Lecture 4

Von-Wun Soo

Prof. Von-Wun Soo

Outline

- Sparse coding and self-taught learning
- Autoencoder
- Convolutional Neural Networks

Why sparse coding?

- A code pattern is said to be sparse when a stimulus (like an image) provokes the activation of just a relatively small number of neurons, that combined represent it in a sparse way.
- Neural science evidence: (Hubel & Wiesel, 1968) in the V1 region there are specific cells which respond maximally to edge-like stimulus (besides having others "useful" properties). Sparse Coding explains well many of the observed characteristics of this system. (Olshausen & Field, 1996)
- ML study: a useful technique for feature extraction that yields good results in transfer learning tasks.
  - Raina et al. (2007) showed that a set of "basis vectors" (features, as pen-strokes and edges) learned with a training set composed of hand-written characters improves classification in a hand-written digits recognition
  - “Google Brain” use sparse coding to construct a network with several layers was capable of distinguish a image of a cat in a purely unsupervised manner (Le et al., 2013).
- In signal analysis: signals must be represented with as few components as possible, (compressed) and can be recovered or inferred from the few components as well.

Sparse coding

- Represent a variable vector in terms of a linear combination of a set of vector bases:

$$X = \Sigma^{k}_{i=1} a_i \phi_i$$

- Unlike some other unsupervised learning techniques such as PCA, sparse coding can be applied to learning overcomplete basis sets, in which the number of bases is greater than the input dimension.
- Over-complete representation of basis in the sense that coefficients a_i are not uniquely determined by the input x
- We want the coefficients a_i being sparse as having few non-zero components or having few components not close to zero.

Sparse penalty

- Minimize $_{a(j)i,\varphi i} \sum^{m}_{j=1} ||\mathbf{x}(j) - \sum^{k}_{i=1} a(j)_i \varphi_i||^2 + \lambda \sum^{k}_{i=1} S(a(j)_i)$

  Subjected to $||\varphi_i||^2 \leq C, \forall i=1,...,k$

The constraints $||\varphi_i||^2 \leq C$ to ensure the purpose of making very few a(j)'s are far from zero's.

- $S(a(j)_i)$ is the penalty of sparseness, commonly choose
  - L1 penalty: $S(a_i)=|a_i|_1$
  - Log penalty: $S(a_i)=\log(1+a_i^2)$.

Pro and con of Sparse coding

- Pro:

  Better generalization error than parametric models when labels are few.

- Con:

  To "encode" a new data example, optimization must be performed to obtain the required coefficients.

  Sparse coding is computationally expensive to implement even at test time especially compared to typical feedforward architectures.

Self taught learning

- Supervised learning:

  data is labeled and of the same type (shares the same class labels).
- Semi-supervised learning: unlabeled examples are drawn from the same distribution with labeled data
- Transfer learning: All data is labeled but some is of another type (i.e. has class labels that do not apply to data set that we wish to classify).
- Active learning: Select limited size un-labeled samples to ask oracle(teacher) to “best” improve the learning performance.
- Self-taught learning: is a kind of semi-supervised learning; source domain unlabelled; target domain labelled

- http://www.wikicoursenote.com/wiki/Self-Taught_Learning

Learning from Labeled and unlabeled data

- Supervised classification (labeled both training and test set)

- Transfer learning (labeled both source and target; different domains)

- Semi-supervised classification
- （both labeled and unlabled in training set)

- Self-taught learning (source unlabled, target lableled)

Learning sparse coding from unlabeled data

- Given the unlabeled data $\{x^{(1)}_u, \ldots, x^{(k)}_u\}$ with each $x_u^{(i)} \in R$,
- We pose the following optimization problem:
- $\text{minimize}_{b,a} \quad ||x_u^{(i)} - a_j^{(i)} b_j^{(i)}||^2_2 + \beta ||a^{(i)}||_1 \qquad (1)$

  $\text{s.t.} \quad ||b_j||_2 \leq 1, \quad \forall j \in 1,\ldots,s$

- $B = \{b_1, \ldots, b_s\}$ are basis vectors

  $a_j^{(i)}$ activation coefficients

Learning features from labeled data and sparse coding

Let $x_l^{(i)}$ *are labelled data* i = 1,…,N

Find sparse coding $\hat{a}(x_l^{(i)})$ of $x_l^{(i)}$ in terms of basis vectors b_j

- $\hat{a}(x_l^{(i)}) = \text{argmin}_{a(i)} ||x_l^{(i)} - \sum_j a_j^{(i)} b_j||^2_2 + \beta||a^{(i)}||_1$

Self-taught learning algorithm Via Sparse coding

Input:
Labeled training set $T = \{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),...,(x^{(m)},y^{(m)})\}$.
Unlabeled data $\{x_u^{(1)}, x_u^{(2)}, ..., x_u^{(k)}\}$.
Output: Learned classifier for the classification task.
Using unlabeled data $\{x_u^{(i)}\}$, solve the optimization problem (1) to obtain bases b.

Compute features for the classification task to obtain a new labeled training set

$\hat{T} = \{(\hat{a}(x^{(i)}), y^{(i)})\}$ i=1,…, m,

where

$$\hat{a}(x_l^{(i)}) = \text{argmin}_{a(i)} \| x_l^{(i)} - \sum_j a_j^{(i)} b_j \|_2^2 + \beta \| a^{(i)} \|_1$$

Learn a classifier C by applying a supervised learning algorithm (e.g., SVM) to the labeled training set $\hat{T}$.

return the learned classifier

Autoencoder

Von-Wun Soo

Representation learning

- Finding the right representation can enhance performance of learning.
- Discover the feature by itself rather than design the feature set manually.

This is called learning representation and autoencoder.

Rationales

- Compression a large data set (Reduce dimensionality) #h << #x
- Uniform encoding various
- Finding decomposition of encoding and decoding matrix at the same time

- An autoencoder neural network with a linear hidden layer is similar to PCA.
- Upon convergence, the weight vectors of the *K* neurons in the hidden layer will form a basis for the space spanned by the first *K* principal components.
- However, unlike PCA, this technique will not necessarily produce orthogonal vectors.

- Undercomplete autoencoder:

  #h < #x (input dimension)

- Overcomplete autoencoder:

  #h >= #x input (input dimension)

- Regularized autoencoder:

  Impose a penalty to the regularization

- Sparse autoencoder:

  $L(x, g(f(x))) + \Omega(h)$

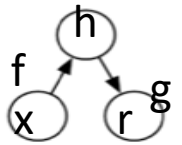
  $\Omega(h)$ is sparse penalty

Under-complete autoencoder

- An autoencoder whose code dimension is less than the input dimension is called undercomplete. (otherwise overcomplete)

- When the decoder is linear and L is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA.

Encoder-decoder



- Learning autoencoder can be viewed as minimization of loss function

  $L(x, g(f(x)))$

- f: encode
- g: decode
- Even #h > #x, we can impose sparseness constraint on h, $L(x, g(f(x))) + \Omega(h)$
- $\Omega(h)$: penalty as regularization term

Sparse autoencoder

$\log p_{model}(x) = \log p_{model}(h, x)$.

$\log p_{model}(h, x) = \log p_{model}(h) + \log p_{model}(x|h)$.

$p_{model}(h_i) = \lambda/2\, e^{-\lambda|h_i|}$

let $\Omega(h) = \lambda \Sigma_i |h_i|$ then

$-\log p_{model}(h)$

$= \Sigma_i (\lambda|h_i| - \log \lambda/2)$

$= \Omega(h) + \text{const.}$

Regularizing by Penalizing Derivatives

$$L(x, g(f(x))) + \Omega(h, x),$$

$$\Omega(h,x) = \lambda \Sigma_i || \nabla_x h_i ||^2.$$

- This forces the model to learn a function that does not change much when x changes slightly.
- Because this penalty is applied only at training examples, it forces the autoencoder to learn features that capture information about the training data distribution.

Deep autoencoder

• Universal approximator theorem guarantees that a feedforward neural network with at least one hidden layer can represent an approximation of any function (within a broad class) to an arbitrary degree of accuracy, provided that it has enough hidden units.

- Many hidden layers are better even though one hidden layer is theoretically sufficient.
- Depth can exponentially reduce the computational cost of representing some functions.
- Depth can also exponentially decrease the amount of training data needed to learn some functions.

Denoising autoencoder DAE

- L (x , g (f (x ‘))) ,

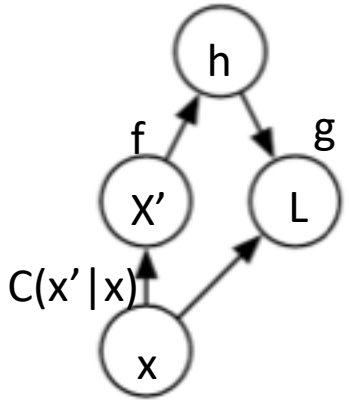
  X’ is a copy of x that has been corrupted by some form of noise.

- Penalty of sparse autoencoder

  L(x, g(f (x))) + Ω(h, x),

  where $\Omega(h, x) = \lambda \sum_{i} \; ||\nabla_{x} h||^{2}$.

Denoising Autoencoder (DAE)



- We may train the autoencoder by minimizing

$$-\log p_{decoder}(x \mid h = f(x'))$$

- X' is obtained by corrupted process: $C(x' \mid x)$

Denoising Autoencder

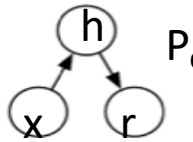
- The autoencoder then learns a reconstruction distribution preconstruct (x | x’) estimated from training pairs (x, x’),
- as follows:
- 1. Sample a training example x from the training data.
- 2. Sample a corrupted version x ‘ from C(x ‘ | x = x).
- 3. Use (x, x’) as a training example for estimating the autoencoder reconstruction distribution
- $p_{reconstruct}(x \mid x') = p_{decoder}(x \mid h)$ with
- h the output of encoder f(x ‘) and
- $p_{decoder}$ typically defined by a decoder g(h).

D A E

- Viewed as performing stochastic gradient descent on the following expectation：
- $-E_{x \sim \hat{p}data(x)} E_{x' \sim C(x' \mid x)} log\ p_{decoder}(x \mid h = f(x'))$
- where $\hat{p}_{data}(x)$ is the training distribution.

Stochastic Encoders and Decoders

- $p_{encoder}(h \mid x) = p_{model}(h \mid x)$
- $p_{decoder}(x \mid h) = p_{model}(x \mid h)$.

$P_{encoder}(h|x)$  $P_{decoder}(x|h)$

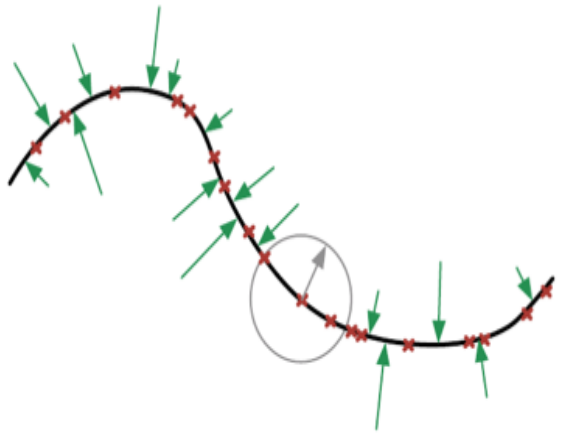
Learning Manifolds with autoencoder

- Only the variations tangent to the manifold around x need to correspond to changes in h = f(x).
- Hence the encoder learns a mapping from the

  input space x to a representation space,
- a mapping that is only sensitive to changes along the manifold directions,

  but that is insensitive to changes orthogonal to the manifold.

Learning manifold



- A gray arrow demonstrates how one training example is transformed into one sample from this corruption process.
- When the denoising autoencoder is trained to minimize the average of squared errors

  $||g(f(\tilde{x})) - x||^2$, the reconstruction $g(f(\tilde{x}))$

  estimates $E[x|\tilde{x}]$.
- The vector x, $\tilde{x} \sim pdata(x)\, C(\tilde{x}|x)$
- $g(f(\tilde{x})) - \tilde{x}$ points approximately towards the nearest point on the manifold, since $g(f(\tilde{x}))$ estimates the center of mass of the clean points x which could have given rise to $\tilde{x}$.
- The autoencoder thus learns a vector field $g(f(x)) - x$ indicated by the green arrows.
- This vector field estimates the score $\nabla x \log p_{data}(x)$ up to a multiplicative factor that is the average root mean square reconstruction error.

Contractive autoencoder

- With an explicit regularizer on the code h = f(x),

  encouraging the derivatives of f to be as small as possible:

$$\Omega(h) = \lambda \, || \, \partial f(x) / \partial x \, ||^2_F$$

- The penalty Ω(h) is the squared Frobenius norm (sum of squared elements) of the Jacobian matrix of partial derivatives associated with the encoder function.
- Denoising autoencoders make the reconstruction function resist small but finite-sized perturbations of the input, while contractive autoencoders make the feature extraction function resist infinitesimal perturbations of the input.
- When the Ω(h) penalty is applied to sigmoidal units, one easy way to shrink the Jacobian, is to make the sigmoid units saturate to 0 or 1.

Contractive encoder

- The name contractive arises from the way that the CAE warps space.
- Specifically, because the CAE is trained to resist perturbations of its input,
- It is encouraged to map a neighborhood of input points to a smaller neighborhood of output points.
- We can think of this as contracting the input neighborhood to a smaller output neighborhood.

Predictive Sparse Decomposition

- PSD is a hybrid of sparse coding and parametric autoencoders.
- The model consists of an encoder f (x) and a decoder g(h) that are both parametric.
- Objective function to be optimized：

$$\min_h \; ||x - g(h)||^2 + \lambda|h|_1 + \gamma||h - f(x)||^2.$$

- The training algorithm alternates between minimization with respect to h and minimization with respect to the model parameters.
- Minimization with respect to h is fast because f(x) provides a good initial value of h and the cost function constrains h to remain near f (x) anyway.
- Simple gradient descent can obtain reasonable values of h in as few as ten steps.

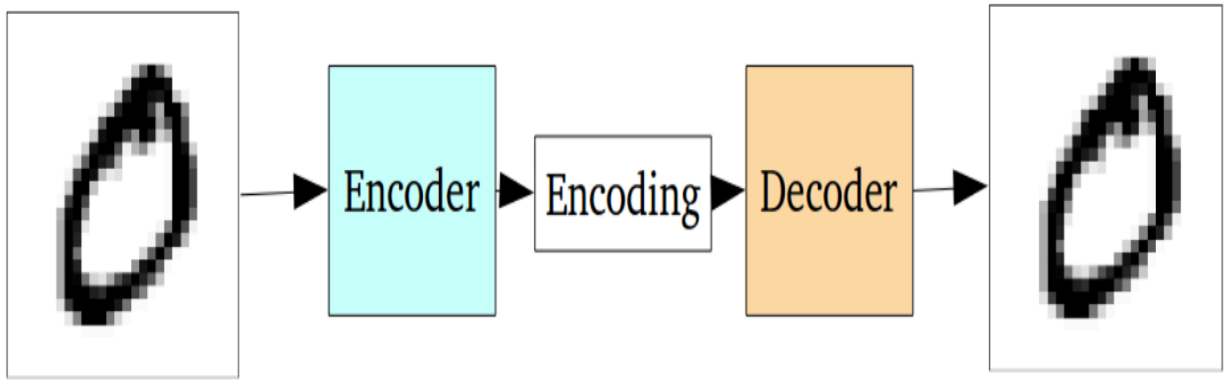
Applications of Autoencoders

- Unsupervised learning
- dimension reduction
- Semantic encoding (word vectors)

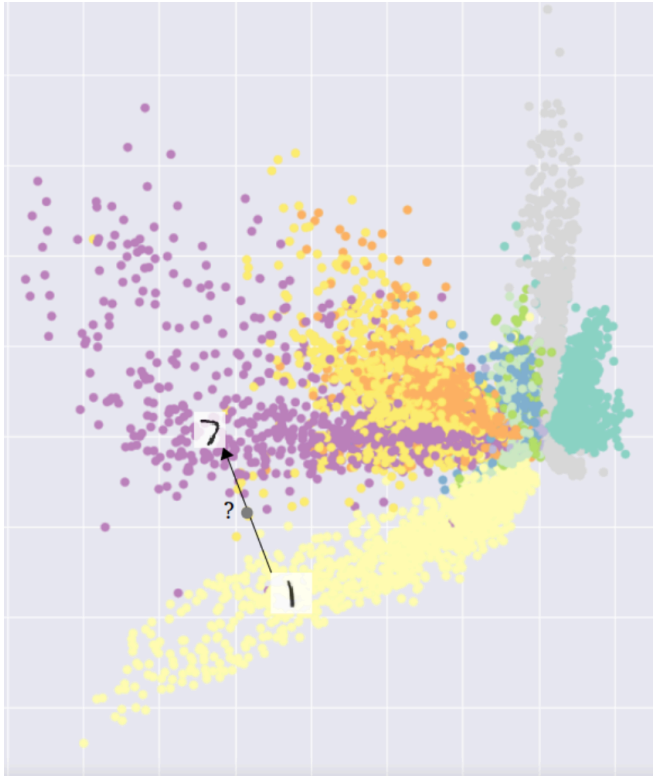
- Two interesting practical applications of autoencoders are:

1. data denoising and dimensionality reduction for data visualization.
2. With appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

Problems in autoencoders



A standard Autoencoder



The fundamental problem for generation with autoencoders is that the latent space may not be continuous, or allow easy interpolation.

Variational autoencoder

- The latent spaces of variational autoencoders are, *by design,* continuous, allowing easy random sampling and interpolation.

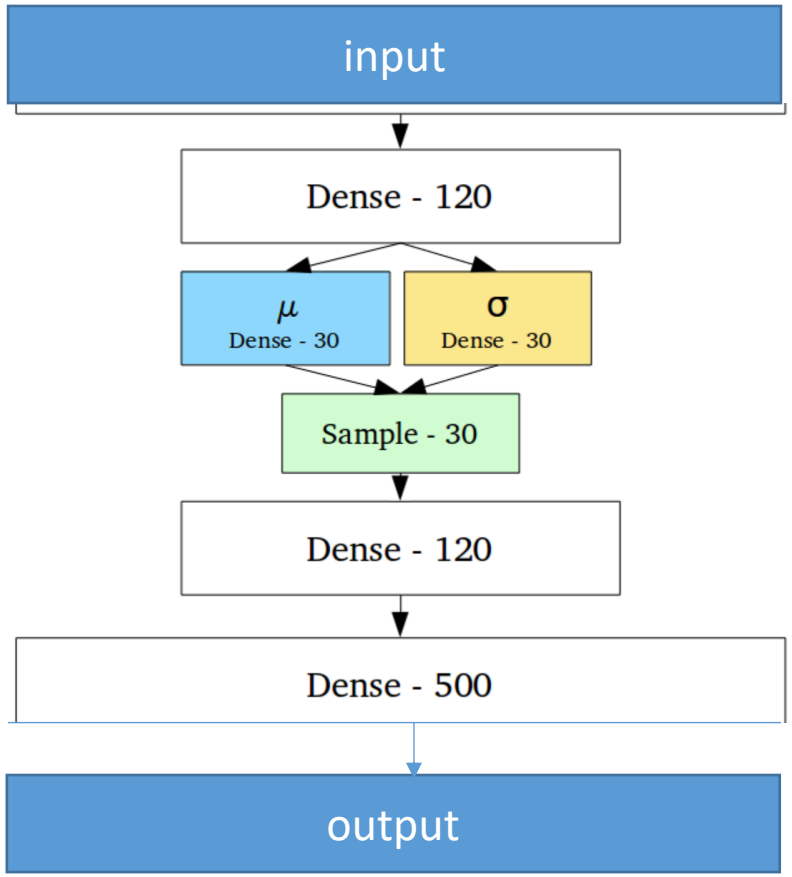
- The encoder does not output an encoding vector of size *n, but* rather, outputting two vectors of size n:

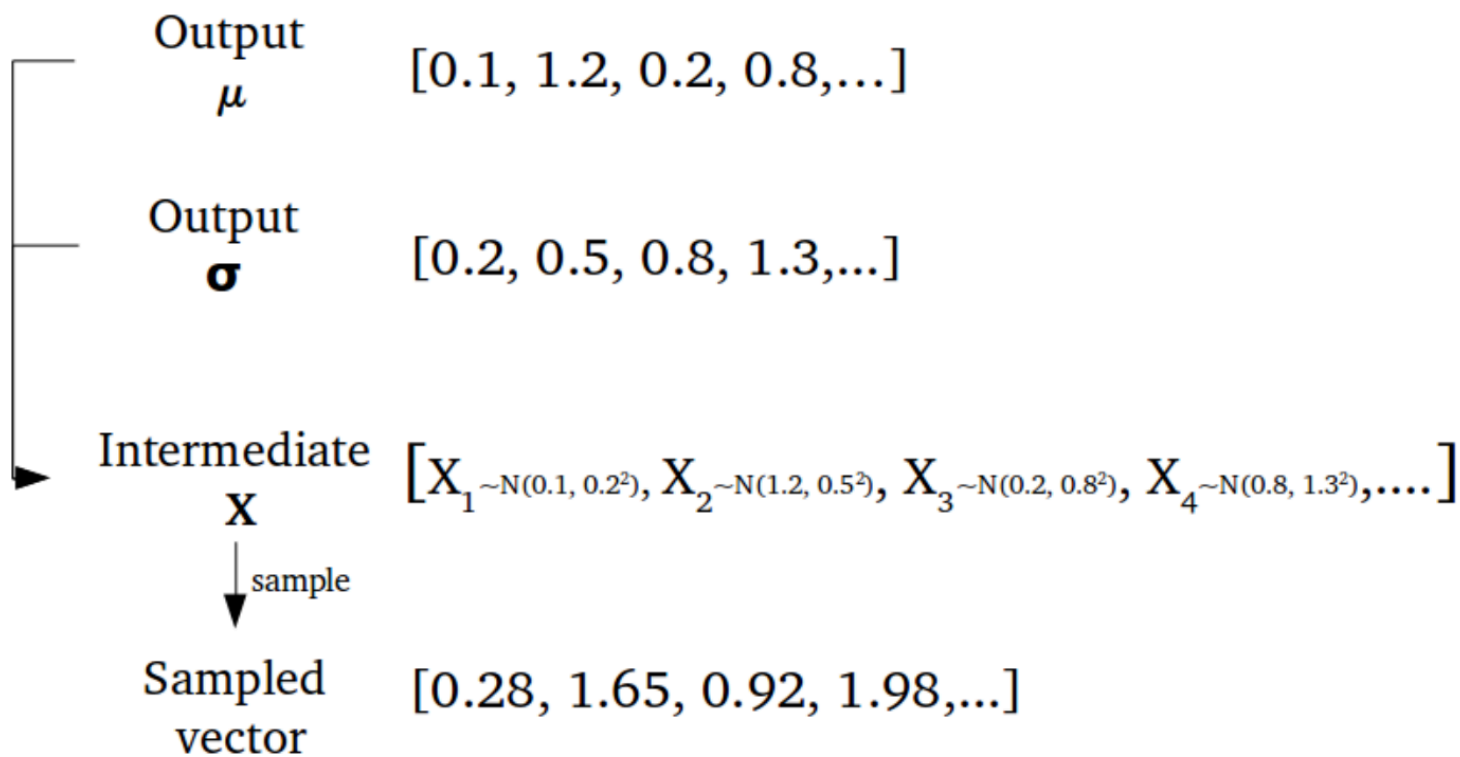
  a vector of means**, μ**, and another vector of standard deviations, **σ**.

- The mathematical basis of VAEs actually has relatively little to do with classical autoencoders, e.g. sparse autoencoders or denoising autoencoders

-

input


input
Dense - 120
μ
Dense - 30
σ
Dense - 30
Sample - 30
Dense - 120
Dense - 500
output



Stochastically generating encoding vectors

KL-loss for VAE

Minimizing the KL divergence here means optimizing the probability distribution parameters **(μ** and **σ) to closely resemble that of the target distribution.**

For VAEs, the KL loss is equivalent to the *sum* of all the KL divergences between the *component* $X_i \sim N(\mu_i, \sigma_i^2)$ in **X,** **and the standard normal**.

It's minimized when $\mu_i = 0$, $\sigma_i = 1$.

$$\sum_i \text{KL\_loss(Xi)} = \sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

Inference network and generative network

- parametrize the approximate posterior $q_\theta(z|x, \lambda)$ with an *inference network* (or encoder) that takes as input data x and outputs parameters λ.
- parametrize the likelihood p(x|z) with a *generative network* (or decoder) that takes latent variables and outputs parameters to the data distribution $p_\varphi(x|z)$

- $ELBO_i(\vartheta,\phi) = E_{q\vartheta(z|xi)}[\log p_\phi(x_i|z)] - KL(q_\vartheta(z|x_i)||p(z))$.

VAE Total loss = Reconstruction loss + regularization loss

- The *loss function* of the variational autoencoder is the negative log- likelihood with a regularizer

The loss function l_i for data point x_i is:

$l_i(\theta, \varphi) = -E_{z\sim q\theta(z|xi)[}\log p_\varphi(x_i|z)] + KL(q_\theta(z|x_i)||p(z))$

= Reconstruction loss + regularization loss

$ELBO(\lambda) = \mathbf{E}_q[\log p(x, z)] - \mathbf{E}_q[\log q_\lambda(z|x)].$

ELBO includes the inference (encoding) and generative network (decoding) parameters as:

$ELBO_i(\theta,\varphi) = E_{q\theta}(z|x_i)[\log_p\varphi(x_i|z)] - KL(_{q\theta}(z|x_i)||p(z)).$

$\text{Log } p(x) = ELBO(\lambda) + KL(q\lambda(z|x)||p(z|x))$

ELBO for a single data point in the variational autoencoder:

$ELBO_i(\lambda) = E_{q\lambda}(z|x_i)[\log p(x_i|z)] - KL(_{q\lambda}(z|x_i)||p(z)).$

Convolutional neural networks

Von-Wun Soo

Convolution operation

- $s(t) = \int x(a)\, w(t-a)\, da$

$$s(t) = (x * w)(t)$$

- A 2-dimensional image I(m,n) with 2 dimensional kernel K:
- $S(i, j) = (I * K)(i, j) = \boldsymbol{\Sigma_m \Sigma_n}\, I(m, n)\, K(i - m, j - n)$

Or

- $S(i, j) = (K * I)(i, j) = \boldsymbol{\Sigma_m \Sigma_n}\, I(i - m, j - n) K(m, n)$
- https://en.wikipedia.org/wiki/Convolution
- https://en.wikipedia.org/wiki/Convolution

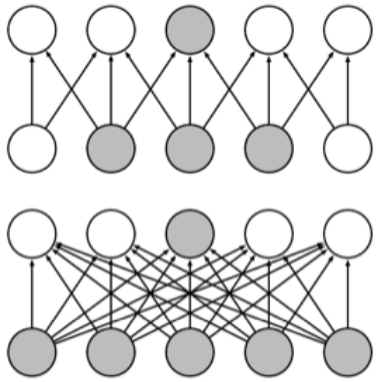
Convolution function properties

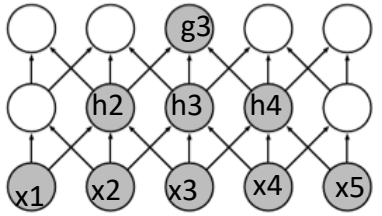
- Sparse interaction: sparse connectivity
- Parameter sharing: kernel filter can be shared
- Equivariant representation: input change then output change at the same time

  A function f(x) is equivariant to a function g if f(g(x)) = g(f(x)).
- Shift-invariant: $I(x, y) = I(x - 1, y)$.

receptive field

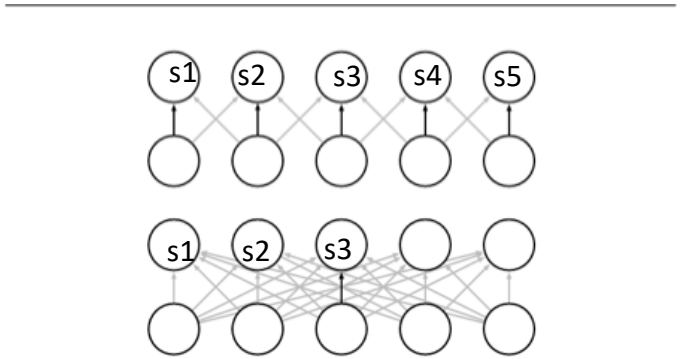
- The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers.
- Even the direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image.





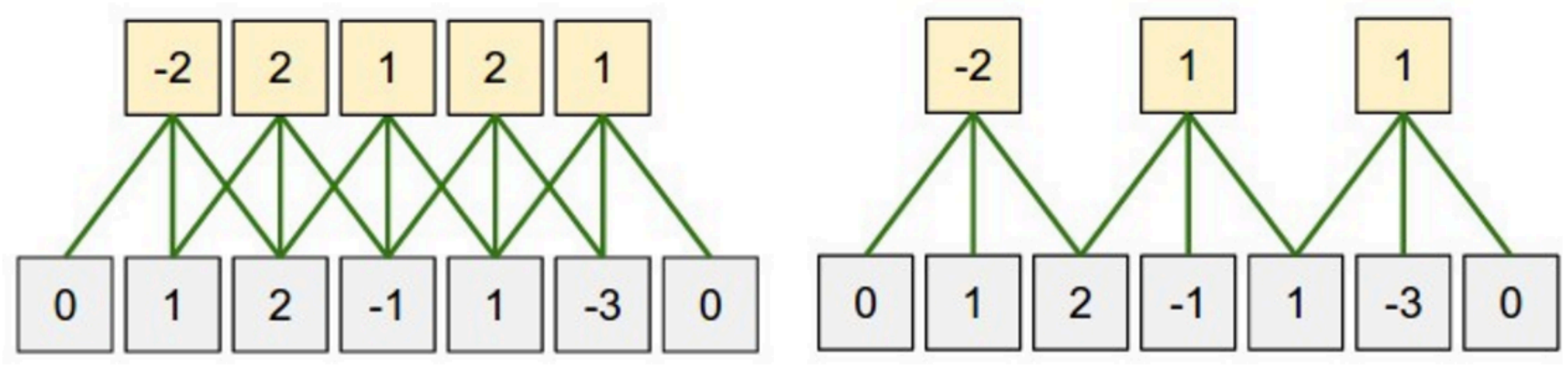
Parameter sharing:

- Top) The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations.
- (Bottom) The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once



Strider size

- How much you want to shift your filter at each step?
- A larger stride size leads to fewer applications of the filter and a smaller output size.



Convolution Stride Size. Left: Stride size 1. Right: Stride size 2.
Source: http://cs231n.github.io/convolutional-networks/

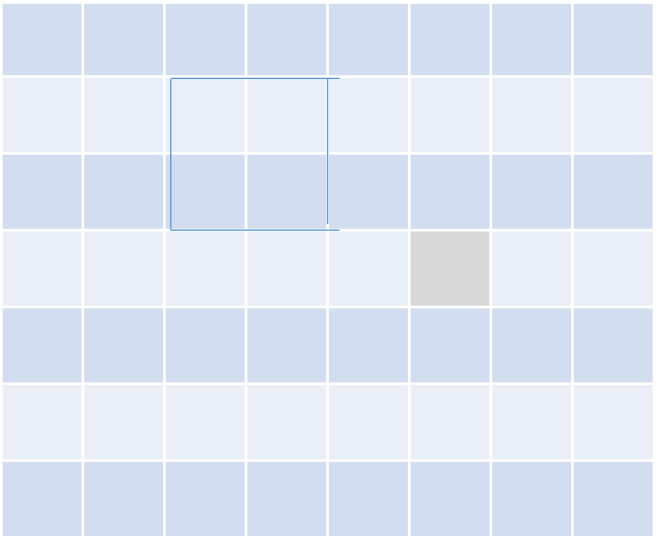
Three stages in CNN

1. Convolution layer: Input affine transform stage
2. Detector stage: rectifier
3. Pooling modifies outputs of detectors further by summarizing nearby outputs
    - helps to make the representation become approximately invariant to small translations of the input

Pooling

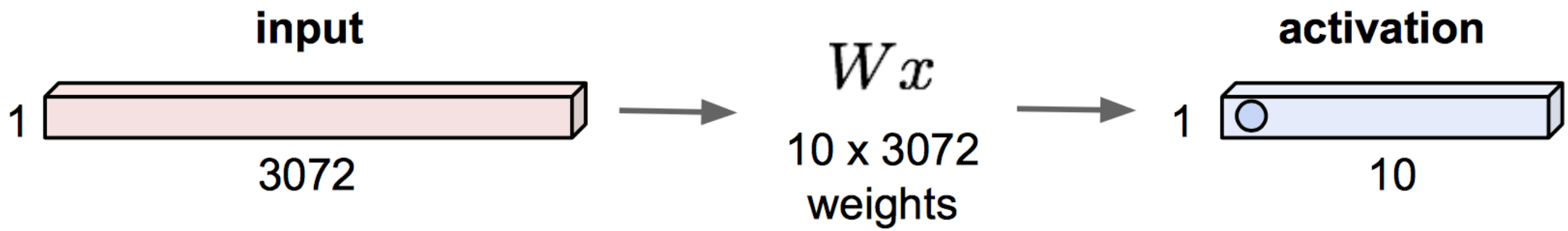
- Aggregate statistics (mean or maximal) of features at various locations to reduce dimension.
- Summarize local “receptive field”.

- A convolution (or a local neural network feature detector) maps a region of an image to a feature map.



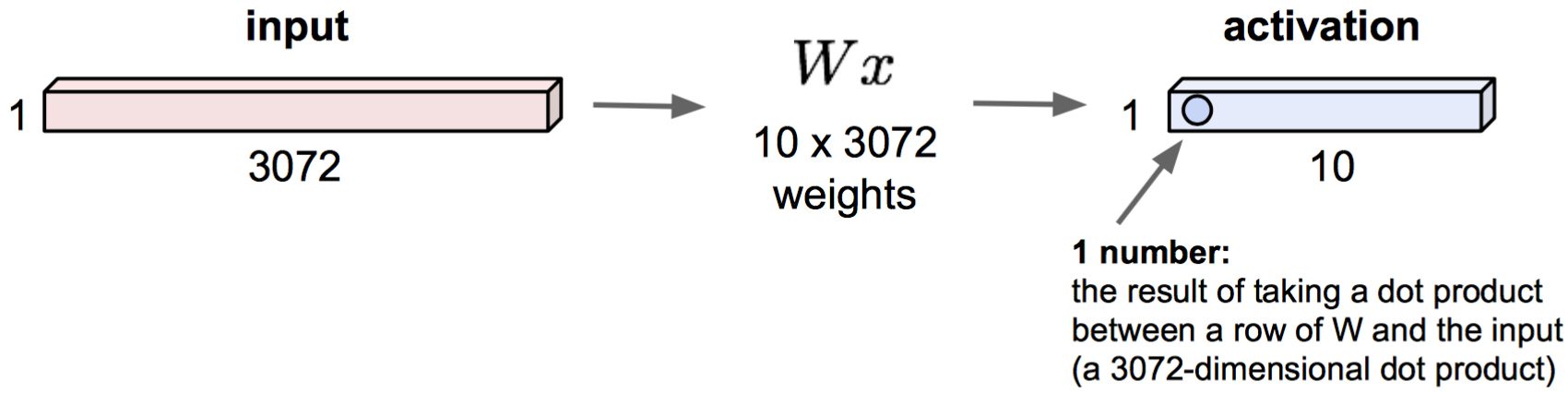
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

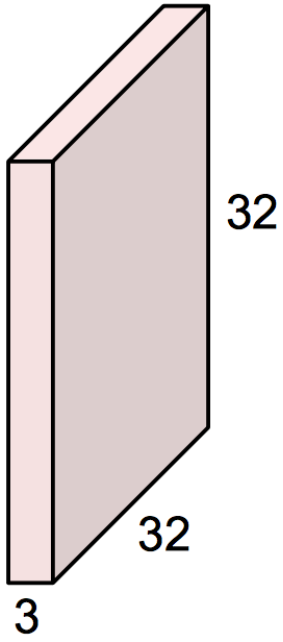
32x32x3 image -> stretch to 3072 x 1



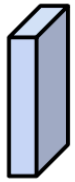
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

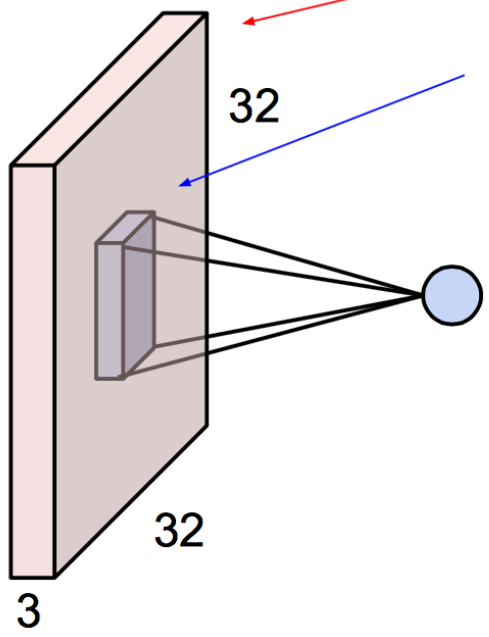


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



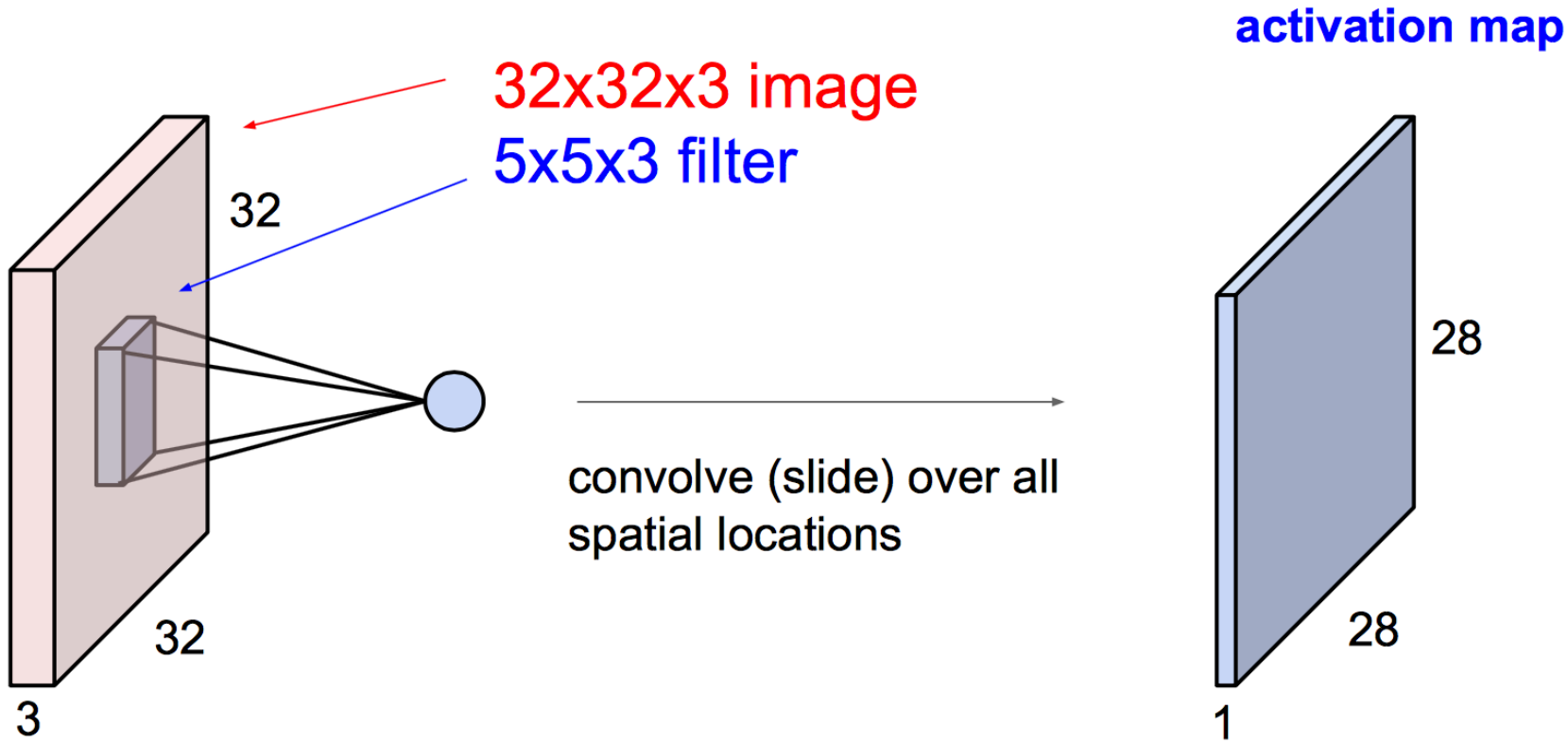
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

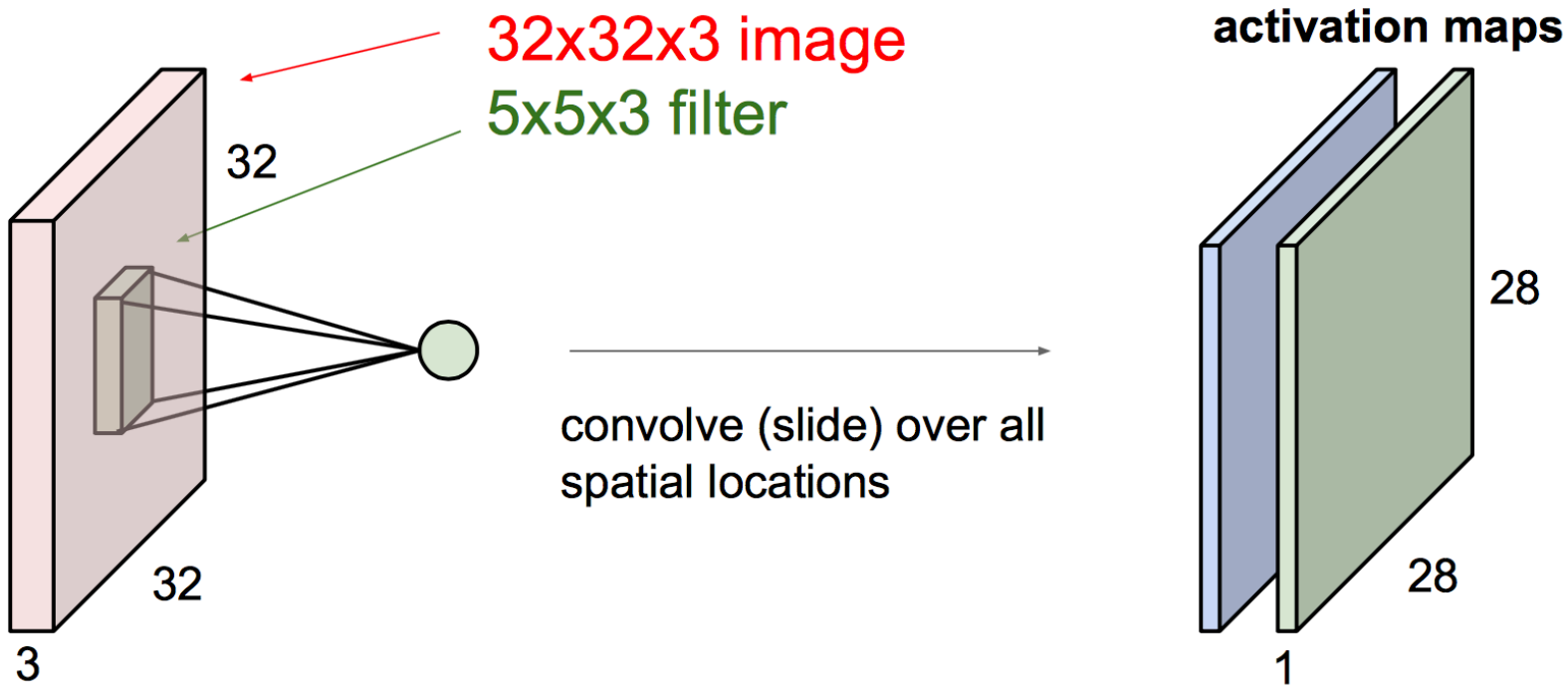
$$w^T x + b$$

Convolution Layer

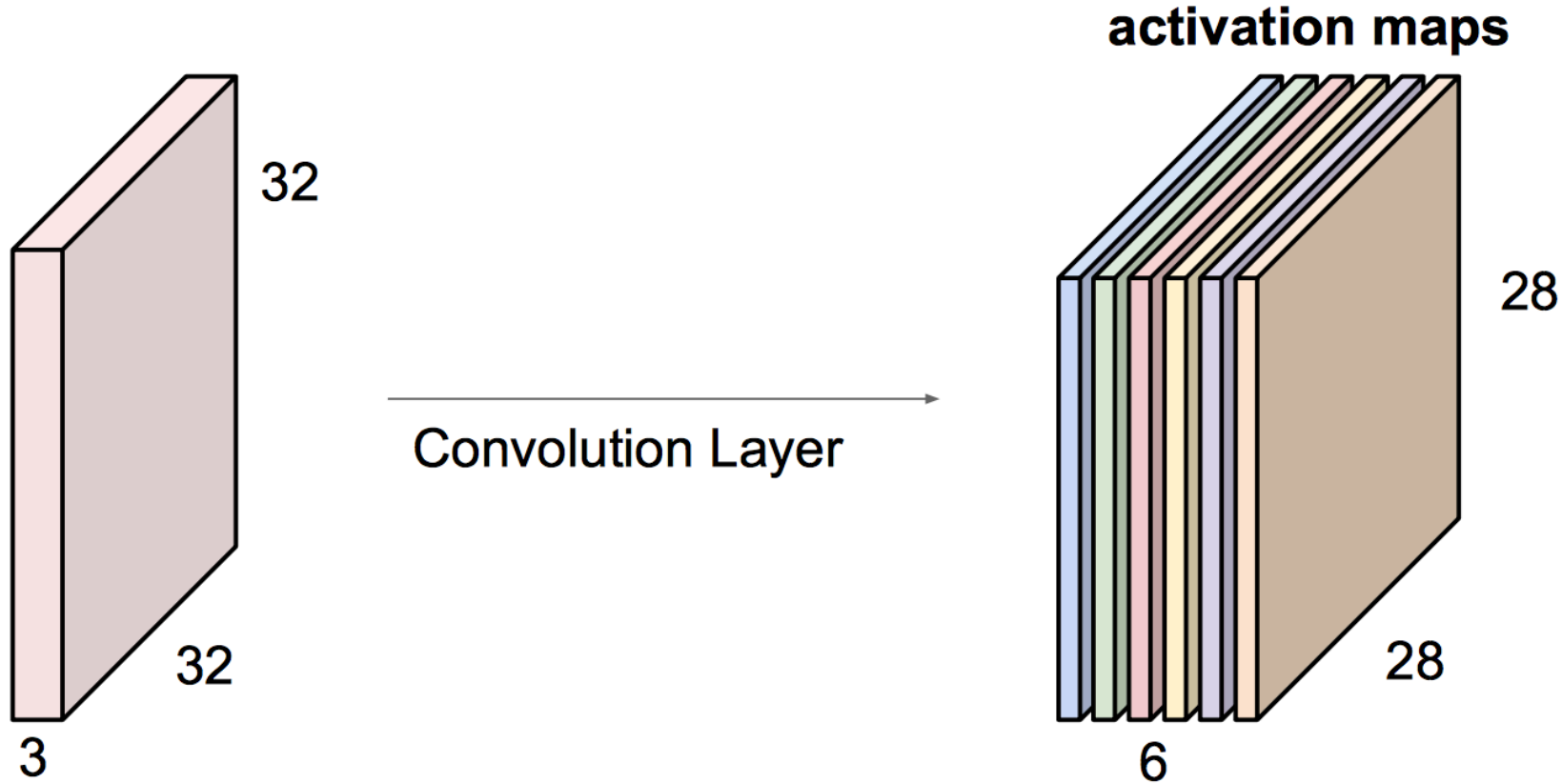


Convolution Layer

consider a second, green filter

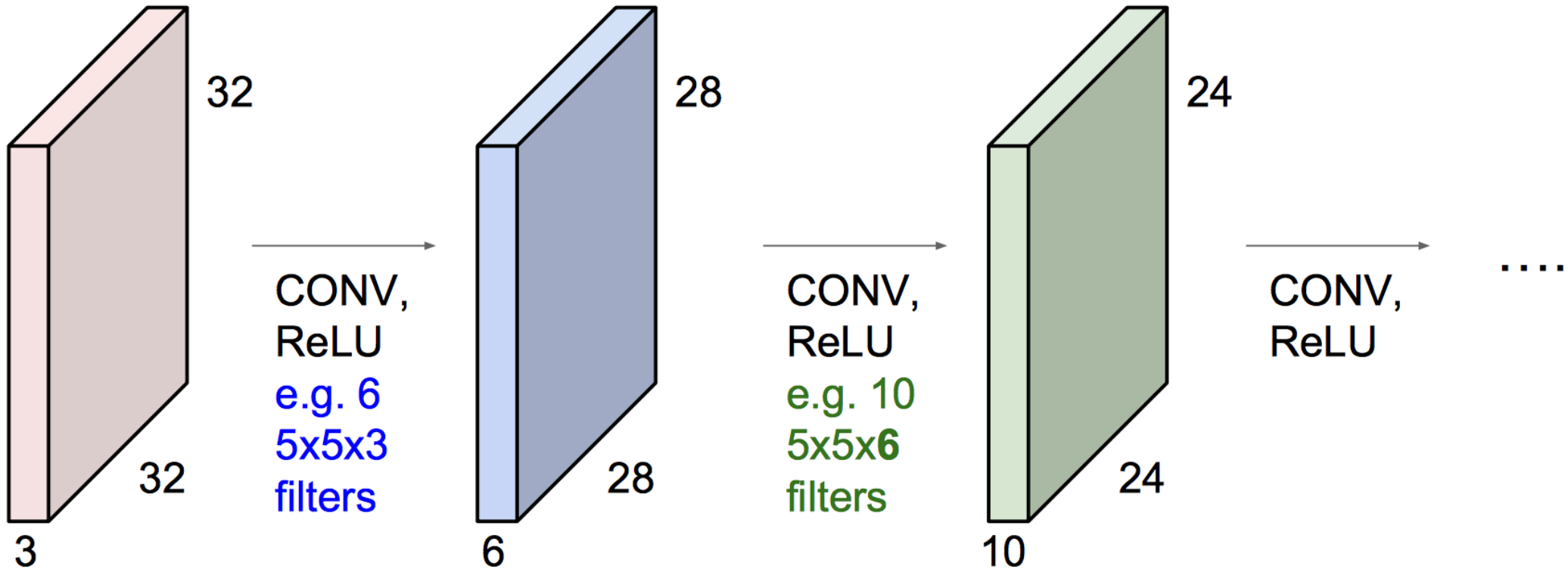


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



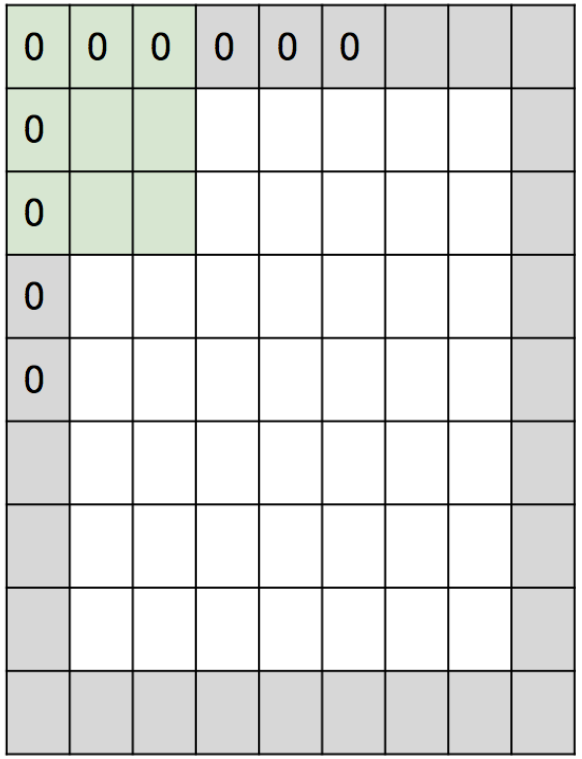
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Zero padding

In practice: Common to zero pad the border

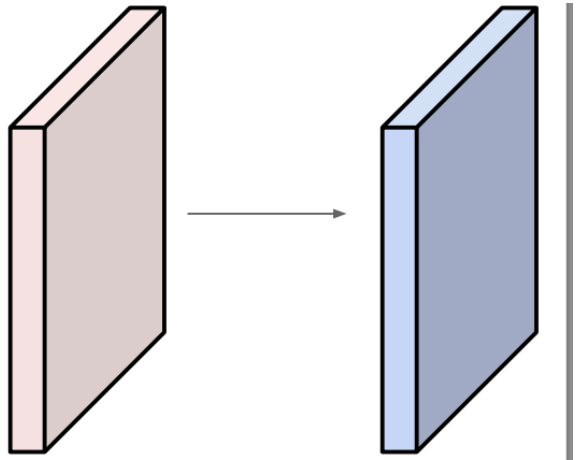


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Volume change between convolution layers

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Pooling

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

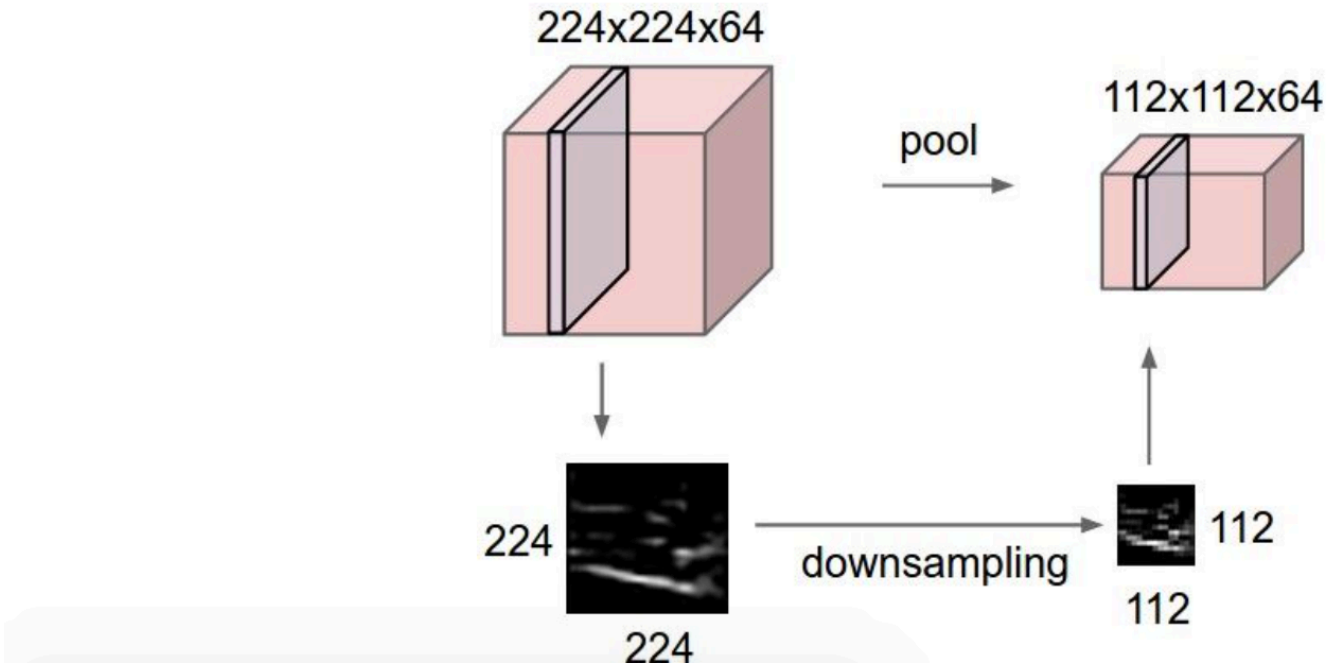


Image size NxN; filter-size WxW; Stride S; zero padding P on both sides

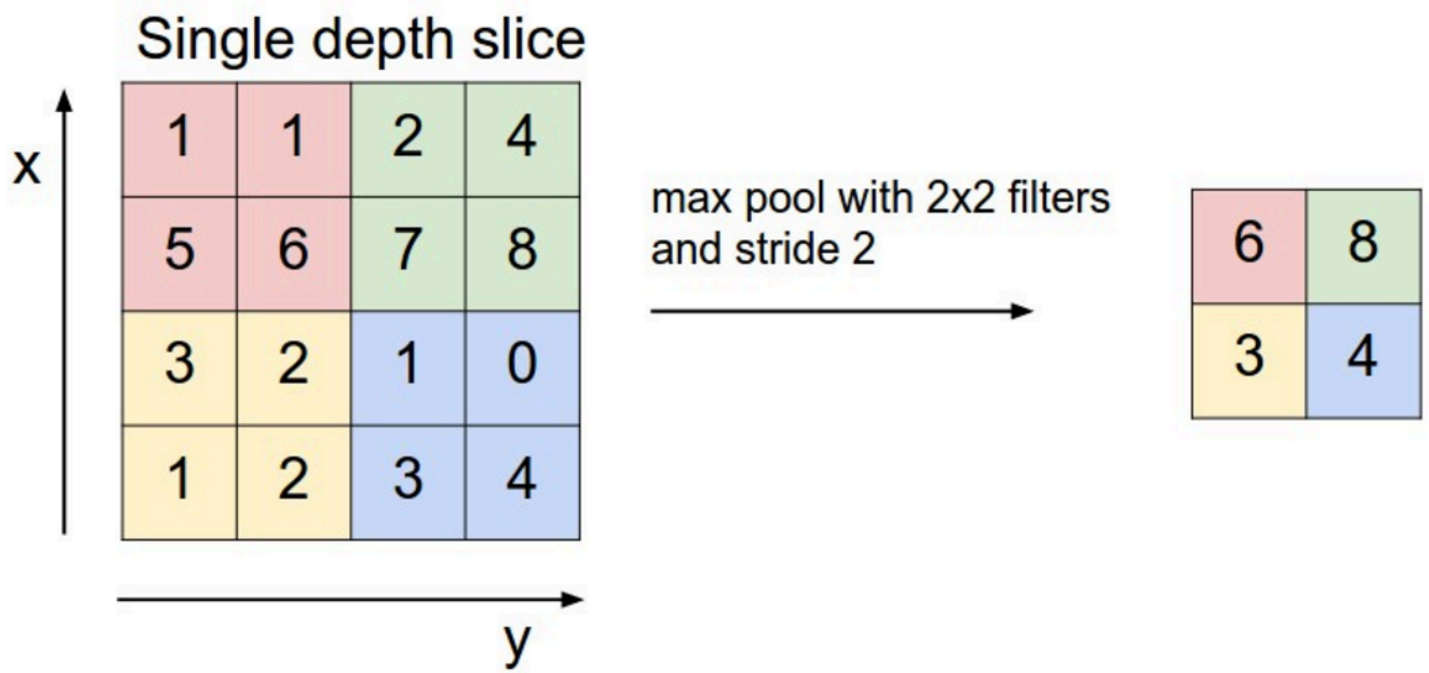
- General formula: (N+2P-W)/S + 1

Pooling properties

- Pooling is translation invariant:
- if you were to take an MNIST digit and translate it left or right, you would want your classifier to still accurately classify it as the same digit regardless of its final position.

- What is suitable pooling size?

Max pooling in CNN. Source: http://cs231n.github.io/convolutional-networks/#pool

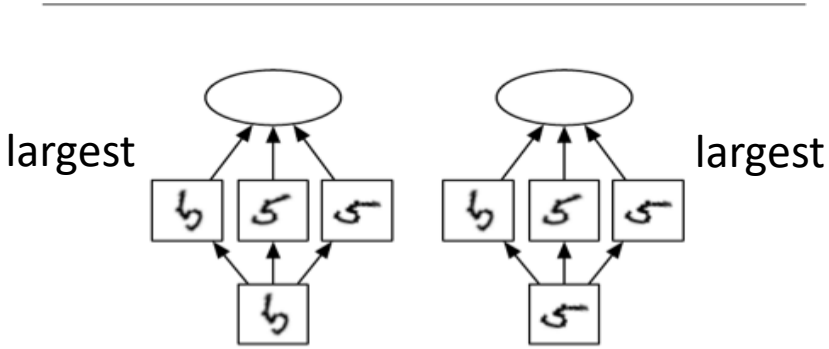


Down subsampling

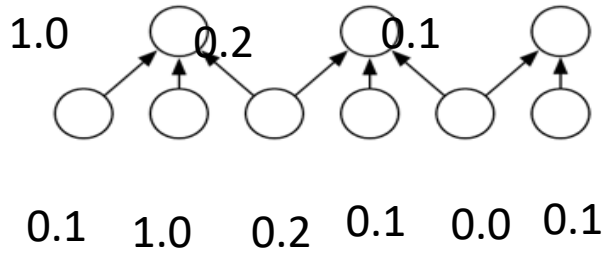
- Downsampling:
- match the imbalance classes of training data by sampling from the more frequent class
- Pooling with downsampling.
- If you have a 16x16 input layer, and apply 2:1 downsampling, you end up with a 8x8 layer. Each "pixel" in the new layer represents 4 in the input layer

Pooling with down sampling

- We use max-pooling with a pool width of three and a stride between pools of two.
- This reduces the representation size by a factor of two, which reduces the computational and statistical burden on the next layer.
- Note that the rightmost pooling region has a smaller size, but must be included if we do not want to ignore some of the detector units.



Invariant to rotation



channel

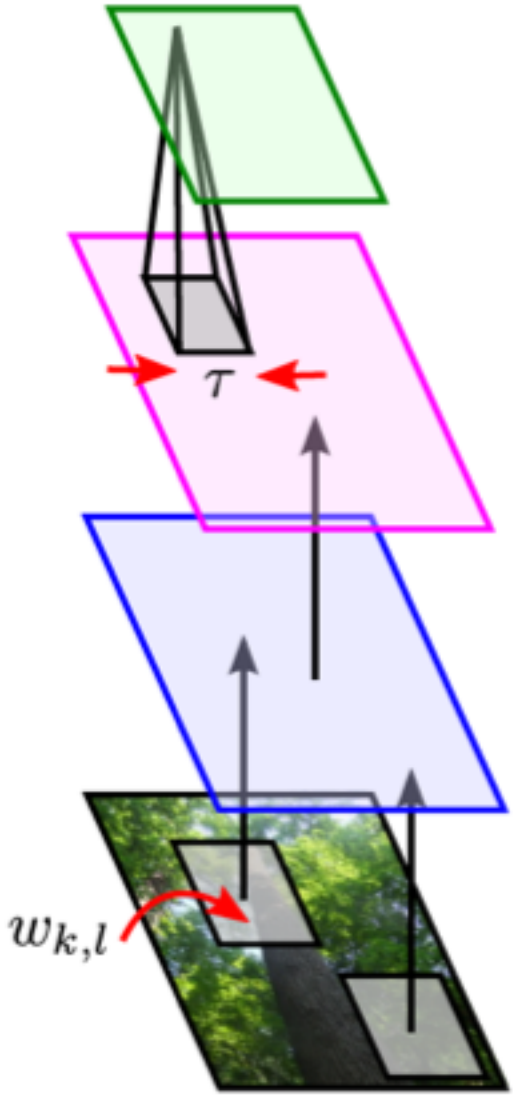
- Channels are different “views” of your input data.
- For example,
- in image recognition you typically have RGB (red, green, blue) channels
- In NLP, a sentence can be phrased in different ways.

- CNNs seem to be classifications tasks, such as Sentiment Analysis, Spam Detection or Topic Categorization.

Convolution layers

- Input to layer -> convolution stage -> detector stage -> pooling stage -> next layer
- Affine transform rectified linear max pooling

Building block of a convolutional neural network



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

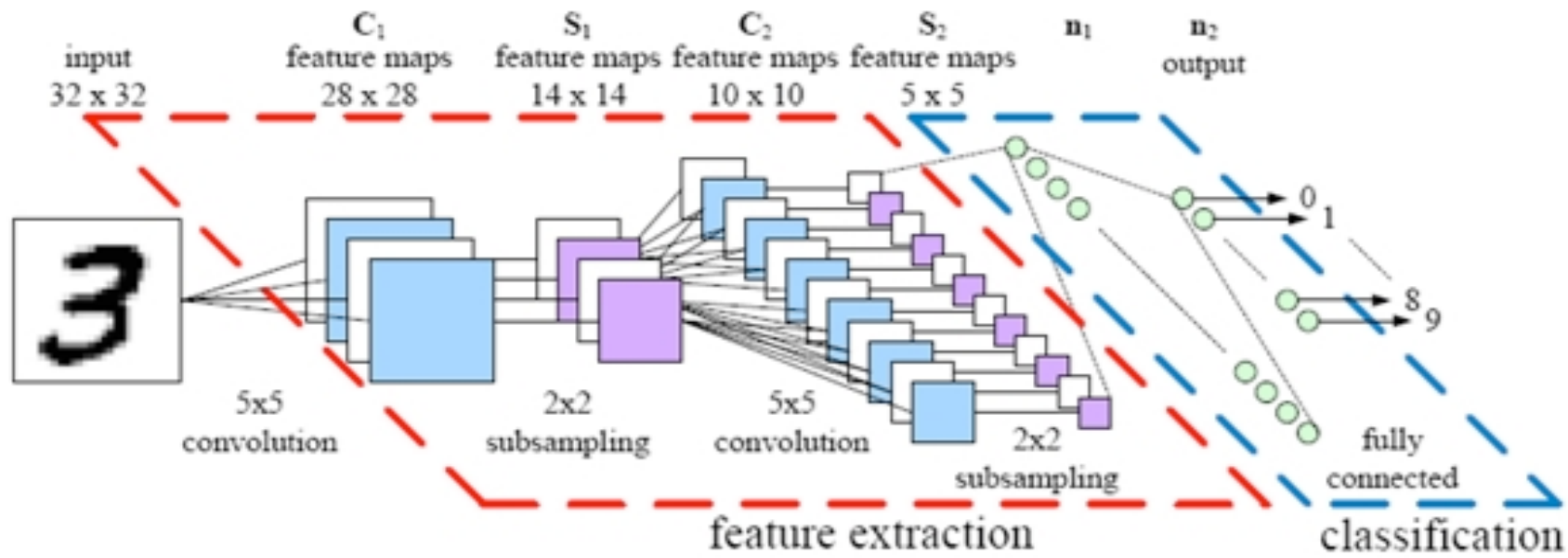
$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

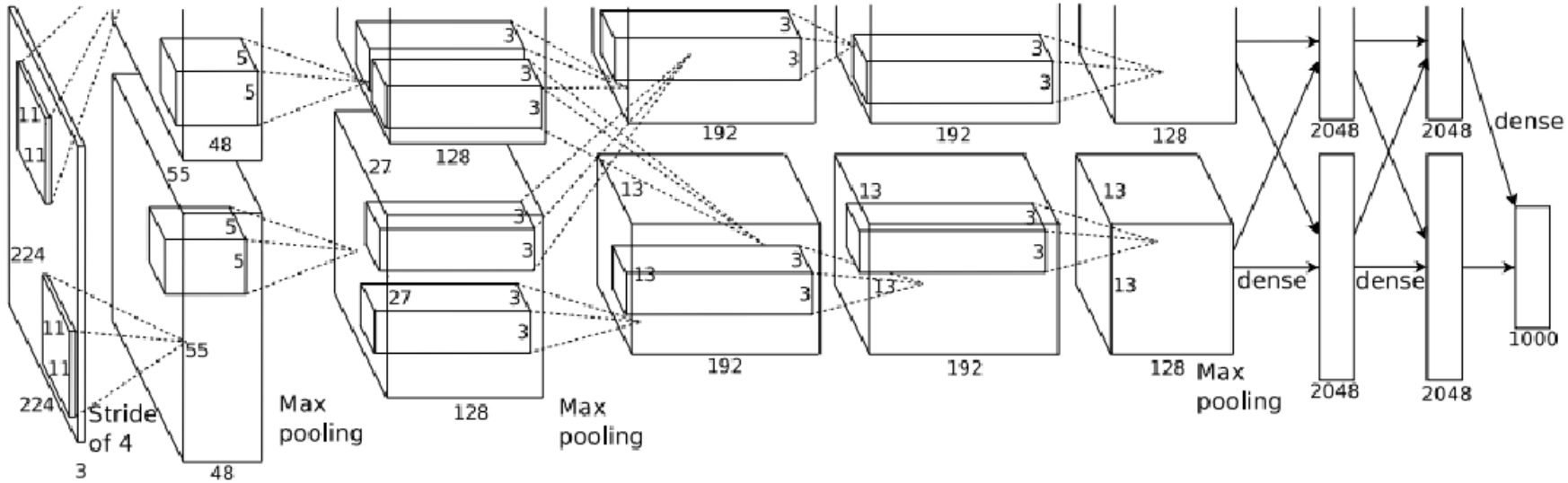
only parameters

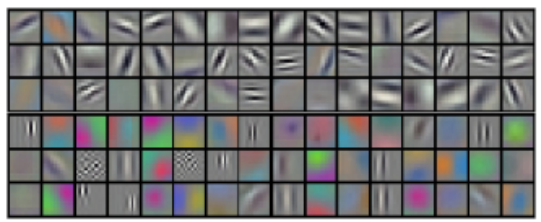
convolutional stage

$z_{i,j}$

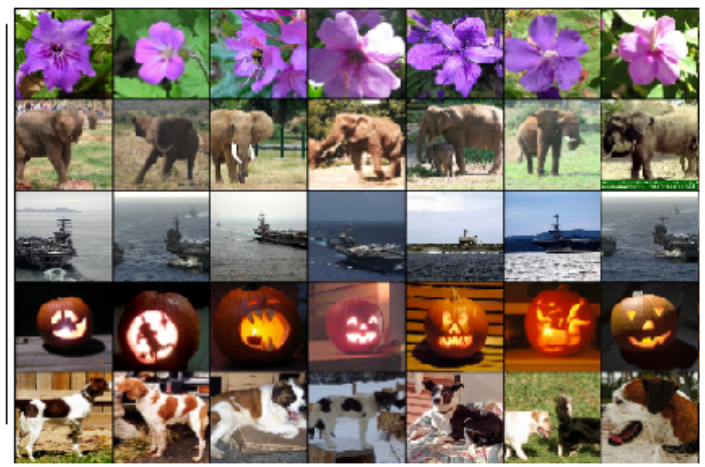
input image


input
32 x 32
C1
feature maps
28 x 28
S1
feature maps
14 x 14
C2
feature maps
10 x 10
S2
feature maps
5 x 5
n1
n2
output
0
1
8
9
5x5
convolution
2x2
subsampling
5x5
convolution
2x2
subsampling
fully
connected
feature extraction
classification


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




mite
container ship
motor scooter
leopard
grille
mushroom
cherry
Madagascar cat




dinning table
grocery store

Classes

Hyper parameters

- Learning rate
- Number of hidden units
- Convolution kernel width (for CNN)
- Implicit zero padding (for CNN)
- Weight decay coefficient
- Dropout rate

TRAINING CNN WITH BACKPROPAGATION

- Let $\delta^{(l+1)}$ be the error at l+1 layer with a cost function J(W, b; x, y) where W, b are parameters and (x , y) are training data and label pairs.
- If the l-th layer is densely connected to the (l+1)-st layer, then the error for the l-th layer is computed as
- $\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \cdot f'(z(l))$
- The gradients are:
- $\nabla_{W(l)} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T$
- $\nabla_{b(l)} J(W,b;x,y) = \delta^{(l+1)}.$

- If the l-th layer is a convolutional and subsampling layer then the error is propagated through as
- $\delta^{(l)}_k = \text{upsample}((W^{(l)}_k)^T \delta^{(l+1)}_k) \cdot f'(z^{(l)}_k)$
- Where k indexes the filter number and $f'(z^{(l)}_k)$ is the derivative of the activation function.
- The upsample operation has to propagate the error through the pooling layer by calculating the error w.r.t. to each unit incoming to the pooling layer.
- For example, if we have mean pooling then upsample simply uniformly distributes the error for a single pooling unit among the units which feed into it in the previous layer. In max pooling the unit which was chosen as the max receives all the error since very small changes in input would perturb the result only through that unit.

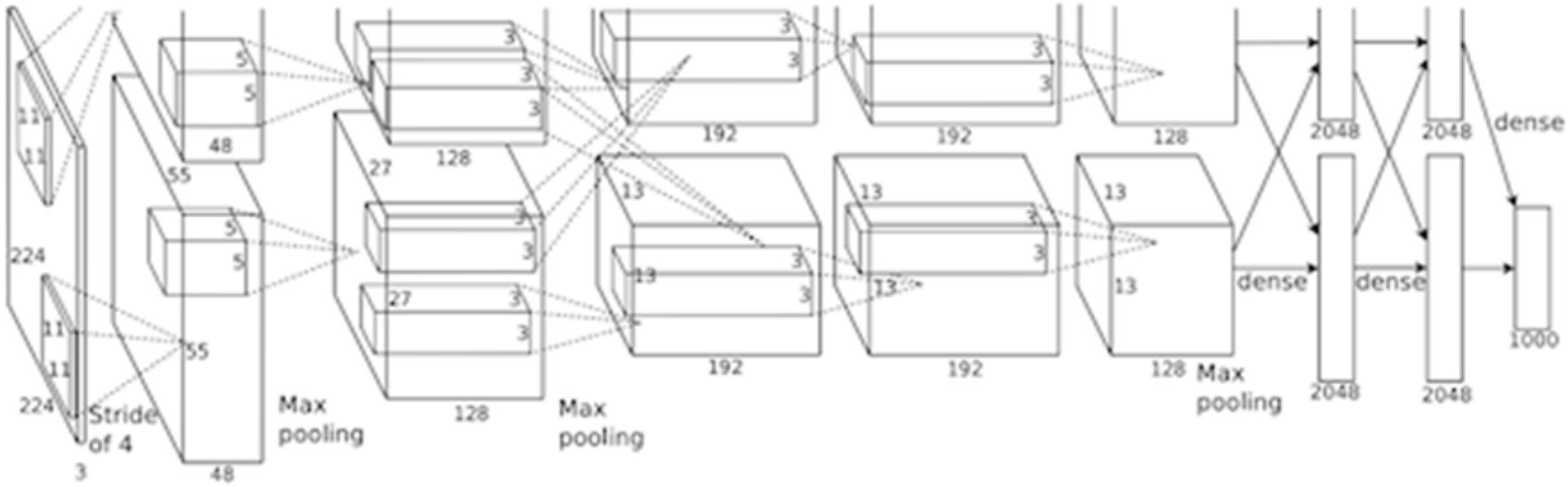
https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

- **AlexNet (2012)**
- **ZF Net (2013)**
- DeConvNet
- **VGG Net (2014)**
- **GoogLeNet (2015)**
- Inception Module
- **Microsoft ResNet (2015)**
- **Region Based CNNs (R-CNN - 2013, Fast R-CNN - 2015, Faster R-CNN - 2015)**
- **Spatial Transformer Networks (2015)**

AlexNet

- Yann LeCun's paper: ImageNet Classification with Deep Convolutional Networks" 1998.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton created a "large, deep convolutional neural network" that was used to win the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge).
- 2012 marked the first year where a CNN was used to achieve a top 5 test error rate of 15.4%
- The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers. For 1000 classes.

AlexNet architecture



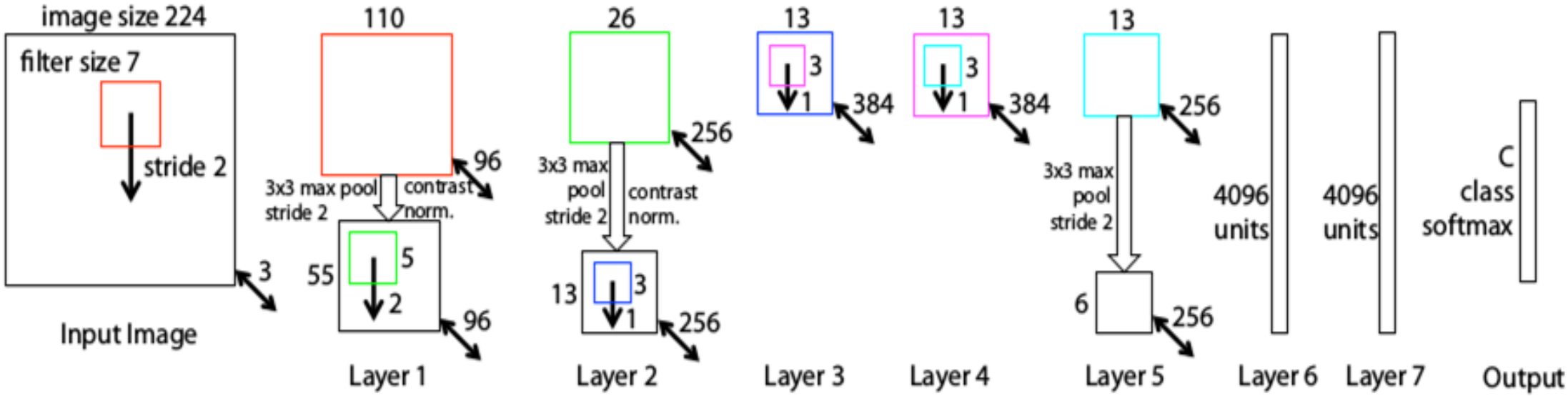
AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

- Trained the network on ImageNet data, which contained over 15 million annotated images from a total of over 22,000 categories.
- Used ReLU for the nonlinearity functions (Found to decrease training time as ReLUs are several times faster than the conventional tanh function).
- Used data augmentation techniques that consisted of image translations, horizontal reflections, and patch extractions.
- Implemented dropout layers in order to combat the problem of overfitting to the training data.
- Trained the model using batch stochastic gradient descent, with specific values for momentum and weight decay.
- Trained on two GTX 580 GPUs for **five to six days**.

ZF Net 2013

- Matthew Zeiler and Rob Fergus from NYU. Named ZF Net, this model achieved an 11.2% error rate. Won the 2013 challenge.

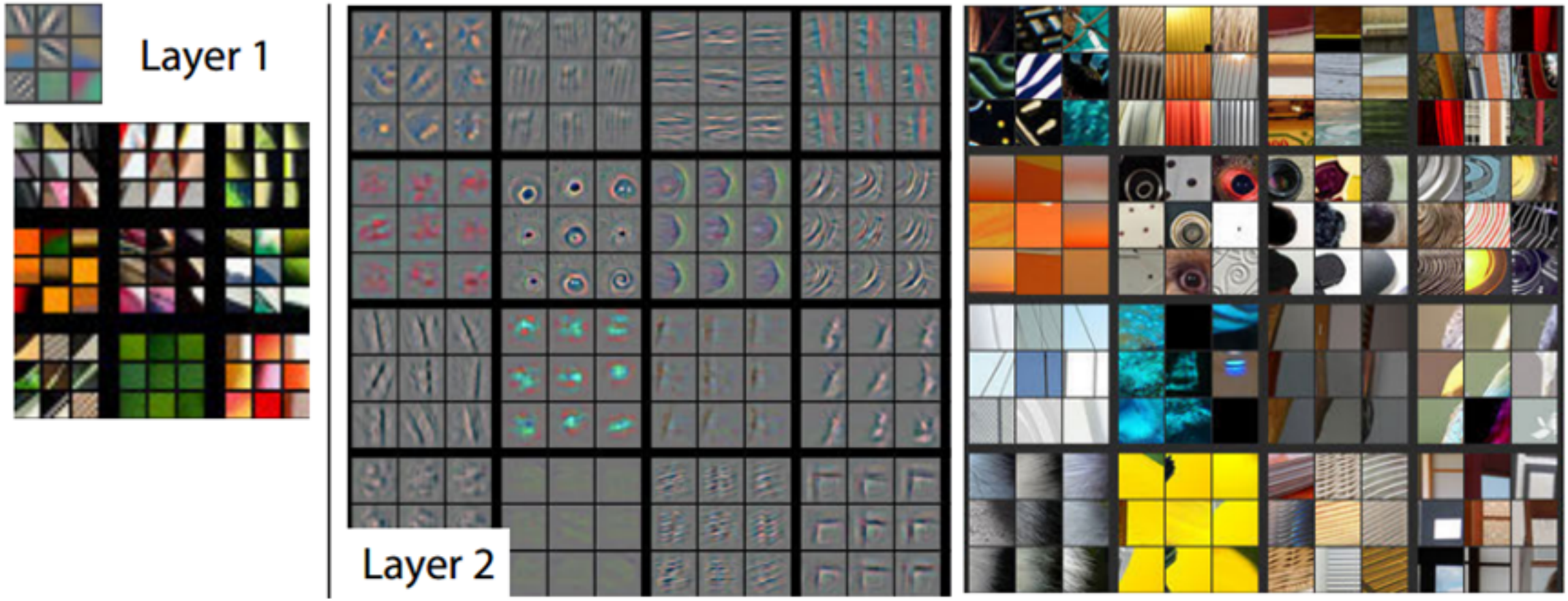
- Allow visualizing feature maps.



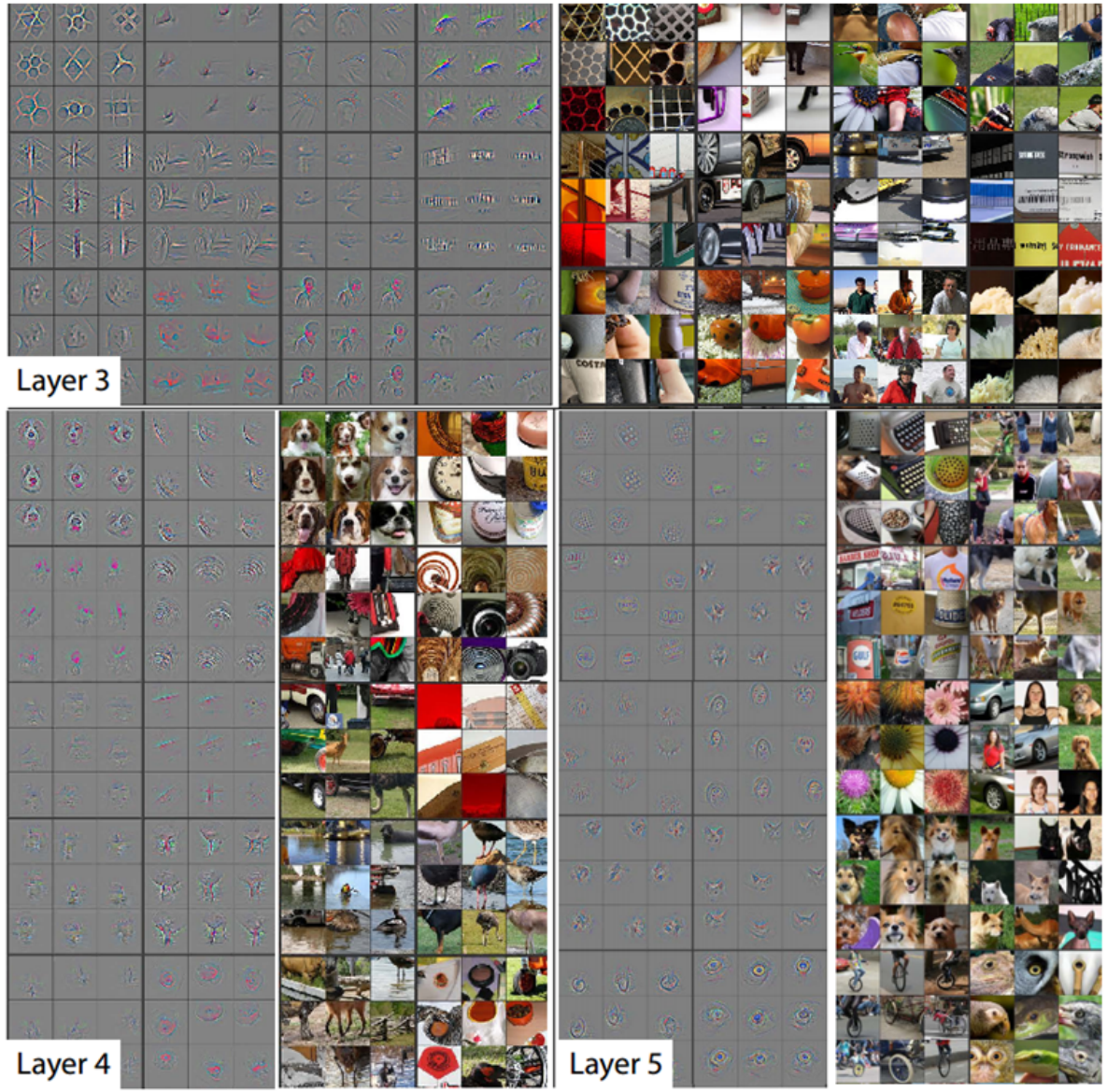
ZF Net Architecture

ZF Net

- similar architecture to AlexNet.
- AlexNet trained on 15 million images, while ZF Net trained on only 1.3 million images.
- Instead of using 11x11 sized filters in the first layer (which is what AlexNet implemented), ZF Net used filters of size 7x7 and a decreased stride value. The reasoning behind this modification is that a smaller filter size in the first conv layer helps retain a lot of original pixel information in the input volume.
- rise in the number of filters used.
- Used ReLUs for their activation functions, cross-entropy loss for the error function, and trained using batch stochastic gradient descent.
- Trained on a GTX 580 GPU for **twelve days**.
- Developed a visualization technique named Deconvolutional Network, to examine different feature activations and their relation to the input space.
- deconvnet" maps features to pixels (the opposite of what a convolutional layer does).



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)



Visualizations of Layers 3, 4, and 5

VGG Net 2014

- The model wasn't the winner of ILSVRC 2014) but with 7.3% error rate.
- Karen Simonyan and Andrew Zisserman of the University of Oxford created a 19 layer CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 maxpooling layers with stride 2

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB imag)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The 6 different architecures of VGG Net. Configuration D produced the best results

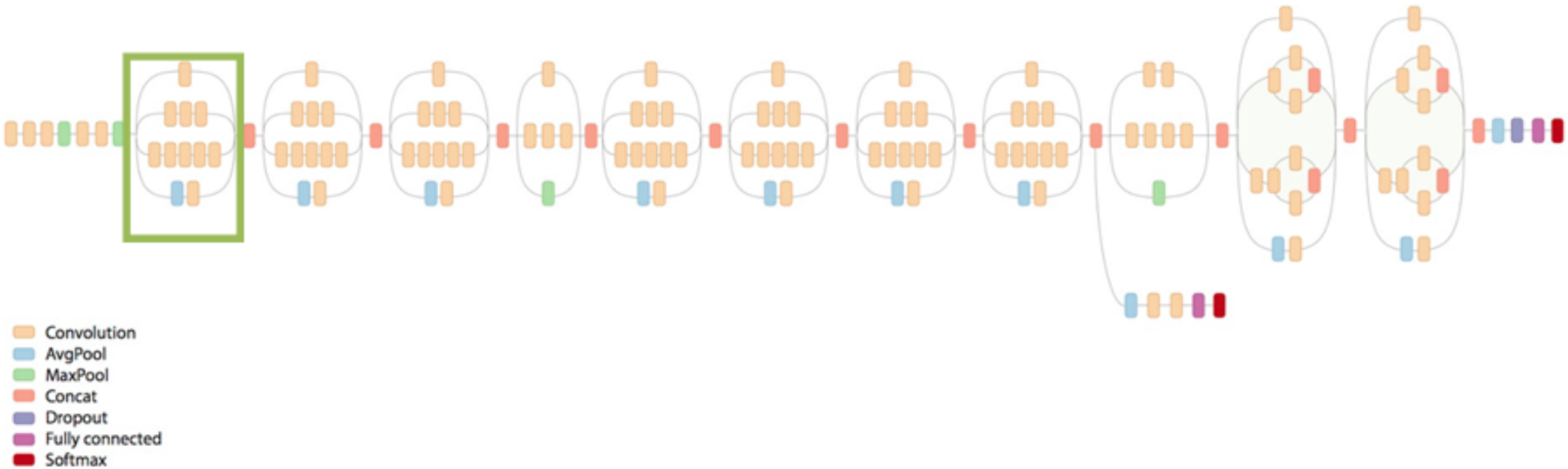
VGG net Keeps simple but deep!

- use only 3x3 sized filters different from AlexNet's 11x11 filters in the first layer and ZF Net's 7x7 filters.
- The benefits is a decrease in the number of parameters. Also, with two conv layers, use ReLU layers instead of one.
- 3 conv layers back to back have an effective receptive field of 7x7.
- As the spatial size of the input volumes at each layer decrease (result of the conv and pool layers), the depth of the volumes increase due to the increased number of filters as you go down the network.
- the number of filters doubles after each maxpool layer. This reinforces the idea of shrinking spatial dimensions, but growing depth.
- Worked well on both image classification and localization tasks. The authors used a form of localization as regression (see page 10 of the paper for all details).
- Used ReLU layers after each conv layer and trained with batch gradient descent.
- Trained on 4 Nvidia Titan Black GPUs for **two to three weeks**.

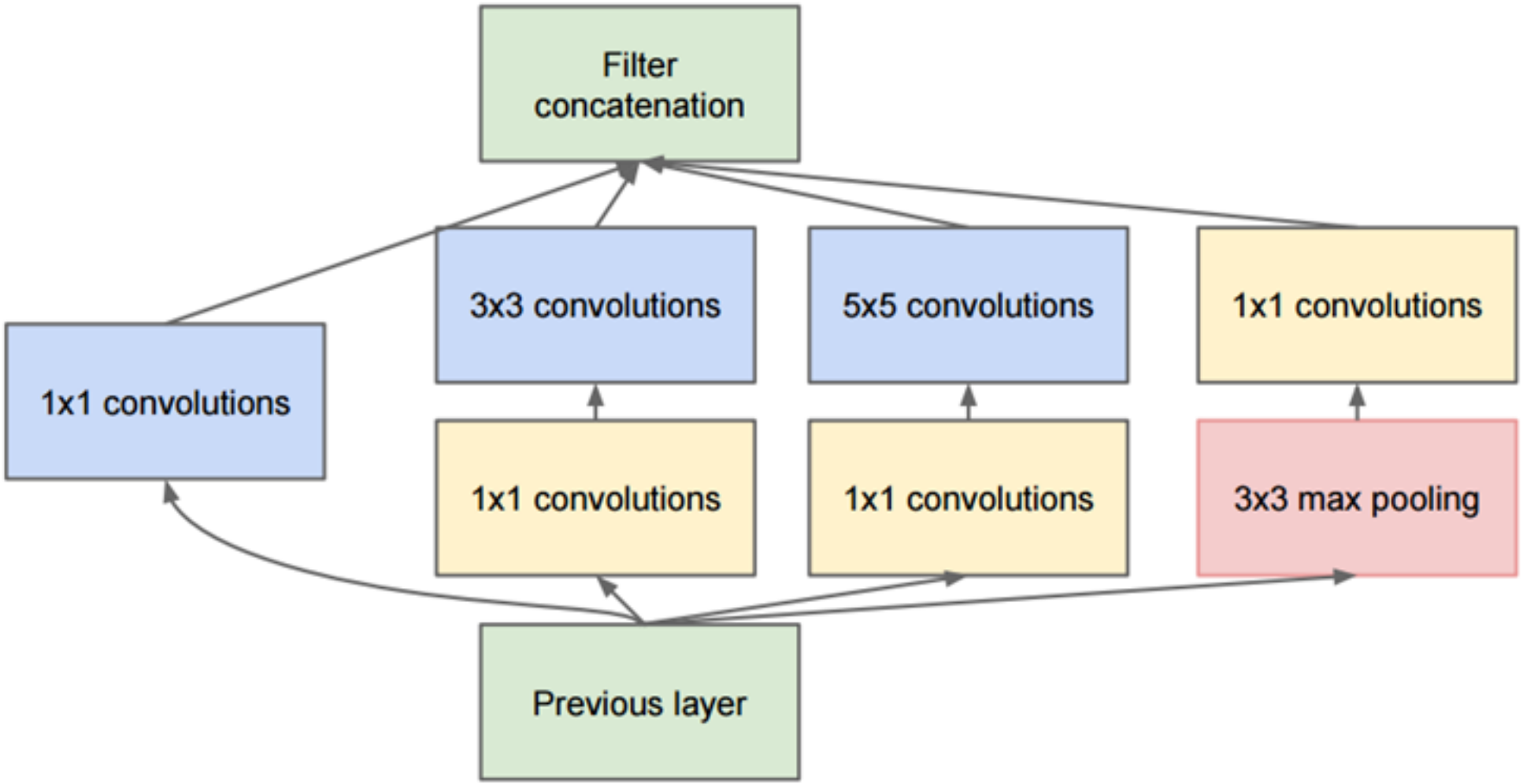
GoogLeNet (2015)

- GoogLeNet is a 22 layer CNN and was the winner of ILSVRC 2014 with a top 5 error rate of 6.7%.

- We see components of the network happening in parallel.
- The 1x1 convolutions (or network in network layer) provide a method of dimensionality reduction. (Especially to reduce depth.)



Green box shows parallel region of GoogLeNet



Full Inception module

GoogleNet2015

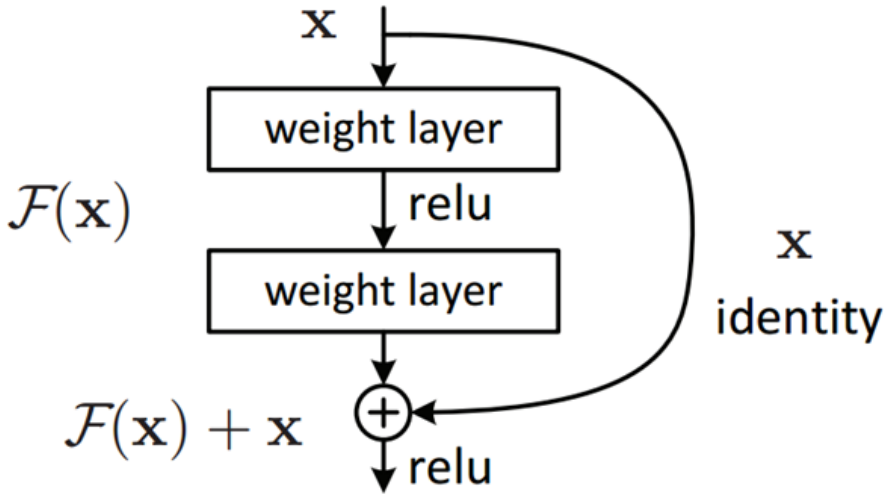
- Used 9 Inception modules in the whole architecture, with over 100 layers in total! Now that is deep…
- No use of fully connected layers! They use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.
- Uses 12x fewer parameters than AlexNet.
- During testing, multiple crops of the same image were created, fed into the network, and the softmax probabilities were averaged to give us the final solution.
- Utilized concepts from R-CNN (a paper we'll discuss later) for their detection model.
- There are updated versions to the Inception module (Versions 6 and 7).
- Trained on "a few high-end GPUs **within a week**".

ResNet (Microsoft 2015)

ResNet is a new 152 layer network architecture that set new records in classification, detection, and localization through one incredible architecture.

Aside from the new record in terms of number of layers, ResNet won ILSVRC 2015 ImageNet challenge with an incredible error rate of 3.6%

(Depending on their skill and expertise, humans generally hover around a 5-10% error rate.



A residual block

Achievement of Deep residual nets

- *For ImageNet dataset, residual nets with a depth of up to 152 layers— 8× deeper than VGG nets 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.*

- *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.*

- CIFAR-10 dataset consists of 50k training images and 10k testing images in 10 classes.

- The MS COCO dataset involves 80 object categories.
- 80k images on the train set for training and the 40k images on the validation set for evaluation.

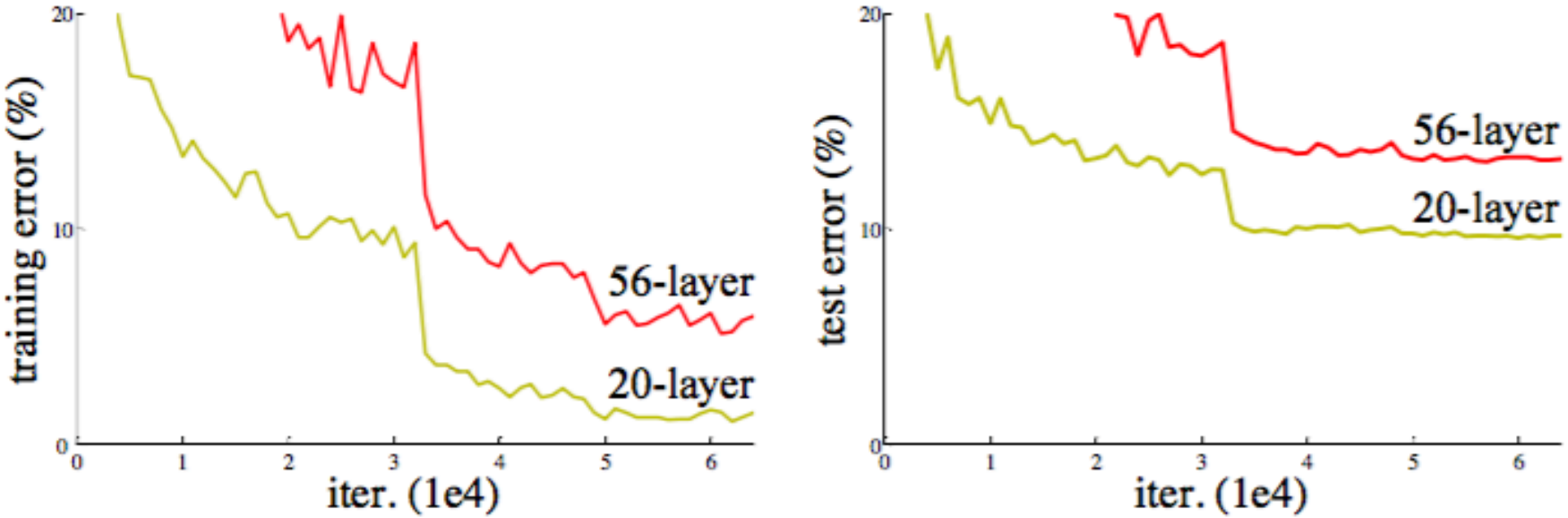
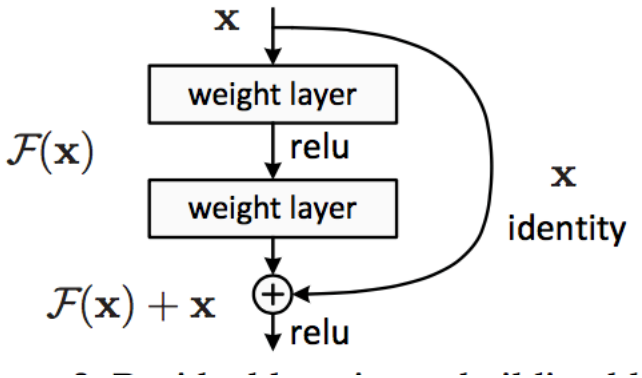


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

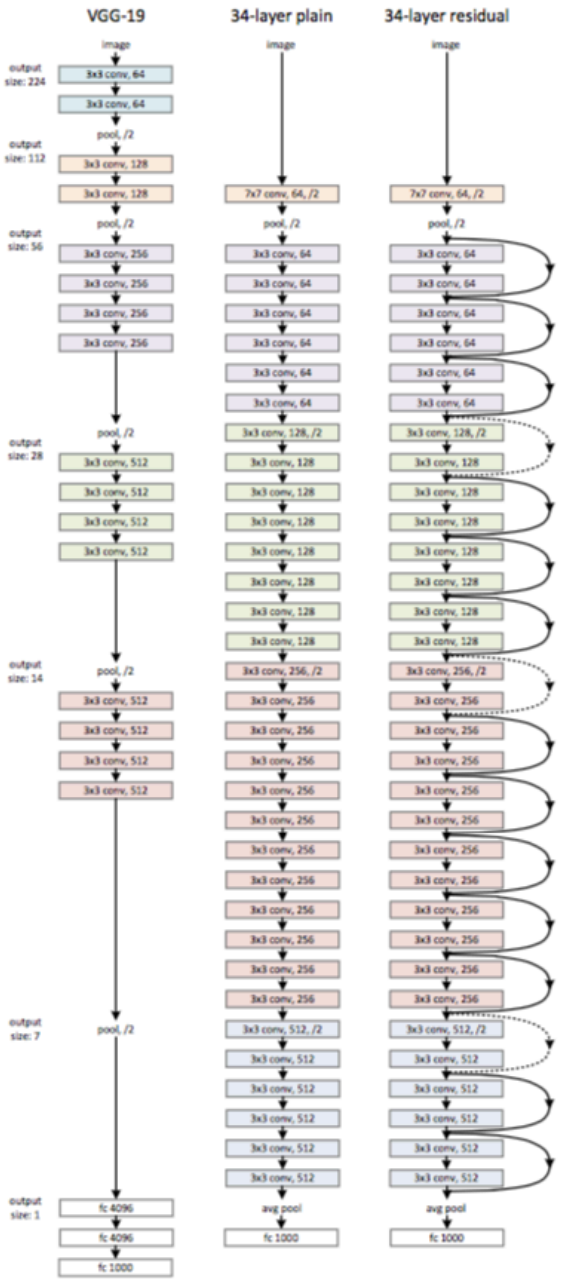
Residual learning



- y = F (x, {Wi }) + x.

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

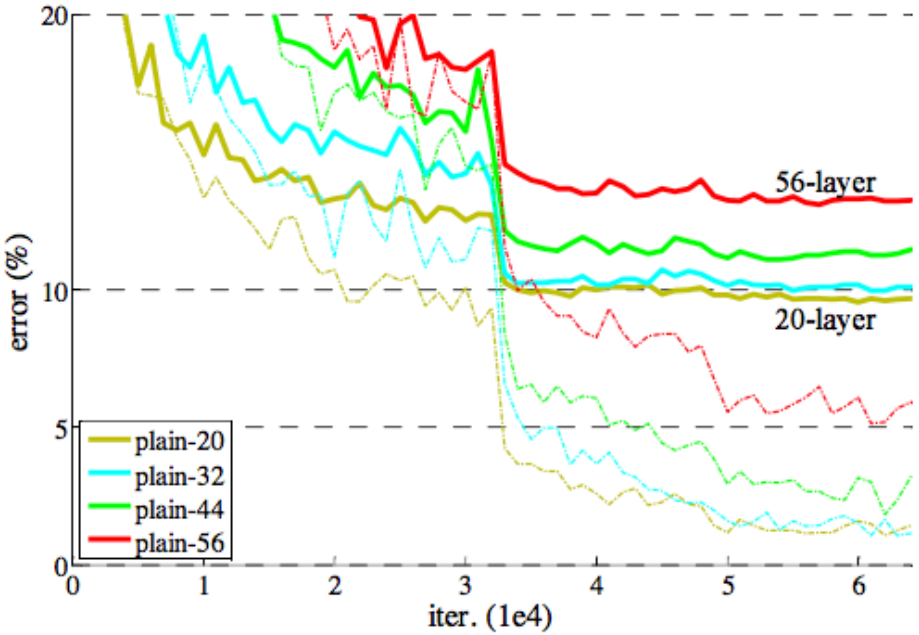
Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.



- Example network architectures for ImageNet.
- Left: the VGG-19 model [41] (19.6 billion FLOPs) as a reference.
- Middle: a plain network with 34 parameter layers (3.6 billion FLOPs).
- Right: a residual network with 34 parameter layers (3.6 billion FLOPs).

- The dotted shortcuts increase dimensions. Table 1 shows more details and other variants.

- The group tried a 1202-layer network, but got a lower test accuracy, presumably due to overfitting.
- Trained on an 8 GPU machine for **two to three weeks**.
-



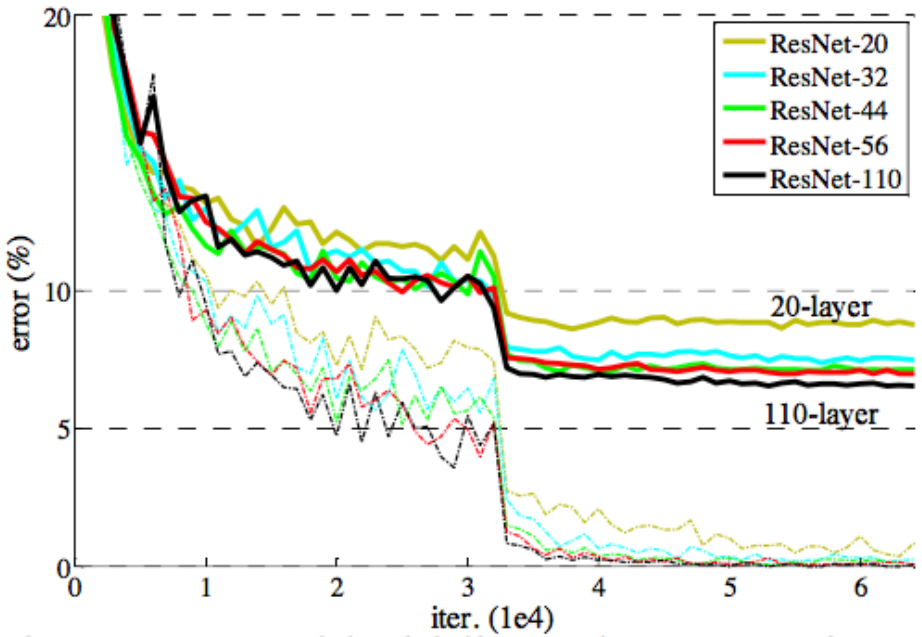
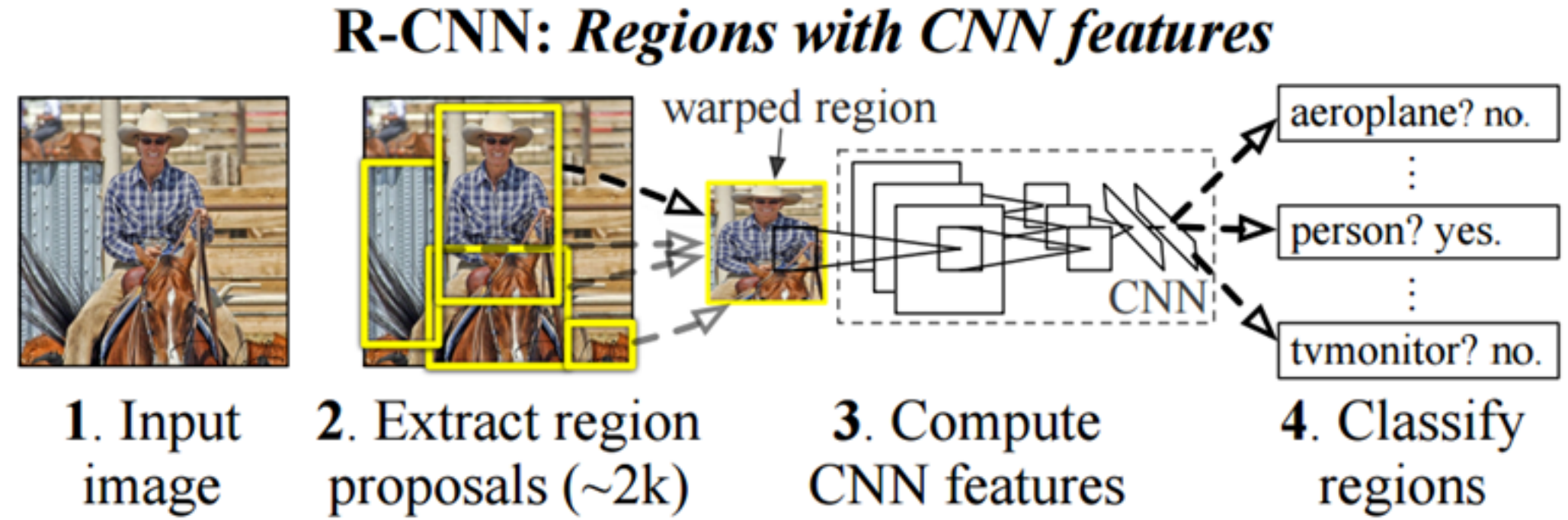


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing er
of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1

Region Based CNNs (R-CNN - 2013, Fast R-CNN - 2015, Faster R-CNN - 2015)

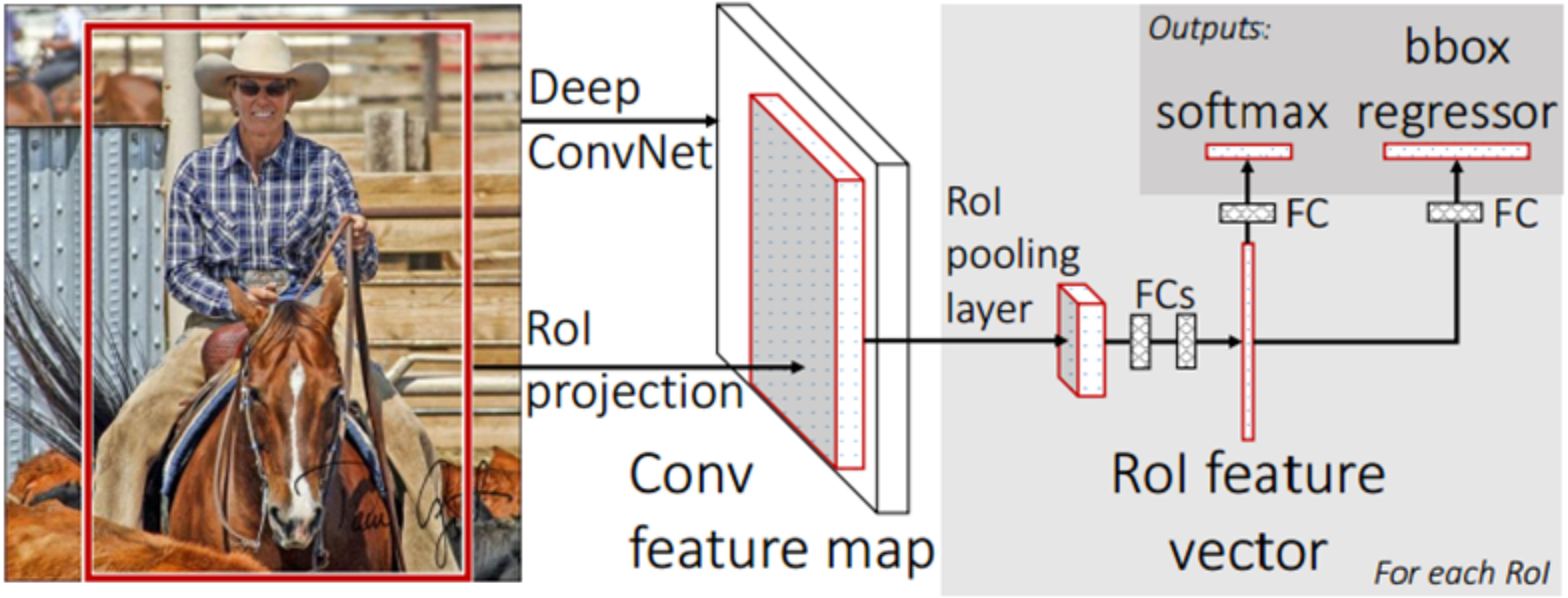
- Ross Girshick and his group at UC Berkeley proposed R-CNN.
- The purpose of R-CNNs is to solve the problem of object detection.
- Given a certain image, we want to be able to draw bounding boxes over all of the objects.
- The process can be split into two general components, the region proposal step and the classification step.

R-CNN



R-CNN workflow

Fast R-CNN



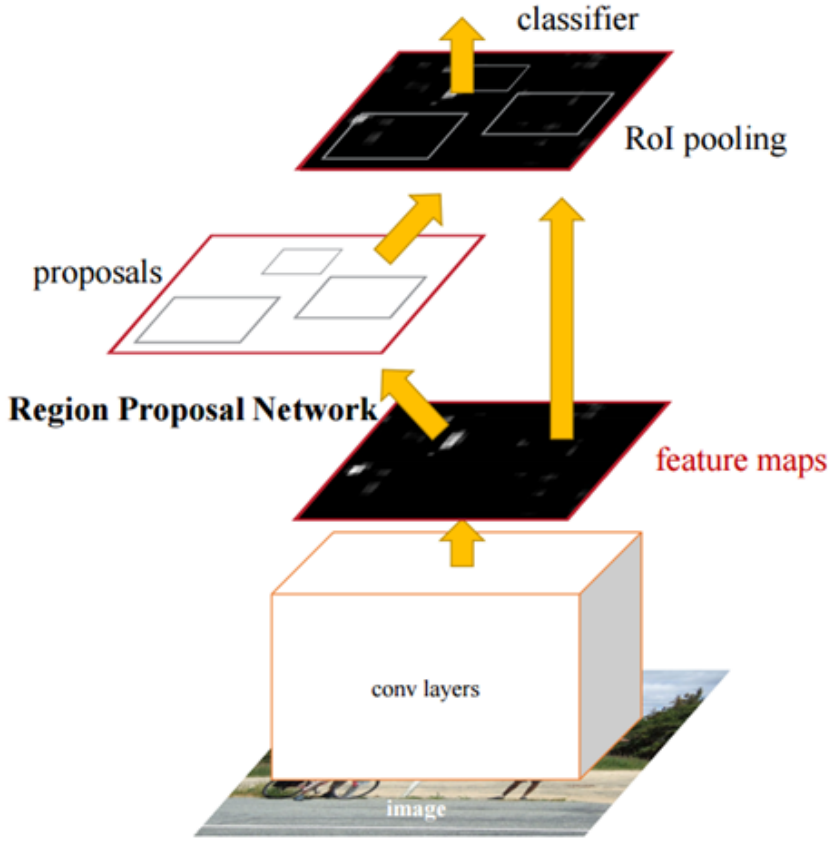
Fast R-CNN workflow

Fast R-CNN

- Improvements were made to the original model because of 3 main problems. Training took multiple stages (ConvNets to SVMs to bounding box regressors), was computationally expensive, and was extremely slow (RCNN took 53 seconds per image).
- Fast R-CNN was able to solve the problem of speed by basically sharing computation of the conv layers between different proposals and swapping the order of generating region proposals and running the CNN.
- In this model, the image is *first* fed through a ConvNet, features of the region proposals are obtained from the last feature map of the ConvNet (check section 2.1 of the paper for more details), and lastly we have our fully connected layers as well as our regression and classification heads.

Faster R-CNN

- insert a region proposal network (RPN) after the last convolutional layer.
- just look at the last convolutional feature map and produce region proposals from that.
- The same pipeline as R-CNN is used (ROI pooling, FC, and then classification and regression heads).



Faster R-CNN workflow

Generating Image Descriptions (2014)

- Andrej Karpathy and Fei-Fei Li
- combination of CNNs and bidirectional RNNs (Recurrent Neural Networks)

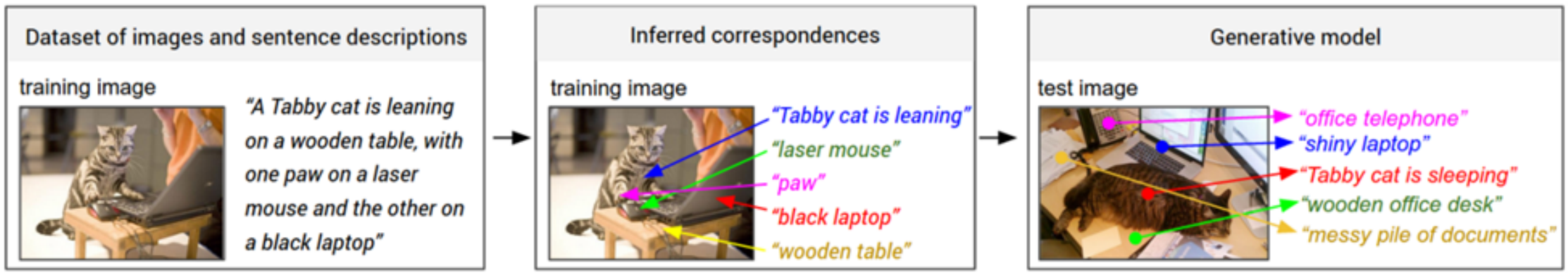
- To generate natural language description

Example output of the model

- Using this training data, a deep neural network “infers the latent alignment between segments of the sentences and the region that they describe”

- Alignment Model:

- Generation Model:



Workflow of alignment and generative model

Spatial Transformer Networks 2015

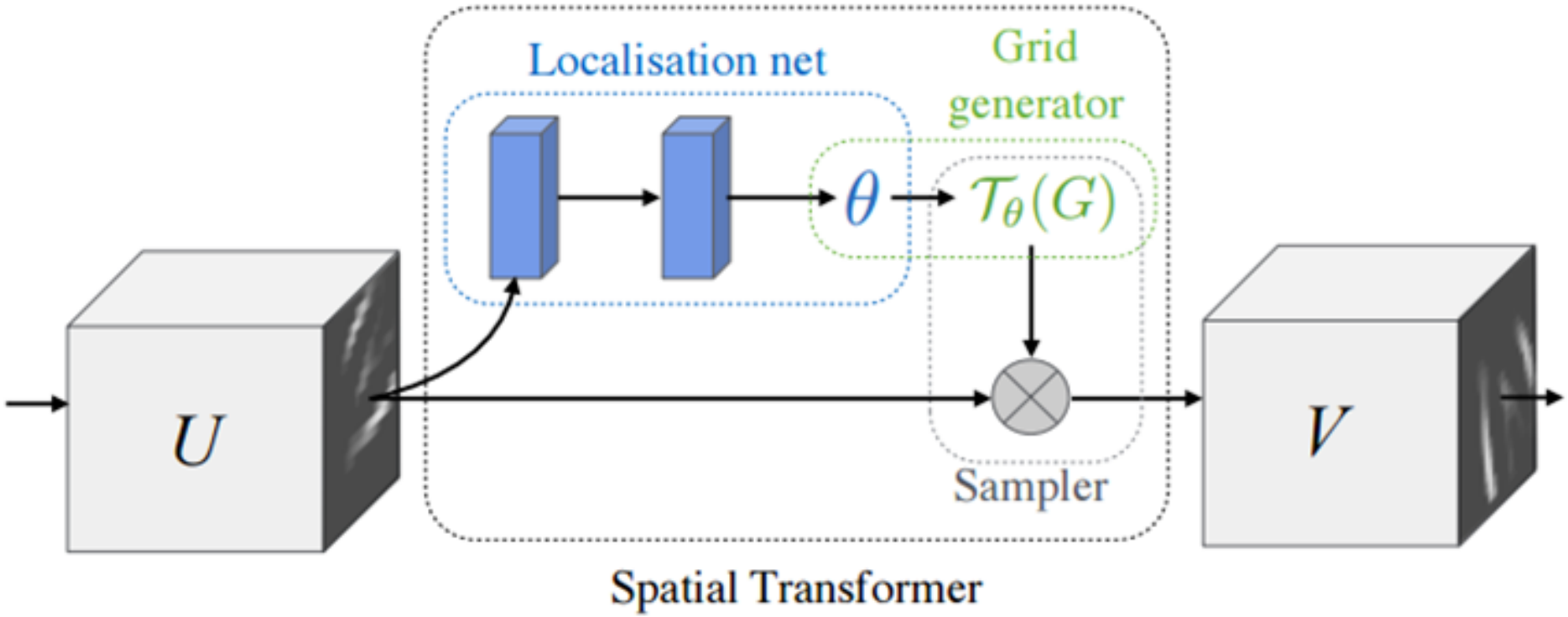
- Google Deepmind 2015
- A module poses 1. normalization (scenarios where the object is tilted or scaled) and 2. spatial attention (bringing attention to the correct object in a crowded image).
- For traditional CNNs, if you wanted to make your model invariant to images with different scales and rotations, you'd need a lot of training examples for the model to learn properly.

- Making affine transformations to the input image in order to help models become more invariant to translation, scale, and rotation.

Transformation module consists of:

- A localization network which takes in the input volume and outputs parameters of the spatial transformation that should be applied. The parameters, or theta, can be 6 dimensional for an affine transformation.
- A sampling grid that is the result of warping the regular grid with the affine transformation (theta) created in the localization network.
- A sampler whose purpose is to perform a warping of the input feature map.

The Transformer Module



A Spatial Tranfomer module

Transformer applies to MNIST

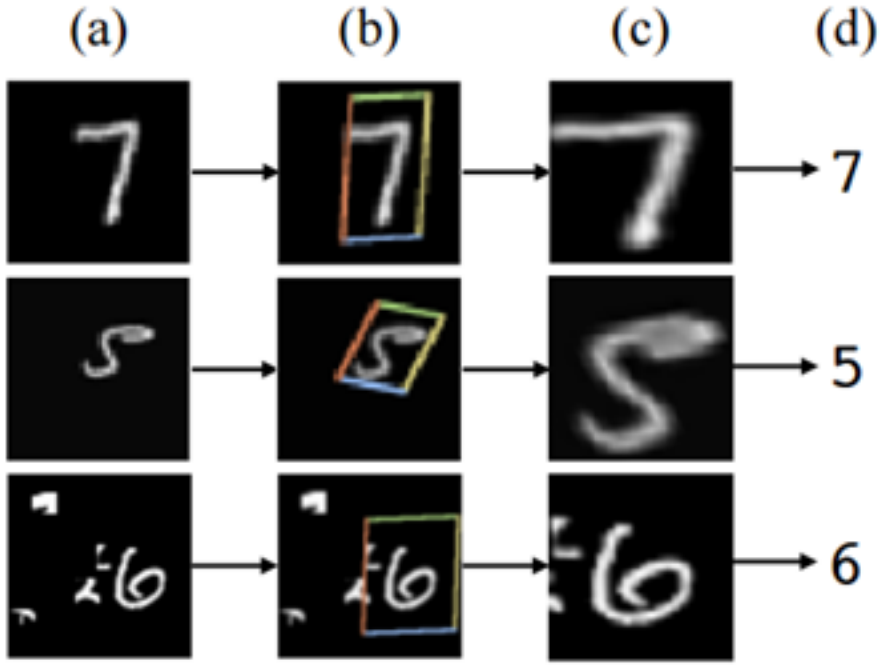


Figure 1: The result of using a spatial transformer as the first layer of a fully-connected network trained for distorted MNIST digit classification. (a) The input to the spatial transformer network is an image of an MNIST digit that is distorted with random translation, scale, rotation, and clutter. (b) The localisation network of the spatial transformer predicts a transformation to apply to the input image. (c) The output of the spatial transformer, after applying the transformation. (d) The classification prediction produced by the subsequent fully-connected network on the output of the spatial transformer. The spatial transformer network (a CNN including a spatial transformer module) is trained end-to-end with only class labels – no knowledge of the groundtruth transformations is given to the system.

Great overview of the function of a Spatial Tranfomer module