

CIFAR 10 Image Classification with CNN and Caffe

Machine Learning Engineer Nanodegree Capstone Project

Neo Xing, 2016/10

I. Definition

1.1 Project Overview

Image Classification

Computer vision has become the one of the most important field in artificial intelligence as we has entered The Zettabyte Era, where image is the most pervasive form of information. Cisco predicated that the global video traffic is 70 percent in 2015 of all consumer Internet traffic.

Image classification, or image recognition, is one core problem in computer vision, which is the task of assigning an input image one label from a fixed set of categories. It has a large variety of practical applications, from post number recognition, facial detection, and researches such as finding new galaxies in astronomy. Moreover, many other computer Vision tasks can be reduced to image classification, such as object detection and segmentation.

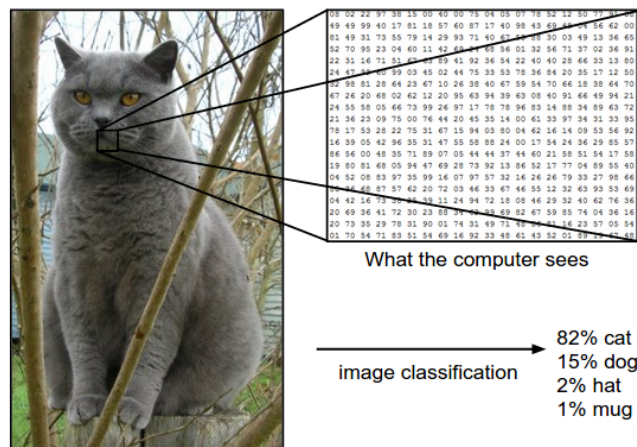


Figure 1: Image classification task

Convolutional Neural Network

Currently, the best algorithms for such tasks are based on convolutional neural networks. Performance of convolutional neural networks on the ImageNet tests, is now close to that of humans. The rapid development of CNN lies in the vast utilization of GPU and large image datasets. There are successful applications in image recognition, natural language processing, and playing computer Go, as well as popular frameworks such as Caffe and Tensorflow.

A convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. It mitigate the challenges posed by the simple Artificial Neural Network such as MLP by exploiting the strong spatially local correlation present in natural images. It features a 3D volumes of neurons of multiple layered structure, especially the convolutional layers with shared local filters and the pooling layers.

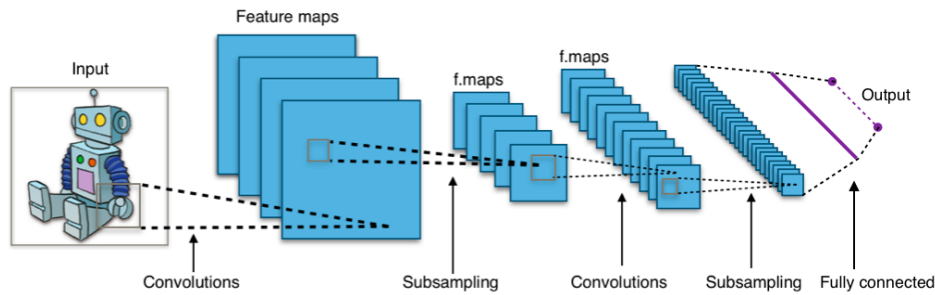


Figure 2: Typical CNN structure [2]

Overview

In this project, we study the CNN based image classification with the famous CIFAR-10 datasets [7] and Caffe framework. We show how to refine CNN with two optimized methods, the dropout and fine tuning pre-trained network, to achieve close to state-of-art accuracy. We'll start with a more strict definition of the problems and metrics shortly, followed by section 2 the analysis of datasets and CNN algorithms. In section 3, we train the Convnet CNN, focusing on optimization methods such as dropout regularization and parameter fine tuning, and basic transfer learning. In section 4 we provide results of optimized CNN in image classification.

1.2 Problem Statement

The problem of image classification is straight forward, classify the given input image with the most probable category label.

Data

We will use CIFAR 10 datasets. It contains 10 categories of small images, 50000 for training and 10000 for testing. The images are randomly shuffled in order and uniformly distributed by categories.

CIFAR10 Image Examples

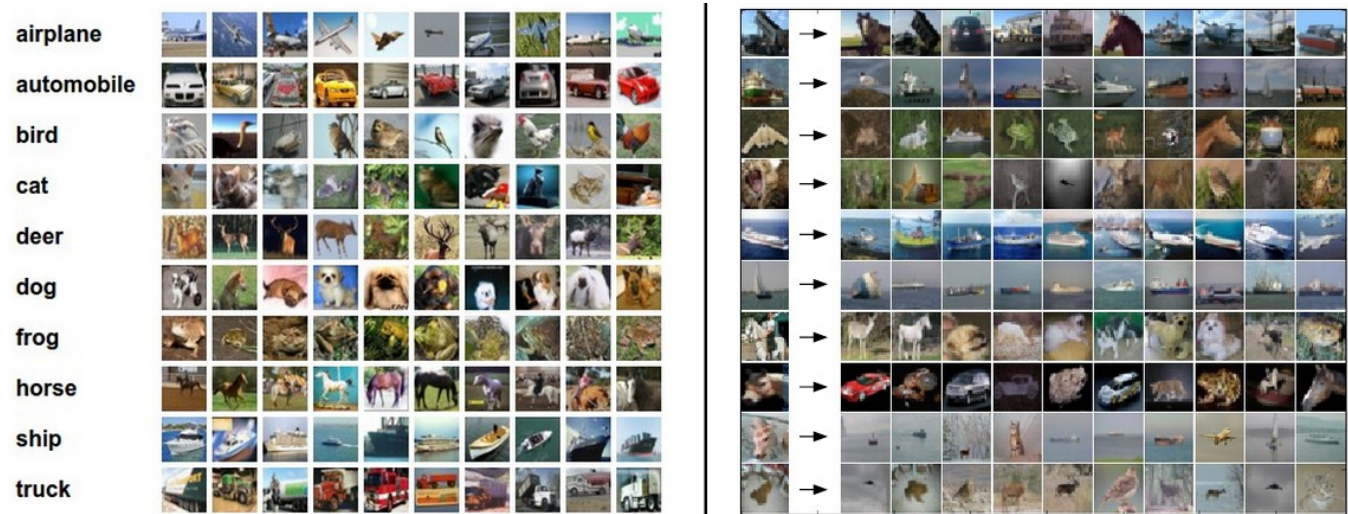


Figure 3: CIFAR10 images from 10 categories [1]

CIFAR10 Image Summary

Table 1: CIFAR10 meta data

key	meta data
label_names	[airplane, automobile, bird, cat, deer, dog, f...]
num_cases_per_batch	10000
num_vis	3072

Table 2: CIFAR10 test data

data	batch_label	data	filenames	labels
0	testing batch 1 of 1	[158, 159, 165, 166, 160, 156, 162, 159, 158, ...]	domestic_cat_s_000907.png	3
1	testing batch 1 of 1	[235, 231, 232, 232, 232, 232, 232, 232, 232, ...]	hydrofoil_s_000078.png	8
2	testing batch 1 of 1	[158, 158, 139, 132, 166, 182, 187, 193, 199, ...]	sea_boat_s_001456.png	8
3	testing batch 1 of 1	[155, 167, 176, 190, 177, 166, 168, 166, 170, ...]	jetliner_s_001705.png	0
4	testing batch 1 of 1	[65, 70, 48, 30, 23, 40, 44, 45, 45, 40, 10, 1...	green_frog_s_001658.png	6
...
9995	testing batch 1 of 1	[76, 76, 77, 76, 75, 76, 76, 76, 76, 78, 81, 8...	cabin_cruiser_s_000783.png	8
9996	testing batch 1 of 1	[81, 91, 98, 106, 108, 110, 80, 84, 88, 90, 92...	tabby_cat_s_000494.png	3
9997	testing batch 1 of 1	[20, 19, 15, 15, 14, 13, 12, 11, 10, 9, 8, 8, ...]	pekinese_s_002005.png	5
9998	testing batch 1 of 1	[25, 15, 23, 17, 23, 51, 74, 91, 114, 137, 90, ...]	convertible_s_002525.png	1
9999	testing batch 1 of 1	[73, 98, 99, 77, 59, 146, 214, 176, 125, 218, ...]	broodmare_s_000763.png	7

