Design

Arthur J. Redfern
axr180074@utdallas.edu
Sep 17, 2018
Sep 19, 2018

Outline

- Motivation
- Goal
- Preliminaries
- Strategy
- Layers
- Networks
- Visualization
- References

Motivation

The Previous Lectures Covered A Lot

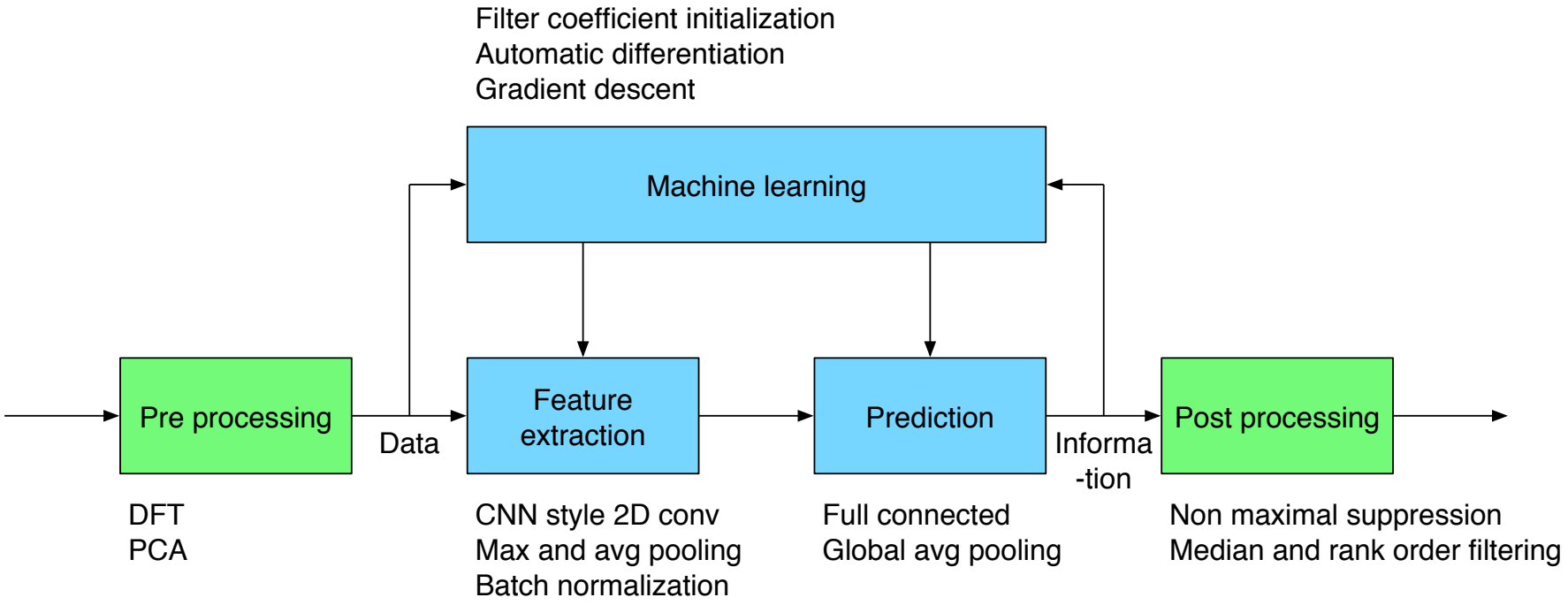
- We've briefly looked at a lot of different parts of basic math
  - 1 intro lecture where a framework for information extraction was introduced including pre processing, feature extraction, prediction and post processing
  - 2 lectures on linear algebra covering sets, fields, vectors, matrices, tensors, functions, vector spaces, normed vector spaces, inner product spaces, matrix vector multiplication, matrix matrix multiplication, CNN style 2D convolution, DFTs and PCA
  - 2 lectures on calculus covering derivatives, sub derivatives, partial derivatives, gradients, Jacobians, chain rule, critical points, gradient descent, automatic differentiation with reverse more accumulation and universal approximation
  - 2 lectures on probability covering probability spaces, events, random variables, expected value, normalization, law of large numbers, central limit theorem, random processes, stationarity, time averages, ergodicity, entropy, mutual information, Kullback Leibler divergence, data processing inequality, compression, Huffman coding and arithmetic coding
  - 1/2 of a lecture on algorithms covering comparison sorts, sequential merge sort, parallel merge sort, pooling layers, median and rank order filtering and non maximal suppression

~ 30 lectures / semester, 7.5 = 25% allocated to math; remaining ~ 25% CNNs, ~ 25% implementation, ~ 25% application; minus tests and projects

Now We Start To Put The Pieces Together

- Introduction: flow of pre processing, information extraction, prediction and post processing
- Linear algebra: CNN style 2D convolutional layers, fully connected layers, pre processing methods
- Algorithms: pooling layers, post processing methods
- Probability: initialization, information in feature maps and filter coefficients, batch normalization, error functions
- Calculus: filter coefficient estimation, approximation

Now We Start To Put The Pieces Together



But Realize That There's (A Lot) More

- A basic amount of material from linear algebra, calculus and probability was needed such everything hangs together

- But it's possible to go much deeper and broader in each of these topics
  - We'll do some of that selectively in subsequent lectures
  - And you're well setup to do more in the future based on your own interests
  - Starting from a strong base makes learning new material a whole lot better

- Summary: the diagram on the previous slide is a nice start but there's a lot more for us to consider together and for you to consider individually

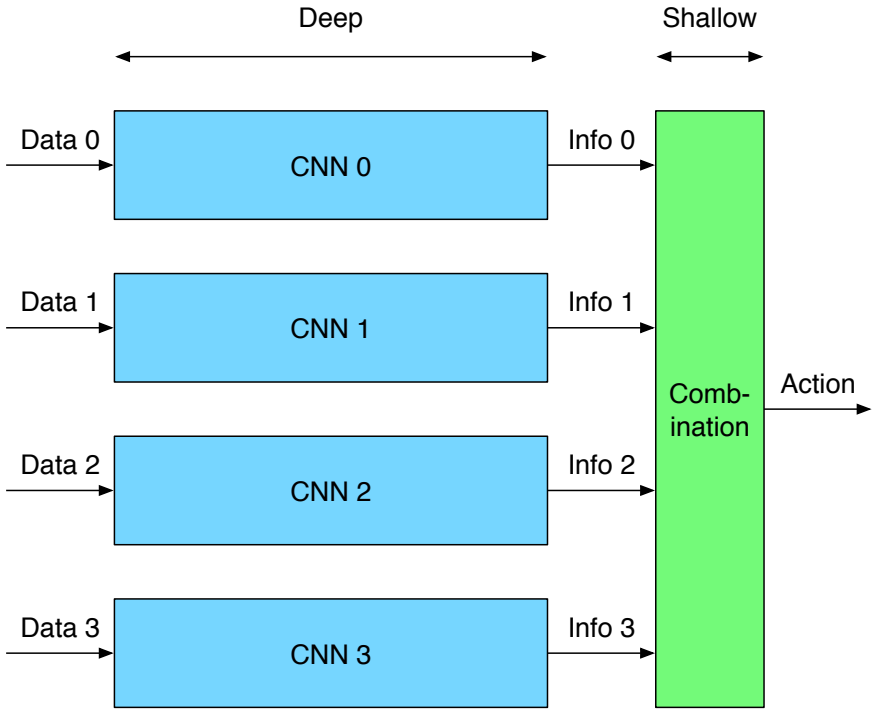
Goal

Keep Your Goals Simple

Both a deep network and general life comment

- You design a network to accomplish a goal
  - Never lose sight of this
- Keep the problems you address using deep networks like CNNs simple
  - Deep networks work best when there's a whole lot of labeled data to train on
  - The simpler the task the more data you have related to that task
  - A simple combinatorics argument illustrates this
- Handle complex problems via a shallow combination of simple problems that were solved with deep networks
  - A smaller shallow structure likely requires less data to train
  - A CNN may or may not be used for this

Keep Your Goals Simple



Example: Crossing A Busy Street

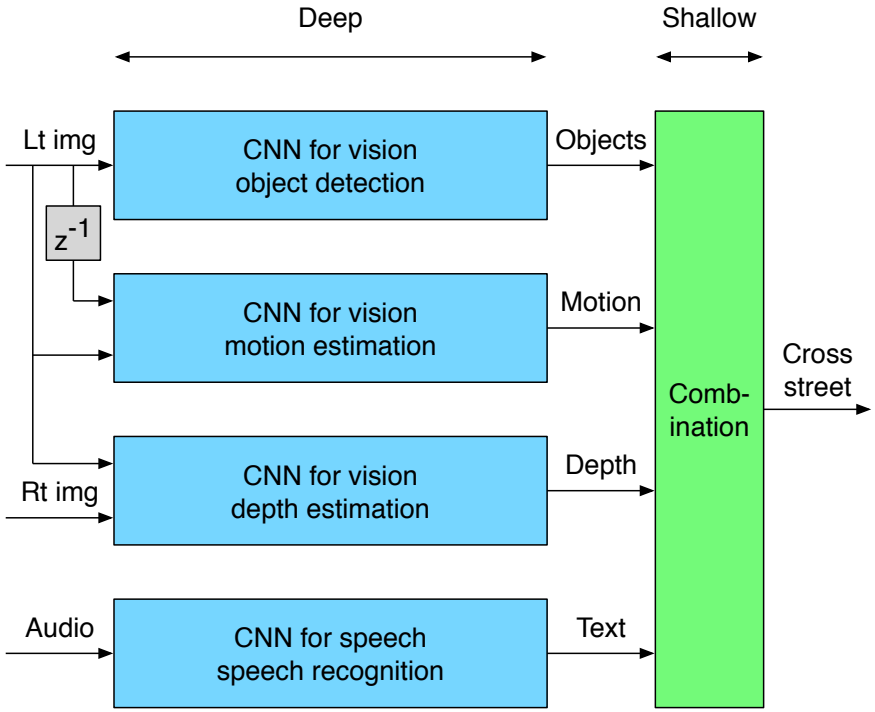
- Goal: get from 1 side of the street to the other side in a safe, efficient and law abiding way

- How do you solve this problem as a human? An incomplete list includes ...
  - Pre processing — look up from mobile phone
  - Vision — recognition of many objects in the scene, approximate distances, approximate motion
  - Sound — recognition of many sounds in the scene, approximate localization
  - Modeling — prediction of object paths vs desirable object separation for different scenarios
  - Risk analysis — from a personal health and legal perspective
  - Action — decide whether to start crossing via some trajectory at some speed looking some way
  - Repeat
  - Post processing — look down at mobile phone

https://www.freeimageslive.co.uk/files/images008/shibuya_busy_people.jpg

Example: Crossing A Busy Street

- How do you solve this problem as a computer?
  - Use CNNs to solve “simple” vision and speech problems, maybe part of the modeling prediction problem, maybe part of the action selection problem
  - Probably use other methods for risk analysis
  - Probably put the individual pieces together with a shallow structure



This Semester And This Class

- This semester will focus on small simple problems solved via CNNs
  - It's not that solving complex problems with shallow structures is not important (it is)
  - It's just that simple problems are a pre requisite and / or subset of complex problems
  - Simple problems will also be sufficiently difficult to solve that we won't get too bored

