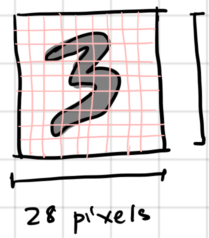
Convolutional Neural Networks

Convolutional neural networks (CNNs) are used when the data $x \in \mathbb{R}^d$ has an underlying Euclidean geometric structure. The most prominent example is when x is an $N \times N$ image so that x can be written as:

$$x(n_1, n_2) \in [0,1], \quad 0 \le n_i < N, \quad i = 1,2 \qquad (*)$$

In this case $x \in [0,1]^d$ where $d = N^2$, but x has additional structure given by (*). An example we have already seen is MNIST:

$x =$ 

$, \quad x \in [0,1]^{28^2} = [0,1]^{784}$

Image processing is quite common in machine learning, e.g. computer vision, and countless other examples exist. Some popular image databases include:

- MNIST : 70,000 ; 28×28 ; grayscale ; handwritten digits
- CIFAR-10 : 60,000 ; 32×32 ; color images with 10 classes
- CIFAR-100 : 60,000 ; 32×32 ; color images with 100 classes
- ImageNet : Over 14 million color images with over 20,000 classes

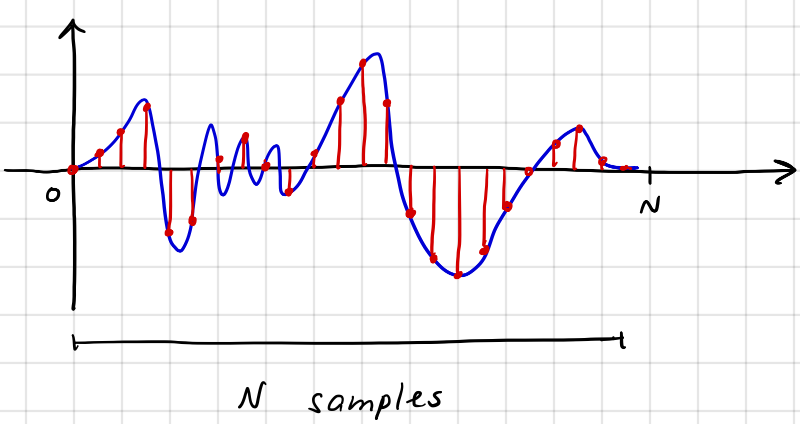
CNNs are also used for data with 1D and 3D geometries. 1D signals consist of, for example:

- audio recordings, as in speech, music, etc.
- medical device recordings, as in EEG, etc.
- more generally time series, although if one wants to make predictions of the future values of a single time series, one should use a recurrent neural network.

In this case $d = N$ and x is of the form:

$$x(n) \in \mathbb{R}, \quad 0 \le n < N$$

which is the same vector structure from before, but now there is the implication that the order $x(0), x(1), x(2), \dots, x(N-1)$ matters.

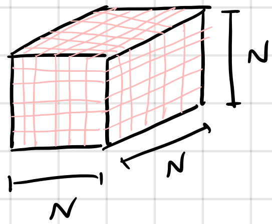
$x =$ 

3D signals may consist of, e.g.,
- volumetric medical data
- volumetric data from physics, e.g., many particle physics and fluid mechanics
- self-driving car data
- LiDAR data

The idea is similar to 1D & 2D, in this case:

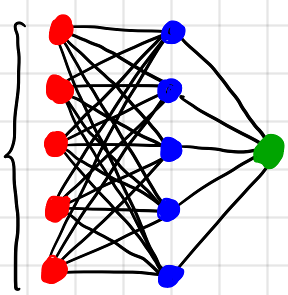
$$x(n_1, n_2, n_3) \in \mathbb{R}, \quad 0 \le n_i < N, \quad i = 1,2,3$$

and so $x \in \mathbb{R}^d$, $d = N^3$

$x =$  $\in \mathbb{R}^{N^3}$

CNNs as special cases of ANNs

CNNs can be viewed as ANNs with sparse, shared weights. Let us first look at a diagram, supposing that $x \in \mathbb{R}^d$, $d = N$, has a 1D geometric structure.

$\mathbb{R}^5 \ni x =$  output layer

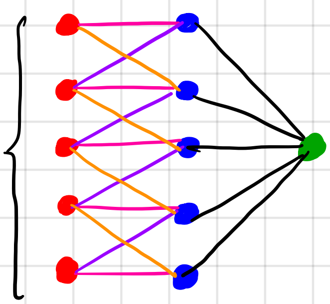
input layer | hidden layer with 5 neurons

All edges indicate different weights

$\underbrace{\qquad\qquad}$

Fully connected ANN w/ one hidden layer.

$$f(x;\theta) = \langle \alpha, \sigma(Wx + \beta) \rangle$$

$\mathbb{R}^5 \ni x =$  output layer

input layer | hidden layer with 5 neurons

All pink/purple/orange edges have the same weight

Black edges may have different weights

$\underbrace{\qquad\qquad}$

CNN = sparsely connected ANN with shared weights.

$$\tilde{f}(x;\theta) = \langle \alpha, \sigma(\tilde{W}x + \beta) \rangle$$

Let us examine the CNN diagram more closely:

We have five neurons:

$$\left.\begin{array}{l} w_0 = (b, c, 0, 0, 0) \\ w_1 = (a, b, c, 0, 0) \\ w_2 = (0, a, b, c, 0) \\ w_3 = (0, 0, a, b, c) \\ w_4 = (0, 0, 0, a, b) \end{array}\right\} \tilde{W} = \begin{pmatrix} b & c & 0 & 0 & 0 \\ a & b & c & 0 & 0 \\ 0 & a & b & c & 0 \\ 0 & 0 & a & b & c \\ 0 & 0 & 0 & a & b \end{pmatrix} = \text{Toeplitz type weight matrix}$$

(let us ignore the biases.)

Notice then that:

$$\langle x, w_0 \rangle = b\,x(0) + c\,x(1)$$
$$\langle x, w_n \rangle = a\,x(n-1) + b\,x(n) + c\,x(n+1), \quad 1 \le n \le 3$$
$$\langle x, w_4 \rangle = a\,x(3) + b\,x(4)$$

Let $x, y \in \mathbb{R}^N$ and define their correlation as:

$$(x \star y)(n) = \sum_{m=0}^{N-1} x(m)\, y(n+m), \qquad y(n) = 0 \ \ \forall\, n \notin [0, N)$$

Note $(x \star y)(n)$ takes, in general, nonzero values for: $-N < n < N$.

Therefore we can think of $x \star y \in \mathbb{R}^{2N-1}$

A subset of these values correspond to $\langle x, w_n \rangle$. In our example above:

Define $w = w_2 = (0, a, b, c, 0) \in \mathbb{R}^5$. Then:

$$\langle x, w_n \rangle = (x \star w)(n-2), \qquad 0 \le n \le 4$$

The vector w is called a filter. Finished class here

The operation of convolution is closely related to correlation. It is defined as:

$$(x * y)(n) = \sum_{m=0}^{N-1} x(m)\, y(n-m)$$

Set $\bar{y}(n) = y(-n)$. Then:

$$(x * \bar{y})(-n) = \sum_{m=0}^{N-1} x(m)\, \bar{y}(-n-m) = \sum_{m=0}^{N-1} x(m)\, y(n+m) = (x \star y)(n) \qquad (*)$$

Notice in the CNN network, we only need to learn 3 parameters, a, b, c, in the hidden layer, versus $5^2 = 25$ parameters in the fully connected network.

In 2D, correlation is defined as:

$$(x \star y)(n_1, n_2) = \sum_{m_1=0}^{N-1} \sum_{m_2=0}^{N-1} x(m_1, m_2)\, y(n_1+m_1, n_2+m_2)$$

In practice, CNNs implement correlation, not convolution. But equation $(*)$ shows they are equivalent, so we will use both definitions and will often use convolution.

In practice these filters are often small, e.g., if $x \in \mathbb{R}^{N^2}$ is an $N \times N$ image, the filters may only be $(2s+1) \times (2s+1)$ in which the filter is indexed as:

$$w(m_1, m_2) \in \mathbb{R}, \qquad m_i \in [-s, s] \cap \mathbb{Z}, \quad i = 1,2$$

We still index x as:

$$x(n_1, n_2) \in \mathbb{R}, \qquad n_i \in [0, N) \cap \mathbb{Z}, \quad i = 1,2$$

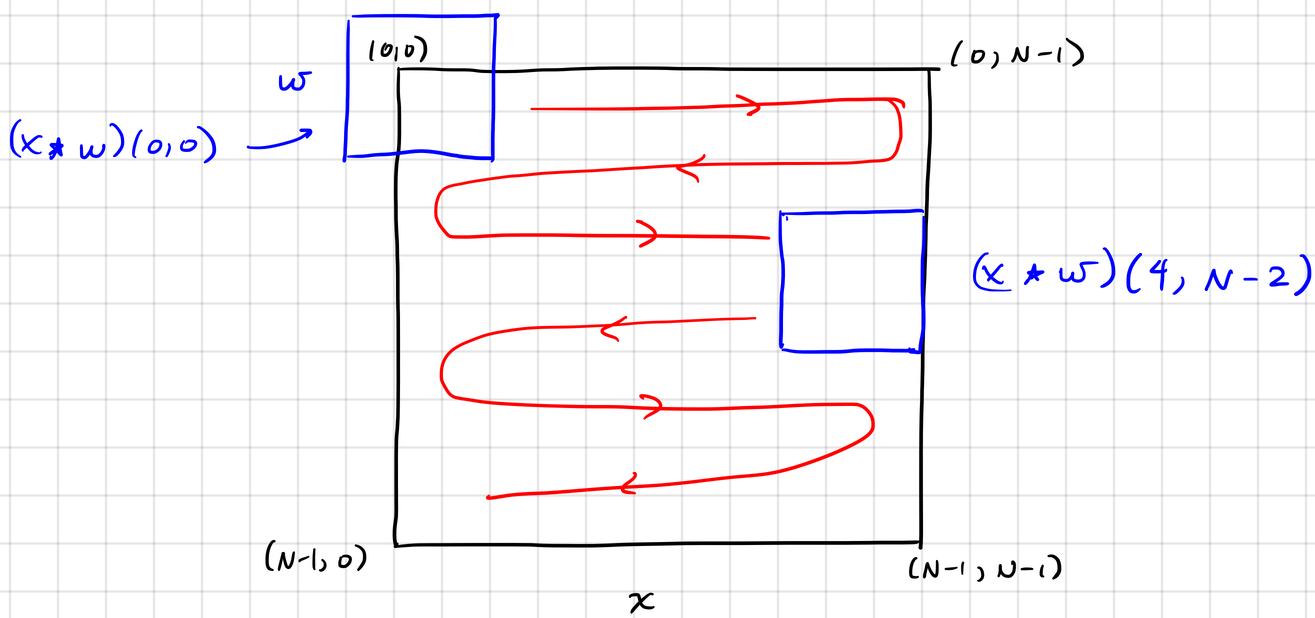
Then:

$$(x * w)(n_1, n_2) = \sum_{m_1=-s}^{s} \sum_{m_2=-s}^{s} w(m_1, m_2)\, x(n_1 + m_1, n_2 + m_2)$$

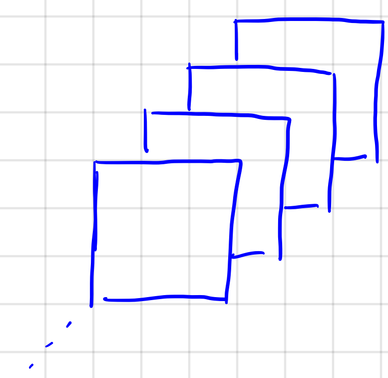
$$n_i \in [0, N) \cap \mathbb{Z}, \quad i = 1, 2$$

Here s may be, e.g., $s = 1, 2$, or 3

Pictorally:



Also in practice there will be more than one filter in each layer. We'll come back to this point later.



A stack of filters for one layer

CNNs work under two related priors, both related to the underlying geometry of the signal:

(i) Neighboring dimensions, e.g. pixels, are related and their relation is relavant to the task. Note this relation may be "long range," that is, it goes over large portions of the image, but it is still governed by the underlying geometry of each data point.

$\Rightarrow$ · weights only between nearby dimensions
· long range relations obtained through depth and new nonlinearities (more on this later)

(ii) Translation equivariance / invariance: (small) translations of the signal, e.g., $x_t(n) = x(n-t)$, do not change the class of x.

