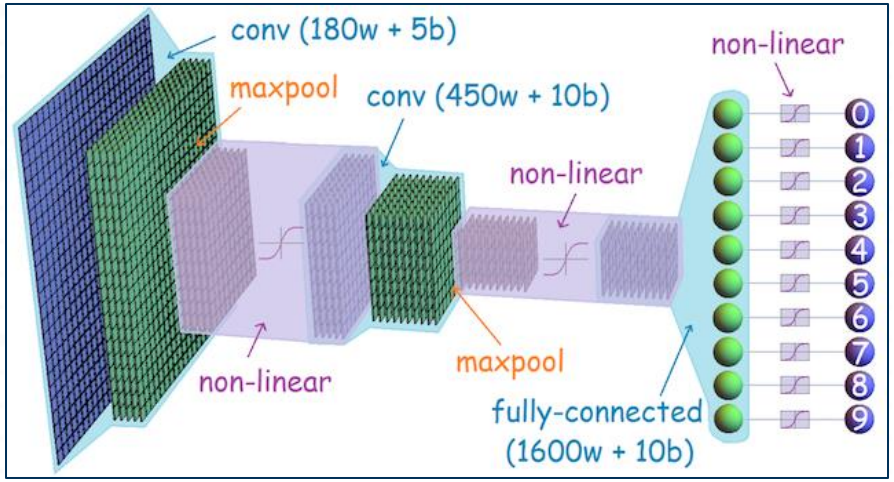
ECE 4252/8803: Fundamentals of Machine Learning (FunML)
Fall 2024

Lecture 15: CNN Architectures



OLIVES @GeorgiaTech

Georgia Tech

Overview

In this Lecture..

Locally Connected Layer

Introduction to Convolutional Neural Networks

Layers used to build Convolutional Neural Networks

- Convolutional layer
- Pooling Layer
- Fully-connected layers

Examples of Deep CNNs Architectures

Training CNNs

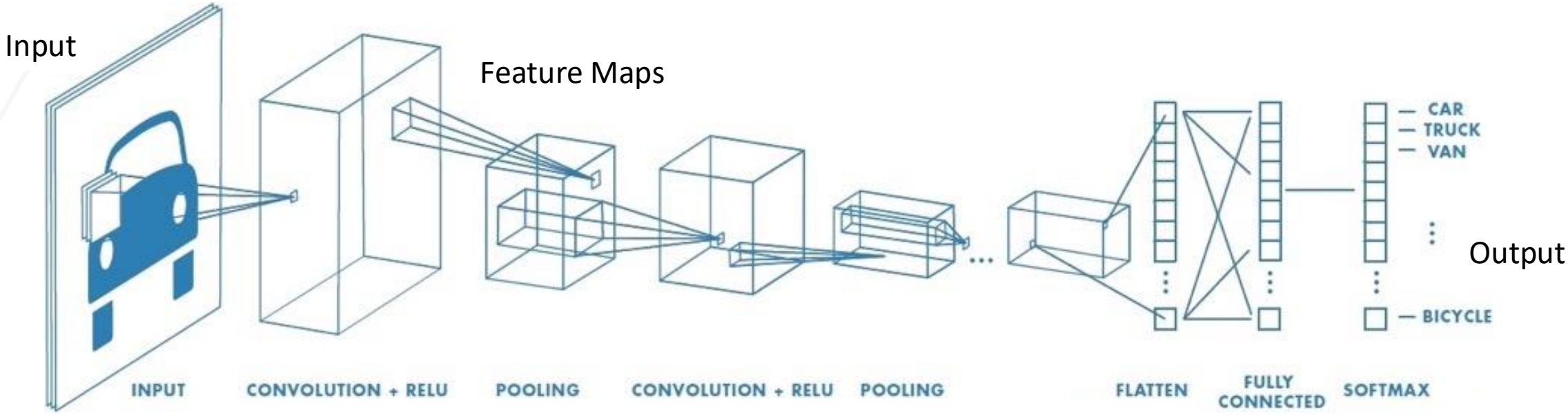
Visualization of Convolutional Neural Networks

Convolutional Neural networks

Fully-connected Layer: 1x1 Convolution

Three main types of layers to build ConvNet architectures:

- Convolutional Layer, Pooling Layer, and Fully-Connected Layer (ANN)



different layers in a CNN

Overview

In this Lecture..

Locally Connected Layer

Introduction to Convolutional Neural Networks

Layers used to build Convolutional Neural Networks

Examples of Deep CNNs Architectures

- LeNet
- AlexNet
- VGG
- GoogleNet
- ResNet

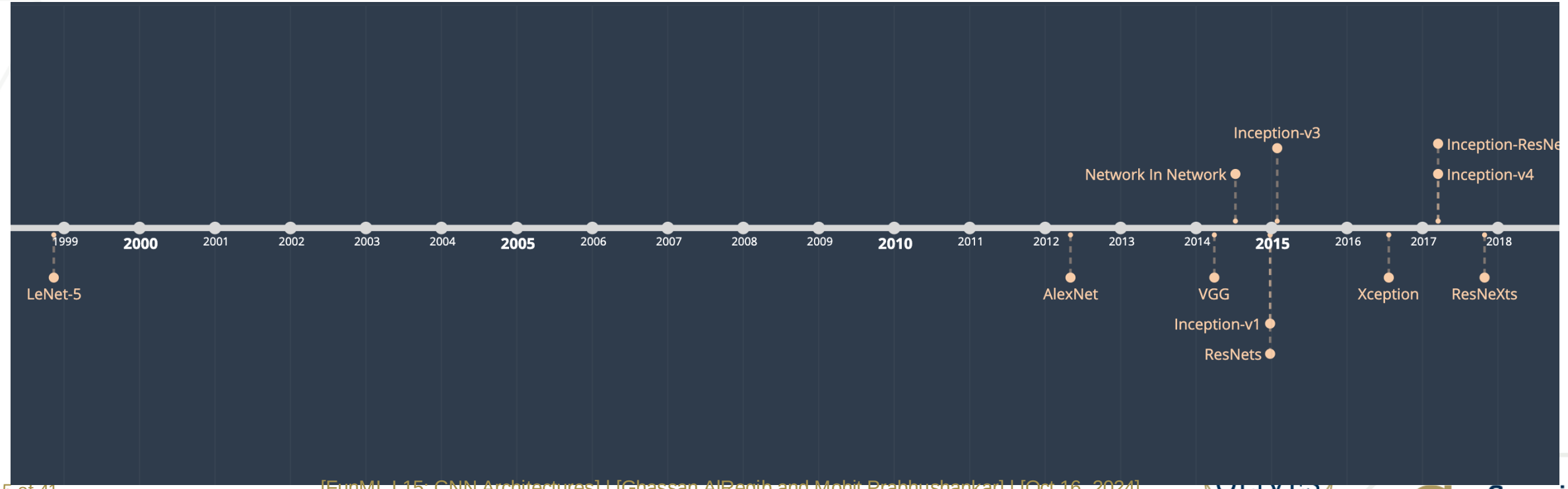
Training CNNs

Visualization of Convolutional Neural Networks

History of CNNs

Evolution of CNNs

- LeNet
- AlexNet
- VGG
- GoogLeNet (Inception-V1)
- ResNet



CNN Architectures

Legend

Layers

conv 3×3 — Convolutional operations, in **red**

avg-pool 2×2 — Pooling operations, in **grey**

concat — Merge operations eg. concat, add in **purple**

Dense layer, **blue**

Activation Functions

T — Tanh

R — ReLU

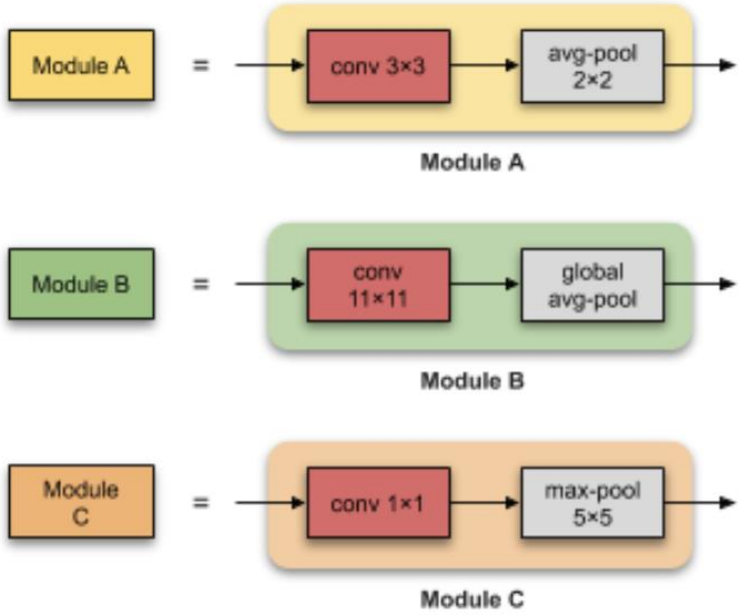
Other Functions

B — Batch normalisation

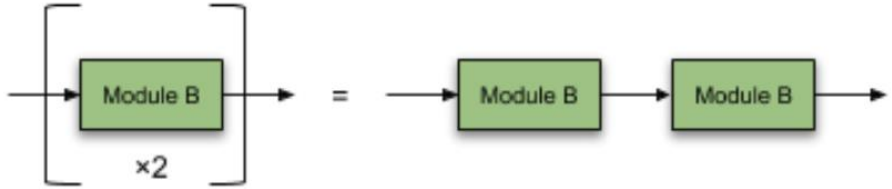
S — Softmax

Modules/Blocks

Modules (groups of convolutional, pooling and merge operations), in **yellow, green,** or **orange**.
The operations that make up these modules will also be shown.



