

Customer Sentiment Analysis

Why Customer Sentiment Analysis ??

In any business, understanding the customer is the basic necessity to succeed. The user views, queries and sentiments plays a major role in delivering the users' point of view.

It is the process of determining the emotional tone behind a series of words, used to gain an understanding of the attitude, opinions, and emotions expressed.

The scope of sentiment analysis is broad any many types of business, as it helps in analyzing any survey instantly.

We will collect those embedded user's views about the products or services to sort them according to their sentiments or need using the emerging technology called “Natural Language Processing”.

SENTIMENT ANALYSIS



**Discovering people opinions, emotions and feelings about
a product or service**

Text Preprocessing:

1. A good first step when working with text is to split it into words.
2. Words are called tokens and the process of splitting text into tokens is called tokenization.

By default, this function automatically does 3 things:

3. Splits words by space (split=" ").
4. Filters out punctuation (filters='!"#\$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n').
5. Converts text to lowercase (lower=True).

```
[ ] # fit a tokenizer
def create_tokenizer(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

```
[ ] # calculate the maximum document length
def max_length(lines):
    return max([len(s.split()) for s in lines])
```

```
[ ] # encode a list of lines
def encode_text(tokenizer, lines, length):
    # integer encode
    encoded = tokenizer.texts_to_sequences(lines)
    # pad encoded sequences
    padded = pad_sequences(encoded, maxlen=length, padding='post')
    return padded
```

Tokenizer API

Keras provides a more sophisticated API for preparing text that can be fit and reused to prepare multiple text documents. This may be the preferred approach for large projects.

Keras provides the [Tokenizer class](#) for preparing text documents for deep learning. The Tokenizer must be constructed and then fit on either raw text documents or integer encoded text documents.

For example:

```
1 from keras.preprocessing.text import Tokenizer
2 # define 5 documents
3 docs = ['Well done!',
4         'Good work',
5         'Great effort',
6         'nice work',
7         'Excellent!']
8 # create the tokenizer
9 t = Tokenizer()
10 # fit the tokenizer on the documents
11 t.fit_on_texts(docs)
```

Once fit, the Tokenizer provides 4 attributes that one can use to query what has been learned about your documents:

- **word_counts:** A dictionary of words and their counts.
- **word_docs:** A dictionary of words and how many documents each appeared in.
- **word_index:** A dictionary of words and their uniquely assigned integers.
- **document_count:** An integer count of the total number of documents that were used to fit the Tokenizer.

Once the Tokenizer has been fit on training data, it can be used to encode documents in the train or test datasets.

Text tokenization utility class.

This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf...

text_to_word_sequence

Converts a text to a sequence of words (or tokens).

Arguments

- **text:** Input text (string).
- **filters:** list (or concatenation) of characters to filter out, such as punctuation. Default: `!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~`
```, includes basic punctuation, tabs, and newlines.
- **lower:** boolean. Whether to convert the input to lowercase.
- **split:** str. Separator for word splitting.

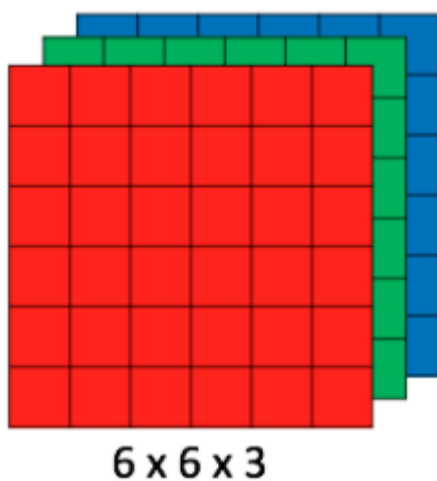
### Returns

A list of words (or tokens).

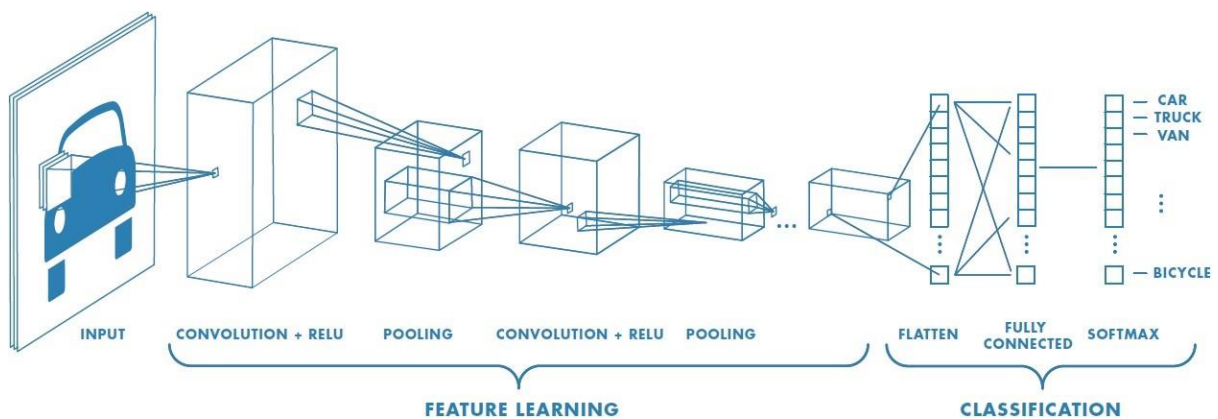
# Understanding of Convolutional Neural Network (CNN)

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see  $h \times w \times d$  (  $h$  = Height,  $w$  = Width,  $d$  = Dimension ). Eg., An image of  $6 \times 6 \times 3$  array of matrix of RGB (3 refers to RGB values) and an image of  $4 \times 4 \times 1$  array of matrix of grayscale image.



Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.



## Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f<sub>h</sub> x f<sub>w</sub> x d)**
- Outputs a volume dimension **(h - f<sub>h</sub> + 1) x (w - f<sub>w</sub> + 1) x 1**



Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

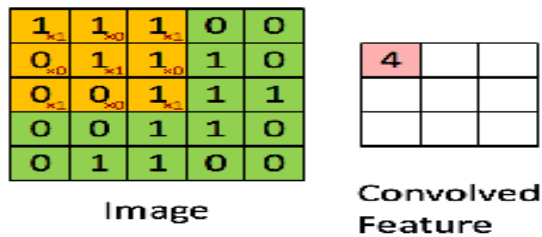
**5 x 5 – Image Matrix**

\*








|   |   |   |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

**3 x 3 – Filter Matrix**

Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called “**Feature Map**” as output shown in below



Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).

| Operation                        | Filter                                                                           | Convolved Image                                                                     |
|----------------------------------|----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| Identity                         | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$              |   |
| Edge detection                   | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$            |  |
|                                  | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$             |  |
|                                  | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$      |  |
| Sharpen                          | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$          |  |
| Box blur<br>(normalized)         | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$  |  |
| Gaussian blur<br>(approximation) | $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ |  |



## Padding

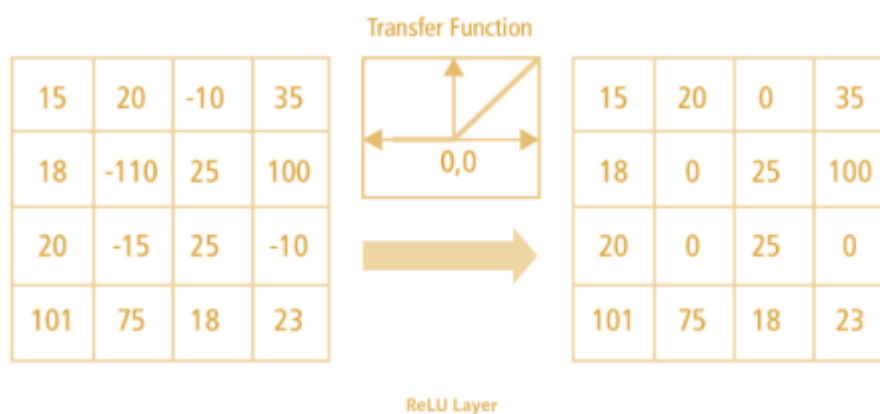
Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

## Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is  $f(x) = \max(0, x)$ .

Why ReLU is important : ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.



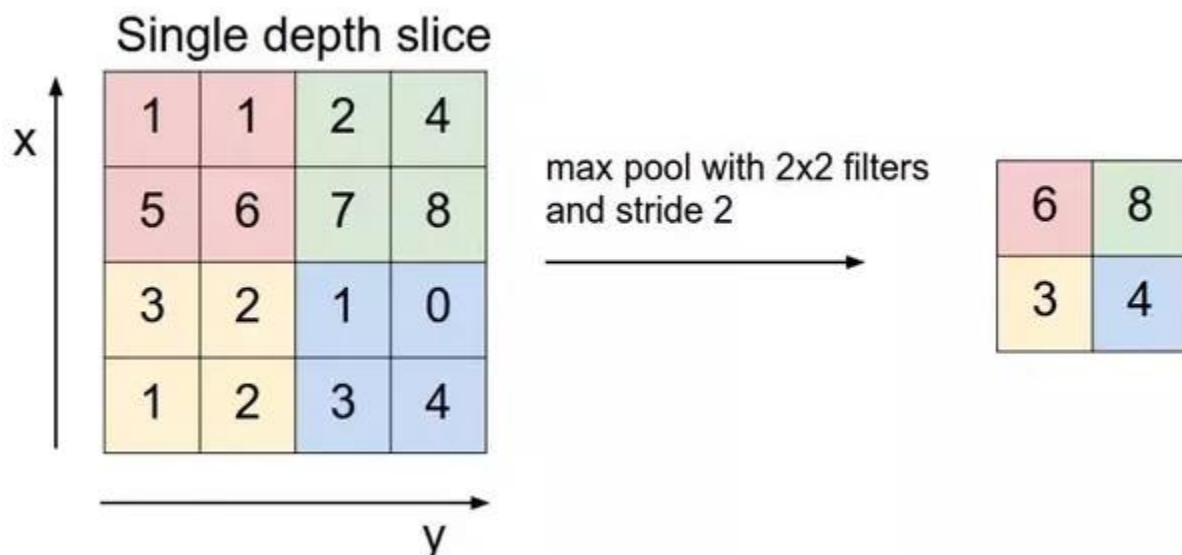
There are other non linear functions such as tanh or sigmoid can also be used instead of ReLU. Most of the data scientists uses ReLU since performance wise ReLU is better than other two.

## Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

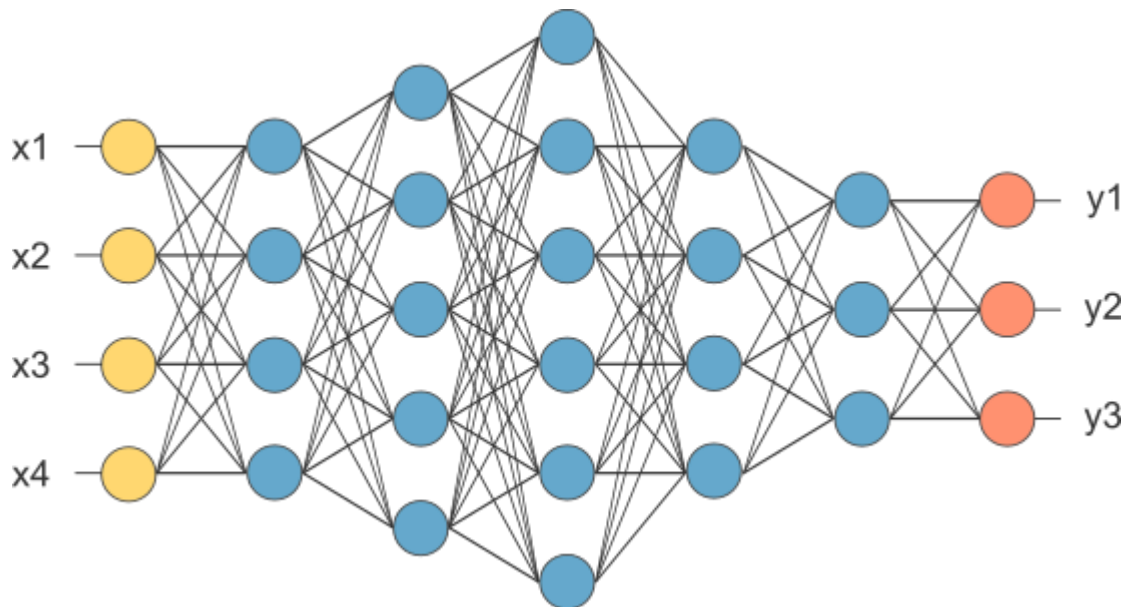
Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.





## Fully Connected Layer

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like neural network.



In the above diagram, feature map matrix will be converted as vector ( $x_1, x_2, x_3, \dots$ ). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.

