run_train_distracted_drivers-maxout8-size-261

August 4, 2016

The changes here are larger images but same maxout.

In [1]:
```
from skimage import io, transform, exposure, color, util
import os, itertools, sys
from PIL import Image
%pylab inline
sys.setrecursionlimit(1000000)
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
# data_dir = "/home/dylan/IdeaProjects/distracted_drivers/train/"
data_dir =  "/media/dylan/Science/Kaggle-Data/distracted_drivers/train/"
```

In [3]:
```
input_volume_shape = (261, 261)
```

In [4]:
```
def read_img_file_PIL(file_path, size=(32,32)):
    img = Image.open(file_path).convert('L')
    img.thumbnail(size, Image.NEAREST)
    data = np.array(img)
    shape = data.shape
    append_top = int(ceil(max(0, size[0] - shape[0])/2.0))
    append_bot = int(floor(max(0, size[0] - shape[0])/2.0))
    data = util.pad(data, ((append_top, append_bot),
                           (0,0)), mode='constant', constant_values=0)
    return data
```

In [5]:
```
def read_img_file(file_path, rescale=0.01):
    img = io.imread(file_path)
    img= color.rgb2gray(img)
    return transform.rescale(img, rescale)
```

In [6]:
```
def image_gen_from_dir(directory, batch_size, num_categories, size=input_volume_shape):
    result = {os.path.join(dp, f) : int(os.path.split(dp)[1]) for dp, dn, filenames in os.walk(
                for f in filenames if os.path.splitext(f)[1] == '.jpg'}
    # infinite loop
    while True:
        image_files = []
        labels = []
        # randomly choose batch size samples in result
        for category in range(num_categories):
            file_samples = np.random.choice([k for k, v in result.iteritems() if v == category]
                            size=batch_size, replace=False)
            for file_sample in file_samples:
                image_files.append(read_img_file_PIL(file_sample, size=size))
            labels.extend([v for v in itertools.repeat(category, batch_size)])
```

```
            # end category loop
            X = np.asarray(image_files, dtype=np.float32)
            # -1 to 1 range
            X = exposure.rescale_intensity(X, out_range=(-1,1))
            y = np.asarray(labels, dtype=np.int32)
            yield X, y
```

0.1 Another loader, augmentation time

We'll do 6 augmentations:

```
1.) Translation up to 10 pixels
2.) Rotation up to 15 degrees
3.) Zooming
4.) JPEG compression
5.) Sharpening
6.) Gamma correction
```

We won't do flips since the dataset only contains images from the passenger seat. Perhaps we can revisit this later.

In [7]:
```
from skimage.transform import rotate, warp, AffineTransform
from skimage import filters
from scipy import ndimage, misc
import StringIO
```

In [8]:
```
def random_translate(img):
    shift_random = AffineTransform(translation=(randint(-10, 10), randint(-10, 10)))
    min_value = 0 if min(img.ravel()) > 0 else min(img.ravel())
    return np.float32(warp(img, shift_random, mode='constant', cval=min_value))

def random_rotate(img):
    min_value = 0 if min(img.ravel()) > 0 else min(img.ravel())
    return np.float32(rotate(img, randint(-15, 15), mode='constant', cval=min_value))

def random_zoom(img):
    min_value = 0 if min(img.ravel()) > 0 else min(img.ravel())
    scale_random = AffineTransform(scale=(uniform(0.9, 1.1), uniform(0.9, 1.1)))
    return np.float32(warp(img, scale_random, mode='constant', cval=min_value))

def random_compress(img):
    max_v = np.ceil(img.max())
    min_v = np.floor(img.min())
    nd_im = exposure.rescale_intensity(img, out_range=(0, 1)).squeeze()
    nd_im = np.ndarray.astype(nd_im * 255, np.uint8)
    # nd_im = np.ndarray.astype(img * 255, np.uint8)
    im = Image.fromarray(nd_im)
    buf = StringIO.StringIO()
    im.save(buf, "JPEG", quality=np.random.randint(95, 99))
    buf.seek(0)
    im2 = Image.open(buf)
    x1 = exposure.rescale_intensity(np.ndarray.astype(np.array(im2), np.float32), out_range=(mi
    return x1
```

```
        def random_sharpening(img):
            blurred_f = ndimage.gaussian_filter(img, 0.5)
            filter_blurred_f = ndimage.gaussian_filter(blurred_f, 1)
            alpha = uniform(0.9, 1.2)
            img = blurred_f + alpha * (blurred_f - filter_blurred_f)
            return exposure.rescale_intensity(img, out_range=(-1 , 1))

        def random_gamma_correction(img):
            max_v = np.ceil(img.max())
            min_v = np.floor(img.min())
            img = exposure.rescale_intensity(img, out_range=(0,1))
            img = exposure.adjust_gamma(img, uniform(0.2, 0.8))
            return exposure.rescale_intensity(img, out_range=(-1, 1))
```

```
In [9]: def random_aug(img):
            choice = np.random.randint(0,6)
            # choose from 4 different augmentations!
            if choice == 0:
                return random_translate(img)
            elif choice == 1:
                return random_rotate(img)
            elif choice == 2:
                return random_zoom(img)
            elif choice == 3:
                return random_compress(img)
            elif choice == 4:
                return random_sharpening(img)
            else:
                return random_gamma_correction(img)
```

```
In [10]: def random_aug_batch(X, aug_algorithm):
             for i in range(X.shape[0]):
                 X[i] = aug_algorithm(X[i])
             return X
```

```
In [11]: def random_aug_gen(gen, aug_algorithm):
             for batchX, batchY in gen:
                 yield random_aug_batch(batchX, aug_algorithm), batchY
```

1 Process Generator with cached elements

```
In [12]: def threaded_generator(generator, num_cached=50):
             import Queue
             queue = Queue.Queue(maxsize=num_cached)
             sentinel = object()  # guaranteed unique reference

             # define producer (putting items into queue)
             def producer():
                 for item in generator:
                     queue.put(item)
                 queue.put(sentinel)

             # start producer (in a background thread)
             import threading
```

```
            thread = threading.Thread(target=producer)
            thread.daemon = True
            thread.start()

            # run as consumer (read items from queue, in current thread)
            item = queue.get()
            while item is not sentinel:
                yield item
                queue.task_done()
                item = queue.get()
```

In [13]:
```
from nolearn.lasagne import NeuralNet
from lasagne.layers import DenseLayer, ReshapeLayer, Upscale2DLayer, Conv2DLayer, InputLayer,
    MaxPool2DLayer, get_all_params, batch_norm, BatchNormLayer, FeaturePoolLayer
import numpy as np
from lasagne.nonlinearities import softmax, leaky_rectify, theano
from lasagne.updates import nesterov_momentum
from nolearn.lasagne import NeuralNet, BatchIterator, PrintLayerInfo, objective
from nolearn.lasagne import TrainSplit
from common import EarlyStopping, EndTrainingFromEarlyStopping
from lasagne.objectives import categorical_crossentropy, aggregate
import cPickle as pickle
from sklearn import metrics
import time, logging, logging.config, logging.handlers
from lasagne.init import Orthogonal
from notebook_functions import load_best_weights
```

```
Couldn't import dot_parser, loading of dot files will not be possible.

Using gpu device 0: GeForce GTX 960 (CNMeM is disabled, CuDNN 4004)
```

def batch_norm(s): return s

In [14]:
```
try:
    from lasagne.layers.dnn import Conv2DDNNLayer, MaxPool2DDNNLayer
    def conv_2_layer_stack(top, num_filters, pad=1):
        conv1 = batch_norm(Conv2DDNNLayer(top, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv2 = batch_norm(Conv2DDNNLayer(conv1, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        return MaxPool2DDNNLayer(conv2, (2, 2), 2)

    def conv_3_layer_stack(top, num_filters, pad=1):
        conv1 = batch_norm(Conv2DDNNLayer(top, num_filters, (3, 3),
                    stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv2 = batch_norm(Conv2DDNNLayer(conv1, num_filters, (3, 3),
                    stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv3 = batch_norm(Conv2DDNNLayer(conv2, num_filters, (3, 3),
                    stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        return MaxPool2DDNNLayer(conv3, (2, 2), 2)

    def conv_4_layer_stack(top, num_filters, pad=1):
        conv1 = batch_norm(Conv2DDNNLayer(top, num_filters, (3, 3),
                    stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv2 = batch_norm(Conv2DDNNLayer(conv1, num_filters, (3, 3),
```

```
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv3 = batch_norm(Conv2DDNNLayer(conv2, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv4 = batch_norm(Conv2DDNNLayer(conv3, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        return MaxPool2DDNNLayer(conv4, (2, 2), 2)

    def conv_6_layer_stack(top, num_filters, pad=1):
        conv1 = batch_norm(Conv2DDNNLayer(top, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv2 = batch_norm(Conv2DDNNLayer(conv1, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv3 = batch_norm(Conv2DDNNLayer(conv2, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv4 = batch_norm(Conv2DDNNLayer(conv3, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv5 = batch_norm(Conv2DDNNLayer(conv4, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        conv6 = batch_norm(Conv2DDNNLayer(conv5, num_filters, (3, 3),
                stride=1, pad=pad, nonlinearity=leaky_rectify, W=Orthogonal()))
        return MaxPool2DLayer(conv6, (2, 2), 2)

except ImportError:
    def conv_2_layer_stack(top, num_filters, pad=1):
        conv1 = batch_norm(Conv2DLayer(
                top, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv2 = batch_norm(Conv2DLayer(
                conv1, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        return MaxPool2DLayer(conv2, (2, 2), 2)

    def conv_3_layer_stack(top, num_filters, pad=1):
        conv1 = batch_norm(Conv2DLayer(
                top, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv2 = batch_norm(Conv2DLayer(
                conv1, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv3 = batch_norm(Conv2DLayer(conv2
                                       , num_filters, (3, 3), stride=1, pad=pad, nonlinearity=
        return MaxPool2DLayer(conv3, (2, 2), 2)

    def conv_4_layer_stack(top, num_filters, pad=1):
        conv1 = batch_norm(Conv2DLayer(
                top, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv2 = batch_norm(Conv2DLayer(
                conv1, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv3 = batch_norm(Conv2DLayer(
                conv2, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv4 = batch_norm(Conv2DLayer(
                conv3, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        return MaxPool2DLayer(conv4, (2, 2), 2)

    def conv_6_layer_stack(top, num_filters, pad=1):
        conv1 = batch_norm(Conv2DLayer(
                top, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv2 = batch_norm(Conv2DLayer(
```

```
                conv1, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv3 = batch_norm(Conv2DLayer(
                conv2, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv4 = batch_norm(Conv2DLayer(
                conv3, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv5 = batch_norm(Conv2DLayer(
                conv4, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        conv6 = batch_norm(Conv2DLayer(
                conv5, num_filters, (3, 3), stride=1, pad=pad, nonlinearity=leaky_rectify))
        return MaxPool2DLayer(conv6, (2, 2), 2)
```

In [15]:
```
k = 8
input_layer = InputLayer((None, 1, input_volume_shape[0], input_volume_shape[1]))
conv1 = batch_norm(Conv2DDNNLayer(input_layer, 32, (7, 7),
                    stride=2, pad=0, nonlinearity=leaky_rectify, W=Orthogonal()))
maxpool1 = MaxPool2DLayer(conv1, (2, 2), 2)

conv_stack_1 = conv_2_layer_stack(maxpool1, 64)
# dropout1 = DropoutLayer(conv_stack_1, p=0.1)

conv_stack_2 = conv_2_layer_stack(conv_stack_1, 128)
# dropout2 = DropoutLayer(conv_stack_2, p=0.2)

conv_stack_3 = conv_2_layer_stack(conv_stack_2, 256)
# dropout3 = DropoutLayer(conv_stack_3, p=0.3)

# conv_stack_4 = conv_2_layer_stack(conv_stack_3, 512, pad=0)
# dropout4 = DropoutLayer(conv_stack_4, p=0.4)

# conv_stack_5 = conv_2_layer_stack(conv_stack_4, 1024, pad=0)
# dropout17 = DropoutLayer(conv_stack_5, p=0.5)

conv4 = batch_norm(Conv2DDNNLayer(conv_stack_3, 512, (3, 3),
        stride=1, pad=0, nonlinearity=leaky_rectify, W=Orthogonal()))
conv5 = batch_norm(Conv2DDNNLayer(conv4, 512, (3, 3),
        stride=1, pad=1, nonlinearity=leaky_rectify, W=Orthogonal()))
conv6 = batch_norm(Conv2DDNNLayer(conv5, 512, (3, 3),
        stride=1, pad=0, nonlinearity=leaky_rectify, W=Orthogonal()))
conv7 = batch_norm(Conv2DDNNLayer(conv6, 512, (3, 3),
        stride=1, pad=1, nonlinearity=leaky_rectify, W=Orthogonal()))
maxpool2 = MaxPool2DLayer(conv7, (2, 2), 2)

dense18 = DenseLayer(maxpool2, 2048, nonlinearity=None)
norm1 = BatchNormLayer(dense18)
maxout1 = FeaturePoolLayer(norm1, k)
dropout19 = DropoutLayer(maxout1, p=0.5)

dense20 = DenseLayer(dropout19, 2048, nonlinearity=None)
norm2 = BatchNormLayer(dense20)
maxout2 = FeaturePoolLayer(norm2, k)

softmax21 = DenseLayer(maxout2, 10, nonlinearity=softmax)
```

1.1 Quality of Life Functions

In [16]:
```
if not os.path.exists("logs"):
    os.mkdir("logs")
logging.config.fileConfig("logging-training.conf")

def regularization_objective(layers, lambda1=0., lambda2=0., *args, **kwargs):
    # default loss
    losses = objective(layers, *args, **kwargs)
    # get layer weights except for the biases
    weights = get_all_params(layers[-1], regularizable=True)
    regularization_term = 0.0
    # sum of abs weights for L1 regularization
    if lambda1 != 0.0:
        sum_abs_weights = sum([abs(w).sum() for w in weights])
        regularization_term += (lambda1 * sum_abs_weights)
    # sum of squares (sum(theta^2))
    if lambda2 != 0.0:
        sum_squared_weights = (1 / 2.0) * sum([(w ** 2).sum() for w in weights])
        regularization_term += (lambda2 * sum_squared_weights)
    # add weights to regular loss
    losses += regularization_term
    return losses

def eval_regularization(net):
    if net.objective_lambda1 == 0 and net.objective_lambda2 == 0:
        return 0
    # check the loss if the regularization term is not overpowering the loss
    weights = get_all_params(net.layers_[-1], regularizable=True)
    # sum of abs weights for L1 regularization
    sum_abs_weights = sum([abs(w).sum() for w in weights])
    # sum of squares (sum(theta^2))
    sum_squared_weights = (1 / 2.0) * sum([(w ** 2).sum() for w in weights])
    # add weights to regular loss
    regularization_term = (net.objective_lambda1 * sum_abs_weights) \
                          + (net.objective_lambda2 * sum_squared_weights)
    return regularization_term


def print_regularization_term(net):
    if net.objective_lambda1 > 0.0 or net.objective_lambda2 > 0.0:
        regularization_term = eval_regularization(net)
        print "Regularization term: {}".format(regularization_term.eval())

def validation_set_loss(_net, _X, _y):
    """We need this to track the validation loss"""
    _yb = _net.predict_proba(_X)
    _y_pred = np.argmax(_yb, axis=1)
    _acc = metrics.accuracy_score(_y, _y_pred)
    loss = aggregate(categorical_crossentropy(_yb, _y))
    loss += eval_regularization(_net)
    return loss, _acc
```

```
def store_model(model_file_name, net):
    directory_name = os.path.dirname(model_file_name)
    model_file_name = os.path.basename(model_file_name)
    if not os.path.exists(directory_name):
        os.makedirs(directory_name)
    # write model
    output_model_file_name = os.path.join(directory_name, model_file_name)
    start_write_time = time.time()
    if os.path.isfile(output_model_file_name):
        os.remove(output_model_file_name)
    with open(output_model_file_name, 'wb') as experiment_model:
        pickle.dump(net, experiment_model)
    total_write_time = time.time() - start_write_time
    m, s = divmod(total_write_time, 60)
    h, m = divmod(m, 60)
    logging.log(logging.INFO, "Duration of saving to disk: %0d:%02d:%02d", h, m, s)

def write_validation_loss_and_store_best(validation_file_name, best_weights_file_name,
                                         net, X_val, y_val, best_vloss, best_acc):
    # write validation loss
    start_validate_time = time.time()
    vLoss, vAcc = validation_set_loss(net, X_val, y_val)
    loss = vLoss.eval()
    current_epoch = net.train_history_[-1]['epoch']
    with open(validation_file_name, 'a') as validation_file:
        validation_file.write("{}, {}, {}\n".format(current_epoch, loss, vAcc))

    total_validate_time = time.time() - start_validate_time
    m, s = divmod(total_validate_time, 60)
    h, m = divmod(m, 60)
    logging.log(logging.INFO, "Duration of validation: %0d:%02d:%02d", h, m, s)

    # store best weights here
    if loss < best_vloss:
        start_bw_time = time.time()
        best_vloss = loss
        best_acc = vAcc
        with open(best_weights_file_name, 'wb') as best_model_file:
            pickle.dump(net.get_all_params_values(), best_model_file, -1)

    return best_vloss, best_acc


class AdjustVariableWithStepSize(object):
    """This class adjusts any variable during training
    """

    def __init__(self, name, start=0.03, steps=3, after_epochs=2000):
        self.name = name
        self.start = start
        self.steps=steps
        self.after_epochs=after_epochs
        self.ls = []
```

```
        def __call__(self, nn, train_history):
            if not self.ls:
                for i in range(self.steps):
                    self.ls.extend(np.repeat(self.start/(np.power(10,i)), self.after_epochs))

            try:
                epoch = train_history[-1]['epoch']
                new_value = np.float32(self.ls[epoch - 1])
                getattr(nn, self.name).set_value(new_value)
            except IndexError:
                pass
```

1.2 CNN

```
In [17]: lambda1 = 0.0
         lambda2 = 5e-3

         net = NeuralNet(
             layers=softmax21,
             max_epochs=1,
             update=nesterov_momentum,
             update_learning_rate=theano.shared(np.float32(0.001)),
             update_momentum = 0.99,
             # update=adam,
             on_epoch_finished=[
                 EarlyStopping(patience=1000),
                 AdjustVariableWithStepSize('update_learning_rate', start=0.001, steps=2, after_epochs=
             ],
             on_training_finished=[
                 EndTrainingFromEarlyStopping()
             ],
             objective=regularization_objective,
             objective_lambda2=lambda2,
             objective_lambda1=lambda1,
             batch_iterator_train=BatchIterator(batch_size=100),
             train_split=TrainSplit(
                 eval_size=0.25),
             # train_split=TrainSplit(eval_size=0.0),
             verbose=3,
         )

In [18]: p = PrintLayerInfo()
         net.initialize()
         # p(net)
```

1.2.1 load cnn instead

```
In [19]: dir_name = 'net.vgg.large.l2.5e3.orthog-norm-maxout8-lr.2.steps-size-261'
         validation_file_name = "{}/vloss-{}.txt".format(dir_name, dir_name)
         model_file_name = "{}/{}.pickle".format(dir_name, dir_name)
         best_weights_file_name = "{}/bw-{}.weights".format(dir_name, dir_name)
         if os.path.exists(dir_name):
             print "Model exists. Loading {}.".format(dir_name)
             with open(model_file_name, 'rb') as reader:
                 net = pickle.load(reader)
```

```
        else:
            print "Training model from the beginning {}".format(dir_name)
```

```
Training model from the beginning net.vgg.large.l2.5e3.orthog-norm-maxout8-lr.2.steps-size-261
```

load_best_weights(best_weights_file_name, net) from nolearn.lasagne.visualize import plot_loss plt.figure(figsize=(15,9)) plt.ylim([0.1,0.5]) plt.plot([v['valid_loss'] for v in net.train_history_])

2 just this time.

net.on_epoch_finished.pop(1) print net.on_epoch_finished net.update_learning_rate=0.001

2.1 Define validation set

In [ ]:
```
val_dir =  "/media/dylan/Science/Kaggle-Data/distracted_drivers/val/"
X_val, y_val = image_gen_from_dir(val_dir, 40, 10, size=input_volume_shape).next()
X_val = X_val.reshape(-1, 1, input_volume_shape[0], input_volume_shape[1])
```

In [ ]:
```
image_gen = image_gen_from_dir(data_dir, 10, 10, size=input_volume_shape)
gen = random_aug_gen(image_gen, random_aug)
threaded_gen = threaded_generator(gen, num_cached=100)

ops_every = 500
best_acc = 0.0
best_vloss = np.inf

start_time = time.time()
try:
    for step, (inputs, targets) in enumerate(threaded_gen):
        shape = inputs.shape
        net.fit(inputs.reshape(shape[0],1, shape[1], shape[2]), targets)
        logging.log(logging.INFO, "Epoch: {}, Training error: {}".format(
            net.train_history_[-1]["epoch"], net.train_history_[-1]["train_loss"]))
        if (step + 1) % ops_every == 0:
            print_regularization_term(net)
            store_model(model_file_name, net)
            # center validation
            best_vloss, best_acc = write_validation_loss_and_store_best(
                validation_file_name, best_weights_file_name, net, X_val, y_val, best_vloss, bes

except StopIteration:
    # terminate if already early stopping
    with open(model_file_name, 'wb') as writer:
        pickle.dump(net, writer)
    total_time = time.time() - start_time
    print("Training successful by early stopping. Elapsed: {}".format(total_time))
```

```
# Neural Network with 14159722 learnable parameters

## Layer information

name                size          total    cap.Y    cap.X    cov.Y    cov.X    filter Y    filter X    fi
-----------------  ----------  -------  -------  -------  -------  -------  ----------  ----------  ----
InputLayer          1x261x261     68121   100.00   100.00   100.00   100.00         261         261
```

Conv2DDNNLayer	32x128x128	524288	100.00	100.00	2.68	2.68	7	7
BatchNormLayer	32x128x128	524288	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	32x128x128	524288	100.00	100.00	100.00	100.00	261	261
MaxPool2DLayer	32x64x64	131072	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	64x64x64	262144	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	64x64x64	262144	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	64x64x64	262144	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	64x64x64	262144	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	64x64x64	262144	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	64x64x64	262144	100.00	100.00	100.00	100.00	261	261
MaxPool2DDNNLayer	64x32x32	65536	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	128x32x32	131072	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	128x32x32	131072	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	128x32x32	131072	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	128x32x32	131072	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	128x32x32	131072	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	128x32x32	131072	100.00	100.00	100.00	100.00	261	261
MaxPool2DDNNLayer	128x16x16	32768	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	256x16x16	65536	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	256x16x16	65536	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	256x16x16	65536	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	256x16x16	65536	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	256x16x16	65536	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	256x16x16	65536	100.00	100.00	100.00	100.00	261	261
MaxPool2DDNNLayer	256x8x8	16384	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	512x6x6	18432	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	512x6x6	18432	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	512x6x6	18432	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	512x6x6	18432	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	512x6x6	18432	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	512x6x6	18432	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	512x4x4	8192	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	512x4x4	8192	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	512x4x4	8192	100.00	100.00	100.00	100.00	261	261
Conv2DDNNLayer	512x4x4	8192	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	512x4x4	8192	100.00	100.00	100.00	100.00	261	261
NonlinearityLayer	512x4x4	8192	100.00	100.00	100.00	100.00	261	261
MaxPool2DLayer	512x2x2	2048	100.00	100.00	100.00	100.00	261	261
DenseLayer	2048	2048	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	2048	2048	100.00	100.00	100.00	100.00	261	261
FeaturePoolLayer	256	256	100.00	100.00	100.00	100.00	261	261
DropoutLayer	256	256	100.00	100.00	100.00	100.00	261	261
DenseLayer	2048	2048	100.00	100.00	100.00	100.00	261	261
BatchNormLayer	2048	2048	100.00	100.00	100.00	100.00	261	261
FeaturePoolLayer	256	256	100.00	100.00	100.00	100.00	261	261
DenseLayer	10	10	100.00	100.00	100.00	100.00	261	261

Explanation
- X, Y: image dimensions
- cap.: learning capacity
- cov.: coverage of image
- magenta: capacity too low (<1/6)
- cyan: image coverage too high (>100%)
- red: capacity too low and coverage too high

```
  epoch    train loss    valid loss    train/val    valid acc  dur
-------  ------------  ------------  -----------  -----------  -----
      1      35.70711      33.73341      1.05851      0.10000  0.98s
2016-07-31 23:07:59,169 - root - INFO - Epoch: 1, Training error: 35.7071113586
      2      35.62210      33.73750      1.05586      0.10000  0.97s
2016-07-31 23:08:00,149 - root - INFO - Epoch: 2, Training error: 35.6221046448
      3      35.18814      33.73967      1.04293      0.16667  0.98s
2016-07-31 23:08:01,145 - root - INFO - Epoch: 3, Training error: 35.1881446838
      4      34.92326      33.74517      1.03491      0.10000  0.95s
2016-07-31 23:08:02,107 - root - INFO - Epoch: 4, Training error: 34.9232635498
      5      34.53671      33.73351      1.02381      0.06667  0.95s
2016-07-31 23:08:03,070 - root - INFO - Epoch: 5, Training error: 34.5367088318
      6      34.31693      33.72486      1.01756      0.16667  0.95s
2016-07-31 23:08:04,051 - root - INFO - Epoch: 6, Training error: 34.3169288635
      7      34.37460      33.72856      1.01915      0.16667  1.01s
2016-07-31 23:08:05,072 - root - INFO - Epoch: 7, Training error: 34.3745956421
      8      34.40658      33.73640      1.01987      0.13333  1.07s
2016-07-31 23:08:06,168 - root - INFO - Epoch: 8, Training error: 34.406578064
      9      34.21739      33.74545      1.01399      0.06667  1.11s
2016-07-31 23:08:07,293 - root - INFO - Epoch: 9, Training error: 34.2173919678
     10      34.22769      33.73416      1.01463      0.03333  1.00s
2016-07-31 23:08:08,318 - root - INFO - Epoch: 10, Training error: 34.2276878357
     11      34.32386      33.76482      1.01656      0.00000  1.13s
2016-07-31 23:08:09,463 - root - INFO - Epoch: 11, Training error: 34.3238601685
     12      34.36395      33.89575      1.01381      0.10000  1.11s
2016-07-31 23:08:10,584 - root - INFO - Epoch: 12, Training error: 34.3639450073
     13      34.49736      34.00295      1.01454      0.10000  1.05s
2016-07-31 23:08:11,643 - root - INFO - Epoch: 13, Training error: 34.4973640442
     14      34.40776      34.89201      0.98612      0.06667  1.01s
2016-07-31 23:08:12,658 - root - INFO - Epoch: 14, Training error: 34.4077644348
     15      34.62708      36.87984      0.93892      0.06667  0.94s
2016-07-31 23:08:13,605 - root - INFO - Epoch: 15, Training error: 34.62707901
     16      34.87041      38.94115      0.89546      0.13333  0.98s
2016-07-31 23:08:14,589 - root - INFO - Epoch: 16, Training error: 34.8704109192
     17      34.80553      42.91374      0.81106      0.13333  0.97s
2016-07-31 23:08:15,575 - root - INFO - Epoch: 17, Training error: 34.8055305481
     18      34.65967      45.42543      0.76300      0.10000  0.99s
2016-07-31 23:08:16,578 - root - INFO - Epoch: 18, Training error: 34.6596679688
     19      34.74032      47.51985      0.73107      0.10000  0.93s
2016-07-31 23:08:17,522 - root - INFO - Epoch: 19, Training error: 34.740322113
     20      34.52382      48.51399      0.71163      0.10000  0.96s
2016-07-31 23:08:18,492 - root - INFO - Epoch: 20, Training error: 34.5238227844
     21      34.68174      51.85040      0.66888      0.06667  0.98s
2016-07-31 23:08:19,476 - root - INFO - Epoch: 21, Training error: 34.6817359924
     22      34.51344      52.82178      0.65339      0.10000  1.10s
2016-07-31 23:08:20,601 - root - INFO - Epoch: 22, Training error: 34.5134353638
     23      34.35563      51.70115      0.66450      0.13333  1.00s
2016-07-31 23:08:21,620 - root - INFO - Epoch: 23, Training error: 34.355632782
     24      34.32787      51.50606      0.66648      0.06667  0.96s
2016-07-31 23:08:22,594 - root - INFO - Epoch: 24, Training error: 34.3278694153
     25      34.24430      48.83182      0.70127      0.10000  0.97s
2016-07-31 23:08:23,577 - root - INFO - Epoch: 25, Training error: 34.2443008423
```

```
     26      34.23243      45.91653      0.74554      0.10000  1.01s
2016-07-31 23:08:24,595 - root - INFO - Epoch: 26, Training error: 34.2324295044
     27      34.08209      48.21726      0.70684      0.13333  0.98s
2016-07-31 23:08:25,587 - root - INFO - Epoch: 27, Training error: 34.0820922852
     28      34.28476      54.34813      0.63084      0.10000  0.95s
2016-07-31 23:08:26,551 - root - INFO - Epoch: 28, Training error: 34.2847557068
     29      33.99178      66.22639      0.51327      0.10000  1.00s
2016-07-31 23:08:27,561 - root - INFO - Epoch: 29, Training error: 33.9917793274
     30      33.98634      71.18796      0.47742      0.10000  0.98s
2016-07-31 23:08:28,548 - root - INFO - Epoch: 30, Training error: 33.9863357544
     31      34.20824      94.85815      0.36063      0.10000  0.97s
2016-07-31 23:08:29,530 - root - INFO - Epoch: 31, Training error: 34.208240509
     32      34.08863      97.63881      0.34913      0.10000  0.92s
2016-07-31 23:08:30,460 - root - INFO - Epoch: 32, Training error: 34.088634491
     33      34.13584     100.28867      0.34038      0.10000  0.95s
2016-07-31 23:08:31,420 - root - INFO - Epoch: 33, Training error: 34.1358413696
     34      34.07228      99.97665      0.34080      0.10000  0.99s
2016-07-31 23:08:32,437 - root - INFO - Epoch: 34, Training error: 34.0722808838
     35      34.13663     106.77155      0.31972      0.10000  0.99s
2016-07-31 23:08:33,448 - root - INFO - Epoch: 35, Training error: 34.1366348267
     36      34.22134     135.58780      0.25239      0.10000  0.97s
2016-07-31 23:08:34,433 - root - INFO - Epoch: 36, Training error: 34.2213439941
     37      34.27357     144.03143      0.23796      0.10000  1.03s
2016-07-31 23:08:35,475 - root - INFO - Epoch: 37, Training error: 34.2735710144
     38      34.06572     135.94006      0.25059      0.10000  0.97s
2016-07-31 23:08:36,464 - root - INFO - Epoch: 38, Training error: 34.0657196045
     39      34.10125     145.17856      0.23489      0.10000  1.03s
2016-07-31 23:08:37,508 - root - INFO - Epoch: 39, Training error: 34.1012458801
     40      34.01466     124.50072      0.27321      0.10000  0.98s
2016-07-31 23:08:38,503 - root - INFO - Epoch: 40, Training error: 34.0146598816
```

2.2 Visualizations

In [ ]:
```
from notebook_functions import plot_validation_loss
```

In [ ]:
```
plot_validation_loss(net, validation_file_name, ylim=[0, 0.5])
```

In [ ]: