Image Classification Using Convolutional Neural Network

Kiran Nanjundaswamy
Northeastern University, Boston, MA, 02115
nanjundaswamy.k@husky.neu.edu

Abstract

Image classification is the task of categorizing objects based on their shared qualities or characteristics. The objects are perceived as inputs which are processed into classes or a probability of classes to which they belong. This skill is obtained naturally by humans from the time we are born. We can effortlessly classify objects in our everyday environment. For a machine to perform the same task would require more effort. Computer vision algorithms tend to fail when factors such as viewpoint variation, scale variation, illumination conditions, background clutter etc. are encountered. A good image classification model must be invariant to the cross product of all these variations. Designing a primitive algorithm to pay attention to every aspect and detail of an image is a monumental task. However, convolutional neural networks have simplified this task by looking for low level features such as curves and edges in the lower convolutional layers and building up to more abstract concepts in consequent layers yielding in a resulting class for the input image. A convolution neural network designed using TensorFlow libraries was given an input of 50,000 training images from the CIFAR-10 dataset. The network was trained for 250,000 iterations and resulted in a 65.3% accuracy in classification.

Introduction

Convolutional Neural Network (CNN) is a concept emerging from biology, specifically the human eye. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. The brain consists of neurons which get triggered when the visual cortex is exposed to an image. Scientists observed that the neurons don't get fired at the same time. Some are triggered by vertical edges and some by horizontal or diagonal edges. These neurons were observed to be in columnar structure which when fused together would produce a visual perception. CNN works in a similar fashion. It is comprised of multiple convolution layers and then followed by one or more fully connected layers.

History of Convolution Neural Networks

In 2012, Alex Krizhevsky used CNN's to win that year's ImageNet competition dropping the classification error record from 26% to 15%. Big companies have been using deep learning for classification of images since then. Google uses it for its photo search algorithm, Facebook for the automatic tagging algorithms, Amazon for product recommendations, Instagram for their search infrastructure and many more companies are working with neural nets to provide better service for their customers. Convolutional neural network is also being used in the medical field to retrieve medical images.

Why use Convolution Neural Networks?

Regular artificial neural networks don't scale well for full images. These neural networks receive an input, transform it through a series of hidden layers and yield an output class. The hidden layers are made up of neurons which are fully connected to all the neurons in the previous layer, and the neurons in each layer are independent of the other neurons in the same layer and do not share and connections. When classifying an RGB image such as CIFAR-10 images, the first hidden layer would consist of 3072 weights when the image is of size 32x32x3 (32 high, 32 wide, 3 color channels). When we scale this up to a 250x250x3 image, it would result in 187,500 weights. This is not manageable as we would require several such layers and we would require excessive computations to classify these images.

Tensorflow library

To help in designing of these neural networks, many libraries have been developed to make the task simpler. Some of the libraries are Theano, Caffe, Chainer, DeepNet, DeepPy, ConvNet, Torch, MxNet, Lasagne. Google's TensorFlow library is one of the popular and open source library for numerical computation using dataflow graphs. A primary benefit of TensorFlow is distributed computing, particularly among multiple GPU's. Nodes in the graph represents

mathematical operations while the graph edges represent the multidimensional data arrays communicated between them. Google has also designed processing units called Tensor Processing Units (TPU's) which are aligned to work perfectly with TensorFlow library.

TensorBoard

The computations for tensorflow can be complex and confusing. It is difficult to debug since we can only see the tensor of the data flowing from one part of the program to another. To make it simpler to understand and debug and optimize TensorFlow programs, Google provides a suite of web application for inspecting and understanding graphs and runs. We can use TensorBoard to visualize our network, plot quantitative metrics and draw scalar and histogram chars of the data being processed. To use TensorBoard we need to provide name scope for different computational areas of our code and add the variable for which we require either a scalar graph or a histogram chart of data points. The data will be stored in a specified log directory and can be commissioned using tensorboard command in the command line. TensorBoard can be viewed in at http://localhost:6006. We will be using TensorBoard to display vital information about the neural network in the following sections

CIFAR-10 Data Set

The CIFAR-10 dataset consists of 60000 32x32 RGB color images in 10 classes. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images

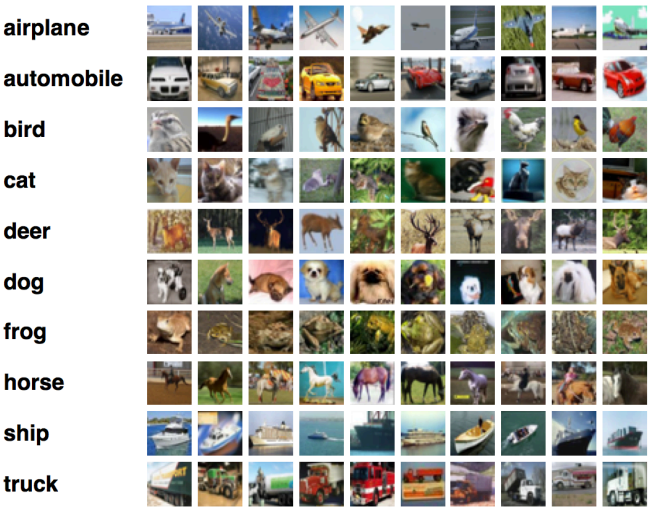

Figure1: CIFAR-10 data set with 10 classes of images. It consists of 50,000 training images and 10,000 test images.

