Convolutional Neural Networks: in pictures

It may be easier to grasp the workings of a CNN in pictures.

We start with the simplest case of input $\mathbf{x}$ and pattern $\mathbf{k}$

- one non-feature dimension: a 1D vector
- one input feature
- one output feature

We work our way up to a more complicated case

- two non-feature dimensions: 2D matrix
- a number of input features
- a number of output features (possibly different from the input)

The notebook (CNN_pictorial.ipynb) illustrates the various possibilities.

In the remainder of this notebook: we explain the pictures.

Preliminaries

Behavior of a CNN layer

Layer l in a Sequential NN transforms transforms input $\mathbf{y}_{(l-1)}$ to output $y_{(l)}$

- $\mathbf{y}_{(l)}$ is called a *feature map*, for all layers l
  - for each location in $\mathbf{y}_{(l-1)}$
  - it measures the intensity of the pattern match when the pattern is centered at that location

So we write the input as $\mathbf{y}_{(l-1)}$ rather than the $\mathbf{x}$ we had used previously.

The size of all quantities in the convolution can vary by layer

- so we add a parenthesized subscript to indicate the layer

We write

- the kernel size as $f_{(l)}$ (can vary by layer) rather than the f used previously
- the collection of kernels for layer l as $\mathbf{W}_{(l)}$

In general a layer l output $\mathbf{y}_{(l)}$ will have

- $N_{(l)} > 0$ non-feature dimensions
  - non-feature dimension i has length (number of indices) $d_{(l),i}$ indices
    - for dimensions $0 \leq i < N_{(l)}$
  - the set of indexes in dimension i is written as D_i
    - usually equal to $0, \ldots, d_{(l),i}$
- one feature dimension

A CNN Layer l

- preserves the non-feature dimensions (when padding is used)

$$
\begin{aligned}
N_{(l-1)} &= N_{(l)} \\
d_{(l-1),i} &= d_{(l),i} \quad 0 \le i < N_{(l-1)}
\end{aligned}
$$

- changes the length of the feature dimension
    - from $n_{(l-1)}$ to $n_{(l)}$

Thus the shape of the input $\mathbf{y}_{(l-1)}$ and $\mathbf{y}_{(l)}$ may only differ in the length of the feature dimension

- provided padding is used
    - in the absence of padding: $\lfloor \frac{f_{(l)}}{2} \rfloor$ locations are lost at each boundary

Thus the CNN layer l

$$
\begin{aligned}
||\mathbf{y}_{(l-1)}|| &= (d_{(l-1),0} \times d_{(l-1),1} \times \ldots d_{(l-1),N_{(l-1)}}, \quad \mathbf{n_{(l-1)}}) \\
||\mathbf{y}_{(l)}|| &= (d_{(l-1),0} \times d_{(l-1),1} \times \ldots d_{(l-1),N_{(l-1)}}, \quad \mathbf{n_{(l)}}) \\
& because
\end{aligned}
$$

We write

$$\mathbf{y}_{(l),\mathbf{i},j}$$

to denote feature j of layer l at non-feature dimension location $\mathbf{i}$

Channel Last/First

We have adopted the convention of using the final dimension as the feature dimension.

- This is called *channel last* notation.

Alternatively: one could adopt a convention of the first channel being the feature dimension.

- This is called *channel first* notation.

When using a programming API: make sure you know which notation is the default

- Channel last is the default for TensorFlow, but other toolkits may use channel first.

Kernel, Filter

There is one pattern per output feature.

A pattern is also called a *kernel*.

The kernels of layer l are just the weights of the layer.

The vector $\mathbf{W}_{(l),1}$ above

So kernel j ($\mathbf{k}_j$)is just an element $\mathbf{W}_{(l),j}$ of the weights of layer l. entered at $\mathbf{y}_{(l-1),j,1}$

There is one kernel per output feature, so $n_{(l)}$ kernels

- $\mathbf{k}_{(l),1}, \ldots, \mathbf{k}_{(l),n_{(l)}}$

The length of the feature dimension of a kernel matches it's input, i.e., $n_{(l-1)}$

The weight vector $\mathbf{W}_{(l)}$ therefore has multiple dimensions. Our convention for each dimension is

- $\mathbf{W}_{(l),j',\ldots,j}$
    - layer l
    - output feature j
    - spatial location: $\ldots \in \{1, 2, 3$
      $\}$
    - input feature j'

Padding

Convolution centers the pattern at each location of the non-feature dimensions of the input.

But what happens when we try to center a patter over the first/last location ?

- the pattern may extend beyond the boundaries of the input

In such a case, we can choose to *pad* the input

- create a special padding input at the locations of the input beyond the original boundary

We will see this in pictures below.

Activation of a CNN layer

Just like the Fully Connected layer, a CNN layer is usually paired with an activation.

The default activation $a_{(l)}$ in Keras is "linear"

- That is: it returns the dot product input unchanged
- Always know what is the default activation for a layer; better yet: always specify !

Conv 1D: single feature to single feature

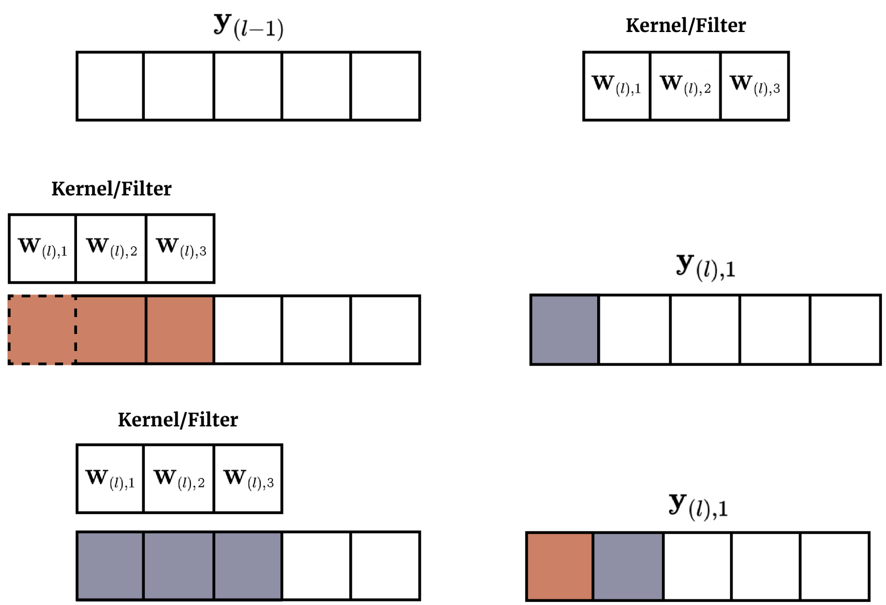
Convolutions pictured: sliding a pattern over the input

A *Convolution* is often depicted as

- A filter/kernel
- That is slid over each location in the non-feature dimensions of the input
- Producing a corresponding output for that location

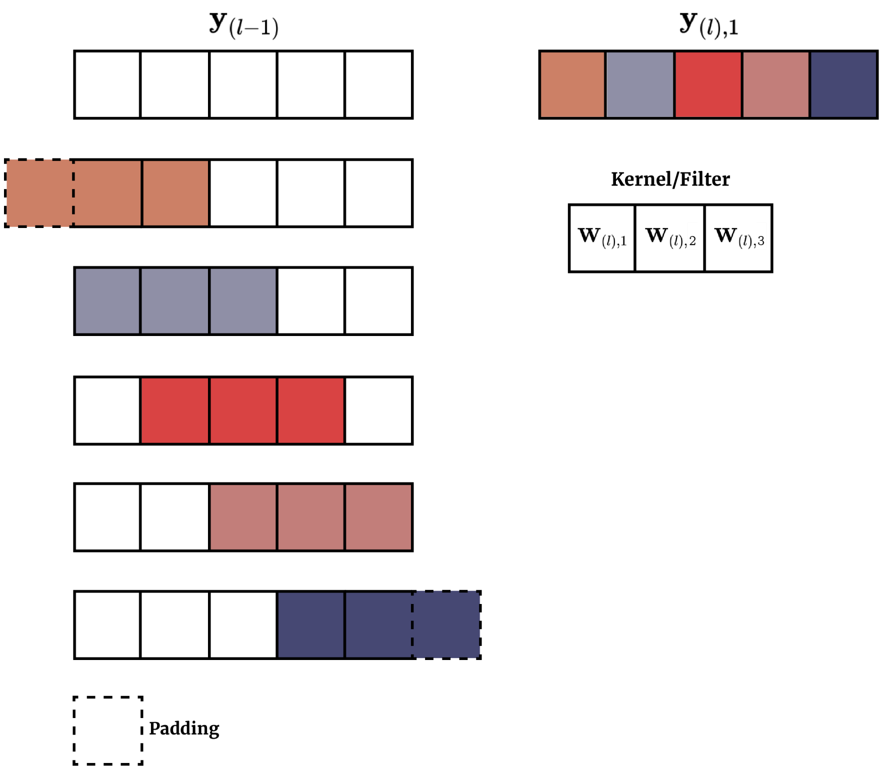
Here's a picture with a kernel of size $f_{(l)} = 3$

