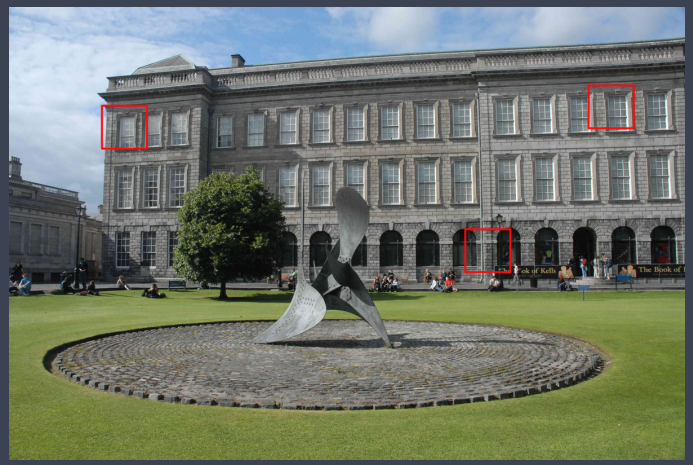
» Why Convolutions?

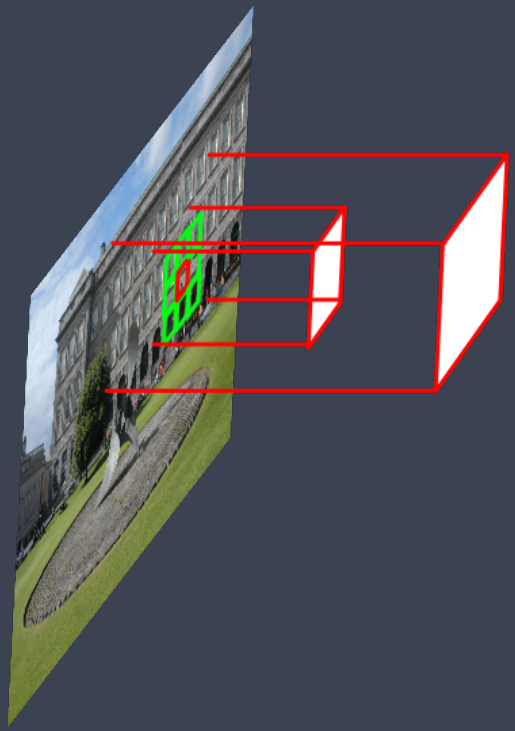
* *Sparsity*. Local information is often enough to detect basic features e.g. edges, no need to look at whole image
* *Reuse*. A feature extractor that works on one part of an image probably works on other parts, so can be re-used
* Result:
  * *Huge* reduction in #parameters to learn. e.g. one $3 \times 3 \times 3$ kernel has just 27 parameters, but with a separate kernel for every part of 300×300 image would need to learn $298 \times 298 \approx 89K$ kernels.

* Can also think of convolutional layer as an FC-layer with lots of constraints added $\rightarrow$ each output depends only on a small number of inputs (sparsity), same weights used for each output (reuse).

» Why Many Layers?

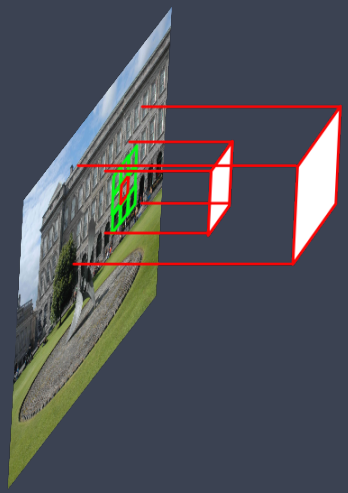
* Local information is often enough to detect *basic* features e.g. edges, no need to look at whole image $\rightarrow$ but what about larger features e.g. *a* building ?





* For kernel that acts directly on the input image the *receptive field* is just the kernel size e.g. 3×3 pixels $\rightarrow$ quite small
* Now receptive field of kernel in second layer is 3×3 kernels from first layer i.e. 9×9 pixels from image. Receptive field of kernel in third layer is 3×3 kernels from second layer i.e. 27×27 pixels from image.

