# NN_with_preprocess

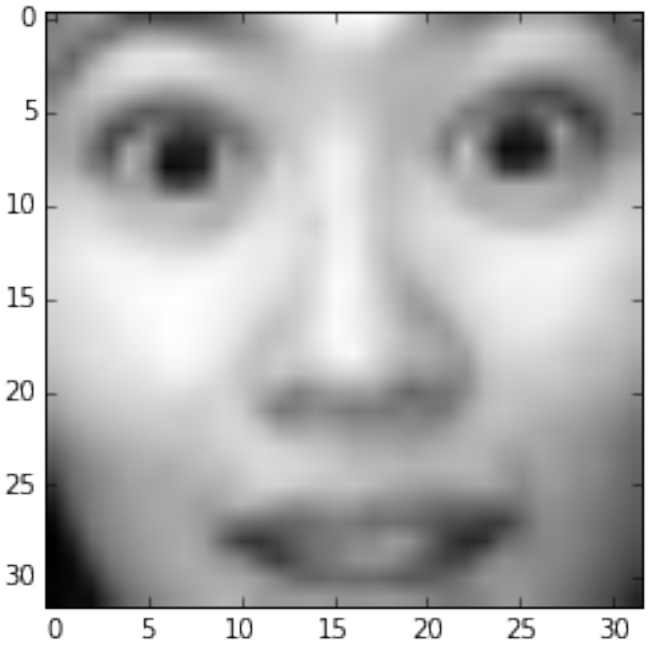
December 11, 2015

In [221]:
```
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
```

In [244]:
```
import scipy.io

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)
# Normalization
#X -= X.mean()
#X /= X.std()

plt.imshow(np.swapaxes((np.swapaxes(train_original["tr_images"], 0, 2)), 1,2)[0], cmap=pylab.g
plt.show()
```

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




```
In [245]: 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 [246]: 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 [253]: 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=50,
            update=adam,
            update_learning_rate=0.0001,
            objective=regularization_objective,
            objective_lambda2=0.009,
            train_split=TrainSplit(eval_size=0.25),
            verbose=3,
        )
```

```
In [254]: 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 | 3.82806 | 3.43833 | 1.11335 | 0.39198 | 0.22s |
| 2 | 3.32719 | 3.16184 | 1.05230 | 0.44757 | 0.21s |
| 3 | 2.95896 | 3.02766 | 0.97731 | 0.50922 | 0.21s |
| 4 | 2.80678 | 2.94084 | 0.95441 | 0.54715 | 0.20s |
| 5 | 2.68662 | 2.88034 | 0.93274 | 0.57194 | 0.21s |
| 6 | 2.57092 | 2.83540 | 0.90672 | 0.57845 | 0.20s |
| 7 | 2.47766 | 2.79376 | 0.88685 | 0.59068 | 0.20s |
| 8 | 2.40327 | 2.75142 | 0.87346 | 0.60676 | 0.20s |
| 9 | 2.33777 | 2.72522 | 0.85783 | 0.60676 | 0.20s |
| 10 | 2.27657 | 2.69274 | 0.84545 | 0.60981 | 0.20s |
| 11 | 2.22343 | 2.66558 | 0.83413 | 0.61332 | 0.20s |
| 12 | 2.17331 | 2.63177 | 0.82580 | 0.62074 | 0.20s |
| 13 | 2.13015 | 2.60986 | 0.81620 | 0.62294 | 0.20s |
| 14 | 2.08821 | 2.58349 | 0.80829 | 0.63472 | 0.20s |
| 15 | 2.04784 | 2.56197 | 0.79932 | 0.63342 | 0.21s |
| 16 | 2.01216 | 2.53521 | 0.79369 | 0.64474 | 0.21s |
| 17 | 1.97773 | 2.51642 | 0.78593 | 0.64213 | 0.20s |
| 18 | 1.94520 | 2.49504 | 0.77963 | 0.64519 | 0.19s |
| 19 | 1.91394 | 2.47481 | 0.77337 | 0.64825 | 0.20s |
| 20 | 1.88388 | 2.45191 | 0.76833 | 0.65345 | 0.20s |
| 21 | 1.85673 | 2.43207 | 0.76344 | 0.66087 | 0.21s |
| 22 | 1.82998 | 2.41927 | 0.75642 | 0.65911 | 0.21s |
| 23 | 1.80263 | 2.39953 | 0.75124 | 0.66393 | 0.20s |
| 24 | 1.77815 | 2.38280 | 0.74625 | 0.66002 | 0.22s |
| 25 | 1.75281 | 2.36685 | 0.74057 | 0.66568 | 0.20s |
| 26 | 1.73016 | 2.34888 | 0.73659 | 0.66653 | 0.21s |
| 27 | 1.70754 | 2.33762 | 0.73046 | 0.66698 | 0.31s |
| 28 | 1.68457 | 2.32141 | 0.72567 | 0.66478 | 0.22s |
| 29 | 1.66362 | 2.30642 | 0.72130 | 0.67259 | 0.20s |
| 30 | 1.64336 | 2.29039 | 0.71750 | 0.67389 | 0.22s |
| 31 | 1.62443 | 2.28123 | 0.71209 | 0.67259 | 0.22s |
| 32 | 1.60414 | 2.26628 | 0.70783 | 0.67259 | 0.24s |
| 33 | 1.58505 | 2.25403 | 0.70321 | 0.67389 | 0.30s |
| 34 | 1.56690 | 2.23681 | 0.70051 | 0.68130 | 0.31s |
| 35 | 1.55103 | 2.22709 | 0.69644 | 0.67825 | 0.22s |
| 36 | 1.53289 | 2.21418 | 0.69231 | 0.68261 | 0.21s |
| 37 | 1.51545 | 2.20517 | 0.68722 | 0.67695 | 0.22s |
| 38 | 1.49841 | 2.18691 | 0.68517 | 0.68215 | 0.27s |
| 39 | 1.48423 | 2.17666 | 0.68188 | 0.68436 | 0.32s |
| 40 | 1.46786 | 2.16570 | 0.67778 | 0.68436 | 0.30s |
| 41 | 1.45190 | 2.15946 | 0.67234 | 0.68612 | 0.21s |
| 42 | 1.43553 | 2.14364 | 0.66967 | 0.68917 | 0.20s |
| 43 | 1.42174 | 2.13122 | 0.66710 | 0.68872 | 0.21s |
| 44 | 1.40805 | 2.12001 | 0.66417 | 0.69178 | 0.22s |
| 45 | 1.39390 | 2.11612 | 0.65871 | 0.68436 | 0.20s |
| 46 | 1.37806 | 2.10391 | 0.65500 | 0.68697 | 0.20s |
| 47 | 1.36520 | 2.09071 | 0.65298 | 0.69087 | 0.20s |
| 48 | 1.35196 | 2.07819 | 0.65055 | 0.69132 | 0.20s |
| 49 | 1.33972 | 2.07609 | 0.64531 | 0.68957 | 0.19s |
| 50 | 1.32508 | 2.06473 | 0.64177 | 0.68827 | 0.21s |

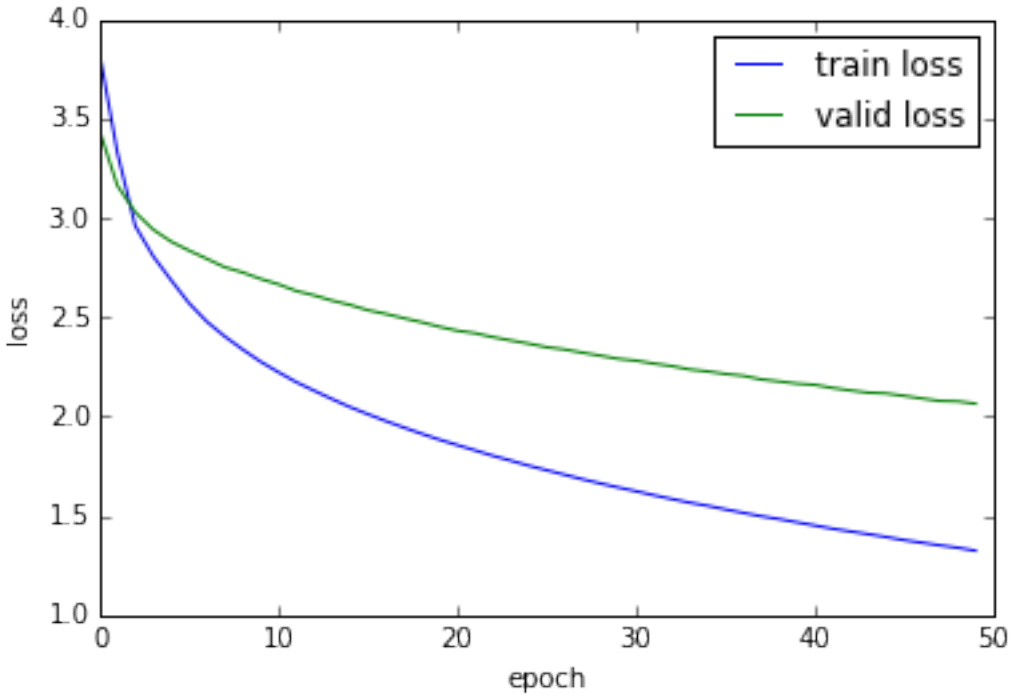
```
Out[254]: NeuralNet(X_tensor_type=None,
                batch_iterator_test=<nolearn.lasagne.base.BatchIterator object at 0x1077bf150>,
                batch_iterator_train=<nolearn.lasagne.base.BatchIterator object at 0x1077bf0d0>,
                custom_score=None,
                layers=[(<class 'lasagne.layers.input.InputLayer'>, {'shape': (None, 1, 2560, 1)}), (<cla
                loss=None, max_epochs=50, more_params={},
                objective=<function regularization_objective at 0x1279a7050>,
                objective_lambda2=0.009,
                objective_loss_function=<function categorical_crossentropy at 0x1074af758>,
                on_batch_finished=[],
                on_epoch_finished=[<nolearn.lasagne.handlers.PrintLog instance at 0x14bc29830>],
                on_training_finished=[],
                on_training_started=[<nolearn.lasagne.handlers.PrintLayerInfo instance at 0x149d723f8>],
                regression=False,
                train_split=<nolearn.lasagne.base.TrainSplit object at 0x14a8b4cd0>,
                update=<function adam at 0x1074b7b18>, update_learning_rate=0.0001,
                use_label_encoder=False, verbose=3,
                y_tensor_type=TensorType(int32, vector))

In [255]: net1.save_params_to ('NN_model_with_preprocess')

In [256]: 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 [257]: 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, 7, 4, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 4, 7, 1, 4, 7, 4, 7, 7, 7, 7, 4, 7, 4, 7, 4, 4, 4, 2, 5
```

In [258]:
```
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])
```

Out[258]: <module 'matplotlib.pyplot' from '/Users/zexuanwang/anaconda/lib/python2.7/site-packages/matpl




In [ ]:

In [ ]:

In [ ]:

In [ ]: