MACHINE INTELLIGENCE COMMUNITY

Compositional Data Notes

October 17th, 2017
By Jinghu Lei

1 Compositional Data

- Compositional data is the filtering of an entity into relative, qualitative aspects. This can be related to spatial, physical, or even temporal data. These different qualities can then be layered into a vector or matrix to represent an entirety.
    - For example, consider an image. Each pixel of the image can be represented using RGB data, producing a 3-dimensional vector. Thus, the entire image can be seen as a mapping of these vectors to a pixel.
- With a method of formatting relevant qualities into actual usable data, we can perform tasks such as classification.
    - However, such a data-driven task can fall to shortages due to the misrepresentations or variations when acquiring such information[1]:
        - Viewpoint
        - Scale
        - Deformation
        - Illumination
        - Class (Different variations of the same object)
- We can outline this task into four steps:
    1. Gather

        For an accurate result, a large amount of data needs to be gathered, which will be used as the training set.
    2. Parameterize

        Run the training set through a neural network, or specifically Convolutional Neural Network, and optimize the parameters.
    3. Predict

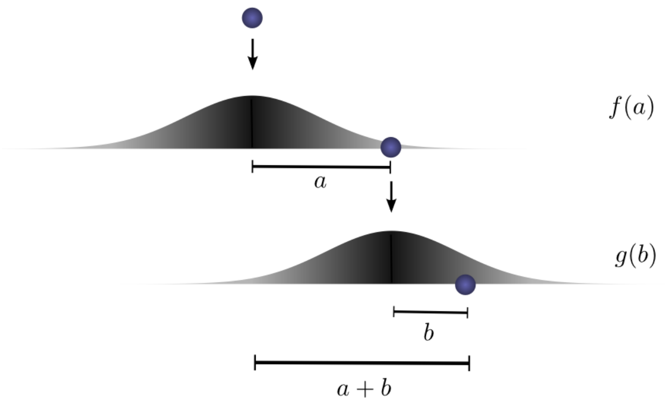
        Output a class and compare to the correct one.

2 Convolutional Neural Networks

2.1 What are convolutions

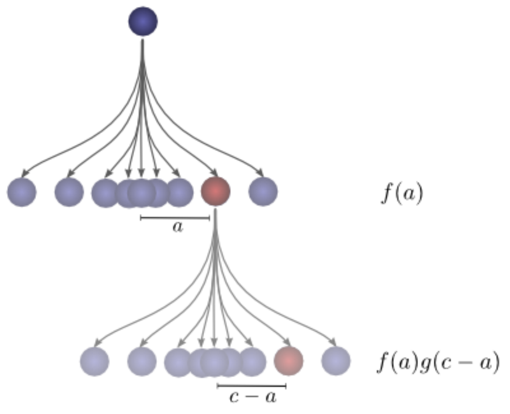
- Convolutions in mathematics are, put simply, the combining of two functions to produce a third, resulting function. This combination creates different paths to reach an output.
- An easy visualization can be seen in the analogy of a dropped ball[2]:

Consider a ball being dropped from a location. The probability of the ball landing *a* distance away is *f(a)*, where *f* is the probability distribution in this case. Then let *g(b)* be the probability of landing *b* feet away when dropped from the landing location of the initial drop.

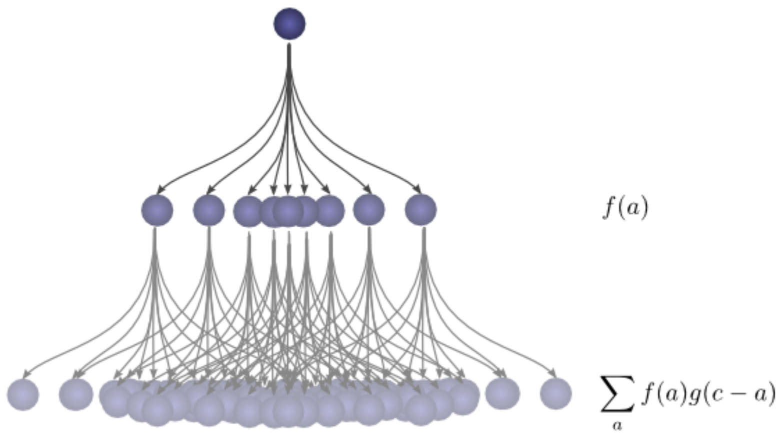


Thus, the probability of landing *a* + *b* distance away is *f(a)* · *g(b)*. Which can be condensed into $(f \cdot g)(c = a + b) = \sum_a f(a) \cdot g(c - a)$.

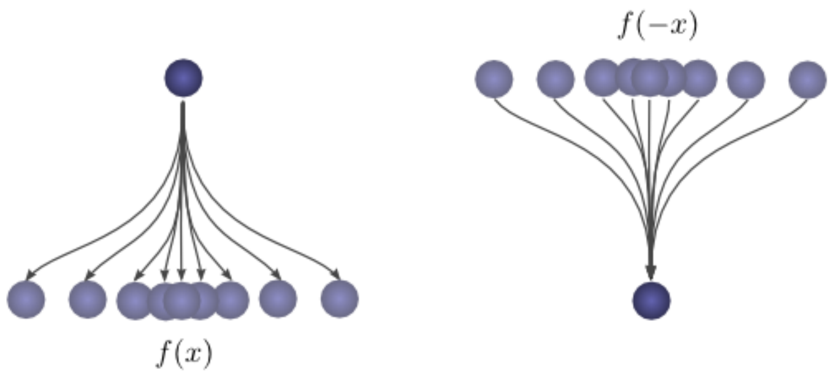
Consider a graph representing the different probabilities:



Then for an entire network it becomes:



Now for a distance x, the probability is *f(x)* and the probability of the starting position is *f(-x)*.

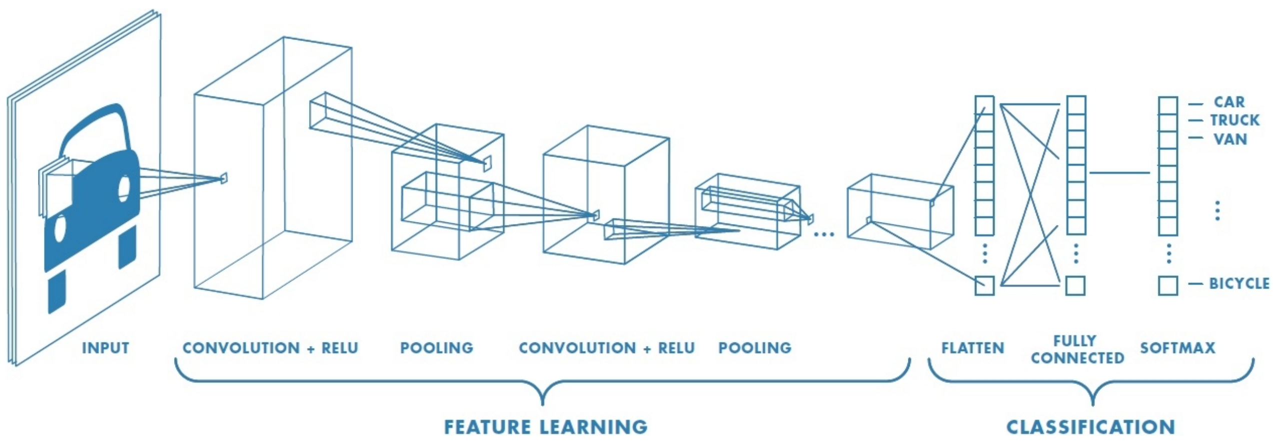


This directly represents the function of a convolutional neural network. From an input it produces such a network of probabilities to select a class.

2.2 Convolutional Neural Network Structure

- For CNNS, the combining functions are represented by the different 'layers' of the network. Each layer can be placed into these four categories:
  - Convolutional
  - Activation - ReLU
  - Pooling
  - Fully Connected

  The very end of the network is a list of possible classes in which the highest class score produced by the network is selected as the output.
- A sample CNN would look like:

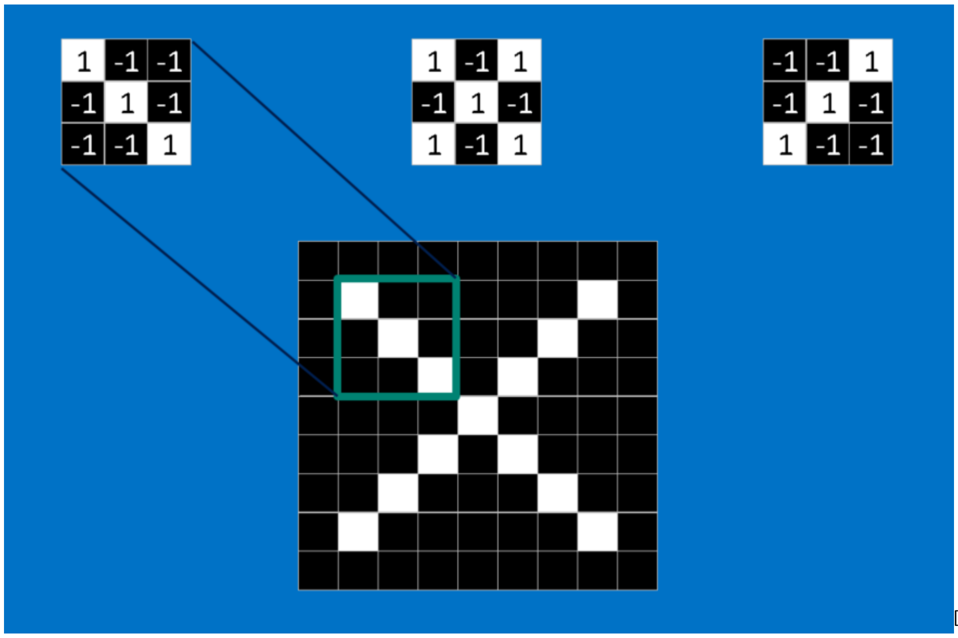


2.3 The Convolutional Layer

- The function of the convolutional layer is to produce an activation map of a given feature.

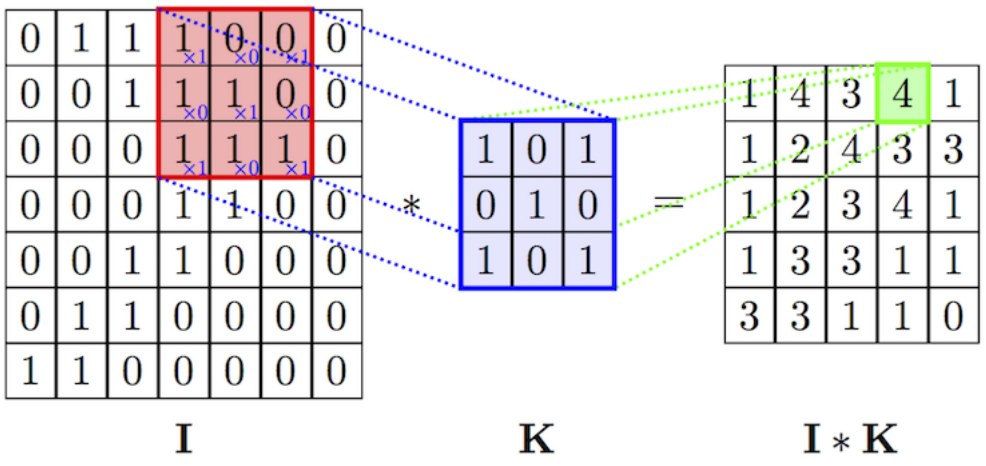
- The activation map in this case is a matrix of numbers indicating how similar the region is to the feature.
- To do this, a feature is represented as a filter, or a matrix of numbers for which the region should look like. The **filter size** is chosen depending on the feature.

  For example, to classify an ‘x’, three features and thus three filters should be used to represent the left pointing, right pointing, and cross regions.

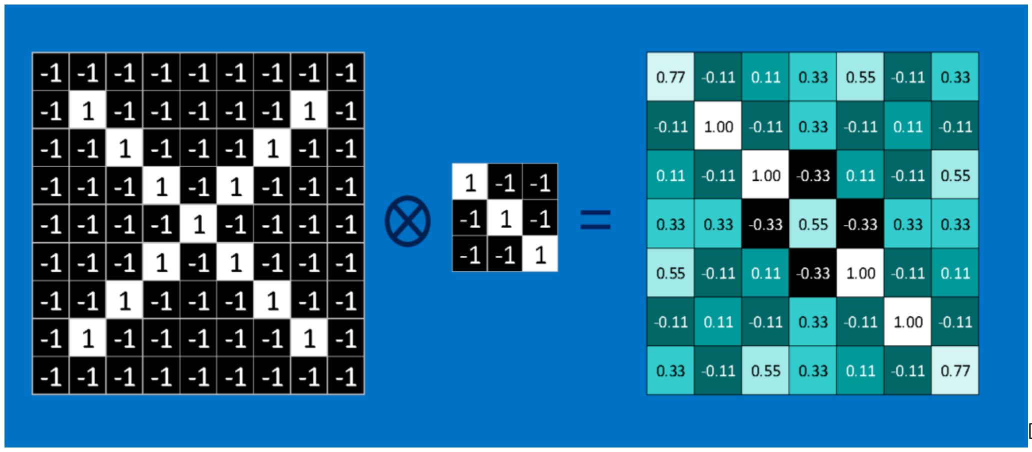


[3]

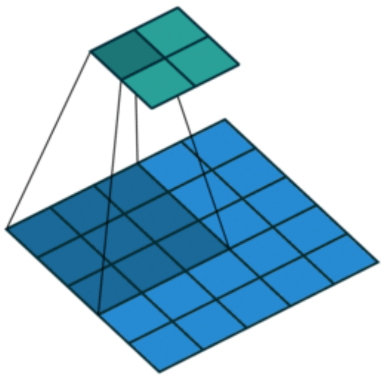
- Because the CNN does not know where a feature might occur, it compares the feature iteratively through the entire image.
  - Given one iteration, the pixels of the actual image performs a dot product with the feature, producing a single number. Intuitively, the larger the number is the more it matches the feature.
  - By comparing the image piece by piece, there is a degree of freedom allowing for transformations like rotations, deformations, and scaling. The CNN looks at the overall mapping for rough similarities in roughly similar locations in the image.
- The convolutional layer is only connected locally to the previous layer.
  - This reduces the amount of necessary weights for the entire network, another approach to reducing overfitting.



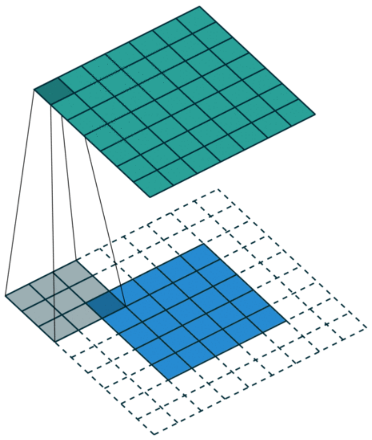
- To normalize the numbers, each pixel in the resulting matrix is divided by the total obtainable score in the filter. The activation map is then a map of numbers between 0 and 1.


[3]

- The iteration through the image can be modified through **stride** and **padding**.
  - Stride is the amount of pixels the filter moves by after completing a comparison.
  - Padding is the amount of blank (0) pixels added to the border of the image to create an even output.
- Another hyper parameter, **depth**, is the amount of features used for the convolutional layer. The mappings for each feature are stacked upon each other. Thus, the output will be a three dimension matrix, with 2 dimensions being the activation mapping of a specific feature and the third being the depth of the features.


4

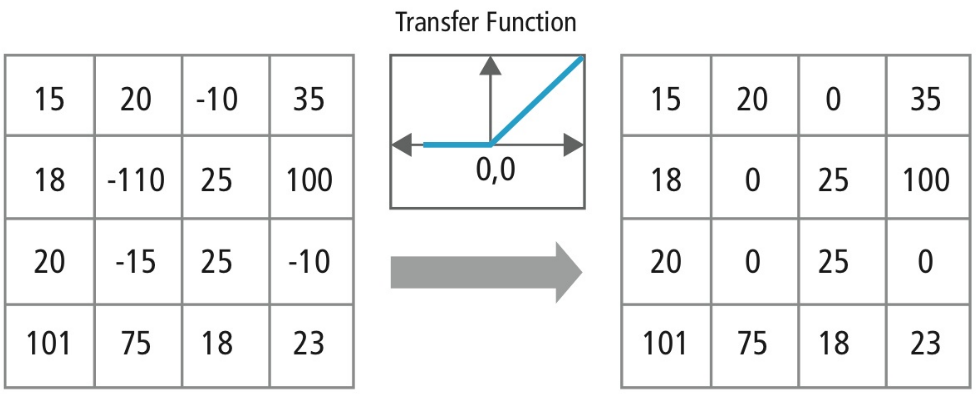
No Padding, Stride 2


[4]

Stride 1, With Padding

2.4 ReLU

- As mentioned in the Neural Networks workshop, ReLU is an activation function.
- It produces the result max(0, x), basically eliminating any negative numbers.
    - This helps the CNN not get stuck near 0 (saddle) as well as head near infinity.
- It is applied right after the completion the convolutional layer to each resulting pixel.



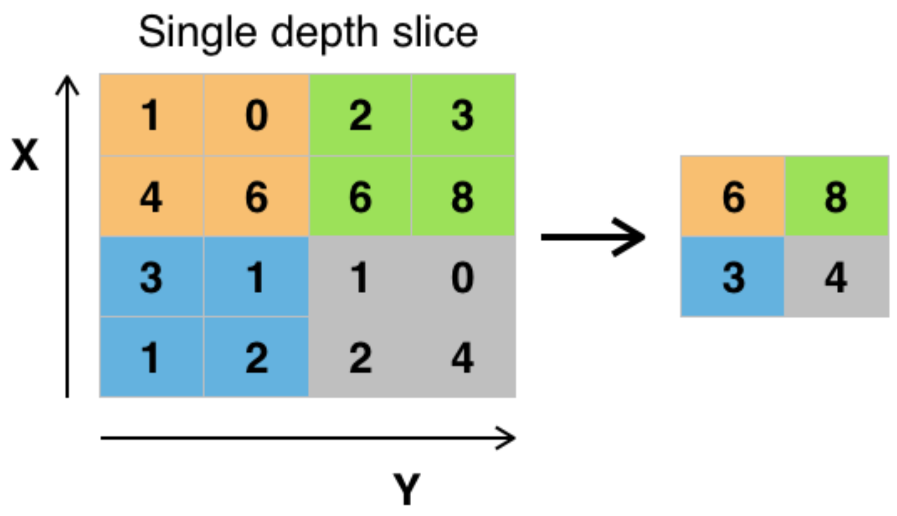
15	20	-10	35
18	-110	25	100
20	-15	25	-10
101	75	18	23

15	20	0	35
18	0	25	100
20	0	25	0
101	75	18	23

2.5 Pooling Layer

- The pooling layer further reduces the data into only the relevant information in a region. Therefore, like convolutional layers it is connected to only the local region in the previous layer.
- Similar to the convolutional layer, it iterates through the mapping but this time reducing a chosen size region to just a piece of information within it. For max pooling, this would be the maximum number.
    - The function of this is to reduce the computing necessary for especially large images, while producing a result as if it processed the entire image.

