Very Deep Convolutional Networks for Text Classification

2019. 02. 19.

1. 등장 배경

Convolutional Neural Networks for Sentence Classification (Kim, 2014, **1** layer, 3438 citations)

-> text분류도 CNN을 써보자, 이때 word단위로 tokenizing하고 **wordvec을 pretrained으로** 쓰면 잘된다.

Character-level Convolutional Networks for Text Classification (Zhang, 2015, **6** layers, 948 citations)

-> 토크나이징을 word단위가 아니라 **char단위로** 하고, 좀더 쌓아보자.

Very Deep Convolutional Networks for Text Classification (Conneau, 2016, **29** layers, 292 citations)

-> 인간의 **rule**이 섞이긴 했지만, text도 image처럼 **hierarchical**한 정보들로 구성되어있으니(char, token, words, phrase,.. sentence, paragraph, document..), computer vision에서 처럼 **작은 conv를 깊게** 쌓아서, **vggNet(19 layers), resNet(152 layers)**같이 text encoder로서의 성능효과를 볼수 있을 것 같다. **성능이 나아지긴 했지만, SOTA**를 아주 크게 뛰어넘는 정도는 아니었다.

실험 결과

This has the consequence that each example induces less gradient information which may make it harder to train large architectures.

대량 데이터
120k ~ 3.6M

참고로 ImageNet은 class가 1000개

Data set	#Train	#Test	#Classes	Classification Task
~~AG's news~~	120k	7.6k	4	English news categorization
~~Sogou news~~	450k	60k	5	Chinese news categorization
DBPedia	560k	70k	14	Ontology classification
Yelp Review Polarity	560k	38k	2	Sentiment analysis
Yelp Review Full	650k	50k	5	Sentiment analysis
Yahoo! Answers	1 400k	60k	10	Topic classification
Amazon Review Full	3 000k	650k	5	Sentiment analysis
Amazon Review Polarity	3 600k	400k	2	Sentiment analysis

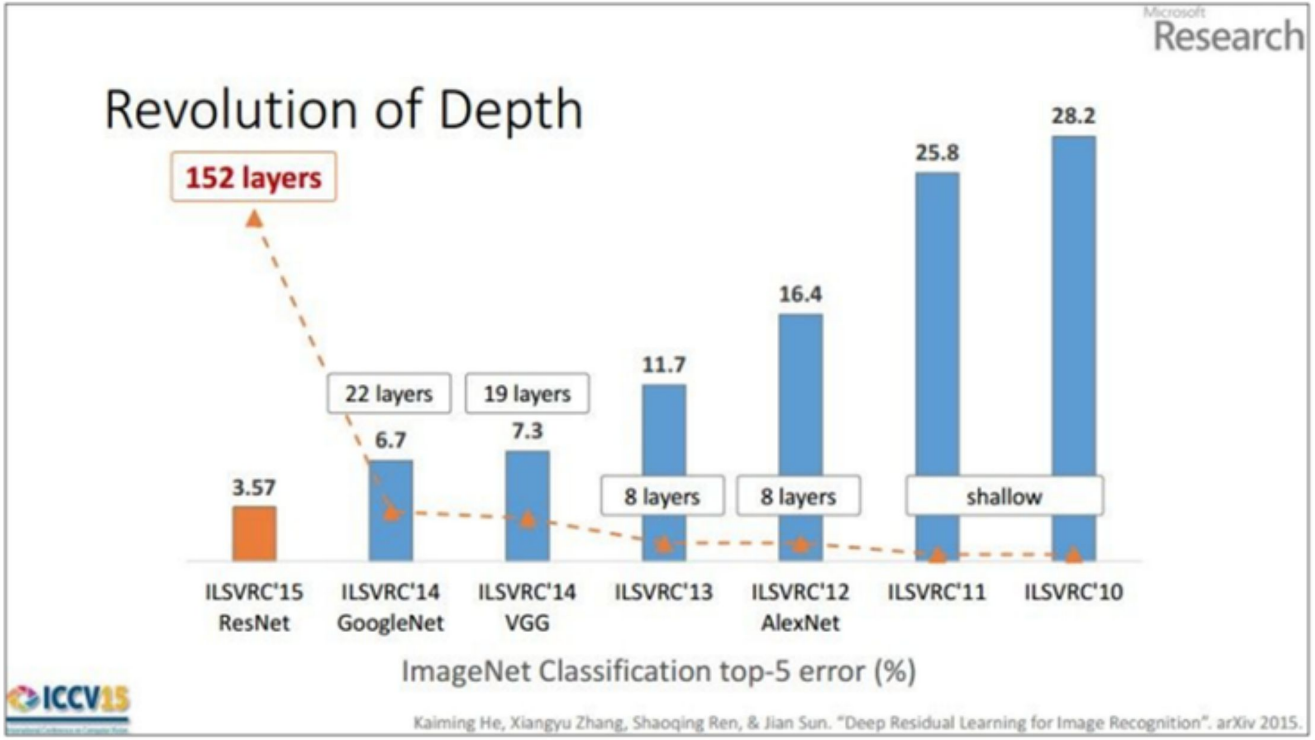
Some of the tasks are very ambiguous, in particular sentiment analysis for which it is difficult to clearly associate fine grained labels

