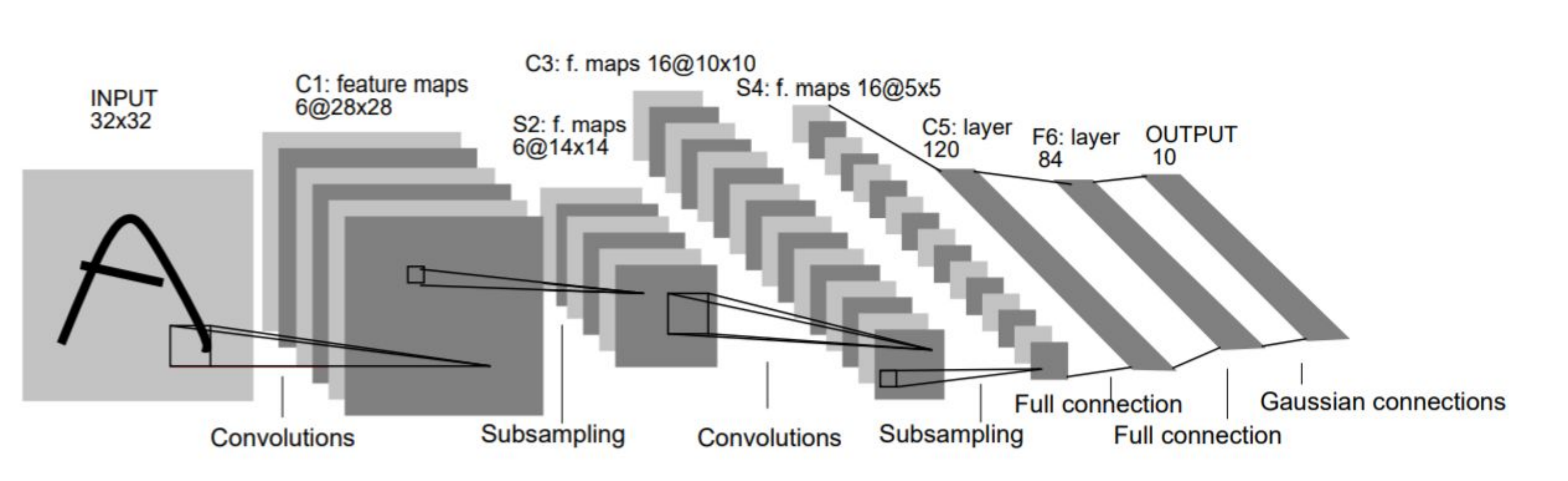
Convolutional Neural Networks



Problems with traditional neural networks

- Previously, you have seen neural networks where every neuron in a layer is connected to every neuron in the next layer.
- These fully-connected layers can work well for layers with a small number of neurons, but as the size of the input and number of neurons increases, the number of weights becomes prohibitively expensive
- Fully connected neural networks would have too many weights if we were to try processing images with them
    - consider a 256 by 256 color image—there would be (256×256×3) = 196608 weights for *each* neuron in the first layer
    - Too many weights can also lead to overfitting training data

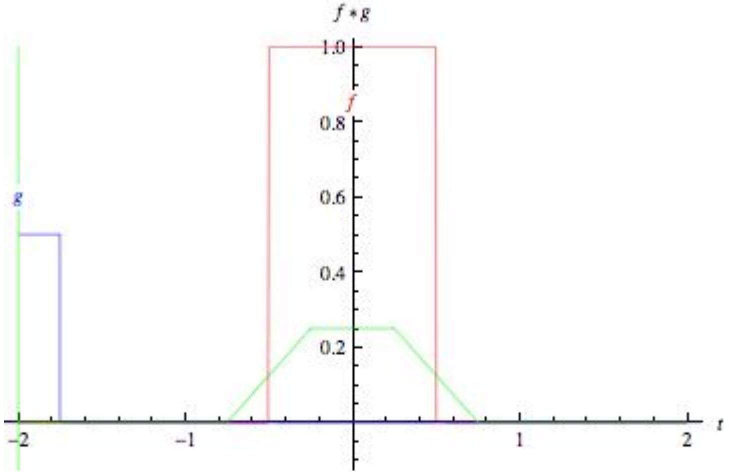
Convolutional Neural Networks (CNNs)

- Designed to process images—makes assumptions about image processing, and uses this to reduce weights
    - Features can be extracted locally: whether there is an edge at a location depends on the pixels around that location, not those on the other side of the image
    - The same weights used to extract a feature at some location are able to do so at any other location: an edge at one location is similar to one somewhere else
- Use two additional layers: convolutional layers and pooling layers
- Are still neural networks—can still use backpropagation to learn weights efficiently

Convolutions

- A convolution is an operation between two functions that blends the functions together based on their overlap
- Defined with the following formula:

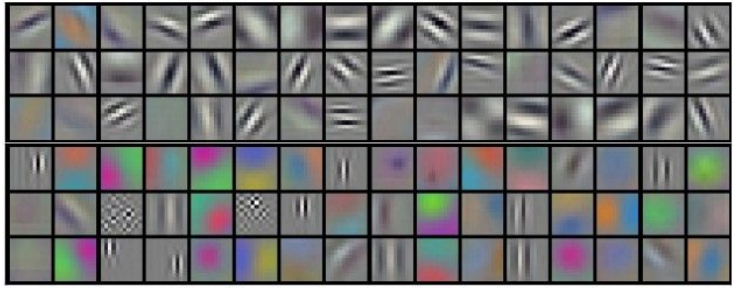
$$(f * g)[t] = \sum_{\tau=-\infty}^{\infty} f[\tau]g[t-\tau]$$



Wolfram animation of a convolution

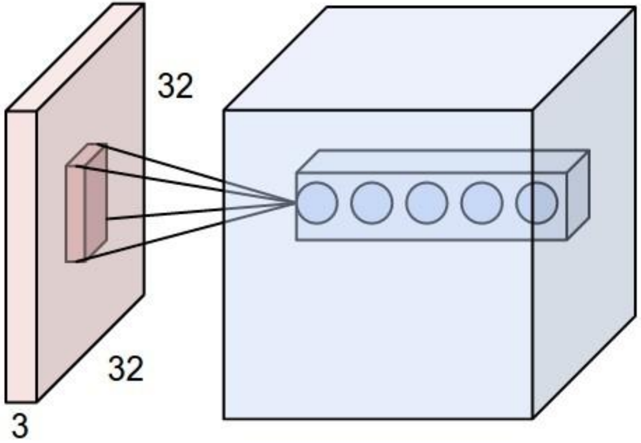
- Intuitively, one function/matrix is flipped and moved across the other, where a dot product is performed at each location, which is the output of the convolution at that location
- Similar to cross correlation, but with one function/matrix flipped
  - Most CNNs don't actually flip the filters, so technically they use cross correlation
- Also used in determining the output of a linear time invariant system for a given input signal (16B)

Convolutional layer

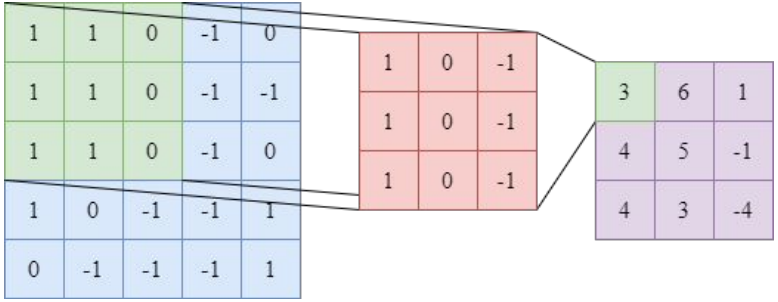


Visualized filters from AlexNet

- Consists of one or more filters (kernel) used to extract features from images
- The filter is slid across and convolved with the part of the image it’s over
- There can be multiple filters that each attempt to find different features
  - e.g. horizontal/vertical lines, colors
- Neurons only have nonzero weights for the inputs near them
  - weights/neuron now only the size of the filter



Convolutional layer

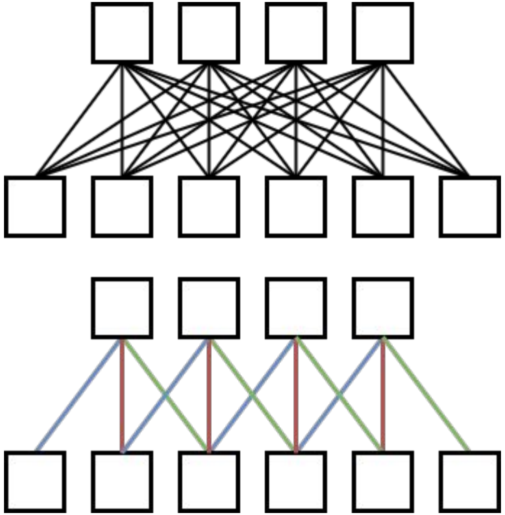


1	1	0	-1	0
1	1	0	-1	-1
1	1	0	-1	0
1	0	-1	-1	1
0	-1	-1	-1	1

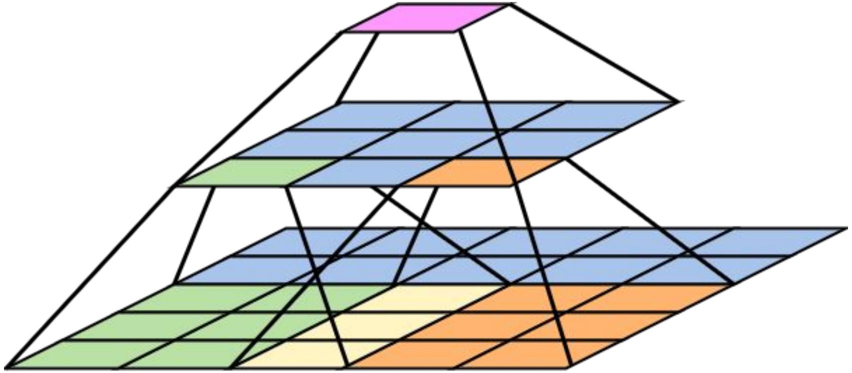
1	0	-1
1	0	-1
1	0	-1

3	6	1
4	5	-1
4	3	-4

- Neurons only have nonzero weights for the inputs near them
  - weights/neuron now only the size of the filter
  - can be viewed as a fully connected layer with the weights from all distant (less influential) neurons set to zero
- Weight sharing: all output neurons use the same weights for inputs in the same relative positions
  - The number of weights is now just the size of the filter



Convolutional layer



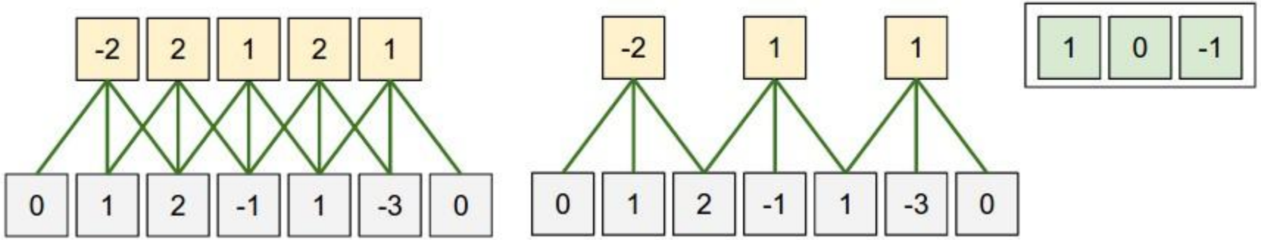
Hyperparameters:

- Filter size: size of the filter.
    - usually small (3×3×3, 5×5×3), as larger filters require weights and more computation
    - stacked convolutional/pooling layers can result in a larger effective receptive field
- Stride
- Zero padding
- Depth

Convolutional layer

Hyperparameters:

- Filter size
- Stride: how far the filter moves between measurements
    - A stride of 1 makes a measurement for every input neuron, a stride of 2 does so for every other input neuron
    - Strides greater than 1 reduce the size of the output layer (reduces computational costs)
- Zero padding
- Depth

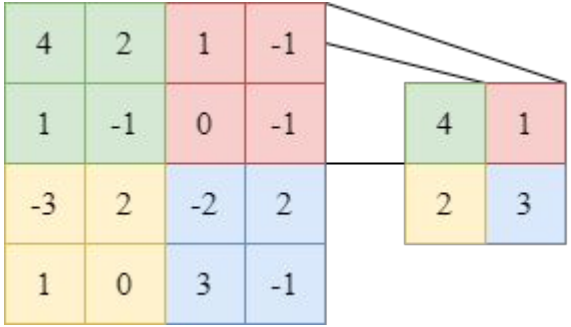


Convolutional layer

Hyperparameters:

- Filter size
- Stride
- Zero padding: number of zeros to place around the edges of the input
    - Used to keep layer sizes consistent (filter sizes > 1 can’t fit on edges without extra zeros)
    - Makes sure that strides fit evenly over the input
        - The size of the output layer $(W - F + 2P)/S + 1$ must be an integer
- Depth: how many filters there are
    - Number of different features searched for
    - More filters results in more sets of weights

Pooling layer



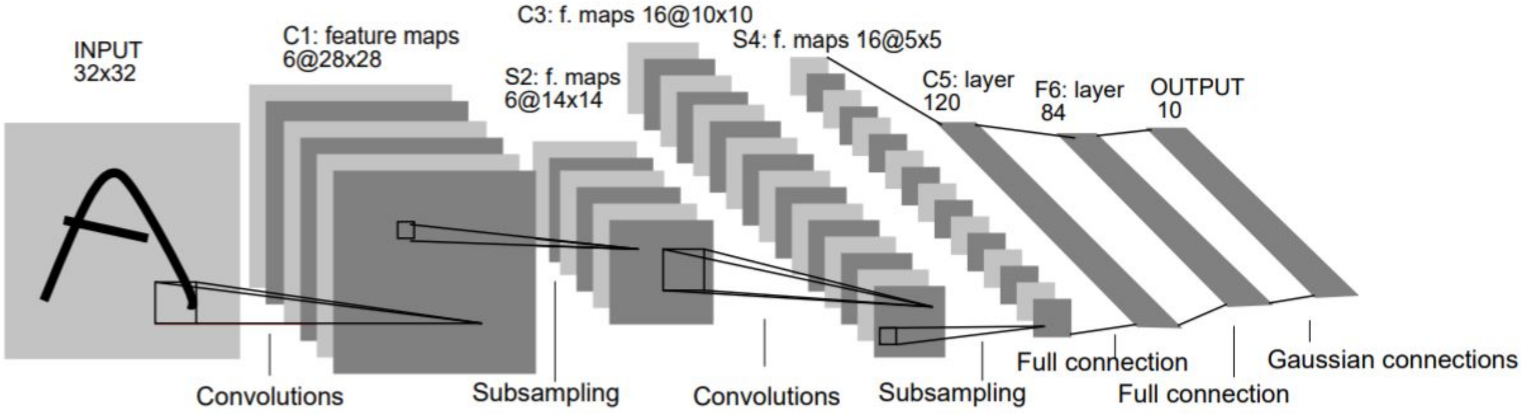
4	2	1	-1
1	-1	0	-1
-3	2	-2	2
1	0	3	-1

4	1
2	3

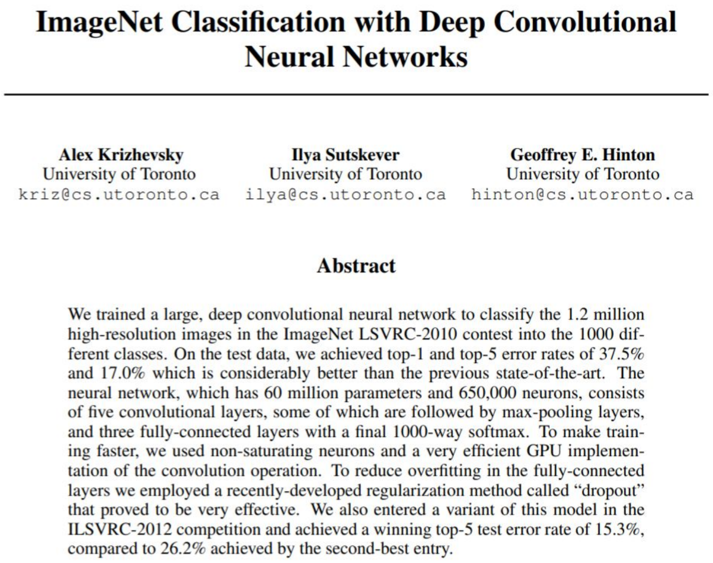
- Reduces the size of the layers while preserving most of the data
- Also uses a filter size and stride (usually 2 by 2 filter and stride of 2), but uses a max function or average instead of a dot product
  - A larger filter will result in more information lost (2 by 2 loses 3/4, 3 by 3 loses 8/9)
  - A smaller stride or filter wouldn't reduce the input size (why?)

Significant CNN Architectures

LeNet-5 (LeCun et al, 1998)



AlexNet (Krizhevsky et al, 2012)



ImageNet Classification with Deep Convolutional Neural Networks

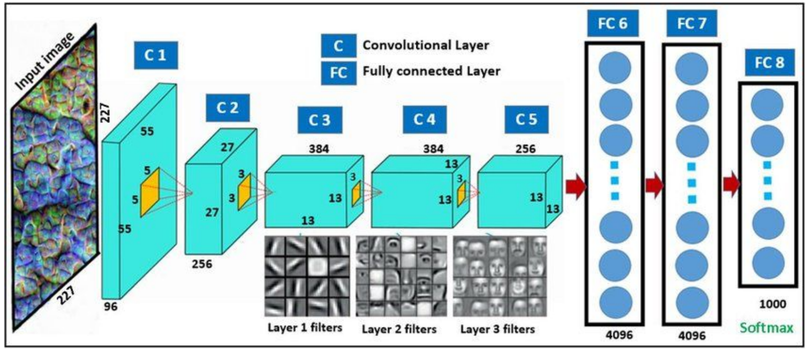
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

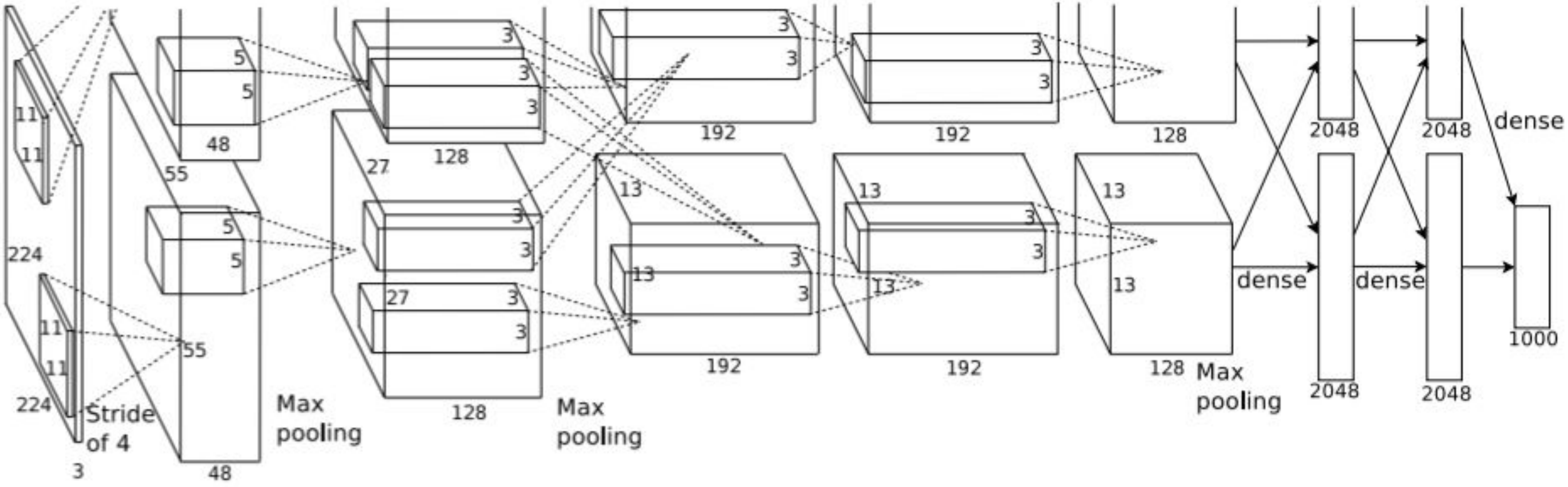
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.



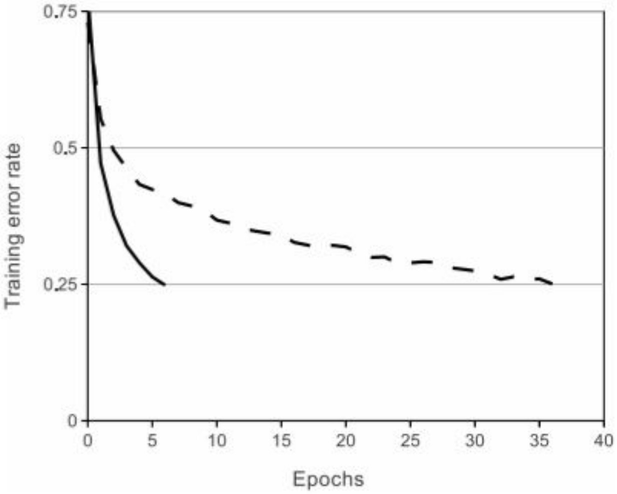


224
224
3
11
11
11
11
Stride of 4
55
55
48
48
5
5
5
5
Max pooling
27
27
128
128
3
3
3
Max pooling
13
13
13
192
192
192
192
3
3
13
13
128
128
Max pooling
dense
dense
dense
2048
2048
2048
2048
1000

Key Points

- Usage of ReLu, a non-saturating function, as the activation function. (over tanh and sigmoid)
- Utilized two GPUs
- Data augmentation to reduce overfitting
- Dropout in the first two FC layers to reduce overfitting. Also halves the number of iterations required to converge.

Improved Training Performance

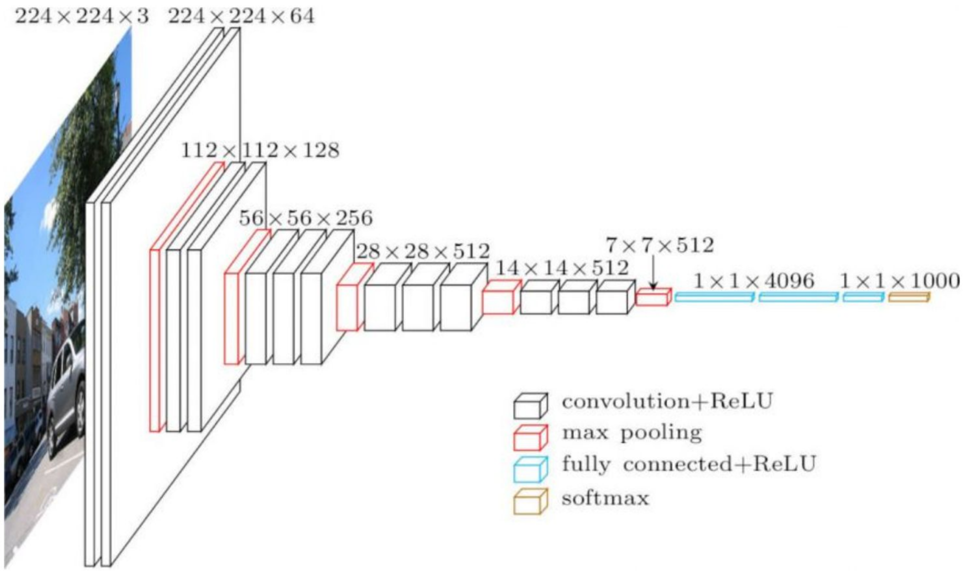


- ReLu (Solid Line) reaches .25 training error rate approximately ~6x faster than Tanh(dotted line) does.

Data Augmentation

- Extracted 224x224 patches out of the 256x256 images, resulting in 2048 more training points. (Why our network has inputs of size 224x224)
- During testing, extracted the 4 corner 224x224 patches and the center patch, along with their horizontal reflections. (total 10 patches)
- Altered the intensity of RGB color channels, utilizing PCA to pertain the important

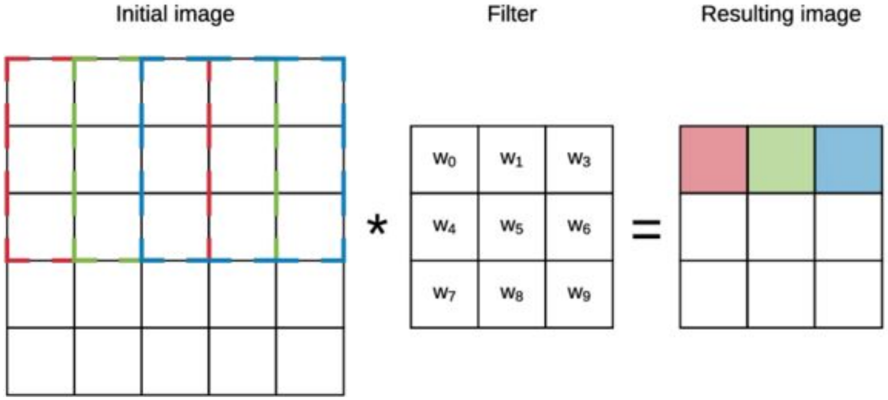
VGGNet (Simonyan & Zisserman, 2014)



VGG16 Architecture

Key Points

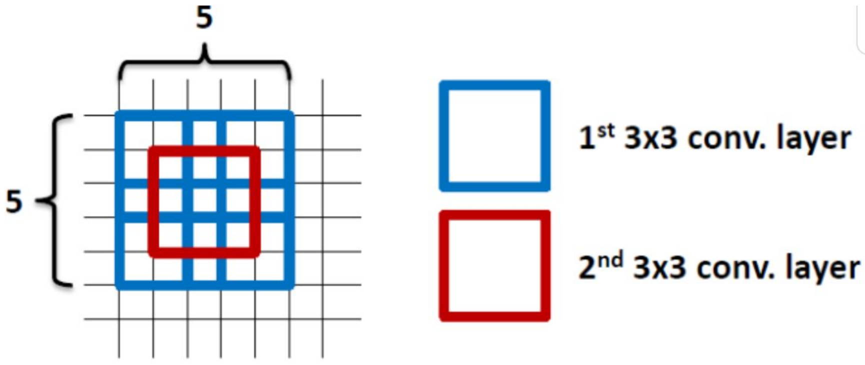
- ImageNet 2014 Runner-Up
- Only uses 3x3 convolution filters



Why 3x3 Filters?

- It is the smallest size that still captures the notion of left, right, up, down, and middle.
- Instead of a single large 7x7 or 11x11 filter, they use multiple 3x3 filters.
- If we apply a 3x3 filter to a 7x7 image, we get a 5x5 matrix. Applying 2 more 3x3 layers results in a 1x1 matrix, which is equivalent to applying a single 7x7 filter.
- Using smaller filters results in more non-linear ReLu layers, which makes our network more discriminative.

Filter Visualization



2 layers of 3×3 filters already covered the 5×5 area

Three 3x3 filters => 27 weights

One 7x7 filter => 49 weights

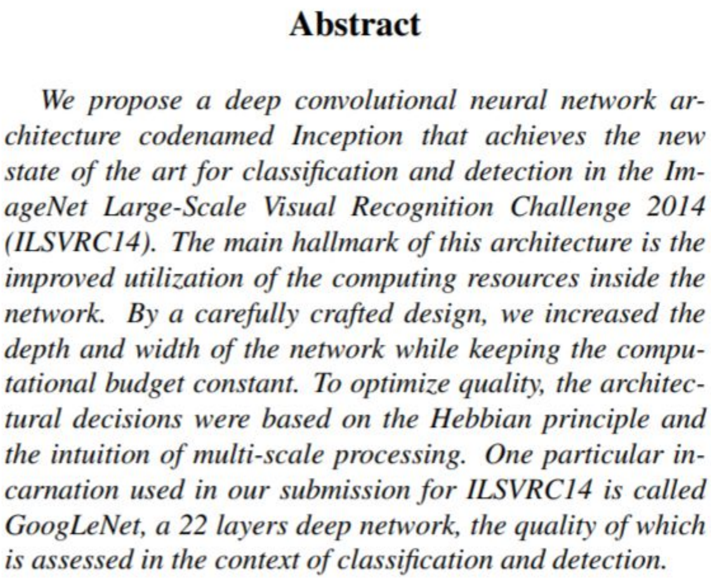
Using only 3x3 conv. layers requires 45% fewer weights.

Two 3x3 filters => 18 weights

One 5x5 filter => 25 weights

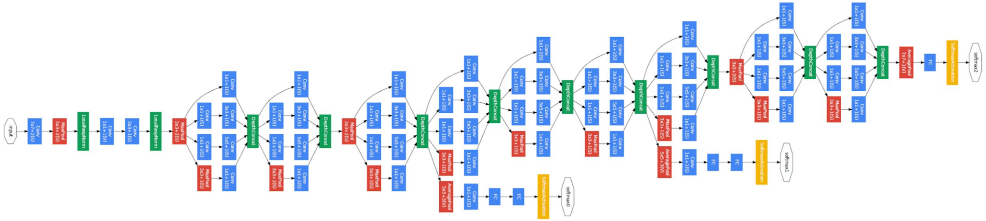
Using only 3x3 layers requires 28% fewer weights

GoogleNet (Szegedy et al, 2014)



Abstract

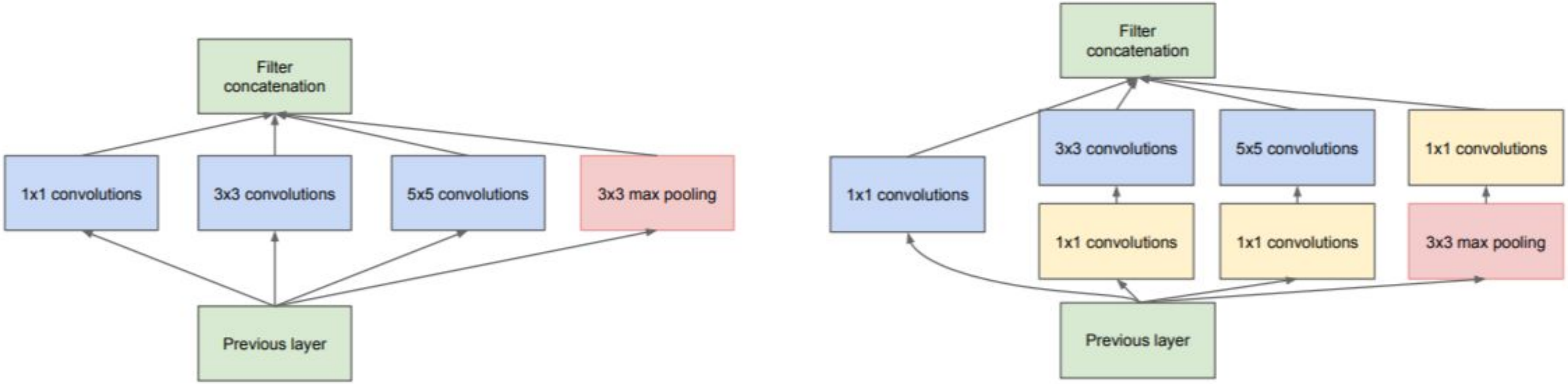
We propose a deep convolutional neural network architecture codenamed Inception that achieves the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.



Key Points

- 22 layers deep, deeper than previous networks.
- Inception Modules
- Average Pooling instead of FC layers, which eliminates a large amount of parameters.

Inception Modules

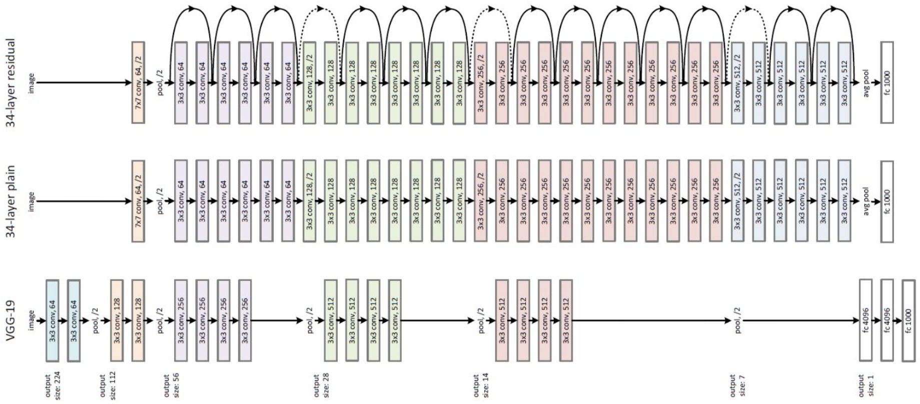


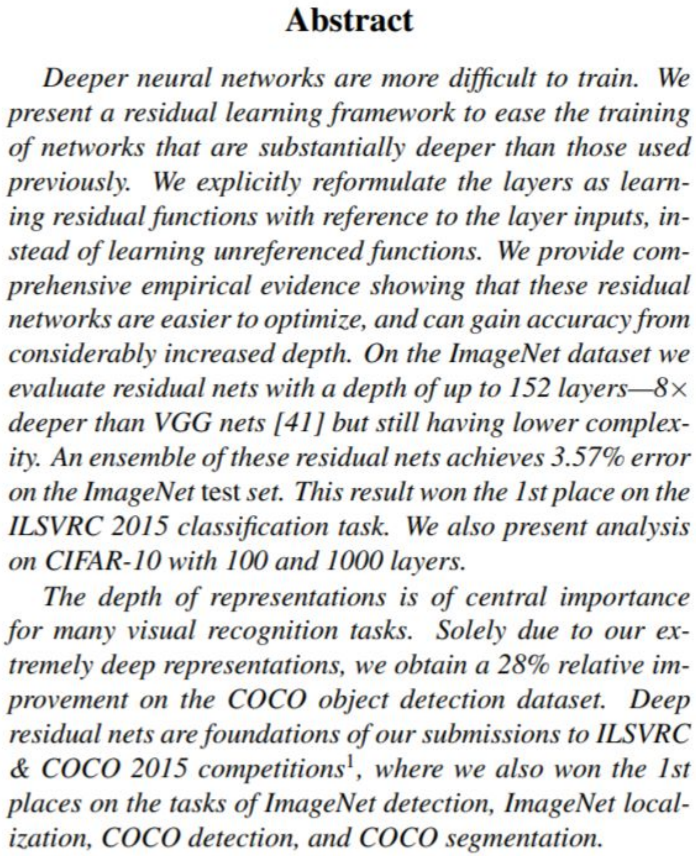
(a) Inception module, naïve version

(b) Inception module with dimensionality reduction

Figure 2: Inception module

ResNet (He et al, 2015)





Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [41] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. *This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.*

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1]*, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.*

Residual Blocks

- If the desired underlying function we want to learn is H(x), the network learns the residual mapping, ie F(x):= H(x) - x. Thus, the output of this network is F(x) + x = H(x)

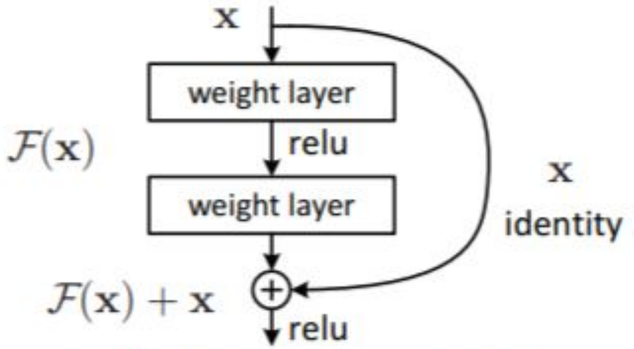


Figure 2. Residual learning: a building block.

Key Points

- Skip connections allow easier learning of simply mappings, such as the identity mapping.
- Speeds up training of deep networks.
- Networks become dynamic: networks can determine how many layers to actually use.
- Solves the vanishing gradient problem.
- Skip connections allow networks to build deeply.

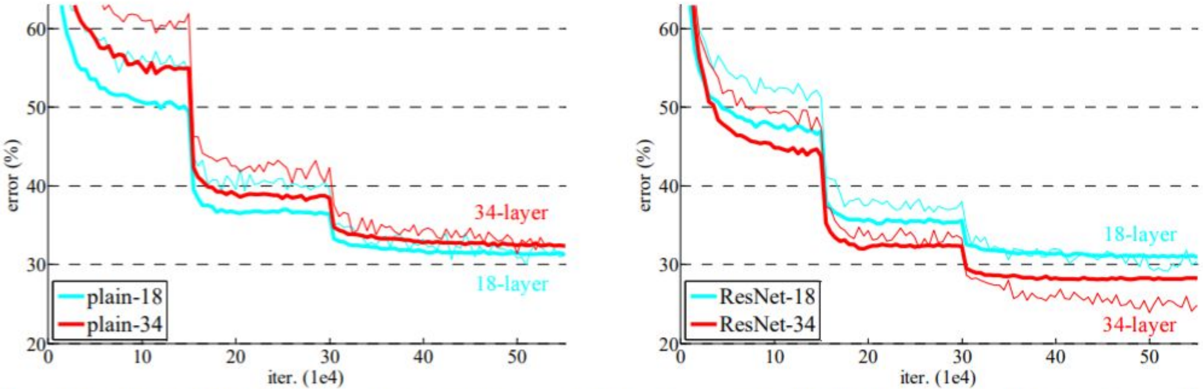
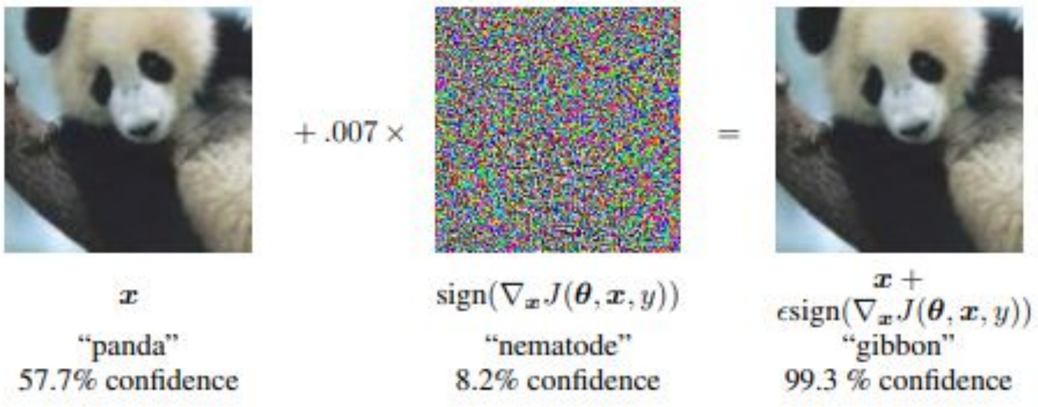


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

This technology allows us to build deeper networks that will still converge.

Adversarial Examples

- Szegedy et al. (2014) has found that most state of the art CNN's are susceptible to adversarial examples.
- Some adversarial examples were indistinguishable to a human eye.
- Hypothesis: Neural Networks are too linear to resist linear perturbations


$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

- Using epsilon = .1, researchers have obtained an "error rate of 87.15% and an average probability of 96.6% assigned to the incorrect labels when using a convolutional maxout network on a preprocessed version of the CIFAR-10."
- Models that are easily optimized are as easily affected by adversarial examples.
- Linear models aren't capable of resisting adversarial perturbations. Only models that include a hidden layer are able to be trained to resist them.
- Can read more here: https://arxiv.org/pdf/1412.6572.pdf