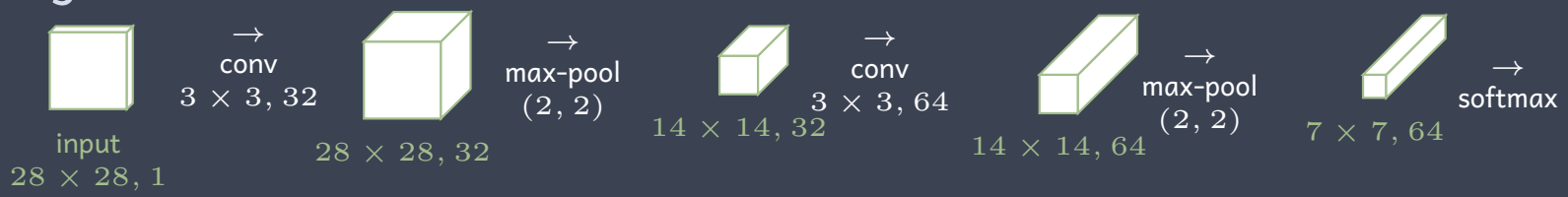
» Why Many Layers?



- First layer in ConvNet "sees" local image features
- Deeper layers combine local image features and so "see" larger features in image e.g. combine multiple lines/edges to recognise an eye
  $\rightarrow$ this is intuition behind using many layers in ConvNet i.e. "deep" learning.

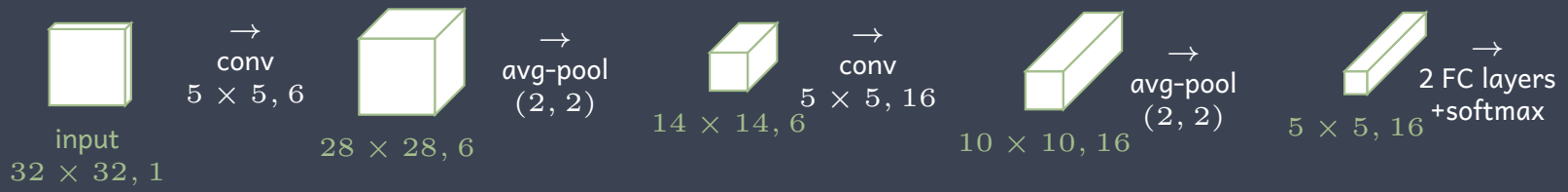
» Typical Modern CNN Structure

* As go through network #channels increases and height/width decreases $\rightarrow$ rule of thumb is to repeatedly halve height/width while doubling number of channels.
* E.g.



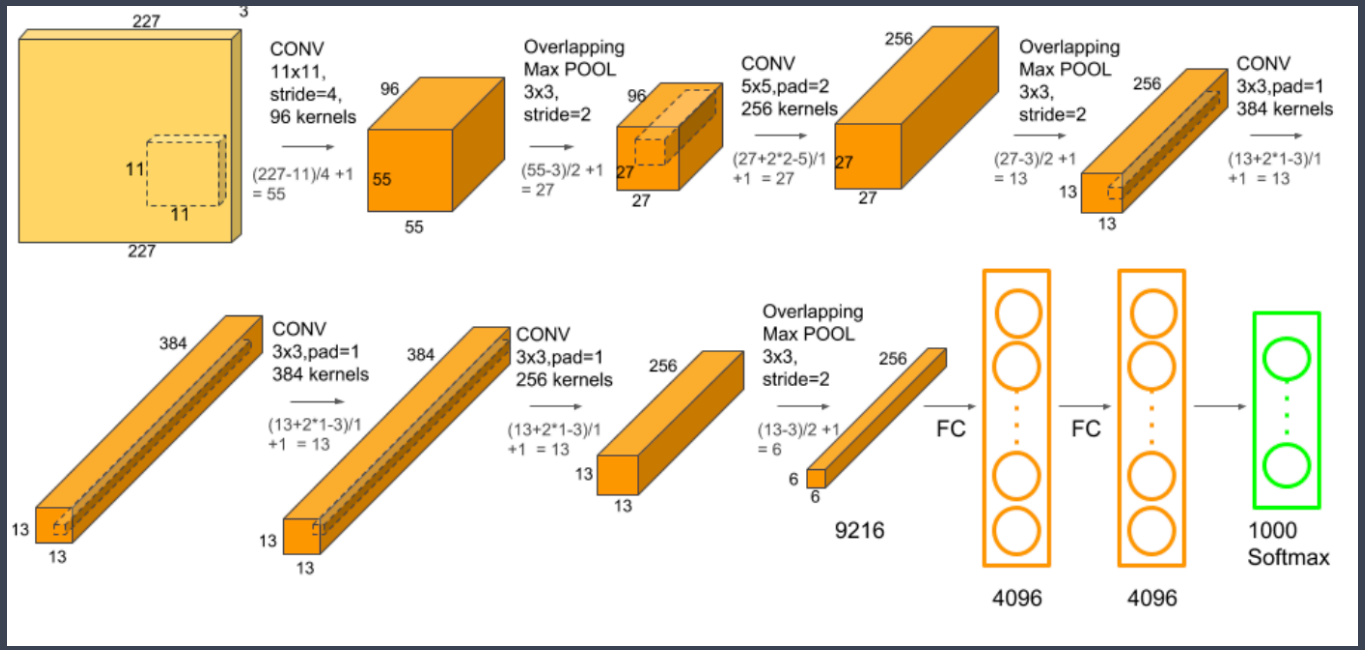
* A common pattern is $(Conv \times n + Maxpool) \times m$:
  * n convolutional layers with same padding, followed by a max-pool layer to downsample
  * Then repeat this block m times
  * Often 3×3 kernels in conv layers and $(2, 2)$ max-pooling
* Final/output layer is a fully-connected layer, often softmax
* Can think of this setup as convolutional layers being used to generate features that are then used as input to a logistic regression classifier (i.e. the softmax layer).

» LeNet



- Designed for MNIST digit dataset we looked at before
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278-2324, 1998
- Its pretty old, so uses choices that are now unpopular: average pooling, sigmoid and tanh activation. Uses relatively large 5×5 kernels. Quite small: network has about 60K parameters.
- The general pattern is still used though:
  - Repeated convolution+pooling/downsampling layers with an FC-layer as the output layer.
  - #channels increases, height/width decrease as move through network
- Achieved 99.05% accuracy on test data, state of the art is now about 99.8%. Recall in our example we managed 98.5% (by running SGD for more epochs it would get close to 99.8%).

» AlexNet

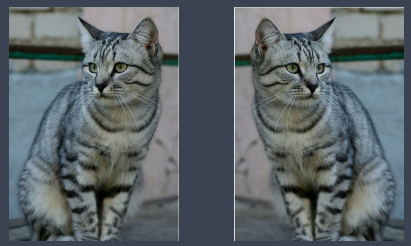


- Designed for ImageNet competition: 1.2M hand-labelled images, 1000 classes. Input is $227 \times 227 \times 3$ RGB image.

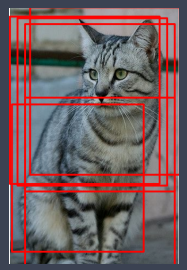
- A. Krizhevsky, I. Sutskever, G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
  - General structure is as for LeNet, but deeper (more layers):
    - Repeated convolution+pooling with an FC-layer as the output layer.
    - #channels increases, height/width decrease as move through network
  - Max-pooling, ReLU activation, much bigger network 60M parameters, dropout regularisation. Output is 2 FC-layers+softmax.
  - Performance of AlexNet kicked off renewed interest in deep learning $\rightarrow$ paper has 61K citations
  - Used GPUs but still took a week to train

» Data Augmentation

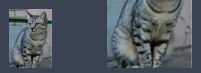
* AlexNet made heavy use of *data augmentation* → pretty much standard practice now
* Training big ConvNets needs lots of data. We can "create" more data by:
  * Reflection:



  * Random crops:

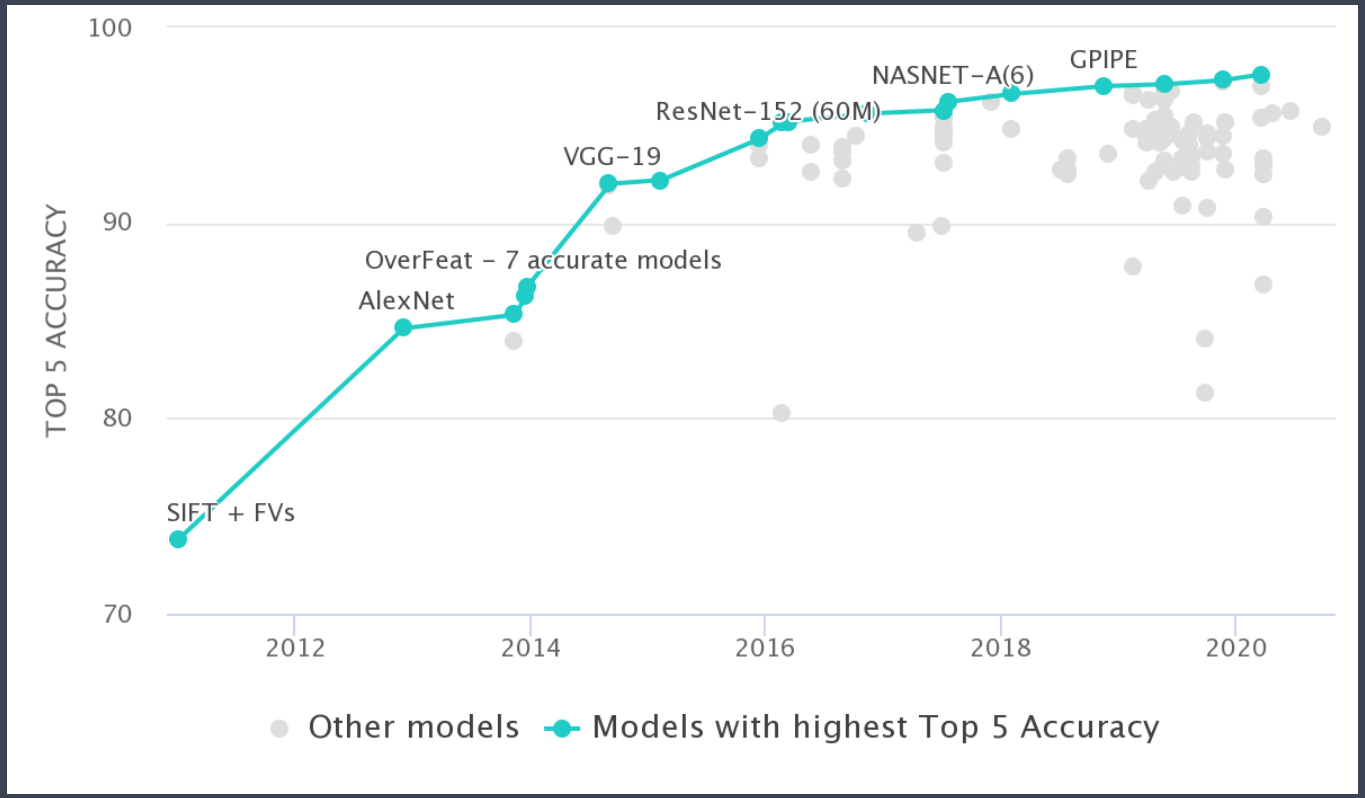


  .
  * Random scaling



  * Random adjustment of exposure and saturation
* Hopefully will make learned model more robust

» ImageNet Over Time[1]



* *AlexNet* accuracy was 10% better than next best at the time. Gain was attributed to #layers → 2012 was deep learning turning point
* *VGG19* achieved another 10% jump in performance in 2014
* *ResNet* achieved 4.49% error rate in 2015, beating human error rate of 5.1%
* Another important network is *GoogLeNet/Inception*, now Inception v4, but will leave you to look that up yourselves.

---

[1] https://paperswithcode.com/sota/image-classification-on-imagenet

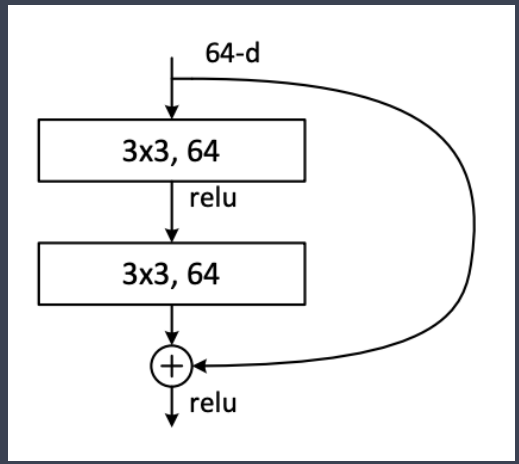
» VGG-16 and VGG-19

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

* K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015
* Used smaller 3×3 kernels (AlexNet uses 11×11 and 5 kernels) and added more layers to maintain large receptive field $\rightarrow$ now pretty standard, reduces number of model parameters
* AlexNet 8 layers (2012), VGG 19 layers (2014)
* Pattern of layers: $(\mathit{Conv} \times n + \mathit{Maxpool}) \times m$ with $n = 2 - 4$ and $m = 5$.
* ReLU activation, max-pooling
* 133-144M parameters (weights file is 500MB) $\rightarrow$ larger than later networks with more layers, makes VGG slow to train. NB: 120M parameters in the three FC-layers at output

» ResNet

* Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016
* AlexNet 8 layers (2012), VGG 19 layers (2014), Resnet 152 layers (2015)
* Similarly to VGG: uses 3×3 kernels and number of channels is doubled when height/width halved.
* Uses convolution layers with stride 2 for downsampling instead of max-pooling, and batch normalisation after each convolution.
* Introduced *skip* connections:



  with aim of reducing vanishing gradient problems (which become a big deal as network depth increases).
* Used 1×1 kernels to reduce number of parameters $\rightarrow$ despite ResNet having many more layers than VGG it has fewer parameters (60M vs 144M)
* ResNet-152 chieved 4.49% error rate in ImageNet, beating human error rate of 5.1%.

» Vanishing Gradients

Imagine you're trying to press a button using a long fishing rod ...

* Suppose rod is v heavy - its hard to move the far end at all $\rightarrow$ vanishing gradient
* Suppose rod is v flexible - even small movements of your hand cause large wobbles at far end $\rightarrow$ exploding gradient

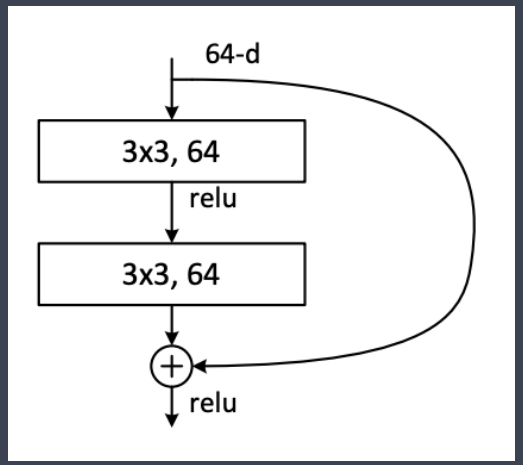
Using equations:

* Output of layer k is $a^{[k]}$ and weights are $w^{[k]}$
* With ReLU, roughly $a^{[k]} = w^{[k]} w^{[k-1]} w^{[k-2]} \ldots w^{[2]} w^{[1]} a^{[0]}$
* So derivative/gradient wrt weight $w^{[1]}$ is $w^{[k]} w^{[k-1]} w^{[k-2]} \ldots w^{[2]} a^{[0]}$
* If $w^{[k]} \leq b < 1$, $w^{[k-1]} \leq b < 1$ etc then derivative/gradient is $< b^{k-1} a^{[0]}$ and $b^{k-1} \rightarrow 0$ as k increases since $b < 1 \rightarrow$ vanishing gradient
* If $b > 1$ then $b^{k-1} \rightarrow \infty$ and k increases $\rightarrow$ exploding gradient

Vanishing/exploding gradients cause numerical problems for gradient descent. This becomes a serious problem as #layers becomes large.

» Vanishing Gradients

* ResNet idea to mitigate vanishing gradients is to add skip connections:



* Skip connections mean its always quite easy for a weight in an early layer to affect the output of a later layer, even one many layers away
* Roughly, $a^{[k]} = (w^{[k]}w^{[k-1]} + 1)(w^{[k-2]}w^{[k-3]} + 1)) \dots w^{[1]}a^{[0]}$
* Suppose weights are all really small, almost 0. Then $a^{[k]} \approx (0 + 1)(0 + 1)) \dots w^{[1]}a^{[0]} = w^{[1]}a^{[0]}$ and derivative wrt $w^{[1]}$ is $a^{[0]} \rightarrow$ no more vanishing
* Combine with initialising weights to reasonable values and use of SGD with adaptive step sizes e.g. adam
* Also uses batch normalisation after each convolution, but this seems to be less important
* This all remains not so well understood ... but it only seems important for really deep networks

» 1×1 Kernels

* 1×1 kernels are not too interesting if have just one input channel:

1	2	3	4	5
1	3	2	3	10
3	2	1	4	5
6	1	1	2	2
3	2	1	5	4

Input

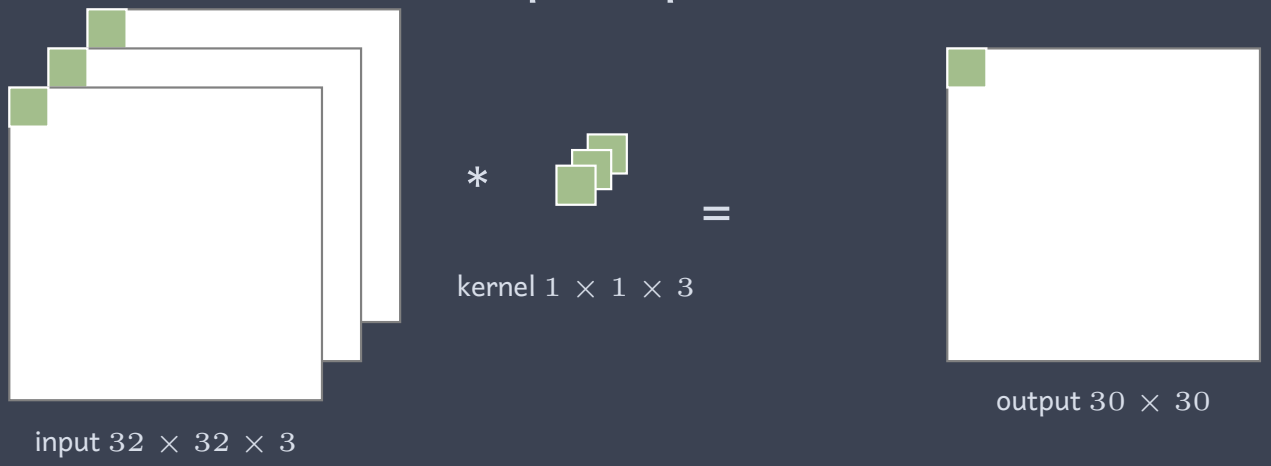
* 1 Kernel =

1	2	3	4	5
1	3	2	3	10
3	2	1	4	5
6	1	1	2	2
3	2	1	5	4

Output

just multiply each element of input by the kernel weight, in this case 1

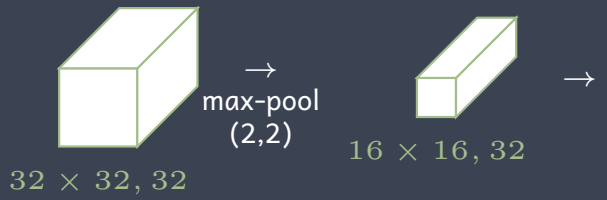
* But when have multiple input channels ...



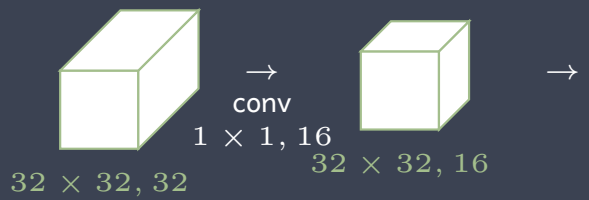
* ... 1×1 kernel takes a weighted sum across the input channels and then passes it through ReLU activation function.

» 1×1 Kernels

* We know that we can use max-pooling to reduce the height/width of an input, e.g. use (2,2) max-pooling to halve the height and width:



* But max-pooling leaves the number of channels unchanged. We can use a 1×1 kernel to reduce the number of channels, e.g. to shrink number of channels from 32 to 16:

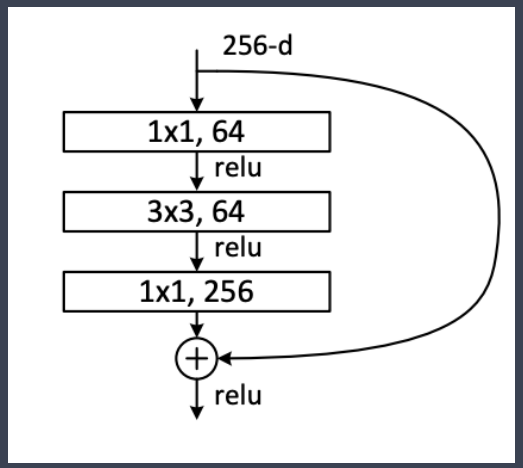


(remember when we write "conv $1 \times 1, 16$" we mean 16 $1 \times 1 \times 32$ kernels, we abbreviate $1 \times 1 \times 32$ to 1×1 since we know #channels in kernel must match #input channels)

* Of course can also use a 1×1 kernel to increase number of channels too e.g. by using conv $1 \times 1, 64$ in above example the output becomes $32 \times 32, 64$.

» 1×1 Kernels

* ResNet uses 1×1 kernels to reduce number of model parameters in v deep networks, e.g.:



* Shrinks 256 channels down to 64 channels using a 1×1 kernel, then applies 3×3 kernel to this smaller tensor and then expands back up to 256 channels using another 1×1 kernel.
* Without this shrinkage/expansion 256 $3 \times 3 \times 256$ kernels have 589K parameters. With shrinkage/expansion 64 $1 \times 1 \times 256$ kernels + 64 $3 \times 3 \times 64$ kernels +256 $1 \times 1 \times 64$ kernels combined have 69K parameters i.e. about $10\times$ fewer.

» Transfer Learning

Basic idea: train a ConvNet on a large and general image dataset, then use the ConvNet as a fixed feature extractor for a new task:

1. Take an already trained ConvNet (training takes *ages* $\rightarrow$ reuse a good idea!)
2. Remove the fully-connected output layer(s)
3. Add a new output layer (e.g. logistic regression or SVM) which takes the ConvNet outputs as its inputs, then train this new output layer for task at hand.

E.g. take VGG-16:

* Flatten output of last maxpool layer to get a 4096×1 vector
* Use as input to new logistic regression/softmax layer.
* Train weights in this new output layer while *holding VGG-16 weights constant*
* Since we hold the VGG-16 layers constant we're just using VGG-16 to map from the input to a 4096×1 feature vector. Can then use these input features with any other model.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

» Transfer Learning

A variant: also fine-tune the ConvNet weights for new task. When does this make sense?

* New dataset is small and similar to dataset used to train ConvNet $\rightarrow$ probably not enough data to re-train a big ConvNet
* New dataset is large and similar to dataset used to train ConvNet $\rightarrow$ fine-tuning of ConvNet might be useful. Usually use a small step-size/learning rate so changes made are kept small.
* New dataset is small and v different from dataset used to train ConvNet $\rightarrow$ hmm
* New dataset is large and v different from to dataset used to train ConvNet $\rightarrow$ re-training fo ConvNet using pre-trained weights as initial values might be useful
* Some further reading:
    * CNN Features off-the-shelf: an Astounding Baseline for Recognition https://arxiv.org/abs/1403.6382
    * How transferable are features in deep neural networks? http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf