1. **Caffe Installation and Setup** [1]

- Download CUDA.
  - cd ~/Downloads/
  - curl -O "http://developer.download.nvidia.com/compute/cuda/6_5/rel/installers/cuda_6.5.14_linux_64.run"
- Make the downloaded installer file runnable:
  - chmod +x cuda_6.5.14_linux_64.run
- Run the CUDA installer:
  - sudo ./cuda_6.5.14_linux_64.run --kernel-source-path=/usr/src/linux-headers-`uname -r`/
    - Accept the EULA
    - Install the graphics card drivers.
    - Install the toolkit (leave path at default)
    - Install symbolic link
    - Install samples (leave path at default)
- Update the library path
  - echo 'export PATH=/usr/local/cuda/bin:$PATH' >> ~/.bashrc
  - echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/usr/local/lib' >> ~/.bashrc
  - source ~/.bashrc
- Install dependencies:
  - sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libboost-all-dev libhdf5-serial-dev protobuf-compiler gfortran libjpeg62 libfreeimage-dev libatlas-base-dev git python-dev python-pip libgoogle-glog-dev libbz2-dev libxml2-dev libxslt-dev libffi-dev libssl-dev libgflags-dev liblmdb-dev python-yaml
  - sudo easy_install pillow
- Download Caffe:
  - cd ~
  - git clone https://github.com/omair18/Caffe-Setup-Testing-Training

- Extract caffe-master.zip to your home directory.
- Install python dependencies for Caffe:
  - cd caffe
  - cat python/requirements.txt | xargs -L 1 sudo pip install
- Add a couple of symbolic links for some reason:
  - sudo ln -s /usr/include/python2.7/ /usr/local/include/python2.7
  - sudo ln -s /usr/local/lib/python2.7/dist-packages/numpy/core/include/numpy/ /usr/local/include/python2.7/numpy
- Create a Makefile.config from the example:
  - cp Makefile.config.example Makefile.config
  - nano Makefile.config
    - If you are using Virtual machine then Uncomment the line #CPU_ONLY := 1 ()
    - Under PYTHON_INCLUDE, replace /usr/lib/python2.7/dist-packages/numpy/core/include with /usr/local/lib/python2.7/dist-packages/numpy/core/include (i.e. add /local)
- Compile Caffe:

- make pycaffe
- make all
- make test

- Download the ImageNet Caffe model and labels:
  - ./scripts/download_model_binary.py models/bvlc_reference_caffenet
  - ./data/ilsvrc12/get_ilsvrc_aux.sh
- First Download ilsvrc12 package by running $CAFFE_ROOT/data/get_ilsvrc_aux.sh
- Copy caffe_ilsvrc12/synset_words.txt to $CAFFE_ROOT/data/ilsvrc12/
- Test your installation by running the ImageNet model on an image of a kitten:
  - cd ~/caffe (or whatever you called your Caffe directory)
  - python python/classify.py --print_results examples/images/cat.jpg foo
  - Expected result: [('tabby', '0.27933'), ('tiger cat', '0.21915'), ('Egyptian cat', '0.16064'), ('lynx', '0.12844'), ('kit fox', '0.05155')]

2. **Training LeNet on MNIST with Caffe** [2]

- Prepare dataset
  - **./data/mnist/get_mnist.sh**
  - **./examples/mnist/create_mnist.sh**
- We will get 2 lmdb files from the above step
  - *mnist_train_lmdb*
  - *mnist_test_lmdb*
- All the training layers are written in *.prototxt files.
  - **$CAFFE_ROOT/examples/mnist/lenet_train_test.prototxt**
- Layers include
  - Data layer
  - Convolution layer
  - Pooling Layer
  - Fully connected layer (inner product)
  - ReLU layer
  - Loss Layer
- Define the Mnist solver file for train/test protocol buffer definition
  - **$CAFFE_ROOT/examples/mnist/lenet_solver.prototxt:**
  - Contains training iterations, base learning rate, maximum number of iterations, solver mode
- Start the training of model using *train_lenet.sh*
  - **$CAFFE_ROOT/examples/mnist/train_lenet.sh**
  - Simple script where we have to start *caffe* with parameters of *train/test* and prototxt file containing the training layers.
  - **./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt**
- Start the training and it will create a caffemodel at the end, containing our trained network.

3. **Creating LMDB with images data set** [3]

- Suppose we have prepared all images in .JPG/JPEG format and some folder e.g test_images.
- Create 2 files; *train.txt* and *val.txt*. Each file will contain [path/to/image label] of the training and testing data. Label = 0 if feature is present in image, Label = 1 if feature isn't present.
- Reference example **$CAFFE_ROOT/examples/imagenet/create_imagenet.sh**
  - Download ilsvrc12 package by running **$CAFFE_ROOT/data/get_ilsvrc_aux.sh**

- Open **$CAFFE_ROOT/caffe_ilsvrc12/train.txt or val.txt**
- Run the *convert_imagesest* program to convert all the images to lmdb format using
  - **$CAFFE_ROOT/build/tools/convert_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB_NAME**
  - Where FLAGS can be gray, shuffel, backend, resize_width, resize_height, check_size, encoded, encoded type.
  - ROOTFOLDER = folder containing all the images
  - LISTFILE = train.txt/val.txt for creating training/testing database

4. Training with your own Data set [3]

- After creating your own LMDB database, it's time to create prototxt files containing all the layers information of your network.
- References
  - **$CAFFE_ROOT/examples/mnist/lenet.prototxt**
  - **$CAFFE_ROOT/examples/minst/lenet_train_test.prototxt**
  - **$CAFFE_ROOT/examples/mnist/lenet_solver.prototxt**
  - **$CAFFE_ROOT/examples/mnist/train_lenet.sh**
  -
- *lenet.protoxt* and *lenet_train_test.prototxt* should contain the following important things
  - input_dim: (batchsize, #channels, imagesize1,imagesize2)
  - inner_product_param: num_output: #labels
  - “data” layer: phase: TRAIN/TEST source: “…”(path to training/testing lmdb database)
  - the default path in “data” layer omit **caffe_root**, and you need to call **./build/tools/caffe** at caffe_root
- *lenet_solver.protoxt* should contain following things
  - test_iter: #test batches (the total test images evaluated in TRAIN phase = test_iter*TEST batch size)
  - base_lr: initial learning rate (default:.01, change to a smaller number if getting NAN loss in training)
  - snapshot_prefix: “…”(filepath to snapshot saved)
  - snapshot: # of iterations per snapshot
  - max_iter: # of iterations
- Change path of prototxt files in *train_lenet.sh* file(see “Training LeNet on MNIST using Caffe” for reference)
- Start training by running train_lenet.sh

5. Testing a Trained model with test image set

- Create a LMDB database of the testing image set. (See “Create LMDB with image dataset” for reference)
- You must have prototxt files available and also the snapshot of trained model.
- Use the following python script to test your trained mode.
  https://github.com/omair18/Caffe-Setup-Testing-Training/blob/master/test_your_own_model.py

6. Create Mean of image set (binaryproto)

- Crop all images to 227x227 or 256x256
- Create LMDB database of the image set . (See 3.)
- Now run **$CAFFE_ROOT/build/tools/compute_image_mean [LMDB] [FILE NAME]**
- where

- LMDB is the database you created in step 2
- FILE NAME is the destination filename that will be created with .binaryproto extension

7. Extract Features from Any layer in Caffe

- Crop all images to 227x227 or 256x256
- Create LMDB database of the image set . (See 3.)
- Create a prototxt file, in which we have to mention our LMDB database and mean of image set (binaryproto)
    - Reference **$CAFFE_ROOT/examples/feature_extraction/imagenet_val.protoxt**
- Now Run **$CAFFE_ROOT/build/tools/extract_features.bin [MODEL File] [NETWORK] [LAYER] [DESTINATION] [BATCHSIZE] [LMDB]**
- where
    - MODEL FILE = pretrained caffe model e.g **$CAFFE_ROOT/models/bvlc_reference_caffe_net/bvlc_caffenet.caffemodel**
    - NETWORK = Prototxt file we just made in step 3
    - LAYER = Layer name whose features we want to extract e.g fc7
    - DESTINATION = Folder name to save results
    - BATCH SIZE e.g 10
    - LMDB = to store resutls in LMDB format

8. Running Deep Joint Task Learning for Generic Object Extraction Code on Object Discovery Dataset

- Extract Code zip file in folder (i.e Caffe-Segmentation)
- Set Path of “Makefile.config” of your Caffe-ROOT in Makefile
- run make to compile all the required files
- Go to folder “segscripts”
- Put all your images in “segscripts/data/corpus”
- Create a file in “segscripts/data” named as “ImgsList_IDL_input.txt”
    - File must contain Image path (placed in corpus) and Label
    - e.g “data/corpus/01.jpg 1”
- Create a file in “segscipts/data” named as “ImgsList.txt”
    - File must contain only the list of all images present in “corpus” folder
- segscripts/Models folder must have 3 files
    - loc.caffemodel

    - mean.binaryproto
    - seg.caffemodel
- segscripts/loc folder must have a Protoxt file, defining all the Network layers for this model
- Now run “run_seg.sh” availalbe in “Segscripts” folder
    - **$Caffe-Segmentation/segscripts/run_seg.sh ../**

9. Fine Tuning Pre-trained Caffe model [5]

Fine-tuning takes an already learned model, adapts the architecture, and resumes training from the already learned model weights. Let’s fine-tune the BVLC-distributed CaffeNet model on a different dataset, Flickr Style, to predict image style instead of object category.

The Flickr-sourced images of the Style dataset are visually very similar to the ImageNet dataset, on which the *bvlc_reference_caffenet* was trained. Since that model works well for object category classification, we’d like to use it architecture for our style classifier. We also only have 80,000 images to train on, so we’d like to start with the parameters learned on the 1,000,000 ImageNet images, and fine-tune as needed. If we give provide the weights argument to the caffe train command, the pretrained weights will be loaded into our model, matching layers by name.

Because we are predicting 20 classes instead of a 1,000, we do need to change the last layer in the model. Therefore, we change the name of the last layer from fc8 to fc8_flickr in our prototxt. Since there is no layer named that in the *bvlc_reference_caffenet*, that layer will begin training with random weights.
We will also decrease the overall learning rate *base_lr* in the solver *prototxt*, but boost the *blobs_lron* the newly introduced layer. The idea is to have the rest of the model change very slowly with new data, but let the new layer learn fast. Additionally, we set *stepsize* in the solver to a lower value than if we were training from scratch, since we’re virtually far along in training and therefore want the learning rate to go down faster. Note that we could also entirely prevent fine-tuning of all layers other than fc8_flickr by setting their *blobs_lr* to 0.

Procedure:

- Create Lmdb database for your train and test images. (section 3)
- Create labeled files for your train/test images as *train.txt* and *test.txt* . Each file contains *path/to/image <space> label*
- Correctly update paths of label files and lmdb files in *train_val.protoxt* from *bvlc_reference_caffenet*.

- Change *train_val.protoxt* and add another *fully connected layer* at the end as
-

```
layer {
  name: "fc8_flickr"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8_flickr"
  # lr_mult is set to higher than for other layers, because this
layer is starting from random while the others are already
trained
  param {
    lr_mult: 10
    decay_mult: 1
  }
  param {
    lr_mult: 20
    decay_mult: 0
  }
  inner_product_param {
    num_output: 20
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc8_flickr"
  bottom: "label"
  top: "loss"
}
```

- Now run the fine tuning procedure as
  - **$CAFFE_ROOT ./build/tools/caffe train -solver models/finetune_flickr_style/solver.prototxt -weights models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel**

References

1. https://github.com/BVLC/caffe/wiki/Ubuntu-14.04-VirtualBox-VM
2. http://caffe.berkeleyvision.org/gathered/examples/mnist.html
3. http://sites.duke.edu/rachelmemo/2015/04/03/train-and-test-lenet-on-your-own-dataset/
4. https://github.com/BVLC/caffe.git
5. http://caffe.berkeleyvision.org/gathered/examples/finetune_flickr_style.html