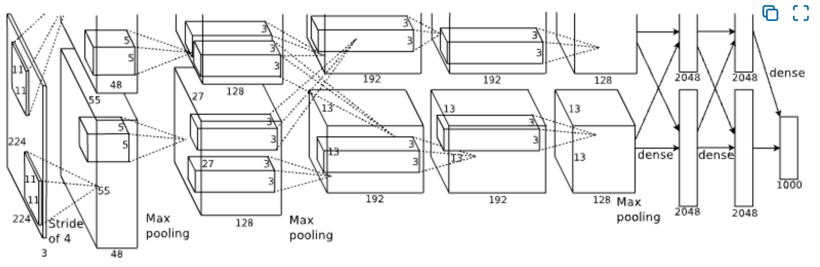
ImageNet Classification with Deep Convolutional Neural Networks

- [[📚 NIPS-2012-imagenet-classification-with-deep-convolutional-neural-networks-Paper.pdf]]
- **Basic Idea**:
  - Try to use deep CNNs for image classification.
  - AlexNet: 60 million parameters, ImageNet accuracy: 62.5% (top1), 83% (top5)
- **Tricks of the Trade**:
  - *ReLU Nonlinearity*:
    - Advantage: Speeds up the training in comparison to the saturating non-linearities such as $tanh(x)$.
    - Other non-linearity: 🟡 **P3** 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.
-
  - *Group Convolutions*:
    - 🟡 **P5**



    - Advantage: Due to limited GPU memory, they split the networks across 2 GPUs thereby utilising group convolutions.
    - 🟡 **P3** 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 if a layer has 20 filters then it split across different GPUs (i.e. 10 on each GPU). However, instead of letting both groups communicate after every layer, the interaction happens only in layer 3 and dense layers.
    - 🟡 **P3** 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.
    - Note that in the above figure, the image is not split between GPUs (i.e., one GPU doesn't handle upper part of the image and other GPU doesn't get the lower part of the image). Both of the GPUs get the whole image. The split happens along the depth (filters).
    - *Advantage of using group convolution with restricted connectivity*:
      - Specialization of features in a group: 🟡 **P7** Notice the specialization exhibited by the two GPUs, a result of the restricted connectivity described in Section 3.5. The kernels on GPU 1 are largely color-agnostic, while the kernels on on GPU 2 are largely color-specific.
  - *Layer Response Normalization*:
    - 🟡 **P4** ReLUs have the desirable property that they do not require input normalization to prevent them from saturating.
      - This is because for other saturating activation functions such as $tanh(x)$ or $sigmoid(x)$, the values nearby 0 are only non-saturating i.e., are approximately linear. Thereby, if we keep the inputs nearby 0, then we can ensure that the learning can happen as we are operating in the non-saturating region of the activation function.
    - 🟡 **P4** However, we still find that the following local normalization scheme aids generalization.
    - Response-normalized activity $b^i_{x,y}$ is given by

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

where,
$a^i_{x,y}$: activity of the neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity.
The sum runs overs n "adjacent" kernel maps at the same spatial position, and N is the total

number of kernels in the layer. 🟡 **P4** The ordering of the kernel maps is of course arbitrary and determined before training begins.

- 🟡 **P4** This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels.
    - To understand why this is the case, let's set $k=0, \alpha=1, \beta=\frac{1}{2}, n=2N$ for the extreme case. Therefore, the expression reduces to

      $$b^i_{x,y} = \frac{a^i_{x,y}}{\left(\sum_{j=0}^{N-1}(a^i_{x,y})^2\right)^{1/2}}$$

      which is similar to diving the activation by the $l2$-norm of the vector consisting of the adjacent activations. Now, since the norm of the resultant vector is 1, therefore, the neurons will compete to get higher value as the upper value of the norm is bounded by 1.
    - In the paper, the hyperparameter values are determined using validation set: $k=2, n=5, \alpha=10^{-4}, \beta=0.75$.
- 🟡 **P4** Response-normalization layers follow the first and second convolutional layers.
- 🟡 **P4** This scheme bears some resemblance to the local contrast normalization scheme of Jarrett et al. [11], but ours would be more correctly termed “brightness normalization”, since we do not subtract the mean activity.
    - For images, consider the element-wise operation

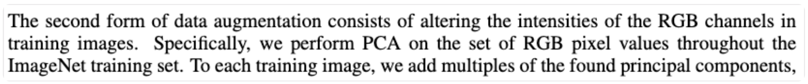
      $$g(\mathbf{x}) = af(\mathbf{x}) + b$$

      where the parameters $a > 0$ and b are called the *gain* and *bias* parameters; sometimes these parameters are said to control *contrast* and *brightness*, respectively.

- *Overlapping Pooling*
    - 🟡 **P4** Pooling layers in CNNs summarize the outputs of neighboring groups of neurons in the same kernel map.
    - 🟡 **P4** a pooling layer can be thought of as consisting of a grid of pooling units spaced s pixels apart, each summarizing a neighborhood of size z × z centered at the location of the pooling unit
        - s is the stride of the pooling layer.
        - 🟡 **P4** If we set s = z, we obtain traditional local pooling as commonly employed in CNNs For example, max-pooling with a filter size z of 2 and stride of 2, will halve the resolution.
        - 🟡 **P4** If we set s < z, we obtain overlapping pooling.
    - They observed that with overlapping pooling it is slightly more difficult to overfit.

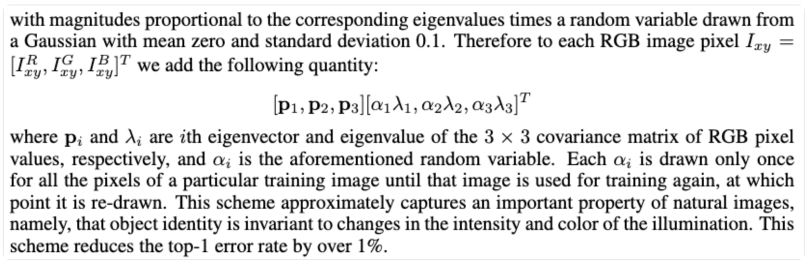
- *Reducing Overfitting*:
    - 🟡 **P5** Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting
    - *Data Augmentation*
        - *10-crop data augmentation*: 🟡 **P5** averaging the predictions made by the network’s softmax layer on the ten patches.
        - *Fancy PCA*:

          🟡 **P5**

          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,

          🟡 **P6**

          with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1. Therefore to each RGB image pixel $I_{xy} = [I^R_{xy}, I^G_{xy}, I^B_{xy}]^T$ we add the following quantity:

          $$[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

          where $\mathbf{p}_i$ and λ_i are ith eigenvector and eigenvalue of the 3×3 covariance matrix of RGB pixel values, respectively, and α_i is the aforementioned random variable. Each α_i is drawn only once for all the pixels of a particular training image until that image is used for training again, at which point it is re-drawn. This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.
    - *Dropout*:
        - 🟡 **P6** This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn

more robust features that are useful in conjunction with many different random subsets of the other neurons

      - 🟡 **P6** 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
      - 🟡 **P6** Dropout roughly doubles the number of iterations required to converge.

- **Results**
  - *Visual Inspection*:
    - 🟡 **P6**

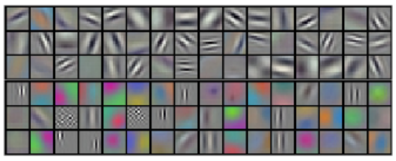
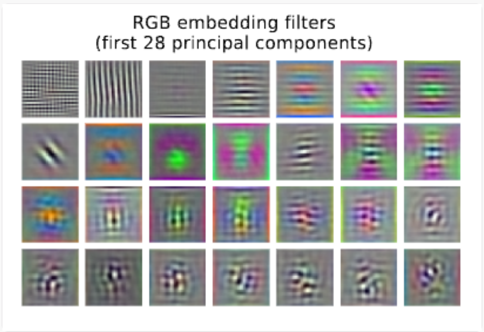
      

      Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

    - 🟡 **P7** The network has learned a variety of frequency- and orientation-selective kernels, as well as various colored blobs.
    - 🟡 **P7** Notice the specialization exhibited by the two GPUs, a result of the restricted connectivity described in Section 3.5. The kernels on GPU 1 are largely color-agnostic, while the kernels on on GPU 2 are largely color-specific.
    - Compare and contrast this with the patch embeddings learned by ViTs.
      - *Patch Embeddings*:
        - 🟡 **P9**

        - Patch embeddings can be thought of as convolutions. $\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$, therefore, this layer can be thought of as learning D filters, each of shape $P \times P \times C$. Now, usually D is as high as 768, therefore, to visualize the important filters, we can do a PCA and pick first 28 filters.
        - 🟡 **P8** The components resemble plausible basis functions for a low-dimensional representation of the fine structure within each patch.

-
  - *k-Nearest Neighbour Search*:
    - 🟡 **P8** If two images produce feature activation vectors with a small Euclidean separation, we can say that the higher levels of the neural network consider them to be similar. Pick an image from the test set and search nearest images from the training set.
    - *Speedup Trick*: 🟡 **P8** Computing similarity by using Euclidean distance between two 4096-dimensional, real-valued vectors is inefficient, but it could be made efficient by training an auto-encoder to compress these vectors to short binary codes. This should produce a much better image retrieval method than applying autoencoders to the raw pixels [14], which does not make use of image labels and hence has a tendency to retrieve images with similar patterns of edges, whether or not they are semantically similar.