Machine Learning Engineer Nanodegree

Capstone Project

Rachit Kumar Agrawal
September 25th, 2017

I. Definition

(approx. 1-2 pages)

1. Project Overview

I have picked up the domain of Computer Vision for my capstone project. In last decade, the amount of data has exploded exponentially. These data are in different forms - text, image and video. With each of these forms of data exploding, it becomes extremely important to categorize them or classify them and classification of these images is not humanly possible. So, it becomes important to let machine do it for us. I am personally motivated for this problem because, I feel this is first step to my ultimate goal to create a prototype for the Autonomous Driving.

1.1 Dataset

In this work, I am using the publicly available CIFAR-10 dataset. This dataset was collected ty students at MIT and NYU over the span of six months; it is described in detail in [1]. They assembled it by searching the web for images of every non-abstract English noun in the lexical database WordNet [2][3]. Two popular subsets were created out of this large dataset of tiny images called – CIFAR-10 and CIFAR-100. I am using CIFAR-10 dataset for this work. This dataset has 6000 examples of each of 10 classes. Out of these 6000 examples from each class, 5000 examples are kept for training and the remaining 1000 are kept for the evaluation of the model.

2. Problem Statement

The problem I am trying to solve is - "Object Classification on CIFAR-10 dataset". The goal is to classify given RGB 32x32 pixel images across 10 object categories. The performance of this prediction is then compared with the label provided along with the dataset.

3. Metrics

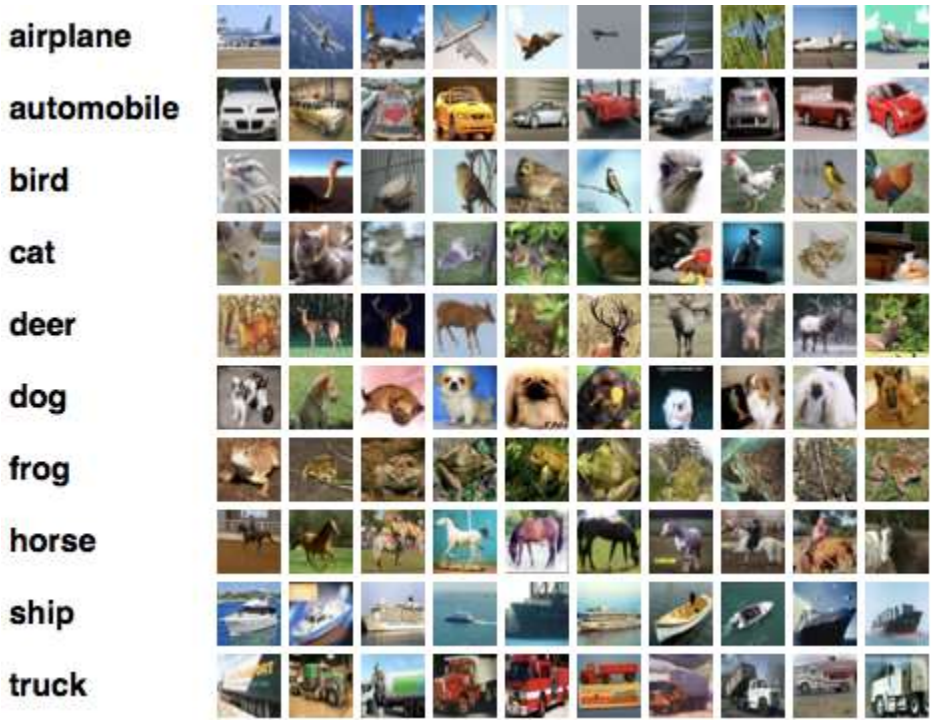
The evaluation metric used in this work is *accuracy.* The accuracy is defined as follows:

$$Accuracy = \frac{\text{Number of images correctly predicted}}{\text{Total images in the dataset}}$$

II. Analysis

(approx. 2-4 pages)

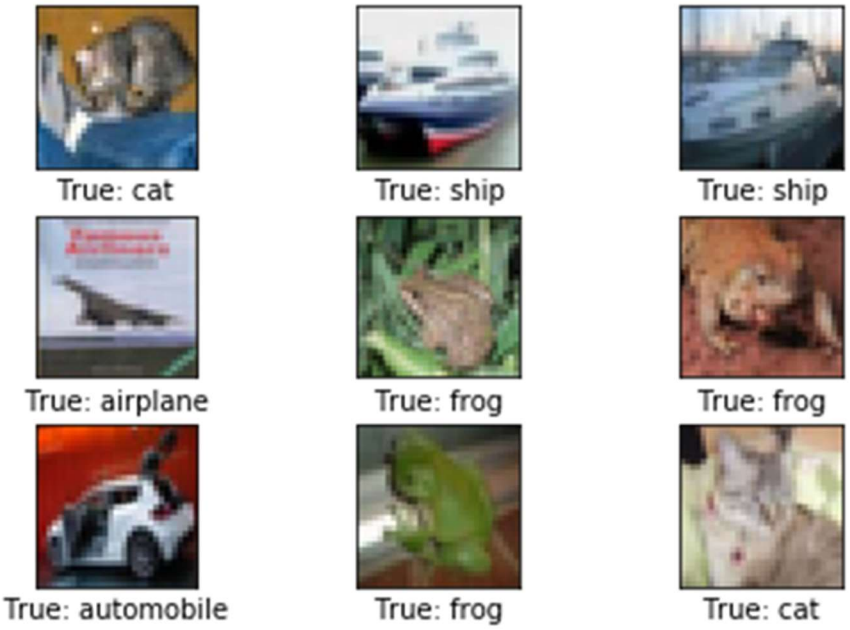
1. Data Exploration



The input images provided are 32x32 RGB images. Even though there are substantial number of images per class but they are not sufficient enough to get a good score on a CNN. For this reason, we have increased our training set by randomly doing some pre-processing on the input images. These pre-processing includes – cropping, rotating, flipping, changing the contrast etc.

2. Exploratory Visualization

We try to explore the images from the training dataset and see how they look like. In the image below, I have plotted the first 9 images from the dataset:



The first image here, shows a cat that's lying beside the owner's foot. So, even though most of the images are humanly possible to recognize the images correctly, some are still difficult to recognize even with human eyes. For example, the first image is of a cat but it is not easy to recognize it. And this is where CNN comes handy, it has been shown through various studies specially in the ImageNet contest that CNN is very useful in classifying the images. And we use CNN for the same reason.

3. Algorithms and Techniques

3.1 Convolution Neural Network

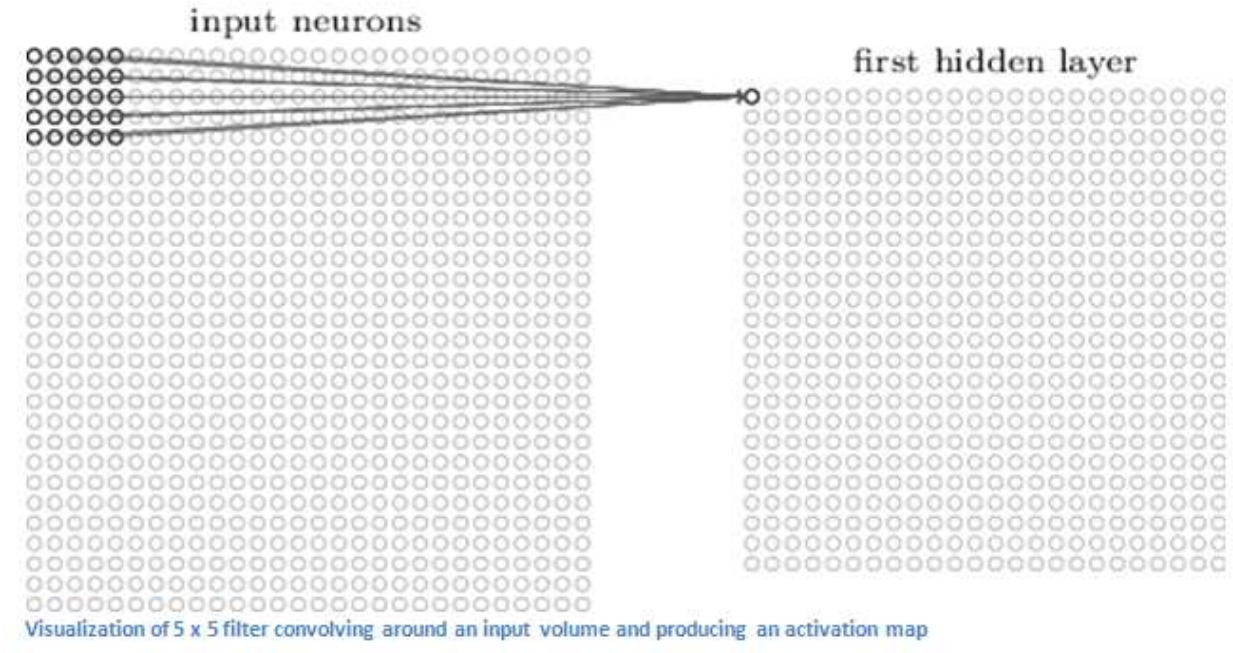
I plan to use Convolution Neural Network (CNN) for this project. In machine learning, a convolution neural network is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. CNNs use a variation of multilayer perceptron designed to require minimal preprocessing [4]. Convolution networks were inspired by biological processes [5].