from each class. The 10 categories of the CIRFAR-10 dataset are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The images in the categories are mutually exclusive. For instance, the class "automobiles include sedans and suv's and vehicles of that sort. The class "trucks" includes only big trucks. So, there is no overlap between automobiles and trucks. The images were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton and have been used to help in understanding image classification problems. For python, the data in each file is in the form of dictionaries. The key 'data' consists of arrays of length 3072 (32x32x3). Each array corresponds to a single image in the dataset. The key 'label' consist of an array of labels which map each of the images in the 'data' array to a specific class. The dictionary also consists of a file name for the corresponding images and a batch label indicating the number of the file in the batch.

Background

Artificial Neural Networks

Artificial Neural Networks was a concept developed in 1943. It was modeled after the structure of the human brain. The idea saw a lot of research in the early days but declined due to the technical limitations of the systems which existed before 1990's. The computational power of the system was not very high and it would take a very long time to train a simple neural network. A breakthrough in training time was achieved when the backpropagation algorithm was discovered. Here the data would be input to the network and the error rate would be calculated at the output layer. Then the weights would be changed to minimize the error and provide better outputs. Attempting to minimize the error led to neural nets which were more efficient and quicker to train. With the advancement of technology and development of many computationally powerful systems, neural nets saw a rise again. Neural nets are mostly used for deep learning which is an area of machine learning in modelling relationships which are non-linear and complicated.

A simple neural network is designed with a few input nodes through which data is inputted to the system. The data is propagated to a set of hidden layers which is a layer consisting of multiple nodes. A network can have one or many hidden layers. The nodes in each layer are connected to nodes in the incoming and outgoing layers by different, adjustable weights. The data is passed through the hidden layers and eventually reaches the output layer where the data can be interpreted as different results. The network shown below consists of 3 input layer and only 1 hidden layer consisting of 4 nodes and 2 nodes in the output layer. The nodes in each layer are connected to the nodes in the

other layer by weights. A bias is also added to each node in the network.

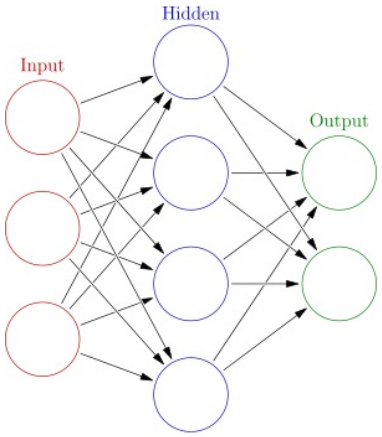

Figure 2: Simple artificial neural network with 3 input nodes, 4 hidden nodes and 2 output nodes.

Backpropagation

Backpropagation is a method of training artificial neural networks. The algorithm repeats a two-phase cycle, propagation and weight update. Before training the weights are set randomly.

• In the first phase, the input is propagated forward through the network layer by layer until it reaches the output layer. The output of each neuron in the network is given as the weighted sum of all its inputs.

$$y = x_1 w_1 + x_2 w_2,$$

• The network output is compared to the desired output and the loss function and error values are calculated. A common way to measure error is given as

$$E = (t - y)^2 ,$$

• The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output. the loss function over n training examples can be written

$$E = \frac{1}{2n} \sum_x \|(y(x) - y'(x))\|^2$$

• These error values are used to calculate the gradient of the loss function with respect to the weights in the network. The gradient calculated as a partial derivative of the output

$$\frac{\partial E}{\partial y'} = (y' - y)$$

• In the second phase, this gradient is fed to the optimization method, which in turn uses it to update the weights, to minimize the loss function.

On completion of training using backpropagation, the neurons in the intermediate layers organize themselves in a way that different neurons recognize different characteristics of the input space.

Supervised and Unsupervised Learning

Machine learning algorithms fall into two categories, supervised or unsupervised learning. Problems such as classification and regression use supervised learning and problems related to clustering and association use unsupervised learning.

Supervised learning is a concept where you have an input variable X and an output variable Y and we use an algorithm to learn the mapping function between X and Y such that Y = f (X). The idea is to reduce the error in the mapping function such that when the algorithm is fed a previously unseen data, it can predict the correct output variable Y for the data. In supervised learning, we have a set of training data and labels associated with the data which categorize them correctly. The algorithm learns to categorize the input by making predictions and understanding the error based on the associated labels. The learning stops when the algorithm achieves an acceptable level of performance. Formally it is described as, given a set of example pair $(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{x} \in \boldsymbol{X}, \boldsymbol{y} \in \boldsymbol{Y}$ the aim is to find a function $\boldsymbol{f}: \boldsymbol{X} \rightarrow \boldsymbol{Y}$.

In unsupervised learning, we are only given the input X and no corresponding output variables. The goal is to model the underlying structure or distribution of the data to learn more about the data given. Here there is no correct answer and we do not know the error in the result achieved. The algorithms are left to discover and present the interesting structure of data. Formally it is described, given a model $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{a}$, where a is a constant and the cost $\boldsymbol{C} = \boldsymbol{E}\,[(\boldsymbol{x} - \boldsymbol{f}(\boldsymbol{x}))(\boldsymbol{x} - \boldsymbol{f}(\boldsymbol{x}))]$. Minimizing this cost will give us a value of that is equal to the mean of the data.

Related Work

Enormous amount of work has been done with respect to classification of images especially involving neural networks.

The paper by Jianxin Wu et al, describes the use of histograms used in the aspect of image processing and computer vision, from visual descriptors to image representations. Histogram intersection kernel (HIK) and support vector machine (SVM) classifiers are shown to be very effective in dealing with histograms.

Similarly, proposing another method for image classification, the paper written by KUN-CHE LU AND DON-

LIN YANG, describes a general framework based on the decision tree for mining and processing image data. The decision tree induction is adopted to realize relationships between attributes and the target label from image pixels, and to construct a model for pixel-wised image processing according to a given training image dataset.

The paper by Richard Socher, Brody Huval, Bharath Bhat, Christopher D. Manning, Andrew Y. Ng, emphasizes on how recent advances in 3D sensing technologies make it possible to improve object recognition. The paper introduces a model based on a combination of convolutional and recursive neural networks (CNN and RNN) for learning features and classifying RGB-D images.

Learning New Facts from Knowledge Bases with Neural Tensor Networks and Semantic Word Vectors, this is a paper from Andrew Ng's Stanford research group, which focuses on using neural networks to try to extract data and insights from unannotated text. It focuses on lexical databases like WordNet and Yago.

Reducing the Dimensionality of Data with Neural Networks, this is a publication by Geoff Hinton deals with the problem of using neural networks to reduce the dimensionality of data. The problem of dimensionality reduction has been traditionally tackled using methods like principal components analysis which looks for the greatest variance in the data set.

As a survey, the paper by Chaitali Dhaware, Mrs. K. H. Wanjale discuss the different approaches used while classifying images namely, Decision Tree, Artificial Neural Network (ANN) and Support Vector Machine (SVM).

There are several methods to classify images with their pros and cons. A classifier is considered more efficient if they can predict the image accurately. Though the neural networking algorithms required more experience and datasets, these algorithms work well for complex problems. Another advantage of these algorithms is that you do not have to worry much about the feature engineering part of the problem.

Project description

Architecture of the Convolutional Neural Network

Unlike regular artificial networks, Convolutional Neural Networks take advantage of structure of the input images. The neurons in the layers are arranged in 3-dimensions, the width, height and the depth which refers to the third dimension of the filter and not the number of layers in the network. Instead of the neurons being connected to all the other neurons in a fully connected network, the neurons in a certain layer will only be connected to a small region in the layer before it.

As we already know, the input image will be an array which holds the raw pixel values of the image. Here the image is of 32 height, 32 width and 3 color channels. Now the first convolution layer or filter will be of size 5x5x3. We must make sure that the depth of the first layer is same as the number of channels of the image. We choose a small filter size as we want to only concentrate on only a small region at any point of time. We choose to the have 64 filters in this layer. So, the output of the 1st layer will be 64 channels. We can have more than one convolution layers. For our purpose, we will be using two convolution layers. The second convolution layer will have a size of 5x5x64. The size is the same but the number of channels has changed to 64 to accommodate the output from the 1st filter. These convolution layers will go through a series of ReLU and Pooling operation which will be discussed later. The result from the second convolutional layer will be made into a flat layer and fed into a fully connected network with 128 neurons. This layer is followed by another fully connected network of 128 neurons again. The output of the fully connected layer will be just 10 classes. These 10 classes correspond to the classes into which the image belongs.

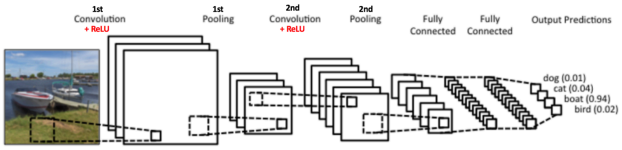

Figure 3: Pictorial representation of the architecture of convolution neural network used in this project.

Designing the neural network

The input data set

The data we require for testing and training are in the data_batch_1 to data_batch_5 files and test_batch file respectively. The data is read from both the batches and stored in training and testing variables to be used in the code. We only retain the image array, the class label for the image and a one hot encoding of the label to be used in our training and testing evaluations. We use cPickle library to read the files and use tensorflow's one hot encoding to encode the labels.

Preprocessing the images

The images will be processed before sending it to the network. We can perform image distortion using tensorflow's image library. We crop the images, we do a random flip and change the hue, brightness, contrast and saturation characteristics of the images. This is done to provide us with a broader perspective of each image and will delete unnecessary image discrepancy's which may hinder the

training process. The resulting size of the image after processing is 24x24x3.

Convolution layers

We can now build the network which will be trained to classify the images. The network was built with 2 convolution layers followed by 2 fully connected layers and finally produce 10 classes corresponding to the classes of the image. The first convolution layer is of size 5x5 with 3 channels. The layer also called as the filter will only pay attention to a small area of the image at any point in time. The filter is an array of weights which are randomly initialized to have a normalized collection of data with a specific standard deviation. Consider the convolution filter to be a sliding window on top of the image. The filter is slid on the top of the image across the entire width and height and the part of the image being viewed from the filter is called the receptive field. We compute the dot products between the entries of the filter and the input at any position. We sum up all the dot products and we have one single number which is a representative of when the filter is on top of that region of the image. Now the filter is moved by one unit or one pixels to the right and process is repeated. This is defined by the stride of the convolution filter. We also adding a padding to the image which is set to 'SAME', meaning that the input image is padded with 0's so the size of the output is the same. After the entire image is covered we end up with a 2-dimensional activation or feature map which is of size 28x28. This number is as so because there are 784 different locations that a 5x5 filter can fit on a 32x32 input image. These in turn forms a 28x28 array of numbers. Next, we add a bias to the filter. The bias is a constant value carried across the matrix.

The formula for calculating the output size of any convolution layer is given by

$$O = \frac{(W - K + 2P)}{S} + 1$$

where O is the output length, W is the input size, K is the filter size, P is the padding and S is the stride.

ReLU and Pooling

The output of the convolution layer is put across two more operations called RELU and Pooling. In Rectified Linear Unit (ReLU) layer, an element wise activation function is applied to calculates the max for each input pixel. Performing this operation adds some non-linearity to the formula and allows us to learn more complicated functions. This operation does not change the size of the image. The ReLU layer consists of neurons which apply the above-mentioned activation function. It is given by

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

Now we perform the pooling using the MaxPooling layer. We use pooling to down-sample the image resolution. We use 2x2 window which selects the largest value in each window position and then moves it across by two pixels to the next window position.

The output of the first convolution layer is sent to another convolution layer for further processing. This layer will also be of size 5x5 but now with 64 input channels as the number of output from the previous convolution layer was 64 channels. This convolution layer performs the same operation as the previous layer but now it performs it not on the image but on the results of the first convolution layer. We perform max pooling and ReLU operation on the outputs of the second convolution layer as well. The resultant of these operations is a further down sampled array of data.

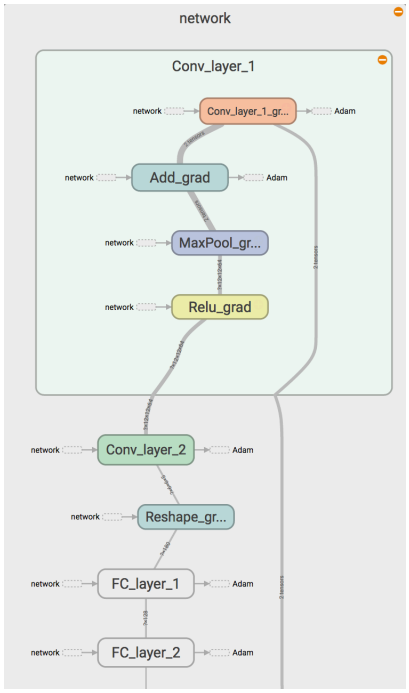

Figure 4: TensorBoard representation of the entire neural network

Flattening the layer

The resultant of the above operations will output a tensor with 4 dimensions. We need to pass this output to the fully connected layers. To do so we need to need to reduce the 4-dimension tensor to a 2-dimension which can be used as input to the fully connected layer. The second convolution layer output's 5 channels of data and a 2D filter array of size 6x6. When we flatten this later we are left with an arbitrary number of images which have been flattened to vectors of length 180 (6x6x5) each.

Fully connected layers

Next the fully connect layers will compute the class scores which results in a volume of size 1x1x10 which corresponds to the 10 classes of the images. Since this is a fully connected like the layers of a regular artificial neural network, the neurons in this layer will be connected to all the elements of the previous layer. Both the layers have the same number of neuros which is set to 128. In these layers, we initialize the weights and biases as done previously for the convolution layers. There is not much operations done in the fully connected layers. We perform a matrix multiplication of the input and the weights and add the biases to these calculations. For the first fully connected layer we have an input of vector size 180 which is the result of flattening the output from the convolution layers. We perform a ReLU operation in this layer as well. The output of the first fully connected layer is a vector of length 128.

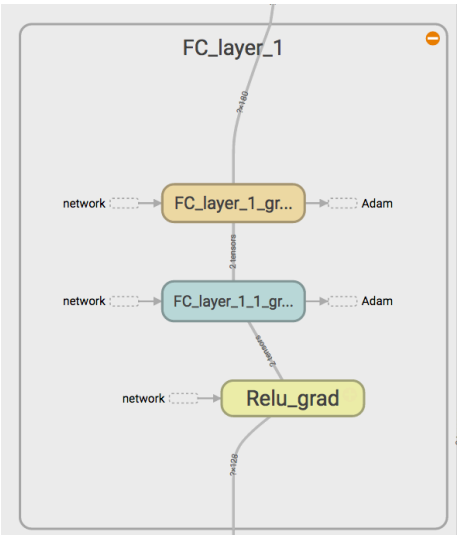

Figure 5: TensorBoard representation of the first fully connected layer with ReLU operation

This is passed to the second fully connected layer. We perform the same operations as before but we do not perform the ReLU operation for the second layer. This layer will now output a vector of size 10. The second fully-connected layer estimates how likely it is that the input image belongs to each of the 10 classes. However, these estimates are a bit rough and difficult to interpret because the numbers may be very small or large, so we want to normalize them so that each element is limited between zero and one and the 10 elements sum to one. This is calculated using the tensorflow's softmax function.

