# Paper Review

# ImageNet classification with Deep convolutional neural networks

## Purpose:-

Using deep learning techniques to efficiently and effectively predict the image subject.
Results achieved: top-1 error rates- 37.5%, top-5 error rates - 17%

**The dataset used to train and test and its selection criteria:-**

The dataset used for labeling:- ImageNet-2010
Reason for choosing:-
One of the largest datasets of 15 million images
Has a labeled test dataset which is not available in larger datasets of ImageNet

**Data Cleaning:-**

Images have been resized to 256 x 256 pixels
The shorter side of each image has been resized to 256 pixels and the center portion of each image has been cropped out.
The Mean Pixel Value of the complete dataset has been subtracted from all the pixels to make their value near 0.
The network has been trained on raw RGB values of the center patch.

**The architecture:-**

The network consists of:- Eight learned layers— five convolutional and three fully connected.

**Function for neuron output:-**

Rectified Linear Unit nonlinearity function has been chosen for the output of each neuron on the basis of input.
This function is $f(x) = \max(0,x)$
Reason for this function:-

It does not have a maximum value which never lets it saturate.
This non-saturation function keeps the training data alive and gives a broader scope for training the model. Which saturating functions can't do at a large value of data points. Also, this function's implementation is very fast than saturating function's implementation.

**GPU use for training:-**

Training has been done on 2 GPUs called GTX 580 GPU of 3GB memory.
A single GPU was not sufficient for efficiently running developed neural network that's why 2 were used.
Kernels between the GPU were divided by parallelization scheme which means that half of the kernels were placed in a single GPU.
The connectivity pattern has been decided to minimize the communication between th GPUs.
Communication pattern between GPUs has been decided on the basis of cross-validation output.
Highly effective GPU architecture
Top-1 error is reduced by 1.17%
Top-5 error is reduced by 1.2%

**Local Response Normalization:-**

Though ReLu does not require normalization because of non-linearity. But then too a generalized normalization has been implemented for improving accuracy.
Response Normalization:-

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

$a^i_{x,y}$= the activity of a neuron computed by applying kernel i at position (x, y)
k,α,n,□=hyperparameters whose values are determined by validation set
*Reduced error rates by:*
*top-1:-1.4%*
*top-5:-1.5%*

**Overlapping Pooling's use for benefit:**

Unlike traditional pooling in which space s between pooling layers is equal to the spatial dimensions of neighbourhood z x z authors here have used overlapping pooling which means s<z. This is dataset or problem statement is a special case in which overlapping pooling has yield better results than traditional pooling.
By overlapping pooling errors are reduced by

Top-1 errors:- 0.4%
Top-5 errors- 0.3%

**Overall Architecture:-**

Overall neural network model is made up of 8 layers. First 5 are convolutional and next 3 are fully connected.
***Response normalization layers: -*** After 1st and 2nd layer
***Pooling Features layers:-*** Response normalization and 5th convolution layer.
***Layers Linking:-***
Second, fourth, and fifth Convolutional Layers: The kernels in these layers are connected specifically to the kernel maps in the previous layer that are located on the same GPU. This means that the kernels in the second convolutional layer only receive input from the kernel maps in the first convolutional layer that reside on the same GPU. Similarly, the kernels in the fourth and fifth convolutional layers are connected to the kernel maps in the third and fourth layers, respectively, but only those residing on the same GPU.

Third Convolutional Layer: The kernels in the third convolutional layer have connections to all kernel maps in the second layer. This means that each kernel in the third convolutional layer receives input from every kernel map in the second layer, regardless of the GPU they are located on.

Fully-Connected Layers: The neurons in the fully-connected layers are connected to all neurons in the previous layer, regardless of the layer they come from or the GPU they are located on. This means that each neuron in the fully-connected layers receives input from every neuron in the previous layer.

***Layer Fitting Sizes:-***

1st convolution layer:- filters the 224 × 224 × 3 input image with 96 kernels of size 11 × 11 × 3 with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map).

2nd convultional takes output of 1st convlutional layer and filters it with 256 kernels of sizes 5 x 5 x 3.

Similarly 3rd convlutional layer takers in input of 2nd layers' output.
3rd,4th,5th convlutional layers are interconnected without any pooling.

**Reducing Overfitting:**

**Overfitting:-** This is a condition when the ml model learns the training data very well but performs very poorly on the test dataset. It usually occurs with extremely complex models which fail to recognize the error possibility in the training dataset. Also when the model memorizes the training data instead of finding an underlying pattern the prediction curve overfits the training data.

***Approaches Used to Reduce Overfit:***

1. ***Reframing training data:-*** Random images from the training data were chosen and patches of 224 x 224 sizes were extracted along with this the horizontal images of each patch was also created. Patches were chosen from 4 corners and a central patch was chosen. This was done on cpu in data preprocessing by writing a python code. Afterwards the patches were added to training dataset. Size of training dataset increased by 2048 times. This data augmentation made model learn about an object from different views.

2. ***Altering RGB intensity:-*** In order to train model on robust factors which are independent of changes to light intensity and illumination this method is used. RGB pixel values of each training image is changed and then added to training dataset.
   How is it done?
   - Firstly, PCA of image RGB pixel matrix is done. Eigen values of each principal component is multiplied by a random variable chosen from a Gaussian distribution with mean 0 and standard deviation 1.

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

$P_i$ ,$\lambda_i$ =eigen values and eigen vectors
$\alpha_i$= Random variable

This quantity is added to RGB value matrix.
This method reduced top-1 error by 1%.

**Dropout:**
To make model train on different kinds of input dropout method is used. In this method output of a fixed number of neurons is turned to 0 and model keeps on training on a different subset of neuron every time. This helps in reducing noise and increasing robustness of model. After all the iterations the output is converged to give the predictions.

During testing the output of each neuron is scaled by a factor of 0.5 to meet regularise the dropout effect. 0.5 is an approximation of taking the geometric mean of the predictions produced by the exponentially-many dropout networks.

**Learning Details:-**

So weights for the model have been changed in every iteration. This is done to reduce the overfitting due to larger weights. Smaller weights have been introduced in every iteration to reduce model training error.
The model has been trained using stochastic gradient descent with batch of 128 examples of momentum of 0.9 and weight decay of 0.0005.

What is stochastic gradient descent?

Method of reducing the gradient of loss function by distributing data into random batches of fixed numbers and then changing the weights in every iteration.

Here batch size is of 128 samples.

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

Here:-
$v_i$= momentum variable(defines the direction of gradient of current iteration based on previous gradients)
e=learning rate
$w_i$=weights at $i^{th}$ iteration
L=objective funnction / loss function
$w_{i+1}$= weights at $i+1^{th}$iteration

**Results:-**

- Well on training over ILSVRC-2010 network achieves top-1 and top-5 test set error rates of 37.5% and 17.0%, respectively.
- Training one CNN with an additional sixth convolutional layer on the ImageNet Fall 2011 dataset (15M images, 22K categories), and fine-tuning it on ILSVRC-2012, resulted in an error rate of 16.6%.
- Combining the predictions of two pre-trained CNNs with the aforementioned five CNNs reduced the error rate to 15.3%.
- Another approach using different classifiers trained on densely sampled features achieved an error rate of 26.2%.
- Additionally, on the Fall 2009 version of ImageNet (10,184 categories, 8.9 million images), the network achieved top-1 and top-5 error rates of 67.4% and 40.9%, respectively, outperforming the best-published results of 78.1% and 60.9%.

**Qualitative Evaluation:**

1. The network has learned a variety of frequency- and orientation-selective kernels, as well as various colored blobs.

2. The kernels on GPU 1 are largely color-agnostic, while the kernels on on GPU 2 are largely color-specific. This kind of specialization occurs during every run and is independent of any particular random weight initialization (modulo a renumbering of the GPUs).

3. Computing similarity by using Euclidean distance between two 4096-dimensional, real-valued vectors is inefficient, but 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 auto encoders to the raw pixels, 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.

**Scope of improvement:**

Model architecture: The AlexNet architecture, with its five convolutional layers and three fully connected layers, has inspired the development of deeper and more sophisticated architectures. More recent architectures, such as VGGNet, GoogLeNet, ResNet, and EfficientNet, have demonstrated improved performance by increasing depth, incorporating skip connections, utilizing inception modules, or leveraging compound scaling techniques.

Regularization techniques: The original AlexNet paper used techniques like dropout and data augmentation for regularization. Since then, various regularization methods have been explored, including batch normalization, weight decay, mixup, and cutout. These techniques help improve model generalization and mitigate overfitting.

Training strategies: The Krizhevsky et al. paper introduced the use of GPUs for training deep neural networks efficiently. Since then, training strategies have evolved, including the use of adaptive learning rate algorithms (e.g., Adam, RMSprop), learning rate schedules, and warm-up steps. These techniques help stabilize training and improve convergence.

Pretraining and transfer learning: While the AlexNet paper trained the model from scratch, subsequent research has shown the effectiveness of transfer learning and pretraining on larger datasets. Pretraining on massive datasets like ImageNet or utilizing architectures pretrained on them has become a common practice, leading to improved performance and faster convergence on target datasets with limited training data.

Ensembling and model fusion: Ensembling multiple models or fusing predictions from different models has been found to enhance classification performance. Techniques like model averaging, stacking, or using ensemble methods such as bagging or boosting can improve accuracy and robustness.

Data augmentation: The original paper utilized data augmentation techniques such as image flipping and cropping. Since then, more advanced augmentation techniques have been explored, including rotation, translation, scaling, color augmentation, and generative adversarial networks (GANs)-based augmentation. These techniques increase the diversity of the training data and improve model generalization.

Optimizers and regularization methods: The original paper used stochastic gradient descent (SGD) with momentum as the optimizer. Research has introduced new optimization algorithms, such as Adam, RMSprop, and LARS, which offer improved convergence and better generalization.

Network interpretation and visualization: While not directly related to the paper's content, post hoc analysis techniques have been developed to interpret and visualize the learned representations and decision-making processes of deep neural networks. Techniques such as class activation maps (CAM), saliency maps, and attention mechanisms provide insights into the important regions and features that contribute to the model's predictions.

It's important to note that the Krizhevsky et al. paper laid the foundation for subsequent advancements in deep learning for image classification. The scopes of improvement mentioned above represent the progress made since the publication of the paper, highlighting the ongoing research and innovations in the field.