A CNN consists of an input layer, a output layer and multiple hidden layers. The hidden layers can be either convolution(CNV), pooling(PL) or fully connected(FC). We will consider each of these layers in detail in the following section.

3.1.1 Convolution Layer (CNV):

The Convolution Layer is the core building block of a Convolution Network that does most of the computation heavy lifting. The CNV 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 CNV might have size

5(W)x5(H)xN(D). N, represent the number of features we want to extract in this layer from the images. For our model in this study, N is chosen as 64. During the forward pass, we slide(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. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), 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. An example of convolution operation is shown in figure below:



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

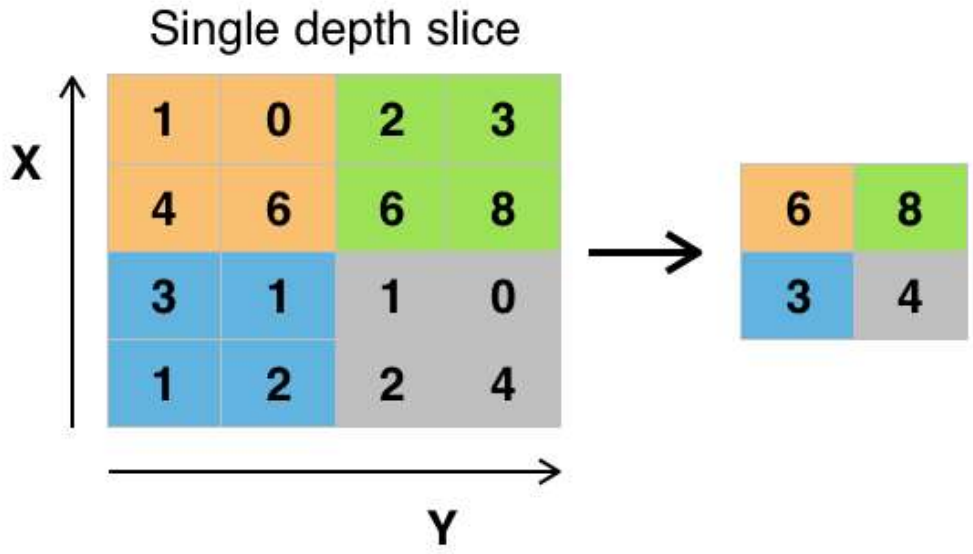
3.1.2 Activation Layer:

After each conv layer, it is convention to apply non-linear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations). There are many type of activation functions like *sigmoid, tanh and ReLU* but researchers found out that ReLU layers work far better because the network can train a lot faster without making a significant difference to the accuracy. The *tanh* and *sigmoid* function suffers from vanishing gradient problem because of the slope being too low for high numbers. ReLU doesn't suffer from this problem and hence it trains faster. The ReLU [6] layer applies the function f(x) = max(0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0. This layer increases the nonlinear properties of the

model and the overall network without affecting the receptive fields of the CNV layer. For these benefits, we have used *ReLU* activation function with all the CNV layer for this study.

3.1.3 Pooling Layer (PL):

After some ReLU layers, programmers may choose to apply a pooling layer. It is also referred to as a down-sampling layer. In this category, there are also several layer options, with *MaxPooling* being the most popular. This basically takes a filter (normally of size 2x2) and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every sub-region that the filter convolves around.



Example of Maxpool with a 2x2 filter and a stride of 2

Other options for pooling layers are average pooling and L2-norm pooling. The intuitive reasoning behind this layer is that once we know that a specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features. We have used a *MaxPooling* Layer of size 2x2 with a stride of 2.

3.1.4 Fully-connected layer (FC):

After several convolution and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. We have used two consecutive FC layer in our model with different neurons, details of which has been explained in the latter section.

3.1.5 Classification Layer (CL):

This is typically final layer of a CNN that transforms all the net activations in the final output layer to a series of values that can be interpreted as probabilities. One of the most common function is *softmax* function. The property of the *softmax* function is that it makes sure that the total probability of all the possible classes is 1.

3.2 Neural Network Framework

Neural Network Framework has become popular off late because these frameworks are more efficient than implementing this Neural Network using NumPy because these frameworks knows the entire computation graph that must be executed, while NumPy only knows the computation of a single mathematical operation at a time. These makes all the frameworks easy to use and much faster than standard python libraries.

We had multiple options to choose from among the various Neural Network framework available. I did my literature survey to compare these different frameworks and came up with the following summary:

Property	Caffe	Neon	TensorFlow	Theano	Torch
Core	C++	Python	C++	Python	Lua
CPU	Yes	Yes	Yes	Yes	Yes
Multi-threaded CPU	Blas	Only Data Loader	Eigen	Blas, conv2D, limited OpenMP	Widely Used
GPU	Yes	Customized nVidia backend	Yes	Yes	Yes
Multi-GPU	Only Data Parallel	Yes	Most Flexible	No	Yes
Nvidia cuDNN	Yes	No	Yes	Yes	Yes
Quick Deploy, on standard Models	Easiest	Yes	Yes	No	Yes
Auto Gradient Compu	Yes	Yes	Yes	Most flexible	Yes

Looking at the above comparison metric, *Torch* and *TensorFlow* are the top two choices. But TensorFlow has a better support for Multi-GPU. This is a very important metric because, GPUs are heavily used for the training and having better support for multi-GPU means huge time savings. Due to this, I chose TensorFlow as the framework to learn and use for my project.

4. Benchmark

There have been many solutions for CIFAR-10 dataset. Among these, the state-of-the-art solution has been provided by Graham, Benjamin called Fractional Max-Pooling[7] where he achieved a result of **96.53%** accuracy on the test dataset. The model that he uses in his work uses a much more complex model than what we have done in this study. His model also needs days of

training on very high-end GPUs. For us, in the absence of such high-end hardware, we make a simpler model and compare to see how much accuracy can be achieved with a reasonable amount of training in a standard hardware.

III. Methodology

(approx. 3-5 pages)

1. Data Preprocessing

Several works in the past has shown that edge pixels contribute very less to the actual features of an image [8]. For this reason, we remove 4 pixels from each of the side of the image. Also, we normalize the images by making their mean 0 and standard deviation 1.

When learning a statistical model of images, it might be nice to force the model to focus on higher-order correlations rather than get distracted by modelling two-way correlation. One way to force the model to ignore second-order structure is to remove it. This can be done by Whitening filter [9]. This is another pre-processing that we do on the input images. So, the processing of the images consists of these two steps:

1. **Image Cropping:** The input images are cropped to a size of 24 x 24 pixels, centrally for evaluation.
2. **Image Whitening:** All the input images are standardized to make their mean 0 and standard deviation 1.

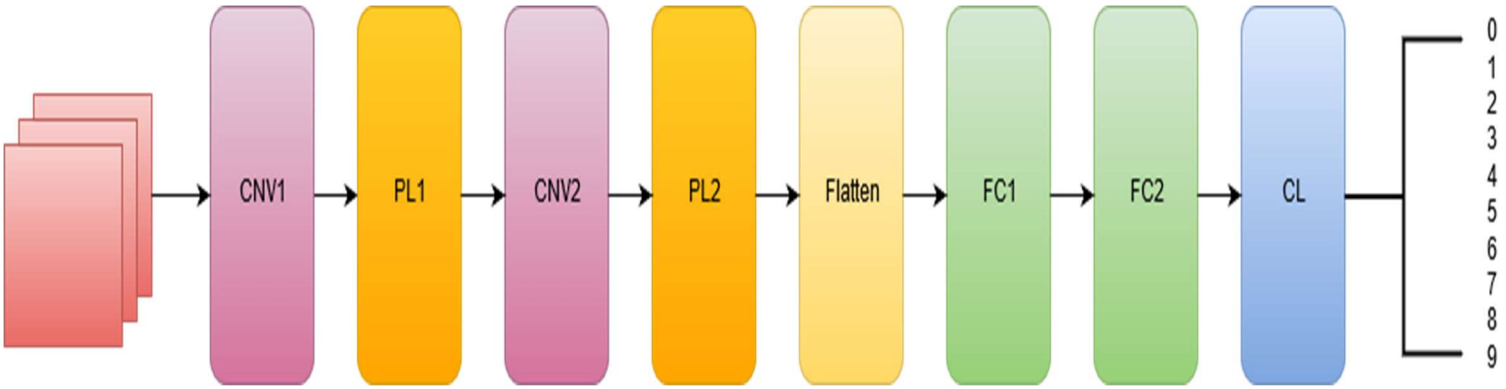
To increase the training set, more images are generated from the given images by applying a series of random distortions. These are fixed series of distortions that are as follows:

- Randomly flip the image from left to right.
- Randomly distort the image brightness.
- Randomly distort the image contrast.

Some of the examples of these distortions output are as follows:

3. Implementation

The modified AlexNet architecture model is used for this study which is shown as below:



After the pre-processing, the images are passed through different layers with following parameters:

1. **CNV1:** A Convolution Layer with 64 filters of kernel size [5x5]
    a. **Input:** [batch_size, 24, 24, 3]
    b. **Output:** [batch_size, 24, 24, 64]
2. **PL1:** A MaxPooling layer of kernel size [2x2] with a stride of 2.
    a. **Input:** [batch_size, 24, 24, 64]
    b. **Output:** [batch_size, 12, 12, 64]
3. **CNV2**: Another convolution layer with 64 filters of kernel size [5x5].
    a. **Input:** [batch_size, 12, 12, 64]
    b. **Output:** [batch_size, 12, 12, 64]
4. **PL2**: Another MaxPooling layer of kernel size [2x2] with a stride of 2.
    a. **Input:** [batch_size, 12, 12, 64]
    b. **Output:** [batch_size, 6, 6, 64]
5. **Flatten**: The data is reshaped into 1-D to prepare it for the Fully Connected Layer. So, the size of this dimension is 6 x 6 x 64 = 2304.
6. **FC1**: A fully connected layer is followed with 256 neurons.
    a. **Input:** [batch_size, 2304]
    b. **Output:** [batch_size, 256]
7. **FC2**: A fully connected layer is followed with 128 neurons.
    a. **Input:** [batch_size, 256]
    b. **Output:** [batch_size, 128]
8. **CL**: At last a classifier layer consisting of Softmax function is used to convert all the input vector into probabilities of 10 classes.
    a. **Input:** [batch_size, 128]
    b. **Output:** [batch_size, 10]

3. Refinement

The initial training was done purely on the provided input training set. The initial result was ~70% with 60,000 iterations. This result was improving with every increase in the training iteration. To help our model to extract the features faster, we introduced these randomly modified images that increased the input training sample data for each of the classes and helped us to increase the accuracy to ~74%. Along with introducing more training data, we also increased the number of iterations to 85,000. Another, important refinement was to have checkpoints in our training runs. This is done to save the learned parameters after every 1000 iterations and then later when the training is started again, the saved parameters are picked up rather than starting from the beginning. These greatly helped us to run the intensive training on my laptop, whenever it was not used. This technique helped us to do training in batches rather in one go.

IV. Results

(approx. 2-3 pages)

1. Model Evaluation and Validation

For the model evaluation, we use all the 10,000 images in the evaluation set of CIFAR-10. It calculates the accuracy described in the Evaluation Metrics section.

System Information:

- Intel Core i5 – 7300 @ 2.6 GHz.
- Memory: 16 GB RAM

Total 85,000 optimization iterations are used for this work. After 85,000 optimization iterations, the classification accuracy is about 79% on the test-set. There are enough mis-classifications as well. Examples of these mis-classifications are plotted below. Some of these are difficult to recognize even for humans and others are reasonable mistakes e.g. between a cat and a dog.

Our model that gives an accuracy of ~79% is much lower in comparison to the benchmark model but when the training time and hardware at disposal is considered, it is a respectable result. The benchmark result is a highly complex model with many days of training while our model is relatively simpler and we could achieve the result of 79% accuracy after ~9 hours of training.

V. Conclusion

(approx. 1-2 pages)

In this study, we achieved a decent classification accuracy with a relatively simpler CNN architecture and in very quick time with the use of TensorFlow. This lays a strong foundation for various problems in the domain of Computer Vision. Also, even though our result in this study is promising, there are many latest techniques that can be used to improve the results even further.

One way in which the above model can be improved in terms of speed is by using depth-wise separable convolutions to build light weight deep neural networks [10]. A new technique recently presented by Andrew G. Howard et. al. A standard convolution both filters and combines inputs into a new set of outputs in one step. The depth-wise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size. There are several other models that shows better results but most of them needs heavy training time and usage of high end GPUs. In the absence of such high-end hardware at my disposal, improving our model to use such depth-wise convolution seems to be next logical step to greatly improve the speed

that can be used on mid-range hardware and yet give improved results in about same training time.

VI. References

1. Torralba, A., Fergus, R., Freeman, W. T., 2008, 80 million tiny images: a large dataset for nonparametric object and scene recognition.
2. WordNet can be found at http://wordnet.princeton.edu/
3. Miller, G. A., 1995, WordNet: a lexical database for English.
4. LeCun, Yann. *LeNet-5, convolution neural networks*.
5. Matusugu, Masakazu; Katshhiko Mori; Yusuke Mitari; Yuji Kaneda (2003). *Subject independent facial expression recognition with robust face detection using a convolution neural network*. Neural Networks. 16 (5): 555-559.
6. Vinod, Nair; Geoffrey E. Hinton. *Rectified linear units improve restricted Boltzmann machines*. ICML' 10 Proceedings of the 27th International Conference on International Conference on Machine Learning, June 21- 24, 2010, USA
7. Graham, Benjamin. *Fractional Max-Pooling.* In *arxiv:cs/arXiv:1412.6071,* 2015.
8. Krizhevsky, Alex. *Convolution Deep Belief Networks on CIFAR-10.* Unpublished manuscript, 2010.
9. Krizhevsky, Alex. *Learning multiple layers of features from tiny images.* Computer Science Department, University of Toronto, Tech. Report, 2009.
10. Howard, Andrew G; Zhu, Menglong; Chen, Bo; Kalenicenko, Dmitry; Wang, Weijun; Weyand, Tobias; Andreetto, Marco; Adam, Hartwig. *MobileNets: Efficient Convolution Neural Networks for Mobile Vision Applications.* arXiv:1704.04861v1, 17 Apr 2017.