Loss calculation

The softmax function measures the probability error in discrete classification. The classes are mutually exclusive as mentioned previously. Each entry can only exist in one class. Each image in the dataset labeled with one label. An image can either be a cat or a ship, but not both. While the classes are mutually exclusive, the probabilities outputted need not be. All that is required is that each of row of labels in a valid probability distribution. If this is not the case then the computation of the gradient in the following steps will be incorrect.

The softmax is calculated but taking the exponential of all elements in the input array and then dividing that by the sum of all the exponents of elements across all dimensions (reduce_sum).

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

In formal definition, softmax function is a generalization of the logistic function that "squashes" a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range (0, 1) that add up to 1.

To make the network model perform better in classification we must change the variables, particularly the weights for all the network layers. To do this we must know how well the current network is performing by comparing the predicted output to the desired output. Cross entropy is a performance measure used in classification. It is a continuous function which is always positive. If the predicted output of the network exactly matches the desire output then the cross-entropy values is 0. The goal of optimization is to minimize the cross entropy and get it as close to zero as possible. To calculate the cross entropy, we use tensorflow's cross entropy function. The cross entropy for p and q is given by

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

Where p is the true prediction and q is the predicted or derived discrete value

By calculating the cross entropy of the model, we have determined how well the model is working on each image individually. To use the cross-entropy to guide the optimization of the model's variables we need a single scalar value, so we simply take the average of the cross-entropy for all the image classifications. This will yield us the cost of the loss of the entire model which we will use to calculate our optimizer.

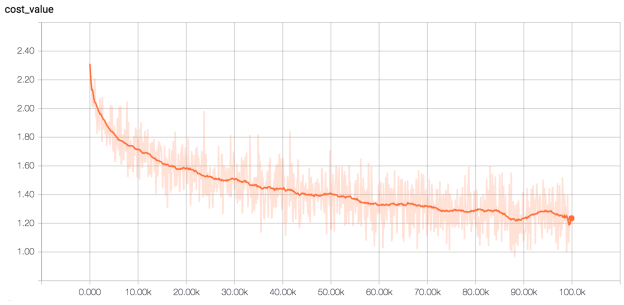

Figure 6: TensorBoard representation of the loss or cost of the model shown decreasing over time as the model gets better at predicting the class of the image.

AdamOptimizer

In this project, we are using the Adam optimizer to reduce the loss of the model. Adam optimizer is an advanced for of gradient descent. It provides several advantages over gradient foremost being that it uses a moving average of parameters. s enables Adam to use a larger effective step size, and the algorithm will converge to this step size without fine tuning. Tensorflow has an Adam optimizer function to help in minimizing the cost. The Adam algorithm takes 3 hyper parameters: the learning rate, the decay rate of 1st-order moment, and the decay rate of 2nd-order moment. The update rule for a variable given a gradient g is given as

```
t <- t + 1
lr_t <- learning_rate * sqrt(1 - beta2^t) / (1 - beta1^t)

m_t <- beta1 * m_{t-1} + (1 - beta1) * g
v_t <- beta2 * v_{t-1} + (1 - beta2) * g * g
variable <- variable - lr_t * m_t / (sqrt(v_t) + epsilon)
```

Each update of the Adam optimizer involves

- Computing the gradient and its element-wise square using the current parameters.
- Updating the exponential moving average of the 1st-order moment and the 2nd-order moment.
- Computing an unbiased average of the 1st-order moment and 2nd-order moment.
- Computing weight update: 1st-order moment unbiased average divided by the square root of 2nd-order moment unbiased average (and scale by learning rate).
- Apply update to the weights.

The main down side of the algorithm is that Adam requires more computation to be performed for each parameter in each training step and more state to be retained for each parameter.

The learning rate of the network is defined here. We are using a learning rate 1e-4. After the weight updates are completed the network will continue to train for other values and until the number of training optimization epochs mentioned by the user.

Accuracy

To test the performance measure of the network from time to time and observe how well the network is training we can check for the accuracy and record it. We do this by find the mean across the vector of Booleans of the predicted classes and the true classes.

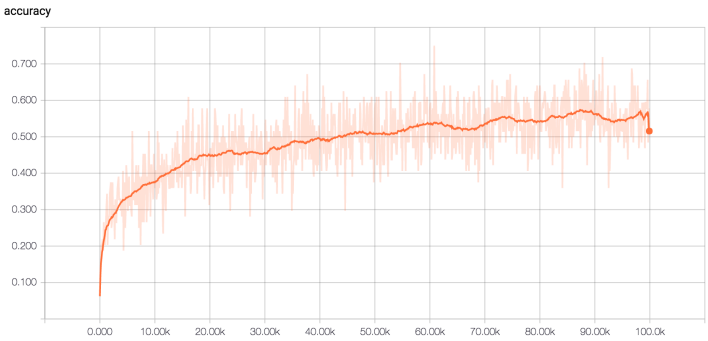

Figure 7: TensorBoard representation of the accuracy of the network shown increasing over time as the model gets better at predicting the class of the image.

Since the training process takes a long time, we save some the parameters as check points. This is helpful as we can pause the training and continue from the last saved checkpoint.

Experiments

The network was trained initially for 100,000 epochs and the changes in the loss and accuracy were observed over time.

```
Optimization Iteration:  63101, Training Accuracy:  62.5%
Optimization Iteration:  63201, Training Accuracy:  60.9%
Optimization Iteration:  63301, Training Accuracy:  60.9%
Optimization Iteration:  63401, Training Accuracy:  65.6%
Optimization Iteration:  63501, Training Accuracy:  75.0%
Optimization Iteration:  63601, Training Accuracy:  64.1%
Optimization Iteration:  63701, Training Accuracy:  62.5%
Optimization Iteration:  63801, Training Accuracy:  59.4%
Optimization Iteration:  63901, Training Accuracy:  64.1%
Optimization Iteration:  64001, Training Accuracy:  76.6%
Saved checkpoint.
Optimization Iteration:  64101, Training Accuracy:  62.5%
Optimization Iteration:  64201, Training Accuracy:  65.6%
Optimization Iteration:  64301, Training Accuracy:  62.5%
Optimization Iteration:  64401, Training Accuracy:  68.8%
Optimization Iteration:  64501, Training Accuracy:  70.3%
Optimization Iteration:  64601, Training Accuracy:  67.2%
Optimization Iteration:  64701, Training Accuracy:  68.8%
```

Figure 8: The result for every 100 epochs is displayed on the console. We can see the number of epochs completed and the training accuracy currently reached.

Below is a tabular representation of the number of training epochs completed and the range of accuracy achieved during training

Number of epochs	Training accuracy
10,000	15% – 28 %
50,000	35% - 47%
100,000	56% - 75%

Below is a tabular representation of the number of training epochs completed and the accuracy achieved during testing of the network

Number of epochs	Testing accuracy
100,000	52%
200,000	56%
300,000	60.7%
400,000	64.7%
500,000	65.3%

As it is observed in the above table, the best test accuracy which was achieved after 500,00 epochs of training is 65.3%. There is not much difference between the last 2 rows of the table. This is because the network has become saturated and cannot achieve a higher accuracy beyond a certain point. To achieve better results, we need to change the hyper parameters of the network and include more layers and better distribution of the sampled input for training. The process of choosing the correct parameters of the network is a since of its own. We will stop here and observe the results returned by the trained network.

Some of the example errors during prediction by the network are displayed below.

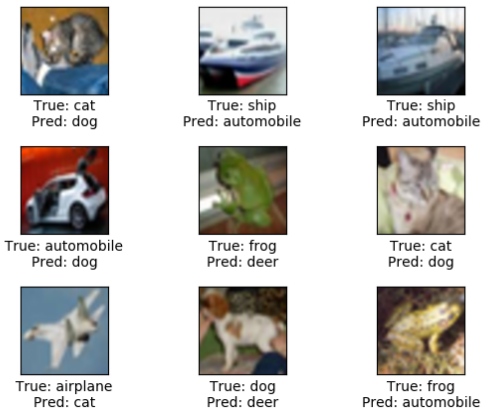

Figure 8: wrong classification of images by the network

Below are some of the histograms of the weights of the network layers which were retrieved using TensorBoard after 100,00 epochs of training

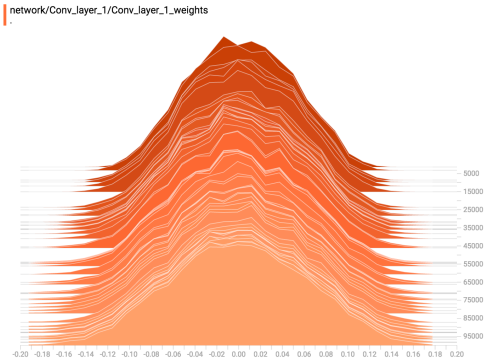

Figure 9: Weight vector distribution of 1st convolutional layer

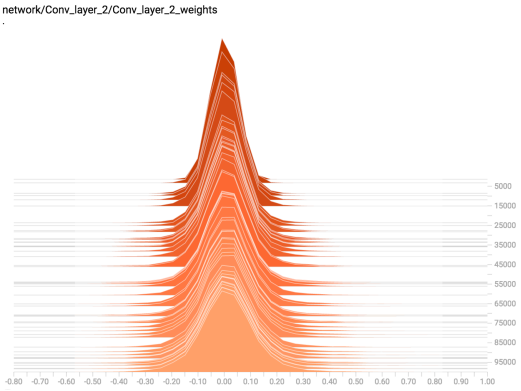

Figure 10: Weight vector distribution of 2nd convolutional layer

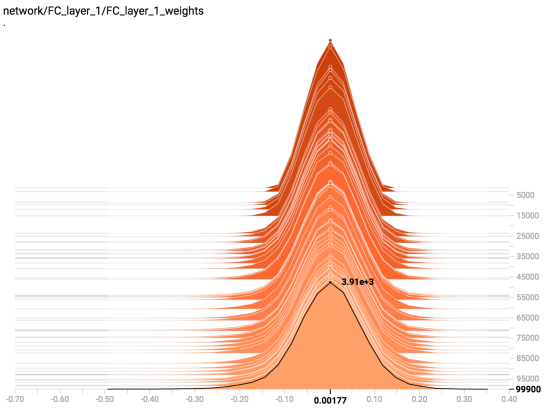

Figure 11: Weight vector distribution of 1st Fully connected layer

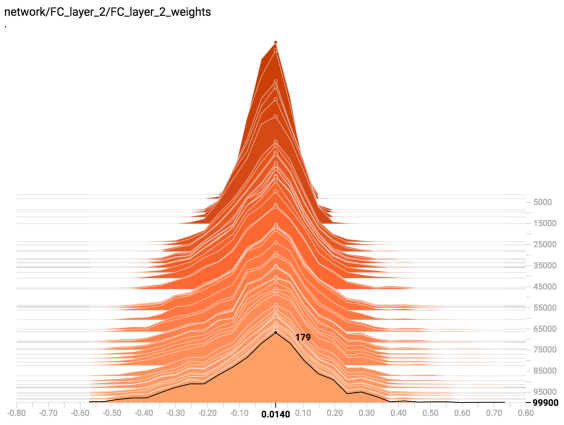

Figure 11: Weight vector distribution of 2nd Fully connected layer

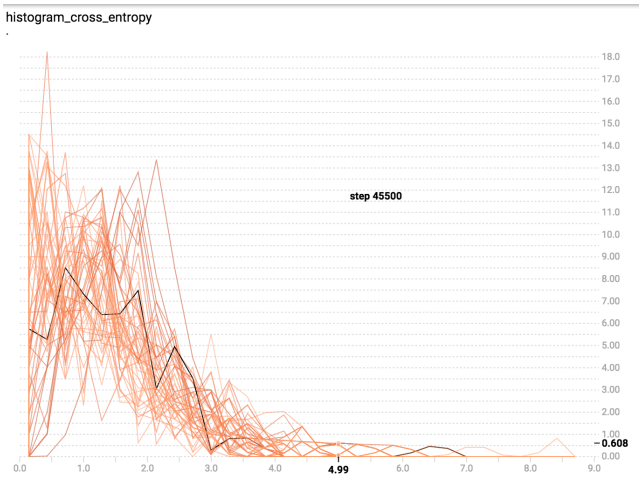

Figure 12: Cross entropy histogram showing the decrease in the distribution of loss as the network gets trained

Conclusion

CNNs give the best performance in pattern/image recognition problems and even outperform humans in certain cases. The Convolution neural network we designed could provide a 65.3% accuracy in classification of CIFAR-10 dataset. The training took 9 and half hours to reach this accuracy level on a MacBook pro 2015 with 2.7 GHz Intel Core i5 processor. It was convenient to use TensorFlow library to design the network and its parameters. TensorBoard provided beautiful insights into the working of the network and was very useful for debugging. It can be said that Convolutional Neural Networks are optimal for image classification. Future work would involve designing a better network with more layers of convolution and fully connected layers to provide better results of classification. The network now is only capable of classifying 32x32x3 images. This can be extended to larger images by adding more convolution, ReLU and pooling layers. This network could classify the images into 10 distinct classes. We can also build networks to classify data into more number of classes. If possible we should make use of GPU processors to help in faster training of the network. Also google has introduced Tensor Processing Units called TPU's which outperform GPU's in processing speed. We can also make use of new wrapper libraries such as learn, Keras and others to help in writing shorter code and performing more analysis of the optimal hyper parameters of the network and number of layers required to yield the best results.

References

1. WildML. (2015). Understanding Convolutional Neural Networks for NLP. [online] Available at: http://bit.ly/2oQjXwT [Accessed 21 Apr. 2017].
2. TensorFlow. (2017). TensorFlow. [online] Available at: https://www.tensorflow.org/ [Accessed 21 Apr. 2017].
3. Schumacher, A. (2016). Hello, TensorFlow!. [online] O'Reilly Media. Available at: http://oreil.ly/2oQ0vAe [Accessed 21 Apr. 2017].
4. Medium. (2016). Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks. [online] Available at: http://bit.ly/2oPtO63 [Accessed 21 Apr. 2017].
5. Deshpande, A. (2016). A Beginner's Guide To Understanding Convolutional Neural Networks. [online] Adeshpande3.github.io. Available at: http://bit.ly/2oQhMtc [Accessed 21 Apr. 2017].
6. Cs231n.github.io. (2017). CS231n Convolutional Neural Networks for Visual Recognition. [online] Available at: http://bit.ly/2oQ9e5r [Accessed 21 Apr. 2017].
7. Nielsen, M. (2015). Neural Networks and Deep Learning. Determination Press, [online] p. Available at: http://bit.ly/2oPXEYm [Accessed 21 Apr. 2017].
8. Brownlee, J. (2016). Crash Course in Convolutional Neural Networks for Machine Learning - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: http://bit.ly/2oQgIp9 [Accessed 21 Apr. 2017].
9. the data science blog. (2016). An Intuitive Explanation of Convolutional Neural Networks. [online] Available at: http://bit.ly/2oQkhf5 [Accessed 21 Apr. 2017].
10. Clarifai.com. (2017). What is visual recognition? | Clarifai. [online] Available at: http://bit.ly/2oQ7wRL [Accessed 21 Apr. 2017].
11. Ufldl.stanford.edu. (2017). Unsupervised Feature Learning and Deep Learning Tutorial. [online] Available at: http://stanford.io/2oQ8mxP [Accessed 21 Apr. 2017].
12. Yim, J., Ju, J., Jung, H. and Kim, J. (2015). Image Classification Using Convolutional Neural Networks With Multi-stage Feature. Springer, Cham, [online] pp.587-594. Available at: http://bit.ly/2oQ6qp0 [Accessed 21 Apr. 2017].

13. Brownlee, J. (2016). Supervised and Unsupervised Machine Learning Algorithms - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: http://bit.ly/2oQjACn [Accessed 21 Apr. 2017].
14. Scholarworks.sjsu.edu. (2017). [online] Available at: http://bit.ly/2oQ6xBa [Accessed 21 Apr. 2017].
15. Ip.cadence.com. (2017). [online] Available at: http://bit.ly/2oQf3QA [Accessed 21 Apr. 2017].
16. Citeseerx.ist.psu.edu. (2017). [online] Available at: http://bit.ly/2oQph3f [Accessed 21 Apr. 2017].
17. Ieeexplore.ieee.org. (2017). Efficient HIK SVM Learning for Image Classification - IEEE Xplore Document. [online] Available at: http://bit.ly/2oQplA1 [Accessed 21 Apr. 2017].
18. Papers.nips.cc. (2017). [online] Available at: http://bit.ly/2oQ8BJi [Accessed 21 Apr. 2017].
19. Pdfs.semanticscholar.org. (2017). [online] Available at: http://bit.ly/2oQ6KEe [Accessed 21 Apr. 2017].
20. Ijcstjournal.org. (2017). [online] Available at: http://bit.ly/2oQhdzx [Accessed 21 Apr. 2017].
21. Chen, D., Socher, R., Manning, C. and Ng, A. (2013). Learning New Facts From Knowledge Bases With Neural Tensor Networks and Semantic Word Vectors. [online] Arxiv.org. Available at: http://bit.ly/2oQa2qY [Accessed 21 Apr. 2017].
22. Cs.toronto.edu. (2017). [online] Available at http://bit.ly/2oPYft4 [Accessed 21 Apr. 2017]
23. Raschka, S. (2017). When Does Deep Learning Work Better Than SVMs or Random Forests?. [online] Kdnuggets.com. Available at: http://bit.ly/2oQ6SUc [Accessed 21 Apr. 2017].
24. Gibiansky, A. (2017). Convolutional Neural Networks - Andrew Gibiansky. [online] Andrew.gibiansky.com. Available at: http://bit.ly/2oQande [Accessed 21 Apr. 2017].
25. Vision.caltech.edu. (2017). Caltech-UCSD Birds-200-2011. [online] Available at: http://bit.ly/2oQ7cm8 [Accessed 21 Apr. 2017].
26. GitHub. (2017). aymericdamien/TensorFlow-Examples. [online] Available at: http://bit.ly/2oPYEf4 [Accessed 21 Apr. 2017].
27. En.wikipedia.org. (2017). Convolutional neural network. [online] Available at: http://bit.ly/2oPYG6G [Accessed 21 Apr. 2017].
28.
29. En.wikipedia.org. (2017). Artificial neural network. [online] Available at: http://bit.ly/2oQ9odb [Accessed 21 Apr. 2017].
30. En.wikipedia.org. (2017). Cross entropy. [online] Available at: http://bit.ly/2oQ9ly4 [Accessed 21 Apr. 2017].
31. En.wikipedia.org. (2017). Backpropagation. [online] Available at: http://bit.ly/2oQe82s [Accessed 21 Apr. 2017].
32. En.wikipedia.org. (2017). Gradient descent. [online] Available at: http://bit.ly/2oQ23dw [Accessed 21 Apr. 2017].
33. En.wikipedia.org. (2017). Stochastic gradient descent. [online] Available at: http://bit.ly/2oQ1VLk [Accessed 21 Apr. 2017].
34. Sebastian Ruder. (2016). An overview of gradient descent optimization algorithms. [online] Available at: http://bit.ly/2oQgyyl [Accessed 21 Apr. 2017].
35. work? H. (2017). How does the Adam method of stochastic gradient descent work? [online] Stats.stackexchange.com. Available at: http://bit.ly/2oPYX9I [Accessed 21 Apr. 2017].