Classifying Images on Deep Convolutional Neural Networks

Sahil Ghanghas and Surbhi Garg
College of Computer and Information Science, Northeastern University, Boston, MA
{ghanghas.s, garg.sur}@husky.neu.edu

Abstract
How would a computer classify an image if it's asked to categorize random images? To provide an answer to this question we trained a deep convolutional neural network to classify a small Kaggle's dataset of 25K images with two classes. We trained several models from scratch with different hyperparameters and number of hidden layers. Our final model reached an accuracy of 91% while testing the trained network which close to human capability of classifying images i.e. 94%. Using the concept of transfer learning, a pre-trained model was used to make this classifier with an accuracy of 97% and a loss of 10.7%.

Introduction

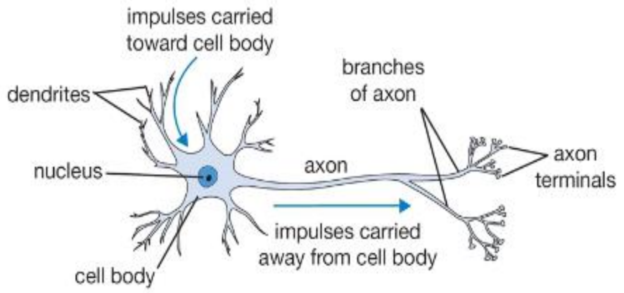
Current approaches to image classification make essential use of machine learning methods. To improve their performance, we can collect large datasets, learn more powerful models, and use better techniques for preventing overfitting. To learn about thousands of categories from millions of images, we need a model with a large learning capacity. However, the immense complexity of the image classification task means that this problem cannot be specified even by a large dataset, so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models.Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Background

Convolutional Neural Networks (CNNs)

Neural network is inspired by the biological neural system. The computational unit present in a biological neural system is a neuron. Fig 1 represents a basic neuron in brain.

In a neuron, the input is received through the dendrites and produces an output along the single axon present in it.The terminals of axon are connected to further neurons with the


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

synapses. In Fig 2 a mathematical model of the neuron

Fig 1. Biological Neuron

is presented, where xo is the axon of the previous neuron. The signals travel along xo through the synapse (*w_o*). (The signals that travel along the axons (e.g. x_o) interact multiplicatively (e.g. w_o x_o) with the dendrites of the other neuron

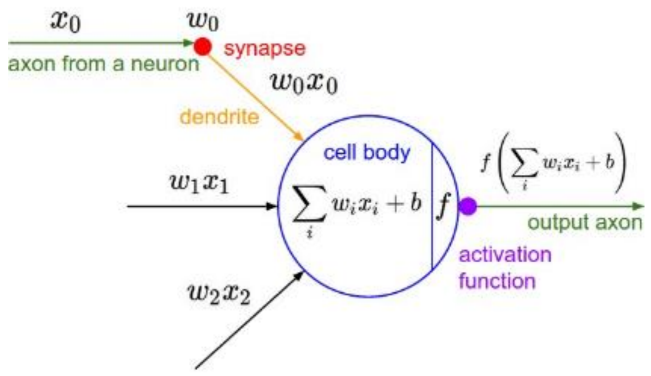


Fig 2. Mathematical Model

based on the synaptic strength at that synapse (e.g. w_o). The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence (and its direction: excitory (positive weight) or inhibitory (negative weight) of one neuron on another. In the basic model, the dendrites are responsible for carrying out the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we are not bothered about the precise timings of the spikes but frequency of the firing communicates information. Based on this rate code interpretation, we model the firing rate of the neuron with an activation function f, which represents the frequency of the spikes along the axon. Previously, a common choice of activation function is the sigmoid function σ, since it takes a real-valued input (the signal strength after the sum) and squashes it to range between 0 and 1)

Sparse Connectivity

CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. In other words, the inputs of hidden units in layer **m** are from a subset of units in layer **m-1**, units that have spatially contiguous receptive fields.

Shared Weights

In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a *feature map*.

Caffe

Caffe is a deep learning framework by BAIR *et al*. It can process over 60M images per day with a single Nvidia K40 GPU. That's 1 ms/image and 4 ms/image while training. Caffe is built on C++ and python. The code for caffe is written in .prototxt file where it is divided into three parts: deploy.prototxt, train.prototxt and solver.prototxt. The hyperparameters are defined in the solver file while the model architect of the convolutional neural network is constructed in the train file. We will discuss them in detail in the architecture section.

ReLU

Traditionally, sigmoid and tanh activation function have been used for neural networks. With respect to the training time, these nonlinearities have been much slower than ReLU. Fig 3 shows the results from Krizhevsky *et al*. Rectified Linear Unit is an activation layer where the activation function is defined as:

$f(x) = \max(0, x)$

where x is the input to a neuron.

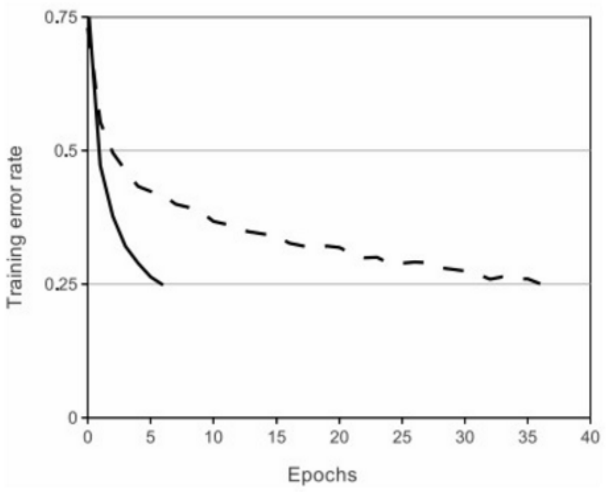


Fig 3 Krizhevsky's paper showing 6x improvement in convergence with ReLU.

Pooling

Pooling layer is present in between the convolutional layers. Its function is to decrease the spatial size which further reduces the number of parameters and computations in the neural network. There are several ways to implement this layer like taking the average or by taking the maximum of the 4 numbers in the region. The pooling layers also helps in controlling the overfitting problem. For our model, we are using max-pooling.

Overfitting

Overfitting is one the problems that occur in the neural networks. It occurs when you have trained your network and you provide it new dataset where the error occurred is greater than the training loss. This happens because the network trained on some parameters but it was not able generalize them over the other dataset. That is, when you train a large feed forward neural network on a small dataset, it is set to perform poorly on the test data.

The overfitting is reduced by randomly removing the feature detectors on each training case. To implement this in our network we have included the dropout (Hinton *et al*) in our architecture.

Project Description

The Dataset

Dogs vs. Cats is a dataset of 37.5K labelled images belonging to two categories of dogs and cats. Kaggle *et al* hosted this dataset for image classification competition where participants were to classify the dataset into either of the two

categories. Kaggle is a platform for predictive modelling and analytics competitions where researchers and companies post their data and data miners compete to produce the optimal models. The dataset is divided into 25K training images and 12.5K testing images.
The dataset consists images of variable resolution and our system requires constant input dimensions, therefore, we resized the images to 227 × 227. We did some image processing for adjusting the contrast of images using histogram equalization (Adil *et al*).

The Architecture

Our model consists of 8 layers. The layers are divided in 5 convolutional layers and 3 fully connected layers.

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels' width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will a full set of filters in each CONV layer and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume. Every Convolution layer is followed by an activation function RelU and a Polling layer and Normalization layer. Fully connected layer's neurons have full connections to all the activations in the previous layer. Their activations can be computed with a matrix multiplication followed by a bias offset.

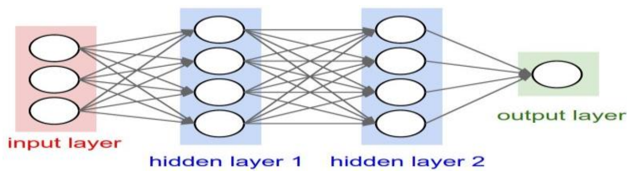
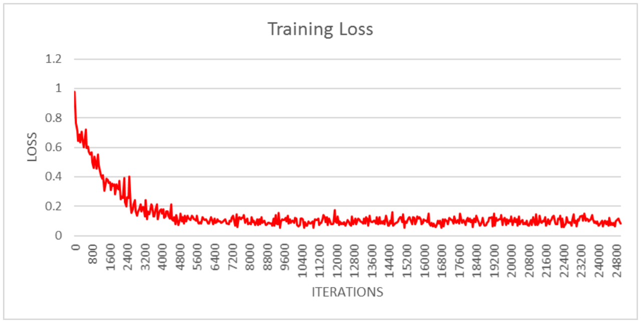


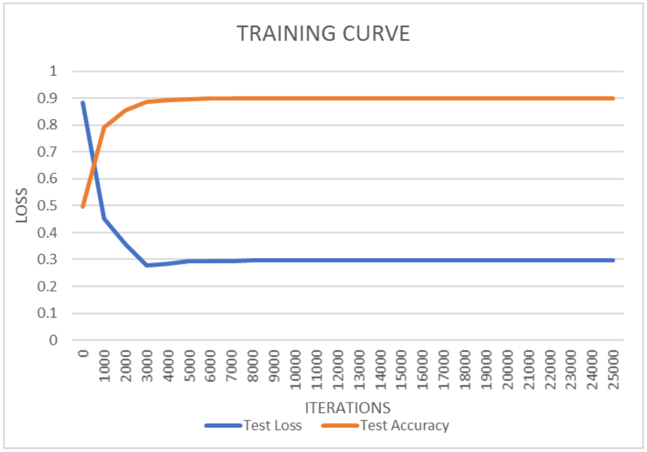
Fig. 4 Sample Layer in Neural Network

Experiment

Model 1

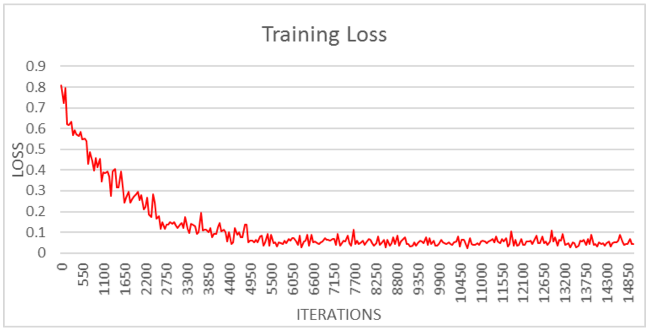


Graph 1. Training Loss

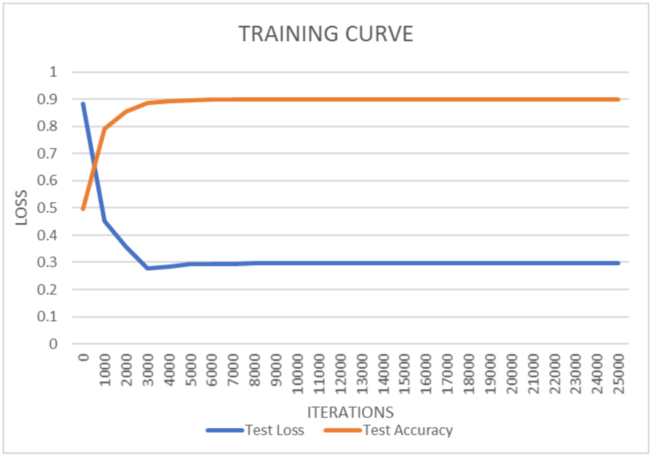


Graph 2. Training Curve

Model 2

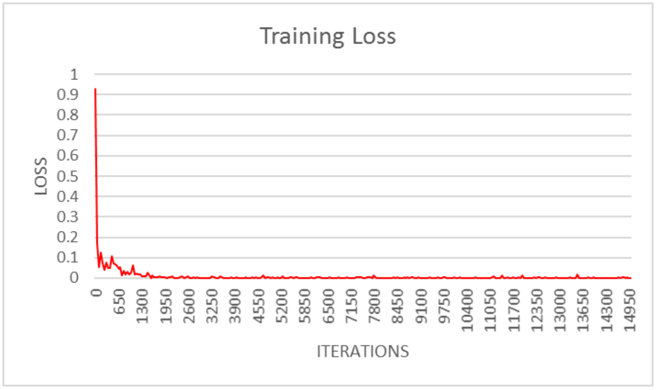


Graph 3. Training Loss

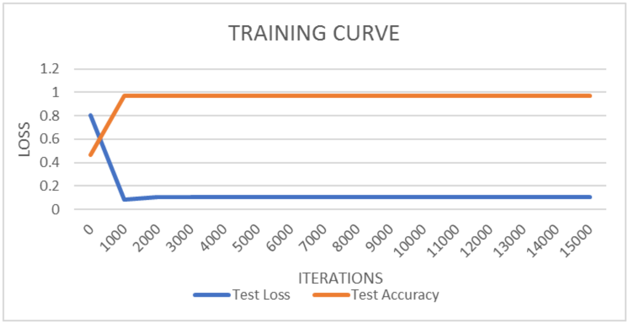


Graph 4. Training Curve

Model 3



Graph 5. Training Loss



Graph 6. Training Curve

For classifying the images, we took three models and kept the hyperparameters same for all of them. We took the lr_policy as "step" with stepsize = 2500 while keeping the base_lr = "0.001" and gamma = 0.1.

Before we started training the model, we need to fix the size of the images as the network requires a constant type of data as input to itself, so resized all the images and then provided the images as input to the neural network. For this we ran separate python for doing so. After resizing the images the next step is to calculate the mean of the images which used for classifying the images into their respective categories. This is a preprocessing which needs to be done in case of supervised learning in machine learning. For doing so, caffe has an inbuilt command compute_image_mean which generates a .binaryproto file which is used of training the network.

For model 1 number of hidden layers were kept at standard size with first layer of 92 hidden layers, second of 256, third and fourth with 384 and last layer with 256 hidden layers. The results we got for the model 1 are represented in the Graph 1 and 2. From the graph 1 we can see that when we start training the loss is high but with iterations it decreases and after 4500 iterations it becomes somewhat constant. Graph 2 represents training curve where accuracy is mapped alongside test loss.

For model 2 we changed the hidden layers after 1st and 2nd layer. We provided them with 192 and 384 hidden layers respectively. Just by changing the number of hidden layers we were increase our accuracy by 2% which is represented in Graph 4.

In model 3, we took an already trained model and passed it as weights to our existing model 1. This type of learning is known as transfer learning. The benefit of using an existing trained model is that first it helps us save time and provides better results than a model which have trained from scratch which is evident from our results as shown in the graph 5 and 6. For this purpose we the bvlc model present in the caffe library.

For experimental purpose, we changed the hyperparameters where we changed the base_lr to 0.01. This model resulted in accuracy of 50% even after 4000 iterations where previous models had already achieved saturated accuracy so we stopped the training of that network at that point. This type phenomenon can be seen when you train network and provide them with less hidden layers and you provide them with number of layers more then what is required. In deep learning networks if keep on increasing the hidden layers on even convolutional layers for better results you reach a point where increasing these parameters only degrades the accuracy of the network and also by increasing these layers the model itself takes longer to train which is not a good thing as the purpose of the training this network is to save time. Table 1 shows the accuracy results of the 3 models we trained for classifying images.

Model	Accuracy (%)
Model_1	89
Model_2(increased depth)	91
Model_3(bvlc_model)	97

Table 1. Accuracy Results

Conclusion

From the results, it is evident that by increasing the depth of the network by increasing the number of hidden layers, we can reduce the loss. By increasing the depth of the network, the accuracy increased by 2%.

Future Work

For the future work, we would be classifying the ImageNet dataset of 1.2M images into 1000 classes and after that we would identifying and recognizing objects present in the dataset.

References

Fei Fei Li, CS231n: Convolutional Neural Networks for Visual Recognition, *Standford University.*

Nikhil Buduma, Deep Learning in a Nutshell – what it is, how it works, why care? *http://www.kdnuggets.com/2015/01/deep-learning-explanation-what-how-why.html*

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, *Advances in Neural Information Processing Systems 25 (NIPS 2012)*

A. Berg, J. Deng, and L. Fei-Fei. Large scale visual recognition challenge 2010. *www.imagenet.org/challenges. 2010.*

Nikhil Buduma, Data Science 101: Preventing Overfitting in Neural Networks, *http://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html*

Larry Brown, Deep Learning for Image Classification, *Nvidia.*

Deep learning framework by BAIR, *http://caffe.berkeleyvision.org/*

Kaggle, *https://en.wikipedia.org/wiki/Kaggle*

G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580, 2012.*

Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. *ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015*

Adil Moujahid, A Practical Introduction to Deep Learning with Caffe and Python. *http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/*