In [1]:

```
import deepthought, os
import numpy as np
from deepthought.util.logging_util import configure_custom
configure_custom(debug=False)

import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt

data_root = os.path.join(deepthought.OUTPUT_PATH, 'iclr2016')
data_specs = None
```

In [2]:

```
from deepthought.datasets.eeg import EEGEpochsDataset, DataFile
from deepthought.datasets.datasources import SubDatasource
db1 = DataFile(os.path.join(deepthought.DATA_PATH, 'OpenMIIR', 'pylearn2', 'i
clr2016', 'all-cond12-nomastoids-varylen-nocue-64hz.pkl'))
db1 = SubDatasource(db=db1, selectors = dict(subject='all', condition=[1]))

# db2 = DataFile(os.path.join(data_root, 'common_components','4x1_cond[1]','d
ataset-transformed.pkl'))
# # db = DataFile(os.path.join(data_root, 'common_components','8x2_cond[1]','
dataset-transformed.pkl'))
# db2 = SubDatasource(db=db2, selectors = dict(subject='all', condition=[1]))
```

```
loading data from /Users/sstober/work/datasets/OpenMIIR/pylearn2/iclr2016/all
-cond12-nomastoids-varylen-nocue-64hz.pkl...
loading data from /Users/sstober/work/datasets/OpenMIIR/pylearn2/iclr2016/all
-cond12-nomastoids-varylen-nocue-64hz.pkl done. Time elapsed: 45.332694 secon
ds
```

In [3]:
```
base_dataset1 = EEGEpochsDataset(name='base_dataset',
                            db=db1,
                            use_targets=False,
                            stop_sample=440,
                            label_attribute='stimulus_id',

                            label_map={ # stimulus_id
                                     1: 0, 2: 1, 3: 2, 4: 3, 11: 4, 12: 5, 13:
 6, 14: 7, 21: 8, 22: 9, 23: 10, 24: 11,
                            },
                            remove_dc_offset=True,
                            layout='ft')
# base_dataset2 = EEGEpochsDataset.Like(base=base_dataset1, db=db2)
```

```
selectors: {}
selected trials: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
7, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36
, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 9
4, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 1
26, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141
, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 17
2, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187,
 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 2
03, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218
, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 24
9, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264,
 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 2
80, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295
, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310,
311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 32
6, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341,
 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 3
57, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372
, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387,
388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 40
3, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418,
 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 4
34, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449
, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464,
465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 48
0, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495,
 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 5
11, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526
, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539]
Data will be normalized to max amplitude 1 per channel (normalize=True).
generated dataset "base_dataset" with shape X=(540, 28160)=(540, 1, 440, 64)
y=(540, 12) targets=(540, 12)
```

In [ ]:

```
def compute_2class_performance(base_dataset, selectors, output_fn, class_attr
ibute, classes=None):

    if classes is None:
        classes = np.arange(base_dataset.targets.shape[-1])
    n_classes = len(classes)

    alpha_values = np.zeros((n_classes, n_classes))
    alpha_values += 1

    alpha_treshold = 0.05

    accuracies = np.zeros((n_classes,n_classes))
    confusion = dict()

    for i, c1 in enumerate(classes):
        for j, c2 in enumerate(classes):
            if j <= i: continue
#             print i,j

            sel = selectors.copy()
            sel[class_attribute] = [c1,c2]

            dataset = EEGEpochsDataset.Like(base=base_dataset, selectors=sel)

            # compute output
#             y_pred = output_fn(dataset.trials)
            y_pred = process_dataset(output_fn, dataset)

            # reduce output to the 2 selected classes
            y_pred = y_pred[:, [i,j]]
            y_real = dataset.targets[:, [i,j]]

            # determine class labels
            y_pred = np.argmax(y_pred, axis=1)
            y_real = np.argmax(y_real, axis=1)

            n_correct = sum(y_pred == y_real)
            n_total = len(y_real)

            accuracies[i,j] = accuracies[j,i] = 100. * float(n_correct) / n_t
otal
            confusion[(i,j)] = confusion[(j,i)] = confusion_matrix(y_real, y_
pred)
            p = binom.cdf(n_correct, n=n_total, p=0.5) # NOTE: assumes equal
class distribution

            alpha_values[i,j] = alpha_values[j,i] = 1-p
    return accuracies, alpha_values, confusion


def _plot_2class_confusion(base_dataset, selectors, output_fn,
                          class_attribute, classes=None, class_labels=None,
                          figsize=(10,10), show=True, plot_pvalues=True):

    two_class_acc, two_class_alpha, two_class_conf = compute_2class_performan
ce(
                    base_dataset, selectors, output_fn, class_attribute, clas
ses)

    return __plot_2class_confusion(two_class_acc, two_class_alpha, two_class_
conf,
                          class_attribute, classes, class_labels,
                          figsize, show, plot_pvalues)

def __plot_2class_confusion(two_class_acc, two_class_alpha, two_class_conf,
                          class_attribute, classes=None, class_labels=None,
                          figsize=(10,10), show=True, plot_pvalues=True):

    import matplotlib.patheffects as PathEffects
    from sklearn.metrics import confusion_matrix
    import matplotlib.gridspec as gridspec
    from deepthought.datasets.eeg import EEGEpochsDataset

    if classes is None:
        classes = np.arange(base_dataset.targets.shape[-1])
    n_classes = len(classes)

    if class_labels is None:
        class_labels = classes

    fig = plt.figure(figsize=figsize)
    subplot_grid = gridspec.GridSpec(n_classes-1,n_classes-1)
```

In [78]:

```
## this turned out to be not very useful
def predict_binary_performance(fold_output_fns, class_attribute='stimulus_id'
, classes=STIMULUS_IDS):
    from deepthought.experiments.spearmint_cv_wrapper import generate_folds
    FOLDS = generate_folds(['P01','P04','P06','P07','P09','P11','P12','P13','
P14'])

#     print fold_output_fns
    agg_alpha = None
    agg_acc = None
    agg_conf = None
    for fold, fold_fn in zip(FOLDS, fold_output_fns):
        selectors = dict(trial_no=TRAIN_VALID, subject=fold['valid'], conditi
on=[1])
        print selectors

        two_class_acc, two_class_alpha, two_class_conf = compute_2class_perfo
rmance(
                    base_dataset, selectors, fold_fn, class_attribute, classe
s)
        if agg_alpha is None:
            agg_alpha = two_class_alpha
            agg_acc = two_class_acc
            agg_conf = two_class_conf
        else:
            agg_alpha += two_class_alpha
            agg_acc += two_class_acc
            for k in agg_conf.keys():
                agg_conf[k] += two_class_conf[k]

    agg_alpha /= len(FOLDS)
    agg_acc /= len(FOLDS)

    __plot_2class_confusion(agg_acc, agg_alpha, agg_conf,
                          class_attribute, classes, class_labels=None,
                          figsize=(10,10), show=True, plot_pvalues=True)
```

In [82]:

```
# predict_binary_performance(fold_output_fns)
```

In [83]:

```
# selectors = dict(trial_no=TEST, condition=[1])
# _plot_2class_confusion(base_dataset, selectors, fold_output_fns[4], 'stimul
us_id', STIMULUS_IDS)
```

In [84]:

```
# selectors = dict(trial_no=TRAIN_VALID, subject='P09', condition=[1])
# _plot_2class_confusion(base_dataset, selectors,
#                         fold_output_fns[4],
#                            class_attribute='stimulus_id', classes=STIMULUS_I
DS, class_labels=None,
#                            figsize=(10,10), show=True, plot_pvalues=True)
```

In [85]:

```
# selectors = dict(trial_no=TEST, subject='P09', condition=[1])
# _plot_2class_confusion(base_dataset, selectors, fold_output_fns[4],
#                            class_attribute='stimulus_id', classes=STIMULUS_I
DS, class_labels=None,
#                            figsize=(10,10), show=True, plot_pvalues=True)
```

In [4]:

```
# import and define required functions
from deepthought.analysis.ipynb import plot_model
from deepthought.analysis.ipynb import plot_model_comparison
# from deepthought.analysis.plot.classification import plot_2class_confusion
as _plot_2class_confusion

def plot_2class_confusion(output_fn, selectors, filename=None):
    fig, grid = _plot_2class_confusion(base_dataset, selectors, output_fn,
                                       class_attribute='stimulus_id', classes
=STIMULUS_IDS,
                                       figsize=(16,16), show=False)
    axes = plt.subplot(grid[3,0])
    analyze_performance(output_fn, selectors, transform=meter, labels=METER_L
ABELS, title='meter', axes=axes)

    if filename is not None:
        fig.savefig(filename, bbox_inches='tight', dpi=300)


from sklearn.metrics import confusion_matrix, classification_report
from deepthought.datasets.openmiir.constants import STIMULUS_IDS

# blocks
TRAIN = [0,3,4]
VALID = [1]
TRAIN_VALID = [0,1,3,4]
TEST = [2]

METER_LABELS = ['3/4', '4/4']

def analyze_performance(output_fn, selectors, transform=None, labels=STIMULUS
_IDS, verbose=False, title=None,
                        axes=None, show=True, filename=None):

    dataset = EEGEpochsDataset.Like(
                    base=base_dataset,
                    name='test',
                    selectors=selectors,
                    )

#     y_pred = output_fn(dataset.trials)
    y_pred = process_dataset(output_fn, dataset)

#     for l in y_pred:
#         v = sorted(l)[::-1]
#         print np.asarray(100000*(v - v[0]), int)[1]
#     print len(y_pred)

#     def target_convert(Y):
#         '''
#         converts target [0,1] to [-1, 1]
#         '''
#         Y_t = 2. * Y - 1.
#         return Y_t
#     y_hat = target_convert(np.argmax(y_pred, axis=1))
#     #Assume target is in [0,1] as binary one-hot
#     y = target_convert(np.argmax(dataset.targets, axis=1))
#     misclass = (y != y_hat).mean()
#     print misclass

#     for v in y_pred:
#         w = np.where(v == np.max(v))[0]
#         if len(w) > 1: print w

    y_pred = np.argmax(y_pred, axis=1)
#     print y_pred

    y_real = np.argmax(dataset.targets, axis=1)
#     print y_real

#     print np.sum(dataset.targets, axis=0)
#     print np.sum(dataset.targets, axis=1)

    if transform is not None:
        y_pred = transform(y_pred)
        y_real = transform(y_real)

    accuracy = 100. * float(sum(y_pred == y_real)) / len(y_real)
    confusion = confusion_matrix(y_real, y_pred)

    if verbose:
        print 'confusion:'
        print confusion
```

In [16]:

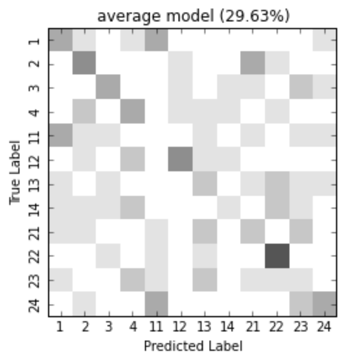
```
# %debug
def make_majority_vote_fn(output_fns, raw=False):
    def majority_vote(*data):
        pred = []
        output = None
        for output_fn in output_fns:
            y = output_fn(*data)       # raw output

            if output is None:
                output = np.zeros_like(y)

            if raw:
                output += y
            else:
                y = np.argmax(y, axis=1)      # reduce to max class

                for i in range(len(output)):  # set class values
                    output[i, y[i]] += 1

#         output = np.argmax(output, axis=1) # done later

#         print pred
#        print output.shape
        return output
    return majority_vote

# mv_out_fn = make_majority_vote_fn(fold_models.values(), raw=True)
# print np.argmax(mv_out_fn(test_dataset.trials), axis=1)

from pylearn2.utils import serial
import theano
from deepthought.pylearn2ext.aggregate_models import average_models

def process_dataset(output_fn, dataset):
    global data_specs
    it = dataset.iterator(mode='sequential',
                          batch_size=min(128, dataset.get_num_examples()),
                          data_specs=data_specs)
    output = []
    for minibatch in it:
        if type(minibatch) is tuple:
            print minibatch[0].shape, minibatch[1].shape
            output.append(output_fn(*minibatch))
        else:
            output.append(output_fn(minibatch))
    output = np.vstack(output)
#     print output.shape
    return output

def load(fold_model_path, final_model_path=None):

    fold_models = []
    for fold in range(10): # consider at most 10 folds
        model_path = fold_model_path.format(fold)
        if not os.path.exists(model_path): # auto-detect number of folds
            break
        print model_path
        fold_models.append(serial.load(model_path))

    # update data specs
    global data_specs
    input_space = fold_models[0].get_input_space()
    input_source = fold_models[0].get_input_source()
#     print input_space
#     print input_source
    if type(input_source) is list:
        input_source = tuple(input_source)      # FIX: use tuple() to ensure
source format
#     print input_source
    data_specs = (input_space, input_source)
    print 'data specs:', data_specs

    # select data source based on data specs
    global base_dataset
#     if type(input_source) is tuple or 'baseline' in fold_model_path:
#         base_dataset = base_dataset1
#     else:
#         base_dataset = base_dataset2
    print 'db data specs:', base_dataset.get_data_specs()

    # prepare symbolic inputs
    minibatch = input_space.make_theano_batch()
#     print minibatch, type(minibatch)
```

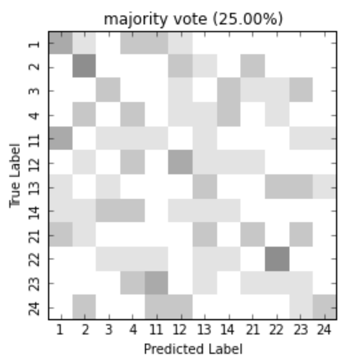
In [8]:
```
# switch off logging
import logging
top_level_logger = logging.getLogger('deepthought')
top_level_logger.setLevel(logging.WARN)
```

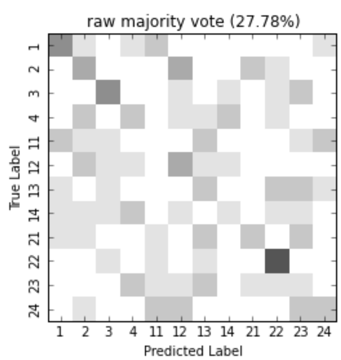
In [70]:
```
# experiment_path = '20034' # avg=28.7, very simple model ****
experiment_path = '20072' # avg=29.63 L0=1x3 l1=3*9 *****

final_model_path = os.path.join(model_base, experiment_path, 'final', 'model.
pkl')
fold_model_path = os.path.join(model_base, experiment_path, '{}', 'model.vali
d-best.pkl')

fold_models, fold_output_fns, avg_model, avg_output_fn, majvote_output_fn, ra
w_majvote_output_fn, fin_model, fin_output_fn = load(fold_model_path, final_m
odel_path)
```

```
./20072/0/model.valid-best.pkl
./20072/1/model.valid-best.pkl
./20072/2/model.valid-best.pkl
./20072/3/model.valid-best.pkl
./20072/4/model.valid-best.pkl
./20072/5/model.valid-best.pkl
./20072/6/model.valid-best.pkl
./20072/7/model.valid-best.pkl
./20072/8/model.valid-best.pkl
data specs: (Conv2DSpace(shape=(1, 440), num_channels=64, axes=('b', 0, 1, 'c
'), dtype=float32), 'features')
db data specs: (CompositeSpace(Conv2DSpace(shape=(1, 440), num_channels=64, a
xes=('b', 0, 1, 'c'), dtype=float32), VectorSpace(dim=12, dtype=float32)), ('
features', 'targets'))
./20072/final/model.pkl
```

In [87]:
```
overview([avg_output_fn, majvote_output_fn, raw_majvote_output_fn, fin_output
_fn],
         ['average model', 'majority vote', 'raw majority vote']) #, 'final m
odel'])
# overview([fin_output_fn], ['final model'])
```



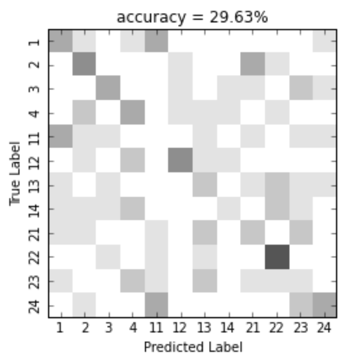




average model

In [89]:
```
analyze_performance(avg_output_fn, {'trial_no':TEST}, verbose=True)
```

```
confusion:
[[3 1 0 1 3 0 0 0 0 0 0 1]
 [0 4 0 0 0 1 0 0 3 1 0 0]
 [0 0 3 0 0 1 0 1 1 0 2 1]
 [0 2 0 3 0 1 1 1 0 1 0 0]
 [3 1 1 0 0 0 1 0 1 0 1 1]
 [0 1 0 2 0 4 1 1 0 0 0 0]
 [1 0 1 0 0 0 2 0 1 2 1 1]
 [1 1 1 2 0 0 0 1 0 2 1 0]
 [1 1 0 0 1 0 2 0 2 0 2 0]
 [0 0 1 0 1 0 1 0 0 6 0 0]
 [1 0 0 2 1 0 2 0 1 1 1 0]
 [0 1 0 0 3 0 0 0 0 0 2 3]]
```




```
classification report:
             precision    recall  f1-score   support

          0       0.30      0.33      0.32         9
          1       0.33      0.44      0.38         9
          2       0.43      0.33      0.38         9
          3       0.30      0.33      0.32         9
          4       0.00      0.00      0.00         9
          5       0.57      0.44      0.50         9
          6       0.20      0.22      0.21         9
          7       0.25      0.11      0.15         9
          8       0.22      0.22      0.22         9
          9       0.46      0.67      0.55         9
         10       0.10      0.11      0.11         9
         11       0.43      0.33      0.38         9

avg / total       0.30      0.30      0.29       108

accuracy: 29.6296296296
```

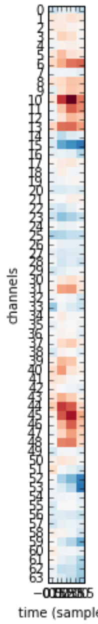
layer-by-layer comparision of the k fold models

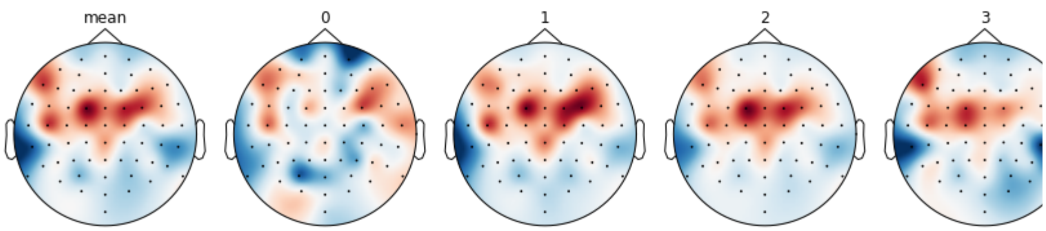
In [223]:
```
# plot_model_comparison([fold_models[i] for i in range(4)])
```

plot the average model (mean of the k fold models)

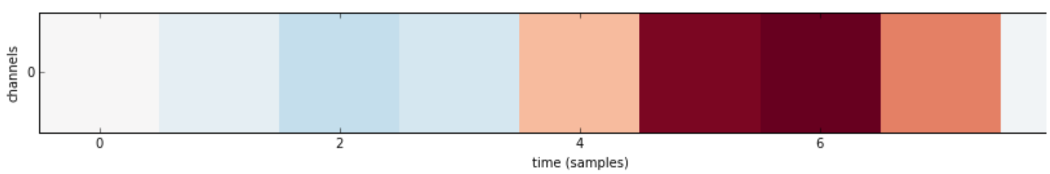
In [81]:
```
plot_model(avg_model, file_prefix='model_20072')
```

```
number of layers: 3
layer l0: 1 x [1x4]x64
layer l0, filter #0
```

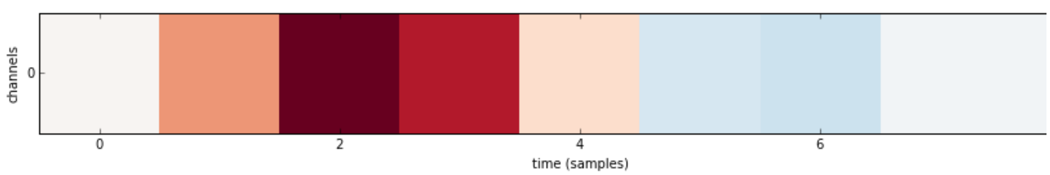




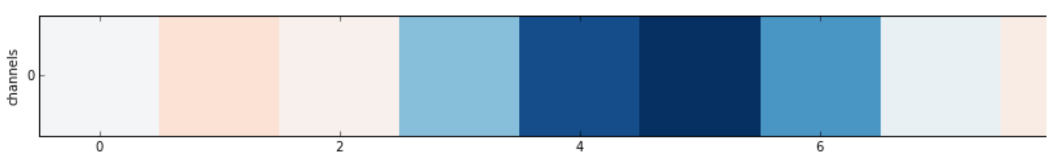

```
layer l1: 3 x [1x9]x1
layer l1, filter #0
```



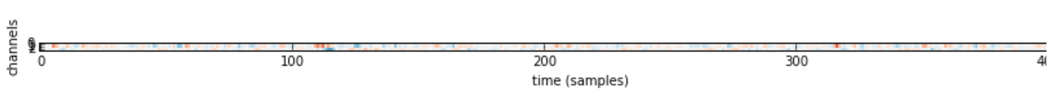

```
layer l1, filter #1
```



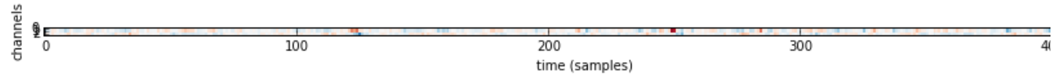

```
layer l1, filter #2
```



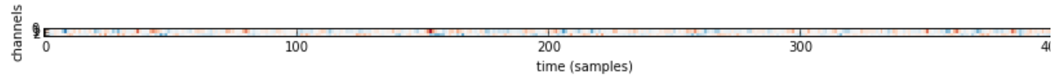

```
layer y: 12 x [1x429]x3
layer y, filter #0
```



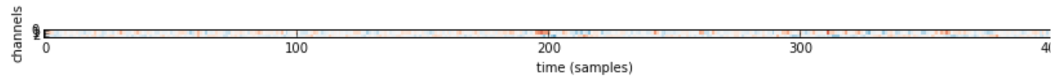
```
layer y, filter #1
```



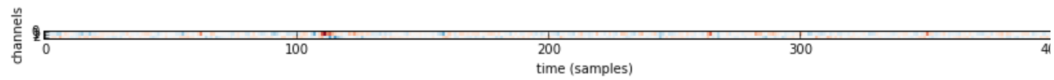

```
layer y, filter #2
```



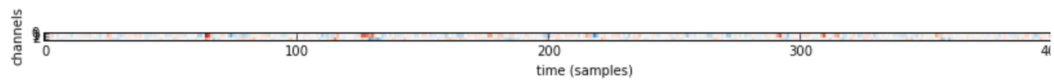

```
layer y, filter #3
```



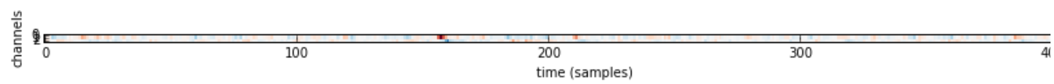

```
layer y, filter #4
```



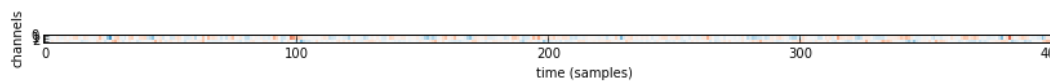

```
layer y, filter #5
```



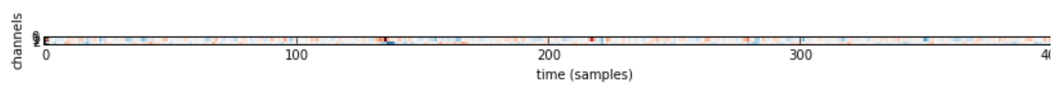

```
layer y, filter #6
```



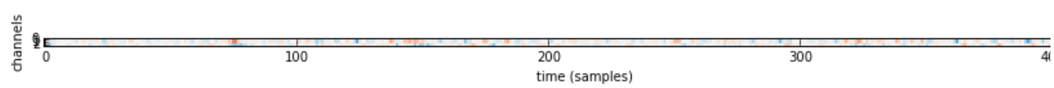

```
layer y, filter #7
```



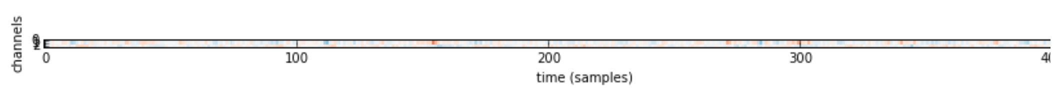

```
layer y, filter #8
```



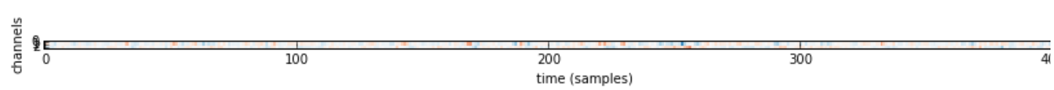

```
layer y, filter #9
```




```
layer y, filter #10
```



