<https://realpython.com/python-keras-text-classification/>

When you are working with sequential data, like text, you work with one dimensional convolutions, but the idea and the application stays the same. You still want to pick up on patterns in the sequence which become more complex with each added convolutional layer.

You can see that 80% accuracy seems to be tough hurdle to overcome with this data set and a CNN might not be well equipped. The reason for such a plateau might be that:

* There are not enough training samples
* The data you have does not generalize well
* Missing focus on tweaking the hyperparameters

CNNs work best with large training sets where they are able to find generalizations where a simple model like logistic regression won’t be able.

<https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/>

A standard model for document classification is to use an Embedding layer as input, followed by a one-dimensional convolutional neural network, pooling layer, and then a prediction output layer.

The kernel size in the [convolutional layer](https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/) defines the number of words to consider as the convolution is passed across the input text document, providing a grouping parameter.

A multi-channel convolutional neural network for document classification involves using multiple versions of the standard model with different sized kernels. This allows the document to be processed at different resolutions or different n-grams (groups of words) at a time, whilst the model learns how to best integrate these interpretations.

This approach was first described by Yoon Kim in his 2014 paper titled “[Convolutional Neural Networks for Sentence Classification](https://arxiv.org/abs/1408.5882).”

In the paper, Kim experimented with static and dynamic (updated) embedding layers, we can simplify the approach and instead focus only on the use of different kernel sizes.

This approach is best understood with a diagram taken from Kim’s paper:

Each channel is comprised of the following elements:

* Input layer that defines the length of input sequences.
* Embedding layer set to the size of the vocabulary and 100-dimensional real-valued representations.
* One-dimensional convolutional layer with 32 filters and a kernel size set to the number of words to read at once.
* Max Pooling layer to consolidate the output from the convolutional layer.
* Flatten layer to reduce the three-dimensional output to two dimensional for concatenation.

The output from the three channels are concatenated into a single vector and process by a Dense layer and an output layer.

The function below defines and returns the model. As part of defining the model, a summary of the defined model is printed and a plot of the model graph is created and saved to file.

<http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>

Given all this, it seems like CNNs wouldn’t be a good fit for NLP tasks. [Recurrent Neural Networks](http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/) make more intuitive sense. They resemble how we process language (or at least how we think we process language): Reading sequentially from left to right.

CNNs we don’t do that. Instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters, typically hundreds or thousands like the ones showed above, and combines their results. There’s also something something called pooling (subsampling) layers, but I’ll get into that later. During the training phase, **a CNN** **automatically learns the values of its filters** based on the task you want to perform. For example, in Image Classification a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features.

In vision, our filters slide over local patches of an image, but in NLP we typically use filters that slide over full rows of the matrix (words). Thus, the “width” of our filters is usually the same as the width of the input matrix. The height, or region size, may vary, but sliding windows over 2-5 words at a time is typical. Putting all the above together, a Convolutional Neural Network for NLP may look like this (take a few minutes and try understand this picture and how the dimensions are computed. You can ignore the pooling for now, we’ll explain that later):

<https://medium.com/voice-tech-podcast/text-classification-using-cnn-9ade8155dfb9>

**Convolution**: It is a mathematical combination of two relationships to produce a third relationship. Joins two sets of information.

**Convolution over input**: We slide over input data the convolution to extract features by applying a filter/ kernel (both can be used interchangeably). This is important in feature extraction. There are some parameters associated with that sliding filter like how much input to take at once and by what extent should input be overlapped.

* Stride: Size of the step filter moves every instance of time.
* Filter count: Number of filters we want to use.

When we are done applying the filter over input and have generated multiple feature maps, an **activation function** is passed over the output to provide a non-linear relationship for our output.

We are not done yet. We need something that helps us to reduce this high computation in the CNN and not overfit the data. Overfitting will lead the model to **memorize** the training data rather than **learning** from it.

We use a **pooling layer** in between the convolutional layers that reduces the dimensional complexity and stil keeps the significant information of the convolutions.

One example is of **max pooling** layer. It finds the maximum of the pool and sends it to the next layer as we can see in the figure below.

Sometimes a **Flatten** layer is used to convert 3-D data into 1-D vector.

In a CNN, the last layers are fully connected layers i.e. each node of one layer is connected to each node of the other layer.

<https://journals.sagepub.com/doi/full/10.1177/2053951719843310>

CNNs are composed of convolution and pooling layers, which provide an abstraction of the input. These models are employed in specific NLP tasks where the presence or absence of features is a more distinguishing factor than their location or order. For example, presence of specific words and phrases in a product review is usually indicative of it being a positive or negative review. Therefore, CNNs are well suited for the purpose of longer text classification. Neural network models have also been applied in previous work within the domain of misinformation and fake news ([Rashkin et al., 2017](https://journals.sagepub.com/doi/full/10.1177/2053951719843310); [Wang, 2017](https://journals.sagepub.com/doi/full/10.1177/2053951719843310); [Yang et al., 2017](https://journals.sagepub.com/doi/full/10.1177/2053951719843310)

<https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/>

A standard model for document classification is to use an Embedding layer as input, followed by a one-dimensional convolutional neural network, pooling layer, and then a prediction output layer.

The kernel size in the [convolutional layer](https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/) defines the number of words to consider as the convolution is passed across the input text document, providing a grouping parameter.

A multi-channel convolutional neural network for document classification involves using multiple versions of the standard model with different sized kernels. This allows the document to be processed at different resolutions or different n-grams (groups of words) at a time, whilst the model learns how to best integrate these interpretations.

A convolutional neural network is utilised in NLP as it is able to utilise the presence or absence of features, words in this case, as a distinguishing factor, instead of more intuitive approaches such as location and order within recurrent neural networks. A CNN applies convolutions over an input layer to compute the output by applying different learned filters. A standard model involves embedding, convolution, and pooling layers to give a prediction output.

Here we have the architecture of the best-performing CNN-LSTM model. The model consisted of 2 convolutional layers for interpretation of the data. The convolutional layers are interspersed with dropout layers for regularisation and maxpool is employed to downsize the data spatially. Four stacked LSTM layers process the output of the convolutions and feed into a final two fully connected layers. Average model peak performance for 10 model iterations was 90.25%. Average model epochs-to-peak accuracy was 24 epochs.

After trying multiple variations of CNN models, the most effective was found to be a multi-channel CNN model, as we can see in this case with three channels of the standard CNN model setup and 32 filters. However, it combines multiple kernel sizes which allow the CNN to process the text at different resolutions or n-grams at a time, in this case 4, 6, and 8, groups of words at a time. The highest accuracy was given for a batch size of 32, giving an accuracy of \_\_\_\_.

This model uses 32 filters and has max-pooling and dropout layers help prevent overfitting and a flatten layer to convert the data back to a one-dimensional vector.