A Study of AlexNet

Sandipan Sarma

Project: Stage-II, Advanced Topics in Machine Learning (EE-525),
IIT Guwahati, India

1 Introduction

Deep learning has been providing major breakthroughs in solving several problems since the past few years, that have withstood many attempts of machine learning and artificial intelligence (AI) community in the past. Hence, it is now at the heart of the current rise of AI and is being used to decipher a wide range of problems in the scientific community at a gigantic scale, e.g. in analysis of mutations in DNA, self-driving cars, speech recognition tasks, and natural language understanding among others. Although deep learning arrived in the early 2000s, the seismic shift in the field of image processing happened in 2012, when Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton demonstrated their research paper on "ImageNet classification with deep convolutional neural networks" on a very large-scale visual recognition task in ImageNet Large Scale Visual Recognition Challenge (ILSVRC). ILSVRC evaluates the success of the participating image classification solutions by evaluating the top-1% error (the percentage of time the classifier was unable to assign the highest prediction score to the correct class) and top-5% error (the percentage of time the classifier did not include the correct class in its top five predictions). The proposed Convolutional Neural Network (CNN) - AlexNet - achieved a top-5 error rate of 16.4%, which was considerably better than the previous year winners of the challenge at that time (a comparative graph has been shown in Figure 1). Hence, it became a milestone in deep learning, and paved the way for more complex architectures to come in the upcoming years, which would go on to even beat humans in visual recognition.

In this work, a comprehensive study of AlexNet has been presented. Section 2 provides a background - explaining the journey from machine learning to deep learning, and the one from simple feed-forward neural networks to CNNs. Sections 3 through 6 are solely dedicated to Alexnet - its architecture, choice of activation and pooling layers, significance of dropout nodes incorporated, and explains why it worked so well so as to become an exemplary CNN. Section 7 contains the implementation details, and the observations and results have been enumerated in Sections 8 and 9 respectively. Finally, the work has been concluded in Section 10.

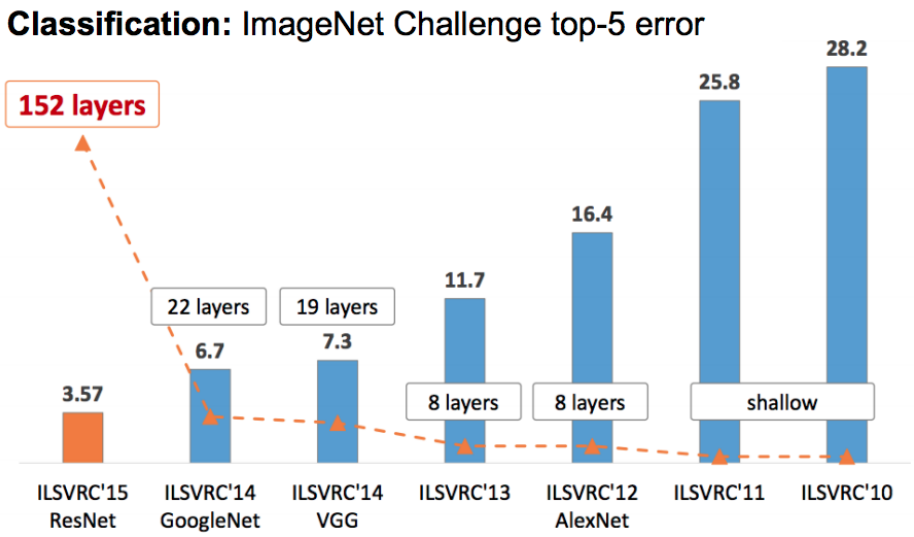


Figure 1: ILSVRC winners

2 Background

We need machine learning approaches for tasks that are too complex to code via general programming paradigms, as it might be impossible for us to work out all the possible scenarios of the real-life problems that we are trying to solve. For example, while creating a recommendation system for an online platform like *Netflix*, the objective is to help a user find a show or movie to enjoy with minimal effort. To build it, *Netflix* needs to analyse several factors based on the data collected from millions of users, as shown in Figure 2. Now writing a procedural or object-oriented program for this explicitly is not possible, since the programmer, on his/her own, cannot find out all the patterns and inter-relations in the collected data. Since the programmer can only explicitly code once he/she can conceptualize, visualize and discretize the working conditions, then think of the code statements to run, and finally list out the expected results - he/she will be unable to write lines of code for the desired recommendation system.

Machine learning algorithms basically build a mathematical model on the *training data* it is fed, and learns to make decisions without the need of any explicit programming. But again, with the rise in technology, the world witnessed information explosion - large amounts of data became accessible. The performance of most of the traditional machine learning algorithms depend on how well the features have been extracted from the data and analyzed (a process known as *feature engineering*), but doing this process over a huge corpus of data is expensive in terms of time and expertise. But with the advent of

deep learning, which became more feasible in the 2010s mainly due to reduced hardware costs, the problem of working effectively with huge amounts of data has been solved to a huge extent. A comparison of the two approaches has been shown in Figure 3.

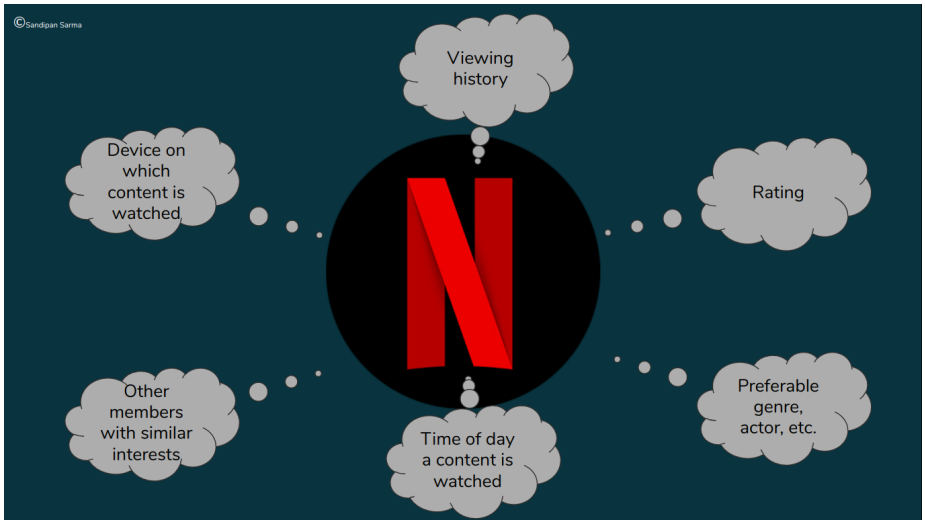


Figure 2: Input factors for Netflix's recommendation system

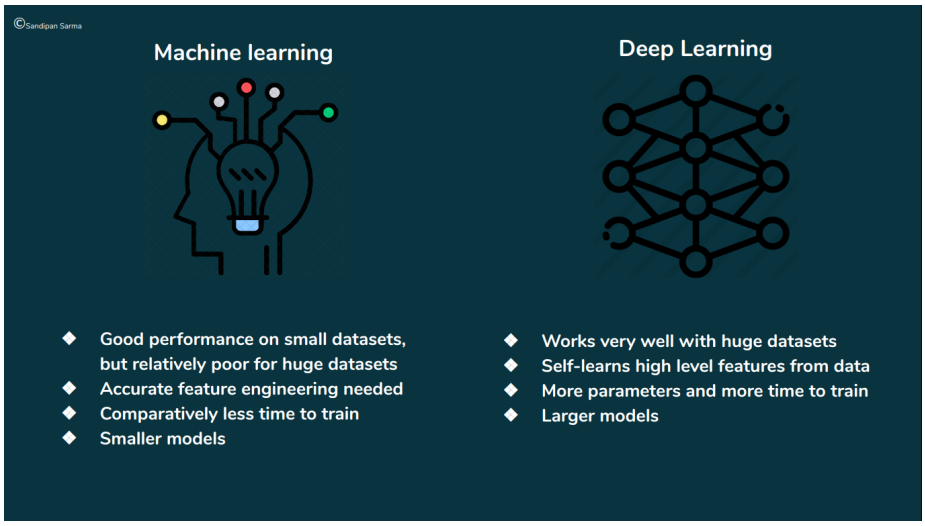


Figure 3: Machine learning vs Deep Learning

A major revolution in machine learning took place with the successful applications of **neural networks**. The primary reasons for so much popularity

of the neural networks were:

- They were proven to be universal function approximators, and hence they could be applied to almost any machine learning problem for learning a complex (often non-linear) mapping from the input to the output space.
- They were inspired by the neural structure of the human body, and hence try to mimic the numerous decision-making and learning paradigms that the human brain uses. This makes the problem-solving process more intuitive and incorporates conceptual similarities with the human brain's approach of solving the problem at hand.

A typical architecture for a *feed-forward* neural network has been given in Figure 4.

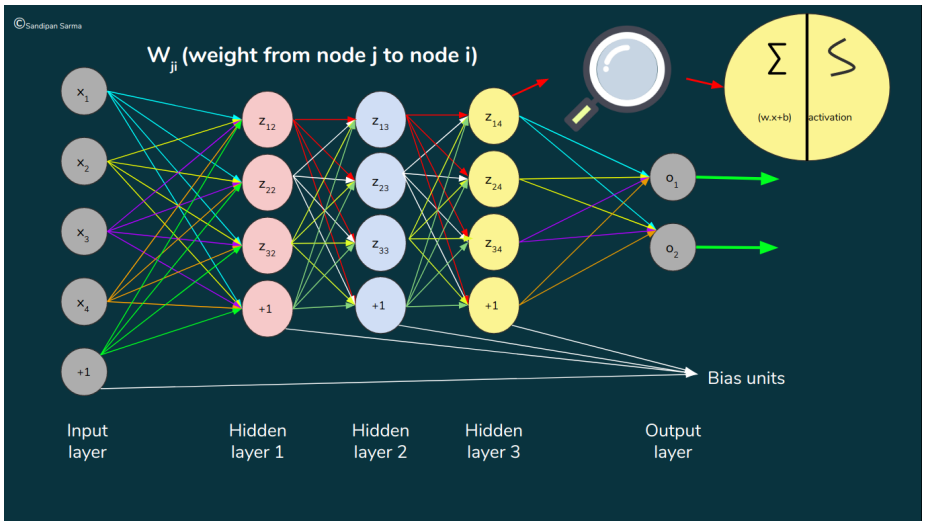


Figure 4: A typical neural network

Such feed-forward nets had been found to be working quite well for several applications like *handwritten character recognition, data compression, speech recognition, sonar target recognition*, etc. But they do not scale well to images. As an example, for *Netflix's* recommendation system, the dataset will primarily comprise of posters of movies and web series, as shown in Figure 5. Assuming a size of $1005 \times 1500 \times 3$ per image, each neuron in the first hidden layer of a feed-forward network will have $1005 \times 1500 \times 3 = 4{,}522{,}500$ connections, each corresponding to a weight (W_{ji}), and similarly a high number of neurons in the other layers too. When they add up, the total number of parameters is exceedingly high, and hence, such full-connectivity in each layer seems wasteful. It may also lead to overfitting (described later in the study).

Figure 5: IMDB 5000 movie dataset (Source: Kaggle)

To handle images in a more sensible way, **Convolutional Neural Networks** (ConvNets or CNNs) came into the picture. CNN layers arrange their neurons in a 3D volume (height, width and depth), and neurons in a layer are only connected to a small region of the layer preceding it (called the *local receptive field*). A simple real-life has been shown in Figure 6 and Figure 7, that explains how convolution works - an aerial search operation has to be undertaken, and several helicopters are carrying it out using searchlights. The process has been shown in the aforementioned figures, and the respective CNN analogies have also been given (although, the example has been shown for a 2D search area, while images are generally 3D (width, height and number of channels). The entire example culminates in the construction of a single ConvNet layer (3D volume). Assuming the area dimensions, the size of the 3D volume turns out to be: *(width * height * number of filters)* $= 4 \times 5 \times 3 = 60$ neurons, each having *filter size + 1 (for bias)* number of parameters $= 4 \times 4 + 1 = 17$ parameters. Most importantly, *neurons across the same filter share the same parameters* - this reduces the parameter count drastically in a CNN, and allows the filter to look for the same pattern in different sections of the image. A typical CNN architecture also includes some other layers than mere ConvNet layers, which will be discussed in the next section.

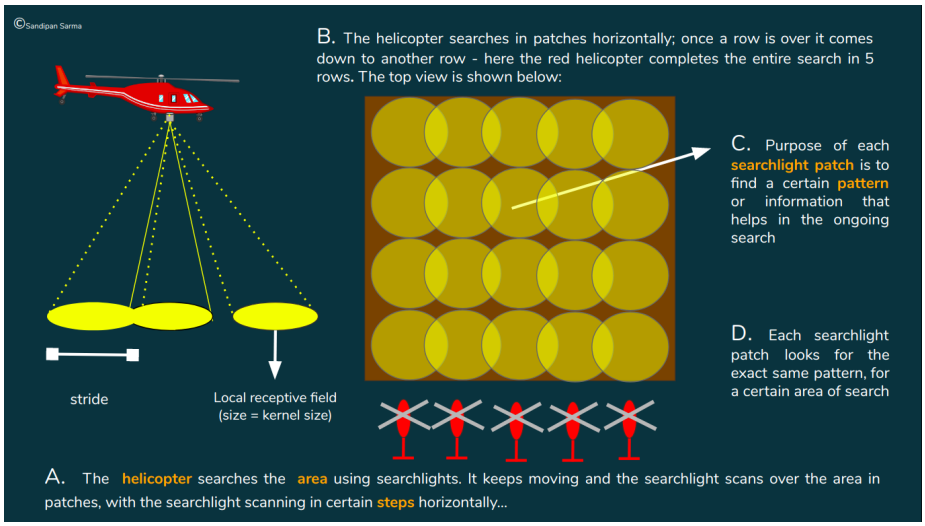


Figure 6: Understanding CNN via search operation (part 1)

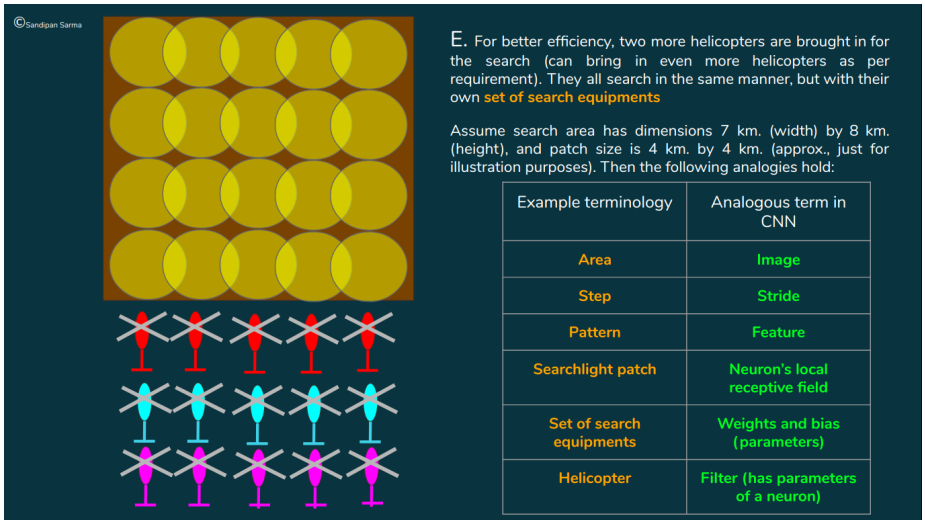


Figure 7: Understanding CNN via search operation (part 2)

CNNs first arrived to the scene way back in 1998, but attracted the research community's attention towards themselves, and more importantly towards deep learning, when **AlexNet** (an improved CNN) won the Imagenet challenge in 2012. The upcoming sections present a deep study on AlexNet.

3 AlexNet: The revolutionary CNN

3.1 Motivation

Until deep learning arrived to its full potential in the 2010s, vast, labeled datasets were not available for object-recognition tasks. The available datasets till then, like *MNIST*, were worked with and the best results were quite close to the human performance, but these datasets did not have enough variability incorporated within them to satisfy the objective of a real-life object-recognition task. But with the advent of datasets like *Imagenet*, which contained millions of labeled images from thousands of categories of objects, object-recognition became both meaningful and necessary. Figure 8 shows the stark difference in variability between the datasets:

Figure 8: Samples from MNIST (left) and Imagenet

Such large datasets need powerful models, like CNN, with large learning capacity. CNNs, as compared to similarly-sized feed-forward neural networks, are more efficient due to fewer connections and parameters, and hence are easier to train while significantly preventing overfitting. Moreover, availability of improved GPUs in the market made it possible to work with CNNs for a huge dataset like Imagenet, but yet, highly-optimized ConvNet layers were needed to be included in the network so that both the training time and performance metrics could be improved. AlexNet was aimed at achieving this objective.

3.2 The input: Imagenet dataset

Good research needs good resources to start with - hence came forth the Imagenet project, with an objective of providing the image and vision research field with a large-scale, high resolution, labeled image database. It consists of roughly 1.2 million training images, 50,000 validation images and 150,000 testing images, spanning over roughly 22,000 object categories. ILSVRC uses a subset of it, having images from 1000 categories. The aim is to classify an input

image to one of theese 1000 categories. The images in the dataset have a variety of resolutions, but the input to AlexNet was considered to have a constant resolution. Hence, some **preprocessing** was done on the images:

1. Images were converted to a fixed resolution of 256 $\times$ 256 as shown in Figure 9:

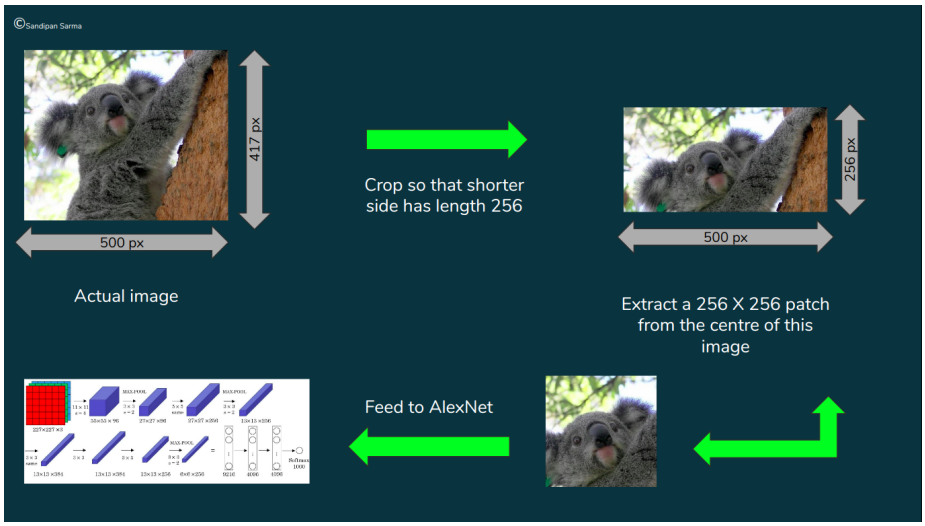


Figure 9: Converting an image to required dimensions

2. Mean centering of pixel values in the RGB space over the entire training dataset.

4 Novelties in the architecture

Inside the network, the 256×256 image is cropped and a resulting image of 227×227 is fed as an input to the first layer of the CNN (all the dimensions and values being used here in the paper were found to be optimal while implementing AlexNet). Before actually diving into the network as shown in Figure 11, it is important to know about the major contributions and operations in AlexNet, which made it different from all the other CNN architectures in use at that time, and hence gave a whooping improvement in performance.

4.1 Choice of non-linearity

In the traditional neural models, the activation functions used for modelling a neuron's output had been generally the Sigmoid function ($f(x) = (1 + e^{-x})^{-1}$) or tanh function (as also shown in Figure 4 - the activation function symbol shown depicts a sigmoid() or tanh()). But upon experiment, the authors of

AlexNet found that these saturating non-lineariites are much slower than the non-saturating non-linear function of ReLU ($f(x) = max(0, x)$ in terms of training time, while optimizing the loss function being used in the neural model. A plot (shown in Figure 10) taken from the original paper of AlexNet [1] suggests that for training large CNNs, ReLU is relatively more suitable than the traditional activation functions.

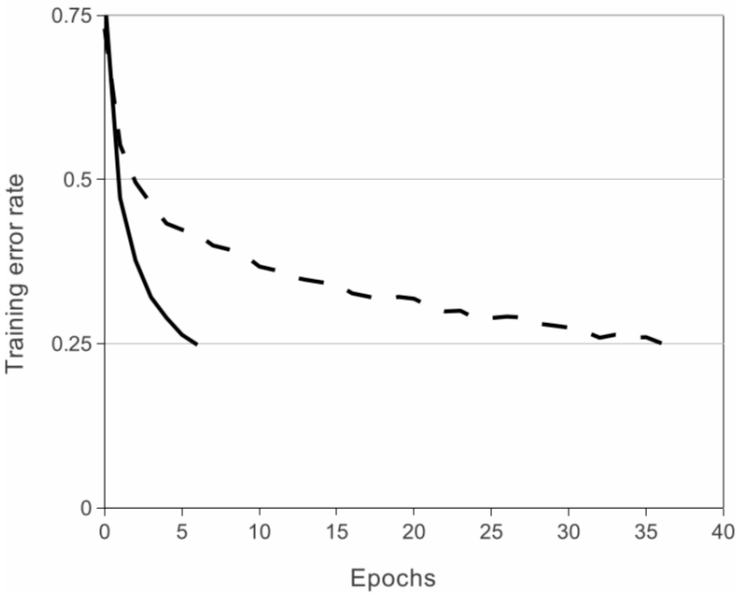


Figure 10: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25 % training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line). The learning rates for each network were chosen independently to make training as fast as possible. 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.

4.2 Training on multiple GPUs

For fast training, the training was done on two GTX 580 GPUs, equipped with cross-GPU parallelization - half of the kernels (or neurons) were put on each of the GPUs, with the GPUs communicating in only certain layers. For the actual distributed neural network architecture, refer to [1]. The authors report that this scheme reduced their 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, which is a desirable improvement.

4.3 Local response normalization

If at least some training examples produce a positive input to a ReLU, learning still happens in that neuron (although there might be possibilities of the dying ReLU problem [2] at times). So, generally, input normalization is not required for preventing ReLUs from saturating. However, it had been observed that a local normalization scheme [1] reduces the top-1 and top-5 error rates by 1.4 % and 1.2 % respectively- hence this scheme was applied after ReLU in certain layers.

4.4 Overlapping pooling layers

Pooling layers are introduced in the CNN to progressively reduce the spatial dimensions of the output of a convolutional layer. This reduces the number of parameters and the computational complexity of the network, hence also controlling overfitting. The most common type of pooling used is *max pooling*, although other kinds of pooling strategies like *average pooling* and *L2-norm pooling* are also there [1]. The authors in [1] have used overlapping max pooling ("summarizing" regions of the convolutional activation map in a kernel by setting the stride of the pooling filter, say s, to be less than the dimension z of a $z \times z$ pooling filter). This reduced the top-1 and top-5 error rates by 0.4% and 0.3%, respectively, as compared to non-overlapping pooling layers. Also, it has been reported that overlapping pooling layers reduce the chance of overfitting to some extent.

5 The groundbreaking architecture of AlexNet

- As shown in Figure 11, the network consists of eight **trainable layers**:
  - five **convolutional** (referred as CONV in Figure 11) **layers** which work as feature extractors, and
  - three **fully-connected layers** at the end of the network, whose job is to take the outputs of the pooling/convolution maps and use them for classification.

  All the trainable layers are followed by a **ReLU** activation layer, except for the last layer, where a **softmax** classifier intakes the output of the final fully-connected layer and produces a probability distribution over the 1000 object categories which indicates a confidence score of the resulting classification.

- AlexNet contains several **non-trainable** layers too:
  - three **maxpooling layers**, which progressively reduce the number of parameters in the network and reduce the size of the output maps;

[1] Find here at: https://cs231n.github.io/convolutional-networks/

– two **normalization layers** after the first and second non-linearity layers, as per the novelty introduced in section 4.3; and

– **dropout layers** for the first two fully-connected layers, for controlling overfitting (it will be discussed in the following section).

The following figure shows the way an input image is processed inside the network to produce an object class as the output:

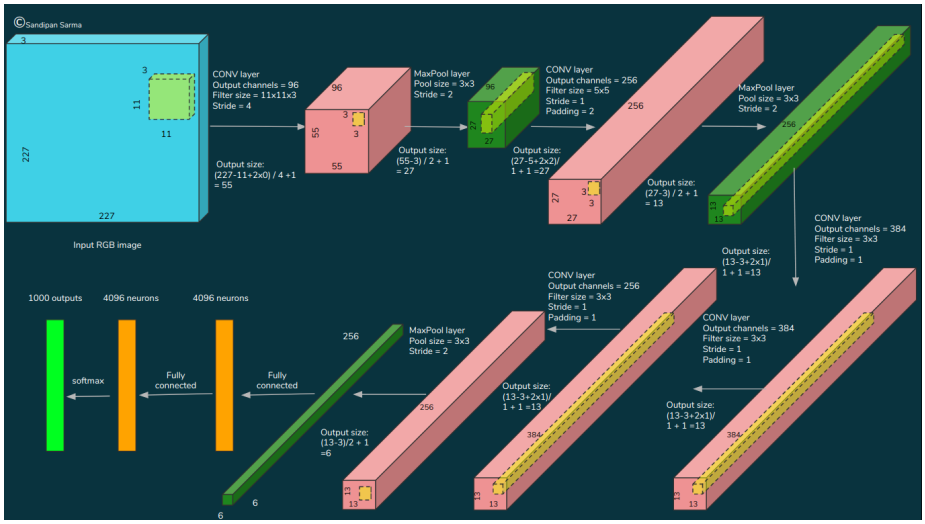


Figure 11: AlexNet architecture

6 Dealing with overfitting

Overfitting happens in a classifier model if during training, the model tries too hard to learn the decision boundary and turns out memorizing the training samples instead, rather than developing good generalization properties - hence failing to classify unseen data correctly. So, it is important to reduce overfitting as much as possible in the models. While building AlexNet, overfitting has been tackled in two primary ways:

6.1 Data augmentation

The idea behind this is that if multiple variations of the same sample are shown to the classifier, it prevents "hard memorization" of that particular sample - hence avoiding overfitting. With this in mind, some label-preserving transformations can be applied on the training samples to create artificial data, increasing the size of the dataset. Two approaches to data augmentation have been undertaken in [1]:

1. **By generating image translations and horizontal reflections**: Images of size 227×227 and their corresponding horizontal reflections were extracted from 256×256 images - training is done on these patches (as shown in Figure 11). Doing this has substantially reduced the chances of overfitting, considering the size of AlexNet.

2. **By altering the intensities of the RGB channels of the training images**: This scheme approximately captures an important property of natural images - object identity is invariant to changes in the intensity and color of the illumination. This reduced the top-1 error rate by over 1%. For detailed explanation, refer to [1].

6.2 Incorporating dropout

Using ensemble methods [3] for reducing test errors of classifiers works well with traditional machine learning models, but with huge neural networks, it will take even more time to train. So, in order to have the effect of using "combined neural network models" without having to use ensemble methods, the dropout technique was used:

- Here, the outputs of the neurons in the hidden layer are set to zero with some probability d - so these neurons do not participate in the learning process.
- Hence, in every pass when an input is fed to the network, AlexNet samples a different architecture.
- Although using dropout substantially prohibits the network from overfitting, it has been observed that it roughly doubles the number of iterations required for the classifier to converge.

7 Implementation

A self-implemented AlexNet has been uploaded in the GitHub repository of the author of this work [2]. Due to resource constraints, the trained model has been trained on relatively smaller datasets and some steps have also been omitted, which will be described in the following sub-sections.

7.1 Training environment

The training was done in *Google Colaboratory*, having the following settings:

- RAM: 12.72 GB
- Hardware accelerator: GPU
- Deep learning platform: Tensorflow 1.15.2

[2] https://github.com/sandipan211/DL-templates/tree/master/AlexNet

7.2 Dataset and model architecture

- **Dataset**: CIFAR-10 has been used as the dataset to work on.
- **About the architecture**:
  - **Learning rate** was kept at 0.00005.
  - The default **batch-size** was kept as 64.
  - A zero-mean Gaussian with a standard deviation of 0.01, as proposed in the paper, was used as the **weight initializer** in all the trainable layers.
  - **Local response normalization** has also been used with the parameters as specified in [1].
  - **Data augmentation** has been avoided due to lack of requisite resources.
  - **Mean centering** has been avoided as it has been inherently taken care of in a resizing function used in the code.
  - **Adam** [4] was used as the **optimizer** instead of Stochastic Gradient Descent as proposed in the paper.
  - softmax_cross_entropy_with_logits_v2 , taken from the Tensorflow library, was used as the **loss function**.
  - Early stopping was used to avoid overfitting.

8 Observations

- **Effect of batch size**: The implementation of AlexNet has been done with mini-batch training. Batch sizes of 64 and 128 have been tested, and no major change on the training accuracy or training time has been found. Although, a batch size too high might result in decreasing accuracy as well as poor generalization ability of the model.
- **Varying the loss function**: On conducting a survey on the experiments done by various researchers on AlexNet, for the CIFAR-10 dataset the performance of different loss functions can be seen in Figure 12.
- **Various datasets**: Researchers have worked on a lot of datasets while implementing AlexNet - some of which include - *CIFAR-10, CIFAR-100, MNIST, Fashion-MNIST, Imagenet, CIFE, GaMO*, etc.
- **Training time**:
  - It took about 93 minutes to train the model with the environment specifications as mentioned in Section 7.1. Early stopping has been used in the implementation to avoid overfitting by keeping track of the validation loss each epoch. The model stopped early at 10 epochs,

since there had been no improvement in the validation loss for k successive epochs (here k has been set to 2). So, each epoch took a training time of about 9.3 minutes.

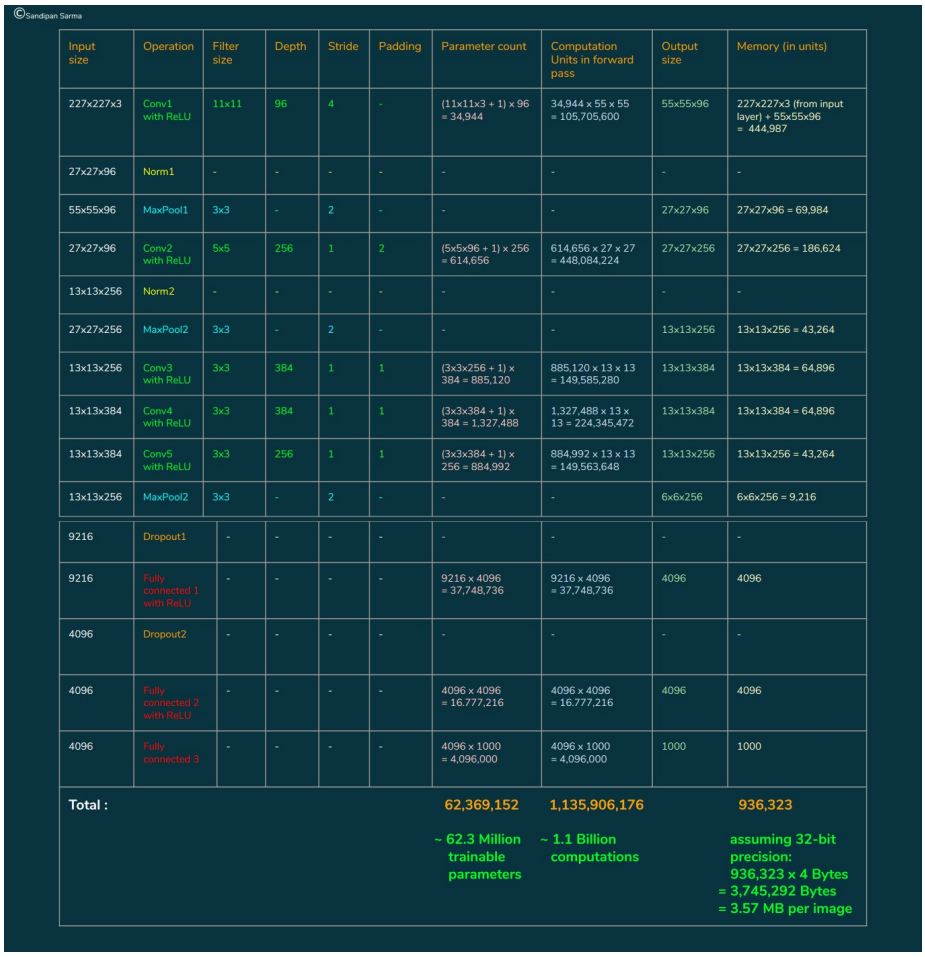
Loss Function/Model	AlexNet
Categorical Cross Entropy	0.6574
Class-wise Hinge	0.6707
Cosine Similarity	0.6164
Pairwise Hinge	0.5904
Hadsell	0.6544
Double Margin	0.6548
Siamese with Global	0.6593
Triplet	0.6552
Distance Ratio	0.6574
SoftPN	0.6616
Improved Triplet	0.6411
Large Margin Nearest Neighbor	0.6545
Lifted Structured Softmax Similarity	0.6542
Triplet with Global	0.6485
Histogram Loss	0.6577
Categorical Cross Entropy + Class-wise Hinge	0.6532
Categorical Cross Entropy + Triplet	0.6493

Figure 12: Loss function comparisons (accuracy may vary depending on the implementation)

- In order to understand the time distribution there might have been, for the different layers in the model, the number of *trainable parameters* of each layer needs to be taken into account - the higher the parameters in a layer, the more time it will take to train, intuitively. The details of the parameters have been presented in Figure 13.
- It can be found out from Figure 13 the convolutional layers add up 3,747,200 parameters, whereas the fully connected layers add up 58,621,952 parameters - that is, about 6% and 94% of the network's **trainable parameters** respectively.
- However, in terms of **number of computations**, the convolutional computations are far greater (about 1.077 billion - consisting of about 95% of the total) than the fully connected computations.
- The convolutional **neurons** (650,080, which is about 650,000 as said in [1])also take up most of the **memory space** (about 69.4% of the total required space per image). However, if we also consider the parameters to store, then fully connected layers eat up most of the

space. Each training image will take up about 3.57 MB of space in the memory during training, and this is only during the forward pass. These have to be stored in the memory after the forward pass to enable *backpropagation* through the network.

– The maximum training time is generally taken up during the back-propagation of errors in the fully connected layers. Although, it will also depend on the huge number of computations in the convolutional layers - this again depends on the number of filters in the layer being considered and the size of the input given to it.



©Sandipan Sarma

Input size	Operation	Filter size	Depth	Stride	Padding	Parameter count	Computation Units in forward pass	Output size	Memory (in units)
227x227x3	Conv1 with ReLU	11x11	96	4	-	(11x11x3 + 1) x 96 = 34,944	34,944 x 55 x 55 = 105,705,600	55x55x96	227x227x3 (from input layer) + 55x55x96 = 444,987
27x27x96	Norm1	-	-	-	-	-	-	-	-
55x55x96	MaxPool1	3x3	-	2	-	-	-	27x27x96	27x27x96 = 69,984
27x27x96	Conv2 with ReLU	5x5	256	1	2	(5x5x96 + 1) x 256 = 614,656	614,656 x 27 x 27 = 448,084,224	27x27x256	27x27x256 = 186,624
13x13x256	Norm2	-	-	-	-	-	-	-	-
27x27x256	MaxPool2	3x3	-	2	-	-	-	13x13x256	13x13x256 = 43,264
13x13x256	Conv3 with ReLU	3x3	384	1	1	(3x3x256 + 1) x 384 = 885,120	885,120 x 13 x 13 = 149,585,280	13x13x384	13x13x384 = 64,896
13x13x384	Conv4 with ReLU	3x3	384	1	1	(3x3x384 + 1) x 384 = 1,327,488	1,327,488 x 13 x 13 = 224,345,472	13x13x384	13x13x384 = 64,896
13x13x384	Conv5 with ReLU	3x3	256	1	1	(3x3x384 + 1) x 256 = 884,992	884,992 x 13 x 13 = 149,563,648	13x13x256	13x13x256 = 43,264
13x13x256	MaxPool2	3x3	-	2	-	-	-	6x6x256	6x6x256 = 9,216
9216	Dropout1	-	-	-	-	-	-	-	-
9216	Fully connected 1 with ReLU	-	-	-	-	9216 x 4096 = 37,748,736	9216 x 4096 = 37,748,736	4096	4096
4096	Dropout2	-	-	-	-	-	-	-	-
4096	Fully connected 2 with ReLU	-	-	-	-	4096 x 4096 = 16,777,216	4096 x 4096 = 16,777,216	4096	4096
4096	Fully connected 3	-	-	-	-	4096 x 1000 = 4,096,000	4096 x 1000 = 4,096,000	1000	1000
Total :						**62,369,152**	**1,135,906,176**		**936,323**
						~ 62.3 Million trainable parameters	**~ 1.1 Billion computations**		**assuming 32-bit precision: 936,323 x 4 Bytes = 3,745,292 Bytes = 3.57 MB per image**

Figure 13: AlexNet: Model summary

9 Results

- In the code implementation, the model has been tested on the CIFAR-10 dataset, yielding a test accuracy of 76.03%.
- From the loss and accuracy curves as shown in Figure 14, it is evident that overfitting and underfitting, both have been avoided to quite an extent.
- Also, in this implementation, provisions have been made to test on an entire test set, random k images, or a single image for quick class prediction. The results obtained have been shown in Figures 15 and 16.

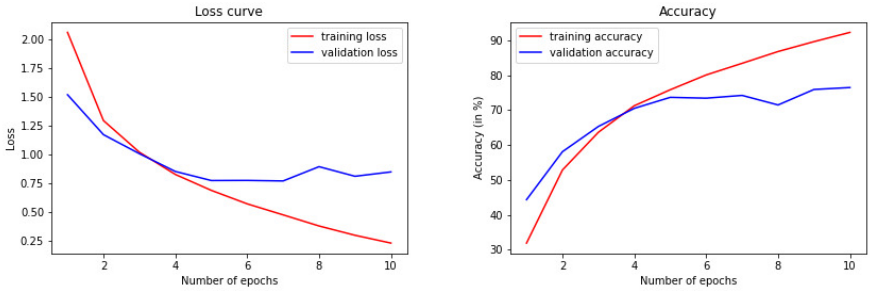


Figure 14: Loss plot (left) and accuracy plot

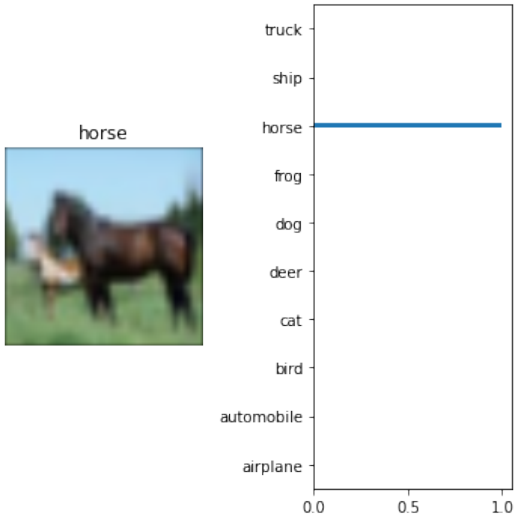


Figure 15: Object class prediction on a randomly given image

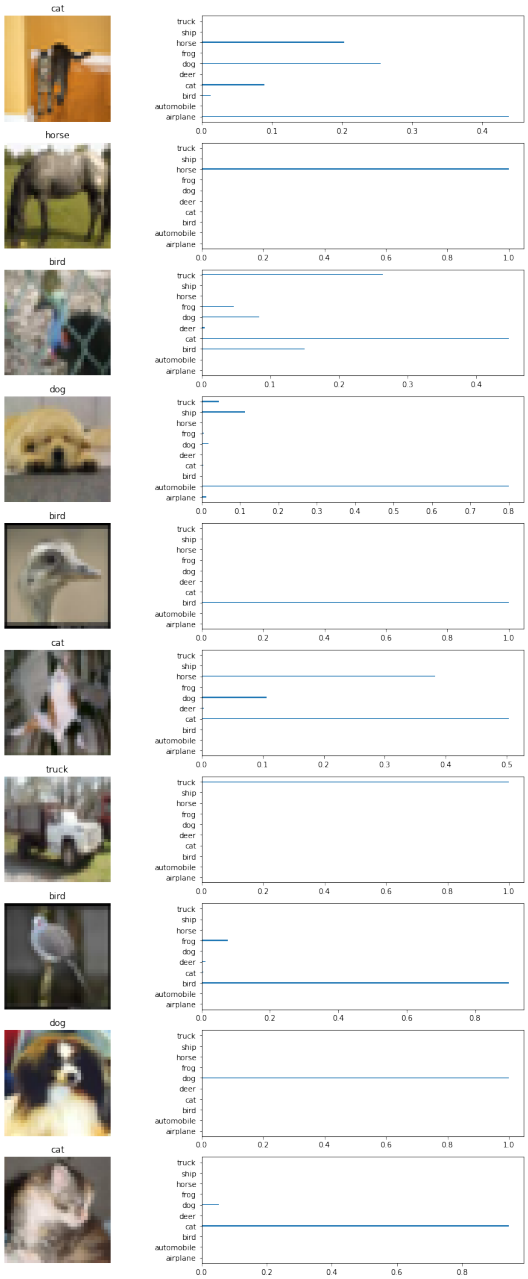


Figure 16: Object class prediction on 10 random images

10 Conclusion

In an era where the computer hardware industry had not made much progress, Krizhevsky et al. came up with a large, deep convolutional network which changed deep learning forever with its record-breaking results on image classification, by training on only two GTX 580 GPUs. However, an observation had been noted down in [1] that removing any of the middle layers of the network resulted in a loss of about 2% for the top-1 performance of the network. Hence, maintaining the network depth had been one of the things that the authors gave priority to. But thankfully, with the rapid advancement in technology, memory chips became cheaper in the market, which opened the gateway for numerous CNNs to come up in the upcoming years [5], having a lot of convolutional layers, acheiving better performance than AlexNet. Nevertheless, to this day, AlexNet remains one of the principal CNN architectures to be introduced to those interested in deep learning.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

[3] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.