Table 3: Large-scale text classification data sets used in our experiments. See (Zhang et al., 2015) for a detailed description.

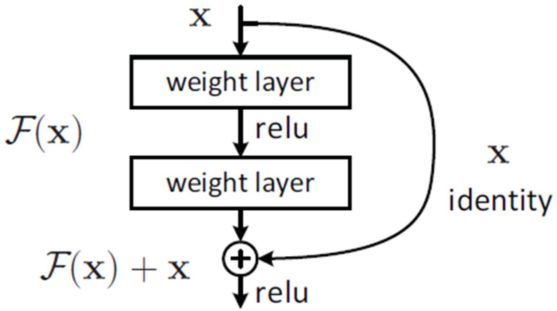
Depth	Pooling	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
9	Convolution	10.17	4.22	1.64	5.01	37.63	28.10	38.52	4.94
9	KMaxPooling	9.83	3.58	1.56	5.27	38.04	28.24	39.19	5.69
9	MaxPooling	9.17	3.70	1.35	4.88	36.73	27.60	37.95	4.70
17	Convolution	9.29	3.94	1.42	4.96	36.10	27.35	37.50	4.53
17	KMaxPooling	9.39	3.51	1.61	5.05	37.41	28.25	38.81	5.43
17	MaxPooling	8.88	3.54	1.40	4.50	36.07	27.51	37.39	4.41
29	Convolution	9.36	3.61	1.36	4.35	**35.28**	27.17	37.58	**4.28**
29	KMaxPooling	**8.67**	**3.18**	1.41	4.63	37.00	27.16	38.39	4.94
29	MaxPooling	8.73	3.36	**1.29**	**4.28**	35.74	**26.57**	**37.00**	4.31

Table 5: Testing error of our models on the 8 data sets. No data preprocessing or augmentation is used.