Repeated layers or modules/blocks



https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

CNN Architectures

LeNet-5 (1998)

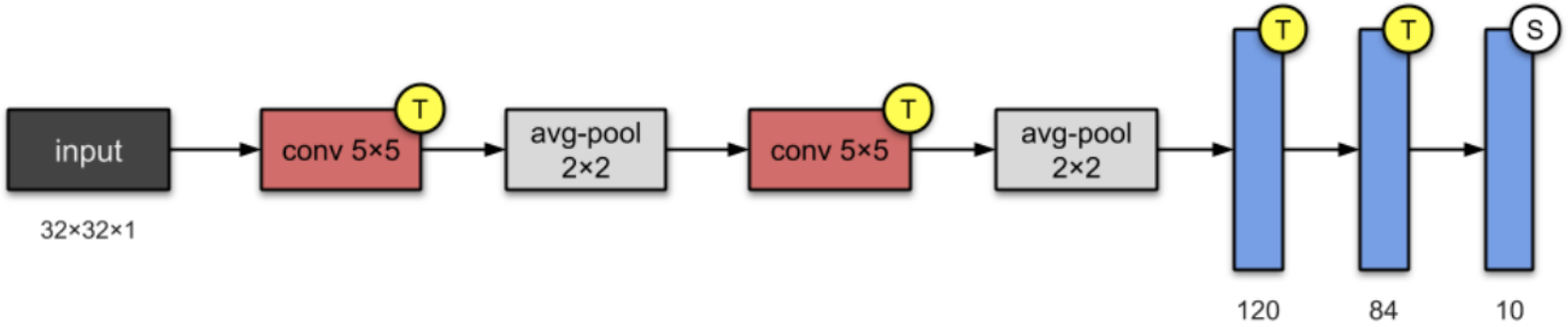


Fig. 1: LeNet-5 architecture, based on their paper

Novelty:

- Reduced number of learnable parameters and **learned from raw pixels automatically**
  - Made hand engineering features redundant
- The 1st popular CNN that became the "standard" template of CNNs
  - Stacking convolutional, activation, pooling layers
  - Ending with fully connected layers
- Good results on small datasets
  - **Top-5 error rate** on MNIST is 0.95%

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

CNN Architectures

LeNet-5 (1998)

Pros

- State-of-the-art performance on hand digit recognition tasks
  - Low resolution, greyscale images
- **Unaffected by small distortions, rotations, varying position, scale**
- Good on other small datasets

Cons

- Doesn't perform well on all classes of images, e.g., color
- Limited by availability of computing resources at the time

CNN Architectures

Long Gap (1998-2012)

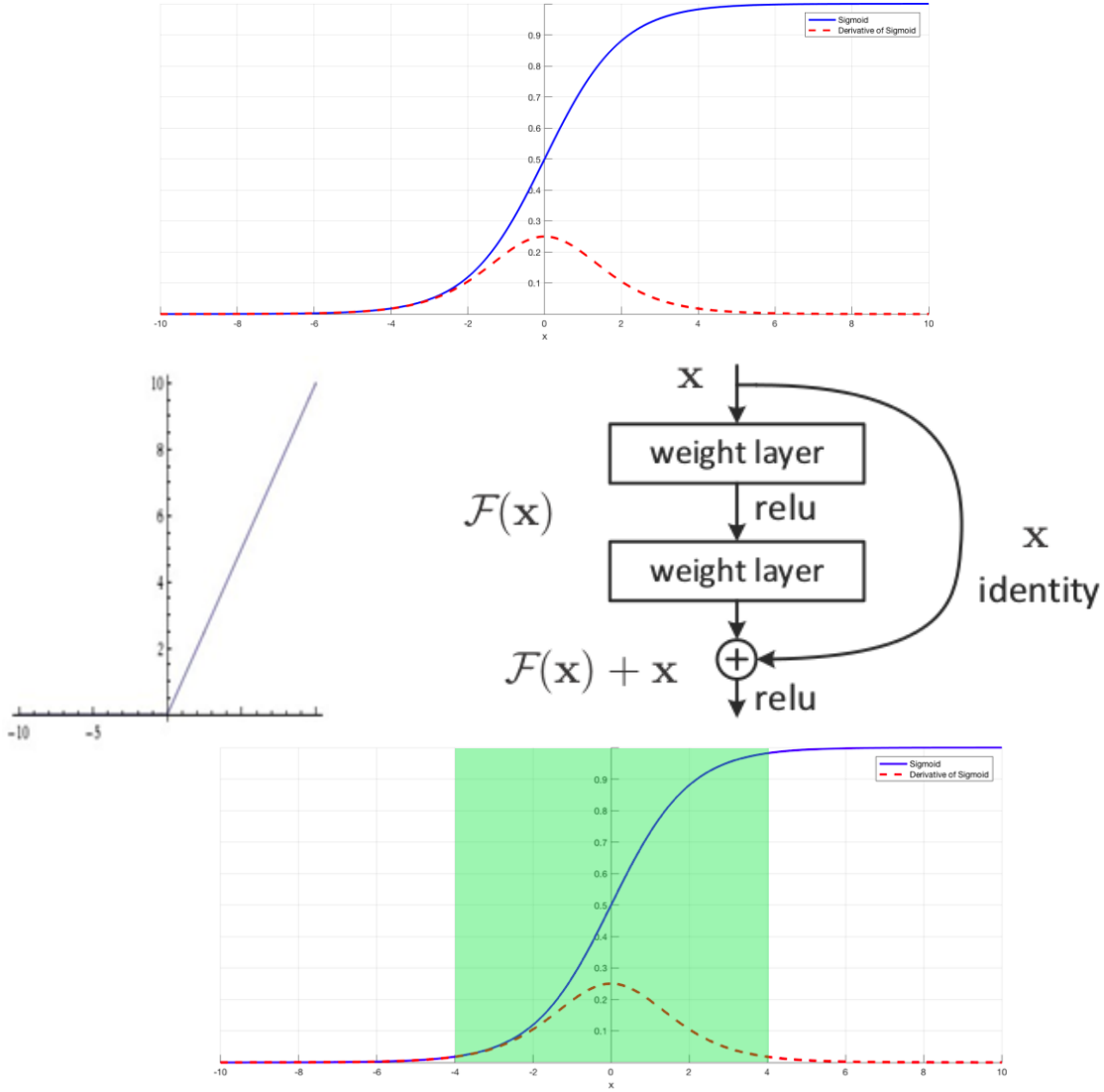
- Working to improve computational power
  - Existing accelerators were not yet sufficiently powerful to make deep multichannel, multilayer CNNs with a large number of parameters.
- Existing datasets were relatively small
  - Limited storage capacity of computers
  - Expense of sensors
- Tricks for neural network training were not established yet
  - Parameter initialization
  - Variants of stochastic gradient descent
  - Non-squashing activation functions
  - Effective regularization techniques
  - **Vanishing Gradient Problem**

Convolutional Neural networks

Vanishing Gradient Problem

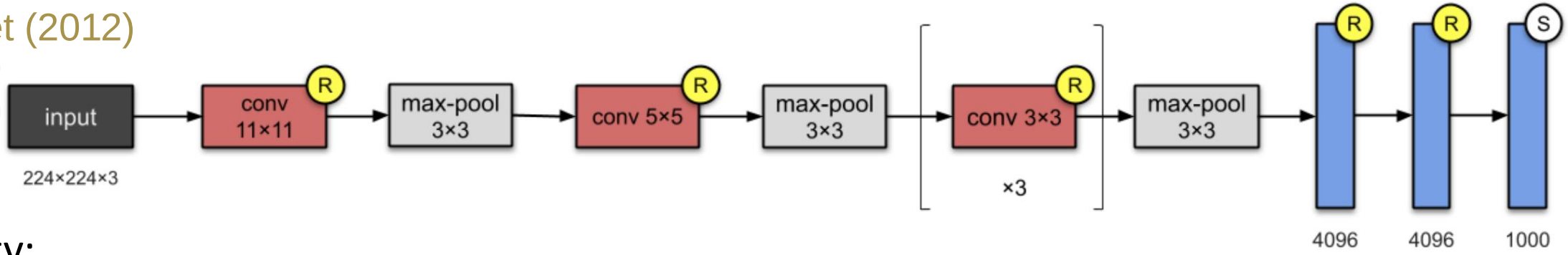
- Using the sigmoid, the gradient could become smaller and smaller and in deep networks, that may cause the training to be ineffective as no change could take place [check the derivative of the sigmoid in the plot]

- Possible solutions:
  - Use ReLU
  - Use Residual networks
  - Normalization



CNN Architectures

AlexNet (2012)



Novelty:

- First to implement Rectified Linear Units (ReLUs) as activation, **solving the vanishing gradient problem**
- Applied **dropout regularization** to fully connected layer to control complexity
- Applies **Normalization between layers**
- Deep CNN that ran on GPU hardware
- Deeper and wider than LeNet
- **More robust than LeNet (data augmentation)**
- Won ImageNet Challenge and significantly outperformed traditional methods

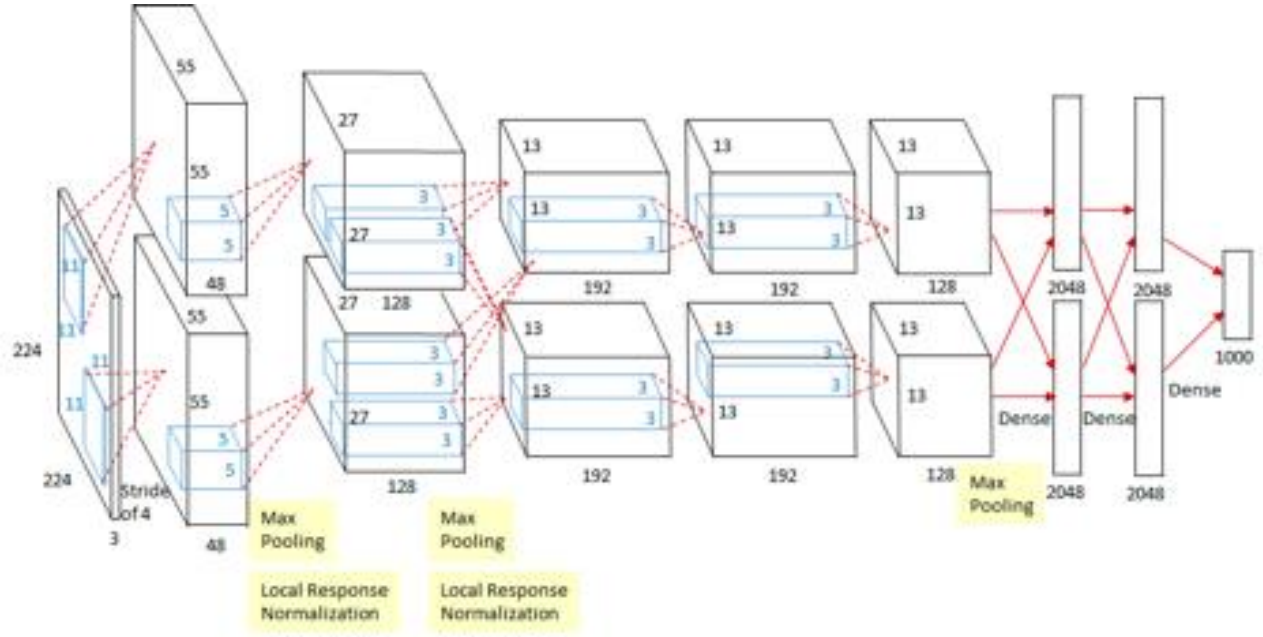
https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

CNN Architectures

AlexNet (2012)

Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160

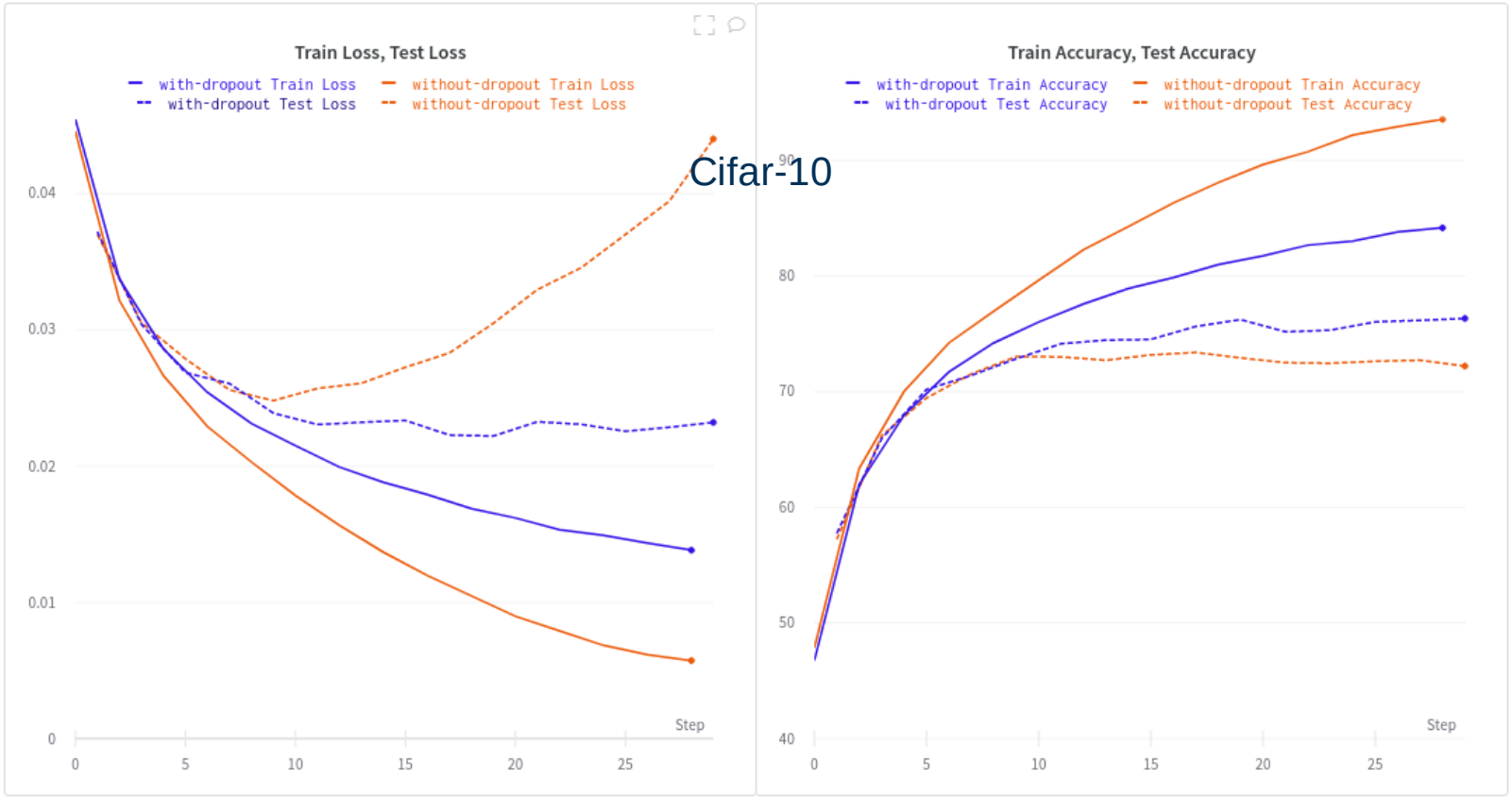
Details/Retrospectives:

- first use of ReLU
- used Norm (Local Response Normalization) layers
- GPU implementation
- used Dropout

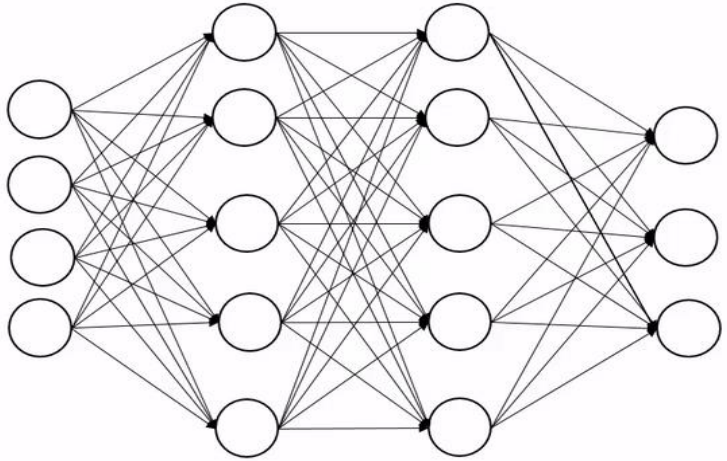
Krizhevsky et al. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012)

CNN Architectures

AlexNet (2012): Dropout



Cifar-10



applying a dropout on the second hidden layer of a neuron network

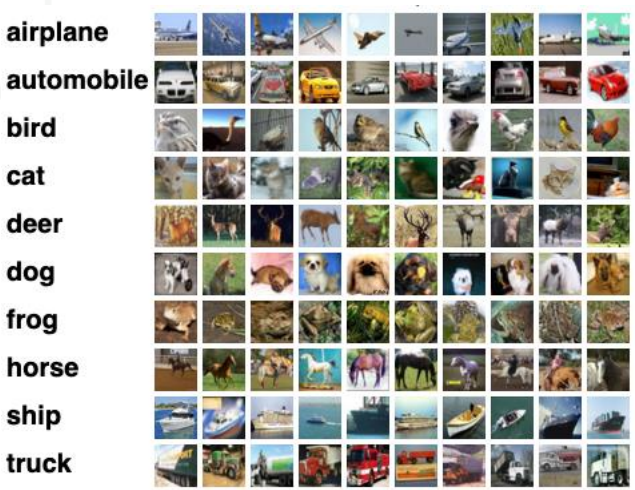
- Dropout refers to the practice of **disregarding certain nodes in a layer at random** during training
- A dropout is a **regularization approach** that **prevents overfitting** by ensuring that no units are **codependent with one another**.
  - Dropout makes the **training process noisy**, requiring nodes within a layer to take on more or less responsible for the inputs on a probabilistic basis.

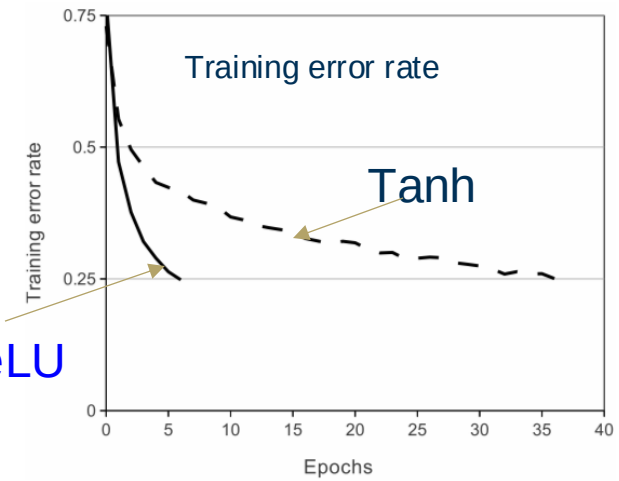
Krizhevsky et al. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012)

CNN Architectures

AlexNet (2012): ReLU

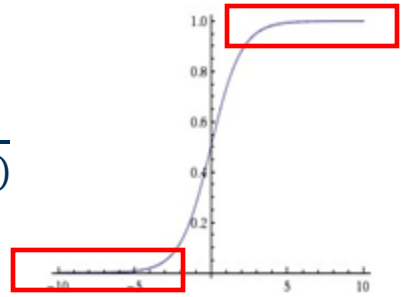
- Standard activation function was tanh or sigmoid, which are saturating nonlinearities
- ReLU is a non-saturating nonlinearity, which allows for faster training with gradient descent
- Trained about 6x faster on CIFAR-10 dataset than model with tanh activations





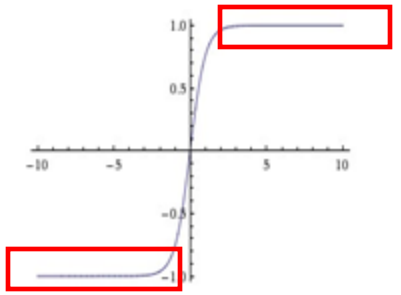
60000 32x32 color images; 10 classes
6000 images per class
50000 training images and 10000 test images

Sigmoid $\sigma(x) = \frac{1}{(1+e^{-x})}$

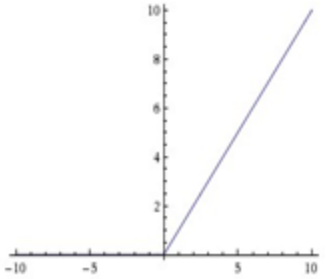


Saturating, gradients at these regions is almost zero. Gradient descent is slower

Tanh $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$



ReLU $\max(0, x)$
non-saturating in (0, +∞)



Krizhevsky et al. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012)

CNN Architectures

AlexNet (2012): LRN

Local Response Normalization (LRN)

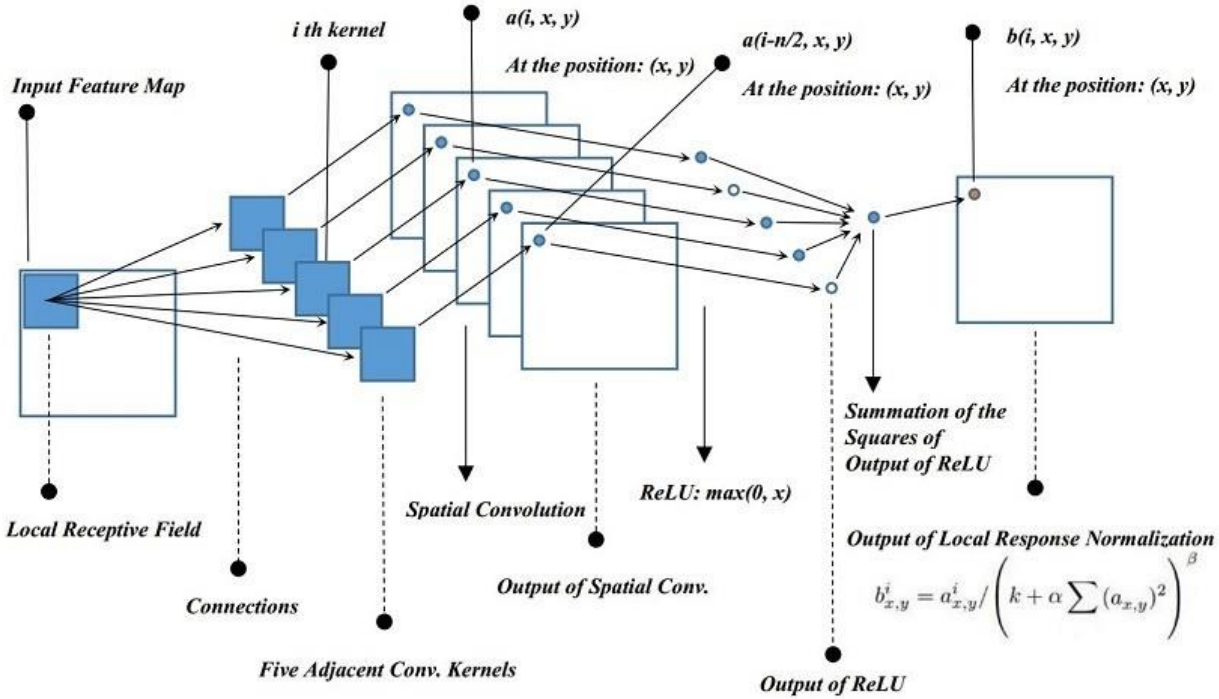
- Divides the output of a kernel $a^i_{x,y}$ by the output of n "neighboring" kernels at the same spatial position (x, y)

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

"brightness" normalization

$a^i_{x,y}$: output of the i-th kernel at position (x, y)
n: the number of "neighboring" kernels
N: the total number of kernels in the layer
k, α, β: hyper-parameters

- LRN is non-trainable



LRN introduces a "**lateral inhibition**" scheme:

- Enhances "peaks" and dampen "flat" responses to increase neural sensitivity to stimuli.
- Enhances peak activations by suppressing the weak responses in the neighborhood.
- Suppresses "flat" but high responses which might not encode much information regarding stimuli
- Helps generalization. Reduced top-1 and top-5 error rates by 1.4% and 1.2%

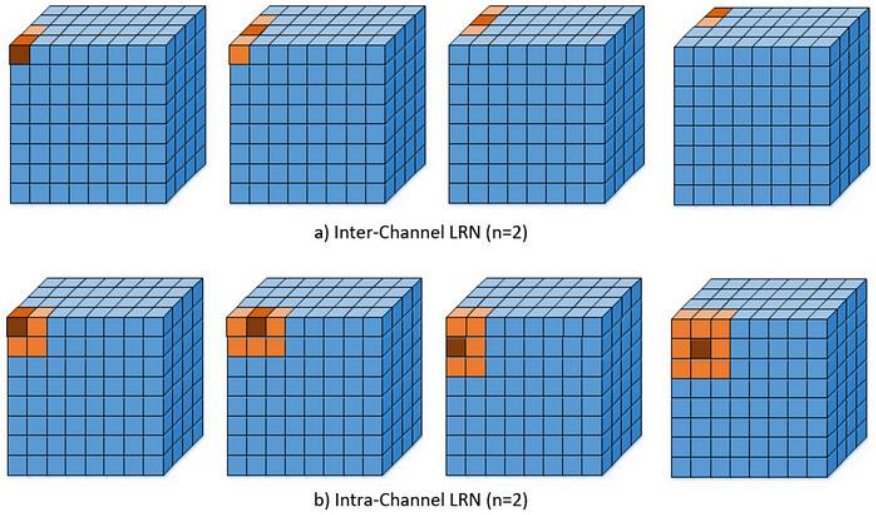
Krizhevsky et al. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012)

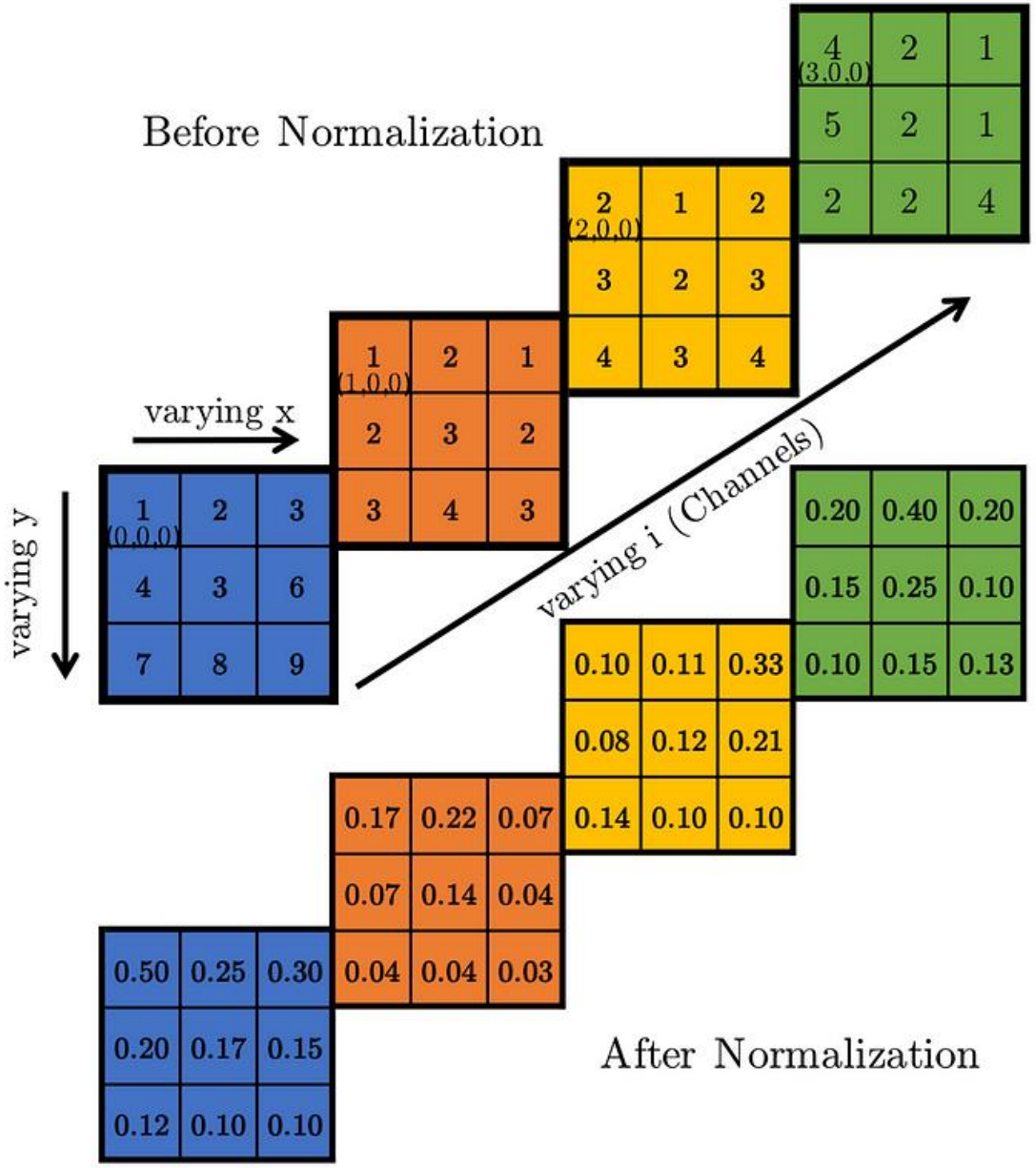
CNN Architectures

AlexNet (2012): LRN

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

- For this toy example, let $\mathrm{k} = 0, \alpha = \beta = 1, N = 4$, $n = 2$
- This LRN is referred to as inter-channel LRN, used in Alexnet.



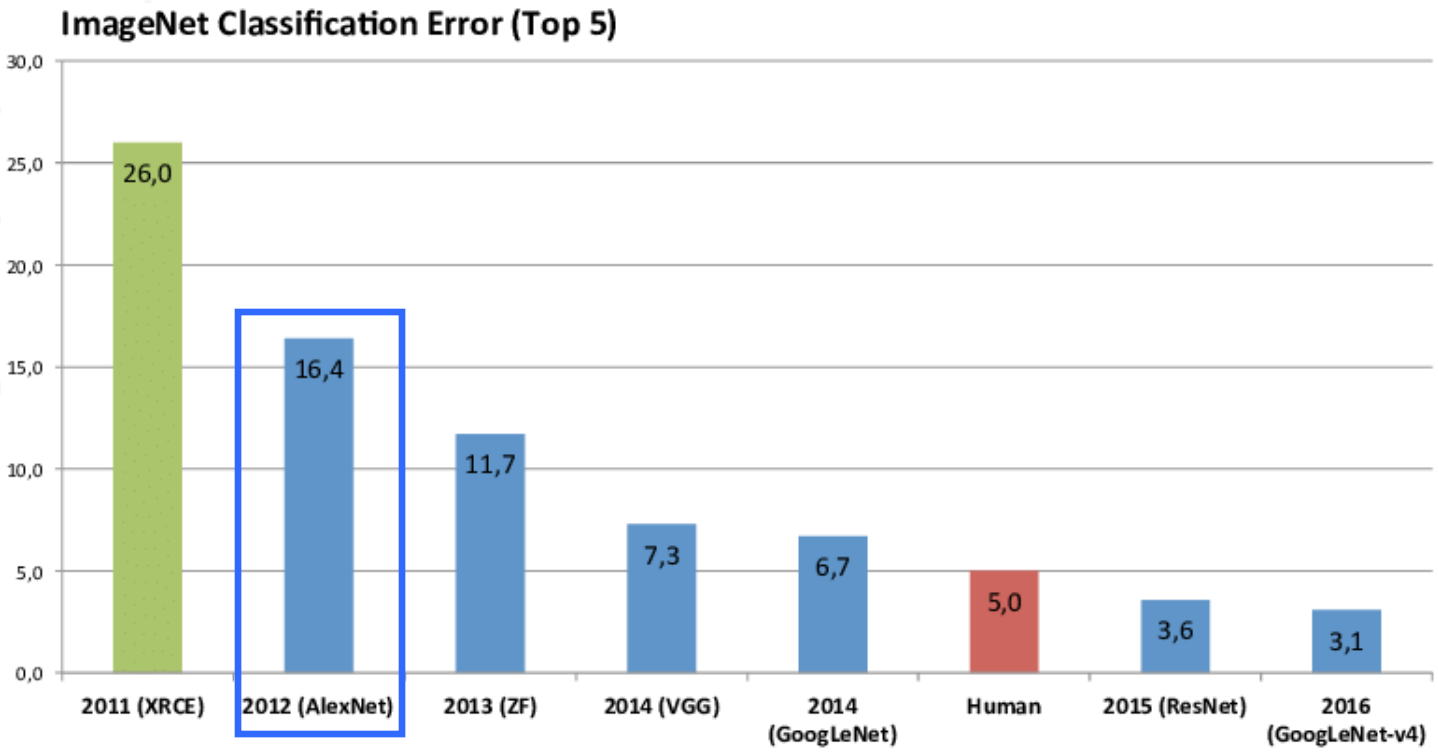


https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

CNN Architectures

AlexNet (2012)

60+ million parameters; 650K neurons



Imagenet:
1000 classes, 1.2M training images, 150K for testing

16.4% top 5 error in ILSVRC 2012

Figure Credit: Zitzewitz, Gustav. "Survey of neural networks in autonomous driving." (2017)

Krizhevsky et al. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012)

CNN Architectures

AlexNet (2012) Summary

Pros

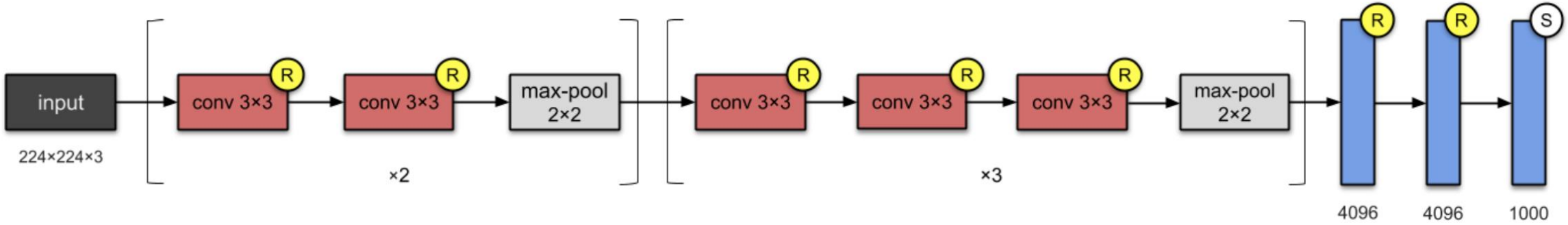
- Faster training of models via GPU
- Better at feature extraction than LeNet due to depth; also works well with color images
- ReLu activation
  - Does not saturate so fewer features are lost
  - Increases training speed as not all perceptrons are activated

Cons

- Less depth than future models so it struggles to learn features compared to more complex models
- Needs longer training time to achieve higher accuracy results compared to future models

CNN Architectures

VGG (2014)



Visual Geometry Group (VGG)

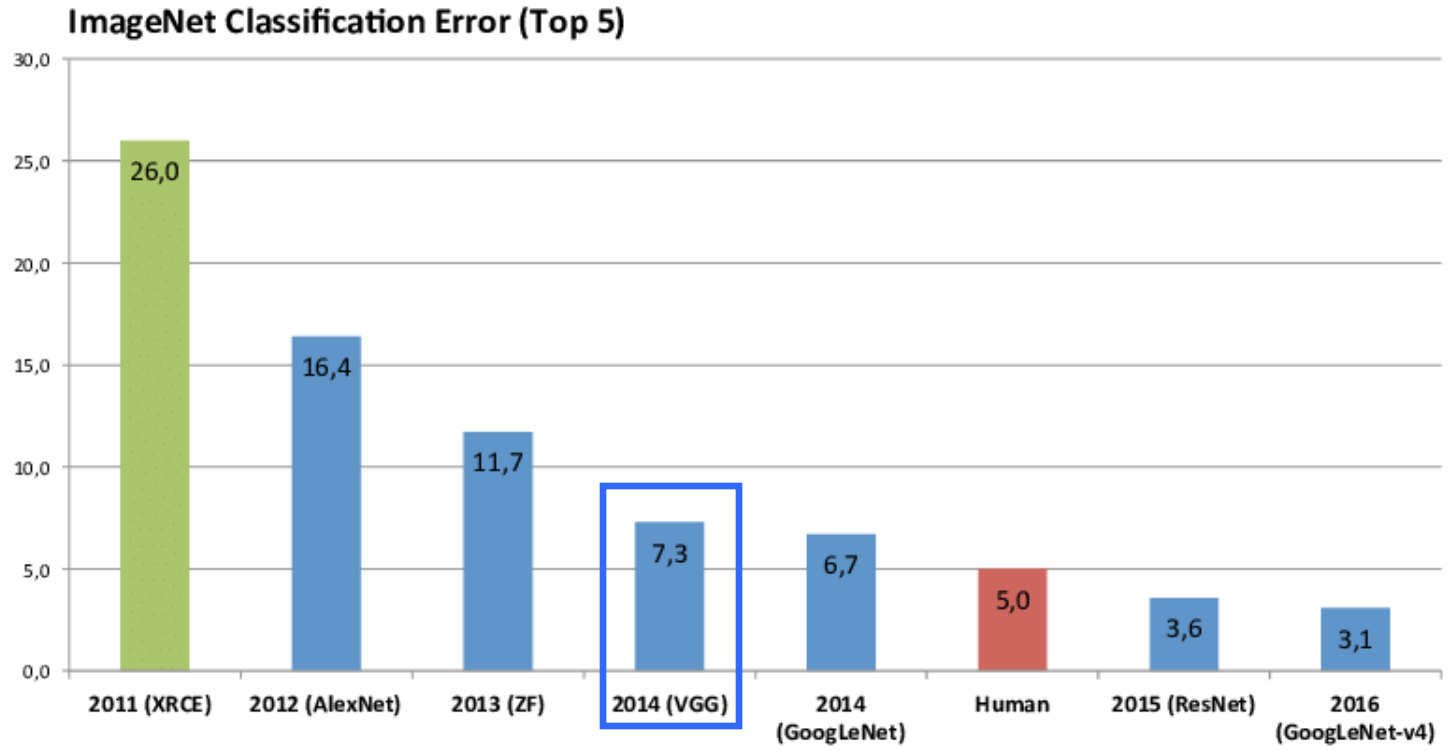
Novelty:

- Designed a deeper model than AlexNet to show the effect of depth on accuracy
- Grouping multiple convolution layers with smaller kernel sizes instead of having one Conv layer with a large kernel size.

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

CNN Architectures

VGG (2014)



11.7% previous top 5 error in ILSVRC 2013 → 7.3% top 5 error

Figure Credit: Zitzewitz, Gustav. "Survey of neural networks in autonomous driving." (2017)

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *ICLR* 2015

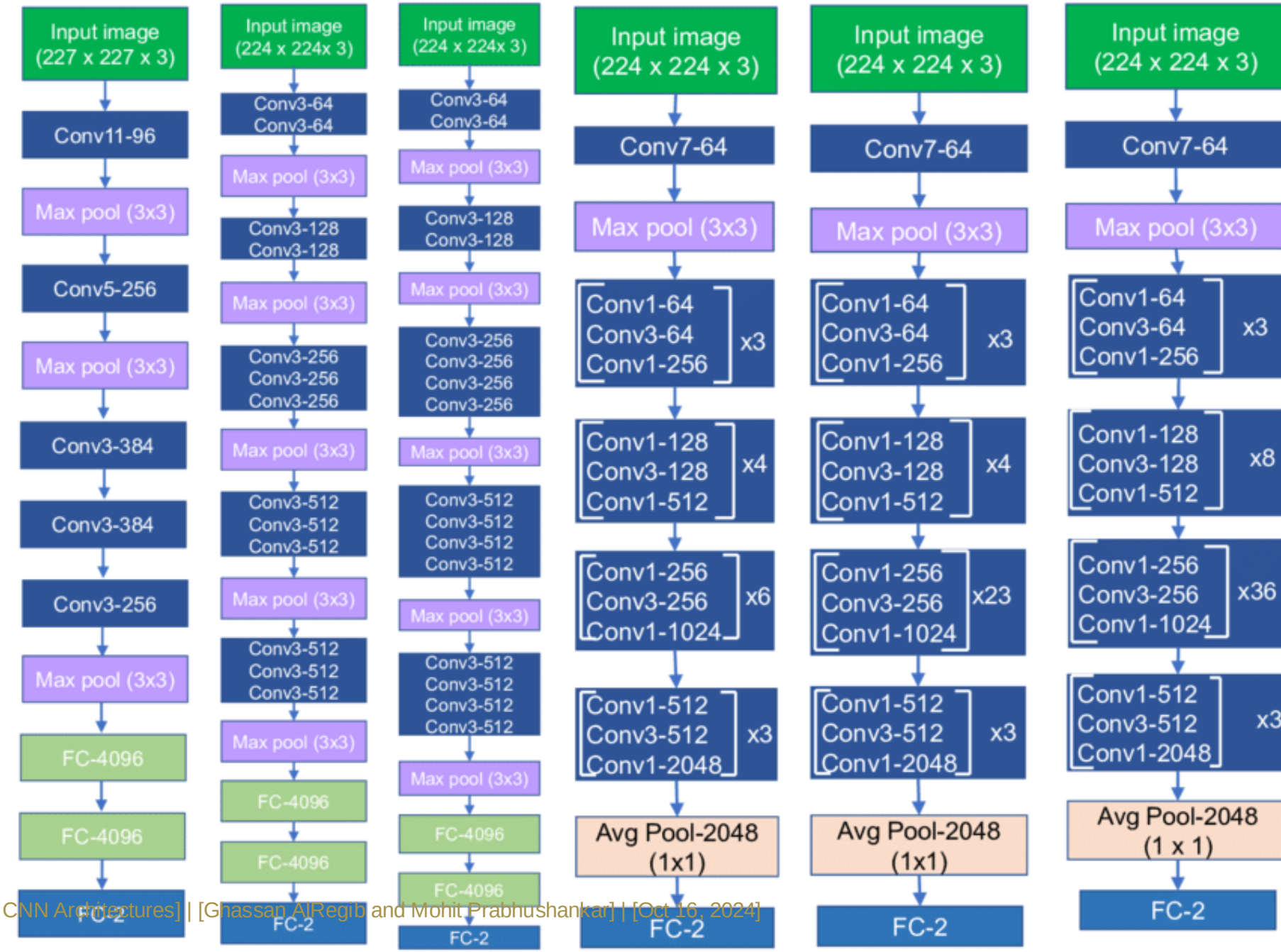
CNN Architectures
VGG (2014)

only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

deeper using smaller filters (3x3 CONV)

Stack of three 3x3 CONV (stride 1) layers has same receptive field as one 7x7 CONV layer

Also deeper, more non-linearities with fewer parameters: $3 \times (3^2 \times C^2)$ vs. $7^2 \times C^2$ for C channels per layer



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *ICLR* 2015

CNN Architectures

VGG (2014)

Pros

- Simple and effective design
- Improvement in accuracy and speed
- More layers with smaller kernels increases non-linearity
- Lower computational complexity from smaller kernels

Cons

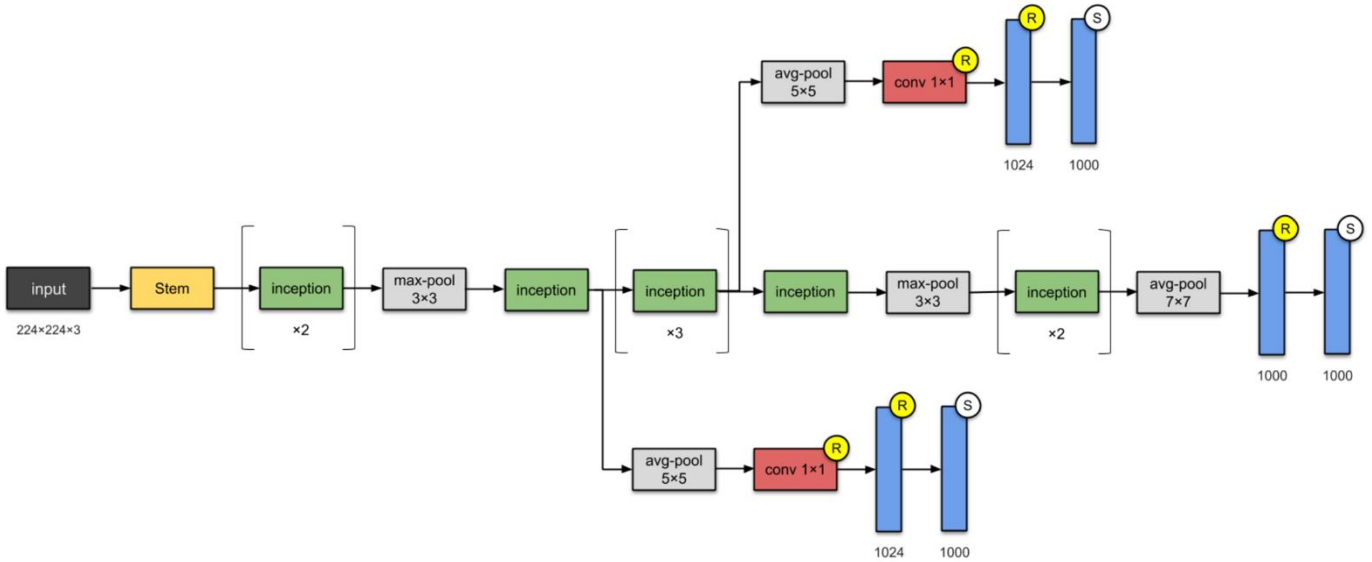
- 138 million parameters
  - Computationally expensive and difficult to deploy it on low resource systems
- Suffers from vanishing gradient problem
  - Gradients from the loss function shrinking to zero after several applications of the chain rule
- Slower than ResNet

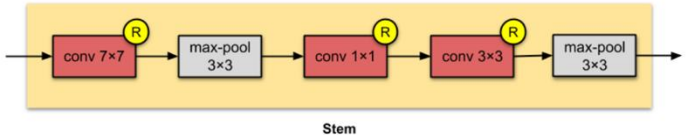
CNN Architectures

GoogleNet (2014)

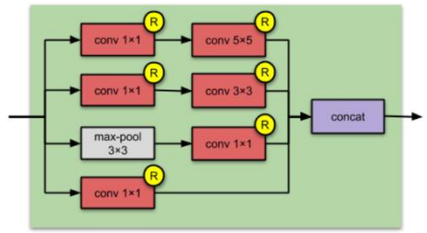
Novelty:

- Created the "**Network-in-Network**" approach via Inception modules
  - Captures spatial information at different scales
- **Used batch normalization**
- Reduced computational cost without affecting generalization
  - Factorizing symmetric convolutions into asymmetric
  - Using **global average pooling** at the last layer, instead of a fully connected layer reduced connection density
- Introduced the concept of auxiliary learners to speed up the convergence rate





Stem



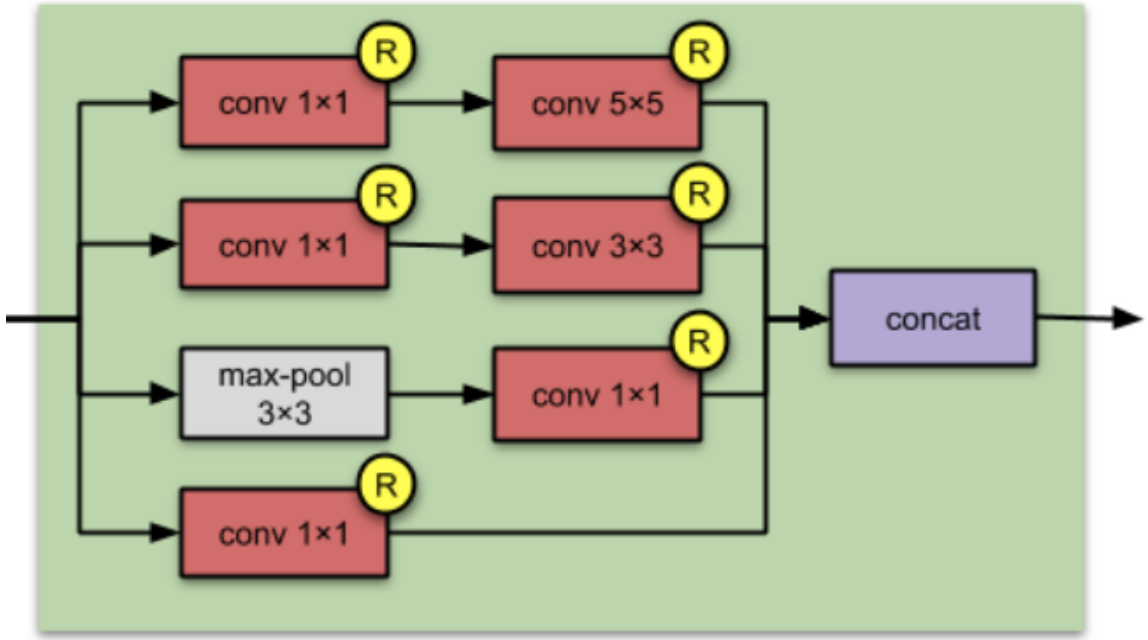
Inception module

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

CNN Architectures

GoogleNet (2014): Inception Module

- Encapsulates filters of different sizes (1x1, 3x3, and 5x5) to capture spatial information at different scales
- Regulates computations by adding a 1x1 convolutional bottleneck layer before employing large size kernels



Inception module

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

CNN Architectures

GoogleNet (2014): Batch Normalization

- Normalize the entire batch *B* to be zero mean and unit variance
  - Calculate the mean of the entire mini-batch output: *u_B*
  - Calculate the variance of the entire mini-batch output: s*igma_B*
  - Normalize the mini-batch by subtracting the mean and dividing with variance
- Introduce two trainable parameters (*Gamma:* scale_variable and *Beta:* shift_variable) to scale and shift the normalized mini-batch output
- Feed this scaled and shifted normalized mini-batch to the activation function.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

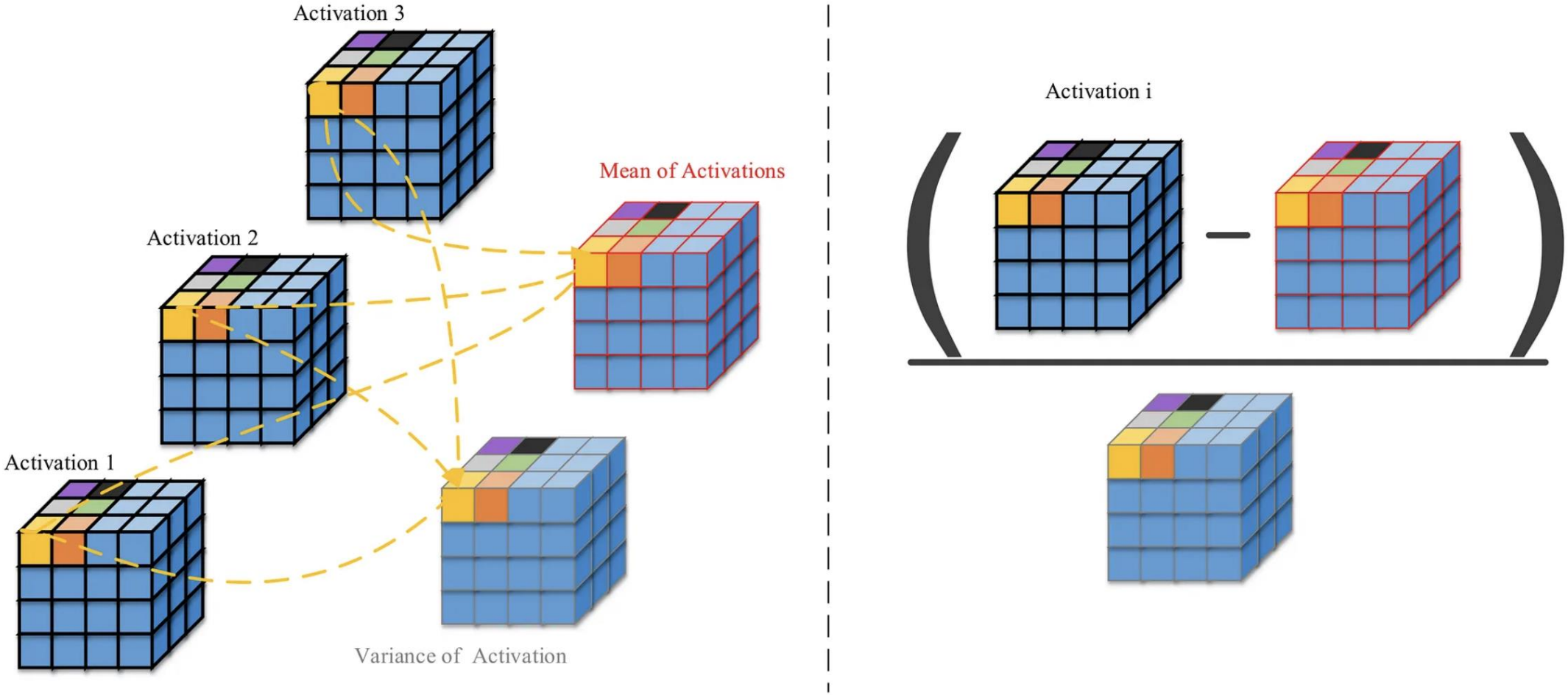
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *International conference on machine learning*. pmlr, 2015.

CNN Architectures

GoogleNet (2014): Batch Normalization



https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

CNN Architectures

LRN vs Batch Normalization

- LRN (Local Response Normalization)
  - was introduced because of the ReLU behavior
  - Non-trainable

- Batch Normalization
  - Introduced because of the Internal Covariate Shift within hidden neurons that may slow down training
  - Trainable
  - For every output of a hidden neuron, and before the activation function:
    - Normalize the entire batch, B, to have zero mean and unit variance
    - Shift the mean and scale the variance using two hyperparameters

CNN Architectures

GoogleNet (2014): Summary

Pros

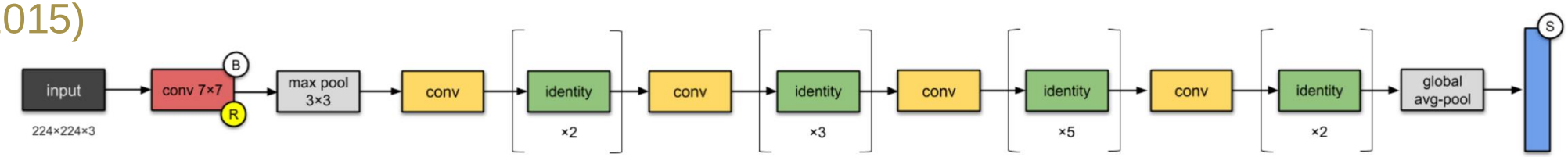
- Reduced computational cost without affecting generalization
  - Decrease number of parameters from 138 million to 4 million

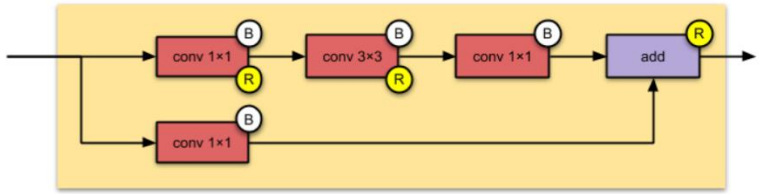
Cons

- Heterogeneous topology needs to be customized from module to module
- Representation bottleneck that drastically reduces the feature space in the next layer and leads to loss of useful information

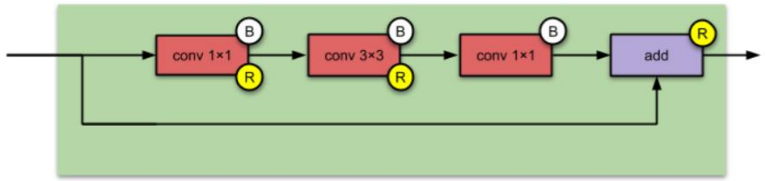
CNN Architectures

ResNet (2015)





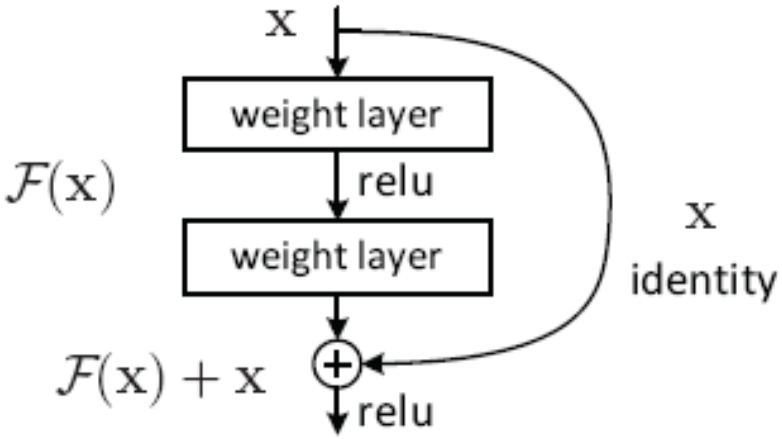
Conv block



Identity block

Novelty:

- Introduced residual learning (**Residual blocks**)
  - Shortcut connections with identity mapping
  - i.e., **skip connections**, gradients flow through a network without passing through activations
- 20 and 8 times deeper than AlexNet and VGG, respectively with less computational complexity and without compromising generalization power

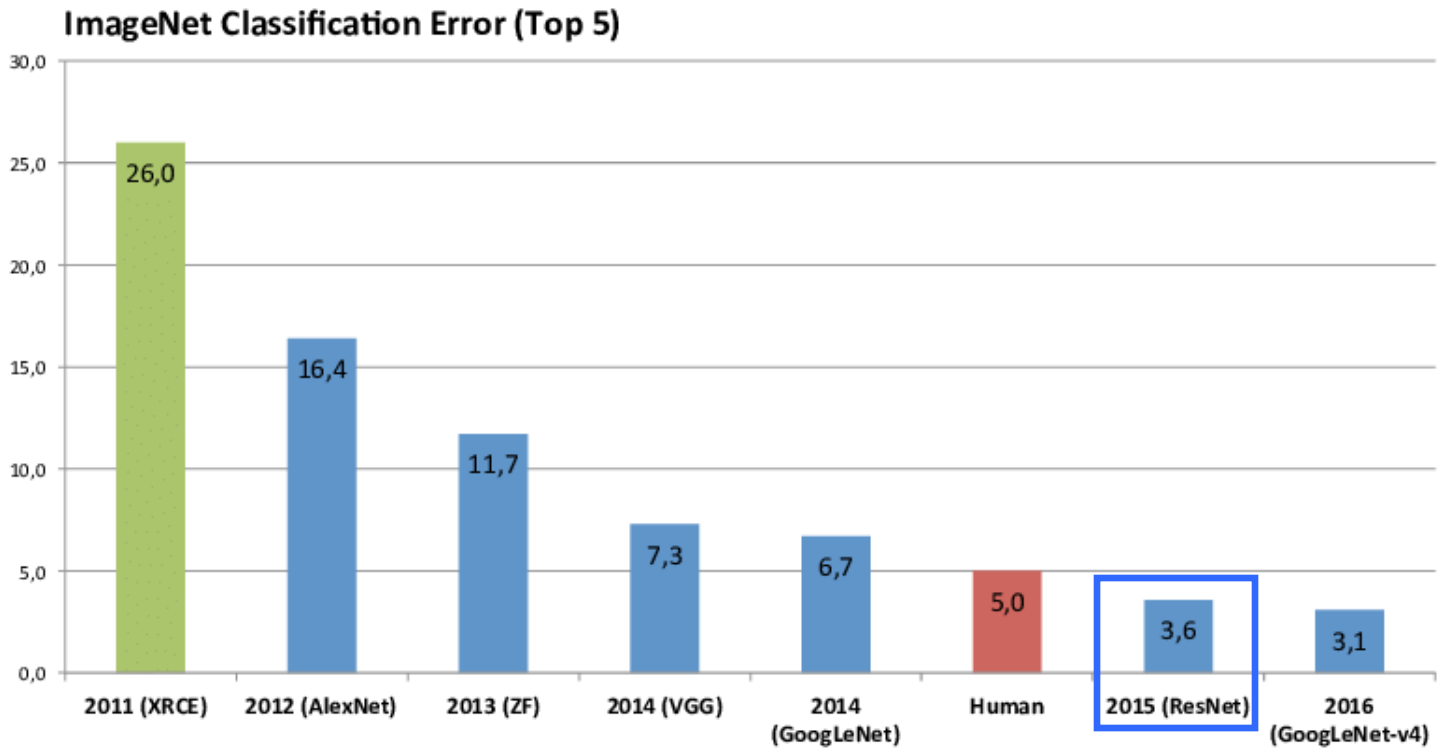


https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

CNN Architectures

ResNet (2015)



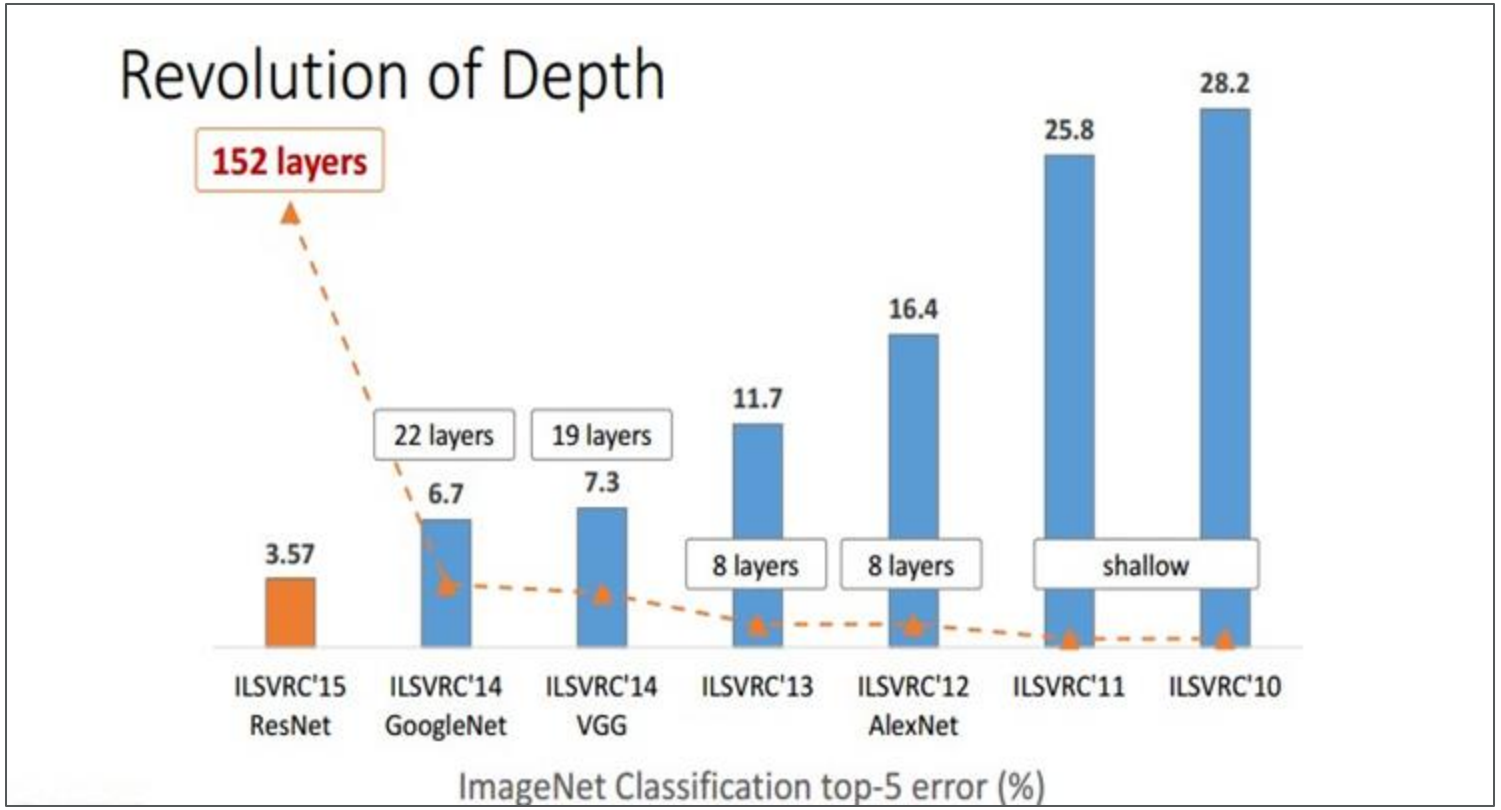
~3.6% top 5 error in ILSVRC 2015, lower than human recognition error!

Figure Credit: Zitzewitz, Gustav. "Survey of neural networks in autonomous driving." (2017)

He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

CNN Architectures

ResNet (2015)

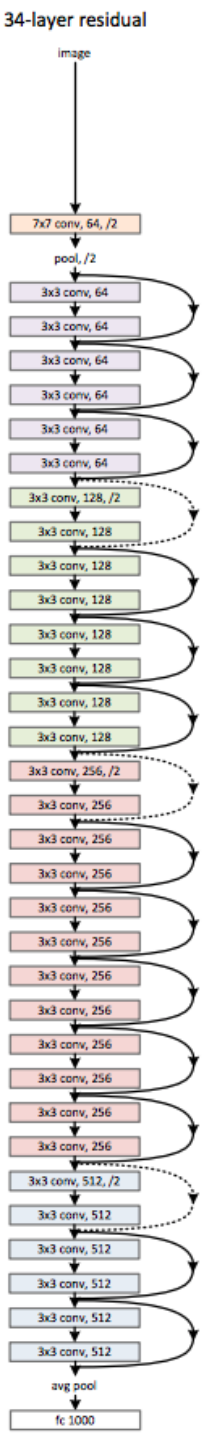


He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

CNN Architectures

ResNet (2015)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

CNN Architectures

ResNet (2015): Summary

Pros

- Greatly reduces the training time and improves accuracy
  - Does not need to fire all neurons in every epoch
  - Once a feature is learnt, it focuses on learning newer features.
- Solved the vanishing gradient problem experienced by VGG

Overview

Next Lecture..

Locally Connected Layer

Introduction to Convolutional Neural Networks

Layers used to build Convolutional Neural Networks

Examples of Deep CNNs Architectures

Training CNNs

- Gradient Descent
- Momentum
- Adagrad
- RMSProp
- ADAM
- BFGS
- L-BFGS

Visualization of Convolutional Neural Networks

Exercise

LeNet

1. Implement LeNet from scratch and train on the Fashion-MNIST dataset. Batch size = 256, learning rate for stochastic gradient descent = 0.9, epochs = 10, using cross entropy loss.
   I. Replace the average pooling with max pooling. What happens?
   II. Try to construct a more complex network based on LeNet to improve its accuracy.
      I. Adjust the convolution window size.
      II. Adjust the number of output channels.
      III. Adjust the activation function (e.g., ReLU).
      IV. Adjust the number of convolution layers.
      V. Adjust the number of fully connected layers.
      VI. Adjust the learning rates and other training details (e.g., initialization and number of epochs.)
   III. Try out the improved network on the original MNIST dataset.
   IV. Display the activations of the first and second layer of LeNet for different inputs (e.g., sweaters and coats).

Exercise

AlexNet

1. Implement AlexNet from scratch and train on the Fashion-MNIST dataset (up-sample images to 224 x 224). Batch size = 128, learning rate for stochastic gradient descent = 0.01, epochs = 10, using cross entropy loss.
   I. Try increasing the number of epochs. Compared with LeNet, how are the results different? Why?
   II. AlexNet may be too complex for the Fashion-MNIST dataset.
      I. Try simplifying the model to make the training faster, while ensuring that the accuracy does not drop significantly.
      II. Design a better model that works directly on 28×28 images.
   III. Modify the batch size, and observe the changes in accuracy and GPU memory.
   IV. Analyze computational performance of AlexNet.
      I. What is the dominant part for the memory footprint of AlexNet?
      II. What is the dominant part for computation in AlexNet?
      III. How about memory bandwidth when computing the results?
   V. Apply dropout and ReLU to LeNet-5. Does it improve? How about preprocessing?

Exercise

VGG

1. Write a function to implement a VGG Block with the following signature:

   ```
   def vgg_block(num_convs, in_channels, out_channels)
   ```

2. Write a function that uses vgg_block to implement the VGG architecture.
3. Train VGG-11 on Fashion-MNIST. Batch size = 128, learning rate for stochastic gradient descent = 0.05, epochs = 10, using cross entropy loss.
   I. When printing out the dimensions of the layers we only saw 8 results rather than 11. Where did the remaining 3 layer information go?
   II. Compared with AlexNet, VGG is much slower in terms of computation, and it also needs more GPU memory. Analyze the reasons for this.
   III. Try changing the height and width of the images in Fashion-MNIST from 224 to 96. What influence does this have on the experiments?
   IV. Refer to Table 1 in the VGG paper [Simonyan & Zisserman, 2014] to construct other common models, such as VGG-16 or VGG-19.

Exercise

GoogLeNet

1. Write a class to implement an Inception Block
2. Implement GoogLeNet from scratch using the Inception block
3. Train GoogLeNet on Fashion-MNIST. Batch size = 128, learning rate for stochastic gradient descent = 0.1, epochs = 10, using cross entropy loss.
    I. There are several iterations of GoogLeNet. Try to implement and run them. Some of them include the following:
        I. Add a batch normalization layer [Ioffe & Szegedy, 2015], as described later in Section 7.5.
        II. Make adjustments to the Inception block [Szegedy et al., 2016].
        III. Use label smoothing for model regularization [Szegedy et al., 2016].
        IV. Include it in the residual connection [Szegedy et al., 2017], as described later in Section 7.6.
    II. What is the minimum image size for GoogLeNet to work?
    III. Compare the model parameter sizes of AlexNet & VGG with GoogLeNet. How does the latter network architecture significantly reduce the model parameter size?

Exercise

ResNet

1. Write a class to implement a Residual Block
2. Implement ResNet from scratch using the Residual block
3. Train ResNet on Fashion-MNIST. Batch size = 256, learning rate for stochastic gradient descent = 0.05, epochs = 10, using cross entropy loss.
   I. What are the major differences between the Inception block in Fig. 7.4.1 and the residual block? After removing some paths in the Inception block, how are they related to each other?
   II. Refer to Table 1 in the ResNet paper [He et al., 2016a] to implement different variants.
   III. For deeper networks, ResNet introduces a “bottleneck” architecture to reduce model complexity. Try to implement it.
   IV. In subsequent versions of ResNet, the authors changed the “convolution, batch normalization, and activation” structure to the “batch normalization, activation, and convolution” structure. Make this improvement yourself. See Figure 1 in [He et al., 2016b] for details.
   V. Why can’t we just increase the complexity of functions without bound, even if the function classes are nested?

Terminology

- $Distribution$: (sample space) the set of all possible samples
- $Dataset$: a set of samples drawn from a distribution
- $Batch$: a subset of samples drawn from the dataset
- $Sample$: a single data object represented as a set of features
- $Feature$: value of a single attribute, property, in a sample. Could be numeric or categorical.

Appendix A: Notations

- x_i: a single feature
- $\boldsymbol{x}_i$: feature vector (a data sample)
- $\boldsymbol{x}_{:,i}$: feature vector of all data samples
- $\boldsymbol{X}$: matrix of feature vectors (dataset)
- N: number of data samples
- P: number of features in a feature vector
- α: learning rate
- Bold letter/symbol: vector
- Bold capital letters/symbol: matrix