Investigating using Convolutional Networks on Weak Lensing data

Adapted from 4_conv_WL

In [5]:
```
# These are all the modules we'll be using later. Make sure you can import them
# before proceeding further.
import numpy as np
import time
import tensorflow as tf
from six.moves import cPickle as pickle
from six.moves import range
import matplotlib.pyplot as plt
from astropy.io import fits
plt.rcParams['image.cmap'] = 'viridis'
plt.rcParams['image.interpolation'] = 'none'
%matplotlib inline
from IPython import display
from mpl_toolkits.axes_grid1 import make_axes_locatable
from matplotlib.ticker import MultipleLocator
import os
```

In [40]:
```
# Reorganize the code a bit by putting the function definitions first.
def rebin(a, shape):
    sh = shape[0],a.shape[0]//shape[0],shape[1],a.shape[1]//shape[1]
    return a.reshape(sh).mean(-1).mean(1)

def getFITS(imagename):
    filename = os.path.join(whereami, path, imagename)
    f = fits.open(filename)
    dataout = f[0].data

    return dataout

def read_WL(path,display=None):
    # this is a version to look at sigma8
    labels=['750', '850']
    imgs = np.zeros([2048//degrade, 2048//degrade, nct, len(labels)])
    for j, label in enumerate(labels):
        for i in range(nct):
            filename = os.path.join(whereami, path, 'smoothWL-conv_m-512b240_Om0.260_
            if display:
                print("i: %d  j: %d  name: %s" % (i, j, 'smoothWL-conv_m-512b240_Om0.
            f = fits.open(filename)
            imgs[:,:,i,j]=rebin(f[0].data, [2048//degrade, 2048//degrade])

    return imgs, labels

def slice_data(data, labels, exp_cut, exp_nshift):
    labels=['750', '850']
    # how many panels across
    npanelx = 2**exp_cut
    # and how big are they?
    panelw = 2048//(degrade*npanelx)
    # how many shifted panels?
    nshift = 2**exp_nshift -1
    # and what are the shifts?
    shiftw =  panelw//2**exp_nshift
    # with 4 rotations, and 2 shifts, we have
    imgs = np.zeros([panelw, panelw, nct,(npanelx**2 +(npanelx-1)**2*nshift**2)*8, le
    # let's figure out where the centers are, and save that data
    x_centers = np.zeros([nct,(npanelx**2 +(npanelx-1)**2*nshift**2)*8, len(labels)]
    y_centers = np.zeros([nct,(npanelx**2 +(npanelx-1)**2*nshift**2)*8, len(labels)]
    for j, label in enumerate(labels):
        for i in range(nct):
            q=0
            for k in range(npanelx):
                for l in range(npanelx):
                    for r in range(4):
                        imgs[:,:,i,q,j] = np.rot90(data[panelw*k:panelw*(k+1),panelw*
                        x_centers[i,q,j] = (panelw*k+panelw*(k+1))/2.
                        y_centers[i,q,j] = (panelw*l+panelw*(l+1))/2.
                        q+=1
                        imgs[:,:,i,q,j] = np.fliplr(np.rot90(data[panelw*k:panelw*(k+
                        x_centers[i,q,j] = (panelw*k+panelw*(k+1))/2.
                        y_centers[i,q,j] = (panelw*l+panelw*(l+1))/2.
                        q+=1
            for k in range(npanelx-1):
                for l in range(npanelx-1):
                    for m in range(nshift):
                        for n in range(nshift):
                            for r in range(4):
                                imgs[:,:,i,q,j] = np.rot90(data[panelw*k+m*shiftw:pan
                                x_centers[i,q,j] = (panelw*k+m*shiftw+panelw*(k+1)+m*
                                y_centers[i,q,j] = (panelw*l+n*shiftw+panelw*(l+1)+n*
                                q+=1
```

In [13]:

