ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

#### Paper Link

[ImageNet Classification with Deep Convolutional Neural Networks](https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf)

#### Paper Explained

[[Classic] ImageNet Classification with Deep Convolutional Neural Networks (Paper Explained)](https://www.youtube.com/watch?v=Nq3auVtvd9Q)

#### Guide to AlexNet with Implementation in TensorFlow

<https://www.kaggle.com/code/blurredmachine/alexnet-architecture-a-complete-guide/notebook>[ref]

<https://www.kaggle.com/code/syedasifathasnain/alexnet-architecture-a-complete-guide> [ need access]

#### Summary

[7.1. Deep Convolutional Neural Networks (AlexNet)](https://d2l.ai/chapter_convolutional-modern/alexnet.html)

[pending]

## Dataset

Imagenet 2012, 22000 categories with 1000 samples from each category in the training set.

## Unique Features Introduced

### ReLU Nonlinearity

Activation functions in general are used to convert linear outputs of a neuron into [nonlinear outputs](https://www.machinecurve.com/index.php/2020/10/29/why-nonlinear-activation-functions-improve-ml-performance-with-tensorflow-example/), ensuring that a neural network can learn nonlinear behavior.

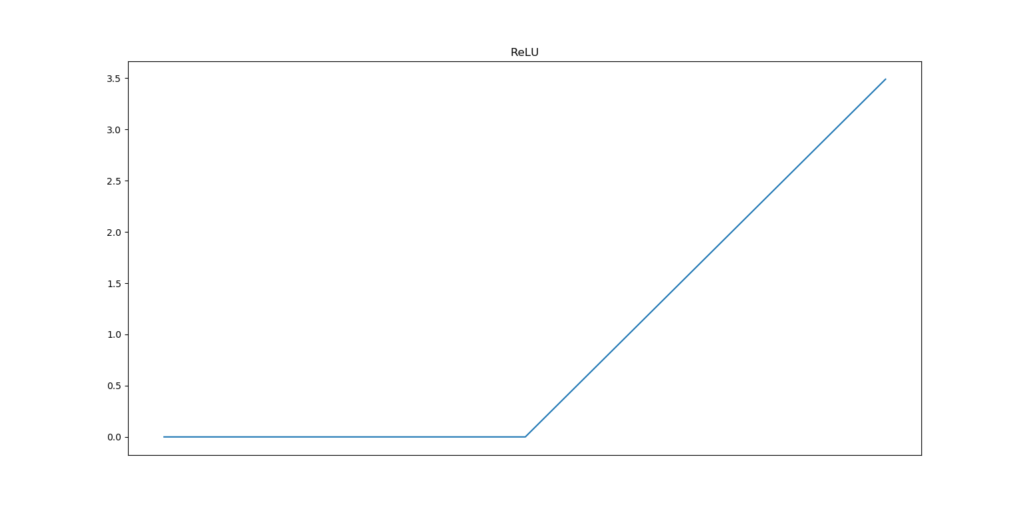
* ***Model Sparsity*** (DaemonMaker, n.d.). The less complex the model is during optimization, the faster it will converge. And *complexity* can be viewed as the *number of unimportant neurons* that are still in your model. The fewer of them, the better - or *sparser* - your model is.

**Sigmoid and Tanh** essentially produce non-sparse models because their neurons pretty much always produce an output value: when the ranges are (0, 1) and (-1, 1), respectively, the output either cannot be zero or is zero with very low probability. Hence, if certain neurons are less important in terms of their weights, they cannot be 'removed', and the model is not sparse.

* [**vanishing gradients problem**](https://machinecurve.com/index.php/2019/08/30/random-initialization-vanishing-and-exploding-gradients/) (DaemonMaker, n.d.). During optimization, data is fed through the model, after which the outcomes are compared with the actual target values to compute the loss. Since the loss can be considered to be an (optimizable) mathematical function, we can compute the gradient towards the zero derivative, i.e. the mathematical optimum. Neural networks, however, comprise many layers of neurons. We would essentially have to repeat this process over and over again for every layer with respect to the downstream ones, and subsequently chain them. That's what backpropagation is. Subsequently, we can optimize our models with gradient descent or a similar optimizer.

When neuron outputs are very small (i.e. -1 < output < 1), the chains produced during optimization will get smaller and smaller towards the upstream layers. This will cause them to learn very slowly, and make it questionable whether they will converge to their optimum at all: enter the *vanishing gradient problem*.

This activation function, named **Rectified Linear Unit** or ReLU, is much less sensitive to the problems mentioned above and hence improves the training process. It looks as follows:



And can be represented as follows:

f(x) = 0, if x < 0, x otherwise

Or, in plain English, it produces a zero output for all inputs smaller than zero; and x for all other inputs. Hence, for all inputs <= 0, it produces zero outputs.

##### Sparsity

As it produces 0, so unimportant neurons are made silent and the model gets simpler for optimization.

##### Fewer vanishing gradients

It also reduces the impact of vanishing gradients, because the gradient is always a constant: the derivative of f(x) = 0 is 0 while the derivative of f(x) = x is 1. Models hence learn faster and more evenly.

##### Computational requirements

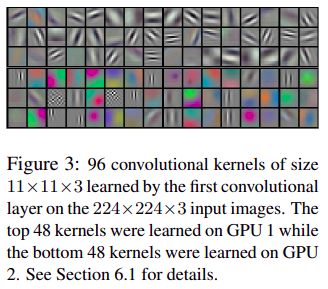
Additionally, ReLU does need much fewer computational resources than the Sigmoid and Tanh functions (Jaideep, n.d.). The function that essentially needs to be executed to arrive at ReLU is a max function: max(0, x) produces 0 when x < 0, and x when x >= 0. That's ReLU!

Now compare this with the formulas of the Sigmoid and tanh functions presented above: those contain exponents. Computing the output of a max function is much simpler and less computationally expensive than computing the output of exponents. For one calculation, this does not matter much, but note that in deep learning many such calculations are made. Hence, ReLU reduces your need for computational requirements.

**With ReLU, Alexnet made it possible to train a four layer CNN to train six times faster.**

RelU raises other issues like the dying RelU problem; you can read in more detail about activation functions [here](https://github.com/christianversloot/machine-learning-articles/blob/main/relu-sigmoid-and-tanh-todays-most-used-activation-functions.md).

### Training on Multiple GPU



Their approach here is very different than that in practice today. What we do is run mini-batches with models distributed on multiple gpus and later combine the output. But for AlexNet they split the model across 2 gpus and parallelization is done by cross communication in certain layers. That time the GPU had only 3gb of memory not capable of storing the entire model. They actually reduced top-1 and top-5 error rate by 1.7 and 1.2% respectively.

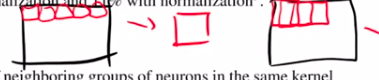
### Local Response Normalization

Maybe it got replaced by batch normalization. But has an impact on layer normalization.

For example, if we have 10 convolution layers, then LRN normalizes by averaging across, say 5 layers around the layer to be normalized. Now in layer normalization this kind of normalization in groups. It did not produce a large improvement but it did reduce some error rate.

### Overlapping Pooling

Not in practice now. Due to the stride of 2 it produces overlapping output the same size as input.

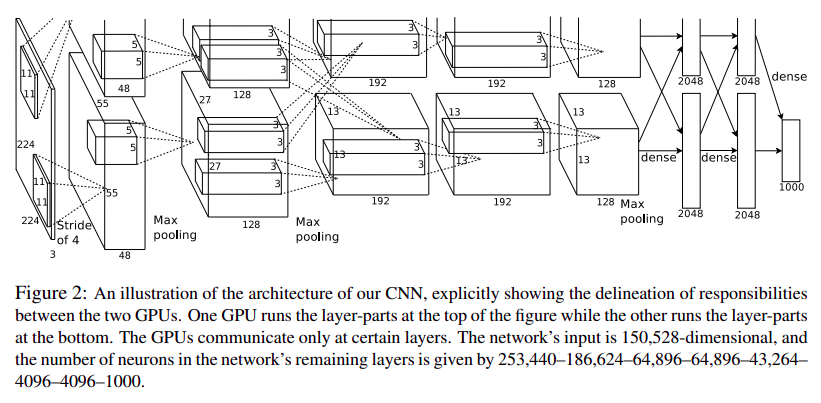


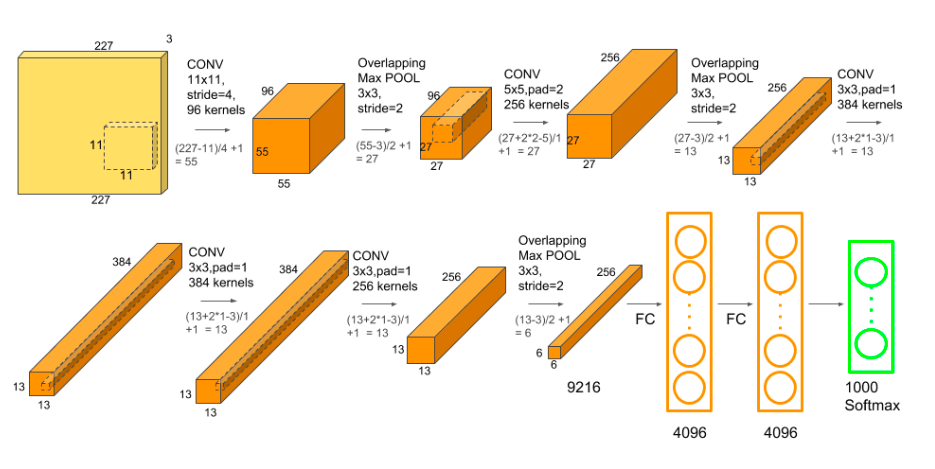
See the difference in model performance with various strides and kernel size in the link

[Overlapping pooling and overfitting – Mariam Mohamed](https://mariammohamedfawzy.wordpress.com/2018/04/23/overlapping-pooling-and-overfitting/)

### Architecture

* Max pooling is done to downsample the input image while convolving it. Nowadays Max pooling is not used, instead strided convolution is used to do the same.
* To reduce Overfitting, they used Data Augmentation(horizontal flip, random crop[224,224] from 256x256, pca based image augmentation) and Dropout Regularization with 0.5 value.
* Has 60 million parameters.





\*\*See the detailed network structure in tabular form [here](https://www.kaggle.com/code/blurredmachine/alexnet-architecture-a-complete-guide?scriptVersionId=39712880&cellId=4)

## Hyperparameters Setting

*We trained our models using* ***stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005.*** *We found that this small amount of weight decay was important for the model to learn. In other words, weight decay here is not merely a regularizer: it reduces the model’s training error*

*….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….*

*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 6 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****.*

## Evaluation And Result

