This supplementary information presents :

- first, the code to generate the figures from the paper,
- second, some control experiments that were mentionned in the paper,
- finally, some perspectives for future work inspired by the algorithms presented in the paper.

Figures for "An adaptive algorithm for unsupervised learning"

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import numpy as np
np.set_printoptions(precision=2, suppress=True)
seed = 42
np.random.seed(seed)
```

In [3]:
```
# some overhead for the formatting of figures
import matplotlib.pyplot as plt

fontsize = 12
FORMATS = ['.pdf', '.eps', '.png', '.tiff']
FORMATS = ['.pdf', '.png']
dpi_export = 600

fig_width_pt = 318.670  # Get this from LaTeX using \showthe\columnwidth
fig_width_pt = 450  # Get this from LaTeX using \showthe\columnwidth
#fig_width_pt = 1024 #221     # Get this from LaTeX using \showthe\column
width / x264 asks for a multiple of 2
ppi = 72.27 # (constant) definition of the ppi = points per inch
inches_per_pt = 1.0/ppi  # Convert pt to inches
#inches_per_cm = 1./2.54
fig_width = fig_width_pt*inches_per_pt  # width in inches
grid_fig_width = 2*fig_width
phi = (np.sqrt(5) + 1. ) /2
#legend.fontsize = 8
#fig_width = 9
fig_height = fig_width/phi
figsize = (fig_width, fig_height)


def adjust_spines(ax, spines):
    for loc, spine in ax.spines.items():
        if loc in spines:
            spine.set_position(('outward', 10))  # outward by 10 points
            spine.set_smart_bounds(True)
        else:
            spine.set_color('none')  # don't draw spine

    # turn off ticks where there is no spine
```

```
    if 'left' in spines:
        ax.yaxis.set_ticks_position('left')
    else:
        # no yaxis ticks
        ax.yaxis.set_ticks([])

    if 'bottom' in spines:
        ax.xaxis.set_ticks_position('bottom')
    else:
        # no xaxis ticks
        ax.xaxis.set_ticks([])

import matplotlib
pylab_defaults = {
    'font.size': 10,
    'xtick.labelsize':'medium',
    'ytick.labelsize':'medium',
    'text.usetex': False,
#     'font.family' : 'sans-serif',
#     'font.sans-serif' : ['Helvetica'],
    }

#matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGenera
l', 'mathtext.fontset': 'stix'})
matplotlib.rcParams.update(pylab_defaults)
#matplotlib.rcParams.update({'text.usetex': True})

import matplotlib.cm as cm


from IPython.display import Image

DEBUG = True
DEBUG = False
hl, hs = 10*'-', 10*' '
```

In [4]:
```
tag = 'ICLR'
datapath = '../../SparseHebbianLearning/database'
# different runs
opts = dict(datapath=datapath, verbose=0)
#opts = dict(cache_dir='cache_dir_cluster', datapath=datapath, verbose=0)
#opts = dict(cache_dir='cache_dir_ICLR', datapath=datapath, verbose=0)
#opts = dict(cache_dir='cache_dir_cluster25', eta=0.002, eta_homeo=0.005,
 datapath=datapath, verbose=0)
```

In [5]:
```
from shl_scripts.shl_experiments import SHL
shl = SHL(**opts)
data = shl.get_data(matname=tag)
```

In [6]:
```
shl?
```

```
Type:        SHL
String form: <shl_scripts.shl_experiments.SHL object at 0x1044974e0>
File:        ~/science/SparseHebbianLearning/shl_scripts/shl_experiment
s.py
Docstring:
Base class to define SHL experiments:
    - initialization
    - coding and learning
    - visualization
    - quantitative analysis
```

In [7]:
```
print('number of patches, size of patches = ', data.shape)
print('average of patches = ', data.mean(), ' +/- ', data.mean(axis=1).st
d())
SE = np.sqrt(np.mean(data**2, axis=1))
print('average energy of data = ', SE.mean(), '+/-', SE.std())
```

```
number of patches, size of patches =  (65520, 484)
average of patches =  -7.953667832096442e-06  +/-  0.005778375977326485
average energy of data =  0.2449634589359681 +/- 0.07497821676401197
```

In [8]:
```
#!ls -l {shl.cache_dir}/{tag}*
!ls {shl.cache_dir}/{tag}*lock*
!rm {shl.cache_dir}/{tag}*lock*
#!rm {shl.cache_dir}/{tag}*
#!ls -l {shl.cache_dir}/{tag}*
```

```
ls: cache_dir/ICLR*lock*: No such file or directory
rm: cache_dir/ICLR*lock*: No such file or directory
```

figure 1: Role of homeostasis in learning sparse representations

TODO : cross-validate with 10 different learnings

In [9]:
```
fname = 'figure_map'
N_cv = 10
one_cv = 9 # picking one to display intermediate results
```

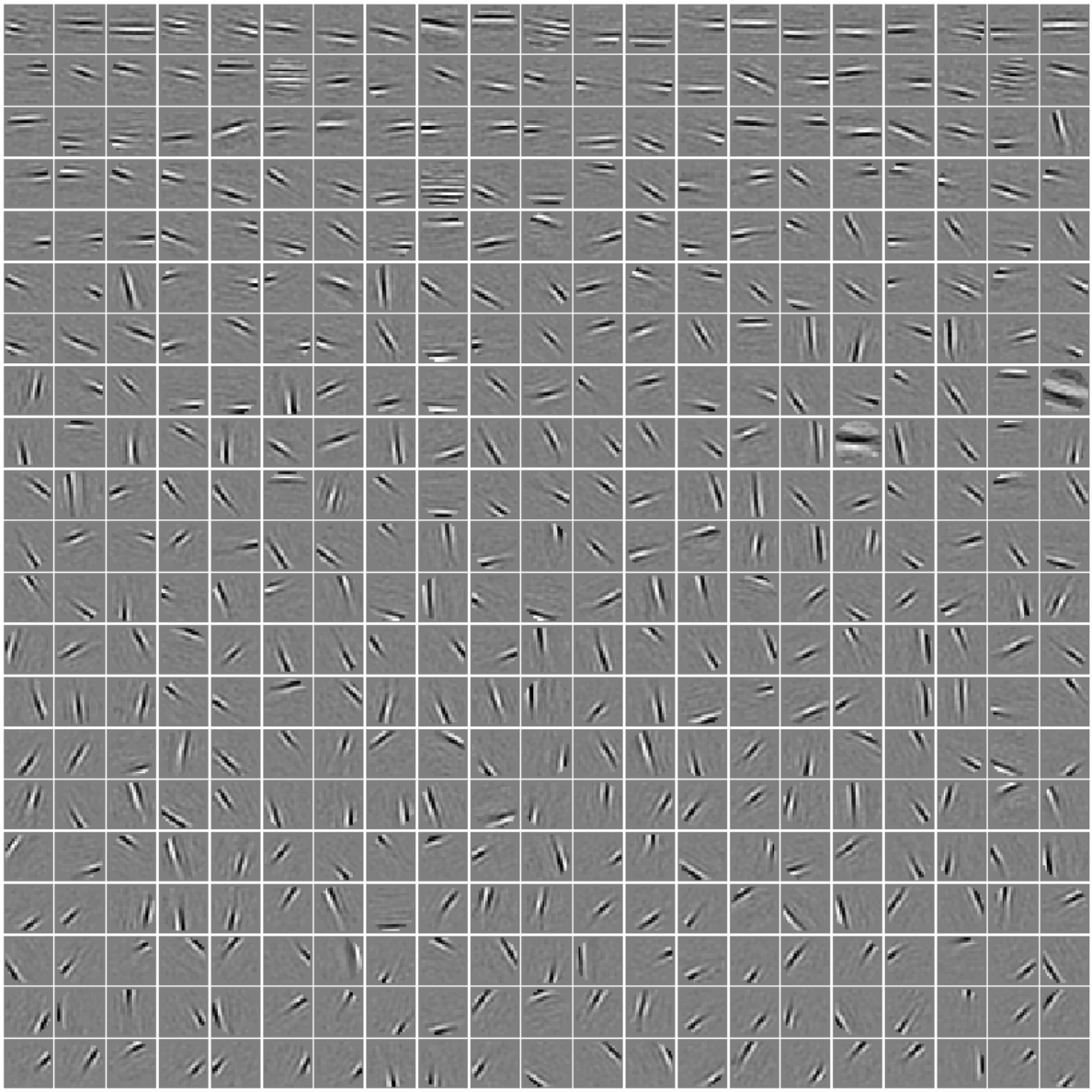
learning

The actual learning is done in a second object (here `dico`) from which we can access another set of properties and functions (see the shl_learn.py (https://github.com/bicv/SHL_scripts/blob/master/shl_scripts/shl_learn.py) script):

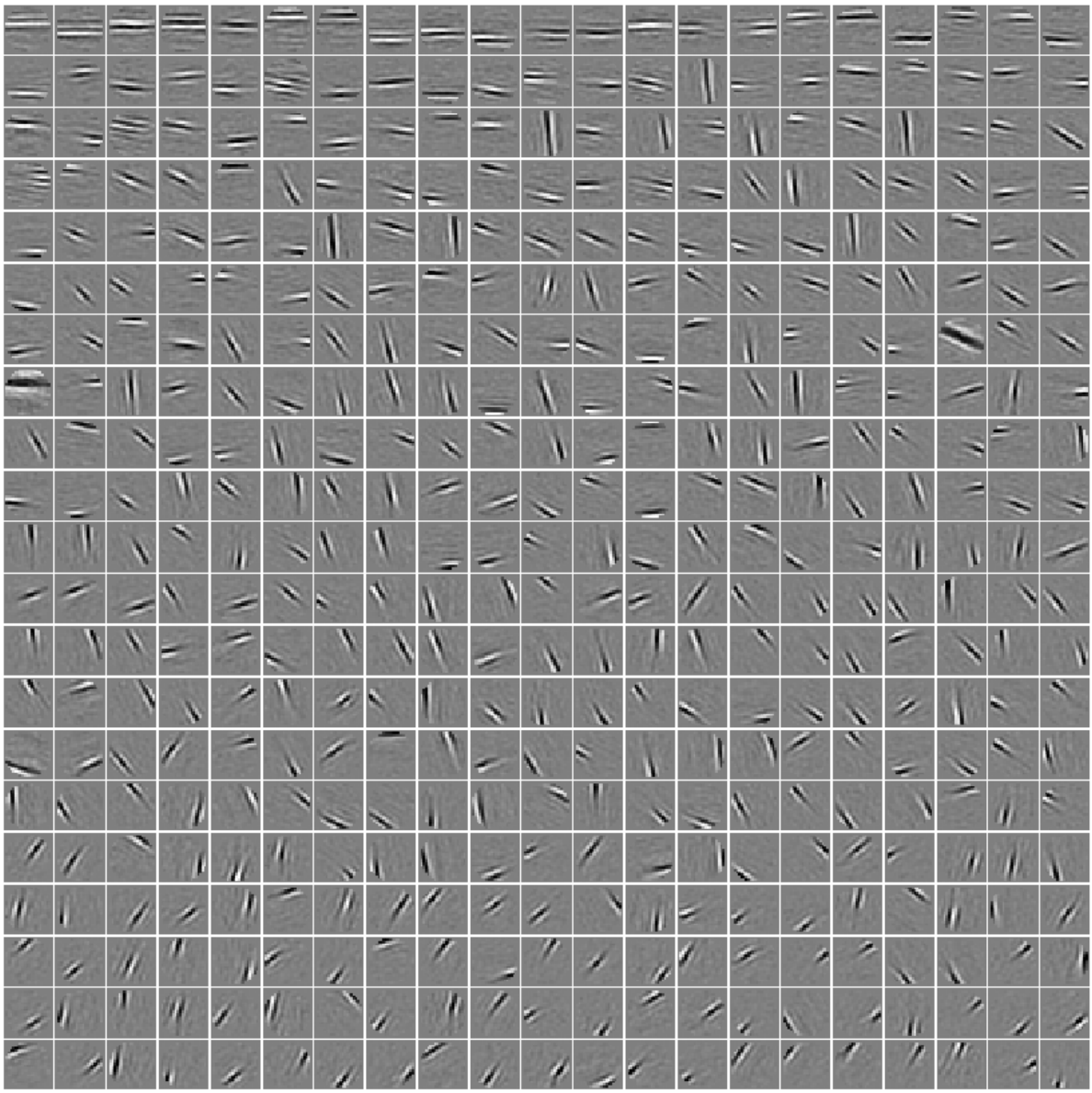
In [10]:
```
homeo_methods = ['None', 'OLS', 'HEH']

list_figures = ['show_dico', 'time_plot_error', 'time_plot_logL', 'time_p
lot_MC', 'show_Pcum']
list_figures = []
dico = {}
for i_cv in range(N_cv):
    dico[i_cv] = {}
    for homeo_method in homeo_methods:
        shl = SHL(homeo_method=homeo_method, seed=seed+i_cv, **opts)
        dico[i_cv][homeo_method] = shl.learn_dico(data=data,
list_figures=list_figures, matname=tag + '_' + homeo_method + '_seed=' +
str(seed+i_cv))
```

In [11]:
```
list_figures = ['show_dico']
for i_cv in [one_cv]:
    for homeo_method in homeo_methods:
        print(hl + hs + homeo_method[:3] + hs + hl)
        shl = SHL(homeo_method=homeo_method, seed=seed+i_cv, **opts)
        shl.learn_dico(data=data, list_figures=list_figures, matname=tag
+ '_' + homeo_method + '_seed=' + str(seed+i_cv))

