Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton.

"Imagenet classification with deep convolutional neural networks."

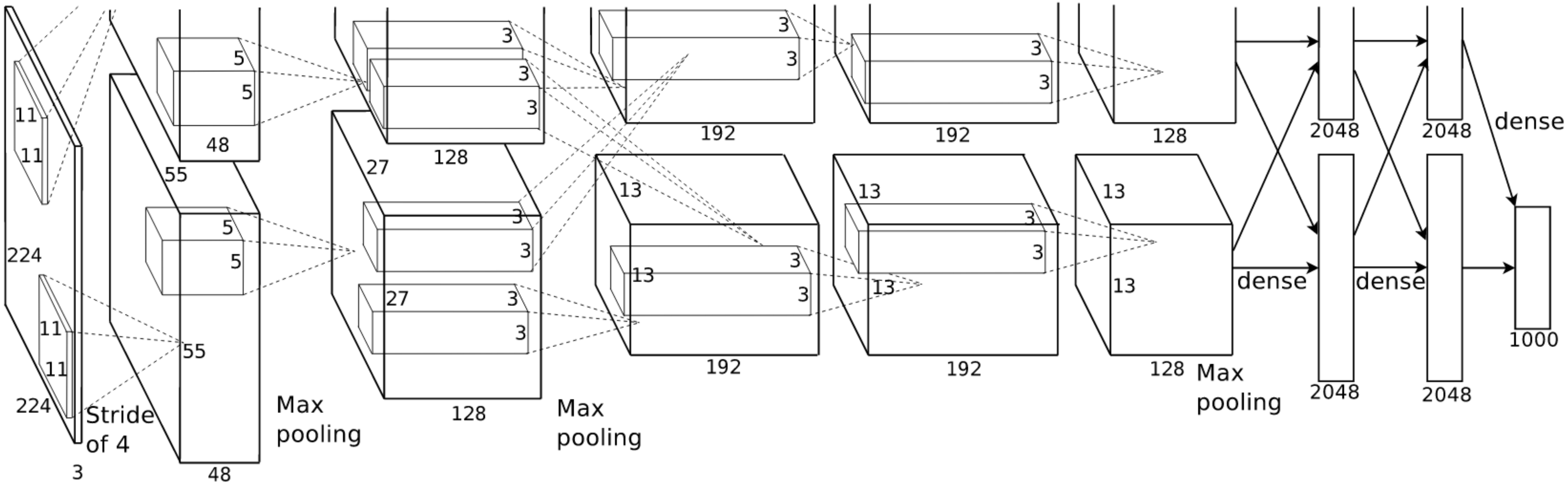
Advances in neural information processing systems. 2012.

박희경

AlexNet

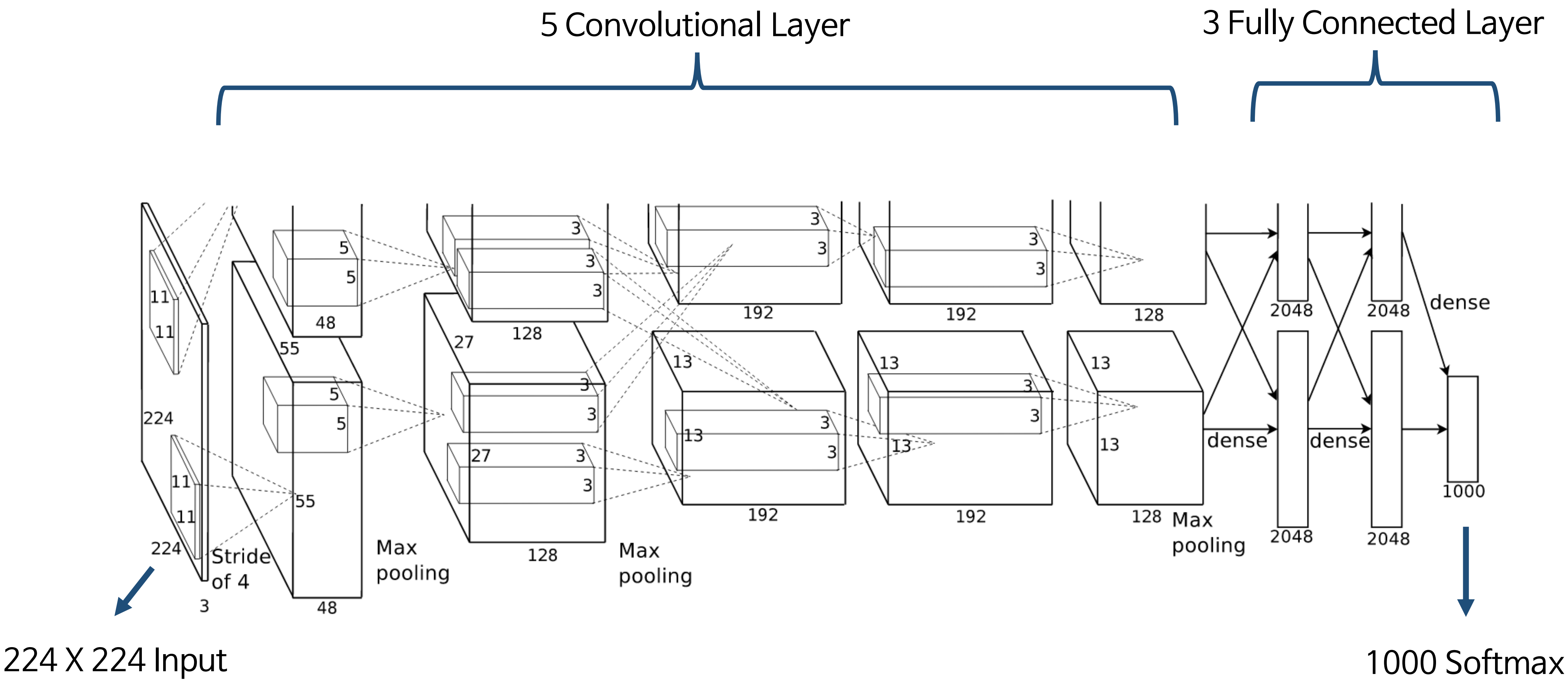
Introduction

- The AlexNet (The ILSVRC 2012 winner) was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up .
  (top 5 error of 16% compared to runner-up with 26% error)



AlexNet

Outline

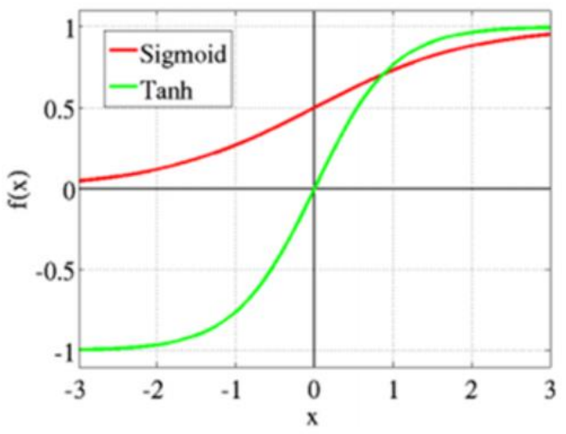


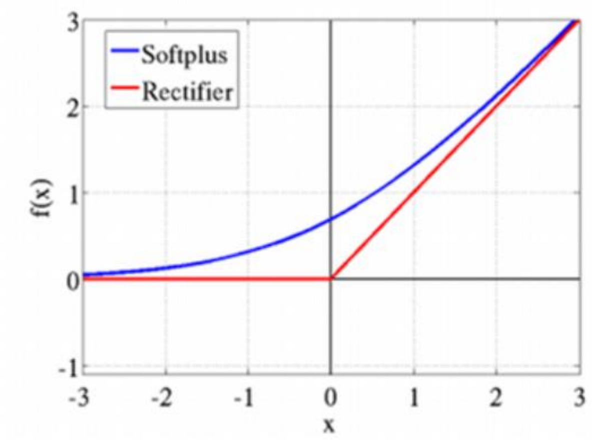
AlexNet

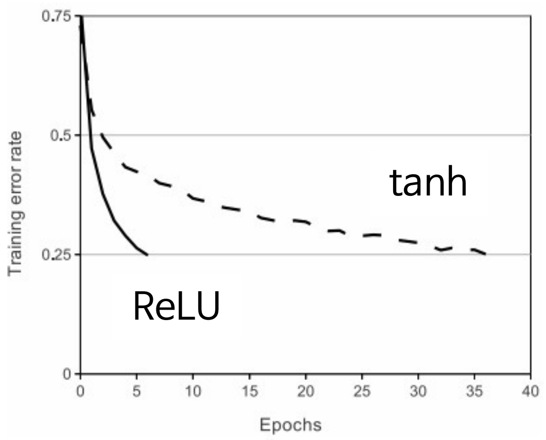
ReLU Nonlinearity

- Sigmoid, Tanh : Saturating nonlinearity
- ReLU (Rectified Linear Units) : Non-saturating nonlinearity

→ DNNs (Deep convolutional neural network) with ReLU train several times faster than their equivalents with tanh, sigmoid units.





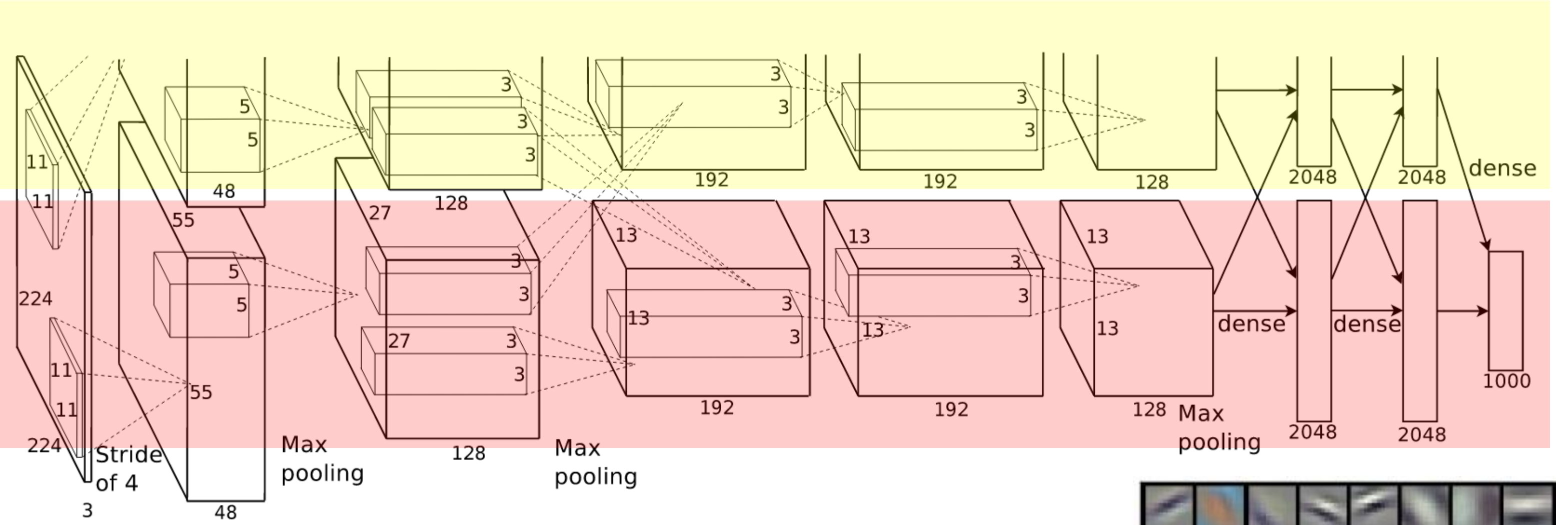


A four layer convolutional neural network with ReLU reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons.

AlexNet

Training on Multiple GPUs

Training on two GPUs in parallel reduces top-1and top-5 error rates by 1.7% and 1.2% as compared with net with half convolution layer in one GPU.

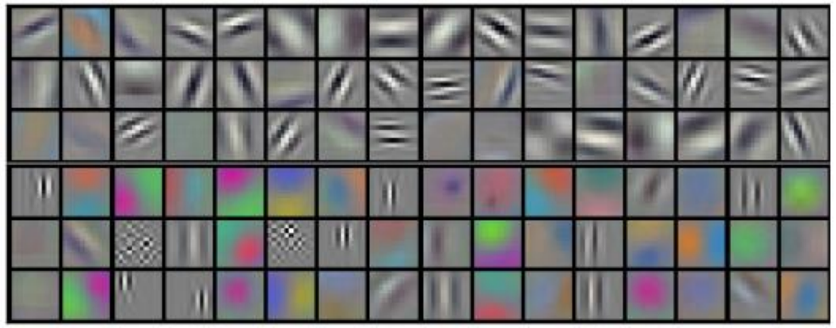


(Intra) GPU 1

(Inter) GPU 2

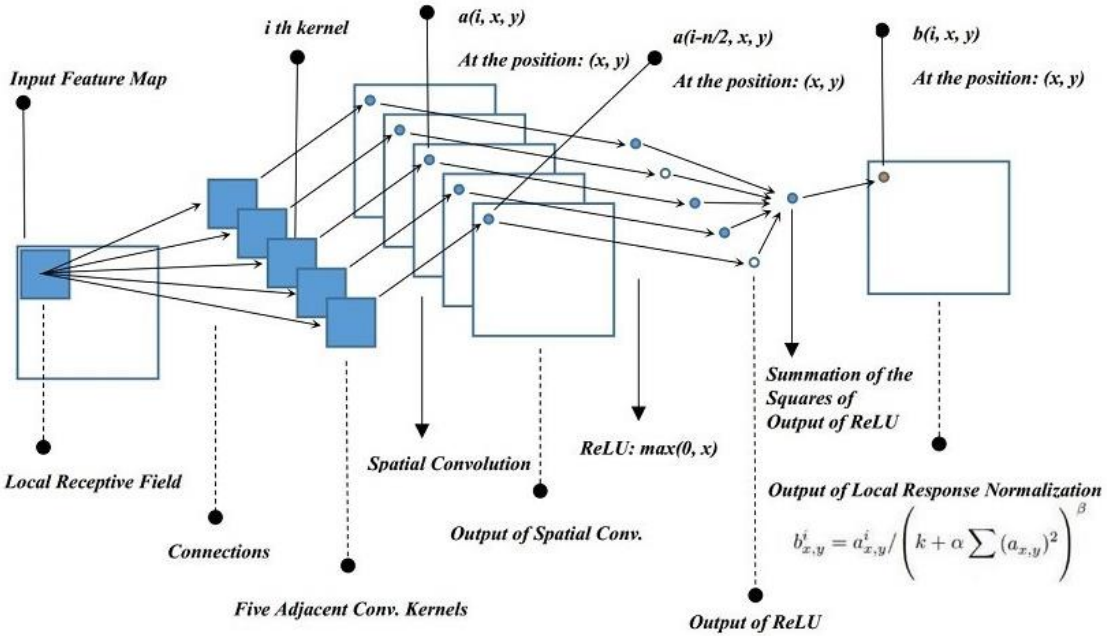
GPU 1 : color-agnostic

GPU 2 : color-specific



AlexNet

Local Response Normalization



2nd, 3rd layer

Conv

↓

ReLU

↓

Response Normalization

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

↓

Max pooling

- Response Normalization reduces top-1and top-5 error rate by 1.4% and 1.2%.
- Local Response normalization would be more correctly termed “brightness normalization”, since we do not subtract the mean activity

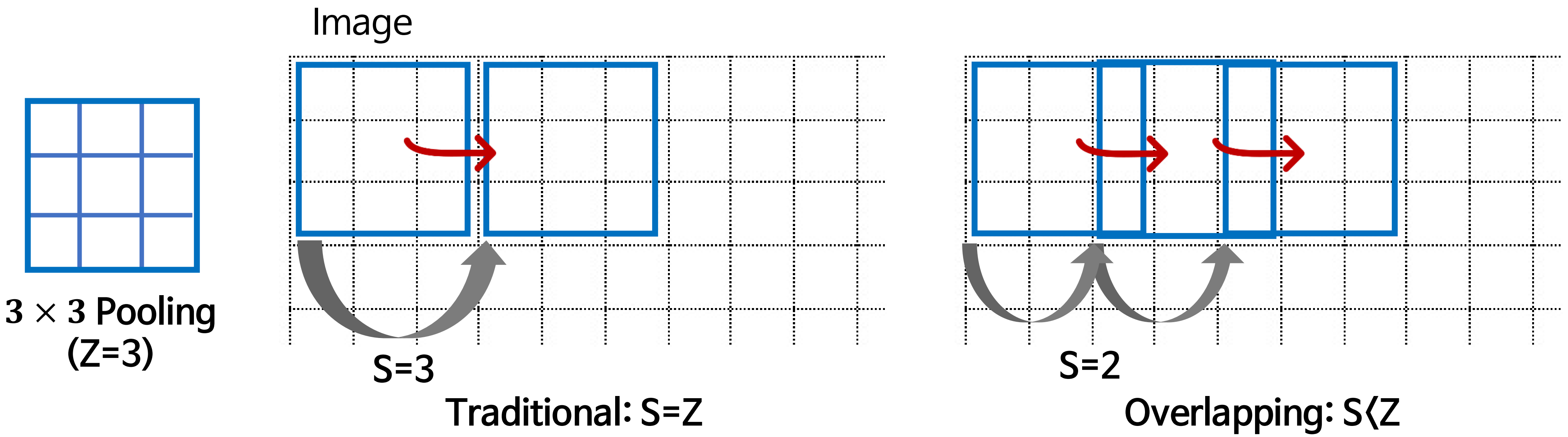
AlexNet

Overlapping Pooling

- Pooling resize and sampling the feature.

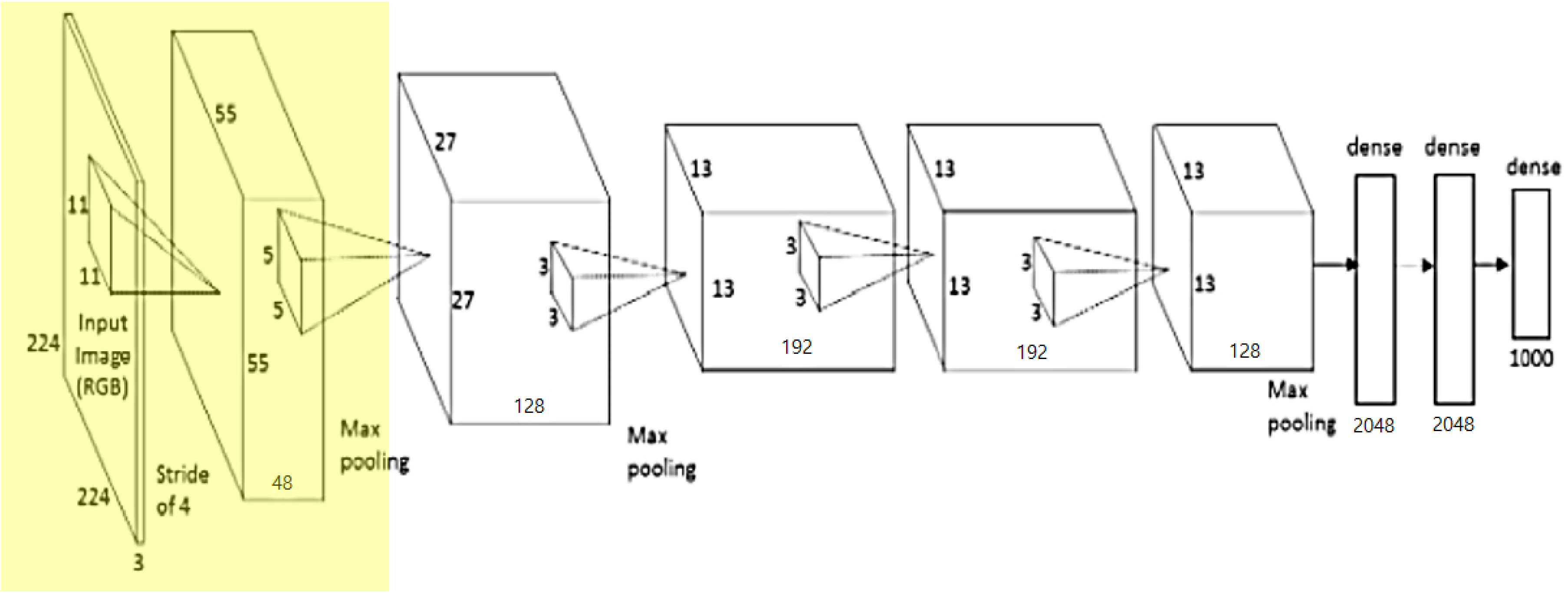
- Overlapping pooling find it slightly more difficult to overfit.

- Overlapping pooling (s=2, z=3) reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively, as compered with the non-overlapping pooling (s=2, z=2)



AlexNet

Overall Architecture

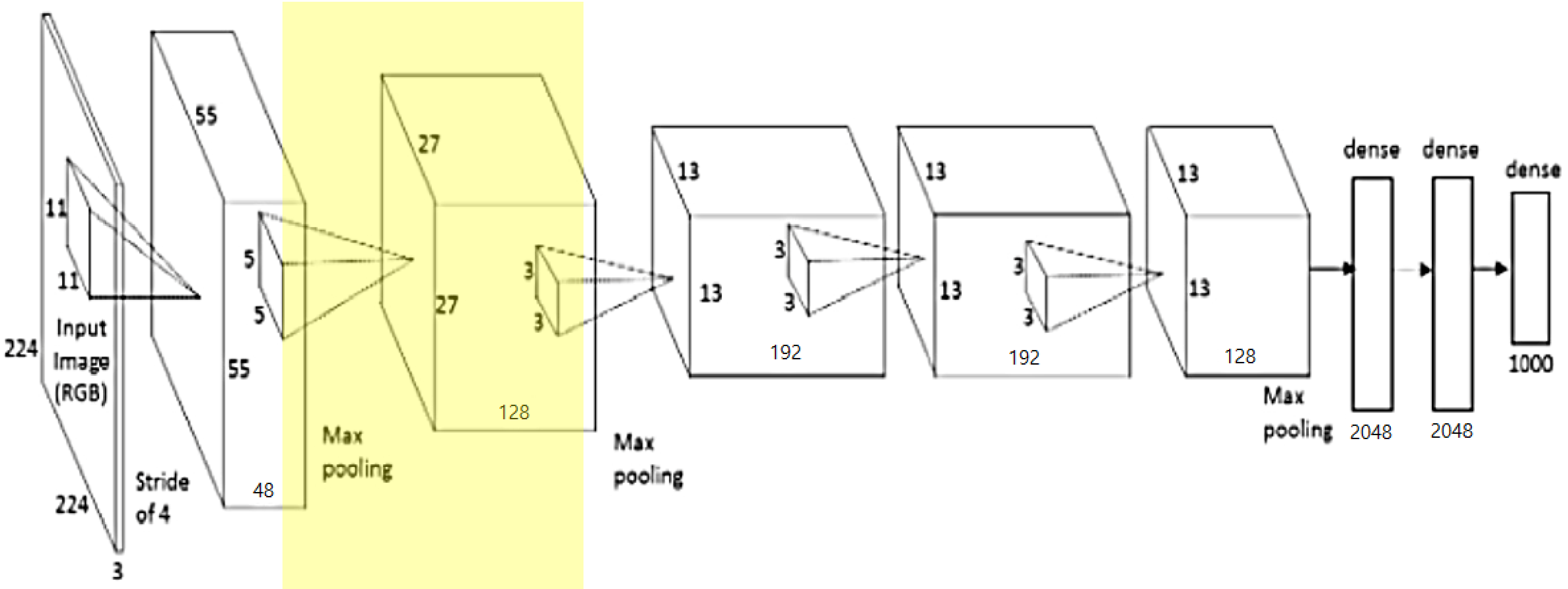


- Input image: $224 \times 224 \times 3$ (padding 3)
- 1st Conv. Kernel: $11 \times 11 \times 3, @\ 96$
- Stride 4
- Max pooling

- Whole Feature map: 96
- $55 \times 55 \times 96 = 290{,}400$ Neurons
- Parameter for each kernel: $(11 \times 11 \times 3) + 1 = 364$
- 1st layer connection: $290{,}400 \times 364 = 105{,}750{,}600$

AlexNet

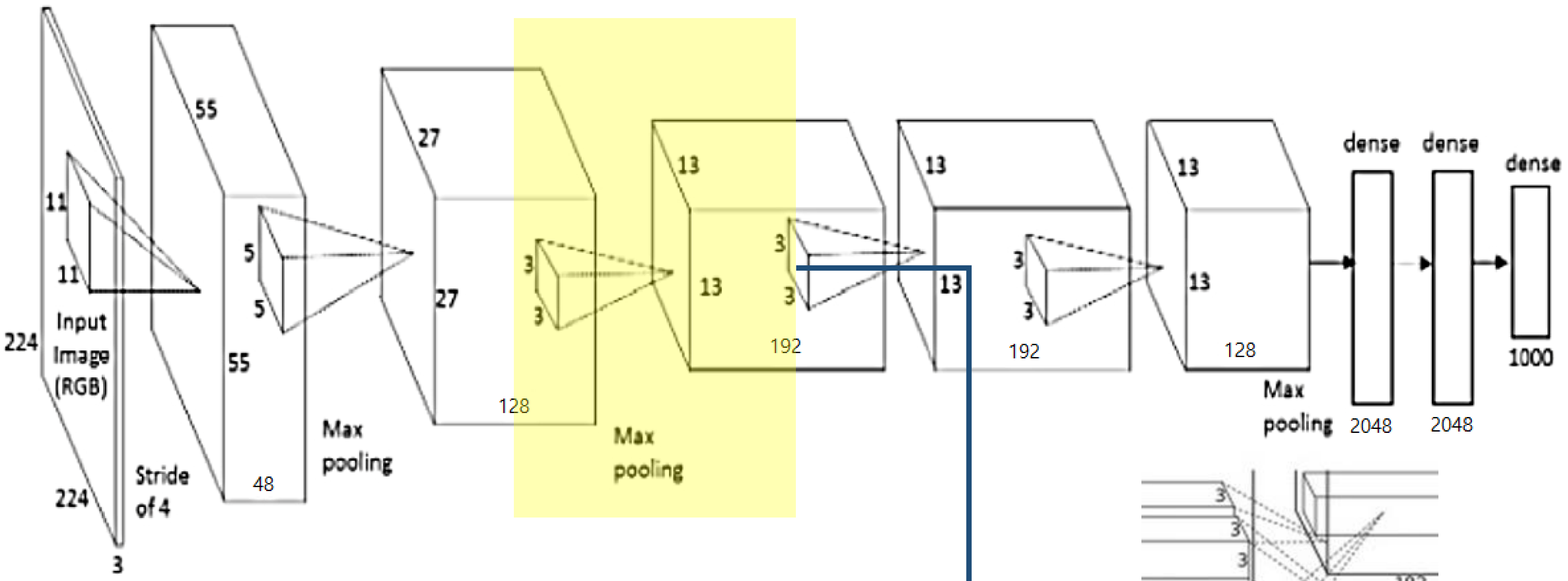
Overall Architecture

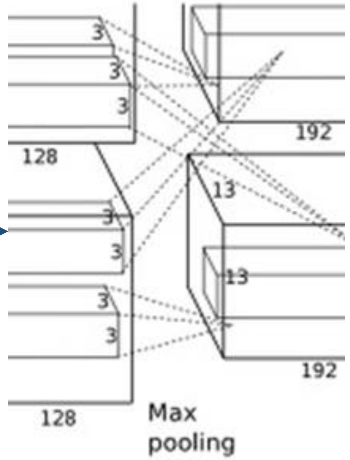


- Input image : Response normalized & Pool
- 2nd Conv. Kernel: $5 \times 5 \times 48, @256$
- Max pooling

