DEEP LEARNING FOR COMPUTER VISION

Summer Seminar UPC TelecomBCN, 4 - 8 July 2016

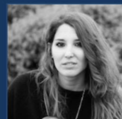
Instructors

Xavier Giró-i-Nieto

Elisa Sayrol

Amaia Salvador

Jordi Torres


Eva Mohedano

Kevin McGuinness

Organizers

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Co-funded by the Erasmus+ Programme of the European Union

+ info: TelecomBCN.DeepLearning.Barcelona

Day 2 Lecture 1

Memory usage and computational considerations

Introduction

Useful when designing deep neural network architectures to be able to estimate memory and computational requirements on the “back of an envelope”

This lecture will cover:

- Estimating neural network memory consumption
- Mini-batch sizes and gradient splitting trick
- Estimating neural network computation (FLOP/s)
- Calculating effective aperture sizes

Improving convnet accuracy

A common strategy for improving convnet accuracy is to **make it bigger**

- Add more layers
- Made layers wider, increase depth
- Increase kernel sizes*

Works if you have sufficient data and strong regularization (dropout, maxout, etc.)

Especially true in light of recent advances:

- ResNets: 50-1000 layers
- Batch normalization: reduce covariate shift

network	year	layers	top-5
Alexnet	2012	7	17.0
VGG-19	2014	19	9.35
GoogleNet	2014	22	9.15
Resnet-50	2015	50	6.71
Resnet-152	2015	152	5.71

Increasing network size

Increasing network size means using more memory

Train time:

- Memory to store outputs of intermediate layers (forward pass)
- Memory to store parameters
- Memory to store error signal at each neuron
- Memory to store gradient of parameters
- Any extra memory needed by optimizer (e.g. for momentum)

Test time:

- Memory to store outputs of intermediate layers (forward pass)
- Memory to store parameters

Modern GPUs are still relatively memory constrained:

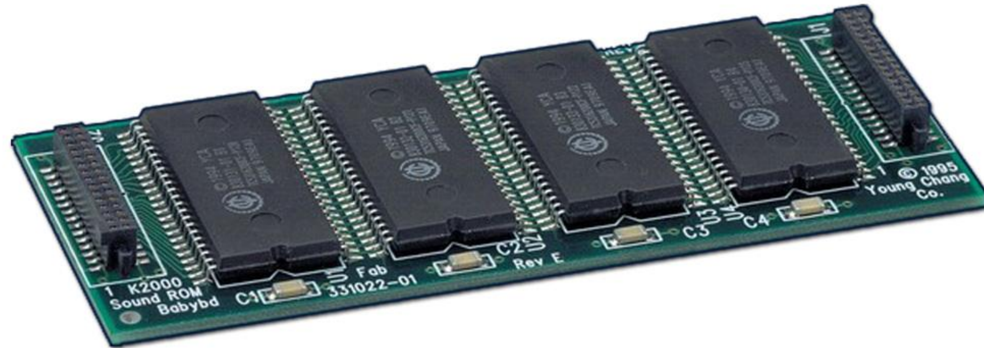
- GTX Titan X: 12GB
- GTX 980: 4GB
- Tesla K40: 12GB
- Tesla K20: 5GB

Calculating memory requirements

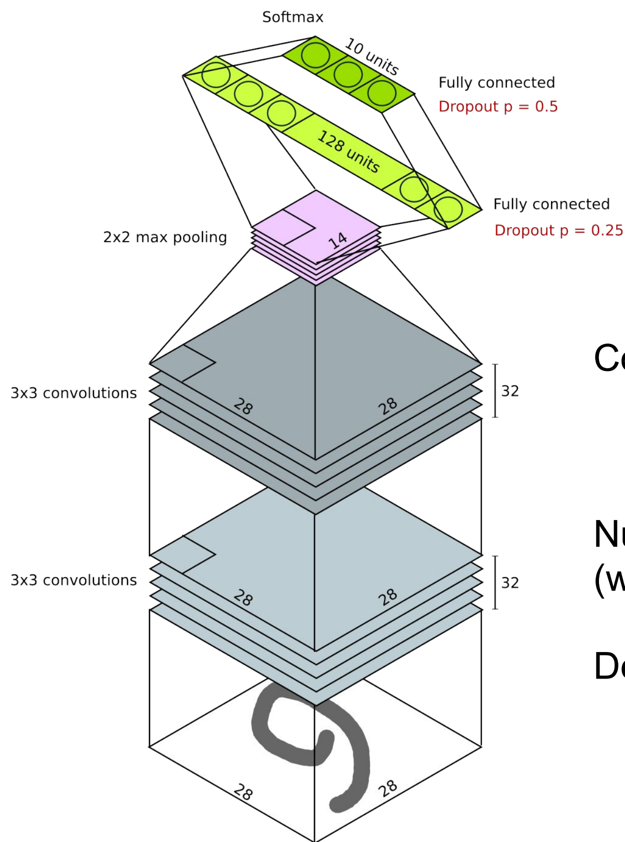
Often the size of the network will be practically bound by available memory

Useful to be able to **estimate memory requirements of network**

True memory usage depends on the implementation



Calculating the model size

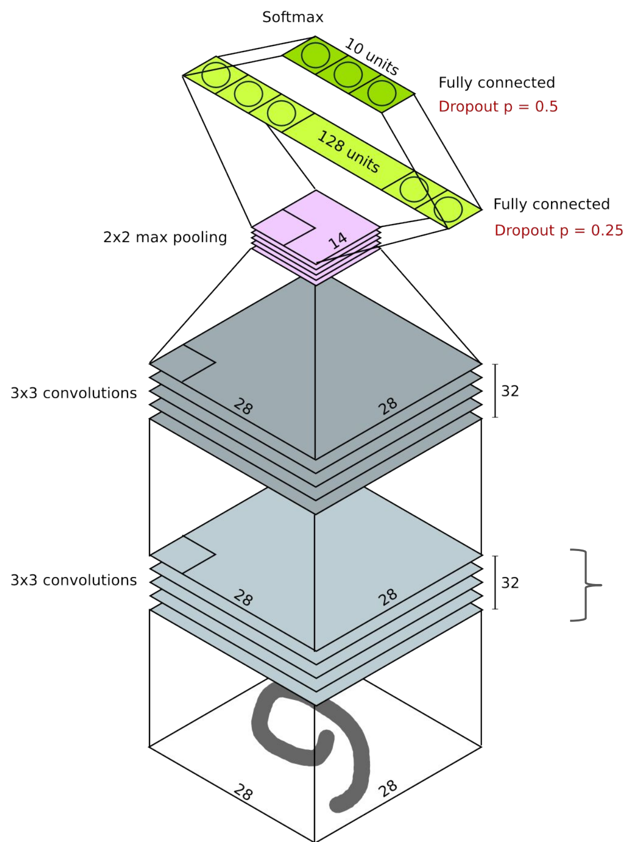


Conv layers:

Num weights on conv layers does not depend on input size (weight sharing)

Depends only on depth, kernel size, and depth of previous layer

Calculating the model size

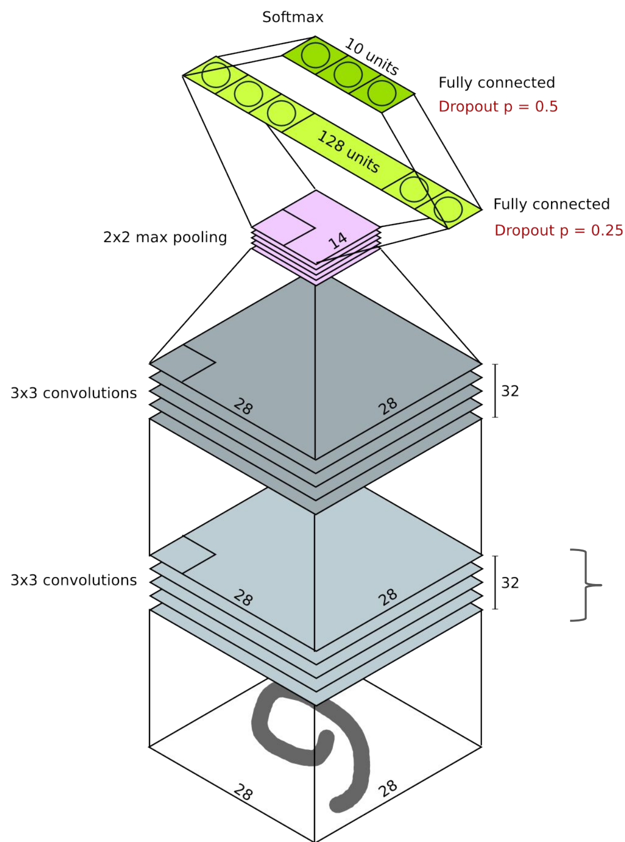


parameters

weights: $depth_n \times (kernel_w \times kernel_h) \times depth_{(n-1)}$

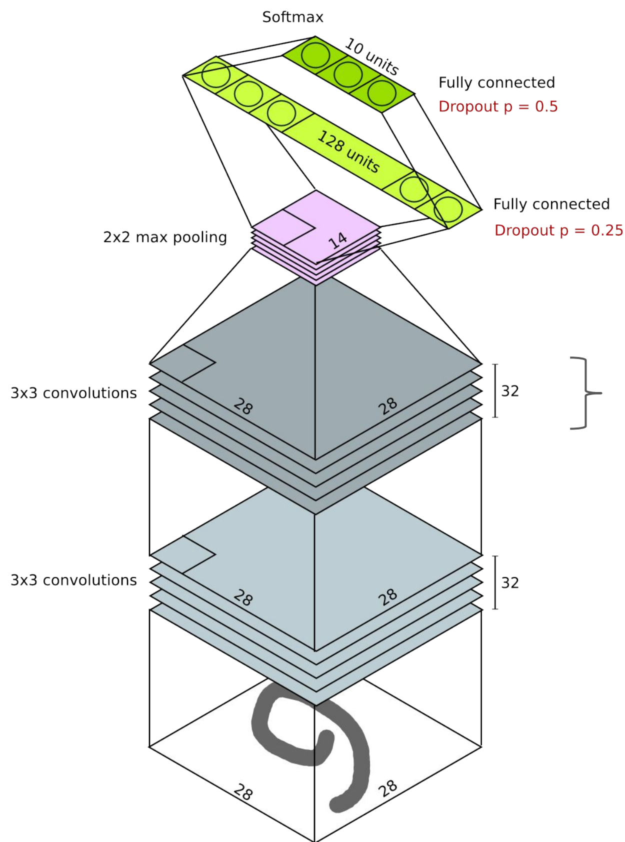
biases: $depth_n$

Calculating the model size




```
parameters
  weights: 32 x (3 x 3) x 1 = 288
  biases: 32
```

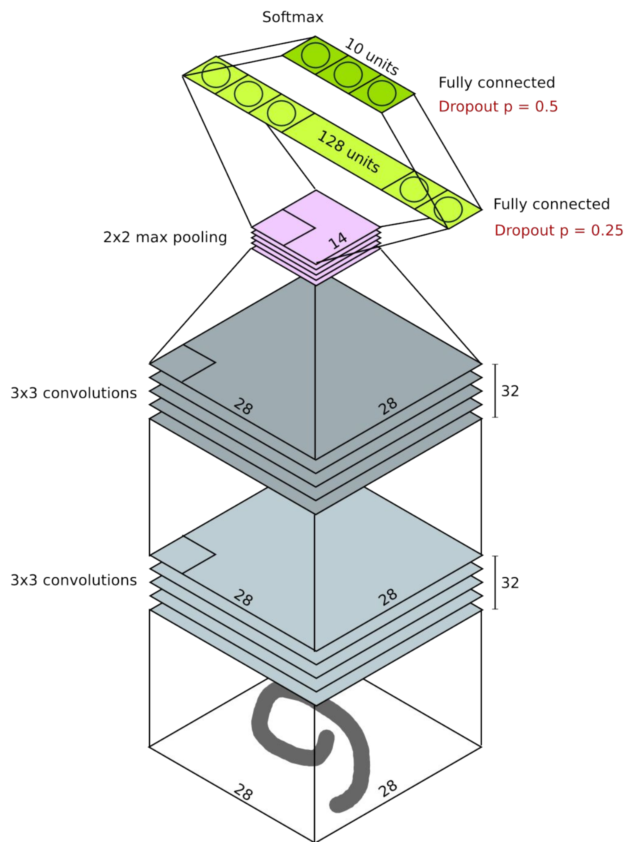
Calculating the model size



Pooling layers are parameter-free

```
parameters
  weights: 32 x (3 x 3) x 32 = 9216
  biases: 32
```

Calculating the model size

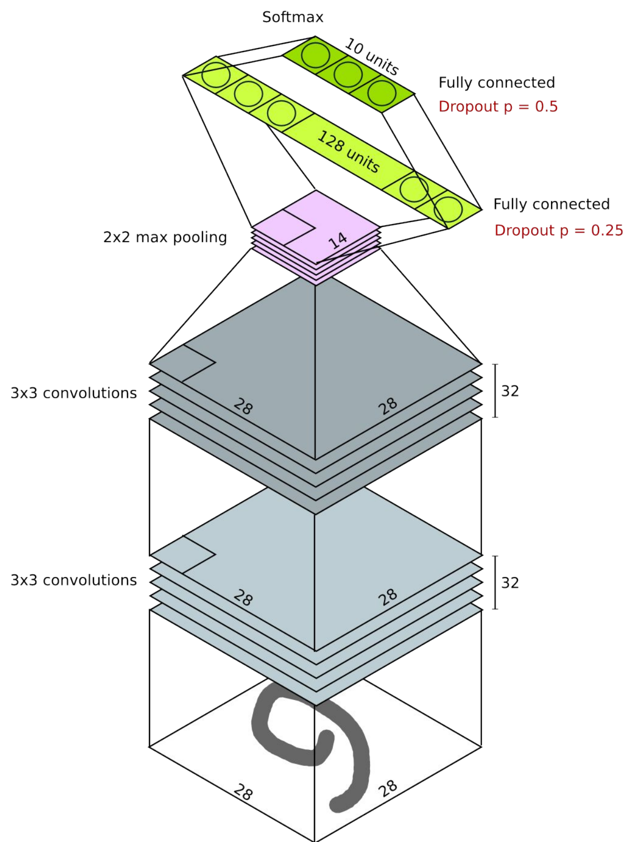


Fully connected layers

- #weights = #outputs x #inputs
- #biases = #outputs

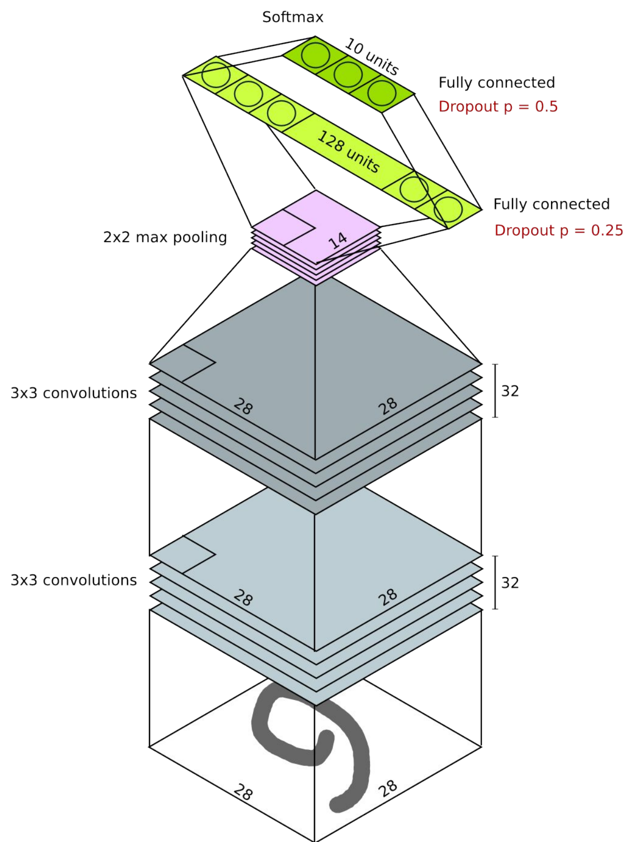
If previous layer has spatial extent (e.g. pooling or convolutional), then #inputs is size of flattened layer.

Calculating the model size



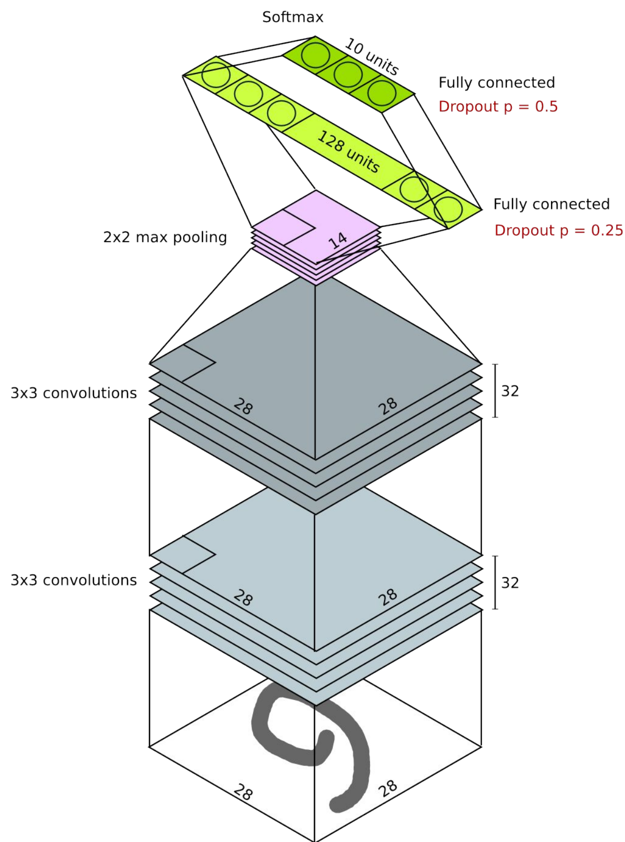

```
parameters
  weights: #outputs x #inputs
  biases: #inputs
```