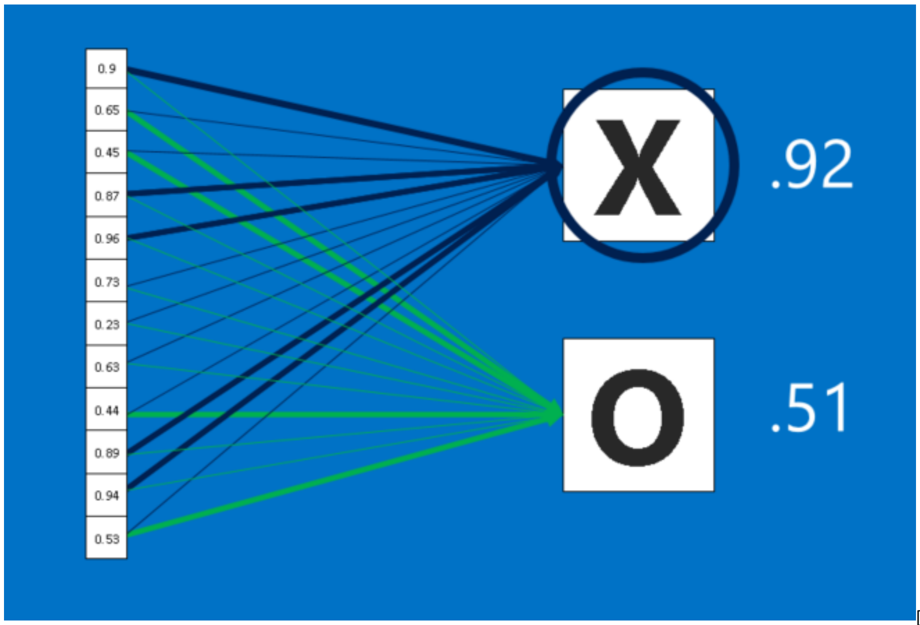
- Ultimately, this provides another degree of freedom. The feature needs to only reside within the given region not a specific location.
- In practice, a stride of 2 and a filter size of 2 or 3 is chosen.



2.7 Fully Connected Layers

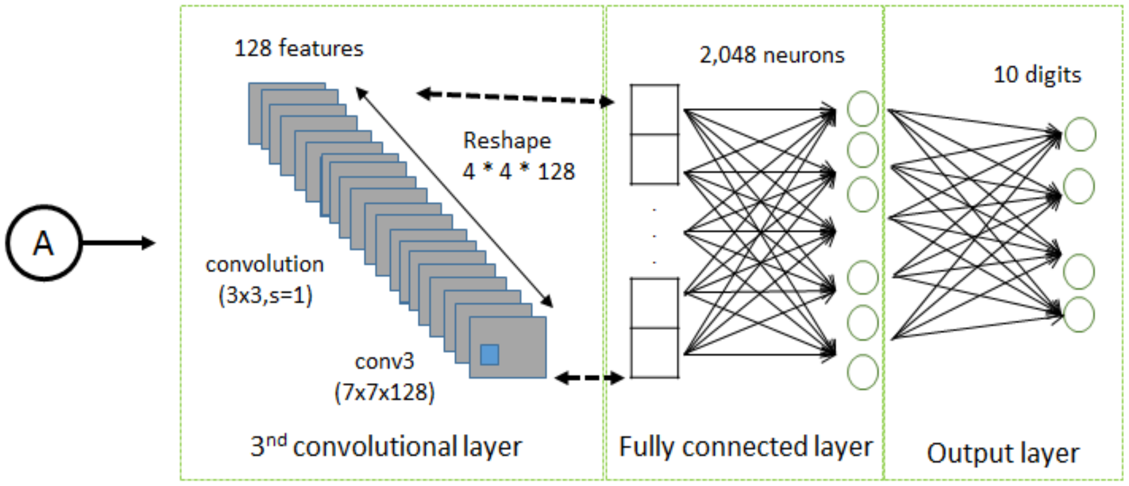
- Fully connected layers flatten the previous layer into a one dimensional vector and output another one dimensional vector representing the class scores. The largest of these are chosen as the resulting category for the input.
- As the name implies, fully connected layers are fully connected to the activated neurons in the previous layer, which are basically just elements above 0. These are given weights depending on how effective they are in figuring out which class is accurate.
  - For example, the middle pixels are more important in figuring out if an image is of an ‘X’ or ‘O’.
- Each connected neuron then gives a numerical vote, which is then altered based on the weight, to each possible category.
- Since the input and output are of the same dimension, these layers are often stacked upon each other before the final scores are given.
  - The intermediate fully connected layers therefore give weights to ‘hidden’ categories.

2x2x3 matrix reduced to 12x1x1 then to a 2x1x1.



[3]

A more overhead view of the reduction in CNNs:
128x4x4 to 2048x1x1 to 10x1x1



[5]

2.8 Score Normalization

- A secondary operation can be performed on the output (score) layer to normalize the values.
- The Softmax function normalizes the scores to add to 1, but in addition also gives the largest value the most weight and reduces the rest of the scores.

    - The remaining scores are considered errors and the goal is to reach 1.0 for the correct result.

	Scoring Function	Unnormalized Probabilities	Normalized Probabilities	Negative Log Loss
Dog	-3.44	0.0321	0.0006	
Cat	1.16	3.1899	0.0596	
Boat	-0.81	0.4449	0.0083	
Airplane	3.91	49.8990	0.9315	**0.0709**

Correct score is at 0.9315 with a 0.0709 error given by the remaining possibilities.

2.9 Network Optimization

- Like its name implies, Convolutional Neural Networks also learn by **backpropagation**.
    - By now, backpropagation has been mentioned multiple times. If you are still unfamiliar, please read through the gradient descent and neural network notes.
- Connections, or weights, for the layers, except for fully connected layers, are local.
- From the output score, it propagates through the entire fully connected layers and then only to the activated local regions in the previous layers.

3 Placement and Application

3.1 Layer and Feature Placement

- The general pattern for a CNN is a convolutional layer for each feature followed by a ReLU/activation function layer. After two or three of these, a pooling layer is applied reducing the input space. At the very end, fully connected layers are stacked to produce the ultimate class scores.
- Because of reduction deeper into the network, more complicated features should be placed at the beginning of the network e.g. patterns and shapes and lower layers should have simpler features such as brightness and edges.

3.2 Applications

- A common application of CNNs is image classification because of the easy transformation of an *n x n* image into a 3 *x n x n* matrix.

- This can be extended into anything that can be represented as an image.
  - For example sound waves or even key combinations in a Mario game.

6. Reference & Further Reading

1. http://cs231n.github.io/understanding-cnn/
2. http://colah.github.io/posts/2014-07-Understanding-Convolutions/
3. http://brohrer.github.io/how_convolutional_neural_networks_work.html
4. https://github.com/vdumoulin/conv_arithmetic
5. https://www.packtpub.com/books/content/cnn-architecture
6. Dumoulin, Vincent, and Francesco Visin. “A Guide to Convolution Arithmetic for Deep Learning.” Arxiv, vol. 1603, 23 Mar. 2016, https://arxiv.org/pdf/1603.07285v1.pdf.
7. Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” Neural Information Processing Systems, papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.