$\Rightarrow$ weight sharing and correlation/convolution operators. Indeed,

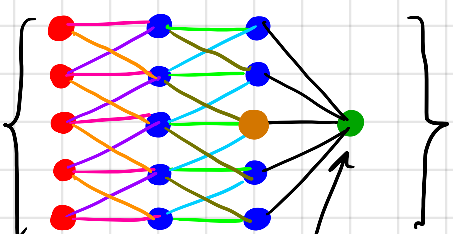
$$(x_t \star y)(n) = (x \star y)(n-t)$$
$$(x_t * y)(n) = (x * y)(n-t)$$

[assuming zero padding]

"Translating the input merely translates the output by the same amount." $\leftarrow$ EQUIVARIANCE

Let us examine (i) and (ii) a bit more closely. In the previous example we used filter with a receptive field (that is support) of 3 dimensions. But what if we believe there are relations between all 5 dimensions? One option is to use a wider filter. Another is to use a deeper network, e.g.

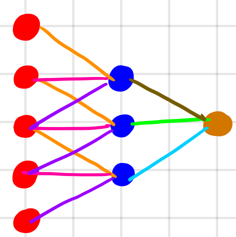
$\mathbb{R}^5 \ni x =$



CNN w/ 2 hidden layers

input layer
first hidden layer
second hidden layer
output layer

$$f(x;\theta) = \sum_n \alpha(n)\, \sigma\big(\sigma(x * w_1) * w_2\big)(n)$$



Closer look at the orange neuron. Notice the orange neuron collects information from all 5 original dims. Thus, even though
$$|\mathrm{supp}(w_1)| = |\mathrm{supp}(w_2)| = 3,$$
the effective support of w_2 relative to the input x, i.e. its receptive field, is 5.

Notice in particular if you have L filters w_ℓ, $1 \le \ell \le L$, each with $|\mathrm{supp}(w_\ell)| = s$

and you compute an L layer network of the form:

$$\sigma(\cdots \sigma(\sigma(x * w_1) * w_2) * \cdots * w_L)$$

then the effective receptive field of w_ℓ relative to x is:

$$\text{effective receptive field of layer } \ell = (s-1)\ell + 1$$

[This is for 1D]

This means the receptive field grows linearly with depth. While this is good, for high dimensional data (e.g., time series with very large numbers of samples such as $N \geq 2^{13}$ or high resolution images) this may not be fast enough. CNNs thus incorporate an additional (nonlinear) pooling operation. This pooling operator works by (nonlinearly) downsampling the output of a given layer. Let $x \in \mathbb{R}^N$, a factor 2 downsampling operator applied to x returns a new vector x_d with half the length:

$$x_d(n) = x(2n), \quad 0 \leq n < N/2$$

This type of downsampling is standard in signal processing. In 2D for signals $x \in \mathbb{R}^{N^2}$, $x_d \in \mathbb{R}^{(\frac{N}{2})^2} = \mathbb{R}^{N^2/4}$ with:

$$x_d(n_1, n_2) = x(2n_1, 2n_2), \quad 0 \leq n_1, n_2 < N/2$$

CNNs incorporate other types. Two common ones are:

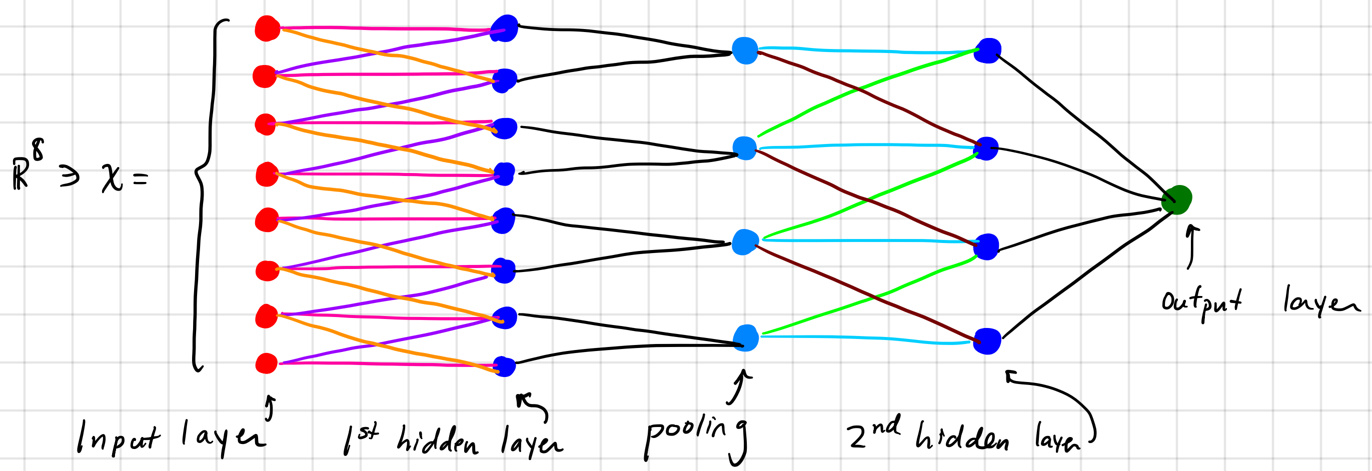
(i) Average pooling:

$$x_d(n) = \frac{1}{2}\left(x(2n) + x(2n+1)\right), \quad 0 \leq n < N/2$$

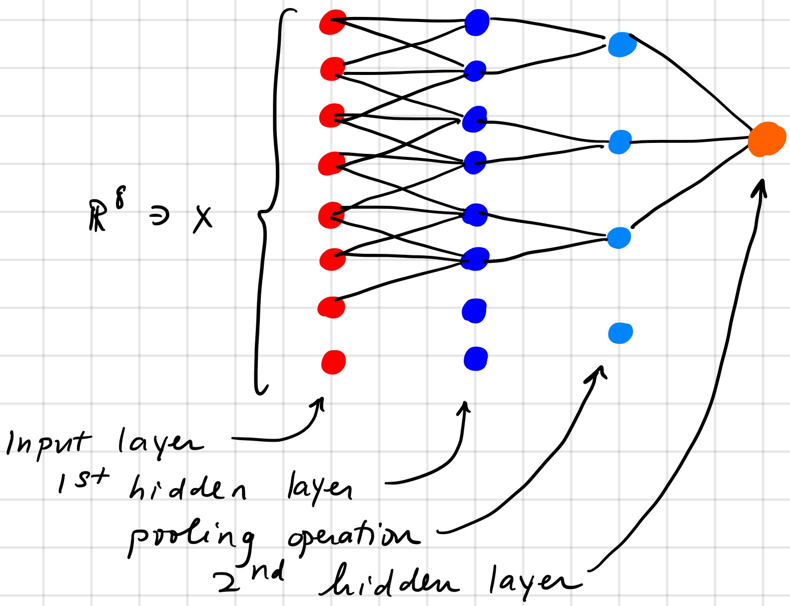
(ii) Max pooling, which is nonlinear:

$$x_d(n) = \max\left(x(2n), x(2n+1)\right)$$

Pooling operations serve multiple roles. One is to expand the receptive field of deeper filters. They also encode invariance into the representation of x. Let us first consider the receptive field:



Notice how the pooling operation expands the receptive field of the 2nd hidden layer. In the example without pooling, the receptive field of the 2nd hidden layer was 5. Now with the pooling operation we have:



Thus the receptive field of the orange neuron in the 2nd hidden layer is 7 with the pooling operation, but the number of learned parameters has not increased. To get the same receptive field without pooling, we would need 3 hidden layers which would increase the number of trained parameters.

Pooling operations in 1D effectively double the support of the neurons coming after them. In 2D a pooling is a function of 4 pixels:

$$x_d(n_1, n_2) = \text{pool}\left[x(2n_1, 2n_2),\ x(2n_1+1, 2n_2),\ x(2n_1, 2n_2+1),\ x(2n_1+1, 2n_2+1)\right]$$

so the support is quadrupled

Equivariance vs. Invariance

The other important role of convolutional operators and pooling operators is their role in developing equivariant and invariant representations.

Let $x_t(n) = x(n-t)$ be the translation of x by t.
Let $\Phi(x) \in \mathbb{R}^D$ be a representation of x, e.g., $\Phi(x)$ could be the last hidden layer of a neural network.

We say $\Phi(x)$ is translation equivariant if

$$\Phi(x_t)(n) = \Phi(x)(n-t)$$

The representation $\Phi(x)$ is translation invariant if

$$\Phi(x_t) = \Phi(x)$$

The representation $\Phi(x)$ is translation invarient up to scale 2^J if:

$$\|\Phi(x) - \Phi(x_t)\|_2 \leq C\,|t|\,2^{-J}\,\|x\|_2$$

The local translation invariance property implies that if the translation t is small relative to the scale 2^J, that is:

$$|t| << 2^J \Rightarrow |t|/2^J << 1$$

then $\Phi(x)$ and $\Phi(x_t)$ are nearly identical since:

$$\|\Phi(x) - \Phi(x_t)\|_2 \leq C \cdot \underbrace{\frac{|t|}{2^J}}_{<<1} \cdot \|x\|_2 << 1$$

An equivariant representations may be useful in its own right. For example, if you are considering the force acting on a body, the force is equivariant with respect to translations and rotations of the body. Equivariant representations are also important for extracting invariant representations. Indeed, suppose $\tilde{\Phi}(x)$ is a translation equivariant representation. Then:

$$\phi(x) = \langle \alpha, \sigma(\tilde{\Phi}(x)) \rangle = \sum_n \alpha(n) \sigma(\tilde{\Phi}(x)(n)) \in \mathbb{R} \qquad (*)$$

is invariant to translations of x, that is:

$$\phi(x_t) = \phi(x)$$

Notice that correlation/convolution yield translation equivariant representations since

$$(x_t * w)(n) = (x * w)(n-t)$$

Fully invariant representations $\Phi(x) = (\phi_\lambda(x))_{\lambda \in \Lambda}$ consisting of components $\phi_\lambda(x)$ defined through $(*)$ are useful when one has a known global invariance prior. This may be the case in data driven problems coming from chemistry, physics, biology, and statistical modeling of time series, among other contexts. For example, the potential energy of a many body system is invariant to global translations and rotations of the system, and the statistics of a stationary stochastic process are invariant to translations. These types of globally invariant representations are also useful for processing data that can be modelled as an abstract graph $G = (V, E)$, consisting of vertices V connected by edges E. In order to compare two graphs G_1 and G_2 we need a representation $\Phi(G)$ that is invariant to the order in which the vertices are enumerated.

On the other hand, in image processing tasks in computer vision, global translation invariance is often too inflexible. Rarely does one encounter large, global translations (or rotations) of images, but smaller translations and rotations are more common.

A pooling operation increases local translation invariance by a factor of 2. Therefore if we incorporate J pooling operations in our neural network, the local translation invariance of the network will be up to scale 2^J. Note this only works because the linear operations are translation equivariant convolution operators.

Notice that the translation invariance properties, even the one with scale 2^J, still refer to translations that act on the whole signal. Many signal deformations of interest act locally on the signal. We can define these mathematically as diffeomorphisms, and think of them as generalized translations. In order to develop this framework, it is useful to model the data point x as a function. We have:

- 1D : $x : \mathbb{R} \to \mathbb{R}$
- 2D : $x : \mathbb{R}^2 \to \mathbb{R}$
- 3D : $x : \mathbb{R}^3 \to \mathbb{R}$

The discrete data can be considered a sampling of these functions, that is $x(n)$ is the evaluation of x at $n \in \mathbb{Z}$ and on the computer we store $(x(n))_{0 \leq n < N}$.

Let us consider $x : \mathbb{R} \to \mathbb{R}$, that is 1D signals, keeping in mind that everything can be generalized to 2D and 3D signals.

Let $\tau : \mathbb{R} \to \mathbb{R}$ with $\tau \in C^2(\mathbb{R})$ and

$$\|\tau'\|_\infty = \sup_{u \in \mathbb{R}} |\tau'(u)| \leq \frac{1}{2}$$

Then the mapping $u \mapsto u - \tau(u)$ is a diffeomorphism with displacement field τ, that is, $u \in \mathbb{R}$ gets moved to $u - \tau(u)$, which displaces u by $\tau(u)$. We can model deformations of our data through such diffeomorphisms as:

$$x_\tau(u) = x(u - \tau(u))$$

Notice that if $\tau(u) = t$, then this operation is a translation. But this model allows us to study "local" translations and other operations that deform the data locally.

Unlike translations in which we may wish for a translation invariant representation, i.e. $\Phi(x_t) = \Phi(x)$ where $x_t(u) = x(u-t)$, encoding invariance over diffeomorphisms is too strong. Indeed the diffeomorphism group is infinite-dimensional (for data $x : \mathbb{R}^p \to \mathbb{R}$ the translation group is p-dimensional) and one can string together small diffeomorphisms to go between vastly different data points, e.g., in MNIST:

$$1 \xrightarrow{\tau_1} 7 \xrightarrow{\tau_2} 2 \xrightarrow{\tau_3} 3$$

Therefore diffeomorphism invariance is far too strong since we would classify too many things as the same. Rather we seek a representation that is stable to diffeomorphisms, meaning:

$$\| \Phi(x) - \Phi(x_\tau) \|_2 \leq C \cdot \text{size}(\tau) \cdot \|x\|_2 \qquad (*)$$

Later we will discuss CNNs in which size(τ) depends on $\|\tau'\|_\infty$ and $\|\tau''\|_\infty$. In particular if τ is a translation then size$(\tau) = 0$, so $(*)$ will imply global translation invariance. If we want only translation invariance up to the scale 2^J we can amend $(*)$ as:

$$\| \Phi(x) - \Phi(x_\tau) \|_2 \leq C \cdot \Big[2^{-J} \|\tau\|_\infty + \underbrace{\text{size}(\tau)}_{F(\|\tau'\|_\infty, \|\tau''\|_\infty)} \Big] \|x\|_2$$

measures the translation part of τ.

Multiple channels

Since a Toeplitz weight matrix only implements one convolutional filter, the expressiveness of the network will be pretty limited. CNNs rectify this by using many filters in each layer. This also allows CNNs to encode additional invariants on top of translation invariance. We will explain how stacking multiple filters works using the VGG network as a model. This will also explain how color images are processed.

A color image x can be modeled as $x : \mathbb{R}^2 \to \mathbb{R}^3$, in which:

$$x(u) = (x_r(u), x_g(u), x_b(u)), \quad u = (u_1, u_2) \in \mathbb{R}^2$$

and where x_r is the red channel, x_g is the green channel, and x_b is the blue channel. The first hidden layer of the VGG network processes x using a bank of $3M_1$ filters, M_1 filters for each channel:

$$x \mapsto \left((x_r * w^{(1)}_{r,j})_{j=1}^{M_1}, \ (x_g * w^{(1)}_{g,j})_{j=1}^{M_1} \ (x_b * w^{(1)}_{b,j})_{j=1}^{M_1} \right)$$

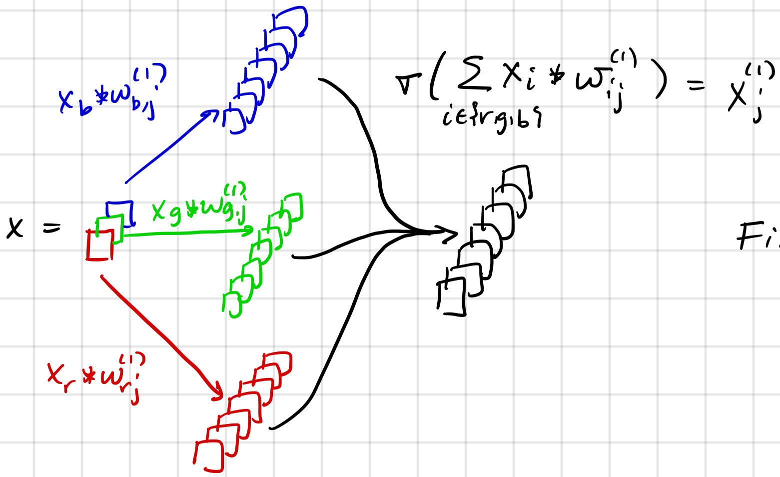
Note this gives us $3M_1$ "images."
These responses are combined across the channels:

$$x \mapsto \sum_{i \in \{r,g,b\}} x_i * w^{(1)}_{ij}$$

and a nonlinearity is applied: $\quad x \mapsto x^{(1)}_j = \sigma\left(\sum_{i \in \{r,g,b\}} x_i * w^{(1)}_{ij} \right), \quad 1 \leq j \leq M_1$
(e.g. ReLU)

Thus we have taken the original image $x: \mathbb{R}^2 \to \mathbb{R}^3$ and mapped it into a new M_1 channel image $x^{(1)}: \mathbb{R}^2 \to \mathbb{R}^{M_1}$ with:

$$x^{(1)}(u) = \left(x_j^{(1)}(u) \right)_{j=1}^{M_1}, \quad u \in \mathbb{R}^2$$



First hidden layer of the VGG network

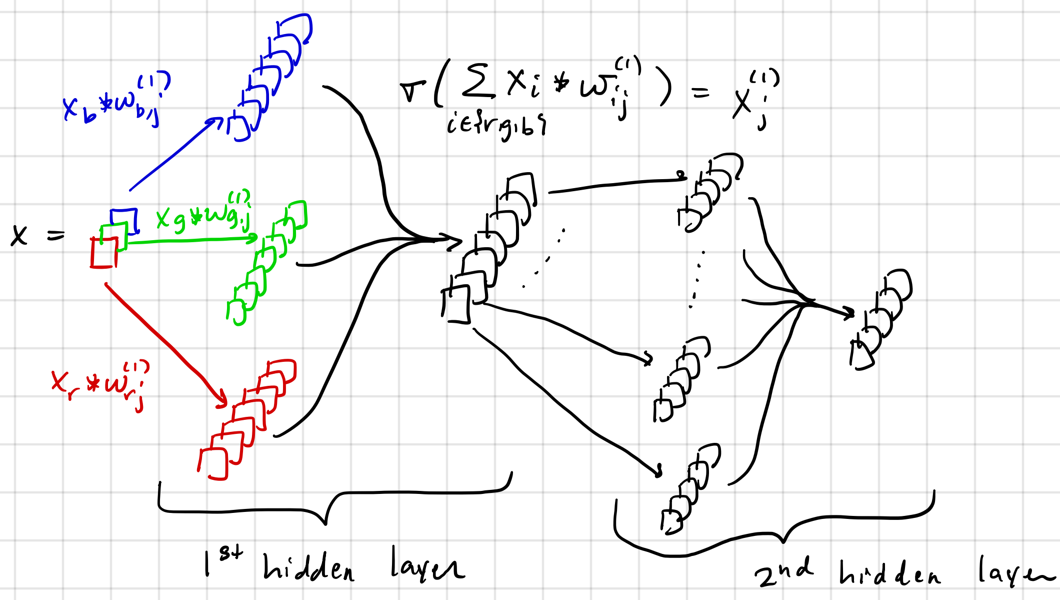
The second hidden layer works similarly, except that instead of having the three RGB channels as input it has the M_1 channels from the first hidden layer as input:

$$x^{(1)} = \left(x_j^{(1)} \right)_{j=1}^{M_1} \longmapsto x^{(2)} = \left(x_k^{(2)} \right)_{k=1}^{M_2}$$

$$x_k^{(2)} = \sigma\left(\sum_{j=1}^{M_1} x_j^{(1)} * w_{j,k}^{(2)} \right), \quad 1 \leq k \leq M_2$$

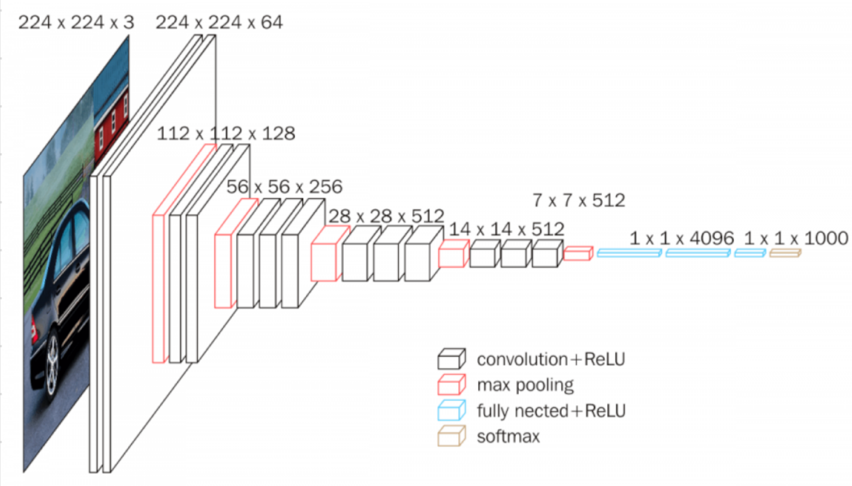
$$= \sigma\left(\sum_{j=1}^{M_1} \sigma\left(\sum_{i \in \{r,g,b\}} x_i * w_{i,j}^{(1)} \right) * w_{j,k}^{(2)} \right)$$

Thus the output of the 2^{nd} hidden layer has M_2 channels: $x^{(2)}: \mathbb{R}^2 \to \mathbb{R}^{M_2}$



1^{st} and 2^{nd} hidden layers of the VGG network

Subsequent layers work similarly. In some of the hidden layers a max pooling operation is also applied. Here is a diagram of the VGG network:



These stacks of filters within each layer give the CNN increased capacity for distinguishing between multiple types of signals. They also allow the CNN to encode invariants over groups other than the translation group. Convolution is equivariant with respect to the translation group, but not other groups such as the rotation group. However, the stack of filters can be learned to be equivariant with respect to other group actions. We will show later how this works for rotations.

CNNs from the perspective of approximation theory

The research on approximation theory for CNNs is limited and we will not spend much additional time on it. We mention three results:

1.) Poggio, et al (2017) - We already discussed this result at length. CNNs are a special case of compositional networks in which the weights are shared and the composed dimensions are organized geometrically.

2.) Zhou - "Universality of deep CNNs" (2018) : Universal approximation by CNNs as the # of layers $L \to \infty$.

3.) Petersen & Voigtlaender - "Equivalence of approximation by CNNs and fully connected networks" (2018) : Rates of approximation by CNNs and ANNs are the same.

CNNs from the perspective of signal processing

Let us now see how CNNs arise naturally as a powerful way of representing signals. A lot of the mathematical ideas for this section come from:

(i) Mallat - "Group Invariant Scattering" (2012)

(ii) Bruna & Mallat - "Invariant Scattering Convolution Network" (2013)

(iii) ... and several subsequent papers

Per our previous discussions, suppose we are looking for a representation $\Phi_J(x)$ of signal type data, which we model as $x : \mathbb{R} \to \mathbb{R}$. Define

$$\|x\|_2 = \int_{\mathbb{R}} |x(u)|^2 \, du < +\infty$$

We want $\Phi_J(x)$ to have the following properties:

(a) Translation invariance up to the scale 2^J

(b) Stability to diffeomorphisms

Combining (a) and (b) and recalling that for $\tau \in C^2(\mathbb{R})$ w/ $\|\tau'\|_\infty \leq \frac{1}{2}$ we defined

$$x_\tau(u) = x(u - \tau(u))$$

we want:

$$\|\Phi(x) - \Phi(x_\tau)\|_2 \leq C \cdot \left[2^{-J} \|\tau\|_\infty + \|\tau'\|_\infty + \|\tau''\|_\infty \right] \|x\|_2$$

But is this enough? Consider the representation:

$$\Phi(x) = \int_{\mathbb{R}} x(u) \, du \qquad \text{CoV} : v = u - t$$

We have:

$$\Phi(x_t) = \int_{\mathbb{R}} x_t(u) \, du = \int_{\mathbb{R}} x(u-t) \, du = \int_{\mathbb{R}} x(v) \, dv$$

$\Rightarrow \Phi(x_t) = \Phi(x)$ and so $\Phi(x)$ is translation invariant

We also have:

$$\Phi(x_\tau) = \int_{\mathbb{R}} x_\tau(u) \, du = \int_{\mathbb{R}} x(u - \tau(u)) \, du \qquad \begin{aligned} v &= u - \tau(u) \\ dv &= [1 - \tau'(u)] \, du \end{aligned}$$

$$= \int_{\mathbb{R}} \frac{x(v)}{1 - \tau'(u)} \, dv \qquad (u \text{ depends on } v)$$

Therefore:

$$\Phi(x) - \Phi(x_\tau) = \int_{\mathbb{R}} x(v) \, dv - \int_{\mathbb{R}} \frac{x(v)}{1 - \tau'(u)} \, dv$$

$$= \int_{\mathbb{R}} \left[1 - \frac{1}{1 - \tau'(u)} \right] x(v) \, dv = \int_{\mathbb{R}} \frac{-\tau'(u)}{1 - \tau'(u)} \cdot x(v) \, dv$$

$$\Rightarrow |\Phi(x) - \Phi(x_\tau)| = \left| \int_{\mathbb{R}} -\frac{\tau'(u)}{1-\tau'(u)} x(v)\, dv \right| \le \int_{\mathbb{R}} \left| \frac{\tau'(u)}{1-\tau'(u)} \right| |x(v)|\, dv$$

$$\le 2\|\tau'\|_\infty \int_{\mathbb{R}} |x(v)|\, dv = 2\|\tau'\|_\infty \|x\|_1 \qquad (*)$$

Therefor $\Phi(x)$ is translation invariant and stable to diffeomorphisms as encoded by $(*)$. But $\Phi(x)$ is not a very good represention because it is just the integral of x. Many different signals have the same integral. Therefore to (a) and (b) we must add another condition:

(c) The representation retains enough information in x to perform the task.

Condition (c) is not as precise as (a) and (b). A precise and very strong version of (c) is:

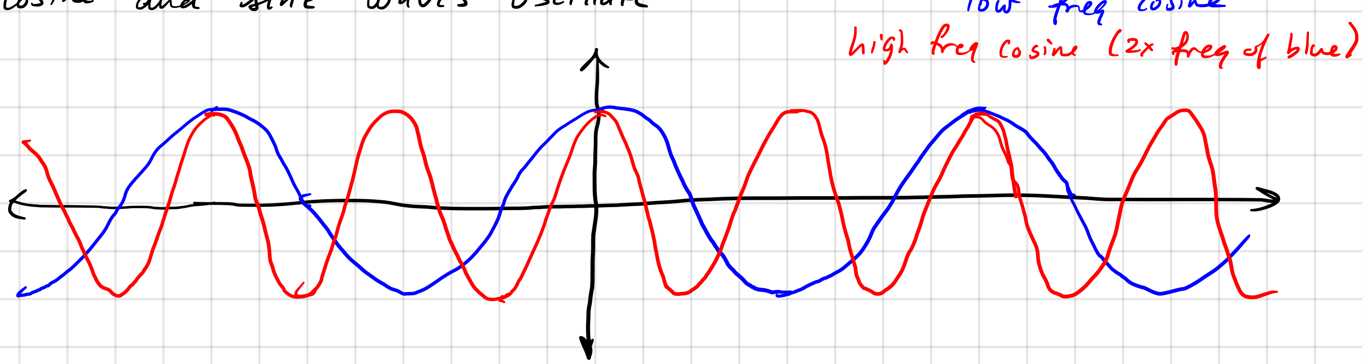
$$\Phi(x) = \Phi(y) \iff y = x_t \text{ for some } t \qquad (**)$$

Equation $(**)$ says $\Phi(x)$ is invertible up to translations. While this is mathematically precise, it may also take things too far. Indeed, in many classification tasks, $\Phi(x)$ being invertible is not a requirement for good classification results. We will instead be content to develop a systematic way of adding new information into $\Phi(x)$ while maintaining properties (a) (translation invariance) and (b) (stability to diffeomorphisms).

A key to understanding local translation invariance and diffeomorphism stability is through frequency representations of signals $x: \mathbb{R} \to \mathbb{R}$. For example, in a piece of music, we listen to the piece in time, but another way of representing the piece is through the notes, or frequencies, contained in it. The Fourier transform is the mathematically precise way to do this. Define a complex valued sinusoid at the frequency ω as:

$$e_\omega(u) = e^{i\omega u} = \cos(\omega u) + i\sin(\omega u), \quad i = \sqrt{-1}$$

The frequency is ω because the cosine and sine functions are periodic with period $2\pi/\omega$. Thus the higher ω, the faster the cosine and sine waves oscillate



The Fourier transform of $x: \mathbb{R} \to \mathbb{R}$ w/ $\int_{\mathbb{R}} |x(u)|\, du < \infty$ computes:

$$\hat{x}(\omega) = \langle x, e_\omega \rangle = \int_{\mathbb{R}} x(u) e^{-i\omega u}\, du \quad , \quad \omega \in \mathbb{R}$$

It thus tests the signal x against each sinusoid, and records which frequencies are present in x through $\hat{x}$.

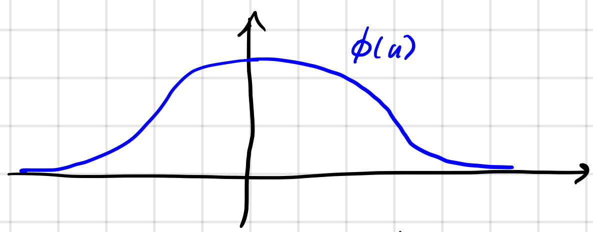
Assume $\int_{\mathbb{R}} |\hat{x}(\omega)|\, d\omega < \infty$. Then knowing $\hat{x}$ is equivalent to knowing x since:

$$x(u) = \int_{\mathbb{R}} \hat{x}(\omega) e^{i\omega u}\, d\omega$$

We will let $\phi: \mathbb{R} \to \mathbb{R}$ denote a low pass filter. This means:

$$\hat{\phi}(\omega) = 0 \text{ for all } |\omega| > \pi \quad \text{and} \quad 1 = \hat{\phi}(0) \geq |\hat{\phi}(\omega)|$$

Intuitively, ϕ will be a "bump function":



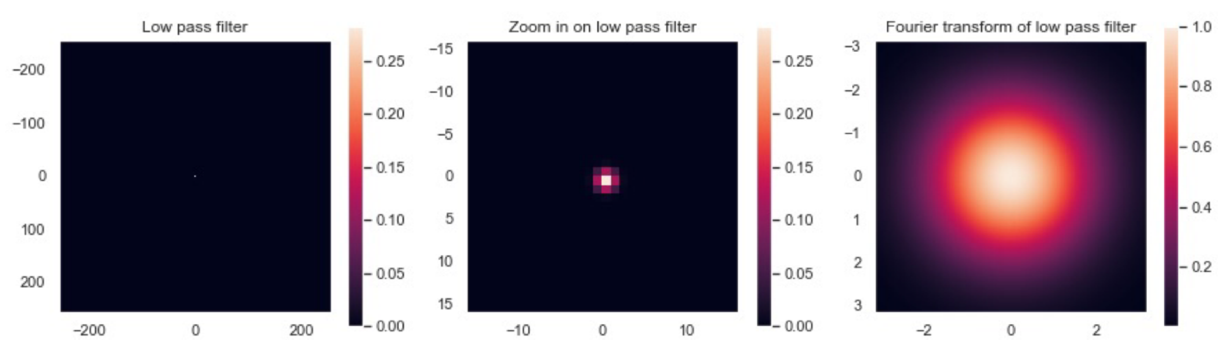
Filtering x with ϕ computes: $x * \phi$

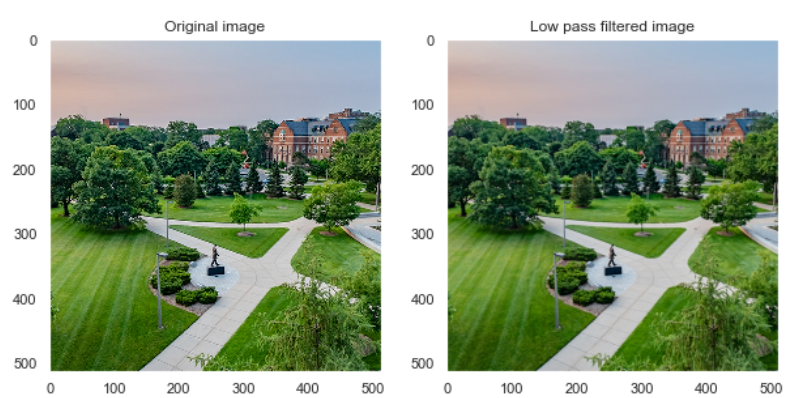
The resulting signal $x * \phi$ is a smoothed, or blurred, version of x. Since

$$\widehat{(x * \phi)}(\omega) = \hat{x}(\omega)\hat{\phi}(\omega) \qquad \text{(Fourier convolution theorem)}$$

It keeps only the low frequencies of x contained in $[-\pi, \pi]$.

Here is an example:



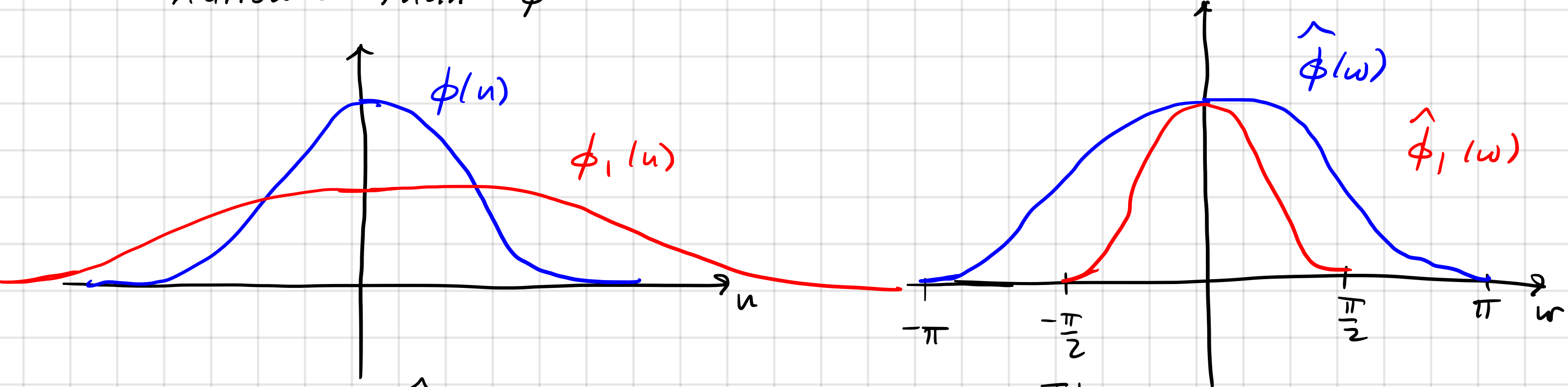


Notice the low pass filter is quite small, and is essentially a 3×3 filter (as in the VGG network). Because it is a low pass filter, it replaces every pixel in the image with a weighted average of the pixels in a 3×3 neighborhood around the central pixel. The resulting image is similar to the original image, but is a slightly smoothed / blurred version. It retains most of the frequency content of the original image, but loses some of the high frequencies. If one looks carefully at the two images, one can see some of the very fine detail is lost.

We can dilate ϕ to enlarge it, which will allow us to smooth the signal more drastically:

$$\phi_J(u) = 2^{-J}\phi(2^{-J}u) \Rightarrow \widehat{\phi}_J(\omega) = \widehat{\phi}(2^J\omega)$$

For $J > 0$, ϕ_J will be wider than ϕ but $\widehat{\phi}_J$ will be narrower than $\widehat{\phi}$:

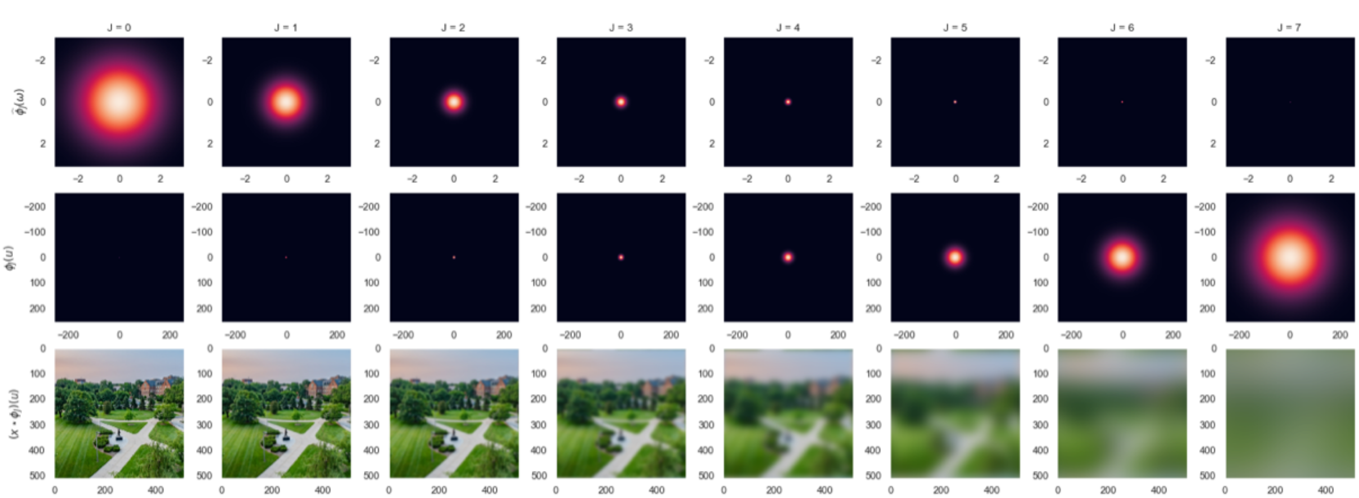


Notice: $\widehat{\phi}_J(\omega) = 0$ for all $|\omega| > \pi/2^J$

When we filter x with ϕ_J, that is we compute $x * \phi_J$, we blur x even more. Indeed, since

$$\widehat{(x * \phi_J)}(\omega) = \widehat{x}(\omega)\widehat{\phi}_J(\omega)$$

we see that $x * \phi_J$ only retains the frequencies of x contained in $|\omega| \leq \pi/2^J$. Here is the filtering of the same image as before for different dyadic scales 2^J:



The top row is the Fourier transform of ϕ_J, $\hat{\phi}_J(\omega)$. The middle row is $\phi_J(u)$. The bottom row is $(x * \phi_J)(u)$. The scales range over $0 \le J \le 7$. The low pass function here is a Gaussian,

$$\phi(u) = \frac{1}{2\pi\sigma^2} e^{-|u|^2/2\sigma^2} \Rightarrow \hat{\phi}(\omega) = e^{-\sigma^2|\omega|^2/2}$$

We choose $\sigma = 3/4$. Notice as the scale increases, ϕ_J becomes larger and $\hat{\phi}_J(\omega)$ becomes smaller. We average in larger and larger neighborhoods, which progressively blurs the image more and more. From a frequency perspective, we retain fewer and fewer frequencies in the original image x. Visually, the increased blur makes it harder to distinguish translations and small deformations of the image. The following theorem quantifies this for translations:

Theorem (Mallat 2012): There is a constant $C > 0$, depending on ϕ, such that for all $t \in \mathbb{R}$ and $x \in L^2(\mathbb{R})$:

$$\| x * \phi_J - x_t * \phi_J \|_2 \le C \cdot 2^{-J} \cdot |t| \cdot \|x\|_2$$

This theorem shows the representation $\Phi(x) = x * \phi_J$ is translation invariant up to the scale 2^J. But how does this relate to neural networks? To understand this, we will need to appeal to results from sampling theory:

Theorem (Shannon-Nyquist): Suppose $\hat{x}(\omega) = 0$ for all $|\omega| > \pi/s$ for some $s > 0$. Then x can be recovered from the downsampled version of x defined by

$$x_d(n) = x(sn), \quad n \in \mathbb{Z}$$

Notice if $s = 1$ then $\hat{x}(\omega) = 0$ for all $|\omega| > \pi$ and we can recover $x : \mathbb{R} \to \mathbb{R}$ from $x_d : \mathbb{Z} \to \mathbb{R}$, $x_d(n) = x(n)$. This is one way to think of a natural image. The underlying scene is x and the image is x_d, which has been sampled along "integer" pixels. Since high resolution images are good representations of the scene, we can interpret this as $\hat{x}(\omega) = 0$ for $|\omega| > \pi$ (warning: If you are comparing different cameras, there is some danger in this)

Since we assumed $\hat{\phi} = 0$ for $|\omega| > \pi$, this is why $x * \phi$, depicted earlier, is a good approximation of x since it retains nearly all of $\hat{x}(\omega)$ (only the corners are lost). On the other hand, this is intuitively clear since ϕ averaged over a 3x3 window.

Notice that for $\phi_J(u) = 2^{-J}\phi(2^{-J}u) \Rightarrow \widehat{\phi_J}(\omega) = \hat{\phi}(2^J\omega)$ we have $\widehat{\phi_J}(\omega) = 0$ for all $|\omega| > \pi/2^J$. Since $\widehat{(x * \phi_J)}(\omega) = \hat{x}(\omega)\widehat{\phi_J}(\omega)$ this means that $\widehat{(x * \phi_J)}(\omega) = 0$ for all $|\omega| > \pi/2^J$. Therefore we can represent $x * \phi_J$ via:

$$(x * \phi_J)_d(n) = (x * \phi_J)(2^J n)$$

Thus we downsample $x * \phi_J$ by a factor 2^J. This is not quite like CNNs which usually pool in factors of 2. Also, ϕ is small, but ϕ_J is larger by a factor 2^J. So there are some differences, at least it would appear so. In fact things are not so different. Indeed the following implements $x * \phi_J$:

$$x \longrightarrow \underbrace{x * \phi_1 \downarrow_2} \longrightarrow (x * \phi_1 \downarrow_2) * \phi_1 \downarrow_2 \longrightarrow \cdots \quad J \text{ times}$$

convolve x with ϕ_1 (reminder $\hat{\phi}_1(\omega) = 0$ for all $|\omega| > \pi/2$) and downsample by a factor of 2

Note: ϕ_1 is essentially 7×7

Therefore we can implement the translation invariant operator $x * \phi_J$ by composing convolution with ϕ_1 and downsampling by a factor of 2, J times. This is a simple type of CNN with same single filter at each layer and no nonlinearities.

Okay, so we see that $\Phi(x) = x * \phi_J$ is a translation invariant representation of x and can be viewed as simple CNN. On the other hand, we know

$$\widehat{(x * \phi_J)}(\omega) = \hat{x}(\omega)\hat{\phi}_J(\omega) \neq 0 \text{ only for } |\omega| \leq \pi/2^J$$

So we have lost a lot of $\hat{x}$ and thus x (indeed recall the pictures of $x * \phi_J$ which were very blurry).

To recover the lost information we turn to something called a wavelet transform. A wavelet $\psi : \mathbb{R} \to \mathbb{R}$ or $\psi : \mathbb{R} \to \mathbb{C}$ is a localized, oscillating waveform with zero average. The last property means

$$\hat{\psi}(0) = \int_{\mathbb{R}} \psi(u)\,du = 0$$

Thus, unlike the low pass filter ϕ_J for which $\sup_\omega |\hat{\phi}_J(\omega)| = \hat{\phi}_J(0)$, the wavelet ψ has its frequency support concentrated around a frequency (or frequencies) away from zero. Like the low pass filter ϕ, we can dilate ψ:

$$\psi_j(u) = 2^{-j}\psi(2^{-j}u) \Rightarrow \widehat{\psi_j}(\omega) = \hat{\psi}(2^j\omega)$$

A wavelet transform computes:

$$J \geq 1, \qquad W_J x = \{ x * \phi_J(u),\ x * \psi_j(u) \ : \ u \in \mathbb{R},\ 1 \leq j \leq J \}$$

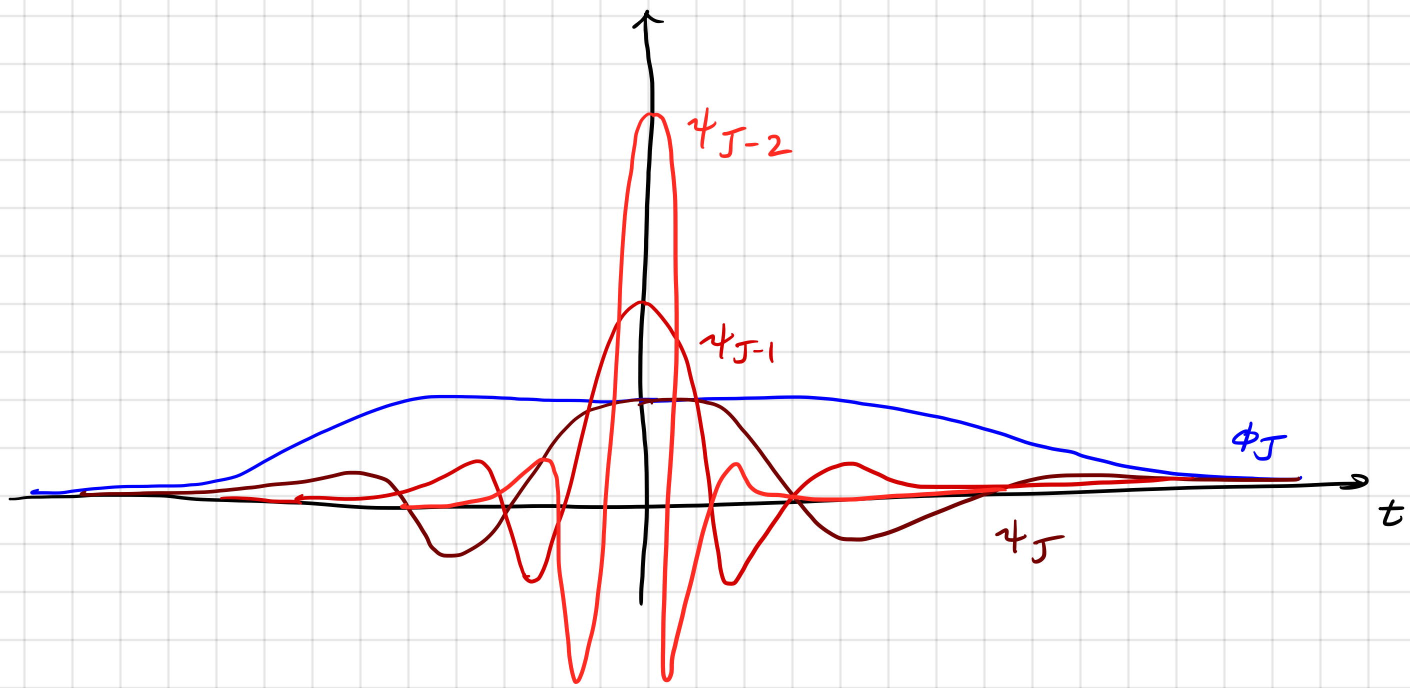
In other words, in addition to averaging over x with $x * \phi_J$, we filter x with J smaller wavelets that recover the details in x lost by $x * \phi_J$. In terms of frequencies, $x * \phi_J$ keeps the low frequencies of x (hence ϕ_J is a low pass filter) while $\{x * \psi_j\}_{1 \le j \le J}$ keeps the high frequencies of x (hence the ψ_j filters are called high pass filters).

Suppose, as we observed for natural images, that $\hat{x}(\omega) = 0 \ \forall\ |\omega| > \pi$.
If

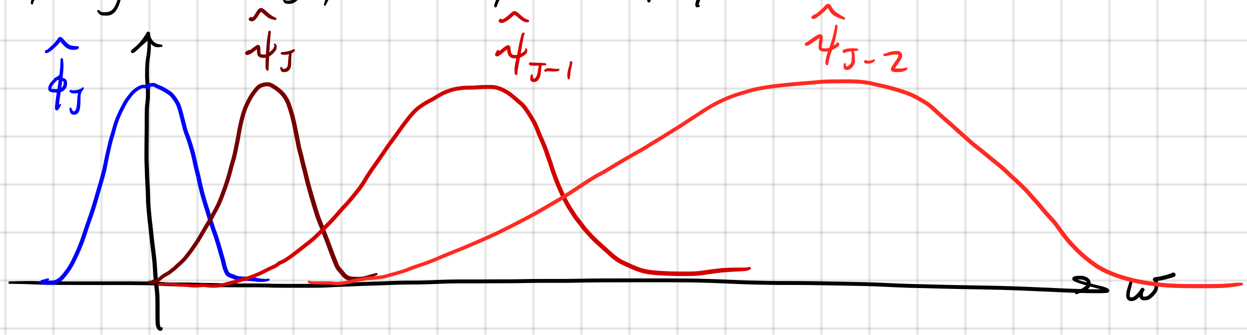
$$\underbrace{0 < A \le |\hat{\phi}_J(\omega)|^2 + \sum_{j=1}^{J} |\hat{\psi}_j(\omega)|^2 \le B < +\infty}_{\text{This means all the frequencies are covered by our low pass } \phi_J \text{ and wavelets } \{\psi_j\}_{1 \le j \le J}} \quad \text{for all } \omega \in [-\pi, \pi]$$

then $W_J x = \{x * \phi_J, x * \psi_j : 1 \le j \le J\}$ is invertible, meaning knowing $W_J x$ is as good as knowing x. The proof of this is based on the fact that we stated earlier, which is that knowing $\hat{x}(\omega)$ is as good as knowing $x(u)$.

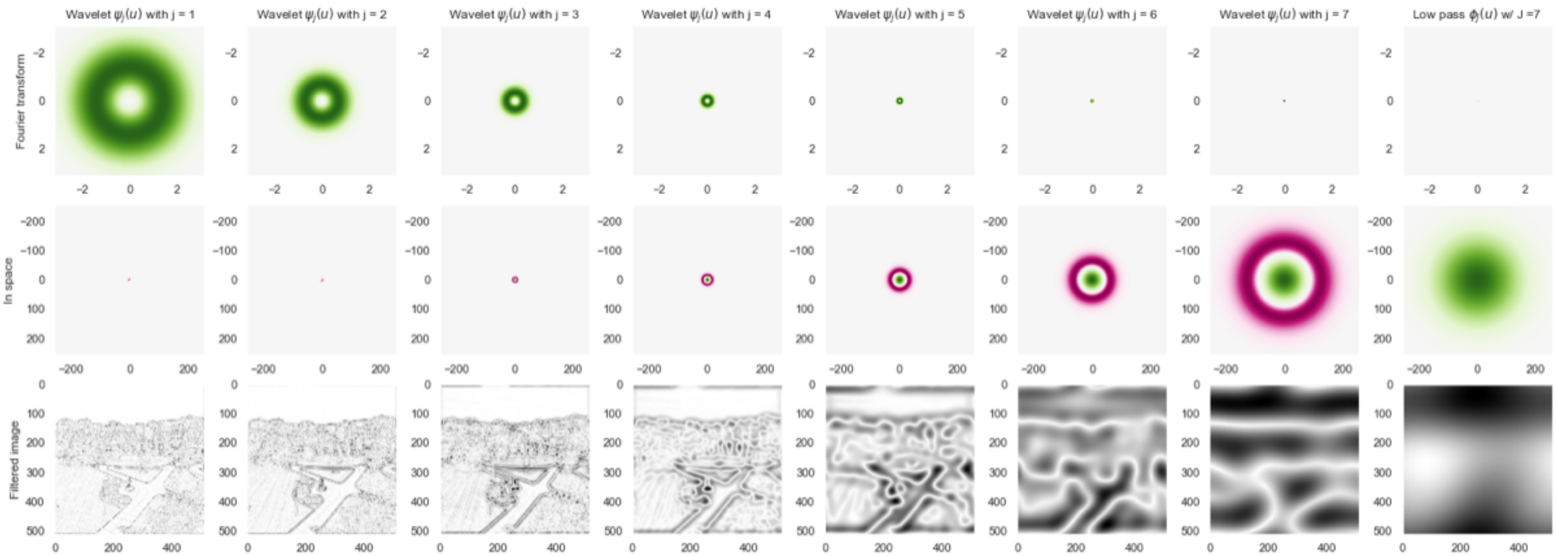
In time/space we have the following plots:



And in frequency (I only plot the positive frequencies):



Here are pictures in 2D on the same image as before:



In the first 2 rows green is positive, pink is negative, white is zero.
In that last row white is zero and black in max positive value.
The first seven columns are wavelets going from small scale in space to large scale in space, so $\psi_j(u)$ (2nd row) and $\hat{\psi}_j(\omega)$ (1st row) for $1 \le j \le J = 7$. The last column is the low pass filter $\phi_J(u)$ (2nd row) and $\hat{\phi}_J(\omega)$ (1st row).
We see in frequency the wavelets capture the high frequencies that the low pass misses. These wavelets are localized oscillating waveforms where the oscillations flow radially out of the center.
When computing the filtration $x * \psi_j$ (the 3rd row), the small wavelets act as edge detectors; here we plot:

$$\left[|x_r * \psi_j|^2 + |x_g * \psi_j|^2 + |x_b * \psi_j|^2 \right]^{1/2}$$

The larger wavelets capture larger scale information in the image.
In this example, ϕ is the same as before and

$$\Delta = \text{Laplacian} \longrightarrow \psi(u) = -(\Delta g)(u), \quad g(u) = \frac{1}{2\pi\alpha^2} e^{-|u|^2/2\alpha^2}, \quad \alpha = \frac{1}{2}$$

$$\Rightarrow \hat{\psi}(\omega) = |\omega|^2 \hat{g}(\omega), \quad \hat{g}(\omega) = e^{-\alpha^2|\omega|^2/2}$$

Like ϕ_J, we can also implement $x * \psi_j$ with a simple CNN:

$$\underbrace{x \mapsto x * \phi_1 \downarrow_2 \mapsto (x * \phi_1 \downarrow_2) * \phi_1 \downarrow_2 \mapsto \cdots \mapsto \underbrace{(((x * \phi_1 \downarrow_2) * \phi_1 \downarrow_2) * \cdots * \phi_1 \downarrow_2)}_{j-1 \text{ times}} * \psi_1}_{x * \psi_j}$$

where ϕ_1 and ψ_1 are very small filters.

But what about translation invariance? Recall $x * \phi_J$ is translation invariant up to the scale 2^J. We also have

$$\int_{\mathbb{R}} x * \phi_J(u)\,du = \int x(u)\,du$$

On the other hand

$$\int_{\mathbb{R}} \psi(u)\,du = 0 \Rightarrow \int_{\mathbb{R}} \psi_j(u)\,du = 0 \Rightarrow \int_{\mathbb{R}} x * \psi_j(u)\,du = 0 \leftarrow \text{tells you nothing about } x$$

What if we convolved $x * \psi_j$ with ϕ_J? We know $x * \phi_J$ is translation invariant so $x * \psi_j * \phi_J$ is also translation invariant.

But if $1 = |\hat{\phi}_J(\omega)|^2 + \sum_{j=1}^{J} |\hat{\psi}_j(\omega)|^2$ and $\hat{\phi}_J(0) = 1$ then

the support of $\hat{\phi}_J(\omega)$ must overlap very little with the support of $\hat{\psi}_j(\omega)$. But:

$$\widehat{(x * \psi_j * \phi_J)}(\omega) = \hat{x}(\omega)\hat{\psi}_j(\omega)\hat{\phi}_J(\omega) \simeq 0$$

which is also not helpful!

Therefore we need something nonlinear. The idea is $x * \psi_j$ captures the high frequencies of x. We need to "push" this high frequency information down to the low frequencies so we can obtain a nontrivial, translation invariant representation of x. The wavelet scattering transform uses the absolute value / modulus operator. Some of these results could potentially be adapted to ReLU. We also note:

$$|z| = \mathrm{ReLU}(z) + \mathrm{ReLU}(-z) \quad \text{for } z \in \mathbb{R}$$

With the absolute value / modulus we send:

$$x * \psi_j \mapsto |x * \psi_j| = x_j$$

These new functions x_j have Fourier transform $\hat{x}_j(\omega)$ with some support at $\omega = 0$. Indeed:

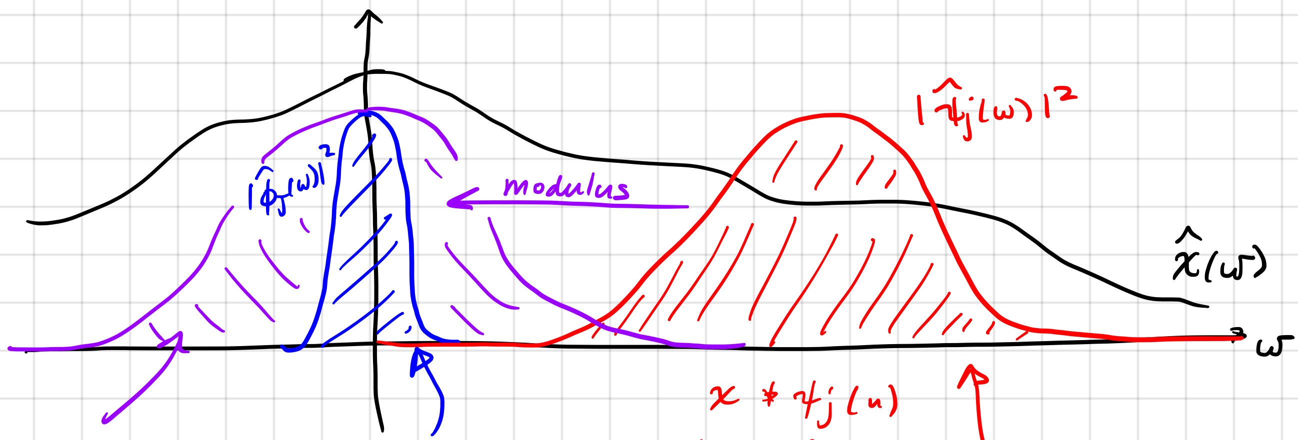
$$\hat{x}_j(0) = \int_{\mathbb{R}} |x * \psi_j(u)|\,du \geq 0$$

Therefore each function $|x * \psi_j| * \phi_J \neq 0$ and using the previous theorem each function $|x * \psi_j| * \phi_J$ is translation invariant up to the scale 2^J. Thus far we have:

$$\Phi(x) = S_J^2(x) = \{ x * \phi_J,\ |x * \psi_j| * \phi_J \ : 1 \leq j \leq J \}$$

$\uparrow$ Wavelet scattering of x w/ two wavelet layers is a translation invariant representation of x up to the scale 2^J.

The idea in frequency is the following:



frequency support of $|x * \psi_j(u)|$

$x * \phi_J(u)$ keeps frequencies in this range and is translation invariant

$x * \psi_j(u)$ keeps frequencies in this range

Note that $|x * \psi_j| * \phi_J(u)$ keeps the frequencies in the intersection of the blue bump and the purple bump. The coefficients $|x * \psi_j| * \phi_J(u)$ are translation invariant up to the scale 2^J and retain some of the information from $x * \psi_j(u)$.

On the other hand, the above picture shows that $x * \phi_J$ and $|x * \psi_j| * \phi_J$ do not capture all of $\hat{x}(\omega)$ and hence all of $x(u)$. Indeed, when we computed $|x * \psi_j| * \phi_J(u)$ we lost information from $|x * \psi_j(u)|$ just as $x * \phi_J(u)$ lost information from $x(u)$. In order to recover this lost high frequency information we can iterate the filterings with wavelets:

$$|x * \psi_{j_1}| * \psi_{j_2} \quad , \quad 1 \leq j_1, j_2 \leq J$$

Since

$$\underbrace{W_J}_{\text{wavelet transform}} |x * \psi_{j_1}| = \{ |x * \psi_{j_1}| * \phi_J \, , \, |x * \psi_{j_1}| * \psi_{j_2} : 1 \leq j_2 \leq J \}$$

and since knowing $W_J |x * \psi_{j_1}|$ is as good as knowing $|x * \psi_{j_1}|$, we see that $|x * \psi_{j_1}| * \psi_{j_2}$ indeed recovers the information lost in only computing $|x * \psi_{j_1}| * \phi_J$.

However, similar to before, $|x * \psi_{j_1}| * \psi_{j_2}$ is translation equivariant but not translation invariant and

$$|x * \psi_{j_1}| * \psi_{j_2} * \phi_J \approx 0$$

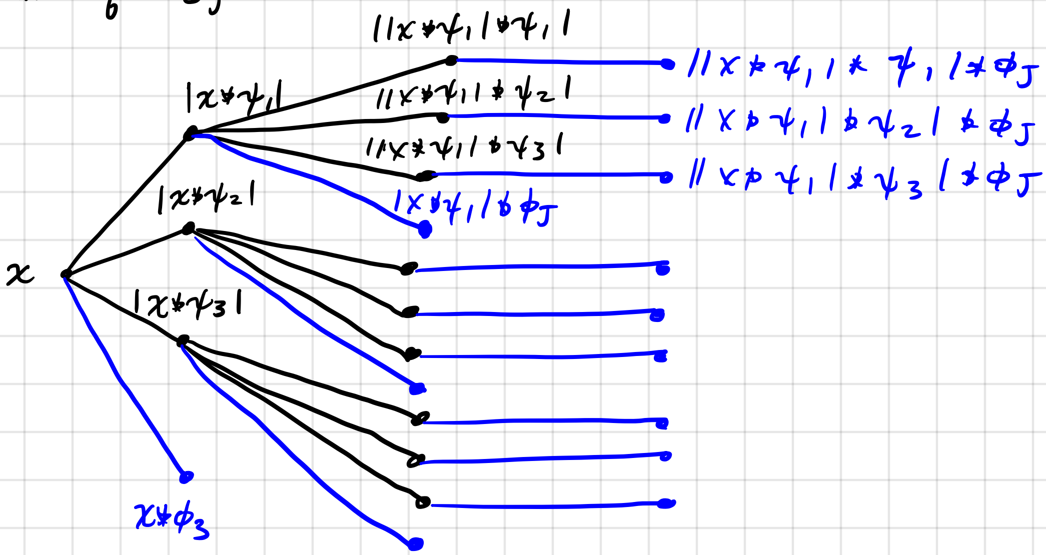
Therefore we need to apply another nonlinearity (another absolute value / modulus) and then apply the low pass filter:

$$||x * \psi_{j_1}| * \psi_{j_2}| * \phi_J$$

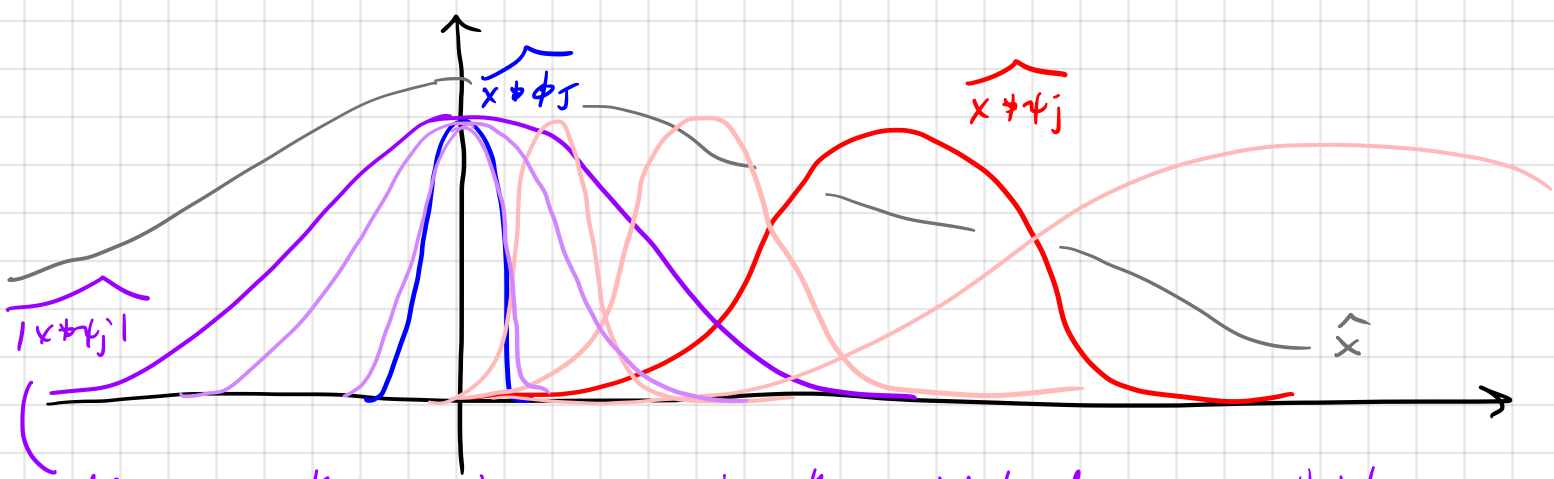
Our representation of x is now:

$$\Phi(x) = S_J^3(x) = \left\{ \begin{array}{l} x * \phi_J \\ |x * \psi_{j_1}| * \phi_J \\ ||x * \psi_{j_1}| * \psi_{j_2}| * \phi_J \end{array} \right\}$$

We call $S_J^3(x)$ a wavelet scattering transform. Here is a diagram of $S_J^3(x)$:



In frequency space, here is what is happening:

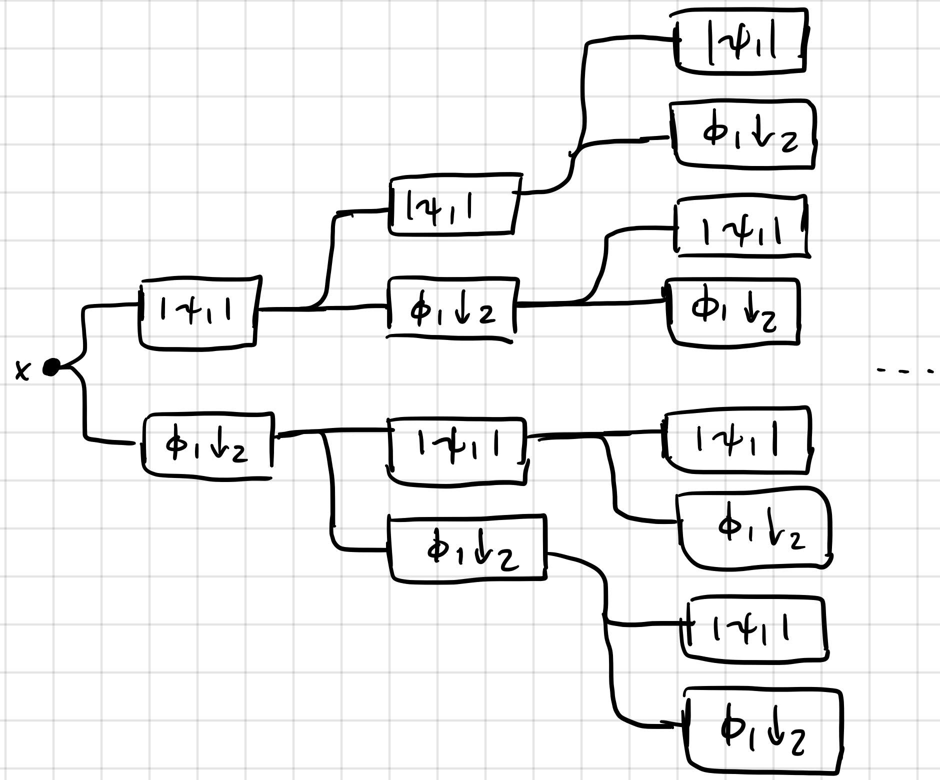


decompose the purple curve using the pink/red curves, which corresponds to computing $|x * \psi_{j_1}| * \psi_{j_2}$. Then push that frequency information down with $||x * \psi_{j_1}| * \psi_{j_2}|$.

One can see that even $S_J^3(x)$ loses some information from $\hat{x}(\omega)$ and hence $x(u)$. Therefore we can keep going. An $(m+1)$-wavelet layer transform computes:

$$S_J^{m+1}(x) = \left\{ \begin{array}{ll} x * \phi_J, & \\ |x * \psi_{j_1}| * \phi_J & : \ 1 \le j_i \le J \\ ||x * \psi_{j_1}| * \psi_{j_2}| * \phi_J & \quad \text{for all } 1 \le i \le m \\ \vdots & \\ |||x * \psi_{j_1}| * \psi_{j_2}| * \cdots * \psi_{j_m}| * \phi_J & \end{array} \right\}$$

The wavelet scattering transform is a mathematical model for a convolutional neural network. Recalling our discussion on sampling theory and downsampling, we can also write it in terms of only small filters:



where $x - \boxed{\phi_1 \downarrow 2}$ computes $x * \phi_1 \downarrow 2$

and $x - \boxed{|\psi_1|}$ computes $|x * \psi_1|$

Now let us discuss some additional theoretical properties of the wavelet scattering transform. We will assume the wavelets and low pass filter perfectly cover the frequency axis, meaning:

$$|\hat{\phi}_J(\omega)|^2 + \sum_{j \leq J} |\hat{\psi}_j(\omega)|^2 = 1 \quad \forall \; \omega \in \mathbb{R} \qquad (\psi \text{ real valued})$$

$$|\hat{\phi}_J(\omega)|^2 + \sum_{j \leq J} |\hat{\psi}_j(\omega)|^2 = 2 \quad \forall \; \omega \geq 0 \qquad (\psi \text{ complex valued})$$

We also define the norm of $S_J^{m+1}(x)$ as:

$$\| S_J^{m+1}(x) \|^2 = \| x * \phi_J \|_2^2 + \sum_{\ell=1}^{m+1} \| \, |||x * \psi_{j_1}| * \psi_{j_2}| * \cdots * \psi_{j_\ell}| * \phi_J \|_2^2$$

Individually each function in $S_J^{m+1}(x)$ is translation invariant up to to the scale 2^J. In fact all of $S_J^{m+1}(x)$ is translation invariant:

Theorem (Mallat 2012): For $x \in L^2(\mathbb{R})$, let $x_t(u) = x(u-t)$. Then:

$$\| S_J^{m+1}(x) - S_J^{m+1}(x_t) \| \leq C \cdot m \cdot 2^{-J} \cdot |t| \cdot \|x\|_2$$

Note the linear scaling in the number of layers, $m+1$, is better than one would get by counting up all of the functions in $S_J^{m+1}(x)$. The bound also holds even with an infinite number of functions, which can be thought of as infinitely wide layers.

Our other goal was stability to diffeomorphisms. For this we have:

Theorem (Mallat 2012): Let $x \in L^2(\mathbb{R})$ and $\tau \in C^2(\mathbb{R})$ with $\|\tau'\|_\infty < 1/2$ and $x_\tau(u) = x(u - \tau(u))$. Then:

$$\| S_J^{m+1}(x) - S_J^{m+1}(x_\tau) \| \leq C \cdot m \cdot \left[2^{-J} \|\tau\|_\infty + J \cdot \|\tau'\|_\infty + \|\tau''\|_\infty \right] \|x\|_2$$

Can be replaced by other things, such as R where $\operatorname{supp}(x) \subseteq B_{2R}(0)$

These two theorems show $\Phi(x) = S_J^{m+1}(x)$ is invariant to translations and stable to diffeomorphisms. It is also $L^2(\mathbb{R})$ stable:

Theorem (Mallat 2012): Let $x, \tilde{x} \in L^2(\mathbb{R})$. Then:

$$\| S_J^{m+1}(x) - S_J^{m+1}(\tilde{x}) \| \leq \| x - \tilde{x} \|_2$$

If the wavelet ψ satisfies additional constraints, then

$$\lim_{m \to \infty} \| S_J^{m+1}(x) \| = \| x \|_2$$

L^2 stability is complementary to translation invariance and stability diffeomorphisms, and is important in its own right.

The energy preservation shows that the transform neither creates nor destroys "mass," here as measured by $\|x\|_2$. It shows that the collection of functions in $S_J^\infty(x)$ partition the energy of x.

The one thing we have not addressed yet is the stacking of filters. Let us start with rotations. A directional filter is a filter that oscillates in a certain direction. We can model such filters as:

$$u \in \mathbb{R}^2, \qquad \psi(u) = g(|u|)\, e^{i\xi \cdot u}, \qquad |u| = [\,u(1)^2 + u(2)^2\,]^{1/2}$$

complex valued

$$\text{OR} \quad \left.\begin{aligned} \psi(u) &= g(|u|)\cos(\xi \cdot u) \\ \psi(u) &= g(|u|)\sin(\xi \cdot u) \end{aligned}\right\} \text{ real valued}$$

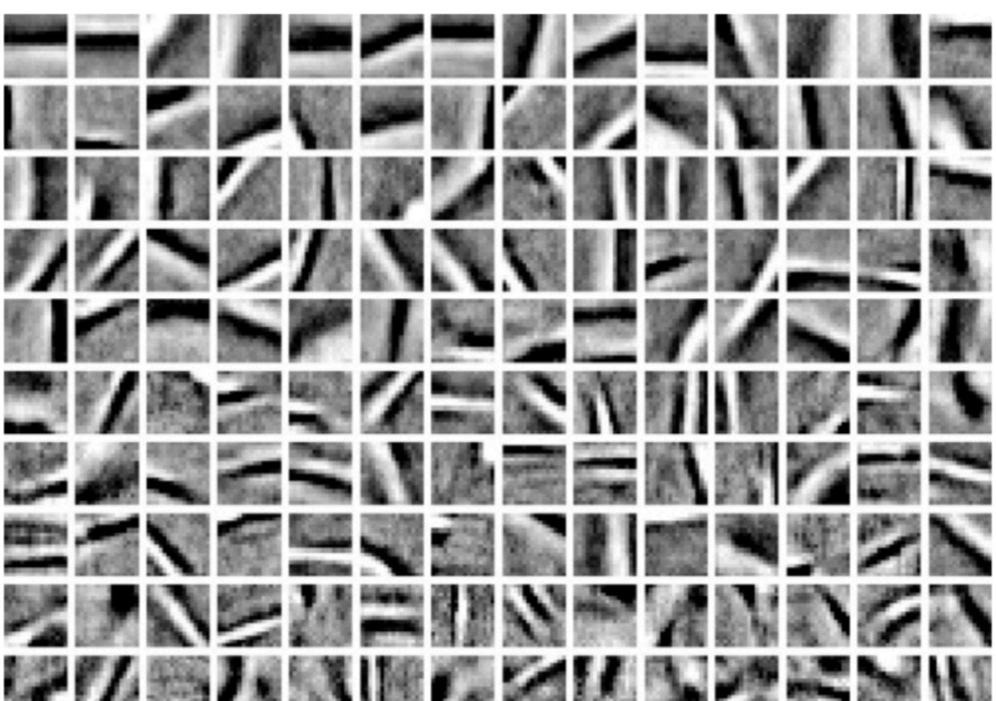
(just the real and imaginary parts of the complex valued filter)

The window g is often non-negative, e.g., a Gaussian

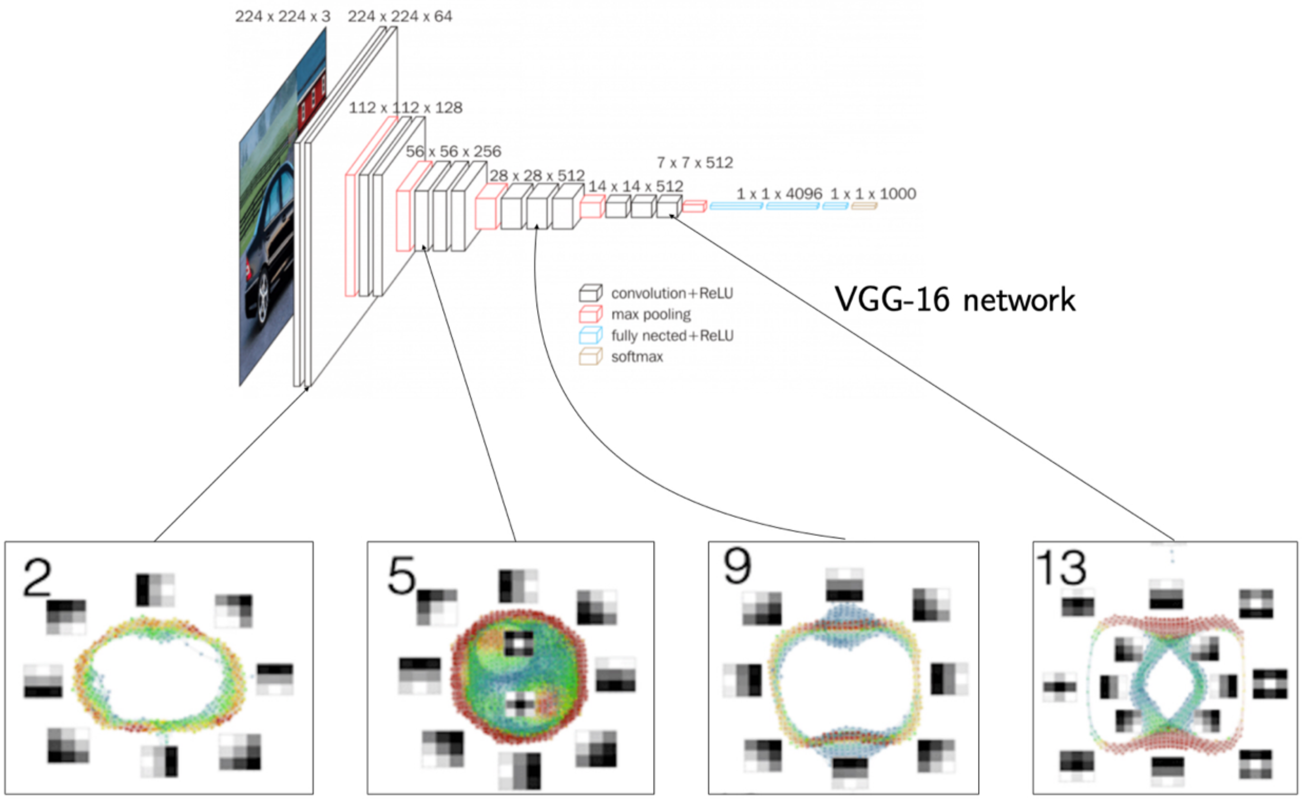
$$g(|u|) = \frac{1}{2\pi\sigma^2} e^{-|u|^2/2\sigma^2}$$

The filter ψ oscillates in the direction $\frac{\xi}{|\xi|}$. For example, if $\xi = (\xi_1, 0)$ it will oscillate in the horizontal direction, if $\xi = (0, \xi_2)$ it will oscillate in the vertical direction, and if $\xi = (\xi_0, \xi_0)$ it will oscillate at a $45°$ angle.

Directional filters are useful for image processing because they isolate edges and other patterns in the image only in certain directions. For example, a directional filter ψ that oscillates horizontally will pick up on a vertical edge. It turns out that many dictionary learning algorithms and deep networks learn directional filters (at least in the early layers for deep networks), so they are a good model for filters.



Filters learned through dictionary learning (Figure taken from A Wavelet Tour of Signal Processing)



The VGG network (top) and the filters learned at selected layers (bottom), organized using methods from topological data analysis [organizations by Carlsson and Gabrielsson 2018]

In both cases we can see many directional filters at different orientations. A striking example is panel 2 and panel 9 of the VGG network. These panels show that often the filters are essentially the same filter, that has been rotated:

$$\psi_\theta(u) = \psi(R_\theta^{-1}u) = g(|R_\theta^{-1}u|)\, e^{i\xi\cdot R_\theta^{-1}u}$$
$$= g(|u|)\, e^{iR_\theta\xi\cdot u}$$

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

2x2 rotation matrix

Notice ψ_θ oscillates in the direction $R_\theta\xi/|\xi|$, but otherwise is like the original filter ψ. Let

$$\Theta = \{ 2\pi m/M \; : \; 0 \leq m < M \}$$

Then $\{\psi_\theta\}_{\theta\in\Theta}$ defines a stack of M filters that are rotations of a base filter ψ. Let us compute the convolution of an image x with this stack. Let

$$x_\varphi(u) = x(R_\varphi^{-1}u)$$

be the rotation of x by $\varphi \in [0, 2\pi)$.

Then: $$(x_\varphi * \psi_\theta)(u) = \int_{\mathbb{R}^2} x(R_\varphi^{-1} v)\, \psi(R_\theta^{-1} u - R_\theta^{-1} v)\, dv$$

$$t = R_\varphi^{-1} v$$

$$= \int_{\mathbb{R}^2} x(t)\, \psi(R_\theta^{-1} u - R_\theta^{-1} R_\varphi t)\, dt$$

$$= \int_{\mathbb{R}^2} x(t)\, \psi(\underbrace{R_\theta^{-1} R_\varphi}_{(R_{\theta-\varphi})^{-1}} (R_\varphi^{-1} u - t))\, dt$$

$$= \int_{\mathbb{R}^2} x(t)\, \psi(R_{\theta-\varphi}^{-1} (R_\varphi^{-1} u - t))\, dt$$

$$= (x * \psi_{\theta-\varphi})(R_\varphi^{-1} u)$$

We see that $x * \psi_\theta$ is not equivariant with respect to rotations, but rather two things are happening:

$$(x_\varphi * \psi_\theta)(u) = (x * \underbrace{\psi_{\theta-\varphi}}_{})(\underbrace{R_\varphi^{-1} u}_{})$$

rotation of filtered image (like translations)

Transport of information along the stack of filters (different than translations).

With a one layer CNN we would apply a pointwise nonlinearity:

$$\sigma(x_\varphi * \psi_\theta)(u) = \sigma(x * \psi_{\theta-\varphi})(R_\varphi^{-1} u)$$

and we can aggregate information across filters:

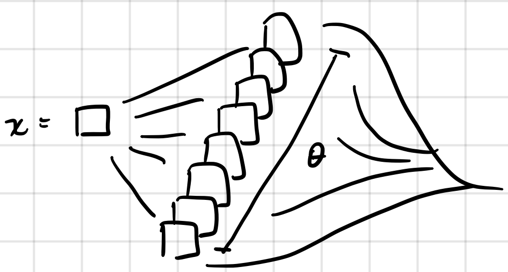
$$\Phi(x_\varphi)(u) = \sum_{\theta \in \Theta} \sigma(x_\varphi * \psi_\theta)(u) = \sum_{\theta \in \Theta} \sigma(x * \psi_{\theta-\varphi})(R_\varphi^{-1} u)$$

$$\approx \sum_{\theta \in \Theta} \sigma(x * \psi_\theta)(R_\varphi^{-1} u)$$

Thus the transformation

$$x(u) \longmapsto \Phi(x)(u) = \sum_{\theta \in \Theta} \sigma(x * \psi_\theta)(u)$$

is equivariant to rotations since $\Phi(x_\varphi)(u) \approx \Phi(x)(R_\varphi^{-1}u)$ (as we showed on the previous page) and it is equivariant with respect to translations since it uses convolution operators. Here is a diagram:



$$\square = \Phi(x) = \sum_{\theta \in \Theta} \sigma(x * \psi_\theta)$$

$\sigma(x * \psi_\theta)$

On the other hand, we summed over Θ, which removes much of our directional information. A two layer CNN handles this better. In this case, the output of the first layer is the M channel representation of x given by:

$$x^{(1)} = \left(x^{(1)}_{\theta_1}\right)_{\theta_1 \in \Theta} = \left(\sigma(x * \psi_{\theta_1})\right)_{\theta_1 \in \Theta}$$

Now we define a collection of M^2 filters for the second layer as:

$$\psi_{\theta_1, \theta_2}(u) = \psi_{\theta_1 + \theta_2}(u) \quad , \quad \theta_1, \theta_2 \in \Theta$$

Following our earlier discussion on stacks of filters and the VGG network, we compute in the second layer:

$$x^{(2)}_{\theta_2} = \sigma\left(\sum_{\theta_1 \in \Theta} x^{(1)}_{\theta_1} * \psi_{\theta_1, \theta_2}\right) = \sigma\left(\sum_{\theta_1 \in \Theta} \sigma(x * \psi_{\theta_1}) * \psi_{\theta_1 + \theta_2}\right)$$

Let x_φ be the rotation of x. Then:

$$\sigma(x_\varphi * \psi_{\theta_1}) * \psi_{\theta_1 + \theta_2}(u) = \overbrace{\sigma(x * \psi_{\theta_1 - \varphi})_\varphi}^{\text{by our previous calculation}} * \psi_{\theta_1 + \theta_2}(u)$$

$$= \sigma(x * \psi_{\theta_1 - \varphi}) * \psi_{\theta_1 - \varphi + \theta_2}(R_\varphi^{-1}u)$$

Therefore :

$$(x_\varphi)^{(2)}_{\theta_2}(u) = \sigma\Big(\sum_{\theta_1 \in \Theta} \sigma(x_\varphi * \psi_{\theta_1}) * \psi_{\theta_1+\theta_2}\Big)(u)$$

$$= \sigma\Big(\sum_{\theta_1 \in \Theta} \sigma(x * \psi_{\theta_1-\varphi}) * \psi_{\theta_1-\varphi+\theta_2}\Big)(R_\varphi^{-1} u)$$

$$\approx \sigma\Big(\sum_{\theta_1 \in \Theta} \sigma(x * \psi_{\theta_1}) * \psi_{\theta_1+\theta_2}\Big)(R_\varphi^{-1} u)$$

$$= x^{(2)}_{\theta_2}(R_\varphi^{-1} u)$$

Thus $x^{(2)}_{\theta_2}$ is equivariant with respect to translations and rotations, and unlike the 1-layer CNN we have a stack of M such maps given by

$$x^{(2)} = (x^{(2)}_{\theta_2})_{\theta_2 \in \Theta}$$

Remark : We can replace the rotation group with another group G. Then

$$x_g(u) = x(g^{-1} \cdot u)$$ where $g \in G$ and $g \cdot u \in \mathbb{R}^2$ is the group action of g on u.

Then we have a stack of filters :

$$\{\psi_g\}_{g \in G}, \quad \psi_g(u) = \psi(g^{-1} \cdot u)$$

Let $h \in G$ as well. Assume $|g \cdot u| = |u|$. Then :

$$x_h * \psi_g(u) = \int_{\mathbb{R}^2} x(h^{-1} \cdot v)\, \psi(g^{-1} \cdot u - g^{-1} \cdot v)\, dv$$

$$t = h^{-1} \cdot v$$

$$= \int_{\mathbb{R}^2} x(t)\, \psi(g^{-1} \cdot u - g^{-1} \cdot h \cdot t)\, dt$$

$$= \int_{\mathbb{R}^2} x(t)\, \psi(g^{-1} h(h^{-1} \cdot u - t))\, dt$$

$$= \int_{\mathbb{R}^2} x(t)\, \psi_{h^{-1}g}(h^{-1} u - t)\, dt$$

Now we can apply similar ideas $\longrightarrow$

$$= x * \psi_{h^{-1}g}(h^{-1} u) = (x * \psi_{h^{-1}g})_h(u)$$

When we train CNNs, they implicitly learn the groups G, and corresponding stacks of filters, over these groups. In practice, the mathematical language of group theory may be too limited.

<u>Remark 2</u>: The groups can be enlarged through the layers. see, e.g., "Understanding deep convolutional networks" by Mallat (2016).

<u>Remark 3</u>: We can keep going with deeper layers by iterating on these ideas.