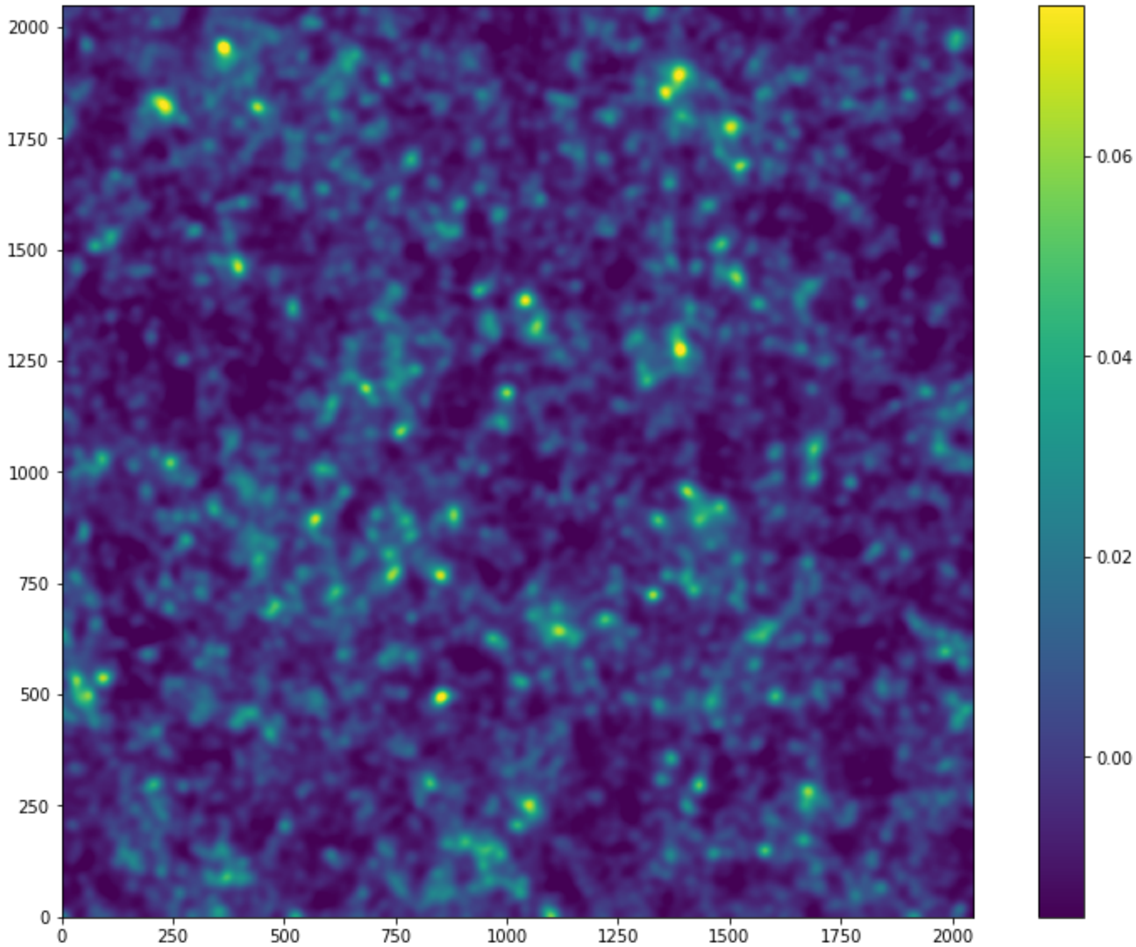
```
# Set the paths to the raw data files
whereami = '/home/jhargis'
#whereami = '/Users/jhargis'
#path     = 'Dropbox/astroNN/wl_maps/'
#whereami = '/Users/goldston'
#whereami = '/Users/jegpeek'
#path = 'Documents/Weak_Lensing/kmaps_smoothed/'

whereami = '/Users/crjones'
path='/Users/crjones/Documents/Science/HargisDDRF/astroNN/data/wl_maps'
```

In [14]:

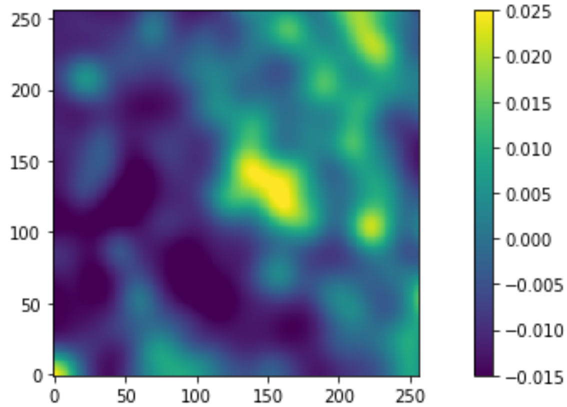
```
# Set (1) the factor by which we want to degrade the original WL maps
# and (2) the number of realizations of each universe.
#
#   The original images are 2048 x 2048, and we degrade them using
#   an 8 x 8 sq.pix box, which makes a smaller set of 64 images
#   (= 256x256 sq.pix in size).

degrade=8
nct = 9

data, labels = read_WL(path,display=True)

print("Data shape : {}".format(data.shape))
print("Labels     : {}".format(labels))
```

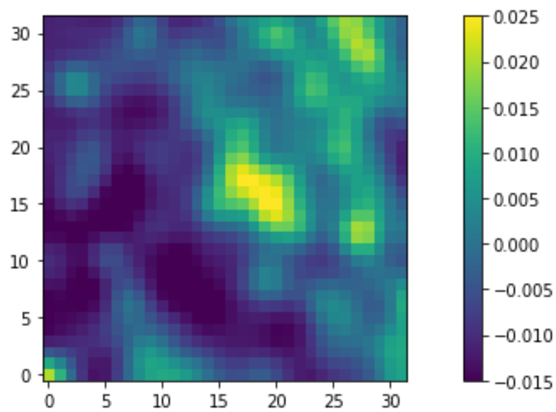
```
i: 0  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0001r_0029p_0100z_og.gre.fit
i: 1  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0002r_0029p_0100z_og.gre.fit
i: 2  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0003r_0029p_0100z_og.gre.fit
i: 3  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0004r_0029p_0100z_og.gre.fit
i: 4  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0005r_0029p_0100z_og.gre.fit
i: 5  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0006r_0029p_0100z_og.gre.fit
i: 6  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0007r_0029p_0100z_og.gre.fit
i: 7  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0008r_0029p_0100z_og.gre.fit
i: 8  j: 0  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.750_
4096xy_0009r_0029p_0100z_og.gre.fit
i: 0  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0001r_0029p_0100z_og.gre.fit
i: 1  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0002r_0029p_0100z_og.gre.fit
i: 2  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0003r_0029p_0100z_og.gre.fit
i: 3  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0004r_0029p_0100z_og.gre.fit
i: 4  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0005r_0029p_0100z_og.gre.fit
i: 5  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0006r_0029p_0100z_og.gre.fit
i: 6  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0007r_0029p_0100z_og.gre.fit
i: 7  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0008r_0029p_0100z_og.gre.fit
i: 8  j: 1  name: smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
4096xy_0009r_0029p_0100z_og.gre.fit
Data shape : (256, 256, 9, 2)
Labels     : ['750', '850']
```

In [16]:
```
# Display an example of the full 2048 x 2048 image
#   Second Universe
fullimage = getFITS("smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.850_
#    First Universe
#fullimage = getFITS("smoothWL-conv_m-512b240_Om0.260_Ol0.740_w-1.000_ns0.960_si0.75(
plt.figure(figsize=(14, 10))
plt.imshow(fullimage, origin="lower")
plt.clim(-.016,0.075)
plt.colorbar()
plt.savefig('/Users/crjones/Documents/Science/HargisDDRF/astroNN/data/tmp/simulation_
```



In [17]:
```
# Now compare a small section of the original image to the rebinned version.
# Becaused we used an 8x8 box, a 256x256 region in the original image
# should correspond to a 32x32 region in the rebinned image.
plt.figure(figsize=(12, 4))
plt.imshow(fullimage[:256,:256],origin="lower")
plt.clim(-0.015,0.025)
plt.colorbar()
plt.show()

image_data = data[:32,:32,8,1]
image_data.shape
plt.figure(figsize=(12, 4))
plt.imshow(image_data, origin='lower')
plt.clim(-0.015,0.025)
plt.colorbar()
plt.show()
```





In [24]:
```
# First we slice the data in the following manner:
#
#   1) Each 256 x 256 image is again split 8 x 8 into 64 images which are 32 x 32 sq
#   2) Series of 4 rotations, 2 flips, and shifts of 2 pixels in size
imgs2, x_centers, y_centers = slice_data(data, labels, 3, 3)
img2sh = imgs2.shape

# Next, reshape the arrays and take the first 7 realizations as the training data se
train_dataset = np.transpose(imgs2[:, :, 0:7, :, :].reshape(img2sh[0], img2sh[1], 7*i
train_xc = x_centers[0:7, :, :].reshape(7*img2sh[3]*2)
train_yc = y_centers[0:7, :, :].reshape(7*img2sh[3]*2)
ones = np.ones([7,img2sh[3], 2] )
train_labels = ((np.asarray([0,1])).reshape(1, 1, 2)*ones).reshape(7*img2sh[3]*2)

# The validation set is the 8th realization
valid_dataset = np.transpose(imgs2[:, :, 7, :, :].reshape(img2sh[0], img2sh[1], 1*img
valid_xc = x_centers[7, :, :].reshape(1*img2sh[3]*2)
valid_yc = y_centers[7, :, :].reshape(1*img2sh[3]*2)
ones = np.ones([1,img2sh[3], 2] )
valid_labels = ((np.asarray([0,1])).reshape(1, 1, 2)*ones).reshape(1*img2sh[3]*2)

# The test data set is the 9th realization
test_dataset = np.transpose(imgs2[:, :, 8, :, :].reshape(img2sh[0], img2sh[1], 1*img2
test_xc = x_centers[8, :, :].reshape(1*img2sh[3]*2)
test_yc = y_centers[8, :, :].reshape(1*img2sh[3]*2)
ones = np.ones([1,img2sh[3], 2] )
test_labels = ((np.asarray([0,1])).reshape(1, 1, 2)*ones).reshape(1*img2sh[3]*2)
```

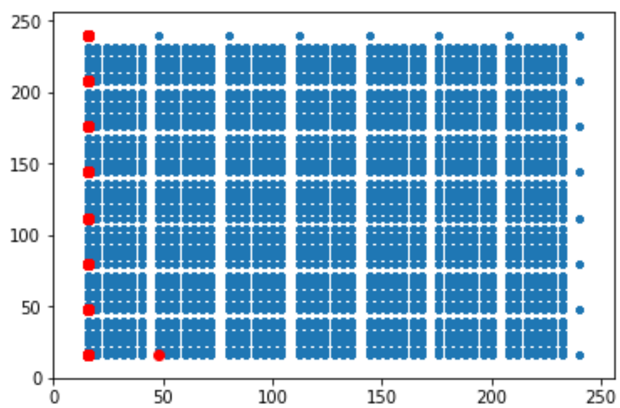
In [26]:
```
print("Master images tensor shape:", img2sh)
print('')
print("Train dataset shape:", train_dataset.shape)
print("Train labels shape :", train_labels.shape)
print("Test dataset shape :", test_dataset.shape)
print("Valid dataset shape:", valid_dataset.shape)
print("TOTAL data sets    : ", train_dataset.shape[0] + test_dataset.shape[0] + valid
```

```
Master images tensor shape: (32, 32, 9, 19720, 2)

Train dataset shape: (276080, 32, 32)
Train labels shape : (276080,)
Test dataset shape : (39440, 32, 32)
Valid dataset shape: (39440, 32, 32)
TOTAL data sets    :  354960
```

In [27]:
```
plt.plot(np.reshape(x_centers, 9*19720*2), np.reshape(y_centers, 9*19720*2), '.')
plt.xlim([0,256])
plt.ylim([0,256])
print(x_centers.shape)
print(x_centers[8,0:65,0])
print(y_centers[8,0:65,0])
indx = np.arange(0,65,8)
print(indx)
plt.plot(x_centers[8,0:65,0], y_centers[8,0:65,0],'ro')
```

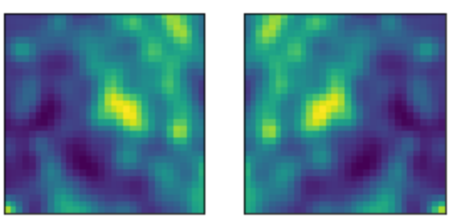
```
(9, 19720, 2)
[ 16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.
  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.
  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.
  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.  16.
  16.  16.  16.  16.  48.]
[  16.   16.   16.   16.   16.   16.   16.   16.   48.   48.   48.   48.
   48.   48.   48.   48.   80.   80.   80.   80.   80.   80.   80.   80.
  112.  112.  112.  112.  112.  112.  112.  112.  144.  144.  144.  144.
  144.  144.  144.  144.  176.  176.  176.  176.  176.  176.  176.  176.
  208.  208.  208.  208.  208.  208.  208.  208.  240.  240.  240.  240.
  240.  240.  240.  240.   16.]
[ 0  8 16 24 32 40 48 56 64]
```

Out[27]: [<matplotlib.lines.Line2D at 0x11968c898>]



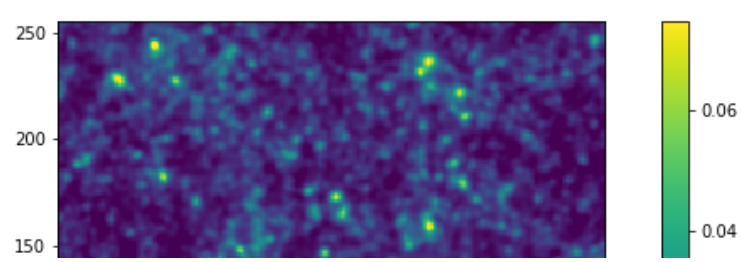
In [28]:
```
fig = plt.figure(figsize=(15, 7))
sub1 = plt.subplot(2, 6, 1)
sub1.set_xticks(())
sub1.set_yticks(())
sub1.imshow(image_data, origin='lower')

image_data = test_dataset[3,:32,:32]
sub2 = plt.subplot(2, 6, 2)
sub2.set_xticks(())
sub2.set_yticks(())
sub2.imshow(image_data, origin='lower')
```

Out[28]: <matplotlib.image.AxesImage at 0x1034acb00>



In [30]:
```
# Loop to get array elements for grid
even = 0
odd  = 1
indv = np.arange(odd,19720*2, 2)
k = 0
test_grid = []
for i in range(2465):
    test_grid.append(indv[k:k+8])
    k+=8

#print("Full test grid:",len(test_grid))
#print
#print(test_grid[-1])
# [2464]

# Plot the full figure (rebinned)
plt.figure(figsize=(12, 6))
image_data = data[:,:,8,1]
#plt.imshow(fullimage, origin="lower")
plt.imshow(image_data, origin='lower')
plt.clim(-.016,0.075)
plt.colorbar()
plt.savefig('/Users/crjones/Documents/Science/HargisDDRF/astroNN/data/tmp/simulation_


# Plot based on setup grid
for gridpt in range(61,62):
#for gridpt in range(0,3):
#for gridpt in range(66,67):
#for gridpt in range(2464,2465):
    fig, axes = plt.subplots(nrows=1, ncols=8,figsize=(10,10))
    #gridpt = 0
    j = 0
    print("test_grid:",test_grid[gridpt])
    for ax in axes.flat:
        idx = test_grid[gridpt][j]
        pldata = test_dataset[idx, :32, :32]
        im = ax.imshow(pldata, origin='lower', vmin=-0.016, vmax=0.075)
        ax.set_xticks(())
        ax.set_yticks(())
        fig.savefig("/Users/crjones/Documents/Science/HargisDDRF/astroNN/data/tmp/sir
        j+=1
    fig.subplots_adjust(right=1.)
    print("X:",x_centers[8,:,0][np.arange(gridpt*8,gridpt*8+8)][0])
    print("Y:",y_centers[8,:,0][np.arange(gridpt*8,gridpt*8+8)][0])

#cbar_ax = fig.add_axes(([0.85, 0.15, 0.05, 0.7]))
#fig.colorbar(im, cax=cbar_ax)
```

```
test_grid: [977 979 981 983 985 987 989 991]
X: 240.0
Y: 176.0
```



In [12]:
```
# Just don't even worry about this for now.
# Figure it out later.

"""
#pickle_file = 'notMNIST.pickle'
#pickle_file = '/Users/jegpeek/Documents/WL88.pickle'
pickle_file = '/Users/jegpeek/Dropbox/WL_other.pickle'

usePickle = True

if usePickle:
    with open(pickle_file, 'rb') as f:
      save = pickle.load(f)
      train_dataset = save['train_dataset']
      train_labels = save['train_labels']
      valid_dataset = save['valid_dataset']
      valid_labels = save['valid_labels']
      test_dataset = save['test_dataset']
      test_labels = save['test_labels']
      del save  # hint to help gc free up memory
      print('Training set', train_dataset.shape, train_labels.shape)
      print('Validation set', valid_dataset.shape, valid_labels.shape)
      print('Test set', test_dataset.shape, test_labels.shape)
else:
    %run Read_WL.py
"""
```

Out[12]:
```
"\n#pickle_file = 'notMNIST.pickle'\n#pickle_file = '/Users/jegpeek/Documents/WL88
.pickle'\npickle_file = '/Users/jegpeek/Dropbox/WL_other.pickle'\n\nusePickle = Tr
ue\n\nif usePickle:\n    with open(pickle_file, 'rb') as f:\n      save = pickle.l
oad(f)\n      train_dataset = save['train_dataset']\n      train_labels = save['tr
ain_labels']\n      valid_dataset = save['valid_dataset']\n      valid_labels = sa
ve['valid_labels']\n      test_dataset = save['test_dataset']\n      test_labels =
save['test_labels']\n      del save  # hint to help gc free up memory\n      print
('Training set', train_dataset.shape, train_labels.shape)\n      print('Validation
set', valid_dataset.shape, valid_labels.shape)\n      print('Test set', test_datas
et.shape, test_labels.shape)\nelse:\n    %run Read_WL.py\n"
```

In [31]:
```
# Reformat into a TensorFlow-friendly shape:
# - convolutions need the image data formatted as a cube (width by height by #channel
# - labels as float 1-hot encodings.

image_size = 32
num_labels = 2
num_channels = 1 # grayscale

train_dataset, train_labels = reformat(train_dataset, train_labels)
valid_dataset, valid_labels = reformat(valid_dataset, valid_labels)
test_dataset, test_labels = reformat(test_dataset, test_labels)
print('Training set', train_dataset.shape, train_labels.shape)
print('Validation set', valid_dataset.shape, valid_labels.shape)
print('Test set', test_dataset.shape, test_labels.shape)
```

```
Training set (276080, 32, 32, 1) (276080, 2)
Validation set (39440, 32, 32, 1) (39440, 2)
Test set (39440, 32, 32, 1) (39440, 2)
```

In [33]:
```
print("test_labels:", test_labels[test_grid[0]])
```

```
test_labels: [[ 0.  1.]
 [ 0.  1.]
 [ 0.  1.]
 [ 0.  1.]
 [ 0.  1.]
 [ 0.  1.]
 [ 0.  1.]
 [ 0.  1.]]
```

In [15]:
```
# Cut this down to a subset
#test_frac = 0.3
#_train_dataset = train_dataset[0:train_dataset.shape[0]*test_frac,:,:,:]
#_train_labels  = train_labels[0:train_labels.shape[0]*test_frac,:]
#_test_dataset = test_dataset[0:test_dataset.shape[0]*test_frac,:,:,:]
#_test_labels  = test_labels[0:test_labels.shape[0]*test_frac,:]
#_valid_dataset = valid_dataset[0:valid_dataset.shape[0]*test_frac,:,:,:]
#_valid_labels  = valid_labels[0:valid_labels.shape[0]*test_frac,:]
#print('--> Using subset <--')
#print('Training set', _train_dataset.shape, _train_labels.shape)
#print('Validation set', _valid_dataset.shape, _valid_labels.shape)
#print('Test set', _test_dataset.shape, _test_labels.shape)
#train_dataset = _train_dataset
#train_labels = _train_labels
#test_dataset = _test_dataset
#test_labels = _test_labels
#valid_dataset = _valid_dataset
#valid_labels = _valid_labels
#print('Training set', train_dataset.shape, train_labels.shape)
#print('Validation set', valid_dataset.shape, valid_labels.shape)
#print('Test set', test_dataset.shape, test_labels.shape)
```

Let's build a small network with two convolutional layers, followed by one fully connected layer. Convolutional networks are more expensive computationally, so we'll limit its depth and number of fully connected nodes.

In [35]:
```
batch_size = 128  # 16
patch_size = 5    # 5
depth = 32        # 16
depth2= 64        # JRH addition
num_hidden = 2048   # 64

graph = tf.Graph()

with graph.as_default():

    # Input data.
    tf_train_dataset = tf.placeholder(tf.float32, shape=(batch_size, image_size, imag
    tf_train_labels = tf.placeholder(tf.float32, shape=(batch_size, num_labels))
    tf_valid_dataset = tf.constant(valid_dataset)
    tf_test_dataset = tf.constant(test_dataset)

    # Global Step
    #global_step = tf.Variable(1.)
    #learn_decay = 0.85
    #learning_rate = tf.train.exponential_decay(0.005, global_step, 10000, learn_deca
    learning_rate = 0.01

    # Variables.
    #layer1_weights = tf.Variable(tf.truncated_normal([patch_size, patch_size, num_ch
    #layer1_biases = tf.Variable(tf.zeros([depth]))
    #layer2_weights = tf.Variable(tf.truncated_normal([patch_size, patch_size, depth,
    #layer2_biases = tf.Variable(tf.constant(1.0, shape=[depth]))
    #layer3_weights = tf.Variable(tf.truncated_normal([image_size // 4 * image_size /
    #layer3_biases = tf.Variable(tf.constant(1.0, shape=[num_hidden]))
    #layer4_weights = tf.Variable(tf.truncated_normal([num_hidden, num_labels], stdde
    #layer4_biases = tf.Variable(tf.constant(1.0, shape=[num_labels]))

    # Alternate variable setup
    #   Layer 1: Compute 16 features = depth
    layer1_weights = tf.Variable(tf.truncated_normal([patch_size, patch_size, num_cha
    #layer1_biases = tf.Variable(tf.zeros([depth]))
    layer1_biases = tf.Variable(tf.constant(0.1, shape=[depth]))

    #   Layer 2: Compute 32 features = DEPTH2
    layer2_weights = tf.Variable(tf.truncated_normal([patch_size, patch_size, depth,
    layer2_biases = tf.Variable(tf.constant(0.1, shape=[depth2]))

    #   Layer 3: Fully-connected layer should use depth2, which results from layer2
    layer3_weights = tf.Variable(tf.truncated_normal([image_size // 4 * image_size //
    layer3_biases = tf.Variable(tf.constant(0.01, shape=[num_hidden]))

    #   Layer 4: Readout layer
    layer4_weights = tf.Variable(tf.truncated_normal([num_hidden, num_labels], stddev
    layer4_biases = tf.Variable(tf.constant(0.01, shape=[num_labels]))

    keep_prob = tf.placeholder(tf.float32)

    #with tf.name_scope('derp'):
    #  spl = tf.split(3, 16, layer1_weights)
    #  filter_summary = tf.image_summary((spl[0]).name, spl[0], max_images=1)

    # Model.
    def model(data):
        #  Layer 1
        conv = tf.nn.conv2d(data, layer1_weights, [1, 2, 2, 1], padding='SAME')
        hidden = tf.nn.relu(conv + layer1_biases)
        #  Layer 2
        conv = tf.nn.conv2d(hidden, layer2_weights, [1, 2, 2, 1], padding='SAME')
        hidden = tf.nn.relu(conv + layer2_biases)
```

In [36]:
```
valid_prediction.get_shape()
```

Out[36]: TensorShape([Dimension(39440), Dimension(2)])

In [37]:
```
permutation = np.random.permutation(train_labels.shape[0])
train_dataset = train_dataset[permutation,:,:]
train_labels = train_labels[permutation]
```

In [ ]:
```
# Run of TensorFlow without the permutations
# but with the TensorBoard logging calls
num_steps = 200001
print_step = 200
summary_step = 200
losses = np.zeros((num_steps-1)//print_step+1)
acc_valid = np.zeros((num_steps-1)//print_step+1)
acc_test = np.zeros((num_steps-1)//print_step+1)
acc_train = np.zeros((num_steps-1)//print_step+1)
acc_grid = np.zeros(64)
acc_image = np.zeros((256,256))
q = 0
p = 0

with tf.Session(graph=graph) as session:
    tf.initialize_all_variables().run()
    summary_writer = tf.train.SummaryWriter('/Users/crjones/Documents/Science/HargisI
    print('Initialized')
    for step in range(num_steps):
        offset = (step * batch_size) % (train_labels.shape[0] - batch_size)
        batch_data = train_dataset[offset:(offset + batch_size), :, :, :]
        batch_labels = train_labels[offset:(offset + batch_size), :]

        start_time = time.time()
        feed_dict = {tf_train_dataset : batch_data, tf_train_labels : batch_labels}
        summary, _, l, predictions = session.run([merged, optimizer, loss, train_prec
        duration = time.time() - start_time

        if (step % print_step == 0):

            # Save the values of the loss
            # Calculate the overall accuracy (single number) based on each individua
            losses[q] = l
            test_prediction_results = test_prediction.eval()
            acc_valid[q] = accuracy(valid_prediction.eval(), valid_labels)/100.0
            acc_test[q] = accuracy(test_prediction_results, test_labels)/100.0
            acctmp = acc_test[q]*100.
            acc_test_str = '%.1f' % acctmp
            acc_train[q] = accuracy(predictions, batch_labels)/100.0
            q += 1
            if q < 10:
                saveoutname = "0000"+str(q)+".png"  # 00001.png
                saveoutnamepdf = "0000"+str(q)+".pdf"
            elif (q >= 10 and q < 100):
                saveoutname = "000"+str(q)+".png"   # 00010.png
                saveoutnamepdf = "000"+str(q)+".pdf"
            elif (q >= 100 and q < 1000):
                saveoutname = "00"+str(q)+".png"    # 00100.png
                saveoutnamepdf = "00"+str(q)+".pdf"
            elif (q >= 1000 and q < 10000):
                saveoutname = "0"+str(q)+".png"     # 01000.png
                saveoutnamepdf = "0"+str(q)+".png"
            else:
                saveoutname = ""+str(q)+".png"      # 10000.png
                saveoutnamepdf = ""+str(q)+".pdf"


            # Setup a grid of 64 accuracy values based on the 8 images in the defaul
            # The accuracy is the fraction of the 8 images that have P > 0.5.
            for i in range(len(acc_grid)):
                acc_grid[i] = accuracy(test_prediction_results[test_grid[i]], test_la
            acc_image = make_acc_image(acc_grid)
            # Next add in one of the nearby grid points
            for gridpt in range(64,2465):
```

