SAIR

Spatial AI & Robotics Lab

CSE 473/573-A

W13B: INTRO TO DEEP LEARNING

Chen Wang

Spatial AI & Robotics Lab

Department of Computer Science and Engineering

University at Buffalo The State University of New York

Many Slides from Kris Kitani

Classic Classification Pipeline

Input → **encoder** → **decoder** → **output**

image(s)

Extract features

Build descriptor

Optimizer

semantic label

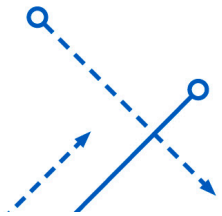
e.g., SIFT, Bag of visual words

e.g., KNN, SVM, Naive Bayes

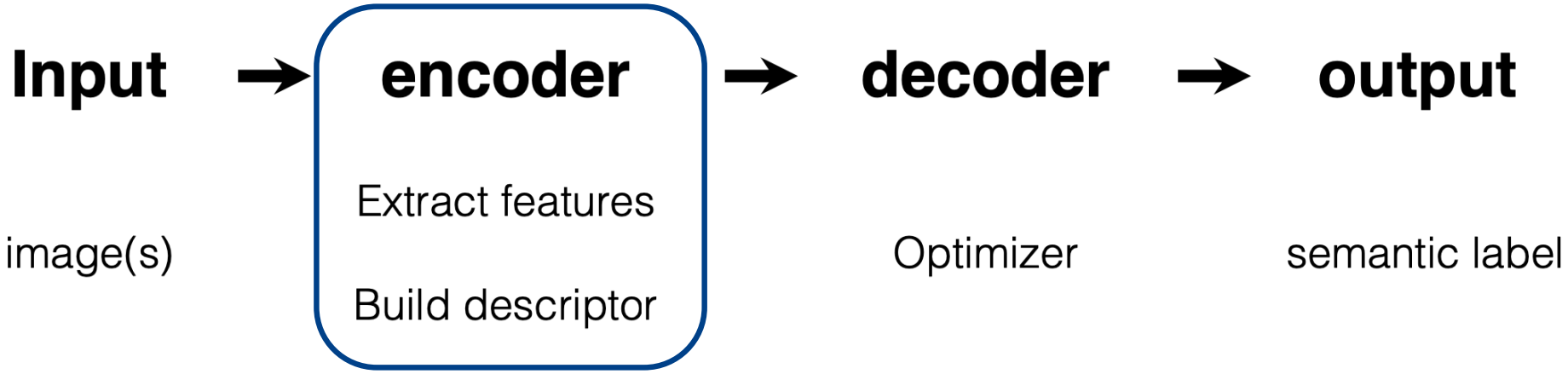
Designed by you

Painful.

Learned from the data



Classic Classification Pipeline



Can you think of a way to learn this part too?

*How can we learn it all '**end to end**?'*

End-to-end Learning

Input → **encoder** → **decoder** → **output**

image(s)

Extract features

Build descriptor

Optimizer

semantic label

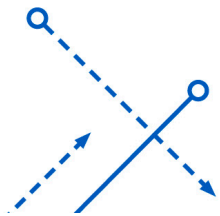
Convolutional Neural Layers

Fully connected layers

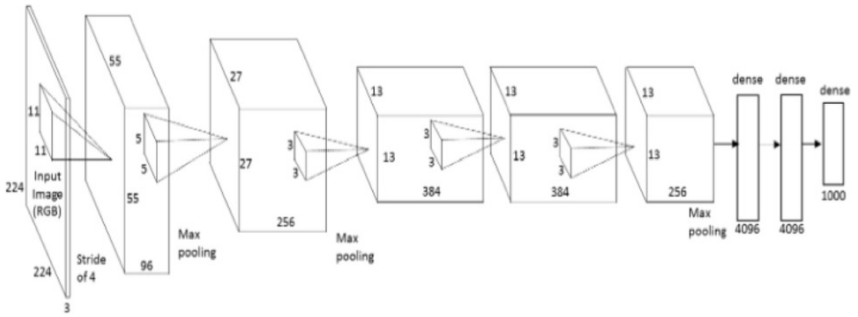
Deep Network

Use gradient descent to learn everything

(except for the architecture, which you have to design. painful.)

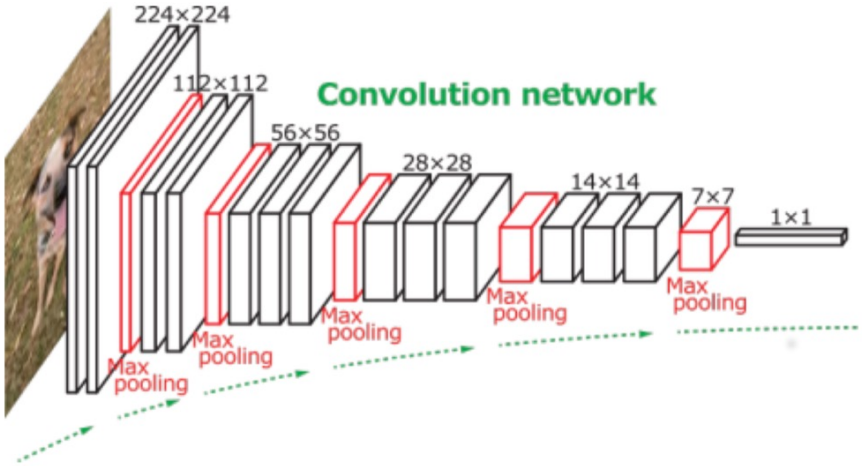


Popular CNN architectures

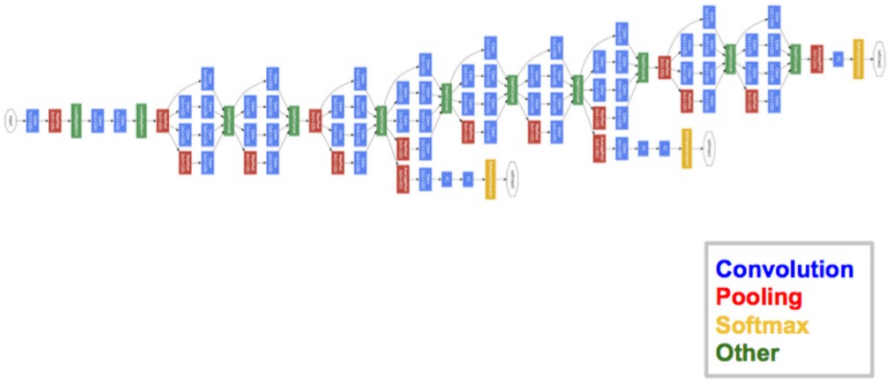


AlexNet

ResNet

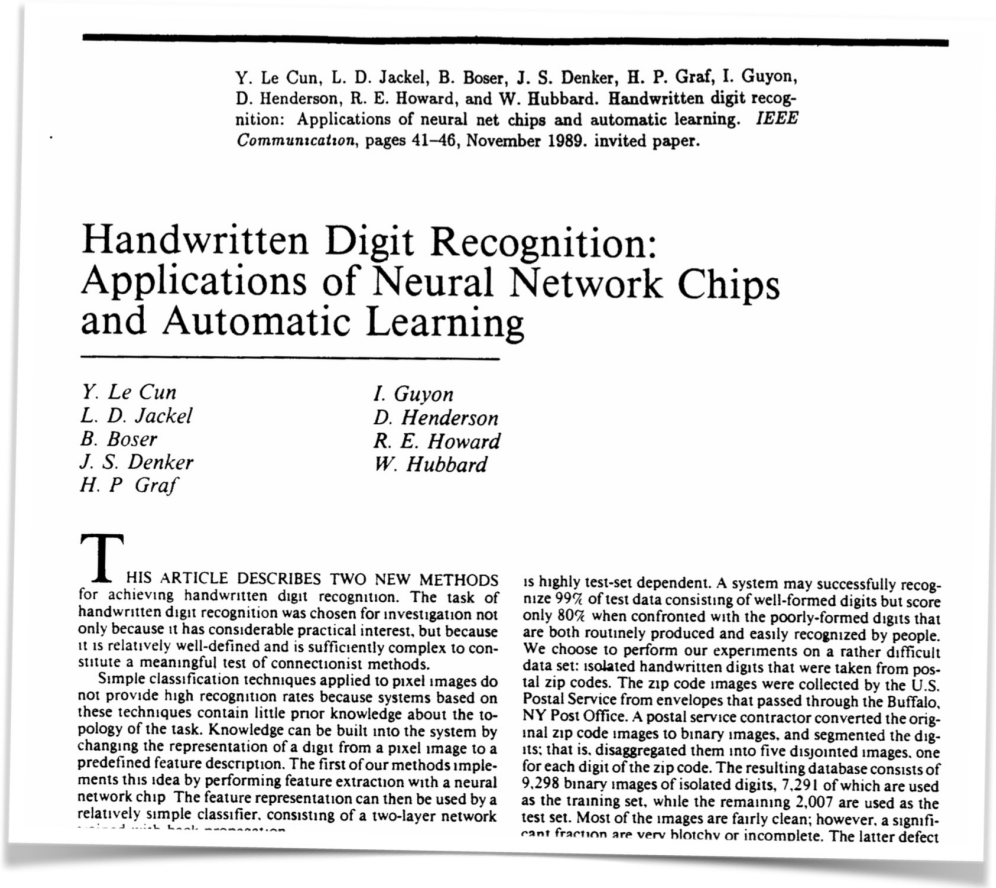


VGG



GoogLeNet

The origin …



Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural net chips and automatic learning. *IEEE Communication*, pages 41–46, November 1989. invited paper.

Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning

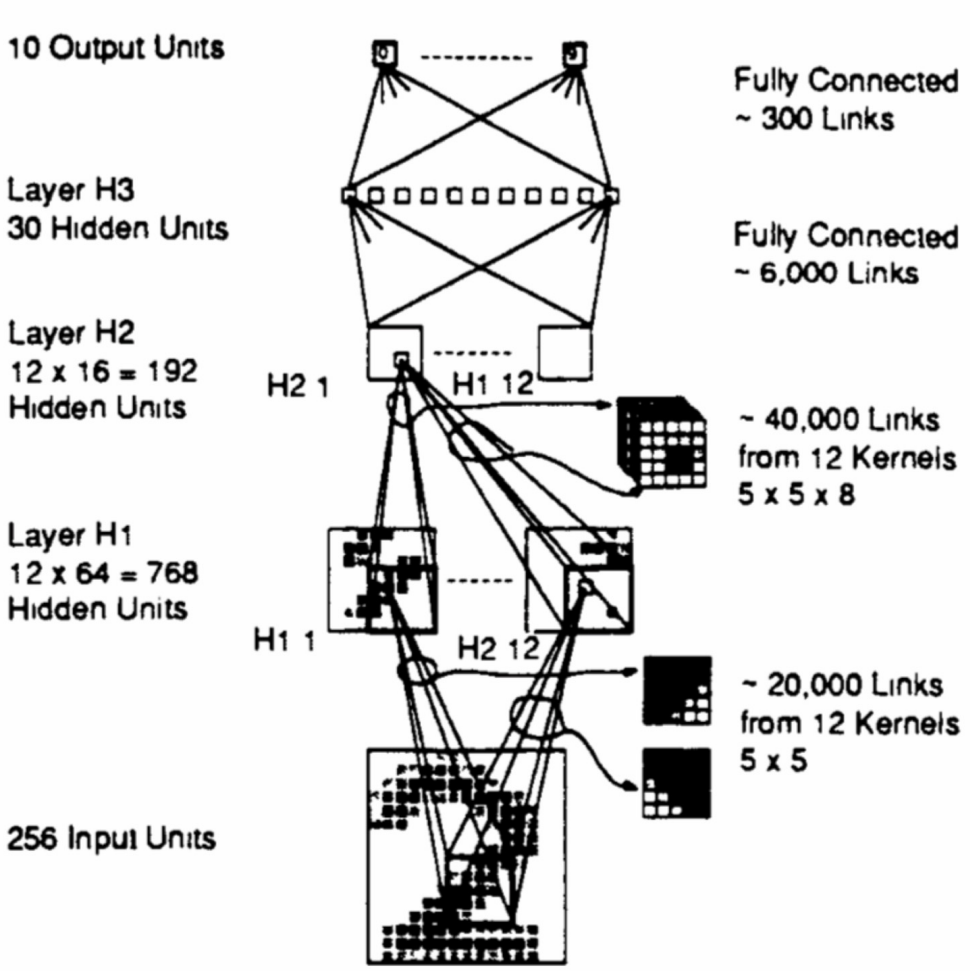
Y. Le Cun
L. D. Jackel
B. Boser
J. S. Denker
H. P Graf

I. Guyon
D. Henderson
R. E. Howard
W. Hubbard

THIS ARTICLE DESCRIBES TWO NEW METHODS for achieving handwritten digit recognition. The task of handwritten digit recognition was chosen for investigation not only because it has considerable practical interest, but because it is relatively well-defined and is sufficiently complex to constitute a meaningful test of connectionist methods.

Simple classification techniques applied to pixel images do not provide high recognition rates because systems based on these techniques contain little prior knowledge about the topology of the task. Knowledge can be built into the system by changing the representation of a digit from a pixel image to a predefined feature description. The first of our methods implements this idea by performing feature extraction with a neural network chip. The feature representation can then be used by a relatively simple classifier, consisting of a two-layer network

is highly test-set dependent. A system may successfully recognize 99% of test data consisting of well-formed digits but score only 80% when confronted with the poorly-formed digits that are both routinely produced and easily recognized by people. We choose to perform our experiments on a rather difficult data set: isolated handwritten digits that were taken from postal zip codes. The zip code images were collected by the U.S. Postal Service from envelopes that passed through the Buffalo, NY Post Office. A postal service contractor converted the original zip code images to binary images, and segmented the digits; that is, disaggregated them into five disjointed images, one for each digit of the zip code. The resulting database consists of 9,298 binary images of isolated digits, 7,291 of which are used as the training set, while the remaining 2,007 are used as the test set. Most of the images are fairly clean; however, a significant fraction are very blotchy or incomplete. The latter defect



1989!

This led to a famous model known as...

LetNet-5

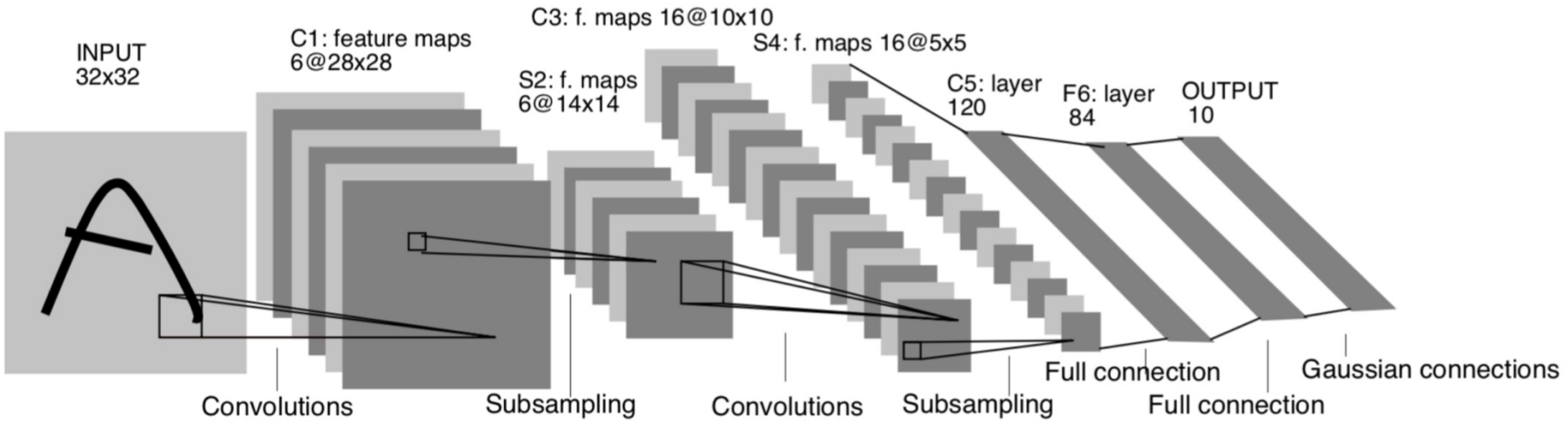


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

as the feature maps in the previous layer. The trainable coefficient and bias control the effect of the sigmoid non-linearity. If the coefficient is small, then the unit operates in a quasi-linear mode, and the sub-sampling layer merely blurs the input. If the coefficient is large, sub-sampling units can be seen as performing a "noisy OR" or a "noisy

B. *LeNet-5*

This section describes in more detail the architecture of LeNet-5, the Convolutional Neural Network used in the experiments. LeNet-5 comprises 7 layers, not counting the input, all of which contain trainable parameters (weights).

LeNet

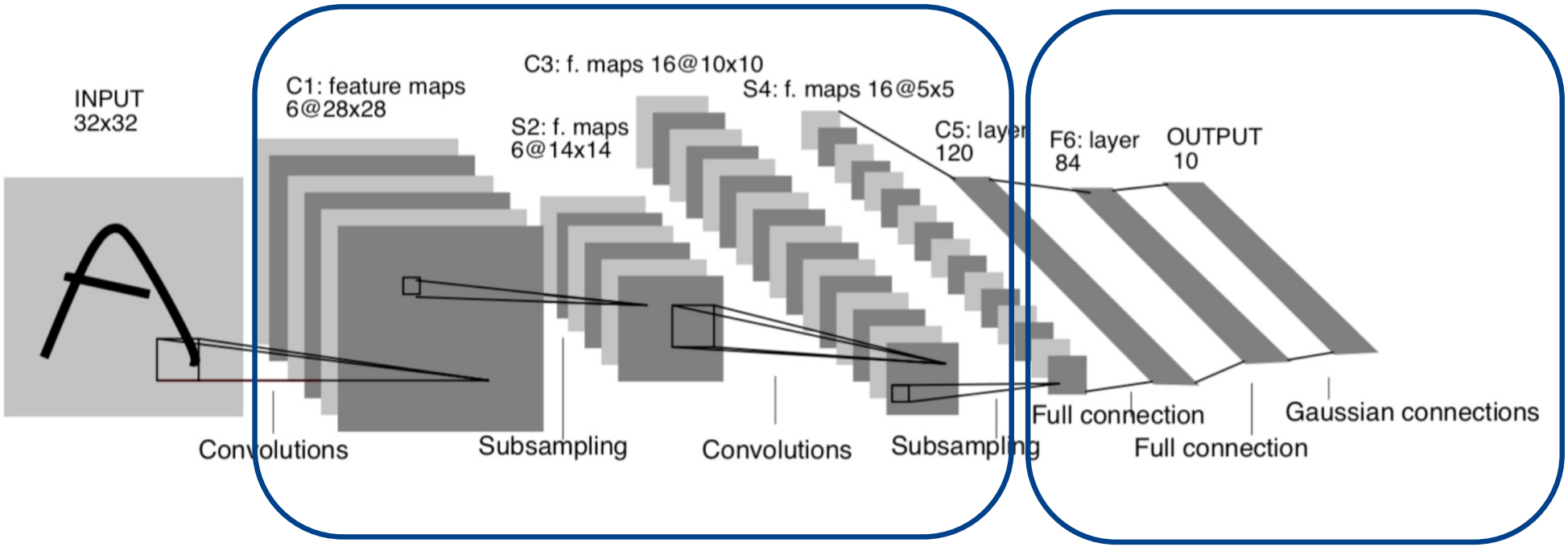


Image input

Encoder

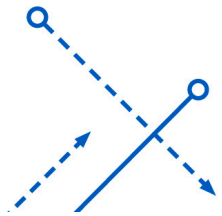
Extracts a representational vector of the image.

Takes place of SIFT and BOW

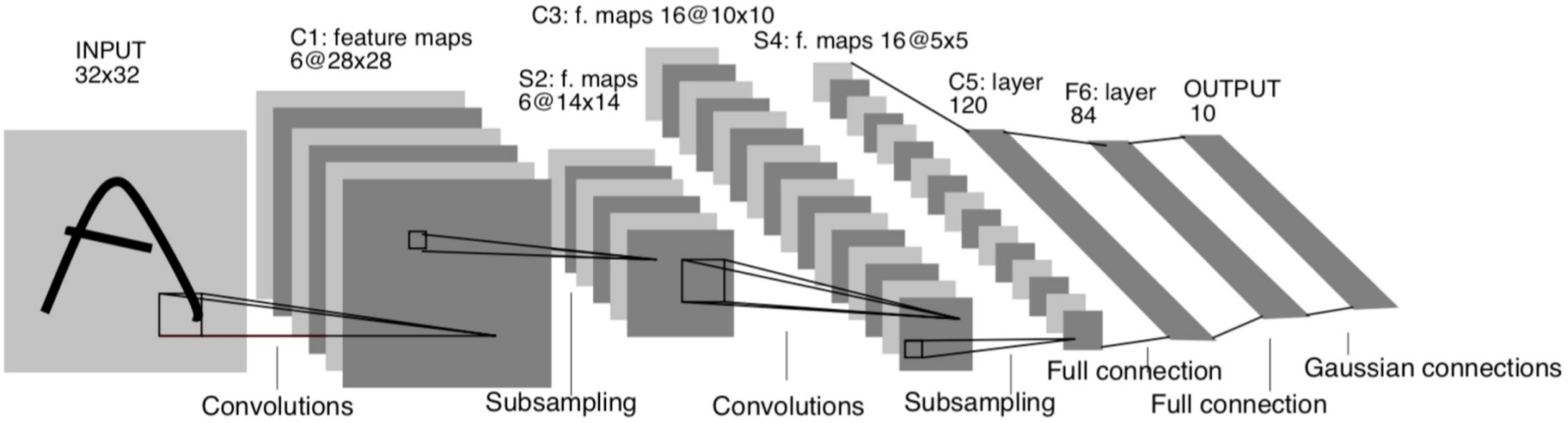
Decoder

Maps vector to categorical probabilities.

Just an MLP

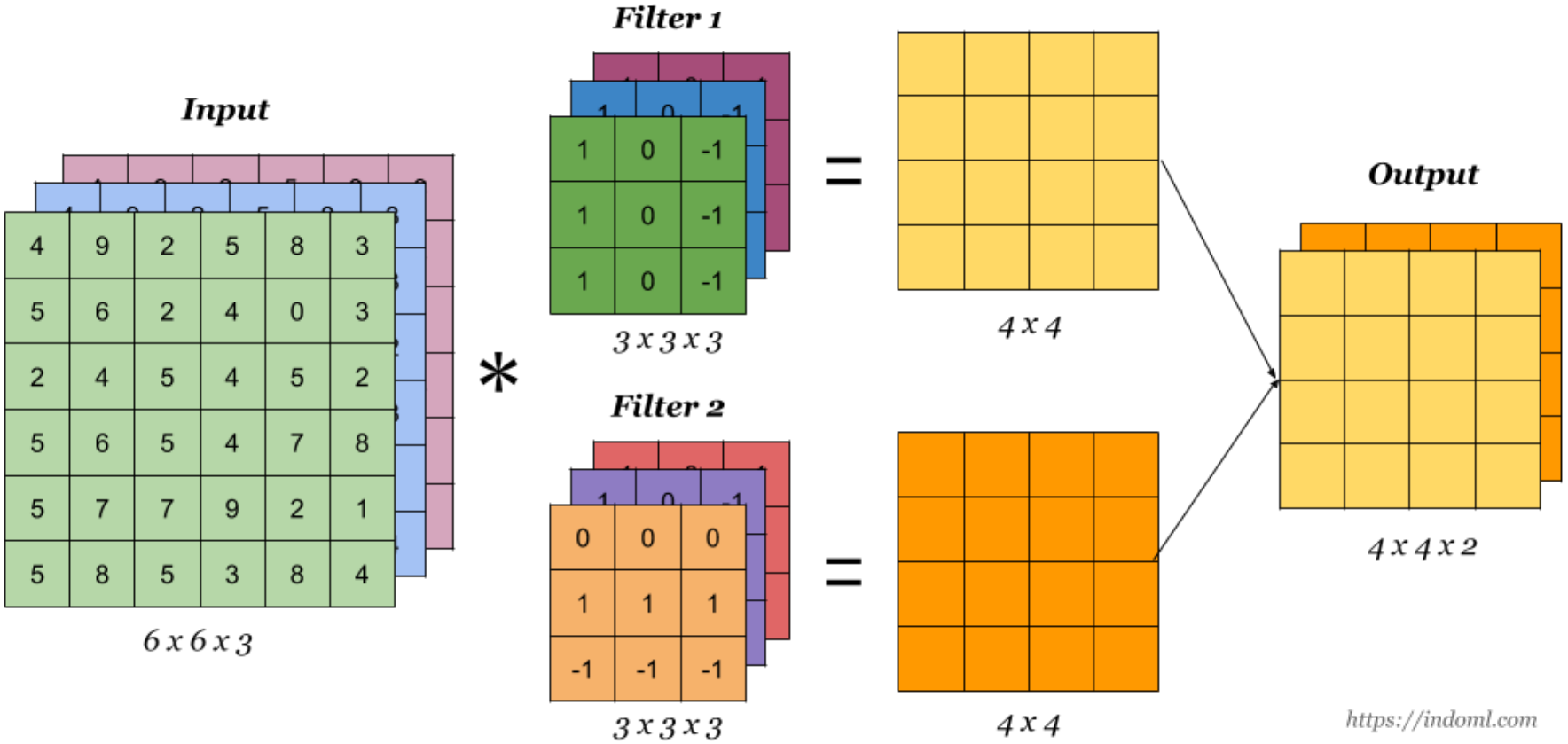


LeNet



- Input image size is fixed, what do you do if image is too big?
  - Convolutions with small receptive field.
- How do convolution layer work?

Convolutional Layer



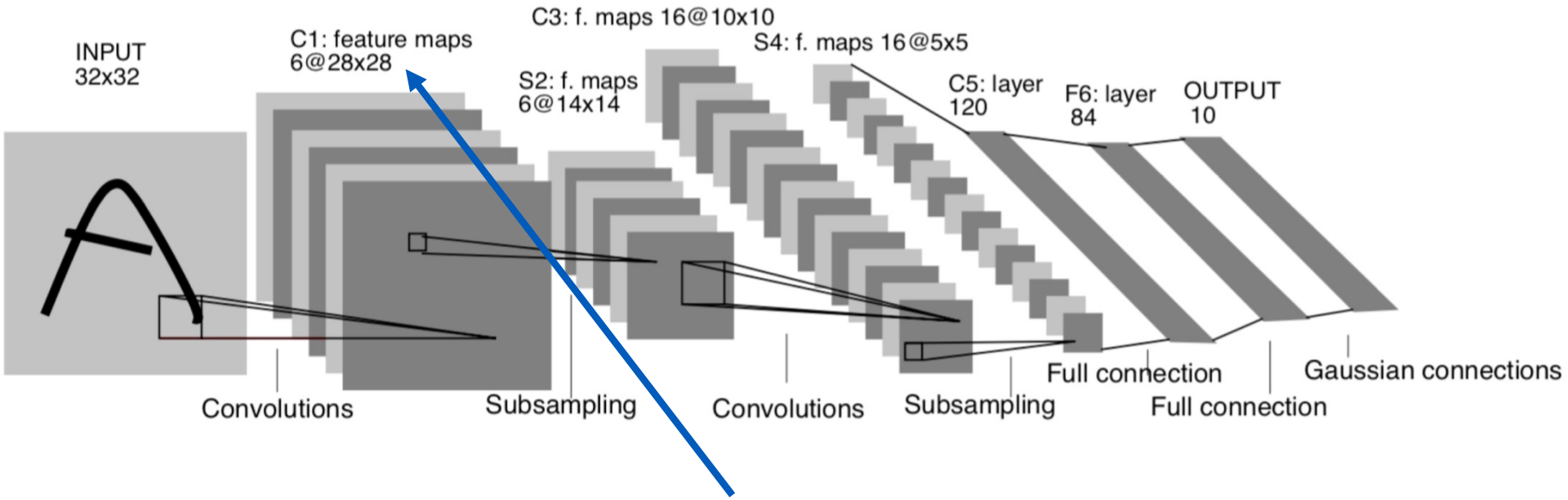
Examples:

```
>>> # With square kernels and equal stride
>>> filters = torch.randn(8, 4, 3, 3)
>>> inputs = torch.randn(1, 4, 5, 5)
>>> F.conv2d(inputs, filters, padding=1)
```

What is shape of the output in the left example?

(1, 8, 5, 5)

Feature map



- If the feature map is 28 x 28, what is the size of the convolution kernel?
- If you don't assume border padding, then 5 x 5.
  - Can't compute convolution for 2 pixels on border.
- Why not use a bigger kernel size?

Feature similarity

Feature similarity

.. features also similar across objects

Harris Detector

Recall:

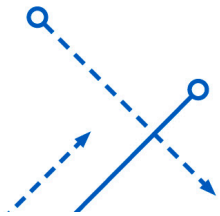
Harris Detector

C.Harris and M.Stephens. “A Combined Corner and Edge Detector.”1988.

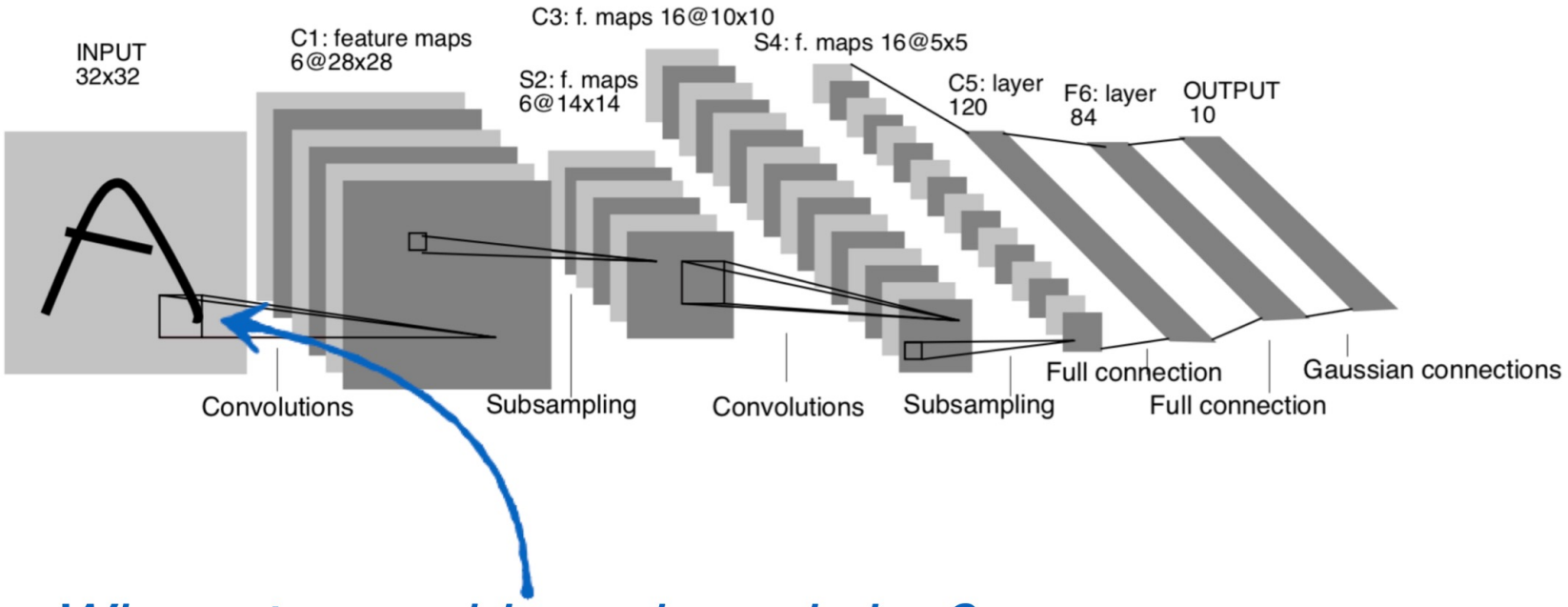
Gradient covariance matrix

$$\begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

Harris corner detector also used
a **small image region P** to compute corners

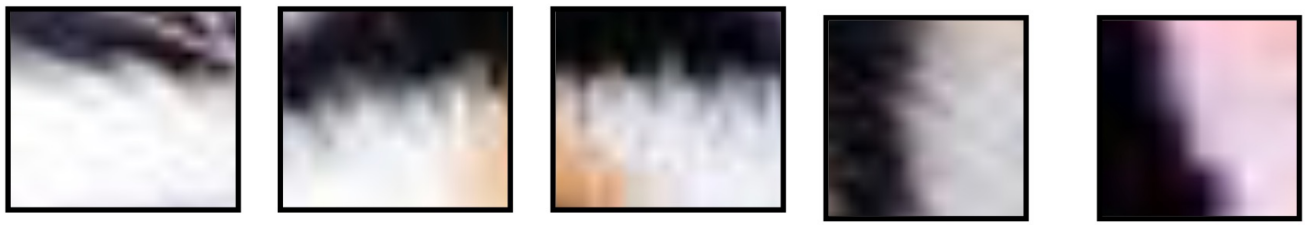


Kernel size



Why not use a bigger kernel size?

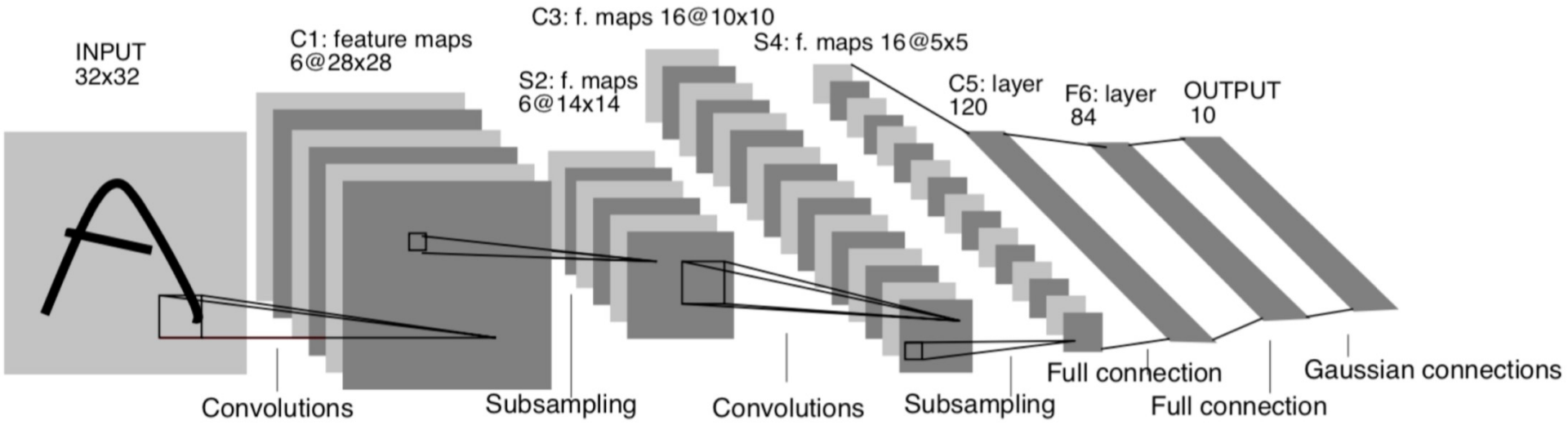
A small kernel can capture common primitive patterns across a wide range of images



Can be reused over the entire image (shared weights)

How many kernels did we use here? (# output channel)

Subsampling



Subsampling reduces the size of feature maps

(actually average pooling with scale and bias, sigmoid function)

$$h[i,j] = \text{sigmoid}\left(\alpha \cdot \sum_{u,v} g[u,v] f[i-u, j-v] - \beta\right)$$

scale; 'Average Pooling'; bias; Box filter; Feature map

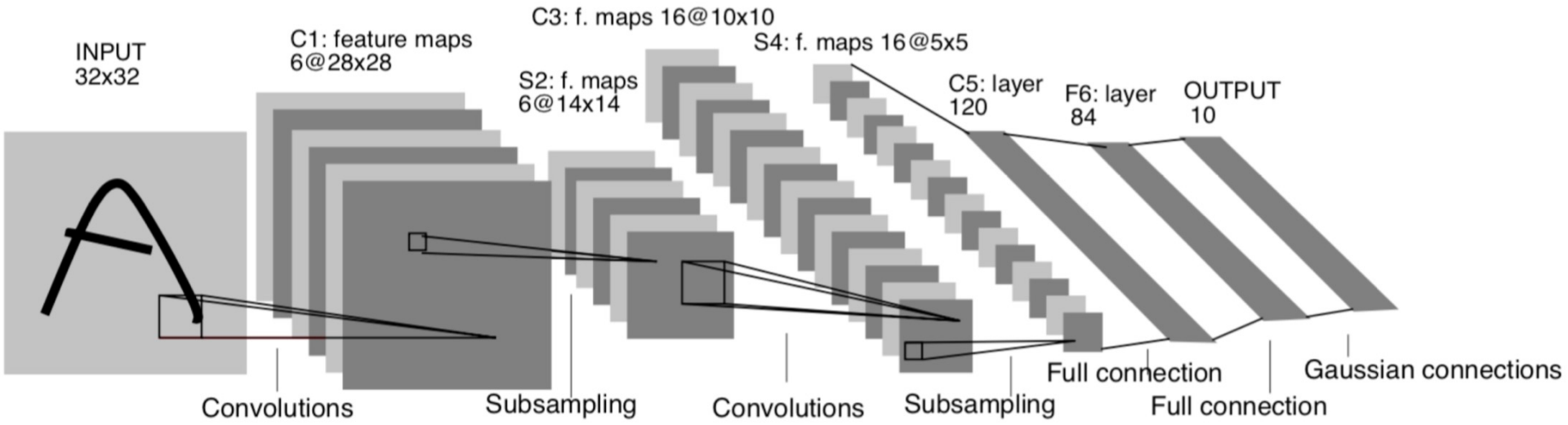
What is the step size and receptive field of subsampling?

Step size: 2, Receptive field: 2 x 2

How does subsampling affect the image representation?

Low pass filter. 'Blurs' the image.

Multi-channel convolution



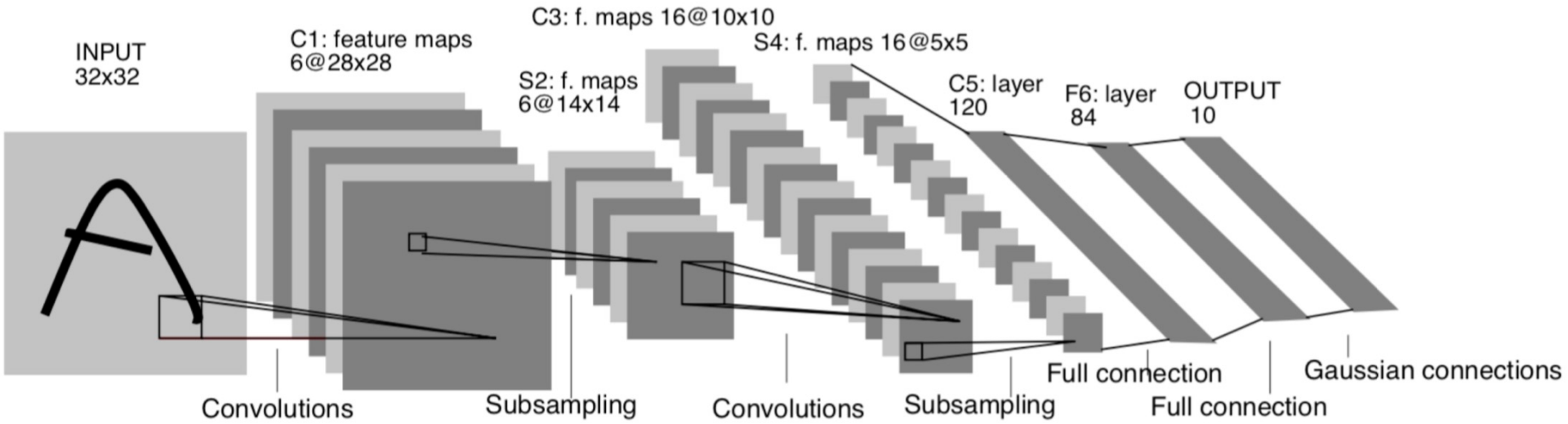
Convolution is now multichannel

$$h[i,j] = \sum_k \sum_u \sum_v g[u,v,k]\ f[i-u, j-v, k]$$

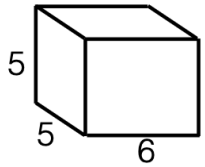
Is this a 3D convolution?

No. Why?

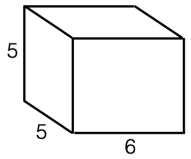
Convolutional filter

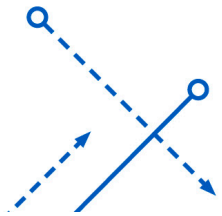


- What is the size of the convolutional filter at C3?
  - 5×5×6

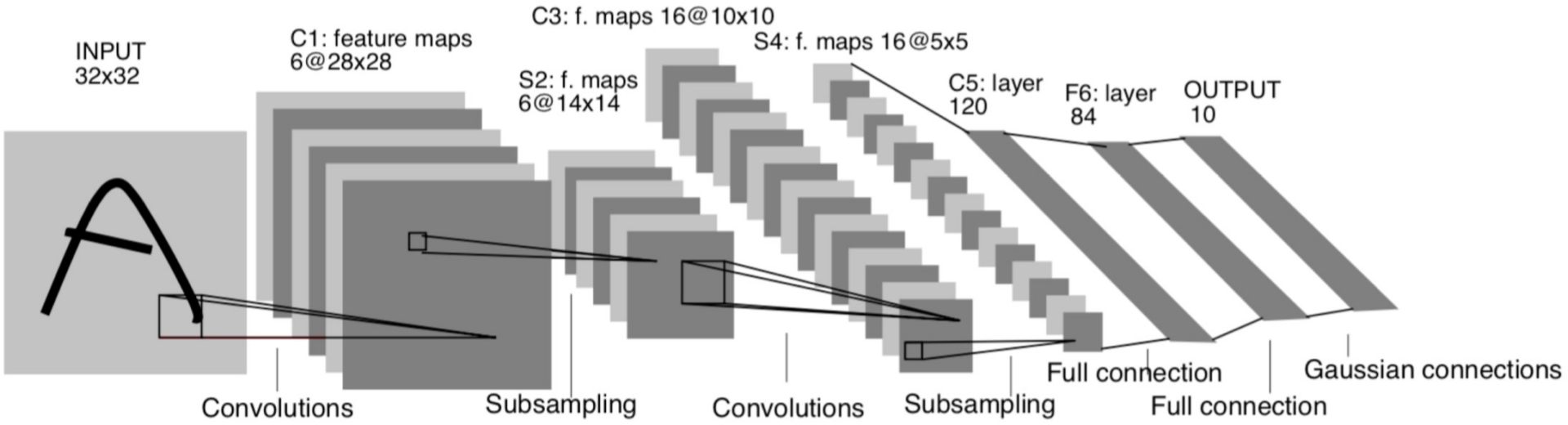


- How many features maps for one multi-channel convolution?

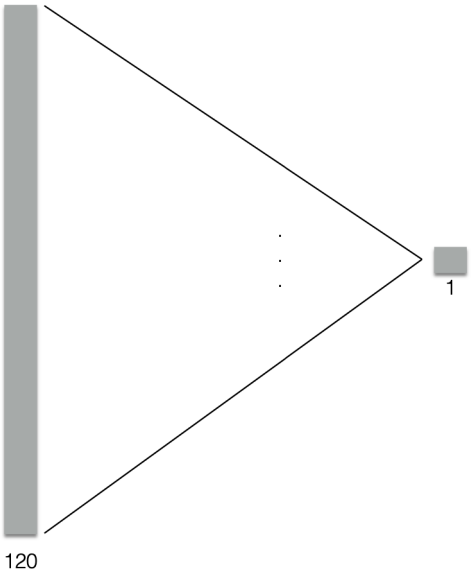




Fully connection



One huge perceptron!
(don't forget the bias)



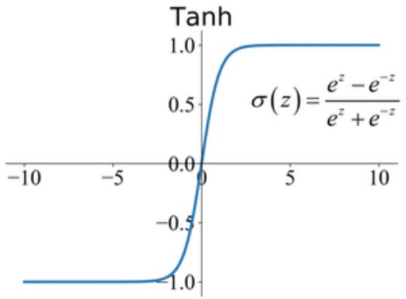
Full connection.

(actually just another special convolution)

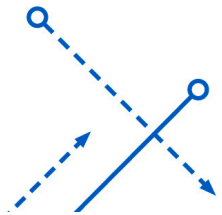
amplitude scale

$$f(a) = A \cdot \tanh(Sa)$$

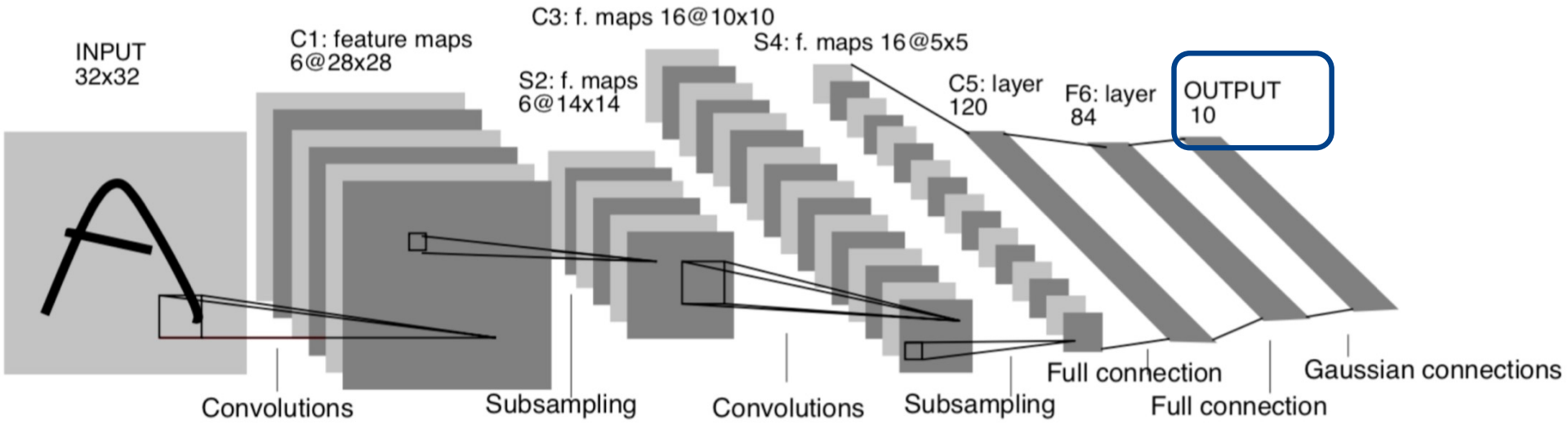
weighted sum



How many perceptrons to generate F6?



Output layer



Euclidean Radial Basis Function

Unnormalized negative log-likelihood of a Gaussian

$$y_i = \sum_{j}^{84} (x_j - w_{ij})^2$$

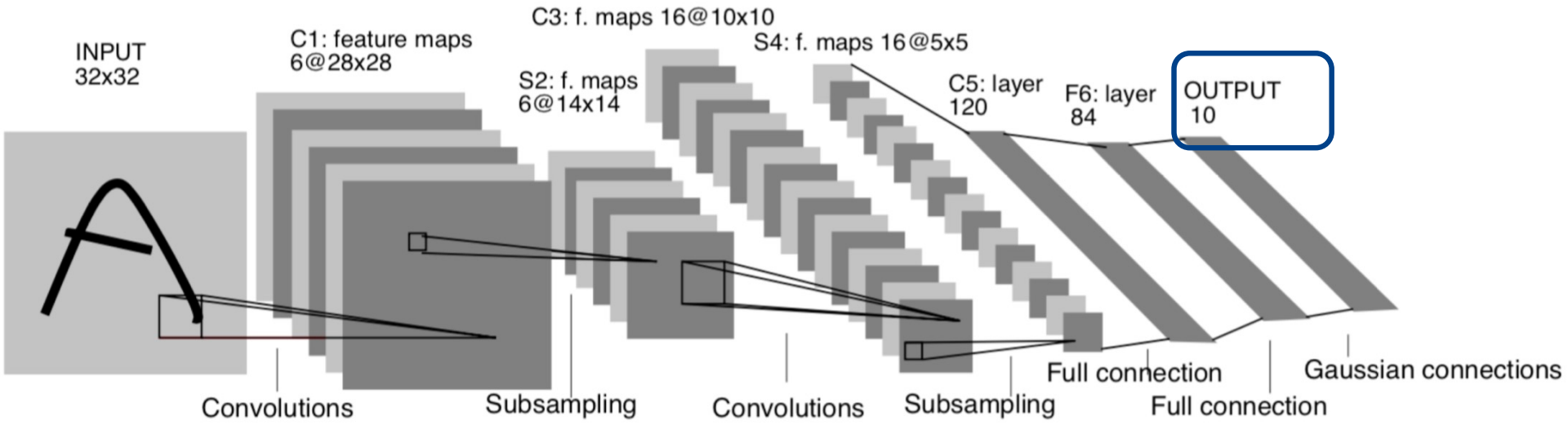
- y_i: single class output
- j: FC6
- w_{ij}: parameter set manually (-1, +1)

What is the range of i?

What happens when x matches w?

How do you set w?

Output layer



Euclidean Radial Basis Function

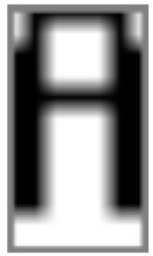
Unnormalized negative log-likelihood of a Gaussian

$$y_i = \sum_{j}^{84} (x_j - w_{ij})^2$$

single class output

FC6

$$w = \begin{matrix} 7 \\ 12 \end{matrix}$$



White: -1
Black: +1

w is set manually to match ASCII character

LeNet 5 is trained to draw ASCII characters!

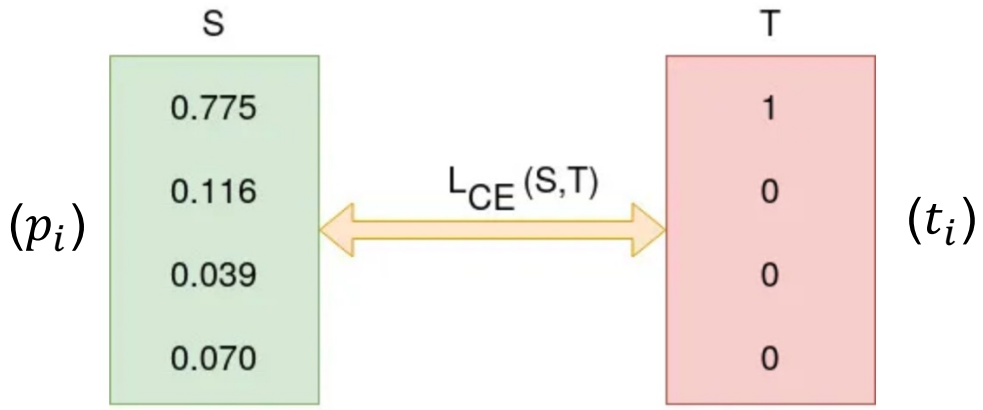
More frequently used loss function

- Cross-entropy loss

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

- Target as one-hot vector



Logits(S) and one-hot encoded truth label(T) with Categorical Cross-Entropy loss function used to measure the 'distance' between the predicted probabilities and the truth labels. (Source: Author)

Why do we call it cross-entropy?

- We need to define Information.
  - High Information (surprising): low probability
  - Low Information (unsurprising): high probability
- We need map probability $(0, 1)$ to information $(\infty, 0)$
- So, we define information as

$$I(x) = -\log p(x)$$

- Recall the expectation of an event x.

$$\int xp(x)\,dx \quad \text{or} \quad \sum x_i p(x_i)$$

- The entropy (expected information) is

$$-\int p(x)\log p(x)\,dx \quad \text{or} \quad -\sum_i p(x_i)\log p(x_i)$$

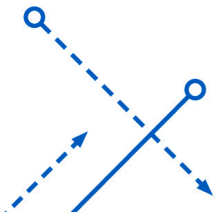
Cross-entropy

- We change the $p(x_i)$ or p_i in **entropy** to distribution t_i, this gives us cross-entropy.

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

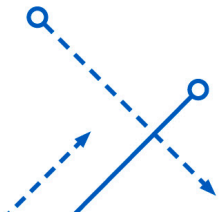
where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

- Expected information of p_i with respect to distribution t_i.
- A measure of the **difference between two probability distributions**, typically between a predicted probability distribution and a true probability distribution.
- Used by all following networks, e.g., AlexNet, VGG, ResNet, DenseNet, etc.
- Easy to compute gradient: more explanation.



Important Concepts

- Small filters capture common patterns
- Small filters can be reused and are efficient
- Multi-channel 2D Convolution
- Average pooling
- Convolution+Pooling+Non-linearity Module



AlexNet (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

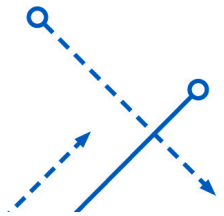
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

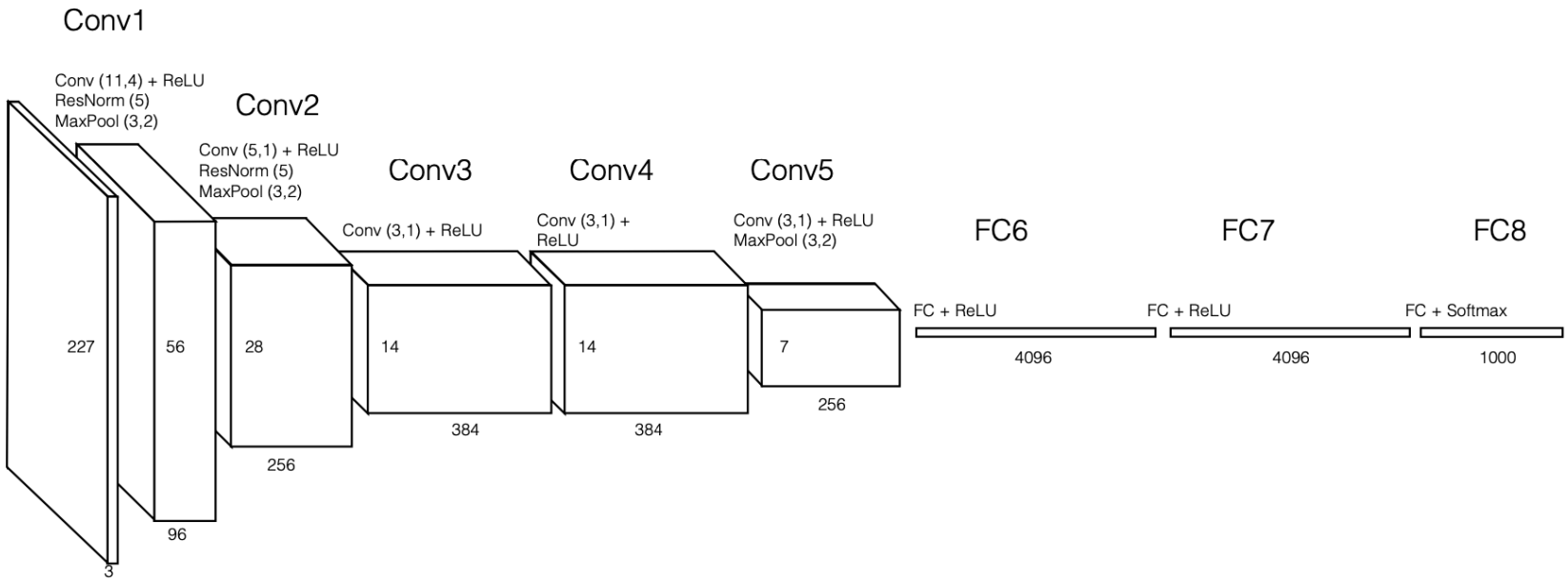
We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. 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.

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

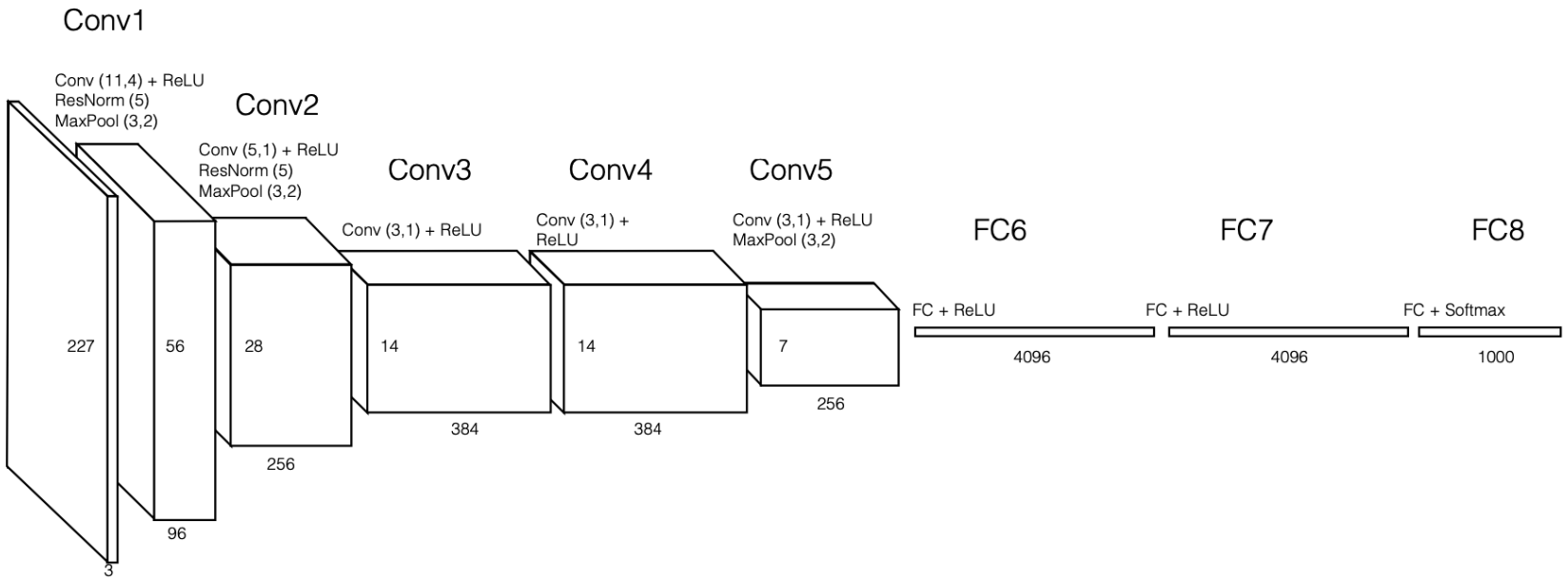


AlexNet structure



- Conv1 (a **convolution**, ReLU, ResNorm, max pooling)
  - Convolution with size? 11 x 11 x 3
  - How many filters in Conv1? 96
  - What is the size of the response map after Conv1? 55
  - What is the stride of the convolutions? 4
- Shorthand notation: Conv(size:11, stride:4)

AlexNet structure



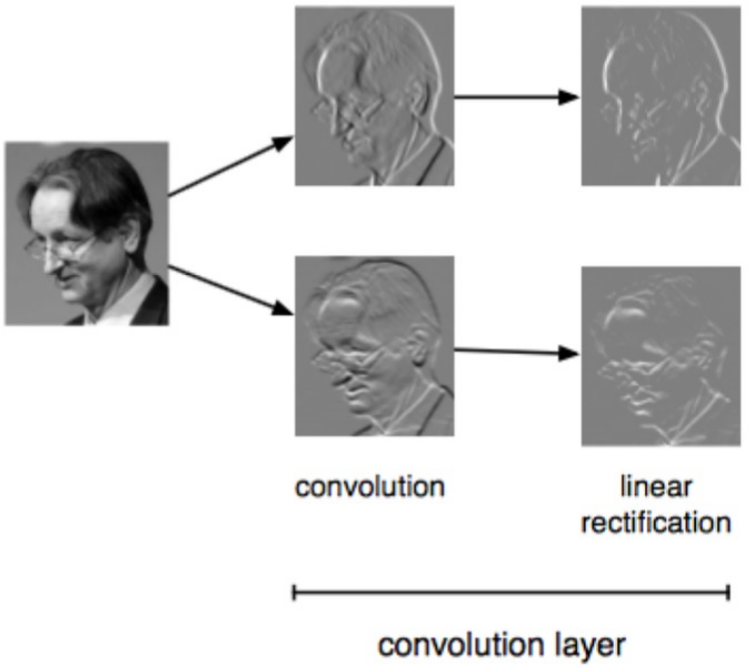
- Conv1 (a convolution, **ReLU**, ResNorm, max pooling)
- Regularized Linear Unit (ReLU)

$$f(x) = \max(0, x)$$ Pixel-wise operator

- Keeps all of the positive activations and throws away all negative!
- Why would we want a non-linear activation function like this?

Why use ReLU

- Two linear functions (convolutions) is the same as one linear function (convolution).
- ReLU adds a non-linearity and is more powerful if you want to stack convolutions.
- Makes gradients sparse. Helps avoid vanishing gradients.
- Really speeds up convergence of gradient descent



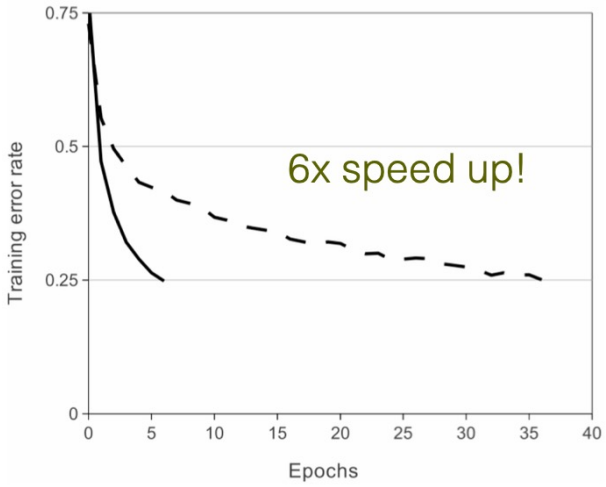
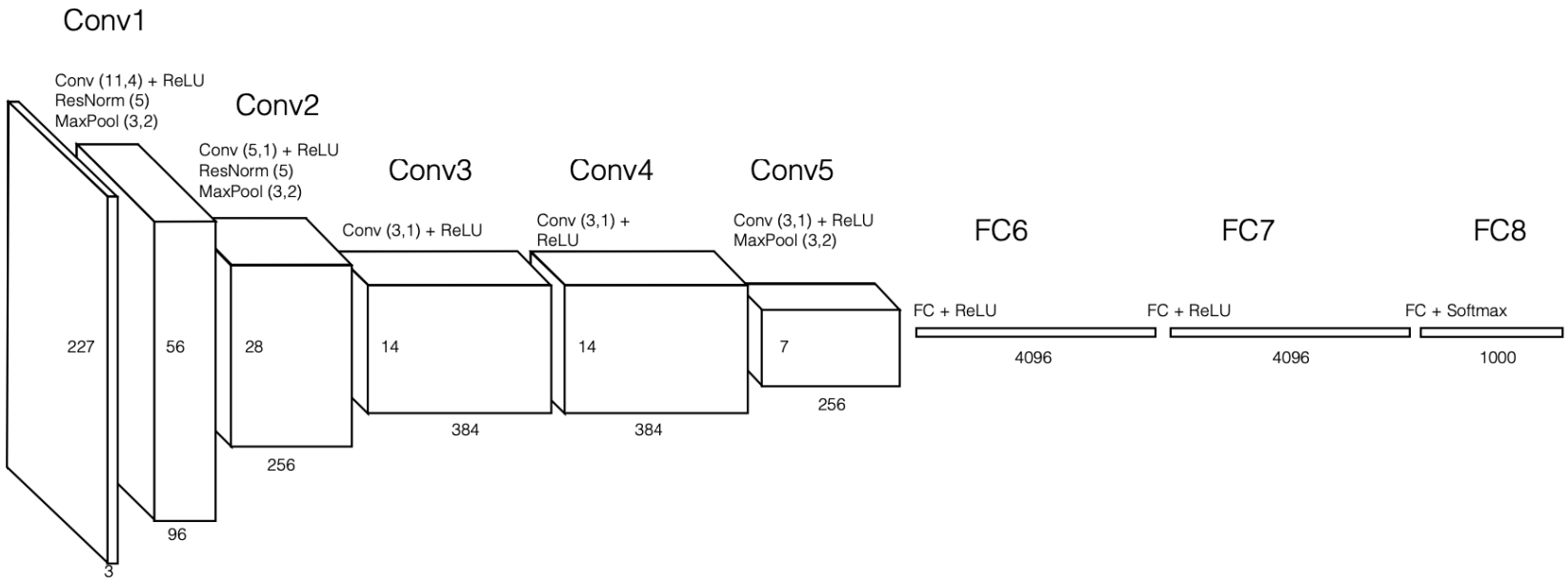


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each net-

AlexNet structure



- Conv1 (a convolution, ReLU, **ResNorm**, max pooling)

Local response normalization

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=-2}^{2} (a^j_{x,y})^2\right)^{\beta}}$$

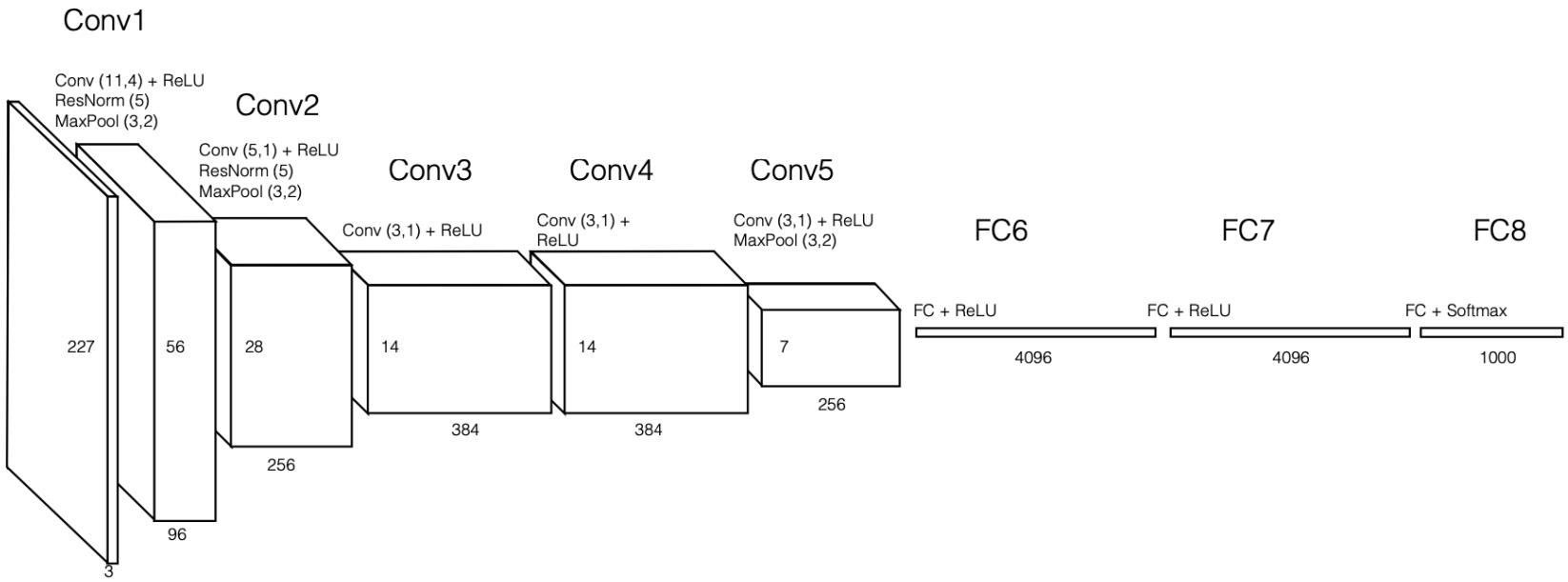
channel

Activation location

offset (2) Scale (10^{-4}) 5 channel window Non-linear scaling (0.75)

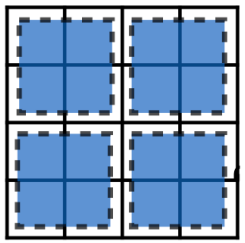
1.2~1.4% reduction in error rate

AlexNet structure



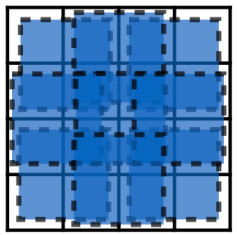
- Conv1 (a convolution, ReLU, normalization, **max pooling**)
- Overlap maximum pooling.

Traditional Pooling



Size: 2 x 2
Stride: 2
MaxPool (2,2)'

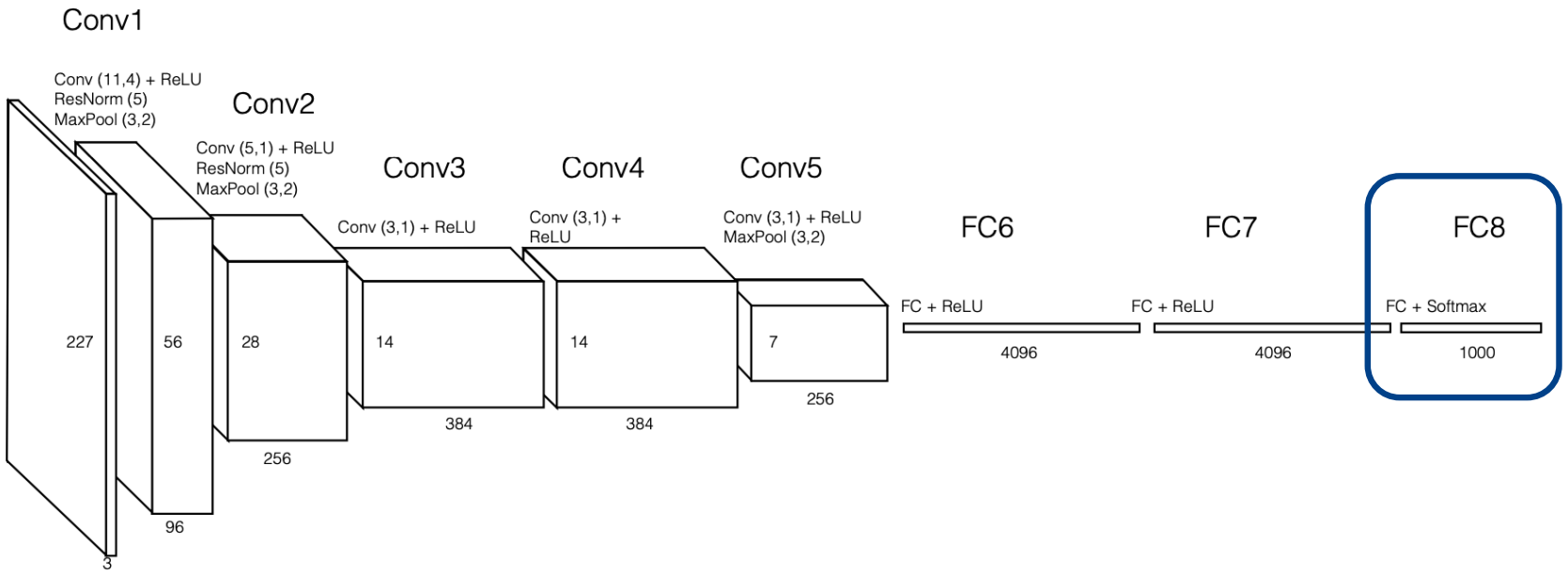
Overlapped Pooling



Size: 2 x 2
Stride: 1
'MaxPool (2,1)'

0.3~0.5% reduction in error rate

AlexNet structure



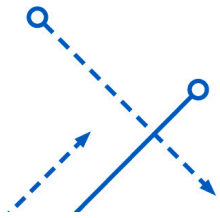
- Softmax

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

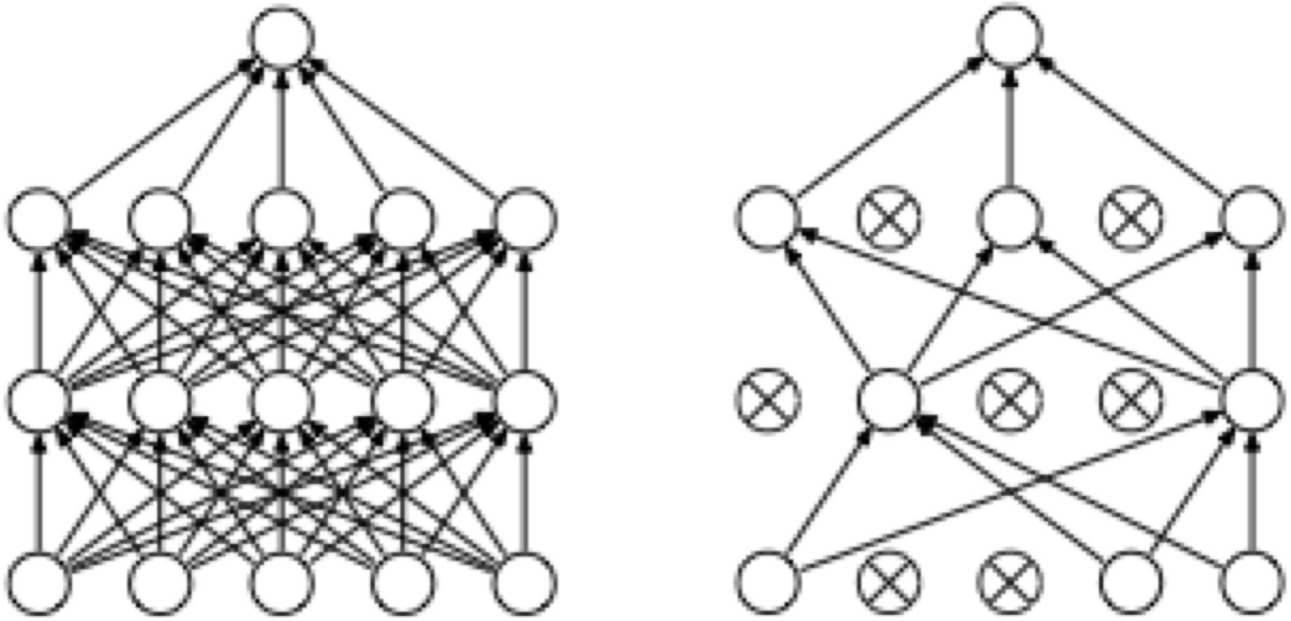
- Final softmax converts FC output values to probability

Tricks for Training AlexNet

- Dropout
- Data augmentation
- SGD with momentum



Dropout

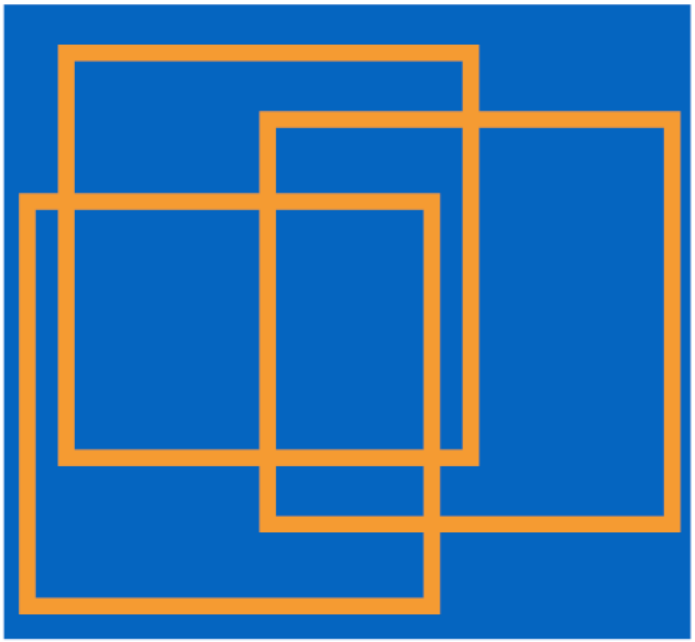


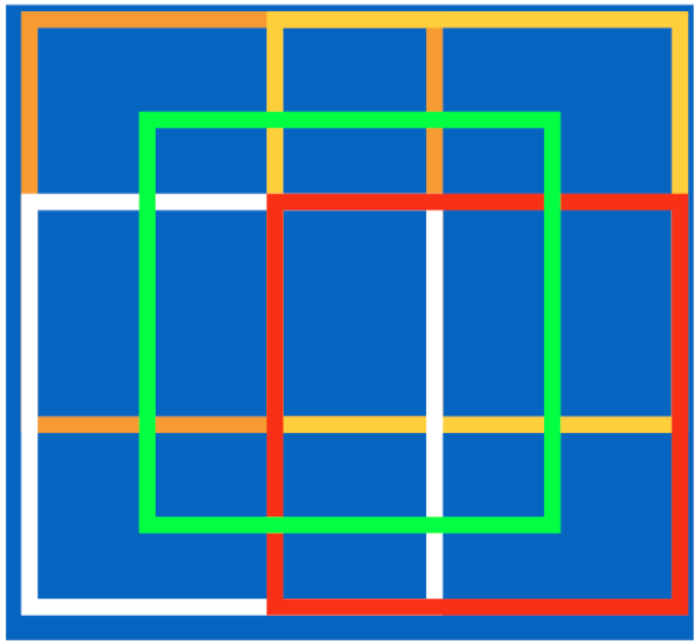
(a) Standard Neural Net (b) After applying dropout.

- During training certain nodes are randomly dropped out in FC.
- Forces the network to learn with less parameters.
- Reduces overfitting (doubles training time).
- Has the effect of model averaging.
- Use all nodes at test time .

Data Augmentation

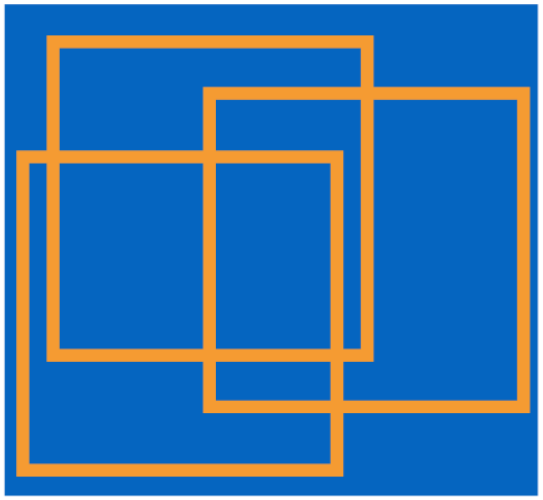
- Random translation and horizontal flipping.
- A test time, prediction is averaged over 10 patches.
  - corners, centers, and flips.

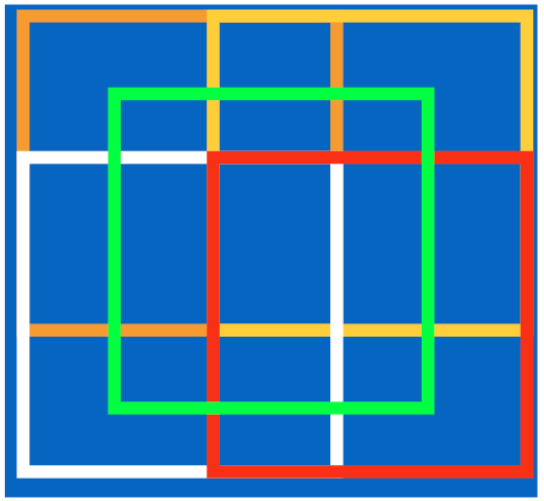




Data Augmentation

- Random translation and horizontal flipping.
- A test time, prediction is averaged over 10 patches.
  - corners, centers, and flips.





- Color augmentation
  - Changes the color and intensity of the image
    - Reduces top-1 error rate by 1%
- Prevents overfitting

SGD with momentum

- Gradient Update

update equation

$$w_{i+1} := w_i + v_{i+1}$$

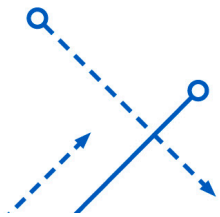
momentum: 'keep going'; weight decay; learning rate; expected gradient of the loss; mini-batch

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

- Momentum helps accelerate gradients vectors in the right directions, thus leading to faster converging
- Weight decay helps improve the convergence of an optimization algorithm by encouraging smaller weights.

Important Concepts

- Overlapped Pooling
- Response Normalization
- ReLU
- Softmax
- Training trick: Dropout
- Training trick: Data Augmentation
- Training trick: momentum and decay
- 2 GPUs, 6 days to train



VGG Net (2015)

Published as a conference paper at ICLR 2015

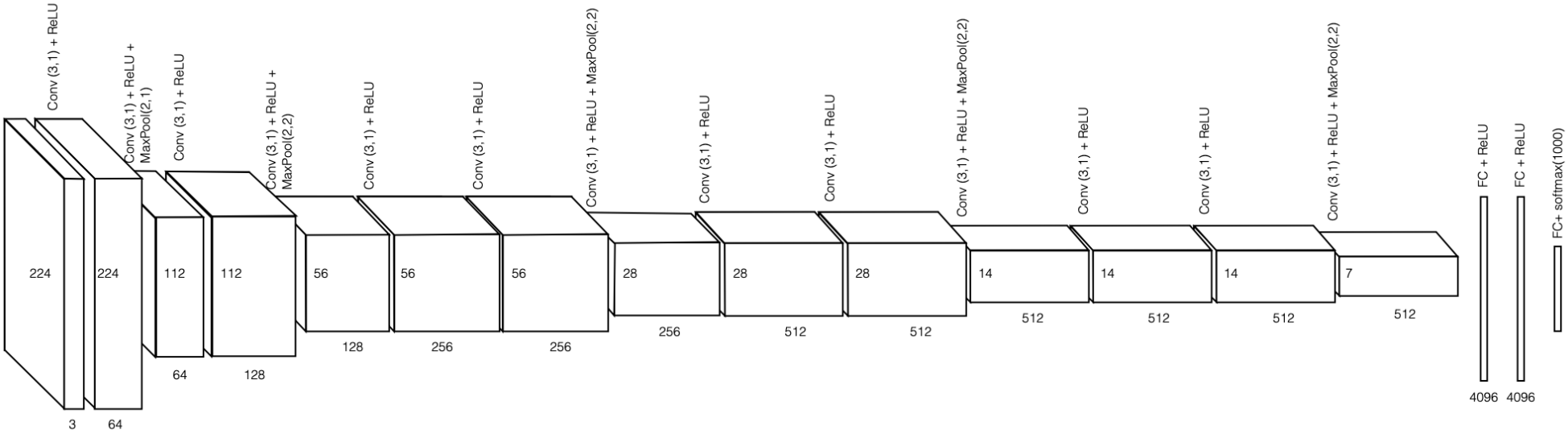
VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

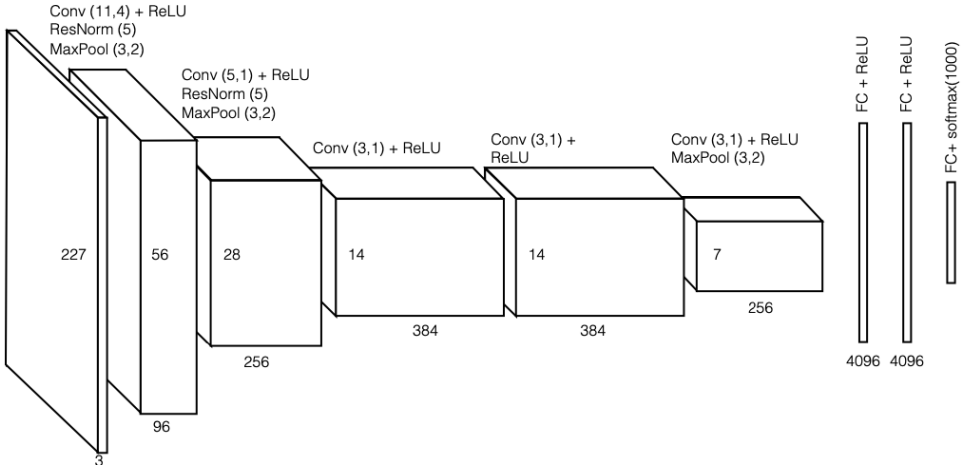
In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

VGG-16


Conv (3,1) + ReLU
Conv (3,1) + ReLU + MaxPool(2,1)
Conv (3,1) + ReLU
Conv (3,1) + ReLU + MaxPool(2,2)
Conv (3,1) + ReLU
Conv (3,1) + ReLU
Conv (3,1) + ReLU + MaxPool(2,2)
Conv (3,1) + ReLU
Conv (3,1) + ReLU
Conv (3,1) + ReLU + MaxPool(2,2)
Conv (3,1) + ReLU
Conv (3,1) + ReLU
Conv (3,1) + ReLU + MaxPool(2,2)
FC + ReLU
FC + ReLU
FC+ softmax(1000)
224
224
112
112
56
56
56
28
28
28
14
14
14
7
3
64
64
128
128
256
256
256
512
512
512
512
512
512
4096
4096

Improvements?

AlexNet

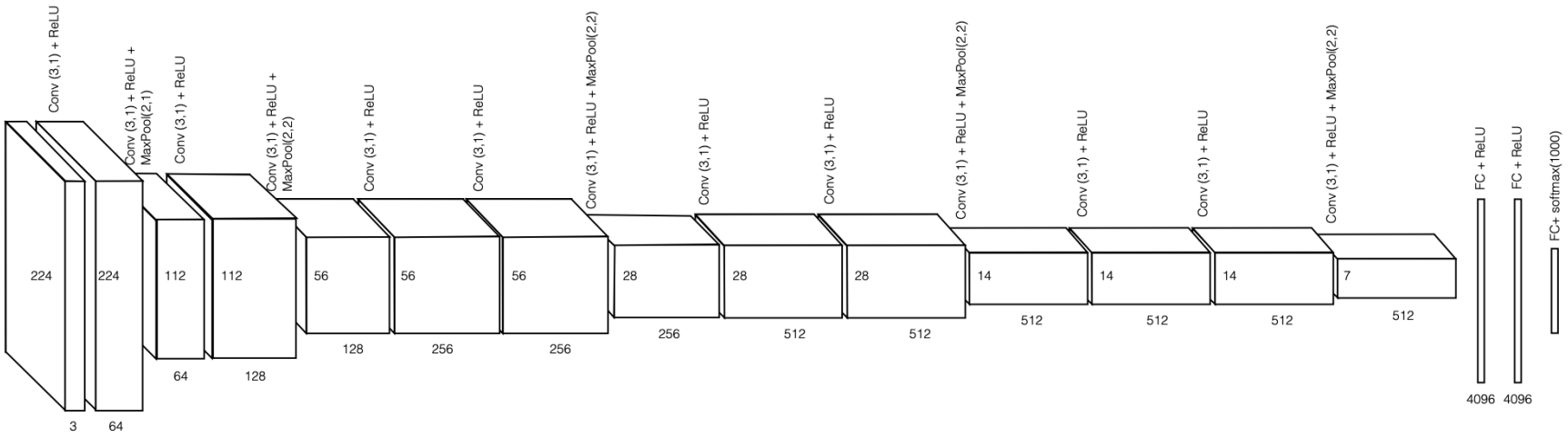


8 layers

deeper!

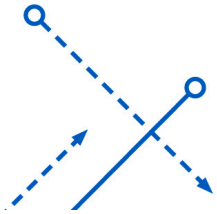
VGG-16

16 layers



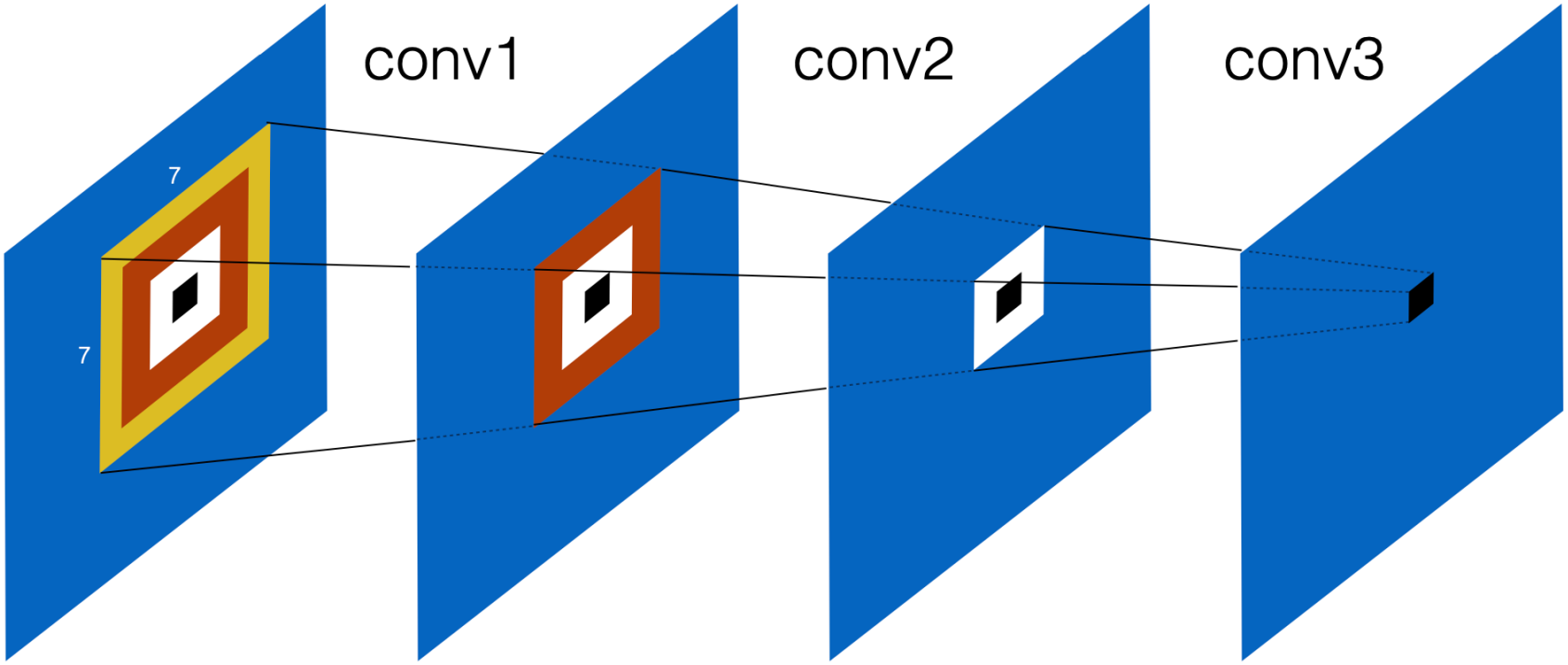
VGG-16

- Emerging ‘rule of thumb’
- Convolutions all standardized to 3×3
- Convolutions always followed by ReLU
- Stack several convolutions at the same resolution
- Downsample by 2, increase channels by 2
- No local response normalization



Stacking convolutions

- What is the receptive field of 3 stacks of 3x3 convolutions?



Stacking convolutions

- What do you gain by stacking convolutions?
  - Decision function is more discriminative.

Two linear functions

$$b \cdot (a \cdot x)$$

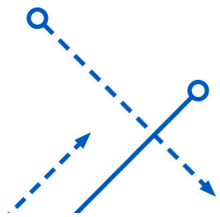
Same as just one linear function

Two non-linear functions

$$F(b \cdot F(a \cdot x))$$

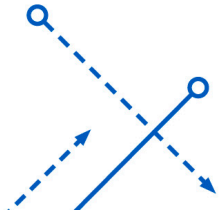
Can capture more complex decision boundary

  - Decrease number of parameters.
    - Three $3\times3\times C$ convolution layers: $27C$ parameters.
    - One $7\times7\times C$ convolution layer: $49C$ parameters.



Tricks for Training VGG

- Pre-training
- Data augmentation (crops, flips, color, scale)
- Dropout
- SGD with momentum
- 4 GPUs, 21 days, to train



ResNet (2016)

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

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 [40] 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 local-*

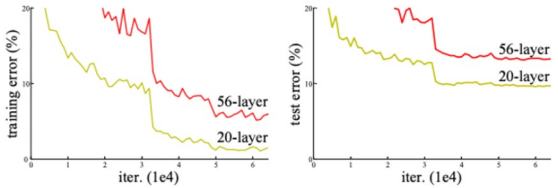


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

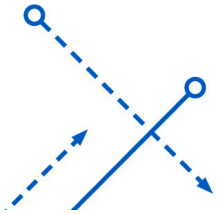
greatly benefited from very deep models.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [14, 1, 8], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 8, 36, 12] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation [22].

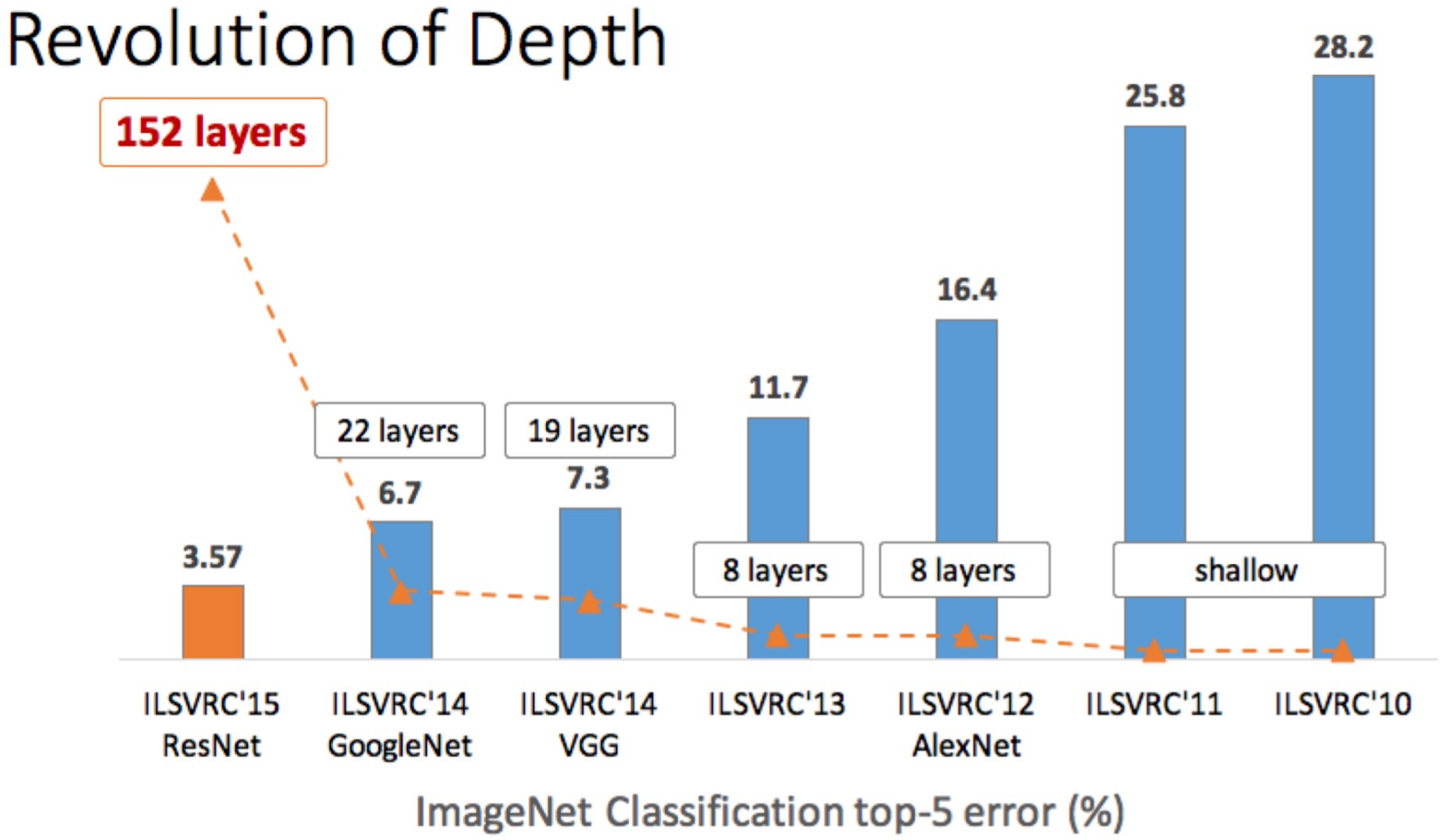
When deeper networks are able to start converging, a

ResNet

- 1st places in all five main tracks
- ImageNet Classification: "Ultra-deep" 152-laver nets
- ImageNet Detection: 16% better than 2nd
- ImageNet Localization: 27% better than 2nd
- COCO Detection: 11% better than 2nd
- COCO Segmentation: 12% better than 2nd

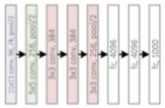


ResNet-152

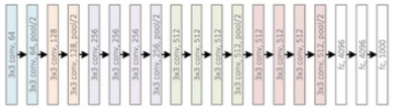

Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)

Going deeper and deeper

AlexNet



VGG-19



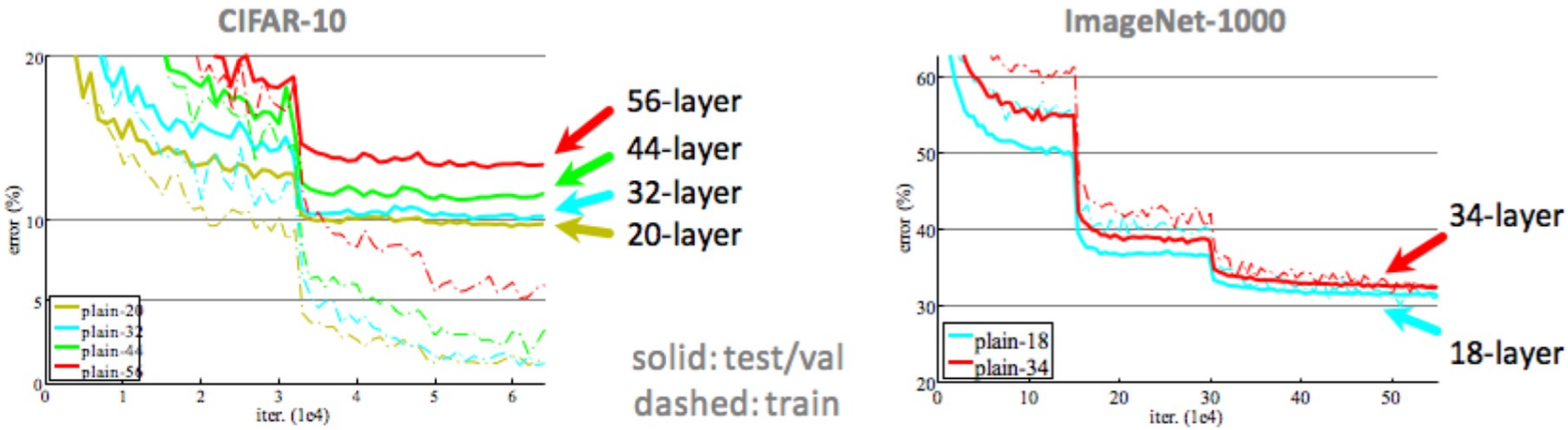
GoogLeNet

ResNet-152

Going deeper and deeper

- Deeper is better so just stack more convolutions.
- Right?

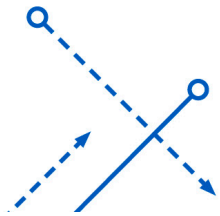
NO!



- Deeper networks have higher training error!

Going deeper and deeper

- Expectation
  - If adding extra identity layers on top of shallow one.
  - Training error should be the same not worse.
  - Deeper models shouldn’t have higher training error.
- Reality
  - Solvers (gradient descent) cannot find the solution.
- Solution
  - Start with the identity solution and then tune parameters.
  - This is residual learning!

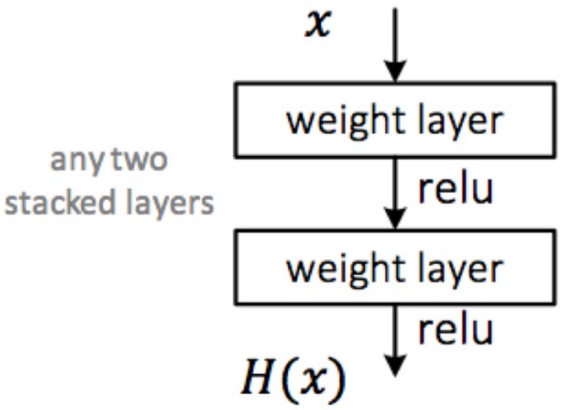


Deep Residual Learning

$H(x)$ is any desired mapping,

hope the 2 weight layers fit $H(x)$

- Plaint net



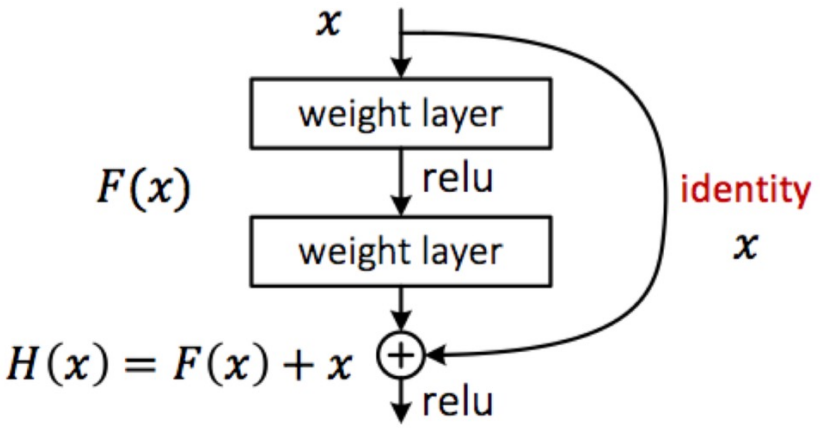
$H(x)$ is any desired mapping,

~~hope the 2 weight layers fit $H(x)$~~

hope the 2 weight layers fit $F(x)$

let $H(x) = F(x) + x$

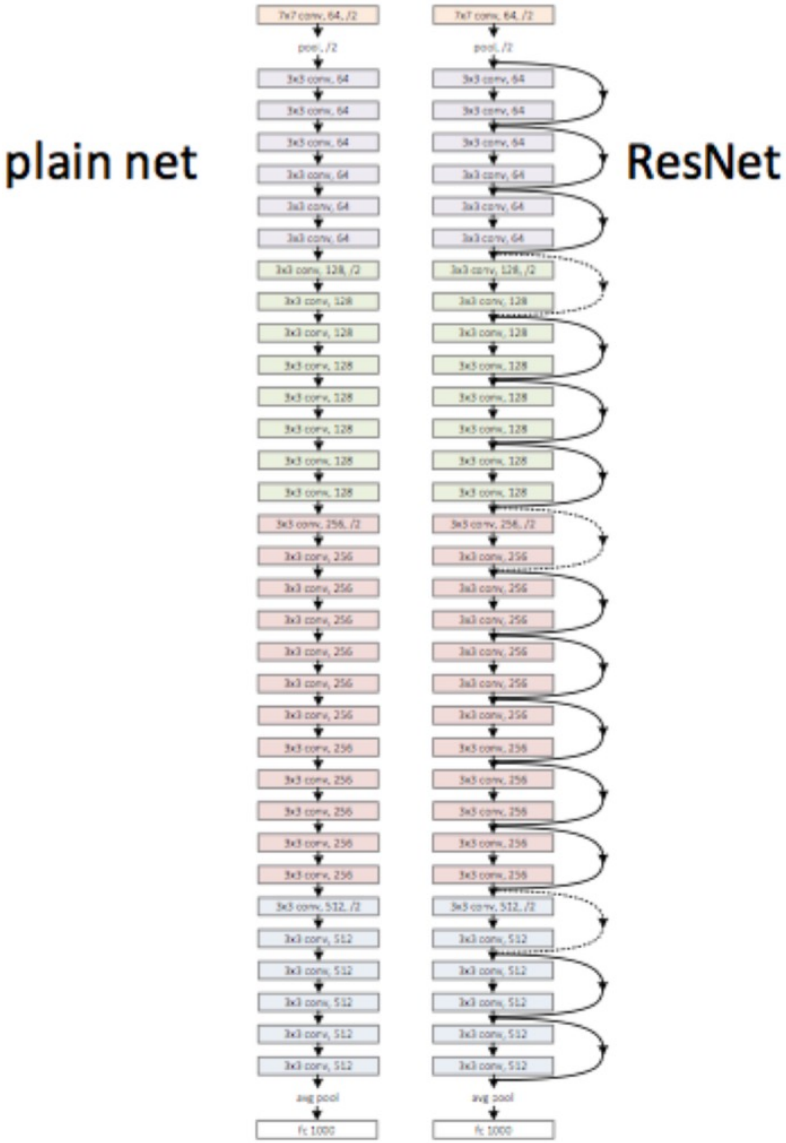
- Residual net



- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

Network design

- Simple design; just deep!



Batch Normalization

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe SIOFFE@GOOGLE.COM
Christian Szegedy SZEGEDY@GOOGLE.COM
Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout. Applied to a state-

minimize the loss

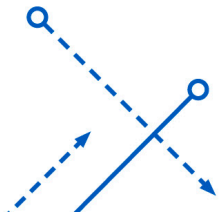
$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathrm{x}_i, \Theta)$$

where $\mathrm{x}_{1\ldots N}$ is the training data set. With SGD, the training proceeds in steps, at each step considering a *mini-batch* $\mathrm{x}_{1\ldots m}$ of size m. Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch $\frac{1}{m}\sum_{i=1}^{m} \frac{\partial \ell(\mathrm{x}_i,\Theta)}{\partial \Theta}$ is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a mini-batch can be more efficient than m computations for individual examples on modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate and the initial parameter values. The train-

Batch Normalization

- Normalizing input
  - LeCun et al 1998 "Efficient Backprop"
- BN: normalizing each layer, for each mini-batch
  - Greatly accelerate training
  - Less sensitive to initialization
  - Improve regularization



Recall SGD

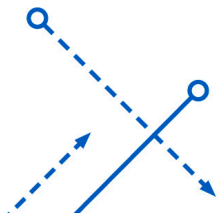
For each random sample $\{x_i, y_i\}$

1. Predict
    1. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
    2. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$

Stochastic sampling. Distribution can change dramatically!

2. Update
    1. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$
    2. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

Solution:
Be conservative. Make step size (learning rate) small

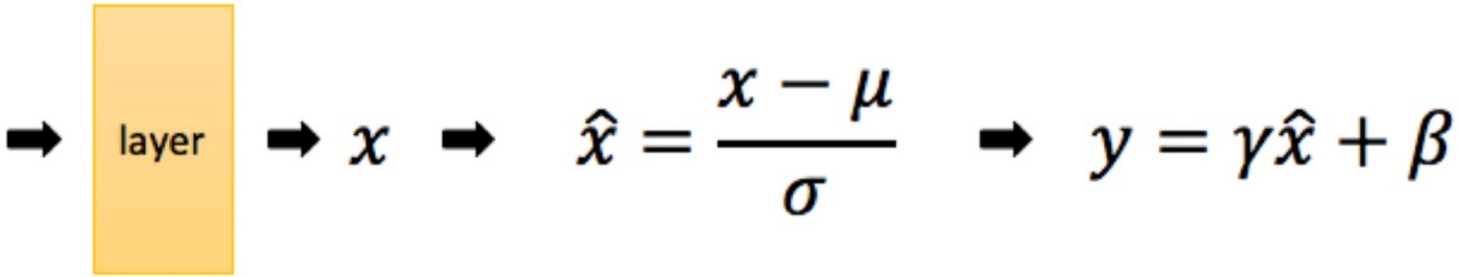


What happens when you make the step size really small?

Batch Normalization

- A really small step makes the training time really long
- Instead, for each training mini-batch...

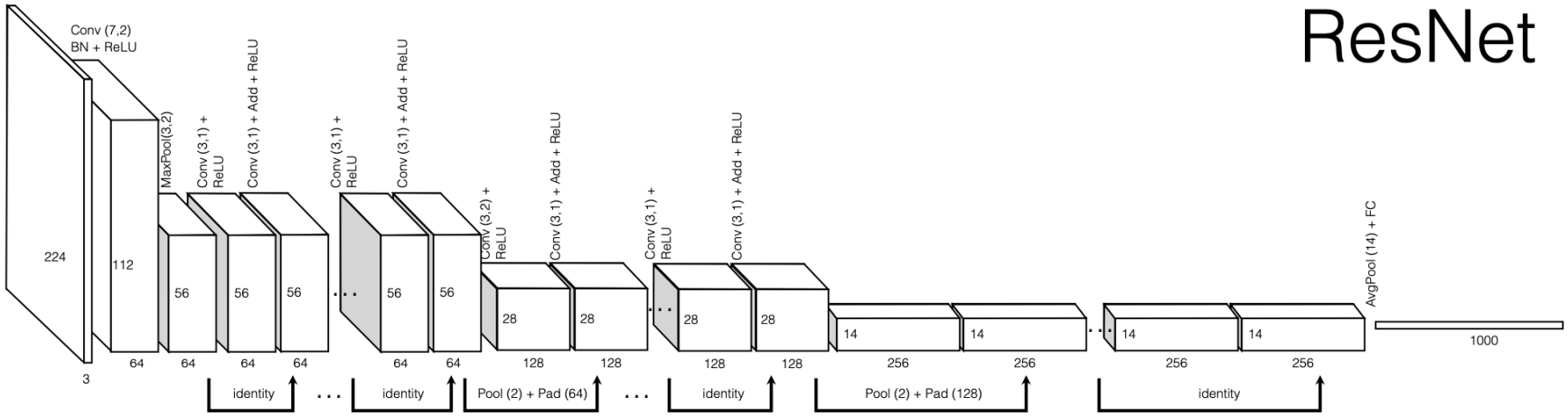
Batch Normalization (BN)



- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

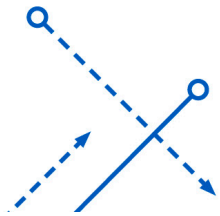
ResNet



- Convolution + Batch Normalization + ReLU
- 7 x 7 convolution just for the first layer
- Stride of 2 reduces the resolution in half
- Batch Normalization to scale the responses
  - ReLU non-linearity

Important Concepts

- Residual Learning
- Skip connections



DenseNet (2018)

Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at https://github.com/liuzhuang13/DenseNet.

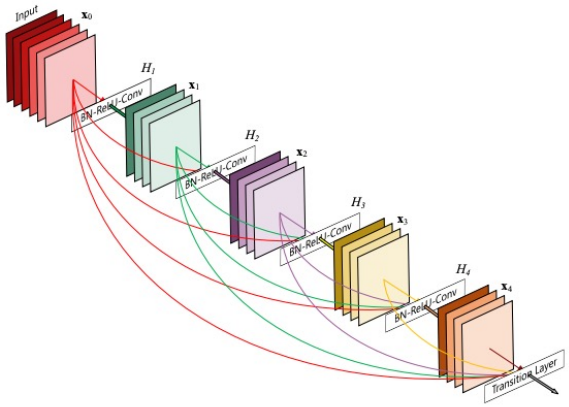
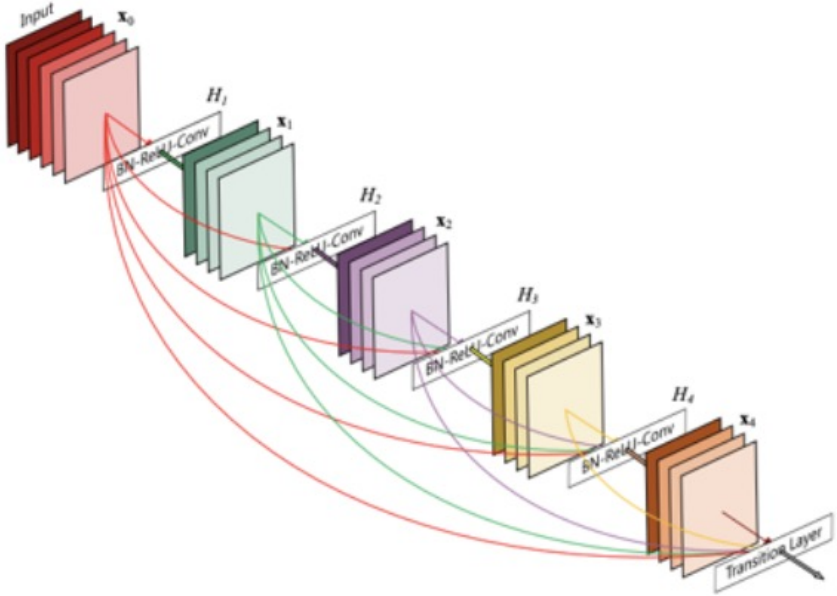


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

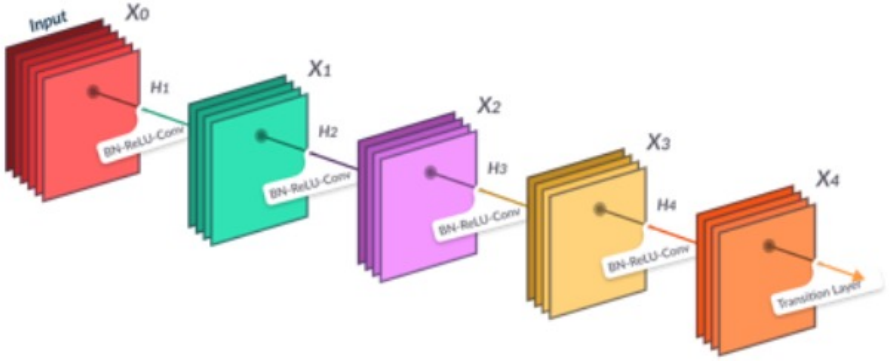
Networks [34] and Residual Networks (ResNets) [11] have surpassed the 100-layer barrier.

As CNNs become increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the network. Many recent publications address this or related

DenseNet



DenseNet Structure



ResNet Structure

$$a^{[l]} = g\left([a^{[0]}, a^{[1]}, a^{[2]}, \dots\dots\dots\dots, a^{[l-1]}\right)$$

$$a^{[l]} = g\left(z^{[l+1]} + a^{[l]}\right)$$

DenseNet

- For each layer, the feature-maps of all preceding layers are used as inputs.
- These connections mean that the network has L(L+1)/2 direct connections
  - L is the number of layers in the architecture.
- Alleviate the vanishing-gradient problem
- Encourage feature reuse
- Reduce the number of parameters