Metrics

Two scores are used to evaluate the algorithms, accuracy and loss function. For this multilabel classification problem, we focus on how much images can be correctly predicted. We expect the algorithm to have good performance (accuracy > 70%), thus the accuracy is chosen for its simpleness. Accuracy is defined by number of true positives divided by total number of testing samples.

$$Accuracy = \frac{N_{true\ positive}}{N_{test\ samples}}$$

The Softmax function is the generalization of binary Logistic function to multiple classes, it gives normalized class probabilities and can be interpreted as the probabilities of each label given the image and weights. Softmax function takes a vector of arbitrary real-valued scores in z and squashes it to a vector of values between zero and one that sum to one.

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Softmax loss is used in the last layer of CNN, for the convenience of its probability interpretation that category with maximum probability should be the predicted label. The full loss for the dataset is the mean of L_i over all training examples together with a regularization term.

$$L_i = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right)$$

f_j is the j -th element of the vector of class scores f , which is the inner product of vector and weights $f(x_i; W) = Wx_i$.

Benchmark

For benchmark, we collect samples on CIFAR10 images classification from Sivan Sabato [8]. The human classification is reported to be 94%, the 47th best score is 79.6%, we will evaluate our final CNN as success in this range.

Table 3: Part of CIFAR10 datasets classification best scores

Accuracy	Method	Venue
96.53%	Fractional Max-Pooling	arXiv 2015
94%	Lessons learned from manually classifying CIFAR-10	unpublished 2011
90.68%	Regularization of Neural Networks using DropConnect	ICML 2013
89%	ImageNet Classification with Deep Convolutional Neural Networks	NIPS 2012
79.6%	An Analysis of Single-Layer Networks in Unsupervised Feature Learning	AISTATS 2011

II. Analysis

In this section, we investigate the datasets, CNN algorithms and Caffe framework that we are going to use.

2.1 Data

Exploration

Each image in CIFAR 10 datasets is a $32\text{px} \times 32\text{px}$ RGB image, with one label to be classified. Such small image size implies that it's prone to overfitting, which means that small number of CNN parameters and strong regularization may be needed for better performance.

Visualization

To gain a better view of possible patterns in the datasets, we performed a PCA feature extraction, and plotted the 2-D feature map of the 1000 images in data sets. The labels from 0 to 9 corresponds to names ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'] in order. The axes are reduced 2-D features generated by PCA codes.

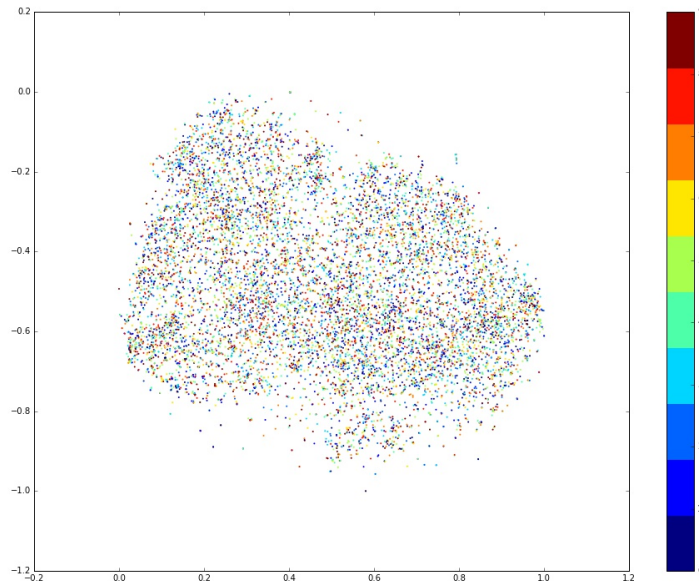


Figure 4: CIFAR10 2D feature maps of 1000 images

We can use a visualization technique called t-SNE to take the CIFAR-10 images and embed them in two dimensions so that their (local) pairwise distances are best preserved, namely the nearby images share more similar features.

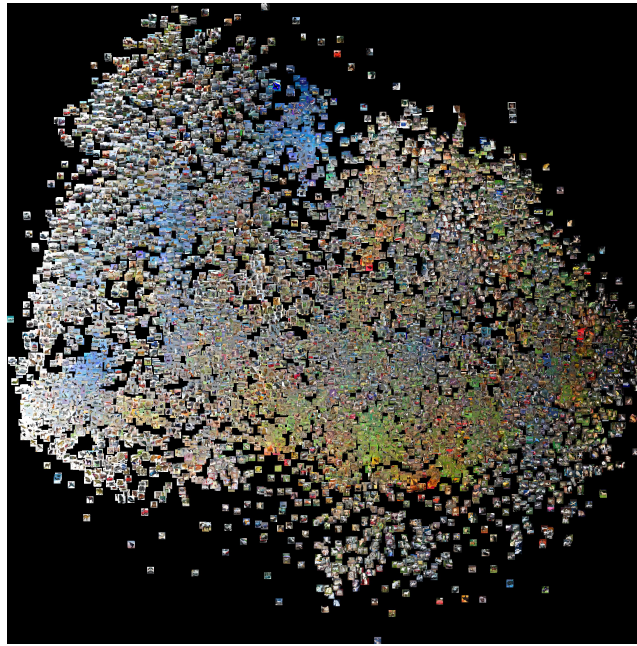


Figure 5: CIFAR10 2D feature maps of 1000 images embedded

Test run on linear classifier

As a introduce, we firstly implemented simple linear classifier using `sklearn`, and run it on testing datasets. SGFClsifier is a regularized linear models with stochastic gradient descent (SGD) learning with linear SVM loss function. After training the whole datasets, we achieve a test accuracy of 21%. It can be shown that linear classifier, although simple and fast, could not capture most of the patterns in image features, thus failed in most of the classification tasks.

2.2 Algorithms and Techniques

Multilayer Perceptron

We first introduce the Multilayer Perceptron (MLP), a basic nerval network that just consists of fully connected layers. MLP models does not take into account the spatial structure of data, and the huge number of parameters quickly leads to overfitting. For images with high-dimensional inputs, it is impractical to connect neurons to all neurons in the previous volume because such a network architecture does not take the spatial structure of the data into account.

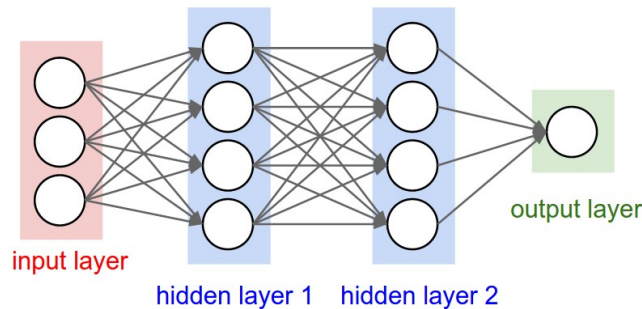


Figure 6: A Neural Network with 2 hidden layers [1]

Convolutional Neural Network

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume through a differentiable function. A few distinct types of layers are commonly used.

Convolutional layer

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters or kernels. Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

Convolutional networks exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume. Such an architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern.

Parameter sharing scheme is used in convolutional layers to control the number of free parameters. It relies on one reasonable assumption: if one patch feature is useful to compute at some spatial position, then it should also be useful to compute at a different position. Therefore, it is common to refer to the sets of weights as a filter (or a kernel), which is convolved with the input. The result of this convolution is an activation map, and the set of activation maps for each different filter are stacked together along the depth dimension to produce the output volume.

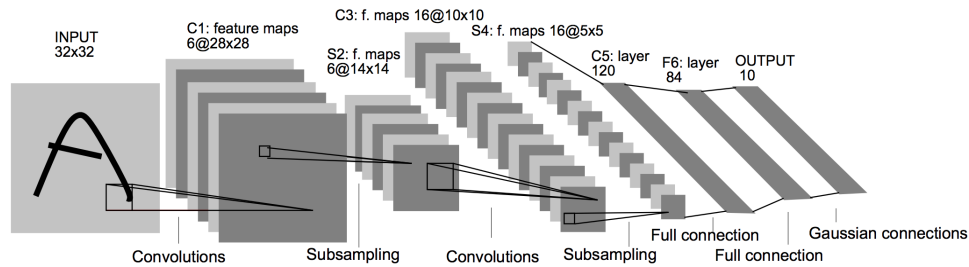


Figure 7: LeNet CNN structure [6]

Pooling layer

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 max pooling. Every MAX operation would in this case be taking a max over 4 numbers, discarding 75% of the data.

ReLU layer

ReLU is the abbreviation of Rectified Linear Units. This is a layer of neurons that applies the non-saturating activation function $f(x) = \max(0, x)$. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Fully connected layer

After several convolutional 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.

Dropout layer

Dropout is an extremely effective, simple and recently introduced regularization technique by Srivastava that complements the other methods (L1, L2, maxnorm). While training, dropout is implemented by only keeping a neuron active with some probability, or setting it to zero otherwise.

Loss layer

The loss layer specifies how the network training penalizes the deviation between the predicted and true labels and is normally the last layer in the network. In our study, softmax loss is used for predicting a single class of K mutually exclusive classes.

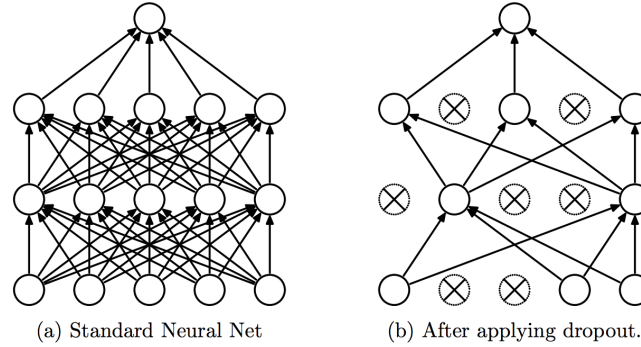


Figure 8: Dropout Neural Net Model [4]

Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind.[Caffe] It features easy defining nets, high speed GPU computation, and lots of open source community support such as shared models.

Defining layers

Caffe defines a net layer-by-layer in its own model schema. The network defines the entire model bottom-to-top from input data to loss. As data and derivatives flow through the network in the forward and backward passes Caffe stores, communicates, and manipulates the information as blobs: the blob is the standard array and unified memory interface for the framework. The layer comes next as the foundation of both model and computation. The net follows as the collection and connection of layers. To create a Caffe model we need to define the model architecture in a protocol buffer definition file (prototxt). The layers we will use are Convolution, Polling, Softmax, ReLU, Inner Product (Fully Connected) and Dropout layers.

Defining solvers

The solver orchestrates model optimization by coordinating the network's forward inference and backward gradients to form parameter updates that attempt to improve the loss. The responsibilities of learning are divided between the Solver for overseeing the optimization and generating parameter updates and the Net for yielding loss and gradients. We will use the default solver Stochastic Gradient Descent, tuning the hyperparameters learning rate α , momentum μ , and learning policy and steps.

Defining models

The models are defined in plaintext protocol buffer schema (`prototxt`) while the learned models are serialized as binary protocol buffer `.caffemodel` files following Google Protocol Buffer. It enables us to directly load and fine tune a pretrained CNN, which will greatly reduce the training cost and improve efficiency. We will use the CaffeNet model comes with Caffe Model Zoo framework of Caffe.

Environment

The study of this problem is performed on a Ubuntu 14.04 GPU instance of AWS EC2, with NVIDIA CUDA 7.5 and Caffe.

III. Methodology

In this section, we show the process of training CNN with Caffe. We'll highlight how to improve CNN performance with two optimized methods, namely the dropout and fine tuning pre-trained network.

3.1 Data Preprocessing

To improve the performance of file IO and memory access, Caffe load data from database and compute data in binary blobs. We will need to convert pickled CIFAR10 python batch to `leveldb`.

It's important to do center the images by mean subtraction, which will be handled by Caffe scripts.

For fine tuning the Caffenet, we will need to resize CIFAR10 32*32 image to 227*227 images, then convert and save them to database.

Table 4: CIFAR10 data mean subtraction



Caffe data augmentation can be performed on the fly automatically in data layer. However, this method is not used as its improvement to CIFAR 10 should be small due to its small amount of features and labels.

3.2 Implementation

With Caffe's powerful framework, we can define and train CNN by writing network and solver `.prototxt` files. Most of challenges lie in matching the layered structures and their dimensions.

Convnet

We will start with Caffe implmentation of Alex Krizhevsky's cuda-convnet.

- Network structure

`cifar10_full_train_test.prototxt`

```
-----
cifar    Data                                -> data, label
cifar    Data                                -> data, label
conv1    Convolution      (, x2.0)          data    -> conv1      5+2 32
```


pool1	Pooling		conv1	-> pool1	3/2
relu1	ReLU		pool1	-> pool1	
norm1	LRN		pool1	-> norm1	
conv2	Convolution	(, x2.0)	norm1	-> conv2	5+2 32
relu2	ReLU		conv2	-> conv2	
pool2	Pooling		conv2	-> pool2	3/2
norm2	LRN		pool2	-> norm2	
conv3	Convolution		norm2	-> conv3	5+2 64
relu3	ReLU		conv3	-> conv3	
pool3	Pooling		conv3	-> pool3	3/2
ip1	InnerProduct	(Dx250.0, x2.0 Dx0.0)	pool3	-> ip1	
accuracy	Accuracy		ip1, label	-> accuracy	
loss	SoftmaxWithLoss		ip1, label	-> loss	

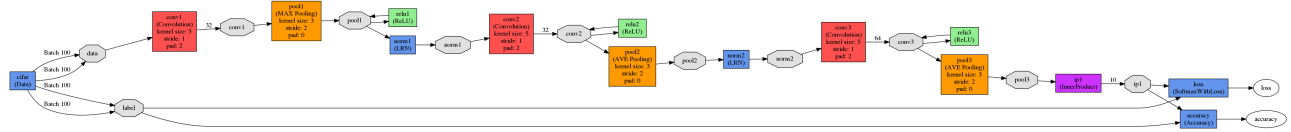


Figure 9: CIFAR10 Convnet diagram

- Network solver

```

net: "examples/cifar10/cifar10_full_train_test.prototxt"
test_iter: 100
test_interval: 1000      # Carry out testing every 1000 training iterations
base_lr: 0.001           # The base learning rate, momentum and the weight decay of the network.
momentum: 0.9
weight_decay: 0.004
lr_policy: "multistep"   # step learning policy returns base_lr*gamma^(floor(iter/step))
gamma: 0.1
stepvalue: 60000
stepvalue: 65000
display: 200             # Display every 200 iterations
max_iter: 70000          # The maximum number of iterations
snapshot: 10000          # snapshot intermediate results
snapshot_format: HDF5
snapshot_prefix: "examples/cifar10/cifar10_full"
solver_mode: GPU

```

CNN with dropout

For brevity, we will describe the difference of CNN with dropout and fine tune CaffeNet with the original ConvNet, other details can be found in their prototxt files. For CNN with dropout, we will add two more FC layers after the FC layer in Convnet, each of them is followed by a ReLU activation layer and dropout layer. To utilize the strong regularization of the dropout layers, we increase the depth (num_output) of Conv layers and FC layers to enable the net to capture more features and patterns.

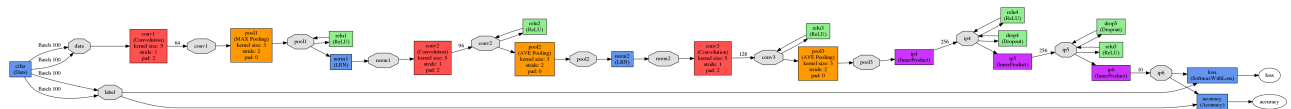


Figure 10: CIFAR10 CNN with dropout diagram

CaffeNet fine tuning

To fine tune pre-trained CaffeNet, we will modify its net structure and solver. For CNN net prototxt, the name of last FC layer is changed to avoid using CaffeNet weights, its output depth is set to the number of CIFAR labels. The database sources are modified accordingly. For solver prototxt, we load the weights of CaffeNet model, and decrease its base learning rate as the pre-trained model should already have good performance on new datasets. We also make some modification due to computation consideration, for example, using a subset of CIFAR10 (10k training images, 2k testing images), smaller data batch size and less training iteration.



Figure 11: CIFAR10 CaffeNet fine-tuning diagram

IV. Results

In this section, we study the results of three CNN models that we have trained.

4.1 Model Evaluation

Original Convnet is pretty straightforward, it works pretty well for CIFAR10 with small image size. However, the tuning of the multisteps learning rate is effective yet difficult. The learning curves are smooth, with test accuracy continuously increasing, training loss and testing loss decreasing. It means that there is not overfitting problem. Also the gap of two loss curves is not obvious, implies that the variance is small thus the algorithm is quite robust.

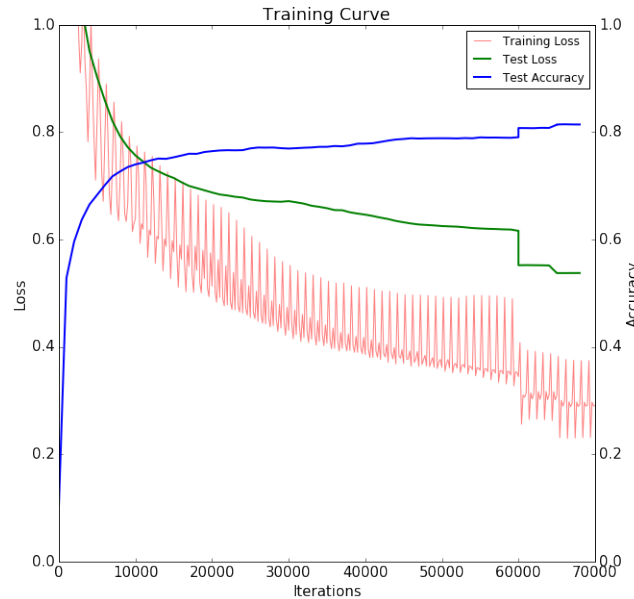


Figure 12: CIFAR10 Convnet learning curve

By adding simple dropout layers and more depth to data blobs, CNN with dropout gains ability to capture more patterns with overfitting restricted by extra regularization. Due to the probabilistic regularization brought by dropout, during training there is large variance for testing loss curve. However, the algorithm manages to overcome this issue once the learning rate decreases by factor of 10 at iteration 60,000.

Using fine-tuned CaffeNet and the pre-trained weights, we achieve good scores of accuracy at the initial and final steps of training. It shows most of the patterns in different images can be captured using the same network. Fine-tuned CaffeNet has strengths of both previous CNNs, without sign overfitting or variance. It gives higher accuracy and lower losses variance in 1/10 of the same iteration steps.

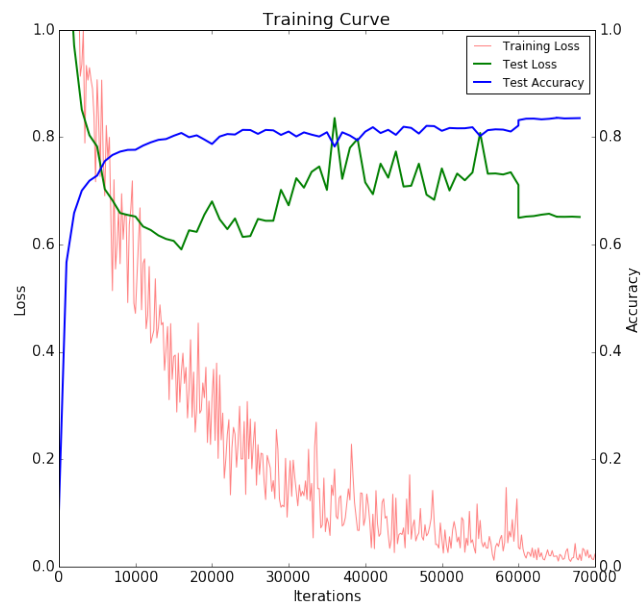


Figure 13: CIFAR10 CNN with dropout learning curve

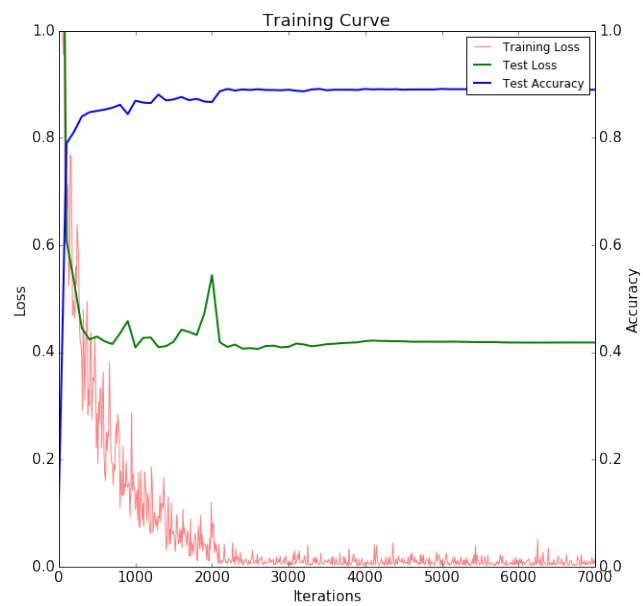


Figure 14: CIFAR10 CaffeNet fine-tuning learning curve

4.2 Justification

The three CNNs give successful results for CIFAR10 image classification, with final testing accuracy scores of 81% to 83.55% and 89% seperately. We also show how the improvement of CNN with dropout and fine-tuning pre-trained CNN. The final result fine-tuning CaffeNet has close to the state-of-art score compared to the previous benchmark, with much less effort in computation. Thus we have shown that the study clearly demonstrated how to design, train, and improve CNN for CIFAR10 image classification task.

V. Conclusion

5.1 Free-Form Visualization

We will load the fine-tuning CaffeNet and use it to detect CIFAR10 images and unseen images (see supplementary notebook). By changing the last layer of CaffeNet from loss layer to Softmax classifier, we will have the network for inference deployment. Combined by the weights saved as snapshot during training, it has good accuracy as expected. The following are some typical images and results.

1. Good prediction: for picture representing the label nicely, CNN is confident about the result with probability ~99%.
2. Positive prediction: CNN gives right prediction but the probability is in range $[0.6, 0.9]$, especially when the picture is vague.
3. False prediction: it seems that CNN failed to capture the pattern deer and believes that it's a bird due to some local similarity.
4. Underfitting: CNN couldn't tell the difference between a SUV and truck, but seems to know it's a vechile, which may be improved by more traning samples.

Table 5: CaffeNet prediction on CIFAR10 with top probabilities



True label: ship
predication probability
ship 9.999993e-01
automobile 6.906423e-07
airplane 5.492342e-08



True label: cat
predication probability
cat 0.709629
dog 0.115369
truck 0.115369



True label: deer
predication probability
bird 0.995539



True label: automobile
predication probability
truck 6.391540e-01

frog 0.001864
deer 0.001414

automobile 3.589041e-01
frog 1.013666e-03

5.2 Reflection

We have showed how to train CNN for CIFAR image classification task in Caffe, and how to improve it with dropout and fine-tuning pre-trained CNN. We summarize the steps as following. 1. Design CNN structure and solver. 2. Preprocess images based on the needs of CNN and Caffe. 3. Train CNN. 4. Plot learning curves, analyze results. 5. Refine hyperparameters such as learning policy, learning rate. 6. Modify CNN structure by adding dropout layers, or loading pre-trained models. 7. Repeat step 3-6 until performance is satisfactory.

5.3 Improvement

Due to limitation of time and computing resources, some of the improvement methods are left out for future study.

1. Fine tuning learning rate and policy, which has been shown by the ConvNet results.
2. Fine tuning dropout layers depths and dropout rate, hopefully we can get a CNN with dropout that has even better accuracy and less variance.
3. More experiments on transfer learning of CIFAR10, including trying different models, and more training iterations and fine-tuning of hyperparameters.

Reference

- [1]: CS231n Convolutional Neural Networks for Visual Recognition
- [2]: Wikipedia: Convolutional neural network
- [3]: ConvNet by Alex Krizhevsky
- [4]: Dropout: A Simple Way to Prevent Neural Networks from Overfitting
- [5]: AlexNet: ImageNet Classification with Deep Convolutional Neural Networks
- [6]: Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition
- [7]: CIFAR10: Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009
- [8]: Sivan Sabato: Classification datasets results