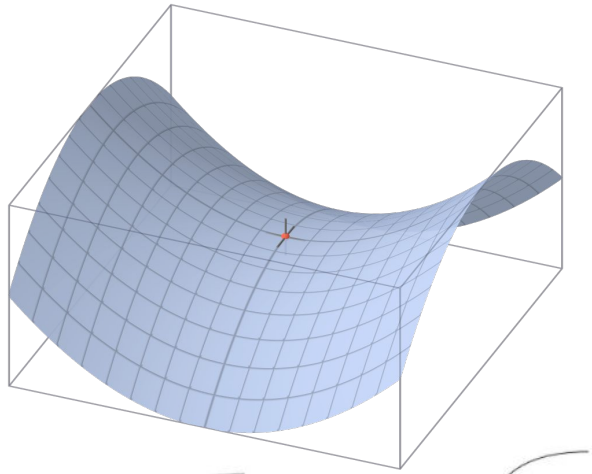
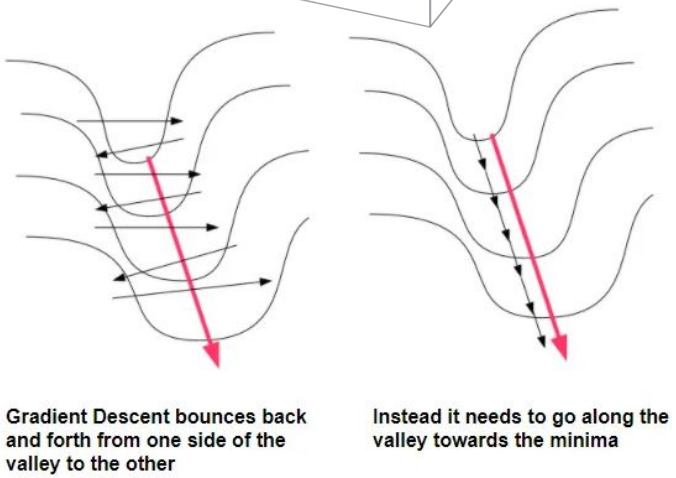
Badly conditioned regions

- regions of the loss function where first-order methods struggle, like saddle points and ravines
- solutions to improve convergence:
  - use second-order methods (slow)
  - change the model and the data to reduce badly conditioned regions
  - alter first-order methods to fare better in badly conditioned regions
- as the approximation error is influenced by the Hessian, conditioning is often expressed in terms of its eigenvalues





https://upload.wikimedia.org/wikipedia/commons/4/40/Saddle_point.png

https://towardsdatascience.com/neural-network-optimizers-made-simple-core-algorithms-and-why-they-are-needed-7fd072cd2788

Momentum

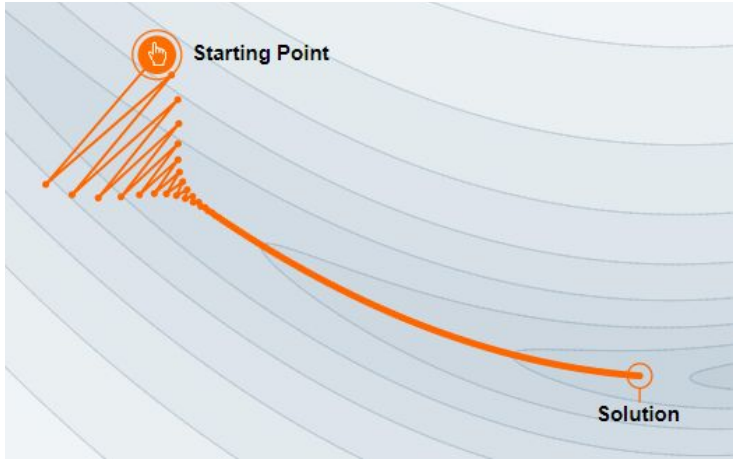
- SGD tends to get stuck in badly conditioned regions
- momentum avoids this by taking a weighted average of previous steps

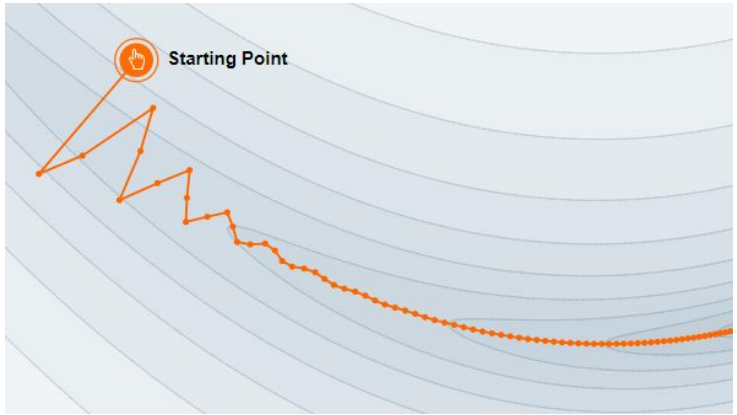
SGD step:

$$v_i = -\eta \nabla \mathcal{L}(w_i)$$

SGD with momentum step:

$$v_i = \beta v_{i-1} - \eta \nabla \mathcal{L}(w_i)$$

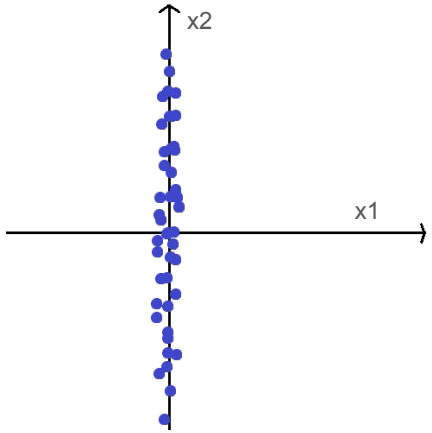


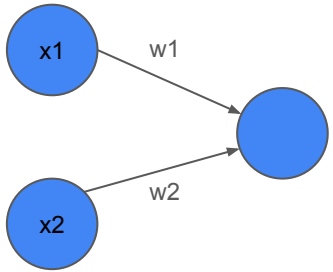


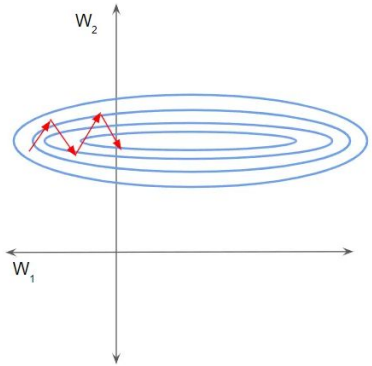
https://distill.pub/2017/momentum/

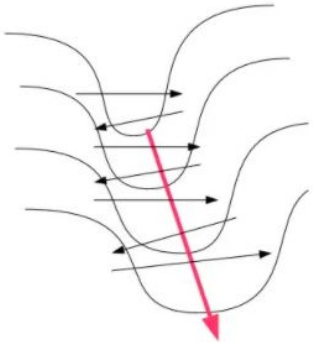
Input normalization

- inputs of drastically different scales create ravines spontaneously
- x2 has large scale -> gradient along w2 is oversensitive, which impedes the optimization along w1
- to prevent this, neural network inputs are usually normalized to zero mean and unit variance
- with images, the improvement is often negligible









https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Initialization

- the choice of the starting point affects the optimization results
- random (uniform) starting weights typically lead to bad results
- idea: initialize weights with a normal or uniform distribution such that the input is transferred into an output of the same variance
- that way, the scale of gradients stays the same across all layers (at least at the beginning of training)

ReLU: Kaiming/He normal distribution

$$W \sim \mathcal{N}\left(0, \frac{2}{n^l}\right)$$

sigmoid: Xavier/Glorot normal distribution

$$W \sim \mathcal{N}\left(0, \frac{1}{n^l}\right)$$

where n^l is the number of inputs

https://distill.pub/2017/momentum/

Momentum & good initialization enabled 1st-order methods

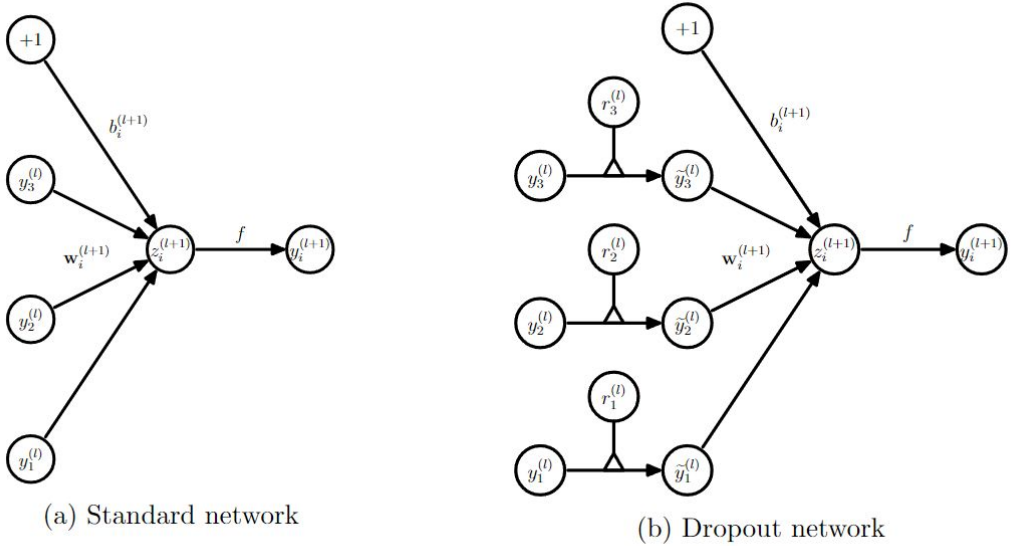
On the importance of initialization and momentum in deep learning

Proceedings of the 30th International Conference on Machine Learning 2013 · Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton · Edit social preview

Deep and recurrent neural networks (DNNs and RNNs respectively) are powerful models that were considered to be almost impossible to train using stochastic gradient descent with momentum. In this paper, we show that when stochastic gradient descent with momentum uses a well-designed random initialization and a particular type of slowly increasing schedule for the momentum parameter, it can train both DNNs and RNNs (on datasets with long-term dependencies) to levels of performance that were previously achievable only with Hessian-Free optimization. We find that both the initialization and the momentum are crucial since poorly initialized networks cannot be trained with momentum and well-initialized networks perform markedly worse when the momentum is absent or poorly tuned. Our success training these models suggests that previous attempts to train deep and recurrent neural networks from random initializations have likely failed due to poor initialization schemes. Furthermore, carefully tuned momentum methods suffice for dealing with the curvature issues in deep and recurrent network training objectives without the need for sophisticated second-order methods.

Dropout

- introduced as a technique to reduce overfitting
- sets a random subset of activations of a chosen layer to zeros during training
- this (in theory) forces the next layer to learn multiple ways of making the prediction
- during inference, all activations are present and rescaled
- significantly increases training time. Used mostly before FC layers



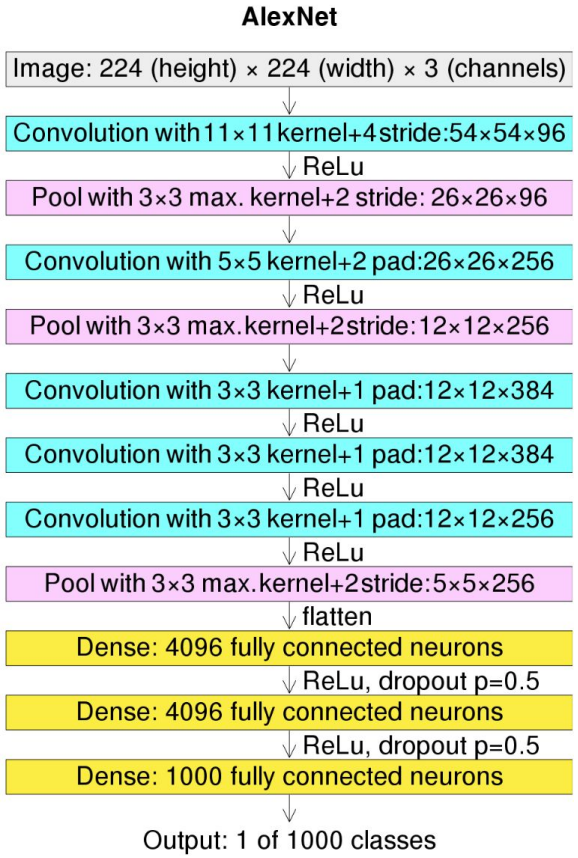
(a) Standard network (b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

Dropout: A Simple Way to Prevent Neural Networks from Overfitting, N. Srivastava et al. (2014)

AlexNet

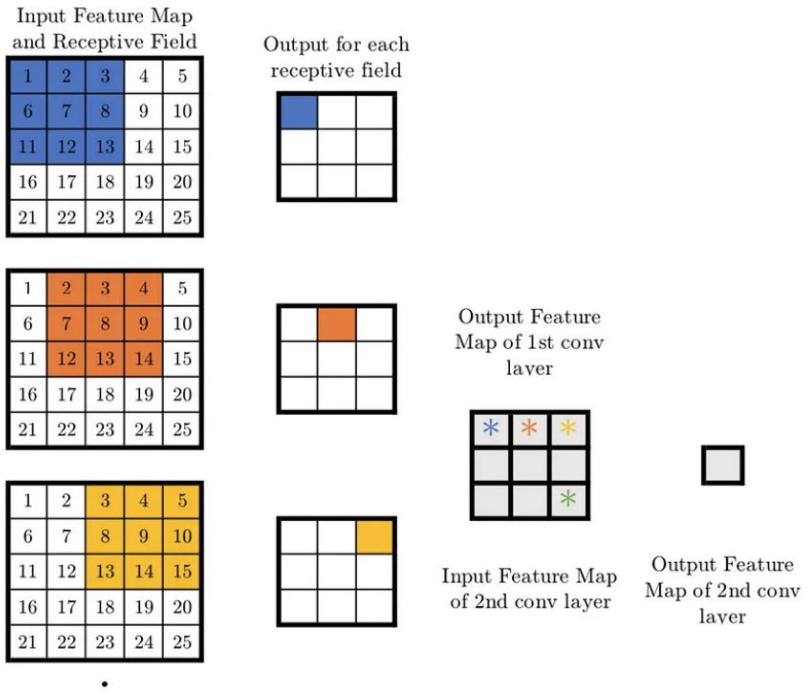
- one of the very first really successful applications of CNNs
- improved upon LeNet by using ReLU, adding dropout and stacking convolutional layers
- achieved a top-5 error of 15.3% on ImageNet in 2012, more than 10.8 percentage points better than that of the runner-up
- 62M parameters in 8 trainable layers



https://en.wikipedia.org/wiki/LeNet

VGG: stacking 3x3 filters to simulate 5x5 and 7x7

- the next big improvement after AlexNet
- published in 2014, it was founded on the observation that two succeeding layers of 3x3 filters have the same receptive field as a single 5x5 filter, while having fewer parameters
- this promised the same modelling capabilities with a lower tendency to overfit

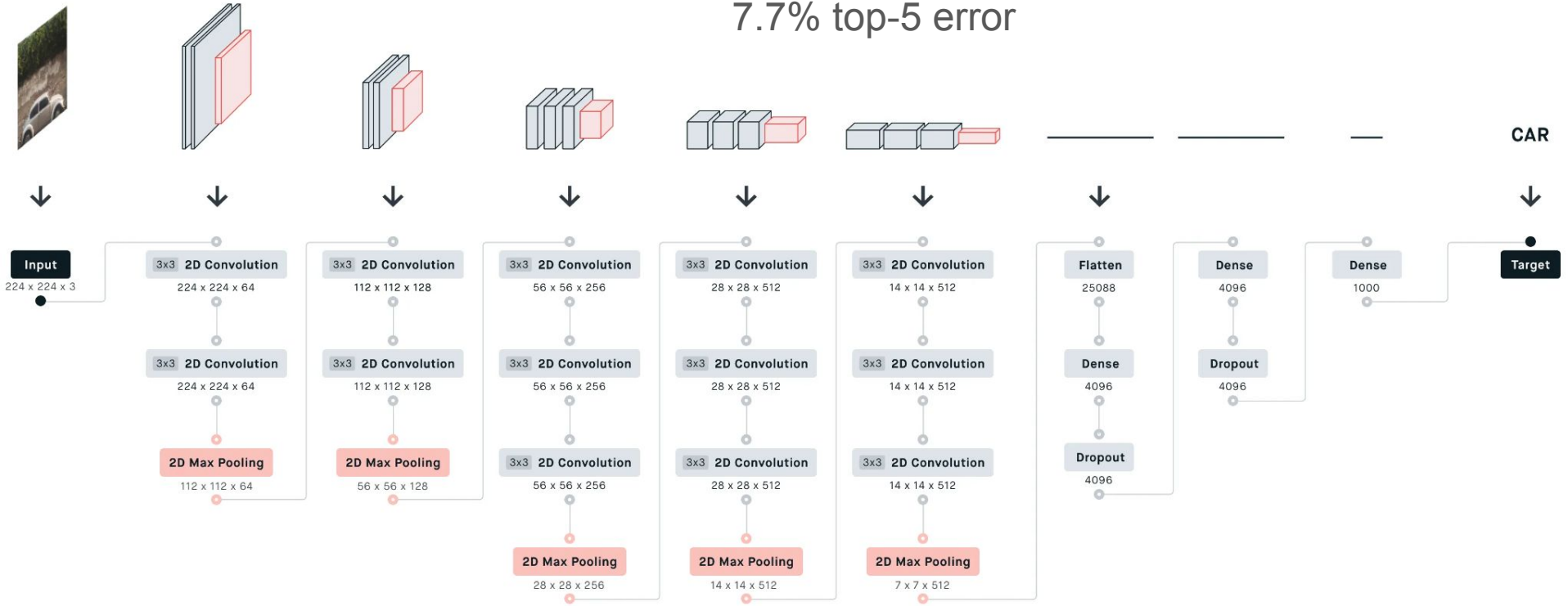


https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

VGG16 architecture

138M parameters in 16 trainable layers

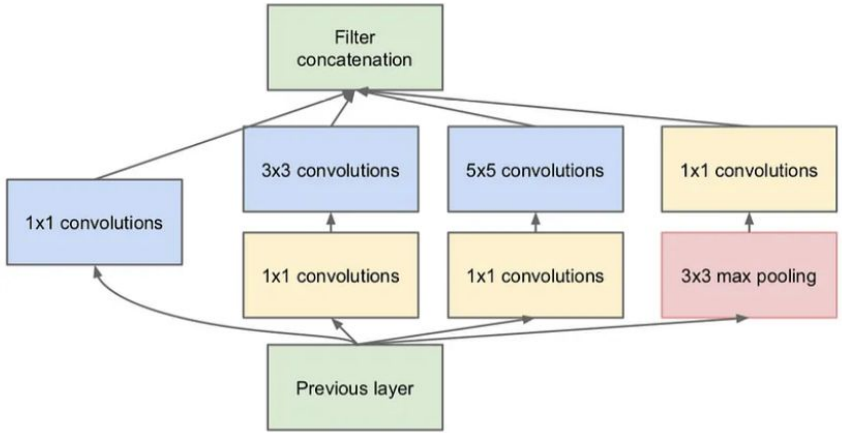
7.7% top-5 error



https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f

Inception module

- makes a single convolutional layer contain filters of different sizes, so the same logic can be applied in slightly different scales
- filters of different scale have independent parameters
- their results are concatenated along the third dimension

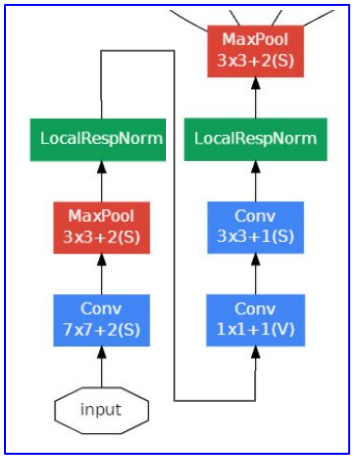


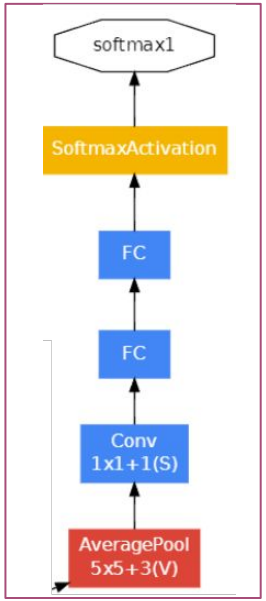
- 1x1 convolutions with the output depth lower than the input depth were added to prevent the number of parameters from blowing up
- after VGG, the 5x5 filter was substituted with two 3x3 filters

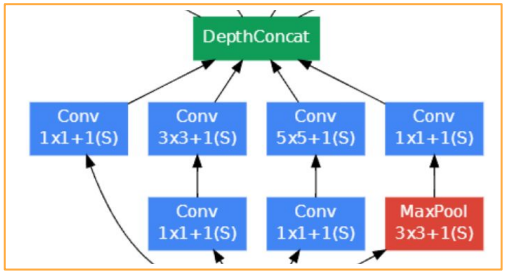
Going Deeper with Convolutions, Szegedy et al. (2014)

GoogleNet

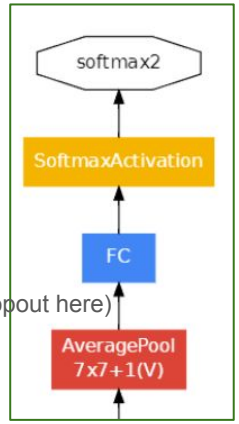
only 6.4M parameters
in 10+9x6 trainable layers
6.4% top-5 error

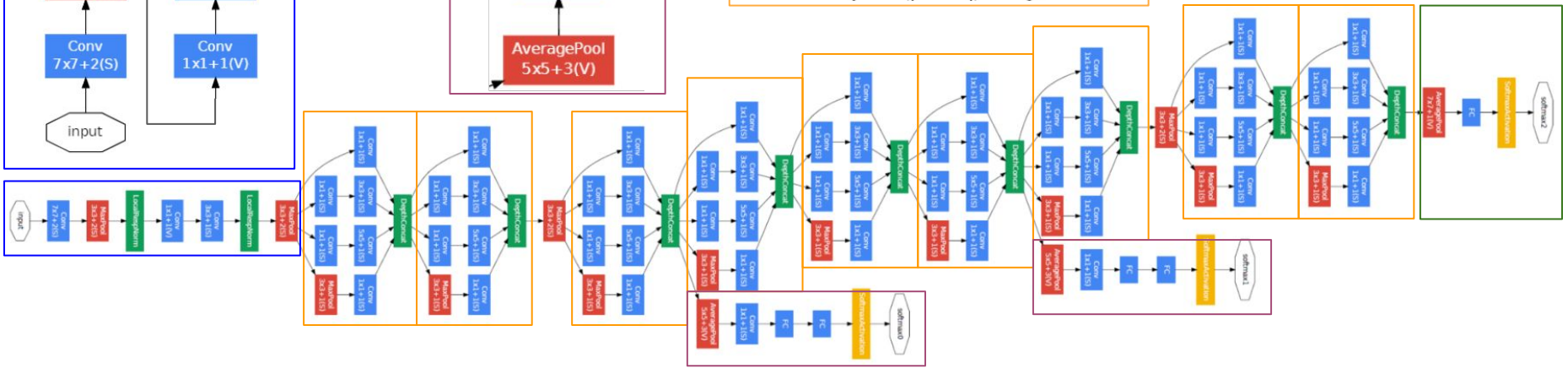






(there's dropout here)

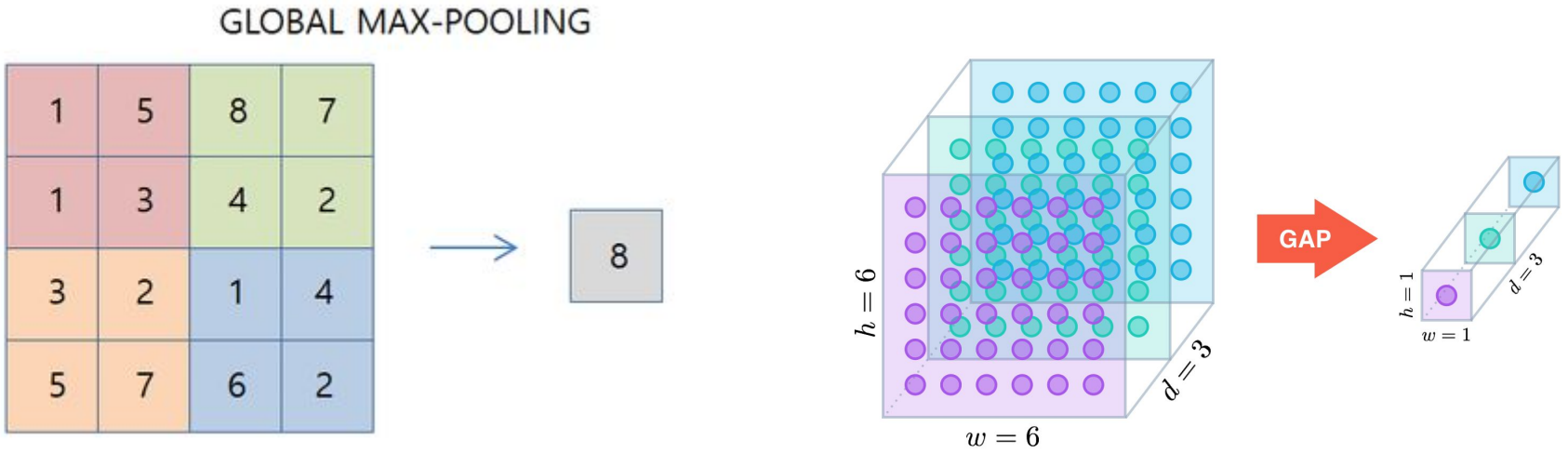




Going Deeper with Convolutions, Szegedy et al. (2014)

Global pooling

- an extreme variant of pooling that reduces a 2D feature map to a single value
- can be used instead of the first fully-connected layer to reduce overfitting
- improved top-1 accuracy of GoogleNet by 0.6 percentage points

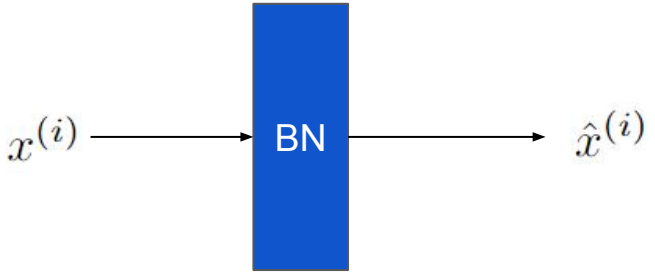


https://www.researchgate.net/figure/The-difference-of-max-pooling-and-global-max-pooling_fig4_338079465

https://www.kaggle.com/discussions/general/175896

Batch Normalization

- most of the time, input normalization and initialization are not enough and the distributions between layers start to drift during training (covariance shift)
- batch normalization normalizes (pre)activations from a single mini-batch
- after normalization, trainable shift and scale vectors are applied to the outputs
- estimated means and variances are kept for the use during testing



$$\hat{x}^{(i)} = \gamma \frac{x^{(i)} - \mu_B}{\sigma_B} + \beta$$

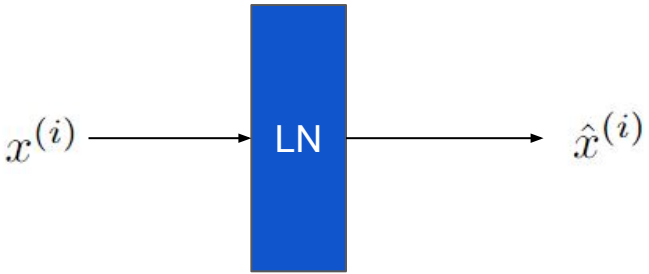
	coffee price	distance to city centre
Charlotte	~~15~~ 1	~~1000~~ 0.84
KFC	~~10~~ -1	~~200~~ -1.40
The Sun's Orbit	~~12.5~~ 0	~~900~~ 0.56

Batch Normalization

- batch normalization can be used before convolutional and fully-connected layers. It can also be used before and after activation functions, but it seems to work better when used before them
- once BN had been used in GoogleNet, dropout was no longer necessary
- the most popular theory says that BN works well because it smooths the loss function landscape, allowing for higher learning rates and faster convergence

Layer Normalization

- as batch normalization relies on the mini-batch to calculate mean and variance, it's not viable for very small batches. It also creates a difference between train and inference
- layer normalization normalizes features within every sample by subtracting a single sample's features' mean and dividing by their standard deviation
- used mostly in Transformers and RNNs

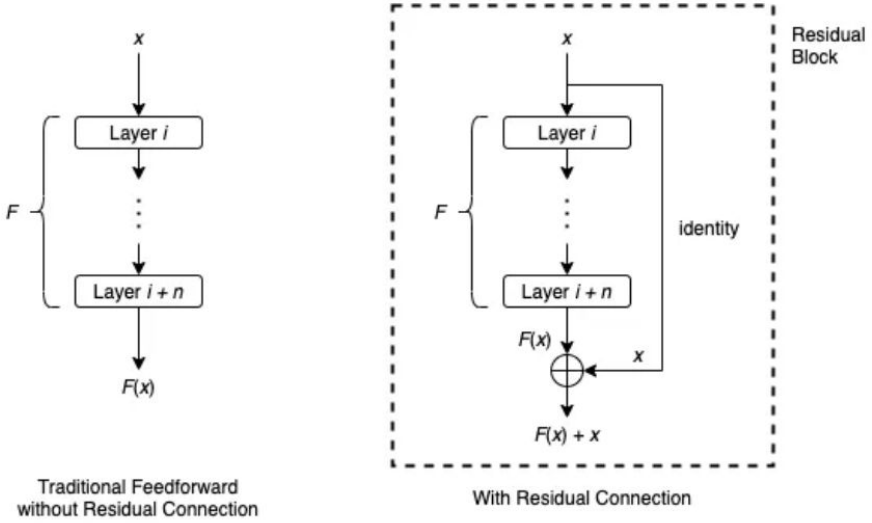


$$\hat{x}_j^{(i)} = \gamma \frac{x_j^{(i)} - \mu_{x^{(i)}}}{\sigma_{x^{(i)}}} + \beta$$

	verbs	adjectives	nouns
Pride and Prejudice	~~30k~~ 0.7	~~30k~~ 0.7	~~25k~~ -1.41
Macbeth	~~2k~~ 0.18	~~400~~ -1.3	~~3k~~ 1.12

Residual connections (a.k.a. skip connections)

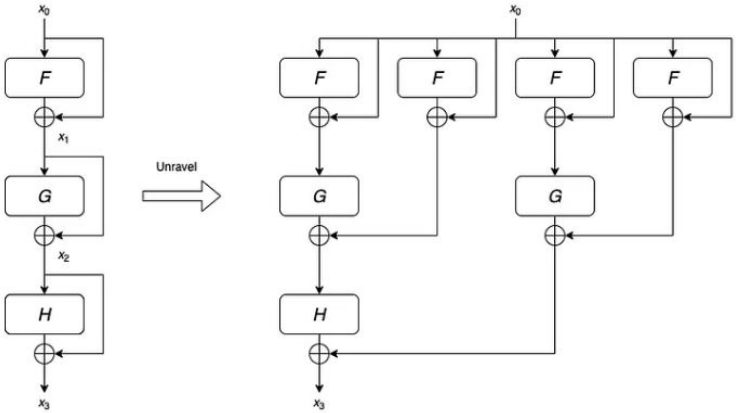
- based on an observation that it's difficult to train an identity function, so layers might struggle to propagate all input information
- input is added to the output; F(x)+x is modelled instead of F(x)
- when input and output dimensions don't match, a simple 1x1 convolutional filter with a specific stride and padding can be used on the residual x

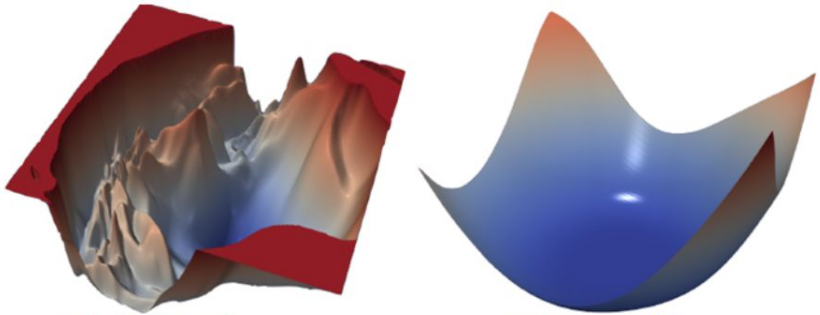


https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55

Skip connections allow to use features of previous layers

- interpretation: skip connections make it possible to use features of all previous layers without the need to dedicate parts of the trained layers for passing them
- fewer bottlenecks for important features are created, which is reflected in a smoother loss landscape





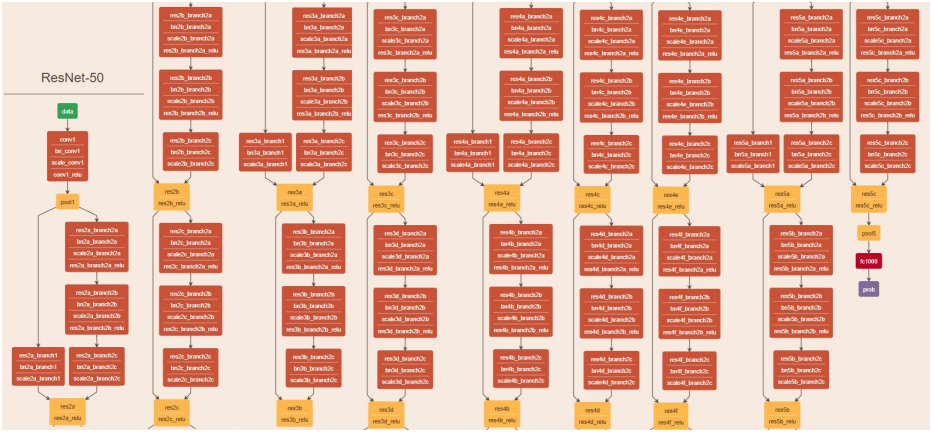
(a) ResNet-110, no skip connections (b) DenseNet, 121 layers

Figure 4: The loss surfaces of ResNet-110-noshort and DenseNet for CIFAR-10.

Visualizing the Loss Landscape of Neural Nets, Li et al. 2017
https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55

ResNet

- a successor to VGG and GoogleNet, published in 2015
- uses Batch Normalization and residual connections
- Resnet-152, despite an extremely high number of layers, keeps the number of parameters on the level of AlexNet
- pooling is no longer needed
- for a readable architecture graph, see the link



Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

https://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006

ImageNet as a benchmark

- with ImageNet now mostly solved and no better supervised datasets in sight, the scientific community shifted towards image generation and semi-supervised learning
- models pre-trained on ImageNet remain popular for practical use

Are we done with ImageNet?

Lucas Beyer[1]* Olivier J. Hénaff[2]* Alexander Kolesnikov[1]* Xiaohua Zhai[1]* Aäron van den Oord[2]*
[1]Google Brain (Zürich, CH) and [2]DeepMind (London, UK)

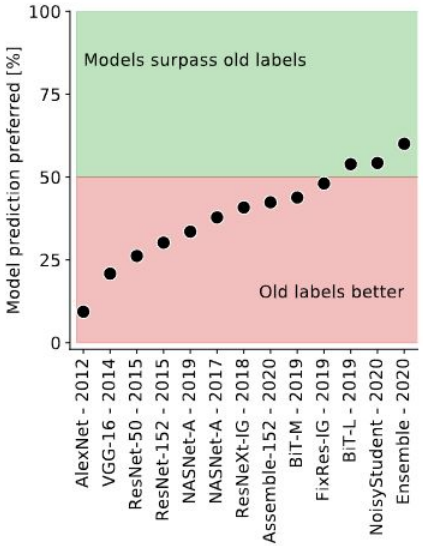


Figure 1: When presented with a model's prediction and the original ImageNet label, human annotators now prefer model predictions on average (Section 4). Nevertheless, there remains considerable progress to be made before fully capturing human preferences.

- too small to benefit from Vision Transformers

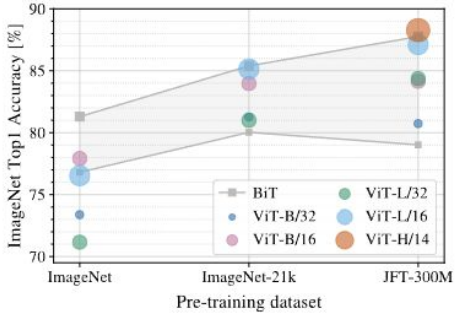


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

Are we done with ImageNet? Beyer et al. 2020
An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, Dosovitskiy et al. 2020

Adam

- the most popular optimizer other than SGD
- “normalizes” gradients of each parameter by dividing them by their uncentered standard deviation
- step in each direction from the standard base < learning rate
- tends to work better than SGD when layers’ gradients have significantly different scales (e.g. in Transformers)

$$
\begin{aligned}
m_i &= \beta_1 m_{i-1} + (1-\beta_1)\nabla\mathcal{L}(w_i) \\
s_i &= \beta_2 s_{i-1} + (1-\beta_2)(\nabla\mathcal{L}(w_i))^2 \\
\hat{m}_i &= \frac{m_i}{1-\beta_1^i} \\
\hat{s}_i &= \frac{s_i}{1-\beta_2^i} \\
v_i &= \frac{-\eta\hat{m}_i}{\sqrt{\hat{s}_i}+\epsilon 1}
\end{aligned}
$$

Weight decay

- overfitting of neural networks usually goes hand in hand with large individual weights
- weight decay reduces weights by a small margin after every epoch
- contrary to a popular misconception, weight decay is equivalent to L2 weight regularization only when used with SGD. In Adam, it leads to different (and usually better) results
- as applying weight decay to Adam generated a lot of confusion, the proper way to do it had been named AdamW

SGD with momentum and WD:

$$v_i = \beta v_{i-1} - \eta \nabla \mathcal{L}(w_i) - \eta \lambda w_{i-1}$$

AdamW:

$$w_i = w_{i-1} - \frac{\eta \hat{m}_i}{\sqrt{\hat{s}_i} + \epsilon 1} - \eta \lambda w_{i-1}$$