# NN_with_preprocess

December 10, 2015

```
In [186]: import os
          import matplotlib.pyplot as plt
          %pylab inline
          import numpy as np
          from lasagne.layers import DenseLayer
          from lasagne.layers import InputLayer
          from lasagne.layers import DropoutLayer
          from lasagne.layers import Conv2DLayer
          from lasagne.layers import MaxPool2DLayer
          from lasagne.nonlinearities import softmax
          from lasagne.updates import adam
          from lasagne.layers import get_all_params
          from nolearn.lasagne import NeuralNet
          from nolearn.lasagne import TrainSplit
          from nolearn.lasagne import objective
```

```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: [’axes’, ’f’]
‘%matplotlib‘ prevents importing * from pylab and numpy
```

```
In [187]: import scipy.io
          ’’’
          train = scipy.io.loadmat(’labeled_images.mat’)
          print "Shape of tr_images is: ", train["tr_images"].shape
          (x_size, y_size, n_images) = train["tr_images"].shape
          X = np.reshape(np.swapaxes(train["tr_images"], 0, 2), (n_images, 1, x_size, y_size))
          y = train["tr_labels"].ravel()-1
          print X.shape
          print y.shape
          X = np.array(X).astype(np.float32)
          y = np.array(y).astype(np.int32)
          # Normalization
          X -= X.mean()
          X /= X.std()
          print X[0].shape
          print y.shape
          ’’’

          train = scipy.io.loadmat(’filtered_testimg.mat’)
          train_original = scipy.io.loadmat(’labeled_images.mat’)

          print "Shape of tr_images is: ", train["tr_images"].shape
          # (x, y, n_images) = train["tr_images"].shape
```

```
(n_images, dim) = train["tr_images"].shape
y = train_original["tr_labels"].ravel()-1

# train_img = np.reshape(np.swapaxes(train["tr_images"], 0, 2), (n_images, x * y))
X = np.reshape(train[’tr_images’], (n_images, 1, dim, 1))
y = np.array(y).astype(np.int32)
X = np.array(X).astype(np.float32)
#X -= X.mean()
#X /= X.std()

#plt.imshow(np.swapaxes(np.reshape(train_img[0], (y, x)), 0, 1), cmap=pylab.gray())
#plt.show()



#plt.imshow(np.swapaxes(np.reshape(X[0], (y_size, x_size)), 0, 1), cmap=pylab.gray())
#plt.show()
```

```
Shape of tr_images is:  (2925, 2560)
```

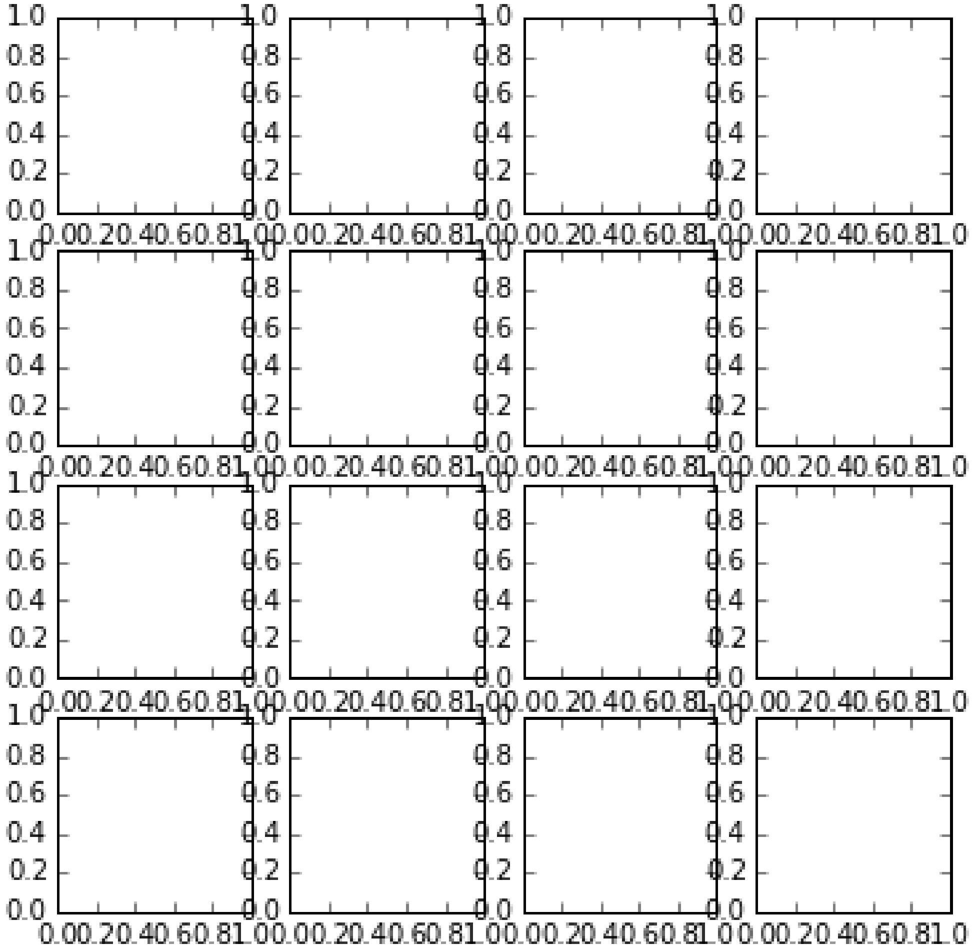
```
In [188]: # Show labels of the dataset
          figs, axes = plt.subplots(4, 4, figsize=(6, 6))
          for i in range(4):
              for j in range(4):
                  axes[i, j].imshow(-X[i + 4 * j].reshape(32, 32), cmap=’gray’, interpolation=’none’)
                  axes[i, j].set_xticks([])
                  axes[i, j].set_yticks([])
                  axes[i, j].set_title("Label: {}".format(y[i + 4 * j]))
                  axes[i, j].axis(’off’)
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-188-97195b50896d> in <module>()
  3 for i in range(4):
  4     for j in range(4):
----> 5         axes[i, j].imshow(-X[i + 4 * j].reshape(32, 32), cmap=’gray’, interpolation=’none’)
  6         axes[i, j].set_xticks([])
  7         axes[i, j].set_yticks([])


ValueError: total size of new array must be unchanged
```



```
In [189]: layers0 = [
              # layer dealing with the input data
              (InputLayer, {'shape': (None, X.shape[1], X.shape[2], X.shape[3])}),

              # first stage of our convolutional layers
              (Conv2DLayer, {'num_filters': 96, 'filter_size': 5}),
              (Conv2DLayer, {'num_filters': 96, 'filter_size': 3}),
              (Conv2DLayer, {'num_filters': 96, 'filter_size': 3}),
              (Conv2DLayer, {'num_filters': 96, 'filter_size': 3}),
              (Conv2DLayer, {'num_filters': 96, 'filter_size': 3}),
              (MaxPool2DLayer, {'pool_size': 2}),

              # second stage of our convolutional layers
              (Conv2DLayer, {'num_filters': 128, 'filter_size': 3}),
              (Conv2DLayer, {'num_filters': 128, 'filter_size': 3}),
              (Conv2DLayer, {'num_filters': 128, 'filter_size': 3}),
              (MaxPool2DLayer, {'pool_size': 2}),

              # two dense layers with dropout
              (DenseLayer, {'num_units': 64}),
              (DropoutLayer, {}),
              (DenseLayer, {’num_units’: 64}),

              # the output layer
              (DenseLayer, {’num_units’: 7, ’nonlinearity’: softmax}),
          ]

          layers1 = [
              # layer dealing with the input data
              (InputLayer, {’shape’: (None, X.shape[1], X.shape[2], X.shape[3])}),

              # first stage of our convolutional layers
              (Conv2DLayer, {’num_filters’: 48, ’filter_size’: 5}),
              (Conv2DLayer, {’num_filters’: 48, ’filter_size’: 3}),
              (Conv2DLayer, {’num_filters’: 48, ’filter_size’: 3}),
              (MaxPool2DLayer, {’pool_size’: 2}),

              # second stage of our convolutional layers
              (Conv2DLayer, {’num_filters’: 64, ’filter_size’: 5}),
              (Conv2DLayer, {’num_filters’: 64, ’filter_size’: 3}),
              (MaxPool2DLayer, {’pool_size’: 2}),

              # two dense layers with dropout
              (DenseLayer, {’num_units’: 32}),
              (DropoutLayer, {}),
              (DenseLayer, {’num_units’: 32}),

              # the output layer
              (DenseLayer, {’num_units’: 7, ’nonlinearity’: softmax}),
          ]

          layers2 = [
              (InputLayer, {’shape’: (None, X.shape[1], X.shape[2], X.shape[3])}),

              (Conv2DLayer, {’num_filters’: 32, ’filter_size’: (3, 3)}),
              (MaxPool2DLayer, {’pool_size’: (2, 2)}),

              (Conv2DLayer, {’num_filters’: 64, ’filter_size’: (3, 3)}),
              (Conv2DLayer, {’num_filters’: 64, ’filter_size’: (3, 3)}),
              (MaxPool2DLayer, {’pool_size’: (2, 2)}),

              (Conv2DLayer, {’num_filters’: 96, ’filter_size’: (3, 3)}),
              (MaxPool2DLayer, {’pool_size’: (2, 2)}),

              (DenseLayer, {’num_units’: 64}),
              (DropoutLayer, {}),
              (DenseLayer, {’num_units’: 64}),

              (DenseLayer, {’num_units’: 7, ’nonlinearity’: softmax}),
          ]

          layers3 = [
              (InputLayer, {’shape’: (None, X.shape[1], X.shape[2], X.shape[3])}),
              (Conv2DLayer, {’num_filters’: 96, ’filter_size’: (5, 5)}),
              (MaxPool2DLayer, {’pool_size’: (2, 2)}),
              (Conv2DLayer, {’num_filters’: 64, ’filter_size’: (5, 5)}),
              (MaxPool2DLayer, {’pool_size’: (2, 2)}),
              (DenseLayer, {’num_units’: 64}),
              (DenseLayer, {’num_units’: 7, ’nonlinearity’: softmax}),
          ]

          layers4 = [
              (InputLayer, {’shape’: (None, X.shape[1], X.shape[2], X.shape[3])}),
              #(Conv2DLayer, {’num_filters’: 96, ’filter_size’: (5, 5)}),
              #(MaxPool2DLayer, {’pool_size’: (2, 2)}),
              (DenseLayer, {’num_units’: 90}),
              (DenseLayer, {’num_units’: 7, ’nonlinearity’: softmax}),
          ]
```

```
In [190]: def regularization_objective(layers, lambda1=0., lambda2=0., *args, **kwargs):
              # default loss
              losses = objective(layers, *args, **kwargs)
              # get the layers’ weights, but only those that should be regularized
              # (i.e. not the biases)
              weights = get_all_params(layers[-1], regularizable=True)
              # sum of absolute weights for L1
              sum_abs_weights = sum([abs(w).sum() for w in weights])
              # sum of squared weights for L2
              sum_squared_weights = sum([(w ** 2).sum() for w in weights])
              # add weights to regular loss
              losses += lambda1 * sum_abs_weights + lambda2 * sum_squared_weights
              return losses
```

```
In [191]: net0 = NeuralNet(
              layers=layers0,
              max_epochs=20,

              update=adam,
              update_learning_rate=0.0002,

              objective=regularization_objective,
              objective_lambda2=0.0025,

              train_split=TrainSplit(eval_size=0.25),
              verbose=4,
          )
          net1 = NeuralNet(
              layers=layers4,
              max_epochs=100,
              update=adam,
              update_learning_rate=0.0001,
              objective=regularization_objective,
              objective_lambda2=0.002,
              train_split=TrainSplit(eval_size=0.005),
              verbose=3,
          )
```

```
In [192]: net1.fit(X, y)
```

```
# Neural Network with 231127 learnable parameters

## Layer information

  #  name    size
---  ------  --------
  0  input0  1x2560x1
  1  dense1  90
  2  dense2  7

  epoch    train loss    valid loss    train/val    valid acc  dur
-------  ------------  ------------  -----------  -----------  -----
      1       2.54987       2.31965      1.09925      0.33333  0.29s
      2       1.98719       1.98705      1.00007      0.38889  0.32s
      3       1.68112       1.84959      0.90892      0.50000  0.30s
      4       1.49158       1.76722      0.84402      0.50000  0.26s
      5       1.37405       1.71639      0.80055      0.55556  0.26s
      6       1.29418       1.65386      0.78252      0.55556  0.27s
      7       1.22849       1.61063      0.76273      0.55556  0.27s
      8       1.17063       1.58512      0.73851      0.55556  0.27s
      9       1.11278       1.55161      0.71718      0.61111  0.25s
     10       1.06248       1.50143      0.70765      0.66667  0.25s
     11       1.02617       1.46533      0.70030      0.66667  0.26s
     12       0.99616       1.45201      0.68606      0.66667  0.28s
     13       0.96675       1.45138      0.66609      0.66667  0.26s
     14       0.93497       1.45026      0.64469      0.66667  0.28s
     15       0.90458       1.44376      0.62655      0.66667  0.26s
     16       0.87694       1.42061      0.61729      0.66667  0.26s
     17       0.85375       1.39171      0.61346      0.66667  0.26s
     18       0.83461       1.36140      0.61305      0.66667  0.26s
     19       0.81777       1.34873      0.60633      0.72222  0.26s
     20       0.80239       1.34948      0.59459      0.72222  0.26s
     21       0.78649       1.34350      0.58541      0.72222  0.26s
     22       0.77115       1.35035      0.57107      0.72222  0.29s
     23       0.75470       1.35405      0.55737      0.72222  0.33s
     24       0.73940       1.35321      0.54640      0.66667  0.29s
     25       0.72426       1.34923      0.53679      0.72222  0.27s
     26       0.71023       1.34199      0.52924      0.72222  0.26s
     27       0.69861       1.32925      0.52556      0.72222  0.26s
     28       0.68784       1.31917      0.52142      0.77778  0.26s
     29       0.67754       1.30962      0.51736      0.77778  0.28s
     30       0.66778       1.30994      0.50978      0.77778  0.26s
     31       0.65762       1.31417      0.50041      0.72222  0.27s
     32       0.64770       1.32188      0.48999      0.72222  0.26s
     33       0.63855       1.32304      0.48264      0.77778  0.26s
     34       0.62958       1.32117      0.47653      0.77778  0.27s
     35       0.62078       1.31987      0.47033      0.72222  0.28s
     36       0.61189       1.31582      0.46502      0.77778  0.32s
     37       0.60358       1.31718      0.45824      0.77778  0.30s
     38       0.59537       1.31575      0.45249      0.77778  0.27s
     39       0.58779       1.30946      0.44888      0.77778  0.31s
     40       0.58097       1.29817      0.44753      0.77778  0.29s
     41       0.57407       1.29932      0.44182      0.77778  0.28s
     42       0.56818       1.29923      0.43732      0.77778  0.27s
```

| | | | | | |
|---|---|---|---|---|---|
| 43 | 0.56227 | 1.30237 | 0.43173 | 0.77778 | 0.28s |
| 44 | 0.55659 | 1.29413 | 0.43009 | 0.77778 | 0.27s |
| 45 | 0.55061 | 1.30487 | 0.42196 | 0.77778 | 0.36s |
| 46 | 0.54444 | 1.31491 | 0.41405 | 0.77778 | 0.47s |
| 47 | 0.53802 | 1.32319 | 0.40661 | 0.77778 | 0.38s |
| 48 | 0.53207 | 1.32608 | 0.40123 | 0.77778 | 0.29s |
| 49 | 0.52585 | 1.31750 | 0.39913 | 0.77778 | 0.29s |
| 50 | 0.52097 | 1.31233 | 0.39698 | 0.72222 | 0.31s |
| 51 | 0.51525 | 1.31774 | 0.39101 | 0.72222 | 0.27s |
| 52 | 0.51036 | 1.31170 | 0.38908 | 0.72222 | 0.26s |
| 53 | 0.50528 | 1.30091 | 0.38841 | 0.72222 | 0.26s |
| 54 | 0.50131 | 1.29803 | 0.38621 | 0.72222 | 0.27s |
| 55 | 0.49707 | 1.30047 | 0.38222 | 0.72222 | 0.27s |
| 56 | 0.49305 | 1.30491 | 0.37784 | 0.72222 | 0.26s |
| 57 | 0.48926 | 1.30371 | 0.37528 | 0.72222 | 0.27s |
| 58 | 0.48573 | 1.30207 | 0.37305 | 0.72222 | 0.28s |
| 59 | 0.48136 | 1.31143 | 0.36705 | 0.72222 | 0.26s |
| 60 | 0.47721 | 1.32565 | 0.35998 | 0.72222 | 0.26s |
| 61 | 0.47286 | 1.33330 | 0.35465 | 0.72222 | 0.27s |
| 62 | 0.46869 | 1.33236 | 0.35178 | 0.72222 | 0.29s |
| 63 | 0.46460 | 1.33348 | 0.34841 | 0.72222 | 0.25s |
| 64 | 0.46035 | 1.33734 | 0.34422 | 0.72222 | 0.25s |
| 65 | 0.45642 | 1.33358 | 0.34225 | 0.72222 | 0.27s |
| 66 | 0.45271 | 1.32799 | 0.34090 | 0.72222 | 0.26s |
| 67 | 0.44915 | 1.32018 | 0.34021 | 0.72222 | 0.26s |
| 68 | 0.44574 | 1.31147 | 0.33988 | 0.72222 | 0.26s |
| 69 | 0.44266 | 1.31206 | 0.33737 | 0.72222 | 0.28s |
| 70 | 0.44037 | 1.30611 | 0.33716 | 0.72222 | 0.26s |
| 71 | 0.43811 | 1.30204 | 0.33648 | 0.72222 | 0.26s |
| 72 | 0.43563 | 1.29472 | 0.33646 | 0.72222 | 0.27s |
| 73 | 0.43321 | 1.30359 | 0.33232 | 0.72222 | 0.37s |
| 74 | 0.43065 | 1.32220 | 0.32571 | 0.72222 | 0.44s |
| 75 | 0.42791 | 1.34024 | 0.31928 | 0.72222 | 0.27s |
| 76 | 0.42459 | 1.34718 | 0.31517 | 0.72222 | 0.26s |
| 77 | 0.42128 | 1.35974 | 0.30982 | 0.72222 | 0.29s |
| 78 | 0.41753 | 1.36796 | 0.30522 | 0.72222 | 0.26s |
| 79 | 0.41407 | 1.36723 | 0.30285 | 0.72222 | 0.26s |
| 80 | 0.41120 | 1.36049 | 0.30225 | 0.72222 | 0.26s |
| 81 | 0.40845 | 1.34545 | 0.30358 | 0.72222 | 0.26s |
| 82 | 0.40583 | 1.33576 | 0.30382 | 0.72222 | 0.28s |
| 83 | 0.40304 | 1.30944 | 0.30780 | 0.72222 | 0.26s |
| 84 | 0.40078 | 1.29022 | 0.31063 | 0.72222 | 0.25s |
| 85 | 0.39902 | 1.27397 | 0.31321 | 0.72222 | 0.29s |
| 86 | 0.39821 | 1.27082 | 0.31335 | 0.72222 | 0.27s |
| 87 | 0.39769 | 1.28061 | 0.31055 | 0.72222 | 0.27s |
| 88 | 0.39713 | 1.29406 | 0.30688 | 0.72222 | 0.27s |
| 89 | 0.39624 | 1.31851 | 0.30052 | 0.72222 | 0.43s |
| 90 | 0.39426 | 1.34113 | 0.29397 | 0.72222 | 0.42s |
| 91 | 0.39128 | 1.37099 | 0.28540 | 0.72222 | 0.33s |
| 92 | 0.38738 | 1.38747 | 0.27920 | 0.72222 | 0.43s |
| 93 | 0.38362 | 1.40467 | 0.27311 | 0.72222 | 0.28s |
| 94 | 0.38063 | 1.41950 | 0.26815 | 0.66667 | 0.26s |
| 95 | 0.37927 | 1.41203 | 0.26860 | 0.72222 | 0.27s |
| 96 | 0.37842 | 1.37411 | 0.27539 | 0.72222 | 0.28s |

```
     97       0.37614       1.32176      0.28458      0.72222  0.29s
     98       0.37249       1.27419      0.29234      0.72222  0.44s
     99       0.36935       1.24197      0.29739      0.72222  0.32s
    100       0.36916       1.23651      0.29855      0.72222  0.26s
```

```
Out[192]: NeuralNet(X_tensor_type=None,
               batch_iterator_test=<nolearn.lasagne.base.BatchIterator object at 0x1077bf150>,
               batch_iterator_train=<nolearn.lasagne.base.BatchIterator object at 0x1077bf0d0>,
               custom_score=None,
               layers=[(<class 'lasagne.layers.input.InputLayer'>, {'shape': (None, 1, 2560, 1)}), (<cla
               loss=None, max_epochs=100, more_params={},
               objective=<function regularization_objective at 0x1301b3d70>,
               objective_lambda2=0.002,
               objective_loss_function=<function categorical_crossentropy at 0x1074af758>,
               on_batch_finished=[],
               on_epoch_finished=[<nolearn.lasagne.handlers.PrintLog instance at 0x1303aa3f8>],
               on_training_finished=[],
               on_training_started=[<nolearn.lasagne.handlers.PrintLayerInfo instance at 0x12f543710>],
               regression=False,
               train_split=<nolearn.lasagne.base.TrainSplit object at 0x1306b4e50>,
               update=<function adam at 0x1074b7b18>, update_learning_rate=0.0001,
               use_label_encoder=False, verbose=3,
               y_tensor_type=TensorType(int32, vector))
```

```
In [193]: net1.save_params_to ('NN_model_with_preprocess')
```

```
In [194]: def classify_pub_test(classifier):
              '''
              test = scipy.io.loadmat('public_test_images.mat')
              print test
              print test["public_test_images"].shape
              (x, y, n_images) = test["public_test_images"].shape
              test_img = np.reshape(np.swapaxes(test["public_test_images"], 0, 2), (n_images, 1, x, y))

              test_img = np.array(test_img).astype(np.float32)
              test_img -= test_img.mean()
              test_img /= test_img.std()
              '''
              test = scipy.io.loadmat('public_test_filtered_no_normalization.mat')
              pub_test = scipy.io.loadmat('./public_test_filtered_no_normalization.mat')
              hid_test = scipy.io.loadmat('./hidden_test_images_filtered.mat')
              (n_images, dim) = pub_test["public_test_images"].shape
              test_img = np.reshape(pub_test['public_test_images'], (n_images, 1, dim, 1))
              test_img = np.array(test_img).astype(np.float32)
              #test_img -= test_img.mean()
              #test_img /= test_img.std()
              pub_res = list(classifier.predict(test_img)+1)

              (n_images, dim) = hid_test["hidden_img"].shape
              test_img = np.reshape(hid_test["hidden_img"], (n_images, 1, dim, 1))
              test_img = np.array(test_img).astype(np.float32)
              #test_img -= test_img.mean()
              #test_img /= test_img.std()
              hid_res = list(classifier.predict(test_img)+1)
              return pub_res+hid_res
```

```
In [195]: classify_result = classify_pub_test(net1)
          cls_res_list = list(classify_result)
          print cls_res_list
          with open('submit_nn_sing_layer_90_units_100iter_non_normalized.csv', 'w') as f:
              f.write('Id,Prediction\n')
              index = 1
              for pred in cls_res_list:
                  f.write('%d,%d\n'%(index, pred))
                  index += 1
              while index<=1253:
                  f.write('%d,0\n'%(index))
                  index+=1
```

```
[7, 5, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 6, 7, 7, 7, 4, 7, 4, 4, 7, 4, 7, 7, 7, 7, 4, 7, 4, 7, 4, 4, 4, 2,
```

```
In [ ]: from nolearn.lasagne.visualize import plot_loss
        from nolearn.lasagne.visualize import plot_conv_weights
        from nolearn.lasagne.visualize import plot_conv_activity
        from nolearn.lasagne.visualize import plot_occlusion
        plot_loss(net1)
        plot_conv_activity(net1.layers_[1], X[0:1])
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```