)

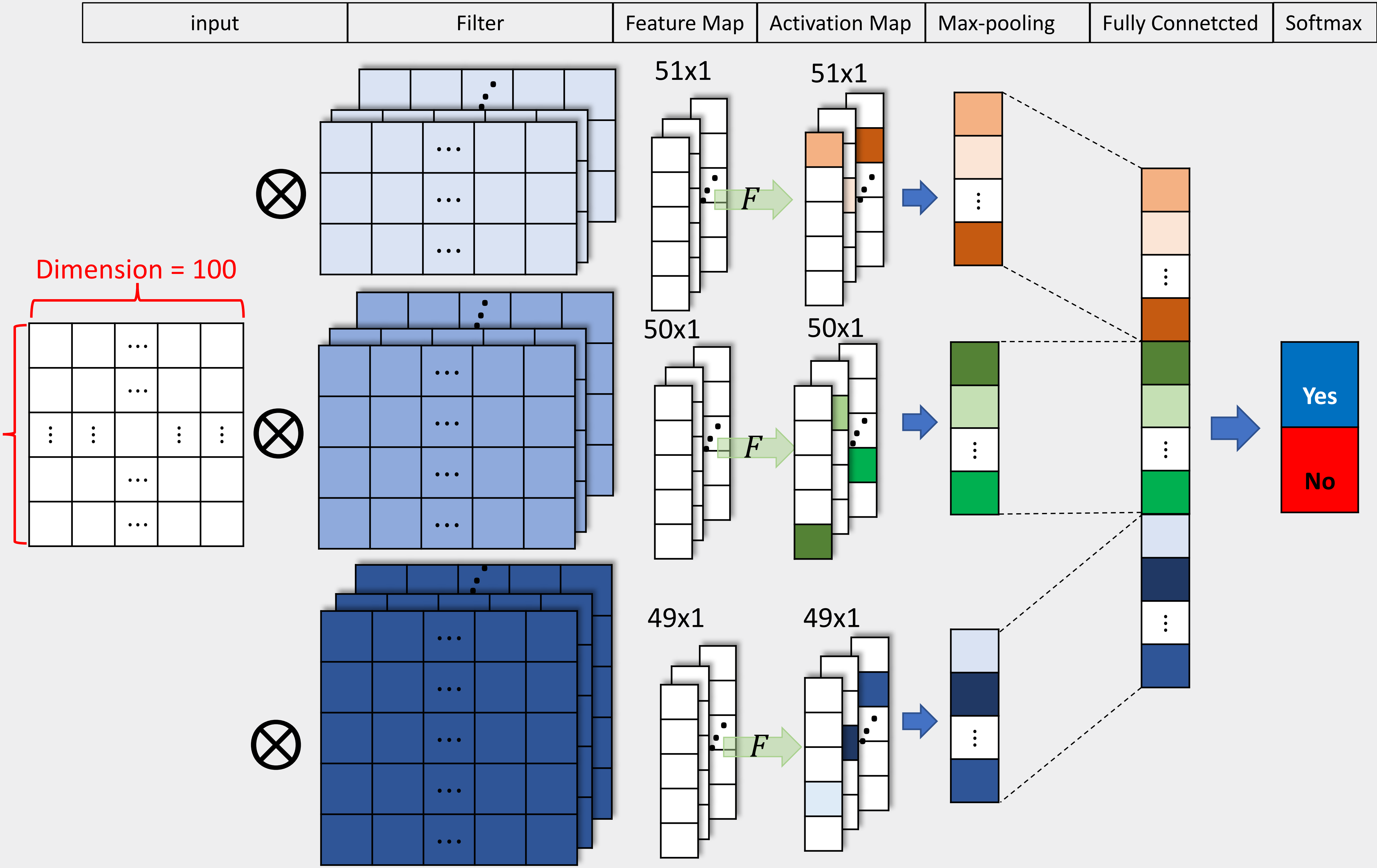
➢TextCNN architecture



➢TextCNN architecture in our Model

- Embedding dimension : 100차원
- Filter size = 3, 4, 5
- Filter channel = 10
- Number of class = 2 (Yes, No)
- Activation function = ReLU
- Max Pooling
- Dropout = 40%
- Optimizer = Adam
- Epoch = 3

➢TextCNN architecture in our Model

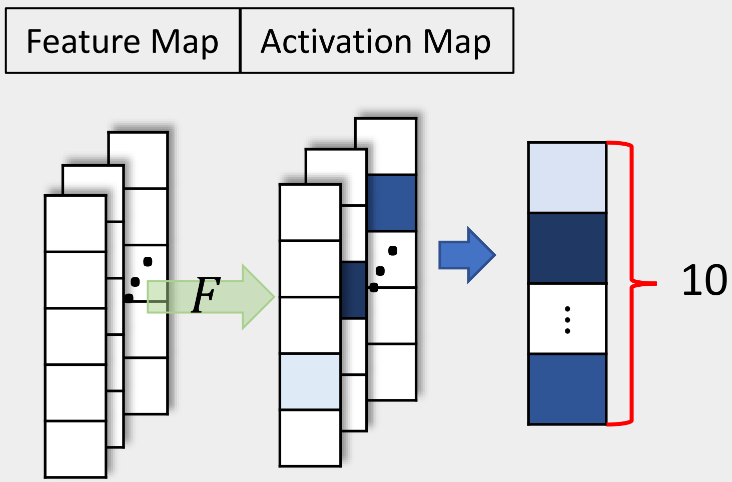


➢TextCNN 모델링

```python
class TextCNN(nn.Module):
    def __init__(self, vocab_built, emb_dim, dim_channel, kernel_wins, num_class):
        super(TextCNN, self).__init__()
        self.embed = nn.Embedding(len(vocab_built), emb_dim)
        self.embed.weight.data.copy_(vocab_built.vectors)
        self.convs = nn.ModuleList([nn.Conv2d(1, dim_channel, (w, emb_dim)) for w in kernel_wins])
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.4)
        self.fc = nn.Linear(len(kernel_wins)*dim_channel, num_class)

```

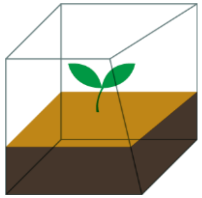
```python
model = TextCNN(vocab, 100, 10, [3, 4, 5], 2).to(device)
```



➤결과

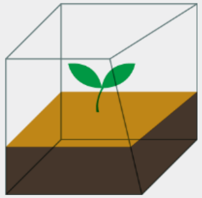
```
TextCNN(
  (embed): Embedding(32536, 100)
  (convs): ModuleList(
    (0): Conv2d(1, 10, kernel_size=(3, 100), stride=(1, 1))
    (1): Conv2d(1, 10, kernel_size=(4, 100), stride=(1, 1))
    (2): Conv2d(1, 10, kernel_size=(5, 100), stride=(1, 1))
  )
  (relu): ReLU()
  (dropout): Dropout(p=0.4, inplace=False)
  (fc): Linear(in_features=30, out_features=2, bias=True)
)
train Epoch : 1          Loss : 0.08380897547794772      Accuracy : 59.13166427612305%
Valid Epoch : 1          Loss : 0.07966495078053533      Accuracy : 66.19451904296875%
model saves at 66.19451904296875 accuracy
---------------------------------------------------------
train Epoch : 2          Loss : 0.0750463566742398       Accuracy : 68.21614074707031%
Valid Epoch : 2          Loss : 0.07656122036313426      Accuracy : 66.4778060913086%
model saves at 66.4778060913086 accuracy
---------------------------------------------------------
train Epoch : 3          Loss : 0.05591068643797057      Accuracy : 79.66021728515625%
Valid Epoch : 3          Loss : 0.07916408421944851      Accuracy : 67.23323822021484%
model saves at 67.23323822021484 accuracy
---------------------------------------------------------
```

Q&A



2021. 11. 03

Min Jun Kim



THANK YOU.

2021. 11. 03

Min Jun Kim