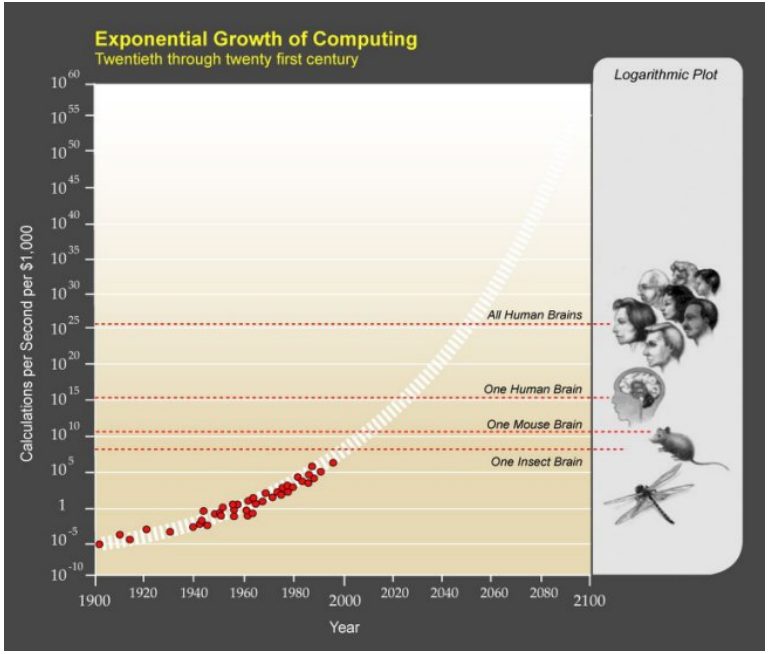
- It's nice to have a default problem to think about new ideas in the context of
  - Will use image classification as our starting point in this lecture
  - Will later generalize in a natural way to more problems in subsequent lectures on applications

Sound Like A Good Plan?

Preliminaries

Inspiration

- It's ok to look to nature for inspiration for basic principles that work
- But don't get too hung up on biology and replicating brain structure
  - If you believe in physics, the brain is an existence proof of what can be done in 3 pounds of material and 20 W of power (20 % of an average human's 100 W power budget)
  - If you believe in evolution, why build replica of the current human brain, why not build the better brain that people will have 1M years from now?
- Why have aliens never contacted us?



Theory Or Practice?

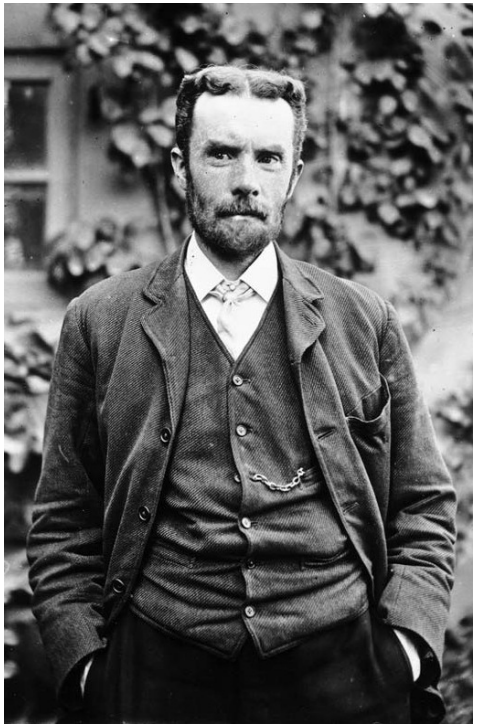
- Both (and that's ok, it's how science works)

- A frequent critique of deep learning and CNNs is that no one knows why they work
  - This is way too general a statement
  - This is not correct
  - There are strong theoretical underpinnings and frameworks for understanding what's going on
  - Is every single detail known about every single thing? No. But what scientific field would answer yes?
  - We've seen some theory, realize that much more exists

- There's a place in science for experimentation and trying new things
  - It's a reasonable way to make progress
  - Experimentation works best when it's guided by a combination of theory and intuition
  - Successes (→ practice) are nice in that they allow focused work on theory to explain

Oliver Heavyside

- "Mathematics is of two kinds, Rigorous and Physical. The former is Narrow: the latter Bold and Broad. To have to stop to formulate rigorous demonstrations would put a stop to most physico-mathematical inquiries. **Am I to refuse to eat because I do not fully understand the mechanism of digestion?**"



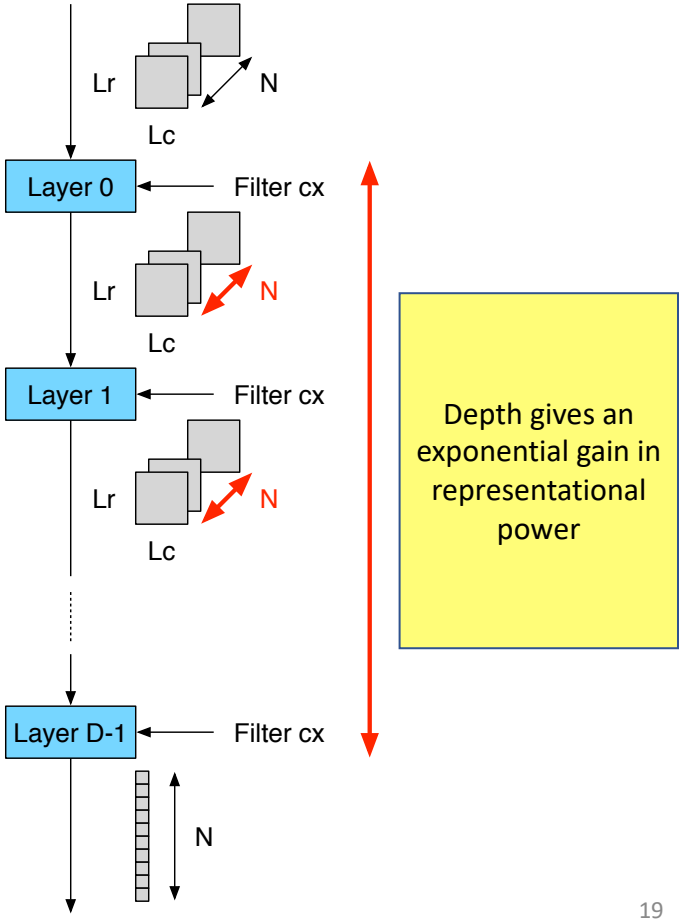
https://en.wikipedia.org/wiki/Oliver_Heaviside#/media/File:Oheaviside.jpg

Quotes != proofs, but regardless, I like this 1

Network Size

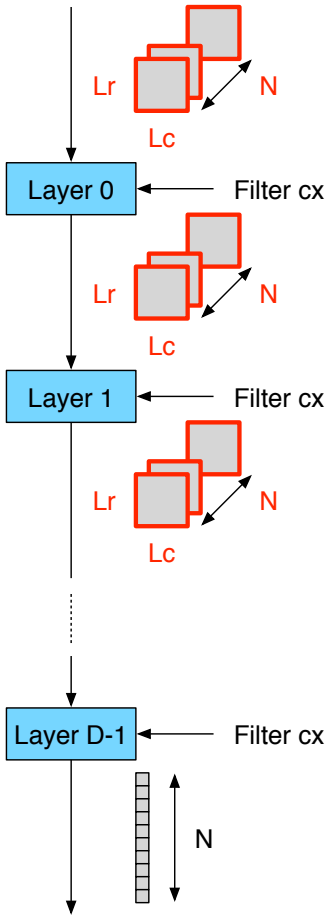
- Network depth and breadth (D, $N_{i/o}$)
  - Deeper and wider networks can approximate more complex functions
  - Remember the universal approximation proof from the calculus lecture
  - Deeper and wider networks are enabled by more data and better hardware
  - Do you want a smaller brain or bigger brain?
  - How do you sell a house?

- Performance
  - A drawback of a deeper and wider network is increased complexity / reduced performance
  - Will look at this in detail later
  - For now, pay attention to compute at the beginning and memory at the end



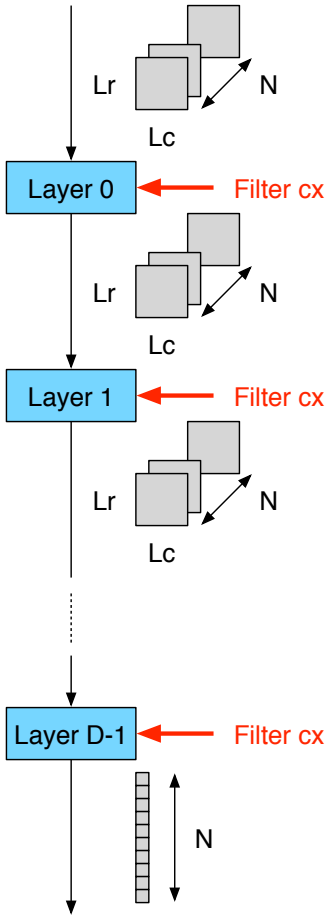
Feature Map Size

- Feature maps ($N_{i/o}$ x L_r x L_c per layer)
  - Feature maps encode information in the testing data
  - The information you're trying to extract is diffused within the feature maps
  - Need to track feature map data size as it flows through the network to make sure that there are no information killing bottlenecks
  - Remember the data processing inequality from the probability / information theory lecture



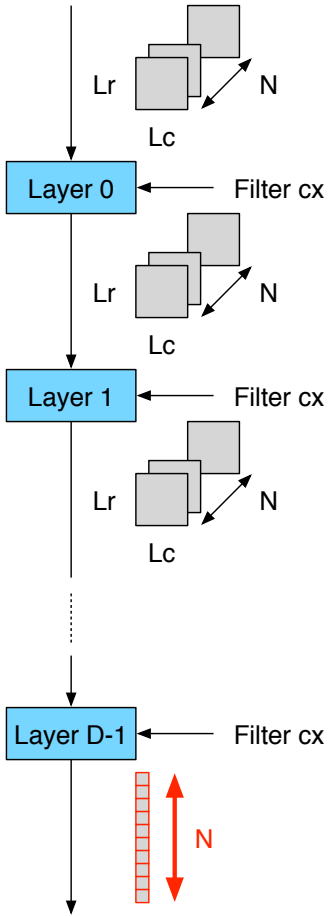
Filter Coefficient Size

- Filter coefficients (F_r x F_c x N_i x N_o and bias N_o per linear transformation)
  - Together with the network structure contain all information extracted from training data that will be applied to testing data
  - Remember automatic differentiation and gradient descent from the calculus lecture
  - Would like to have as few as possible because of performance reasons
  - But less filter coefficients tends to mean less extracted information, so there's a balance



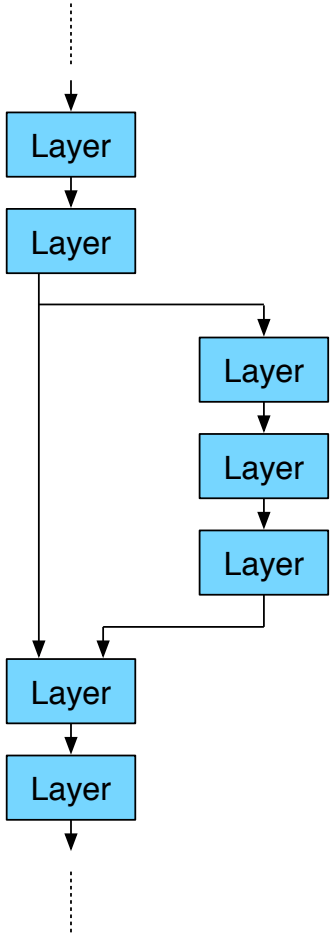
Problem Complexity

- It's important to understand how complex the problem is you're trying to solve
  - Classification to 2 classes, 1000 classes, regression to a real number, …
  - Easily separable classes or easily confused classes?
  - The simpler the function needed to solve the problem the simpler the network that can be used to solve it (and the converse is also true)
  - Remember the universal approximation proof from the calculus lecture
  - Lot's of problems we care about are really pretty complex



Graph Specification

- Implicitly or explicitly, computational graphs are used for high level network descriptions (other algorithms too)
- Graph edges are memory and nodes are compute
  - Not shown are practicalities that we'll care about during the implementation including feature map location and data movement, instruction location and data movement, computation partitioning, computation algorithm selection, ...
  - We'll deal with this in the implementation section
- Graphs are used for training and testing
  - We've already seen this with automatic differentiation with reverse mode accumulation
  - Typically just need to specify the forward graph, the backward graph and weight update graph can be auto generated (with a few other differences between training and testing)



Strategy

A Framework For Thinking About Networks

- Feature extraction and prediction (the focus of these slides using CNNs)
  - Tail feature extraction (optional) | encoder
  - Body feature extraction | encoder
  - Head prediction | decoder
- Will discuss ~ application specific components of the data to information extraction problem later in the context of applications
  - Pre processing
  - Post processing
- Will also discuss training, evaluation, performance, implementation and more applications later
  - For now just focusing on design basics
  - But realize that everything is coupled

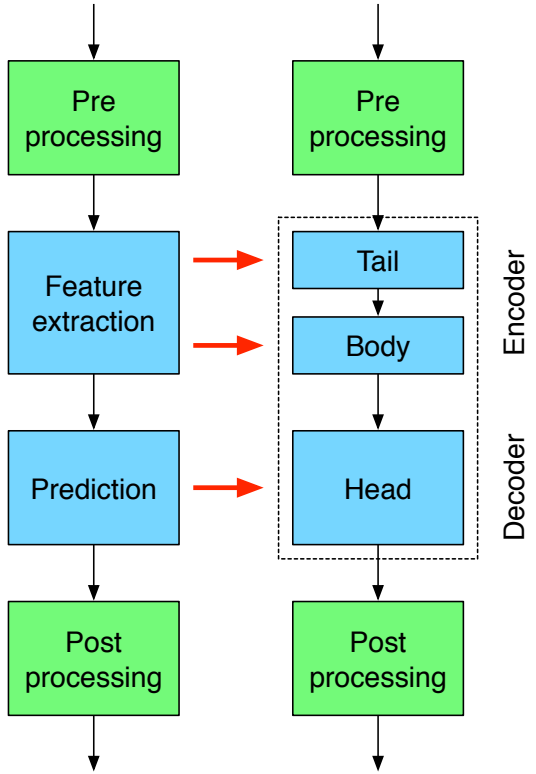
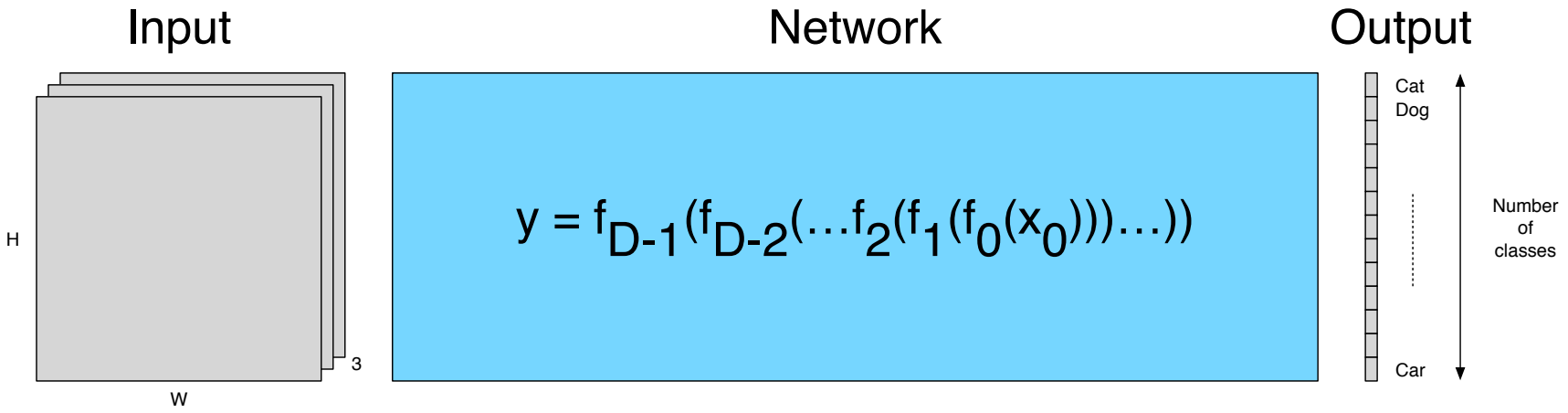


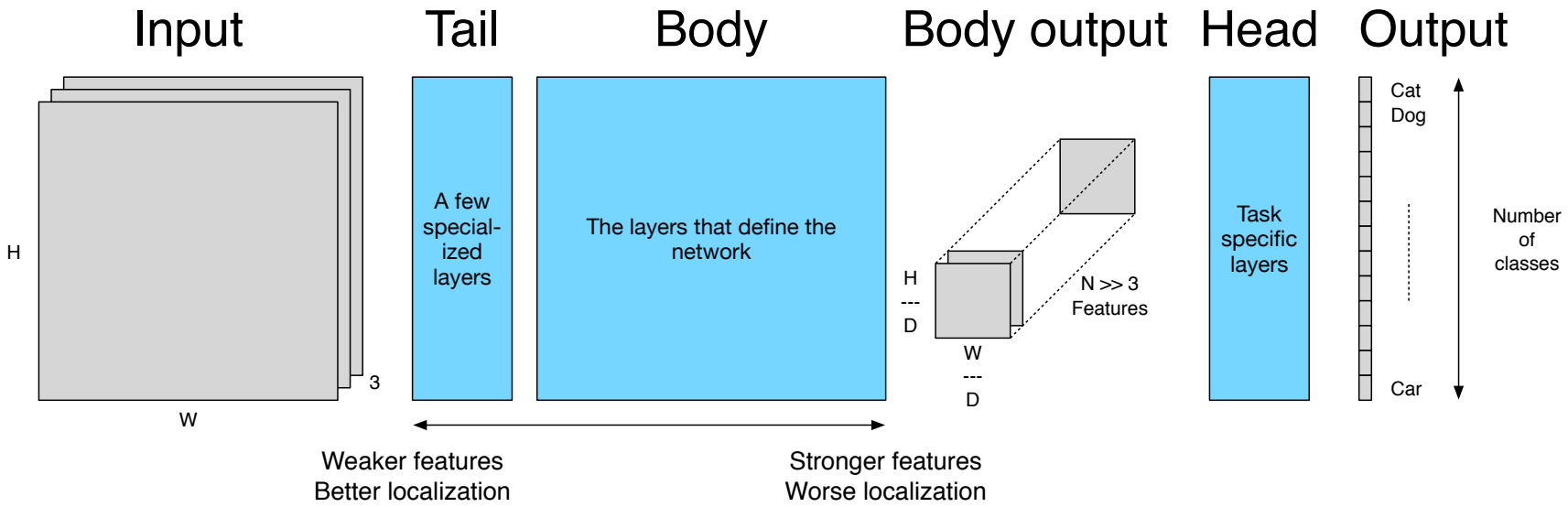
Image Classification As A Starting Point

A classification network maps an input image to an output vector; the largest element of the output vector is the predicted class

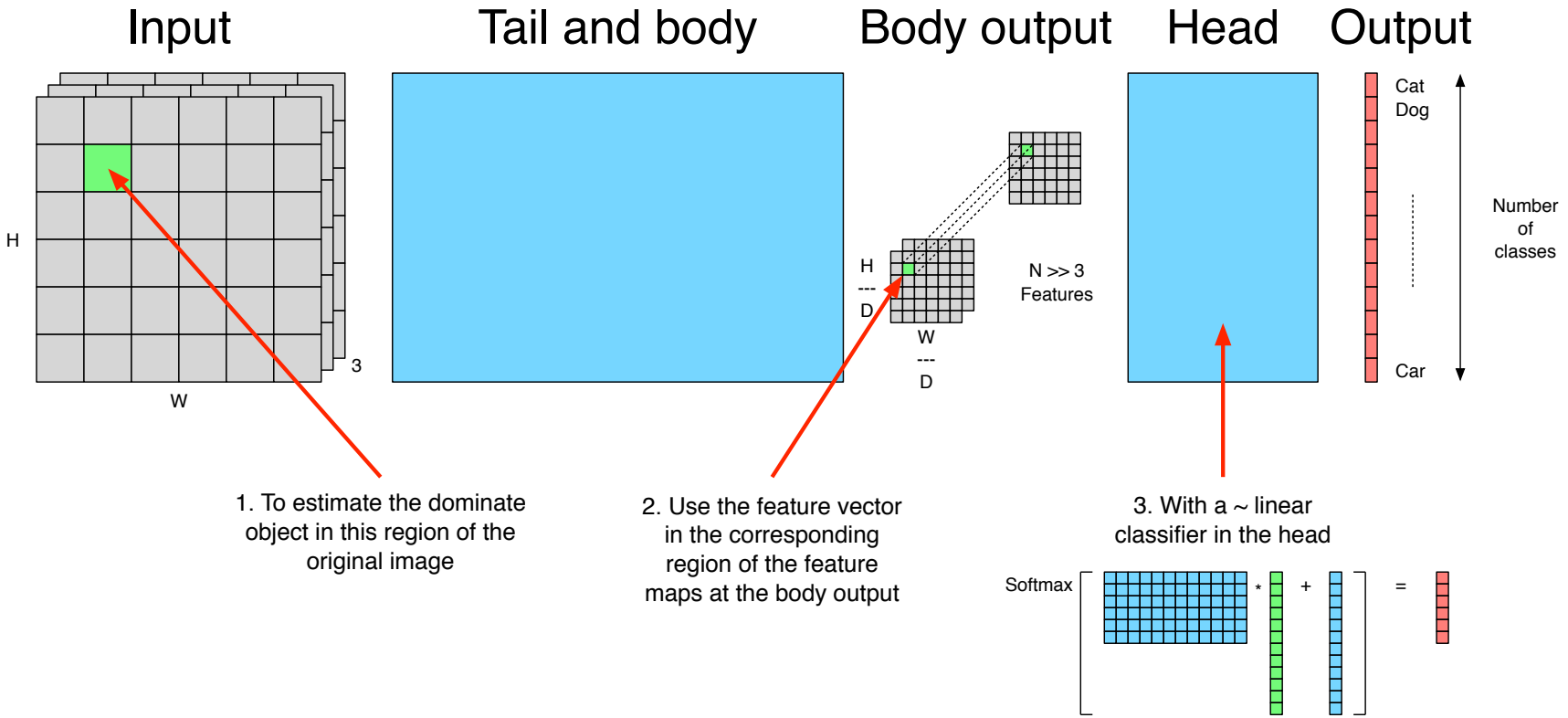


Tail Body And Head Decomposition

Modern networks decompose into a tail, body and head; the tail and body encode / extract features and the head decodes / makes predictions

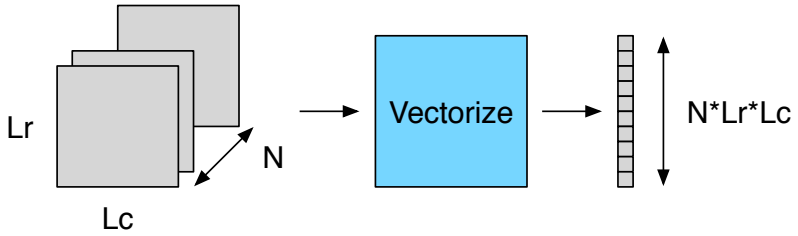


Conceptually How The Head Works

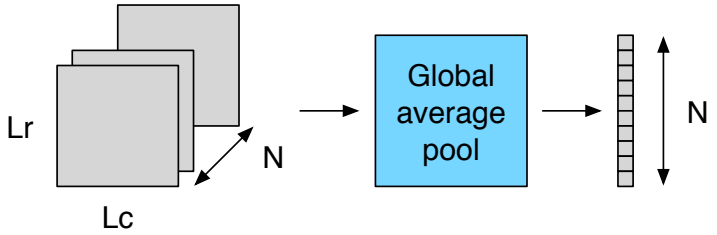


Feature Map To Feature Vector Transform

- 3D tensor to 1D vector transformation
  - Output of convolutional layers is N x Lr x Lc
  - Input to linear classifier is N x 1
  - 2 common methods for transforming between the output of the convolutional layers to the input of the linear classifier



- Vectorize
  - Historically first
  - Allows features for all pixels to be combined arbitrarily at the expense of more complexity and memory
  - Likely a lot of redundancy in the subsequent classifier
    - Perhaps why dropout is frequently used for this case



- Global average pool
  - Most popular right now
  - Average features together for all pixels
  - Less complexity and memory (math subset of vectorize)

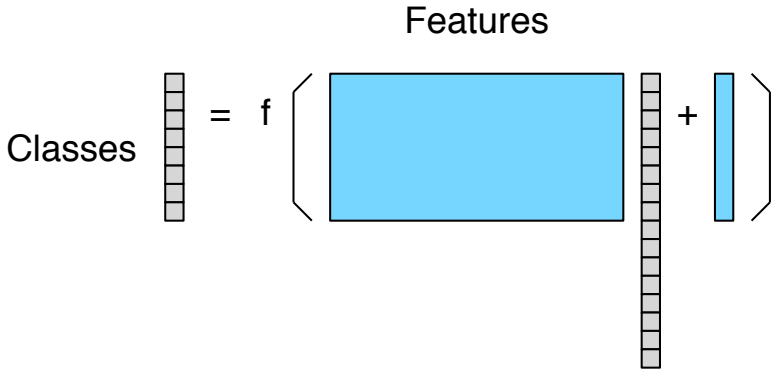
The linear classifier requires an input vector of a fixed size; convolutional layers in the body work on feature maps of ~ arbitrary size; this needs to connect them

Linear Classifier

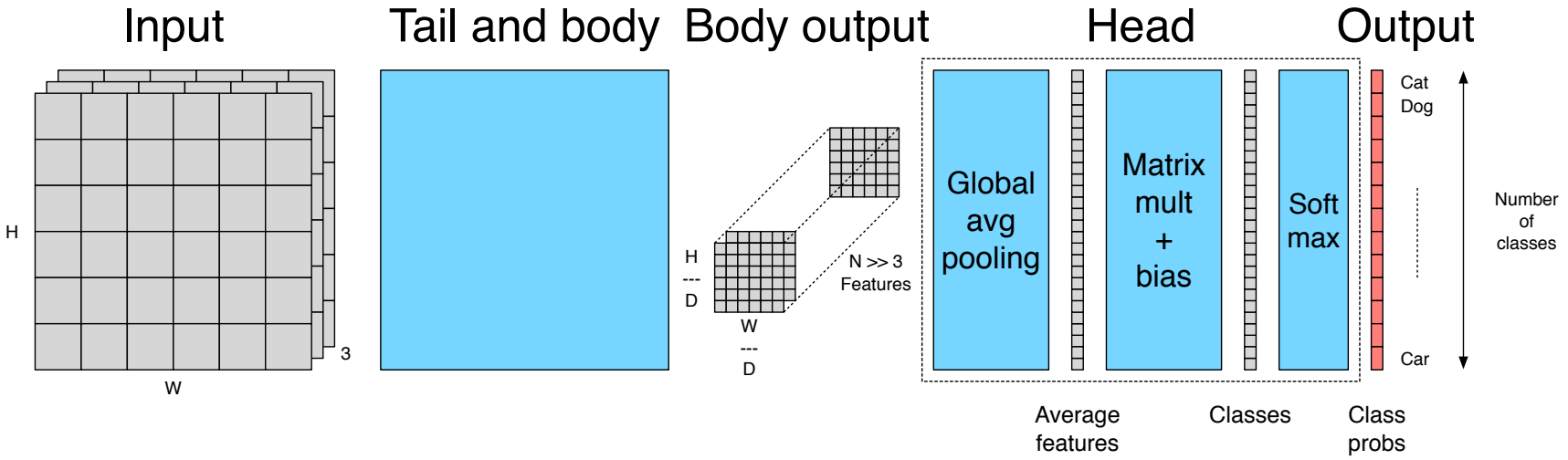
- Linear classifier
  - + softmax to convert to probabilities (training)
  - + arg max to select max output as class (testing)

- Implicit assumption is that the output classes are linearly separable in terms of the input features to the linear classifier (the job of the tail and body)

- The linear classifier works best when the number of features is same order of magnitude or larger than the number of classes
  - See the linear algebra notes for outline of proof
  - Intuition is that more features add robustness to the estimation



Example Head Design



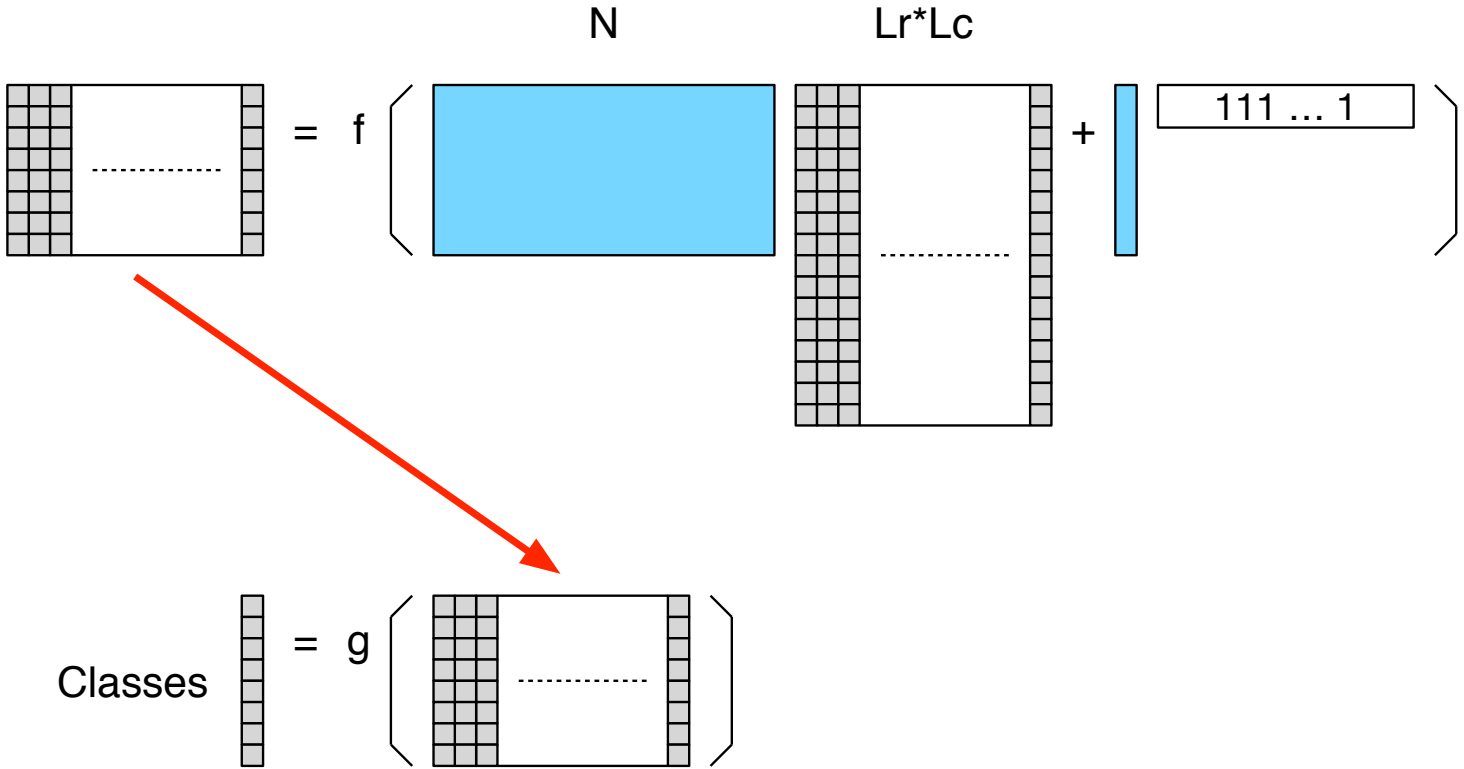
2 Head Design Ideas

- Maybe already done by someone else (I don't know)?
  Maybe doesn't work as well (I don't know)?
  Maybe would make a nice project (I don't know)? ...

- An idea that sits in between vectorize and global average pool
  - Idea is to estimate the class of each final feature map pixel (which corresponds to a region in the original image) then combine these classes to estimate the dominant class of the full image
  - Mechanics: create vectors for each pixel, process them all with the same classifier then combine classifier outputs (different options for this, could be as simple as picking the most common, could be trained which allows class combination and location to come into play; SqueezeNet V1 did this where the combination is averaging, is there a better method?)
  - Related: could potentially create better classification error functions with training data with localized labels
  - Similar memory and compute to global average pool (matrix matrix vs matrix vector which is memory bound)

- An idea that is an enhancement of global average pooling followed by a linear classifier
  - Different regions in the image potentially have different levels of importance for predicting the dominant class in the image (maybe edges are less important and the center is more important?)
  - Mechanics: weight the features of the final feature map different spatially via a pointwise multiplication before the global average pooling, potentially learn the optimal weighting

Head Design Idea 1

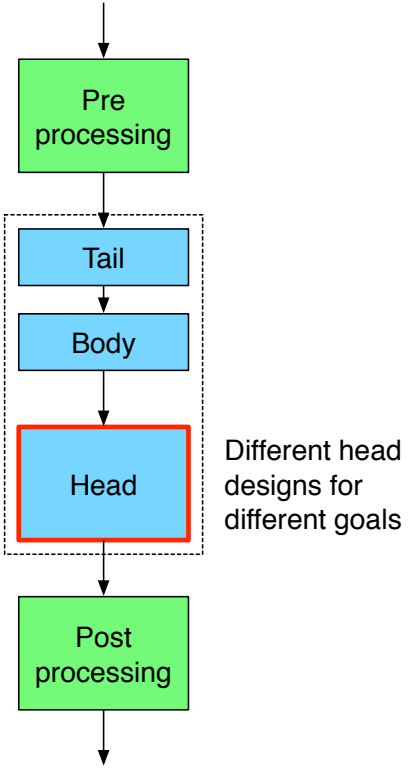


In The Near Future

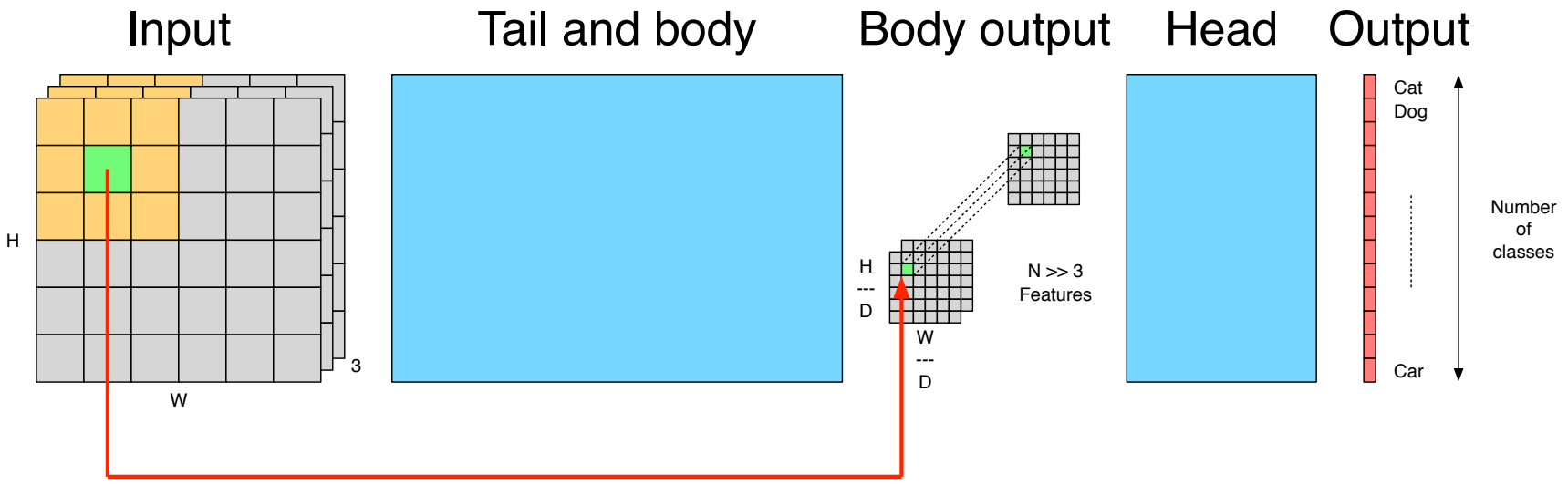
- Specifically, when we look at different applications in more detail

- We're going to look at different head designs to accomplish different goals
  - Multiple object detection
  - Segmentation
  - Depth
  - Motion
  - Character estimation
  - ...

- Note that more than 1 head can be attached to the same tail and body
  - Implicitly, the same features are used for more than 1 task



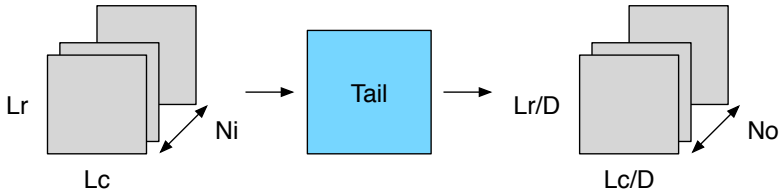
Conceptually How The Tail And Body Work



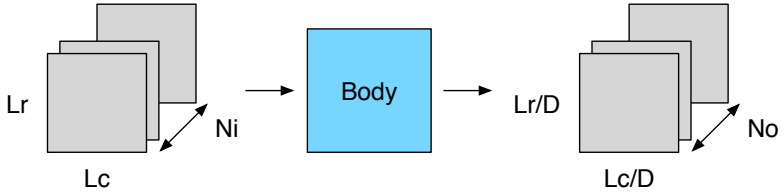
The tail and body transform regions from an image to feature vectors; the receptive field size shown in orange is the area around the region of interest that the body can use to create the feature vector

Why Distinguish Between The Tail And Body?

- The distinction is somewhat arbitrary
  - Both take input feature maps (an RGB input is 3 feature maps) and generate output feature maps
  - Features get stronger after each transformation



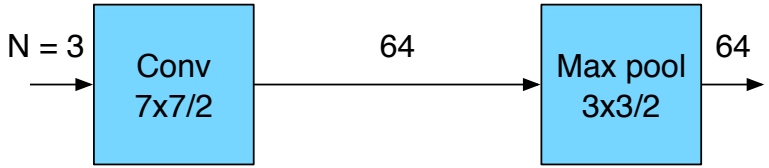
- So why make the distinction?
  - There are a few common tail designs that most people use
  - There are a lot of different building blocks for the body that effectively define the network name



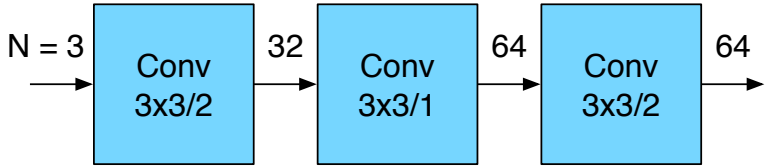
- Plan
  - We'll discuss a few tail designs in this section
  - We'll discuss body designs in the network section

Tail: Data To Weak Features

- Initial data to feature mapping
  - Optional (can be thought of as part of the body)



- Features
  - High compute
  - High feature map memory
  - Low parameter memory
  - Lots of spatial redundancy



- Common design components
  - Aggressive down sampling (but don't lose useful information)
  - Aggressive increase in the number of channels

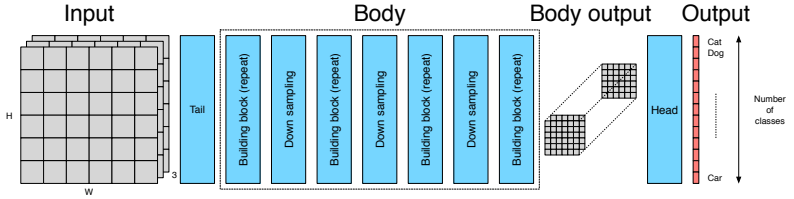
Project idea: look at the transformations in trained tail designs, look for ways of saving compute (e.g., I believe a paper in the past exploited symmetry to 1/2 compute)

Body: Weak Features To Strong Features

- Building blocks we'll see in network design followed by down sampling (strided convolution or pooling)

- Interpretation
  - Lower level features are more class agnostic
    - Less channels earlier in the network
  - Deeper features are more class specific
    - More channels later in the network

- Feature map changes
  - Gradual memory reduction
  - Loss of spatial resolution
  - Increase of channel / feature dimension and strength



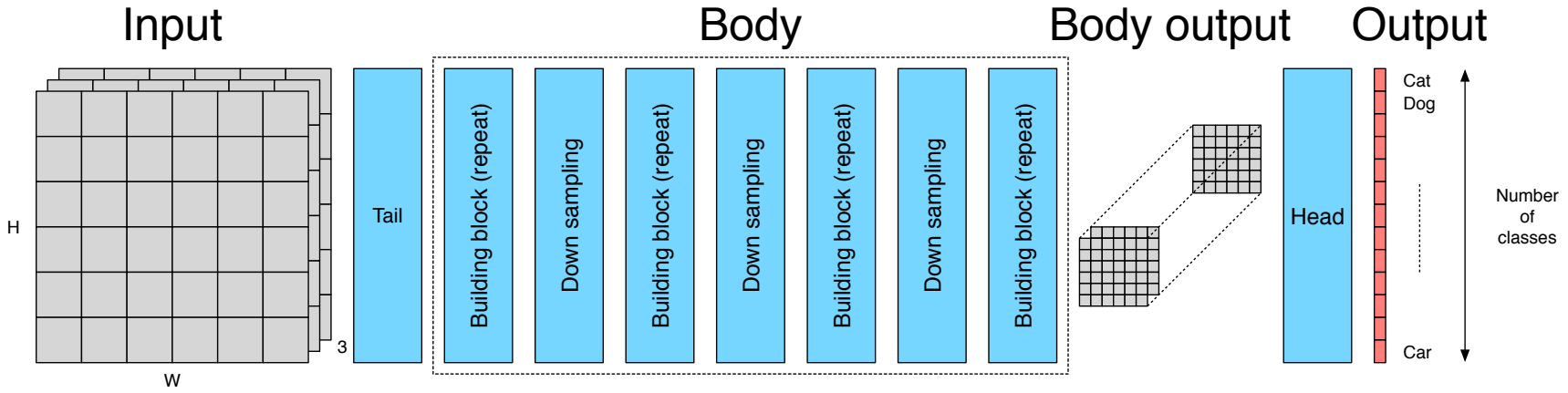
- Around down sampling stages it's common to
  - Reduce the rows and cols both by 2
  - Increase the channels by 2

- Result is the data volume shrinks by 2
  - Data before = channels x rows x cols
  - Data after = 2*channels x rows/2 x cols/2
    = data before / 2
  - Effectively reduces compute by a factor of 2

- Project idea: investigate ordering of channel increase and down sampling for classification and segmentation

Body: Weak Features To Strong Features

For ImageNet classification it's common to have 5 levels of down sampling before the global average pool: ~ 2 in the tail and ~ 3 in the body



Receptive Field Size

- Receptive field size at the head
  - How to compute in theory
  - But effectively smaller in practice (convolving a bunch of filters gives a ~ Gaussian shape)
- What does the classifier have to work with when making a decision
  - Idea of looking at the input through a screen of that size
  - Objects of different classes can have different sizes, objects of the same class can have different sizes
  - Implications on changing the input resolution

Calculation

- Start at the output last convolutional layer and initialize the receptive field size to 1
- Move backwards through the network to the very beginning
- Every time you go through a filter increase the receptive field size by adding F – 1 (the length of the filter – 1); observe that 1x1 filters do not increase the receptive field size
- Every time you go backwards through a down sampling operation multiply the receptive field size by S (the down sampling factor)
- You can find the receptive field size at any place in the network; later useful when looking at skip connections
- Once at the very beginning you have the receptive field size with respect to the input
- Note: this is the maximum theoretical size, and the actual size will likely be much smaller (think a Gaussian like shape centered in the receptive field span that collects energy non uniformly)

Layers

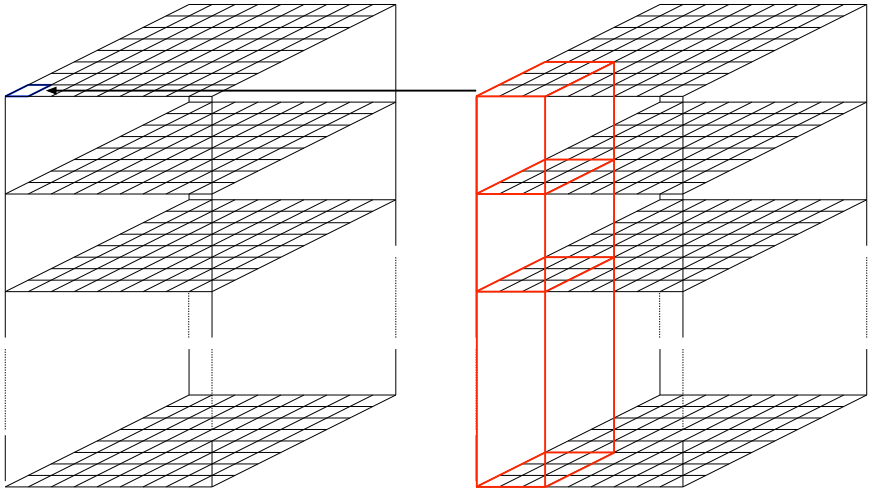
Layer Overview

- Layers map input feature maps to outputs feature maps
  - This section will look at the individual operations
  - The network section will combine multiple layers together into building blocks

- The key is to understand the dimensions in which layers combine information when mapping from input to output
  - Channel and space
  - Channel
  - Space
  - Element
  - Time
  - Rearrangement
  - Training

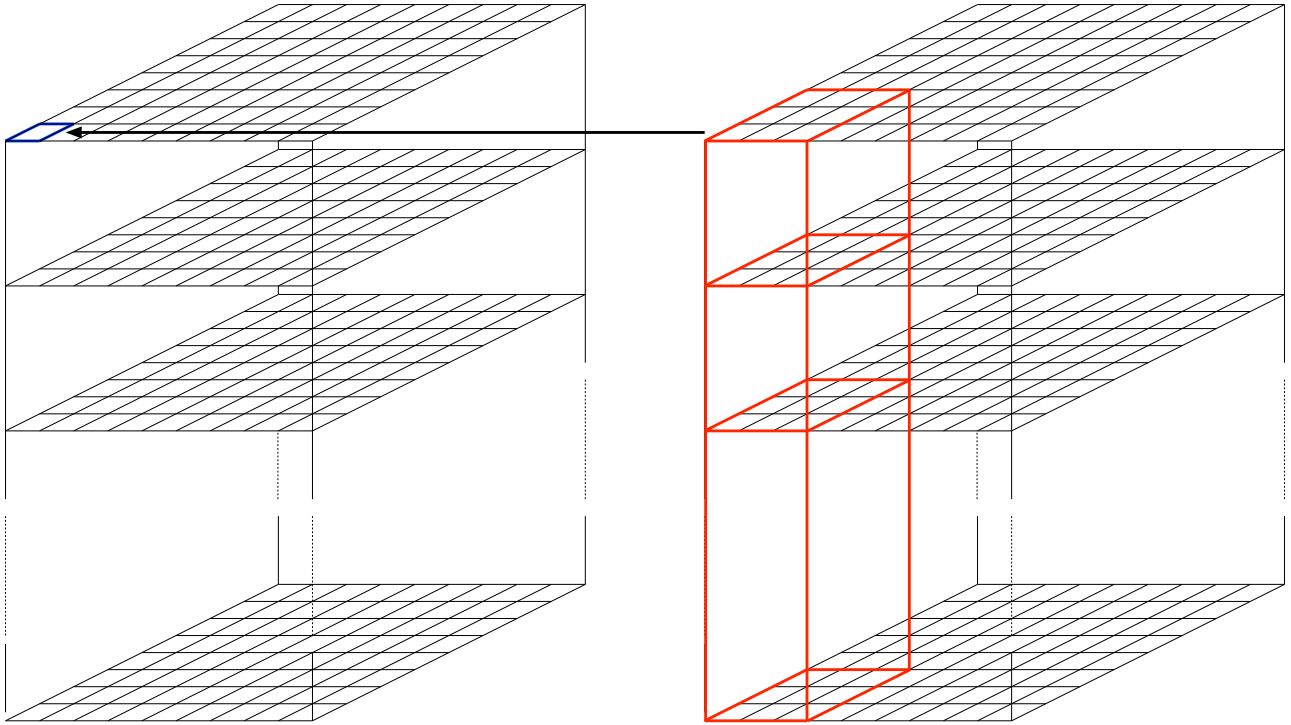
Channel And Space

- CNN style 2D convolution layer
  - Filter
    - Tensor up sampling
    - Matrix creation (nothing to do)
  - Input feature map
    - Tensor up sampling
    - Tensor 0 padding
    - Matrix create (Toeplitz style)
  - Matrix reduction from filter up sampling
    - Remove cols from filter matrix from up sampling
    - Remove associated rows from the input matrix
  - Matrix reduction from output down sampling
    - Remove cols from input matrix
  - Output feature map
    - Matrix create via filter matrix * input matrix + bias
    - Tensor creation (nothing to do)



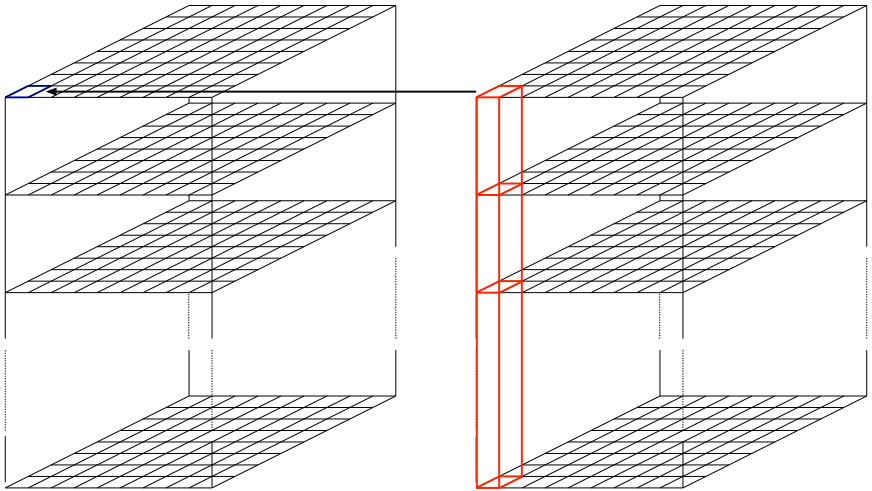
Spatial invariance of CNN style 2D convolution allows for memory and compute reduction but is also plausible with respect to feature extraction from many images

Channel And Space

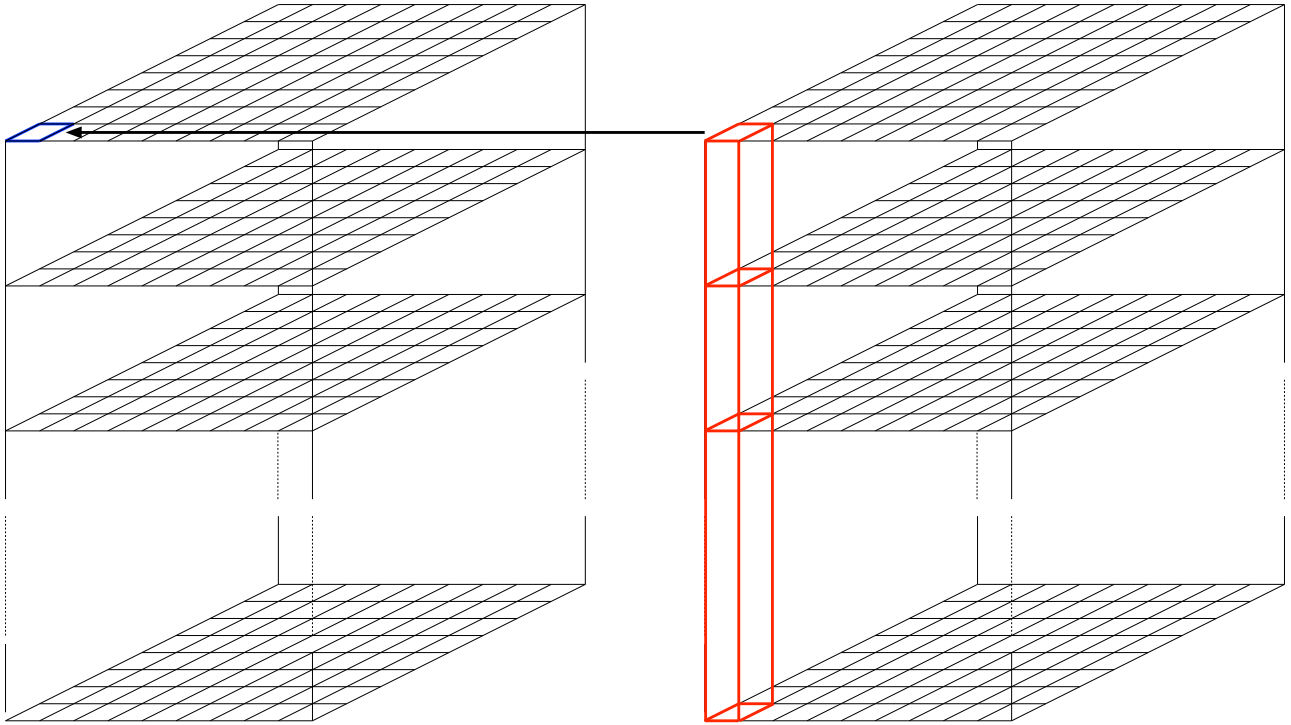


Channel

- CNN style 2D convolution with Fr = Fc = 1 (matrix matrix multiplication)
- Fully connected layer for a Ni x 1 x 1 input (or batch of Ni x 1 x 1 inputs)
- Local response normalization
  - Famously used in AlexNet / CaffeNet
  - Typically not used now (input normalization via pre processing helps some)

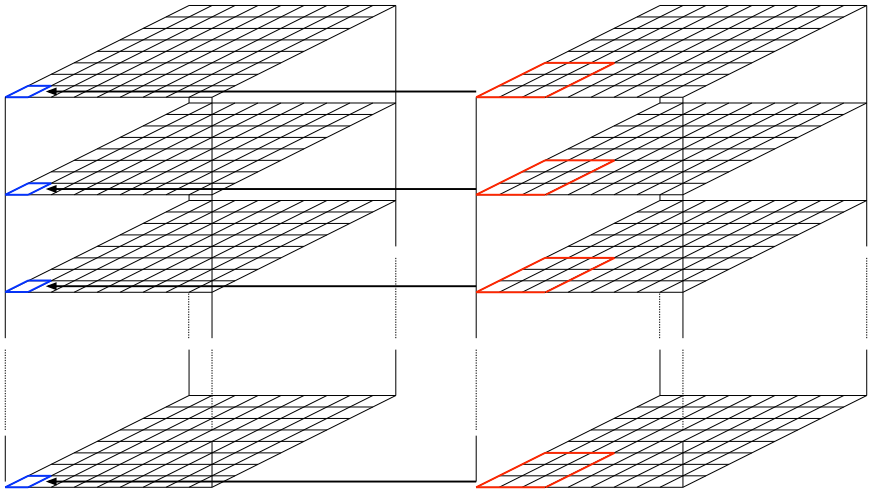


Channel



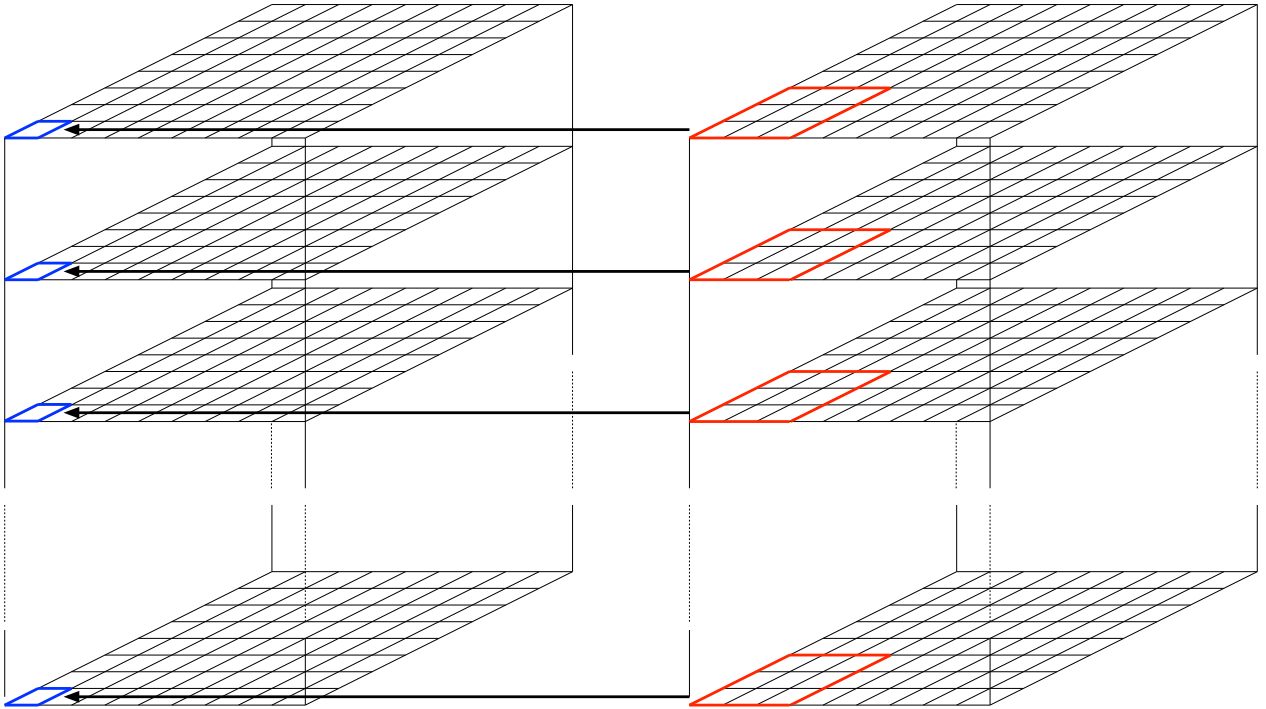
Space

- CNN style 2D convolution that's fully grouped (standard 2D convolution, depth wise separable convolution)
- Decimation
  - Max pooling
  - Avg pooling
  - Global average pooling
  - Spatial pyramid pooling
- Interpolation
  - Nearest neighbor
  - Bilinear



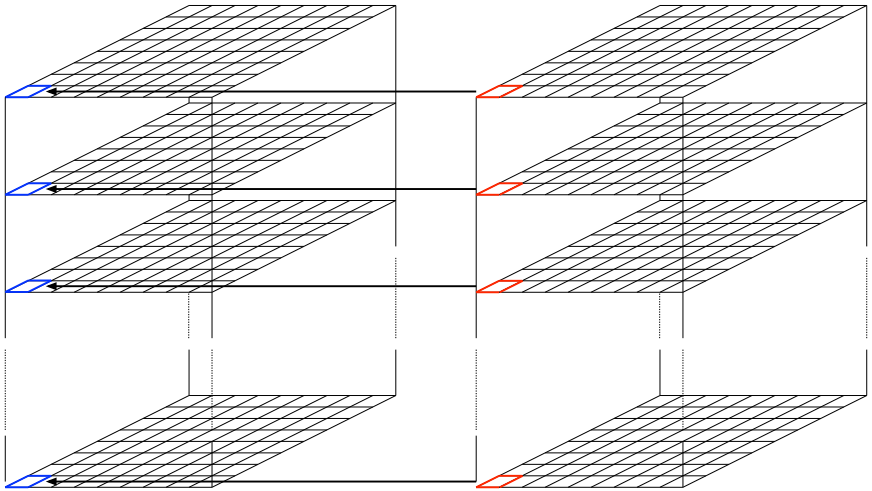
Pooling, as shown in the above figure, allows for the aggregation of larger regions of input features in the generation of output features (increasing receptive field size)

Space

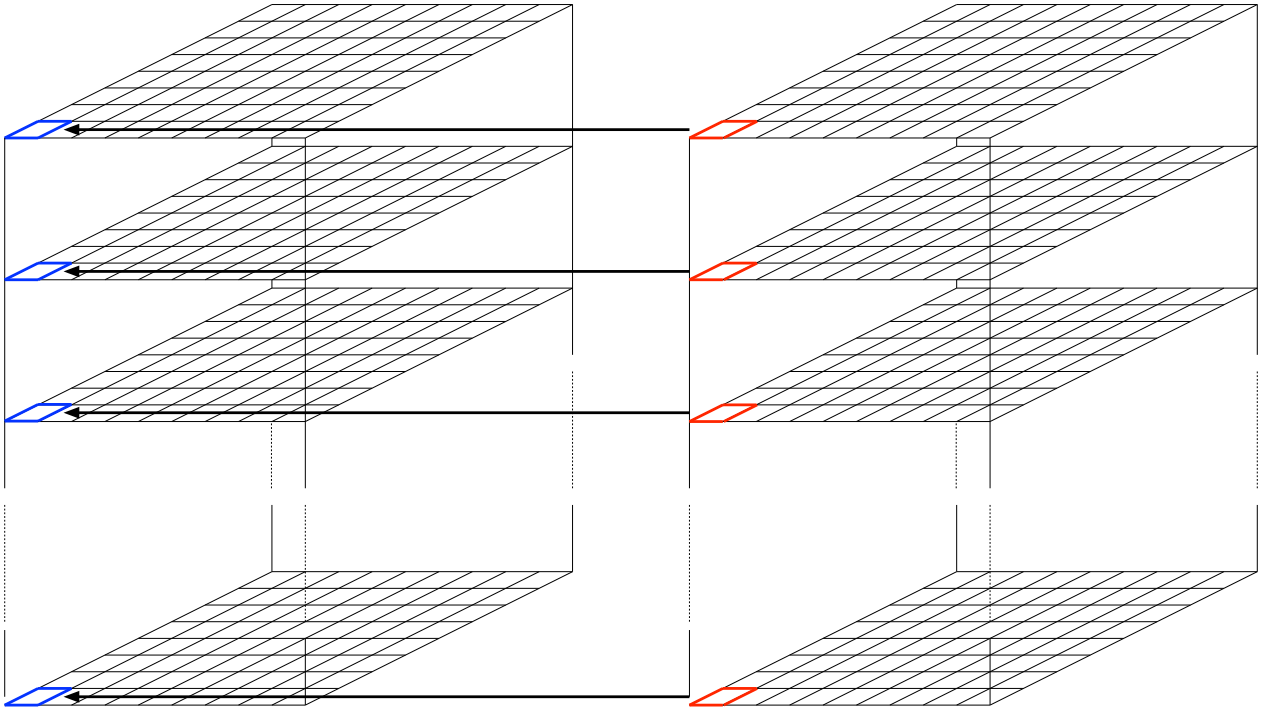


Element

- Nonlinearities
  - ReLU, ReLU6, PReLU, ReLU with a small negative slope, sigmoid, tanh, ...
- 1 input 1 output linear
  - Scale
  - Offset
- 2 input 1 output math
  - Add, multiply, max, ...



Element

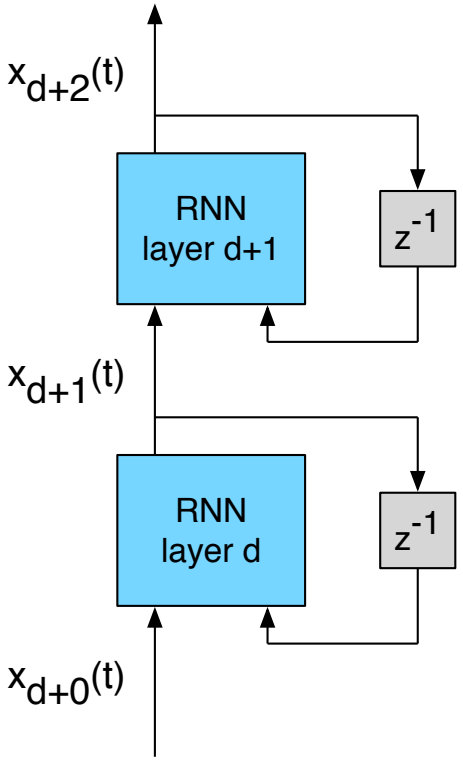


Time

- Will come back to these in the context of speech and language

- Examples (many variants exist)
  - RNN
  - LSTM

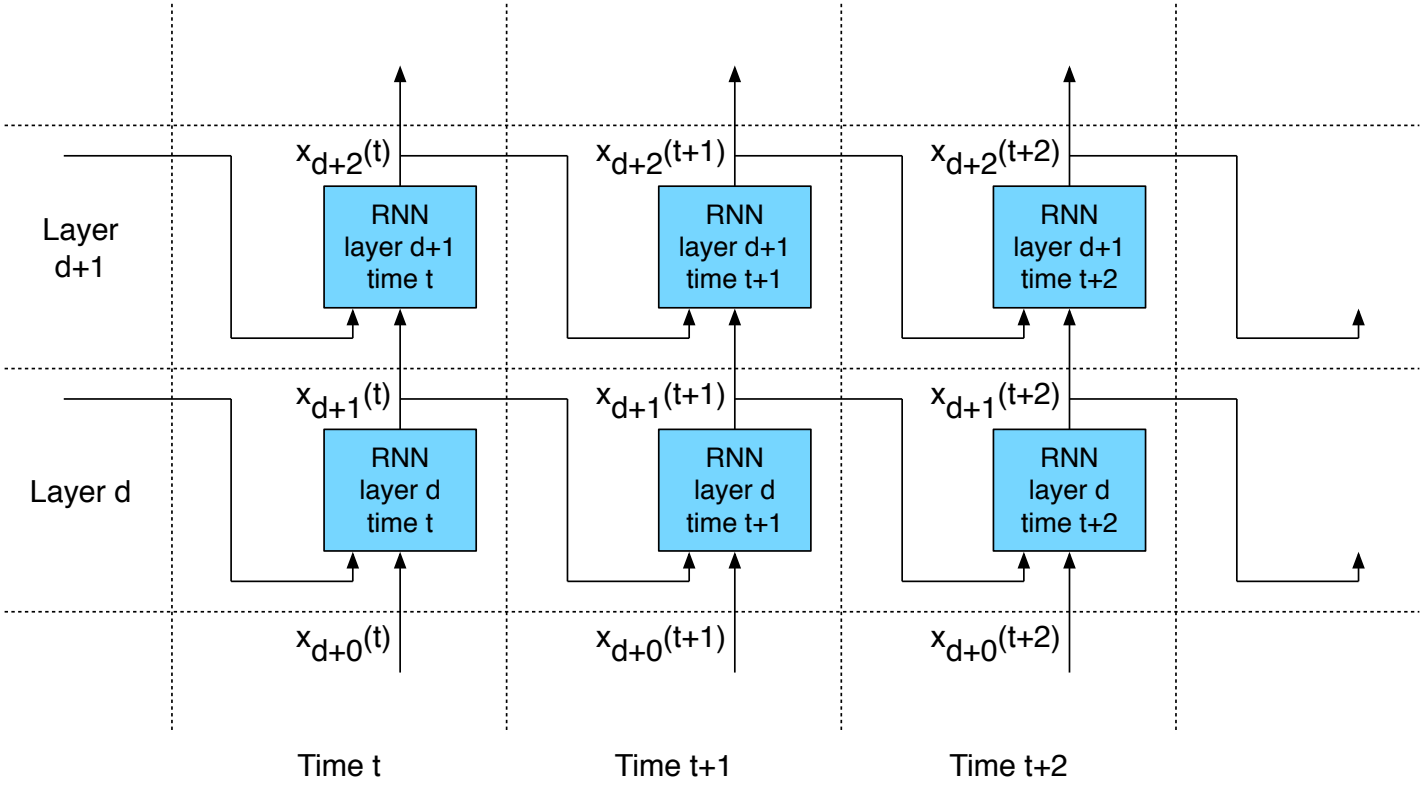
- An example deep RNN
  - $\mathbf{x}_{d+1}(t) = f_{d+0}(\mathbf{G}_{d+0}\,\mathbf{x}_{d+1}(t-1) + \mathbf{H}_{d+0}\,\mathbf{x}_{d+0}(t) + \mathbf{v}_{d+0})$
  - $\mathbf{x}_{d+2}(t) = f_{d+1}(\mathbf{G}_{d+1}\,\mathbf{x}_{d+2}(t-1) + \mathbf{H}_{d+1}\,\mathbf{x}_{d+1}(t) + \mathbf{v}_{d+1})$
  - ...



Question to myself, maybe a project idea (?): investigate the structure of G for different applications and at different places in the network; how much mixing is there across features?

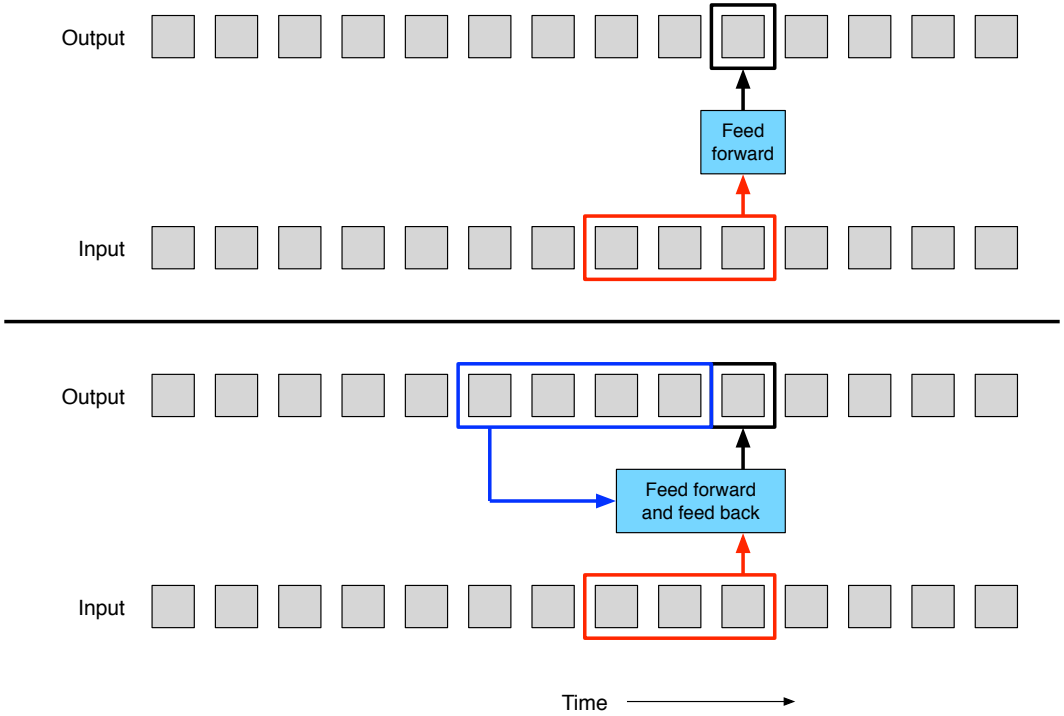
Time

The RNN on the previous slide unwrapped in time



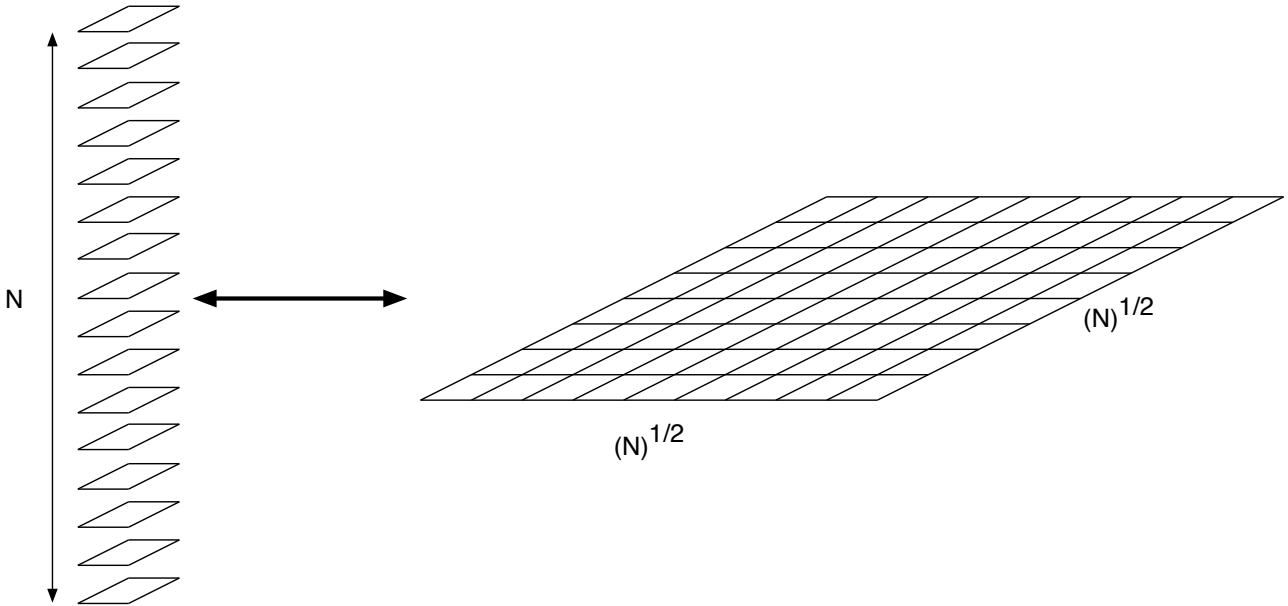
Time

Note commonalities between fully connected layers and FIR filtering and RNN layers and IIR filtering



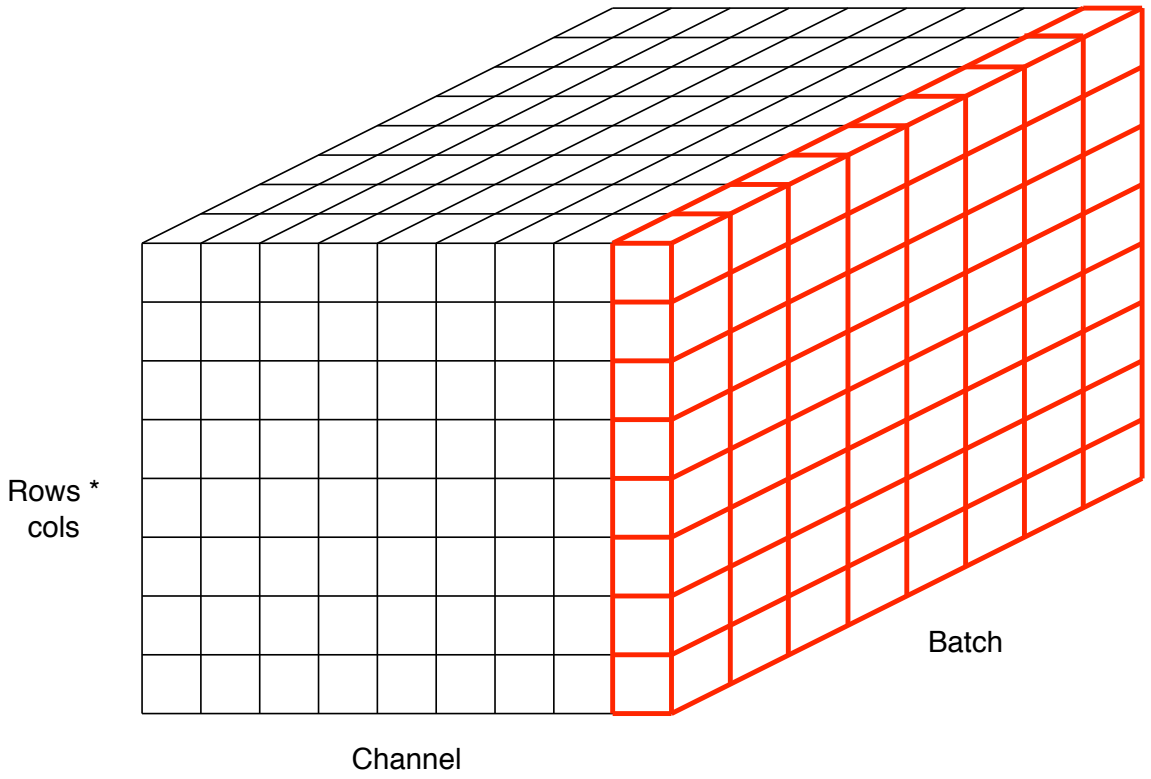
Rearrangement

- Reshape
  - Channel to space
  - Space to channel
- Concatenate
  - Channel
  - Space
- Split
  - Channel
  - Space



Training

- Will come back to these in the context of training
- Normalization
  - Batch
  - Group
- Dropout
- Loss
  - Lp
  - Softmax + cross entropy



Networks

Visualization

References

List

- References for individual networks are listed on the slide on which they occur

- Mathematics of deep learning
  - https://arxiv.org/pdf/1712.04741.pdf
  - http://www.vision.jhu.edu/tutorials/ICCV17-Tutorial-Math-Deep-Learning-Intro-Rene.pdf
- On the number of linear regions of deep neural networks
  - https://arxiv.org/abs/1402.1869
- An introduction to deep learning (lecture 1)
  - http://www.cs.toronto.edu/~ranzato/files/ranzato_deeplearn17_lec1_vision.pdf
- Image classification with deep learning
  - http://www.cs.toronto.edu/~ranzato/files/ranzato_CNN_stanford2015.pdf
- Batch normalization: accelerating deep network training by reducing internal covariate shift
  - https://arxiv.org/abs/1502.03167
- Group normalization
  - https://arxiv.org/abs/1803.08494

List

- Improving neural networks by preventing co-adaptation of feature detectors
  - https://arxiv.org/abs/1207.0580
- VGG convolutional neural networks practical
  - http://www.robots.ox.ac.uk/~vgg/practicals/cnn/index.html
- Deep learning
  - http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf
- Deep learning for AI
  - http://www.iro.umontreal.ca/~bengioy/talks/IJCAI2018-DLtutorial.html
- Netscope
  - https://dgschwend.github.io/netscope/quickstart.html