Part 3a. Training with Caffe: miniVggNet

XILINX

What is miniVggNet

- It is a CNN inspired by the original VGG, suitable for the CIFAR10 dataset
- **It is published on the great book "Deep Learning for Computer Vision with Python: Practitioner Bundle" by Dr. Adrian Rosebrock (pyimagesearch), briefly cited as [DL4CV_PB]**
- Original description and training scripts are available only in Keras/Tensorflow
- Prediction accuracy in the range of 80-82%, depending on how it is measured
- For Caffe, I re-wrote the prototxt description and solver files and trained the CNN from scratch, doing the proper changes as Caffe and Keras are pretty different environments for ML development
- To get the same results (within 2% more or less) shown in the next pages, run the following script:
  - `source caffe_flow_miniVggNet.sh`

MiniVggNet model in Keras, from [DL4CV_PB]

- MiniVggNet consists of
  - two sets of CONV=> RELU=>CONV=>RELU=>POOL
  - followed by a set of FC=>RELU=>FC=>SOFMAX layers
  - The first 2 CONV layers will learn 32 filters of 3x3 size
  - The second 2 CONV layers will learn 64 filters again of 3x3 size
  - POOL performs pooling over a 2x2 window with a 2x2 stride
  - Batch Normalization (BN) are inserted after the RELU activations (ACT) while DROPOUT layers after the POOL and FC layers

Layer Type	Output Size	Filter Size / Stride
INPUT IMAGE	$32 \times 32 \times 3$	
CONV	$32 \times 32 \times 32$	$3 \times 3, K = 32$
ACT RELU	$32 \times 32 \times 32$	
BN	$32 \times 32 \times 32$	
CONV	$32 \times 32 \times 32$	$3 \times 3, K = 32$
ACT RELU	$32 \times 32 \times 32$	
BN	$32 \times 32 \times 32$	
POOL	$16 \times 16 \times 32$	2×2
DROPOUT	$16 \times 16 \times 32$	
CONV	$16 \times 16 \times 64$	$3 \times 3, K = 64$
ACT RELU	$16 \times 16 \times 64$	
BN	$16 \times 16 \times 64$	
CONV	$16 \times 16 \times 64$	$3 \times 3, K = 64$
ACT RELU	$16 \times 16 \times 64$	
BN	$16 \times 16 \times 64$	
POOL	$8 \times 8 \times 64$	2×2
DROPOUT	$8 \times 8 \times 64$	
FC	512	
ACT RELU	512	
BN	512	
DROPOUT	512	
FC	10	
SOFTMAX	10	

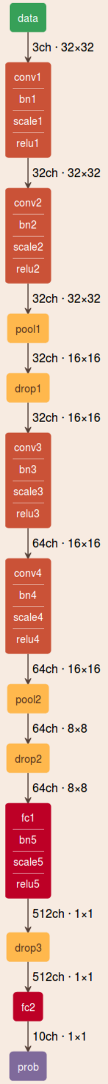
MiniVggNet: Python code from [DL4CV_PB]

```python
# import the necessary packages
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dropout
from keras.layers.core import Dense
from keras import backend as K

class MiniVGGNet:
	@staticmethod
	def build(width, height, depth, classes):
		# initialize the model along with the input shape to be
		# "channels last" and the channels dimension itself
		model = Sequential()
		inputShape = (height, width, depth)
		chanDim = -1

		# if we are using "channels first", update the input shape
		# and channels dimension
		if K.image_data_format() == "channels_first":
			inputShape = (depth, height, width)
			chanDim = 1
```

```python
# first CONV => RELU => CONV => RELU => POOL layer set
model.add(Conv2D(32, (3, 3), padding="same",
	input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(32, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# second CONV => RELU => CONV => RELU => POOL layer set
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# first (and only) set of FC => RELU layers
model.add(Flatten())
model.add(Dense(512))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
# softmax classifier
model.add(Dense(classes))
model.add(Activation("softmax"))

# return the constructed n
return model
```

Keras assumes implicitly stride is equal to max pooling size

miniVggNet: block diagram (in Netscope)



Summary:

ID	name	type	batch	ch_in	dim_in	ch_out	dim_out	ops	mem
1	data	Input		3	32x32	3	32x32		**activation** 3.07k
2	conv1	Convolution		3	32x32	32	32x32	**macc** 884.74k	**activation** 32.77k **param** 896
3	bn1	BatchNorm		32	32x32	32	32x32	**add** 32.77k **div** 32.77k	**activation** 32.77k **param** 64
4	scale1	Scale		32	32x32	32	32x32	**macc** 32.77k	**activation** 32.77k
5	relu1	ReLU		32	32x32	32	32x32	**comp** 32.77k	**activation** 32.77k
6	conv2	Convolution		32	32x32	32	32x32	**macc** 9.44M	**activation** 32.77k **param** 9.25k
7	bn2	BatchNorm		32	32x32	32	32x32	**add** 32.77k **div** 32.77k	**activation** 32.77k **param** 64
8	scale2	Scale		32	32x32	32	32x32	**macc** 32.77k	**activation** 32.77k
9	relu2	ReLU		32	32x32	32	32x32	**comp** 32.77k	**activation** 32.77k
10	pool1	Pooling		32	32x32	32	16x16	**comp** 32.77k	**activation** 8.19k
11	drop1	Dropout		32	16x16	32	16x16	**comp** 8.19k	**activation** 8.19k
12	conv3	Convolution		32	16x16	64	16x16	**macc** 4.72M	**activation** 16.38k **param** 18.5k
13	bn3	BatchNorm		64	16x16	64	16x16	**add** 16.38k **div** 16.38k	**activation** 16.38k **param** 128
14	scale3	Scale		64	16x16	64	16x16	**macc** 16.38k	**activation** 16.38k
15	relu3	ReLU		64	16x16	64	16x16	**comp** 16.38k	**activation** 16.38k
16	conv4	Convolution		64	16x16	64	16x16	**macc** 9.44M	**activation** 16.38k **param** 36.93k
17	bn4	BatchNorm		64	16x16	64	16x16	**add** 16.38k **div** 16.38k	**activation** 16.38k **param** 128
18	scale4	Scale		64	16x16	64	16x16	**macc** 16.38k	**activation** 16.38k
19	relu4	ReLU		64	16x16	64	16x16	**comp** 16.38k	**activation** 16.38k
20	pool2	Pooling		64	16x16	64	8x8	**comp** 16.38k	**activation** 4.1k
21	drop2	Dropout		64	8x8	64	8x8	**comp** 4.1k	**activation** 4.1k
22	fc1	InnerProduct		64	8x8	512	1x1	**macc** 2.1M	**activation** 512 **param** 2.1M
23	bn5	BatchNorm		512	1x1	512	1x1	**add** 512 **div** 512	**activation** 512 **param** 1.02k
24	scale5	Scale		512	1x1	512	1x1	**macc** 512	**activation** 512
25	relu5	ReLU		512	1x1	512	1x1	**comp** 512	**activation** 512
26	drop3	Dropout		512	1x1	512	1x1	**comp** 512	**activation** 512
27	fc2	InnerProduct		512	1x1	10	1x1	**macc** 5.12k	**activation** 10 **param** 5.13k
28	prob	Softmax		10	1x1	10	1x1	**add** 10 **div** 10 **exp** 10	**activation** 10
	TOTAL							**macc** 26.68M **comp** 160.77k **add** 98.83k	**activation** 423.44k **param** 2.17M

XILINX

miniVggNet: deploy and train_val prototxt files

> Remember that there are differences between **train_val_3_miniVggNet.prototxt** and **deploy_3_miniVggNet.prototxt** description files:

- In the training you need to declare the TRAIN and TEST phases and proper databases, while in the deploy you need to remove those layers

- Batch Normalization layers require the following setting:
  - (deploy) `use_global_stats: true`
  - (train_val) `use_global_stats: false`

- The last layer – Loss – requires the following setting
  - (deploy) `name: "prob" type: "Softmax" top: "prob"`
  - (train_val) `name: "loss" type: "SoftmaxWithLoss" top: "loss"`

train_val_3_miniVggNet.prototxt fragments

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
   mean_file: “/home/ML/cifar10/input/mean.binaryproto”
  }
  data_param {
    source: "/home/ML/cifar10/input/lmdb/train_lmdb”
    batch_size: 128
    backend: LMDB
  }
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mirror: false
   mean_file: “/home/ML/cifar10/input/mean.binaryproto”
  }
  data_param {
    source: "/home/ML/cifar10/input/lmdb/valid_lmdb”
    batch_size: 50
    backend: LMDB
  }
}
-:---  train_val_3_miniVggNet.prototxt   14% L67    (Fundamental)
```

```
  bottom: "drop3"
  top: "fc2"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      #type: "gaussian"
      type: "xavier"
      #std: 0.001
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc2"
  bottom: "label"
  top: "loss"
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "fc2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
-:---  train_val_3_miniVggNet.prototxt   91% L519
```

solver_3_miniVggNet.prototxt

```
net: "/home/ML/cifar10/caffe/models/miniVggNet/m3/train_val_3_miniVggNet.prototxt"
#
test_iter: 180            # test_iter = test dataset size / test batch size
#
test_interval: 1000        # amount of iterations after which the NN will test the performance on the test dataset
#
base_lr: 0.01             # the beginning of learning rate
#
lr_policy: "poly"          # it could be "step", "fixed", "exp", "poly", "sigmoid"
power: 1
#
#gamma: 0.1                # how much the learning rate should be changed every time we reach the next step
#
#stepsize: 8000
display:  100
max_iter: 40000           # end of NN training. Note that max_iter  = num_epochs * training set size / test batch size
#
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
#snapshot_format: HDF5
snapshot_prefix: "/home/ML/cifar10/caffe/models/miniVggNet/m3/snapshot_3_miniVggNet_"
#solver_mode: CPU
solver_mode: GPU
#
#type: "Nesterov"
type: "SGD"

random_seed: 1201

-:---  solver_3_miniVggNet.prototxt   All L1     (Fundamental)
```

logfile_3_miniVggNet.log file (first fragment of training)

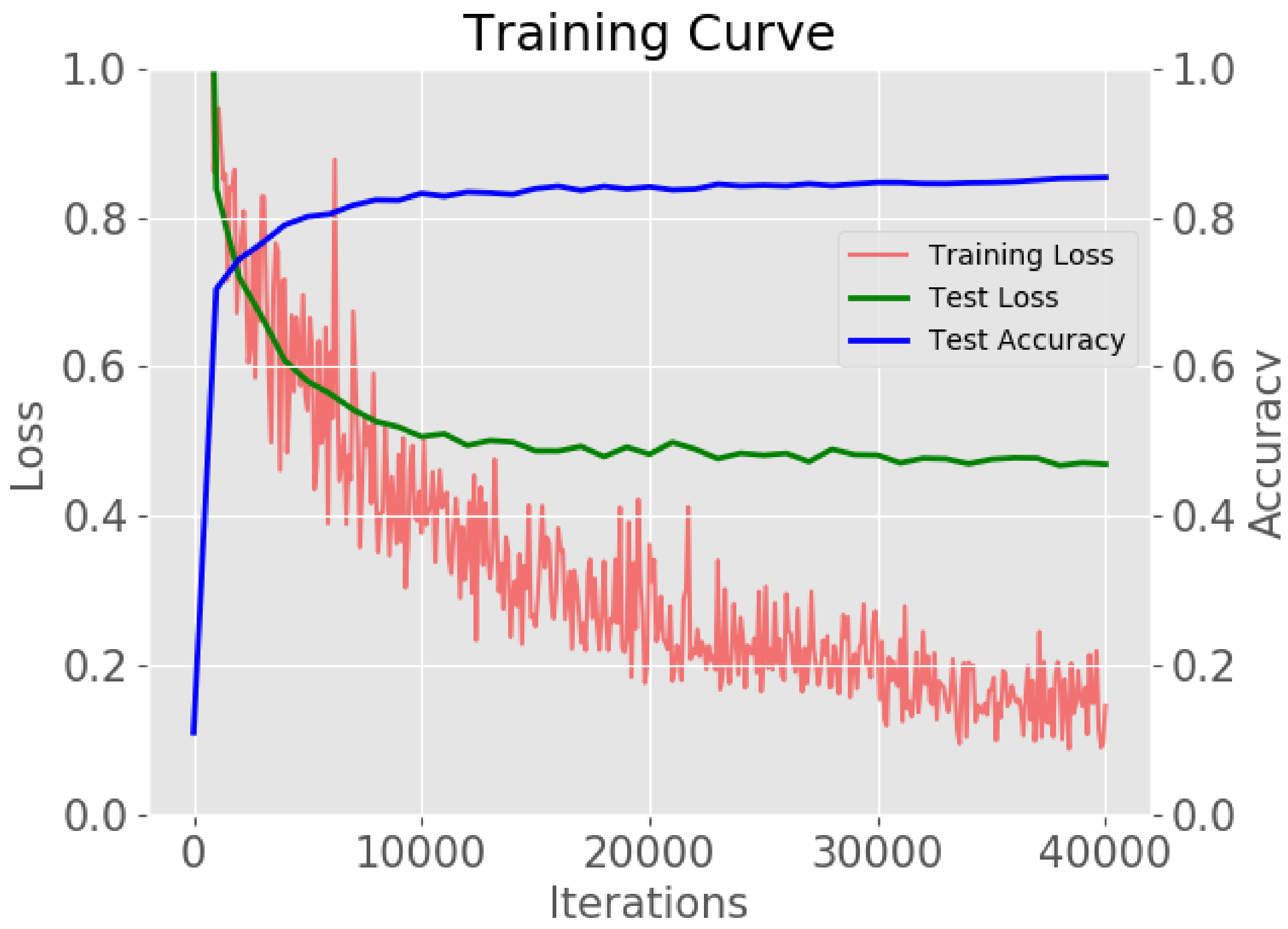
```
I0109 17:13:05.404975 13328 caffe.cpp:204] Using GPUs 0
I0109 17:13:05.430150 13328 caffe.cpp:209] GPU 0: Quadro P6000
I0109 17:13:05.703902 13328 solver.cpp:45] Initializing solver from parameters:
test_iter: 180
test_interval: 1000
base_lr: 0.01
display: 100
max_iter: 40000
lr_policy: "poly"
power: 1
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "/home/ML/cifar10/caffe/models/miniVggNet/m3/snapshot_3_miniVggNet_"
solver_mode: GPU
device_id: 0
random_seed: 1201
net: "/home/ML/cifar10/caffe/models/miniVggNet/m3/train_val_3_miniVggNet.prototxt"
train_state {
  level: 0
  stage: ""
}
type: "SGD"
I0109 17:13:05.704044 13328 solver.cpp:102] Creating training net from net file: /home/ML/cifar10/caffe/models/miniVggNet/m3/train_val_3_miniVggNet.prototxt
I0109 17:13:05.704278 13328 upgrade_proto.cpp:79] Attempting to upgrade batch norm layers using deprecated params: /home/ML/cifar10/caffe/models/miniVggNet/m3/train_val_3_miniVggNet.pr
I0109 17:13:05.704286 13328 upgrade_proto.cpp:82] Successfully upgraded batch norm layers using deprecated params.
I0109 17:13:05.704350 13328 net.cpp:294] The NetState phase (0) differed from the phase (1) specified by a rule in layer data
I0109 17:13:05.704363 13328 net.cpp:294] The NetState phase (0) differed from the phase (1) specified by a rule in layer accuracy
I0109 17:13:05.704365 13328 net.cpp:294] The NetState phase (0) differed from the phase (1) specified by a rule in layer accuracy-top1
I0109 17:13:05.704368 13328 net.cpp:294] The NetState phase (0) differed from the phase (1) specified by a rule in layer accuracy-top5
I0109 17:13:05.704507 13328 net.cpp:51] Initializing net from parameters:
name: "miniVggNet on Cifar10 m3 NO-inPlace"
state {
  phase: TRAIN
  level: 0
  stage: ""
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
```


XILINX

logfile_3_miniVggNet.log file (last fragment of training)

```
I0109 17:25:31.738238 13328 sgd_solver.cpp:112] Iteration 39000, lr = 0.00025
I0109 17:25:32.860574 13336 data_layer.cpp:73] Restarting data prefetching from start.
I0109 17:25:33.639510 13328 solver.cpp:239] Iteration 39100 (52.5988 iter/s, 1.90119s/100 iters), loss = 0.173129
I0109 17:25:33.639539 13328 solver.cpp:258]     Train net output #0: loss = 0.173129 (* 1 = 0.173129 loss)
I0109 17:25:33.639544 13328 sgd_solver.cpp:112] Iteration 39100, lr = 0.000225
I0109 17:25:35.562705 13328 solver.cpp:239] Iteration 39200 (52.0001 iter/s, 1.92307s/100 iters), loss = 0.111885
I0109 17:25:35.562731 13328 solver.cpp:258]     Train net output #0: loss = 0.111885 (* 1 = 0.111885 loss)
I0109 17:25:35.562736 13328 sgd_solver.cpp:112] Iteration 39200, lr = 0.0002
I0109 17:25:37.426491 13328 solver.cpp:239] Iteration 39300 (53.6575 iter/s, 1.86367s/100 iters), loss = 0.136369
I0109 17:25:37.426517 13328 solver.cpp:258]     Train net output #0: loss = 0.136369 (* 1 = 0.136369 loss)
I0109 17:25:37.426520 13328 sgd_solver.cpp:112] Iteration 39300, lr = 0.000175
I0109 17:25:39.306041 13328 solver.cpp:239] Iteration 39400 (53.2075 iter/s, 1.87943s/100 iters), loss = 0.184061
I0109 17:25:39.306092 13328 solver.cpp:258]     Train net output #0: loss = 0.184061 (* 1 = 0.184061 loss)
I0109 17:25:39.306097 13328 sgd_solver.cpp:112] Iteration 39400, lr = 0.00015
I0109 17:25:40.229164 13336 data_layer.cpp:73] Restarting data prefetching from start.
I0109 17:25:41.183041 13328 solver.cpp:239] Iteration 39500 (53.2805 iter/s, 1.87686s/100 iters), loss = 0.129273
I0109 17:25:41.183068 13328 solver.cpp:258]     Train net output #0: loss = 0.129273 (* 1 = 0.129273 loss)
I0109 17:25:41.183071 13328 sgd_solver.cpp:112] Iteration 39500, lr = 0.000125
I0109 17:25:43.079367 13328 solver.cpp:239] Iteration 39600 (52.7368 iter/s, 1.89621s/100 iters), loss = 0.118099
I0109 17:25:43.079393 13328 solver.cpp:258]     Train net output #0: loss = 0.118099 (* 1 = 0.118099 loss)
I0109 17:25:43.079397 13328 sgd_solver.cpp:112] Iteration 39600, lr = 9.99999e-05
I0109 17:25:44.965859 13328 solver.cpp:239] Iteration 39700 (53.0117 iter/s, 1.88638s/100 iters), loss = 0.116446
I0109 17:25:44.965886 13328 solver.cpp:258]     Train net output #0: loss = 0.116446 (* 1 = 0.116446 loss)
I0109 17:25:44.965890 13328 sgd_solver.cpp:112] Iteration 39700, lr = 7.49999e-05
I0109 17:25:46.821291 13328 solver.cpp:239] Iteration 39800 (53.8992 iter/s, 1.85532s/100 iters), loss = 0.125056
I0109 17:25:46.821318 13328 solver.cpp:258]     Train net output #0: loss = 0.125056 (* 1 = 0.125056 loss)
I0109 17:25:46.821322 13328 sgd_solver.cpp:112] Iteration 39800, lr = 5e-05
I0109 17:25:47.552762 13336 data_layer.cpp:73] Restarting data prefetching from start.
I0109 17:25:48.683285 13328 solver.cpp:239] Iteration 39900 (53.7092 iter/s, 1.86188s/100 iters), loss = 0.177063
I0109 17:25:48.683310 13328 solver.cpp:258]     Train net output #0: loss = 0.177063 (* 1 = 0.177063 loss)
I0109 17:25:48.683315 13328 sgd_solver.cpp:112] Iteration 39900, lr = 2.5e-05
I0109 17:25:50.525246 13328 solver.cpp:464] Snapshotting to binary proto file /home/ML/cifar10/caffe/models/miniVggNet/m3/snapshot_3_miniVggNet__iter_40000.caffemodel
I0109 17:25:50.549196 13328 sgd_solver.cpp:284] Snapshotting solver state to binary proto file /home/ML/cifar10/caffe/models/miniVggNet/m3/snapshot_3_miniVggNet__iter_40000.solverstat
I0109 17:25:50.569916 13328 solver.cpp:327] Iteration 40000, loss = 0.133656
I0109 17:25:50.569937 13328 solver.cpp:347] Iteration 40000, Testing net (#0)
I0109 17:25:50.975668 13337 data_layer.cpp:73] Restarting data prefetching from start.
I0109 17:25:50.983530 13328 solver.cpp:414]     Test net output #0: accuracy = 0.857111
I0109 17:25:50.983548 13328 solver.cpp:414]     Test net output #1: loss = 0.449977 (* 1 = 0.449977 loss)
I0109 17:25:50.983551 13328 solver.cpp:414]     Test net output #2: top-1 = 0.857111
I0109 17:25:50.983554 13328 solver.cpp:414]     Test net output #3: top-5 = 0.991889
I0109 17:25:50.983556 13328 solver.cpp:332] Optimization Done.
I0109 17:25:50.983559 13328 caffe.cpp:250] Optimization Done.
```

miniVggNet: learning curves



XILINX.

Average top1 accuracy on the test dataset: 87%

```
IMAGE: /home/ML/cifar10/input/cifar10_jpg/test/deer_00076.jpg
PREDICTED: 5
EXPECTED : 4
-------
IMAGE: /home/ML/cifar10/input/cifar10_jpg/test/horse_00085.jpg
PREDICTED: 7
EXPECTED : 7
-------
IMAGE: /home/ML/cifar10/input/cifar10_jpg/test/ship_00003.jpg
PREDICTED: 8
EXPECTED : 8
-------
IMAGE: /home/ML/cifar10/input/cifar10_jpg/test/truck_00033.jpg
PREDICTED: 9
EXPECTED : 9
-------
             precision    recall  f1-score   support

   airplane       0.89      0.87      0.88       100
 automobile       0.91      0.94      0.93       100
       bird       0.83      0.82      0.82       100
        cat       0.73      0.70      0.71       100
       deer       0.81      0.83      0.82       100
        dog       0.83      0.83      0.83       100
       frog       0.87      0.90      0.88       100
      horse       0.99      0.94      0.96       100
       ship       0.89      0.93      0.91       100
      truck       0.94      0.92      0.93       100

avg / total       0.87      0.87      0.87      1000

SKLEARN Accuracy = 0.87


TOP-5 ACCURACY                    = 1.00
TOP-5 FALSE                       =  0
TOP-5 TRUE                        =  1000


TOTAL NUMBER OF TRUE  PREDICTIONS =  868
TOTAL NUMBER OF FALSE PREDICTIONS =  132
MANUALLY COMPUTED RECALL = 0.87
```