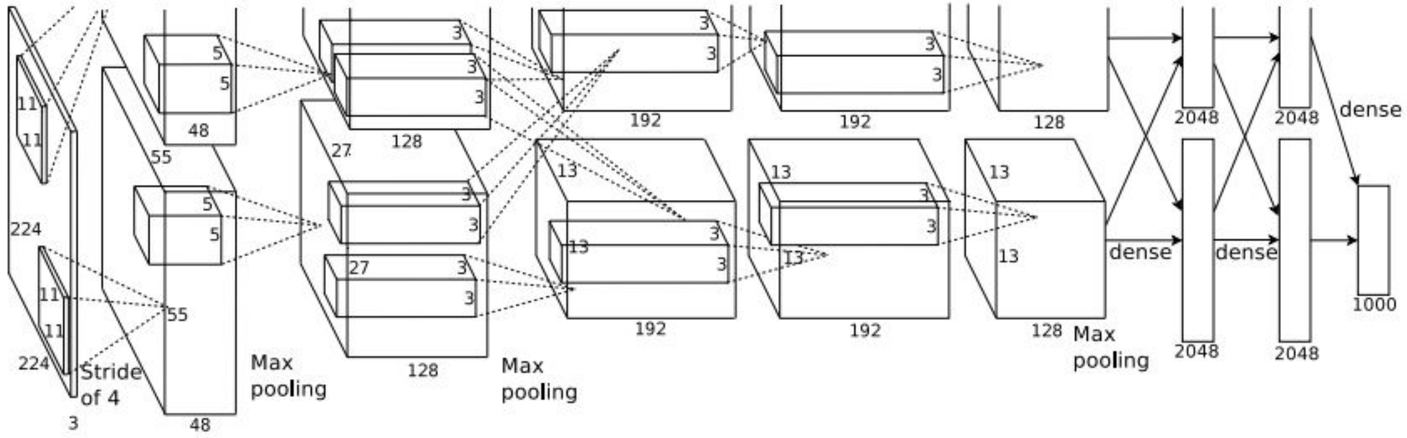
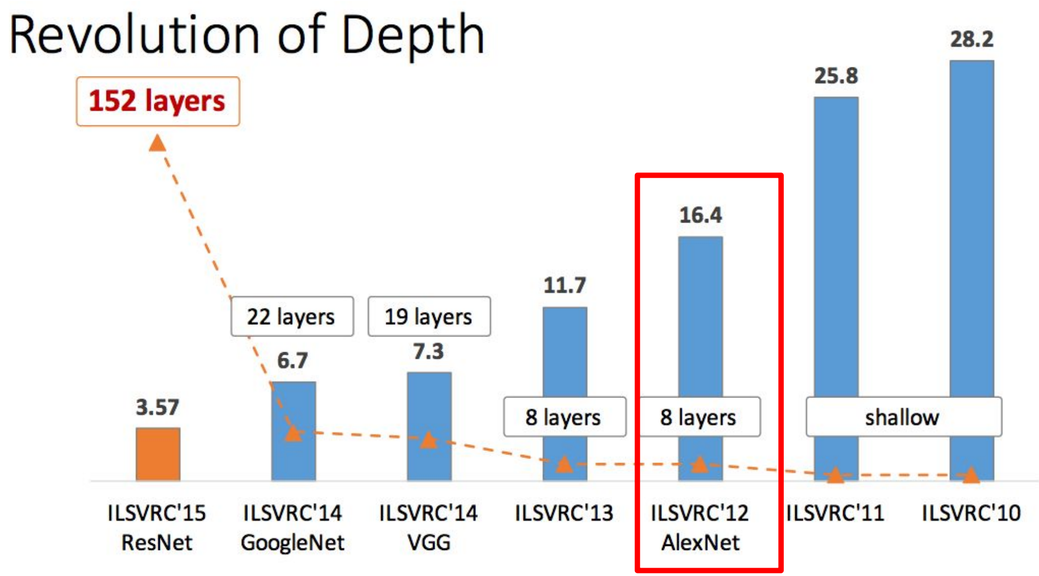
Last time..

CNN artictecturhes

Last Time: CNN Architectures

AlexNet



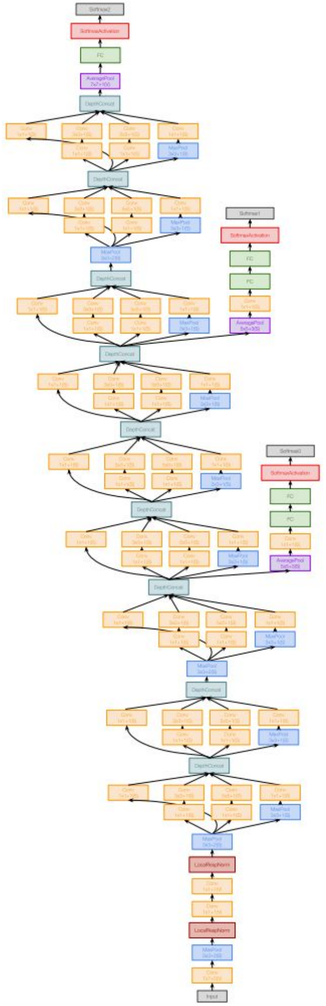


- Last time: CNN **architectures**, time-line: **ImageNet** challenge winners
- Breakthrough: AlexNet in **2012**, 8 layer, kick-started the DL **revolution** in CV, models into mainstream

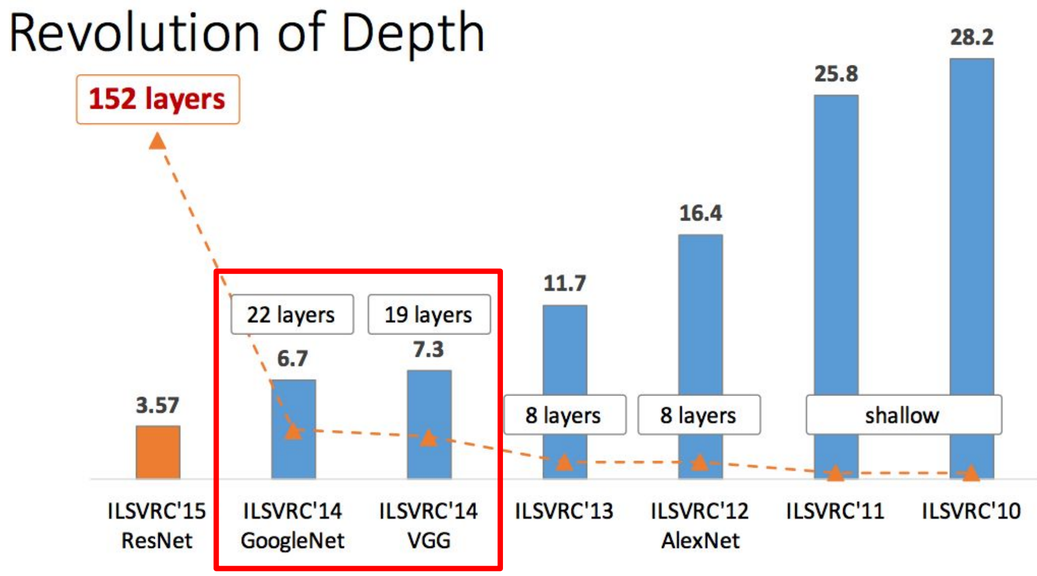
Last Time: CNN Architectures

VGG16

VGG19

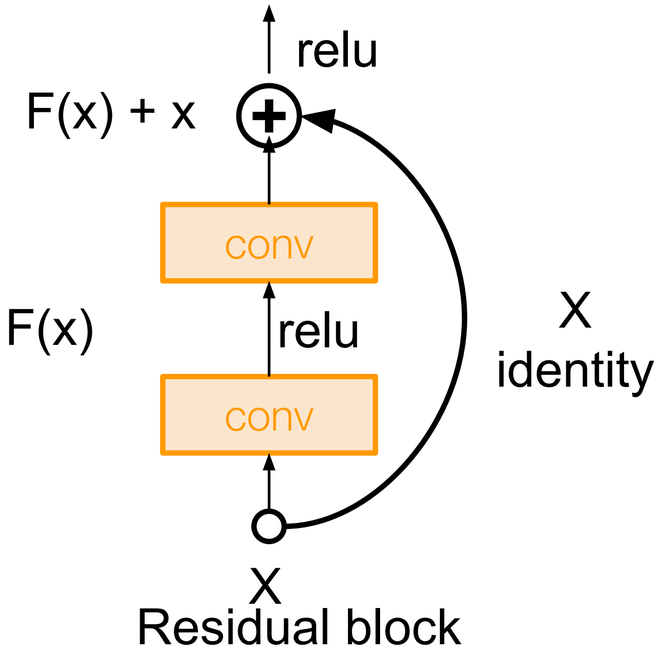


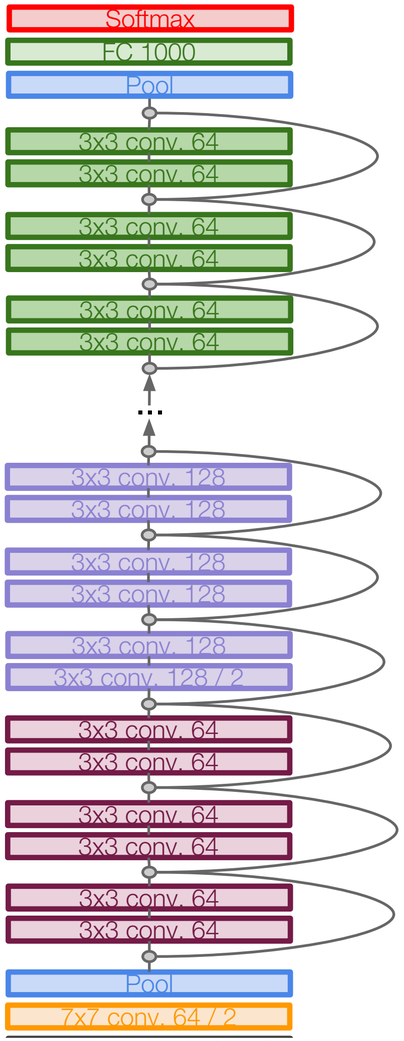
GoogLeNet



- Then we go to **2014**: two interesting models: VGG and GoogLeNet, **deeper**, 16/19 and 22 layers respectively
- Challenge right before **batch normalization (BN)**, training deep models (20) was very difficult, hacks needed to convert, VGG: 11 layer first, GoogLeNet: auxiliary classifiers (gradients directly injected into lower layers), with BN: hacks not needed to get the models to converge

Last Time: CNN Architectures





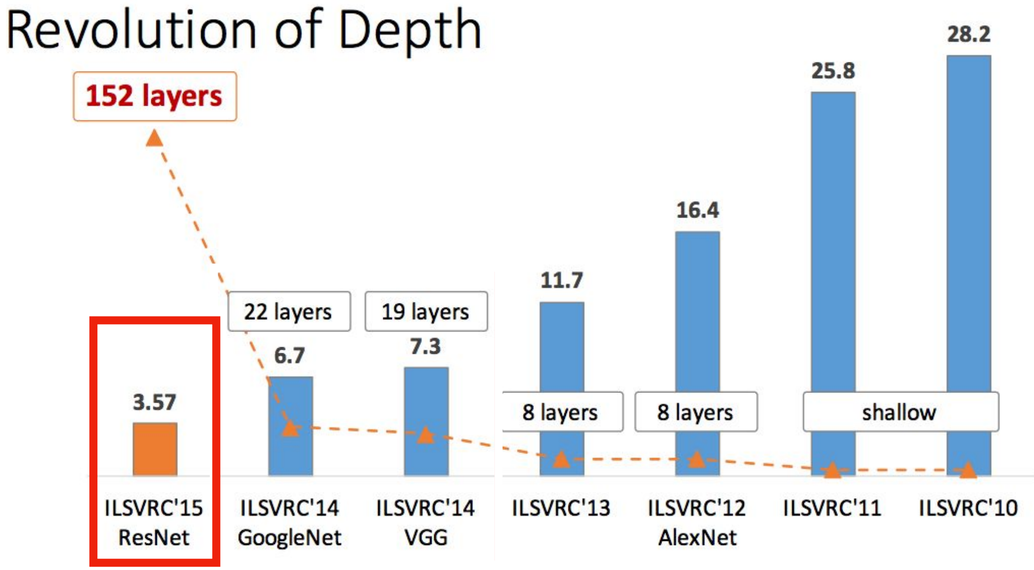
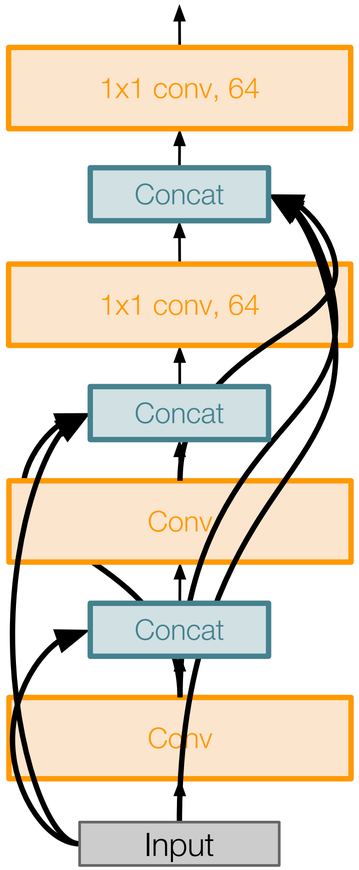


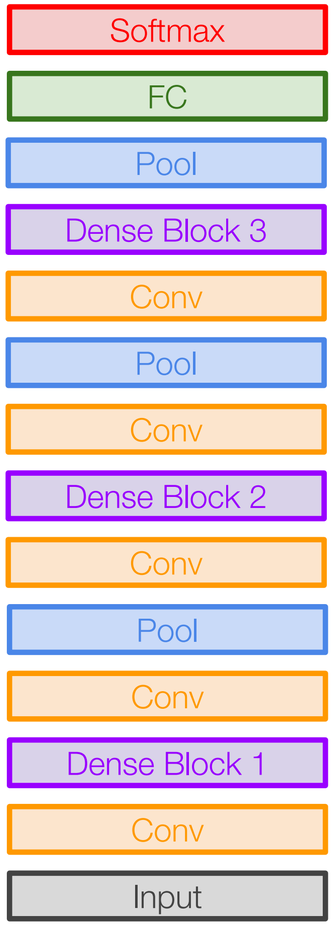
Figure copyright Kaiming He, 2016. Reproduced with permission.

- In **2015**: **ResNet**: residual **networks** with residual **blocks** and shortcut **connections**, take input - pass trough residual block - **add** output of the block to the input of the block (strange), two nice **properties**:
  - If all **weights** in block = 0: block computes the **identity** (learn to not use the layers not needed) + L2 **regularization**: drive w to 0 make sens
  - **Gradient flow** in the backward pass, addition gates: split and fork - two paths, direct (skip) connection, stacking: gradient super highway to flow

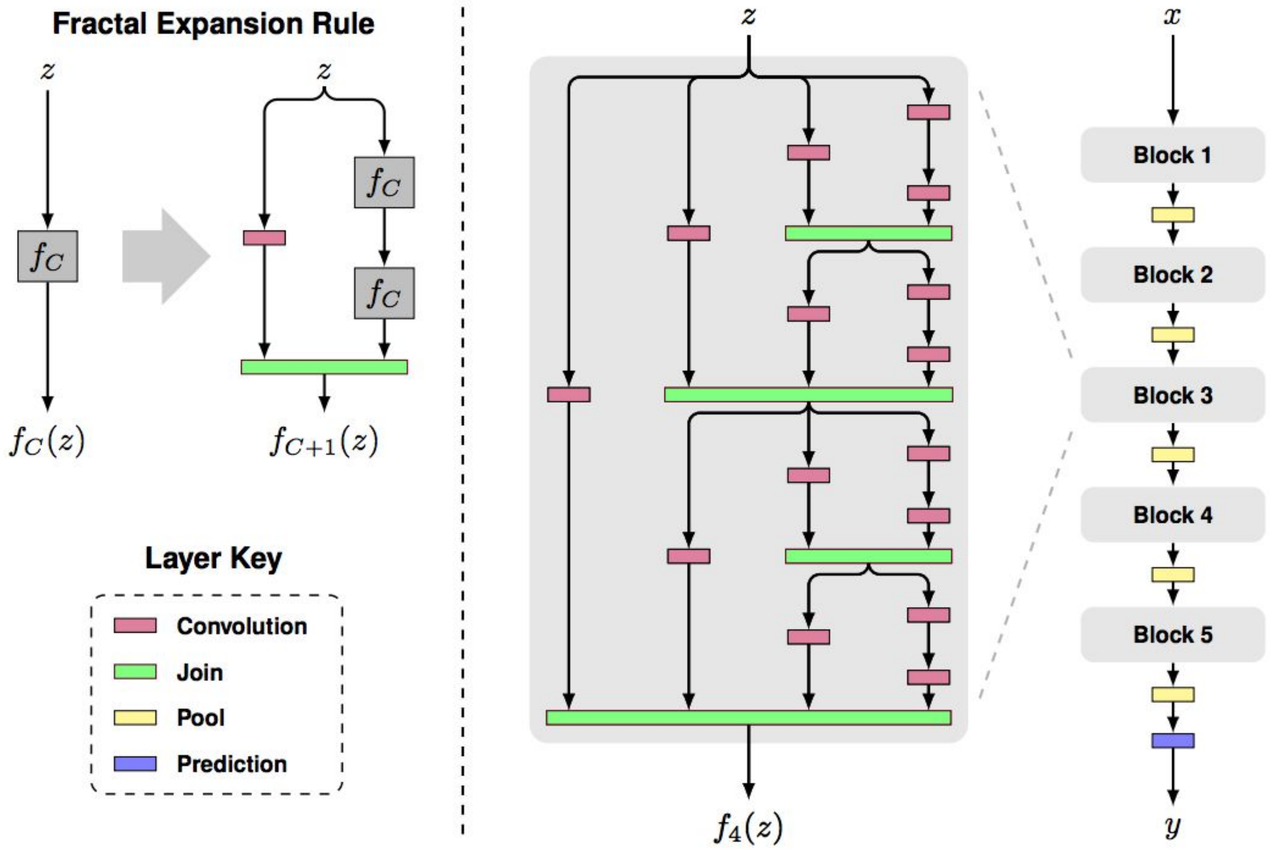
DenseNet



Dense Block



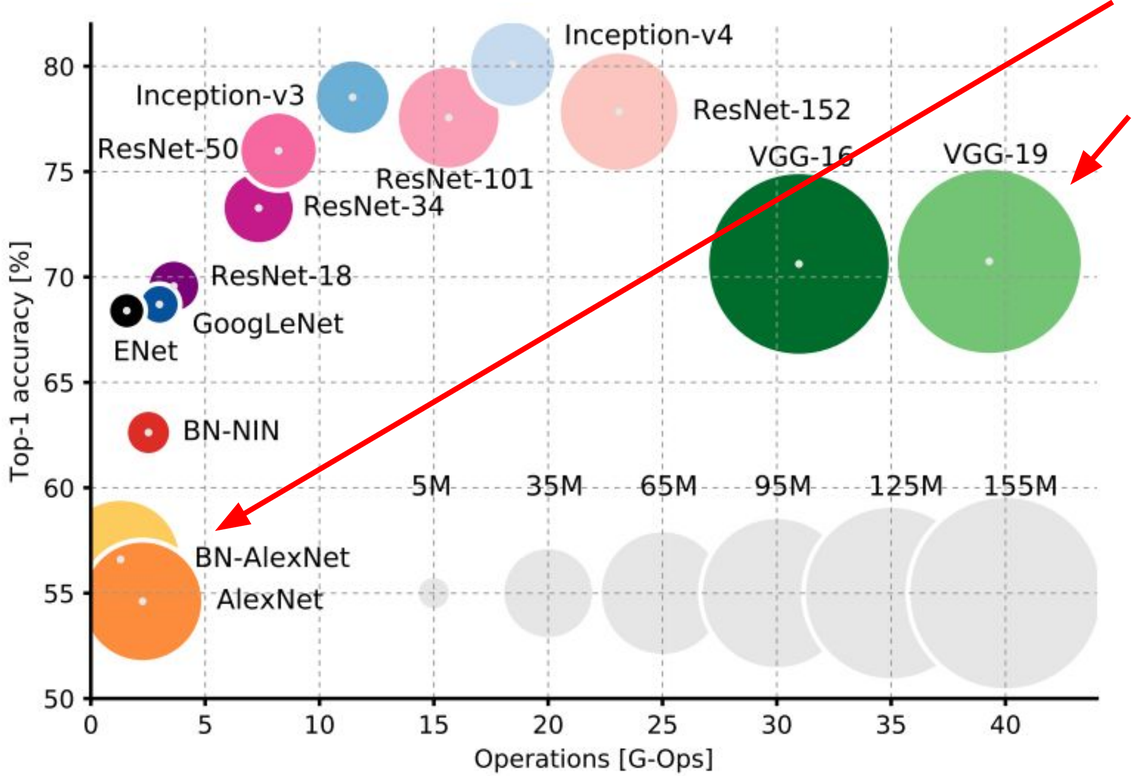
FractalNet



Figures copyright Larsson et al., 2017. Reproduced with permission.

- Idea of **managing gradient** flow in models is very important, also for RNN
- Recent and exotic architectures: make **sens** in terms of **gradient flow** (backward pass), important concept in recent and future models

Last Time: CNN Architectures



AlexNet and VGG have tons of parameters in the fully connected layers

AlexNet: ~62M parameters

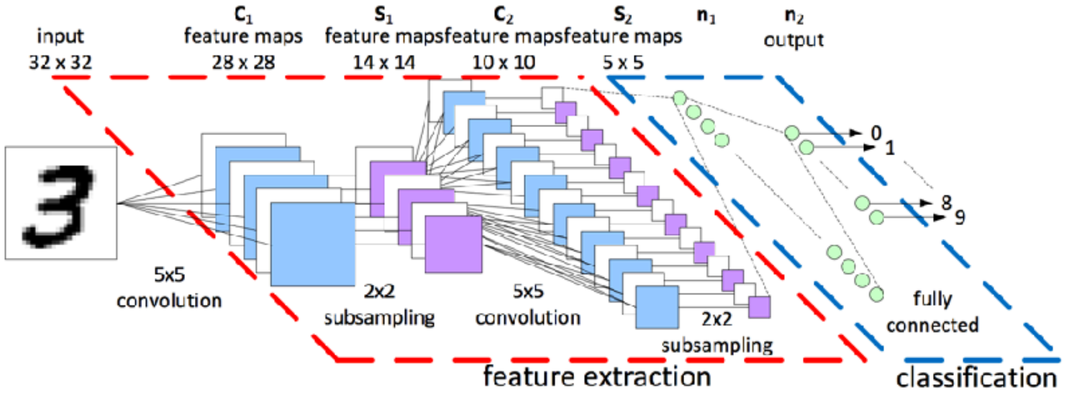
FC6: 256x6x6 -> 4096: 38M params
FC7: 4096 -> 4096: 17M params
FC8: 4096 -> 1000: 4M params
~59M params in FC layers!

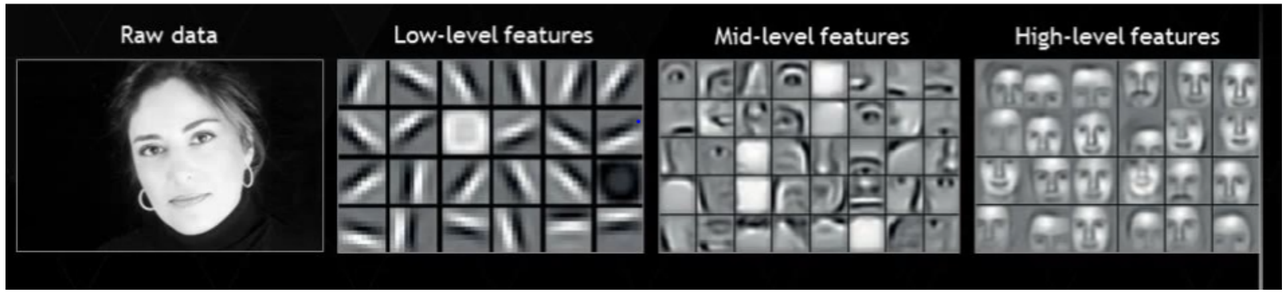
- Nice plot: # operations vs. # parameters vs. accuracy
  - AlexNet and VGG: huge number of **parameters** (**FC** layers), e.g. AlexNet: 62M pram. - FC6: 38M param, all FC: 59M pram.
  - Resent models: fewer param. but bette accuracy.

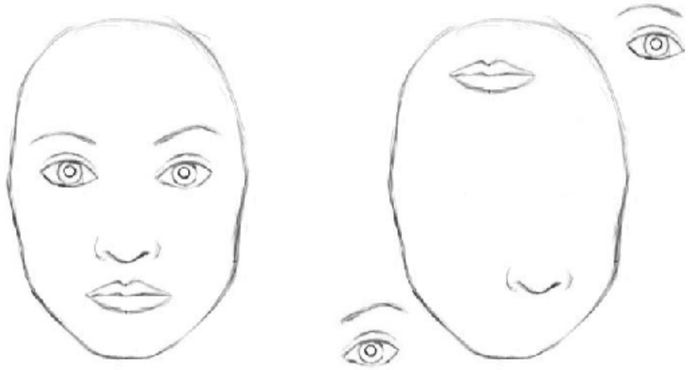
CapsNet

CapsNet: Why?

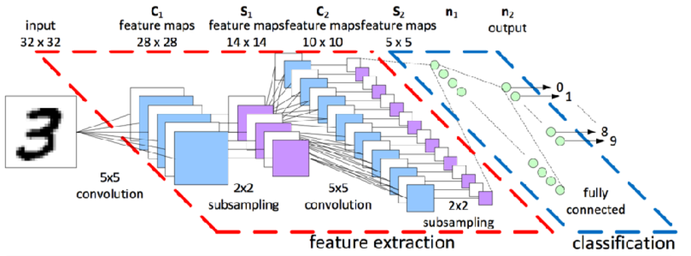
- **CNNs**: amazing things, DL revolution, from not possible to superhuman performance
  - **How**: convolutional layer: learn & detect important **features**, first layers will detect **simple** features such as edges and colors, higher layers will combine simple features into more **complex** features, **dense** layers at end **combine** very high level features and produce **predictions**.
  - **Limitation**: higher-level features **combine** lower-level features as a **weighted sum**, no **pose** relationship between simpler features that make up higher level feature. **Max-pooling**: reduce spacial size and increase the “field of view” to detect higher order features in a larger region, work surprisingly well, but **looses** valuable information (Hinton: disaster).
  - **Key problem:** does not take into account important spatial hierarchies between simple and complex objects.
    Example: just the **presence** of 2 eyes, a mouth and a nose in a picture does not mean there is a face (we also need to know how these objects are oriented relative to each other)

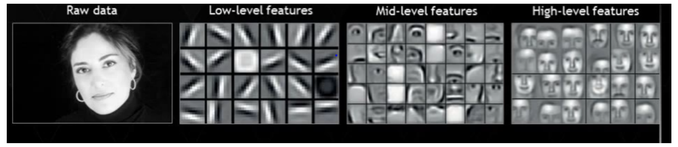




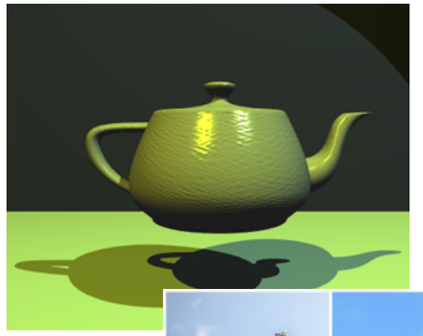


CapsNet: Why?

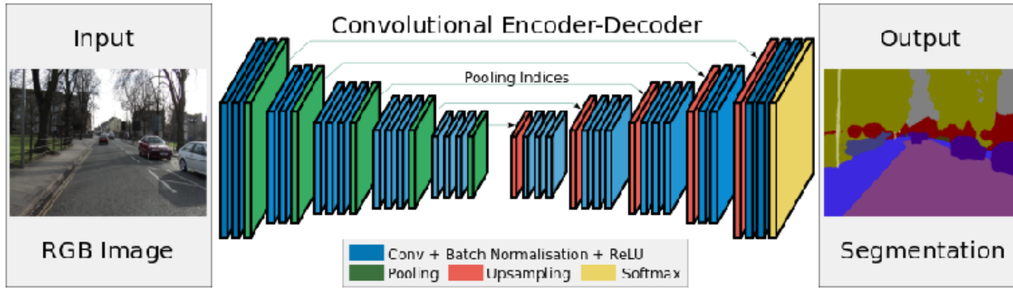




- **Brain:** Inverse rendering / graphics: from visual information (eyes) to hierarchical representation of the world + match it with already learned patterns and relationships stored in the brain (recognition).
- **Key idea**: representation of objects in the brain does not depend on view angle. **Example**: You can easily recognize the Statue of Liberty, even though all the images show it from different angles & you have probably never seen these exact pictures of it before.
- **CapsNet**:
  - **Key intuition**: in order to correctly do classification and object recognition, it is important to preserve *hierarchical pose relationships* between object parts
  - **Viewpoint invariance** (difficult for CNNs): cut error rate by 45% relative to previous SotA.
  - **Less data**: need only a fraction of the data that a CNN would use to achieve SotA performance.
  - **Algorithm** (+GPUs): «dynamic routing between capsules», create representations similar to scene graphs.
  - **Challenges**: speed, difficult data sets, different domains.

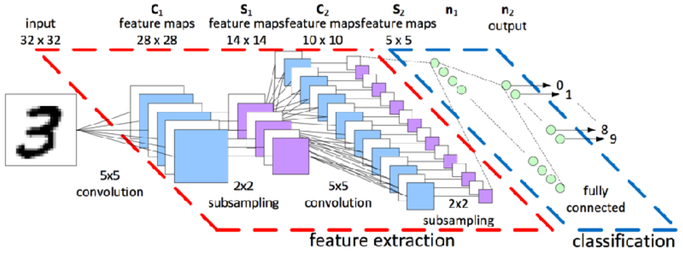


CapsNets: Summary



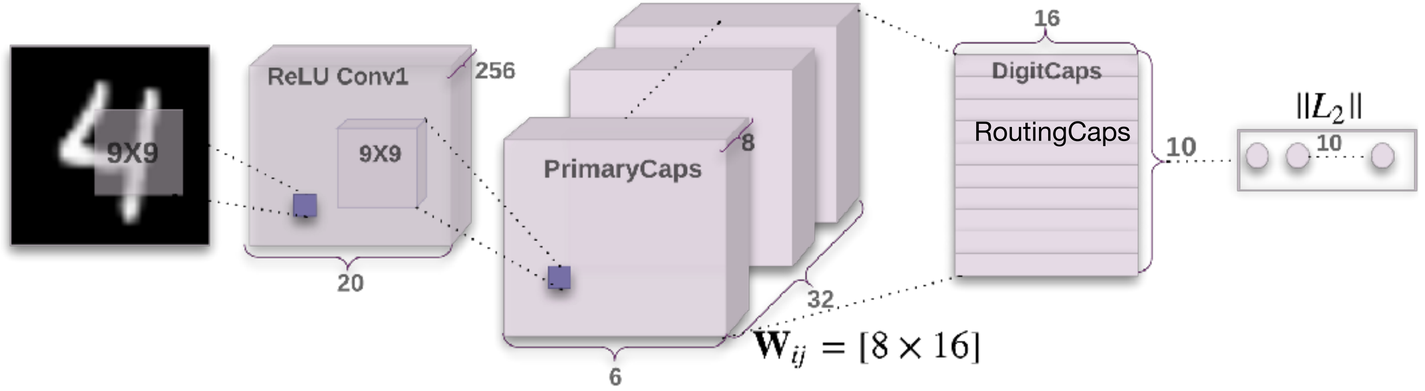
- CNNs **lose** plenty of information in the pooling layers, which reduce the spatial resolution, so the outputs are invariant to small changes in the inputs
    - **Problem** when detailed information must be preserved throughout the network, such as in semantic segmentation. Today, this issue is addressed by building **complex** architectures around CNNs to recover some of the lost information.
        - With **CapsNets**, detailed pose information (such as precise object position, rotation, thickness, skew, size, and so on) is preserved throughout the network, rather than lost and later recovered. Small changes to the inputs result in small changes to the outputs - information is preserved. This is called "equivariance." As a result, CapsNets can use the same simple and consistent architecture across different vision tasks.
            - CNNs require extra components to automatically identify which object a part belongs to (e.g., this leg belongs to this sheep). CapsNets give you the hierarchy of parts for free.
            - CNNs are trained on huge numbers of images (or they reused parts of neural nets that have). CapsNets can generalize well using much less training data.
            - CNNs don’t handle **ambiguity** very well. CapsNets do, so they can perform well even on **crowded** scenes
- **Cons**: Slow compared to CNNs and performance on many real-world tasks are not as good yet.
    - Key ideas are extremely **promising**, just need a few **tweaks** (CNN: 1998 vs. 2012, CapsNet: 2011 (2017) vs. ?)

CapsNet: Architecture

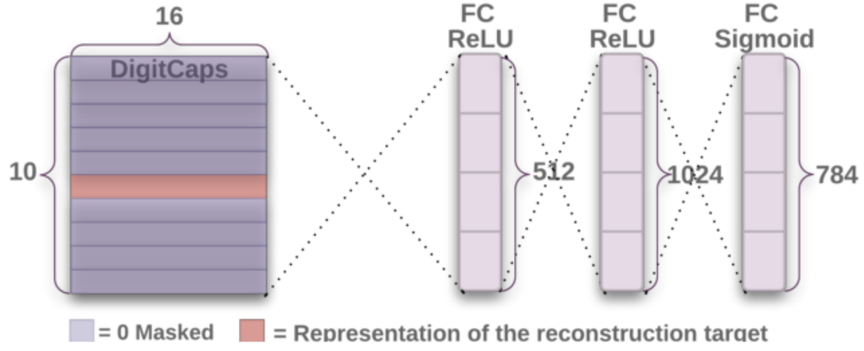


- Architecture
  - Encoder
    - 10 16-element vectors (length and angle)
  - Decoder
    - Reconstruct image (784=28x28)

Encoder

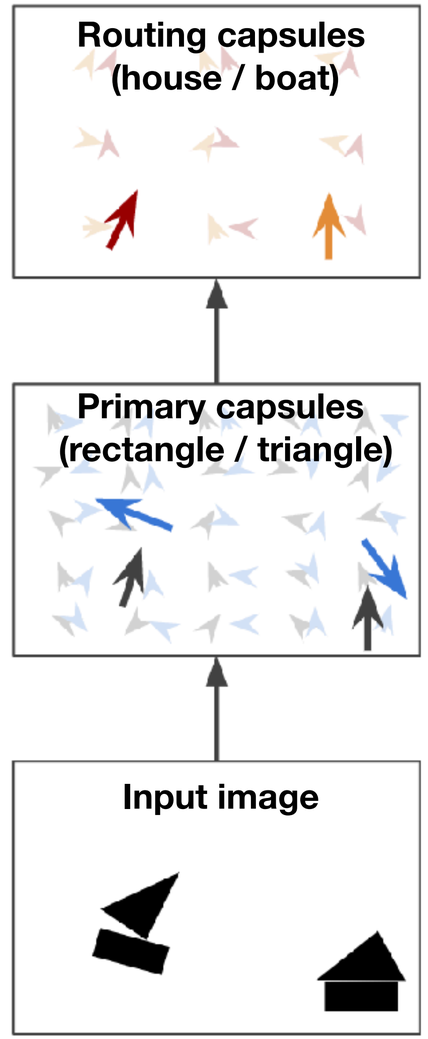


Decoder



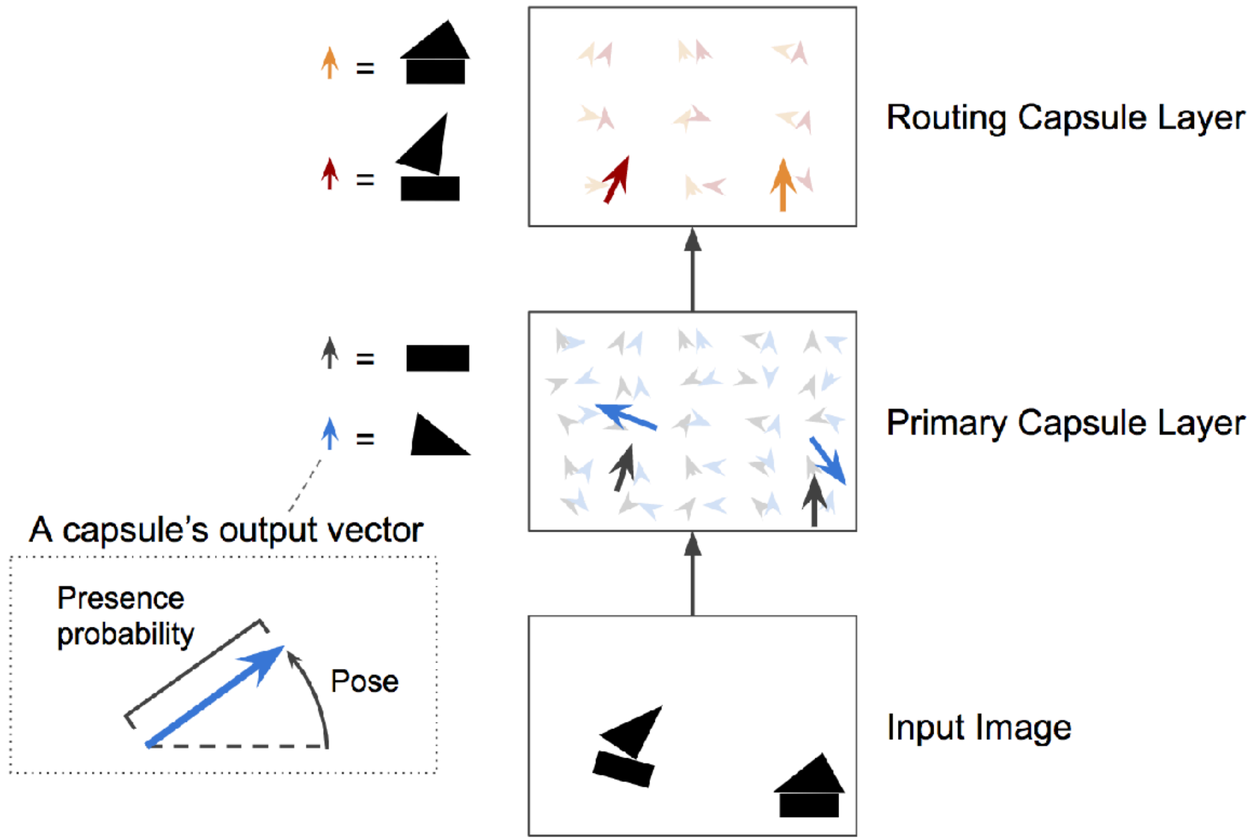
CapsNet: Overview

- Capsules instead of neurons (vectors instead of scalars, more powerful representational capabilities)
- A capsule is a small group of neurons that **learns to detect** a particular pattern or object (e.g., a rectangle)
  - within a given **region** (receptive field) of the image
    - and it **outputs a vector** (e.g., an 8-dimensional vector) whose **length** represents the estimated probability that the object is **present**, and whose **orientation** (e.g., in 8D space) encodes the object's **pose** parameters (e.g., precise position, rotation, etc.)
- If the object is **changed slightly** (e.g., shifted, rotated, resized, etc.) then the capsule will output a vector of the same length, but oriented slightly differently. Thus, capsules are equivariant.
- CapsNets are organized in multiple layers.
  - The capsules in the **lowest** layer are called **primary** capsules: each of them receives a small region of the image as input (called its receptive field), and it tries to detect the **presence and pose** of a particular pattern, for example a **rectangle**.
  - Capsules in **higher** layers, called **routing** capsules, detect larger and more complex objects, such as **boats**.

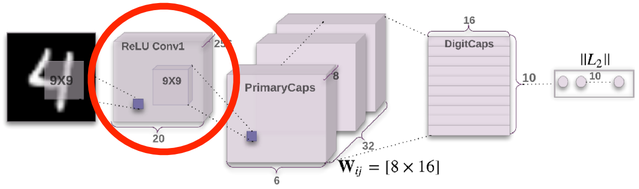


CapsNet: Layers

- **Primary** capsule layer:
  - a few regular **convolutional** layers, e.g. **two** convolutional layers that output 256 6x6 features maps containing scalars, **reshape** this output into 32 6x6 maps containing 8-dimensional vectors, **squashing** function to ensure these vectors have a length between 0 and 1 (to represent a probability)
- **Routing** capsule layer:
  - also try to detect objects and their pose (e.g. boat or house), but they work very differently, using an algorithm called **routing by agreement**.



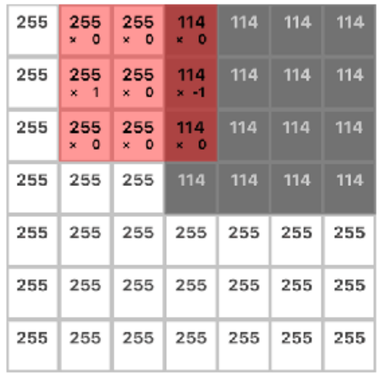
CapsNet: Conv

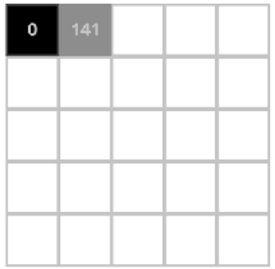


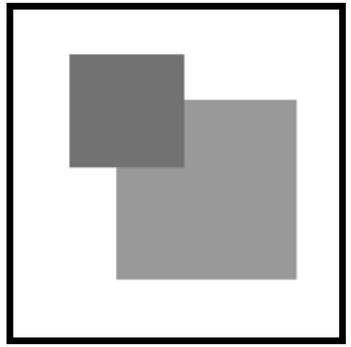
- Convolution + ReLU
  - Convolution
    - Sliding filters / kernels
  - ReLU

```
kernel = [
  [0  0  0]
  [1  0 -1]
  [0  0  0]
]
```

Kernel: 3x3

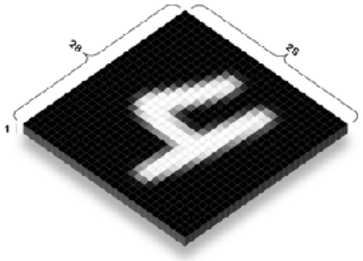




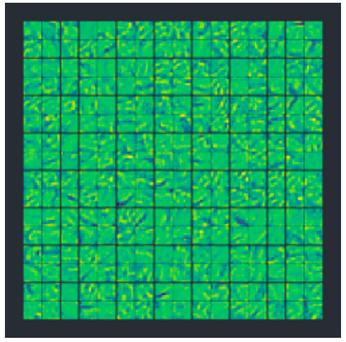


Input

Output



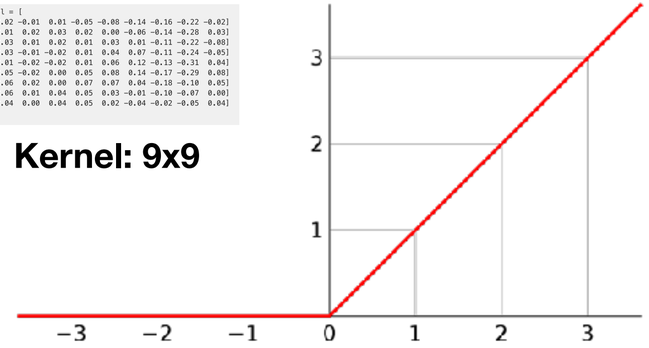
Input: 28x28x1



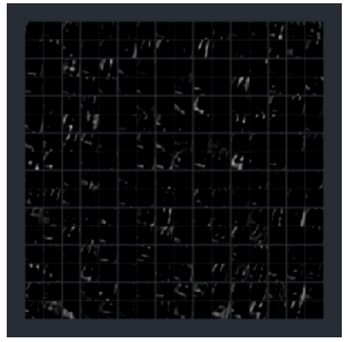
Kernels: 9x9x256

```
kernel = [
  [ 0.02 -0.01  0.01 -0.05 -0.08 -0.14 -0.16 -0.22 -0.02]
  [ 0.01  0.02  0.03  0.02  0.00 -0.06 -0.14 -0.28  0.03]
  [ 0.03  0.01  0.02  0.01  0.03  0.01 -0.11 -0.22 -0.08]
  [ 0.03 -0.01 -0.02  0.01  0.04  0.07 -0.11 -0.24 -0.05]
  [-0.01 -0.02 -0.02  0.01  0.06  0.12 -0.13 -0.31  0.04]
  [-0.05 -0.02  0.00  0.05  0.08  0.14 -0.17 -0.29  0.08]
  [-0.06  0.02  0.00  0.07  0.07  0.04 -0.18 -0.10  0.05]
  [-0.06  0.01  0.04  0.05  0.03 -0.01 -0.10 -0.07  0.00]
  [-0.04  0.00  0.04  0.05  0.02 -0.04 -0.02 -0.05  0.04]
]
```

Kernel: 9x9

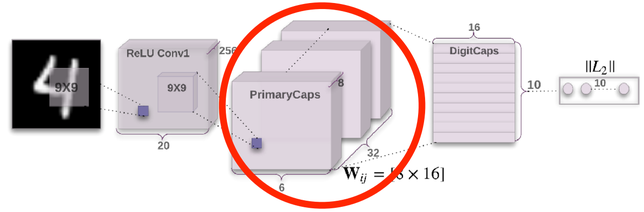


ReLU(x) = max(0,x)

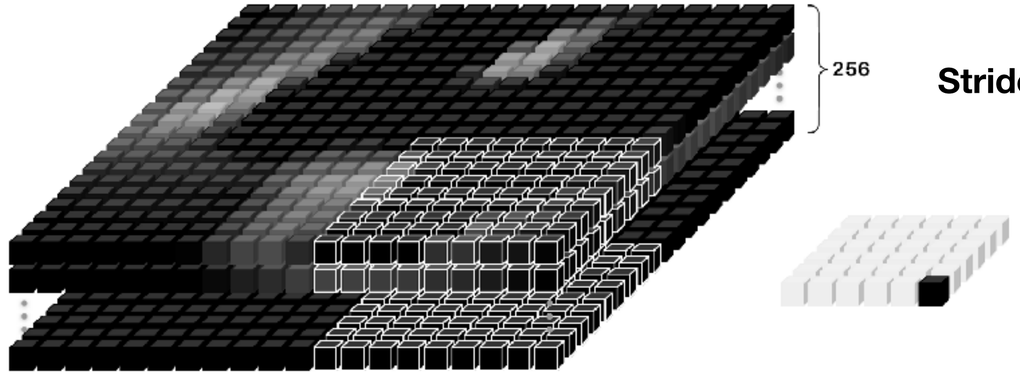


Output: 20x20x256

CapsNet: PrimaryCaps

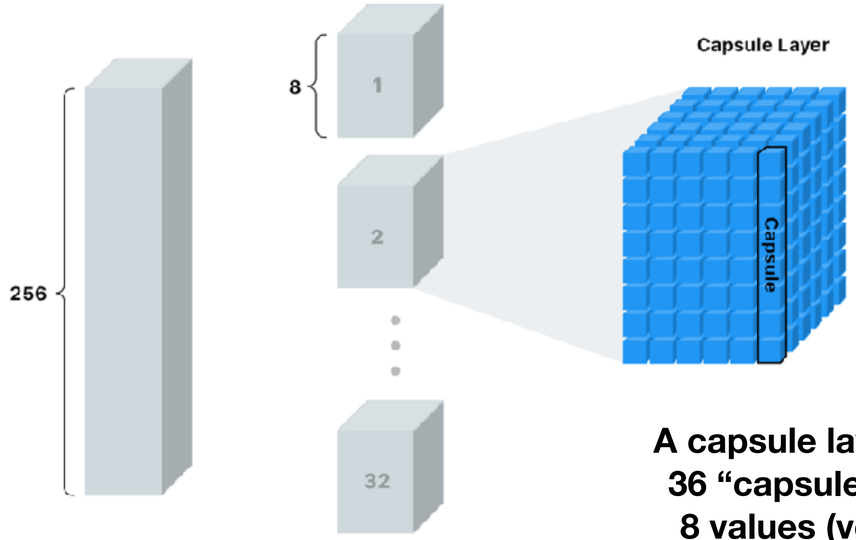


- **PrimaryCaps**
  - Normal **convolution** layer: from simple edges to slightly more complex shapes
  - **Divide** the 256 channels into 32 capsules (rectangle, triangle etc), 8 elements vector each (pose etc.)



Stride: 2

Input: 20x20x256 **Kernel: 9x9x256** **Output: 6x6x256**
i.e. 256 kernels

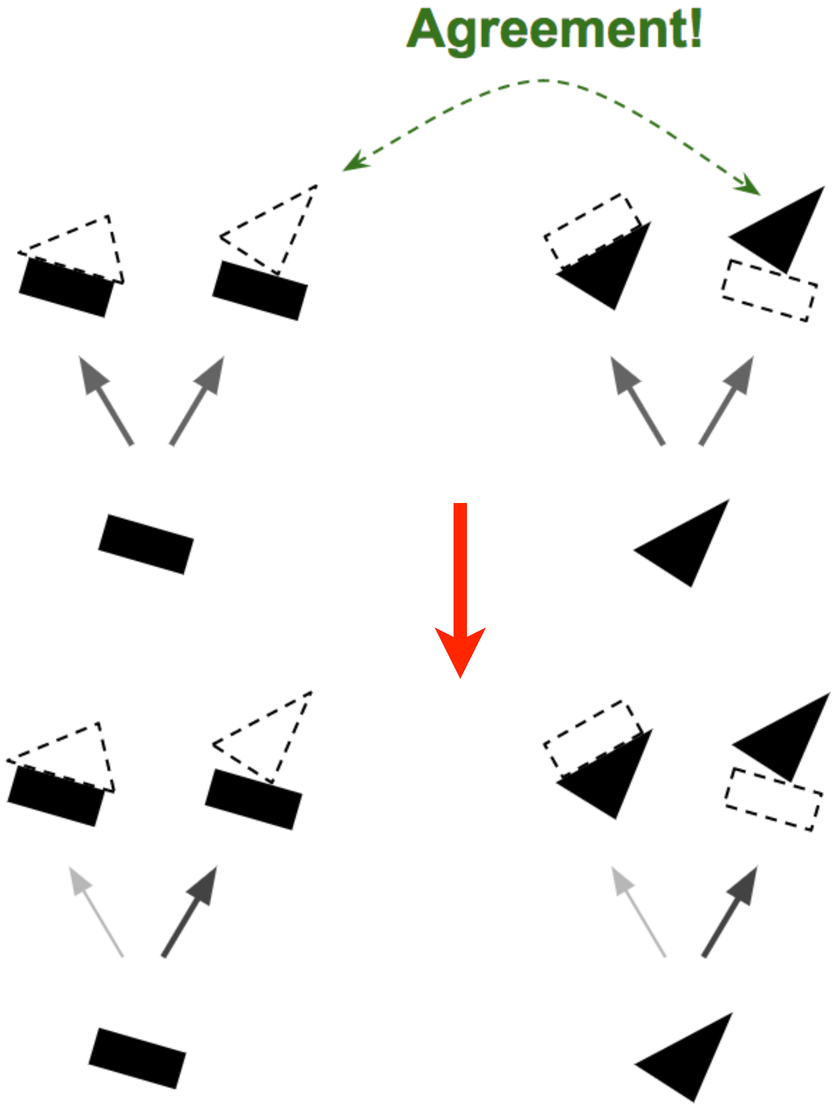


A capsule layer has 36 “capsules with 8 values (vector)

Dept of 256 =
32 capsule layers of 8 cards/values each

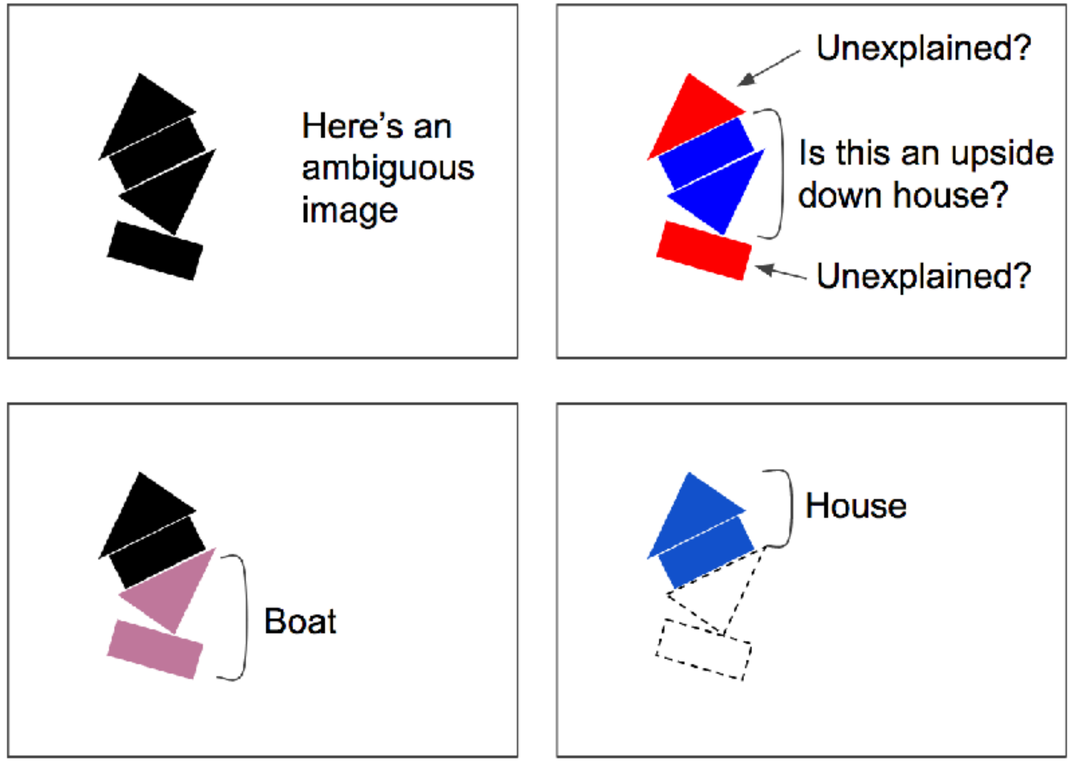
CapsNet: RoutingCaps

- **Routing capsules**:
  - Two **primary** capsules: one **rectangle** capsule and one **triangle** capsule, both the rectangle and the triangle could be **part of** either a **house** or a **boat**, **given** the pose of the **rectangle**, which is **slightly rotated** to the right, **the house and the boat** would have to be slightly rotated to the right as well, **given** the pose of the triangle, the **house** would have to be almost **upside** down, whereas the **boat** would be slightly rotated to the right, the rectangle and the triangle **agree** on the pose of the boat, while they strongly **disagree** on the pose of the house, it's very likely that the rectangle and triangle are part of the same boat, and there is no house in that position.
  - Routing by **agreement:**
    - step 1: a) predict the presence and pose of **objects** based on the presence and pose of **object parts**, then b) look for **agreement** between the predictions
    - step 2: b) **update** the routing **weights**



CapsNet: Ambiguity

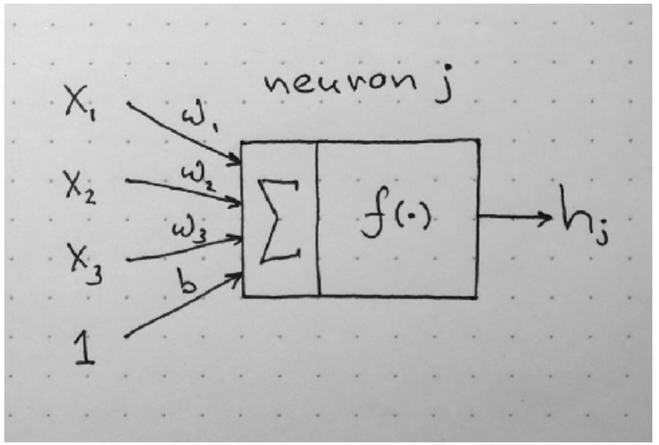
- Routing by agreement and crowded scenes:
  - Given an **ambiguous** image, which could be **misinterpreted** as an **upside**-down house plus some **unexplained** parts
  - But the lower rectangle will be **routed to** the boat, and this will also **pull the lower triangle** into the boat as well
  - Once that boat is “explained away,” it’s easy to interpret the top part as a house.

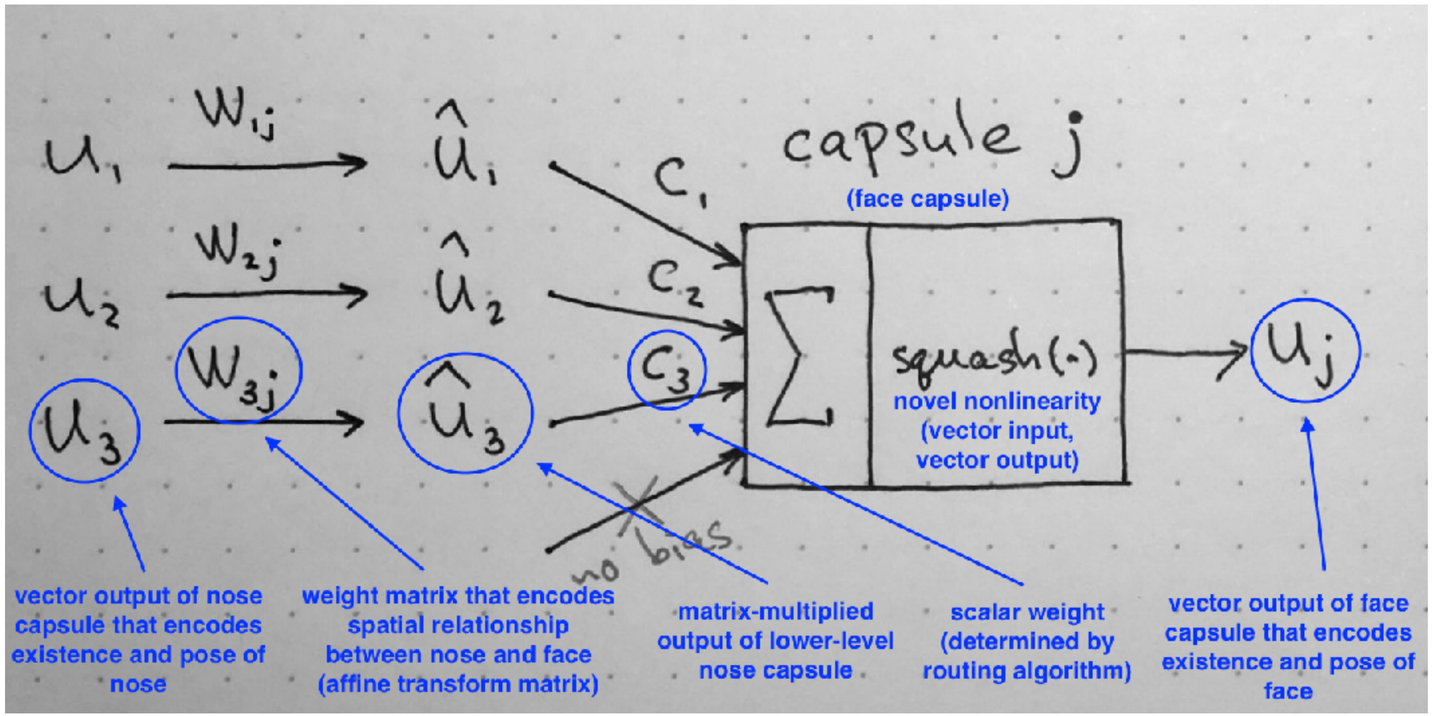


CapsNet: Some more detail

- **Neuron (scalar):**
  - scalar weighting of input scalars
  - sum of weighted input scalars
  - scalar-to-scalar nonlinearity
- **Capsule (vector):**
  - **matrix multiplication of input vectors**
  - scalar weighting of input vectors
  - sum of weighted input vectors
  - vector-to-vector nonlinearity

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	–
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$	$h_j = f(a_j)$
Output		vector($\mathbf{v}_j$)	scalar(h_j)





CapsNet: How?

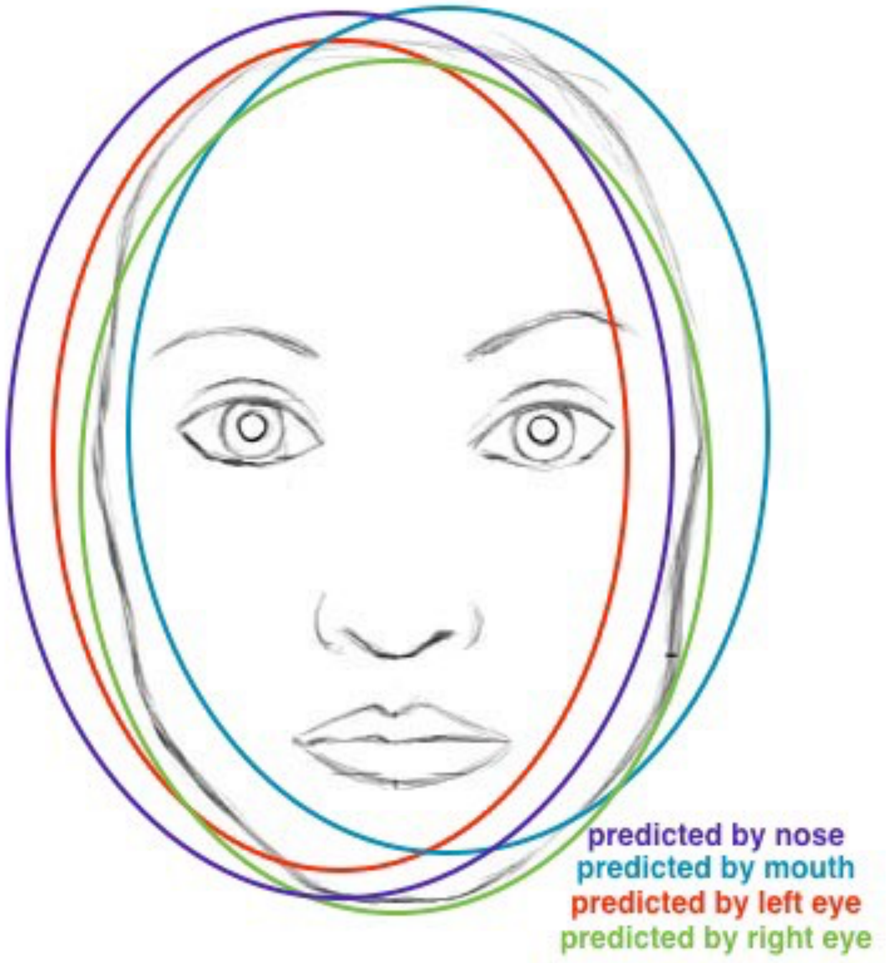
- **Matrix Multiplication of Input Vectors**

  - **Input** capsules: nose, mouth and eyes (**lower** level features). **Output** capsule: face (**higher** level feature)

    - **Weight matrices** encode important **spatial** and other relationships between lower level features (eyes, mouth and nose) and higher level feature (face), i.e. matrix weights encode important **hierarchical relationships** between features of different layers.

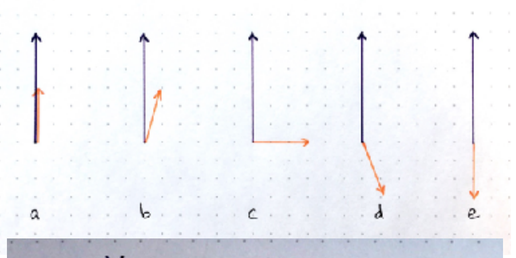
      - After multiplication we get the **predicted position of the higher level feature**, i.e. u1hat represents where the face should be according to the detected position of the eyes etc.

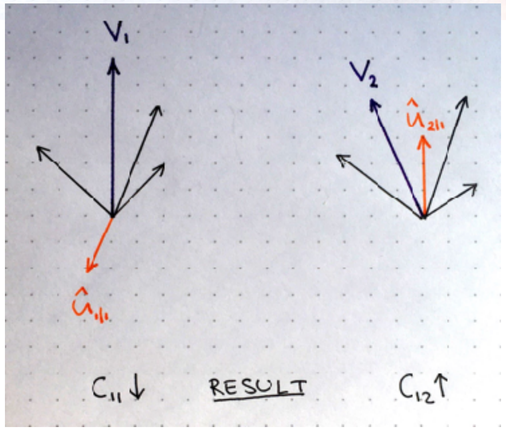
  - **Intuition**: if these 3 predictions of lower level features point at the same position and state of the face, then it must be a face there.

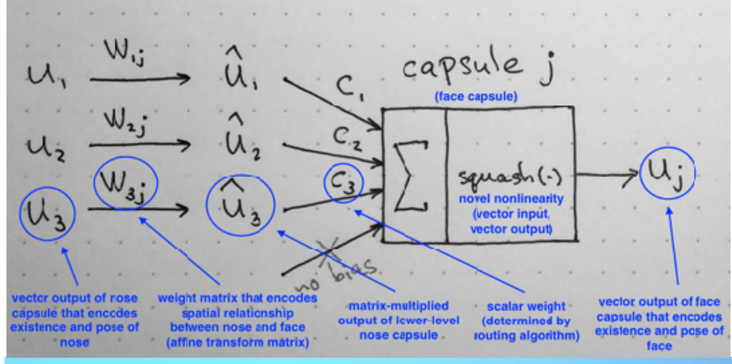


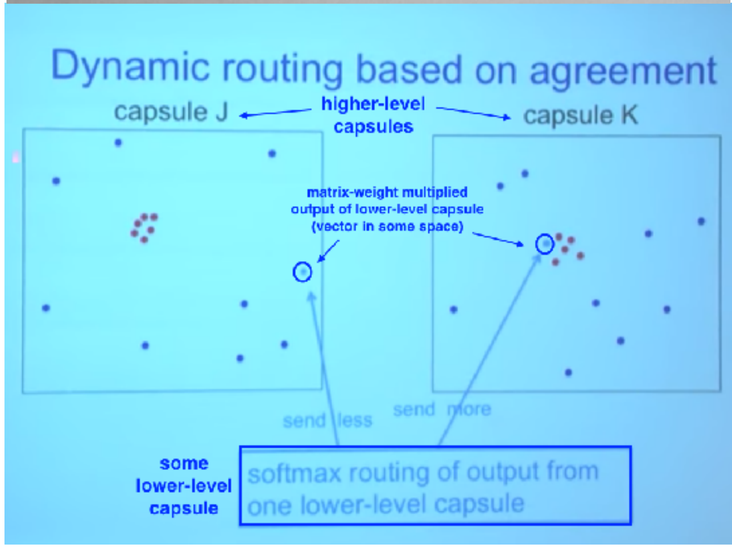
CapsNet: How?

- **Scalar Weighting of Input Vectors**
  - "**dynamic routing**" instead of backpropagation to determine where each capsule's **output** goes
  - Lower level capsule needs to "decide" to which higher level capsule it will **send** its output (i.e. **adjusting** the weights)
  - Higher level capsules receive many input vectors from other lower-level capsules (clusters of points, red)
  - Lower level capsule has a mechanism of measuring which upper level capsule better **accommodates** its results and will automatically adjust its weight accordingly.
- **Sum of Weighted Input Vectors**
  - Sum of **vectors** and not sum of scalars









Procedure 1 Routing algorithm.

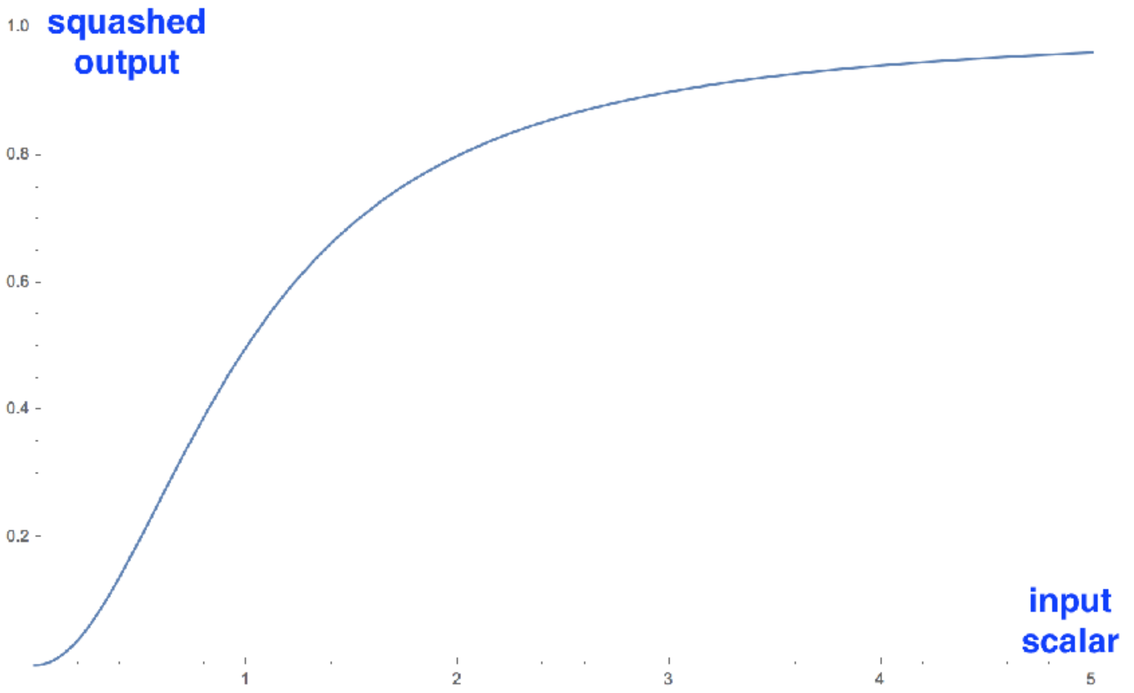
1: **procedure** ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

CapsNet: How?

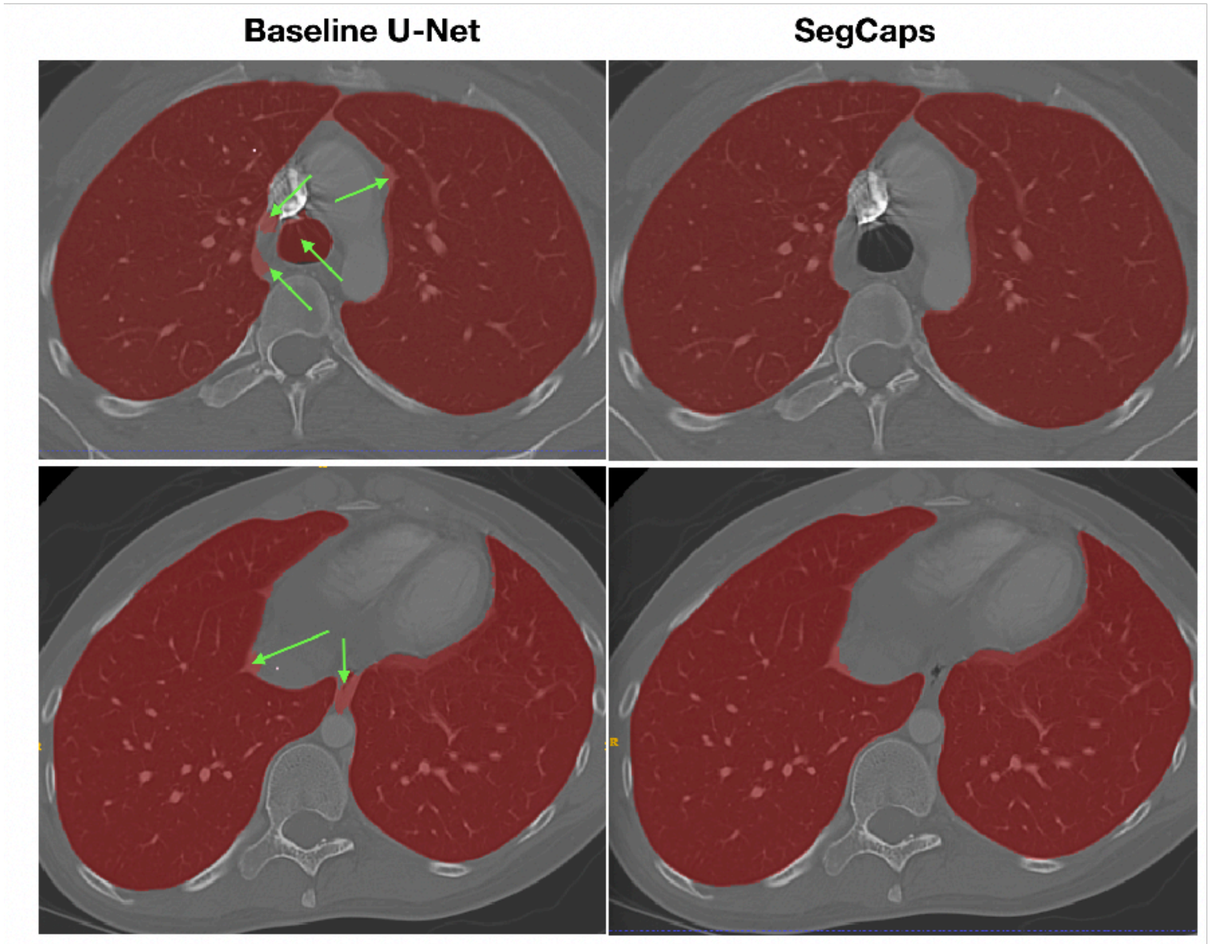
- **“Squash”: Novel Vector-to-Vector Nonlinearity**
  - Result: length of no more than 1, but does not change its direction
  - (blue rectangle) scales the input vector so that it will have unit length
  - (red rectangle) performs additional scaling
  - 1D example: demonstrate the interesting nonlinear shape of the function

$$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2} \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$$

additional “squashing” unit scaling



CapsNet: Application: Lung Segmentation



LaLonde et al. *"Capsules for Object Segmentation "* 2018.

Next time..

RNN / LSTM (CV: in combination with CNNs)