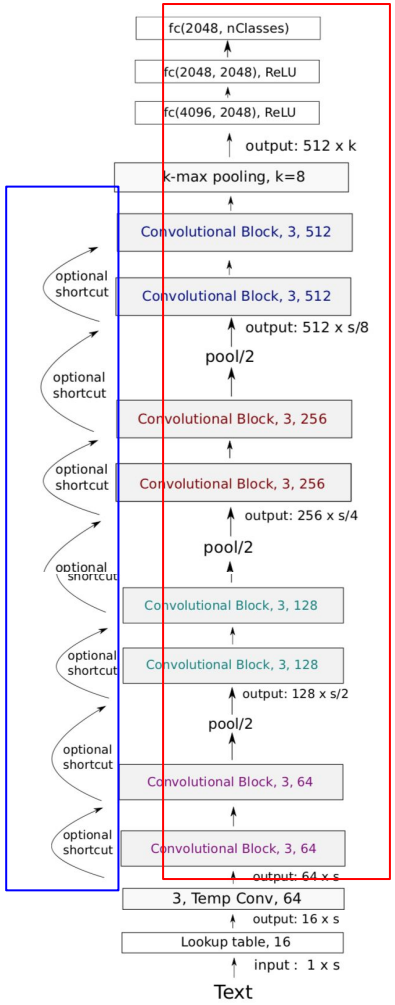
Vision 아키텍쳐 히스토리



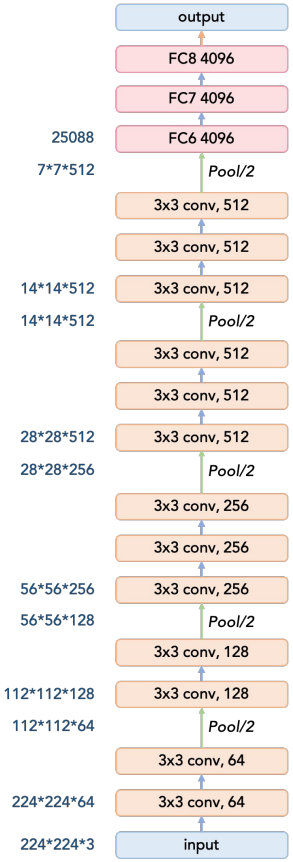
아키텍쳐



Residual
Block/Res Net



VDCNN



(k, s)	(k, s)'	receptive field
7 x 1	404 x 32	404 x 404
2 x 2	212 x 32	212 x 212
3 x 1	196 x 16	196 x 196
3 x 1	164 x 16	164 x 164
3 x 1	132 x 16	132 x 132
2 x 2	100 x 16	100 x 100
3 x 1	82 x 8	92 x 92
3 x 1	76 x 8	76 x 76
3 x 1	60 x 8	60 x 60
2 x 2	44 x 8	44 x 44
3 x 1	40 x 4	40 x 40
3 x 1	32 x 4	32 x 32
3 x 1	24 x 4	24 x 24
2 x 2	16 x 4	16 x 16
3 x 1	14 x 2	14 x 14
3 x 1	10 x 2	10 x 10
2 x 2	6 x 2	6 x 6
3 x 1	5 x 1	5 x 5
3 x 1	3 x 1	3 x 3

VGG Net

http://zike.io/posts/calculate-receptive-field-for-vgg-16/

VGG Net

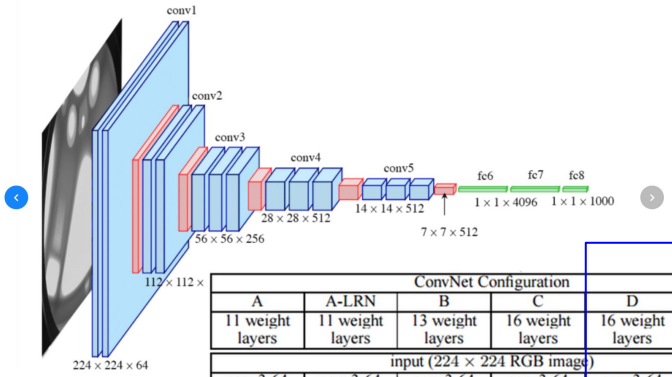


Fig. A1. The standard VGG-16 netwo
the feature extractor.

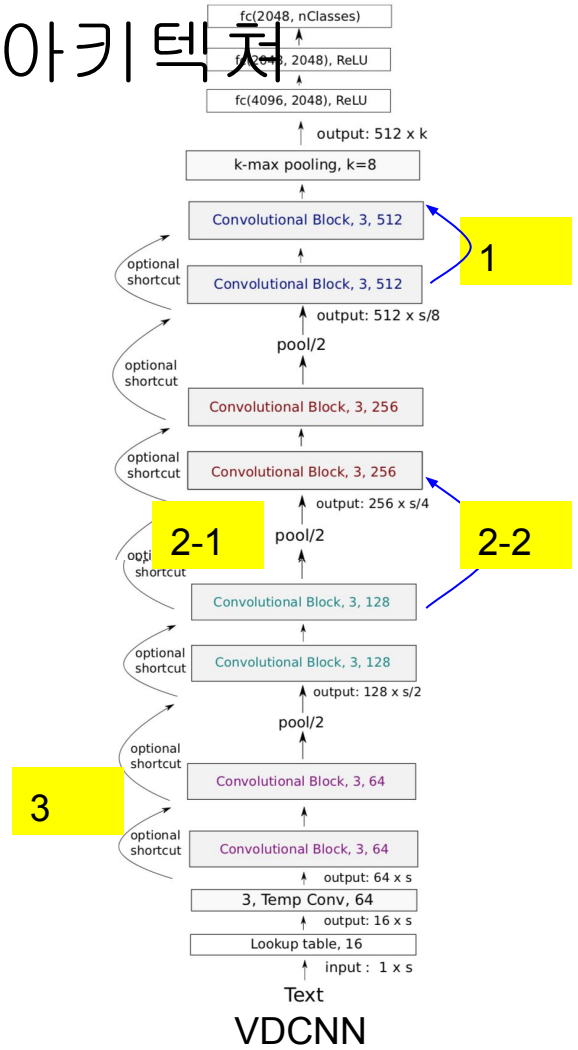
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- AlexNet과 구조는 비슷하지만, **작은 conv를 깊게** 쌓은 논문
- pooling사이에 Conv 여러층을 stack했다
- **3*3 conv를 2 layer쌓으면 5*5 conv와 같은 효과, 적은 파라미터, 많은 layer사이에 non-linearity 증가로 feature추출에 용이**

- 깊어서 학습이 잘 안되니까, 얕은 구조로 별도 분류기를 만들어서 학습결과를 초기값으로 사용
- FC 3층으로 파라미터가 아주 많다
- Scale jittering : training scale을 바꿔가면서 학습
- Multi-crop : data augmentation
- Dense evaluation

https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture-as-proposed-in-32-Note-that-only_fig3_322512435

아키텍처



VDCNN

- 1 + 2 x (2+2+2+2) = 17 층의 convolution layers의 architecture
-
- **두 가지 design rule - VGG/ResNet 참고하여 temporal adapt**
  - 1
  - 2
  - **같은** resolution(s, time seq, length)을 가진 output이 출력되면, layer의 feature map 수는 **같다**
  - resolution이 **절반이** 되면, layer의 feature map 수를 **두배해준다**
    - **=> 메모리 사용량을 줄여줌**
- **3번의 pooling, 할때마다 temporal resolution을 절반으로,**
  - feature map은 3레벨로 됨, 128, 256, 512
-
- 기존에는 6층이 최고였지만, 이 방법을 통하여 더 많이 내려갈 수 있게 됨
- 기존에는 3-gram, 5-gram, 7-gram등 다양한 사이즈의 conv를 병렬적으로 썼다.
  - 이 모델에서는 3 conv를 4개 쌓아서 9-gram까지 본다
- 3
-
- **optional shortcut** : resNet의 방식대로 적용
  - **identity and 1 × 1 convolutions**

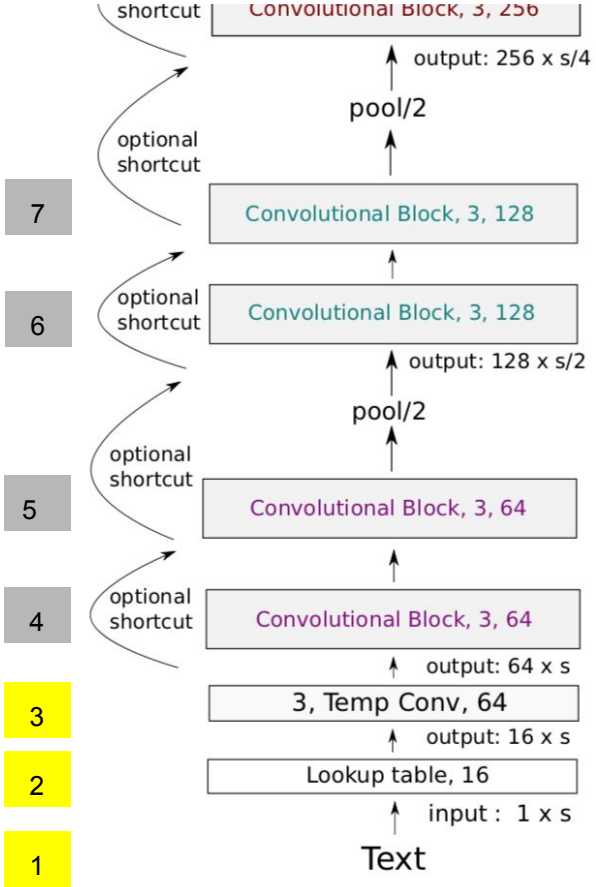
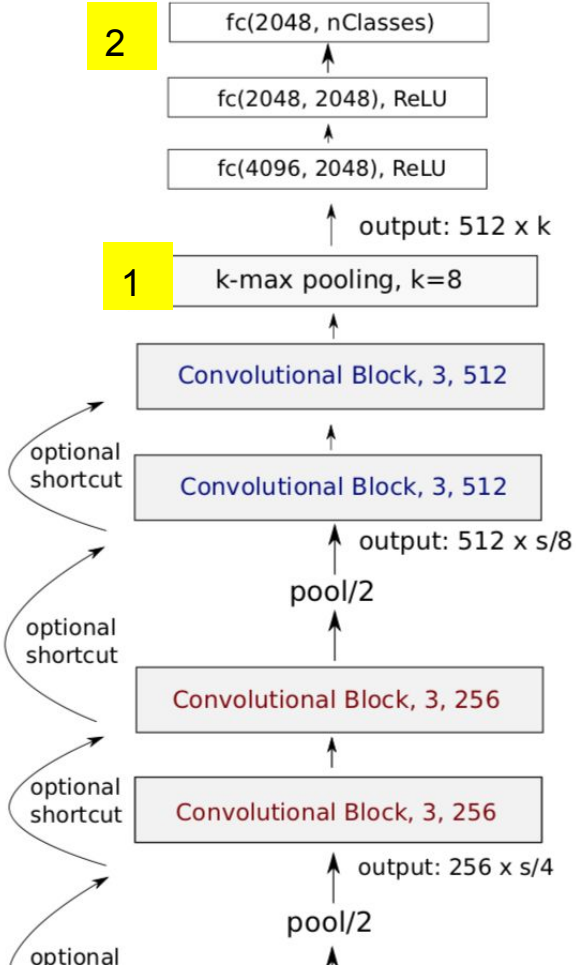


Figure 1: VDCNN architecture.

1 input (1 x s)
- ex) “good morning”
- s는 character의 갯수, 패딩을 붙여서 1024로 고정

2 Look-up-table (16 x s)
- input layer 역할, 2D tensor 생성
- size = (in_ch, s)
  - In_ch = f0(논문) = input text의 차원 = 16 = RGB 역할

3 Temp Conv 64 x(in_ch x 3)
- VGG 처럼 64 convolution layer of size 3
- conv1d (in_ch = f0 = 16, out_ch=64, kernel_size=3)



1. K-max pooling

Max-pooling 또는 k-max pooling을 이용해서

Conv block output 의 temporal resolution을

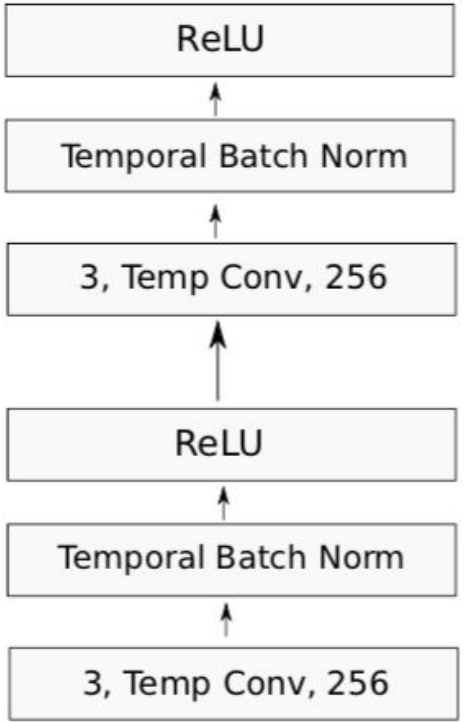
Fixed dimension으로 down-sample한다.

- 마지막에 k-max pooling을 통하여 512 x k개의 feature를 뽑아낸 후, 벡터로 만들어 fully connected ReLU classifier로 분류

2. FC

We do **not use drop-out** with the fully connected layers,
but only **temporal batch normalization** after convolutional layers to **regularize** our network.

Conv Block



- **연속 2개의 convolution layers**, each followed by a **temporal Batch Norm** and **ReLU**

- filter의 크기가 작아서 필요한 parameter가 적기 때문에 convolution layer를 통하여 network의 depth를 많이 늘리는 것이 가능
- 전체적인 architecture에서 depth 조절을 convolutional block 개수를 통해 조절

```
class ConvolutionalBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, first_stride=1):
        super(ConvolutionalBlock,self).__init__()
        self.sequential = nn.Sequential(
            nn.Conv1d(in_channels, out_channels,
                      kernel_size=kernel_size, stride=first_stride, padding=1),
            nn.BatchNorm1d(num_features=out_channels),
            nn.ReLU(),

            nn.Conv1d(out_channels, out_channels,
                      kernel_size=kernel_size, stride=1, padding=1),
            nn.BatchNorm1d(num_features=out_channels),
            nn.ReLU()
        )

    def forward(self, x):
        return self.sequential(x)
```

https://github.com/dreamgonfly/deep-text-classification-pytorch/blob/master/models/VDCNN.py

Temporal Batch Norm

- **Temporal batch normalization** applies the same kind of regularization as batch normalization except that the **activations in a mini-batch are jointly normalized over temporal (instead of spatial) locations**.
- BatchNorm2d
  - `(N, C, H, W)`
  - `Because the Batch Normalization is done over the **`C` dimension**, computing statistics on `(N, H, W)` slices, it's common terminology to call this **Spatial** Batch Normalization.`
- BatchNorm1d
  - `(N, C, L)`
  - `Because the Batch Normalization is done over the **`C` dimension**, computing statistics on `(N, L)` slices, it's common terminology to call this **Temporal** Batch Normalization.`

https://pytorch.org/docs/stable/_modules/torch/nn/modules/batchnorm.html

실험 결과

This has the consequence that each example induces less gradient information which may make it harder to train large architectures. QQ

대량 데이터
120k ~ 3.6M

참고로 ImageNet은 class가 1000개

Data set	#Train	#Test	#Classes	Classification Task
AG's news	120k	7.6k	4	English news categorization
Sogou news	450k	60k	5	Chinese news categorization
DBPedia	560k	70k	14	Ontology classification
Yelp Review Polarity	560k	38k	2	Sentiment analysis
Yelp Review Full	650k	50k	5	Sentiment analysis
Yahoo! Answers	1 400k	60k	10	Topic classification
Amazon Review Full	3 000k	650k	5	Sentiment analysis
Amazon Review Polarity	3 600k	400k	2	Sentiment analysis

Table 3: Large-scale text classification data sets used in our experiments. See (Zhang et al., 2015) for a detailed description.

Some of the tasks are very ambiguous, in particular sentiment analysis for which it is difficult to clearly associate fine grained labels

Depth	Pooling	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
9	Convolution	10.17	4.22	1.64	5.01	37.63	28.10	38.52	4.94
9	KMaxPooling	9.83	3.58	1.56	5.27	38.04	28.24	39.19	5.69
9	MaxPooling	9.17	3.70	1.35	4.88	36.73	27.60	37.95	4.70
17	Convolution	9.29	3.94	1.42	4.96	36.10	27.35	37.50	4.53
17	KMaxPooling	9.39	3.51	1.61	5.05	37.41	28.25	38.81	5.43
17	MaxPooling	8.88	3.54	1.40	4.50	36.07	27.51	37.39	4.41
29	Convolution	9.36	3.61	1.36	4.35	**35.28**	27.17	37.58	**4.28**
29	KMaxPooling	**8.67**	**3.18**	1.41	4.63	37.00	27.16	38.39	4.94
29	MaxPooling	8.73	3.36	**1.29**	**4.28**	35.74	**26.57**	**37.00**	4.31

Table 5: Testing error of our models on the 8 data sets. No data preprocessing or augmentation is used.

3개의 depth 비교 : 9, 17, 29, ~~49~~

- 깊게 쌓을 수록 잘된다
- Absolute Accuracy 3.43% 향상 (data augmentation도 하지 않음)

3가지 down-sampling 비교

성능 : **Max-pooling** > **conv with strides 2** >>>>>>>>>>> **k-max pooling**

We explore three types of down-sampling between blocks K_i and K_{i+1} (Figure 1) :

(i) The first convolutional layer of K_{i+1} has stride 2 (ResNet-like).

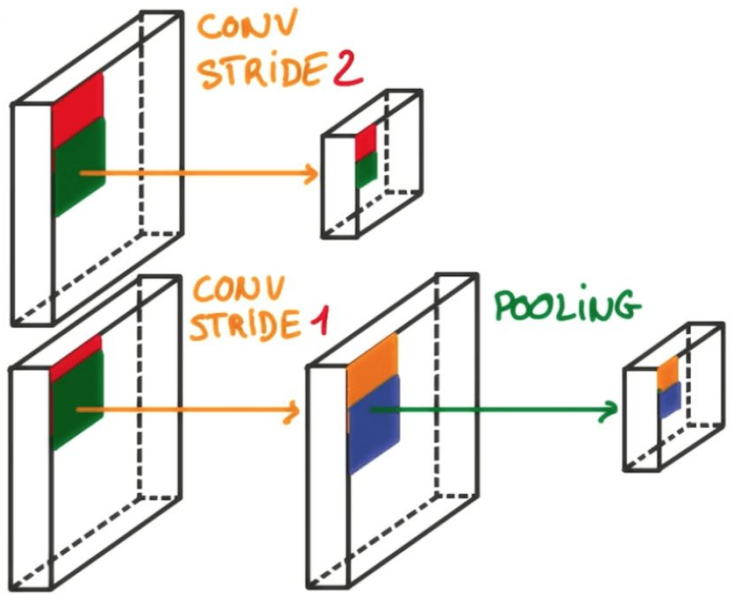
(ii) K_i is followed by a k-max pooling layer where k is such that the resolution is halved (Kalchbrenner et al., 2014).

(iii) K_i is followed by max-pooling with kernel size 3 and stride 2 (VGG-like).

1. Conv with strides 2
   - local하게 연속적인 3개의 token을 본다
2. K-max pooling
   - 문장의 전체를 1번에 본다
   - 중간 layer에서 이것을 하면 성능이 떨어진다
   - 아주 작은 데이터셋에서는 예외적으로 잘 됨
3. (Temporal) max-pooling
   - Small depth에서는 temporal max-pooling 이 더 효과가 좋다
   - local하게 연속적인 3개의 token을 본다

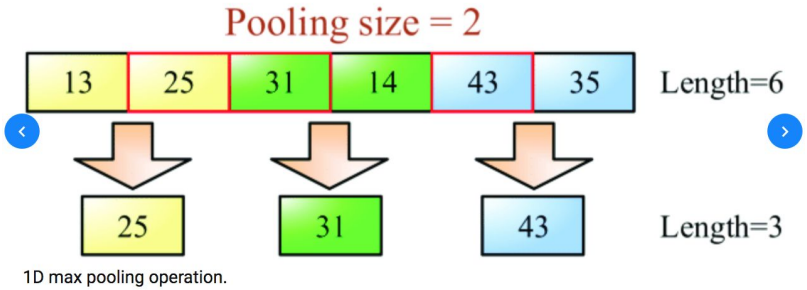
Convolutional with strides 2

- POOLING

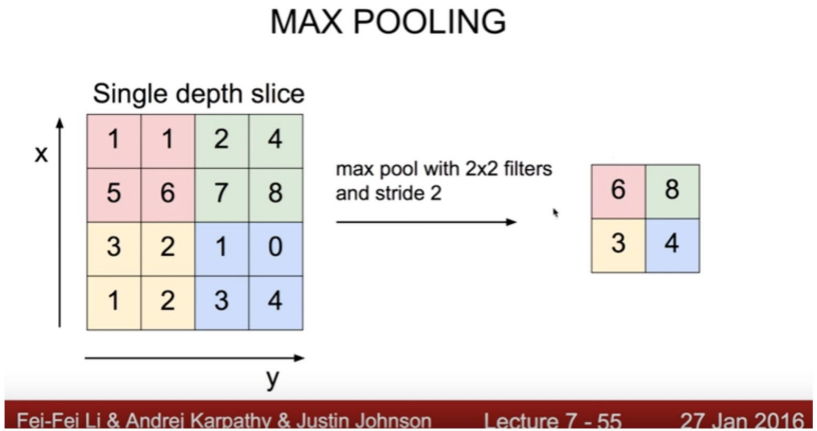


https://www.ritchieng.com/machine-learning/deep-learning/convs

(temporal) max pooling



1D max pooling operation.



Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 55 27 Jan 2016

Max-pooling 1D : temporal max pooling

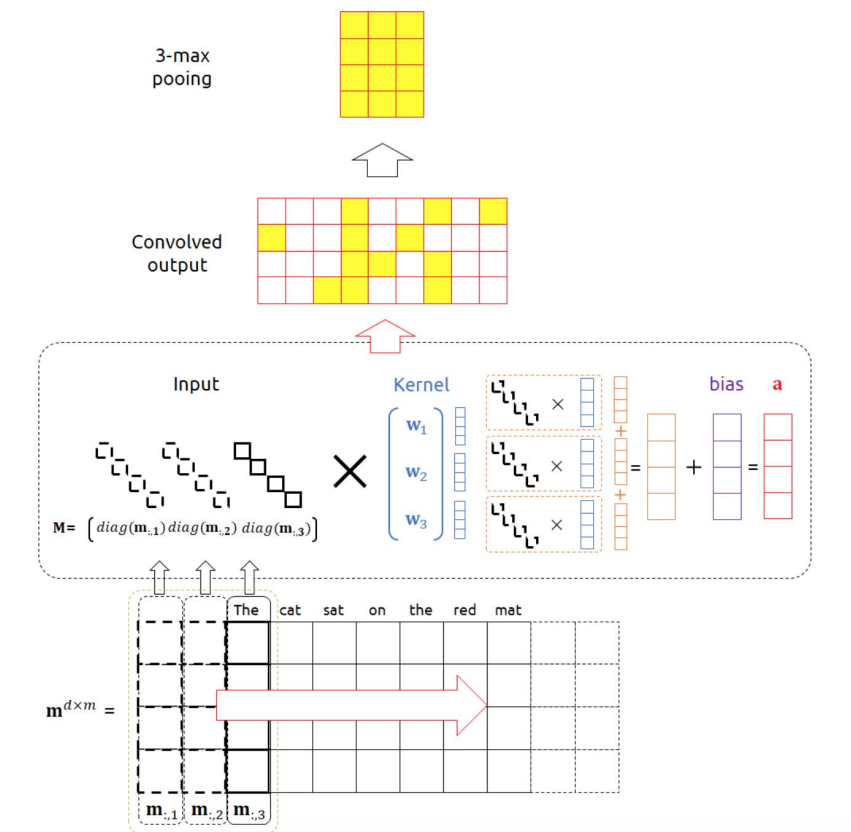
```
keras.layers.MaxPooling1D(pool_size=2, strides=None,
padding='valid', data_format='channels_last')
```

