CS-UY 4563: Lecture 18
Convolutional Feature Extraction

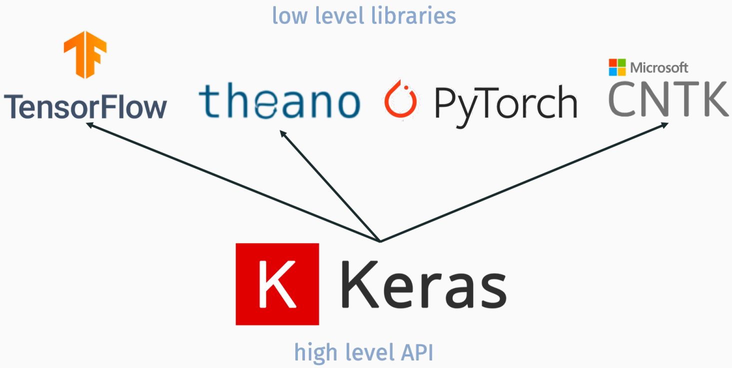
NYU Tandon School of Engineering, Prof. Christopher Musco

COURSE LOGISTICS

- Midterm 2 exam **is canceled.**
    - In place of midterm grade, you will be awarded maximum grade from your homework, first midterm, or project.
- Project Proposal due **tonight**.
    - See guidelines for what to include at: `https://www.chrismusco.com/introml/project_guidelines.pdf`
- New written homework posted. Due **next Monday**.

Quick note from last class. Two demos uploaded on neural networks:

- `keras_demo_synthetic.ipynb`
- `keras_demo_mnist.ipynb`



Low-level libraries have built in optimizers (SGD and improvements) and can automatically perform backpropagation for arbitrary network structures. Also ptimize code for any available GPUs.

Keras has high level functions for defining and training a neural network architecture.

Define model:

```
model = Sequential()
model.add(Dense(units=nh, input_shape=(nin,), activation='sigmoid', name='hidden'))
model.add(Dense(units=nout, activation='softmax', name='output'))
```

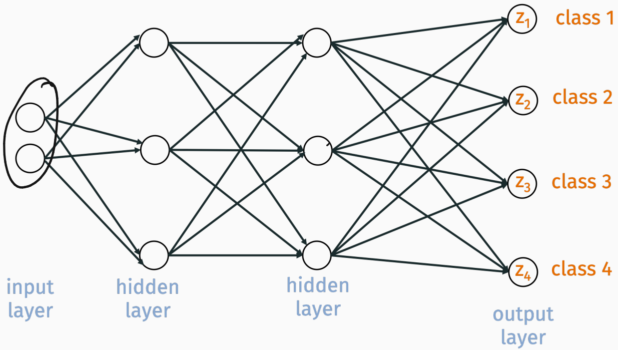
Compile model:

```
opt = optimizers.Adam(lr=0.001)
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Train model:

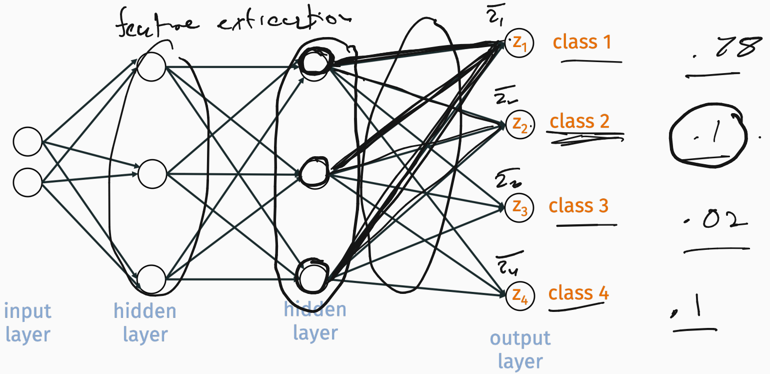
```
hist = model.fit(Xtr, ytr, epochs=30, batch_size=100, validation_data=(Xts,yts))
```

The MNIST demo performs multiclass classification. Typically approach to multiclass problems with neural networks is to have one output neuron per class:



Classification rule: Place in input $\vec{x}$ in class i if z_i is the neuron with maximum value after running $\vec{x}$ through the network.

MULTICLASS CLASSIFICATION



Last layer typically uses a “softmax” nonlinearity to map all values $\bar{z}_1, \ldots, \bar{z}_q$ to values between 0 and 1:

$$z_i = \frac{e^{\bar{z}_i}}{\sum_{j=1}^{q} e^{\bar{z}_j}}$$

Trained using multiclass cross-entropy loss. Let $z_1(\vec{x},\theta),\ldots,z_q(\vec{x},\theta)$ be the outputs obtain when running the network on input $\vec{x}$ with parameters (weights and baises) $\vec{\theta}$.

$$L(y,\vec{x},\vec{\theta}) = -\sum_{i=1}^{q} \mathbb{1}[y=i]\log(z_i(\vec{x},\theta)).$$

output at variable z_i when running network with parameters θ

$y \in 1,\ldots,q$

1

Overall loss for training data $(\vec{x}_1,y_1),\ldots,(\vec{x}_n,y_n)$ is:

$$\mathcal{L}(\vec{\theta}) = \sum_{i=1}^{n} L(y_i,\vec{x}_i,\vec{\theta})$$

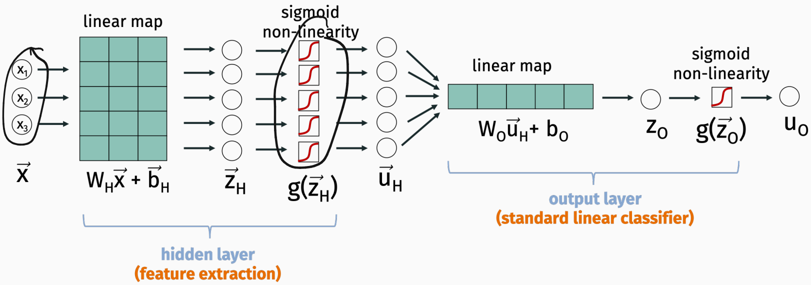
Used in our demo and very standard for neural network classification.

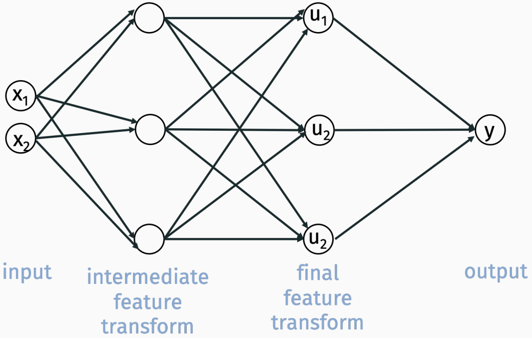
Why do neural networks work so well?

Treat feature transformation/extraction as part of the learning process instead of making this the users job.

But sometimes they still need a nudge in the right direction...

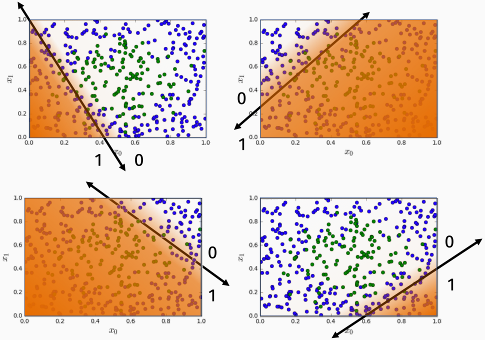
BASIC FEATURE EXTRACTION





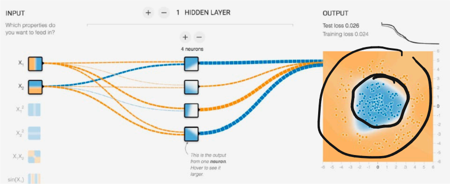
Final output or class label y is a linear function of the final layer variables $u_1, \ldots, u_k$. You could just as well have taken these variables and used them to predict y via linear regression, logistic regression, SVM, any other linear method.

Sigmoid activation: Each hidden variable z_i equal to $\frac{1}{1+e^{-\bar{z}_i}}$ where $z_i = \vec{w}^T\vec{x} + b$ for input $\vec{x}$.

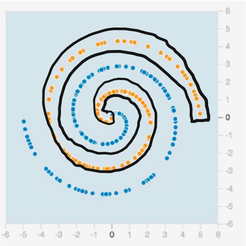


Other non-linearities yield similarly simple feature extractions.

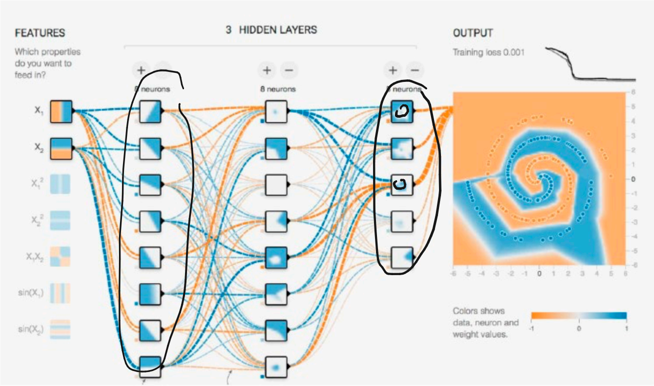
If you combine more hidden variables, you can start building more complicated classifiers.



How about for even more complex datasets?

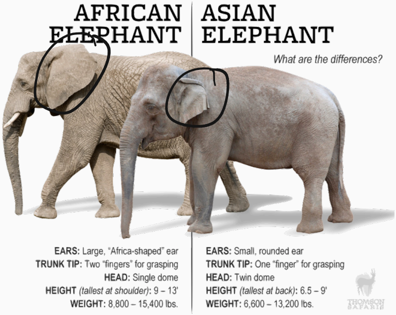


With more layers, complexity starts ramping up...



But there's a limit...

Modern machine learning algorithms can differentiate between images of African and Asian elephants...



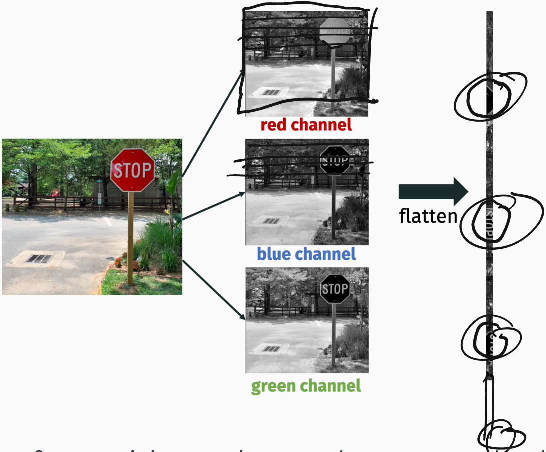
The features needed for a task like this are far more complex then we could expect a network to learn completely on its own using simple combinations of linear layers + non-linearities.

Today's topic: Understand why convolution is a powerful way of extracting features from:

- Image data.
- Audio data.
- Time series data.

Ultimately, can build convolutional networks that already have convolutional feature extraction pre-coded in. Just need to learn weights.

What features would tell use this image contains a stop sign?



Typically way of vectorizing an image chops up and splits up any pixels in the stop sign. We need very complex features to piece these back together again...

Objects or features of an image often involve pixels that are spatially correlated. Convolution explicitly encodes this.

Definition (Discrete 1D convolution[1])

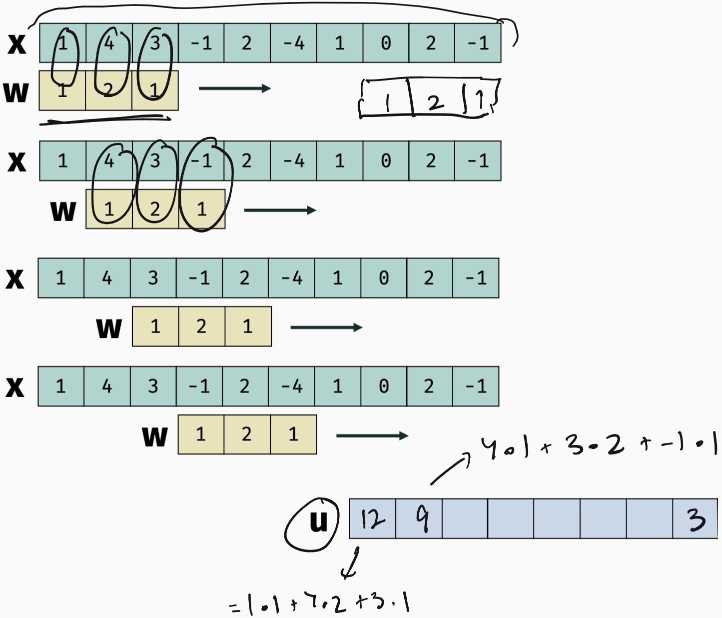
Given $\vec{x} \in \mathbb{R}^d$ and $\vec{w} \in \mathbb{R}^k$ the discrete convolution $\vec{x} \circledast \vec{w}$ is a $d - k + 1$ vector with:

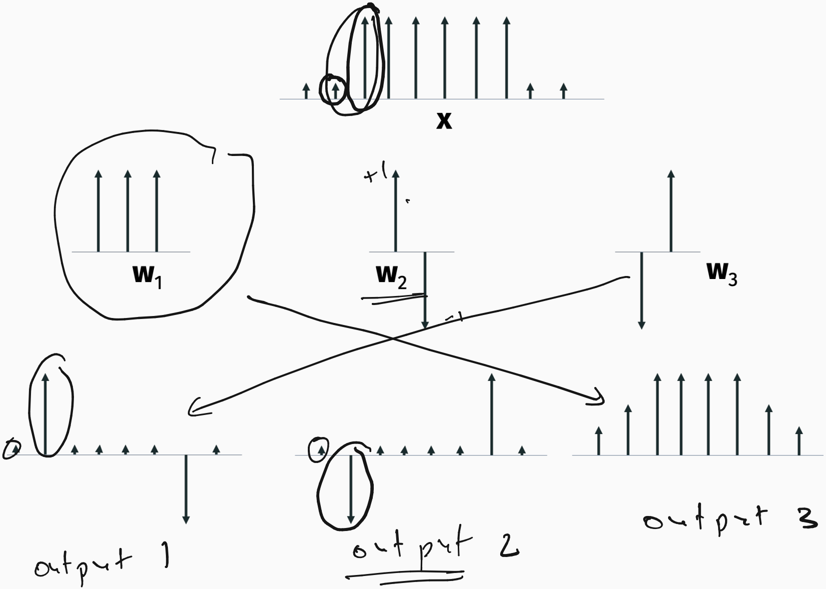
$$[\vec{x} \circledast \vec{w}]_i = \sum_{j=1}^{k} \vec{x}_{(j+i-1)} \vec{w}_j$$

Think of $\vec{x} \in \mathbb{R}^d$ as long **data vector** (e.g. d = 512) and $\vec{w} \in \mathbb{R}^k$ as short **filter vector** (e.g. k = 8). $\vec{u} = [\vec{x} \circledast \vec{w}]$ is a feature transformation.

[1] This is slightly different from the definition of convolution you might have seen in a Digital Signal Processing class because $\vec{w}$ does not get “flipped”. In signal processing our operation would be called correlation.

1D CONVOLUTION




x
w1
+1
w2
-1
w3
0
0
output 1
output 2
output 3

Definition (Discrete 2D convolution)

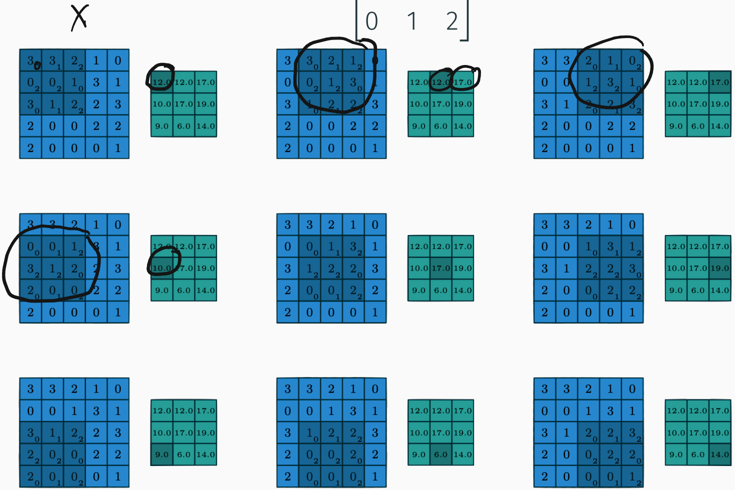
Given matrices $\mathbf{x} \in \mathbb{R}^{d_1 \times d_2}$ and $\mathbf{w} \in \mathbb{R}^{k_1 \times k_2}$ the discrete convolution $\mathbf{x} \circledast \mathbf{w}$ is a $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1)$ matrix with:

$$[\mathbf{x} \circledast \mathbf{w}]_{i,j} = \sum_{\ell=1}^{k_1} \sum_{h=1}^{k_2} \mathbf{x}_{(i+\ell-1),(j+h-1)} \cdot \mathbf{w}_{\ell,h}$$

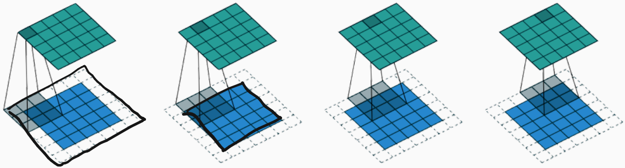
Again technically this is "correlation" not "convolution". Should be performed in Python using `scipy.signal.correlate2d` instead of `scipy.signal.convolve2d`.

$\mathbf{w}$ is called the <u>filter</u> or <u>convolution kernel</u> and again is typically much smaller than $\mathbf{x}$.

$$\mathbf{w} = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$



Sometimes "zero-padding" is introduced so $\mathbf{x} \circledast \mathbf{w}$ is $d_1 \times d_2$ if $\mathbf{x}$ is $d_1 \times d_2$.



Need to pad on left and right by $(k_1 - 1)/2$ and on top and bottom by $(k_2 - 1)/2$.

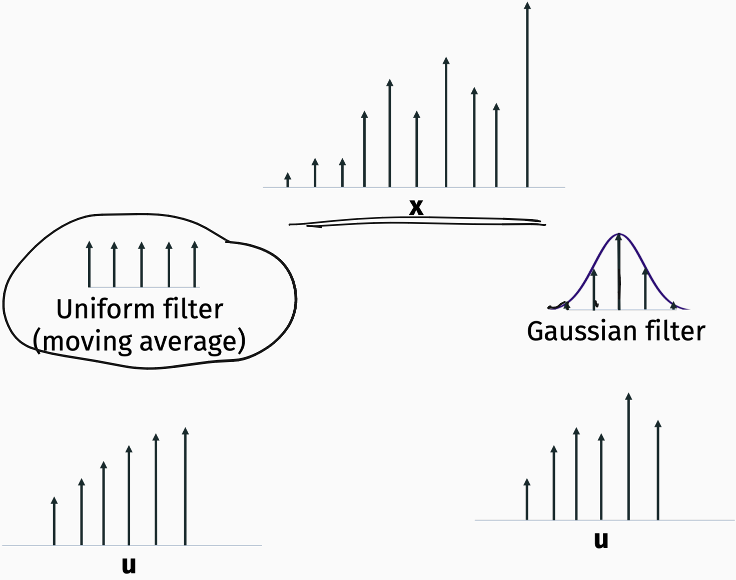
Examples code will be available in
`demo1_convolutions.ipynb`.

Application 1: Blurring/smooth. $w = [1/k \;\; 1/4 \;\; 1/4 \;\; 1/4]$

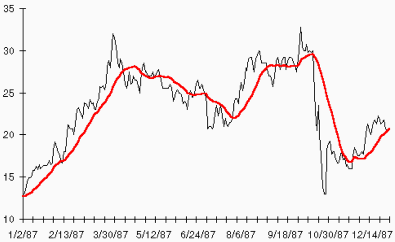
In one dimension:

- Uniform (moving average) filter: $\vec{w}_i = \frac{1}{k}$ for $i = 1, \ldots, k$.
- Gaussian filter: $\vec{w}_i \sim \exp^{(i-k/2)^2/\sigma^2}$ for $i = 1, \ldots, k$.

SMOOTHING FILTERS

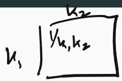


Useful for smoothing time-series data, or removing noise/static from audio data.



Replaces every data point with a local average.

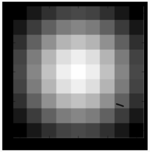
In two dimensions:



- Uniform filter: $\mathrm{w}_{i,j} = \frac{1}{k_1 k_2}$ for $i = 1, \ldots, k_1$, $j = 1, \ldots, k_2$.
- Gaussian filter: $\vec{w}_i \sim \exp^{\frac{(i-k_1/2)^2+(j-k_2/2)^2}{\sigma^2}}$ for $i = 1, \ldots, k_1$, $j = 1, \ldots, k_2$.

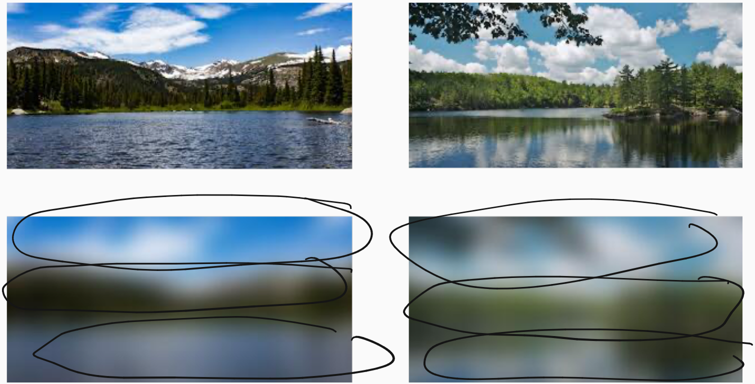
Larger filter equates to more smoothing.

For Gaussian filter, you typically choose $k \gtrsim 2\sigma$ to capture the fall-off of the Gaussian.



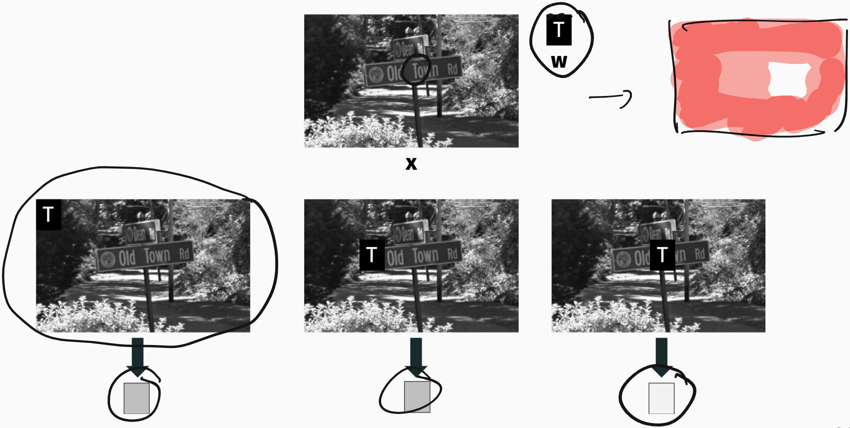
Both approaches effectively denoise and smooth images.

When combined with other feature extractors, smoothing at various levels allows the algorithm to focus on high-level features over low-level features.



Application 2: Pattern matching.

Slide a pattern over an image. Output of convolution will be higher when pattern correlates well with underlying image.



Applications of local pattern matching:

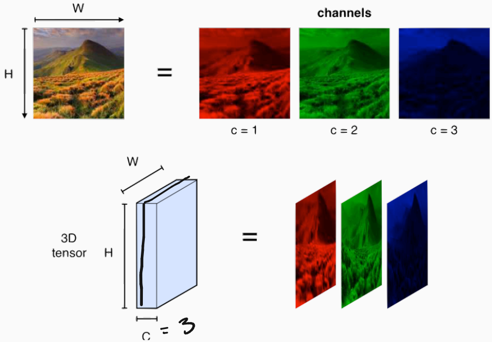
- Check if an image contains text.
- Look for specific sound in audio recording.
- Check for other well-structured objects

3D CONVOLUTION

Recall that color images actually have three color channels for **red**, **green**, **blue**s. Each pixel is represented by 3 values (e.g. in $0, \ldots, 255$) giving the intensity in each channel.

$[0, 0, 0]$ = black, $[0, 0, 0]$ = white, $[1, 0, 0]$ = pure red, etc.

View image as 3D tensor:



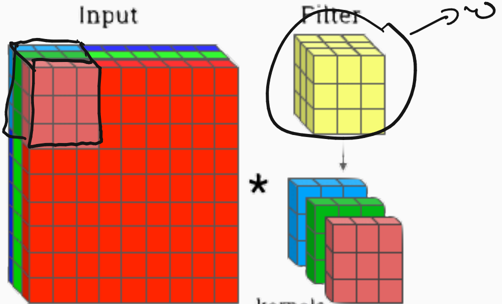
3D convolution

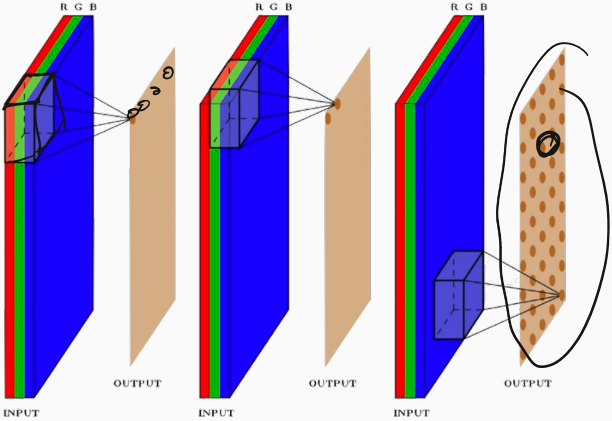
Can be convolved with 3D filter:

Definition (Discrete 3D convolution)

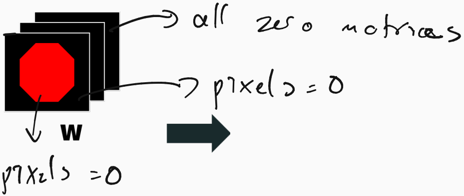
Given tensors $\mathbf{x} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ and $\mathbf{w} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ the discrete convolution $\mathbf{x} \circledast \mathbf{w}$ is a $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1) \times (d_3 - k_3 + 1)$ tensor with:

$$[\mathbf{x} \circledast \mathbf{w}]_{i,j,g} = \sum_{\ell=1}^{k_1} \sum_{m=1}^{k_2} \sum_{n=1}^{k_3} \mathbf{x}_{(i+\ell-1),(j+m-1),(g+n-1)} \cdot \mathbf{w}_{\ell,m,n}$$




R G B
OUTPUT
INPUT
R G B
OUTPUT
INPUT
R G B
OUTPUT
INPUT

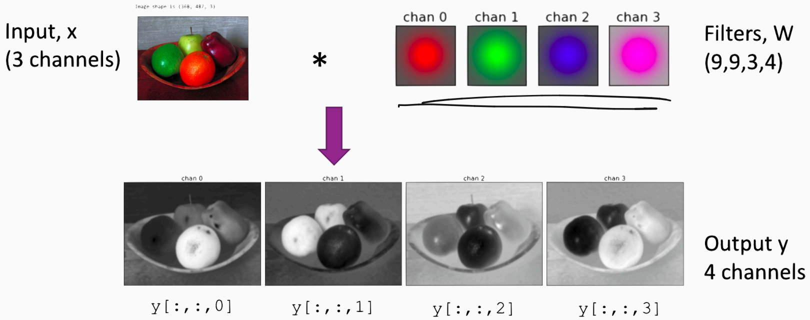
x



W

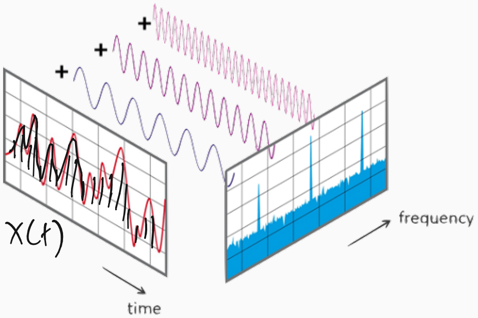
Relatively robust to imperfections, damage, occlusion, etc.

In general extracting different color information is very useful for image understanding:



Less obvious example of pattern matching: Frequency detection in audio.

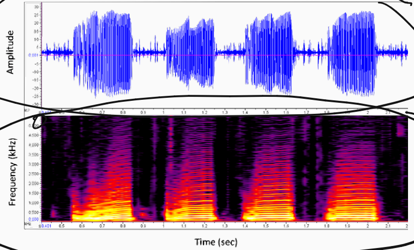
Any 1D signal (including a sound wave) can be decomposed into component frequencies:



$$\vec{x}(t) = \sin(f_1 t + s_1) + \sin(f_2 t + s_2) + \sin(f_3 t + s_3) + \ldots$$

Convolve audio signal with snippet of pure frequency to determine where difference frequencies are prevalent. Detect things like:

- Common notes in a song.
- Different instruments.
- Human voices vs. other noise.

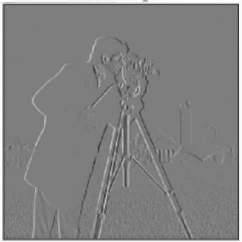


Main idea behind short-time Fourier transforms/spectrograms.

Application 3: Edge detection.

Consider a 2d Sobel filter:

$$W_1 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad W_2 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



x * ?

x * ?

Edges with different orientations are low-level features compared to what we might get from e.g. explicit pattern matching. They still provide some immediately useful information about an image.

More useful when combined to build higher-level features.

Next class: Design neural network architectures that have convolutionoal operations <u>built-in</u>. The exact weights are left as free variables. Lowest layers do low-level feature extraction like edge detection, higher layers learn higher-level concepts.