In [ ]:
```
print(len(test_prediction_results))
acc_test
```

In [ ]:
```
print(test_grid[4])
print(test_prediction_results[test_grid[4]])
print(acc_grid[4])
```

In [ ]:
```
test_labels[test_grid[0]]
```

In [ ]:
```
accuracy(test_prediction_results[test_grid[0]],test_labels[test_grid[0]])/100.
acc_grid
```

num_steps = 20001 print_step = 200 losses = np.zeros((num_steps-1)/print_step+1) acc_valid = np.zeros((num_steps-1)/print_step+1) acc_test = np.zeros((num_steps-1)/print_step+1) q = 0 p=0

with tf.Session(graph=graph) as session: tf.initialize_all_variables().run() print('Initialized') for step in range(num_steps): offset = (step batch_size) % (train_labels.shape[0] - batch_size) batch_data = train_dataset[offset:(offset + batch_size), :, :, :] batch_labels = train_labels[offset:(offset + batch_size), :] feed_dict = {tf_train_dataset : batch_data, tf_train_labels : batchlabels} , l, predictions = session.run([optimizer, loss, train_prediction], feed_dict=feed_dict) if (step % print_step == 0): losses[q] = l acc_valid[q] = accuracy(valid_prediction.eval(), valid_labels)/100.0 acc_test[q] = accuracy(test_prediction.eval(), test_labels)/100.0 q += 1 plt.plot(np.arange(0,(num_steps-1)/print_step+1), acc_valid, '.', color='b') plt.plot((-1)np.arange(0,(num_steps-1)/print_step+1),acc_test, '.', color='g') plt.ylim([0.45, 0.65]) plt.xlim([-1000, 1000]) display.clear_output(wait=True) display.display(plt.gcf()) print('Minibatch loss at step %d: %f' % (step, l))

```
        #print('Minibatch accuracy: %.1f%%' % accuracy(predictions, batch_lab
els))
        #print('Validation accuracy: %.1f%%' % accuracy(valid_prediction.eval
(), valid_labels))
print('Test accuracy: %.1f%%' % accuracy(test_prediction.eval(), test_labels)
)
```

plt.plot(np.arange(0,(num_steps-1)/print_step+1), acc_valid, '.', color='b') plt.plot((-1)*np.arange(0,(num_steps-1)/print_step+1),acc_test, '.', color='g') plt.ylim([0.45, 0.75]) plt.xlim([-100, 100])

In [ ]:
```
plt.plot(acc_test, acc_valid, '.', color='b')
#plt.plot((-1)*np.arange(0,(num_steps-1)/print_step+1),losses, '.', color='g')
plt.ylim([0.4, 0.7])
plt.xlim([0.0, 1.0])
```

Open questions:

- Why is there so much scatter in the loss function over time?
- Is there structure in the loss function over time?
- If I plot loss vs. accuracy, what do I get?
- Do I really see a difference when I scramble vs. leave in order, and if so, is it because of the way SGD interacts with the two cosmologies?
- will deeper / better networks get us over 65%?
- Why, oh why, are my test and valid data sets so damn well correlated??

graph.get_tensor_by_name.im_func

Problem 1

The convolutional model above uses convolutions with stride 2 to reduce the dimensionality. Replace the strides by a max pooling operation (`nn.max_pool()`) of stride 2 and kernel size 2.

Problem 2

Try to get the best performance you can using a convolutional net. Look for example at the classic LeNet5 (http://yann.lecun.com/exdb/lenet/) architecture, adding Dropout, and/or adding learning rate decay.