Conv 1D, single feature: sliding the filter



After sliding the Kernel over the whole $\mathbf{y}_{(l-1)}$ we get the output feature map $\mathbf{y}_{(l),1}$ for the first (and only) feature:

Conv 1D, single feature: output feature map



Element j of output $\mathbf{y}_{(l),\ldots,1}$ (i.e., $\mathbf{y}_{(l),j,1}$)

- Is colored (e.g., $j = 1$ is colored Red)
- Is computed by applying the *same* $\mathbf{W}_{(l),1}$ to
    - The $f_{(l)}$ elements of $\mathbf{y}_{(l-1),1}$, centered at $\mathbf{y}_{(l-1),j,1}$
    - Which have the same color as the output

Note however that, at the "ends" of $\mathbf{y}_{(l-1)}$ the kernel may extend beyond the input vector.

In that case $\mathbf{y}_{(l-1)}$ may be extended with *padding* (elements with 0 value typically)

- illustrated with the boxes with broken-line edges

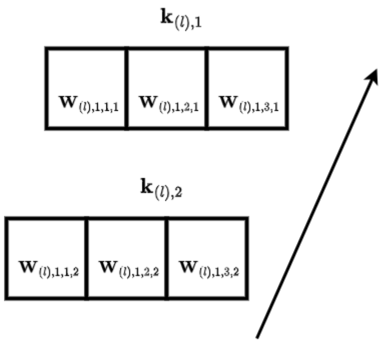
Conv1d transforming single feature to multiple features

When there are multiple output features ($n_{(l)} > 1$) there is *one kernel per output feature*

- $\mathbf{k}_{(l),1}, \ldots, \mathbf{k}_{(l),n_{(l)}}$

Here are the 2 kernels for two output features, assuming $n_{(l-1)} = 1$

Conv 1D: 1 input feature, 2 output features



- $\mathbf{W}_{(l),j',\ldots,j}$
  - layer l
  - output feature j
  - spatial location: $\ldots \in \{1, 2, 3\}$
  - input feature j'

Here is a [picture (CNN_pictorial.ipynb#Conv-1D:-single-feature-to-multiple-features)](CNN_pictorial.ipynb#Conv-1D:-single-feature-to-multiple-features) of a Convolutional layer l transforming

- a 1-dimensional input layer $(l-1)$ consisting of a single feature
    - $N_{(l-1)} = 1, n_{(l-1)} = 1$
- into a 1-dimensional output layer l consisting of a *multiple* features
    - $N_{(l)} = 1, n_{(l)} > 1$

Conv1d transforming multiple features to multiple features

What happens when the input layer has multiple features ?

- e.g., applying Convolutional layer $(l+1)$ to the $n_{(l)}$ features created by Convolutional layer l

The answer is

- The kernels of layer l also have a *feature* dimension
    - Kernel dimensions are $(f_{(l)} \times f_{(l)} \times n_{(l-1)})$
- This kernel is applied
    - at each spatial location
    - to *all features* of layer $(l-1)$
    - Computing a generalized "dot product": sum of element-wise products

When the input $\mathbf{y}_{(l-1)}$ has more than one feature ($n_{(l-1)} > 1$)

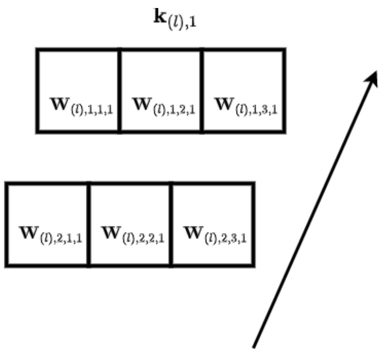
- the kernel for each output feature must have feature dimension of length $n_{(l-1)}$

Here is the kernel for the first output feature, assuming $n_{(l-1)} = 2$

- it's feature dimension is length 2.

There would be a similar kernel for each of the output features.

Conv 1D: 2 input features: kernel 1



- $\mathbf{W}_{(l),j',\ldots,j}$
    - layer l
    - output feature j
    - spatial location: $\ldots \in \{1, 2, 3\}$
    - input feature j'

Notice that (apart from combining spatial locations)

- multiple feature maps from layer $(l-1)$ are combined into one feature map at layer l.
- This is how the "left" half-smile and "right" half-smile features combine into the single "smile" feature

Here is a [picture (CNN_pictorial.ipynb#Conv-1D:-Multiple-features-to-multiple-features)](CNN_pictorial.ipynb#Conv-1D:-Multiple-features-to-multiple-features) of a Convolutional layer l transforming

- a 1-dimensional input layer $(l-1)$ consisting of a 2 features
    - $N_{(l-1)} = 1, n_{(l-1)} = 2$
- into a 1-dimensional output layer l consisting of a *multiple* features
    - $N_{(l)} = 1, n_{(l)} = 3$

With an input layer having N spatial dimensions, a Convolutional Layer l producing $n_{(l)}$ features

- Preserves the "spatial" dimensions of the input
- Replaces the channel/feature dimensions

That is

$$
\begin{array}{lll}
||\mathbf{y}_{(l-1)}|| & = & (n_{(l-1),1} \times n_{(l-1),2} \times \ldots n_{(l-1),N}, \quad \mathbf{n_{(l-1)}}) \\
||\mathbf{y}_{(l)}|| & = & (n_{(l-1),1} \times n_{(l-1),2} \times \ldots n_{(l-1),N}, \quad \mathbf{n_{(l)}})
\end{array}
$$

Conv2d: Two dimensional convolution ($\boldsymbol{N = 2}$)

Thus far, the spatial dimension has been of length $N = 1$.

Generalizing to $N = 2$ is straightforward.

- The number of spatial dimensions (elements denoted by $\ldots$) expands from 1 to 2

When $N = 1$ and $d_1 = 1$

- we have our case of $n_{(l)}$ features at a single location

We have shown that permuting the order of features has no effect on a Dense layer

- There is no ordering relationship among features

But when $d_1 > 1$, there is a *spatial ordering*. For example

- a 2D image
- time ordered data

We need some terminology to distinguish the final dimension from the non-final dimensions

Suppose $\mathbf{y}_{(l)}$ is $(N_{(l)} + 1)$ dimensional of shape

$$||\mathbf{y}_{(l)}|| = (d_{(l),1} \times d_{(l),2} \times \ldots d_{(l),N_{(l)}} \times n_{(l)})$$

(Thus far: $N_{(l)} = 1$ and $n_{(l)} = 1$ but that will soon change)

The first $N_{(l)}$ dimensions $(d_{(l),1} \times d_{(l),2} \times \ldots d_{(l),N})$

- Are called the *spatial* dimensions of layer l

The last dimension (of size $n_{(l)}$)

- Indexes the features i.e., varies over the number of features
- Called the *feature* or *channel* dimension

Notation

- $N_{(l)}$ denotes the *number* of spatial dimensions of layer l
- $n_{(l)}$ denotes the *number of features* in layer l
- We elide the spatial dimensions as necessary, writing
$$\mathbf{y}_{(l),\ldots,j}$$
to denote *feature map* j of layer l
    - where the dots $(\ldots)$ indicate the $N_{(l)}$ spatial dimensions
    - e.g., the feature map detecting a "smile" in the image of a face

For example

- A grey-scale image
    - $N = 2, n_{(l)} = 1$
    - Each pixel in the image has one feature
        - the grey-scale intensity
    - There is an ordering relationship between 2 pixels
        - "left/right", "above/below"
- A color image
    - $N = 2, n_{(l)} = 3$
    - Each pixel in the image has 3 features/attributes
        - the intensity of each of the colors

One can imagine even higher dimensional data ($N > 2$)

- Equity data with "spatial location" identified by (Month, Day, Time)
    - With attributes: $\{$ Open, High, Low, Close $\}$
    - Month/Day/Time are ordered

Note the distinction between the cases

- When layer l has dimension $(d_{(l)} \times 1)$
  - a single feature
  - at $d_{(l)} = d_{(l-1)}$ *spatial* locations
- When layer l has dimension $(1 \times d_{(l)})$
  - (which is how we have implicitly been considering vectors when discussing the Dense layer type)
  - $d_{(l)} = d_{(l-1)}$ features
  - at a single spatial location

