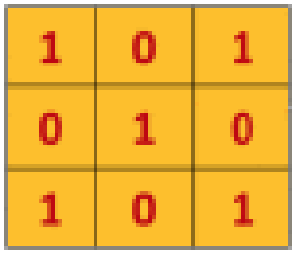
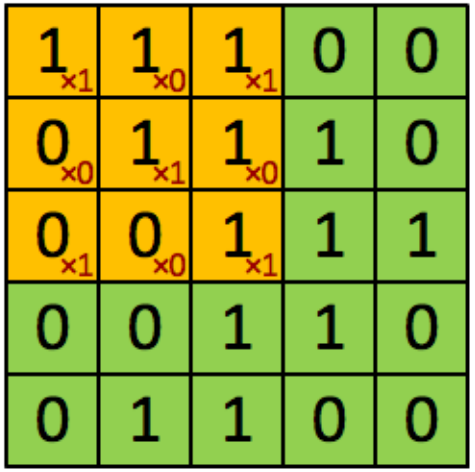
Convolutional Neural Networks

Convolutional Neural Networks

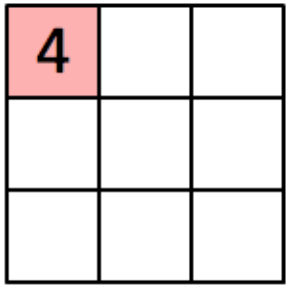
- Convolutional Layer [3, 4]



Filter / Kernel / Set of Weights / Feature Detector



Image



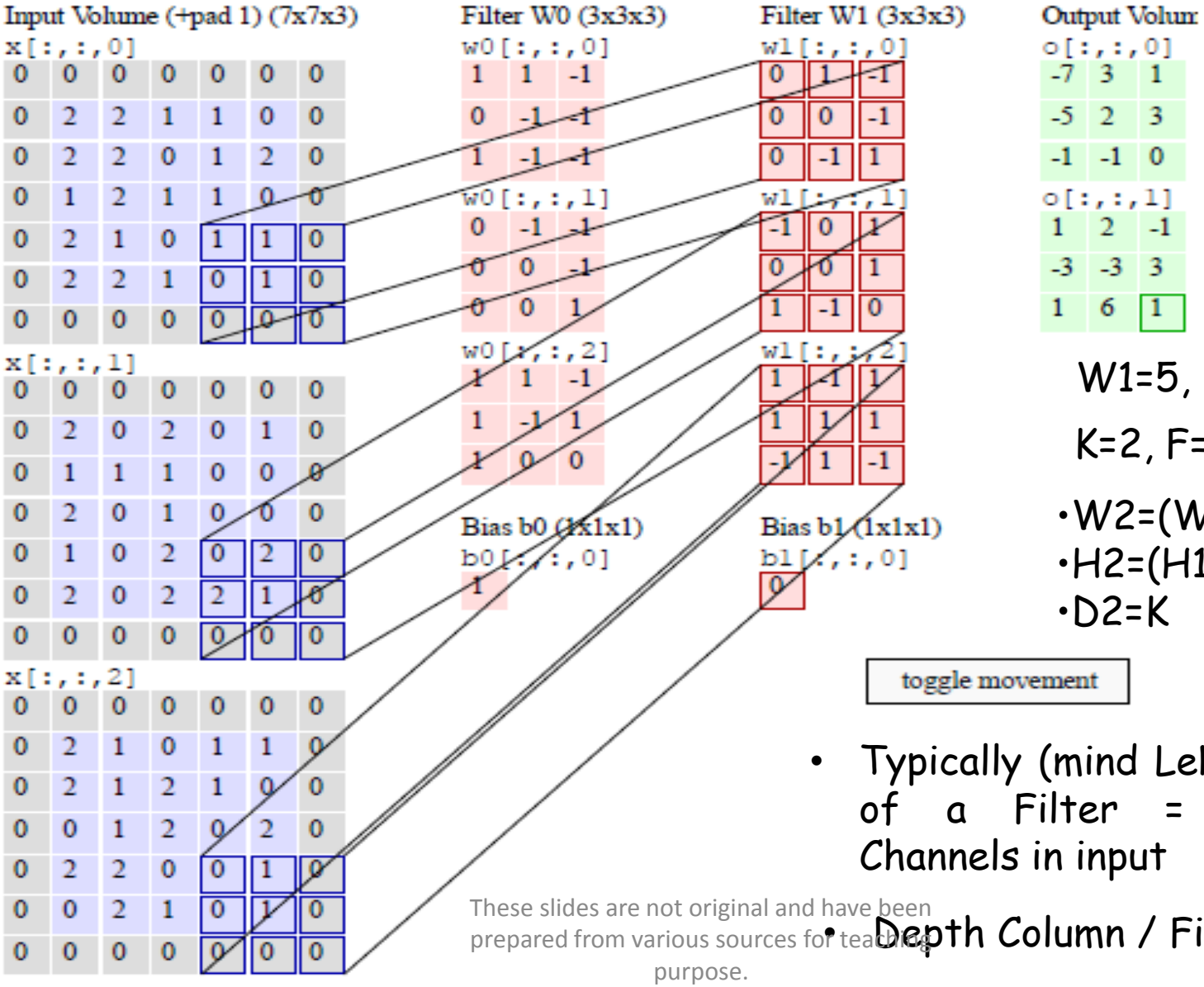
Convolved Feature

Activation Map / Feature Map

- Size of output volume = $\frac{W - F + 2P}{S} + 1$
  - where W is the size of our input volume, F is the size of our filter, P is the amount of padding, and S is the stride

Convolutional Neural Networks

- Convolution Layer [3]



W1=5, H1=5, D1=3

K=2, F=3, S=2, P=1

- W2=(W1−F+2P)/S +1
- H2=(H1−F+2P)/S +1
- D2=K

toggle movement

- Typically (mind LeNet 5), Depth of a Filter = Number of Channels in input
- Depth Column / Fiber

Convolutional Neural Networks

- Convolution Layer [4]

These slides are not original and have been prepared from various sources for teaching purpose.

Convolutional Neural Networks

- Convolution Layer [3]
  - Parameter Sharing

Convolutional Neural Networks

- Convolution Layer [3]
  - Local Connectivity & Receptive Field

Convolutional Neural Networks

- Convolution Layer [3]
  - Use of zero padding
    - Setting zero padding to be P = (F–1)/2 when the stride is S=1 ensures that the input volume and output volume will have the same size spatially.

Convolutional Neural Networks

- Convolution Layer [3]
  - Constraints on Stride
    - Note again that the spatial arrangement hyper-parameters have mutual constraints.
    - For example, when the input has size W=10, no zero-padding is used P=0, and the filter size is F=3, then it would be impossible to use stride S=2, since (W−F+2P)/S+1=(10−3+0)/2+1=4.5, i.e. not an integer, indicating that the neurons don't "fit" neatly and symmetrically across the input.
    - Therefore, this setting of the hyper-parameters is considered to be invalid, and a ConvNet library could throw an exception or zero pad the rest to make it fit, or crop the input to make it fit, or something.

Convolutional Neural Networks

- Convolution Layer [3]
  - Constraints on Stride
    - As we will see in the ConvNet architectures section, sizing the ConvNets appropriately so that all the dimensions "work out" can be a real headache, which the use of zero-padding and some design guidelines will significantly alleviate.

Convolutional Neural Networks

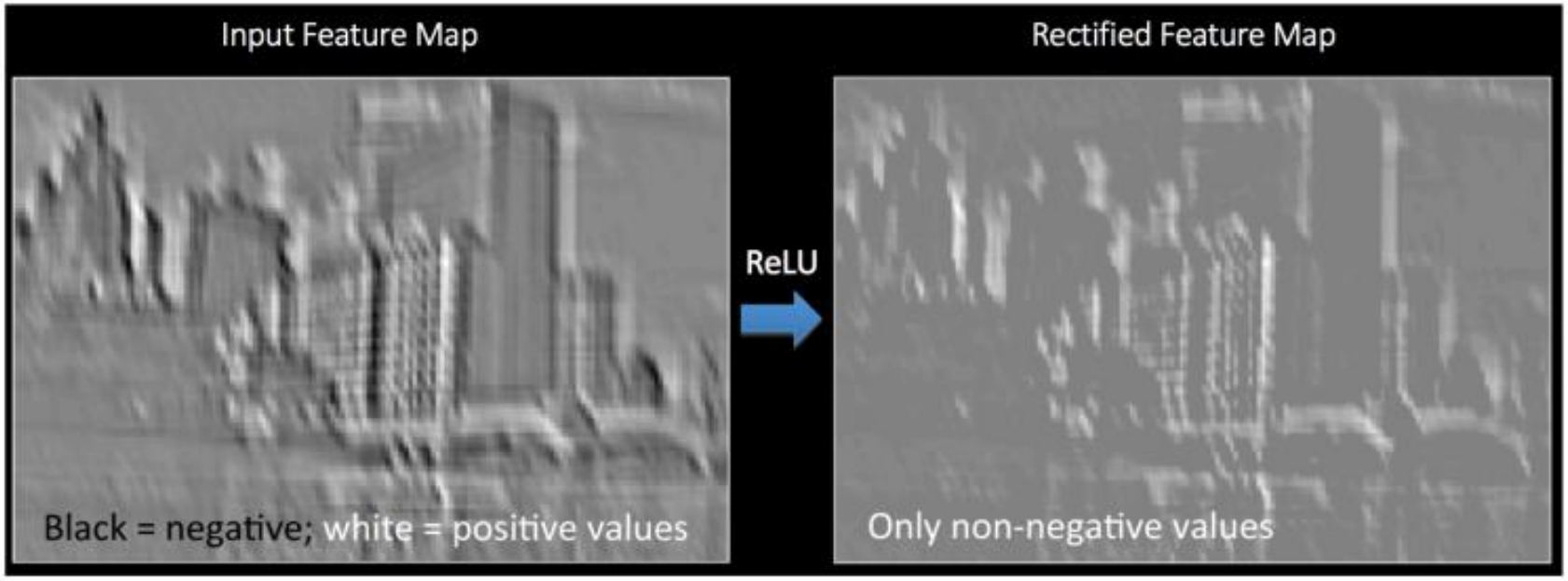
- Convolutional Layer [3]
  - Summary
    - Accepts a volume of size W1 × H1 × D1
    - Requires four hyper-parameters:
      - Number of filters K,
      - their spatial extent F,
      - the stride S,
      - the amount of zero padding P.
    - Produces a volume of size W2×H2×D2 where:
      - W2=(W1−F+2P)/S+1
      - H2=(H1−F+2P)/S+1
      - D2=K

Convolutional Neural Networks

- Convolutional Layer [3]
  - Summary
    - With parameter sharing, it introduces F·F·D1 weights per filter, for a total of (F·F·D1)·K weights and K biases.
    - In the output volume, the d^{th} depth slice (of size W2×H2) 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.

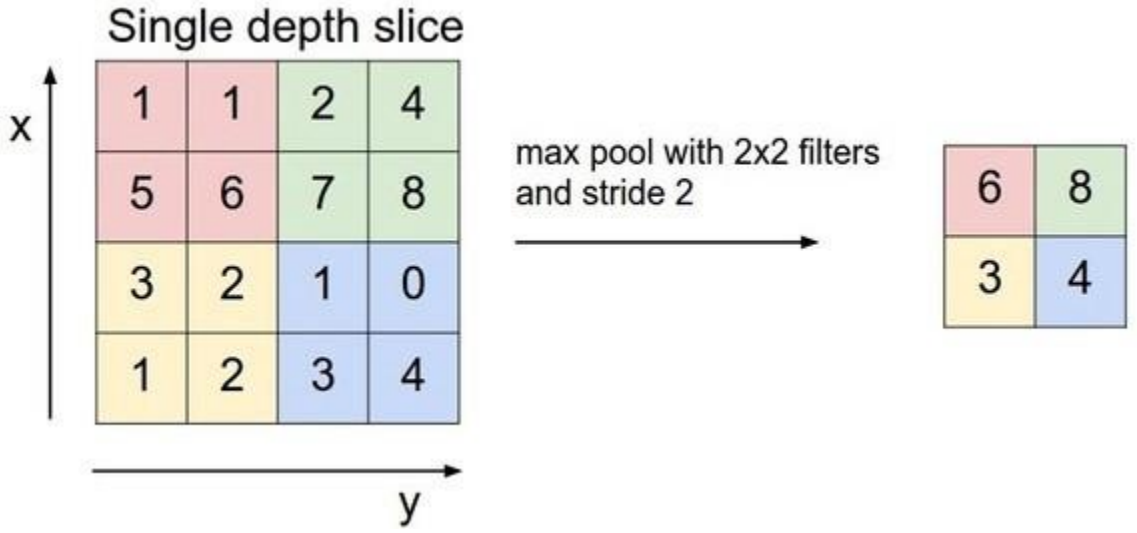
Convolutional Neural Networks

- The ReLU Operation [4]



Convolutional Neural Networks

- Max Pooling Layer [3]



- Size of output volume = $\frac{W - F}{S} + 1$

Convolutional Neural Networks

- Pooling Layer [3]
  - Summary
    - Accepts a volume of size W1×H1×D1
    - Requires two hyper-parameters:
      - their spatial extent F,
      - the stride S,
    - Produces a volume of size W2×H2×D2 where:
      - W2=(W1–F)/S+1
      - H2=(H1–F)/S+1
      - D2=D1
    - Introduces zero parameters since it computes a fixed function of the input
    - Note that it is not common to use zero-padding for Pooling layers

Convolutional Neural Networks

- Pooling Layer [3]
  - Summary
    - It is worth noting that there are only two commonly seen variations of the max pooling layer found in practice: **A pooling layer with F=3, S=2** (also called overlapping pooling), and **more commonly F=2, S=2**. Pooling sizes with larger receptive fields are too destructive.

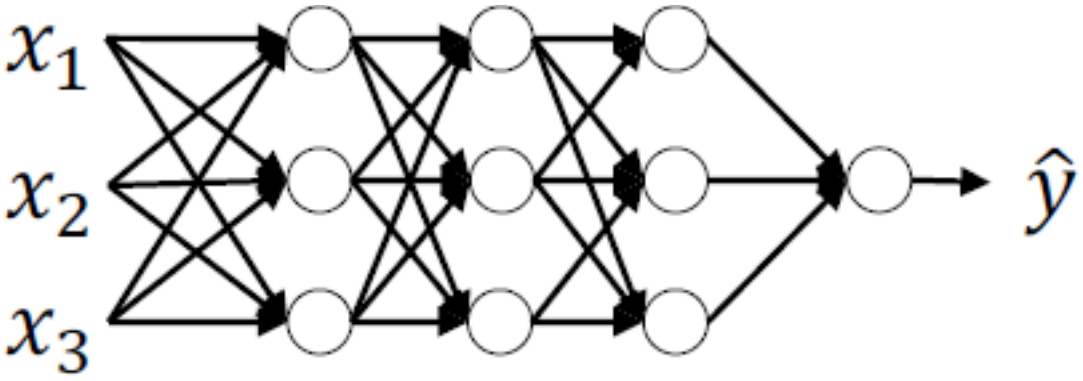
    - In addition to max pooling, the pooling units can also perform other functions, such as **average pooling** or even **L2-norm pooling**. Average pooling was often used historically but has recently fallen out of favour compared to the max pooling operation, which has been shown to work better in practice.

Convolutional Neural Networks

- Pooling Layer [3]
  - Getting rid of Pooling
    - Many people dislike the pooling operation and think that we can get away without it.
    - For example, the paper "Striving for Simplicity: The All Convolutional Net" proposes to discard the pooling layer in favour of architecture that only consists of repeated CONV layers.
    - To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.
    - Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs). It seems likely that future architectures will feature very few to no pooling layers.

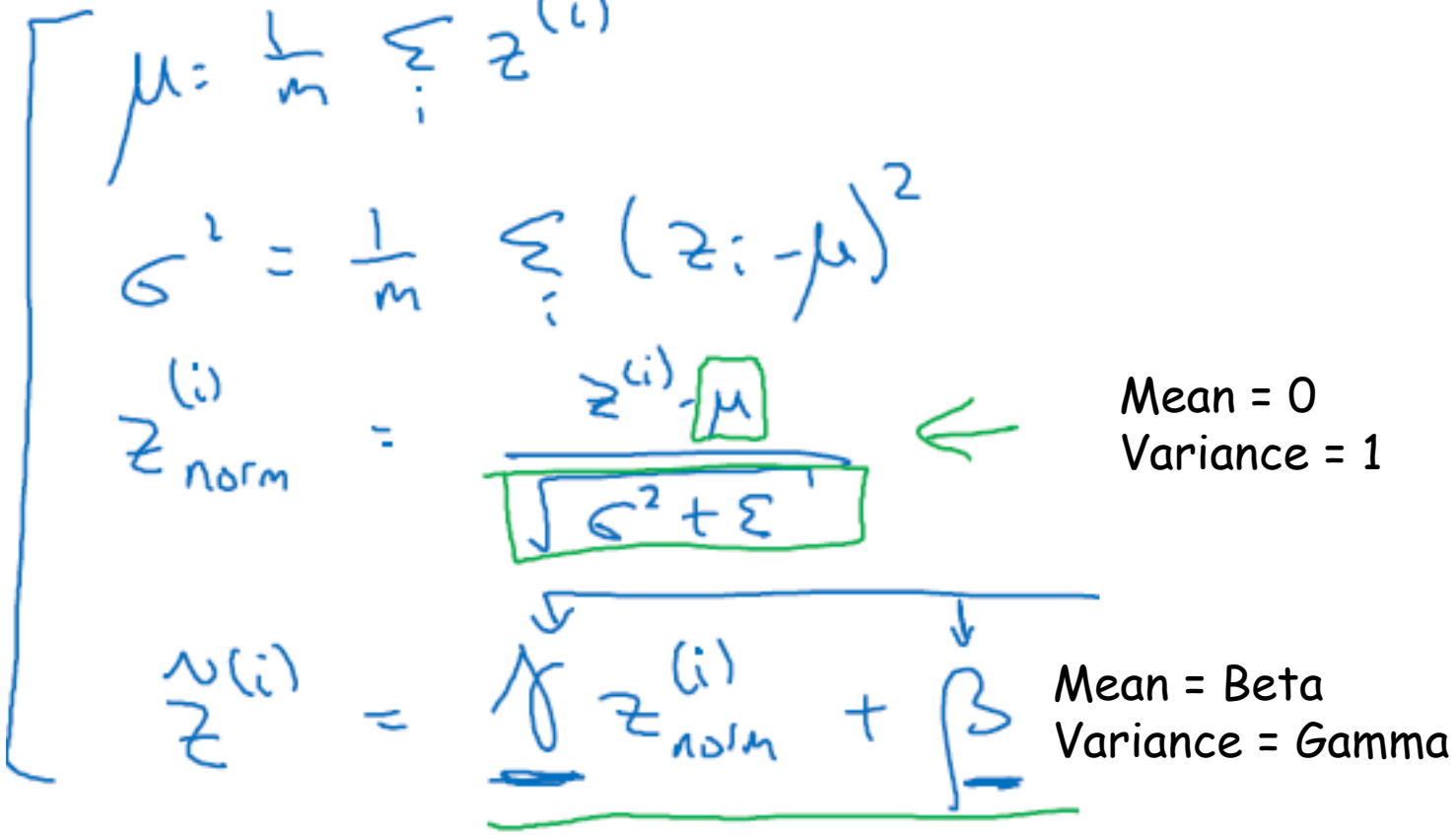
Convolutional Neural Networks

- Normalization Layer [3, Andrew Ng's Lecture on BN]
  - Many types of normalization layers have been proposed for use in ConvNet architectures, sometimes with the intentions of implementing inhibition schemes observed in the biological brain.
  - However, these layers have since fallen out of favour because in practice their contribution has been shown to be minimal, if any.



Convolutional Neural Networks

- Normalization Layer [Andrew Ng's Lecture on BN]

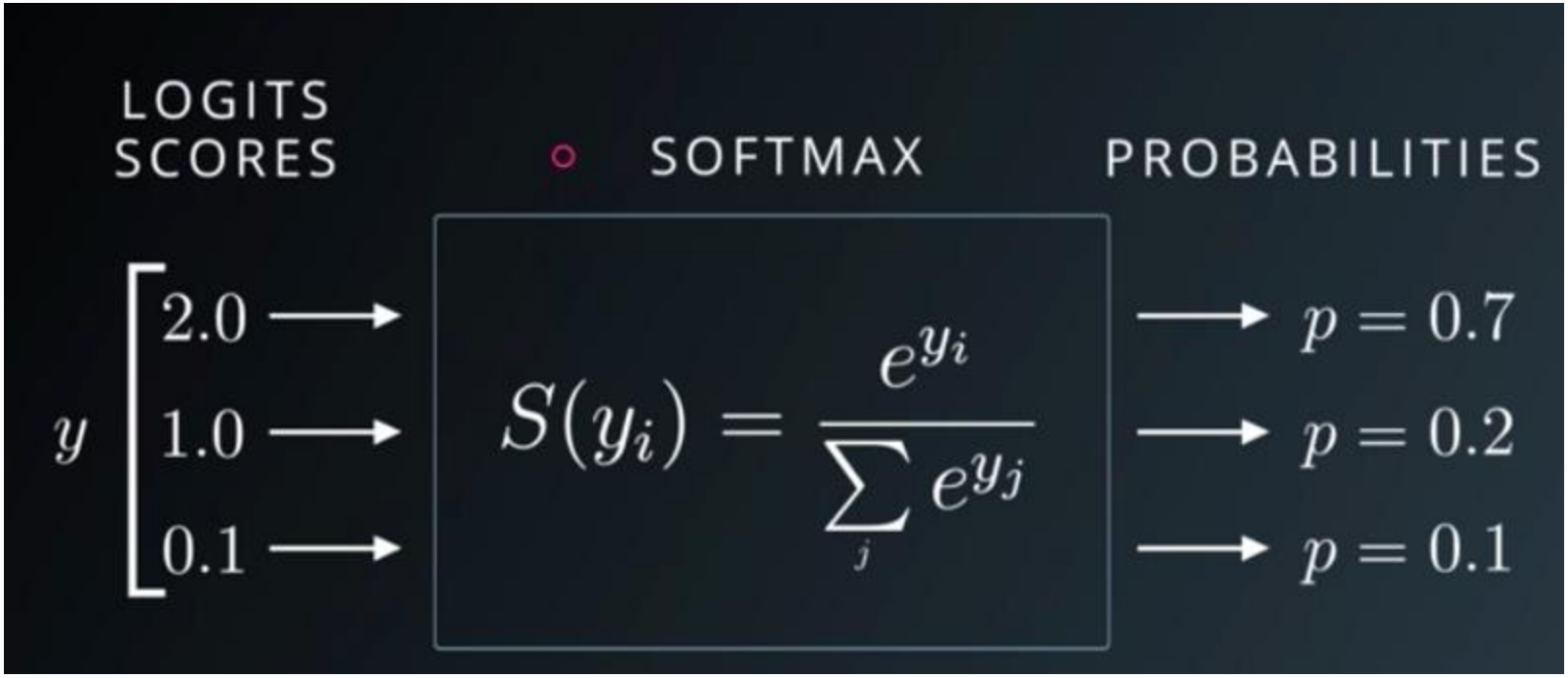


Convolutional Neural Networks

- Fully Connected Layer [3]
  - Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks.
  - Their activations can hence be computed with a matrix multiplication followed by a bias offset.

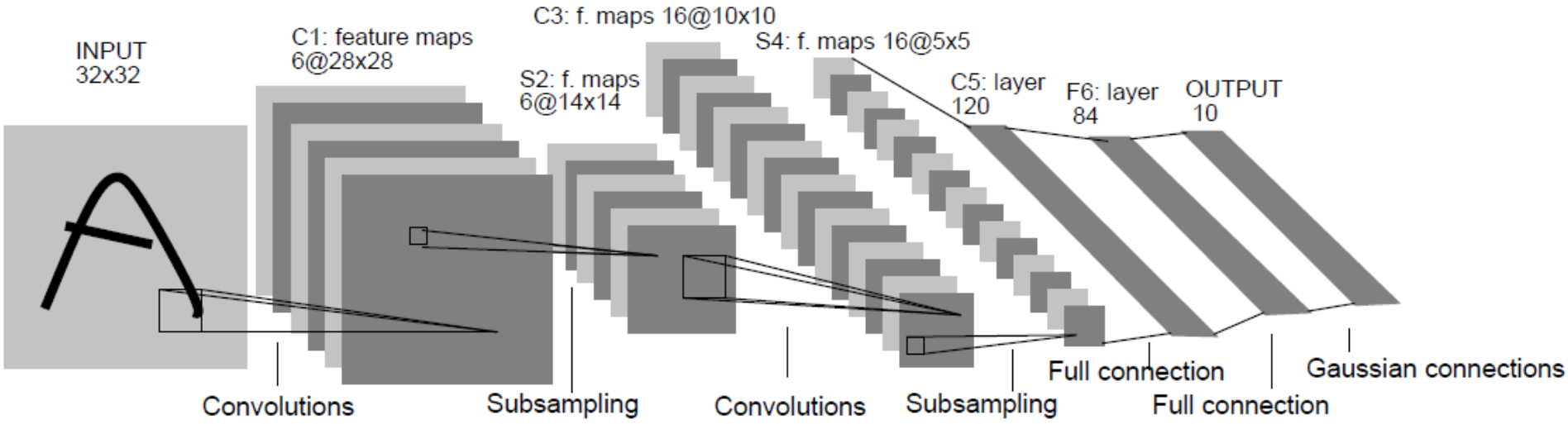
Convolutional Neural Networks

- Softmax Activation Function [5]



- Original Source: Udacity Deep Learning Slides on Softmax

Convolutional Neural Networks [1]



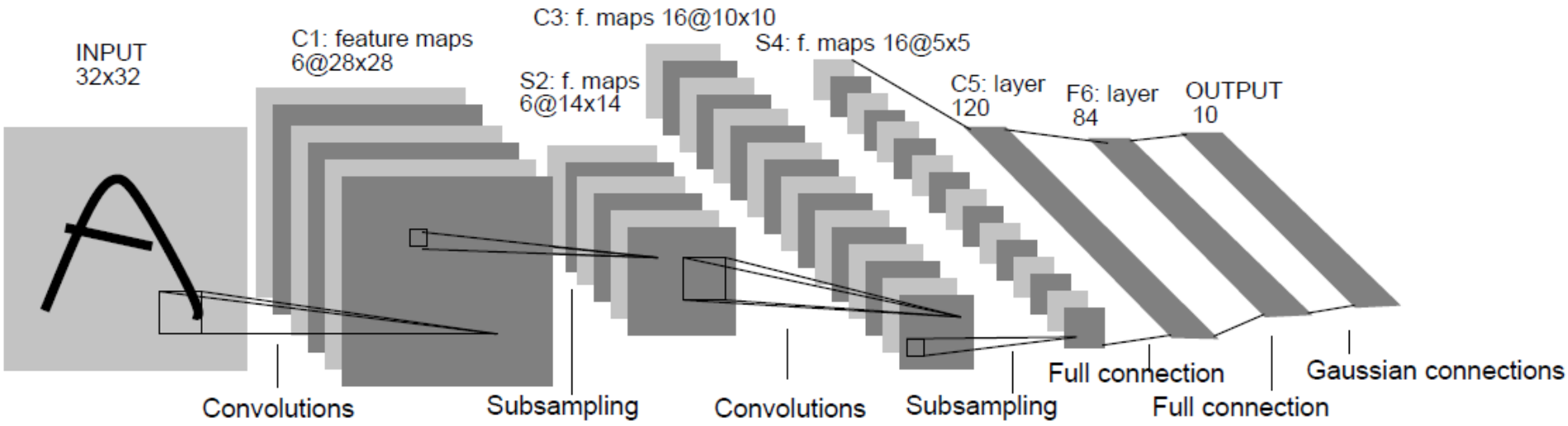
- Layer C1:

(5*5+1)*6=156 parameters to learn

Connections: 28*28*(5*5+1)*6=122304

If it was fully connected we had (32*32+1)*(28*28)*6 parameters

Convolutional Neural Networks [1]

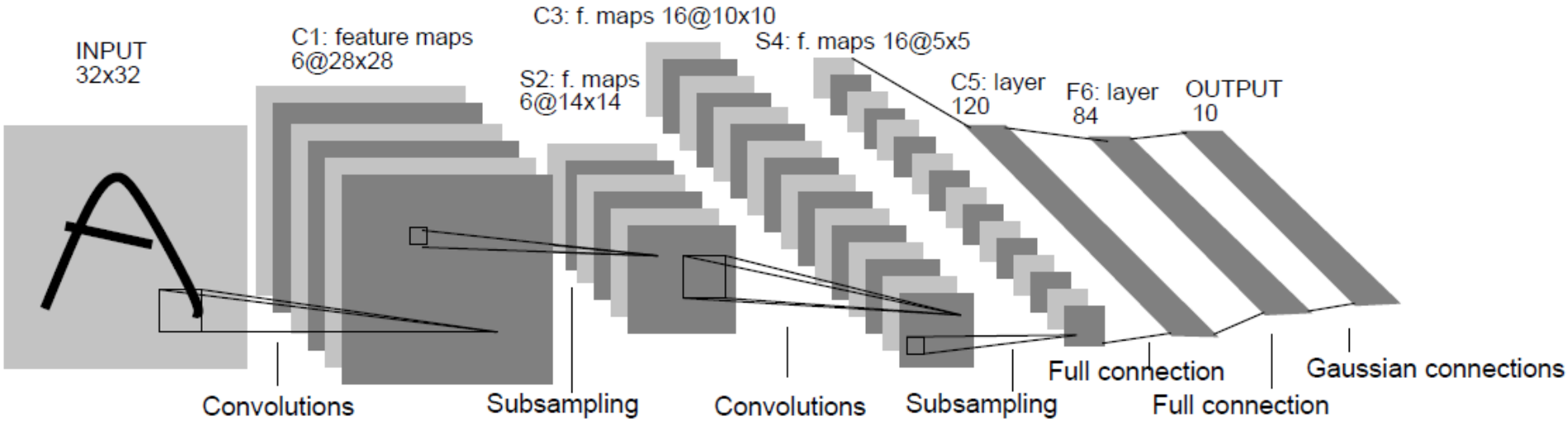


- Layer S2:

  Layer S2: 6*2=12 trainable parameters.

  Connections: 14*14*(2*2+1)*6=5880

Convolutional Neural Networks [1]



- Layer C3:

- C3: Convolutional layer with 16 feature maps of size 10x10
- Each unit in C3 is connected to several! 5x5 receptive fields at identical locations in S2

Layer C3:

1516 trainable parameters.

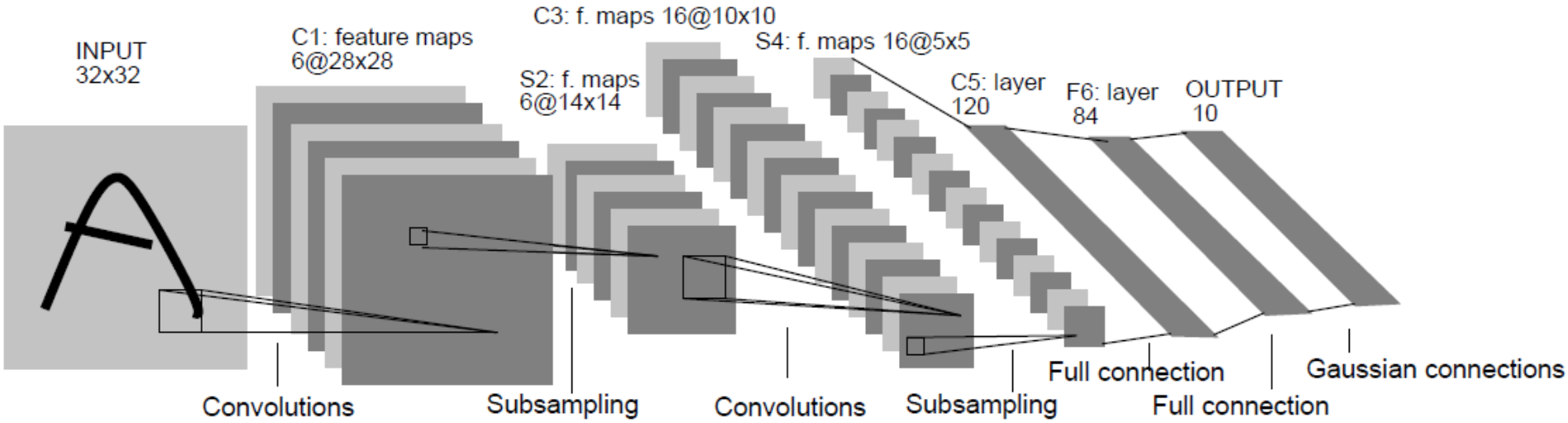
Connections: 151600

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Convolutional Neural Networks [1]

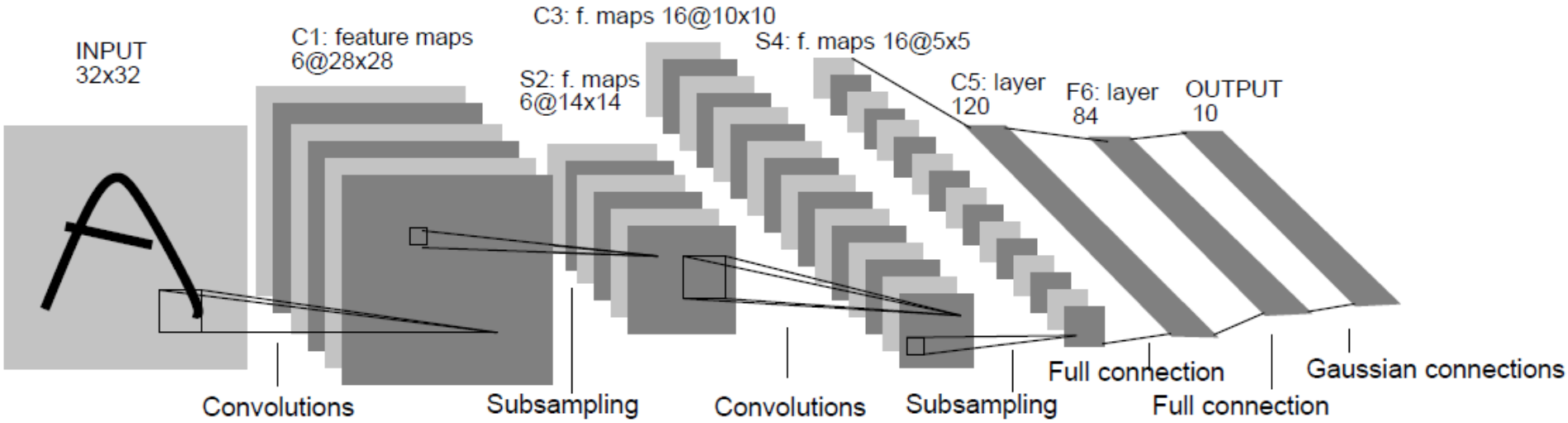


- Layer S4:
  - S4: Subsampling layer with 16 feature maps of size 5x5
  - Each unit in S4 is connected to the corresponding 2x2 receptive field at C3

Layer S4: 16*2=32 trainable parameters.

Connections: 5*5*(2*2+1)*16=2000

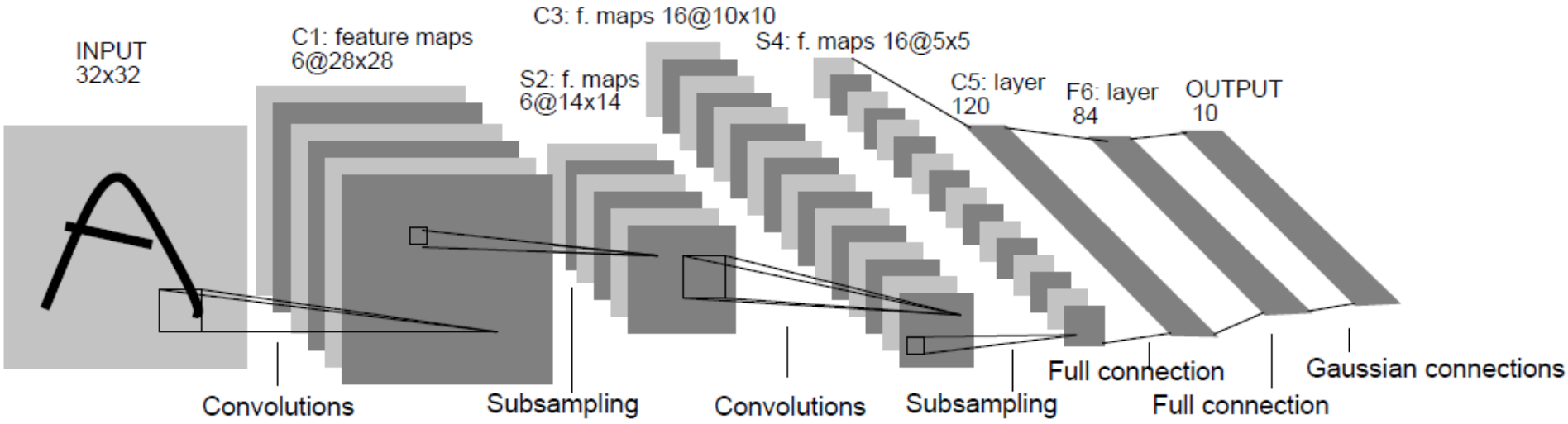
Convolutional Neural Networks [1]



- Layer C5:
  - C5: Convolutional layer with 120 feature maps of size 1x1
  - Each unit in C5 is connected to all 16 5x5 receptive fields in S4

Layer C5: 120*(16*25+1) = 48120 trainable parameters and connections
(Fully connected)

Convolutional Neural Networks [1]



- Layer F6 & Output:

  Layer F6: 84 fully connected units. 84*(120+1)=10164 trainable parameters and connections.

  Output layer: 10RBF (One for each digit)

  84=7x12, stylized image

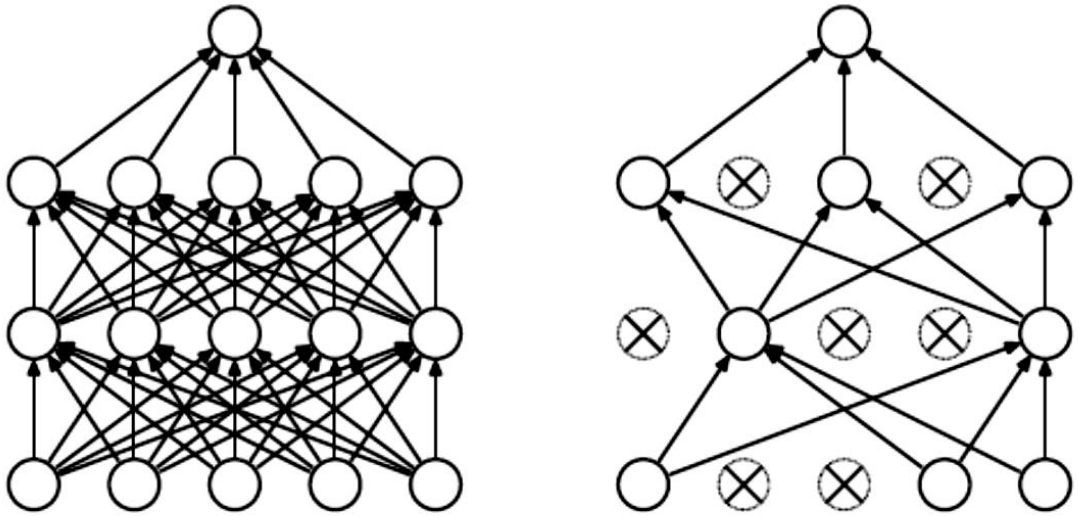
  Weight update: Backpropagation

Convolutional Neural Networks

- Dropout [2]
  - It prevents overfitting
  - Provides a way of approximately combining exponentially many different neural network architectures efficiently.

Convolutional Neural Networks

- Dropout [2]
  - The term "dropout" refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in following Figure 1.



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

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Convolutional Neural Networks

- Dropout [2]
  - The choice of which units to drop is random.
  - In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks.
  - For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5.

Convolutional Neural Networks

- Dropout [2]
  - Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout (Figure 1b).
  - A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. These networks all share weights so that the total number of parameters is still $O(n^2)$, or less.
  - **For each presentation of each training case, a new thinned network is sampled and trained.** So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.

Convolutional Neural Networks

- Dropout [2]
  - At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in practice.
  - The idea is to use a single neural net at test time without dropout.
  - The weights of this network are scaled-down versions of the trained weights. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time as shown in Figure 2.

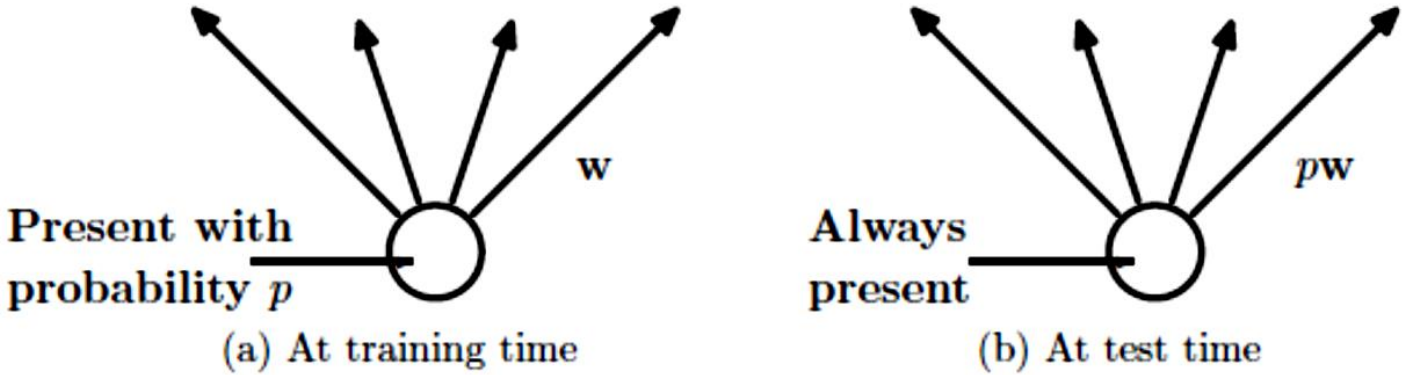


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Convolutional Neural Networks

- Dropout [2]
  - This ensures that for any hidden unit the expected output (under the distribution used to drop units at training time) is the same as the actual output at test time. By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time.

Convolutional Neural Networks

- Some Considerations [3]
  - **Prefer a stack of small filter CONV to one large receptive field CONV layer.**
    - Suppose that you stack three 3x3 CONV layers on top of each other (with non-linearities in between, of course).
    - In this arrangement, each neuron on the first CONV layer has a 3x3 view of the input volume.
    - A neuron on the second CONV layer has a 3x3 view of the first CONV layer, and hence by extension a 5x5 view of the input volume.
    - Similarly, a neuron on the third CONV layer has a 3x3 view of the 2nd CONV layer, and hence a 7x7 view of the input volume.
    - Suppose that instead of these three layers of 3x3 CONV, we only wanted to use a single CONV layer with 7x7 receptive fields.
    - These neurons would have a receptive field size of the input volume that is identical in spatial extent (7x7), but with several disadvantages.

Convolutional Neural Networks

- Some Considerations [3]
  - **Prefer a stack of small filter CONV to one large receptive field CONV layer.**
    - First, the neurons would be computing a linear function over the input, while the three stacks of CONV layers contain non-linearities that make their features more expressive.
    - Second, if we suppose that all the volumes have C channels, then it can be seen that the single 7x7 CONV layer would contain $C\times(7\times7\times C)=49C^2$ parameters, while the three 3x3 CONV layers would only contain $3\times(C\times(3\times3\times C))=27C^2$ parameters.

Convolutional Neural Networks

- Some Considerations [3]
  - **Recent Departures**
    - It should be noted that the conventional paradigm of a linear list of layers had been challenged, in Google's Inception architectures and also in Residual Networks from Microsoft Research Asia.
    - Both of these feature more intricate and different connectivity structures.

Convolutional Neural Networks

- Some Considerations [3]
  - **In practice: use whatever works best on ImageNet**
    - If you're feeling a bit of a fatigue in thinking about the **architectural decisions**, you'll be pleased to know that in 90% or more of applications you should not have to worry about these.
    - I like to summarize this point as "don't be a hero": Instead of rolling your own architecture for a problem, you should look at whatever architecture currently works best on ImageNet, download a pretrained model and finetune it on your data. You should rarely ever have to train a ConvNet from scratch or design one from scratch.

Convolutional Neural Networks

- Some Considerations [3]
  - **Layer Sizing Patterns**
    - **The input layer (that contains the image) should be divisible by 2 many times.**
      - Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.

Convolutional Neural Networks

- Some Considerations [3]
  - **Layer Sizing Patterns – The CONV Layers**
    - The conv layers should be using **small filters** (e.g. 3x3 or at most 5x5), using **a stride of S=1**, and crucially, **padding** the input volume with zeros in such way that the conv layer **does not alter the spatial dimensions** of the input.
    - That is, when F=3, then using P=1 will retain the original size of the input.
    - When F=5, P=2. For a general F, it can be seen that P=(F−1)/2 preserves the input size.
    - If you must use bigger filter sizes (such as 7x7 or so), it is only common to see this on the very first conv layer that is looking at the input image.

Convolutional Neural Networks

- Some Considerations [3]
  - **Layer Sizing Patterns – The Pool Layers**
    - The pool layers are in charge of downsampling the spatial dimensions of the input.
    - The most common setting is to use max-pooling with **2x2** receptive fields (i.e. **F=2**), and with a stride of 2 (i.e. **S=2**).
    - Note that this discards exactly **75% of the activations** in an input volume (due to downsampling by 2 in both width and height).
    - Another slightly less common setting is to use **3x3** receptive fields with a **stride of 2**, but this makes.
    - It is very uncommon to see receptive field sizes for max pooling that are larger than 3 because the pooling is then **too lossy and aggressive**. This usually leads to worse performance.

Convolutional Neural Networks

- Some Considerations [3]
  - **Layer Sizing Patterns - Reducing Sizing Headaches**
    - The scheme presented above is pleasing because all the CONV layers preserve the spatial size of their input, while the POOL layers alone are in charge of down-sampling the volumes spatially.
    - In an alternative scheme where we use strides greater than 1 or don't zero-pad the input in CONV layers, we would have to very carefully keep track of the input volumes throughout the CNN architecture and make sure that all strides and filters "work out", and that the ConvNet architecture is nicely and symmetrically wired.

Convolutional Neural Networks

- Some Considerations [3]
  - **Layer Sizing Patterns - Why use stride of 1 in CONV?**
    - Smaller strides work better in practice.
    - Additionally, as already mentioned stride 1 allows us to leave all spatial down-sampling to the POOL layers, with the CONV layers only transforming the input volume depth-wise.

Convolutional Neural Networks

- Some Considerations [3]
  - **Layer Sizing Patterns - Why use padding?**
    - In addition to the aforementioned benefit of keeping the spatial sizes constant after CONV, doing this actually improves performance.
    - If the CONV layers were to not zero-pad the inputs and only perform valid convolutions, then the size of the volumes would reduce by a small amount after each CONV, and the information **at the borders would be "washed away" too quickly**.

Convolutional Neural Networks

- Some Considerations [3]
  - **Layer Sizing Patterns - Compromising based on memory constraints.**
    - In some cases (especially early in the ConvNet architectures), the amount of memory can build up very quickly with the rules of thumb presented above.
    - For example, filtering a 224x224x3 image with three 3x3 CONV layers with 64 filters each and padding 1 would create three activation volumes of size [224x224x64].
    - This amounts to a total of about 10 million activations, or 72MB of memory (per image, for both activations and gradients).

Convolutional Neural Networks

- Some Considerations [3]
  - **Layer Sizing Patterns - Compromising based on memory constraints.**
    - Since GPUs are often bottlenecked by memory, it may be necessary to compromise.
    - In practice, people prefer to make the compromise at only the first CONV layer of the network.
    - For example, one compromise might be to use a first CONV layer with filter sizes of 7x7 and stride of 2 (as seen in a ZF net).
    - As another example, an AlexNet uses filter sizes of 11x11 and stride of 4.

Convolutional Neural Networks

- Some Considerations [3]
  - **Other Computational Considerations**
    - The largest bottleneck to be aware of when constructing ConvNet architectures is the memory bottleneck.
    - Many modern GPUs have a limit of 3/4/6GB memory, with the best GPUs having about 12GB of memory.
    - **There are three major sources of memory to keep track of:**

Convolutional Neural Networks

- Some Considerations [3]
  - **Other Computational Considerations**
    - **There are three major sources of memory to keep track of:**
      - **From the intermediate volume sizes:** These are the raw number of activations at every layer of the ConvNet, and also their gradients (of equal size).
      - Usually, most of the activations are on the earlier layers of a ConvNet (i.e. first Conv Layers).
      - These are kept around because they are needed for backpropagation, but a clever implementation that runs a ConvNet only at test time could in principle reduce this by a huge amount, by only storing the current activations at any layer and discarding the previous activations on layers below.

Convolutional Neural Networks

- Some Considerations [3]
  - **Other Computational Considerations**
    - **There are three major sources of memory to keep track of:**
      - **From the parameter sizes:** These are the numbers that hold the network parameters, their gradients during backpropagation, and commonly also a step cache if the optimization is using momentum, Adagrad, or RMSProp.
      - Therefore, the memory to store the parameter vector alone must usually be multiplied by a factor of at least 3 or so.

Convolutional Neural Networks

- Some Considerations [3]
  - **Other Computational Considerations**
    - **There are three major sources of memory to keep track of:**
      - Every ConvNet implementation has to maintain **miscellaneous memory**, such as the image data batches, perhaps their augmented versions, etc.

Convolutional Neural Networks

- Some Considerations [3]
  - **Other Computational Considerations**
    - Once you have a rough estimate of the total number of values (for activations, gradients, and misc), the number should be converted to size in GB.
    - Take the number of values, multiply by 4 to get the raw number of bytes (since every floating point is 4 bytes, or maybe by 8 for double precision), and then divide by 1024 multiple times to get the amount of memory in KB, MB, and finally GB.
    - If your network doesn't fit, a common heuristic to "make it fit" is to decrease the batch size, since most of the memory is usually consumed by the activations.

ImageNet Dataset

- ImageNet is a dataset of over **15 million labelled** high-resolution images belonging to roughly **22,000 categories**.

ImageNet Dataset

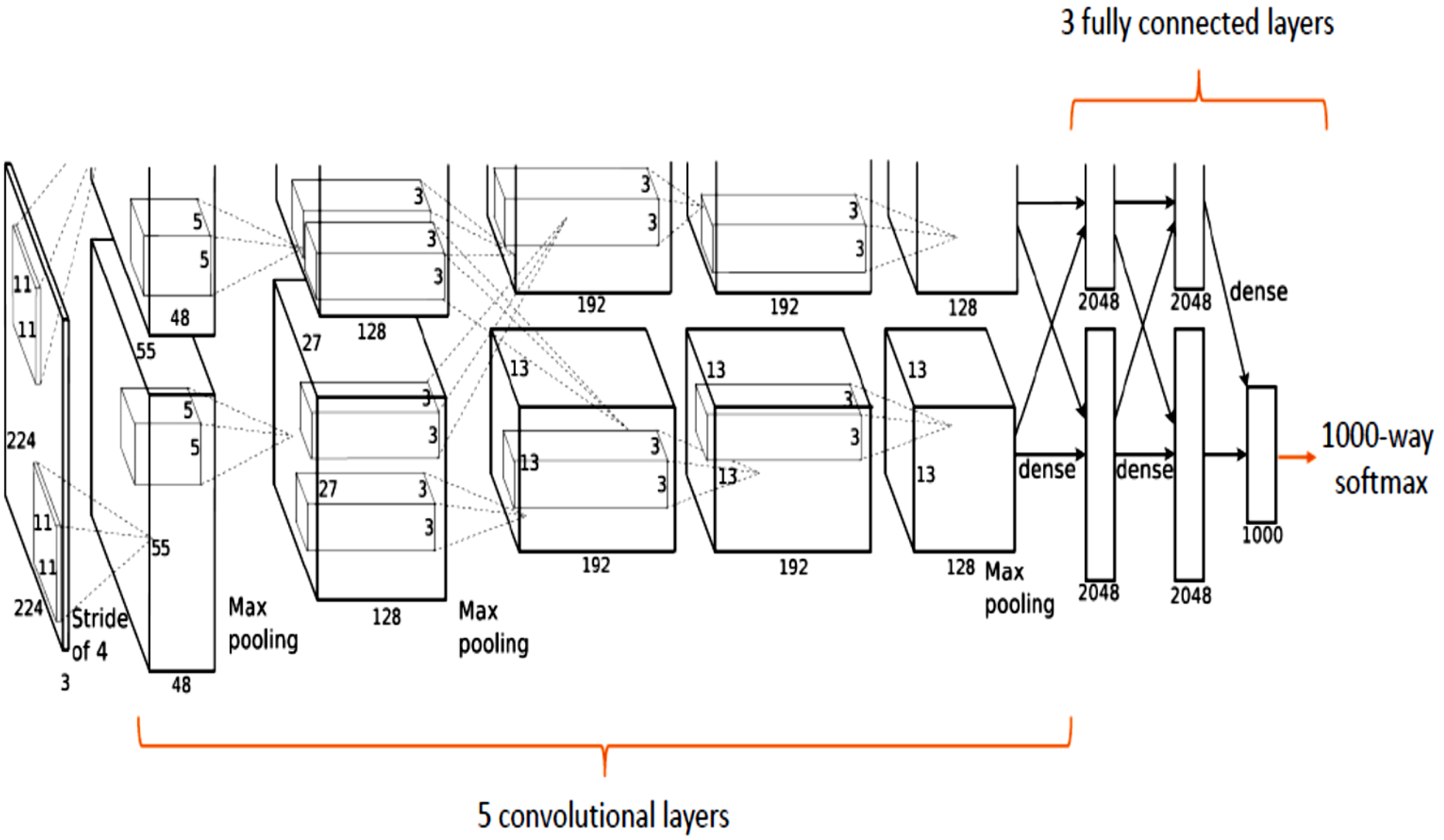
- **ILSVRC-2010 is the only version of ILSVRC for which the test set labels are available.**

- On ImageNet, it is customary to report two error rates: **top-1 and top-5**, where the top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the model.

ALEXNET – Preprocessing

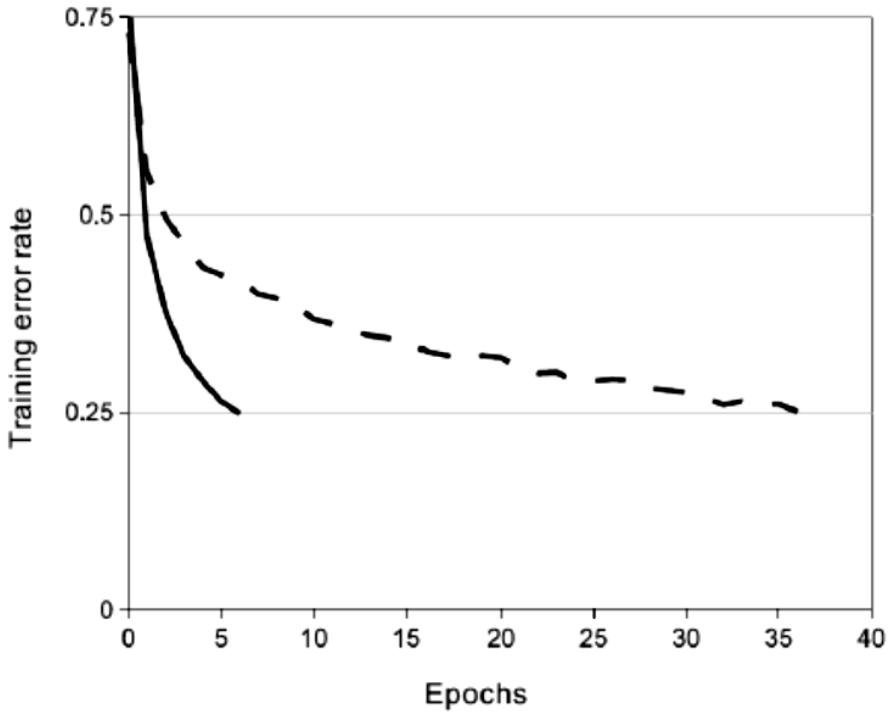
- ImageNet consists of variable-resolution images, while ALEXNET system requires a constant input dimensionality.
- Therefore, authors down-sampled the images to a fixed resolution of **256 x 256**.
- Given a rectangular image, they first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256x256 patch from the resulting image.
- They did not pre-process the images in any other way, except for **subtracting the mean activity over the training set** from each pixel.
- So, they trained our network on the (centered) raw RGB values of the pixels.

ALEXNET - Architecture



ALEXNET - Architecture

- ReLU Nonlinearity



- Dataset: CIFAR-10
- Experiment:

  CNN (4 layers) + ReLUs (solid line)

  vs.

  CNN (4 layers) + tanh (dashed line)

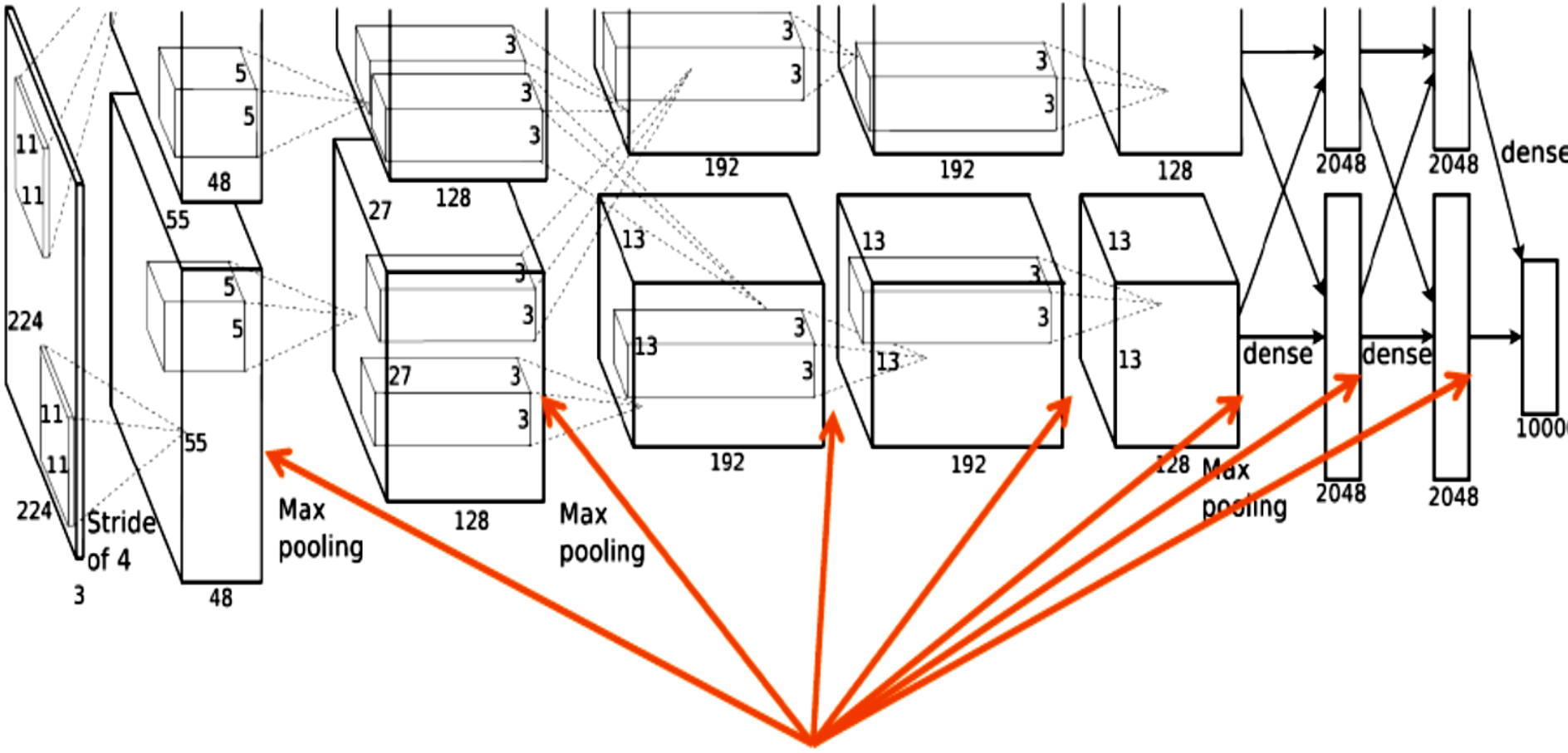
  → ReLUs six times faster

ALEXNET - Architecture

- ReLU Nonlinearity
  - Trains several times faster than their equivalents with tanh units. (Inspired by the work carried out in: V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In Proc. 27th International Conference on Machine Learning, 2010.)

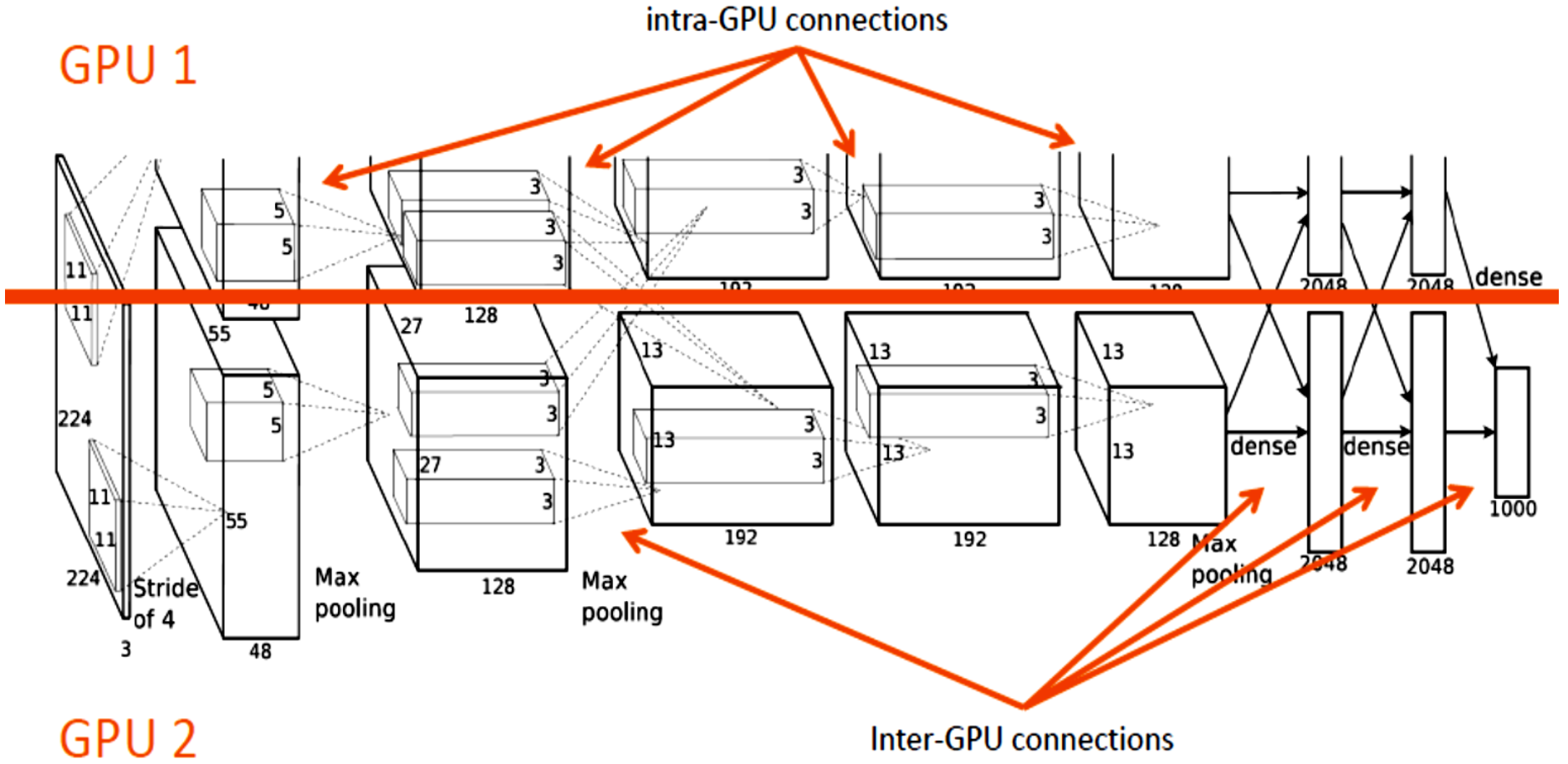
ALEXNET - Architecture

- ReLU Nonlinearity



ALEXNET - Architecture

- Training on Multiple GPUs



ALEXNET - Architecture

- 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 they 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.

ALEXNET - Architecture

- Training on Multiple GPUs
  - 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.

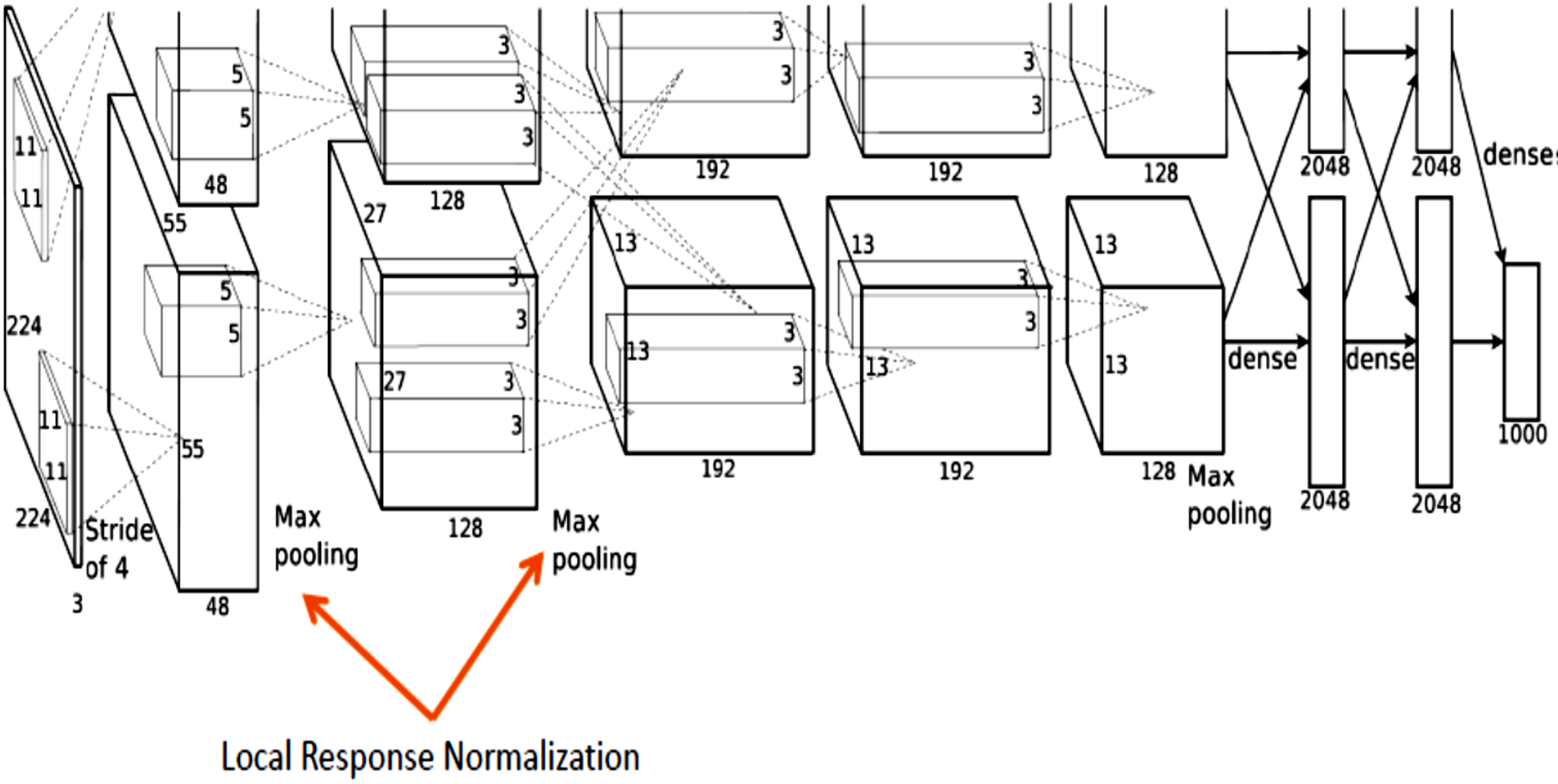
ALEXNET - Architecture

- Training on Multiple GPUs
  - **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-connected layers which follow). Therefore this comparison is biased in favour of the one-GPU net, since it is bigger than "half the size" of the two-GPU net.

ALEXNET - Architecture

- Local Response Normalization

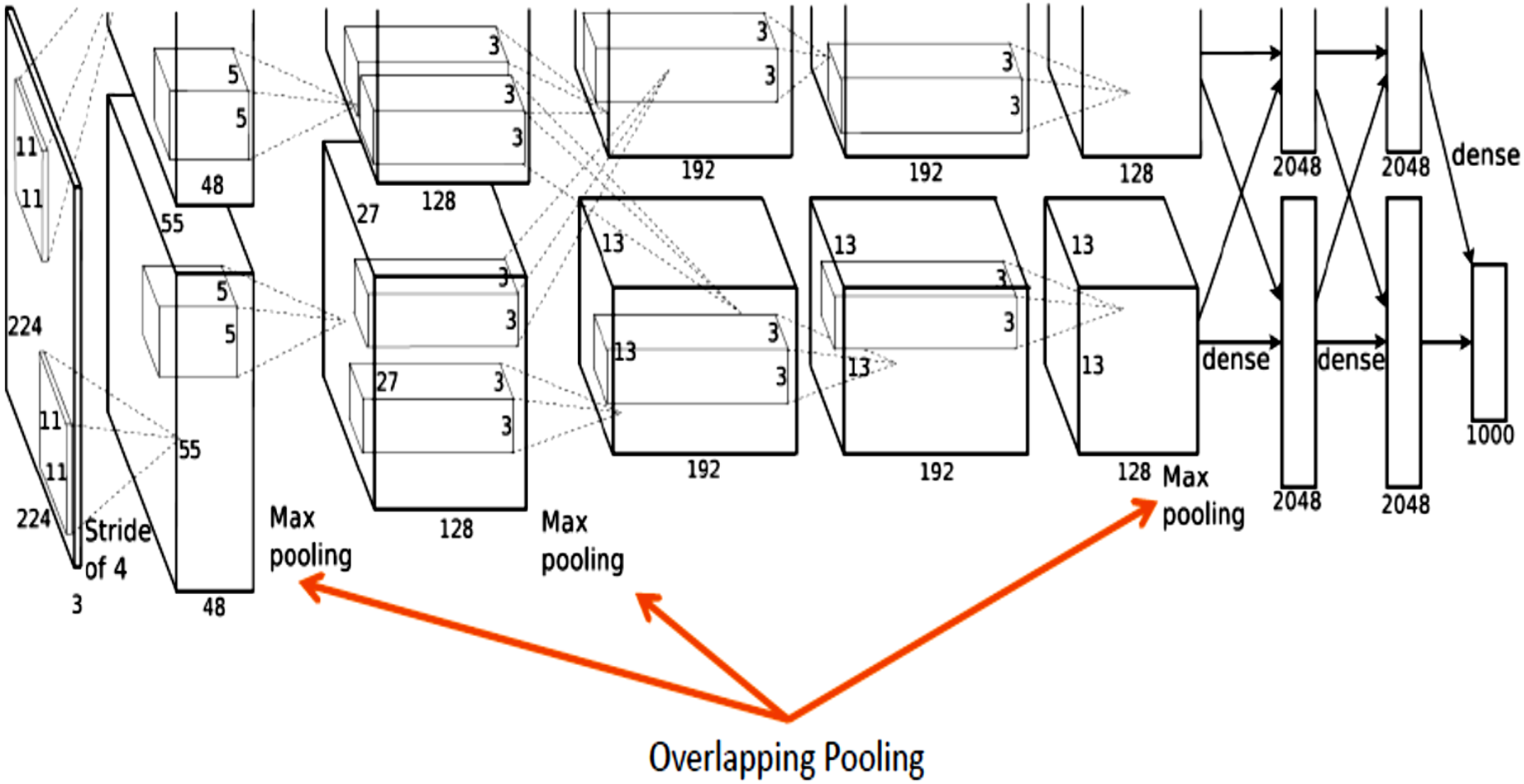


ALEXNET - Architecture

- Local Response Normalization
  - Response normalization reduced their top-1 and top-5 error rates by 1.4% and 1.2%, respectively.
  - They also verified the effectiveness of this scheme on the CIFAR-10 dataset: a four-layer CNN achieved a 13% test error rate without normalization and 11% with normalization

ALEXNET - Architecture

- Overlapping Pooling



ALEXNET - Architecture

- Overlapping Pooling
  - Throughout their network, they used s = 2 and z = 3.
  - This scheme reduced the top-1 and top-5 error rates by 0.4% and 0.3%, respectively, as compared with the non-overlapping scheme with s = 2; z = 2.

ALEXNET - Architecture

- Overall Architecture

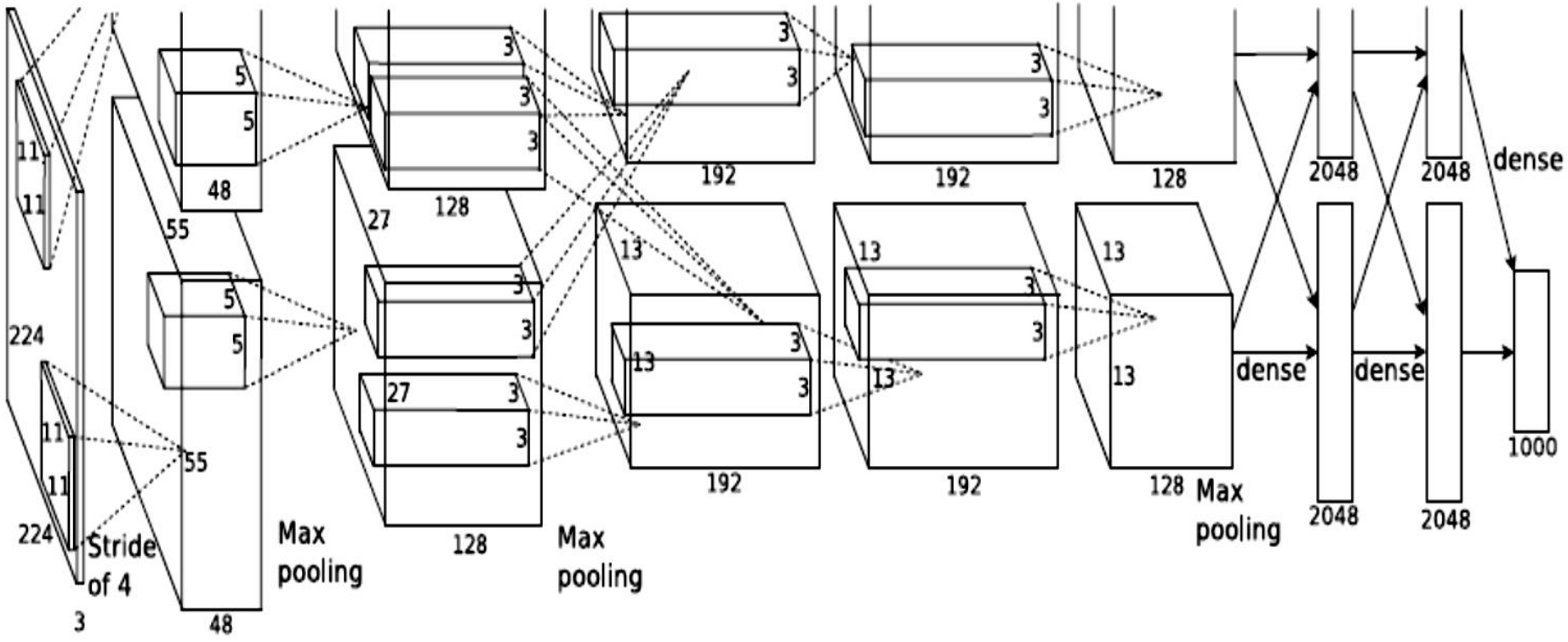
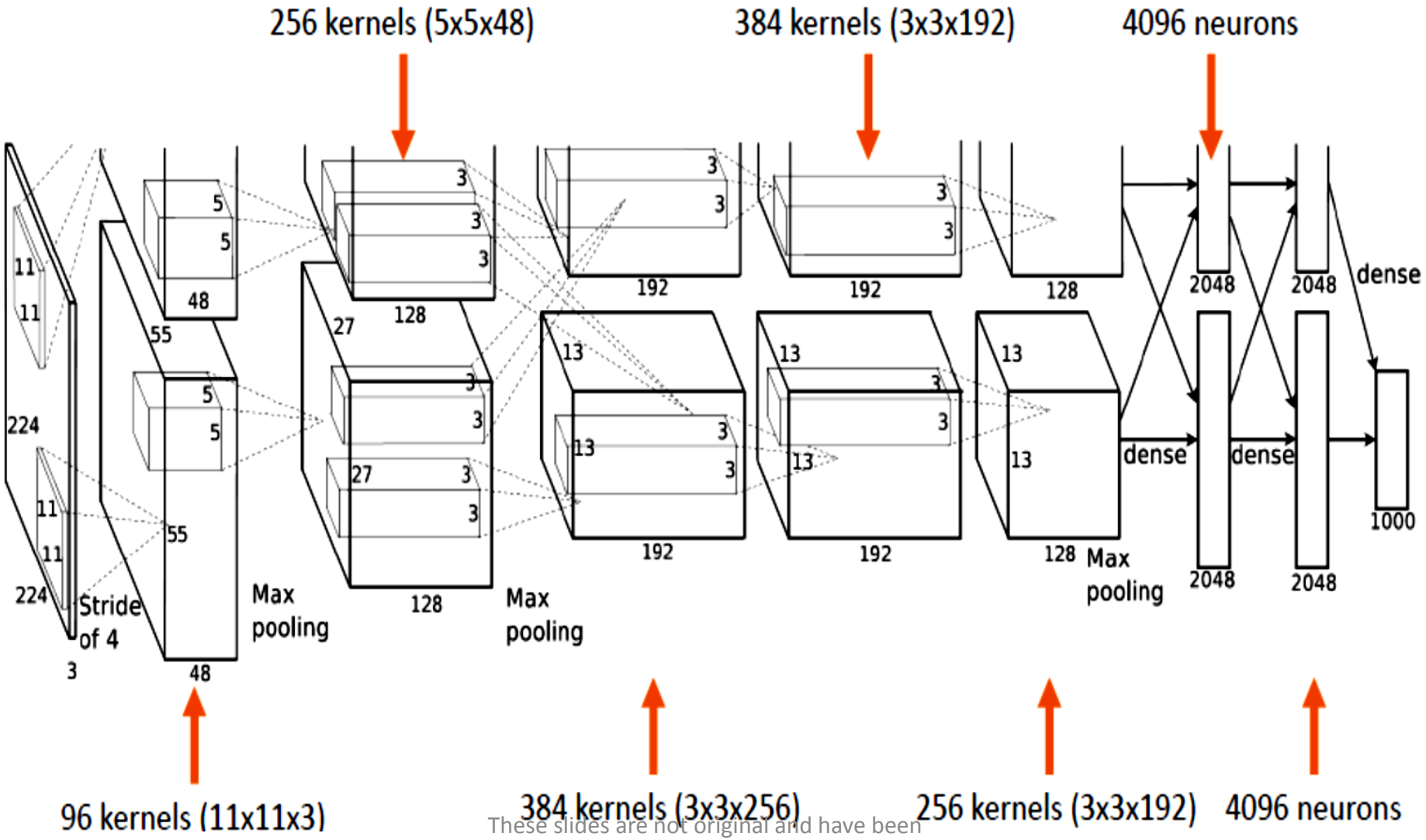


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

ALEXNET - Architecture

- Overall Architecture



ALEXNET - Architecture

- Overall Architecture

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600
96 * 55 * 55						
Max Pooling	3 * 3		2			
96 * 27 * 27						
Norm						
Conv2 + Relu	5 * 5	256	1	2	(5 * 5 * 96 + 1) * 256=614656	(5 * 5 * 96 + 1) * 256 * 27 * 27=448084224
256 * 27 * 27						
Max Pooling	3 * 3		2			
256 * 13 * 13						
Norm						
Conv3 + Relu	3 * 3	384	1	1	(3 * 3 * 256 + 1) * 384=885120	(3 * 3 * 256 + 1) * 384 * 13 * 13=149585280
384 * 13 * 13						
Conv4 + Relu	3 * 3	384	1	1	(3 * 3 * 384 + 1) * 384=1327488	(3 * 3 * 384 + 1) * 384 * 13 * 13=224345472
384 * 13 * 13						

ALEXNET - Architecture

- Overall Architecture

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
384 * 13 * 13						
Conv5 + Relu	3 * 3	256	1	1	(3 * 3 * 384 + 1) * 256=884992	(3 * 3 * 384 + 1) * 256 * 13 * 13=149563648
256 * 13 * 13						
Max Pooling	3 * 3		2			
256 * 6 * 6						
Dropout (rate 0.5)						
FC6 + Relu					256 * 6 * 6 * 4096=37748736	256 * 6 * 6 * 4096=37748736
4096						
Dropout (rate 0.5)						
FC7 + Relu					4096 * 4096=16777216	4096 * 4096=16777216
4096						
FC8 + Relu					4096 * 1000=4096000	4096 * 1000=4096000
1000 classes						
Overall					62369152=62.3 million	1135906176=1.1 billion
Conv VS FC					Conv:3.7million (6%) , FC: 58.6 million (94%)	Conv: 1.08 billion (95%) , FC: 58.6 million (5%)

ALEXNET – Reducing Overfitting

- ALEXNET has 60 million parameters.
- It was difficult to learn so many parameters without considerable overfitting.
- Below are the two primary ways in which they combat overfitting.
  - Data Augmentation (2 ways)
  - Dropout

ALEXNET – Reducing Overfitting

- The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations.

- They employed 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 their implementation, the transformed images were generated in Python code on the CPU while the GPU was training on the previous batch of images.

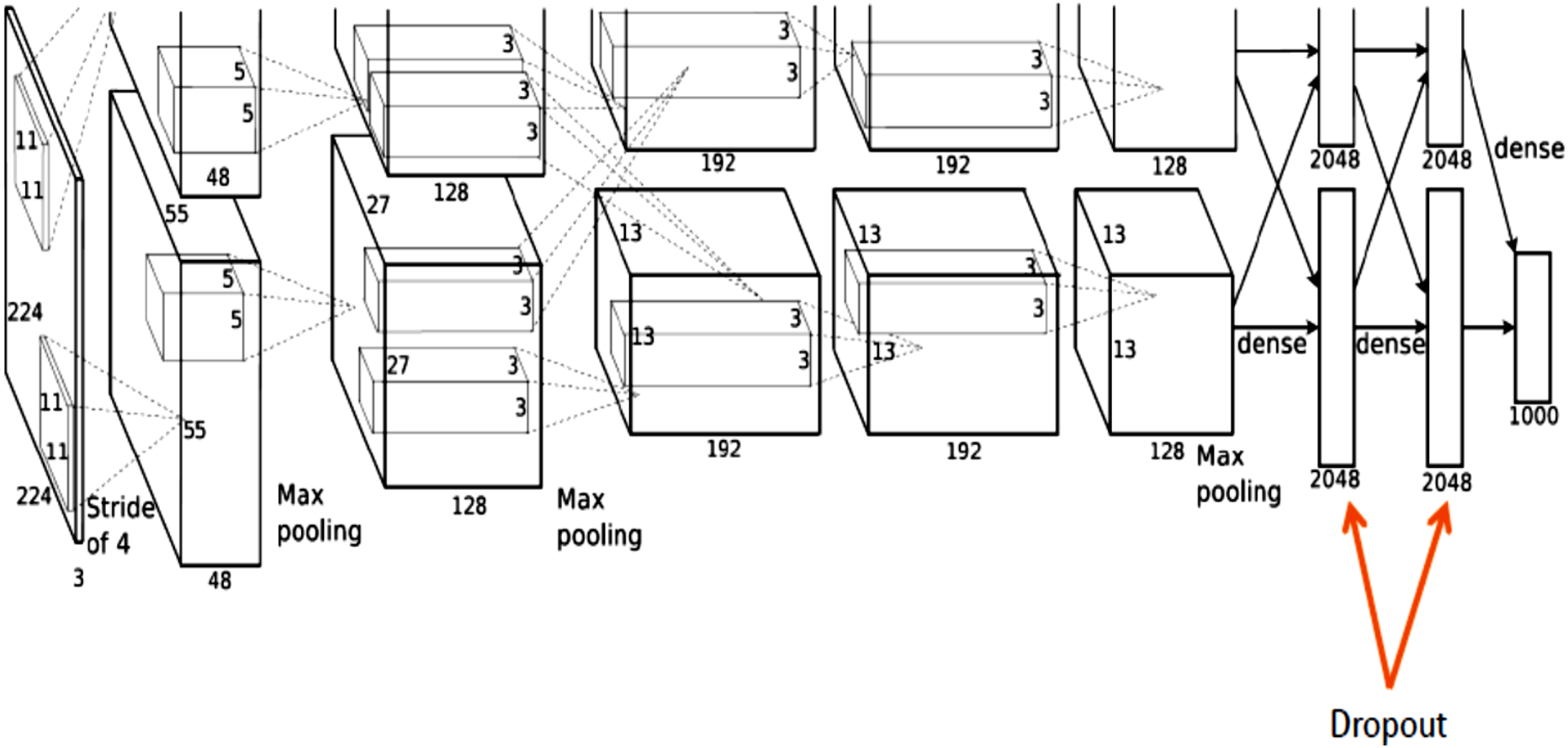
- So, these data augmentation schemes are, in effect, computationally free.

ALEXNET – Reducing Overfitting

- The first form of data augmentation consists of generating image translations and horizontal reflections.
- We do this by extracting random 224 x 224 patches (and their horizontal reflections) from the 256 x 256 images and training our network on these extracted patches.
- This increases the size of training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent.
- Without this scheme, the network suffered from substantial overfitting, which would have forced them to use much smaller networks.
- At test time, the network makes a prediction by extracting five 224 x 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.

ALEXNET – Reducing Overfitting

- Dropout



ALEXNET – Details of Learning

- Dropout
  - They uses dropout in the first two fully-connected layers.
  - Without dropout, their network exhibits substantial overfitting.
  - Dropout roughly doubles the number of iterations required to converge.

ALEXNET – Details of Learning

- Details of Learning

Stochastic Gradient Descent

- Training process
  - Minimizing the cross-entropy loss function:

$$L(w) = \sum_{i=1}^{N} \sum_{c=1}^{1000} -y_{ic} \log f_c(x_i) + \epsilon \|w\|_2^2$$

predicted probability of class c for image x

indicator that example i has label c

ALEXNET – Details of Learning

➢ Details of Learning

Stochastic Gradient Descent

- SGD with a batch size of 128
- Learning rate initialized at 0.01; divided by 10 if validation error rate stopped improving
- Update rule for weight w:

weight decay

learning rate

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

momentum

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

Gradient of Loss

- ~ 90 cycles → five to six days on two NVIDIA GTX 580 3GB GPUs

ALEXNET – Details of Learning

- Details of Learning
  - 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.

ALEXNET – Details of Learning

- Details of Learning
  - 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 reduced three times prior to termination.
  - We trained the network for roughly 90 cycles through the training set of 1.2 million images, which took five to six days on two NVIDIA GTX 580 3GB GPUs.

ALEXNET – Details of Learning

- Results

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

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

ALEXNET – Details of Learning

- Results

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

ALEXNET – Details of Learning

- Results
  - Finally, we also report our error rates on the Fall 2009 version of ImageNet with 10,184 categories and 8.9 million images.
  - On this dataset we follow the convention in the literature of using half of the images for training and half for testing.
  - Since there is no established test set, our split necessarily differs from the splits used by previous authors, but this does not affect the results appreciably.
  - Our top-1 and top-5 error rates on this dataset are 67.4% and 40.9%, attained by the net described above but with an additional, sixth convolutional layer over the last pooling layer.
  - The best published results on this dataset are 78.1% and 60.9%

ALEXNET-Summary

- 60 million parameters
- 650,000 neurons
- Five convolutional layers, some of which are followed by max-pooling layers
- Three fully-connected layers
- A final 1000-way softmax
- Winner of ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

ALEXNET

- Some Considerations [1]

References

1. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

2. Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

3. http://cs231n.github.io/convolutional-networks/

4. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

5. https://medium.com/@uniqtech/understand-the-softmax-function-in-minutes-f3a59641e86d

Disclaimer

- These slides are not original and have been prepared from various sources for teaching purpose.