Using Convolution Neural Network for Image Classification

An implementation of Transfer Learning to obtain excellent results

Muhammad Usama  
SEECS, NUST

[*muhammadosama118@gmail.com*](mailto:muhammadosama118@gmail.com)

Islamabad, Pakistan

*Abstract*

Convolution Neural Networks are extensively used for image classification applications. I am going to implement a CNN and try to model a classifier which will provide a descent accuracy and minimal loss while being trained on a limited data set. I will use techniques like Transfer Learning and I will justify the path I have followed to tackle the problem. Here our goal is to differentiate between images of Dolphins and Sea-horses. The dataset is downloaded from github and consists of less than 120 images. [1]



*Keywords: CNNs, ConvNet, Transfer Learning, AlexNet, DIGITS, fine tuning, Caffe.*

# **Introduction**

In [machine learning](https://en.wikipedia.org/wiki/Machine_learning), a convolutional neural network (CNN, or ConvNet) is a class of deep, [feed-forward](https://en.wikipedia.org/wiki/Feedforward_neural_network) [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network), most commonly applied to analyzing visual imagery.[4] Huge progress has been made in object recognition with deep convolutional neural networks (CNNs), thanks to the availability of large-scale annotated dataset.

In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it takes more time and more computation power and we don’t want to waste any of the above. Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest.[5][2][3]

Now just to give an overview of how we started taking on this task, I should talk about the dataset. In the beginning we prepare the dataset on DIGITS, best part about using NIVIDIA’s DIGITS is that we don’t actually have to label the images we just have to put them in separate directories, and pass them to DIGITS (after naming the directories). DIGITS can work with Torch, Caffe, Tensorflow, we are using Caffe based models here.

Once the dataset is ready we are going to start by training a model that uses our dolphins-and-seahorses dataset, and the default settings DIGITS provides. For our first network, we are going with AlexNet. [2]

AlexNet’s design won major Computer Vision competition called ImageNet in 2012. After analysis of the results we get from this Model, we are going to try to make another attempt to make the classifier a little better by tweaking it a bit, formally speaking we are going to fine tune the AlexNet Model.

Transfer Learning / Fine tuning takes advantage of the layout of deep neural networks, and uses pretrained networks to do the hard work of initial object detection.

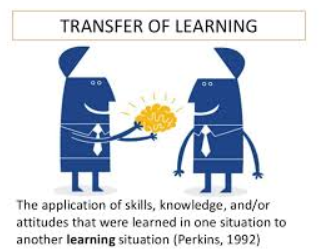
After fine tuning we will compare our initial results with the final results.

# **Transfer Learning**

## Transfer learning or inductive transfer is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.[9]

## For example, knowledge gained while learning to recognize animals could apply when trying to recognize elephants.

Hence formally speaking, **transfer learning is essentially a transfer of knowledge from a learned entity to a novice**. [6]



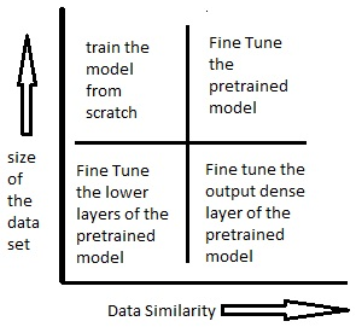
Hence we take a pre-trained model, which was trained by someone else and use it as a starting point to solve our problem.

# **How to fine tune ?**

Well there are multiple ways to fine tune an existing pre-trained network according to one’s needs. Let’s look at them and decide which one we’ll be using.

* Use Pre-trained Model as Feature Extractor
* Use Architecture of Pre-trained Model.
* Train some layers while freezing others.

The diagram below should help us solve the query. [6]



We know that our dataset is already small and it is also similar (both classes belonging to a bigger class ‘animal’).

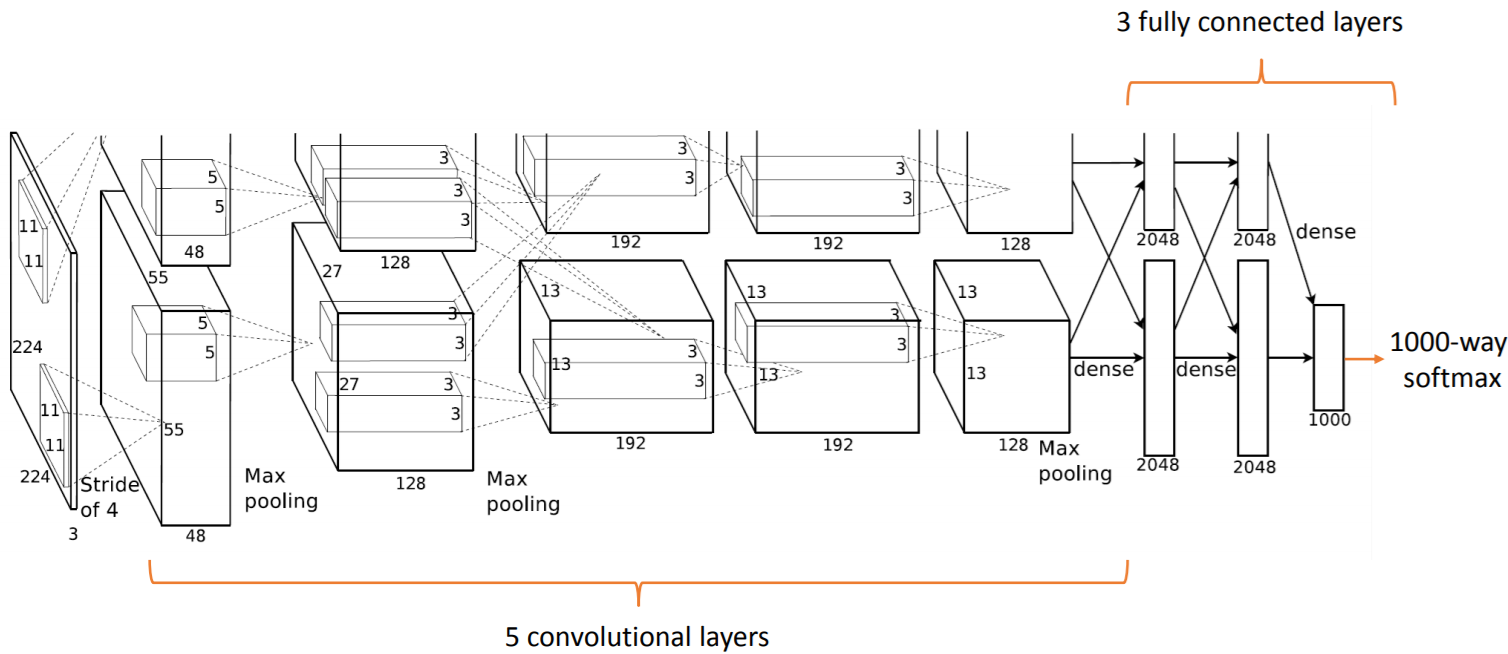
In this case, since the data similarity is very high, we do not need to retrain the model. All we need to do is to customize and **modify the output layers according to our problem statement.** We use the **pretrained model as a feature extractor**. [6][7]

Imagine using a neural network to be like looking at something far away with a pair of binoculars. You first put the binoculars to your eyes, and everything is blurry. As you adjust the focus, you start to see colors, lines, shapes, and eventually you are able to pick out the shape of a bird, then with some more adjustment you can identify the species of bird. [2]

In a multi-layered network, the initial layers extract features (e.g., edges), with later layers using these features to detect shapes (e.g., a wheel, an eye), which are then feed into final classification layers that detect items based on accumulated characteristics from previous layers (e.g., a cat vs. a dog).  [2]

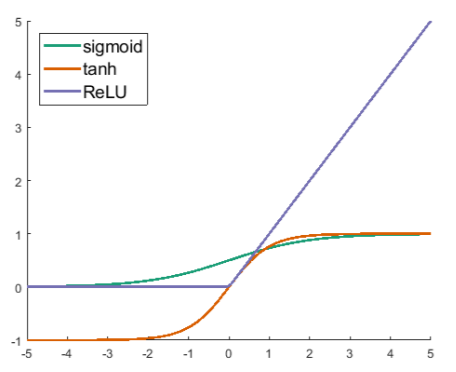
# **Architechture**

By default AlexNet has 8 layers, as shown below, we do not include pooling and activation layers in this counting. We can see the basic architecture below. Where, [8] [10]