$n_{(l)}$ will always refer to the *number of features* of a layer l

Here is a [picture (CNN_pictorial.ipynb#Conv-1D:-single-feature)](CNN_pictorial.ipynb#Conv-1D:-single-feature) of a Convolutional layer l transforming

- a 1-dimensional input layer $(l-1)$ consisting of a single feature
    - $N_{(l-1)} = 1, n_{(l-1)} = 1$
- into a 1-dimensional output layer l consisting of a single feature
    - $N_{(l)} = 1, n_{(l)} = 1$

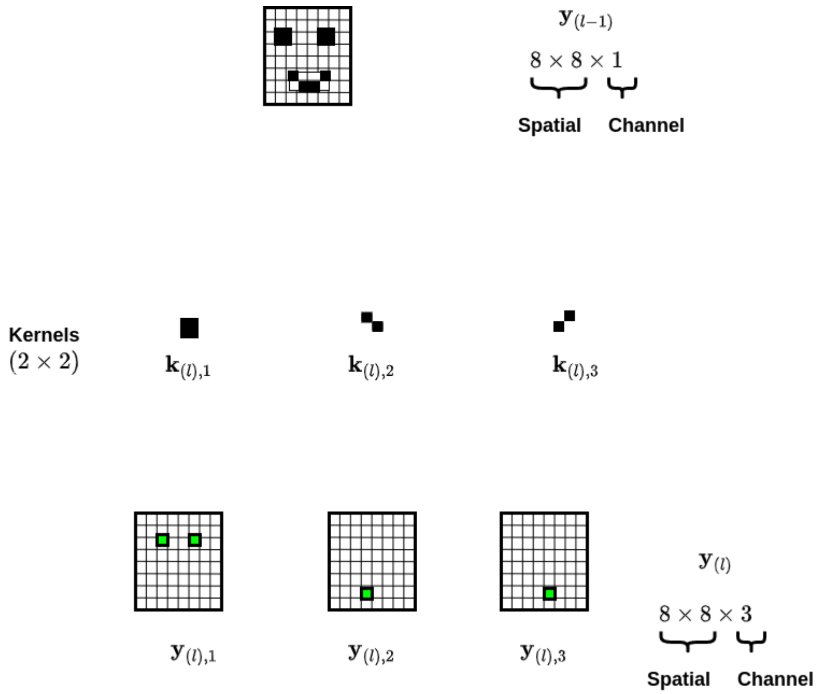
We will generalize Convolution to deal with

- $N_{(l)} > 1$ spatial dimensions
- $n_{(l)} > 1$ features

As a preview of concepts to be introduced, consider

- the input layer $(l-1)$ is a two-dimensional ($N_{(l-1)} = 2$) grid of pixels
- $n_{(l-1)} = 1$
- layer l is a Convolutional Layer identifying $n_{(l)} = 3$ features

Convolution: 1 input feature to 3 output features



Layer $(l-1)$ is three-dimensional tensor: $8 \times 8 \times 1$

- Spatial dimension 8×8
- 1 feature map (channel dimension $= 1$)

- Kernel $k_{(l),j}$ is applied to each spatial location of layer $(l-1)$
- Detecting the presence of the pattern (defined by the kernel) at that location
    - kernel $k_{(l),1}$ detects an eye
- Which results in feature map $\mathbf{y}_{(l)}, \ldots, j$ being created at layer l
    - $\mathbf{y}_{(l), \ldots, 1}$ are indicators of the presence of an "eye" feature

Convolutional Layer description

With this terminology we can say that Convolutional Layer l:

- Transforms the $n_{(l-1)}$ feature maps of layer $(l-1)$
- Into $n_{(l)}$ feature maps of layer l
- Preserving the spatial dimensions: $d_{(l),p} = d_{(l-1),p} \; 1 \leq p \leq N_{(l-1)}$
- Uses a different kernel $\mathbf{k}_{(l),j}$ for each output feature/channel $1 \leq j \leq n_{(l)}$
- Applies this kernel to *each* element in the *spatial* dimensions
- Recognizing a single feature at each location within the spatial dimension

Conv 2D: single input feature: kernel 1

$$\mathbf{k}_{(l),1,1}$$

$$\begin{bmatrix} \mathbf{W}_{(l),1,1,1,1} & \mathbf{W}_{(l),1,1,2,1} & \mathbf{W}_{(l),1,1,3,1} \\ \mathbf{W}_{(l),1,2,1,1} & \mathbf{W}_{(l),1,2,2,1} & \mathbf{W}_{(l),1,2,3,1} \\ \mathbf{W}_{(l),1,3,1,1} & \mathbf{W}_{(l),1,3,2,1} & \mathbf{W}_{(l),1,3,3,1} \end{bmatrix}$$

- $\mathbf{W}_{(l),j',\ldots,j}$
  - layer l
  - output feature j
  - spatial location: $\ldots \in \{(\alpha, \alpha') \in (d_{(l-1),1} \times d_{(l-1),2}\}$
  - input feature j'

Here is a picture (CNN_pictorial.ipynb#Conv-2D:-single-feature-to-single-feature) of a Convolutional layer l transforming

- a 2-dimensional input layer $(l-1)$ consisting of a 1 feature
  - $N_{(l-1)} = 2, n_{(l-1)} = 1$
- into a 2-dimensional output layer l consisting of 1 feature
  - $N_{(l)} = 1, n_{(l)} = 1$

We can further generalize to producing multiple output features

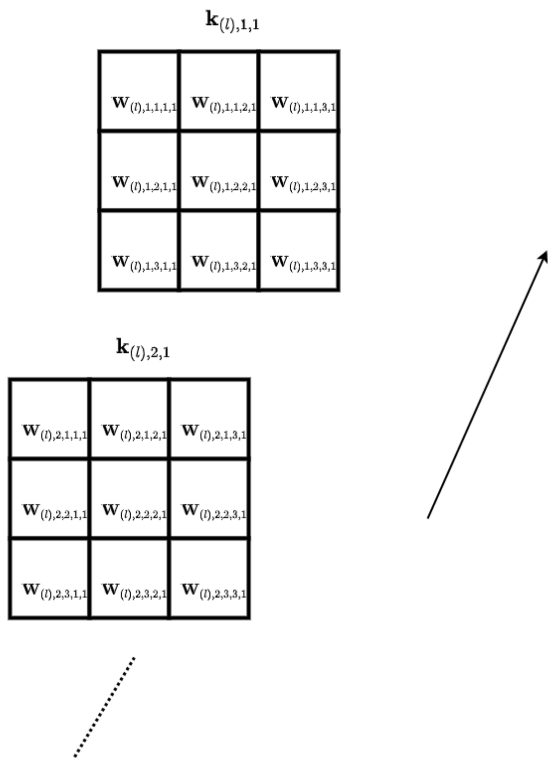
Here is a picture (CNN_pictorial.ipynb#Conv-2D:-single-feature-to-multiple-features) of a Convolutional layer l transforming

- a 2-dimensional input layer $(l-1)$ consisting of a 1 feature
  - $N_{(l-1)} = 2, n_{(l-1)} = 1$
- into a 2-dimensional output layer l consisting of 2 feature
  - $N_{(l)} = 1, n_{(l)} = 2$

Dealing with multiple input features works similarly as for $N = 1$:

- The dot product
- Is over a spatial region that now has a "depth" $n_{(l-1)}$ equal to the number of input features
- Which means the kernel has a depth $n_{(l-1)}$

Conv 2D: multiple input features: kernel 1



Here is a [picture (CNN_pictorial.ipynb#Conv-2D:-multiple-features-to-single-feature)](CNN_pictorial.ipynb#Conv-2D:-multiple-features-to-single-feature) of a Convolutional layer l transforming

- a 2-dimensional input layer $(l-1)$ consisting of multiple features
    - $N_{(l-1)} = 2, n_{(l-1)} = 2$
- into a 2-dimensional output layer l consisting of 1 feature
    - $N_{(l)} = 1, n_{(l)} = 1$

And finally: the general case for a 2 spatial dimensions

Here is a [picture (CNN_pictorial.ipynb#Conv-2D:-multiple-features-to-multiple-features)](CNN_pictorial.ipynb#Conv-2D:-multiple-features-to-multiple-features) of a Convolutional layer l transforming

- a 2-dimensional input layer $(l-1)$ consisting of multiple features
  - $N_{(l-1)} = 2, n_{(l-1)} = 3$
- into a 2-dimensional output layer l consisting of multiple features
  - $N_{(l)} = 1, n_{(l)} = 2$

In [5]:

```
print("Done")
```

```
Done
```