**Caffe and Deep Compression**

**Caffe**

Caffe is a deep learning framework, originally introduced for computer vision problems; Later made convenient for other problems including Text, Sound, Reinforcement Learning etc.;

According to Caffe website, it is mainly preferred for following reasons:

* Expression: models and optimizations are defined as plaintext schemas instead of code.
* Speed: for research and industry alike speed is crucial for state-of-the-art models and massive data.
* Modularity: new tasks and settings require flexibility and extension.
* Openness: scientific and applied progress call for common code, reference models, and reproducibility.
* Community: academic research, start-up prototypes, and industrial applications all share strength by joint discussion and development in a BSD-2 project.

With Caffe-2 recently released, there is a great need for exploring both Caffe and Caffe-2 and document the experience, issues, comparisons etc.

**Deep-Compression**

On other hand, deep learning models generally include deep networks, having large number of neurons and connections and hence, generally very large in size. This makes these models very difficult to be deployed in small devices like mobiles, smart watches etc.;

Hence, Deep-Compression comes into play with recent developments from Song Han et al; This process with certain compressing tasks in pipeline results in reduction of the model size by about 35-50 times, making it very easy for deploying in small devices and faster computations;

Follows a pipeline procedure involving Pruning, Quantization and Huffman Coding.

**Task-Breakdown**

|  |  |
| --- | --- |
| Introduction to Caffe and getting familiar with basic syntax | 1 week |
| Understanding CNN-Basics, Exploring pre-trained models | 1 week |
| Retraining existing models, Building a basic CNN model from scratch | 1 week |
| Understanding RNN-Basics, Retraining existing RNN model | 1 week |
| Deep Compression – Introduction; Familiarising with concepts | 1 week |
| Deep Compression [Pruning-Quantization-Huffman Coding] | 2 weeks |
| Introduction to Caffe-2; Simple model in Caffe-2 | 2 weeks |
| Buffer/Evaluation |  |

**Acceptance-Criteria**

* Should beat the existing benchmarks for the respective pre-trained model;
* For models built from scratch, 90% accuracy criteria must be met;

**Documentation**

**1.Work**

Week 1:

* Installation and Introduction of Caffe and getting familiar with basic syntax
* Understanding CNN-Basics

Week 2:

* Exploring pre-trained models---Alexnet,Caffenet,VGGnet
* Hands on LeNet for MNIST data
* Classified cats & dogs(kaggle dataset) by using Caffenet--**accuracy(43.3%)**
* Cloud basics

Week 3:

* Classified cats & dogs by using Caffenet after retraining(not complete--**only 3k** iters due to technical issues) through Transfer learning--**accuracy(70%)**
* Developed a basic CNN(BasicNet) architecture from scratch
* Setting up the cloud(GCP)

Week 4:

* Classified cats & dogs by using Caffenet after **complete(10K iterations)** retraining through Transfer learning--**accuracy(94.7%)**
* Classified cats & dogs by using BasicNet--**accuracy(72.4%)(After 10K iterations)**
* Understanding RNN-Basics

Week 4:

* Increased the training data set by **10x** using data augmentation techniques followed in Alexnet training as the training loss in the BasicNet was stagnant at 0.3(ideally, we want it to converge to 0 )
* Basicnet accuracy after **8K** **iterations** on cats and dogs dataset--**82.23%**
* Changed the architecture of (updated)BasicNet--made it deeper,changed kernel sizes,added dropout layers to avoid overfitting --**accuracy(88.9%)(After 10K iterations)**
* Complete Report on cats & dogs dataset---[here](https://docs.google.com/document/d/14hatSnUU0xZxRUJixg_IR6LZvveKtBVct88EjQPQkyI/edit)
* As there are no existing RNN models in caffe to retrain,tried to build a model which does sentimental analysis on IMDB movie reviews--erros(couldn’t find proper tutorials on parameters definition used in lstm layer in caffe)---**Incomplete**

Weeks 5-9:

* Deep Compression – Introduction
* Tried implementing pruning as in the deep compression paper(Song Han) for caffe models but due to lack of proper tutorials on this,couldn’t do it.
* Implemented pruning(similar to that of above but only on FC layers) for a tensor flow model(Lenet-5) following [this](https://github.com/gstaff/tfzip) tutorial.
* Implemented pruning on Conv layers --**accuracy decreased**
* Removed the prune masks in the pb of pruned model
* Complete Report on pruning & quantization(on Lenet-5)---[here](https://docs.google.com/document/d/1AcVAHEFoEIa8pe3RZZ2NI4zra0FUE8avN2KQZf220ms/edit?usp=sharing)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Parameters** | **Parameters Compression Ratio** | **Pb Size** | **Pb Size (gzipped)** | **Pb Compression** | **Acc(%)** |
| Uncompressed | ~60k |  | 247 kB | 229 kb | 18% | 98.6 |
| Uncompressed + Quantization | ~60K |  | 70 kb | 55 kb | ~80% | 98.6 |
| Compressed (With masks)  Compressed (No masks) | ~7.5k    ~7.5k | ~90%    ~90% | 478 kb    247 kb | 64kb    56 kb | ~74%    ~80% | 98.7    98.7 |
| Compressed + Quantization | ~7.5K ~ | 90% | 71 kb | 19 kb | ~93% | 98.7 |

**2.Literature**

1. <http://caffe.berkeleyvision.org/tutorial/>
2. <http://cs231n.github.io/convolutional-networks/>
3. <http://cs231n.github.io/linear-classify/#softmax>
4. [Very good lecture on CNN](https://www.youtube.com/watch?v=GYGYnspV230&list=PL16j5WbGpaM0_Tj8CRmurZ8Kk1gEBc7fg&index=7)
5. <http://cs231n.github.io/transfer-learning/#tf>
6. [All in one place about caffe](http://graphics.cs.cmu.edu/courses/16-824/2016_spring/slides/caffe_tutorial.pdf)(useful for quick reference)
7. [Lecture on RNN](https://www.youtube.com/watch?v=iX5V1WpxxkY&list=PL16j5WbGpaM0_Tj8CRmurZ8Kk1gEBc7fg&index=10)
8. <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
9. <http://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/00-classification.ipynb>
10. <http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/>
11. <http://caffe.berkeleyvision.org/gathered/examples/mnist.html>
12. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
13. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
14. <https://arxiv.org/pdf/1510.00149.pdf> (Deep Compression paper)
15. <https://arxiv.org/pdf/1506.02626.pdf> (Pruning paper)
16. <https://arxiv.org/pdf/1602.07360.pdf> (SquuezNet)
17. <https://arxiv.org/pdf/1502.03167.pdf> (Batch Normalization)
18. <https://stackoverflow.com/questions/32177764/what-is-weight-decay-meta-parameter-in-caffe>
19. <https://stats.stackexchange.com/questions/29130/difference-between-neural-net-weight-decay-and-learning-rate>

* 8 is the link to Alexnet paper,whereas link 9 helps us to start using caffe with python and link 10 is tutorial on transfer learning using caffe

**3.Open Source**

* All the Pre-Trained models used have been open-sourced and can be imported from caffe-master/models.
* <https://github.com/gstaff/tfzip> -- pruning code

**4.FAQs(caffe)**

1. **mirror in transform\_param in data layer**: It adds copies of samples to the training by mirroring existing images (which usually makes sense, since the class of the image is not affected by mirroring, but this gives more training data). This is of no significance in testing, so it is deactivated.
2. **Better clarity on batch size,test\_iter,test\_interval,max\_iter:**

(i) epoch: Forward and Backward pass of all training examples ( not used in Caffe)

(ii) batch: how many images in one pass.

(iii) iterations: how many iterations of batches

(iv) Batch Size:Batch size in mainly depended to your memory in GPU/RAM. Most time it is used

power of two (64,128,256). We always try to choose 256, because it works better with SGD. But for

bigger network we use 64.

(v) Number of Iterations:Number of iterations set number of epoch of learning. Here we use MNIST

as an example to explain this :Training: 60k, batch size: 64, maximum\_iterations= 10k. So, there will

be 10k\*64 = 640k images of learning. This mean, that there will be 10.6 of epochs.(Number of

epochs is hard to set, you should stop when net does not learn any more, or it is overfitting)

Val:10k, batch size: 100, test\_iterations: 100, So, 100\*100: 10K, exactly all images from validation

base.

So, if you would like to test 20k images, you should set ex. batch\_size=100 and test\_iterations: 200. This will allow you to test all validation base in each testing procedure.To sum up, parameters "test\_iterations" and "batch size" in test depend on number of images in test database.Parameters "maximum\_iterations" and "batch size" in train depend on number of epochs you would like to train your net

3. **inv lr\_policy:**returns base\_lr \* (1 + gamma \* iter) ^ (- power) ,where base\_lr, gamma, step and power are

defined in the solver parameter protocol buffer, and iter is the current iteration.

4. **group parameter in Alexnet:**The argument gives the quantity of groups, not the size. If you have 40 inputs and

set g to 20, you'll get 20 "lanes" of 2 channels each; with 50 outputs, you'd get 10 groups of 2 and 10 groups of 3.

More often, you split into a small number of groups, such as 2. In that case, you'd have two processing "lanes" or

groups. For the 40=>50 layer you mention, each group would have 20 inputs and 25 outputs. Each layer will split

in half, with each set of forward and backward propagation working only within its own half, for the range of

layers over which the *group* parameter applies (I think it's all the way to the final layer).The processing advantage

is that instead of 40^2 input connections, you have 2 groups of 20^2connections, or half as many. This accelerates

the processing by roughly 2x, with a very small loss in convergence progress.

\*\*g in above para----->group parameter

5. For creating lmdb database for train and validation datasets see this [link](http://deepdish.io/2015/04/28/creating-lmdb-in-python/) or you can use directly use caffe tools as

[follow](https://stackoverflow.com/questions/31427094/a-guide-to-convert-imageset-cpp/31431716#31431716)

6. **Training and Resuming**

* In caffe, during training files defining the state of the network will be output: .caffemodel and .solverstate. These two files define the current state of the network at a given iteration, and with this information we are able to continue training our network in the case of a hiccup, pause for diagnosis, or a system crash.

**i)Training**

* To begin training, we simply need to call the caffe binary and supply a solver:
  + caffe train -solver solver.prototxt

## ii)Stopping a)Number of Iterations Limit

## We can have our network stop after a specified number of iterations with a parameter in the solver.prototxt named max\_iter.

## For example, we can specify that we would like our network to stop after 60,000 iteration, thus we set the parameter accordingly: max\_iter: 600000.

### b)Manually Stopping

### It is possible to manually stop a network from training by pressing the Ctrl+C key combination. When the stop signal is sent, the network will halt the forward and backwards pass, and output the current state of the network in a .caffemodel and .solverstate titled with the current iteration number.

# c)Resuming

# When a network as stopped training, either due to manual halting or by reaching the maximum iterations, we may continue training our network by telling caffe to train from where we left off. This is as simple as supplying the snapshot flag with the current .solverstate file. For example:

# caffe train -solver solver.prototxt -snapshot train\_190000.solverstate

# In this case we will continue training from iteration 190000.

# caffe train -solver solver.prototxt -snapshot train\_190000.solverstate

The solverstate file, as its name conveys, stores the state of the solver and not any information related to classification results. The model is saved as caffemodel file, which you can use to obtain classification results for your data. If you want to fine-tune your network you may use a pre-trained caffemodel file. This will save time as your network does not need to learn from scratch. But, in case your present training needs to be halted, due to a power cut or an unexpected reboot, you may resume your training from the previous snapshot of the solverstate. The difference between using the solverstate and the caffemodel files is that the former allows you to complete your training in the pre-determined manner while the latter may require changes in certain training parameters such as the maximum number of iterations.

**5.Troubleshoot**

1. Follow the steps in this [link](https://github.com/BVLC/caffe/wiki/Ubuntu-16.04-or-15.10-Installation-Guide) **carefully** to install caffe in your ubuntu system.
2. While installing caffe in ubuntu systems it’s advisable to install it without GPU support(i.e don’t install GPU drivers for the GPU support of caffe as they mess up your system)
3. **Shuffle(randomly)** the data before dividing into train,valid,test datasets--**very very important**