Intuitively speaking Layer 1 is a convolution layer, Layer 2 is a max pooling followed by convolution and Layers 3,4,5 do the same. Layer 6,7 and 8 are the fully-connected layers. [8] [10]

The **activation function used is rel**u. It provides the non-saturating non-linearity. It does not saturate in the positive region and gives fast computations. The diagram below shows the other possible but relatively poor choices for activation functions as compared to relu since they all saturate in the positive region. [8]



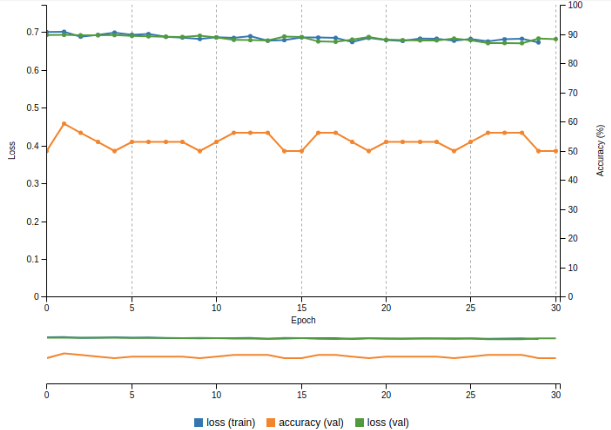
# **Methodology And Results**

Here we describe the two approaches we used.

## **We start from Scratch**

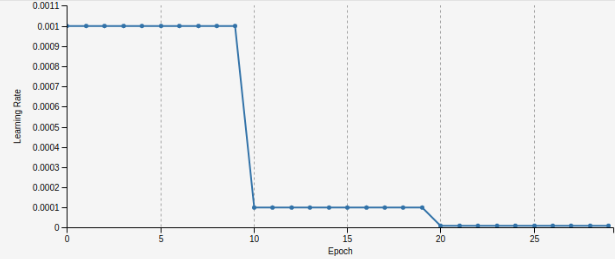
We start by using architecture of AlexNet (not fine-tuned). We get the following output after training.

**Epochs = 30**



## We can see the loss and accuracy plot during training, We can see that accuracy does not increases significantly over time.

We have also estimated the learning rate, it is plotted vs. epochs

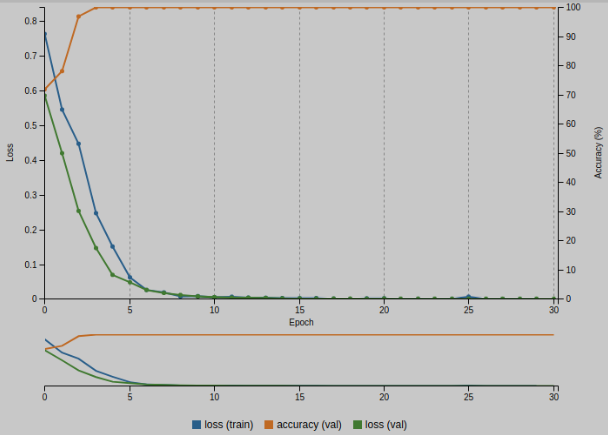


## **We fine-tune AlexNet**

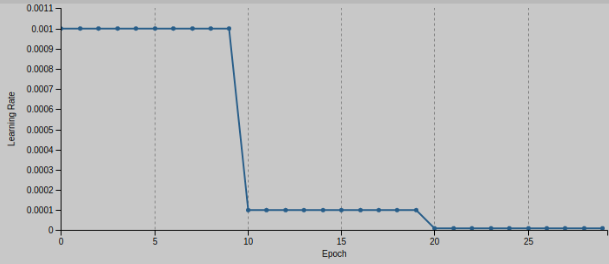
While fine-tuning we throw away the current final layer and introduce our own new fully connected layer and give it a new name. We redefine the outputs for the new fully connected layer.

**Outputs for last fully connected layer = 2**

Now we get the following output after training.



We can see the accuracy increases and loss drops over time during training.



Time taken in 2nd approach was about 37 minutes,

Whereas, in 1st approach. It took us 27 minutes.

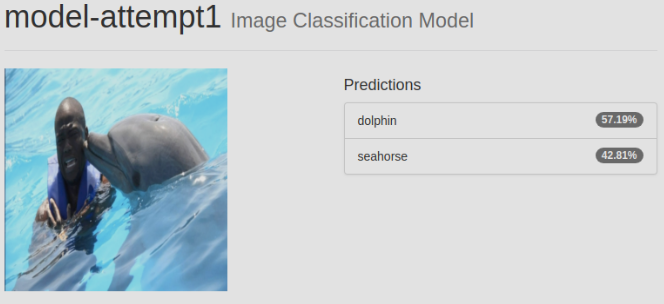
# **Tests and observations**

Here we will test the results we get from the trained models by taking random pictures of internet.

For **Seahorse & Dolphin**, **Model 1** gives the following results

# 

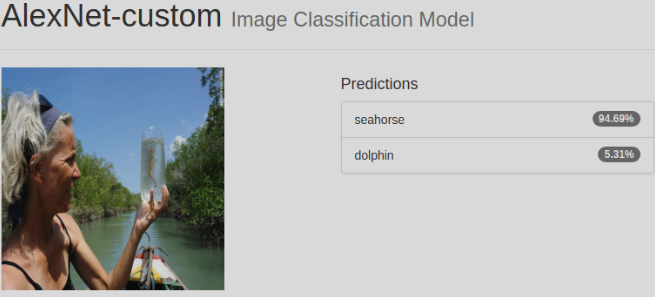
**Dolphin 55% and Seahorse 45%** Almost. This result is pathetic. By the way sea horse is inside the bottle she is holding.



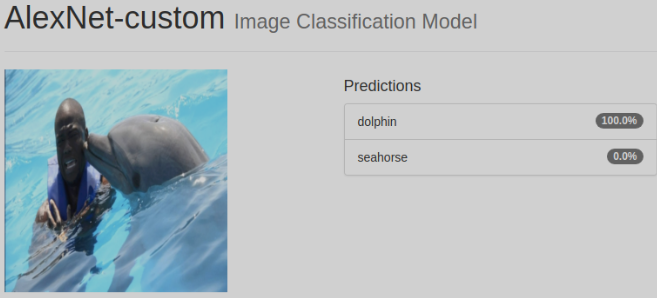
**Dolphin 57% and Seahorse 43%.**

These above results are **not satisfactory.**

**Model 2** gives the following results.



**Dolphin 5.31% Seahorse 94.69%**



**Dolphin 100% Seahorse 0%**

Results are really **good.**

# **Additional Information**

We used dockers repository to install Caffe and DIGITS. The data set was available at github [2] and the test images were taken from Google. The use of other people’s work is acknowledged in the reference section. Snapshots were taken in Ubuntu Linux and they have turned a little greyish in color.

# **References**

1. Transfer Learning with Deep Convolutional Neural Network for SAR Target Classiﬁcation with Limited Labeled Data

By Zhongling Huang 1,2,3, Zongxu Pan 2,3 and Bin Lei 2,3,\*

1. <https://github.com/humphd/have-fun-with-machine-learning>
2. <http://cs231n.github.io/transfer-learning/>
3. https://en.wikipedia.org/wiki/Convolutional\_neural\_network
4. <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
5. <https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/>
6. <https://github.com/BVLC/caffe/blob/master/models/bvlc_alexnet/train_val.prototxt>
7. <http://cvml.ist.ac.at/courses/DLWT_W17/material/AlexNet.pdf>
8. https://en.wikipedia.org/wiki/Transfer\_learning
9. https://sushscience.wordpress.com/2016/12/04/understanding-alexnet/