Calculating the model size



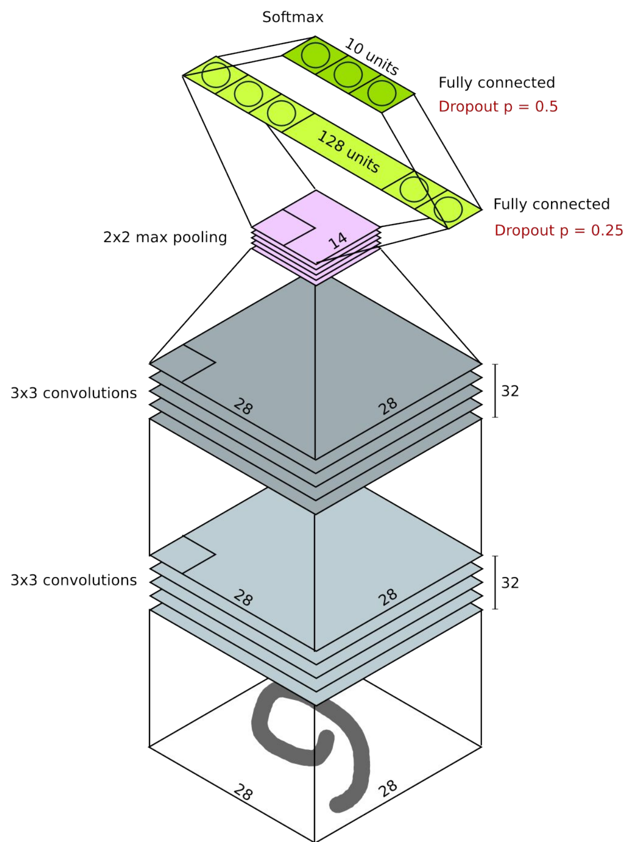

```
parameters
  weights: 128 x (14 x 14 x 32) = 802816
  biases: 128
```

Calculating the model size



```
parameters
  weights: 10 x 128 = 1280
  biases: 10
```

Total model size




```
parameters
  weights: 10 x 128 = 1280
  biases: 10

parameters
  weights: 128 x (14 x 14 x 32) = 802816
  biases: 128

parameters
  weights: 32 x (3 x 3) x 32 = 9216
  biases: 32

parameters
  weights: 32 x (3 x 3) x 1 = 288
  biases: 32

Total: 813,802
       ~ 3.1 MB (32-bit floats)
```

Layer blob sizes

Easy…

Conv layers: width x height x depth

FC layers: #outputs

32 x (14 x 14) = 6,272

32 x (28 x 28) = 25,088

Total memory requirements (train time)

Depends on implementation and optimizer

Memory for parameters

Memory for param gradients

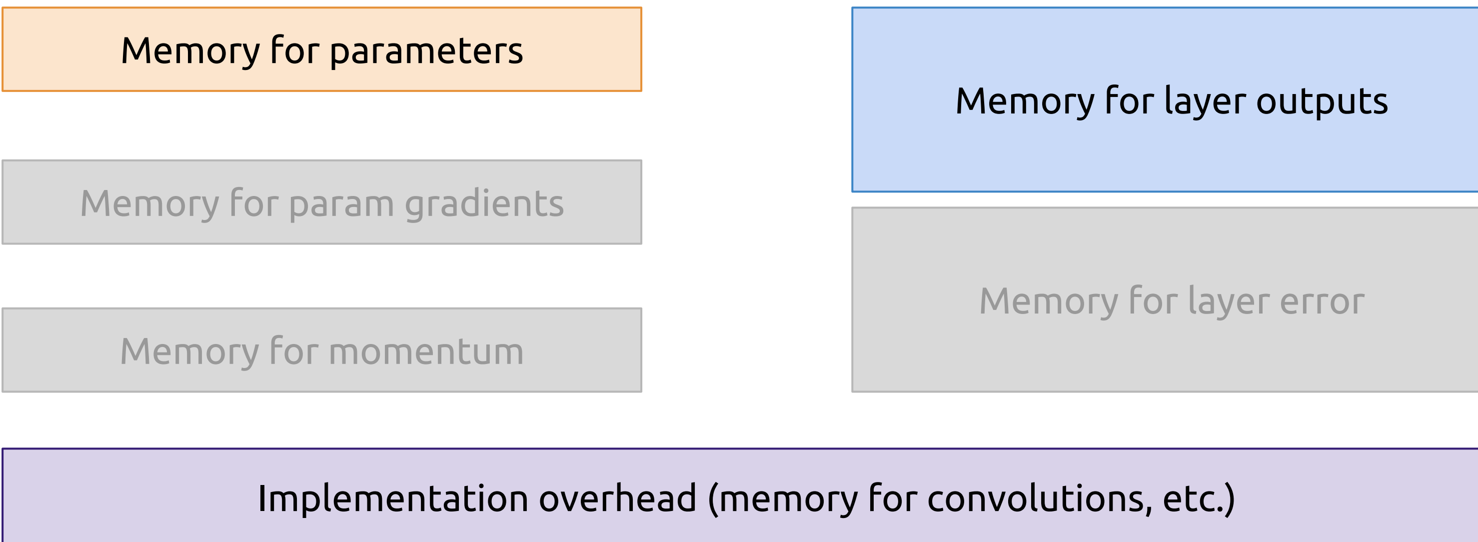
Memory for momentum

Memory for layer outputs

Memory for layer error

Implementation overhead (memory for convolutions, etc.)

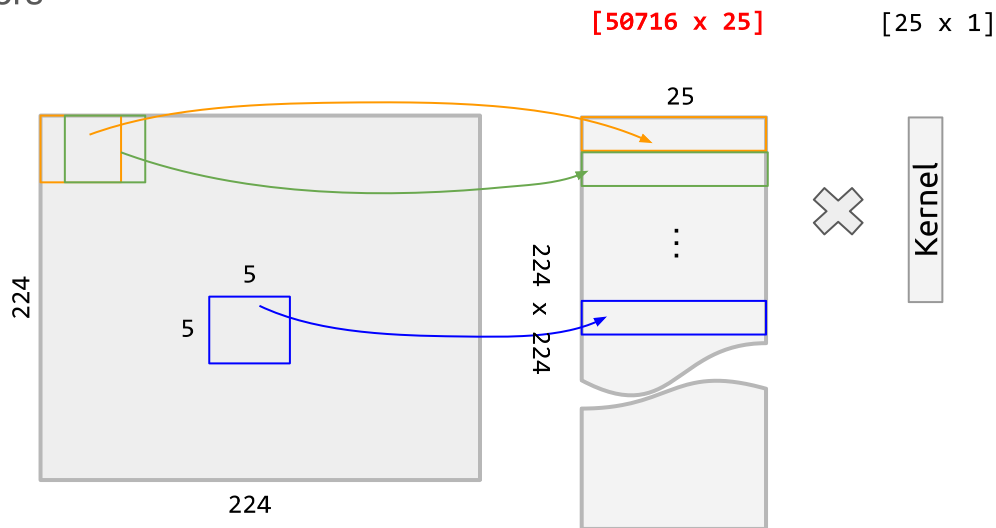
Total memory requirements (test time)



Memory for convolutions

Several libraries implement convolutions as matrix multiplications (e.g. caffe). Approach known as **convolution lowering**

Fast (use optimized BLAS implementations) but can **use a lot of memory,** esp. for larger kernel sizes and deep conv layers



cuDNN uses a more memory efficient method!

https://arxiv.org/pdf/1410.0759.pdf

Mini-batch sizes

Total memory in previous slides is for a single example.

In practice, we want to do mini-batch SGD:

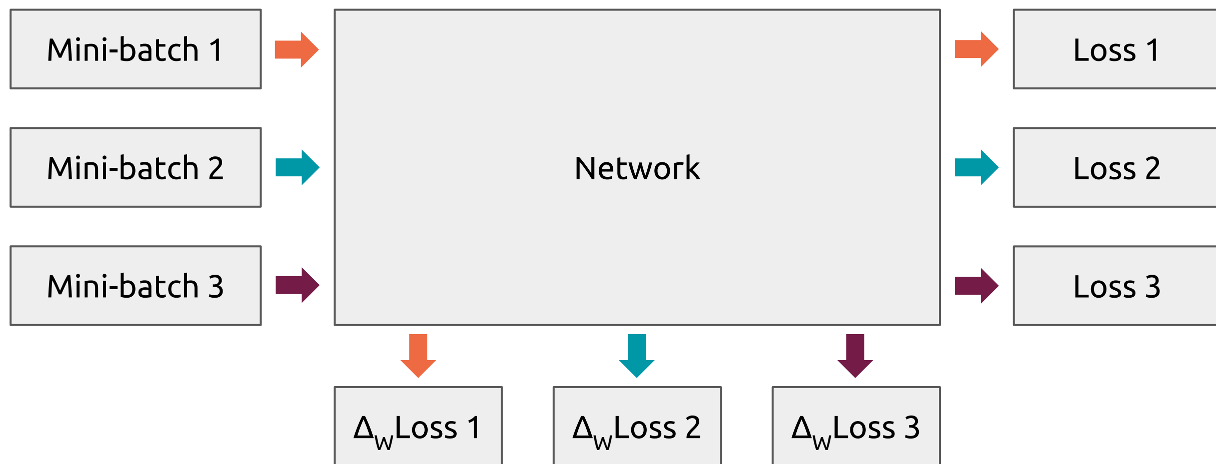
- More stable gradient estimates
- Faster training on modern hardware

Size of batch is limited by model architecture, model size, and hardware memory.

May need to reduce batch size for training larger models.

This may affect convergence if gradients are too noisy.

Gradient splitting trick



$$W_{t+1} = W_t + \frac{\alpha}{3} \sum_{n=1}^{3} \Delta_W L_n$$

Estimating computational complexity

Useful to be able to estimate computational complexity of an architecture when designing it

Computation in deep NN is dominated by multiply-adds in FC and conv layers.

Typically we estimate the number of FLOPs (multiply-adds) in the forward pass

Ignore non-linearities, dropout, and normalization layers (negligible cost).

Estimating computational complexity

Fully connected layer FLOPs

Easy: equal to the number of weights (ignoring biases)

= #num_inputs x #num_outputs

Convolution layer FLOPs

Product of:

- Spatial width of the map
- Spatial height of the map
- Previous layer depth
- Current layer deptjh
- Kernel width
- Kernel height

Example: VGG-16

Layer	H	W	kernel H	kernel W	depth	repeats	FLOP/s
input	224	224	1	1	3	1	0.00E+00
conv1	224	224	3	3	64	2	1.94E+09
conv2	112	112	3	3	128	2	2.77E+09
conv3	56	56	3	3	256	3	4.62E+09
conv4	28	28	3	3	512	3	4.62E+09
conv5	14	14	3	3	512	3	1.39E+09
flatten	1	1	0	0	100352	1	0.00E+00
fc6	1	1	1	1	4096	1	4.11E+08
fc7	1	1	1	1	4096	1	1.68E+07
fc8	1	1	1	1	100	1	4.10E+05
							1.58E+10

Bulk of computation is here

Effective aperture size

Useful to be able to compute **how far** a convolutional node in a convnet ***sees:***

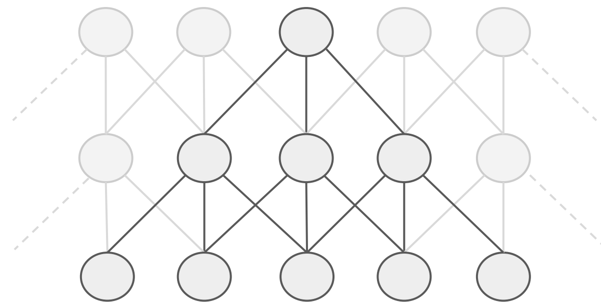
- Size of the input pixel patch that affects a node's output
- Known as the **effective aperture size**, **coverage, or receptive field size**

Depends on kernel size and strides from previous layers

- 7x7 kernel can see a 7x7 patch of the layer below
- Stride of 2 doubles what all layers after can see

Calculate recursively

$$\mathcal{A}_l = \mathcal{A}_{l-1} + (K_l - 1) \prod_{j=1}^{l} S_j$$



Summary

Shown how to estimate memory and computational requirements of a deep neural network model

Very useful to be able to quickly estimate these when designing a deep NN

Effective aperture size tells us how much a conv node can see. Easy to calculate recursively