Machine Learning

Building and improving learning architectures

Dr Chris Willcocks
Department of Computer Science
Durham University

Lecture Overview

Durham University

Today's lecture

- Convolutional layers
  - Convolution arithmetic
  - Stride, padding, and dilations
  - Common patterns
  - Transpose convolutions
- Pooling layers
  - Mean pool, max pool, adaptive pooling, interpolation
- Regularisation techniques
  - L1, L2
  - Dropout
  - Normalisation layers
- Residual learning

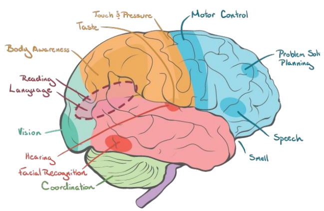
Materials:

https://github.com/cwkx/ml-materials

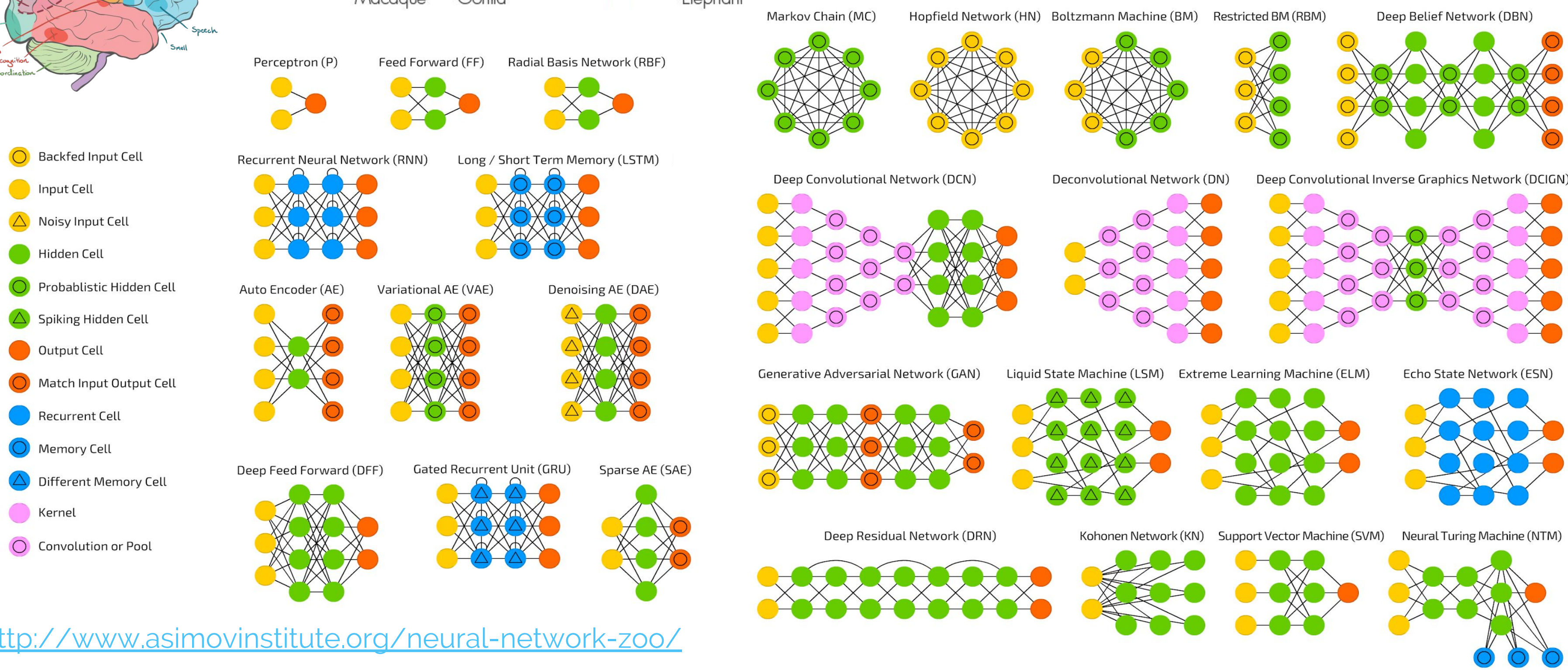
https://colab.research.google.com/gist/cwkx/1a4da17e2e9c1081f93631f5b51e5bae/convnet.ipynb

Building and improving architectures

https://faculty.washington.edu/chudler/compare2.html



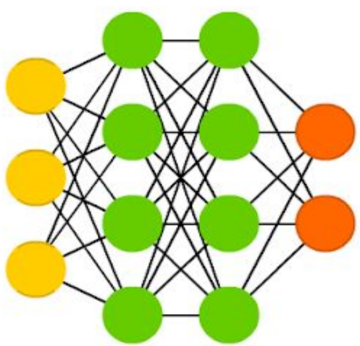
- Architecture is very important to function
  - We'll look today at main components



http://www.asimovinstitute.org/neural-network-zoo/

Fully-connected layer properties

- Big inputs =
  lots of parameters
- Expensive
  - Memory
  - Processing
- Fixed



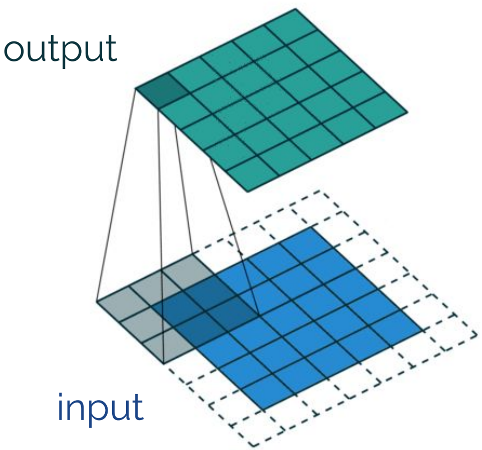
```
(base)  chris@chris-office  ~  master ●  ipython
Python 3.7.1 (default, Oct 23 2018, 19:19:42)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.1.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import torch

In [2]: import torch.nn as nn

In [3]: x = torch.zeros(1, 3, 1024, 1024)

In [4]: l1 = nn.Linear(1024*1024*3, 512)

In [5]: l1.weight.view(-1).size()
Out[5]: torch.Size([1610612736])
```

- **6 GB of memory if float 32-bit**
- **Just 1 layer**
- **And that's only 512 output features...**

Convolutional layers (1D, 2D, 3D)

Durham University

Kernel slides a window over input...



learn these weights

$$* \begin{bmatrix} -1, 0, 1 \\ -2, 0, 2 \\ -1, 0, 1 \end{bmatrix} =$$

- http://setosa.io/ev/image-kernels/ - interactive 2D demo
- https://github.com/vdumoulin/conv_arithmetic - detailed diagrams
- https://arxiv.org/pdf/1603.07285.pdf - great paper (diagrams above)

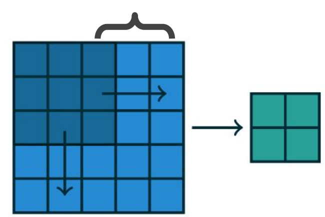
Convolution arithmetic

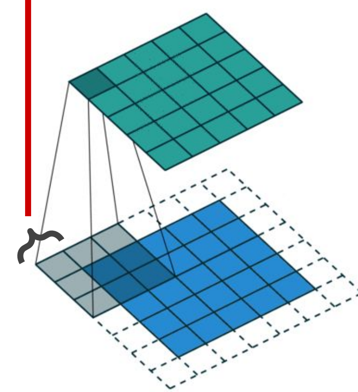
- More advanced usage helps us control the shape of the architecture
  - Kernel size, stride, padding, and dilations

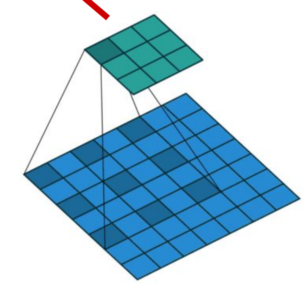
```
nn.Conv2d(in_channels, out_channels, kernel_size, stride=2, padding=1, dilation=2, groups=1, bias=True)
```

e.g. on first layer { R G B }

Ridge 90°
Ridge 33°
Ridge 45°
Blob
Texture
...







Common Patterns

- Input: (N, C_{in}, L_{in}) ← Spatial dimension(s)
- Output: (N, C_{out}, L_{out}) where

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1 \right\rfloor$$

```
nn.Conv2d(in_channels=3, out_channels=14, kernel_size=3, stride=1, padding=1, dilation=1)
```

[B×3×64×64] → [B×14×64×64] - *don't change spatial dimension*

```
nn.Conv2d(in_channels=3, out_channels=1, kernel_size=1, stride=1, padding=0, dilation=1)
```

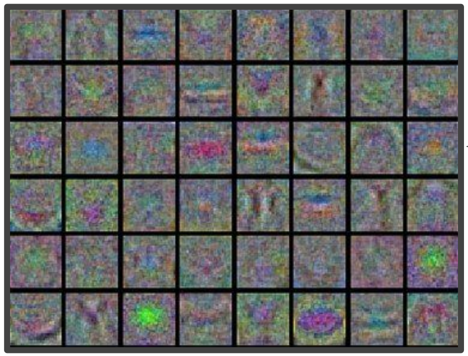
[B×3×64×64] → [B×1×64×64] - *don't change spatial dimension*

```
nn.Conv2d(in_channels=3, out_channels=14, kernel_size=4, stride=2, padding=1, dilation=1)
```

[B×3×64×64] → [B×14×32×32] - *half spatial dimension*

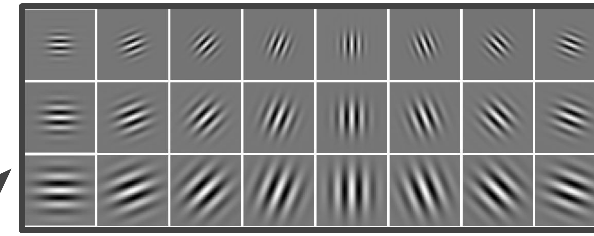
First layer weights respond to edges, color variation, texture...

Durham University



Unconverged

First layer looks a bit like Gabor filters



Converged Krizhevsky et al

Color Boundary

Texture

Blob

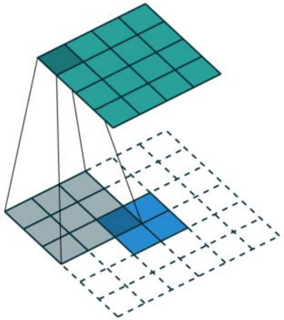
Transpose convolutions

Durham University

- Input: (N, C_{in}, L_{in})
- Output: (N, C_{out}, L_{out}) where

$$L_{out} = (L_{in} - 1) \times \text{stride} - 2 \times \text{padding} + \text{kernel\_size} + \text{output\_padding}$$

- "The gradient of a convolution with respect to its input"
- Useful for <u>going back up</u> (increasing spatial dimension)
  - E.g. generators and decoder components



```
nn.Conv2d(in_channels=3, out_channels=14, kernel_size=4, stride=2, padding=1, dilation=1)
```

[B×3×64×64] → [B×14×128×128] - *doubled the spatial dimension*

Other ways to change spatial dimension

- In contrast to fully-convolutional (recommended)
- Other ways to change the spatial dimension
  - Mean pool
  - Max pool
  - Interpolation (can go up or down)
- Adaptive pooling (can go up or down)
  - The kernel can be altered based on the size of the input
- Common approaches
  - Can implemented in the forward pass using **nn.functional** interface as these methods don't require parameters

```
nn.AdaptiveAvgPool2d((5,71))
```

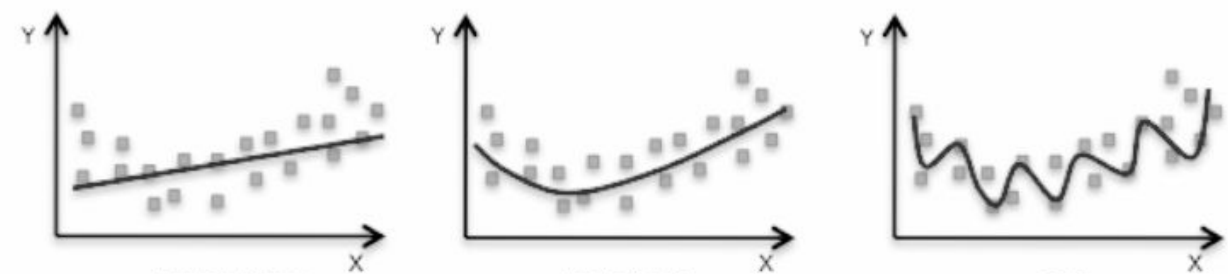
[B×3×64×64] → [B×3×5×71] - *output is fixed*

```
nn.AdaptiveAvgPool2d((1,1))
```

[B×3×27×19] → [B×3×1×1] - *good for last layer*

Regularisation

- "Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."
  - Goodfellow's book, Section 5.2.2
- Approaches
  - Add "error", e.g.:
    - Dataset augmentation
    - Dropout
  - Constrain manifold, e.g.:
    - Smooth the optimization landscape

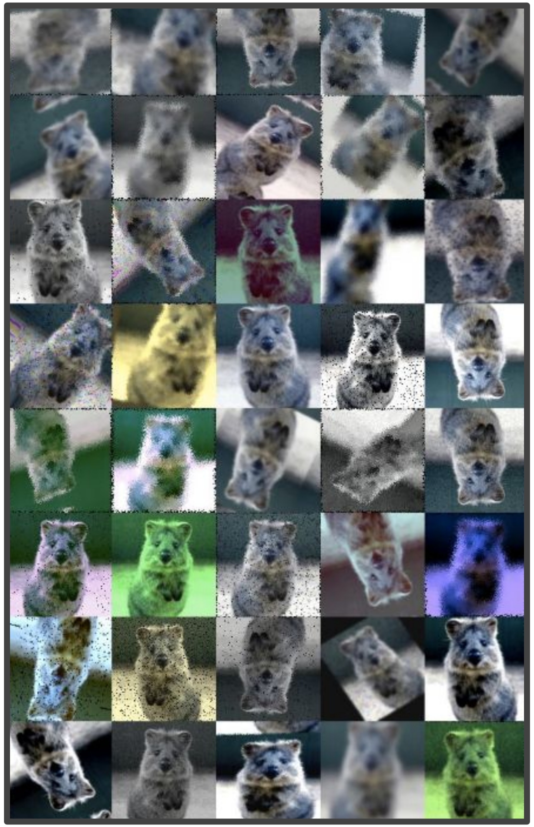


Underfitting   Good fit   Overfitting

Augmentation

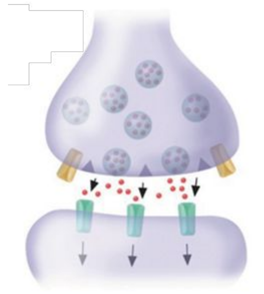
Durham University

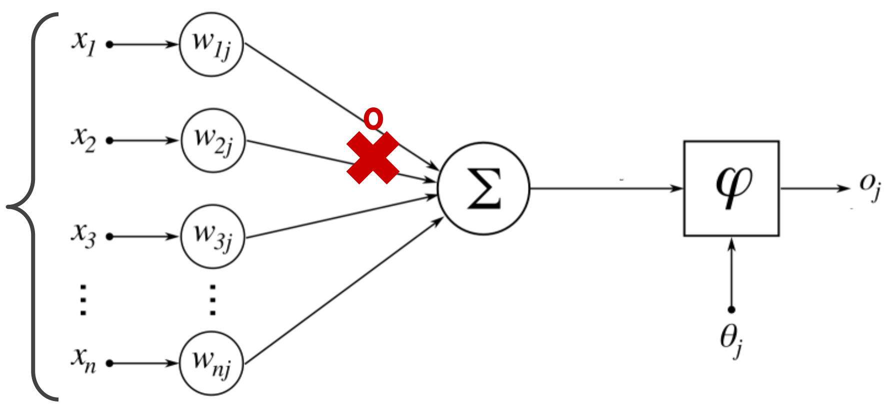
- A form of adding prior knowledge to the model, e.g. fundamentally these are all recognizable images of dogs
- Examples:
    - Random rotations
    - Random horionztal flips
    - Random blur
    - Random noise
- Would 180 degree rotations be suitable data augmentation for MNIST?



L1 and L2 regularisation

Recall Hebbian theory and synaptic plasticity...





Input signals

It'd be nice if some of the weights could → 0

L1 regularizer is not differentiable everywhere but allows sparsity

$$\mathcal{L}' = \mathcal{L} + \lambda \sum_i |w_i| = \mathcal{L} + \lambda \|\mathbf{w}\|_1$$

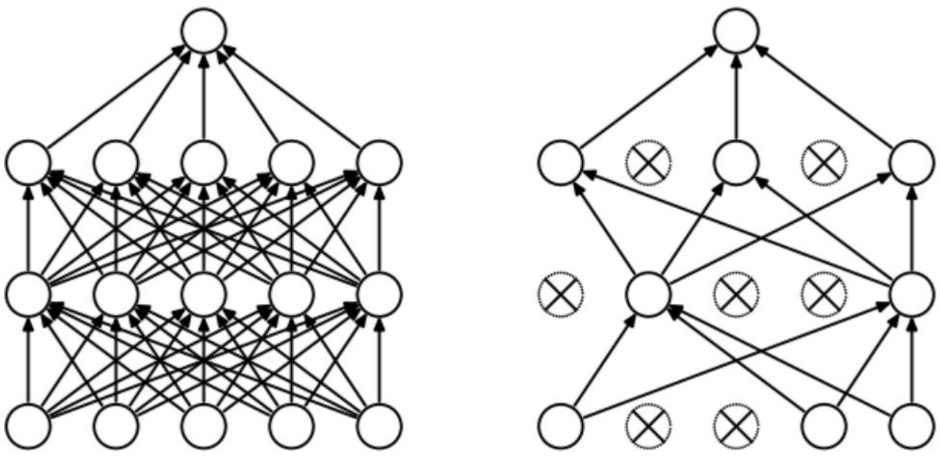
L2 regularizer is convex and differentiable everywhere

$$\mathcal{L}' = \mathcal{L} + \lambda \sum_i w_i^2 = \mathcal{L} + \lambda \|\mathbf{w}\|_2^2$$

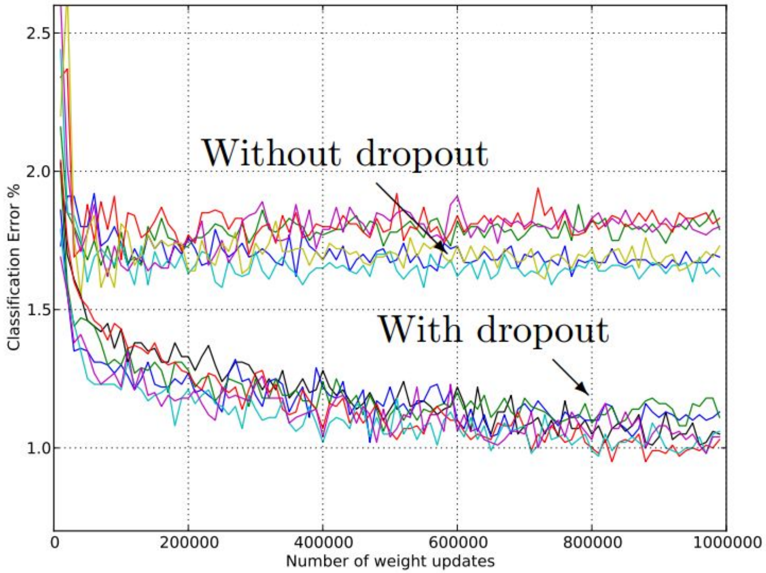
Stochastic Regularisation
Dropout & Label Noise

Durham University

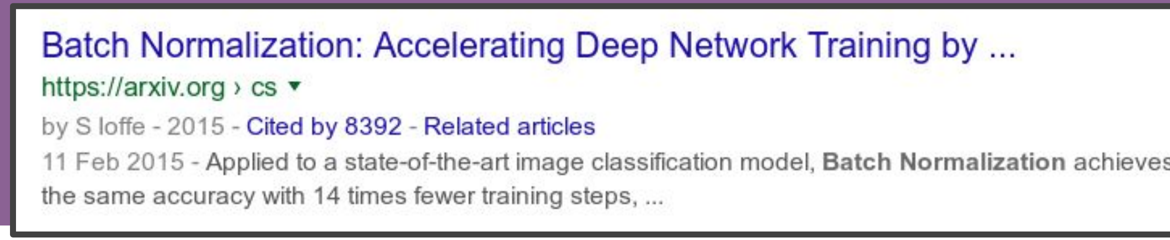
- Each hidden unit is set to zero with some probability (e.g. 0.2)
- Cannot rely on any one weight.
  - Spreads out its weights
  - Shown formally to have similar effect to l2 regularisation



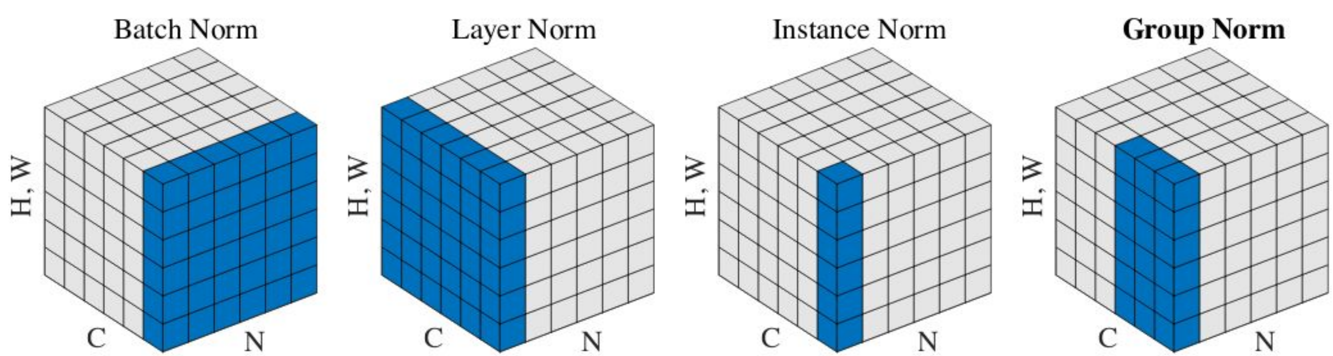
(a) Standard Neural Net (b) After applying dropout.



Normalisation


Batch Normalization: Accelerating Deep Network Training by ...
https://arxiv.org › cs ▾
by S Ioffe - 2015 - Cited by 8392 - Related articles
11 Feb 2015 - Applied to a state-of-the-art image classification model, **Batch Normalization** achieves the same accuracy with 14 times fewer training steps, ...

- Batch normalisation in particular is a very important technique
- Recently it seems main results come from smoothing optimisation landscape rather than preventing covariance shift
  - https://arxiv.org/abs/1805.11604
- Other normalisation layers (useful in different scenarios):

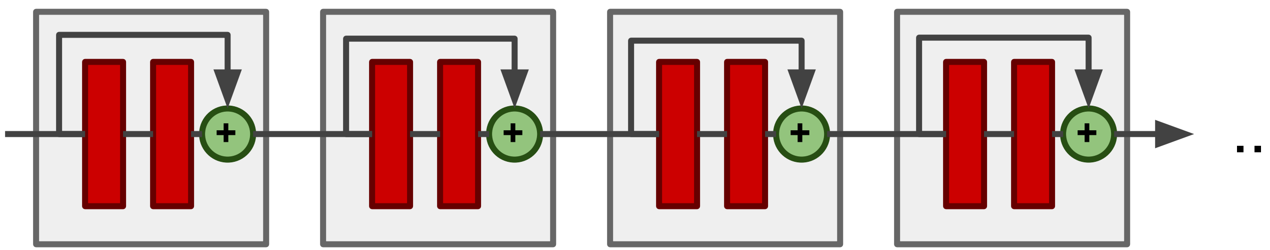


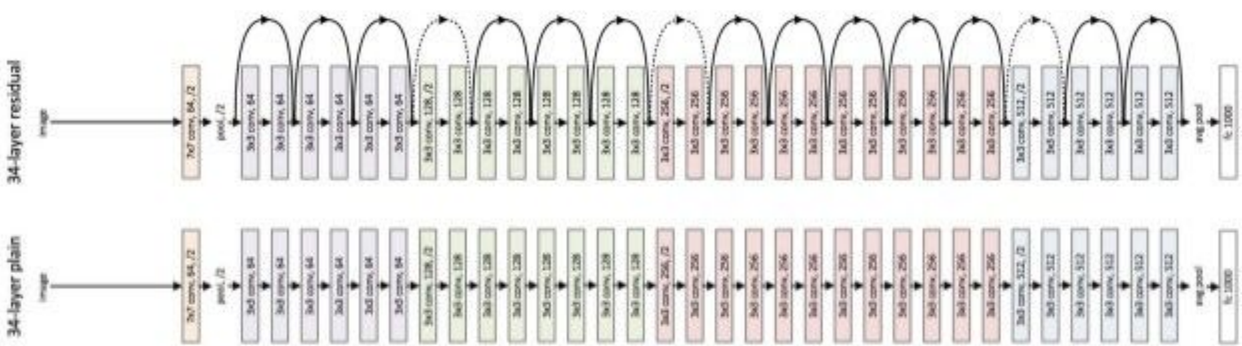
Different normalisation layers (figure from Group Norm paper: https://arxiv.org/pdf/1803.08494.pdf
).
N=batch axis, C=channel axis, H,W = spatial axes. Blue voxels are normalized to have same mean and variance, computed by aggregating values of those voxels.

Deep Residual Learning

Deep Residual Learning
https://arxiv.org › cs ▾
by K He - 2015 - Cited by 17392 -
10 Dec 2015 - We present a resid
evidence showing that these resid

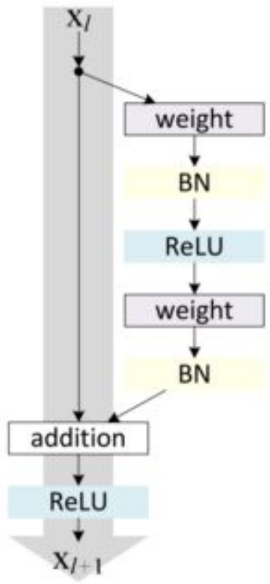
- What if we want to go really deep?
  - Vanishing gradient problem
- Residual blocks





(a) original

Example Convnet Implementation




```
class ResidualBlock(nn.Module):
  def __init__(self, in_features):
      super(ResidualBlock, self).__init__()

      conv_block = [  nn.Conv2d(in_features, in_features, 3, stride=1, padding=1, bias=False),
                      nn.BatchNorm2d(in_features),
                      nn.ReLU(inplace=True),
                      nn.Conv2d(in_features, in_features, 3, stride=1, padding=1, bias=False),
                      nn.BatchNorm2d(in_features) ]

      self.conv_block = nn.Sequential(*conv_block)

  def forward(self, x):
      return x + self.conv_block(x)
```

Take away points

- Convolutions are great when you have **spatial** or **temporal coherence**
- Enforcing **smoothness** (where applicable) generally improves stability and test accuracy
  - Smoothness as regularizer
  - Smoothness during data transformation
- Make sure you only use dropout or other regularization techniques **<u>if you are sure you are overfitting</u>**
- The shape of the architecture is important to the application
- Residual layers help go deep without vanishing gradients

Next week:

- Looking at mathematics of the backpropagation algorithm