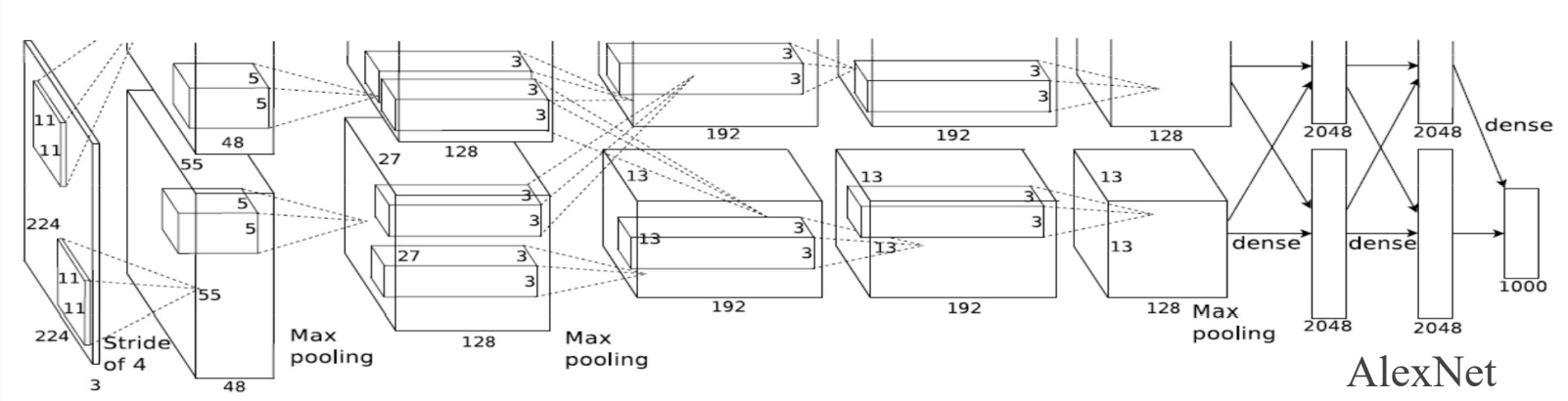
Convolutional Neural Networks for Image Understanding



AlexNet

Krizhevsky et al, 2012

Berrin Yanikoglu

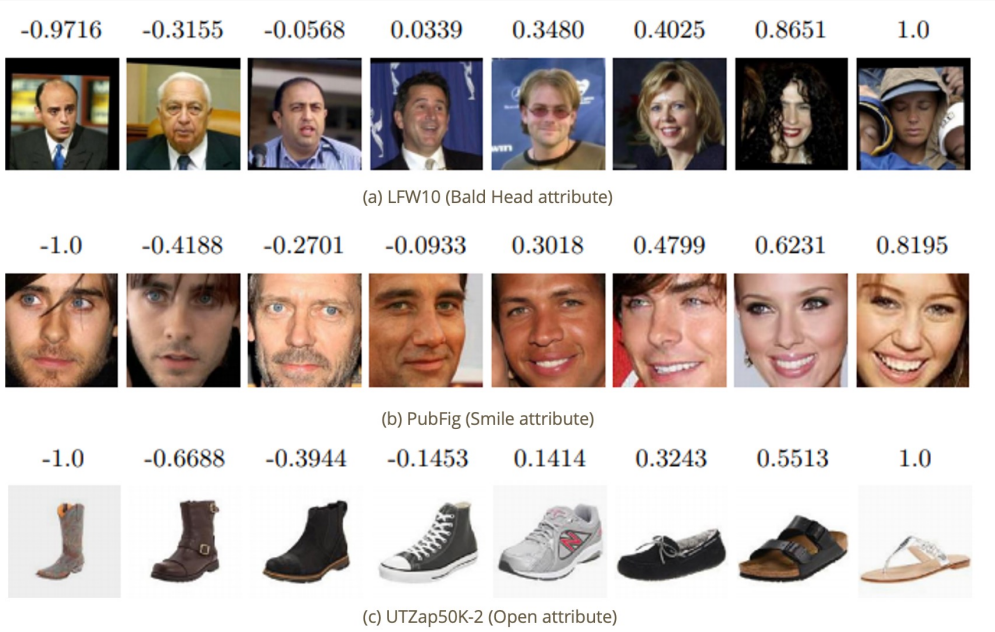
Computer Sci. & Engin., Sabancı University

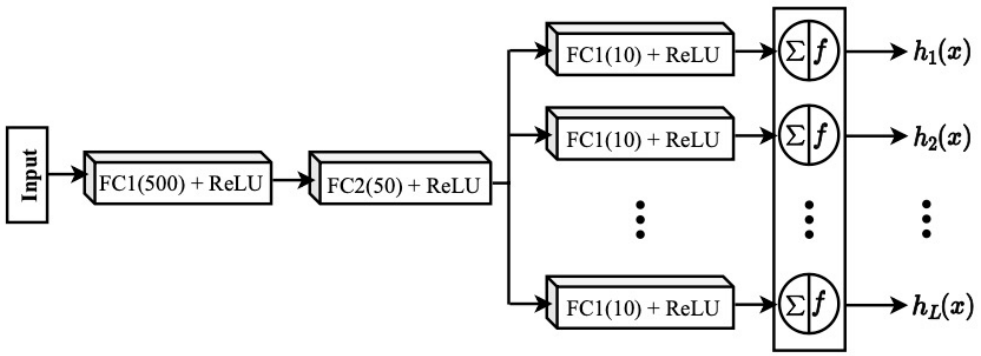
Director, Center of Excellence in Data Analytics (CEDA/VERIM)

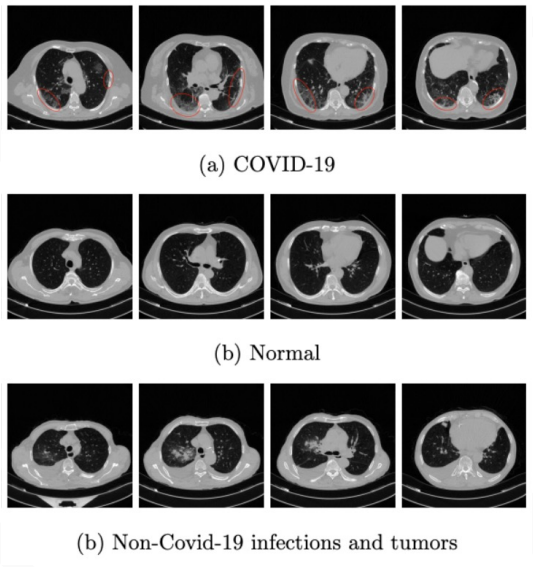
Many figures and content thanks to Stanford U. course http://cs231n.github.io/

Research Areas

- Image and video understanding
  - Attribute based learning
  - Medical image understanding
- Ensemble methods for deep learning
- Semi-supervised and web learning







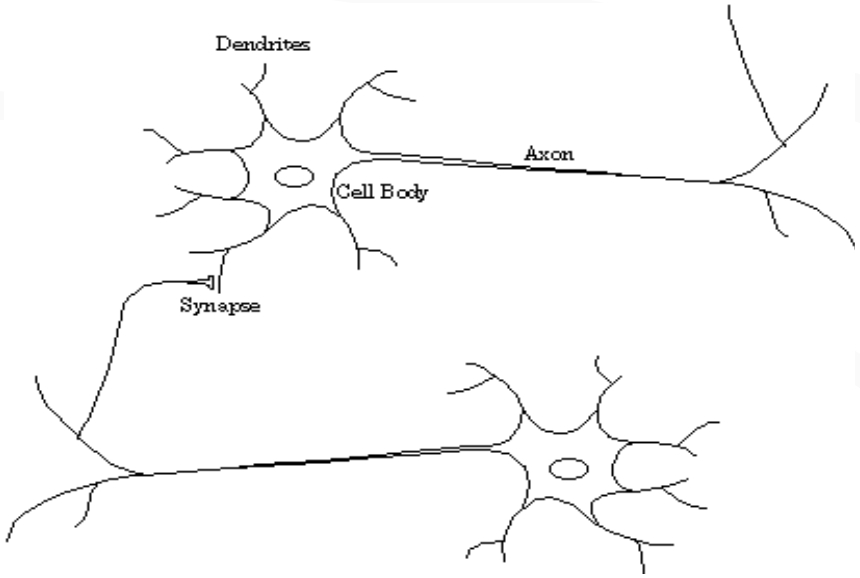
Human Visual System

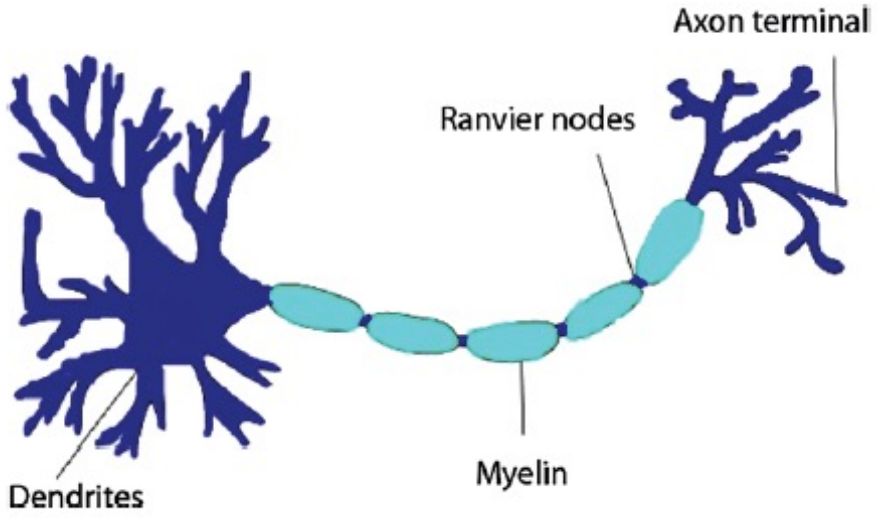
Human Brain

- Humans perform complex tasks such as vision, motor control, or language understanding, very well.
- One way to build intelligent machines is to try to imitate the (organizational principles of) human brain.
- The brain is a highly complex, non-linear, and parallel computer, composed of some 10^{11} neurons (100-120 billion) that are densely connected: $>10^3$ connection per neuron (100-1000 trillion).

Biological Neuron

- dendrites: nerve fibres carrying electrical signals to the cell
- axon: single long fiber that carries the electrical signal from the cell body to other neurons
- synapse: the point of contact between the axon of one cell and the dendrite of another, regulating a chemical connection whose strength affects the input to the cell.
- cell body: computes a non-linear function of its inputs

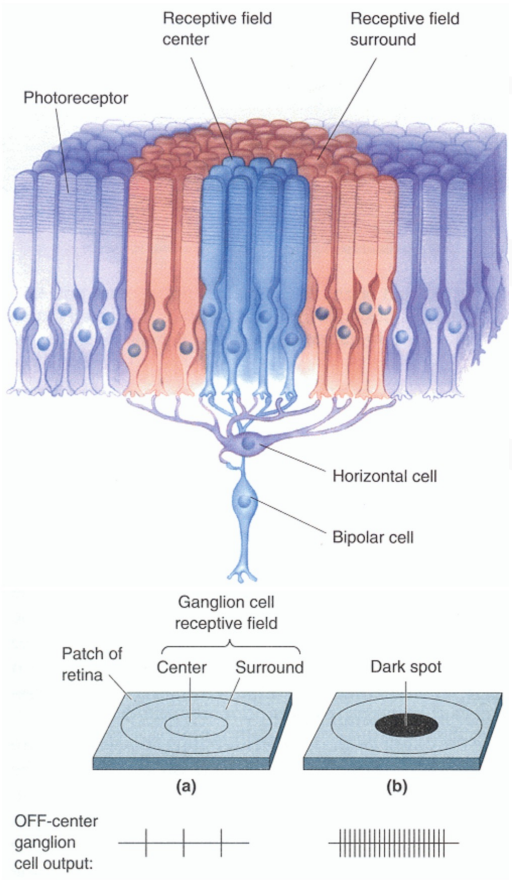


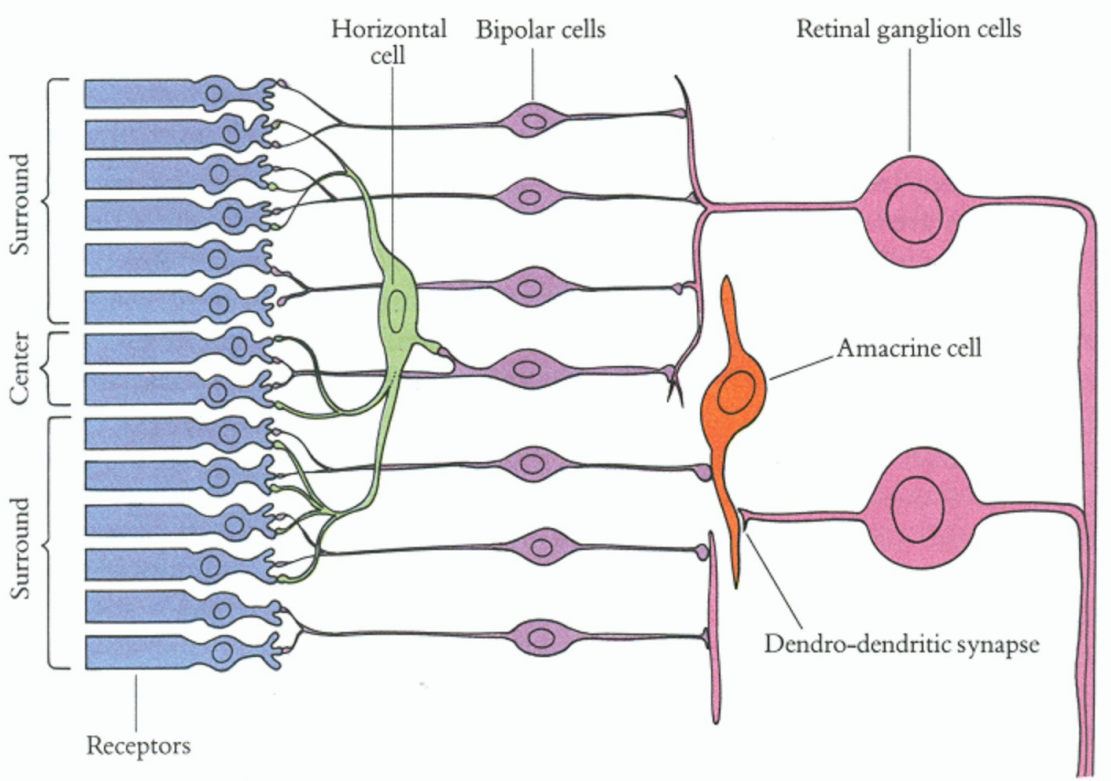


Biological Neuron

A variety of different neurons exist (motor neuron, on-center off-surround visual cells…), with different branching structures.

The connections of the network and the strengths of the individual synapses establish the function of the network.





http://fourier.eng.hmc.edu/e180/lectures/retina/node6.html

Human Visual System

The visual cortex contains a complex arrangement of cells (Hubel & Wiesel 1968).

- Inspiration for artificial neural network systems.
- Information flows from earlier layers to later layers.

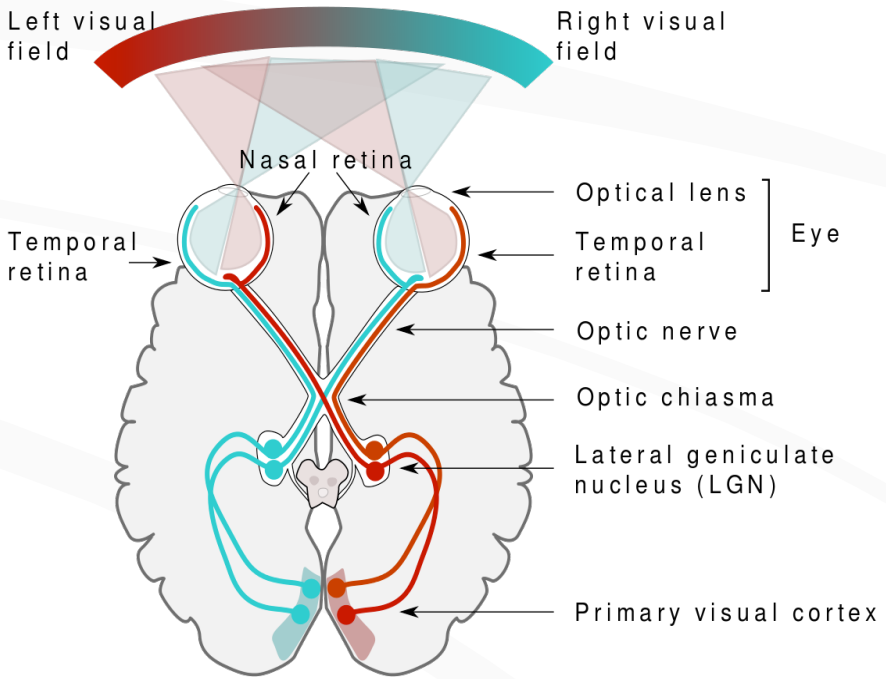
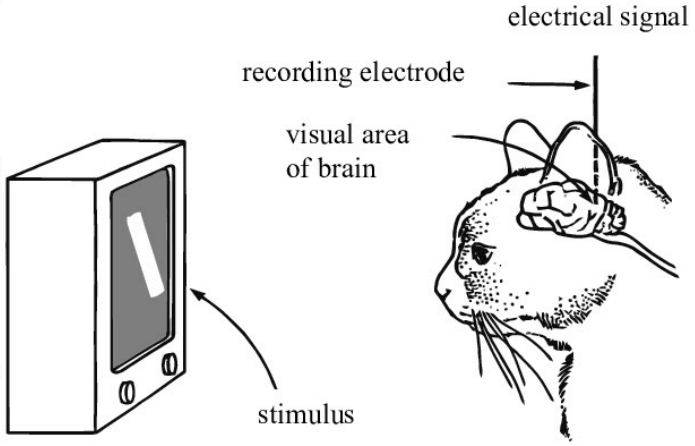
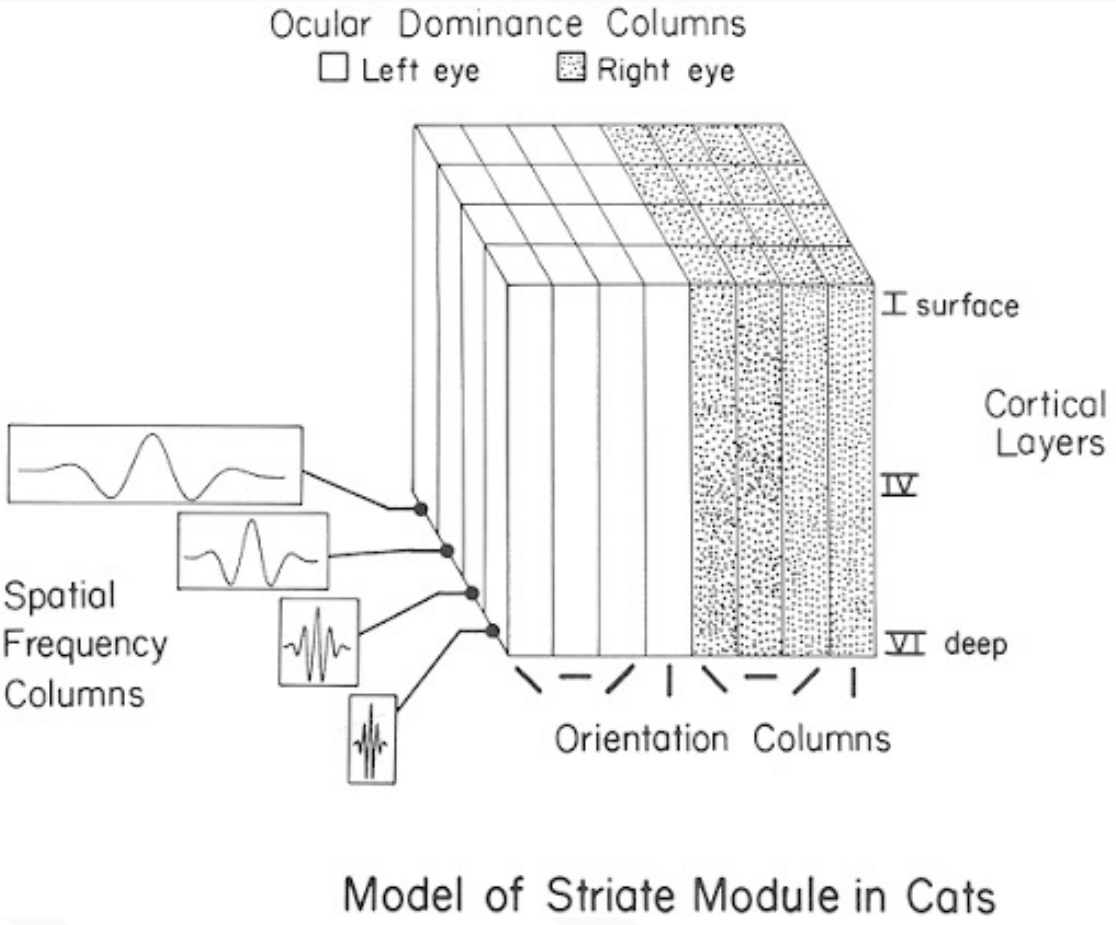


Image credit: Wikipedia.org

Human Visual System





Model of Striate Module in Cats

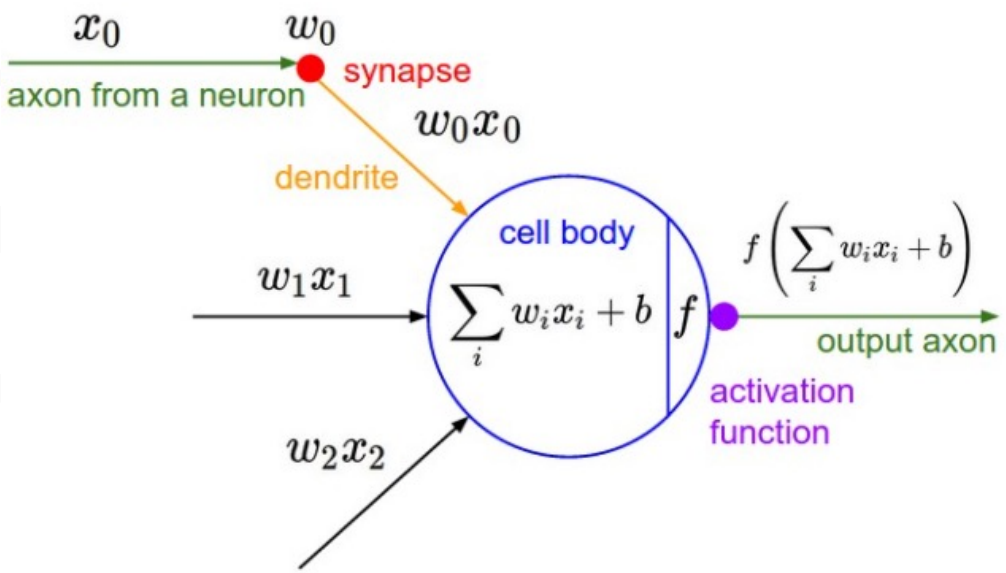
- Cells in earlier/later layers are sensitive to simpler/more complex patterns, respectively.
- Some cells are tuned to certain orientation and frequency.

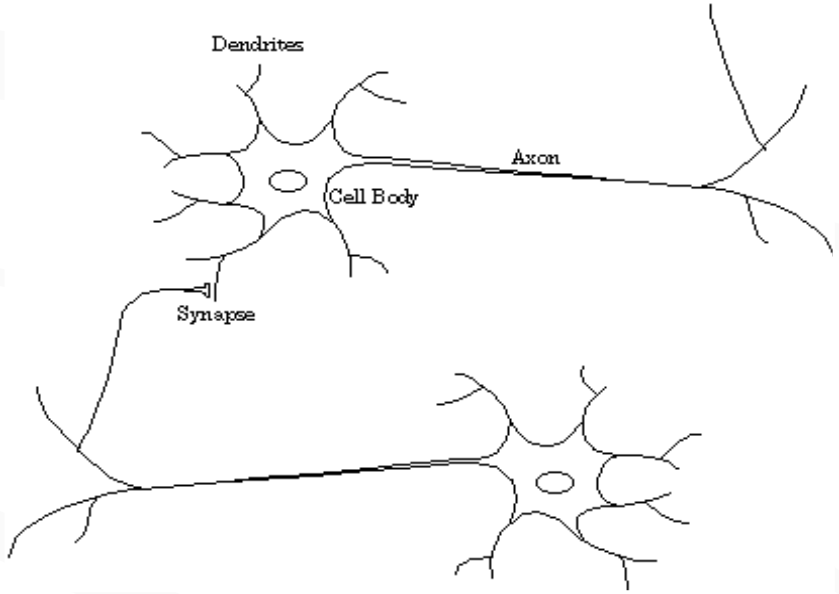
Artificial Neural Networks

Artificial Neuron Model

A single artificial neuron computes its weighted input (net input) and passes it through an activation function to obtain its output.

(typically non-linear)

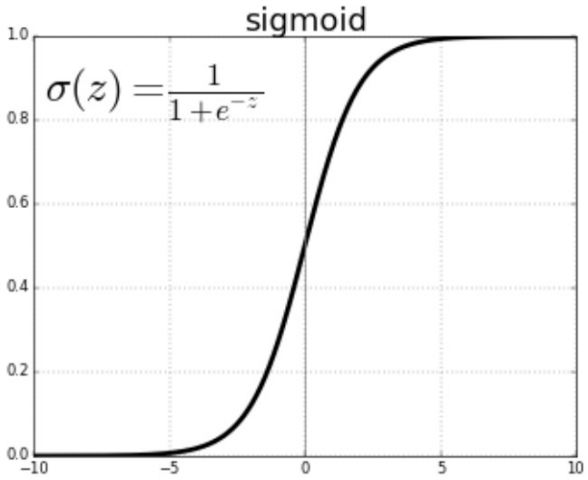


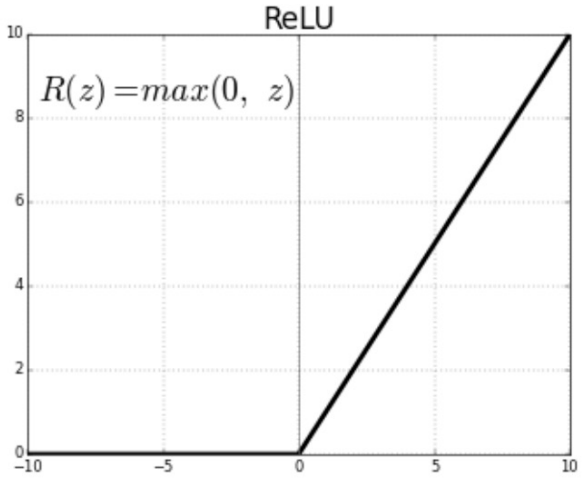


$$net = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$

$$output = f(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$$

Activation Functions





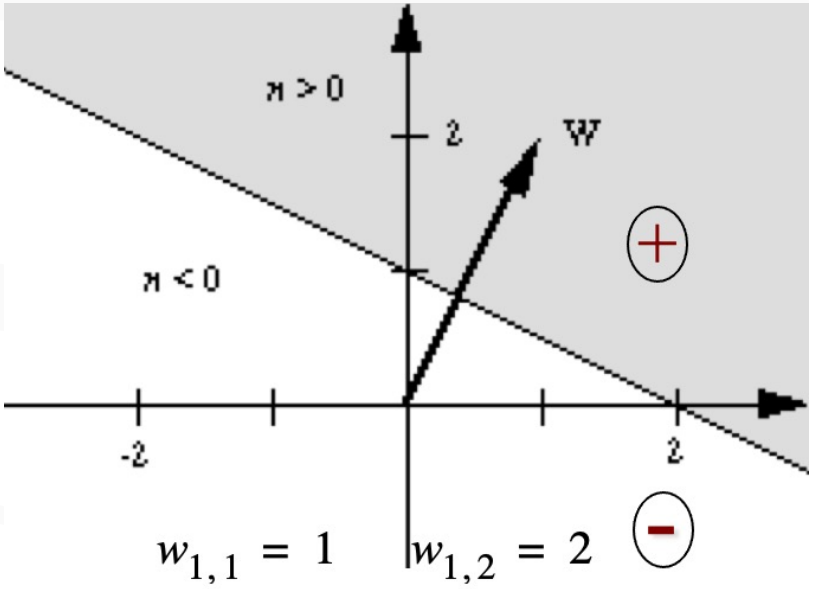
Images: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

What can a single neuron do?

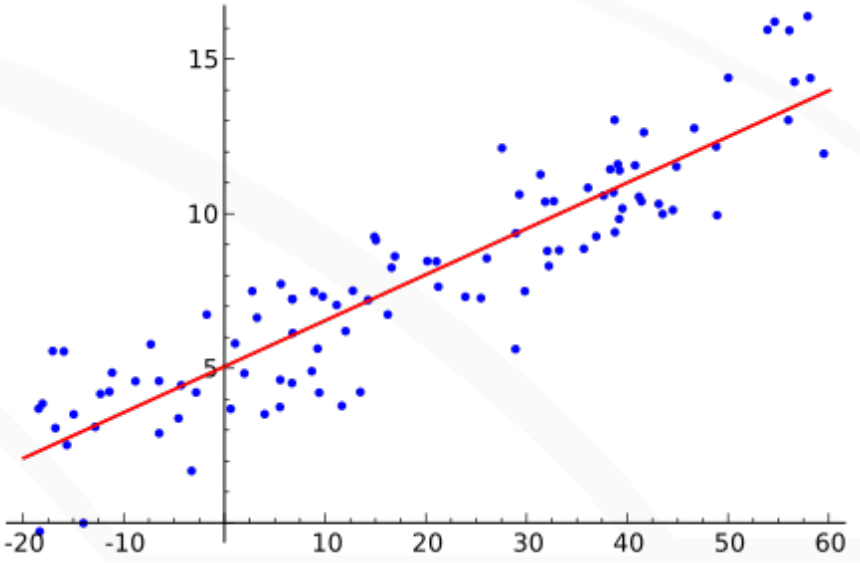
- With a hardlimiting (threshold) activation function,
  - It effectively separates the input space into two by the hyperplane:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

where **x** is the input, **w** is the weight vector, and b is the bias.



- With a linear activation function,
  - It can learn a linear regression model.

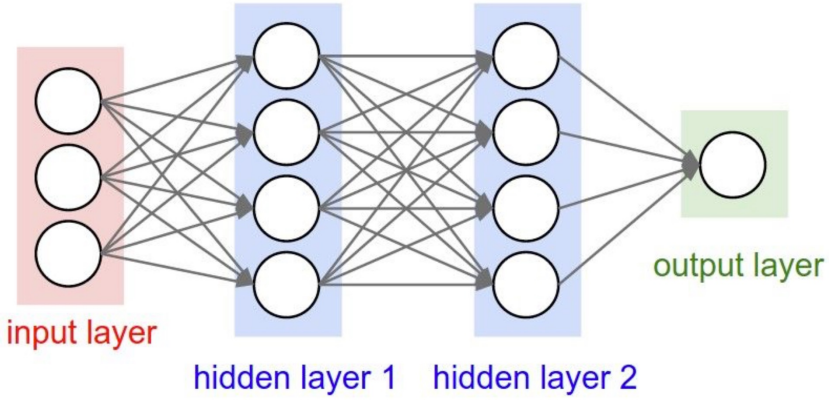


Multi-layer Perceptron

Feed-forward Neural Networks, …

Multilayer Perceptron

- In Multilayer perceptrons, there may be one or more hidden layer(s) which are called hidden since they are not observed from the outside.
- Activations are passed only from one layer to the next.




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Motivation for Convolutional Layers

1-hidden layer feed forward neural network for digit recognition

>> More general problem of object detection and recognition with Convolutional NNs

(shown on board)

Backpropagation Algorithm

Gradient Descent / Steepest Descent

- Starting from random initial weights,
- Backpropagation algorithm is used to learn the network weights.
- Successive adjustments to the weights (w) are in the direction of the steepest descent (direction opposite to the gradient of the error function)

$$w(n+1) = w(n) - \eta \nabla E(w(n))$$

$$\nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1} E(\mathbf{w}) \\ \frac{\partial}{\partial w_2} E(\mathbf{w}) \\ \vdots \\ \frac{\partial}{\partial w_n} E(\mathbf{w}) \end{bmatrix}$$

Convolutional Neural Networks

Receptive Fields & Convolution Operation

Convolutional Neural Networks (CNNs)

- Cells in a layer take input from small sub-regions of the visual field (receptive field).
- Nearby neurons are connected to nearby regions and respond to spatially local input patterns.
- Receptive fields are tiled to cover the entire visual field.
- Stacking many such layers leads to (non-linear) “filters” that become increasingly “global” (i.e. responsive to a larger region of pixel space).

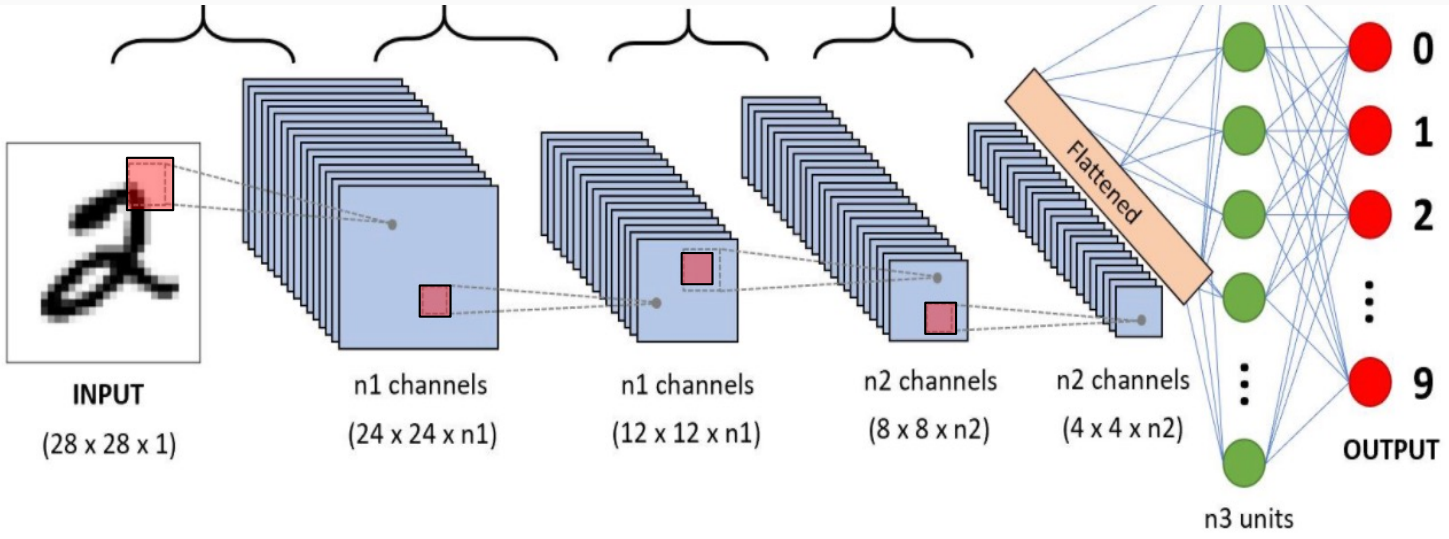
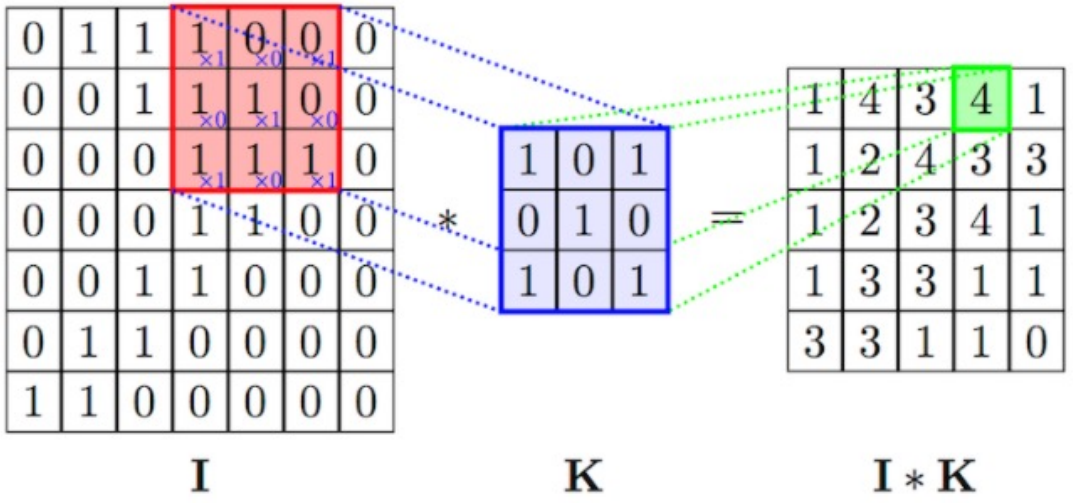


Image credit: https://guandi1995.github.io/Classical-CNN-architecture/

Convolution

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$



Output of this neuron is 4

- Dot product between pixels in the receptive field (subregion of I) and the kernel (K).

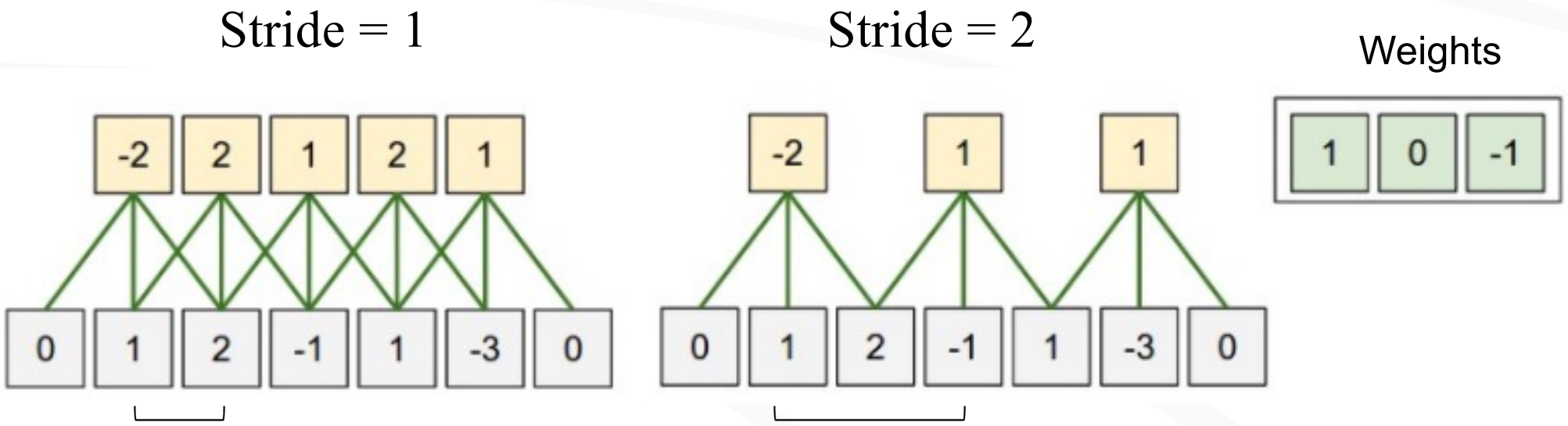
Convolutional Neural Networks

Receptive Field Size, Stride, Padding

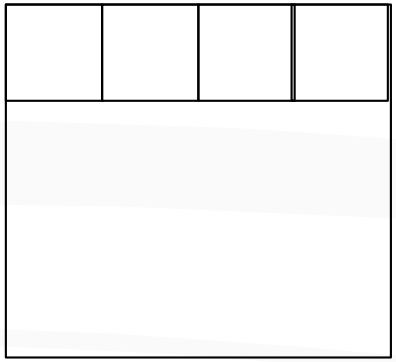
Output Volume, # of Layers

Stride and Zero-Padding

- Stride: How many pixels the filters are shifted spatially. Typically 1 or 2.
- Zero-padding: How many zero pixels are padded to the image to have a good/matching size of output filters.



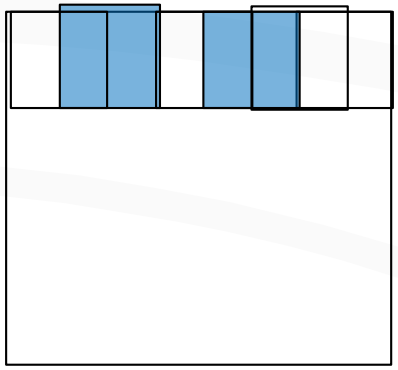
Number of Nodes for a Given Padding and Stride



Assume image width is 20, padding is 0, receptive field size is 5x5 and stride is 5.

W=0, P=0, F=5, S=5

(W+2P-F)/S+1=(20-5)/5+1 = 4

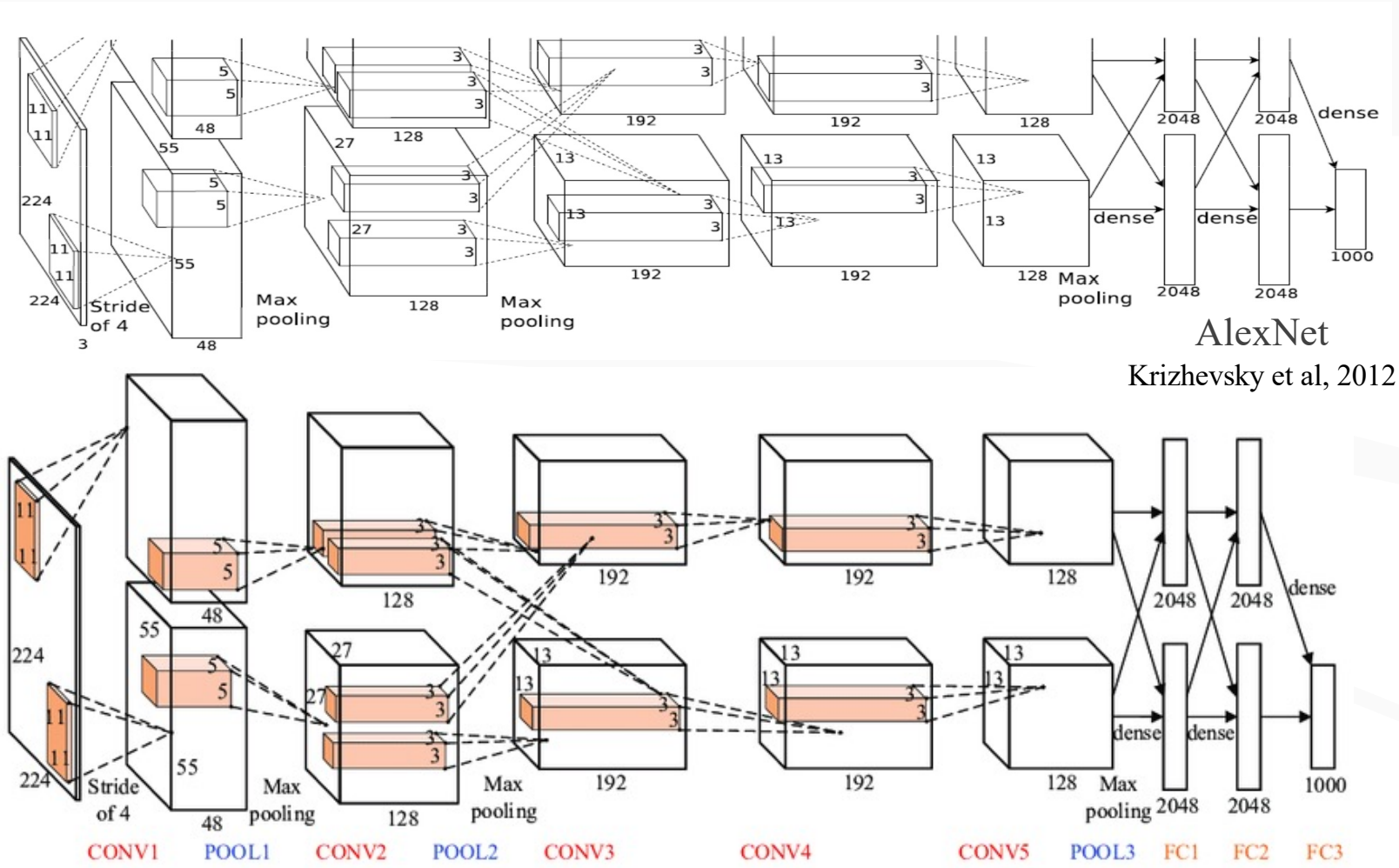


Assume image width is 20, padding is 0, receptive field size is 5x5 and stride is 3.

W=0, P=0, F=5, S=5

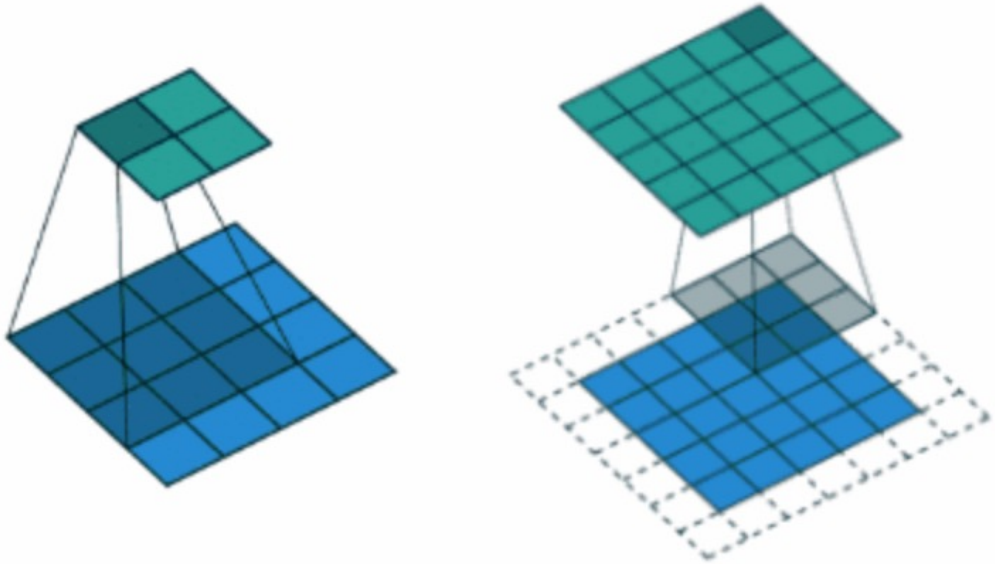
(W-F)/S+1=(20-5)/3+1 = 6

Size of the Output Volume



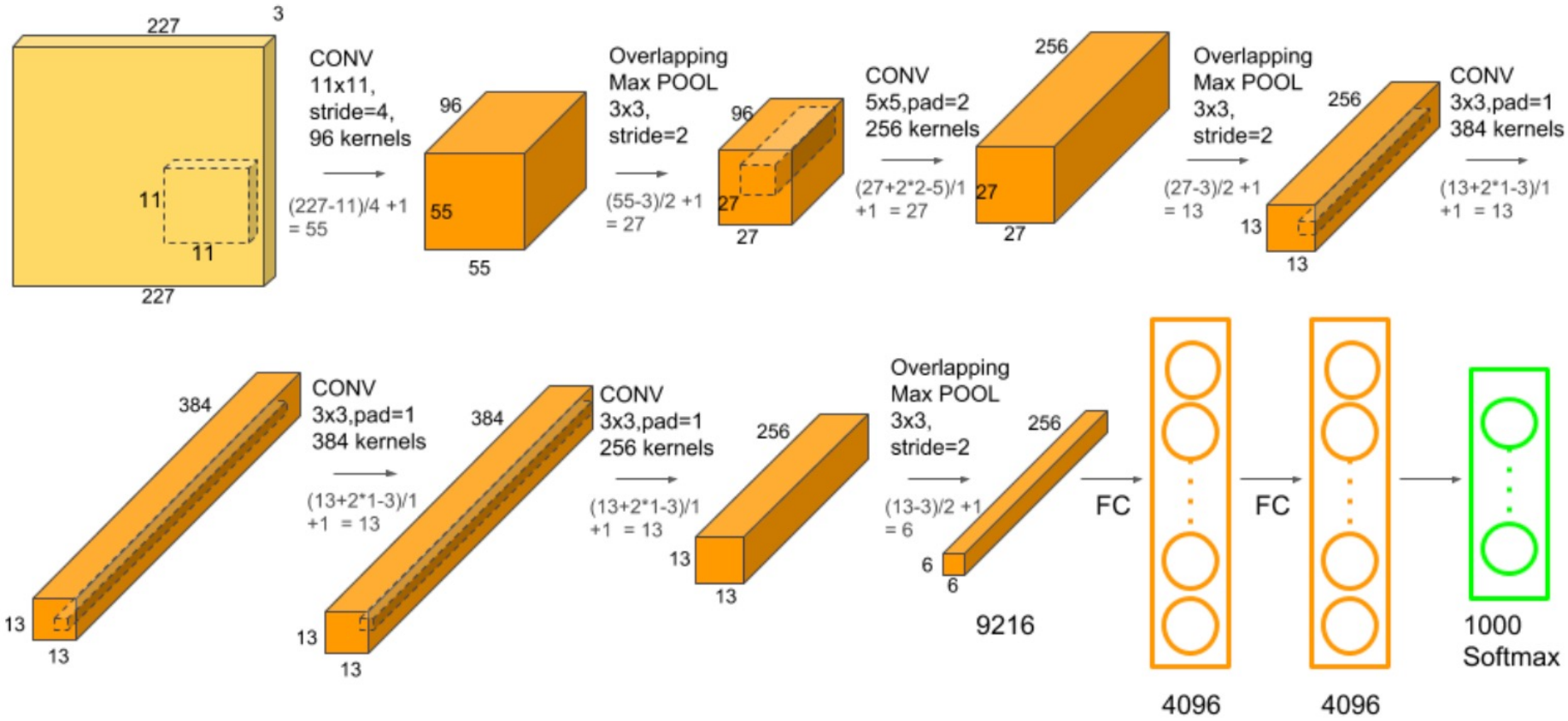
- On the first Convolutional Layer, AlexNet uses neurons with receptive field size F=11, stride S=4 and no zero padding.
  - **(W+2P-F)/S+1** = (227+0-11)/4 + 1 = 55 nodes in one dimension.

- VALID and SAME keywords indicate padding in TensorFlow
  - SAME uses enough padding to match keep Output size same as Input size using a stride of 1.
  - VALID uses no padding so as to use only the valid input pixels.



- Using a 3x3 filter and zero padding and stride of 1, a 4x4 input volume maps to a 2x2 output volume.
- Using a 3x3 filter and padding of 1 and stride of 1, a 5x5 input volume maps to the same size (a 5x5 output volume).

Another Look at AlexNet

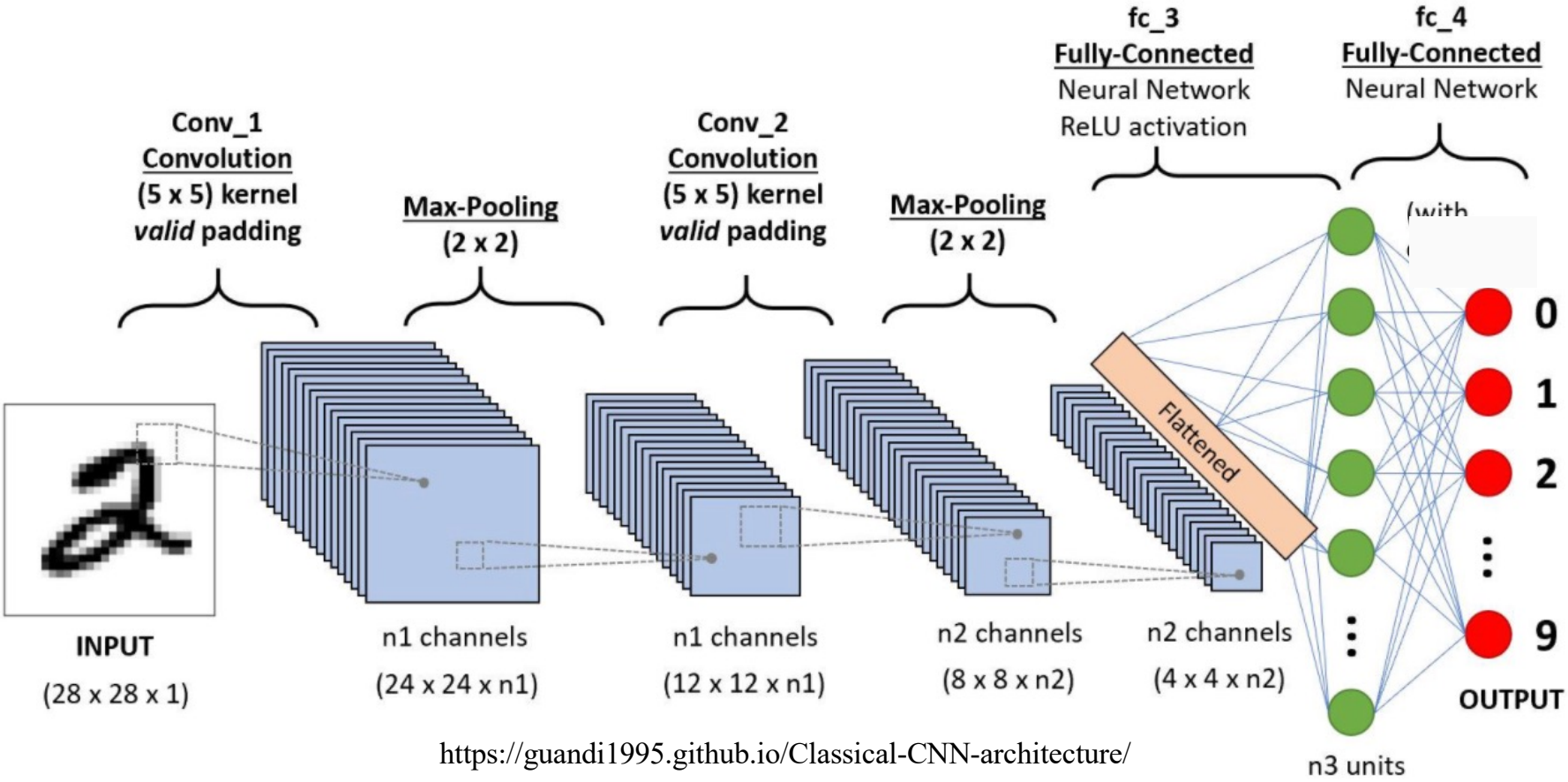


(W+2P-F)/S+1

Image credit: https://learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

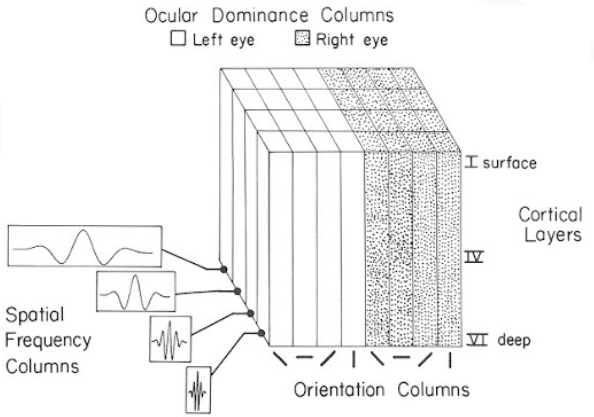
Convolutional Neural Networks

Depth, Feature Maps, Max pool, Fully connected layer

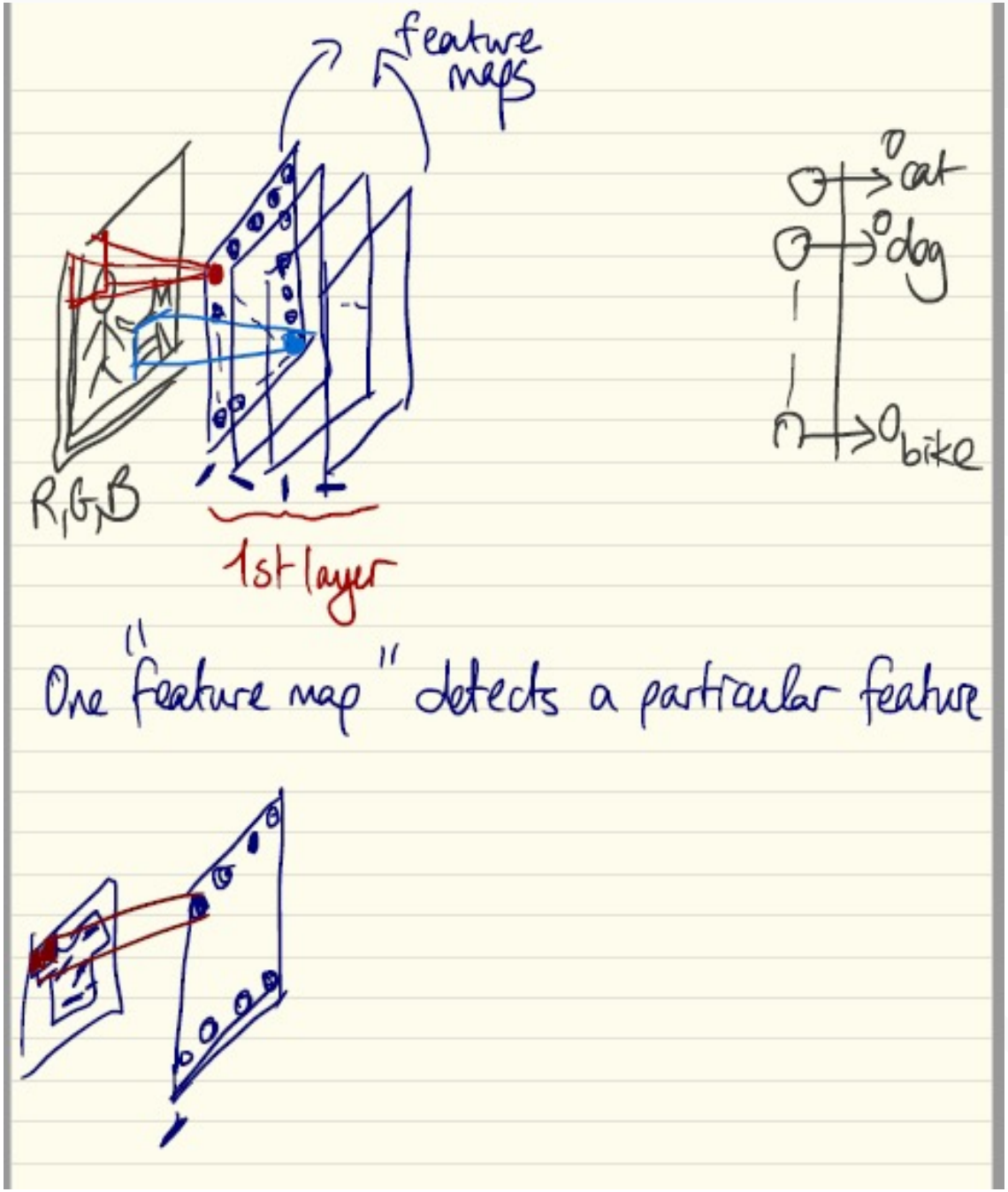


https://guandi1995.github.io/Classical-CNN-architecture/

- In one layer, there are multiple feature maps, to detect different patterns:
  - One can detect vertical edges, another horizontal,…
  - n1 feature maps (channels) on layer 1 above



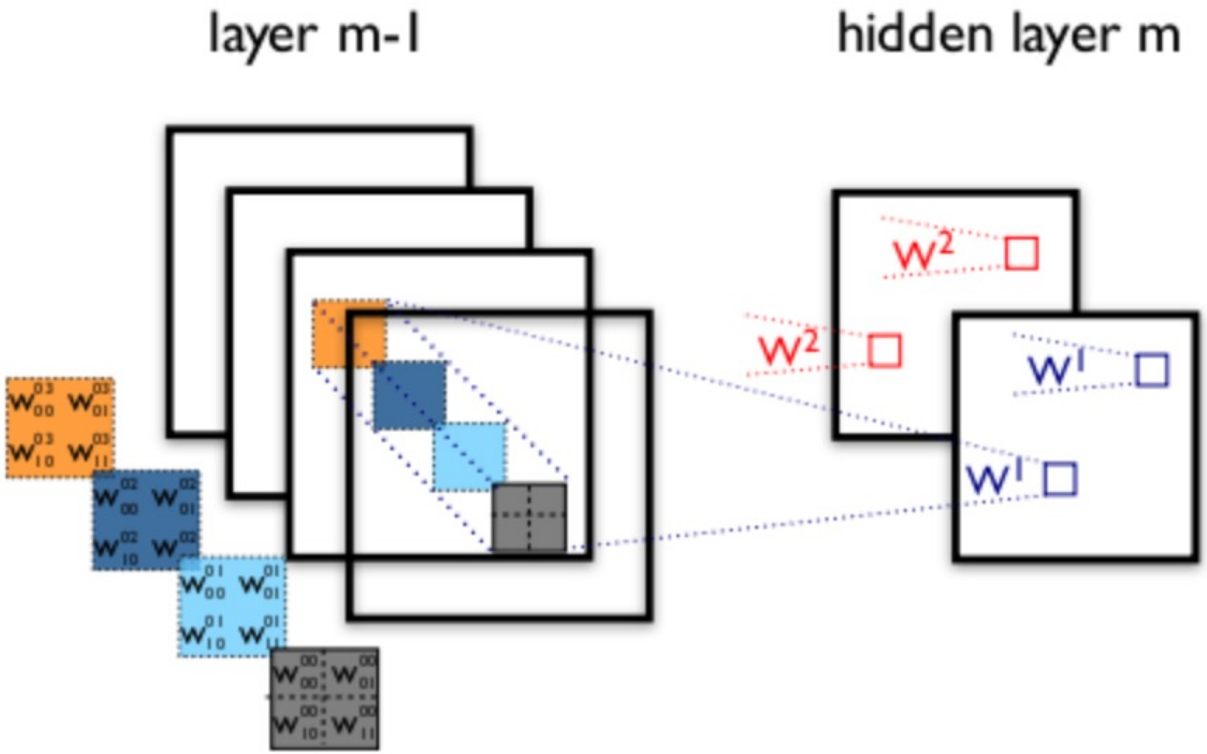
Model of Striate Module in Cats

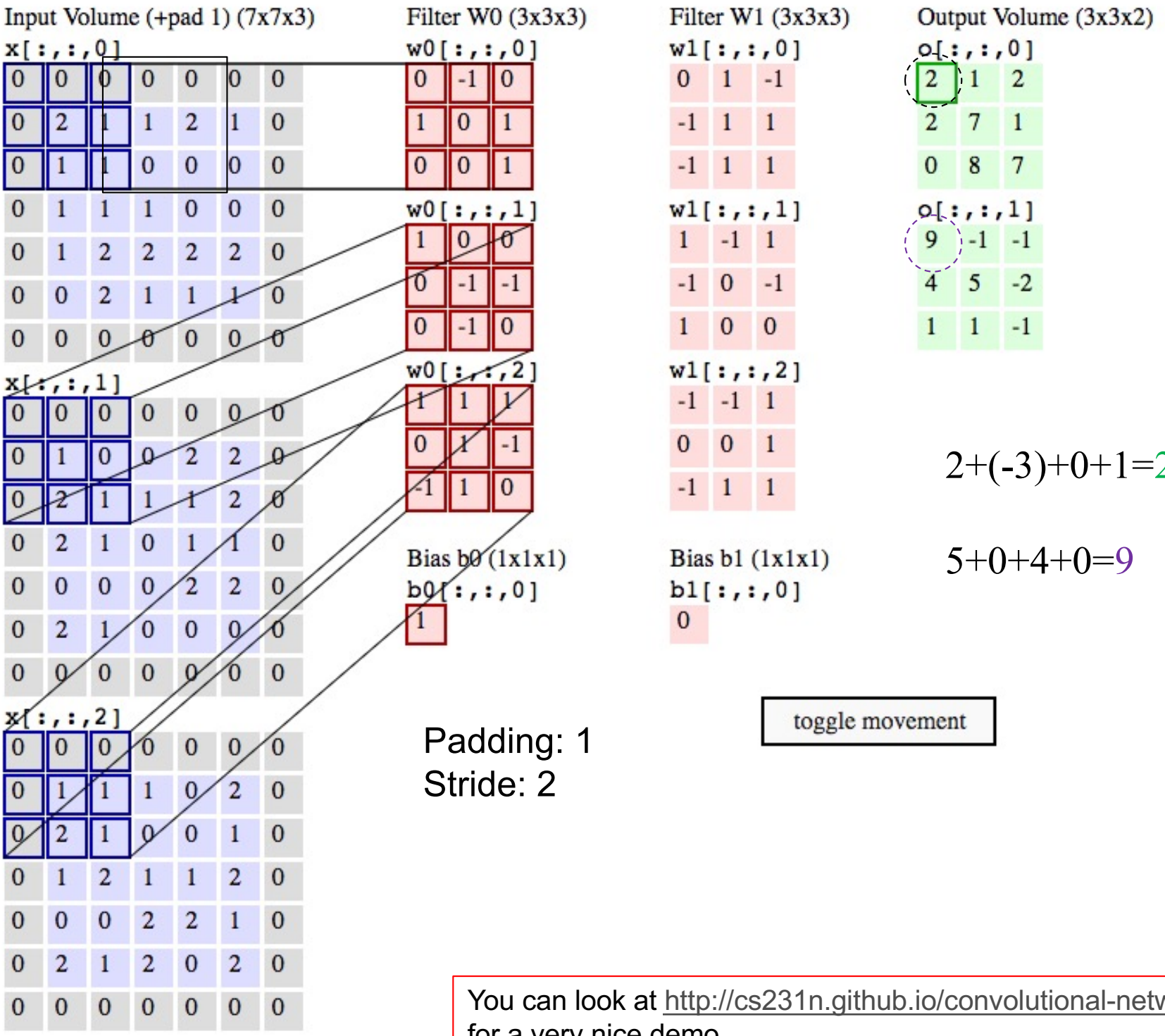


- Each filter is replicated across the entire visual field.
- These replicated units share the same weights and form a feature map.
- Replicating units in this way allows for features to be detected regardless of their position in the visual field.
- Additionally, weight sharing greatly reduces the number of free parameters being learnt.

Feature Maps and Receptive Fields

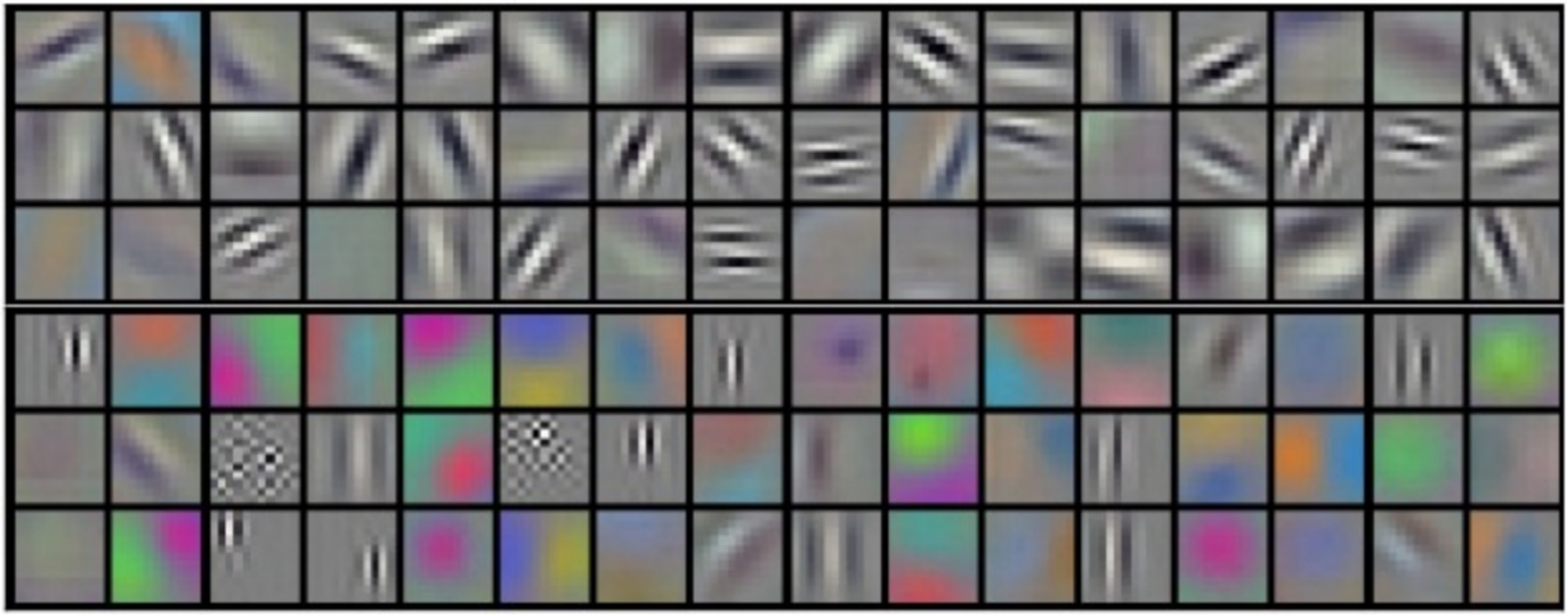
- There are 4 and 2 feature maps in layers m-1 and m, respectively.
- The receptive field of nodes in layer m spans all four input feature maps.
- The weights are 3D weight tensors.
    - The leading dimension indexes the input feature maps, while the other two refer to the pixel coordinates.



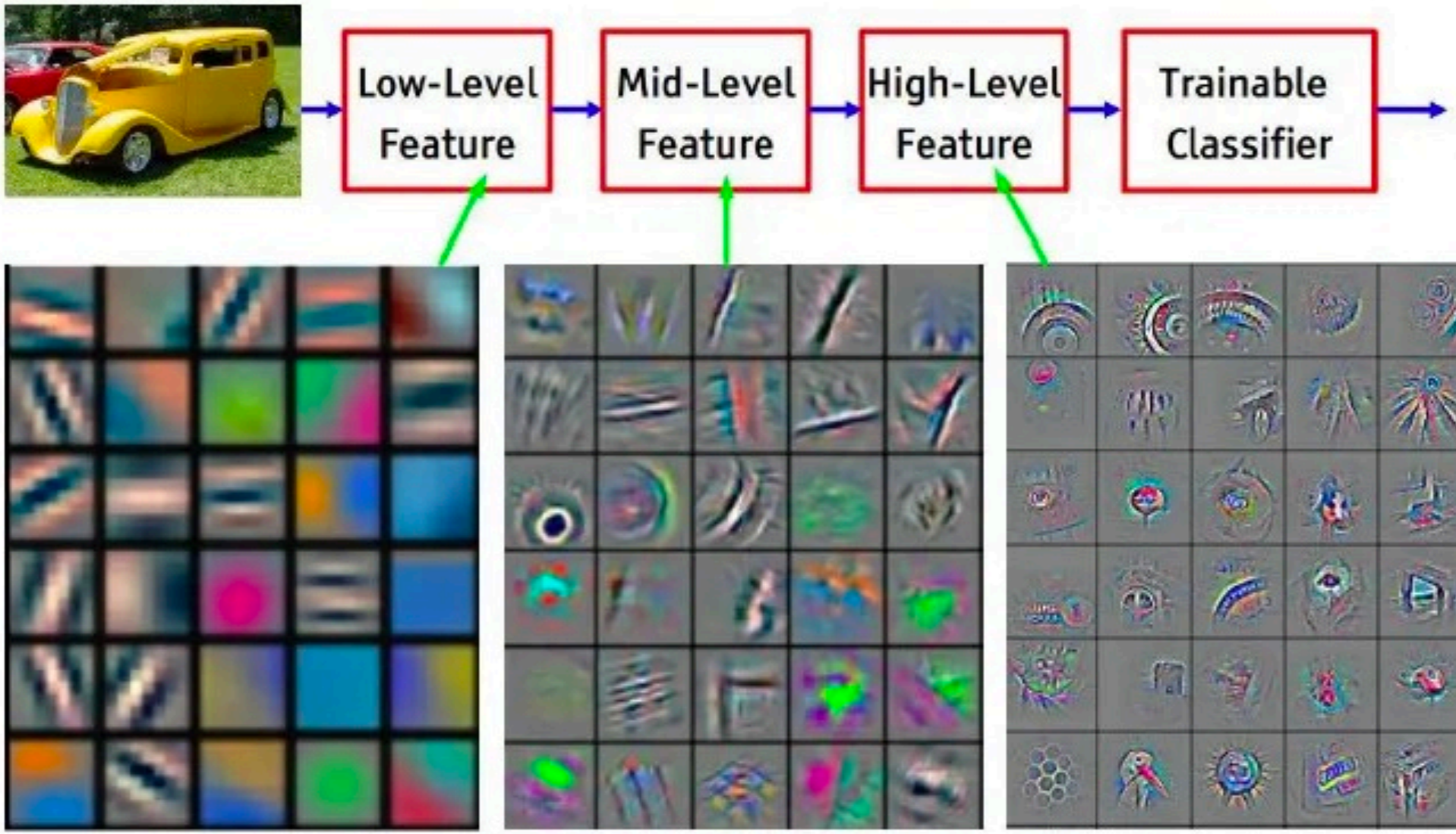

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 2 1 0
0 1 1 0 0 0 0
0 1 1 1 0 0 0
0 1 2 2 2 2 0
0 0 2 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 2 2 0
0 2 1 1 1 2 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 0 0 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 1 0 2 0
0 2 1 0 0 1 0
0 1 2 1 1 2 0
0 0 0 2 2 1 0
0 2 1 2 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 -1 0
1 0 1
0 0 1
w0[:,:,1]
1 0 0
0 -1 -1
0 -1 0
w0[:,:,2]
1 1 1
0 1 -1
-1 1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
0 1 -1
-1 1 1
-1 1 1
w1[:,:,1]
1 -1 1
-1 0 -1
1 0 0
w1[:,:,2]
-1 -1 1
0 0 1
-1 1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 1 2
2 7 1
0 8 7
o[:,:,1]
9 -1 -1
4 5 -2
1 1 -1
2+(-3)+0+1=2
5+0+4+0=9
toggle movement
Padding: 1
Stride: 2
You can look at http://cs231n.github.io/convolutional-networks/
for a very nice demo

Weight Visualization

from AlexNet

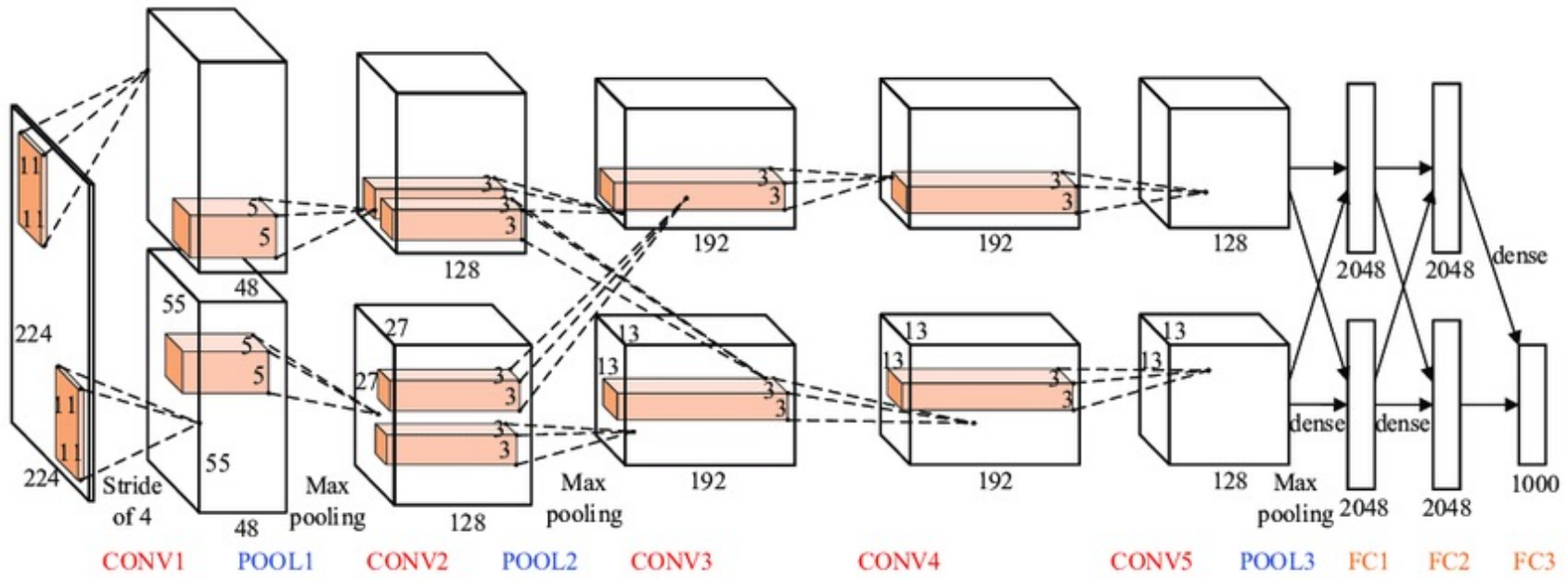


The 96 convolutional kernels of size 11x11x3 learned by the first convolutional layer on the 224x224x3 input images.



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Weight Sharing

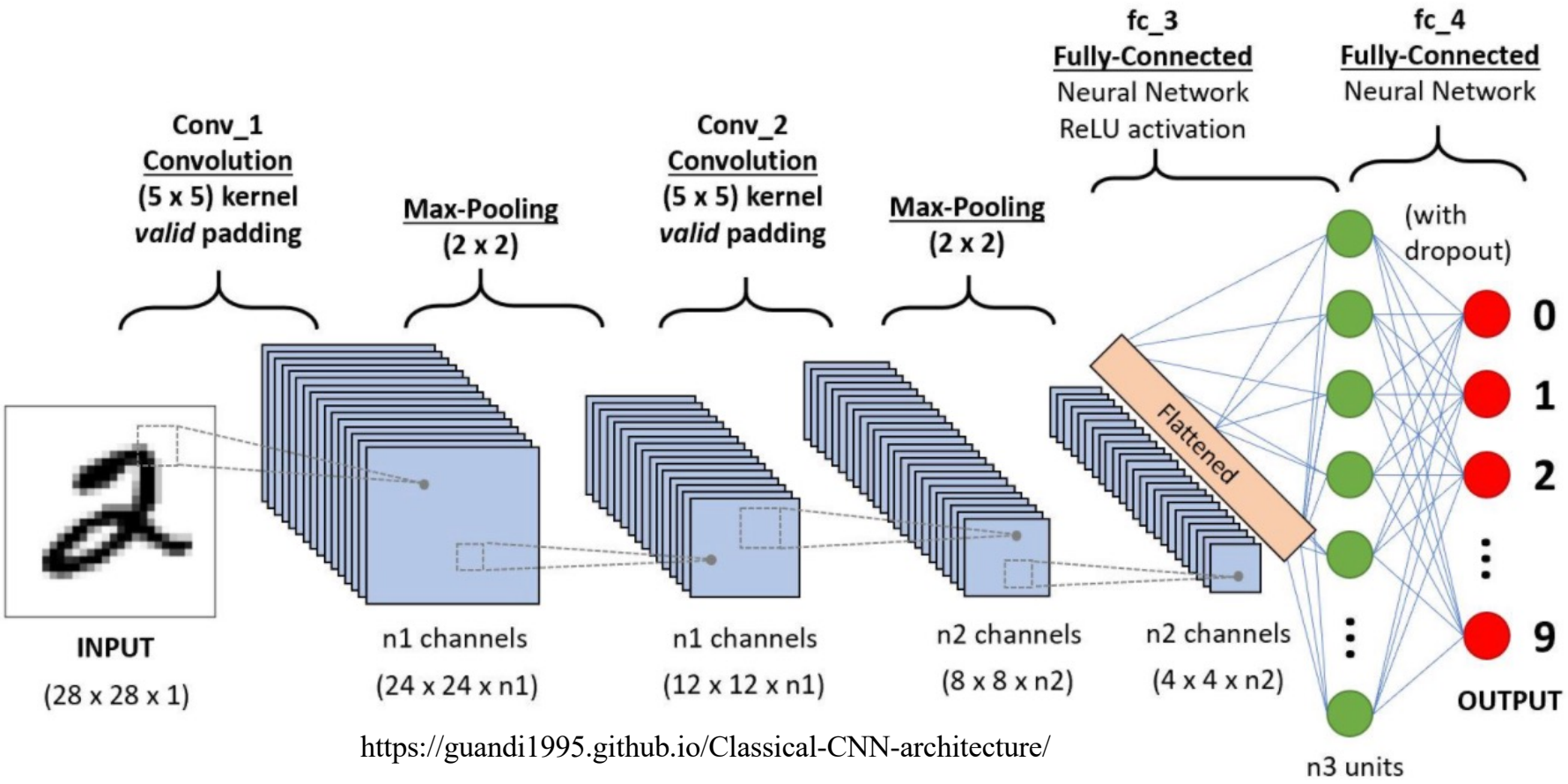


- There are 55*55*96 = 290,400 neurons in the first Conv Layer.
- Each has 11*11*3 = 363 weights and 1 bias.
- Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the ConvNet alone, if the weights were not shared!
- With weight sharing within a feature map, we have 96 unique **set** of weights, for a total of 96 x11x11x3 = 34,944 (+96 biases).

Convolutional Neural Networks

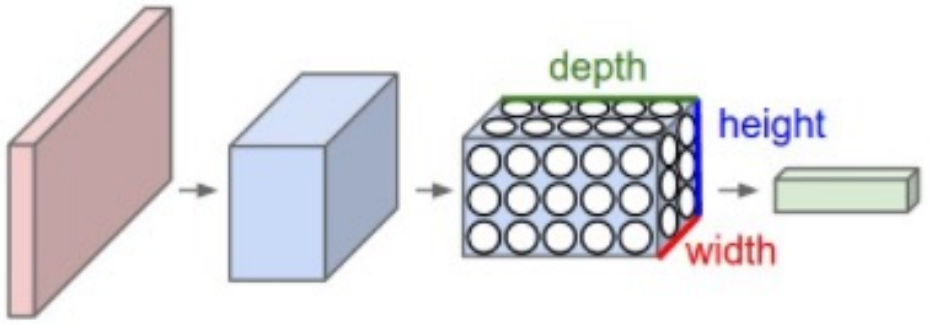
Max pooling & Fully connected layers

Full CNN Architecture

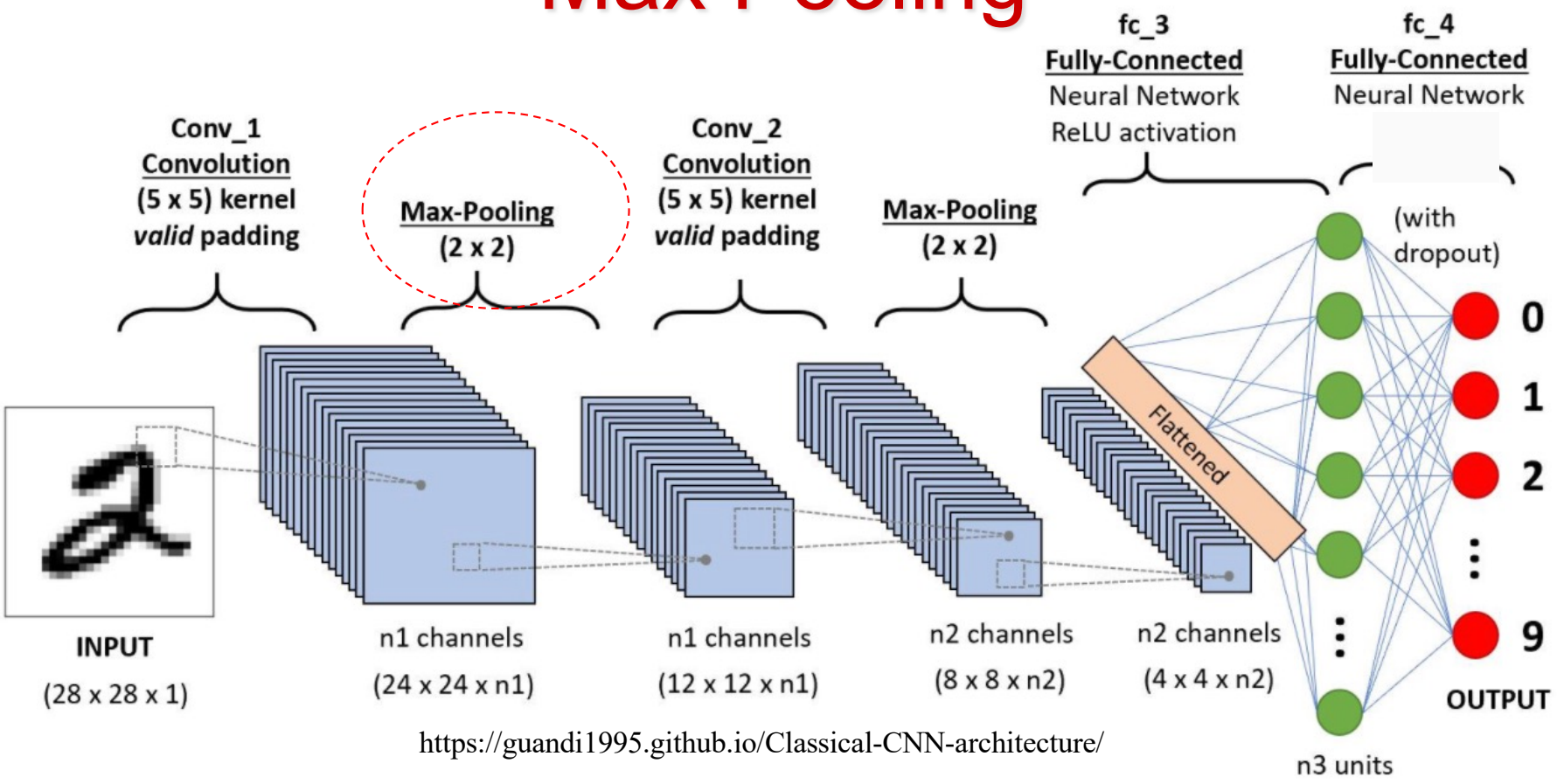


https://guandi1995.github.io/Classical-CNN-architecture/

Alternative drawing:

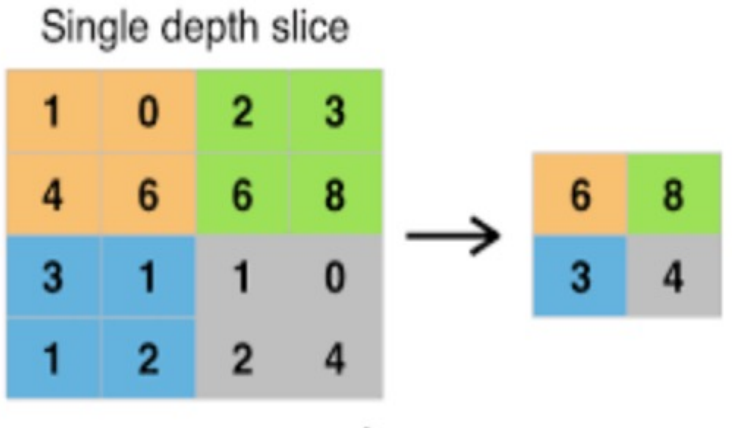


Max Pooling



https://guandi1995.github.io/Classical-CNN-architecture/

- A node in a max-pool layer gets as input the maximum of the activation of the nodes in its receptive field.
- Non-linear down-sampling.

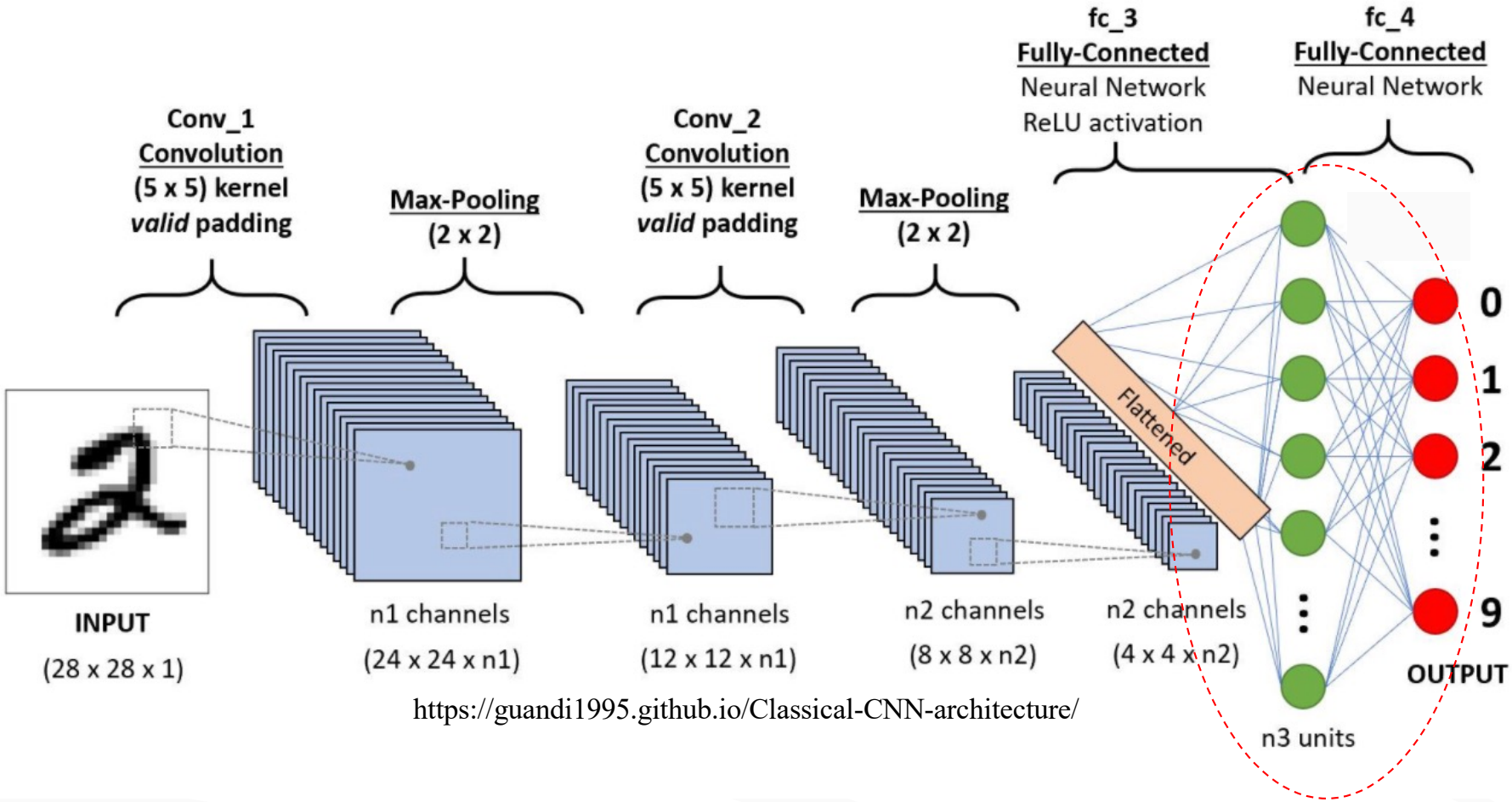


Max Pool Size

Max-pooling is useful in vision for two reasons:

- Reduces computation for upper layers by non-maximal supression
- Provides translation invariance

- Common application of MaxPool is done on a 2x2 region with a stride of 2.
  - Very large input images may warrant 4x4 pooling in the lower-layers.
  - But this will reduce the dimension of the signal by a factor of 16, and may result in throwing away too much information!



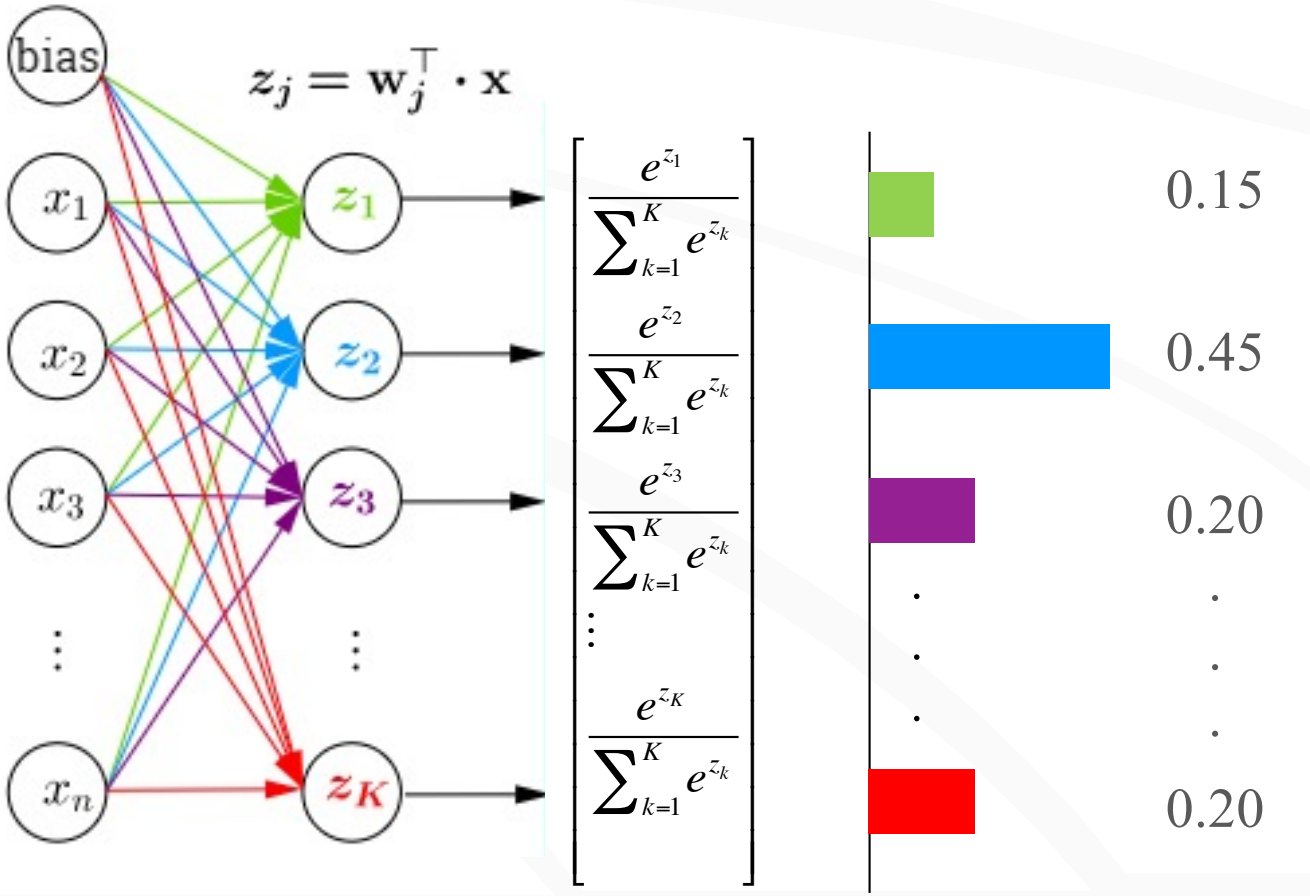
https://guandi1995.github.io/Classical-CNN-architecture/

- The last few layers of a CNN are fully connected layers.
- k output nodes for a k-class classification

SoftMax Layer

- Logistic or Softmax layer (activation function) used at the output layer.
  - Given a test input **x**, we want our hypothesis to estimate the probability that P(y=k|**x**) for each value of k=1,…,K.
  - I.e., we want to interpret the outputs as the posterior probabilities of the class

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



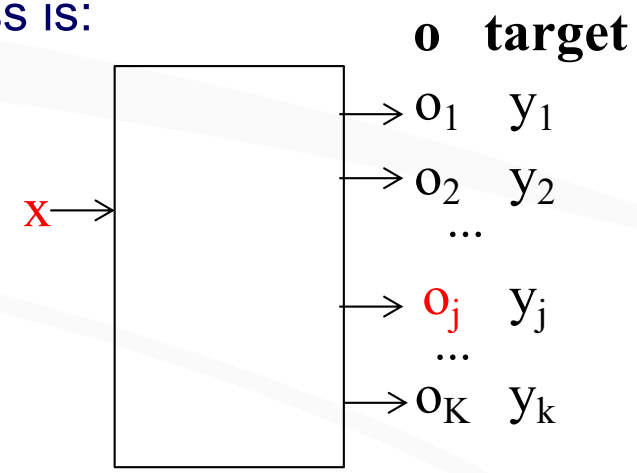
Cross-Entropy loss (Log loss) for classification

Most networks use the Cross-Entropy loss (Categorical Cross-Entropy)

- also a bit wrongly called softmax loss sometimes

For N training samples and K classes, the cross-entropy loss is:

$$J = -\sum_{j=1}^{K} 1\{y_j = 1\} \log o_j$$



where **y** is the target and **o** is the output for input **x**.

Note: Indicator function: 1{...} = 1 if ... is true; 0 otherwise.

Convolutional Neural Networks

Choosing Hyper Parameters

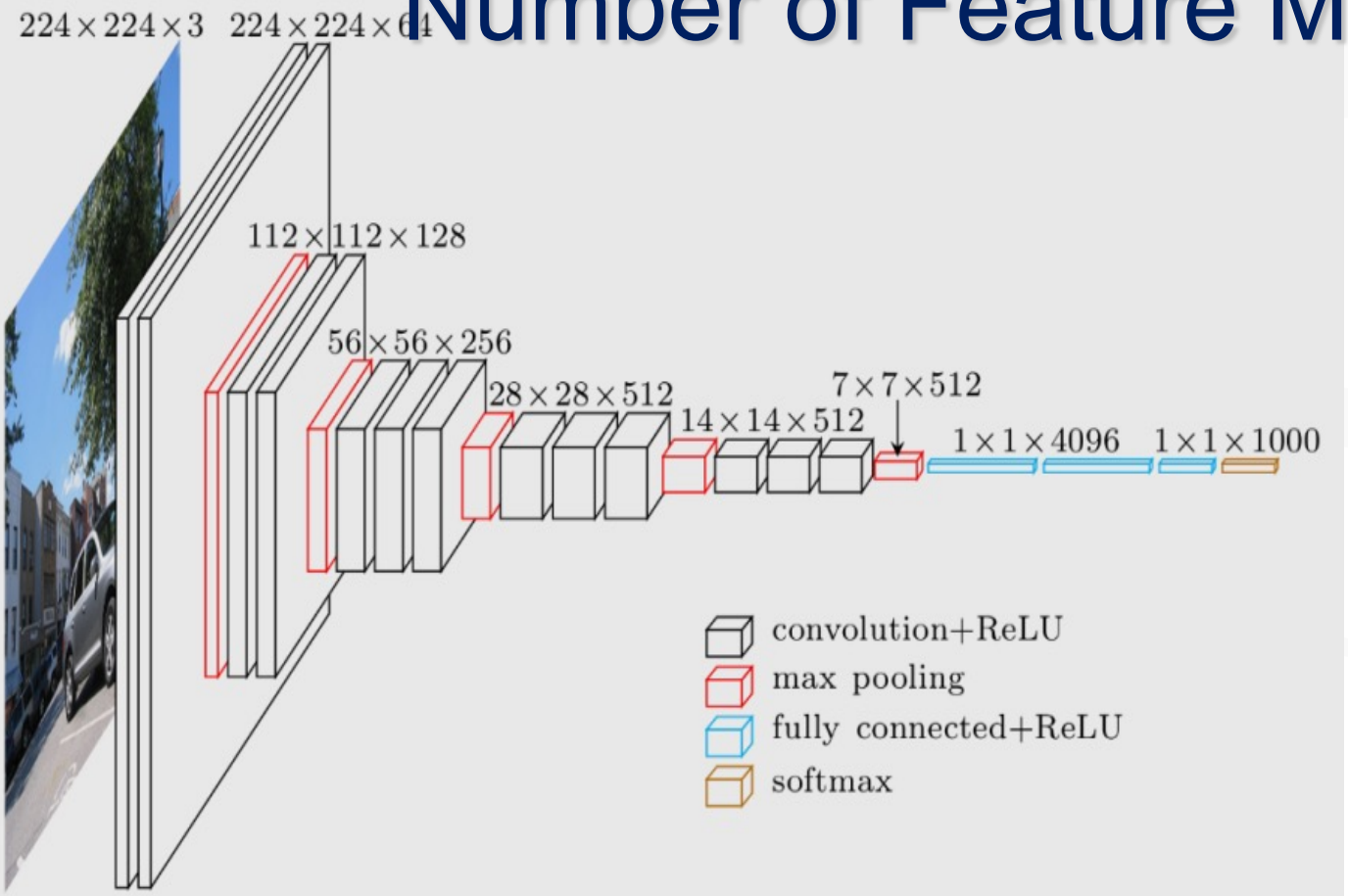
Choosing Hyper Parameters: Receptive Field Size

Common filter sizes found in the literature vary greatly, usually based on the dataset.

Best results on MNIST-sized images (28x28) are usually in the 5x5 range on the first layer, while natural image datasets (often with hundreds of pixels in each dimension) tend to use larger first-layer filters of shape 12x12 or 15x15.

The trick is thus to find the right level of “granularity” (i.e. filter shapes) in order to create abstractions at the proper scale, given a particular dataset.

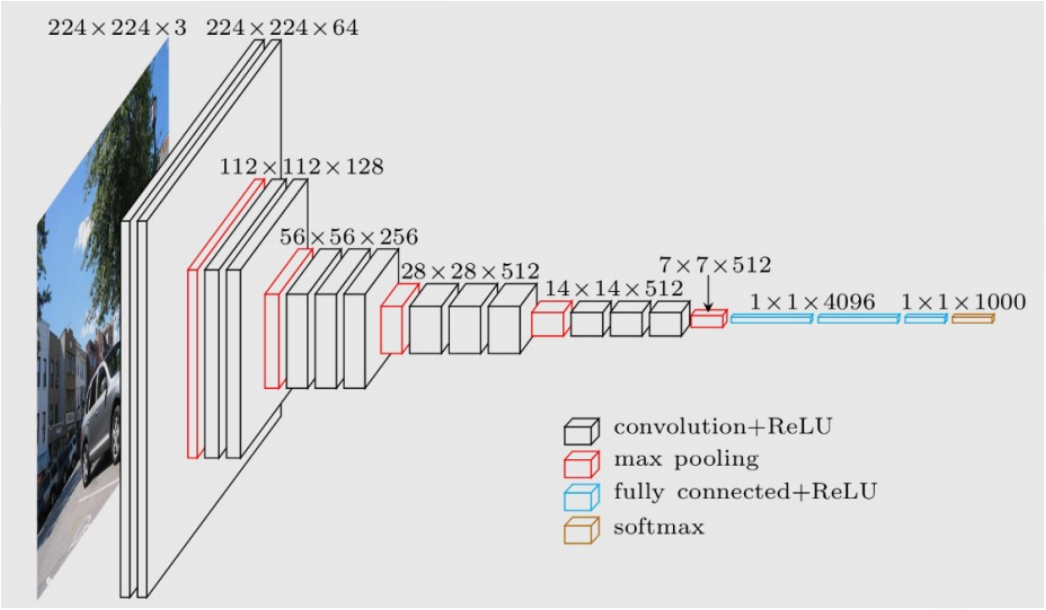
Choosing Hyper Parameters: Number of Feature Maps



- Total weights in the first conv. layer using 3x3 kernels:
  [(3x3x3)+1]x64 = 1,728
- Number of computation in the first conv. layer is expensive:
  (224x224)x(3x3x3)x64 = 86,704,128

Choosing Hyper Parameters: Number of Feature Maps and Layers

- Since feature map size decreases with depth, layers near the input layer will tend to have fewer filters while layers higher up can have much more.
- We want deeper networks, but at the same time want to reduce the number of weights in the network, to prevent overfitting.



Training Convolutional Neural Networks

Mini-Batch

Gradient descent normally can be done in:

- batch mode (more accurate gradients) or
- stochastic fashion (much faster learning with large amounts of training data).

In deep learning, we use mini-batches, which is the set of training images from which the gradient is calculated, before a single update.

- Mini-batch size should be as high as possible, as your computer allows (20-256 are typical)

Learning Rate

Large learning rates may cause divergence

Reduce in time

Batch normalization helps decide the learning rate for different layers

...

Batch Normalization

From Ioffe and Szegedy, 2015:

- 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 difficult to train models with saturating nonlinearities.
- Make normalization a part of the model architecture and perform normalization of node activations for each training mini-batch.
  - Normalization applied to each feature dimension independently
  - Same accuracy with 14x fewer iterations.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

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

https://arxiv.org/pdf/1502.03167.pdf

See also Batch Renormalization, 2017

Dropout

Deep convolutional neural networks have a large number of parameters.

- If we don't have sufficiently many training examples to constrain the network, the neurons can learn the noise (idiosyncrasies) in the data.
- Regularization puts constrains on a learning system to reduce overfitting (e.g. penalizing for large weight magnitudes in NNs, ...).

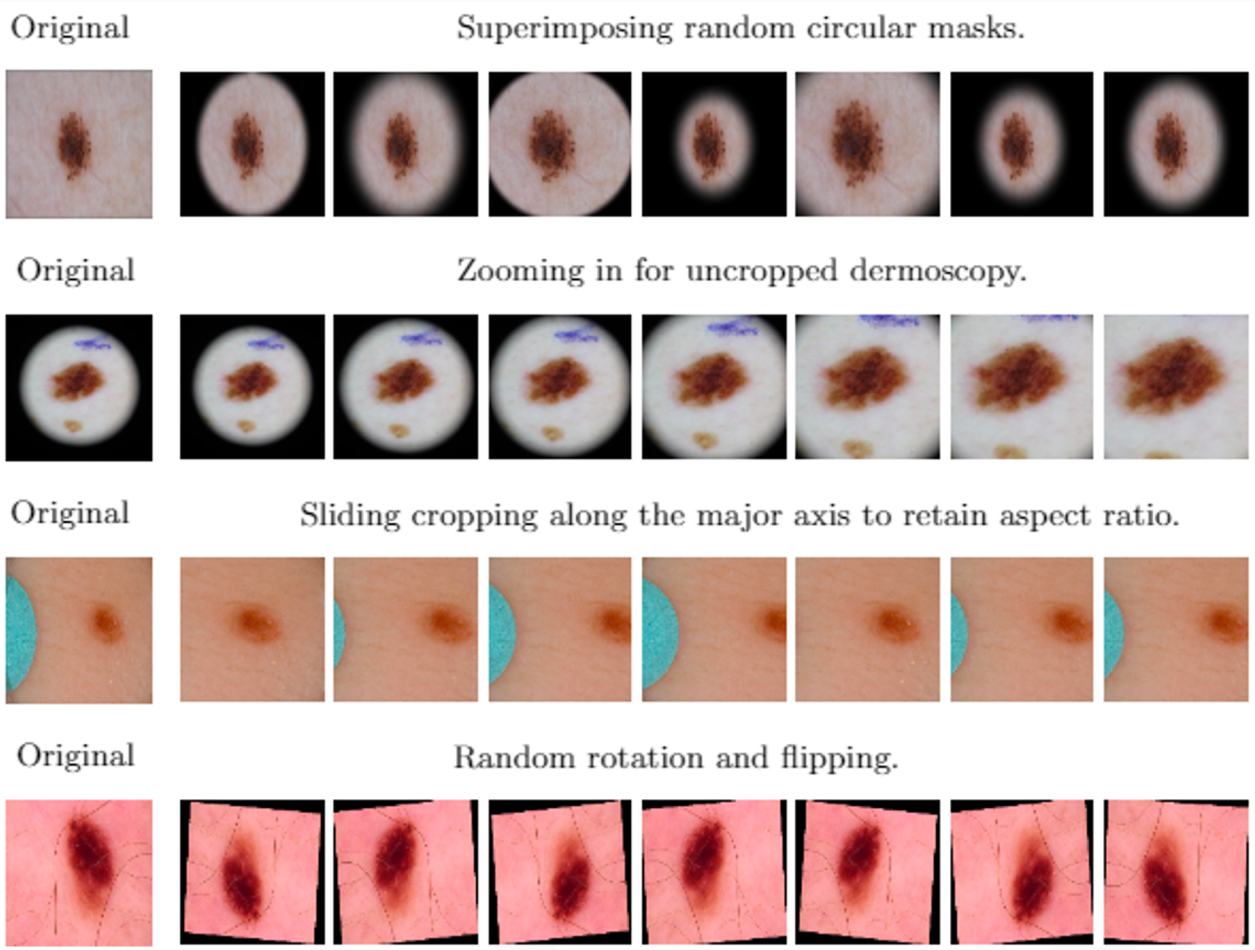
Another novel idea in deep learning is dropout, whereby some nodes are “dropped out” during the network training with a preset probability.

- Network learns to cope with failures.
- Sampling from an ensemble of deep networks, to harvest benefits of ensembles.
- Must scale weights during testing.

Data Augmentation

Training data is augmented with artificial samples, through translation, rotation, scale, reflection, elastic deformations, intensity variations..., in order to obtain more robust systems.

- Data augmentation is done internally within CAFFE and can be supplemented as well.
- Data augmentation is done to increase train data size by 10xfold or more and is very effective in reducing overfitting.



Random augmented samples from ISIC2019 training set

Transfer LearningFine-Tuning

Transfer learning

Instead of training a deep network from scratch, you can:

- take a network trained on a different domain for a different source task
  - E.g. Network is trained with ImageNet data (millions of images, 1000 classes)
- adapt it for your domain and your target task

Transfer learning is the general term for transferring knowledge acquired in one domain to another one.

ILSVRC-2010 images

from AlexNet paper

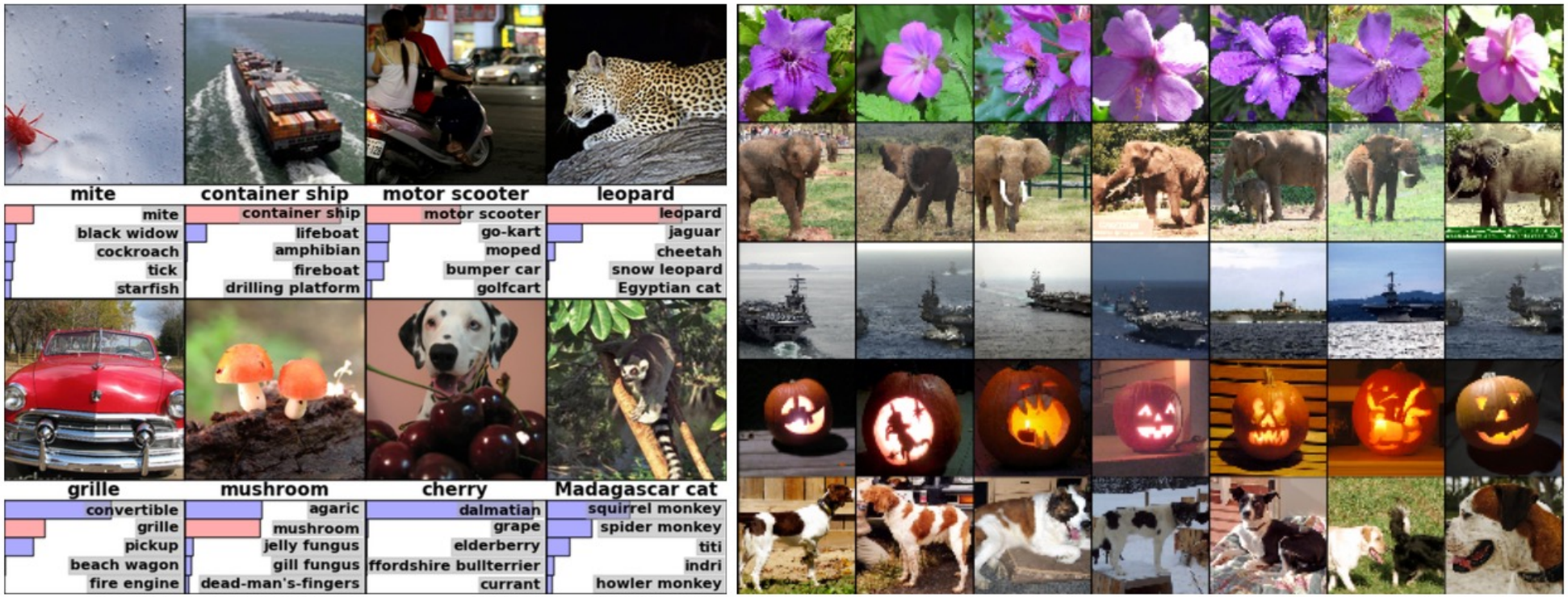


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Feature Detection

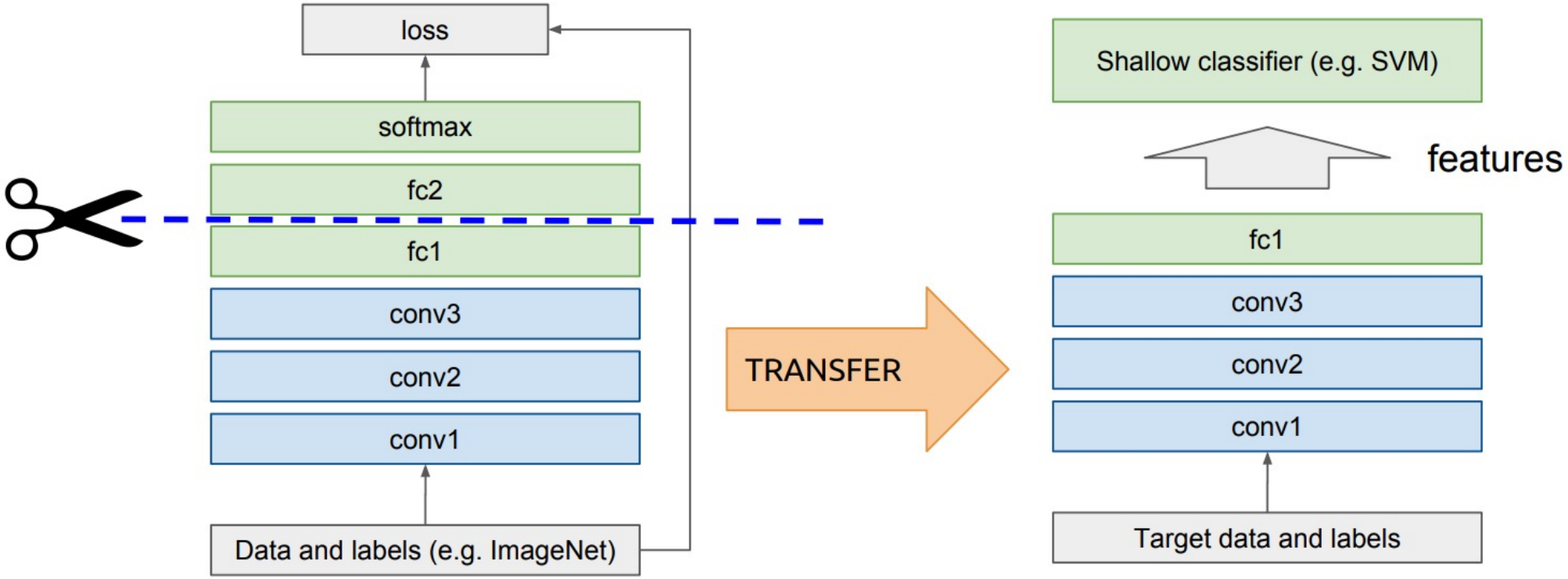


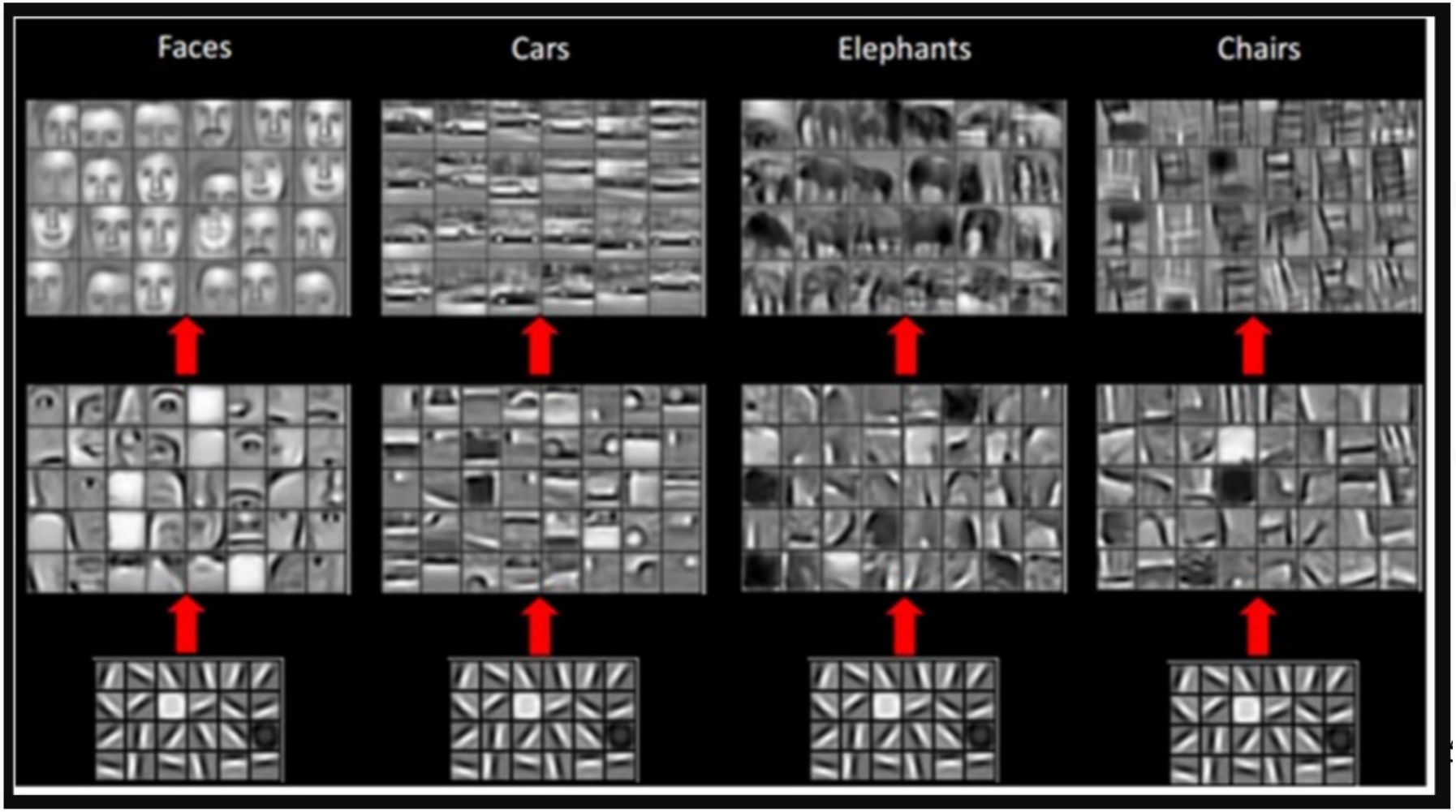
Image from http://imatge-upc.github.io/telecombcn-2016-dlcv/slides/D2L5-transfer.pdf

Fine-Tuning

- Take a pretrained network, cut-off and replace the last layer as before, fine-tune the network using backpropagation.
- Bottom n layers can be frozen: not updated during backpropagation.

Transfer learning

- Instead of training a deep network from scratch, you can take a network trained on a different domain (e.g. ImageNet) for a different source task, and adapt it for your domain (e.g. plant recognition) and your target task



How to fine-tune?

-> **How many iterations**: as many as you can afford.

-> **Data augmentation**: how many more samples are to be generated per input image?

-> **Batch size**: the number of input samples used during gradient calculation for weight update

+ **learning rate**, **weight decay**, **momentum** (typically initialized to default values)

The first 3 affect training time strongly!

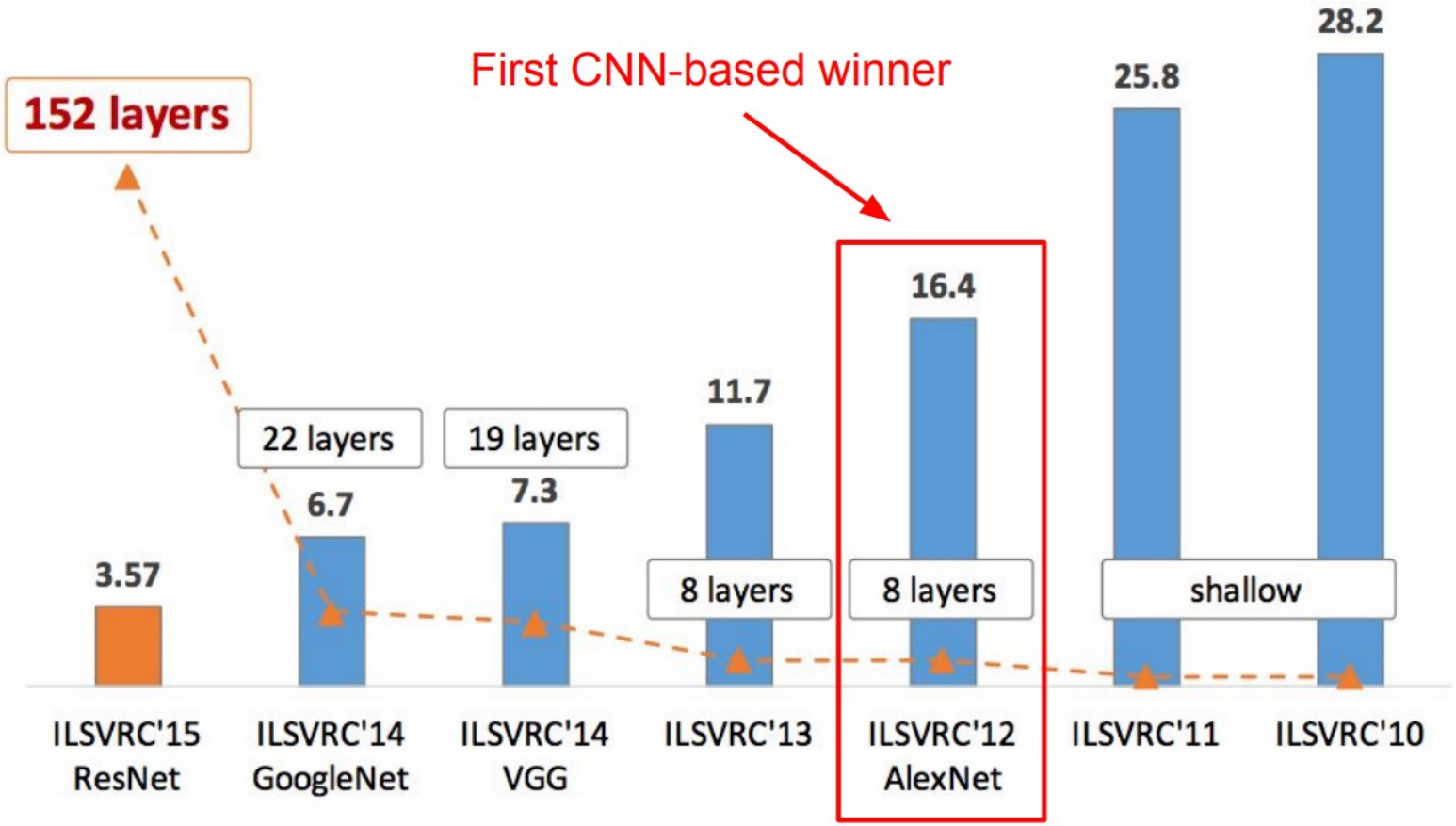
	Branch	Entire	Flower	Fruit	Leaf	LeafScan	Stem	Overall
GoogLeNet (100K, 20, 10x)	44.09	38.36	67.93	57.65	60.57	94.16	37.01	61.06
GoogLeNet (300K, 20, 10x)	55.08	46.05	76.23	67.96	68.07	95.77	45.76	68.57
GoogLeNet (500K, 20, 10x)	55.02	47.66	76.43	69.11	69.13	95.58	45.49	69.11
GoogLeNet (100K, 40, 10x)	45.63	41.03	70.05	61.23	60.73	93.20	36.49	62.48
GoogLeNet (100K, 60, 10x)	50.98	41.09	73.07	63.59	65.50	94.19	41.52	65.18
GoogLeNet (100K, 20, 80x)	54.01	48.06	73.38	64.99	68.63	94.85	39.84	67.32
GoogLeNet (300K, 60, 80x) (*)	67.56	60.28	83.30	76.88	76.30	96.93	54.33	76.87

- The best GoogLeNet results are obtained with (300K,60,80x) as 76.87% on the test set.
- Increasing the number of iterations is much more beneficial than increasing the batch size, for the same total running time (68.57% vs. 65.18 %).
- Data augmentation is the second most important training parameter (69.11 % vs. 67.32 %)

Convolutional Neural Networks

The rest of the talk is a brief overview of some well-known CNN architectures and the accompanying papers

Top-5 Errors in ILSVRC over the Years



ILSVRC-2010 images

from AlexNet paper

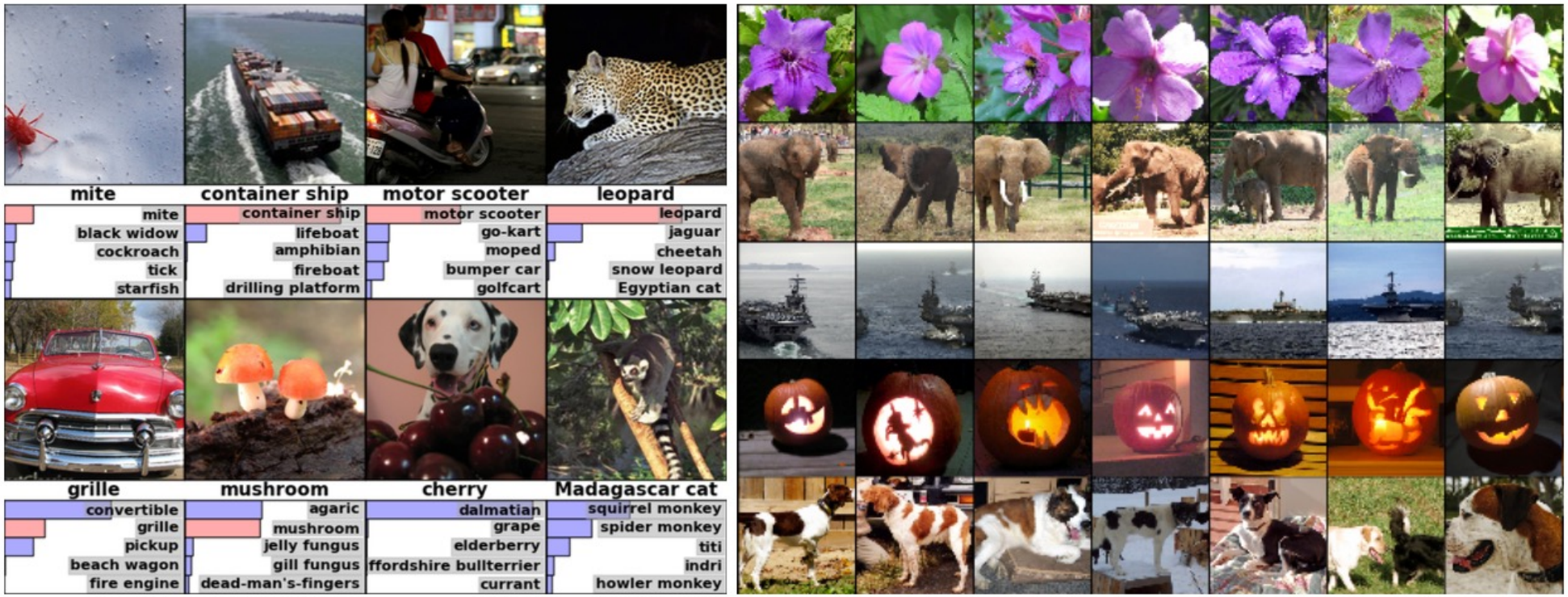


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

“Famous” Deep Networks

AlexNet, the **winner of the ILSVRC 2012** competition by a large margin, **15.3%** top-5 error rate, where the error rate of the second place was 26.2%. The network has **62.3 million parameters**, It contains 5 convolutional layers and 3 fully connected layers.

GoogleNet, the **winner of the ILSVRC 2014** competition from Google. It achieved a top-5 error rate of **6.67%**. The network used batch normalization. This architecture is based on several very small convolutions in order to reduce the number of parameters. GoogleNet architecture consisted of a 22 layer CNN but reduced the number of parameters from around **60 million (in AlexNet) to 4 million.**

VGG16-Net, the **runner-up at the ILSVRC 2014** competition with **7.3% error rate**. VGGNet consists of 16 convolutional layers with a very uniform architecture, only 3x3 convolutions, with a lot of filters. VGGNet consists **of 138 million parameters**, which can be a bit challenging to handle. Their are a lot of versions of VGGNet like, VGG-11, VGG-13, VGG-16, VGG-19, where the number is an indication of the number of convolutional layers. Mostly, when VGG is mentioned, it is either VGG-16 or VGG-19

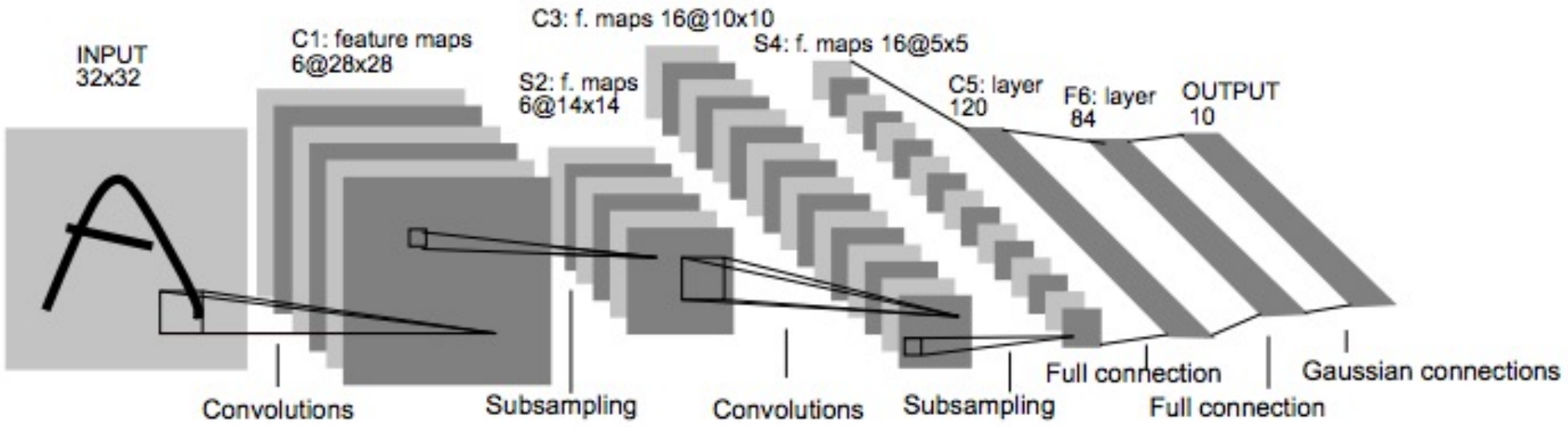
“Famous” Deep Networks

ResNet, so-called Residual Neural Network (at the ILSVRC 2015). It is an architecture with “skip connections” and heavy batch normalization. Thanks to “skip connections”, it is possible to train a NN with **152 layers** while still having lower complexity than VGGNet. It achieves a top-5 error rate of **3.57%** which beats human-level performance on this dataset.

SENet “Squeeze-and-Excitation”, the **winner of the ILSVRC 2017** competition with **2.25% top-5 error rate**. (SE) block, which explicitly models the interdependence between channels. These blocks can be integrated with modern architectures, such as ResNet, and improves their representational power. With the evolved architectures and advanced training techniques, such as batch normalization (BN), the network becomes deeper and the training becomes more controllable, and the performance of object classification is continually improving.

LeNet

LeCun et al.1989



- The lower-layers are composed to alternating convolution and max-pooling layers.
- First CONV layer has 6 feature maps where each node has with 5x5 receptive fields. Total of 156 free parameters (6x5x5 + 6).
- The upper-layers are fully-connected and correspond to a traditional MLP (hidden layer + logistic regression).
- The input to the first fully-connected layer is the set of all features maps at the layer below.

ALEXNET

ImageNet Classification with Deep Convolutional Neural Networks

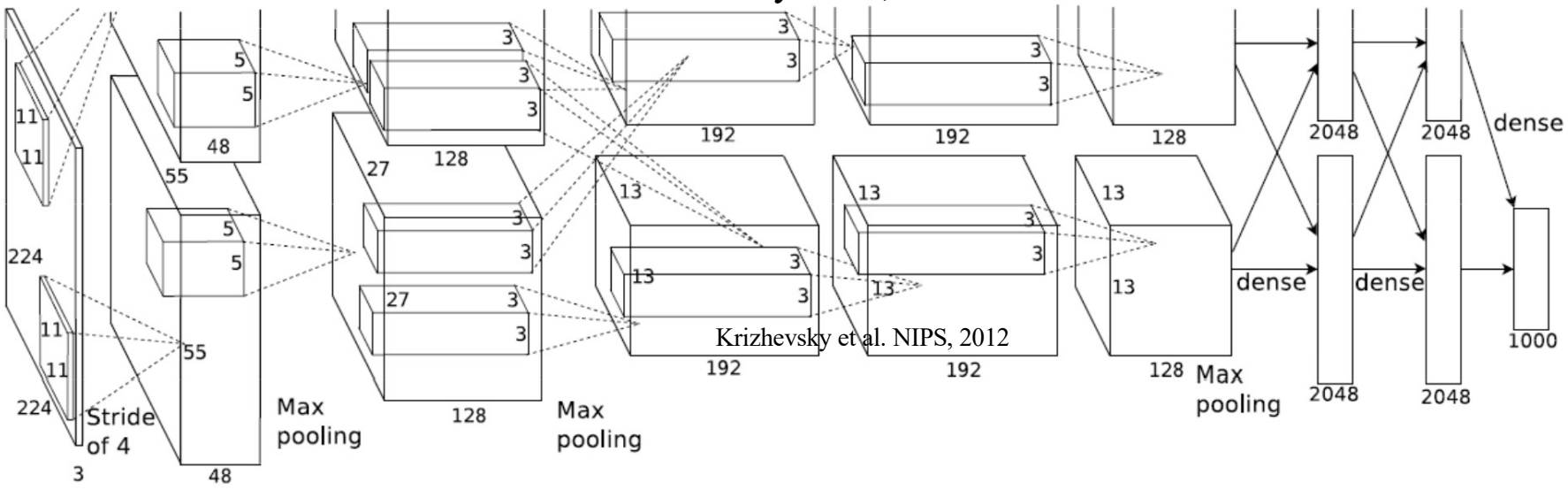
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

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

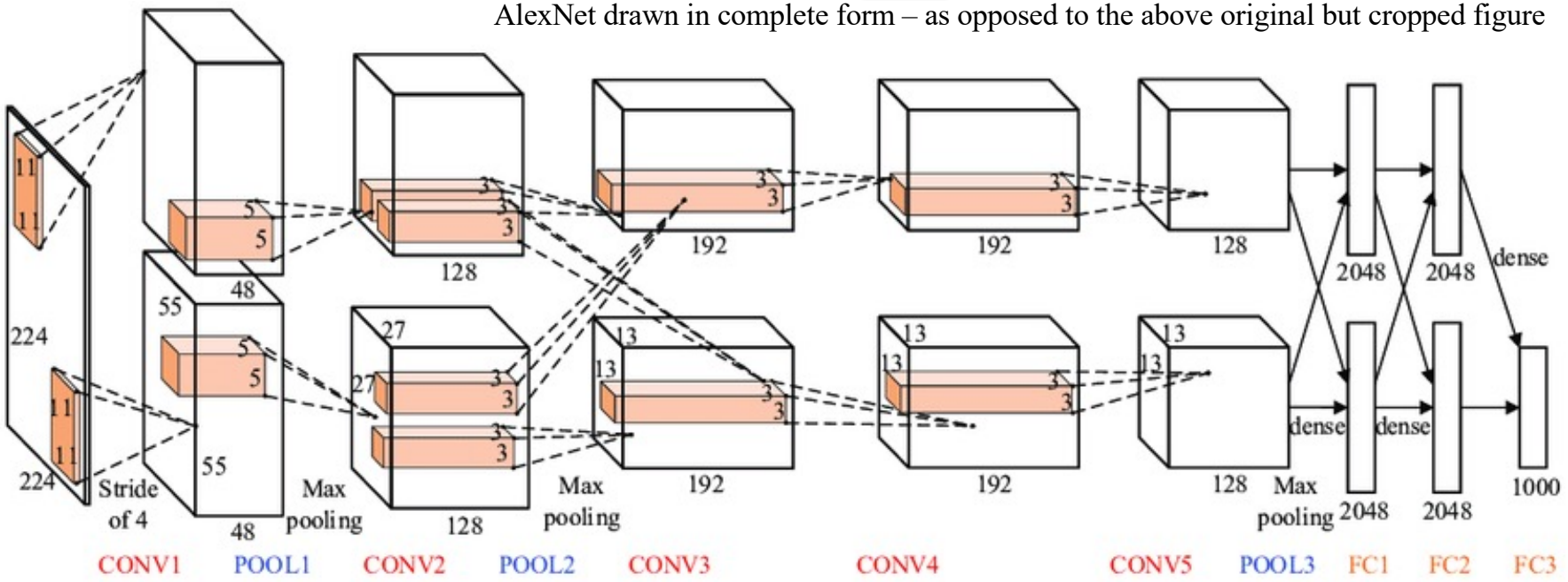
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

AlexNet

Krizhevsky et al., 2012



AlexNet drawn in complete form – as opposed to the above original but cropped figure



https://www.researchgate.net/figure/Network-architecture-of-AlexNet-CONV-convolutional-layer-FC-fully-connected-layer_fig1_331603323

3.1 ReLU Nonlinearity

The standard way to model a neuron's output f as a function of its input x is with $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = \max(0, x)$. Following Nair and Hinton [20], we refer to neurons with this nonlinearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with $\tanh$ units. This is demonstrated in Figure 1, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. This plot shows that we would not have been able to experiment with such large neural networks for this work if we had used traditional saturating neuron models.

We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett et al. [11] claim that the nonlinearity $f(x) = |\tanh(x)|$ works particularly well with their type of contrast normalization followed by local average pooling on the Caltech-101 dataset. However, on this dataset the primary concern is preventing overfitting, so the effect they are observing is different from the accelerated ability to fit the training set which we report when using ReLUs. Faster learning has a great influence on the performance of large models trained on large datasets.

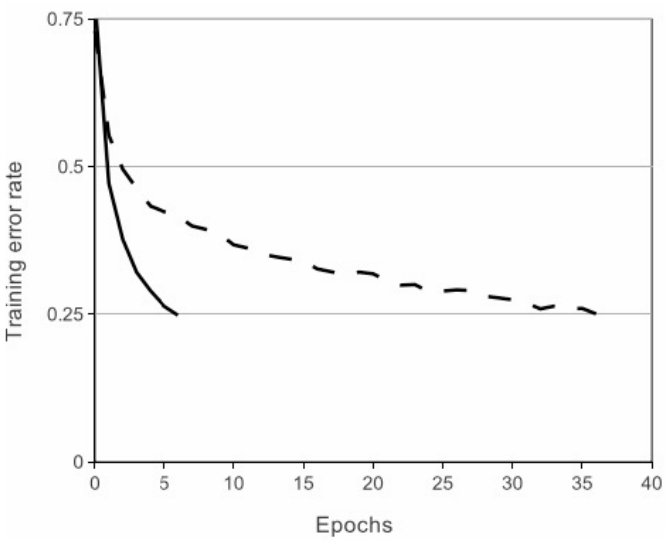


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with $\tanh$ neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

3.2 Training on Multiple GPUs

A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it. It turns out that 1.2 million training examples are enough to train networks which are too big to fit on one GPU. Therefore we spread the net across two GPUs. Current GPUs are particularly well-suited to cross-GPU parallelization, as they are able to read from and write to one another's memory directly, without going through host machine memory. The parallelization scheme that we employ essentially puts half of the kernels (or neurons) on each GPU, with one additional trick: the GPUs communicate only in certain layers. This means that, for example, the kernels of layer 3 take input from all kernel maps in layer 2. However, kernels in layer 4 take input only from those kernel maps in layer 3 which reside on the same GPU. Choosing the pattern of connectivity is a problem for cross-validation, but this allows us to precisely tune the amount of communication until it is an acceptable fraction of the amount of computation.

The resultant architecture is somewhat similar to that of the "columnar" CNN employed by Cireşan et al. [5], except that our columns are not independent (see Figure 2). This scheme reduces our top-1 and top-5 error rates by 1.7% and 1.2%, respectively, as compared with a net with half as many kernels in each convolutional layer trained on one GPU. The two-GPU net takes slightly less time to train than the one-GPU net[2].

[2]The one-GPU net actually has the same number of kernels as the two-GPU net in the final convolutional layer. This is because most of the net's parameters are in the first fully-connected layer, which takes the last convolutional layer as input. So to make the two nets have approximately the same number of parameters, we did not halve the size of the final convolutional layer (nor the fully-conneced layers which follow). Therefore this comparison is biased in favor of the one-GPU net, since it is bigger than "half the size" of the two-GPU net.

AlexNet

5 Details of learning

We trained our models using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005. We found that this small amount of weight decay was important for the model to learn. In other words, weight decay here is not merely a regularizer: it reduces the model's training error. The update rule for weight w was

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

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

where i is the iteration index, v is the momentum variable, ϵ is the learning rate, and $\left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$ is the average over the ith batch D_i of the derivative of the objective with respect to w, evaluated at w_i.

Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

We initialized the weights in each layer from a zero-mean Gaussian distribution with standard deviation 0.01. We initialized the neuron biases in the second, fourth, and fifth convolutional layers, as well as in the fully-connected hidden layers, with the constant 1. This initialization accelerates the early stages of learning by providing the ReLUs with positive inputs. We initialized the neuron biases in the remaining layers with the constant 0.

We used an equal learning rate for all layers, which we adjusted manually throughout training. The heuristic which we followed was to divide the learning rate by 10 when the validation error rate stopped improving with the current learning rate. The learning rate was initialized at 0.01 and

AlexNet

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

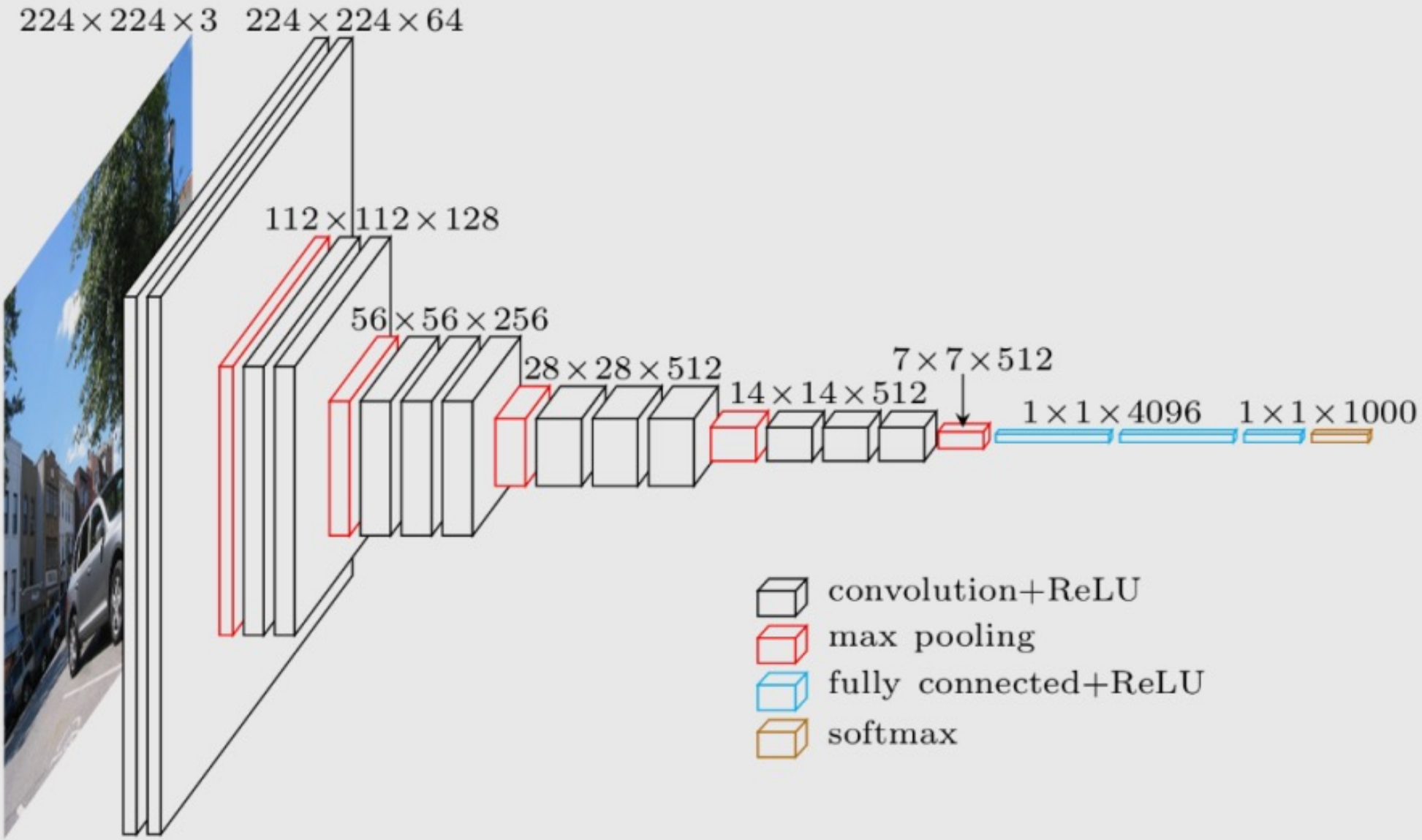
The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224×224 patches (and their horizontal reflections) from the 256×256 images and training our network on these extracted patches[4]. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five 224×224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components,

[4]This is the reason why the input images in Figure 2 are $224 \times 224 \times 3$-dimensional.

VGG-16 Network

Simonyan & Zisserman, 2014



RESNET

Deep Residual Learning for Image Recognition

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

Main Idea

Performance seems to improve with increased depth, so should we just keep stacking more layers?

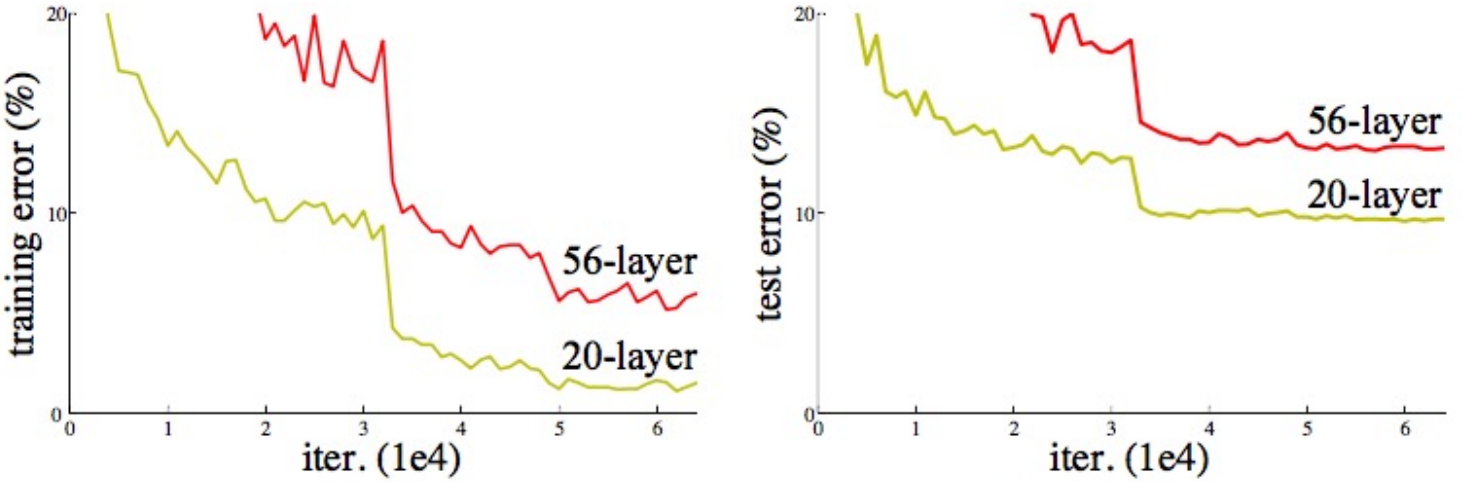


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

From the Paper

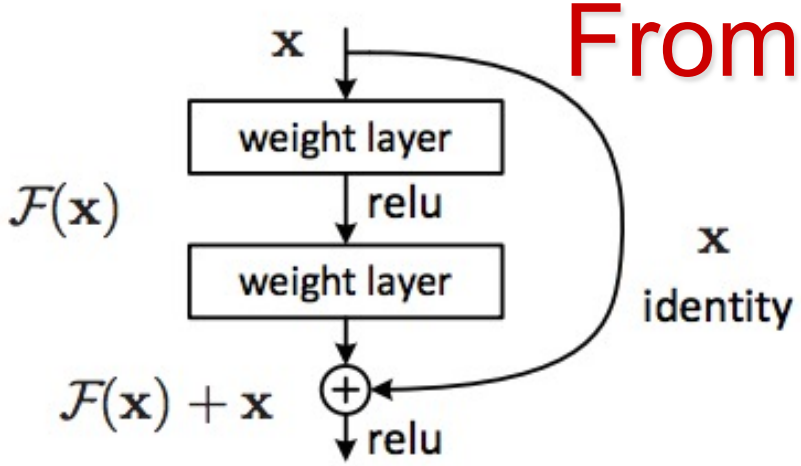


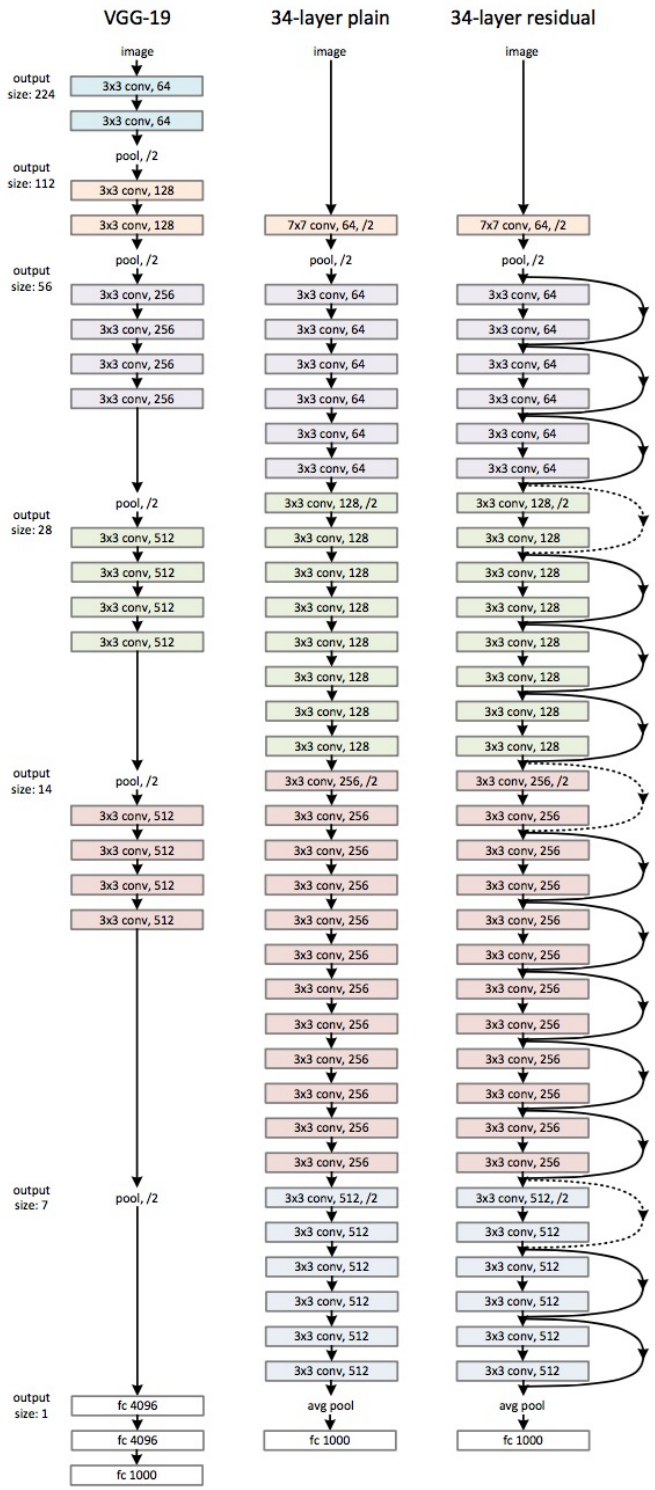
Figure 2. Residual learning: a building block.

"Identity shortcut connections add neither extra parameter nor computational complexity. "

- Assume the 2-*stack layer* meant to learn the desired mapping H(x).
- By adding the residual connection, we are asking it to learn H(x)-x.
  - Let's call it F(x).
  - Then we have F(x) = H(x) – x
- Hence F(X) is called the residual mapping (what is left/residual with respect to ``identity`` - i.e. x).

From the paper: *"We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.``*

Explanation: If the stacked layer is meant to implement identity (=pass the prev. block output x), it is easier to set the weights to 0, than if we did not have the residual connection.



Resnet adds skip-connections

From left to right: VGG, 34-layer plain, 34-layer residual networks.

Filters are doubled everytime the spatial resolution halved (128 to 256 etc) at the dashed lines

Comparison of Plain and Residual Networks

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

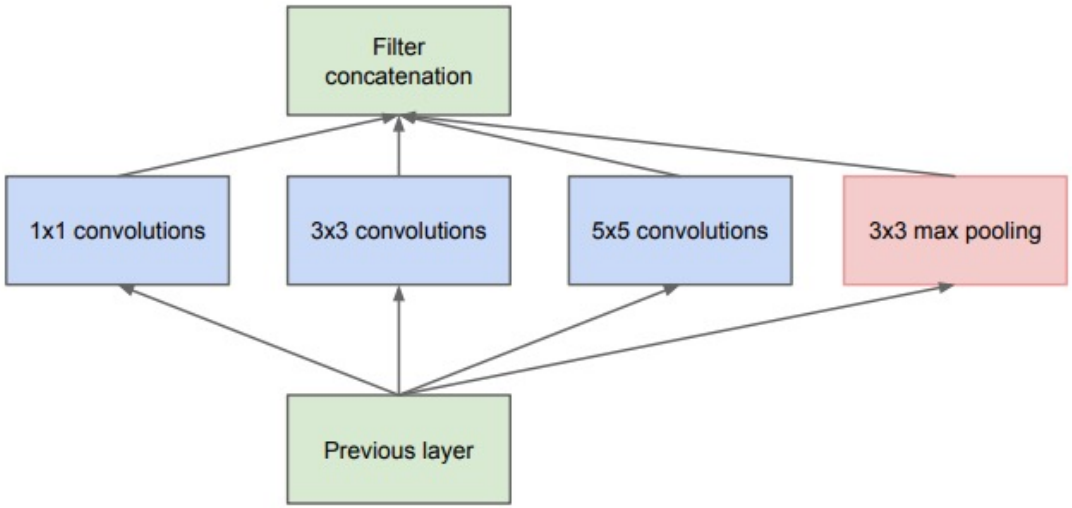
Looking at this table, we can see that:

- The 34-layer resnet has lower error on the validation data compared to 18-layer resnet (as desired), while plain counterpart has higher error when depth is increased.
- The residual networks are better in both cases (more so when depth is increased).

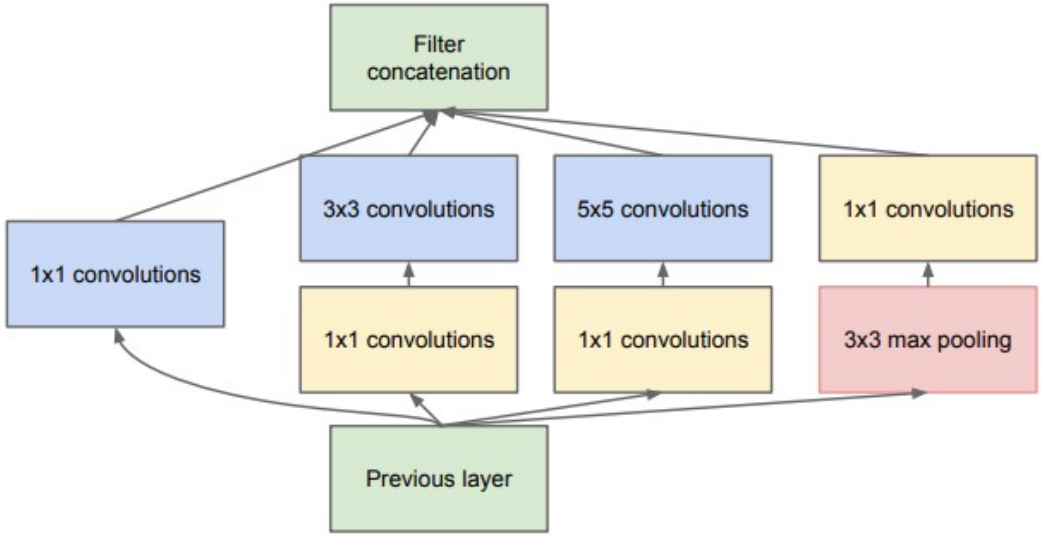
Inception

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

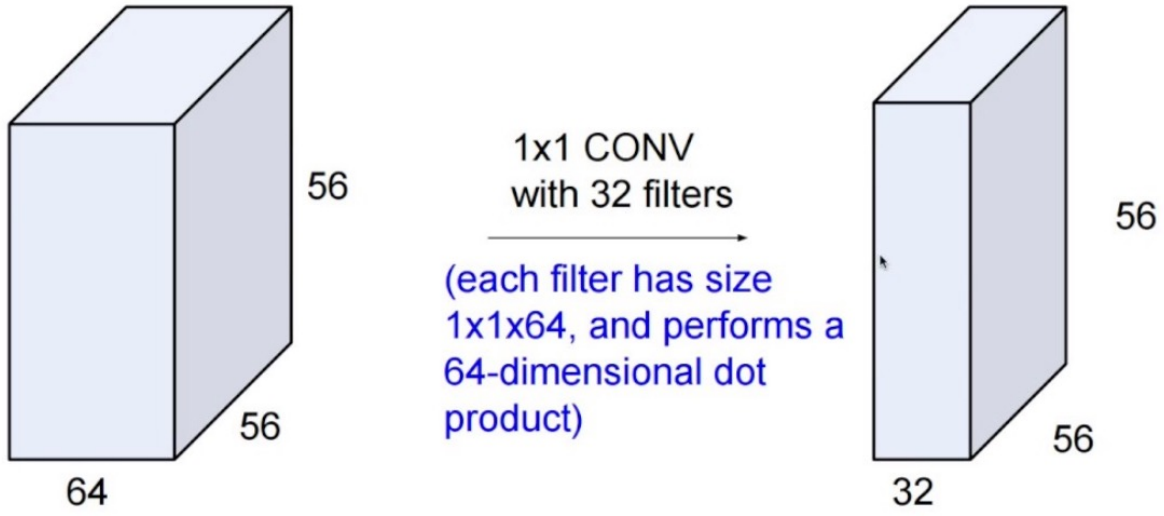


(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

1x1 Convolutions



- 1x1 convolution layers takes the full depth of the previous layer, but single pixel receptive fields.
- They thus reduce/increase the depth (here 64) into the desired size (here 32).

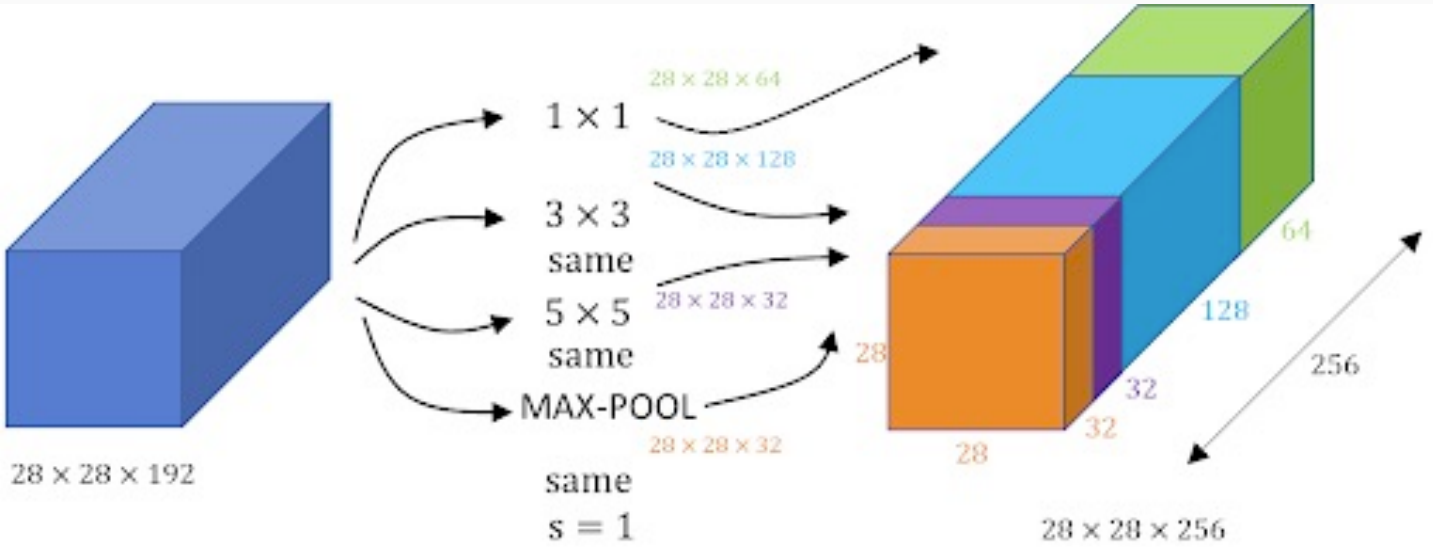


Image credit: http://datahacker.rs/deep-learning-inception-network/

- SAME uses enough padding to match keep Output size same as Input size using a stride of 1.
- So all filters create the same output dimension and we concatenate.

Thanks for Listening!