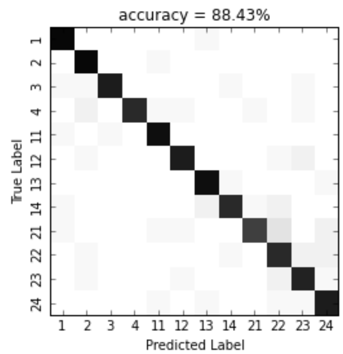

```
layer y, filter #11
```

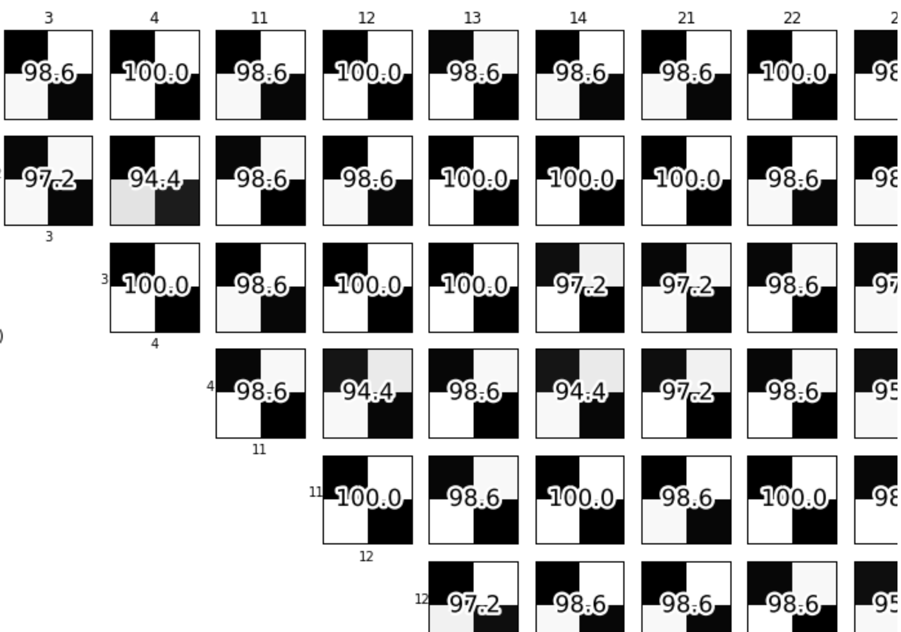


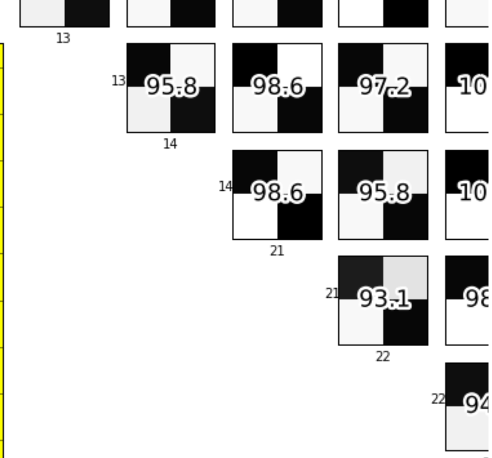
performance analysis of the average model

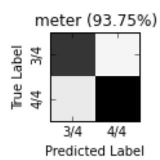
In [90]:
```
# performance on training data
# def analyze(output_fn, selectors, trial_no=TEST, filenames=[None,None]):
analyze(avg_output_fn, {}, trial_no=TRAIN_VALID)
```

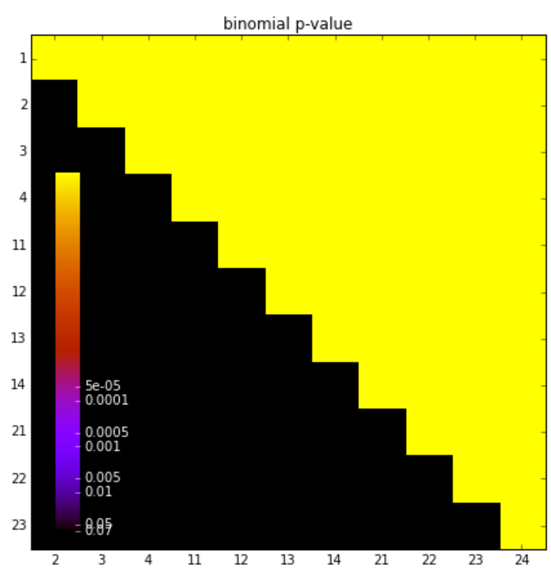
```
{'trial_no': [0, 1, 3, 4]}
```





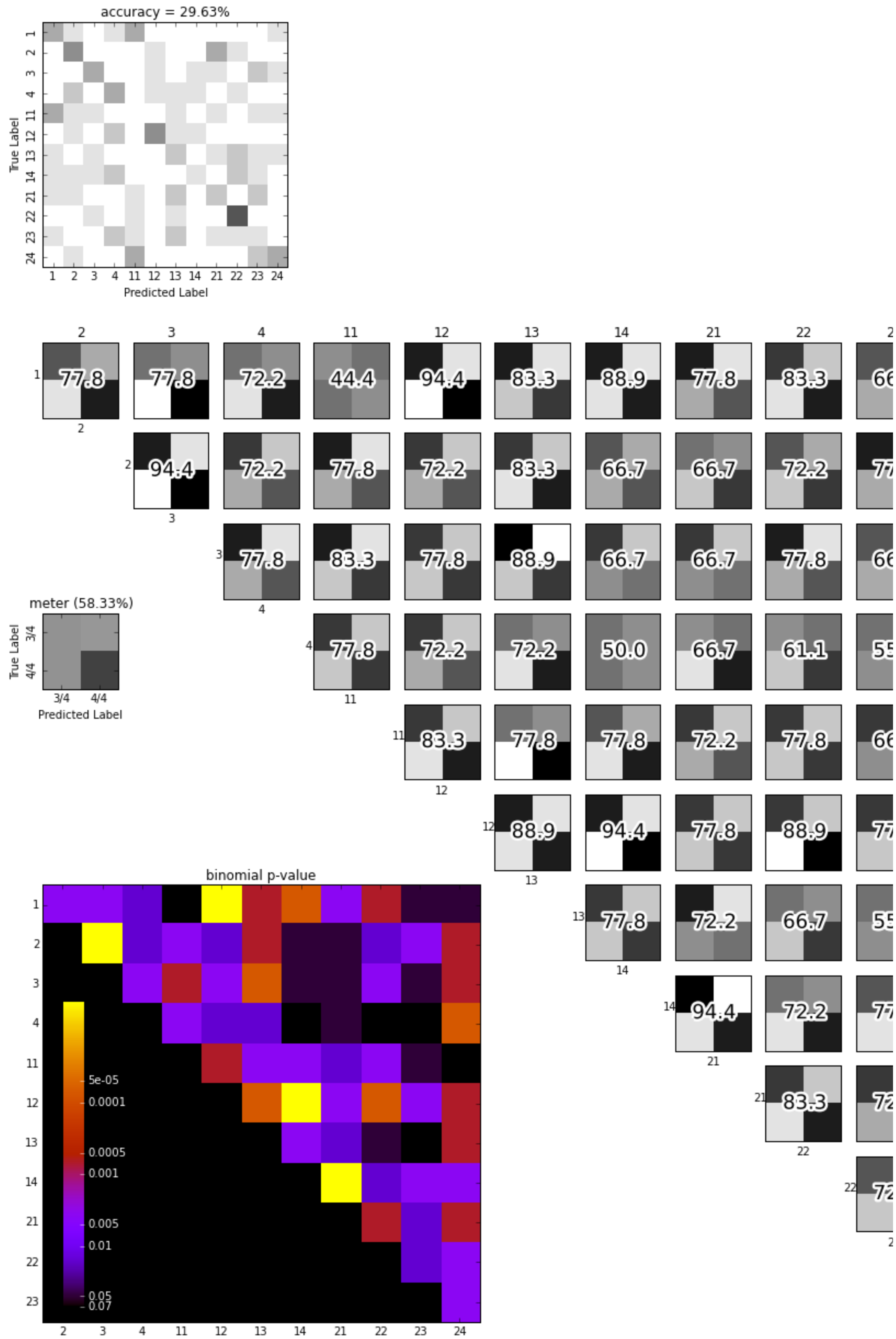






In [91]:
```
# overall performance to start with
analyze(avg_output_fn, {}) #, filenames=['confusion_overall.pdf', 'confusion_
overall_2class.pdf'])
print_binomial_p_levels(108,12)
print_binomial_p_levels(108,2)
print_binomial_p_levels(18,2)
```

```
{'trial_no': [2]}
```



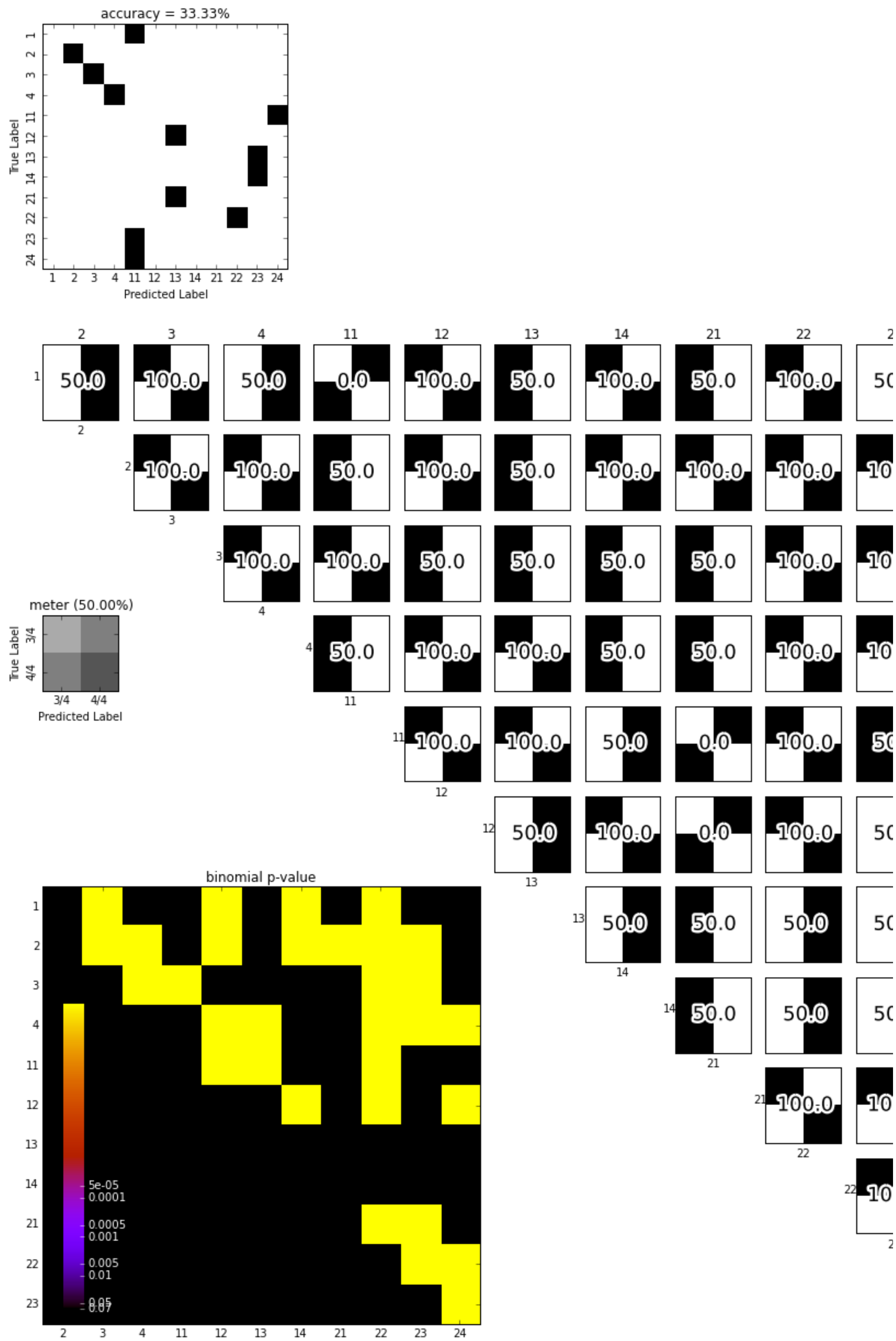
accuracy = 29.63%
True Label
Predicted Label
meter (58.33%)
True Label
Predicted Label
binomial p-value

```
binomial p levels for 12 classes with 108 trials total:
p=0.050 level: 14 = 12.96%
p=0.010 level: 16 = 14.81%
p=0.005 level: 17 = 15.74%
p=0.001 level: 19 = 17.59%
binomial p levels for 2 classes with 108 trials total:
p=0.050 level: 63 = 58.33%
p=0.010 level: 66 = 61.11%
p=0.005 level: 67 = 62.04%
p=0.001 level: 70 = 64.81%
binomial p levels for 2 classes with 18 trials total:
p=0.050 level: 12 = 66.67%
p=0.010 level: 14 = 77.78%
p=0.005 level: 14 = 77.78%
p=0.001 level: 15 = 83.33%
```

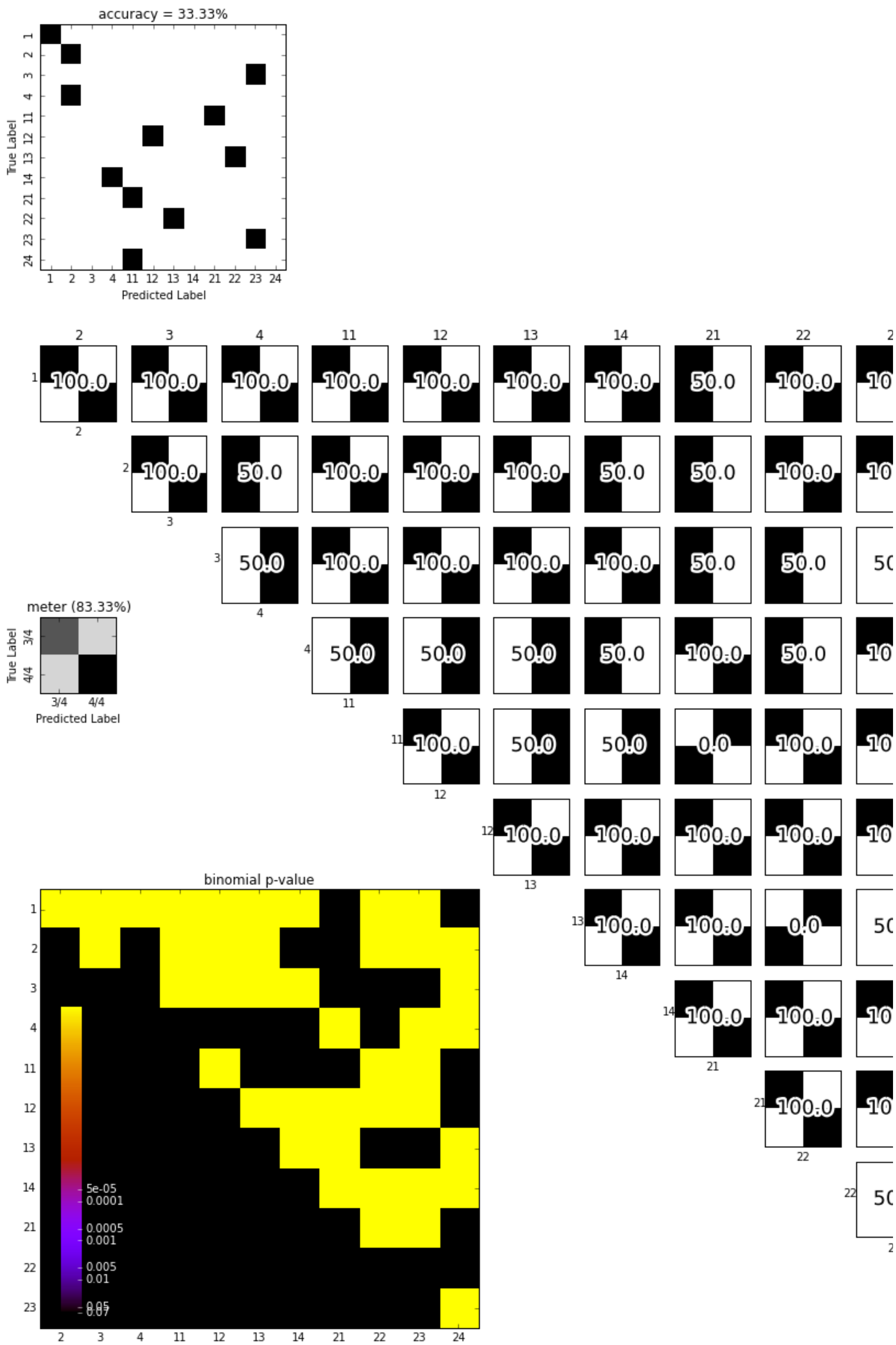
In [92]:

```
analyze(avg_output_fn, {'subject':'P01'})
analyze(avg_output_fn, {'subject':'P09'})
```

```
{'trial_no': [2], 'subject': 'P01'}
```

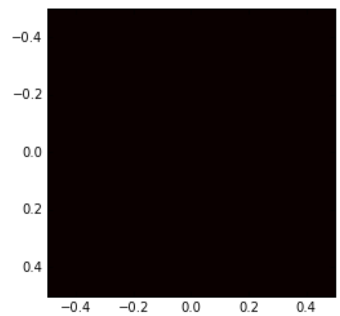



```
{'trial_no': [2], 'subject': 'P09'}
```

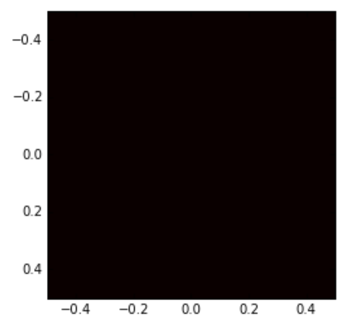

accuracy = 33.33%
True Label
Predicted Label
meter (83.33%)
True Label
Predicted Label
binomial p-value

In [93]:
```
# try to aggregate models
def analyze_filter_similarity(models):

    import numpy as np

    n_models = len(models)
    n_layers = len(models[0].layers)
    print n_models, 'models'

    for l in range(n_layers):
        params = []
        for m in range(n_models):
            params.append(models[m].layers[l].get_param_values()) # list

        if len(params[0]) == 0:
            print 'no params in layer', l
            continue

        n_filters = params[0][0].shape[0]
        n_params = len(params[0])

        print 'layer #{} "{}"'.format(l, models[0].layers[l].layer_name)
        print n_params, 'params'
        print n_filters, 'filters'

#         if n_filters > 1 TODO

        for m2 in range(1, n_models):
            dist = np.zeros((n_filters, n_filters))
            for i in range(n_filters):
                fi = np.hstack(params[0][p][i].flatten() for p in range(n_par
ams))
                for j in range(n_filters):
                    fj = np.hstack(params[m2][p][j].flatten() for p in range(
n_params))
                    dist[i,j] = np.sum((fi - fj)**2)

#             print dist
            print 'model #0 vs #{}'.format(m2)
            plt.imshow(dist, interpolation='nearest', cmap='hot')
            plt.show()

analyze_filter_similarity(fold_models)
```

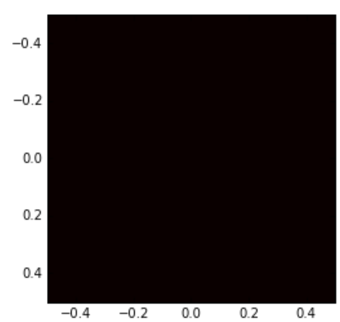
9 models
layer #0 "l0"
1 params
1 filters
model #0 vs #1



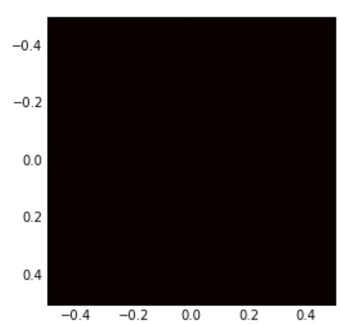
model #0 vs #2



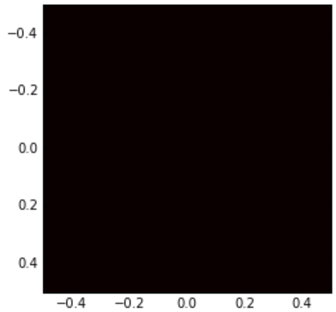
model #0 vs #3



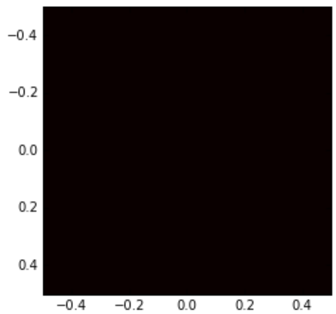
model #0 vs #4



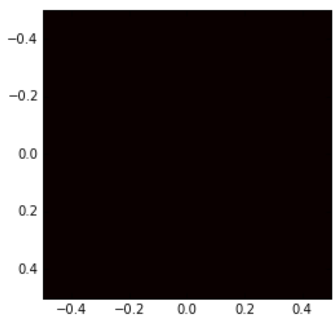
model #0 vs #5



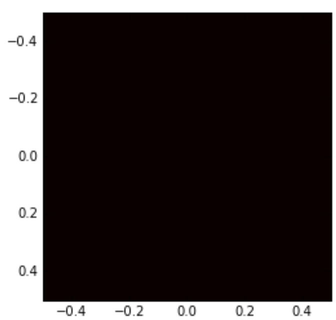

```
model #0 vs #6
```



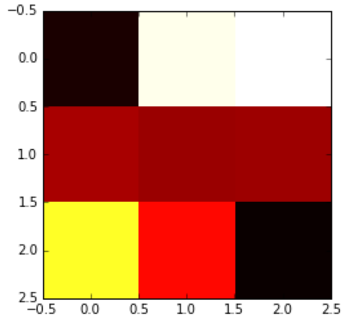

```
model #0 vs #7
```



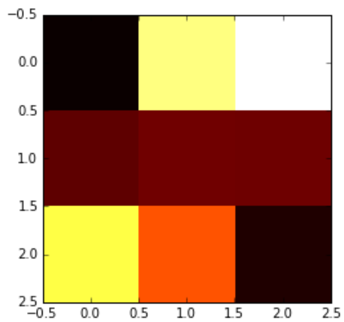

```
model #0 vs #8
```



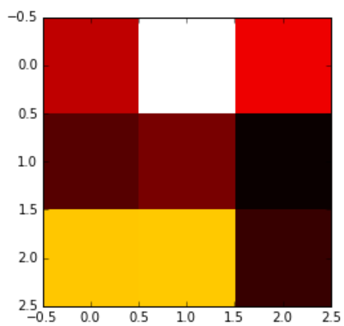

```
layer #1 "l1"
1 params
3 filters
model #0 vs #1
```



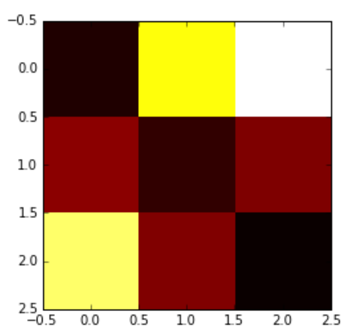
model #0 vs #2



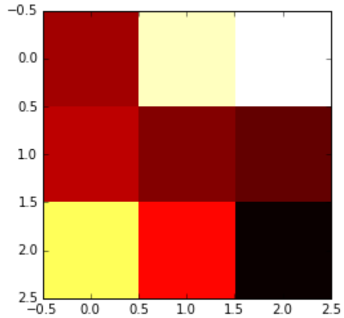
model #0 vs #3



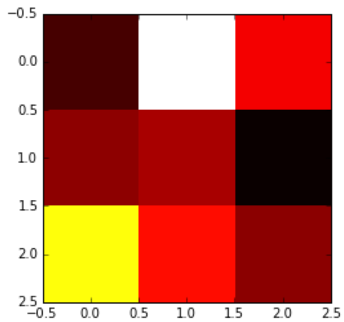
model #0 vs #4



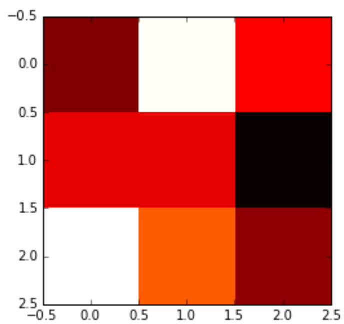
model #0 vs #5



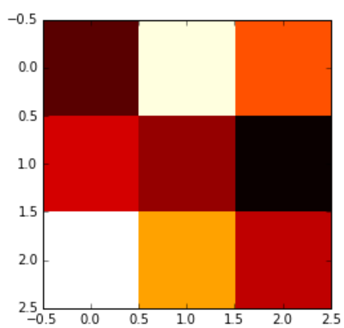
model #0 vs #6



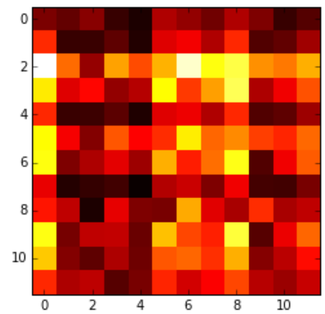
model #0 vs #7



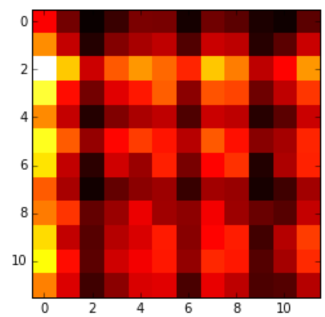
model #0 vs #8



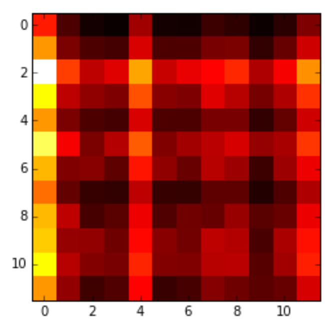
layer #2 "y"
2 params
12 filters
model #0 vs #1



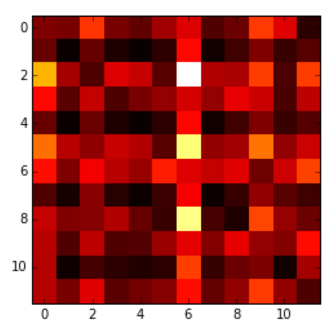
model #0 vs #2



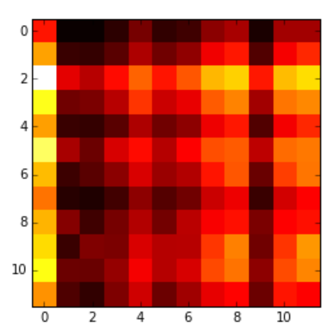
model #0 vs #3



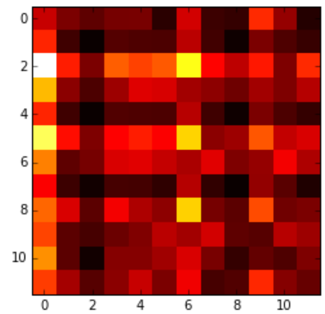
model #0 vs #4



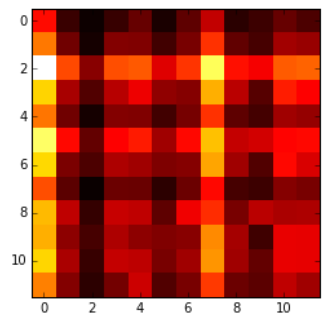
model #0 vs #5



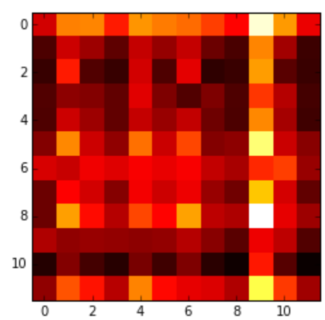
model #0 vs #6



model #0 vs #7



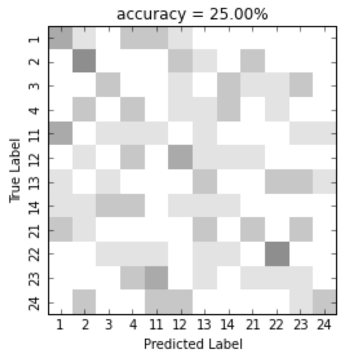
model #0 vs #8

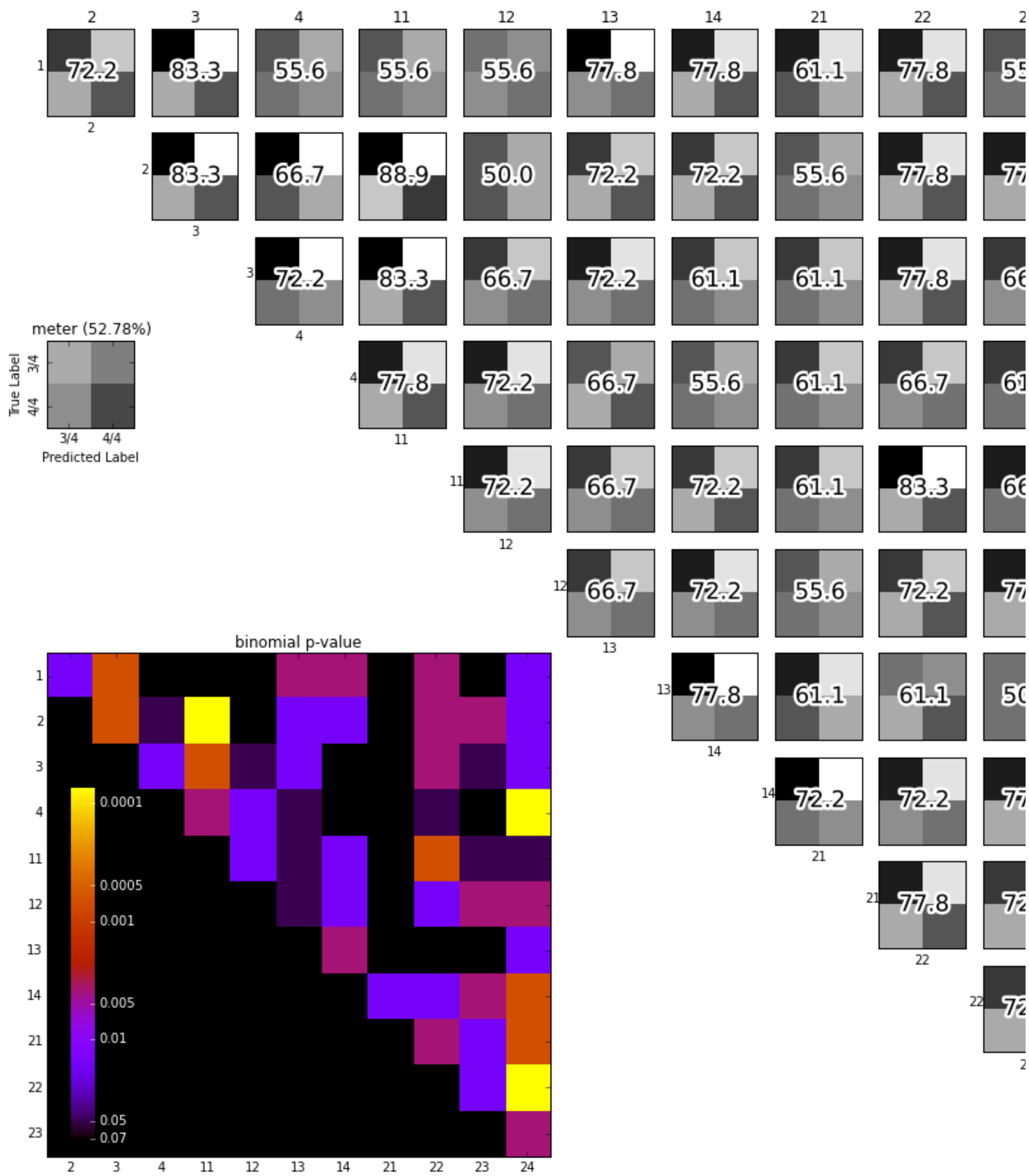


majority vote model

In [94]:
```
analyze(majvote_output_fn, {})
```

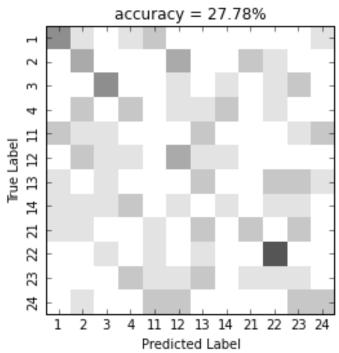
```
{'trial_no': [2]}
```

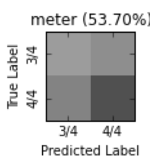


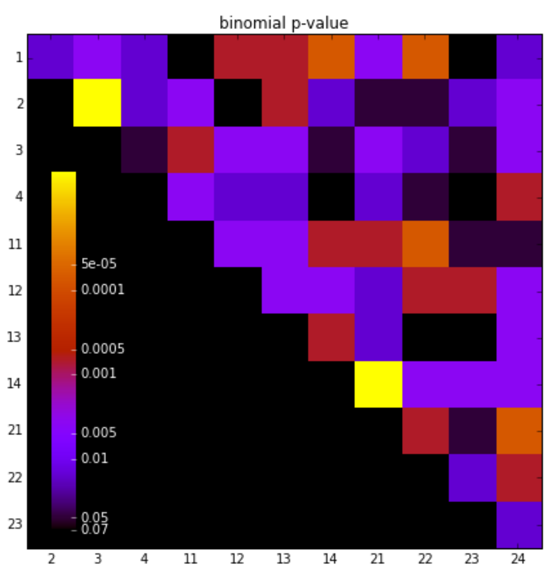


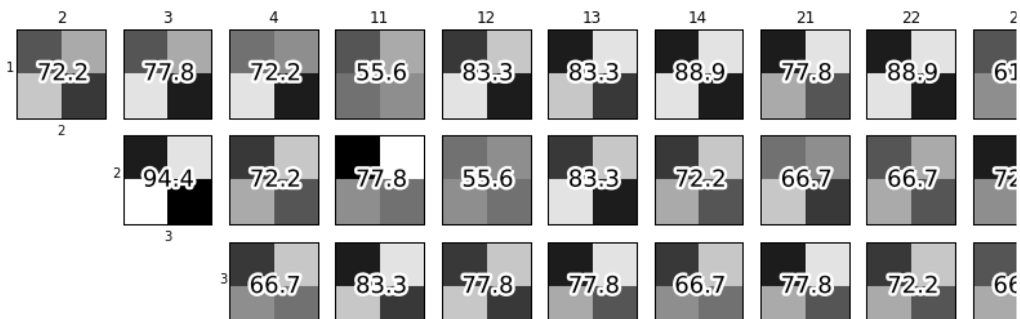
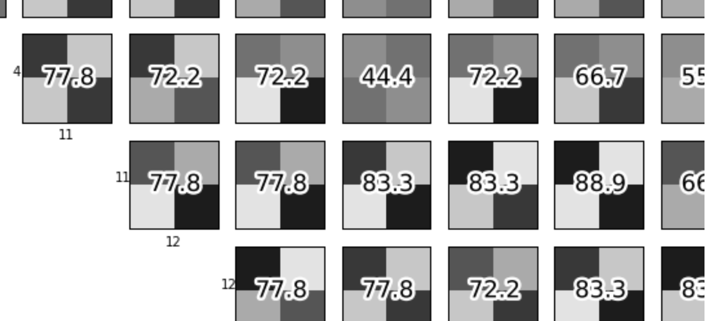
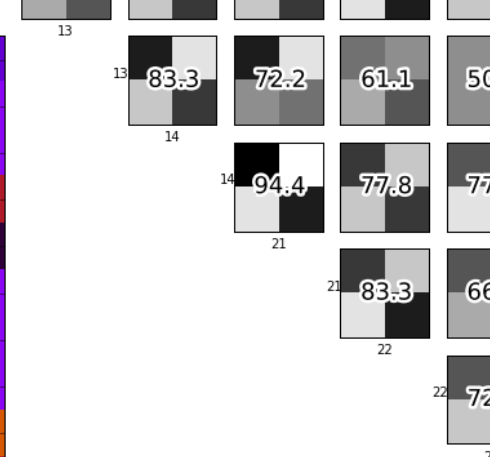
In [95]:
```
analyze(raw_majvote_output_fn, {})
```

```
{'trial_no': [2]}
```

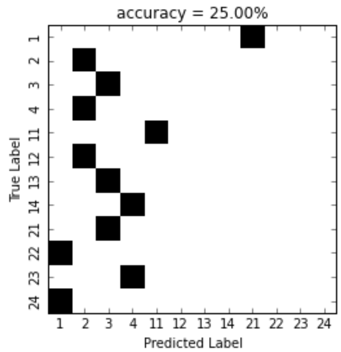


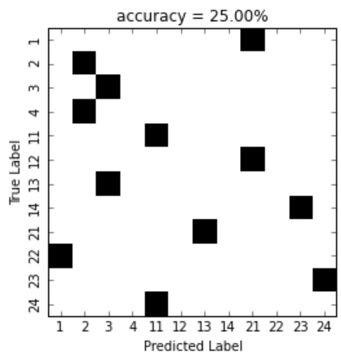




```
In [82]: # analyze(raw_majvote_out_fn, {'subject':'P01'})
         analyze_performance(majvote_output_fn, {'trial_no':TEST, 'subject':'P01'})
         analyze_performance(raw_majvote_output_fn, {'trial_no':TEST, 'subject':'P01'}
         )
```



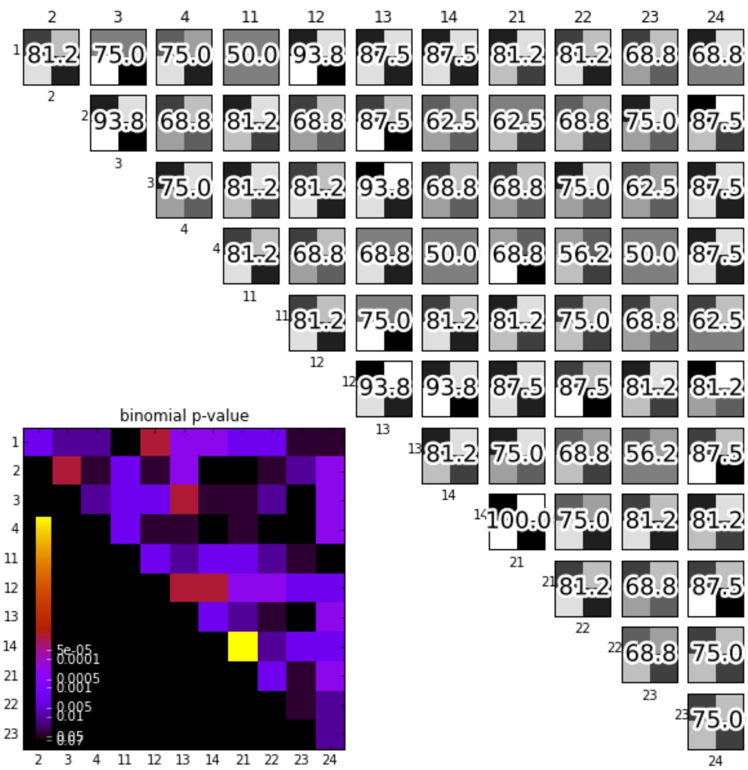


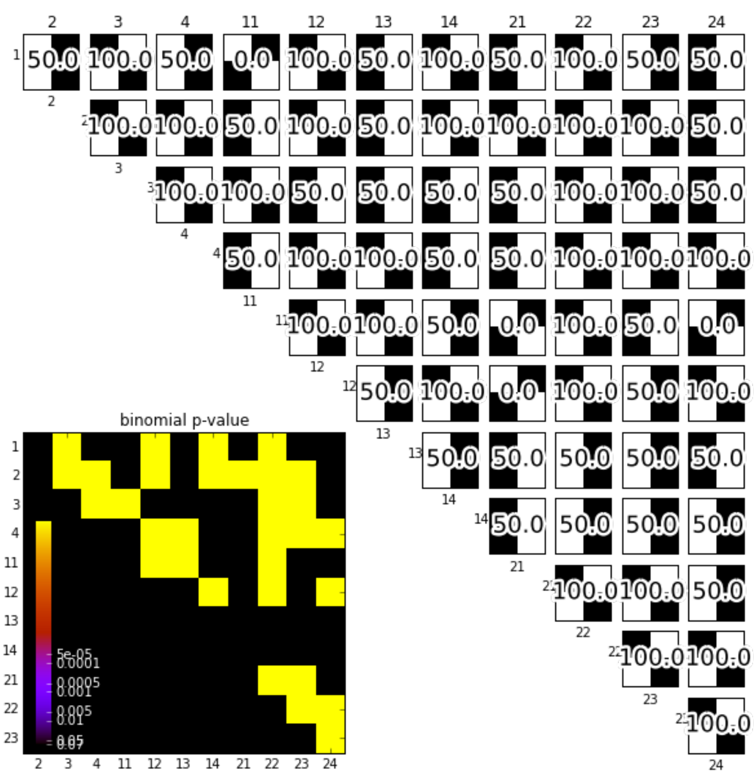

```
In [96]: def extract_structural_params(model):
             shapes = []
             dimensions = []
             sums = []
             for layer in model.layers:
                 try:
                     params = layer.get_param_values()
                     for p in params:
                         print p.shape
                         shapes.append(p.shape)
                         dimensions.append(np.prod(p.shape))
                         sums.append(np.sum(abs(p)))
                 except Exception as e:
                     print e
                     continue
             return np.sum(dimensions), shapes, dimensions, np.sum(sums), sums
         extract_structural_params(fin_model)

         (1, 64, 1, 4)
         (3, 1, 1, 9)
         (12,)
         (1287, 12)

Out[96]: (15739,
          [(1, 64, 1, 4), (3, 1, 1, 9), (12,), (1287, 12)],
          [256, 27, 12, 15444],
          16.114706,
          [0.10042809, 0.010227739, 10.0, 6.0040503])
```

In [73]:

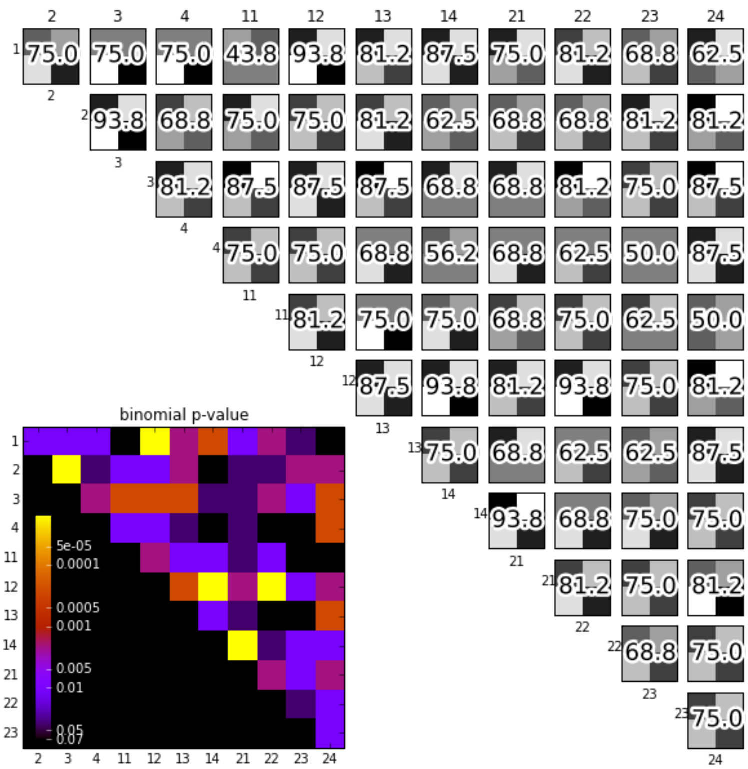
```
# select on 8 of 9 subjects
from deepthought.experiments.spearmint_cv_wrapper import generate_folds
FOLDS = generate_folds(['P01','P04','P06','P07','P09','P11','P12','P13','P14'
])
for fold in FOLDS:
    print fold
    selectors = dict(trial_no=TEST, subject=fold['train'], condition=[1])
    _plot_2class_confusion(base_dataset, selectors, avg_output_fn,
                           class_attribute='stimulus_id', classes=STIMULUS_IDS
, class_labels=None,
                           figsize=(10,10), show=True, plot_pvalues=True)

    selectors = dict(trial_no=TEST, subject=fold['valid'], condition=[1])
    _plot_2class_confusion(base_dataset, selectors, avg_output_fn,
                           class_attribute='stimulus_id', classes=STIMULUS_IDS
, class_labels=None,
                           figsize=(10,10), show=True, plot_pvalues=True)
#     break
```

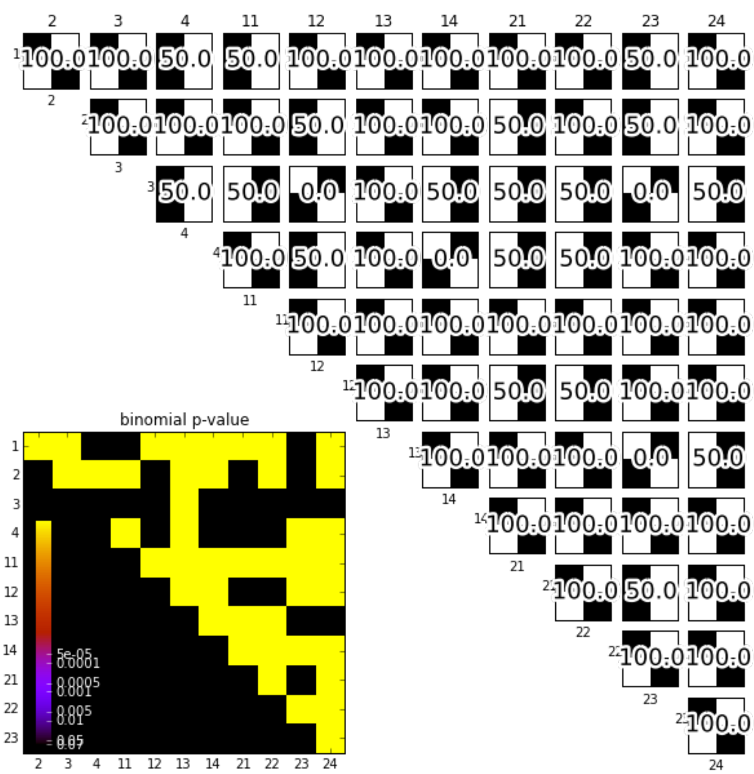
{'train': ['P04', 'P06', 'P07', 'P09', 'P11', 'P12', 'P13', 'P14'], 'valid':
['P01']}



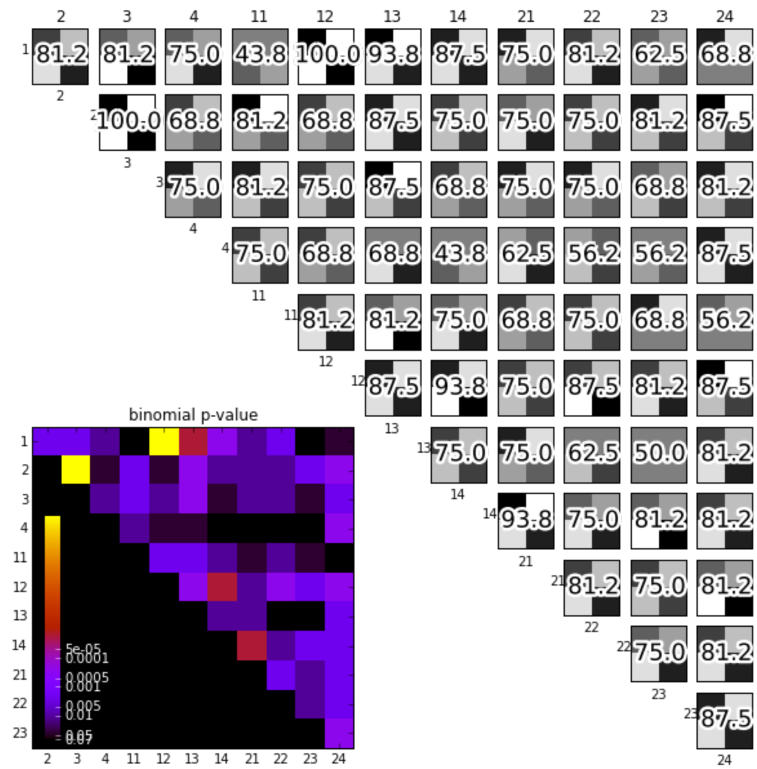


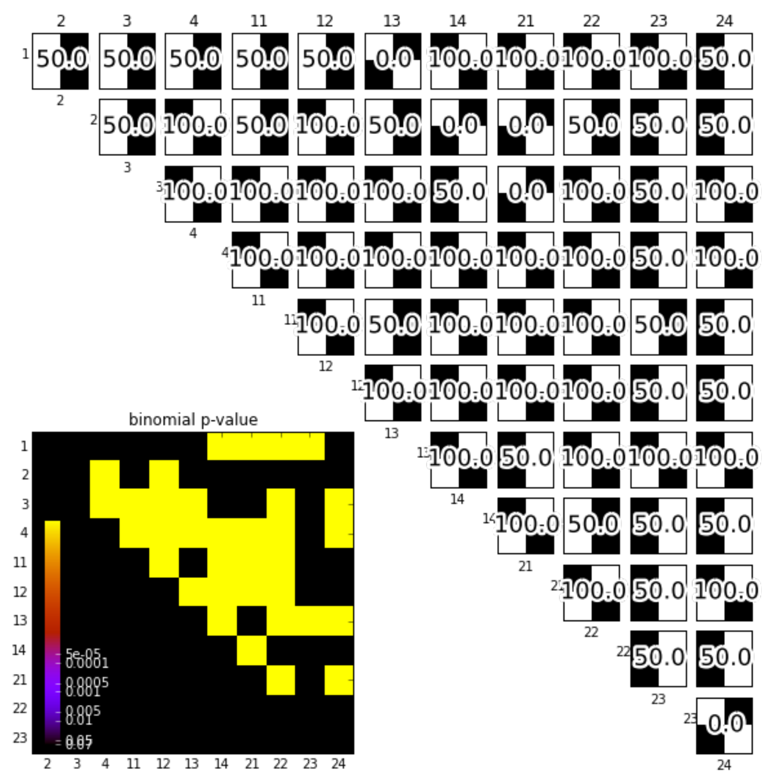
{'train': ['P01', 'P06', 'P07', 'P09', 'P11', 'P12', 'P13', 'P14'], 'valid':
['P04']}



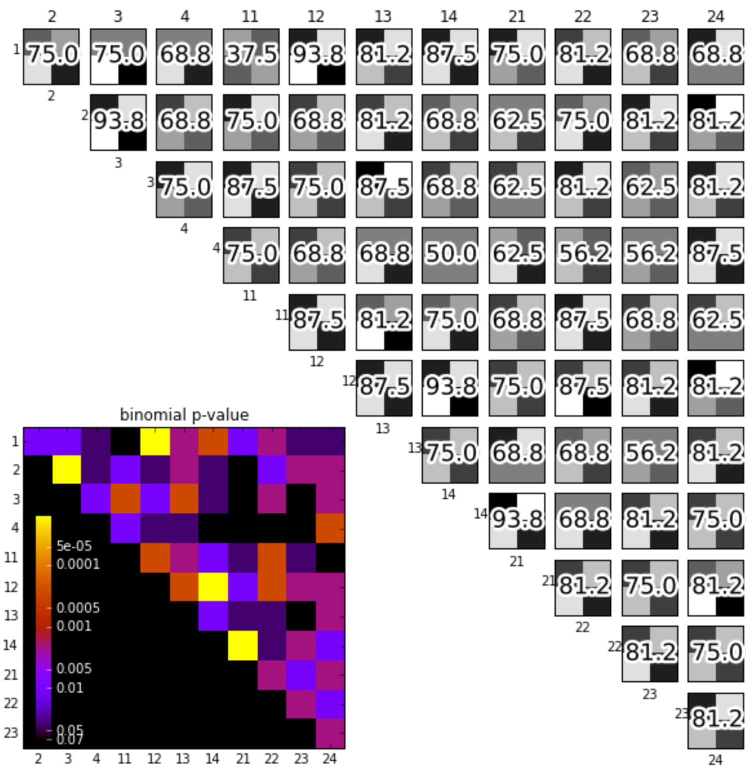


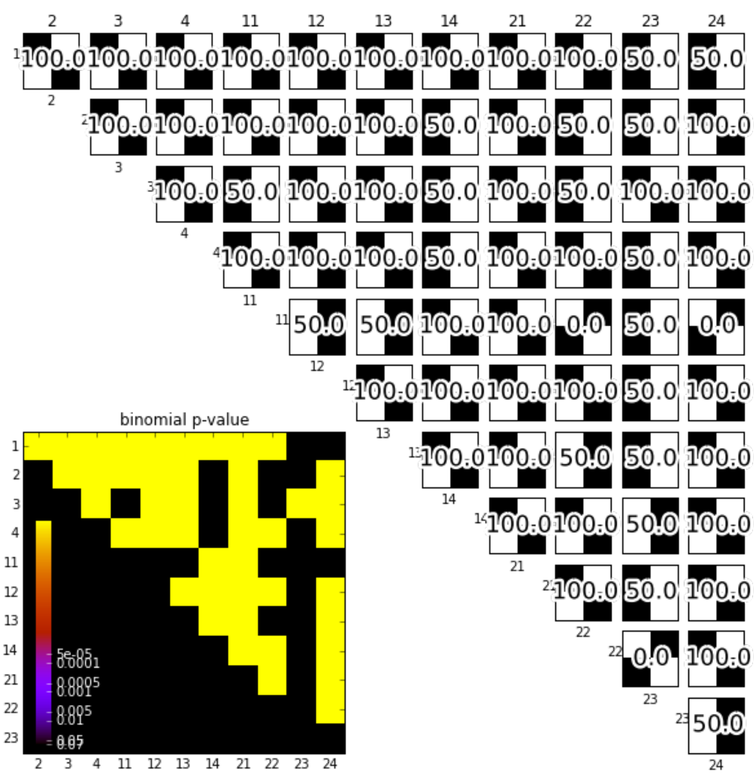
```
{'train': ['P01', 'P04', 'P07', 'P09', 'P11', 'P12', 'P13', 'P14'], 'valid':
['P06']}
```



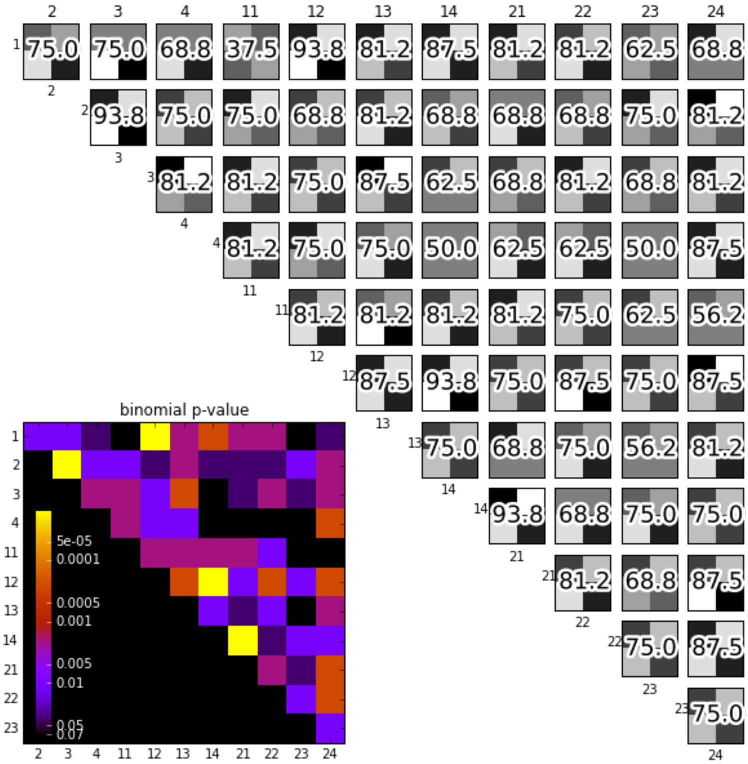


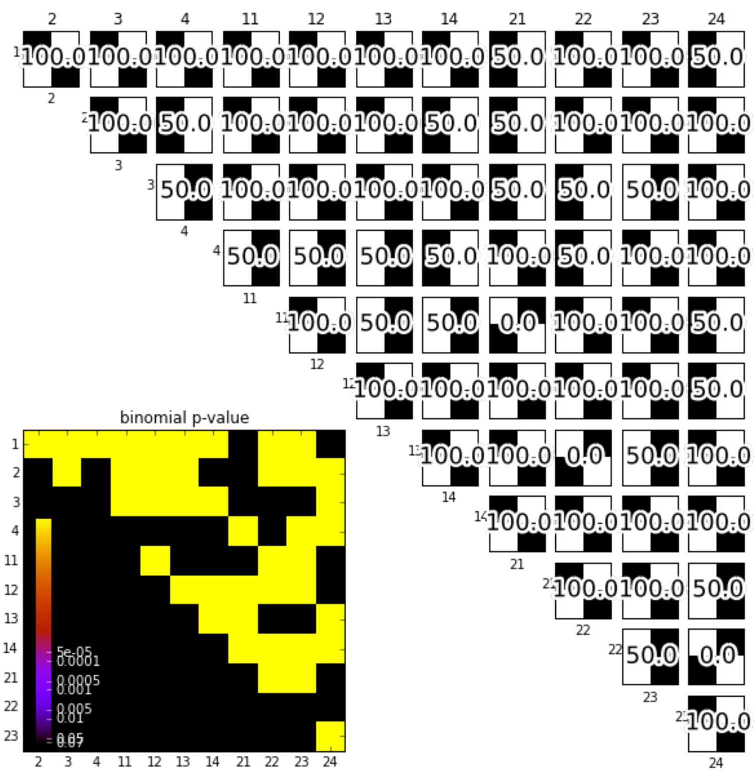
{'train': ['P01', 'P04', 'P06', 'P09', 'P11', 'P12', 'P13', 'P14'], 'valid': ['P07']}



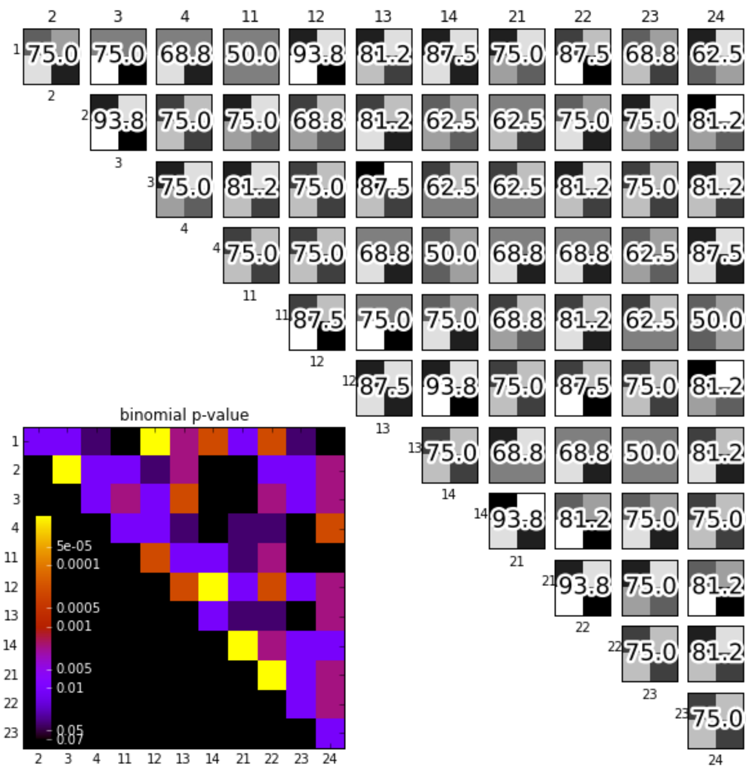


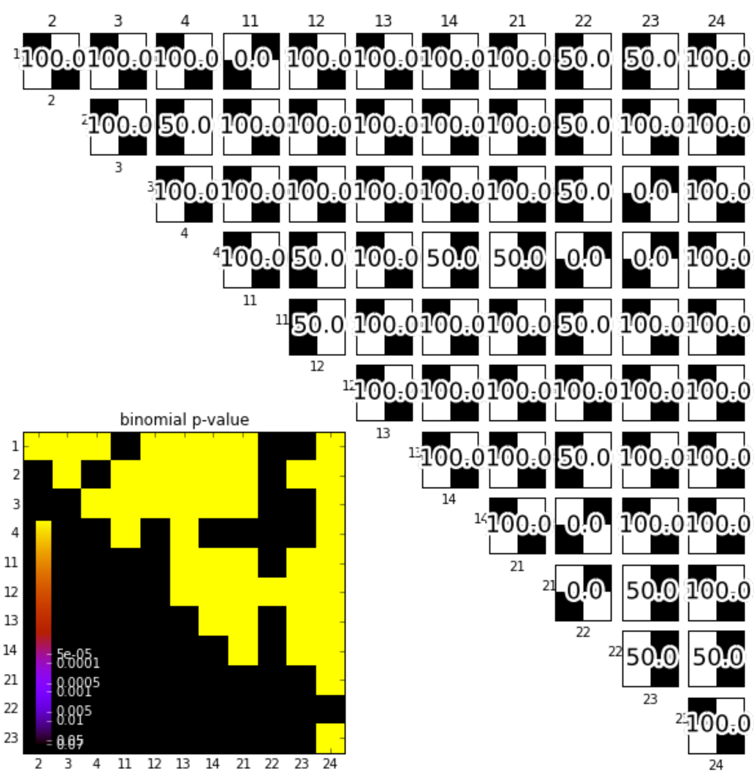
{'train': ['P01', 'P04', 'P06', 'P07', 'P11', 'P12', 'P13', 'P14'], 'valid': ['P09']}



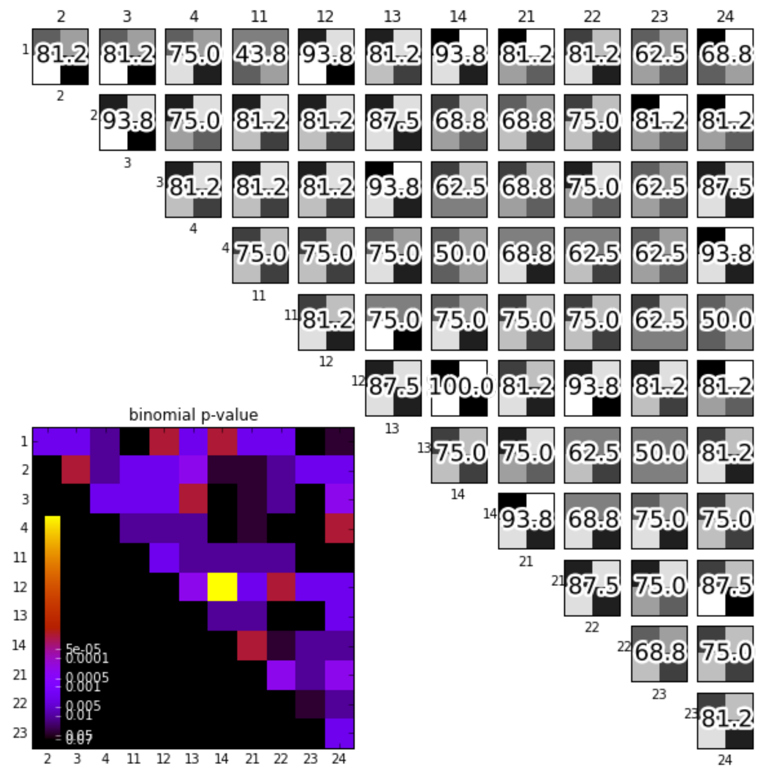


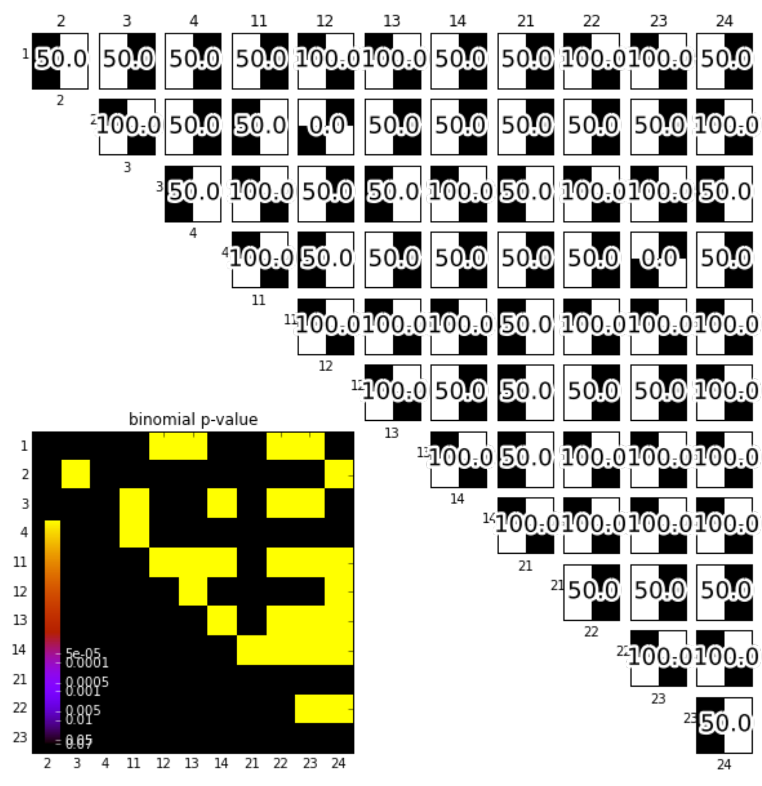
{'train': ['P01', 'P04', 'P06', 'P07', 'P09', 'P12', 'P13', 'P14'], 'valid': ['P11']}



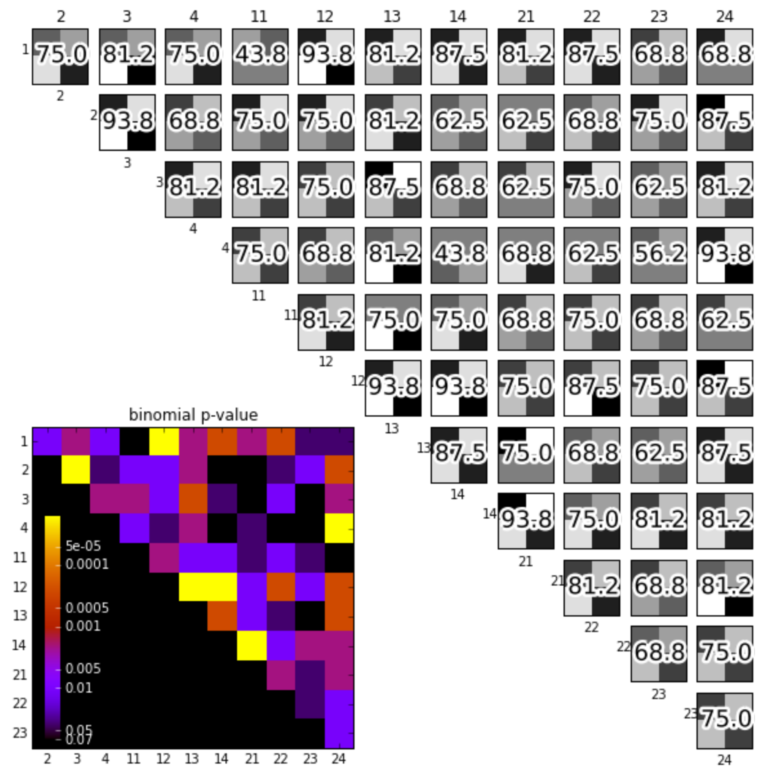


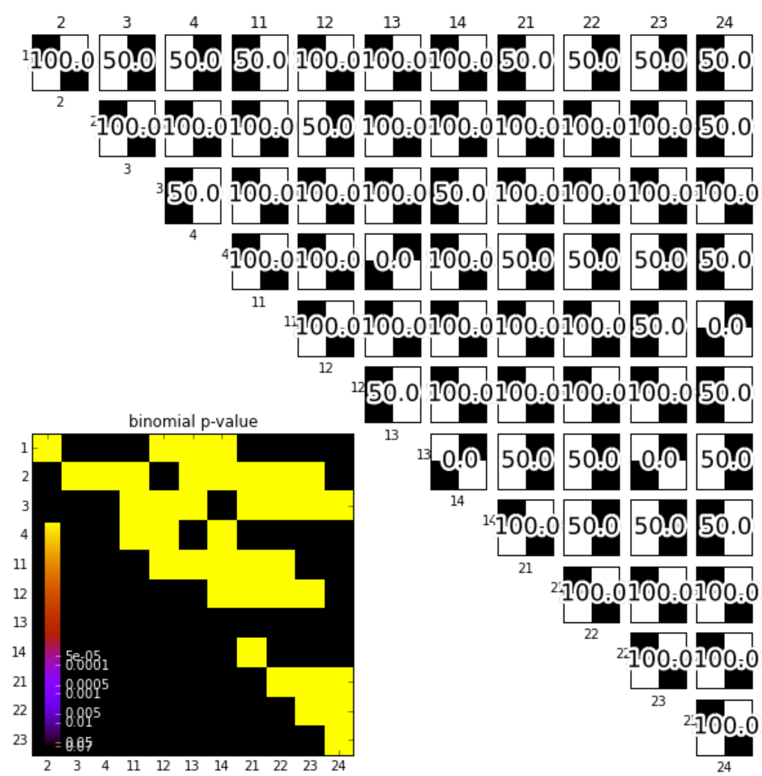
{'train': ['P01', 'P04', 'P06', 'P07', 'P09', 'P11', 'P13', 'P14'], 'valid': ['P12']}



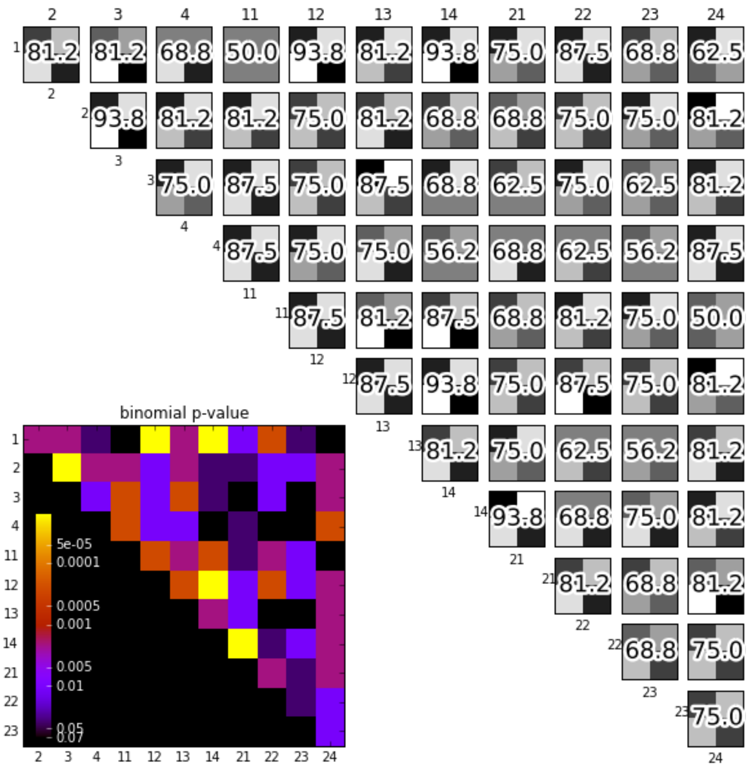


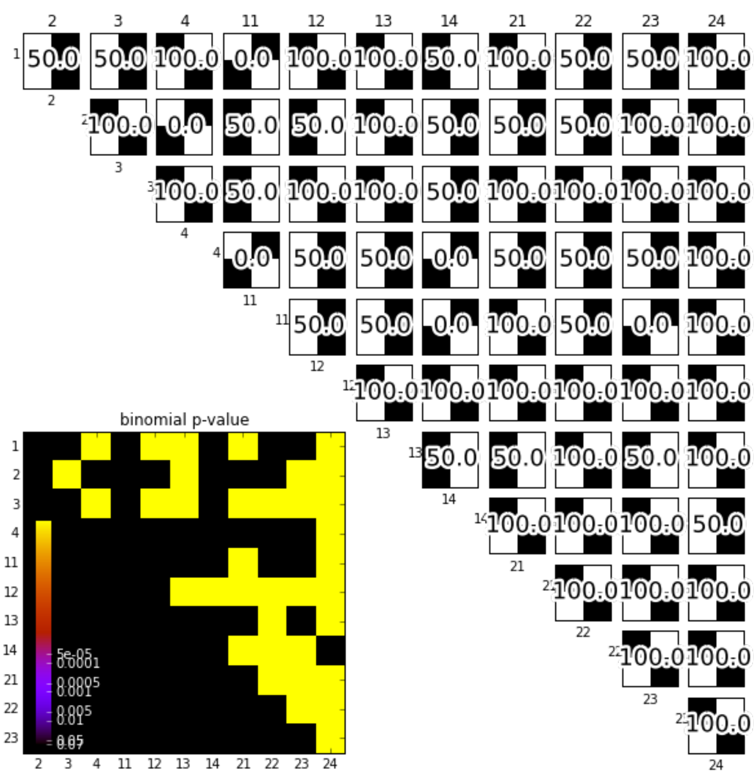
{'train': ['P01', 'P04', 'P06', 'P07', 'P09', 'P11', 'P12', 'P14'], 'valid':
['P13']}





{'train': ['P01', 'P04', 'P06', 'P07', 'P09', 'P11', 'P12', 'P13'], 'valid':
['P14']}

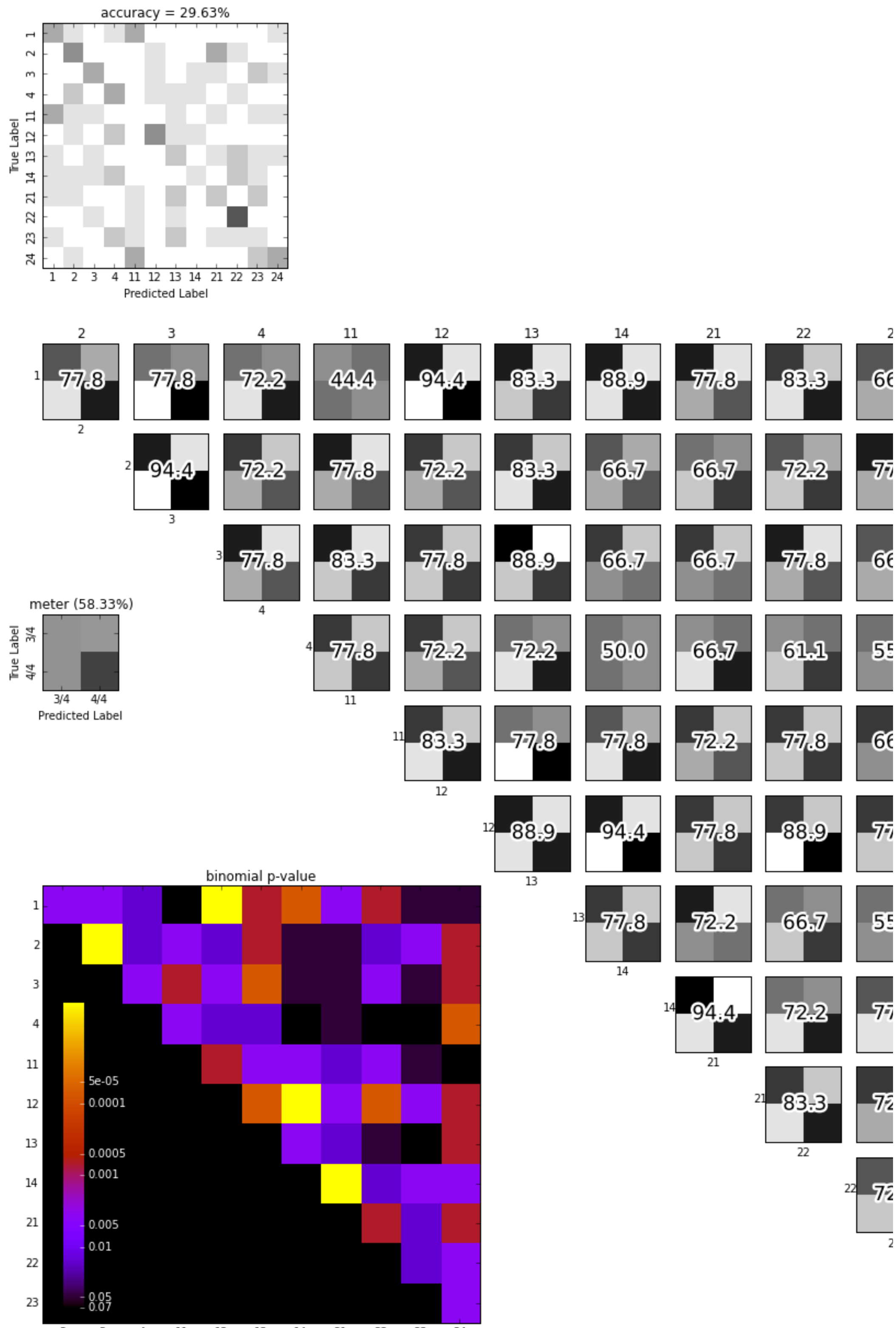

binomial p-value
5e-05
0.0001
0.0005
0.001
0.005
0.01
0.05
0.07


binomial p-value
5e-05
0.0001
0.0005
0.001
0.005
0.01
0.05
0.07

In [79]:

```
# overall binary test performance
analyze(avg_output_fn, {}) #, filenames=['confusion_overall.pdf', 'confusion_overall_2class.pdf'])
```

```
{'trial_no': [2]}
```



In [ ]: