Procedure for a single-stage system

1. Pre-process images
   - remove mean, high-pass filter, normalize contrast

2. Train encoder-decoder on 9x9 image patches

3. use the filters in a recognition architecture
   - Apply the filters to the whole image
   - Apply the tanh and D scaling
   - Add more non-linearities (rectification, normalization)
   - Add a spatial pooling layer

4. Train a supervised classifier on top
   - Multinomial Logistic Regression or Pyramid Match Kernel SVM
64 filters on 9x9 patches trained with PSD

with Linear-Sigmoid-Diagonal Encoder

weights: -0.2828 - 0.3043
Feature Extraction

Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
Feature Extraction

Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabor?
- **Abs** Rectification layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?

\[
x - \mu \\
\frac{\text{max}(t, \sigma)}{max(t, \sigma)}
\]

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Rectification layer: needed?
- Normalization layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- C  Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N  Normalization layer: needed?
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?
- **P** Pooling down-sampling layer: average or max?

Pooling Down-Sampling Layer
Feature Extraction

- C  Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N  Normalization layer: needed?
- P  Pooling down-sampling layer: average or max?
Feature Extraction

- C  Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N  Normalization layer: needed?
- P  Pooling down-sampling layer: average or max?

THIS IS ONE STAGE OF FEATURE EXTRACTION
Training Protocol

- Training
  - Logistic Regression on Random Features: \( R \)
  - Logistic Regression on PSD features: \( U \)
  - Refinement of whole net from random with backprop: \( R^+ \)
  - Refinement of whole net starting from PSD filters: \( U^+ \)

- Classifier
  - Multinomial Logistic Regression or Pyramid Match Kernel SVM
### Using PSD Features for Recognition

\[
\left[ 64.F_{CSG}^{9 \times 9} - R/N/P_5^{5 \times 5} \right] - \log\_reg
\]

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>(R_{abs} - N - P_A)</th>
<th>(R_{abs} - P_A)</th>
<th>(N - P_M)</th>
<th>(N - P_A)</th>
<th>(P_A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(U^+)</td>
<td>54.2%</td>
<td>50.0%</td>
<td>44.3%</td>
<td>18.5%</td>
<td>14.5%</td>
</tr>
<tr>
<td>(R^+)</td>
<td>54.8%</td>
<td>47.0%</td>
<td>38.0%</td>
<td>16.3%</td>
<td>14.3%</td>
</tr>
<tr>
<td>(U)</td>
<td>52.2%</td>
<td>43.3(\pm 1.6)%</td>
<td>44.0%</td>
<td>17.2%</td>
<td>13.4%</td>
</tr>
<tr>
<td>(R)</td>
<td>53.3%</td>
<td>31.7%</td>
<td>32.1%</td>
<td>15.3%</td>
<td>12.1(\pm 2.2)%</td>
</tr>
</tbody>
</table>

\[
\left[ 64.F_{CSG}^{9 \times 9} - R/N/P_5^{5 \times 5} \right] - \text{PMK}
\]

| U | 65.0% |

\[
\left[ 96.F_{CSG}^{9 \times 9} - R/N/P_5^{5 \times 5} \right] - \text{PCA - lin\_svm}
\]

| U | 58.0% |

96. Gabors - PCA - lin\_svm (Pinto and DiCarlo 2006)

| Gabors | 59.0% |

SIFT - PMK (Lazebnik et al. CVPR 2006)

| Gabors | 64.6% |
Rectification makes a **huge** difference:
- 14.5% -> 50.0%, without normalization
- 44.3% -> 54.2% with normalization

Normalization makes a difference:
- 50.0 → 54.2

Unsupervised pretraining makes small difference

PSD works just as well as SIFT

Random filters work as well as anything!
- If rectification/normalization is present

PMK_SVM classifier works a lot better than multinomial log_reg on low-level features
- 52.2% → 65.0%
Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!

- PSD features are more stable.

Feature Sign (FS) is an optimization method for computing sparse codes [Lee...Ng 2006]
PSD Features are more stable

Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!

Because PSD features are more stable. Feature obtained through sparse optimization can change a lot with small changes of the input.