AlexNet

Overall Architecture

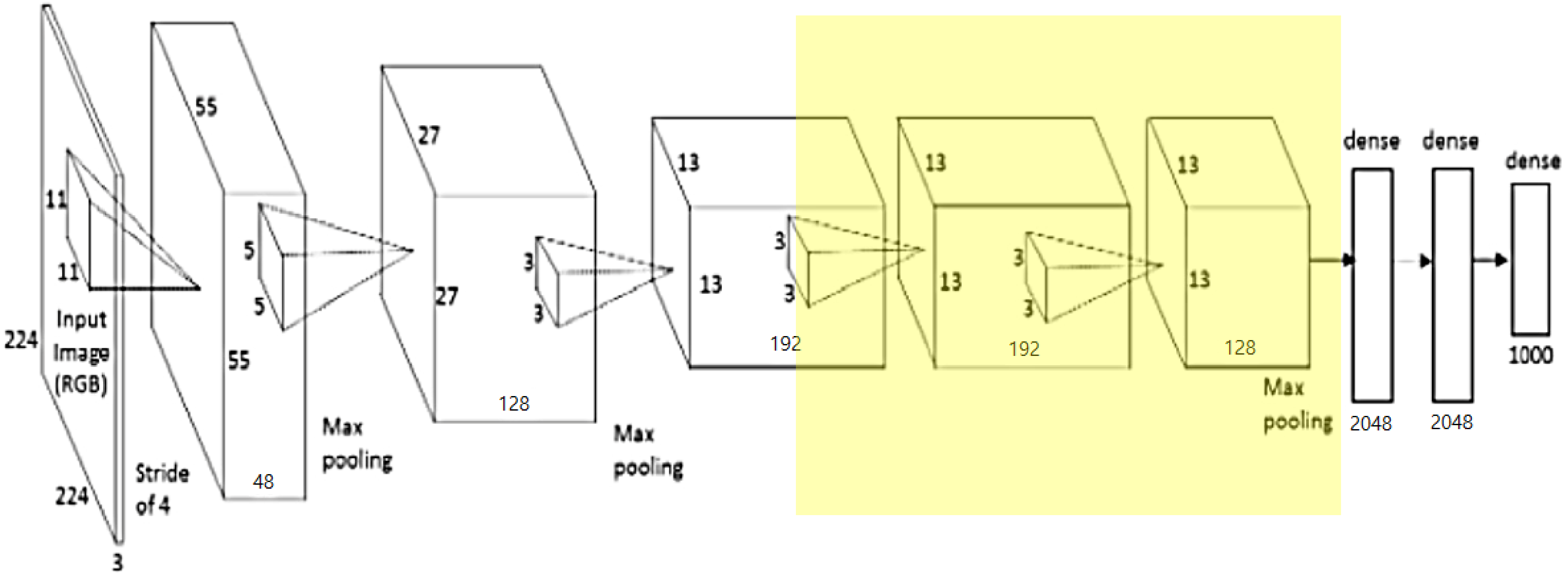




- Input image : Response normalized & Pool
- 3rd Conv. Kernel: $3 \times 3 \times 256, @384$
- GPU-1,GPU-2 mix → Able to tune the communication precisely

AlexNet

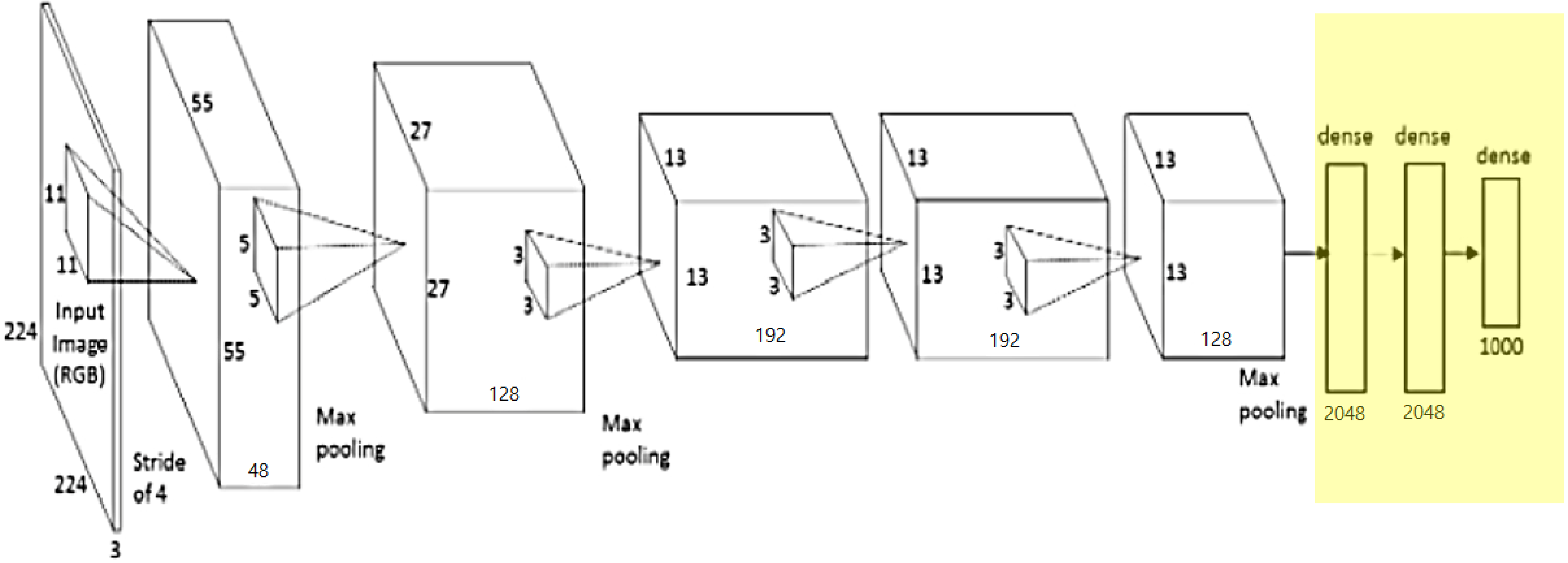
Overall Architecture



- Input image : Not Response normalized & Not Pool
- 4th Conv. Kernel : $3 \times 3 \times 192, @384$
- 5th Conv. Kernel : $3 \times 3 \times 192, @256$

AlexNet

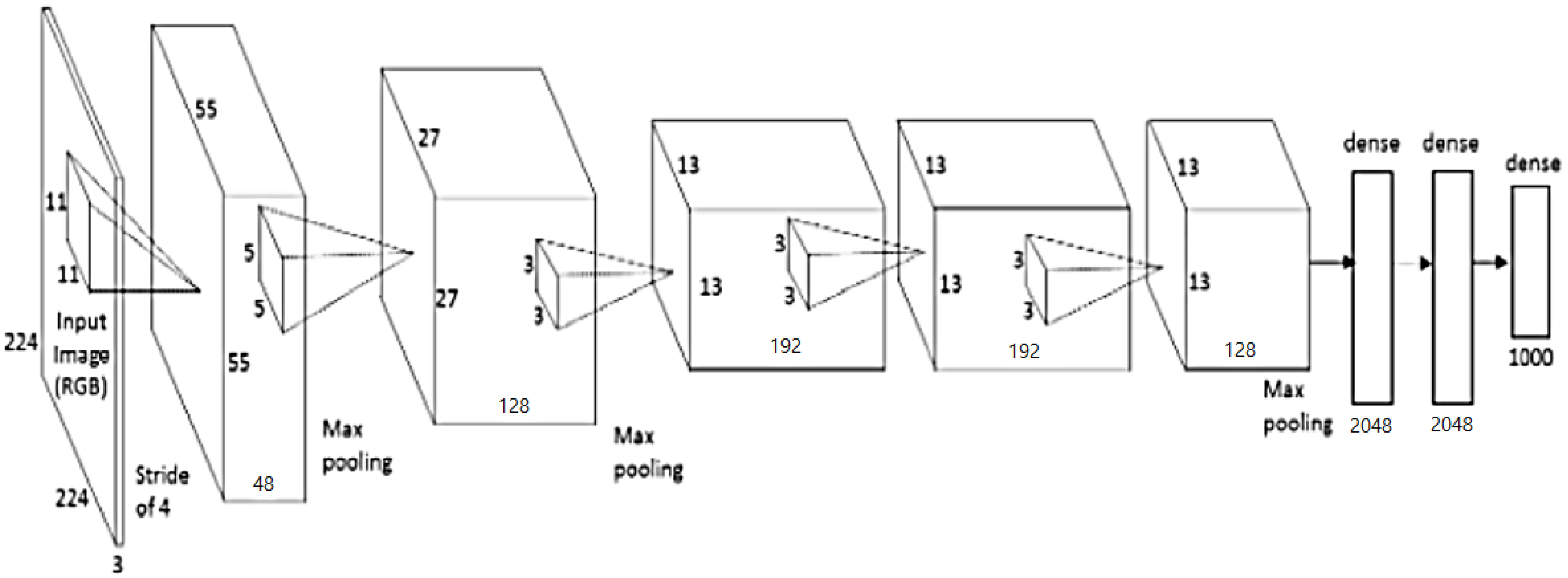
Overall Architecture



- 3 Fully connected Layer
- Total 4096 connect to Fully Connected layer
- Softmax for category at last layer

AlexNet

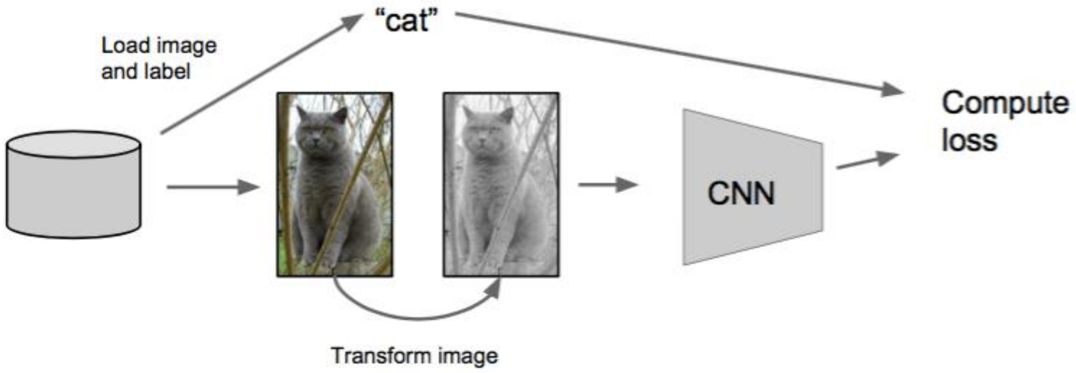
Overall Architecture



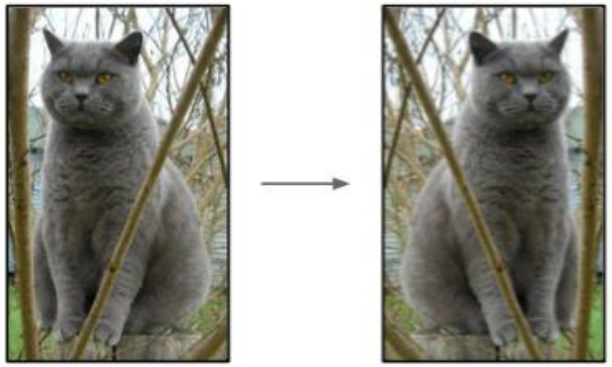
- SGD(Stochastic gradient descent) with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005
- We used an equal learning rate for all layers. The heuristic was to divide the learning rate by 10 when the validation error rate stopped imporving with the current learning rate.

AlexNet

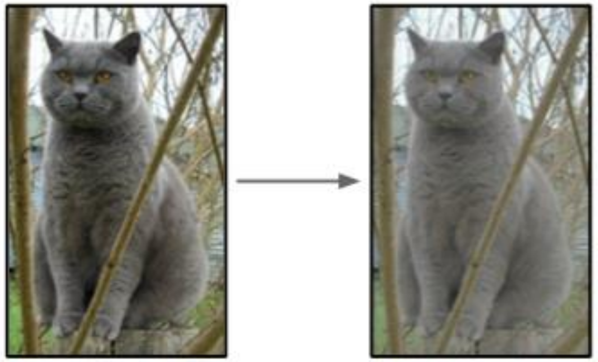
Overfitting - Data Augmentation



Generating Image translations
& horizontal reflections



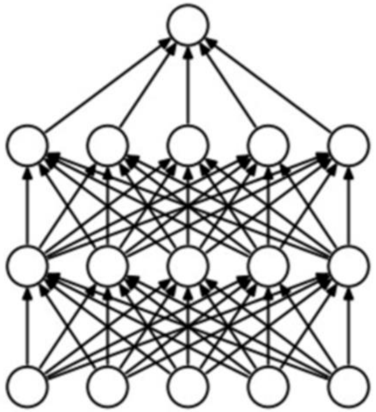
Altering intensities of RGB channel

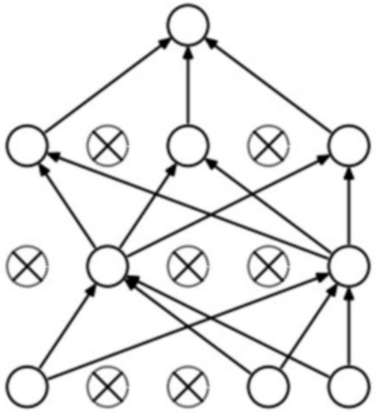


AlexNet

Overfitting - Dropout

- Dropout consists of setting to zero the output of each hidden neuron with probability 0.5
- The neurons which are “dropped out” in this way do not contributed to the forward pass and do not participate in back-propagation.
- At test time, we use all the neurons but multiply their outputs by 0.5, which is a reasonable approximation to taking the geometric mean of the predictive distributions produced by the exponentially-many dropout networks.


(a) Standard Neural Net

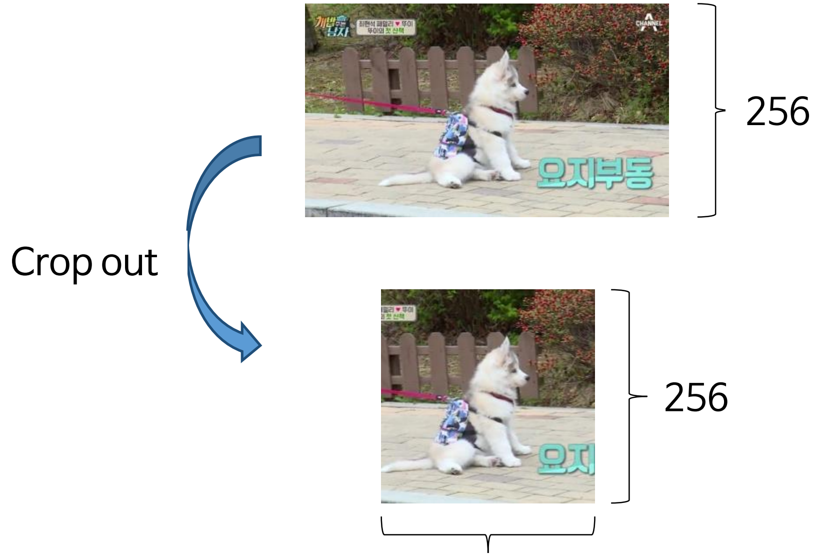

(b) After applying dropout.

AlexNet

Data

- ILSVRC : 15 million labeled high-resolution images belonging to roughly 22,000 categories

- Reshape image



AlexNet

Result

- ILSVRC 2010

Model	Top - 1	Top - 5
Spare coding	47.1 %	28.2 %
SIFT + FVs	45.7 %	25.7 %
CNN	37.5 %	17.0 %

ILSVRC-2010 winner

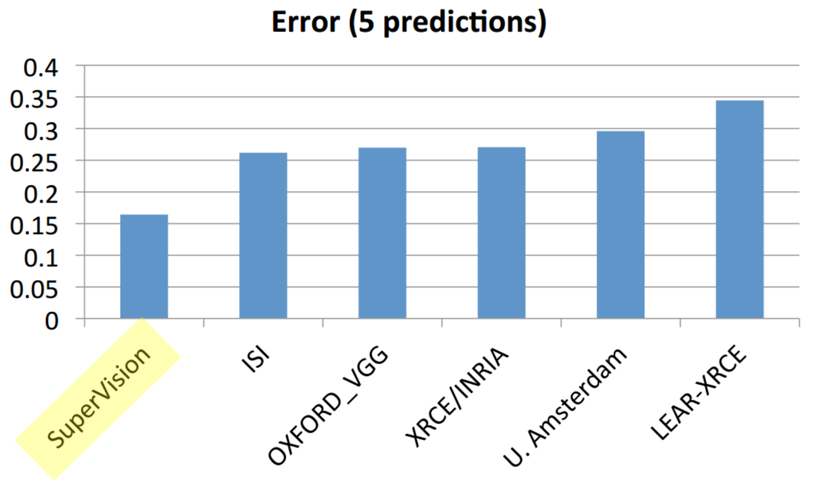
Previous best published result

Proposed Method

- ILSVRC 2012

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

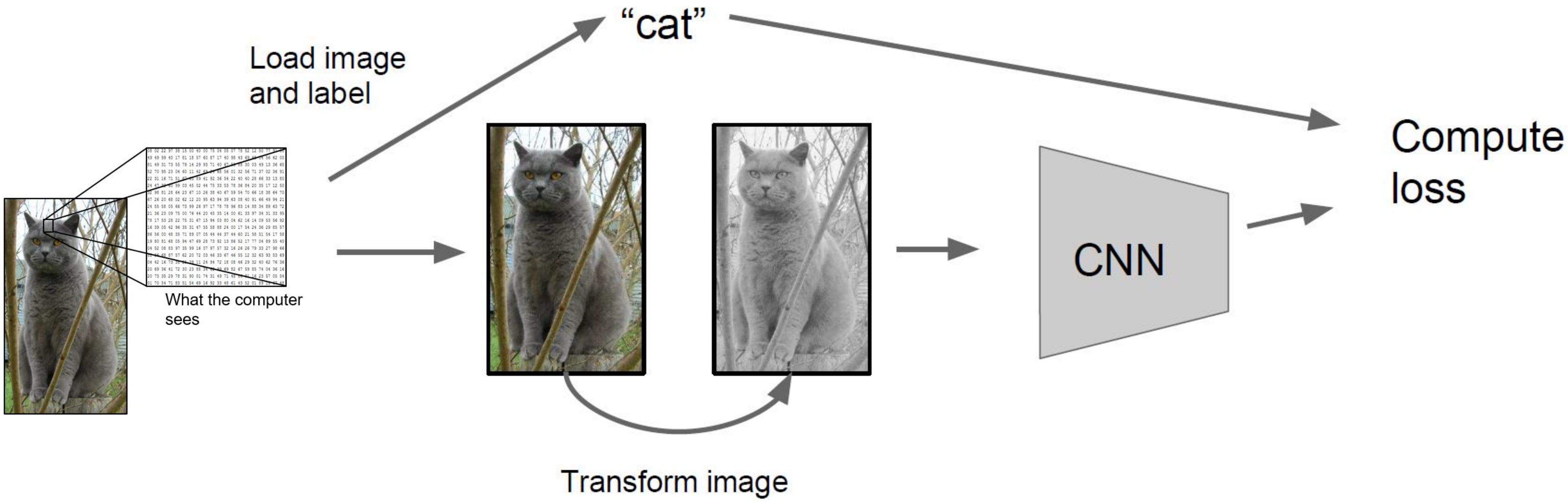
* : "pre-trained" to classify the entire ImageNet 2011 Fall



Data Augmentation

Data Augmentation

Introduction

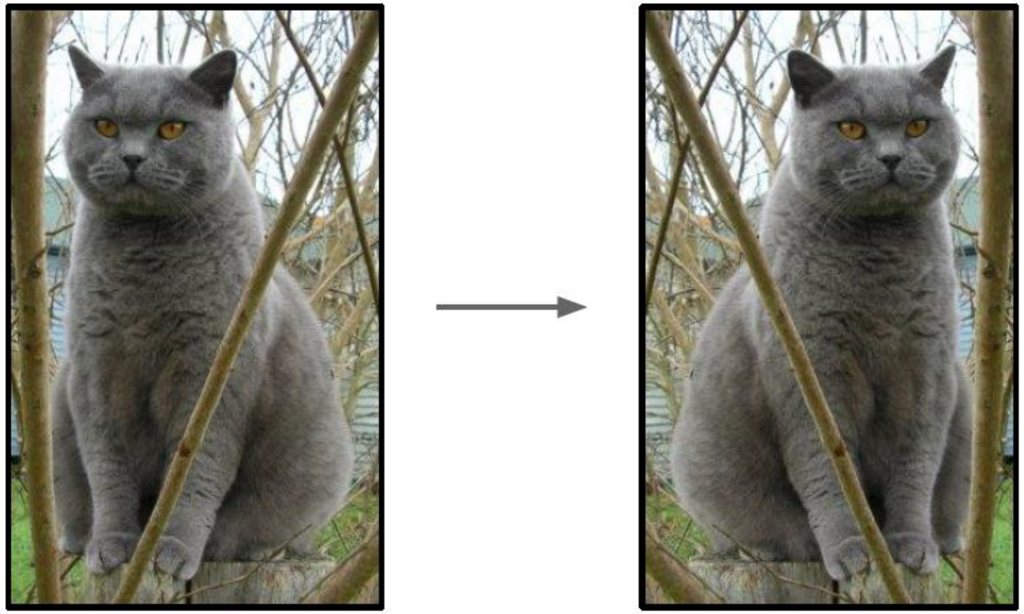


- Change the pixels without changing the label (= label-preserving transformations)
- Train on transformed data

→ Artificially enlarge the dataset

Data Augmentation

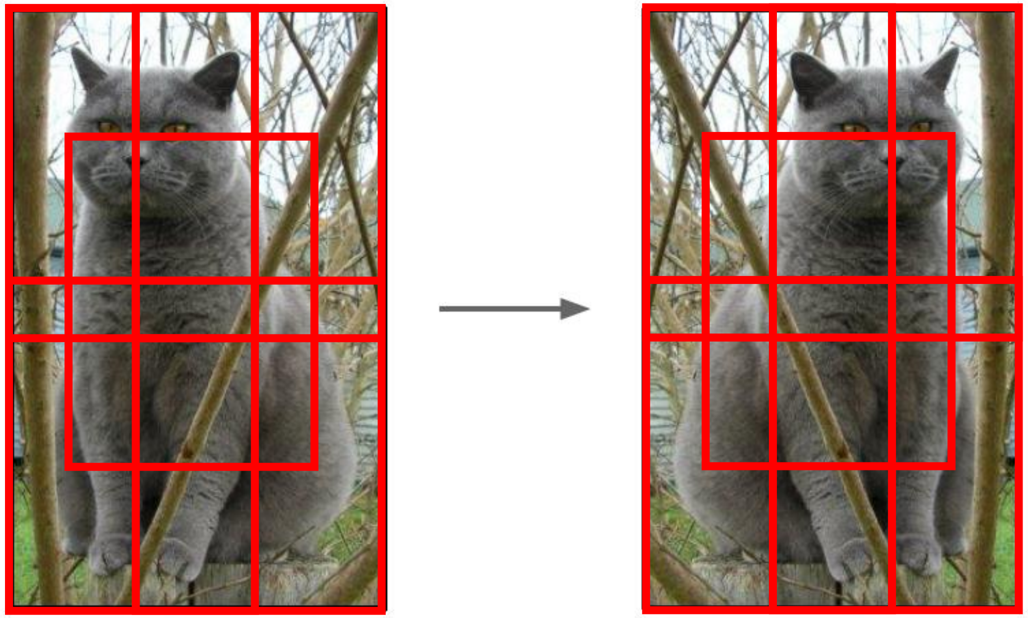
Horizontal Flips



- In AlexNet, By extracting random 224 X 224 patches from the 256 X 256 images and training our networks on these extracted patches.
  → Training set can increase by a factor2048

Data Augmentation

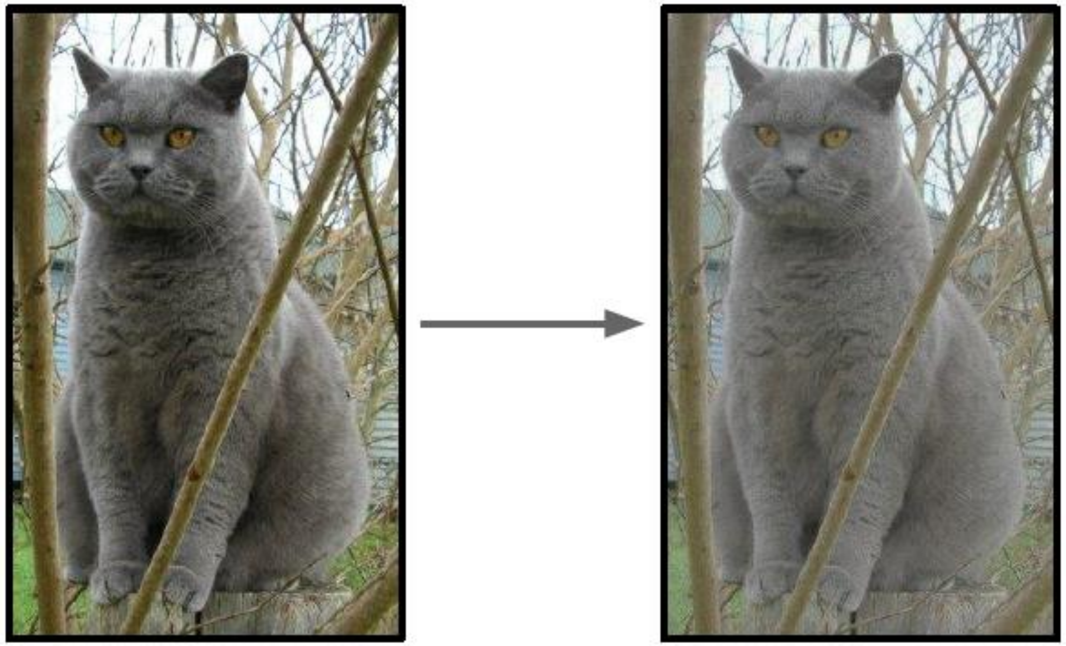
Horizontal Flips



- In AlexNet, 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.
  → Averaging the predictions made by the network's softmax layer on the ten patches

Data Augmentation

Color Jittering



- Simple : Randomly jitter contrast
- Complex : Apply PCA to all [R, G, B] pixels in training set. Then sample a “color offset” along principal component directions. So, Add offset to all pixels of a training image

Data Augmentation

ETC

- Translation
- Rotation
- Stretching
- Shearing
- Lens distortions…

Data Augmentation

Tensorflow

```
def distorted_inputs(data_dir, batch_size):
  """Construct distorted input for CIFAR training using the Reader ops.

  Args:
  data_dir: Path to the CIFAR-10 data directory.
  batch_size: Number of images per batch.

  Returns:
  images: Images. 4D tensor of [batch_size, IMAGE_SIZE, IMAGE_SIZE, 3] size.
  labels: Labels. 1D tensor of [batch_size] size.
  """

  filenames = [os.path.join(data_dir, 'data_batch_%d.bin' % i) for i in xrange(1, 6)]
  for f in filenames:
    if not tf.gfile.Exists(f):
      raise ValueError('Failed to find file: ' + f)

  # Create a queue that produces the filenames to read.
  filename_queue = tf.train.string_input_producer(filenames)

  # Read examples from files in the filename queue.
  read_input = read_cifar10(filename_queue)
  reshaped_image = tf.cast(read_input.uint8image, tf.float32)

  height = IMAGE_SIZE
  width = IMAGE_SIZE
```

```
  # Image processing for training the network. Note the many random
  # distortions applied to the image.

  # Randomly crop a [height, width] section of the image.
  distorted_image = tf.random_crop(reshaped_image, [height, width, 3])

  # Randomly flip the image horizontally.
  distorted_image = tf.image.random_flip_left_right(distorted_image)

  # Because these operations are not commutative, consider randomizing
  # the order their operation.
  distorted_image = tf.image.random_brightness(distorted_image, max_delta=63)
  distorted_image = tf.image.random_contrast(distorted_image, lower=0.2, upper=1.8)

  # Subtract off the mean and divide by the variance of the pixels.
  float_image = tf.image.per_image_whitening(distorted_image)

  # Ensure that the random shuffling has good mixing properties.
  min_fraction_of_examples_in_queue = 0.4
  min_queue_examples = int(NUM_EXAMPLES_PER_EPOCH_FOR_TRAIN *
                   min_fraction_of_examples_in_queue)
  print('Filling queue with %d CIFAR images before starting to train.'
     'This will take a few minutes.' % min_queue_examples)

  # Generate a batch of images and labels by building up a queue of examples.
  return _generate_image_and_label_batch(float_image, read_input.label,
                              min_queue_examples, batch_size,
                              shuffle=True)
```

Data Augmentation

Python

Imgaug

https://github.com/aleju/imgaug