Max-pooling 2D : spatial max pooling

ex) `keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)`

http://cs231n.stanford.edu/
https://www.researchgate.net/figure/1D-max-pooling-operation_fig4_324177888
https://keras.io/layers/pooling/

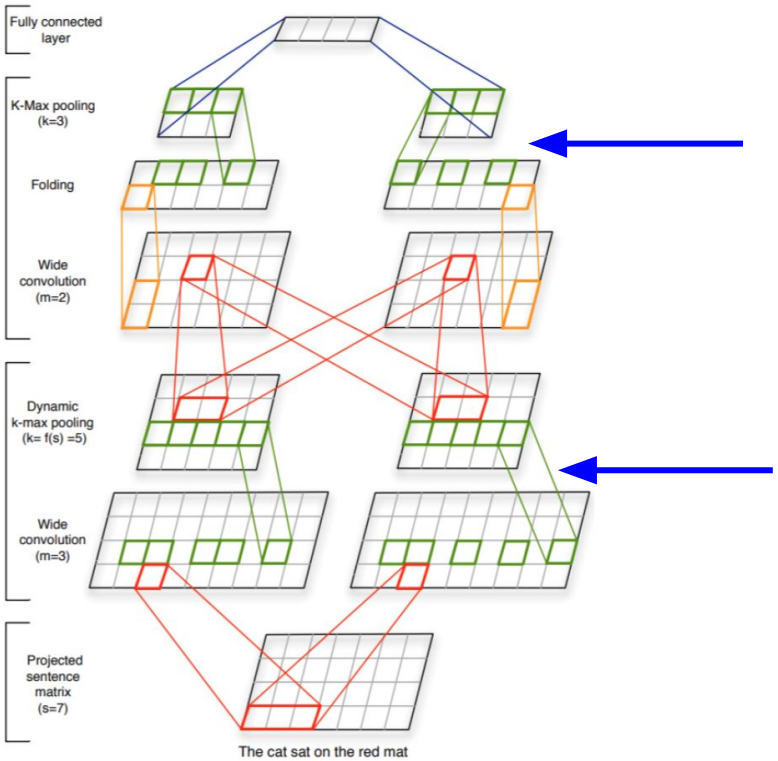
K-max pooling



먼저 k-max pooling이란 convolve된 아웃풋의 각 row에서 가장 값이 높은 k개로 압축하는 것을 의미한다. 예를 들어 k=3일 경우 좌측 그림처럼 표현할 수 있다

https://donghwa-kim.github.io/week_02_01.html

Dynamic k-max pooling



$$k_l = \max\left(k_{top}, \lceil \frac{L-l}{L} \times s \rceil\right)$$

Dynamic k-Max Pooling이란 k가 고정된 값이 아니라, 각 layer에 따라서 바뀌는것을 의미한다.

l_{th} layer마다 변화하는 k_l은 위의 식과 같다.

L은 총 layer의 갯수를 의미하며, l: 현재 layer가 몇번째인지를 나타낸다. 따라서 위 수식은 layer이 얕을수록, k를 input sequence 길이 만큼 가져가고, layer가 깊을수록, k를 사용자가 정의한 값 (hyper-parameters)의 k_{top}으로 다이나믹하게 layer마다 변화된다.

https://donghwa-kim.github.io/week_02_01.html

References

https://arxiv.org/pdf/1606.01781.pdf

https://github.com/YBIGTA/DeepNLP-Study/wiki/Very-Deep-CNN(VDCNN)-for-NLP

https://github.com/dreamgonfly/deep-text-classification-pytorch/blob/master/models/VDCNN.py

https://donghwa-kim.github.io/week_02_01.html

https://m.blog.naver.com/laonple/220738560542