![Graph showing feature sign and PSD for different conditions](image)

How many features change sign in patches from successive video frames (a,b), versus patches from random frame pairs (c)
PSD features are much cheaper to compute

Computing PSD features is hundreds of times cheaper than Feature Sign.
How Many 9x9 PSD features do we need?

Accuracy increases slowly past 64 filters.
1. Train stage-1 filters with PSD on patches from natural images
2. Compute stage-1 features on training set
3. Train stage-2 filters with PSD on stage-1 feature patches
4. Compute stage-2 features on training set
5. Train linear classifier on stage-2 features
6. Refine entire network with supervised gradient descent

What are the effects of the non-linearities and unsupervised pretraining?
Multistage Hubel-Wiesel Architecture on Caltech-101

Y (luminance)

CONVOLUTIONS (9x9)

INPUT 3@140x140

32@132x132

MAX/SUBSAMPLING (4x4)

32@33x33

CONVOLUTIONS (9x9)

64@25x25

MAX/SUBSAMPLING (5x5)

64@5x5
Multistage Hubel-Wiesel Architecture

**Image Preprocessing:**
- High-pass filter, local contrast normalization (divisive)

**First Stage:**
- Filters: 64 9x9 kernels producing 64 feature maps
- Pooling: 10x10 averaging with 5x5 subsampling

**Second Stage:**
- Filters: 4096 9x9 kernels producing 256 feature maps
- Pooling: 6x6 averaging with 3x3 subsampling
- Features: 256 feature maps of size 4x4 (4096 features)

**Classifier Stage:**
- Multinomial logistic regression

**Number of parameters:**
- Roughly 750,000
Multistage Hubel-Wiesel Architecture on Caltech-101

<table>
<thead>
<tr>
<th>Single Stage System: $[64.F_{CSG}^{9\times9} - R/N/P_{5\times5}] - \text{log}_\text{reg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/N/P</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>U$^+$</td>
</tr>
<tr>
<td>R$^+$</td>
</tr>
<tr>
<td>U</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Two Stage System: $[64.F_{CSG}^{9\times9} - R/N/P_{5\times5}] - [256.F_{CSG}^{9\times9} - R/N/P_{4\times4}] - \text{log}_\text{reg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/N/P</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>U$^+$U$^+$</td>
</tr>
<tr>
<td>R$^+$R$^+$</td>
</tr>
<tr>
<td>UU</td>
</tr>
<tr>
<td>RR</td>
</tr>
<tr>
<td>GT</td>
</tr>
</tbody>
</table>

$\leftarrow$ like HMAX model

<table>
<thead>
<tr>
<th>Single Stage: $[64.F_{CSG}^{9\times9} - R/N/P_{5\times5}] - \text{PMK-SVM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Two Stages: $[64.F_{CSG}^{9\times9} - R/N/P_{5\times5}] - [256.F_{CSG}^{9\times9} - R/N] - \text{PMK-SVM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>UU</td>
</tr>
</tbody>
</table>
Two-Stage Result Analysis

- Second Stage + logistic regression = PMK_SVM
- Unsupervised pre-training doesn't help much :-(
- Random filters work amazingly well with normalization
- Supervised global refinement helps a bit
- The best system is really cheap
- Either use rectification and average pooling or no rectification and max pooling.
Multistage Hubel-Wiesel Architecture: Filters

- **Stage 1**
  - After PSD
  - Weights: \(-0.2232 - 0.2075\)

- **Stage 2**
  - After supervised refinement
  - Weights: \(-0.2828 - 0.3043\)
  - Weights: \(-0.0778 - 0.064\)

- Weights: \(-0.0929 - 0.0784\)
MNIST dataset

- 10 classes and up to 60,000 training samples per class
**Architecture**

- $U^+U^+$: 0.53% error \(\text{(this is a record on the undistorted MNIST!)}\)

**Comparison:** $RR$ versus $UU$ and $RR^+$
Why Random Filters Work?
## The Competition: SIFT + Sparse-Coding + PMK-SVM

### Replacing K-means with Sparse Coding

- [Yang 2008]
- [Boureau, Ponce, LeCun 2010]

<table>
<thead>
<tr>
<th>Method</th>
<th>Caltech 15</th>
<th>Caltech 30</th>
<th>Scenes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boiman et al. [1]</td>
<td>Nearest neighbor + spatial correspondence</td>
<td>65.00 ± 1.14</td>
<td>70.40</td>
</tr>
<tr>
<td>Jain et al. [8]</td>
<td>Fast image search for learned metrics</td>
<td>61.00</td>
<td>69.60</td>
</tr>
<tr>
<td>Lazebnik et al. [12]</td>
<td>Spatial Pyramid + hard quantization + kernel SVM</td>
<td>56.40</td>
<td>64.40 ± 0.80</td>
</tr>
<tr>
<td>van Gemert et al. [24]</td>
<td>Spatial Pyramid + soft quantization + kernel SVM</td>
<td>-</td>
<td>64.14 ± 1.18</td>
</tr>
<tr>
<td>Yang et al. [26]</td>
<td>SP + sparse codes + max pooling + linear</td>
<td><strong>67.00 ± 0.45</strong></td>
<td><strong>73.2 ± 0.54</strong></td>
</tr>
<tr>
<td>Zhang et al. [27]</td>
<td>kNN-SVM</td>
<td>59.10 ± 0.60</td>
<td>66.20 ± 0.50</td>
</tr>
<tr>
<td>Zhou et al. [29]</td>
<td>SP + Gaussian mixture</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Baseline:</td>
<td>SP + hard quantization + avg pool + kernel SVM</td>
<td>56.74 ± 1.31</td>
<td>64.19 ± 0.94</td>
</tr>
<tr>
<td>Unsupervised coding</td>
<td>SP + soft quantization + avg pool + kernel SVM</td>
<td>59.12 ± 1.51</td>
<td>66.42 ± 1.26</td>
</tr>
<tr>
<td>1x1 features</td>
<td>SP + soft quantization + max pool + kernel SVM</td>
<td>63.61 ± 0.88</td>
<td>-</td>
</tr>
<tr>
<td>8 pixel grid resolution</td>
<td>SP + sparse codes + avg pool + kernel SVM</td>
<td>62.85 ± 1.22</td>
<td>70.27 ± 1.29</td>
</tr>
<tr>
<td></td>
<td>SP + sparse codes + max pool + kernel SVM</td>
<td>64.62 ± 0.94</td>
<td><strong>71.81 ± 0.96</strong></td>
</tr>
<tr>
<td></td>
<td>SP + sparse codes + max pool + linear</td>
<td><strong>64.71 ± 1.05</strong></td>
<td>71.52 ± 1.13</td>
</tr>
</tbody>
</table>

Table 1. Performance comparison of unsupervised features using a 4 × 4 spatial pyramid. Our results show a consistent ranking of results with results going from worst to best when using hard quantization, soft quantization, sparse codes with average pooling and intersection kernel SVM, sparse codes with max pooling and linear SVM. Results use codebooks of size 1024 on the Caltech-101 dataset, and 4096 on the Scenes dataset. SP: spatial pyramid. Soft quantization paired with max pooling can beat sparse coding paired with average pooling.
The Competition: Variations on SIFT+PMK-SVM

Using sparse coding on a neighborhood of SIFTs

<table>
<thead>
<tr>
<th>Feature 1x1</th>
<th>Grid resolution</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Receptive field</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>67.70 ± 0.91</td>
<td>67.90 ± 1.35</td>
<td>67.71 ± 1.06</td>
<td>66.73 ± 0.84</td>
<td>64.70 ± 1.05</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>74.36 ± 0.91</td>
<td>74.48 ± 1.15</td>
<td>73.84 ± 1.03</td>
<td>73.27 ± 1.05</td>
<td>71.52 ± 1.13</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grid resolution 8</th>
<th>Macrofeature</th>
<th>1x1</th>
<th>2x2</th>
<th>3x3</th>
<th>4x4</th>
<th>5x5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Receptive field</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
</tr>
<tr>
<td>15</td>
<td>64.70 ± 1.05</td>
<td>66.27 ± 1.14</td>
<td>67.13 ± 0.98</td>
<td>66.65 ± 0.88</td>
<td>66.93 ± 0.96</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>71.52 ± 1.13</td>
<td>72.62 ± 1.04</td>
<td>73.32 ± 1.12</td>
<td>73.20 ± 1.04</td>
<td>73.29 ± 1.04</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grid resolution 4, Macrofeature 2x2</th>
<th>Stride</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Receptive field</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>15</td>
<td>67.76 ± 0.98</td>
<td>68.33 ± 0.94</td>
<td>68.58 ± 1.05</td>
<td>68.78 ± 0.109</td>
<td>68.38 ± 1.00</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>73.80 ± 0.75</td>
<td>74.42 ± 0.89</td>
<td>74.63 ± 1.00</td>
<td>75.14 ± 0.86</td>
<td>74.71 ± 1.08</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grid resolution 3</th>
<th>Macrofeature</th>
<th>2x2</th>
<th>3x3</th>
<th>4x4</th>
<th>2x2</th>
<th>3x3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Stride</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Receptive field</td>
<td>19</td>
<td>22</td>
<td>25</td>
<td>28</td>
<td>40</td>
</tr>
<tr>
<td>15</td>
<td>67.74 ± 1.03</td>
<td>67.74 ± 0.99</td>
<td>68.03 ± 0.06</td>
<td>68.67 ± 1.00</td>
<td>68.56 ± 0.87</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>73.57 ± 0.86</td>
<td>73.69 ± 0.69</td>
<td>73.95 ± 0.98</td>
<td>74.78 ± 0.91</td>
<td>74.49 ± 0.12</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Results on the Caltech 101 dataset, 1024 codewords, max pooling, linear classifier. The stride measures the closeness of SIFT features forming the input of a macrofeature; e.g. a stride of 3 for a grid resolution of 4 pixels means that a macrofeature selects features every 12 pixels. The receptive field gives the side of the square spanned by a macrofeature, in pixels; it is a function of other parameters, not an additional parameter.
Small NORB dataset

5 classes and up to 24,300 training samples per class
**NORB Generic Object Recognition Dataset**

- **50** toys belonging to 5 categories: *animal, human figure, airplane, truck, car*
- **10** instance per category: 5 instances used for training, 5 instances for testing
- **Raw dataset:** 972 stereo pair of each object instance. 48,600 image pairs total.

**For each instance:**
- **18 azimuths**
  - 0 to 350 degrees every 20 degrees
- **9 elevations**
  - 30 to 70 degrees from horizontal every 5 degrees
- **6 illuminations**
  - On/off combinations of 4 lights
- **2 cameras (stereo)**
  - 7.5 cm apart
  - 40 cm from the object

**Training instances**

**Test instances**
Small NORB dataset

Architecture

Two Stages

Error Rate (log scale) VS. Number Training Samples (log scale)
Unsupervised PSD ignores the spatial pooling step.

Could we devise a similar method that learns the pooling layer as well?

Idea [Hyvarinen & Hoyer 2001]: sparsity on pools of features

- Minimum number of pools must be non-zero
- Number of features that are on within a pool doesn't matter
- Pools tend to regroup similar features
Learning the filters and the pools

- Using an idea from Hyvarinen: topographic square pooling (subspace ICA)
  1. Apply filters on a patch (with suitable non-linearity)
  2. Arrange filter outputs on a 2D plane
  3. Square filter outputs
  4. Minimize sqrt of sum of blocks of squared filter outputs

\[
\text{Overall Sparsity term: } \sum_{i=1}^{K} \sqrt{v_i^2}
\]

\[
v_1^2 = \sum_{j \in P_1} (w_j z_j)^2 \quad v_k^2 = \sum_{j \in P_k} (w_j z_j)^2
\]

Units in the code $Z$ Define pools and enforce sparsity across pools

Yann LeCun
Learning the filters and the pools

The filters arrange themselves spontaneously so that similar filters enter the same pool.

The pooling units can be seen as complex cells

They are invariant to local transformations of the input

For some it's translations, for others rotations, or other transformations.
Pinwheels?
Invariance Properties Compared to SIFT

- Measure distance between feature vectors (128 dimensions) of 16x16 patches from natural images
  - Left: normalized distance as a function of translation
  - Right: normalized distance as a function of translation when one patch is rotated 25 degrees.

- Topographic PSD features are more invariant than SIFT
Learning Invariant Features

Recognition Architecture

- HPF/LCN -> filters -> tanh -> sqr -> pooling -> sqrt -> Classifier
- Block pooling plays the same role as rectification
### Recognition Accuracy on Caltech 101

- A/B Comparison with SIFT (128x34x34 descriptors)
- 32x16 topographic map with 16x16 filters
- Pooling performed over 6x6 with 2x2 subsampling
- 128 dimensional feature vector per 16x16 patch
- Feature vector computed every 4x4 pixels (128x34x34 feature maps)
- Resulting feature maps are spatially smoothed

<table>
<thead>
<tr>
<th>Method</th>
<th>Av. Accuracy/Class (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>local norm(<em>{5 \times 5}) + boxcar(</em>{5 \times 5}) + PCA(_{3060}) + linear SVM</strong></td>
<td></td>
</tr>
<tr>
<td>IPSD (24x24)</td>
<td>50.9</td>
</tr>
<tr>
<td>SIFT (24x24) (non rot. inv.)</td>
<td>51.2</td>
</tr>
<tr>
<td>SIFT (24x24) (rot. inv.)</td>
<td>45.2</td>
</tr>
<tr>
<td>Serre et al. features [25]</td>
<td>47.1</td>
</tr>
<tr>
<td><strong>local norm(_{9 \times 9}) + Spatial Pyramid Match Kernel SVM</strong></td>
<td></td>
</tr>
<tr>
<td>SIFT [11]</td>
<td>64.6</td>
</tr>
<tr>
<td>IPSD (34x34)</td>
<td>59.6</td>
</tr>
<tr>
<td>IPSD (56x56)</td>
<td>62.6</td>
</tr>
<tr>
<td>IPSD (120x120)</td>
<td>65.5</td>
</tr>
</tbody>
</table>
A/B Comparison with SIFT (128x5x5 descriptors)
32x16 topographic map with 16x16 filters.

### Performance on Tiny Images Dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPSD (5x5)</td>
<td>54</td>
</tr>
<tr>
<td>SIFT (5x5) (non rot. inv.)</td>
<td>53</td>
</tr>
</tbody>
</table>

### Performance on MNIST Dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPSD (5x5)</td>
<td>1.0</td>
</tr>
<tr>
<td>SIFT (5x5) (non rot. inv.)</td>
<td>1.5</td>
</tr>
</tbody>
</table>
**Problem:**

- With patch-level training, the learning algorithm must reconstruct the entire patch with a single feature vector.
- But when the filters are used convolutionally, neighboring feature vectors will be highly redundant.

weights \( \pm 0.2828 - 0.3043 \)
Some convolutional models produce “random-looking” filters

- Convolutional Auto-encoders, Field of Experts trained with CD/HMC or with Score Matching

[Roth and Black 05]  Convolutional Product of Experts

Problems with Convolutional Training
Problems with Convolutional Training

Convolutional PoE trained with Score Matching

The filters look less random when the stride is larger!

To produce an efficient code, the system must produce uncorrelated features. But outputs from the same filters at adjacent locations are correlated, unless the filters are high frequency (random looking).

Stride = 1

Stride = 2

Stride = 3
Non-Negative Encoder-Decoder with Sparsity

Energy:

\[ E(Y, Z) = \|Y - W_d Z\|^2 + \|Z - h(W_e Y)\|^2 + \sum_i Z_i \]

Function \( h(.) \) is a component-wise hinge
- a.k.a. half-wave rectification or positive part

Hinge is equivariant to contrast [Hinton, personal communication]

Hinge makes contrast normalization easy with a global feedback [Seung]
Energy:

\[ E(Y, Z) = \|Y - W_d \star Z\|^2 + \|Z - h(W_e \star Y)\|^2 + \sum_i Z_i \]

Function \( h(.) \) is a component-wise hinge
- a.k.a. half-wave rectification or positive part

Hinge is equivariant to contrast [Hinton, personal communication]

Hinge makes contrast normalization easy with a global feedback [Seung]
Filters and Basis Functions obtained with 1, 2, 4, 8, 16, 32, and 64 filters.
Problem with patch-based training: high correlation between outputs of filters from overlapping receptive fields.
The End