Convolutional Neural Networks for Text Classification

2018-03-31 `convolutional-neural-networks` **(/tag/convolutional-neural-networks)** `document-classification` **(/tag/document-classification)** `deep-learning` **(/tag/deep-learning)** `neural-networks` **(/tag/neural-networks)**

Convolutional Neural Networks (ConvNets) have in the past years shown break-through results in some NLP tasks, one particular task is sentence classification, i.e., classifying short phrases (i.e., around 20~50 tokens), into a set of pre-defined categories. In this post I will explain how ConvNets can be applied to classifying short-sentences and how to easily implemented them in Keras.

You can find the complete code associated with this blog post on this repository:

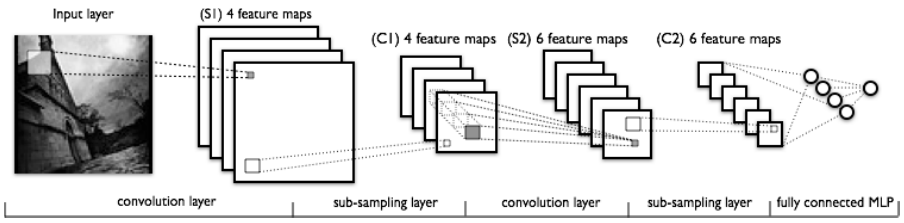
https://github.com/davidsbatista/ConvNets-for-sentence-classification (https://github.com/davidsbatista/ConvNets-for-sentence-classification)

Convolutional Neural Networks

ConvNets were initially developed in the neural network image processing community where they achieved break-through results in recognising an object from a pre-defined category (e.g., cat, bicycle, etc.).

A Convolutional Neural Network typically involves two operations, which can be though of as feature extractors: **convolution** and **pooling**.

The output of this sequence of operations is then typically connected to a fully connected layer which is in principle the same as the traditional multi-layer perceptron neural network (MLP).



The Convolutional Neural Network architecture applied to image classification.
(Image adapted from http://deeplearning.net/)

Convolutions

We can think about the input image as a matrix, where each entry represents each pixel, and a value between 0 and 255 representing the brightness intensity. Let's assume it's a black and white image with just one **channel (https://www.wikiwand.com/en/Channel_(digital_image))** representing the grayscale. If you would be processing a colour image, and taking into account the colours one would have 3 channels, following the **RGB colour mode (https://www.wikiwand.com/en/RGB_color_model)**.

One way to understand the convolution operation is to imagine placing the **convolution filter** or **kernel** on the top of the input image, positioned in a way so that the **kernel** and the image upper left corners coincide, and then multiplying the values of the input image matrix with the corresponding values in the **convolution filter**.

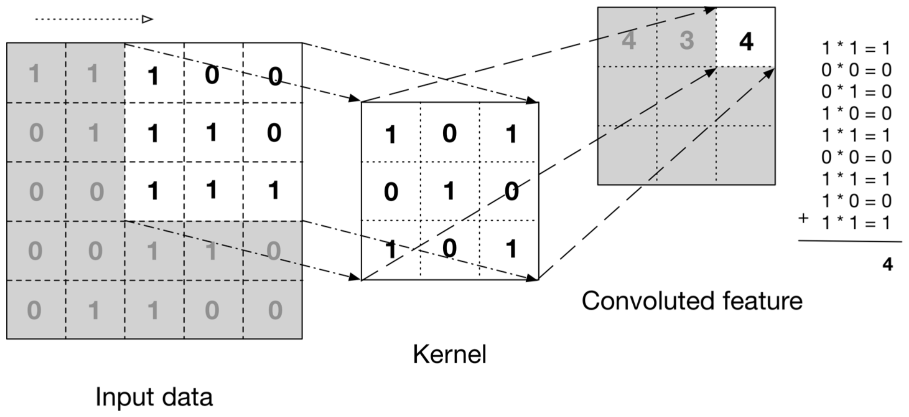
All of the multiplied values are then added together resulting in a single scalar, which is placed in the first position of a result matrix.

The **kernel** is then moved x pixels to the right, where x is denoted **stride length** and is a parameter of the ConvNet structure. The process of multiplication is then repeated, so that the next value in the result matrix is computed and filled.

This process is then repeated, by first covering an entire row, and then shifting down the columns by the same **stride length**, until all the entries in the input image have been covered.

The output of this process is a matrix with all it's entries filled, called the **convoluted feature** or **input feature map**.

An input image can be convolved with multiple convolution kernels at once, creating one output for each kernel.



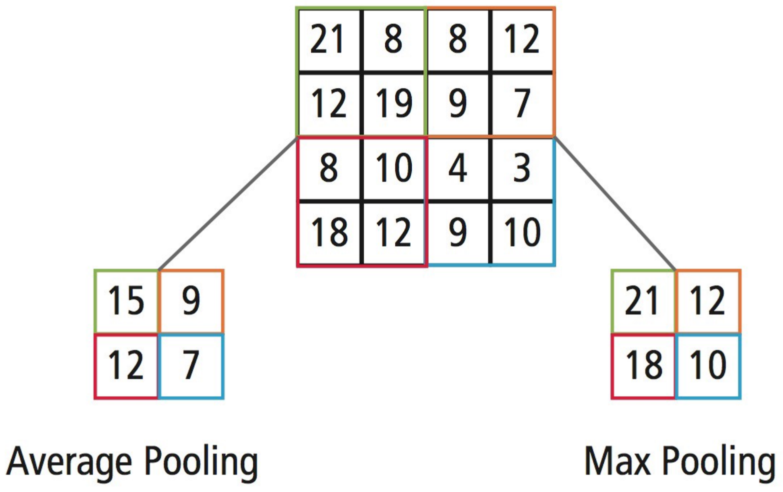
Example of a convolution operation.
(Image adapted from *"Deep Learning"* by Adam Gibson, Josh Patterson)

Pooling

Next comes the **pooling** or **downsampling** layer, which consists of applying some operation over regions/patches in the **input feature map** and extracting some representative value for each of the analysed regions/patches.

This process is somehow similar to the convolution described before, but instead of transforming local patches via a learned linear transformation (i.e., the **convolution filter**), they're transformed via a hardcoded operation.

Two of the most common pooling operations are max- and average-pooling. **Max-pooling** selects the maximum of the values in the **input feature map** region of each step and **average-pooling** the average value of the values in the region. The output in each step is therefore a single scalar, resulting in significant size reduction in output size.



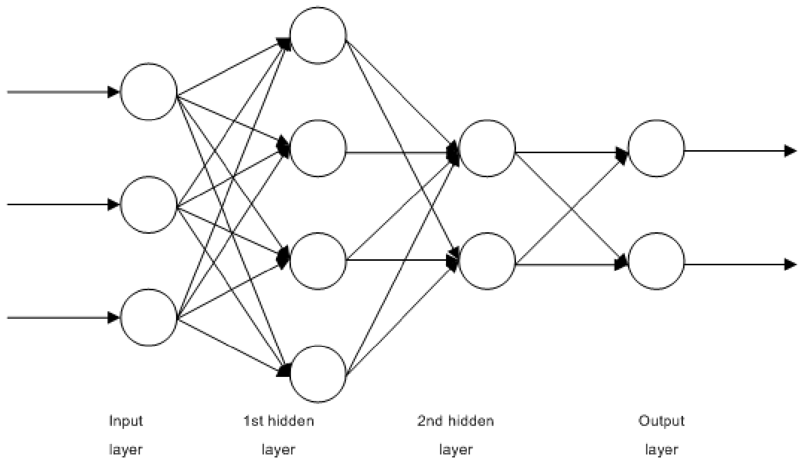
Example of a pooling operation with stride length of 2.
(Image adapted from **AJ Cheng blog (https://medium.com/@Aj.Cheng/convolutional-neural-network-d9f69e473feb)**)

Why do we downsample the feature maps and simply just don't remove the pooling layers and keep possibly large feature maps? François Chollet in *"Deep Learning with Python"* summarises it well in this sentence:

"The reason to use downsampling is to reduce the number of feature-map coefficients to process, as well as to induce spatial-filter hierarchies by making successive convolution layers look at increasingly large windows (in terms of the fraction of the original input they cover)."

Fully Connected

The two processes described before i.e.: convolutions and pooling, can been thought of as a feature extractors, then we pass this features, usually as a reshaped vector of one row, further to the network, for instance, a multi-layer perceptron to be trained for classification.



Example of multi-layer perceptron network used to train for classification.

This was a briefly description of the ConvNet architecture when applied to image processing, let's now see how we can adapt this architecture to Natural Language Processing tasks.

Convolutional Neural Networks for NLP

In the case of NLP tasks, i.e., when applied to text instead of images, we have a 1 dimensional array representing the text. Here the architecture of the ConvNets is changed to 1D convolutional-and-pooling operations.

One of the most typically tasks in NLP where ConvNet are used is sentence classification, that is, classifying a sentence into a set of pre-determined categories by considering n-grams, i.e. it's words or sequence of words, or also characters or sequence of characters.

1-D Convolutions over text

Given a sequence of words $w_{1:n} = w_1, \dots, w_n$, where each is associated with an embedding vector of dimension d. A 1D convolution of width-k is the result of moving a sliding-window of size k over the sentence, and applying the same **convolution filter** or **kernel** to each window in the sequence, i.e., a dot-product between the concatenation of the embedding vectors in a given window and a weight vector u, which is then often followed by a non-linear activation function g.

Considering a window of words $w_i, \dots, w_{i+k}$ the concatenated vector of the ith window is then:

$$x_i = [w_i, w_{i+1}, \dots, w_{i+k}] \in R^{\,k \times d}$$

The **convolution filter** is applied to each window, resulting in scalar values r_i, each for the ith window:

$$r_i = g(x_i \cdot u) \in R$$

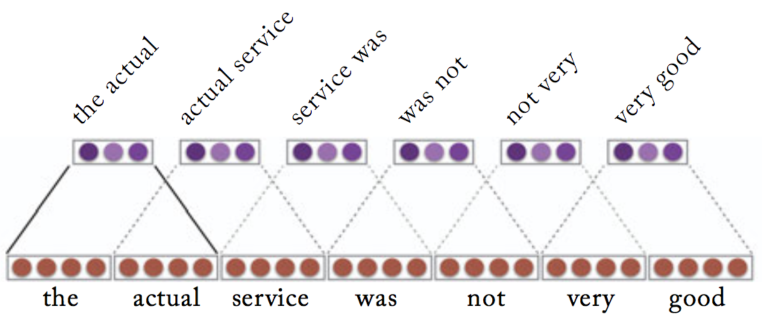
In practice one typically applies more filters, $u_1, \ldots, u_l$, which can then be represented as a vector multiplied by a matrix U and with an addition of a bias term b:

$$\mathrm{r}_i = g(x_i \cdot U + b)$$

with

$$\mathrm{r}_i \in R^l, \quad x_i \in R^{k \times d}, \quad U \in R^{k \cdot d \times l} \text{ and } \quad b \in R^l$$

An example of a sentence convolution in a vector-concatenation notation:



Example of a sentence convolution with

k

=2 and dimensional output

l

=3.
(Image adapted from **Yoav Goldberg (http://u.cs.biu.ac.il/~yogo/)** book "Neural Network Methods for NLP")

Channels

In the introduction above I assumed we were processing a black and white image, and therefore we have one matrix representing the grayscale intensity of each pixel. With the **RGB colour mode (https://www.wikiwand.com/en/RGB_color_model)** each pixel would be a combination of three intensity values instead, one for each of Red, Green and Blue components, and such representation would be stored in three different matrices, providing different characteristics or view of the image, referred to as a **Channel (https://www.wikiwand.com/en/Channel_(digital_image))**. It's common to apply a different set of filters to each channel, and then combine the three resulting vectors into a single vector.

We can also apply the multiple channels paradigm in text processing as well. For example, for a given phrase or window of text, one channel could be the sequence of words, another channel the sequence of corresponding POS tags, and a third one the shape of the words:

Word:	**The**	**plane**	**lands**	**in**	**Lisbon**
PoS-tag:	**DET**	**NOUN**	**VERB**	**PROP**	**NOUN**
Shape:	**Xxx**	**xxxx**	**xxxx**	**xx**	**Xxxxxx**

Applying the convolution over the words will result in m vectors w, applying it over the PoS-tags will result also in m vectors, and the same for the shapes, again m vectors. These three different channels can then be combined either by summation:

$$p_i = words_{1:m} + pos_{1:m} + shapes_{1:m}$$

or by concatenation:

$$p_i = [words_{1:m} : pos_{1:m} : shapes_{1:m}]$$

.

NOTE: each channel can still have different convolutions that read the source document using different kernel sizes, for instance, applying different context windows over words, pos-tags or shapes.

Pooling

The pooling operation is used to combine the vectors resulting from different convolution windows into a single l-dimensional vector. This is done again by taking the *max* or the *average* value observed in resulting vector from the convolutions. Ideally this vector will capture the most relevant features of the sentence/document.

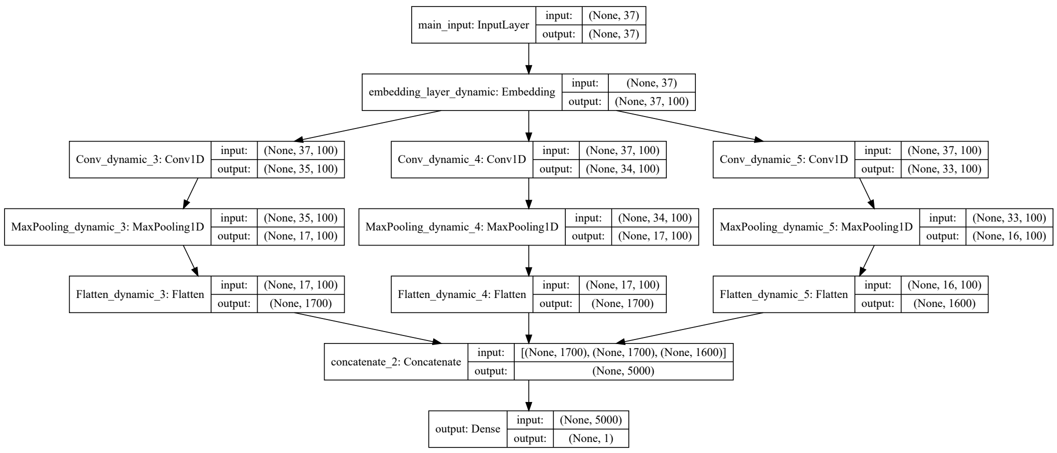
This vector is then fed further down in the network - hence, the idea that ConvNet itself is just a feature extractor - most probably to a full connected layer to perform prediction.

Convolutional Neural Networks for Sentence Classification

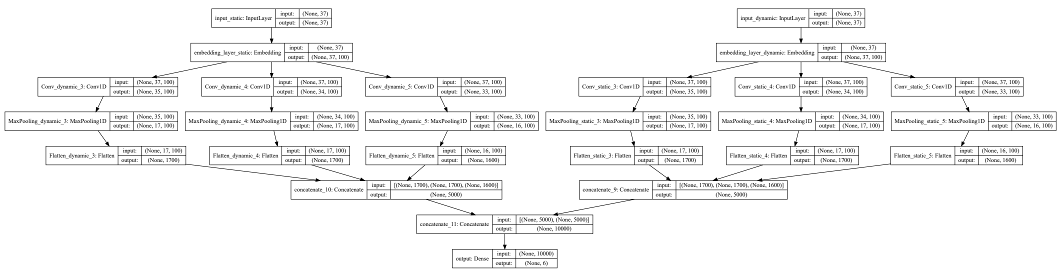
I did a quick experiment, based on the paper by Yoon Kim, implementing the 4 ConvNets models he used to perform sentence classification.

- **CNN-rand**: all words are randomly initialized and then modified during training
- **CNN-static**: pre-trained vectors with all the words— including the unknown ones that are randomly initialized—kept static and only the other parameters of the model are learned
- **CNN-non-static**: same as CNN-static but word vectors are fine-tuned
- **CNN-multichannel**: model with two sets of word vectors. Each set of vectors is treated as a channel and each filter is applied

Let's just first quickly look at how these different models look like in as a computational graph. The first three (i.e., CNN-rand, CNN-static and CNN-non-static) look pretty much the same:


main_input: InputLayer | input: (None, 37) | output: (None, 37)
embedding_layer_dynamic: Embedding | input: (None, 37) | output: (None, 37, 100)
Conv_dynamic_3: Conv1D | input: (None, 37, 100) | output: (None, 35, 100)
Conv_dynamic_4: Conv1D | input: (None, 37, 100) | output: (None, 34, 100)
Conv_dynamic_5: Conv1D | input: (None, 37, 100) | output: (None, 33, 100)
MaxPooling_dynamic_3: MaxPooling1D | input: (None, 35, 100) | output: (None, 17, 100)
MaxPooling_dynamic_4: MaxPooling1D | input: (None, 34, 100) | output: (None, 17, 100)
MaxPooling_dynamic_5: MaxPooling1D | input: (None, 33, 100) | output: (None, 16, 100)
Flatten_dynamic_3: Flatten | input: (None, 17, 100) | output: (None, 1700)
Flatten_dynamic_4: Flatten | input: (None, 17, 100) | output: (None, 1700)
Flatten_dynamic_5: Flatten | input: (None, 16, 100) | output: (None, 1600)
concatenate_2: Concatenate | input: [(None, 1700), (None, 1700), (None, 1600)] | output: (None, 5000)
output: Dense | input: (None, 5000) | output: (None, 1)

The CNN-multichannel model uses two embedding layers, in one channel the embeddings are updated, in the second they remain static. It's exactly the same network as above but duplicated and adding an extra layer do concatenate both results into a single vector:


input_static: InputLayer | input: (None, 37) | output: (None, 37)
embedding_layer_static: Embedding | input: (None, 37) | output: (None, 37, 100)
Conv_dynamic_3: Conv1D | input: (None, 37, 100) | output: (None, 35, 100)
Conv_dynamic_4: Conv1D | input: (None, 37, 100) | output: (None, 34, 100)
Conv_dynamic_5: Conv1D | input: (None, 37, 100) | output: (None, 33, 100)
MaxPooling_dynamic_3: MaxPooling1D | input: (None, 35, 100) | output: (None, 17, 100)
MaxPooling_dynamic_4: MaxPooling1D | input: (None, 34, 100) | output: (None, 17, 100)
MaxPooling_dynamic_5: MaxPooling1D | input: (None, 33, 100) | output: (None, 16, 100)
Flatten_dynamic_3: Flatten | input: (None, 17, 100) | output: (None, 1700)
Flatten_dynamic_4: Flatten | input: (None, 17, 100) | output: (None, 1700)
Flatten_dynamic_5: Flatten | input: (None, 16, 100) | output: (None, 1600)
concatenate_10: Concatenate | input: [(None, 1700), (None, 1700), (None, 1600)] | output: (None, 5000)
input_dynamic: InputLayer | input: (None, 37) | output: (None, 37)
embedding_layer_dynamic: Embedding | input: (None, 37) | output: (None, 37, 100)
Conv_static_3: Conv1D | input: (None, 37, 100) | output: (None, 35, 100)
Conv_static_4: Conv1D | input: (None, 37, 100) | output: (None, 34, 100)
Conv_static_5: Conv1D | input: (None, 37, 100) | output: (None, 33, 100)
MaxPooling_static_3: MaxPooling1D | input: (None, 35, 100) | output: (None, 17, 100)
MaxPooling_static_4: MaxPooling1D | input: (None, 34, 100) | output: (None, 17, 100)
MaxPooling_static_5: MaxPooling1D | input: (None, 33, 100) | output: (None, 16, 100)
Flatten_static_3: Flatten | input: (None, 17, 100) | output: (None, 1700)
Flatten_static_4: Flatten | input: (None, 17, 100) | output: (None, 1700)
Flatten_static_5: Flatten | input: (None, 16, 100) | output: (None, 1600)
concatenate_9: Concatenate | input: [(None, 1700), (None, 1700), (None, 1600)] | output: (None, 5000)
concatenate_11: Concatenate | input: [(None, 5000), (None, 5000)] | output: (None, 10000)
output: Dense | input: (None, 10000) | output: (None, 6)

Experiments and Results

I applied the implemented models on same of the datasets that Kim reported, but I could not get exactly the same results, first his results were reported over, I believe a Tensorflow implementation, and then there is the issue of how the datasets are pre-processed, i.e., tokenised, cleaned, etc.; that will always impact the results.

Another issue which puzzles me is that all those experiments only take into consideration the accuracy. Since the class samples are not uniformly distributed across the different classes I think this is the wrong way to evaluate a classifier.

All the code for the models and experiments is available here:

https://github.com/davidsbatista/ConvNets-for-sentence-classification (https://github.com/davidsbatista/ConvNets-for-sentence-classification)

Summary

The CNN is just a feature-extraction architecture, alone itself is not useful, but is the fist building block of a larger network. It needs to be trained together with a classification layer in order to produce some useful results.

As Yoav Goldberg summarises it:

"The CNN layer's responsibility is to extract meaningful sub-structures that are useful for the overall prediction task at hand. A convolutional neural network is designed to identify indicative local predictors in a large structure, and to combine them to produce a fixed size vector representation of the structure, capturing the local aspects that are most informative for the prediction task at hand. In the NLP case the convolutional architecture will identify n-grams that are predictive for the task at hand, without the need to pre-specify an embedding vector for each possible ngram."

- **convolution** : an operation which applies a filter to a fixed size window.
- **convolution filter** or **kernel**: a template matrix which is used in the convolution operation.
- **pooling**: combines the vectors resulting from different convolution windows into a single l-dimensional vector.
- **feature_maps** : the number of feature maps directly controls capacity and depends on the number of available examples and task complexity.

References

- **"Convolutional Neural Networks for Sentence Classification" Y. Kim 2014 in Conference on Empirical Methods in Natural Language Processing (EMNLP'14) (http://www.aclweb.org/anthology/D14-1181)**
- **"Deep Learning" by Adam Gibson, Josh Patterson (O'Reilly Media, Inc. 2017) (https://www.oreilly.com/library/view/deep-learning/9781491924570/)**
- **"Neural Network Methods for Natural Language Processing" (http://www.morganclaypoolpublishers.com/catalog_Orig/product_info.php?products_id=1056) by Yoav Goldberg (http://www.cs.biu.ac.il/~yogo/) (Morgan & Claypool Publishers 2017)**
- **"Deep Learning with Python" (https://www.manning.com/books/deep-learning-with-python) by François Chollet (https://github.com/fchollet) (Manning Publications 2017)**

tagged in: [`convolutional-neural-networks` **(/tag/convolutional-neural-networks)** `document-classification` **(/tag/document-classification)** `deep-learning` **(/tag/deep-learning)** `neural-networks` **(/tag/neural-networks)**]

(http://www.linkedin.com/shareArticle?mini=true&url=http://www.davidsbatista.net//blog/2018/03/31/SentenceClassificationConvNets/&title=Convolutional Neural Networks for Text Classification&summary=&source=http://www.davidsbatista.net/)
(https://plus.google.com/share?url=http://www.davidsbatista.net//blog/2018/03/31/SentenceClassificationConvNets/)
(https://www.facebook.com/sharer/sharer.php?u=http://www.davidsbatista.net//blog/2018/03/31/SentenceClassificationConvNets/&quote=Convolutional Neural Networks for Text Classification)

ALSO ON **WWW.DAVIDSBATISTA.NET**

Named-Entity evaluation metrics …
5 years ago · 9 comments
Named-Entity evaluation metrics based on entity-level

Evaluation Metrics, ROC-Curves and …
4 years ago · 11 comments
This blog post describes some evaluation metrics used in NLP, it points out …

Named-Entity Recognition based …
4 years ago · 7 comments
This blog post review some of the recent proposed methods to perform …

Google's SyntaxNet Python NLTK
6 years ago · 20 commer
This post shows how t
the output of SyntaxNe
Python …

12 Comments 1 **Login** ▾

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

4 Share Best Newest Oldest

Valentina lovely slut
2 months ago
Hmm Looks good...

(mailto:dsbatista@gmail.com) (https://twitter.com/DavidSBatista) (https://github.com/davidsbatista) (http://linkedin.com/in/dsbatista) (http://scholar.google.com/citations?user=) (/feed.xml)