        print('size of dictionary = (number of filters, size of imagelet
s) = ', dico[i_cv][homeo_method].dictionary.shape)
        print('average of filters = ',  dico[i_cv][homeo_method].dictiona
ry.mean(axis=1).mean(),
              '+/-',  dico[i_cv][homeo_method].dictionary.mean(axis=1).st
d())
        SE = np.sqrt(np.sum(dico[i_cv][homeo_method].dictionary**2,
axis=1))
        print('average energy of filters = ', SE.mean(), '+/-', SE.std())
        plt.show()
```

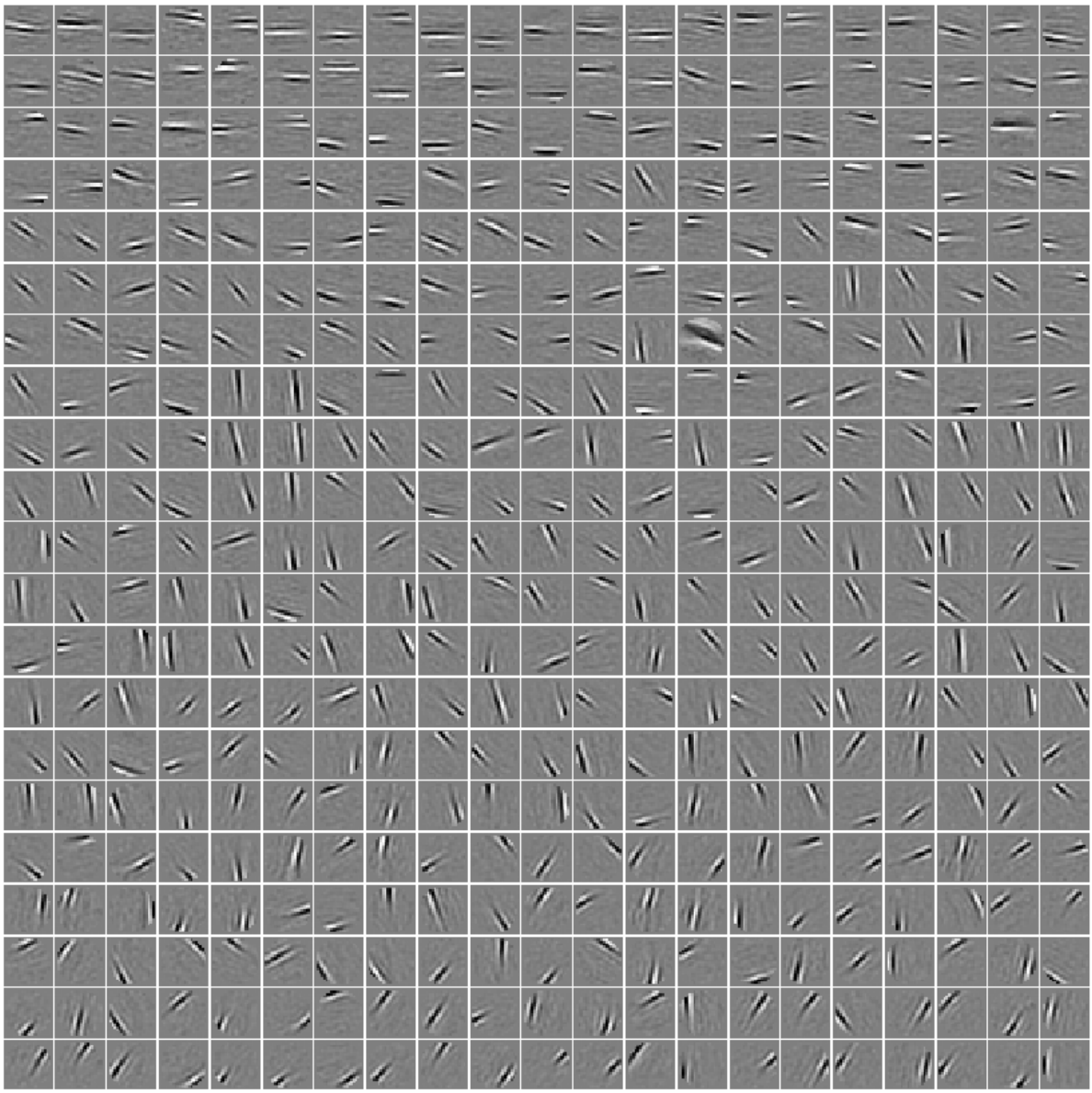
```
----------           Non           ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 48
4)
average of filters =  -9.65917219840568e-06 +/- 0.0005992682475744378
average energy of filters =  1.0 +/- 4.3915262235773366e-17
```



```
----------           OLS           ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 48
4)
average of filters =  -5.903032155117969e-06 +/- 0.0005811583965811138
average energy of filters =  1.0 +/- 4.5478569413264154e-17
```



```
----------           HEH           ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 48
4)
average of filters =  -4.325685413494333e-06 +/- 0.0005996641505749903
average energy of filters =  1.0 +/- 2.797499069501051e-17
```



panel A: plotting some dictionaries

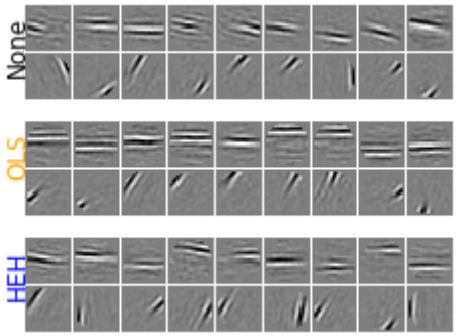
In [12]:
```
pname = '/tmp/panel_A' #pname = fname + '_A'
```

In [13]:
```
from shl_scripts import show_dico
if DEBUG: show_dico(shl, dico[one_cvi_cv][homeo_method], data=data, dim_g
raph=(2,5))
```

In [14]:
```
dim_graph = (2, 9)
colors = ['black', 'orange', 'blue']
homeo_methods
```

Out[14]:
```
['None', 'OLS', 'HEH']
```

In [15]:
```
subplotpars = dict(left=0.042, right=1., bottom=0., top=1., wspace=0.05,
hspace=0.05,)
fig, axs = plt.subplots(3, 1, figsize=(fig_width/2, fig_width/(1+phi)), g
ridspec_kw=subplotpars)

for ax, color, homeo_method in zip(axs.ravel(), colors, homeo_methods):
    ax.axis(c=color, lw=2, axisbg='w')
    ax.set_facecolor('w')
    fig, ax = show_dico(shl, dico[one_cv][homeo_method], data=data, dim_g
raph=dim_graph, fig=fig, ax=ax)
    # ax.set_ylabel(homeo_method)
    ax.text(-9, 7*dim_graph[0], homeo_method, fontsize=12, color=color, r
otation=90)#, backgroundcolor='white'

for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
```



In [16]:
```
### TODO put the p_min an p_max value in the filter map
```

In [17]:
```
if DEBUG: Image(pname +'.png')
```

In [18]:
```
if DEBUG: help(fig.subplots_adjust)
```

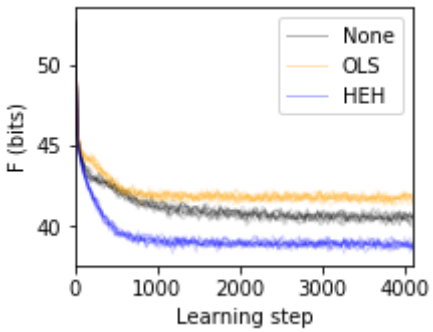
In [19]:
```
if DEBUG: help(plt.subplots)
```

In [20]:
```
if DEBUG: help(matplotlib.gridspec.GridSpec)
```

panel B: quantitative comparison

In [21]:
```
pname = '/tmp/panel_B' #fname + '_B'
```

In [22]:
```
from shl_scripts import time_plot
variable = 'F'
alpha_0, alpha = .3, .15
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.0
5, hspace=0.05,)
fig, ax = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gr
idspec_kw=subplotpars)
for i_cv in range(N_cv):
    for color, homeo_method in zip(colors, homeo_methods):
        ax.axis(c='b', lw=2, axisbg='w')
        ax.set_facecolor('w')
        if i_cv==0:
            fig, ax = time_plot(shl, dico[i_cv][homeo_method], variable=v
ariable, unit='bits', color=color, label=homeo_method, alpha=alpha_0,
fig=fig, ax=ax)
        else:
            fig, ax = time_plot(shl, dico[i_cv][homeo_method], variable=v
ariable, unit='bits', color=color, alpha=alpha, fig=fig, ax=ax)
        # ax.set_ylabel(homeo_method)
        #ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color
='k', rotation=90)#, backgroundcolor='white'
ax.legend(loc='best')
for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```



Montage of the subplots

In [23]:
```
import tikzmagic
```

In [24]:
```
%load_ext tikzmagic
```

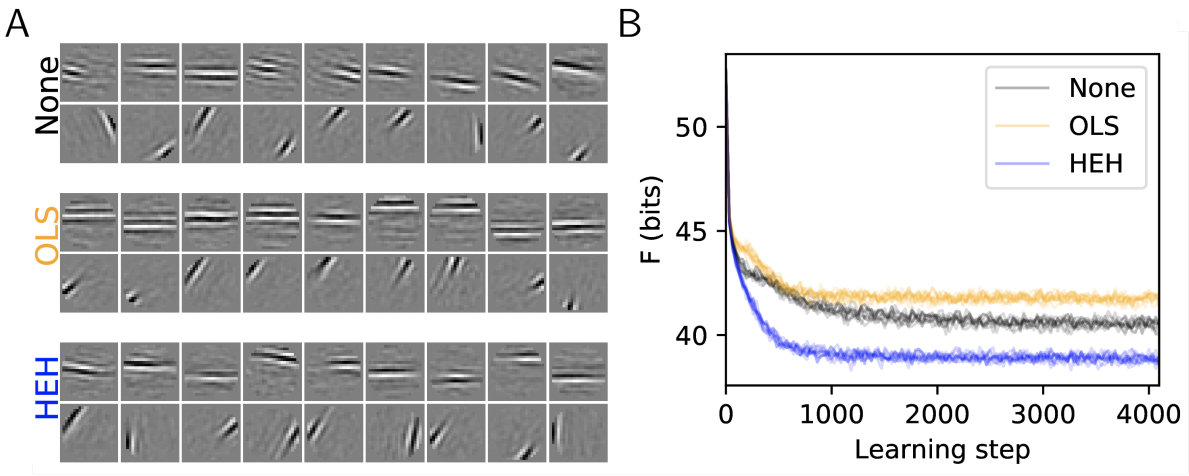
In [25]:
```
#DEBUG = True
if DEBUG: help(tikzmagic)
```

%tikz \draw (0,0) rectangle (1,1);%%tikz --save {fname}.pdf \draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\linewidth) ;

In [26]:
```
%%tikz -f pdf --save {fname}.pdf
\draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\l
inewidth) ;
\draw [anchor=north west] (.0\linewidth, .382\linewidth) node {\includegr
aphics[width=.5\linewidth]{/tmp/panel_A}};
\draw [anchor=north west] (.5\linewidth, .382\linewidth) node {\includegr
aphics[width=.5\linewidth]{/tmp/panel_B}};
\begin{scope}[font=\bf\sffamily\large]
\draw [anchor=west,fill=white] (.0\linewidth, .382\linewidth) node [above
 right=-3mm] {$\mathsf{A}$};
\draw [anchor=west,fill=white] (.53\linewidth, .382\linewidth) node [abov
e right=-3mm] {$\mathsf{B}$};
\end{scope}
```

In [27]:
```
!convert  -density {dpi_export} {fname}.pdf {fname}.jpg
!convert  -density {dpi_export} {fname}.pdf {fname}.png
#!convert  -density {dpi_export} -resize 5400  -units pixelsperinch -flat
ten  -compress lzw  -depth 8 {fname}.pdf {fname}.tiff
Image(fname +'.png')
```

Out[27]:



!echo "width=" ; convert {fname}.tiff -format "%[fx:w]" info: !echo ", \nheight=" ; convert {fname}.tiff -format "%[fx:h]" info: !echo ", \nunit=" ; convert {fname}.tiff -format "%U" info:!identify {fname}.tiff

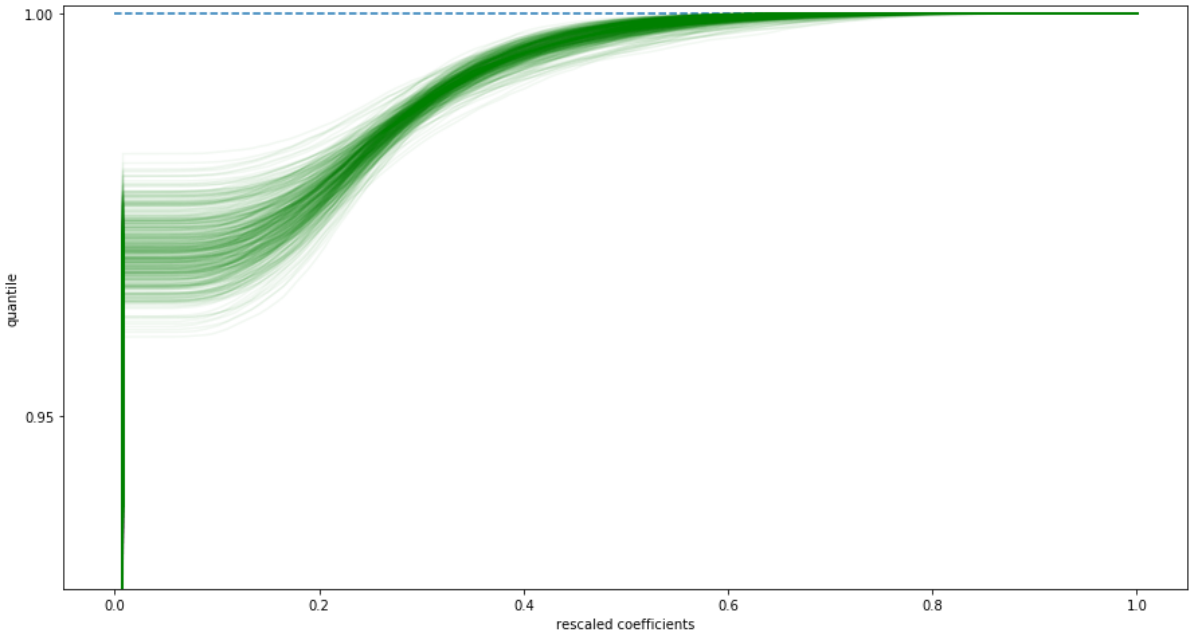
figure 2: Histogram Equalization Homeostasis

In [28]:
```
fname = 'figure_HEH'
```

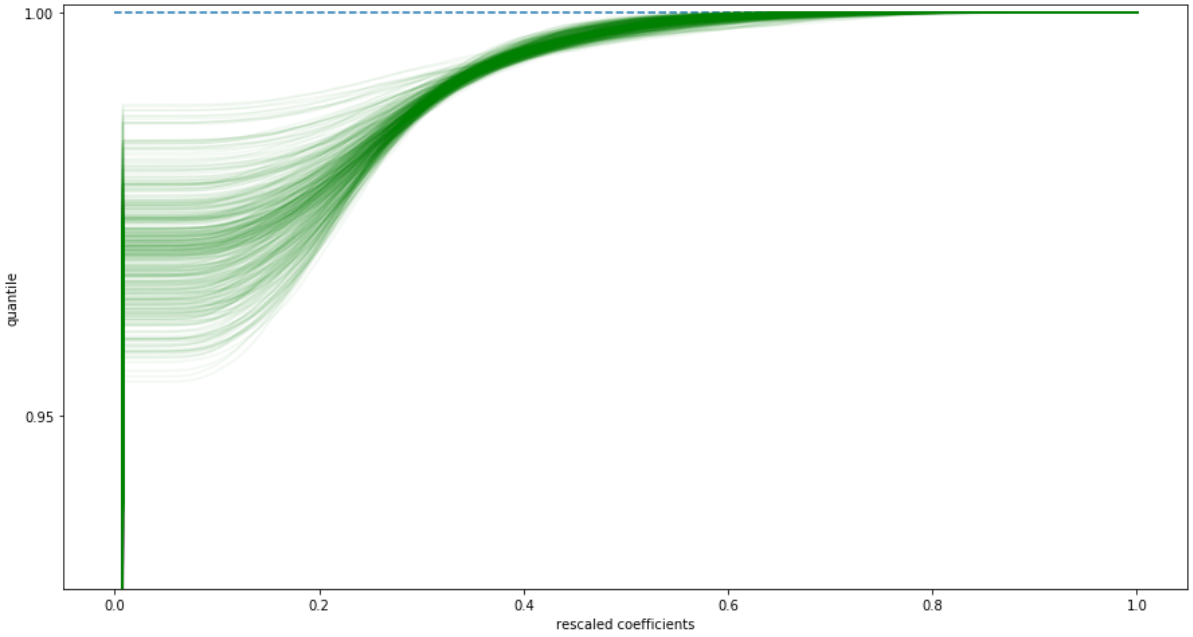
First collecting data:

In [29]:
```
list_figures = ['show_Pcum']

dico = {}
for homeo_method in homeo_methods:
    print(hl + hs + homeo_method + hs + hl)
    shl = SHL(homeo_method=homeo_method, **opts)
    #dico[homeo_method] = shl.learn_dico(data=data, list_figures=list_fig
ures, matname=tag + '_' + homeo_method + '_' + str(one_cv))
    dico[homeo_method] = shl.learn_dico(data=data, list_figures=list_figu
res, matname=tag + '_' + homeo_method + '_seed=' + str(seed+one_cv))
    plt.show()
```

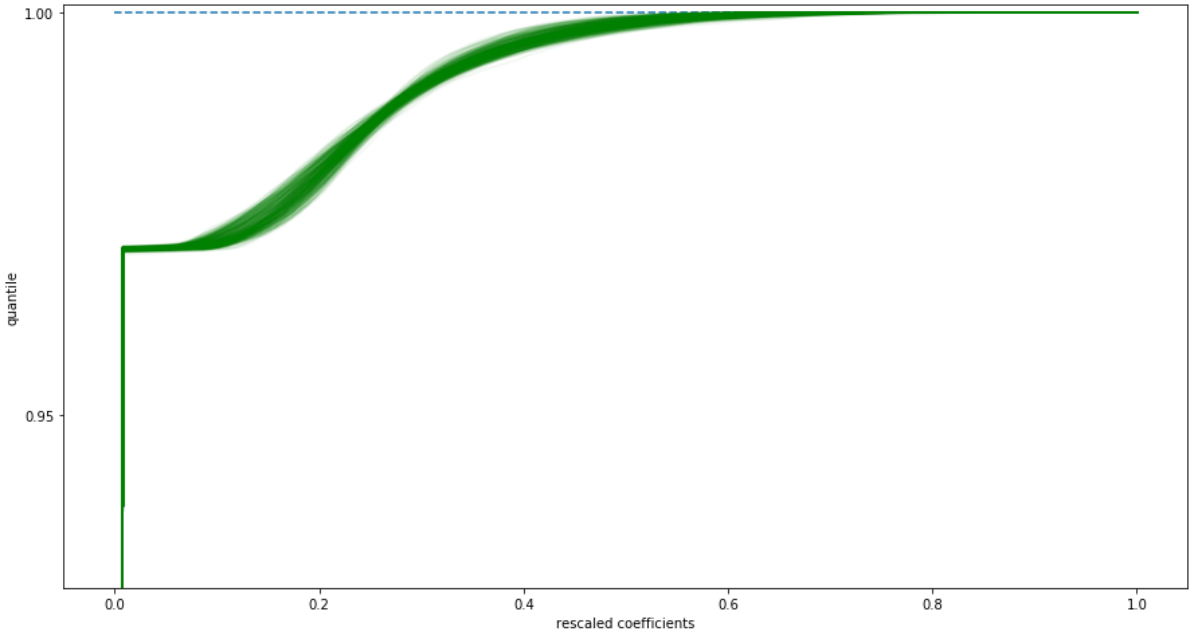
---------- None ----------


1.00
0.95
quantile
0.0
0.2
0.4
0.6
0.8
1.0
rescaled coefficients

---------- OLS ----------


1.00
0.95
quantile
0.0
0.2
0.4
0.6
0.8
1.0
rescaled coefficients

---------- HEH ----------

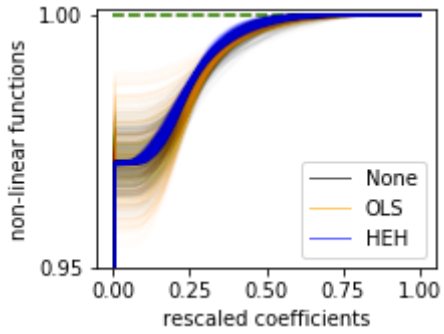


In [30]:
```
dico[homeo_method].P_cum.shape
```

Out[30]: (441, 128)

panel A: different P_cum

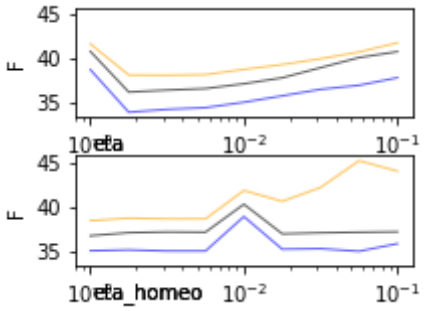
In [31]:
```
pname = '/tmp/panel_A' #pname = fname + '_A'

from shl_scripts import plot_P_cum
variable = 'F'
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.0
5, hspace=0.05,)
fig, ax = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gr
idspec_kw=subplotpars)
for color, homeo_method in zip(colors, homeo_methods):
    ax.axis(c='b', lw=2, axisbg='w')
    ax.set_facecolor('w')
    fig, ax = plot_P_cum(dico[homeo_method].P_cum, ymin=0.95, ymax=1.001,

                         title=None, suptitle=None, ylabel='non-linear fu
nctions',
                         verbose=False, n_yticks=21, alpha=.02, c=color,
fig=fig, ax=ax)
    ax.plot([0], [0], lw=1, color=color, label=homeo_method, alpha=.6)
    # ax.set_ylabel(homeo_method)
    #ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color='k', ro
tation=90)#, backgroundcolor='white'
ax.legend(loc='lower right')
for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```



In [32]:
```
if DEBUG: help(fig.legend)
```

panel B: comparing the effects of parameters

In [33]:
```
pname = '/tmp/panel_B' #fname + '_B'
n_jobs = 1

from shl_scripts.shl_experiments import SHL_set
homeo_methods = ['None', 'OLS', 'HEH']
variables = ['eta', 'eta_homeo']

list_figures = []

for homeo_method in homeo_methods:
    opts_ = opts.copy()
    opts_.update(homeo_method=homeo_method)
    experiments = SHL_set(opts_, tag=tag + '_' + homeo_method, base=10)
    experiments.run(variables=variables, n_jobs=n_jobs, verbose=0)

import matplotlib.pyplot as plt
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95, wspace=0.5,
hspace=0.35,)

x, y = .05, -.3

fig, axs = plt.subplots(len(variables), 1, figsize=(fig_width/2, fig_widt
h/(1+phi)), gridspec_kw=subplotpars, sharey=True)

for i_ax, variable in enumerate(variables):
    for color, homeo_method in zip(colors, homeo_methods):
        opts_ = opts.copy()
        opts_.update(homeo_method=homeo_method)
        experiments = SHL_set(opts_, tag=tag + '_' + homeo_method, base=1
0)
        fig, axs[i_ax] = experiments.scan(variable=variable,
list_figures=[], display='final', fig=fig, ax=axs[i_ax], color=color, dis
play_variable='F', verbose=0) #, label=homeo_metho
        axs[i_ax].set_xlabel('') #variable
        axs[i_ax].text(x, y,  variable, transform=axs[i_ax].transAxes)
        #axs[i_ax].get_xaxis().set_major_formatter(matplotlib.ticker.Scal
arFormatter())

#fig.legend(loc='lower right')
for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```

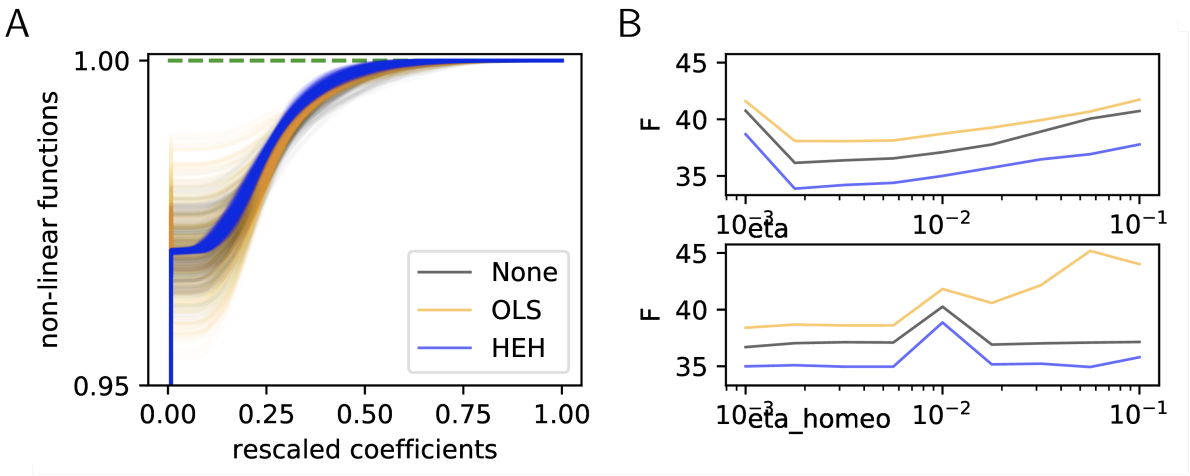


Montage of the subplots

In [34]:
```
%%tikz -f pdf --save {fname}.pdf
\draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\l
inewidth) ;
\draw [anchor=north west] (.0\linewidth, .382\linewidth) node {\includegr
aphics[width=.5\linewidth]{/tmp/panel_A.pdf}};
\draw [anchor=north west] (.5\linewidth, .382\linewidth) node {\includegr
aphics[width=.5\linewidth]{/tmp/panel_B.pdf}};
\begin{scope}[font=\bf\sffamily\large]
\draw [anchor=west,fill=white] (.0\linewidth, .382\linewidth) node [above
 right=-3mm] {$\mathsf{A}$};
\draw [anchor=west,fill=white] (.53\linewidth, .382\linewidth) node [abov
e right=-3mm] {$\mathsf{B}$};
\end{scope}
```

In [35]:
```
!convert  -density {dpi_export} {fname}.pdf {fname}.jpg
!convert  -density {dpi_export} {fname}.pdf {fname}.png
#!convert  -density {dpi_export} -resize 5400  -units pixelsperinch -flat
ten  -compress lzw  -depth 8 {fname}.pdf {fname}.tiff
Image(fname +'.png')
```

Out[35]:



!echo "width=" ; convert {fname}.tiff -format "%[fx:w]" info: !echo ", \nheight=" ; convert {fname}.tiff -format "%[fx:h]" info: !echo ", \nunit=" ; convert {fname}.tiff -format "%U" info:!identify {fname}.tiff

figure 3:

learning

In [36]:
```
fname = 'figure_HAP'
```

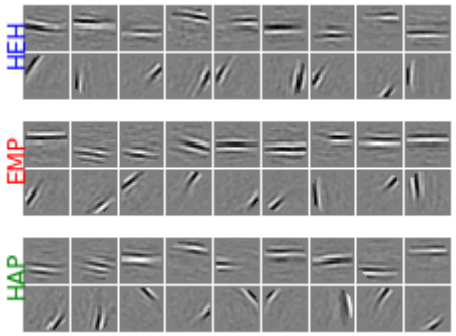
In [37]:
```
colors = ['orange', 'blue', 'red', 'green']
homeo_methods = ['OLS', 'HEH', 'EMP', 'HAP']
list_figures = []
dico = {}
for i_cv in range(N_cv):
    dico[i_cv] = {}
    for homeo_method in homeo_methods:
        shl = SHL(homeo_method=homeo_method, seed=seed+i_cv, **opts)
        dico[i_cv][homeo_method] = shl.learn_dico(data=data,
list_figures=list_figures, matname=tag + '_' + homeo_method + '_seed=' +
str(seed+i_cv))

list_figures = ['show_dico'] if DEBUG else []
for i_cv in [one_cv]:
    for homeo_method in homeo_methods:
        print(hl + hs + homeo_method + hs + hl)
        shl = SHL(homeo_method=homeo_method, seed=seed+i_cv, **opts)
        shl.learn_dico(data=data, list_figures=list_figures, matname=tag
+ '_' + homeo_method + '_seed=' + str(seed+i_cv))
        plt.show()
        print('size of dictionary = (number of filters, size of imagelet
s) = ', dico[i_cv][homeo_method].dictionary.shape)
        print('average of filters = ',  dico[i_cv][homeo_method].dictiona
ry.mean(axis=1).mean(),
              '+/-',  dico[i_cv][homeo_method].dictionary.mean(axis=1).st
d())
        SE = np.sqrt(np.sum(dico[i_cv][homeo_method].dictionary**2,
axis=1))
        print('average energy of filters = ', SE.mean(), '+/-', SE.std())
```

```
----------          OLS          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 48
4)
average of filters =  -5.903032155117969e-06 +/- 0.0005811583965811138
average energy of filters =  1.0 +/- 4.5478569413264154e-17
----------          HEH          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 48
4)
average of filters =  -4.325685413494333e-06 +/- 0.0005996641505749903
average energy of filters =  1.0 +/- 2.797499069501051e-17
----------          EMP          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 48
4)
average of filters =  7.555029151361008e-06 +/- 0.0006147188902722655
average energy of filters =  1.0 +/- 4.1962486042515756e-17
----------          HAP          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 48
4)
average of filters =  7.671372500200844e-06 +/- 0.0005961926488662842
average energy of filters =  1.0 +/- 3.9562611248144994e-17
```

panel A: plotting some dictionaries

In [38]:
```
pname = '/tmp/panel_A' #pname = fname + '_A'
```

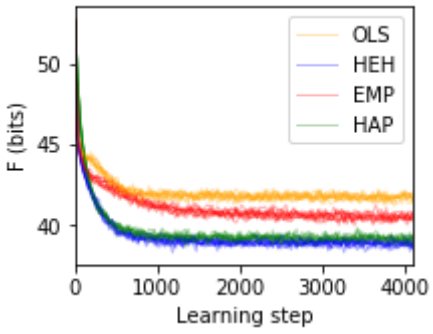
In [39]:
```
subplotpars = dict( left=0.042, right=1., bottom=0., top=1., wspace=0.05,
 hspace=0.05,)
fig, axs = plt.subplots(3, 1, figsize=(fig_width/2, fig_width/(1+phi)), g
ridspec_kw=subplotpars)

for ax, color, homeo_method in zip(axs.ravel(), colors[1:],
homeo_methods[1:]):
    ax.axis(c=color, lw=2, axisbg='w')
    ax.set_facecolor('w')
    from shl_scripts import show_dico
    fig, ax = show_dico(shl, dico[one_cv][homeo_method], data=data, dim_g
raph=dim_graph, fig=fig, ax=ax)
    # ax.set_ylabel(homeo_method)
    ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, r
otation=90)#, backgroundcolor='white'

for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
```



panel B: quantitative comparison

In [40]:
```
pname = '/tmp/panel_B' #fname + '_B'
```

In [41]:
```
from shl_scripts import time_plot
variable = 'F'
alpha = .3
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.0
5, hspace=0.05,)
fig, ax = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gr
idspec_kw=subplotpars)
for i_cv in range(N_cv):
    for color, homeo_method in zip(colors, homeo_methods):
        ax.axis(c='b', lw=2, axisbg='w')
        ax.set_facecolor('w')
        if i_cv==0:
            fig, ax = time_plot(shl, dico[i_cv][homeo_method], variable=v
ariable, unit='bits', color=color, label=homeo_method, alpha=alpha_0,
fig=fig, ax=ax)
        else:
            fig, ax = time_plot(shl, dico[i_cv][homeo_method], variable=v
ariable, unit='bits', color=color, alpha=alpha, fig=fig, ax=ax)
ax.legend(loc='best')
for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```



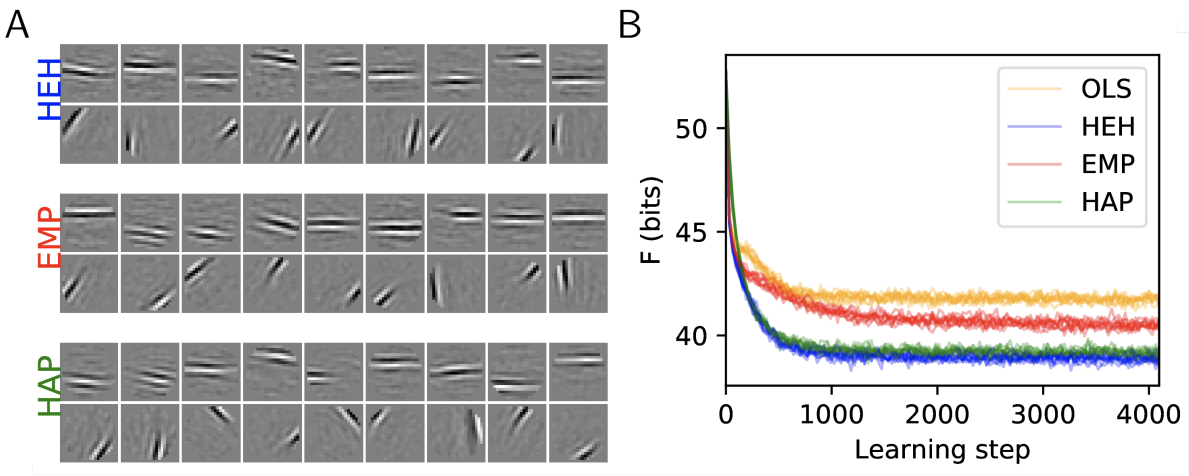
In [42]:
```
if DEBUG: Image(pname +'.png')
```

Montage of the subplots

In [43]:
```
%%tikz -f pdf --save {fname}.pdf
\draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\l
inewidth) ;
\draw [anchor=north west] (.0\linewidth, .382\linewidth) node {\includegr
aphics[width=.5\linewidth]{/tmp/panel_A}};
\draw [anchor=north west] (.5\linewidth, .382\linewidth) node {\includegr
aphics[width=.5\linewidth]{/tmp/panel_B}};
\begin{scope}[font=\bf\sffamily\large]
\draw [anchor=west,fill=white] (.0\linewidth, .382\linewidth) node [above
 right=-3mm] {$\mathsf{A}$};
\draw [anchor=west,fill=white] (.53\linewidth, .382\linewidth) node [abov
e right=-3mm] {$\mathsf{B}$};
\end{scope}
```

In [44]:
```
!convert  -density {dpi_export} {fname}.pdf {fname}.jpg
!convert  -density {dpi_export} {fname}.pdf {fname}.png
#!convert  -density {dpi_export} -resize 5400  -units pixelsperinch -flat
ten  -compress lzw  -depth 8 {fname}.pdf {fname}.tiff
Image(fname +'.png')
```

Out[44]:



!echo "width=" ; convert {fname}.tiff -format "%[fx:w]" info: !echo ", \nheight=" ; convert {fname}.tiff -format "%[fx:h]" info: !echo ", \nunit=" ; convert {fname}.tiff -format "%U" info:!identify {fname}.tiff

figure 4: Convolutional Neural Network

In [45]:
```
fname = 'figure_CNN'
```

In [46]:
```
!mkdir -p /tmp/database && rsync -a "/Users/laurentperrinet/science/VB_These/Rapport d'avancement/database/Face_DataBase/Raw_DataBase" /tmp/database/
```

In [47]:
```
from CHAMP.DataLoader import LoadData
from CHAMP.DataTools import LocalContrastNormalization, FilterInputData,
GenerateMask
from CHAMP.Monitor import DisplayDico, DisplayConvergenceCHAMP, DisplayWh
ere

import os
datapath = os.path.join("/tmp", "database")
path = os.path.join(datapath, "Raw_DataBase")
TrSet, TeSet = LoadData('Face', path, decorrelate=False, resize=(65, 65))

# MP Parameters
nb_dico = 20
width = 9
dico_size = (width, width)
l0 = 20
seed = 42
# Learning Parameters
eta = .05
nb_epoch = 500

TrSet, TeSet = LoadData('Face', path, decorrelate=False, resize=(65, 65))
N_TrSet, _, _, _ = LocalContrastNormalization(TrSet)
Filtered_L_TrSet = FilterInputData(
    N_TrSet, sigma=0.25, style='Custom', start_R=15)

mask = GenerateMask(full_size=(nb_dico, 1, width, width), sigma=0.8, styl
e='Gaussian')

from CHAMP.CHAMP_Layer import CHAMP_Layer

from CHAMP.DataTools import SaveNetwork, LoadNetwork
homeo_methods = ['None', 'HAP']

for homeo_method, eta_homeo  in zip(homeo_methods, [0., 0.0025]):
    ffname = 'cache_dir_CNN/CHAMP_low_' + homeo_method + '.pkl'
    try:
        L1_mask = LoadNetwork(loading_path=ffname)
    except:
        L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                              dico_size=dico_size, mask=mask, verbose=1)
        dico_mask = L1_mask.TrainLayer(
            Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo, nb_epoch=nb_e
poch, seed=seed)
        SaveNetwork(Network=L1_mask, saving_path=ffname)
```

panel A: plotting some dictionaries

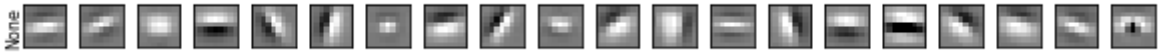
In [48]:
```
pname = '/tmp/panel_A' #pname = fname + '_A'
```

subplotpars = dict(left=0.042, right=1., bottom=0., top=1., wspace=0.05, hspace=0.05,) fig, axs = plt.subplots(2, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotpars) for ax, color, homeo_method in zip(axs.ravel(), ['black', 'green'], homeo_methods): ax.axis(c=color, lw=2, axisbg='w') ax.set_facecolor('w') ffname =

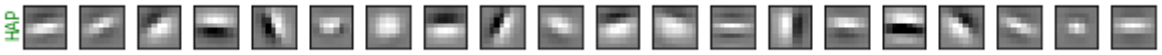
'cache_dir/CHAMP_low_' + homeo_method + '.pkl' L1_mask = LoadNetwork(loading_path=ffname) fig, ax = DisplayDico(L1_mask.dictionary, fig=fig, ax=ax) # ax.set_ylabel(homeo_method) ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, rotation=90)#, backgroundcolor='white' for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)

In [49]:
```
subplotpars = dict(left=0.042, right=1., bottom=0., top=1., wspace=0.05,
hspace=0.05,)

for color, homeo_method in zip(['black', 'green'], homeo_methods):
    #fig, axs = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+ph
i)), gridspec_kw=subplotpars)
    ffname = 'cache_dir_CNN/CHAMP_low_' + homeo_method + '.pkl'
    L1_mask = LoadNetwork(loading_path=ffname)
    fig, ax = DisplayDico(L1_mask.dictionary)
    # ax.set_ylabel(homeo_method)
    #for ax in list(axs):
    #    ax.axis(c=color, lw=2, axisbg='w')
    #    ax.set_facecolor('w')
    ax[0].text(-4, 3, homeo_method, fontsize=8, color=color,
rotation=90)#, backgroundcolor='white'
    plt.tight_layout( pad=0., w_pad=0., h_pad=.0)


    for ext in FORMATS: fig.savefig(pname + '_' + homeo_method + ext,
dpi=dpi_export)
```

```
<Figure size 576x28.8 with 0 Axes>
```




```
<Figure size 576x28.8 with 0 Axes>
```

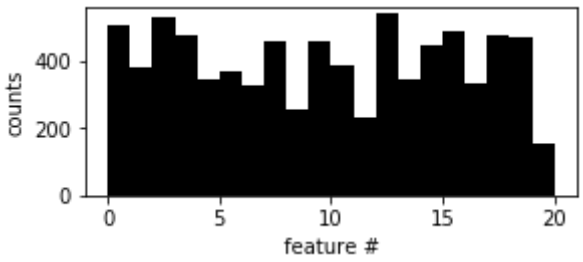


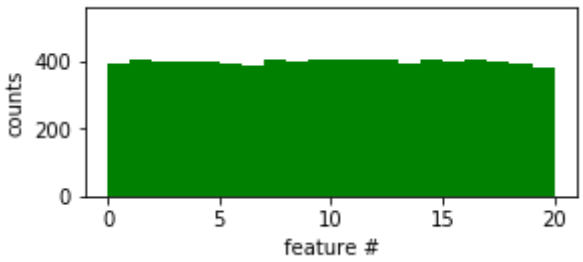
panel B: quantitative comparison

In [50]:
```
pname = '/tmp/panel_B' #fname + '_B'
```

from shl_scripts import time_plot variable = 'F' alpha = .3 subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.05, hspace=0.05,) fig, axs = plt.subplots(2, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotpars) for ax, color, homeo_method in zip(axs, ['black', 'green'], homeo_methods): print(ax, axs) ffname = 'cache_dir_CNN/CHAMP_low_' + homeo_method + '.pkl' L1_mask = LoadNetwork(loading_path=ffname) fig, ax = DisplayConvergenceCHAMP(L1_mask, to_display=['histo'], fig=fig, ax=ax) ax.axis(c=color, lw=2, axisbg='w') ax.set_facecolor('w') # ax.set_ylabel(homeo_method) #ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, rotation=90)#, backgroundcolor='white' for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export) if DEBUG: Image(pname +'.png')

In [51]:
```
from shl_scripts import time_plot
variable = 'F'
alpha = .3
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.05, hspace=0.05,)

for color, homeo_method in zip(['black', 'green'], homeo_methods):
    #fig, axs = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotpars)
    ffname = 'cache_dir_CNN/CHAMP_low_' + homeo_method + '.pkl'
    L1_mask = LoadNetwork(loading_path=ffname)
    fig, ax = DisplayConvergenceCHAMP(L1_mask, to_display=['histo'], color=color)
    ax.axis(c=color, lw=2, axisbg='w')
    ax.set_facecolor('w')
    ax.set_ylabel('counts')
    ax.set_xlabel('feature #')
    ax.set_ylim(0, 560)
    #ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, rotation=90)#, backgroundcolor='white'
    #ax[0].text(-8, 3, homeo_method, fontsize=12, color=color, rotation=90)#, backgroundcolor='white'

    for ext in FORMATS: fig.savefig(pname + '_' + homeo_method + ext, dpi=dpi_export)
    if DEBUG: Image(pname +'.png')
```





Montage of the subplots

In [52]:
```
%ls -ltr /tmp/panel_*
```

```
-rw-r--r--  1 laurentperrinet  wheel   77744 Dec 10 17:10 /tmp/panel_A.
pdf
-rw-r--r--  1 laurentperrinet  wheel   90735 Dec 10 17:10 /tmp/panel_A.
png
-rw-r--r--  1 laurentperrinet  wheel   47977 Dec 10 17:10 /tmp/panel_B.
pdf
-rw-r--r--  1 laurentperrinet  wheel  440130 Dec 10 17:10 /tmp/panel_B.
png
-rw-r--r--  1 laurentperrinet  wheel   27370 Dec 10 17:10 /tmp/panel_A_
None.pdf
-rw-r--r--  1 laurentperrinet  wheel   18909 Dec 10 17:10 /tmp/panel_A_
None.png
-rw-r--r--  1 laurentperrinet  wheel   26909 Dec 10 17:10 /tmp/panel_A_
HAP.pdf
-rw-r--r--  1 laurentperrinet  wheel   16431 Dec 10 17:10 /tmp/panel_A_
HAP.png
-rw-r--r--  1 laurentperrinet  wheel    8816 Dec 10 17:10 /tmp/panel_B_
None.pdf
-rw-r--r--  1 laurentperrinet  wheel   39035 Dec 10 17:10 /tmp/panel_B_
None.png
-rw-r--r--  1 laurentperrinet  wheel    8813 Dec 10 17:10 /tmp/panel_B_
HAP.pdf
-rw-r--r--  1 laurentperrinet  wheel   38743 Dec 10 17:10 /tmp/panel_B_
HAP.png
```

In [53]:
```
fname
```

Out[53]:
```
'figure_CNN'
```

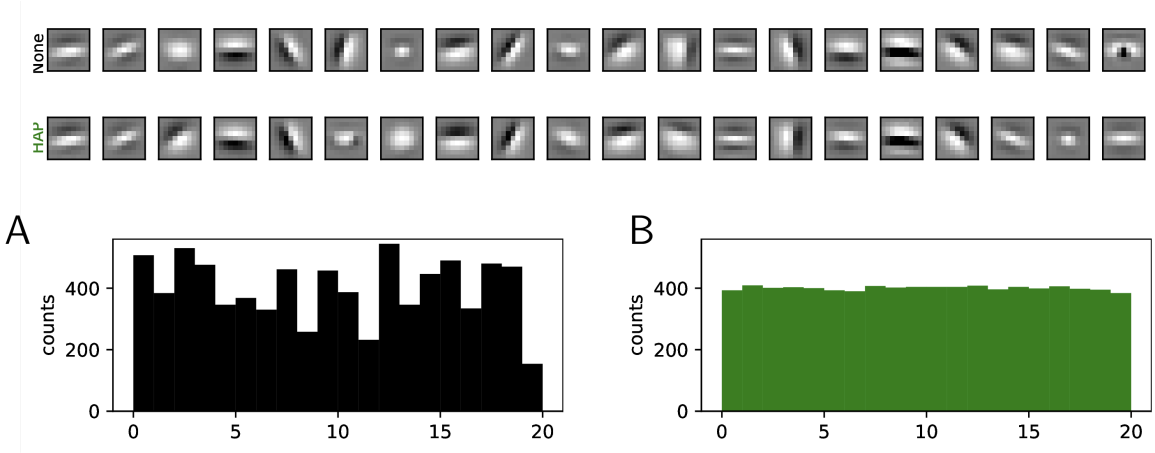
In [54]:
```
382+191
```

Out[54]:
```
573
```

In [55]:
```
%%tikz -f pdf --save {fname}.pdf
\draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\l
inewidth) ;
\draw [anchor=north west] (.0\linewidth, .375\linewidth) node {\includegr
aphics[width=.95\linewidth]{/tmp/panel_A_None}};
\draw [anchor=north west] (.0\linewidth, .300\linewidth) node {\includegr
aphics[width=.95\linewidth]{/tmp/panel_A_HAP}};
\draw [anchor=north west] (.0\linewidth, .191\linewidth) node {\includegr
aphics[width=.45\linewidth]{/tmp/panel_B_None}};
\draw [anchor=north west] (.5\linewidth, .191\linewidth) node {\includegr
aphics[width=.45\linewidth]{/tmp/panel_B_HAP}};
\begin{scope}[font=\bf\sffamily\large]
%\draw [anchor=west,fill=white] (.0\linewidth, .382\linewidth) node [abov
e right=-3mm] {$\mathsf{A}$};
\draw [anchor=west,fill=white] (.0\linewidth, .191\linewidth) node [above
 right=-3mm] {$\mathsf{A}$};
\draw [anchor=west,fill=white] (.53\linewidth, .191\linewidth) node [abov
e right=-3mm] {$\mathsf{B}$};
\end{scope}
```

In [56]:
```
!convert  -density {dpi_export} {fname}.pdf {fname}.jpg
!convert  -density {dpi_export} {fname}.pdf {fname}.png
#!convert  -density {dpi_export} -resize 5400  -units pixelsperinch -flat
ten  -compress lzw  -depth 8 {fname}.pdf {fname}.tiff
Image(fname +'.png')
```

Out[56]:



!echo "width=" ; convert {fname}.tiff -format "%[fx:w]" info: !echo ", \nheight=" ; convert {fname}.tiff -format "%[fx:h]" info: !echo ", \nunit=" ; convert {fname}.tiff -format "%U" info:!identify {fname}.tiff

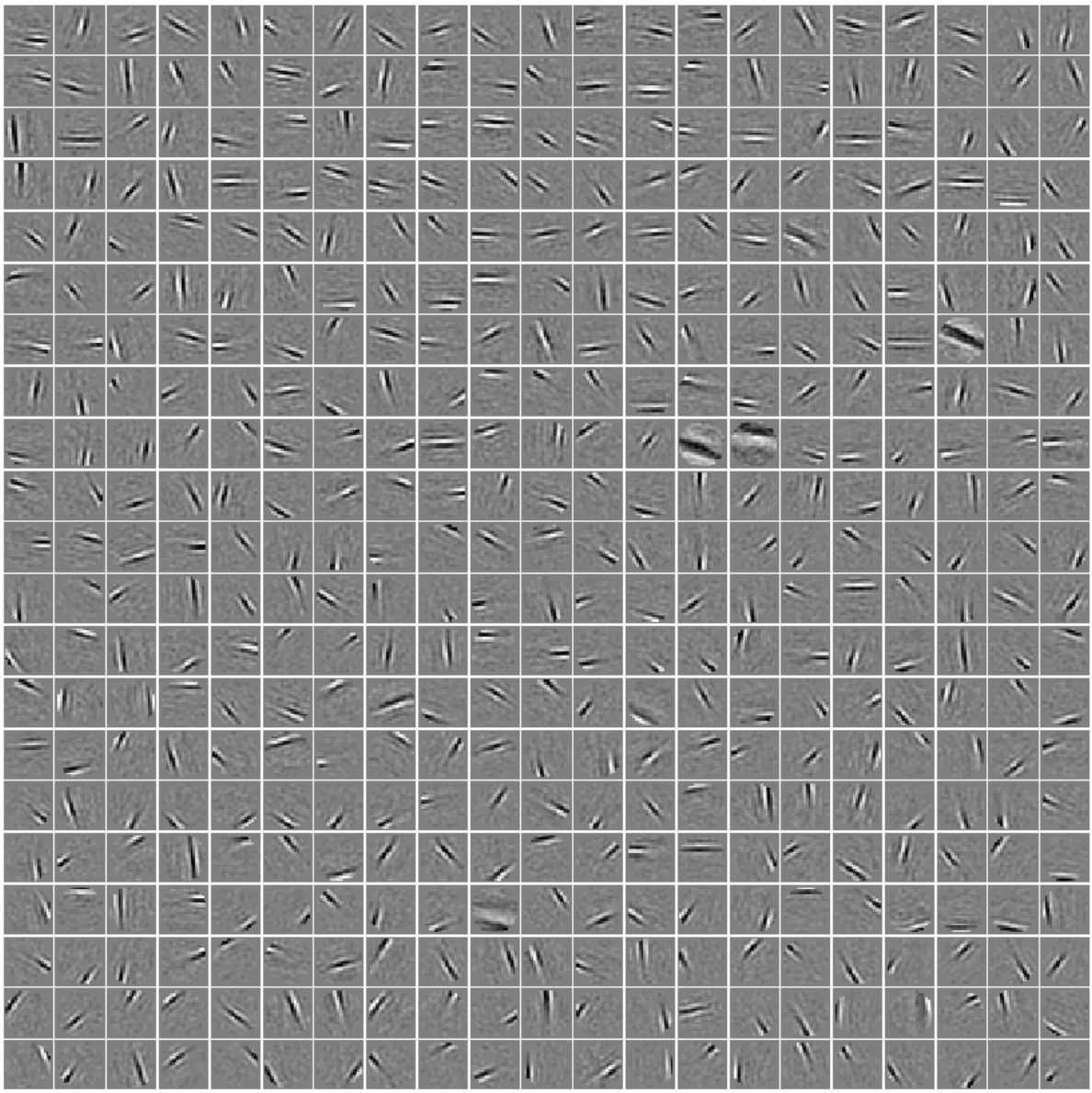
coding

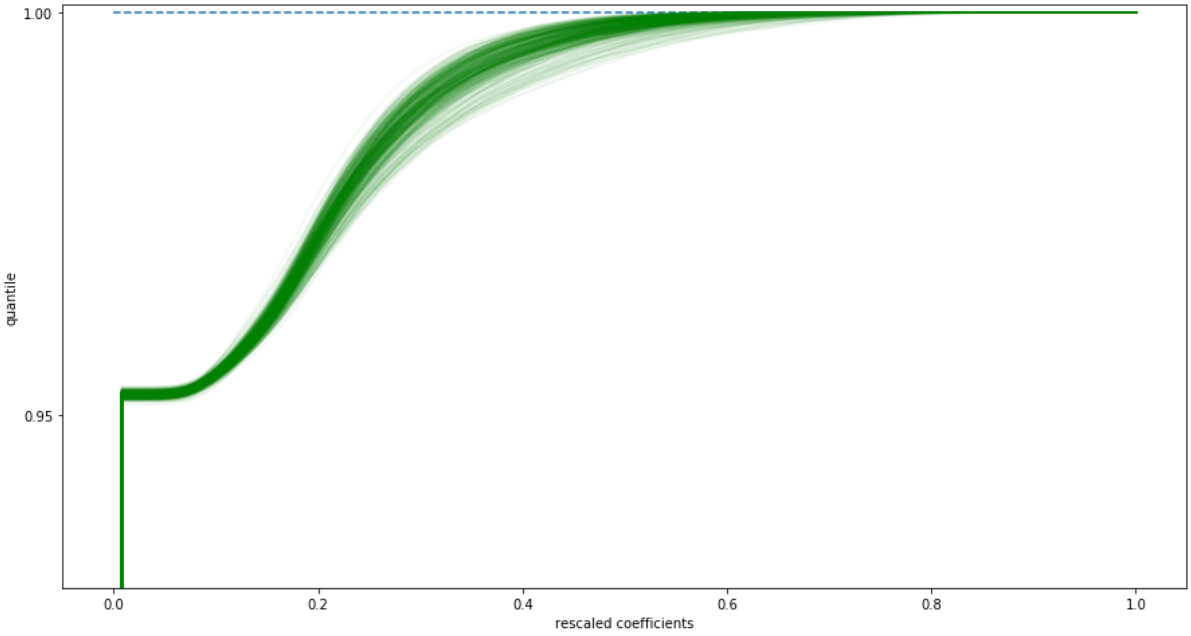
The learning itself is done via a gradient descent but is highly dependent on the coding / decoding algorithm. This belongs to a another function (in the shl_encode.py (https://github.com/bicv/SHL_scripts/blob/master/shl_scripts/shl_encode.py) script)

Supplementary controls

starting a learning

In [58]:
```
shl = SHL(**opts)
list_figures = ['show_dico', 'show_Pcum', 'time_plot_F']
dico = shl.learn_dico(data=data, list_figures=list_figures, matname=tag +
 '_vanilla')
```




1.00
0.95
quantile
0.0
0.2
0.4
0.6
0.8
1.0
rescaled coefficients

In [59]:
```
print('size of dictionary = (number of filters, size of imagelets) = ', d
ico.dictionary.shape)
print('average of filters = ',  dico.dictionary.mean(axis=1).mean(),
      '+/-',  dico.dictionary.mean(axis=1).std())
SE = np.sqrt(np.sum(dico.dictionary**2, axis=1))
print('average energy of filters = ', SE.mean(), '+/-', SE.std())
```

```
size of dictionary = (number of filters, size of imagelets) =  (441, 48
4)
average of filters =  1.293484038371286e-05 +/- 0.0007692820872169094
average energy of filters =  1.0 +/- 4.162811827527472e-17
```

getting help

In [60]:
```
help(shl)
```

```
Help on SHL in module shl_scripts.shl_experiments object:

class SHL(builtins.object)
 |  SHL(height=256, width=256, patch_width=22, N_patches=65536, datapat
h='../database/', name_database='kodakdb', do_mask=True, do_bandpass=Tr
ue, over_patches=16, patch_ds=1, n_dictionary=441, learning_algorithm
='mp', fit_tol=None, l0_sparseness=21, alpha_MP=0.9, one_over_F=True, n
_iter=4097, eta=0.01, beta1=0.9, beta2=0.999, epsilon=1e-08, do_precisi
on=False, eta_precision=0.0005, homeo_method='HAP', eta_homeo=0.01, alp
ha_homeo=2.5, C=3.0, nb_quant=128, P_cum=None, do_sym=False, seed=42, p
atch_norm=False, batch_size=1024, record_each=32, record_num_batches=10
24, n_image=None, DEBUG_DOWNSCALE=1, verbose=0, cache_dir='cache_dir')
 |
 |  Base class to define SHL experiments:
 |      - initialization
 |      - coding and learning
 |      - visualization
 |      - quantitative analysis
 |
 |  Methods defined here:
 |
 |  __init__(self, height=256, width=256, patch_width=22, N_patches=655
36, datapath='../database/', name_database='kodakdb', do_mask=True, do_
bandpass=True, over_patches=16, patch_ds=1, n_dictionary=441, learning_
algorithm='mp', fit_tol=None, l0_sparseness=21, alpha_MP=0.9, one_over_
F=True, n_iter=4097, eta=0.01, beta1=0.9, beta2=0.999, epsilon=1e-08, d
o_precision=False, eta_precision=0.0005, homeo_method='HAP', eta_homeo=
0.01, alpha_homeo=2.5, C=3.0, nb_quant=128, P_cum=None, do_sym=False, s
eed=42, patch_norm=False, batch_size=1024, record_each=32, record_num_b
atches=1024, n_image=None, DEBUG_DOWNSCALE=1, verbose=0, cache_dir='cac
he_dir')
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  code(self, data, dico, coding_algorithm='mp', matname=None, P_cum=N
one, fit_tol=None, l0_sparseness=None, gain=None)
 |
 |  decode(self, sparse_code, dico)
 |
 |  get_data(self, matname=None, patch_width=None)
 |
 |  learn_dico(self, dictionary=None, precision=None, P_cum=None, data=
None, matname=None, record_each=None, folder_exp=None, list_figures=[],
fig_kwargs={'fig': None, 'ax': None})
 |
 |  plot_error(self, dico, **fig_kwargs)
 |
 |  plot_variance(self, sparse_code, **fig_kwargs)
 |
 |  plot_variance_histogram(self, sparse_code, **fig_kwargs)
 |
 |  show_Pcum(self, dico, title=None, verbose=False, n_yticks=21, alpha
=0.05, c='g', **fig_kwargs)
 |
 |  show_dico(self, dico, data=None, title=None, **fig_kwargs)
 |
 |  show_dico_in_order(self, dico, data=None, title=None, **fig_kwargs)
 |
```

```
 |  time_plot(self, dico, variable='kurt', N_nosample=1, **fig_kwargs)
 |
 |  ----------------------------------------------------------------------
---
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [61]:
```
help(dico)
```

```
Help on SparseHebbianLearning in module shl_scripts.shl_learn object:

class SparseHebbianLearning(builtins.object)
 |  SparseHebbianLearning(fit_algorithm, dictionary=None, precision=Non
e, eta=0.003, beta1=0.9, beta2=0.999, epsilon=1e-08, homeo_method='HE
H', eta_homeo=0.05, alpha_homeo=0.0, C=5.0, nb_quant=256, P_cum=None, n
_dictionary=None, n_iter=10000, batch_size=32, l0_sparseness=None, fit_
tol=None, alpha_MP=1.0, do_precision=False, eta_precision=0.01, do_sym=
False, record_each=200, record_num_batches=4096, verbose=False, one_ove
r_F=True)
 |
 |  Sparse Hebbian learning
 |
 |  Finds a dictionary (a set of atoms) that can best be used to repres
ent data
 |  using a sparse code.
 |
 |  Parameters
 |  ----------
 |
 |  n_dictionary : int,
 |      Number of dictionary elements to extract
 |
 |  eta : float or dict
 |      Gives the learning parameter for the homeostatic gain.
 |
 |  n_iter : int,
 |      total number of iterations to perform
 |
 |  eta_homeo : float
 |      Gives the learning parameter for the homeostatic gain.
 |
 |  alpha_homeo : float
 |      Gives the smoothing exponent  for the homeostatic gain
 |      If equal to 1 the homeostatic learning rule learns a linear rel
ation to
 |      variance.
 |
 |  dictionary : array of shape (n_dictionary, n_pixels),
 |      initial value of the dictionary for warm restart scenarios
 |      Use ``None`` for a new learning.
 |
 |  fit_algorithm : {'mp', 'lars', 'cd'}
 |      see sparse_encode
 |
 |  batch_size : int,
 |      The number of samples to take in each batch.
 |
 |  l0_sparseness : int, ``0.1 * n_pixels`` by default
 |      Number of nonzero coefficients to target in each column of the
 |      solution. This is only used by `algorithm='lars'`, `algorithm
='mp'`  and
 |      `algorithm='omp'`.
 |
 |  fit_tol : float, 1. by default
 |      If `algorithm='lasso_lars'` or `algorithm='lasso_cd'`, `fit_tol
` is the
```

```
 |      penalty applied to the L1 norm.
 |      If `algorithm='threshold'`, `fit_tol` is the absolute value of
 the
 |      threshold below which coefficients will be squashed to zero.
 |      If `algorithm='mp'` or `algorithm='omp'`, `fit_tol` is the tole
rance
 |      parameter: the value of the reconstruction error targeted. In t
his case,
 |      it overrides `l0_sparseness`.
 |
 |  verbose :
 |      degree of verbosity of the printed output
 |
 |  Attributes
 |  ----------
 |  dictionary : array, [n_dictionary, n_pixels]
 |      dictionary extracted from the data
 |
 |
 |  Notes
 |  -----
 |  **References:**
 |
 |  Olshausen BA, Field DJ (1996).
 |  Emergence of simple-cell receptive field properties by learning a s
parse code for natural images.
 |  Nature, 381: 607-609. (http://redwood.berkeley.edu/bruno/papers/nat
ure-paper.pdf)
 |
 |  Olshausen BA, Field DJ (1997)
 |  Sparse Coding with an Overcomplete Basis Set: A Strategy Employed b
y V1?
 |  Vision Research, 37: 3311-3325.   (http://redwood.berkeley.edu/brun
o/papers/VR.pdf)
 |
 |  See also
 |  --------
 |  http://scikit-learn.org/stable/auto_examples/decomposition/plot_ima
ge_denoising.html
 |
 |  Methods defined here:
 |
 |  __init__(self, fit_algorithm, dictionary=None, precision=None, eta=
0.003, beta1=0.9, beta2=0.999, epsilon=1e-08, homeo_method='HEH', eta_h
omeo=0.05, alpha_homeo=0.0, C=5.0, nb_quant=256, P_cum=None, n_dictiona
ry=None, n_iter=10000, batch_size=32, l0_sparseness=None, fit_tol=None,
alpha_MP=1.0, do_precision=False, eta_precision=0.01, do_sym=False, rec
ord_each=200, record_num_batches=4096, verbose=False, one_over_F=True)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y=None)
 |      Fit the model from data in X.
 |
 |      Parameters
 |      ----------
 |      X: array-like, shape (n_samples, n_pixels)
 |          Training vector, where n_samples in the number of samples
```

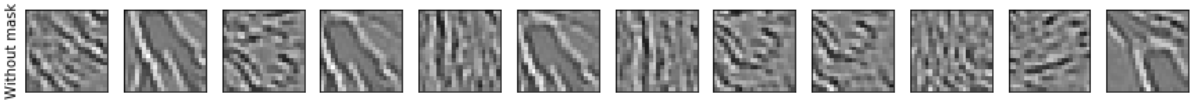
```
 |          and n_pixels is the number of features.
 |
 |      Returns
 |      -------
 |      self : object
 |          Returns the instance itself.
 |
 |  transform(self, X, algorithm=None, l0_sparseness=None, fit_tol=Non
e, alpha_MP=None)
 |      Fit the model from data in X.
 |
 |      Parameters
 |      ----------
 |      X: array-like, shape (n_samples, n_pixels)
 |          Training vector, where n_samples in the number of samples
 |          and n_pixels is the number of features.
 |
 |      Returns
 |      -------
 |      self : object
 |          Returns sparse code.
 |
 |  ----------------------------------------------------------------------
---
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
---
 |  Data and other attributes defined here:
 |
 |  __slotnames__ = []
```

loading a database

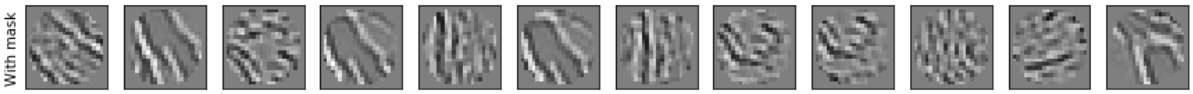
Loading patches, with or without mask:

In [62]:
```
N_patches = 12
from shl_scripts.shl_tools import show_data
opts_ = opts.copy()
opts_.update(verbose=0)
for i, (do_mask, label) in enumerate(zip([False, True], ['Without mask',
'With mask'])):
    data_ = SHL(DEBUG_DOWNSCALE=1, N_patches=N_patches, n_image=1, do_mas
k=do_mask, seed=seed, **opts_).get_data()
    fig, axs = show_data(data_)
    axs[0].set_ylabel(label);
    plt.show()
```

```
<Figure size 1080x216 with 0 Axes>
```

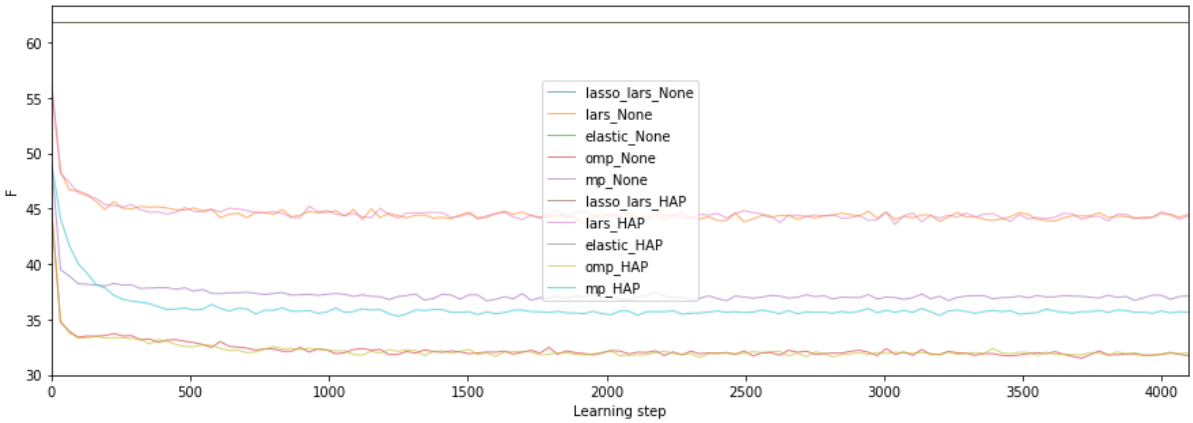



```
<Figure size 1080x216 with 0 Axes>
```



Testing different algorithms

In [63]:
```
fig, ax = None, None

for homeo_method in ['None', 'HAP']:
    for algorithm in ['lasso_lars', 'lars', 'elastic', 'omp', 'mp']: # 't
hreshold', 'lasso_cd',
        opts_ = opts.copy()
        opts_.update(homeo_method=homeo_method, learning_algorithm=algori
thm, verbose=0)
        shl = SHL(**opts_)
        dico= shl.learn_dico(data=data, list_figures=[],
                             matname=tag + ' - algorithm={}'.format(algorithm)
+ ' - homeo_method={}'.format(homeo_method))
        fig, ax = shl.time_plot(dico, variable='F', fig=fig, ax=ax,
label=algorithm +'_' + homeo_method)

    ax.legend()
```

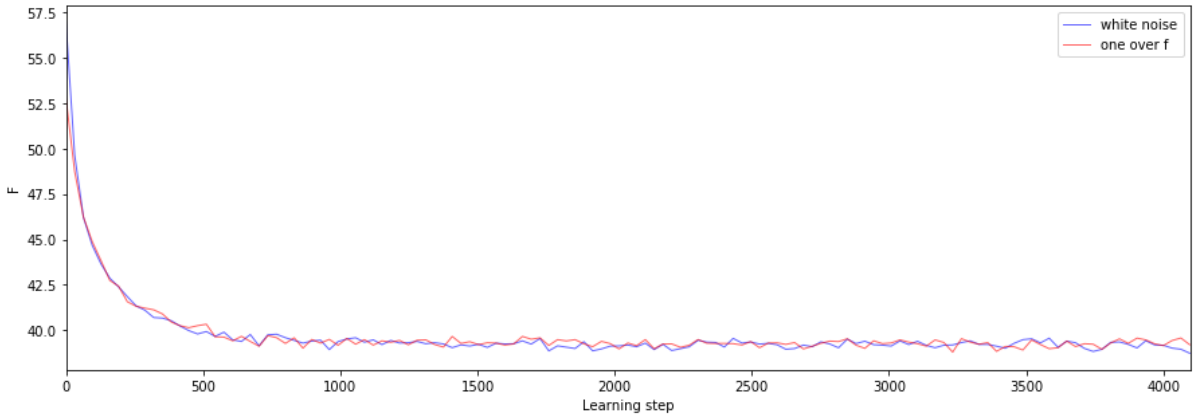


Testing two different dictionary initalization strategies

White Noise Initialization + Learning

In [64]:
```
shl = SHL(one_over_F=False, **opts)
dico_w = shl.learn_dico(data=data, matname=tag + '_WHITE', list_figures=
[])
shl = SHL(one_over_F=True, **opts)
dico_1oF = shl.learn_dico(data=data, matname=tag + '_OVF', list_figures=
[])
fig_error, ax_error = None, None
fig_error, ax_error = shl.time_plot(dico_w, variable='F', fig=fig_error,
ax=ax_error, color='blue', label='white noise')
fig_error, ax_error = shl.time_plot(dico_1oF, variable='F',
fig=fig_error, ax=ax_error, color='red', label='one over f')
#ax_error.set_ylim((0, .65))
ax_error.legend(loc='best')
```

Out[64]: <matplotlib.legend.Legend at 0x128f13cc0>

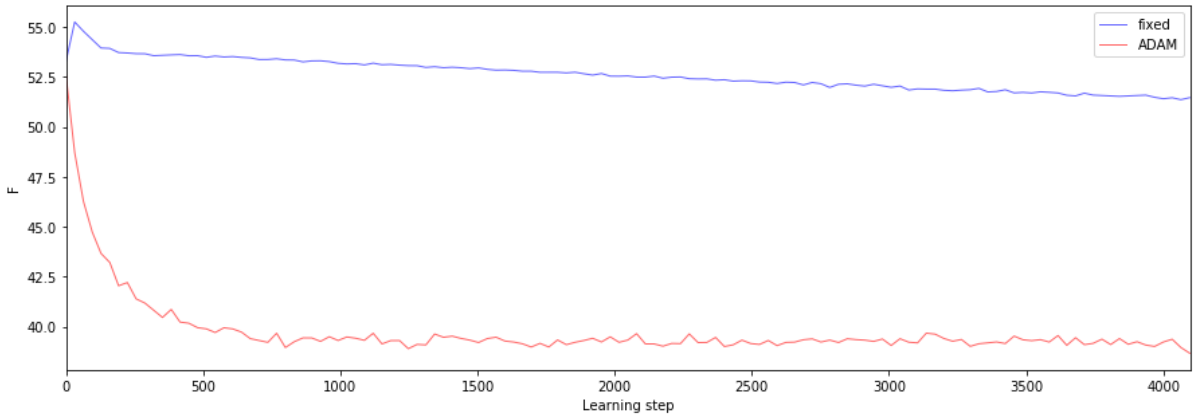


Testing two different learning rates strategies

We use by defaut the strategy of ADAM, see https://arxiv.org/pdf/1412.6980.pdf (https://arxiv.org/pdf/1412.6980.pdf)

In [ ]:
```
shl = SHL(beta1=0., **opts)
dico_fixed = shl.learn_dico(data=data, matname=tag + '_fixed', list_figur
es=[])
shl = SHL(**opts)
dico_default = shl.learn_dico(data=data, matname=tag + '_default', list_f
igures=[])
fig_error, ax_error = None, None
fig_error, ax_error = shl.time_plot(dico_fixed, variable='F', fig=fig_err
or, ax=ax_error, color='blue', label='fixed')
fig_error, ax_error = shl.time_plot(dico_default, variable='F', fig=fig_e
rror, ax=ax_error, color='red', label='ADAM')
#ax_error.set_ylim((0, .65))
ax_error.legend(loc='best')
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x128f34668>
```

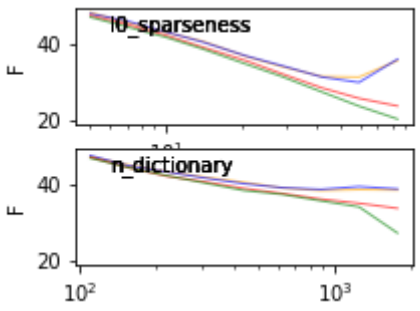


Testing different number of neurons and sparsity

As suggested by AnonReviewer3, we have tested how the convergence was modified by changing the number of neurons. By comparing different numbers of neurons we could re-draw the same figures for the convergence of the algorithm as in our original figures. In addition, we have also checked that this result will hold on a range of sparsity levels. In particular, we found that in general, increasing the `l0_sparseness` parameter, the convergence took progressively longer. Importantly, we could see that in both cases, this did not depend on the kind of homeostasis heuristic chosen, proving the generality of our results.

This is shown in the supplementary material that we have added to our revision ("Testing different number of neurons and sparsity") . This useful extension proves the originality of our work as highlighted in point 4, and the generality of these results compared to the parameters of the network.

In [ ]:
```
from shl_scripts.shl_experiments import SHL_set
homeo_methods = ['None', 'OLS', 'HEH']
homeo_methods = ['None', 'EMP', 'HAP', 'HEH', 'OLS']

variables = ['l0_sparseness', 'n_dictionary']
list_figures = []

#n_dictionary=21**2

for homeo_method in homeo_methods:
    opts_ = opts.copy()
    opts_.update(homeo_method=homeo_method, datapath=datapath)
    experiments = SHL_set(opts_, tag=tag + '_' + homeo_method)
    experiments.run(variables=variables, n_jobs=1, verbose=0)

fig, axs = plt.subplots(len(variables), 1, figsize=(fig_width/2, fig_widt
h/(1+phi)), gridspec_kw=subplotpars, sharey=True)

for i_ax, variable in enumerate(variables):
    for color, homeo_method in zip(colors, homeo_methods):
        opts_ = opts.copy()
        opts_.update(homeo_method=homeo_method, datapath=datapath)
        experiments = SHL_set(opts_, tag=tag + '_' + homeo_method)
        fig, axs[i_ax] = experiments.scan(variable=variable,
list_figures=[], display='final', fig=fig, ax=axs[i_ax], color=color, dis
play_variable='F', verbose=0) #, label=homeo_metho
        axs[i_ax].set_xlabel('') #variable
        axs[i_ax].text(.1, .8,  variable, transform=axs[i_ax].transAxes)
        #axs[i_ax].get_xaxis().set_major_formatter(matplotlib.ticker.Scal
arFormatter())
```



Perspectives

Convolutional neural networks

In [ ]:
```
from CHAMP.DataLoader import LoadData
from CHAMP.DataTools import LocalContrastNormalization, FilterInputData,
GenerateMask
from CHAMP.Monitor import DisplayDico, DisplayConvergenceCHAMP, DisplayWh
ere

import os
home = os.getenv('HOME')
datapath = os.path.join("/tmp", "database")
path = os.path.join(datapath, "Raw_DataBase")
TrSet, TeSet = LoadData('Face', path, decorrelate=False, resize=(65, 65))
to_display = TrSet[0][0, 0:10, :, :, :]
print('Size=', TrSet[0].shape)
DisplayDico(to_display)
```

```
Size= torch.Size([1, 400, 1, 65, 65])
```

Out[ ]:
```
(<Figure size 576x57.6 with 10 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x1217803c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12b994d30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x121592f60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1236b81d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1236cc400>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12ecab630>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123d9a860>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123dc2a58>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123deccc0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123e15ef0>],
       dtype=object))

<Figure size 576x57.6 with 0 Axes>
```

Training on a face database

In [ ]:
```
# MP Parameters
nb_dico = 20
width = 9
dico_size = (width, width)
l0 = 20
seed = 42
# Learning Parameters
eta = .05
nb_epoch = 500

TrSet, TeSet = LoadData('Face', path, decorrelate=False, resize=(65, 65))
N_TrSet, _, _, _ = LocalContrastNormalization(TrSet)
Filtered_L_TrSet = FilterInputData(
    N_TrSet, sigma=0.25, style='Custom', start_R=15)
to_display = Filtered_L_TrSet[0][0, 0:10, :, :, :]
DisplayDico(to_display)

mask = GenerateMask(full_size=(nb_dico, 1, width, width), sigma=0.8, styl
e='Gaussian')
DisplayDico(mask)
```

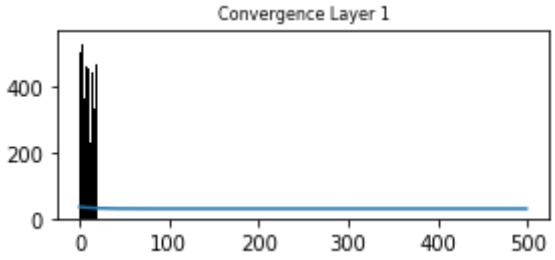
Out[ ]:
```
(<Figure size 576x28.8 with 20 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x123ee19b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123e8ff98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1268c5ef0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1214a9ac8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123cbb080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123ea75f8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123949b70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1239770f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1242396a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123d63c18>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123d421d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123d0a748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1244efcc0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12427a278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123bfb7f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123be8d68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12371d320>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x123736898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12374ae10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1238a83c8>],
       dtype=object))

<Figure size 576x57.6 with 0 Axes>
```

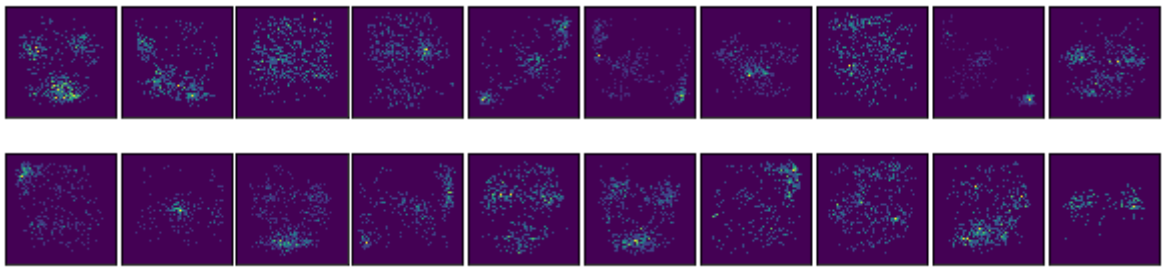
```
<Figure size 576x28.8 with 0 Axes>
```

Training the ConvMP Layer with homeostasis

In [ ]:
```
from CHAMP.CHAMP_Layer import CHAMP_Layer

from CHAMP.DataTools import SaveNetwork, LoadNetwork
fname = 'cache_dir_CNN/CHAMP_low_None.pkl'
try:
    L1_mask = LoadNetwork(loading_path=fname)
except:
    L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                          dico_size=dico_size, mask=mask, verbose=2)
    dico_mask = L1_mask.TrainLayer(
        Filtered_L_TrSet, eta=eta, nb_epoch=nb_epoch, seed=seed)
    SaveNetwork(Network=L1_mask, saving_path=fname)

DisplayDico(L1_mask.dictionary)
DisplayConvergenceCHAMP(L1_mask, to_display=['error', 'histo'])
DisplayWhere(L1_mask.where)
```

Out[ ]:
```
(<Figure size 576x216 with 20 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x1286d2208>)

<Figure size 576x28.8 with 0 Axes>
```

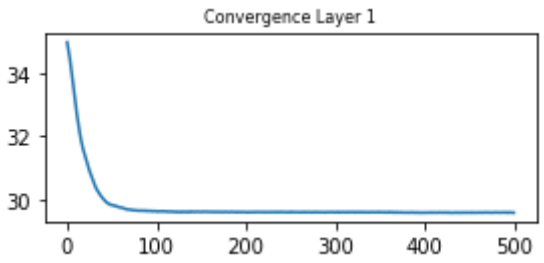


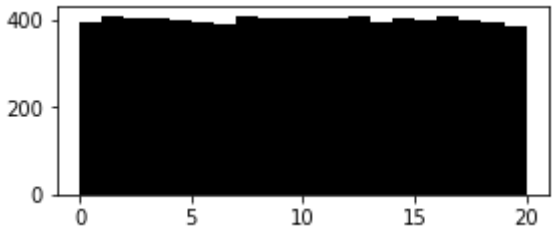


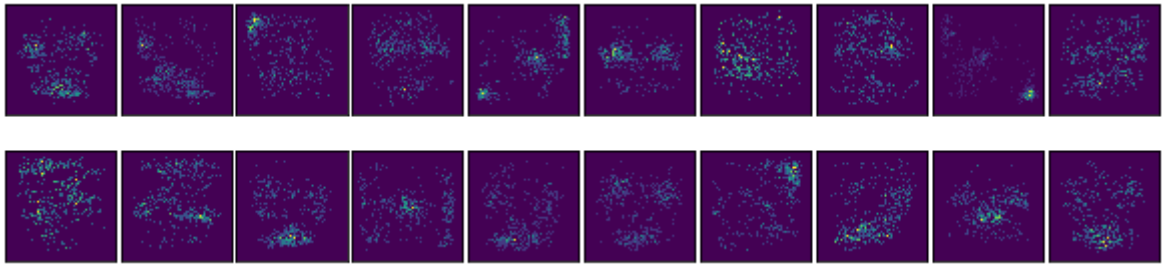
Training the ConvMP Layer with homeostasis

In [ ]:
```
fname = 'cache_dir_CNN/CHAMP_low_HAP.pkl'
try:
    L1_mask = LoadNetwork(loading_path=fname)
except:

    # Learning Parameters
    eta_homeo = 0.0025
    L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                          dico_size=dico_size, mask=mask, verbose=1)
    dico_mask = L1_mask.TrainLayer(
        Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo,
nb_epoch=nb_epoch, seed=seed)
    SaveNetwork(Network=L1_mask, saving_path=fname)

DisplayDico(L1_mask.dictionary)
DisplayConvergenceCHAMP(L1_mask, to_display=['error'])
DisplayConvergenceCHAMP(L1_mask, to_display=['histo'])
DisplayWhere(L1_mask.where)
```

Out[ ]:
```
(<Figure size 576x216 with 20 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x1271425f8>)

<Figure size 576x28.8 with 0 Axes>
```







Reconstructing the input image

In [ ]:
```
from CHAMP.DataTools import Rebuilt
import torch
rebuilt_image = Rebuilt(torch.FloatTensor(L1_mask.code), L1_mask.dictiona
ry)
DisplayDico(rebuilt_image[0:10, :, :, :]);
```

```
<Figure size 576x57.6 with 0 Axes>
```

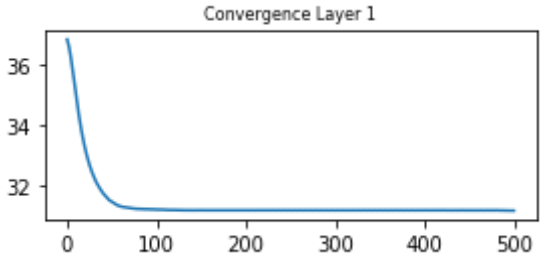
Training the ConvMP Layer with higher-level filters

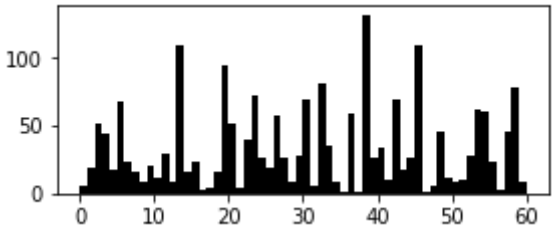
We train higher-level feature vectors by forcing the network to :

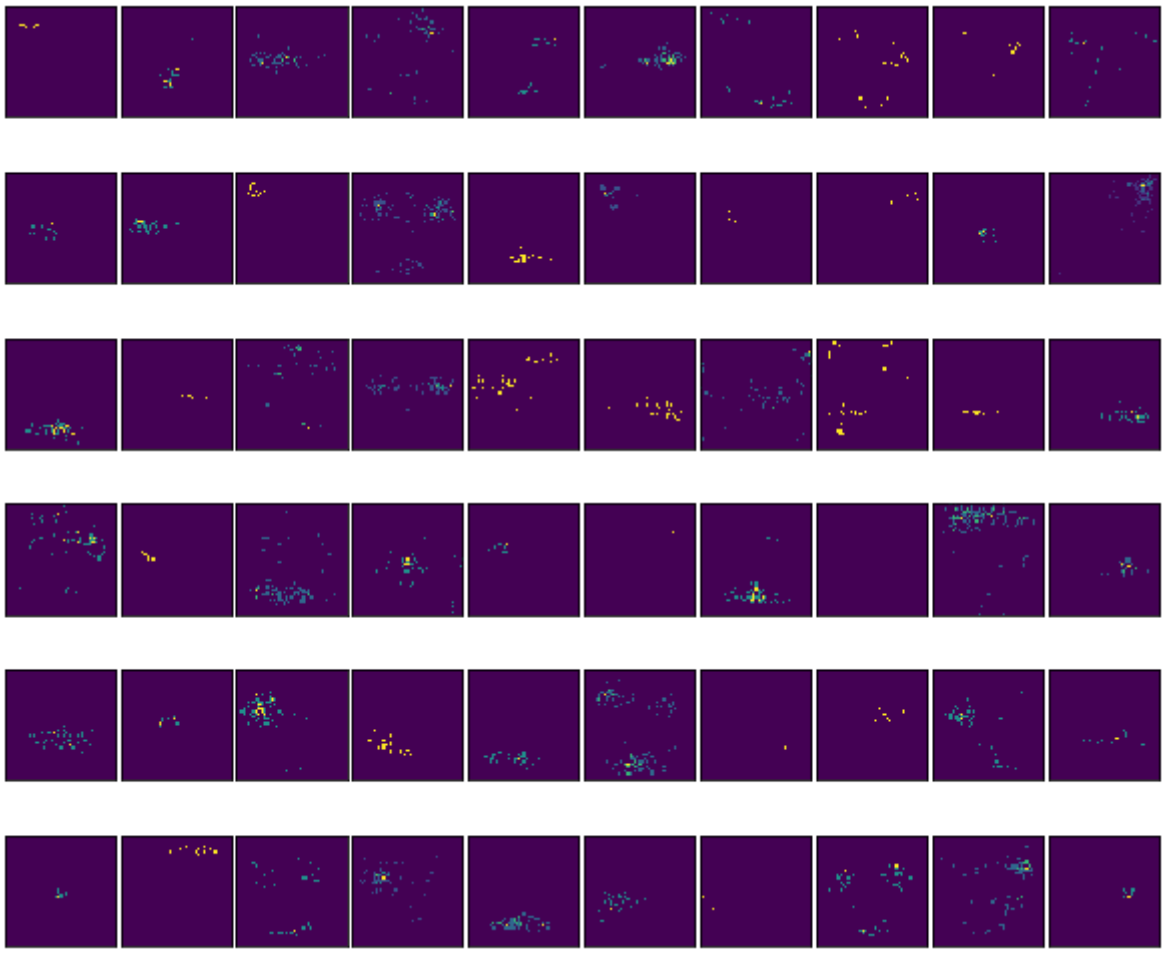
- learn bigger filters,
- represent the information using a bigger dictionary (higher sparseness)
- represent the information with less features (higher sparseness)

In [ ]:
```
fname = 'cache_dir_CNN/CHAMP_high_None.pkl'
try:
    L1_mask = LoadNetwork(loading_path=fname)
except:

    nb_dico = 60
    width = 19
    dico_size = (width, width)
    l0 = 5
    mask = GenerateMask(full_size=(nb_dico, 1, width, width), sigma=0.8,
style='Gaussian')
    # Learning Parameters
    eta_homeo = 0.0
    eta = .05
    nb_epoch = 500
    # learn
    L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                          dico_size=dico_size, mask=mask, verbose=0)
    dico_mask = L1_mask.TrainLayer(
        Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo,
nb_epoch=nb_epoch, seed=seed)
    SaveNetwork(Network=L1_mask, saving_path=fname)


DisplayDico(L1_mask.dictionary)
DisplayConvergenceCHAMP(L1_mask, to_display=['error'])
DisplayConvergenceCHAMP(L1_mask, to_display=['histo'])
DisplayWhere(L1_mask.where);
```

```
<Figure size 576x9.6 with 0 Axes>
```







In [ ]:
```
fname = 'cache_dir_CNN/CHAMP_high_HAP.pkl'
try:
    L1_mask = LoadNetwork(loading_path=fname)
except:

    nb_dico = 60
    width = 19
    dico_size = (width, width)
    l0 = 5
    mask = GenerateMask(full_size=(nb_dico, 1, width, width), sigma=0.8,
style='Gaussian')
    # Learning Parameters
    eta_homeo = 0.0025
    eta = .05
    nb_epoch = 500
    # learn
    L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                          dico_size=dico_size, mask=mask, verbose=0)
    dico_mask = L1_mask.TrainLayer(
        Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo,
nb_epoch=nb_epoch, seed=seed)
    SaveNetwork(Network=L1_mask, saving_path=fname)

DisplayDico(L1_mask.dictionary)
DisplayConvergenceCHAMP(L1_mask, to_display=['error'])
DisplayConvergenceCHAMP(L1_mask, to_display=['histo'])
DisplayWhere(L1_mask.where);
```

Training on MNIST database

fname = 'cache_dir_CNN/CHAMP_MNIST_HAP.pkl' try: L1_mask = LoadNetwork(loading_path=fname) except: path = os.path.join(datapath, "MNISTtorch") TrSet, TeSet = LoadData('MNIST', data_path=path) N_TrSet, _, _, _ = LocalContrastNormalization(TrSet) Filtered_L_TrSet = FilterInputData(N_TrSet, sigma=0.25, style='Custom', start_R=15) nb_dico = 60 width = 7 dico_size = (width, width) l0 = 15 # Learning Parameters eta_homeo = 0.0025 eta = .05 nb_epoch = 500 # learn L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico, dico_size=dico_size, mask=mask, verbose=2) dico_mask = L1_mask.TrainLayer(Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo, nb_epoch=nb_epoch, seed=seed) SaveNetwork(Network=L1_mask, saving_path=fname) DisplayDico(L1_mask.dictionary) DisplayConvergenceCHAMP(L1_mask, to_display=['error']) DisplayConvergenceCHAMP(L1_mask, to_display=['histo']) DisplayWhere(L1_mask.where);

Computational details

caching simulation data

A convenience script to run and cache most learning items in this notebooks:

In [ ]:
```
!ls -l {shl.cache_dir}/{tag}*
#!rm {shl.cache_dir}/{tag}*lock*
#!rm {shl.cache_dir}/{tag}*
#!ls -l {shl.cache_dir}/{tag}*
```

In [ ]:
```
%%writefile model.py
#!/usr/bin/env python3
# -*- coding: utf-8 -*
tag = 'ICLR'
from shl_scripts.shl_experiments import SHL, prun
# pre-loading data
datapath = '../../SparseHebbianLearning/database'
# different runs
#opts = dict(cache_dir='cache_dir_ICLR', datapath=datapath, verbose=0)
#opts = dict(cache_dir='cache_dir_cluster', datapath=datapath, verbose=0)
opts = dict(datapath=datapath, verbose=0)

shl = SHL(**opts)
data = shl.get_data(matname=tag)

# running main simulations
# Figure 1 & 3
N_cv = 10
homeo_methods = ['None', 'OLS', 'HEH', 'HAP', 'EMP']
seed = 42

# running in parallel on a multi-core machine
import sys
try:
    n_jobs = int(sys.argv[1])
    print('n_jobs=', n_jobs)
except:
    n_jobs = 4
    n_jobs = 9
    n_jobs = 10
    n_jobs = 1
    n_jobs = 35


if n_jobs>0:
    # Figure 1 & 3

    list_figures = []

    from shl_scripts.shl_experiments import SHL_set
    for homeo_method in homeo_methods:
        opts_ = opts.copy()
        opts_.update(homeo_method=homeo_method)
        experiments = SHL_set(opts_, tag=tag + '_' + homeo_method,
N_scan=N_cv)
        experiments.run(variables=['seed'], n_jobs=n_jobs, verbose=0)

    # Figure 2-B
    variables = ['eta', 'eta_homeo']

    list_figures = []
```

```
    for homeo_method in homeo_methods:
        opts_ = opts.copy()
        opts_.update(homeo_method=homeo_method)
        experiments = SHL_set(opts_, tag=tag + '_' + homeo_method, base=1
0)
        experiments.run(variables=variables, n_jobs=n_jobs, verbose=0)



    # Annex X.X

    shl = SHL(**opts)
    dico = shl.learn_dico(data=data, list_figures=list_figures, matname=t
ag + '_vanilla')


    variables = ['alpha_homeo', 'l0_sparseness', 'n_dictionary']
    bases = [4] * len(variables)

    for homeo_method, base in zip(homeo_methods, bases):
        opts_ = opts.copy()
        opts_.update(homeo_method=homeo_method)
        experiments = SHL_set(opts_, tag=tag + '_' + homeo_method, base=b
ase)
        experiments.run(variables=variables, n_jobs=n_jobs, verbose=0)

    for algorithm in ['lasso_lars', 'lasso_cd', 'lars', 'elastic', 'omp',
 'mp']: # 'threshold',
        opts_ = opts.copy()
        opts_.update(homeo_method='None', learning_algorithm=algorithm, v
erbose=0)
        shl = SHL(**opts_)
        dico= shl.learn_dico(data=data, list_figures=[],
                             matname=tag + ' - algorithm={}'.format(algorithm))

    for homeo_method in ['None', 'HAP']:
        for algorithm in ['lasso_lars', 'lars', 'elastic', 'omp', 'mp']:
# 'threshold', 'lasso_cd',
            opts_ = opts.copy()
            opts_.update(homeo_method=homeo_method, learning_algorithm=al
gorithm, verbose=0)
            shl = SHL(**opts_)
            dico= shl.learn_dico(data=data, list_figures=[],
                                 matname=tag + ' - algorithm={}'.format(algorit
hm) + ' - homeo_method={}'.format(homeo_method))

    shl = SHL(one_over_F=False, **opts)
    dico_w = shl.learn_dico(data=data, matname=tag + '_WHITE', list_figur
es=[])
    shl = SHL(one_over_F=True, **opts)
    dico_1oF = shl.learn_dico(data=data, matname=tag + '_OVF', list_figur
es=[])

    shl = SHL(beta1=0., **opts)
    dico_fixed = shl.learn_dico(data=data, matname=tag + '_fixed', list_f
igures=[])
    shl = SHL(**opts)
```