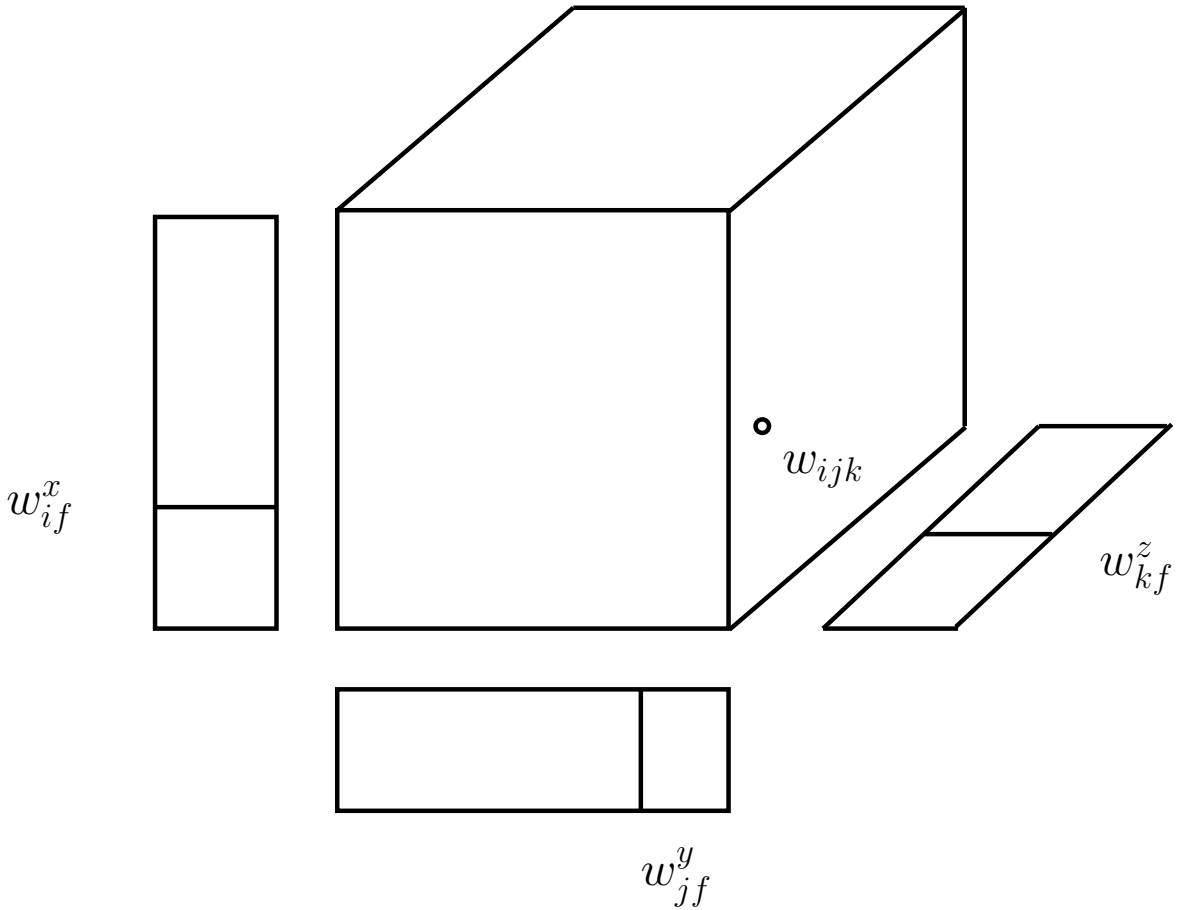
Outline

Outline

Complexity

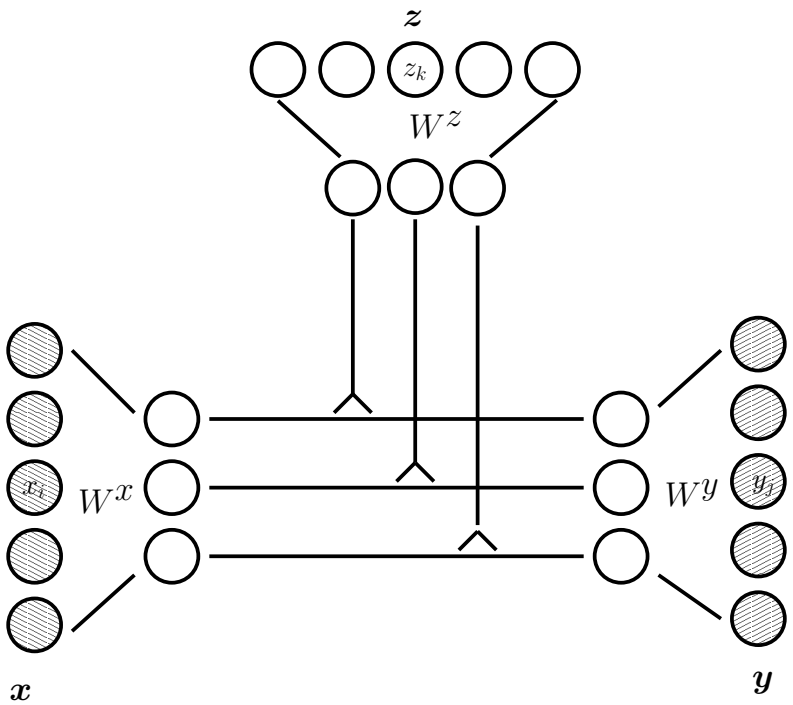
- The number of parameters is about $n \times n \times n$ (!)
- More, if we want sparse, overcomplete hiddens.
- There is a simple, yet far-reaching, way to reduce that number.

Factorization



$$w_{ijk} = \sum_{ijk} \sum_{f} w^x_{if} w^y_{jf} w^z_{kf}$$

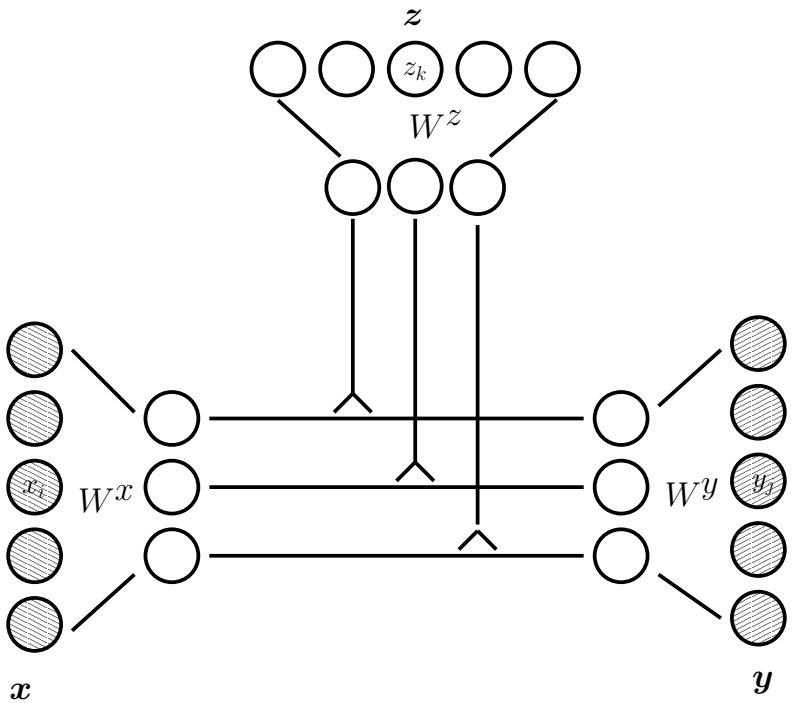
Factorization is *filter matching*



$$z_k = \sum_{ij} w_{ijk} x_i y_j = \sum_{ij} \Big(\sum_f w^x_{if} w^y_{jf} w^z_{kf} \Big) x_i y_j$$

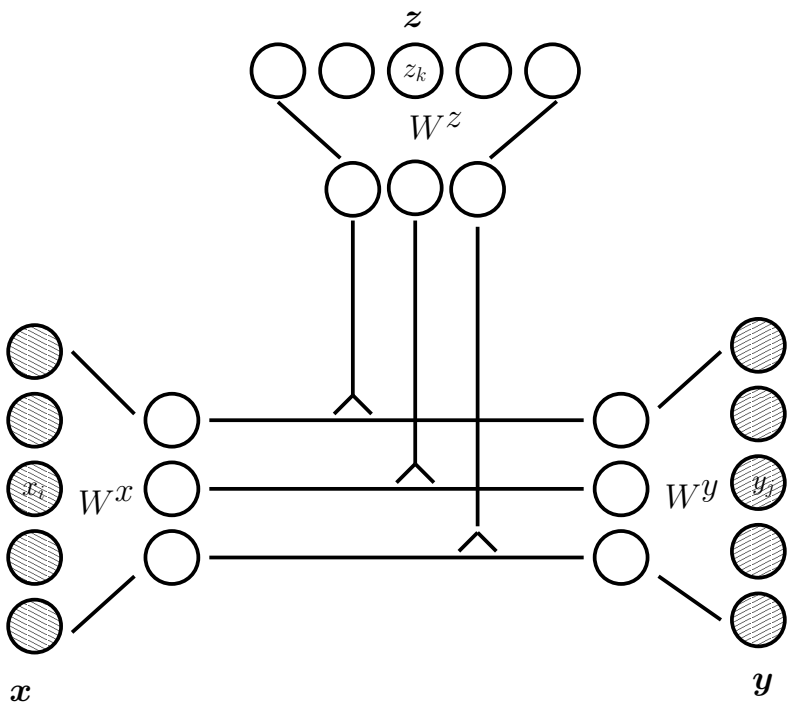
$$= \sum_f w^y_{jf} \cdot \Big(\sum_i w^x_{if} x_i \Big) \cdot \Big(\sum_j w^y_{kf} y_j \Big)$$

Factorization is *filter matching*



$$E = \sum_{ijk}(\sum_{f} w^x_{if} w^y_{jf} w^z_{kf}) x_i y_j z_k = \sum_{f}(\sum_{i} w^x_{if} x_i)(\sum_{j} w^y_{jf} y_j)(\sum_{k} w^z_{kf} z_k)$$

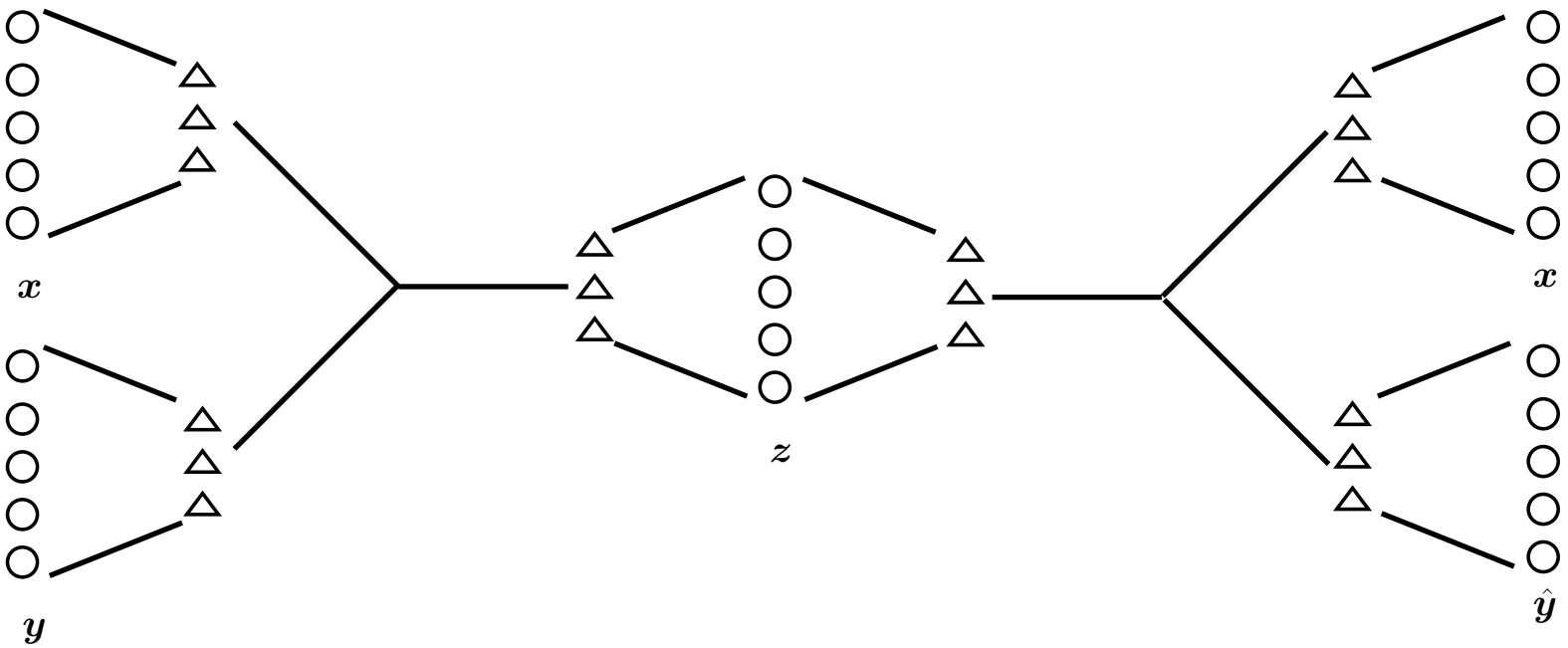
Factorized models



Factored Gated Boltzmann machines

- Exponentiate and normalize energy (just like RBM).
- Learning and inference exactly like before.
- (Taylor, 2009), (Memisevic, Hinton; 2009)

Factorized models



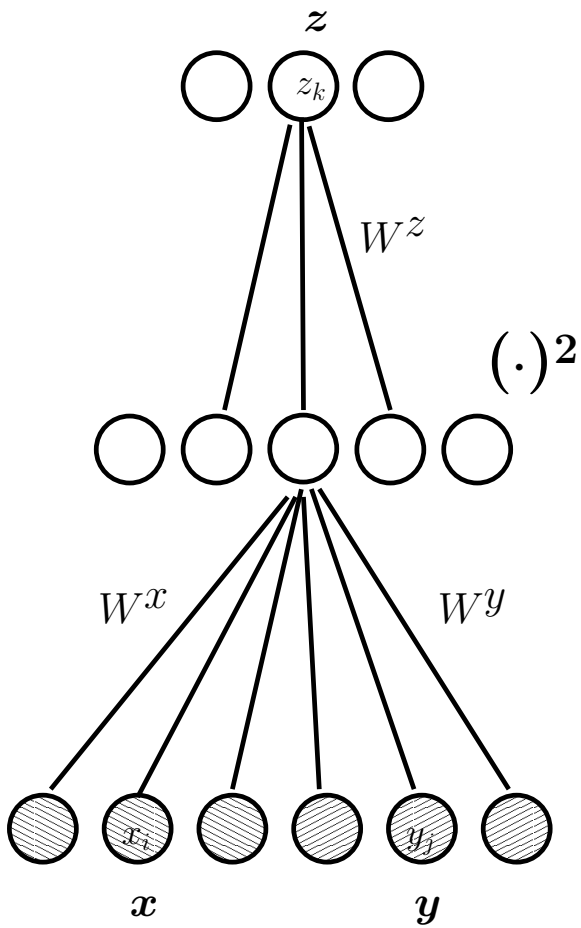
Factored Relational Autoencoders

- Again, everything like before. Back-propagate through the filters.
- Conditional learning trivial.
- Joint learning by adding two asymmetric objectives.

Square pooling models

Square pooling:

- Another way to learn filter matching models are **square pooling** models, for example:
  - ASSOM (Kohonen, 1996)
  - ISA (Hyvarinen, 2000)
  - Product of T-distributions (Osindero et al., 2006)
  - (Karklin, Lewicki; 2008)
  - cRBM (Ranzato et al., 2009)
- Often, W^z is constrained so each hidden sees only a few squared inputs. That way hiddens can be thought of as encoding **subspace** norms.

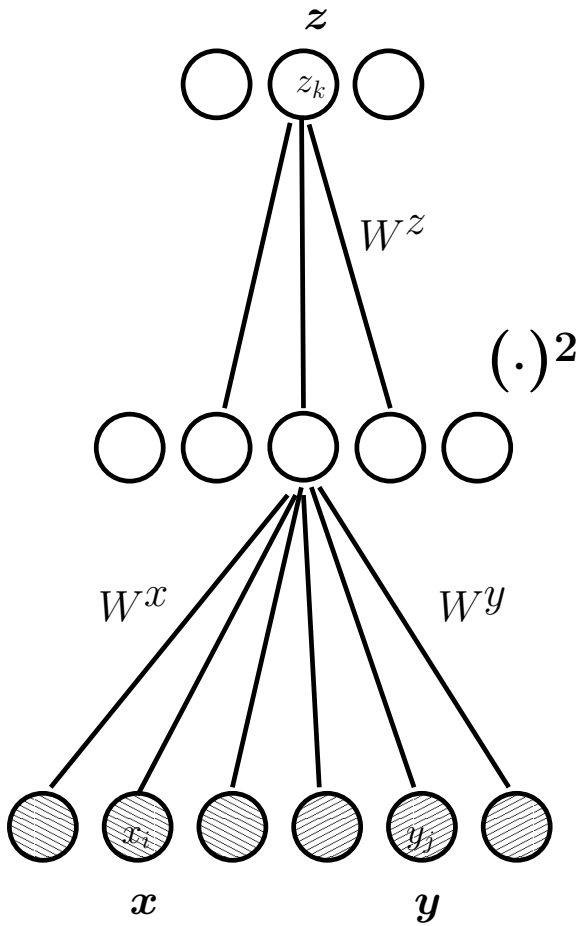


Square pooling models

Square pooling:

- Why is square pooling the same?
- The activity that a hidden unit gets is:

$$\sum_f w^z_{kf} \left(W^{x}_{\cdot f}{}^{\mathrm{T}} \boldsymbol{x} + W^{y}_{\cdot f}{}^{\mathrm{T}} \boldsymbol{y} \right)^2$$
$$= \sum_f w^z_{kf} \left(2 (W^{x}_{\cdot f}{}^{\mathrm{T}} \boldsymbol{x})(W^{y}{}^{\mathrm{T}}_{\cdot f} \boldsymbol{y}) \right.$$
$$\left. + (W^{x}_{\cdot f}{}^{\mathrm{T}} \boldsymbol{x})^2 + (W^{y}_{\cdot f}{}^{\mathrm{T}} \boldsymbol{y})^2 \right)$$

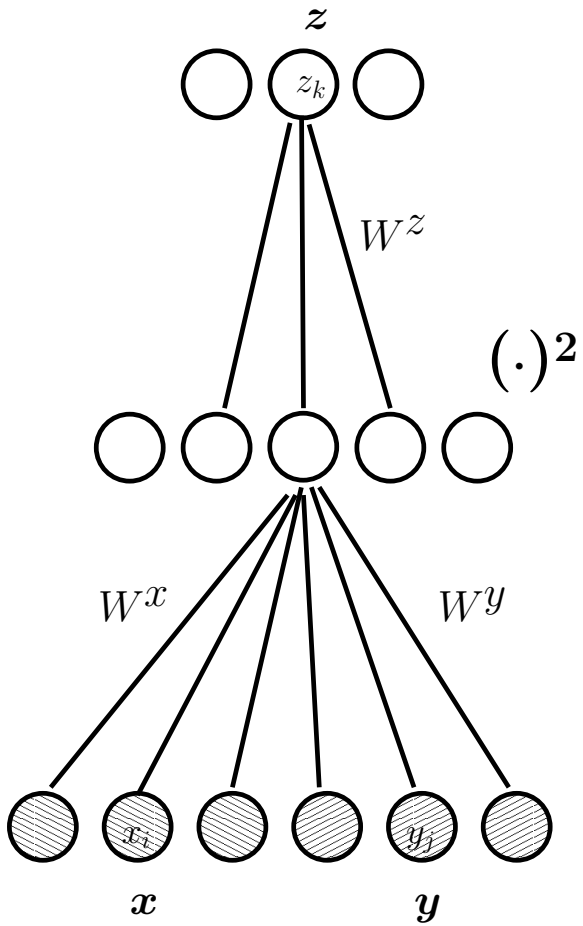


Square pooling models

Square pooling:

- Why is square pooling the same?
- The activity that a hidden unit gets is:

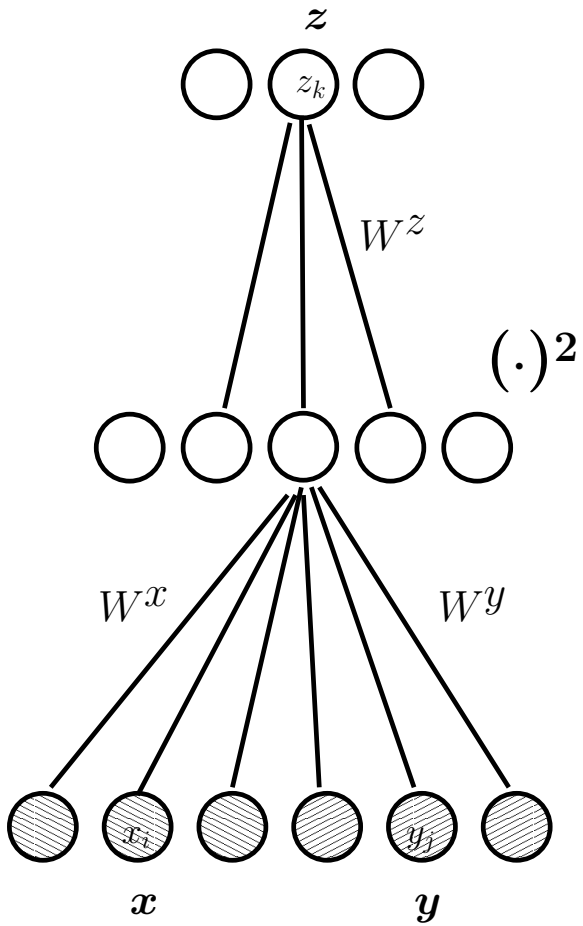
$$\sum_f w^z_{kf} \left({W^x_{\cdot f}}^{\mathrm{T}} \boldsymbol{x} + {W^y_{\cdot f}}^{\mathrm{T}} \boldsymbol{y} \right)^2$$
$$= \sum_f w^z_{kf} \Big(2 ({W^x_{\cdot f}}^{\mathrm{T}} \boldsymbol{x})({W^y_{\cdot f}}^{\mathrm{T}} \boldsymbol{y})$$
$$+ ({W^x_{\cdot f}}^{\mathrm{T}} \boldsymbol{x})^2 + ({W^y_{\cdot f}}^{\mathrm{T}} \boldsymbol{y})^2 \Big)$$



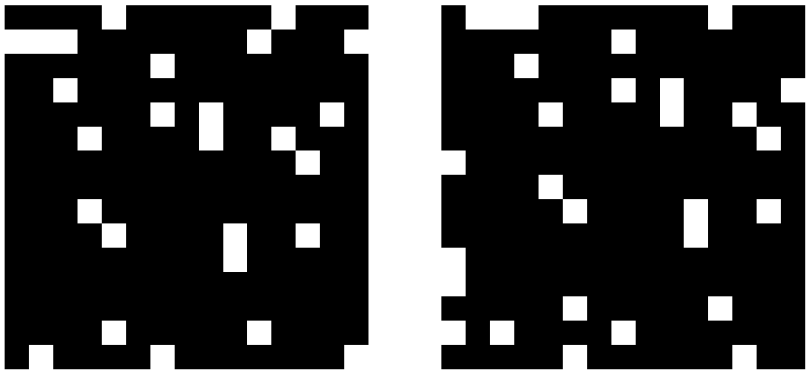
Square pooling models

Square pooling:

- Learning is somewhat more difficult than with factored gated feature learning.
- Example ISA: Gradient-based, while enforcing $W^{xy\mathrm{T}}W^{xy} = I$ after every gradient step (eigen-decomposition).

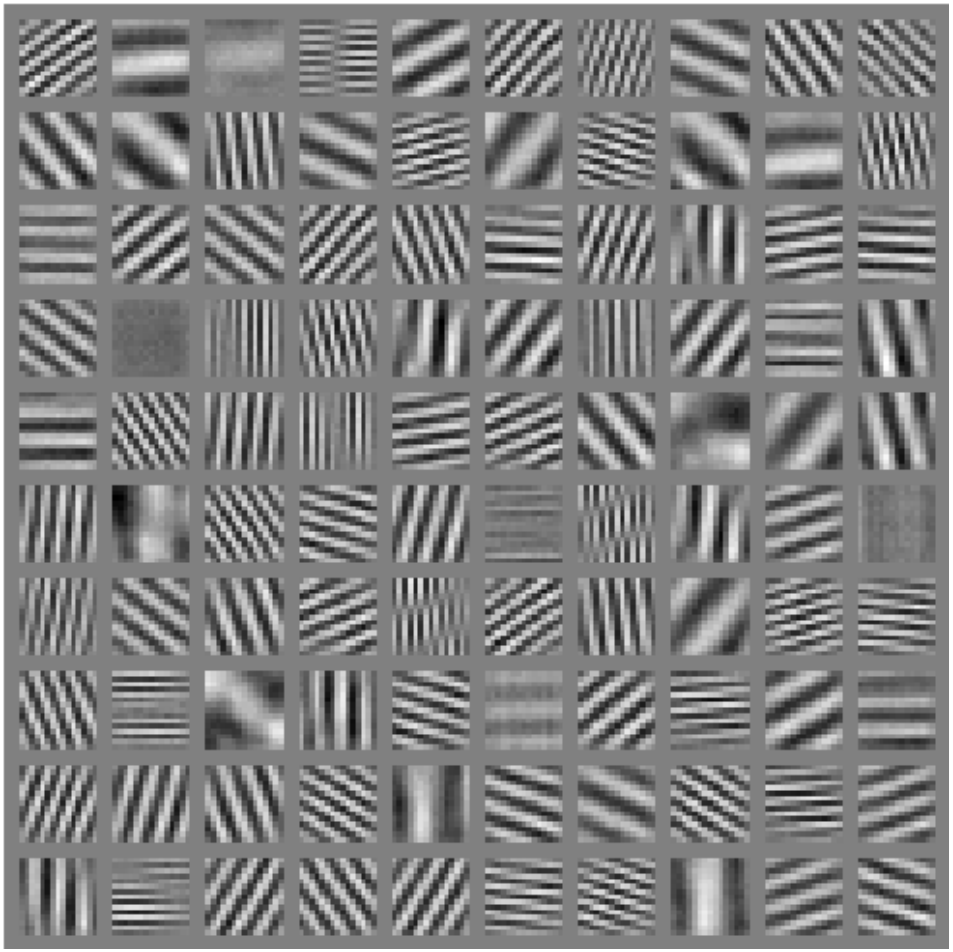


Examples

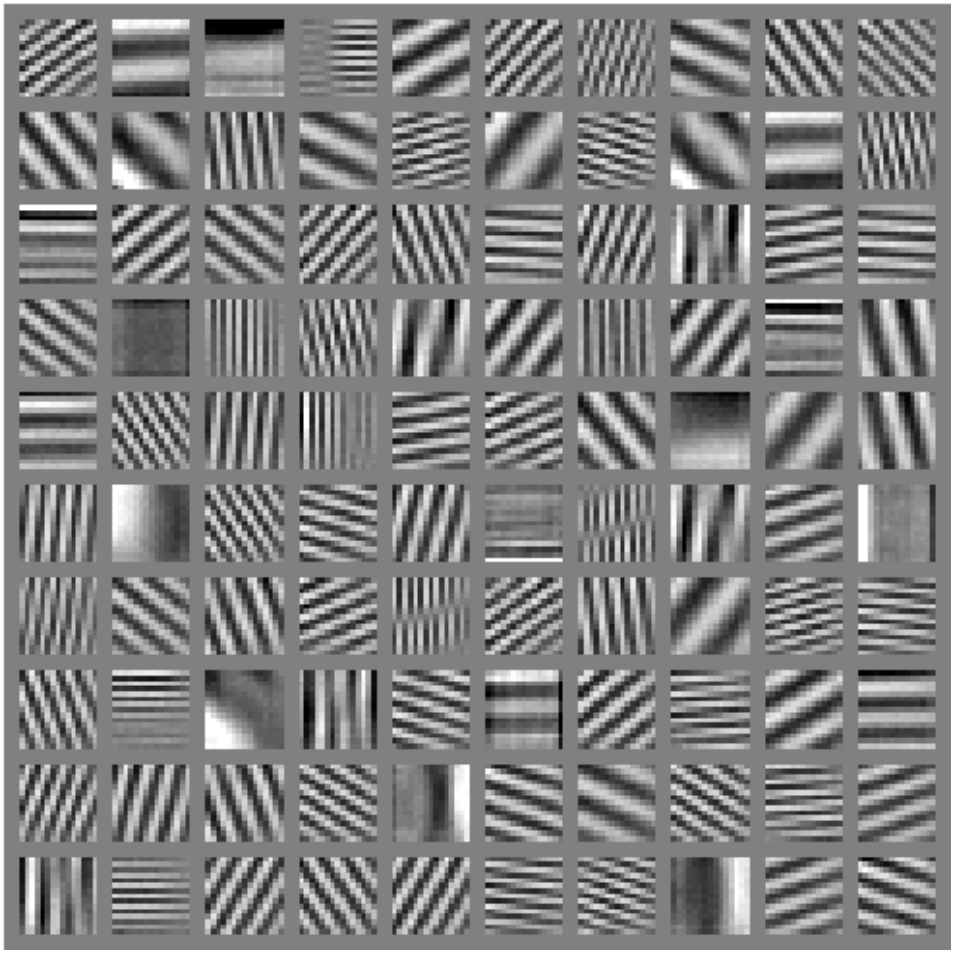


- Toy examples:
- There is no structure in these images.
- Only in *how they change*.

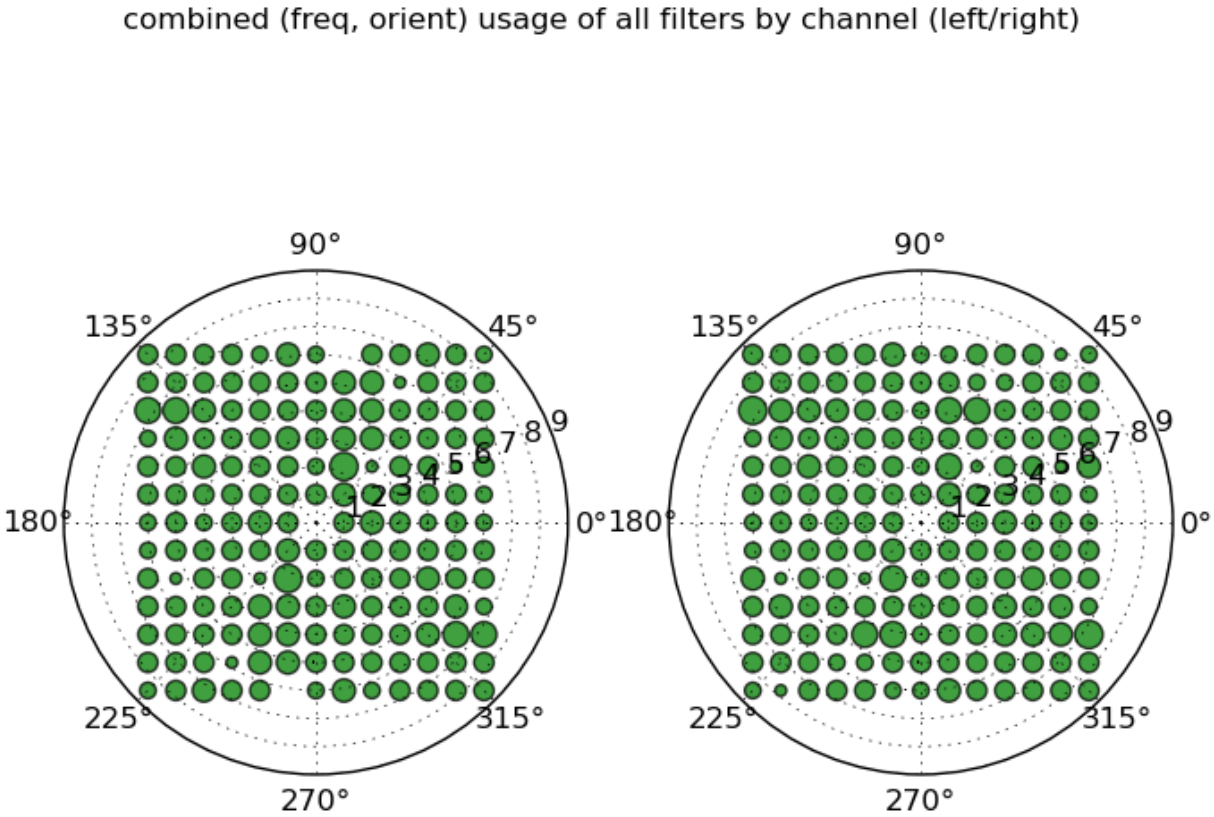
Learned filters w_{if}^{x}



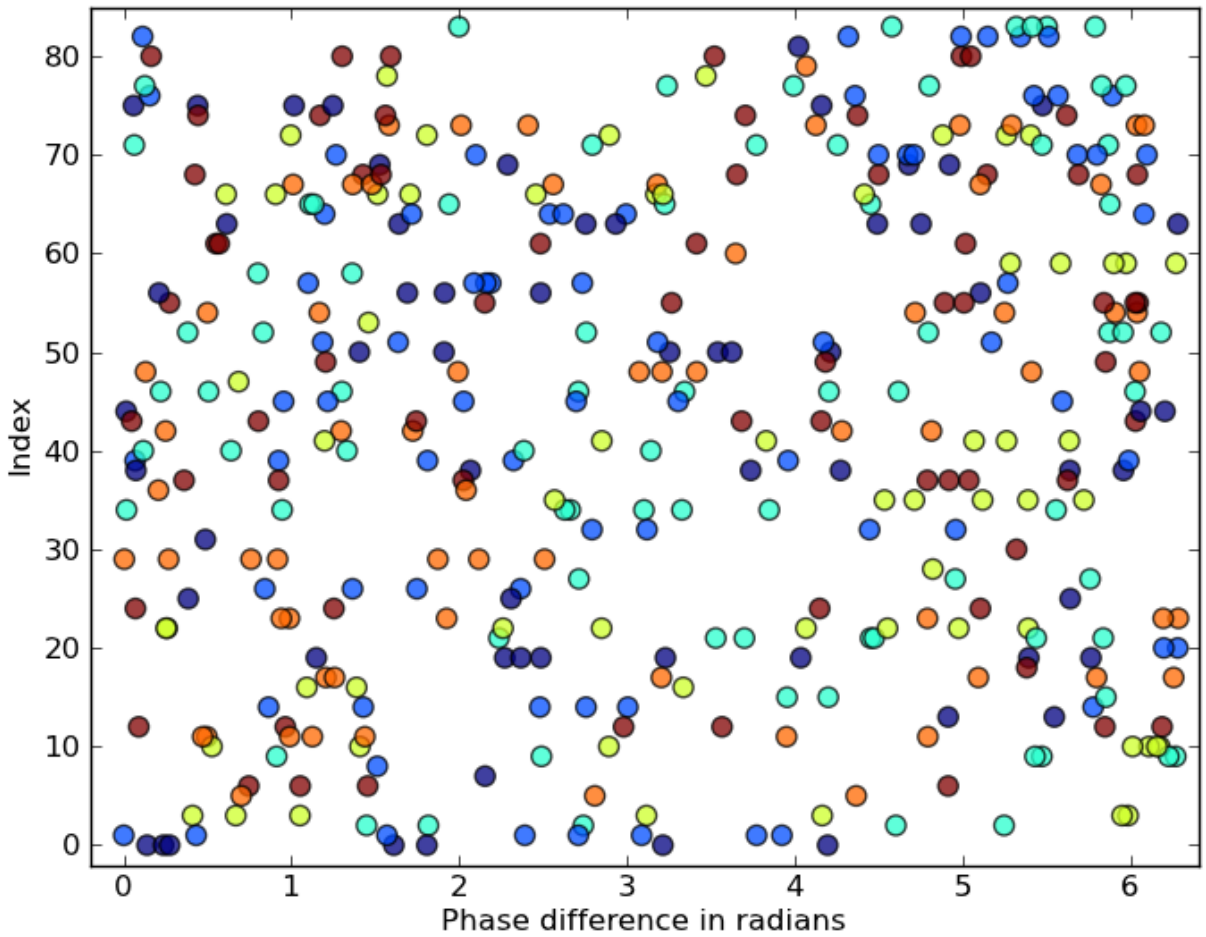
Learned filters w^y_{jf}



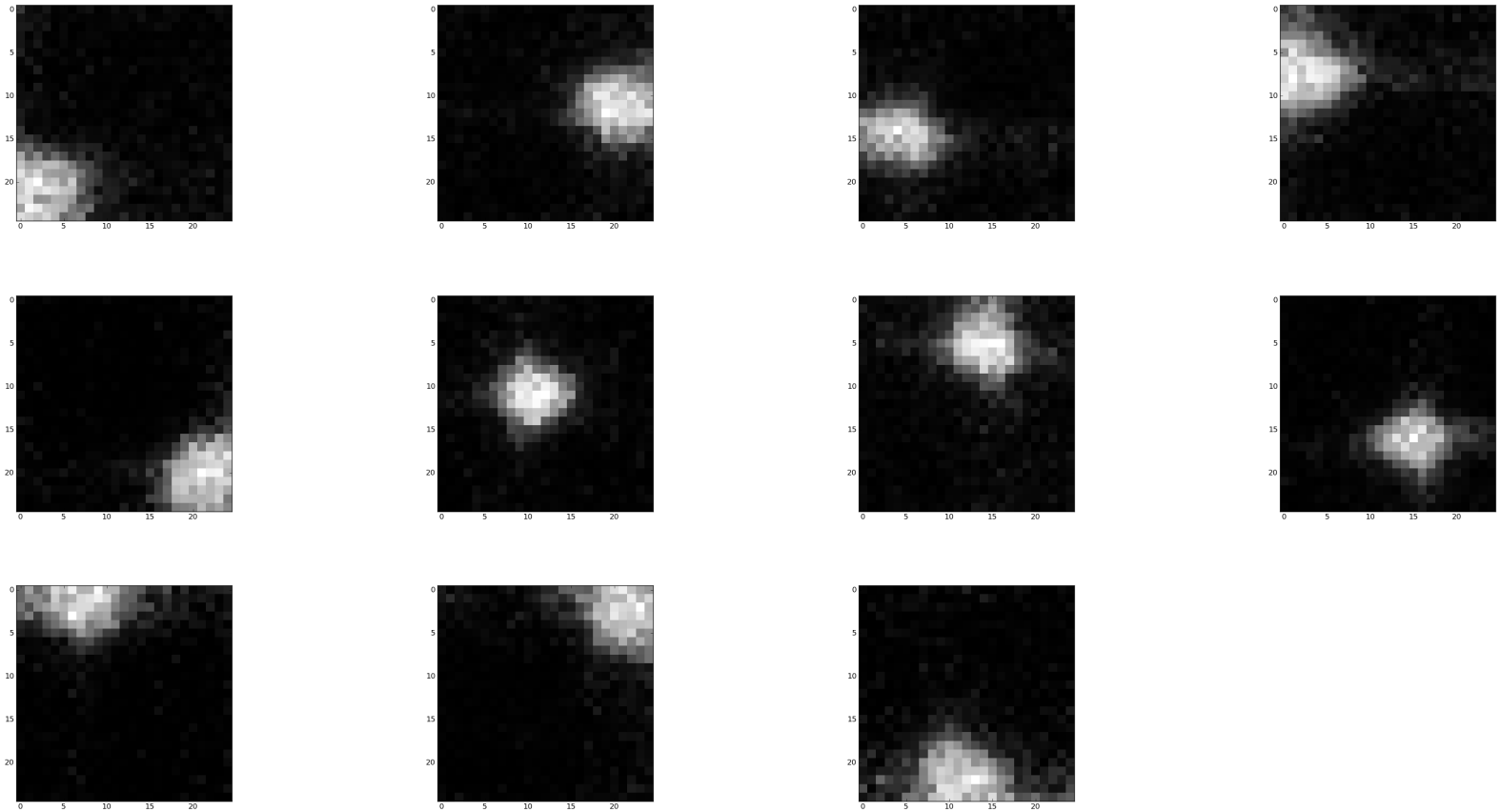
Frequency/orientation histograms



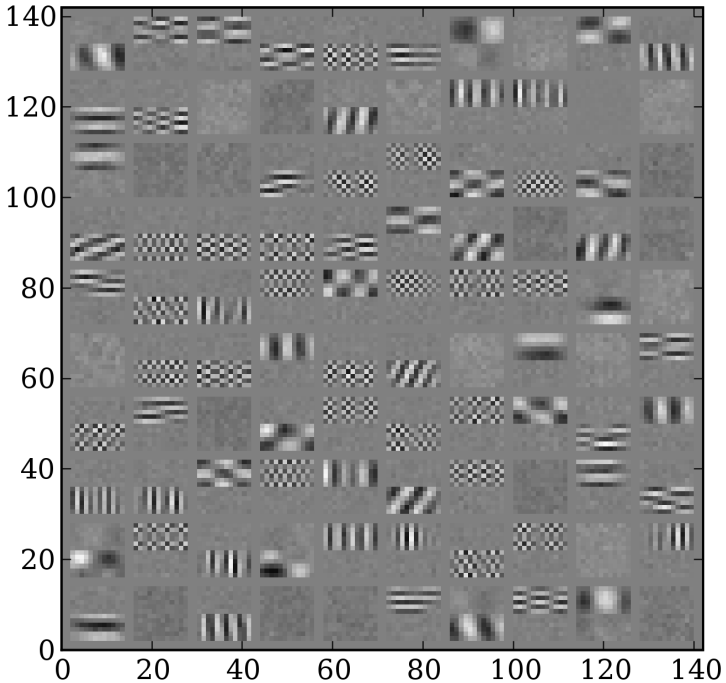
Frequency/orientation histograms

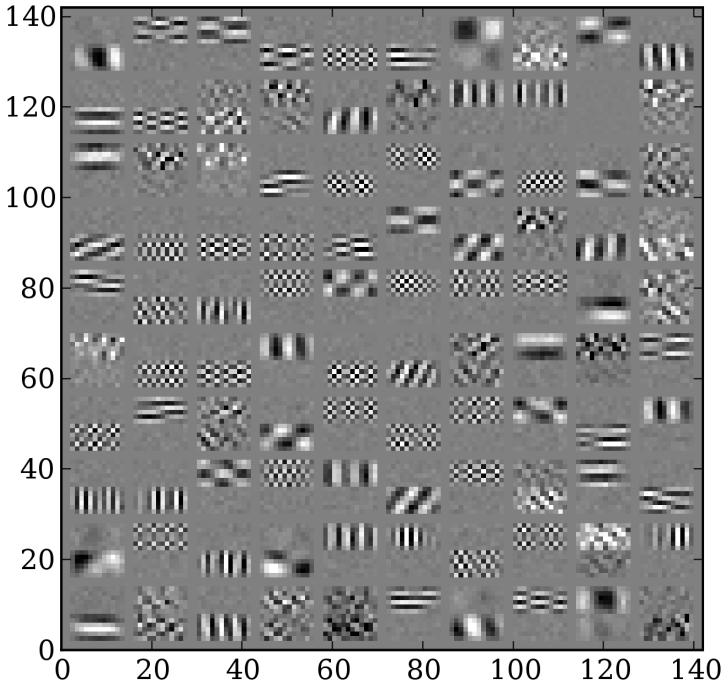


Velocity tuning of mapping units

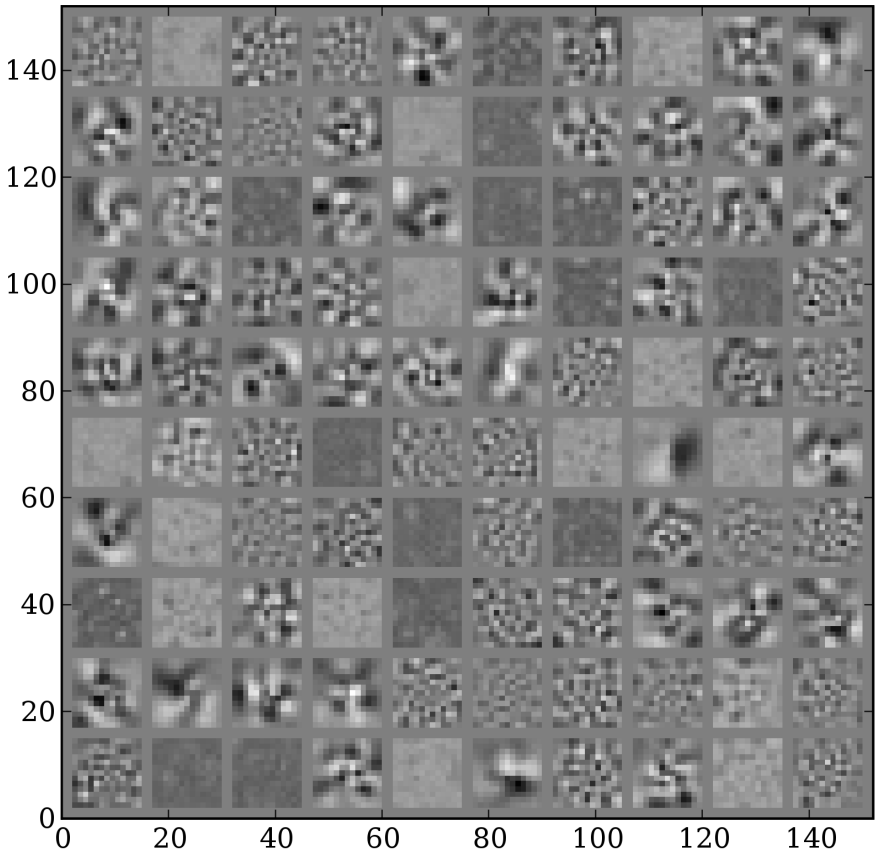


Filters learned from split-screen shifts

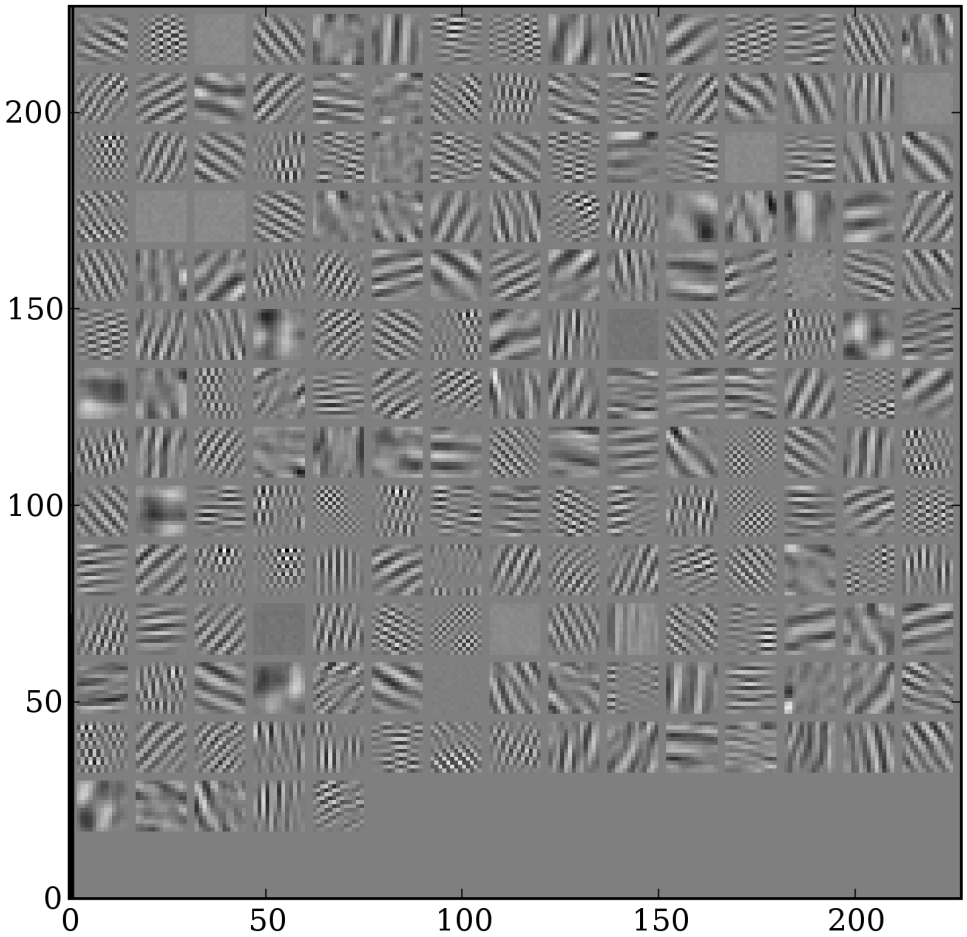




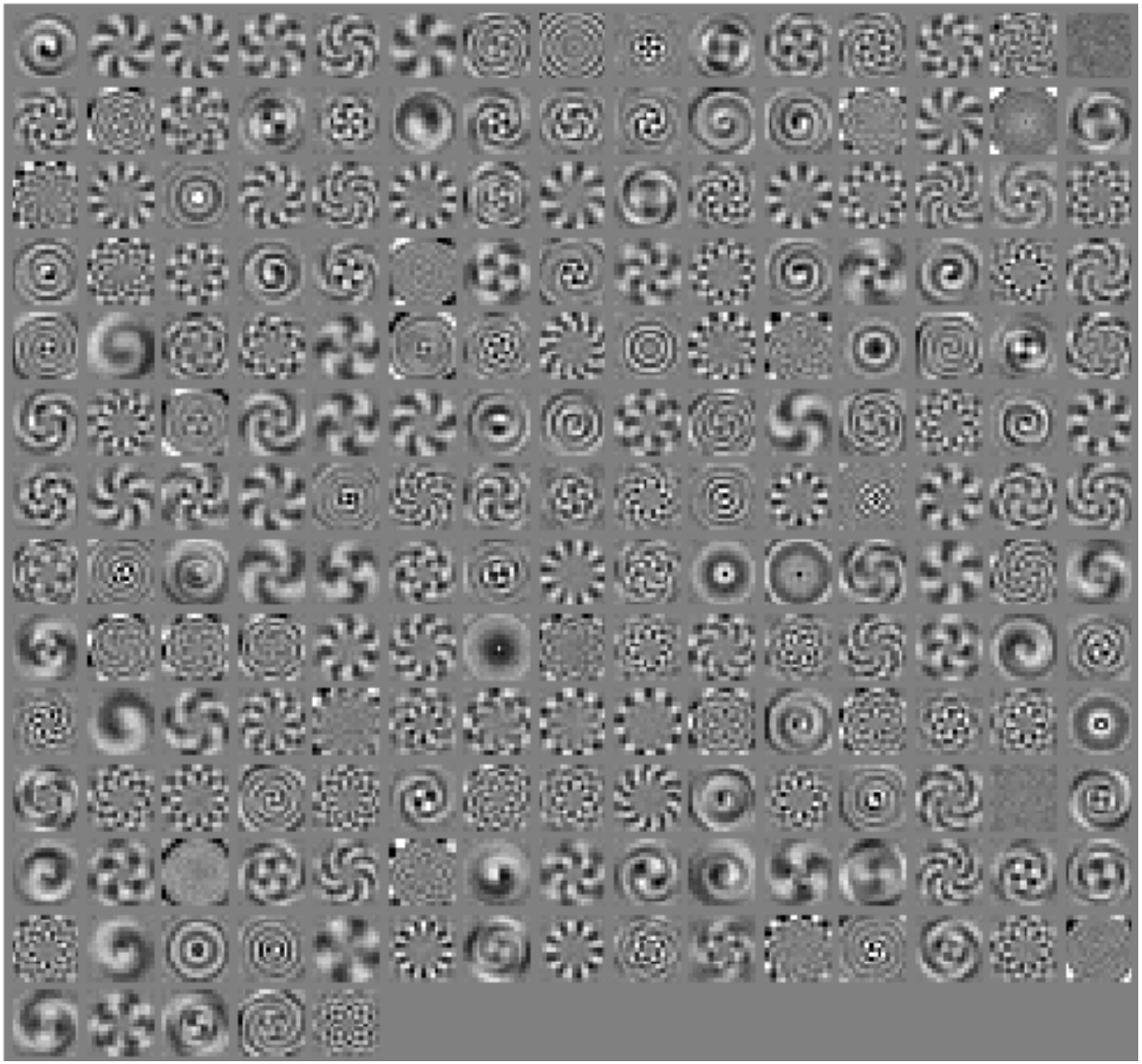
Affine filters



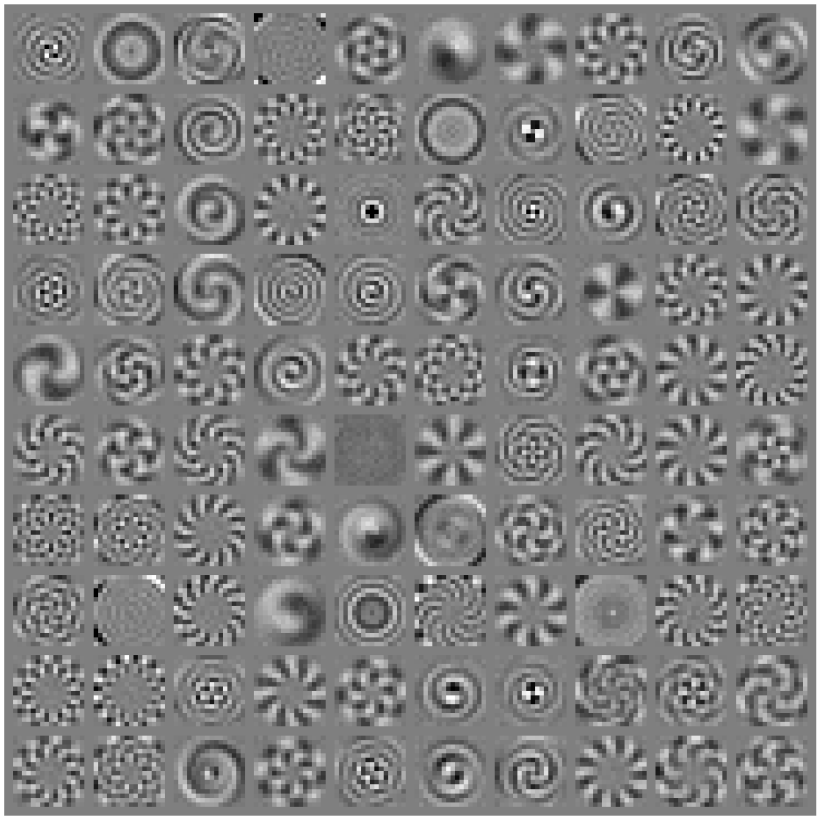
“Filtering”-filters



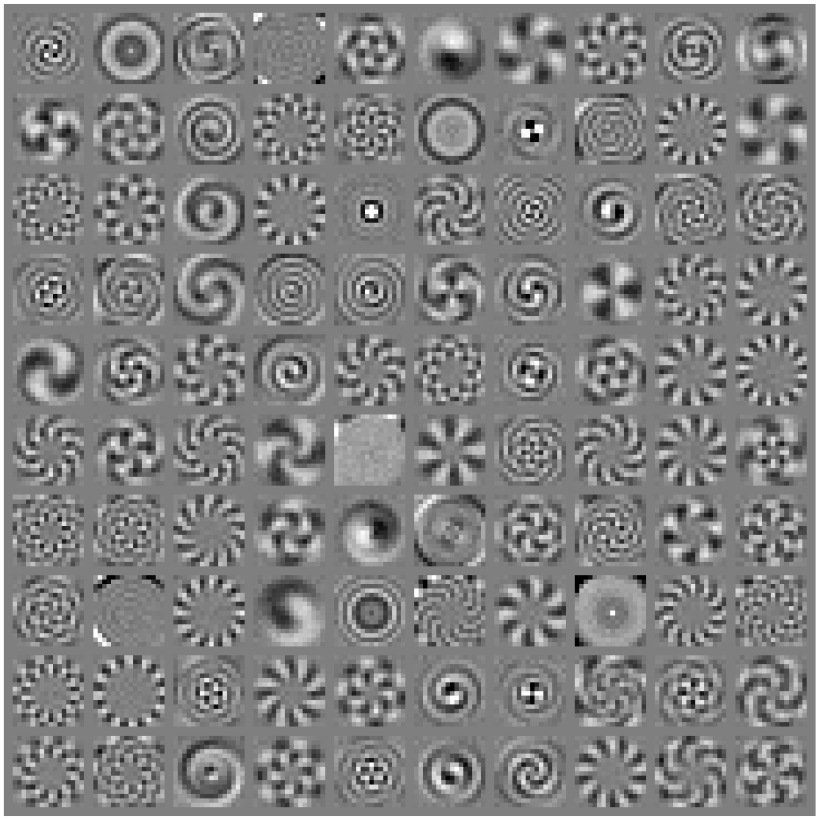
Rotation filters



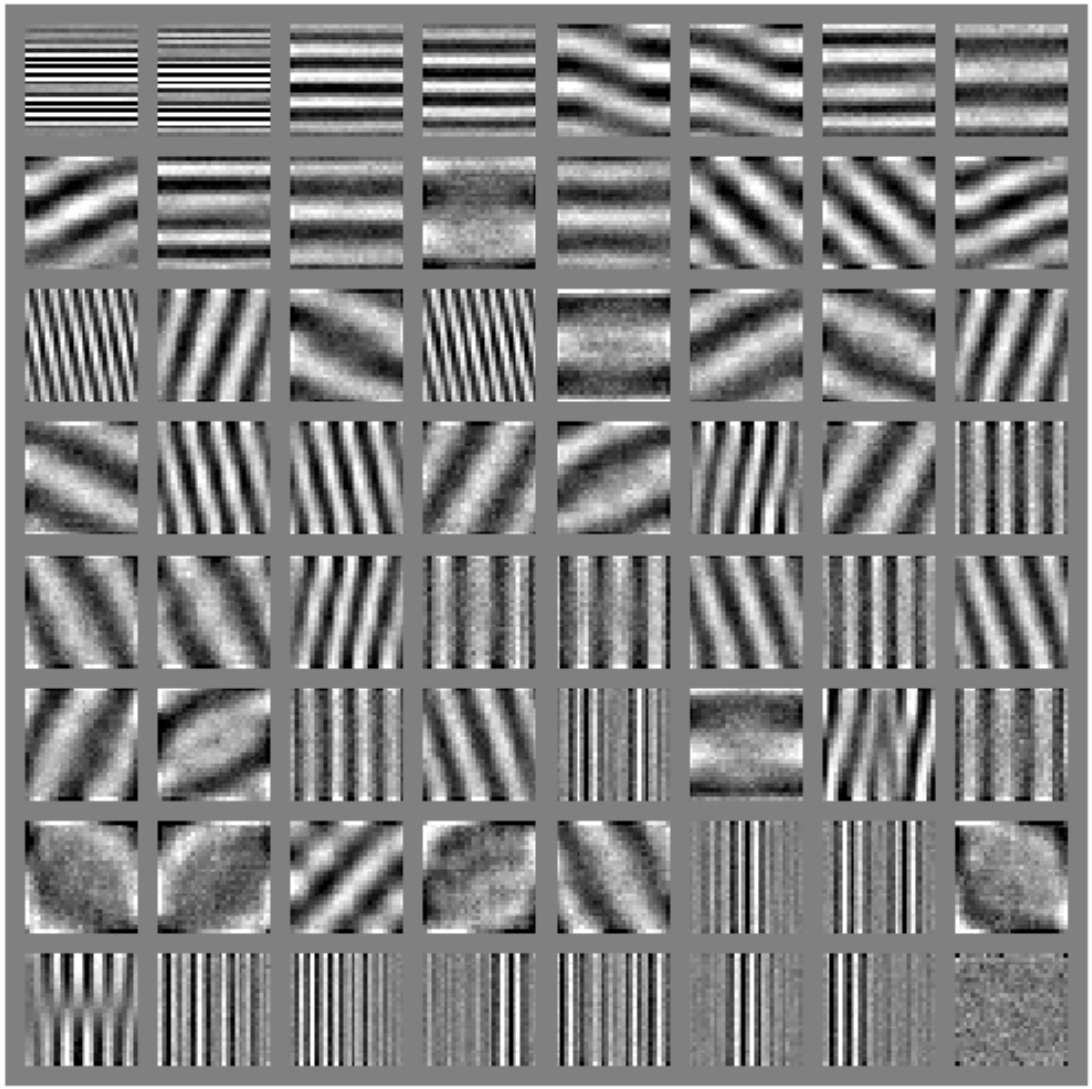
Rotation filters



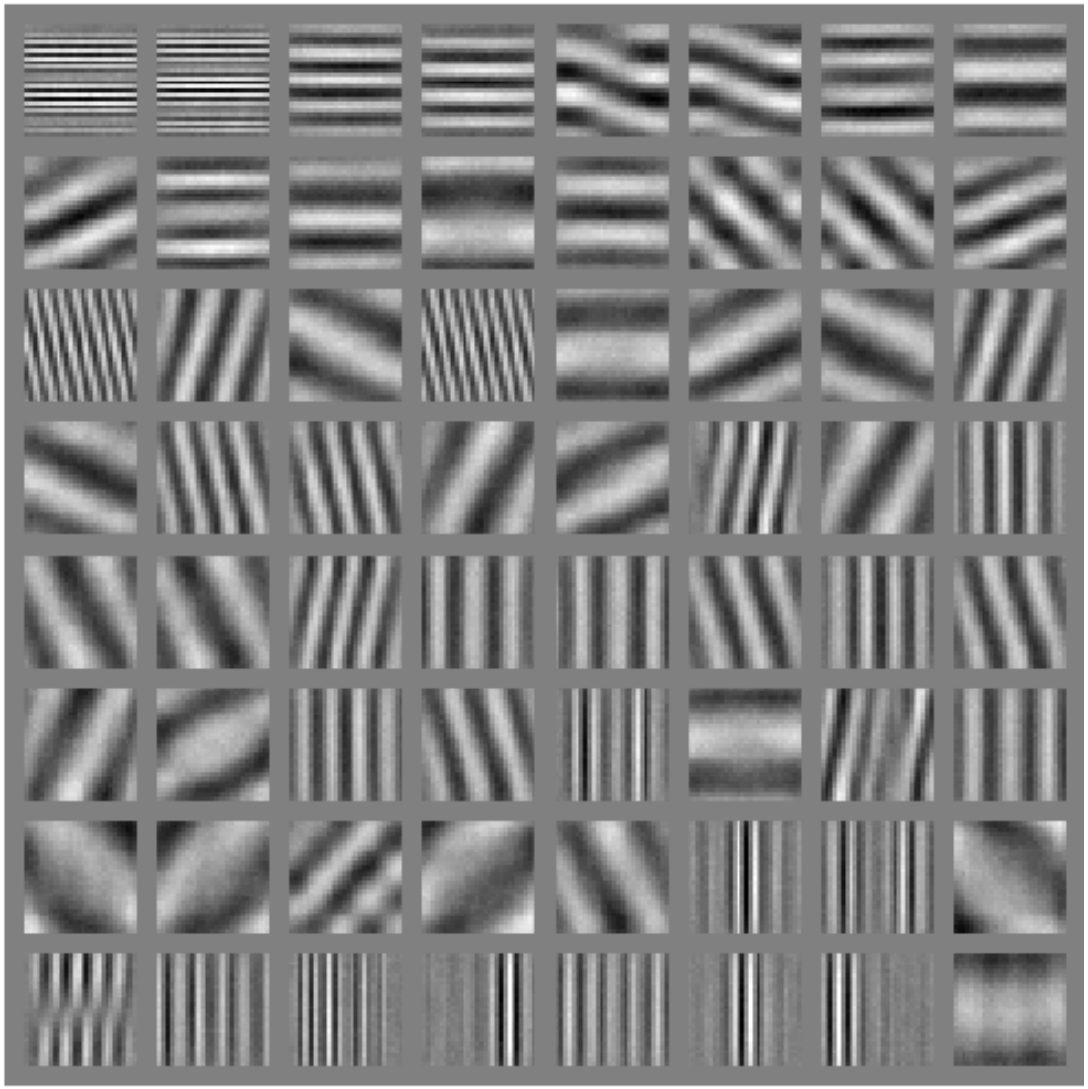
Rotation filters



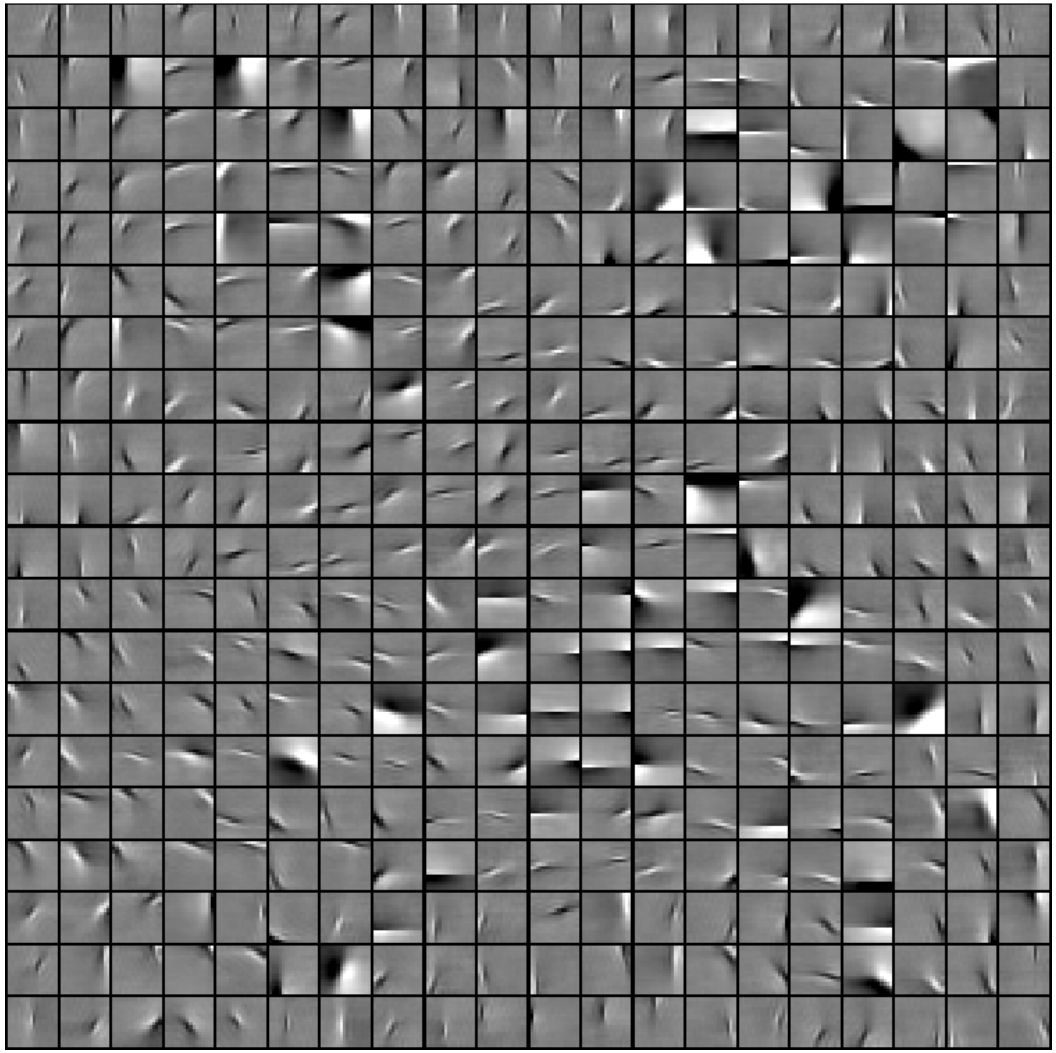
Filters learned by watching TV



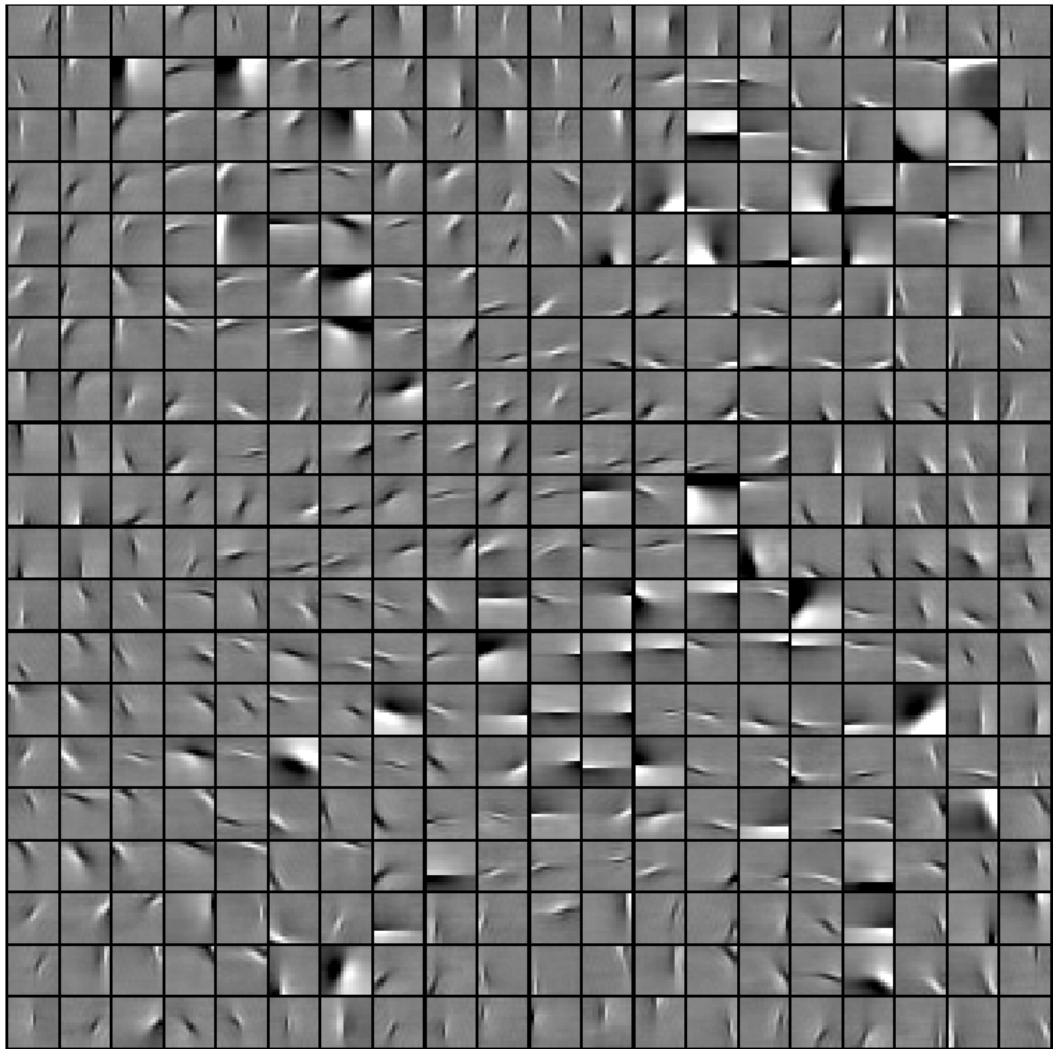
Filters learned by watching TV



“Bag-Of-Warps”



“Bag-Of-Warps”



Outline

Linear image warps

- Consider a linear transformation in pixel space ("*warp*"):

$$\boldsymbol{y} = L\boldsymbol{x}$$

- Now consider the following task:

> Given two images $\boldsymbol{x}$, $\boldsymbol{y}$, what is the warp that relates them?

- This is exactly the problem that mapping units should be able to solve.

Orthogonal image warps

$$\boldsymbol{y} = L\boldsymbol{x}$$

- We restrict our attention to **orthogonal warps** in the following, that is:

$$L^{\mathrm{T}} L = I$$

- These include all permutations ("shuffling pixels").
- Orthogonal warps are the *only* transformations we can see anyway, if all our images are *white*:

$$I = C_y = L C_x L^T = L L^T$$

- (Bethge, 2007)
- To get a better understanding of what mapping units really do, we make use of two properties of orthogonal image warps:

Properties of orthogonal image warps

(I) Orthogonal transformations decompose into 2-D rotations

- An orthogonal matrix is similar to a matrix that performs **axis-aligned two-dimensional rotations:**

$$V^{\mathrm{T}} L V = \begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_k \end{bmatrix} \qquad R_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix}$$

- This follows, for example, from the fact that the **eigen-decomposition**

$$L = V D V^{\mathrm{T}}$$

  has complex eigenvalues of length 1.
- The eigenspaces are also known as **invariant subspaces**.

Properties of orthogonal image warps

Example: Translation and the Fourier spectrum

- **Translation** is an example of an orthogonal warp.
- 1-D translation matrices are *circulants*, which have ones along an off-diagonal, like so:

$$L = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- The two-dimensional eigen-features of this matrix turn out to be sine-/cosine-pairs (Fourier features).

Properties of orthogonal image warps

Quadrature pairs

- Since the invariant subspaces of orthogonal warps are two-dimensional, **eigenvectors come in pairs**:

$$\left(\boldsymbol{v}_R, \boldsymbol{v}_I\right)$$

  They form an orthogonal basis for the invariant subspace.
- In the case of translation, $\boldsymbol{v}_I$ is a sine and $\boldsymbol{v}_R$ is a cosine feature.
- Waves with 90 degrees phase difference are known as "**quadrature pair**".
- But the concept is more general and applies to all orthogonal matrices.
- The eigenvector pairs of orthogonal transformations have been referred to as "**generalized quadrature pairs**" (Bethge et al., 2007).

Properties of commuting image warps

(II) Commuting transformations share an eigen-basis

- Any two transformations that commute share a single eigen-basis.
- They differ only in their eigen*values*.

- "Proof": Consider A and B with $AB = BA$ and the eigenvector v of B with λ an eigenvalue with multiplicity one. We have

$$BAv = ABv = \lambda Av.$$

So Av is also an eigenvector of B with the same eigenvalue.

Properties of commuting image warps

Translation Example continued

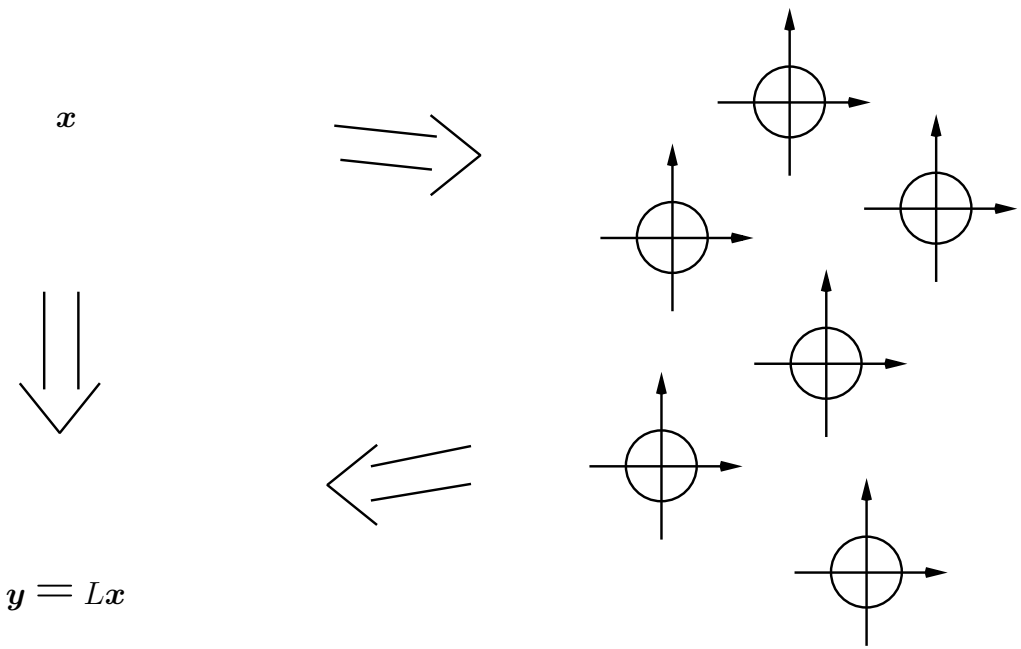
- *All* circulants have the Fourier basis as eigen-basis.

- Properties (I) and (II) taken together now allow us to state the following:

Properties of commuting image warps

Any two **orthogonal, commuting** transformations differ only with respect to the **rotation angles in the eigenpaces**.

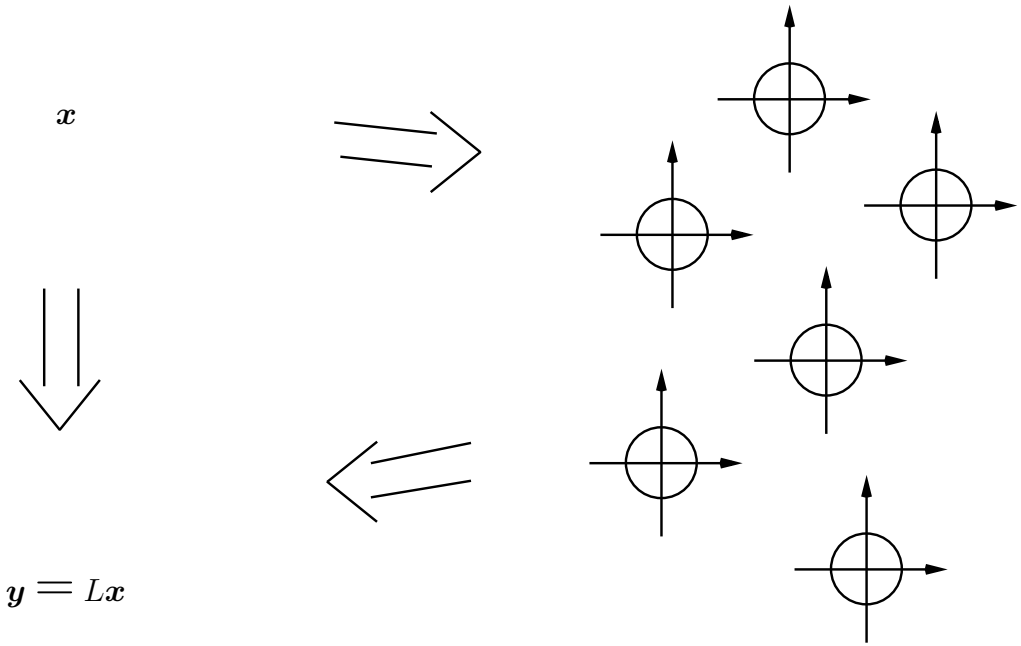
- So to *apply* a transformation you can equivalently perform a set of independent two-D rotations.



Properties of commuting image warps

Any two **orthogonal, commuting** transformations differ only with respect to the **rotation angles in the eigenpaces**.

- So to *apply* a transformation you can equivalently perform a set of independent two-D rotations.

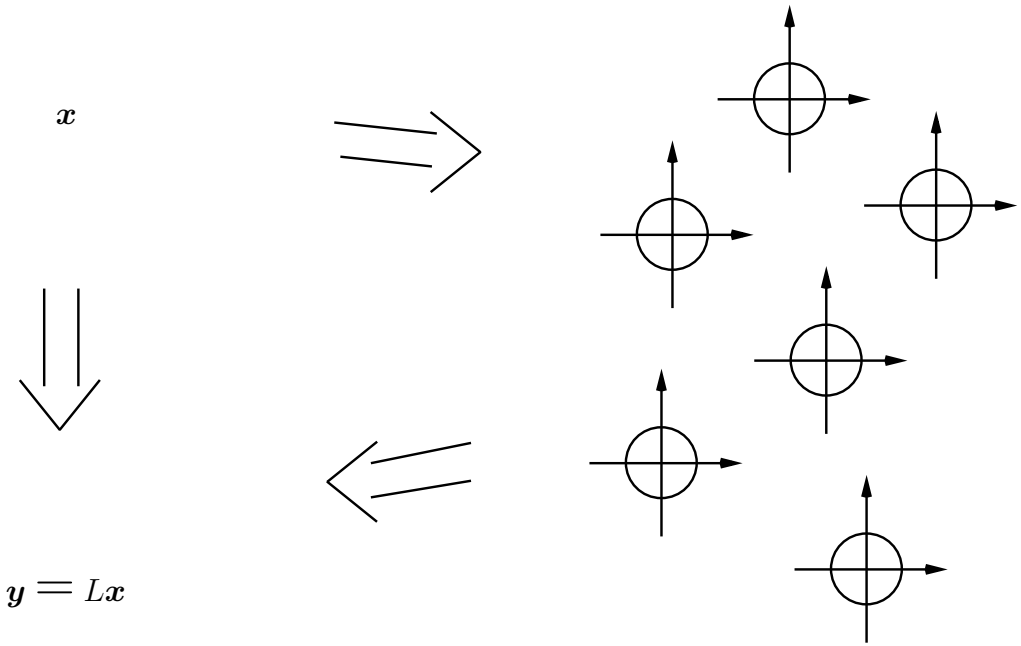


- To *infer* the transformation, given two images x and y: Project x and y onto the eigenvectors, then compute the rotation angles!

Properties of commuting image warps

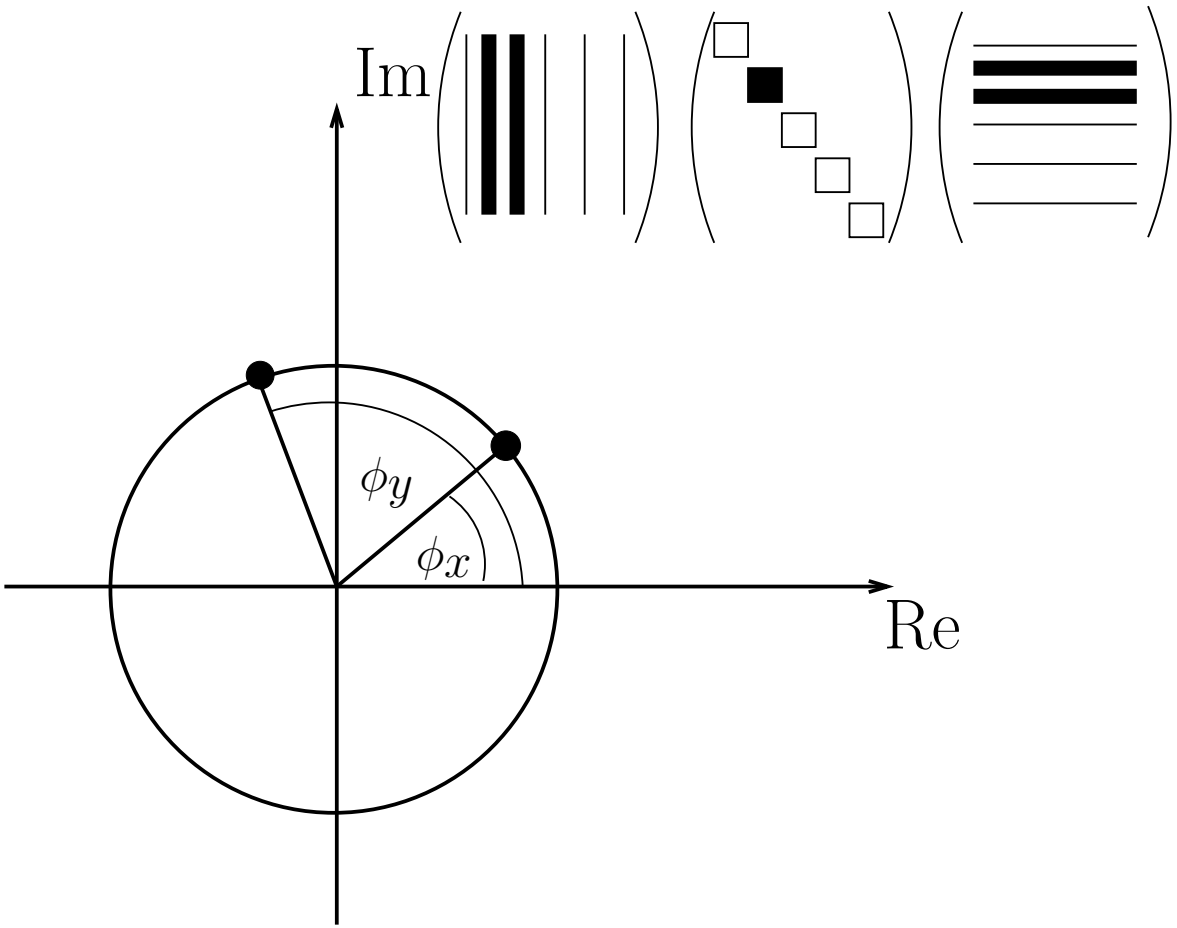
Any two **orthogonal, commuting** transformations differ only with respect to the **rotation angles in the eigenpaces**.

- So to *apply* a transformation you can equivalently perform a set of independent two-D rotations.



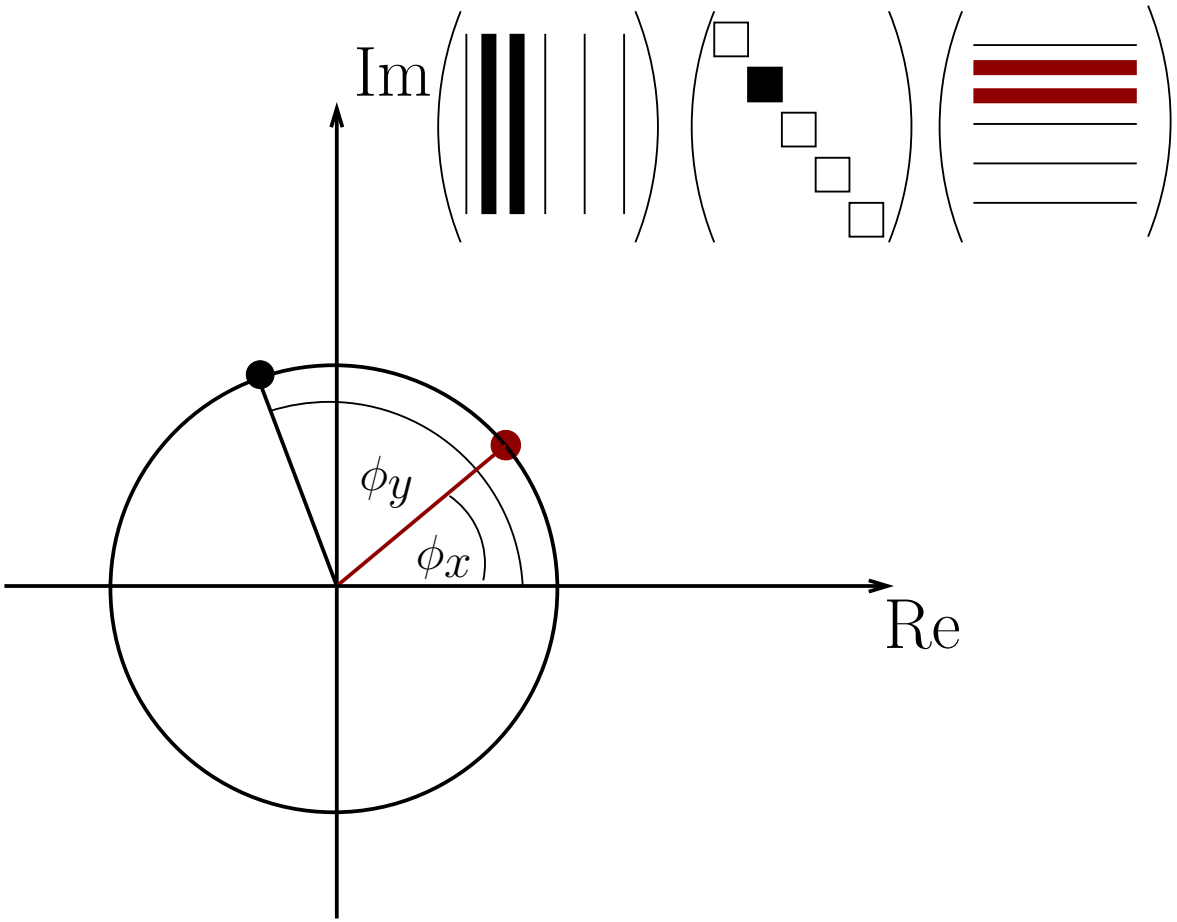
- To *infer* the transformation, given two images x and y: Project x and y onto the eigenvectors, then compute the rotation angles!

Extracting sub-space rotations, naive approach



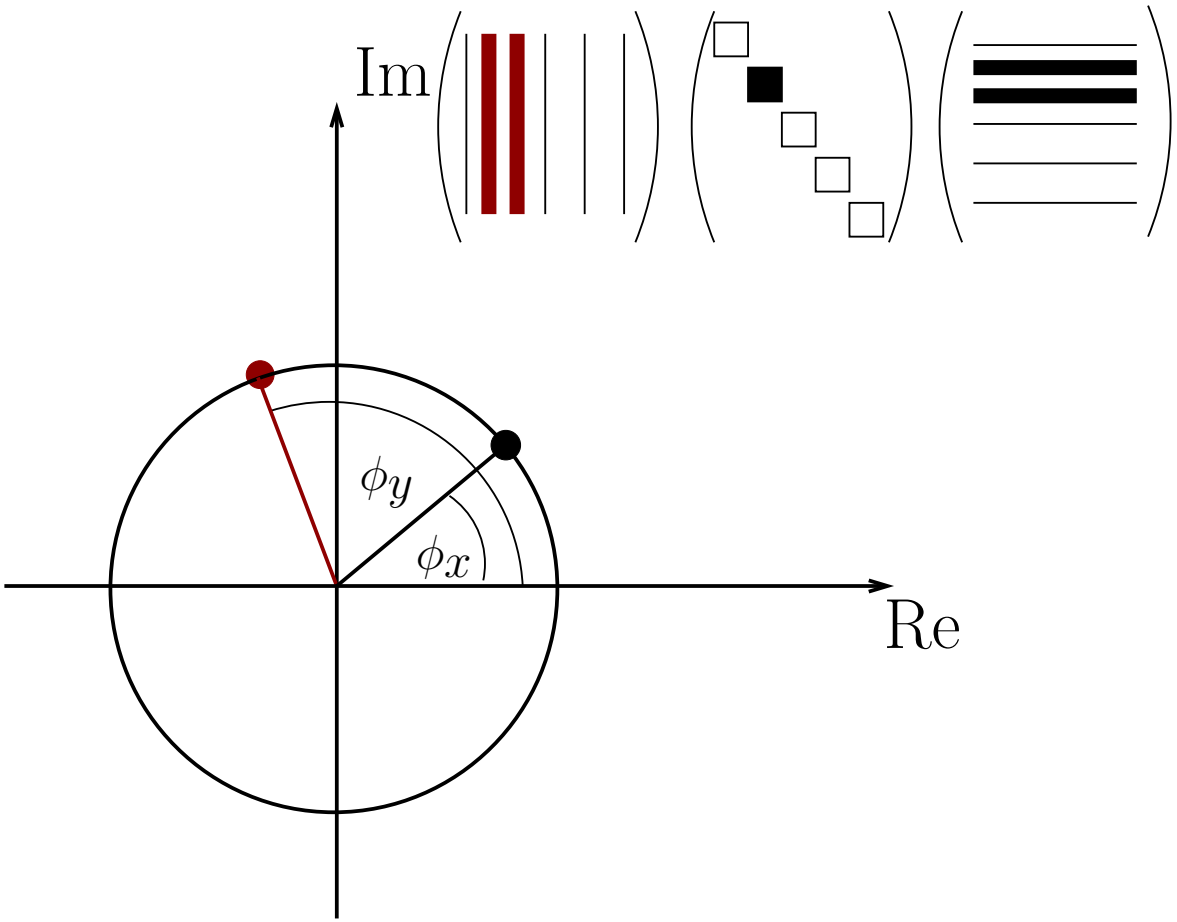
- In each subspace:
- Normalize the 2-D projections to unit norm, then read off the angle between them.

Extracting sub-space rotations, naive approach



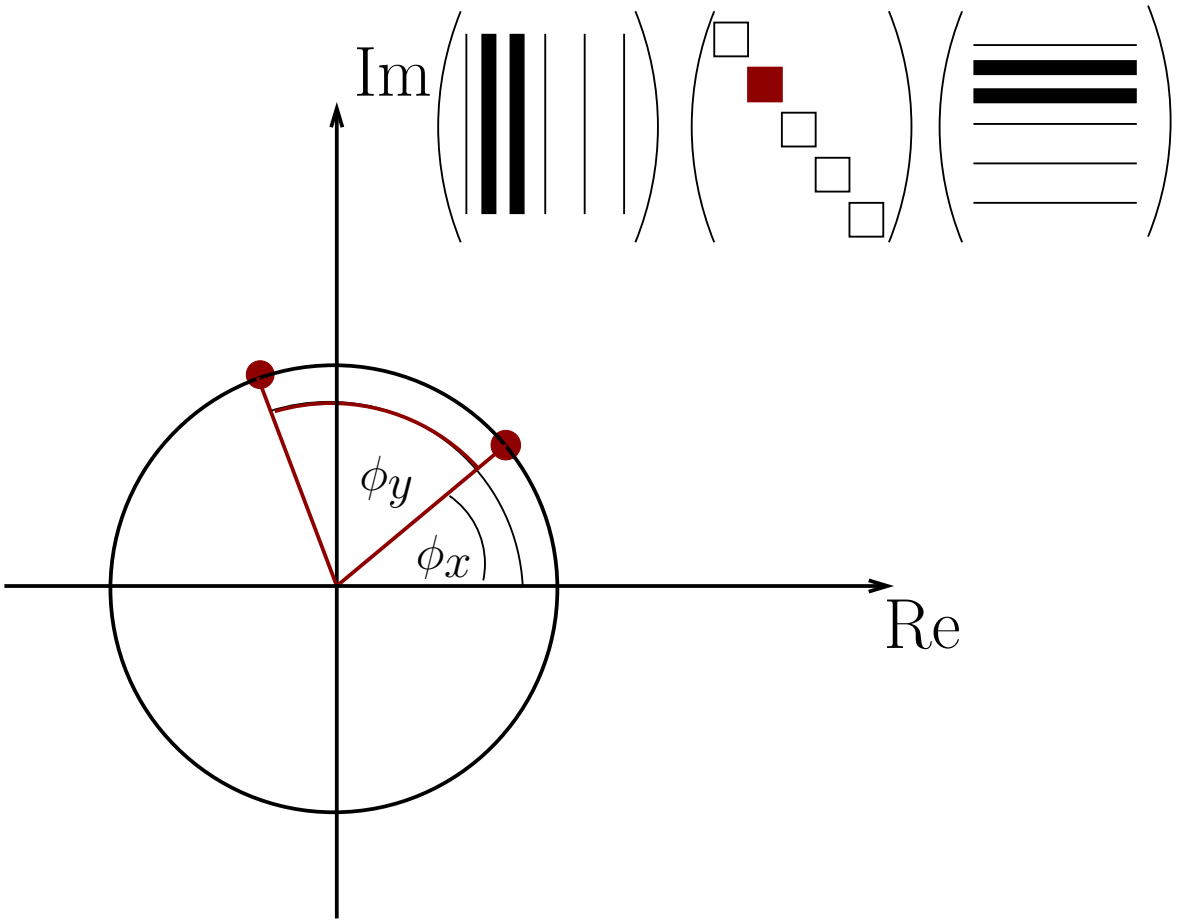
- In each subspace:
- Normalize the 2-D projections to unit norm, then read off the angle between them.

Extracting sub-space rotations, naive approach



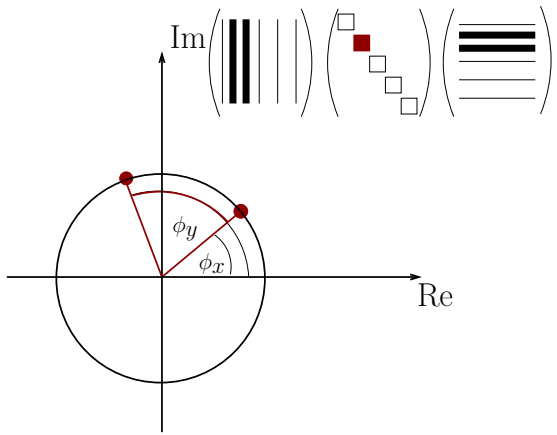
- In each subspace:
- Normalize the 2-D projections to unit norm, then read off the angle between them.

Extracting sub-space rotations, naive approach



- In each subspace:
- Normalize the 2-D projections to unit norm, then read off the angle between them.

Extracting sub-space rotations, naive approach



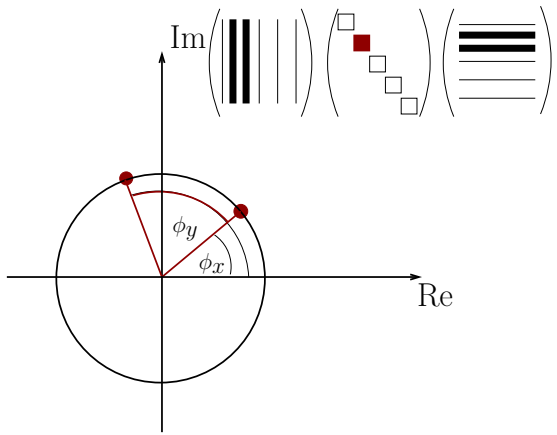
Extracting rotations by computing angles

- To read off the angle, compute the **inner product** (after normalizing projections to unit-norm).
- Formally,

$$\begin{aligned}\cos(\phi_y - \phi_x) &= \cos\phi_y \cos\phi_x + \sin\phi_y \sin\phi_x \\ &= (\boldsymbol{v}_R{}^{\mathrm{T}}\boldsymbol{y})(\boldsymbol{v}_R{}^{\mathrm{T}}\boldsymbol{x}) + (\boldsymbol{v}_I{}^{\mathrm{T}}\boldsymbol{y})(\boldsymbol{v}_I{}^{\mathrm{T}}\boldsymbol{x})\end{aligned}$$

- Compute the sum over products of filter responses.

Extracting sub-space rotations, naive approach



Extracting rotations by computing angles

- To read off the angle, compute the **inner product** (after normalizing projections to unit-norm).
- Formally,

$$\begin{aligned} \cos(\phi_y - \phi_x) &= \cos\phi_y \cos\phi_x + \sin\phi_y \sin\phi_x \\ &= (\boldsymbol{v}_R{}^{\mathrm{T}}\boldsymbol{y})(\boldsymbol{v}_R{}^{\mathrm{T}}\boldsymbol{x}) + (\boldsymbol{v}_I{}^{\mathrm{T}}\boldsymbol{y})(\boldsymbol{v}_I{}^{\mathrm{T}}\boldsymbol{x}) \end{aligned}$$

- Compute the sum over products of filter responses.

Sub-space rotation detectors

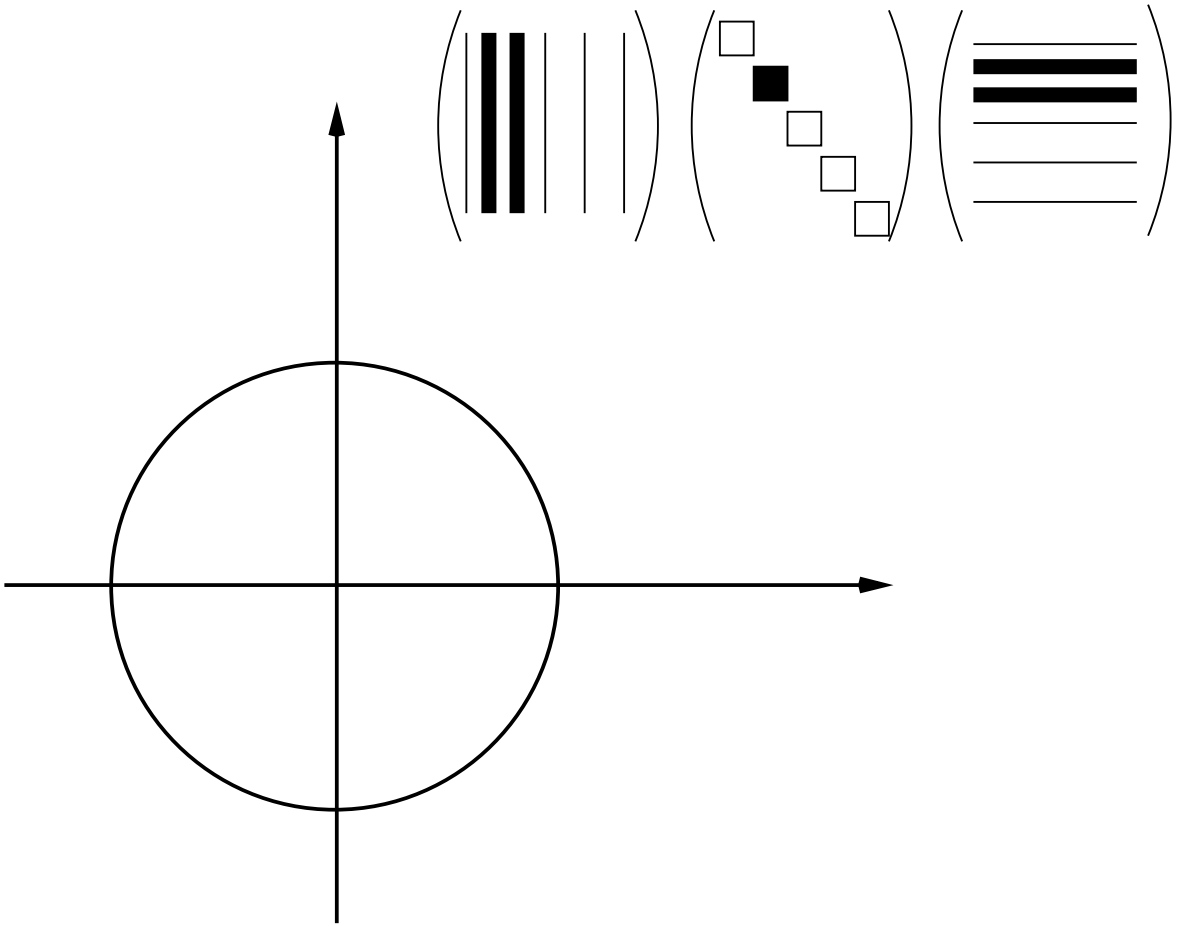
- Normalizing to unit norm can be a bad idea, if projections are small:

The aperture problem

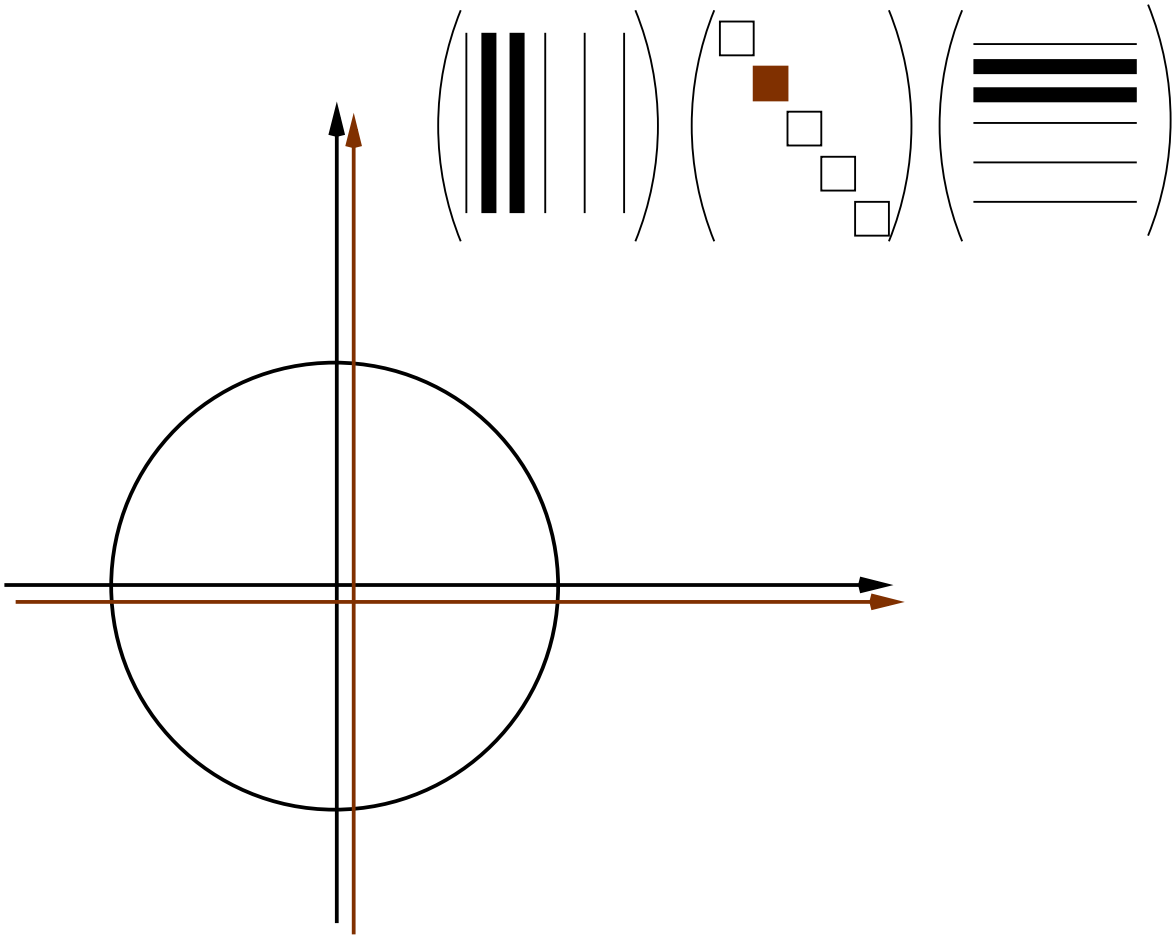
- Consider the left shift of a horizontal bar.
- It is impossible to see the transformation in this case.
- This is known as the **aperture problem**.
- Normalizing subspace projections would amount to pretending we could see the transformation!

- A second way to get the rotations:
- Absorb the rotation into one of the eigenvectors, then try to *detect* rotation angles.

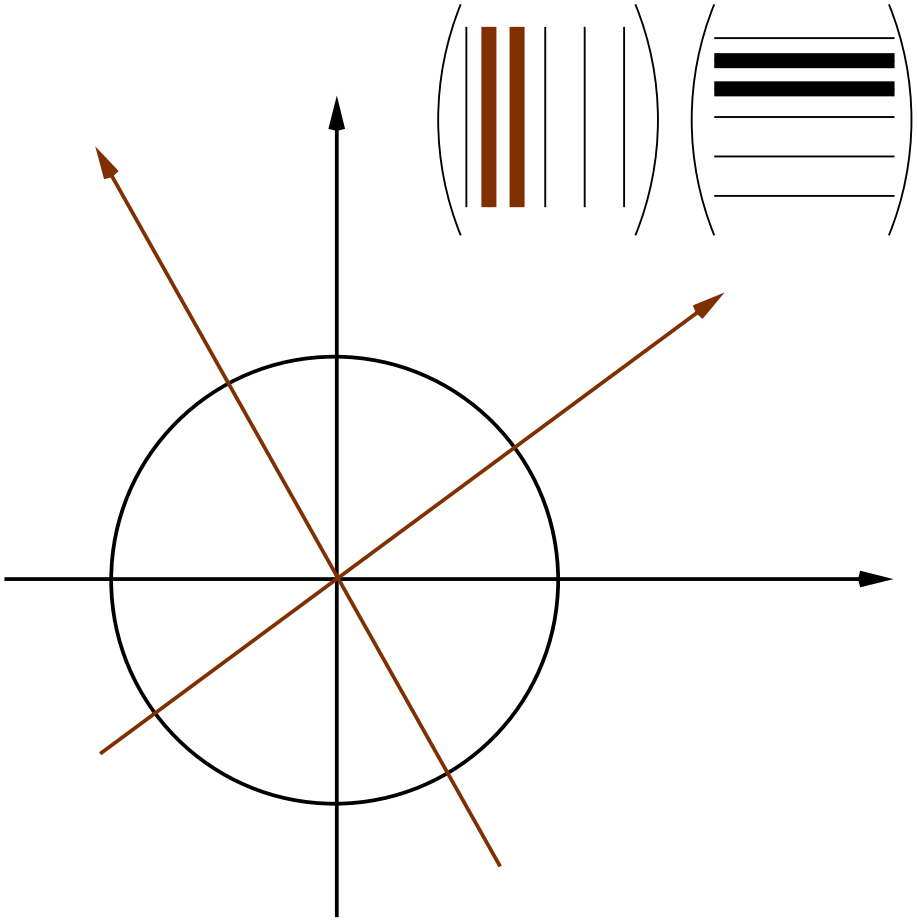
Sub-space rotation detectors



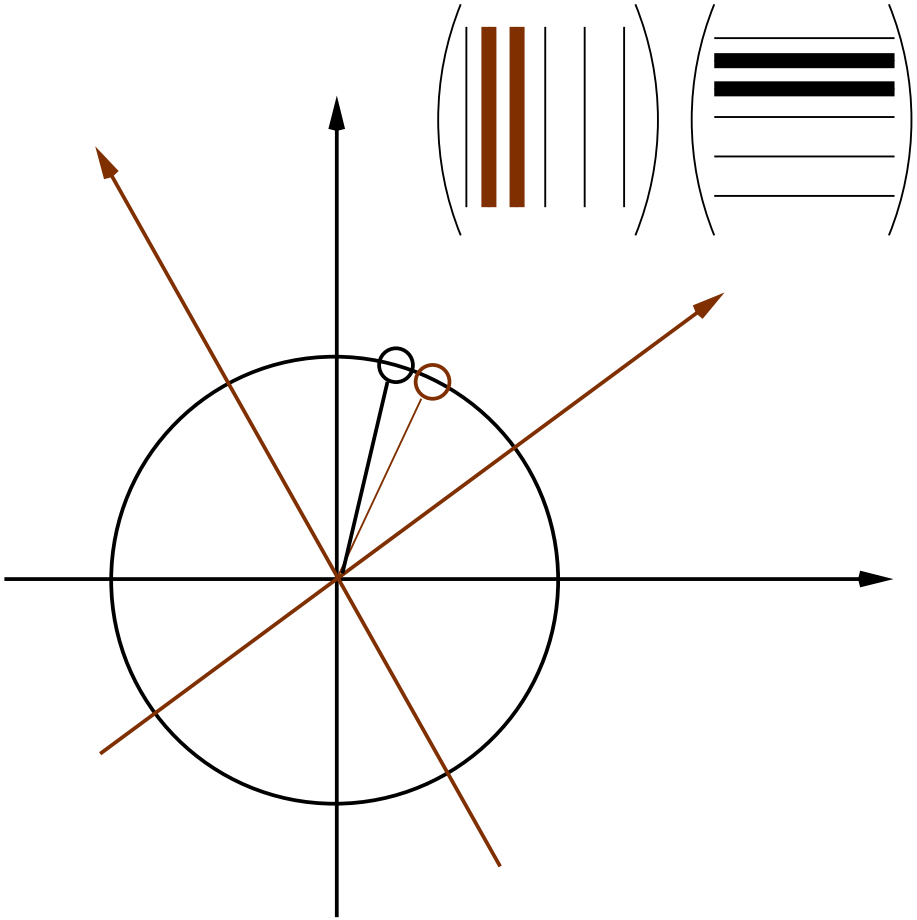
Sub-space rotation detectors



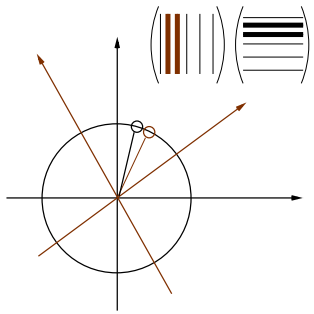
Sub-space rotation detectors



Sub-space rotation detectors



Sub-space rotation detectors



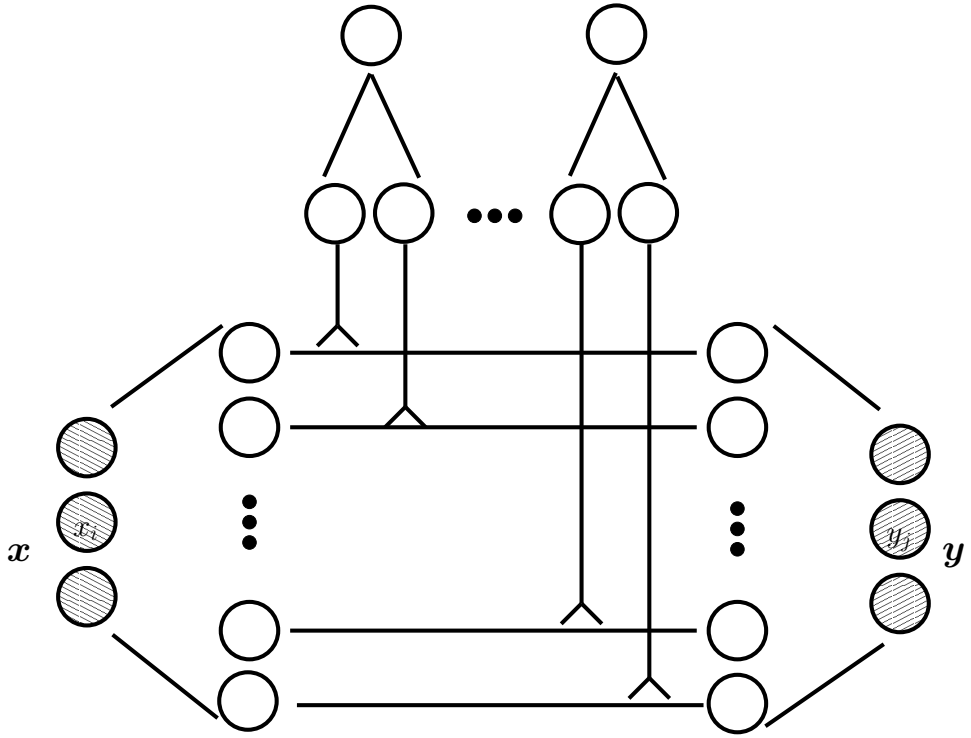
Extracting rotations by detecting angles

- Formally, let the output filter pair $\boldsymbol{v}_R^\theta, \boldsymbol{v}_I^\theta$ be the input filter rotated by θ degrees (in complex notation: $\boldsymbol{v}^\theta = \exp(i\theta)\boldsymbol{v}$).
- Measure how well the image pair $\boldsymbol{x}, \boldsymbol{y}$ conforms with this rotation:

$$\begin{aligned} r^\theta &:= \cos(\phi_y - \phi_x - \theta) \\ &= \cos(\phi_y)\cos(\phi_x + \theta) + \sin(\phi_y)\sin(\phi_x + \theta) \\ &= (\boldsymbol{v}_R^{\theta\,\mathrm{T}}\boldsymbol{y})(\boldsymbol{v}_R{}^{\mathrm{T}}\boldsymbol{x}) + (\boldsymbol{v}_I^{\theta\,\mathrm{T}}\boldsymbol{y})(\boldsymbol{v}_I{}^{\mathrm{T}}\boldsymbol{x}) \end{aligned}$$

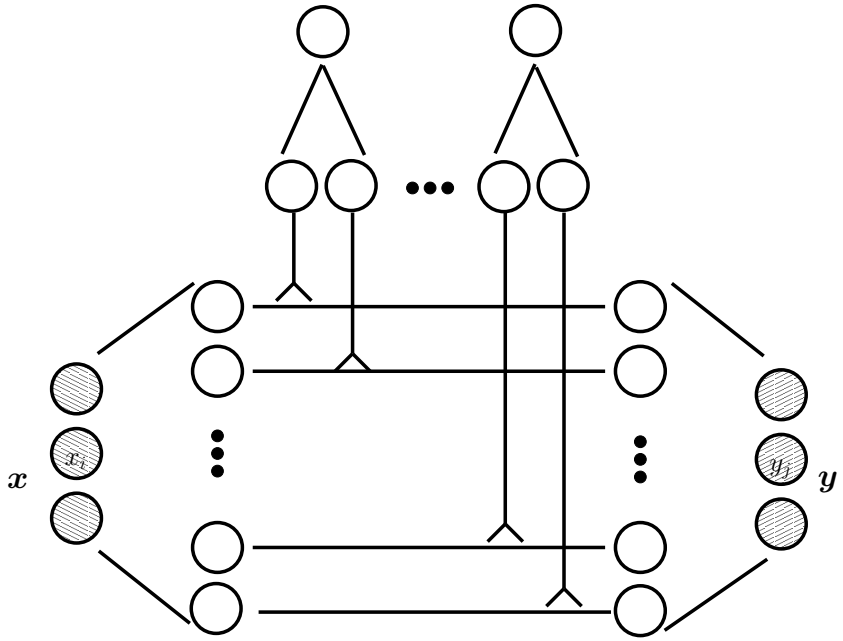
- Again we have to sum over products of filter responses.

Sub-space rotation detectors



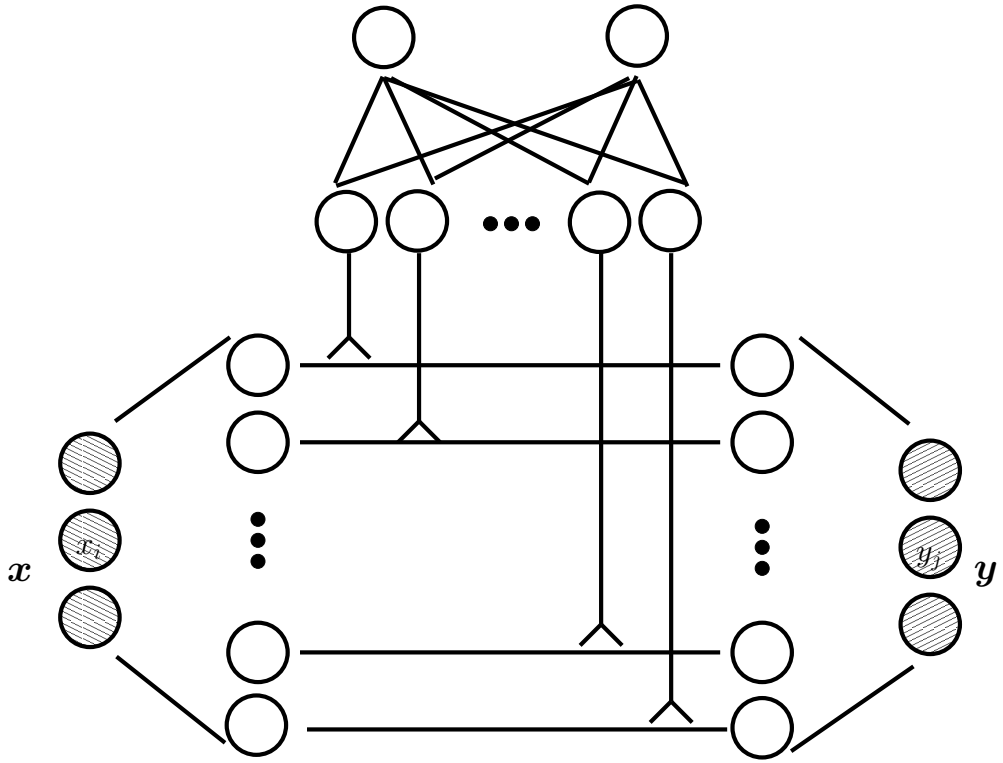
- For each subspace, we will need several mapping units, each tuned to a different angle, θ_i.
- The set of mapping unit responses will now constitute a **population code** that represents the observed transformation.
- A mapping unit is *conservative*: It fires only if a transform is present *and* if it is visible in the image pair.

Subspace rotation detector graphical model



- But the aperture problem causes another problem:
- Take a video showing translations and generate two copies:
- Low-pass filter each frame in the first; High-pass filter each frame in the second.
- Now the transformation will be visible only in some components in the first and in other components in the second video.
- These subspace features are content-dependent!

Subspace rotation detector graphical model



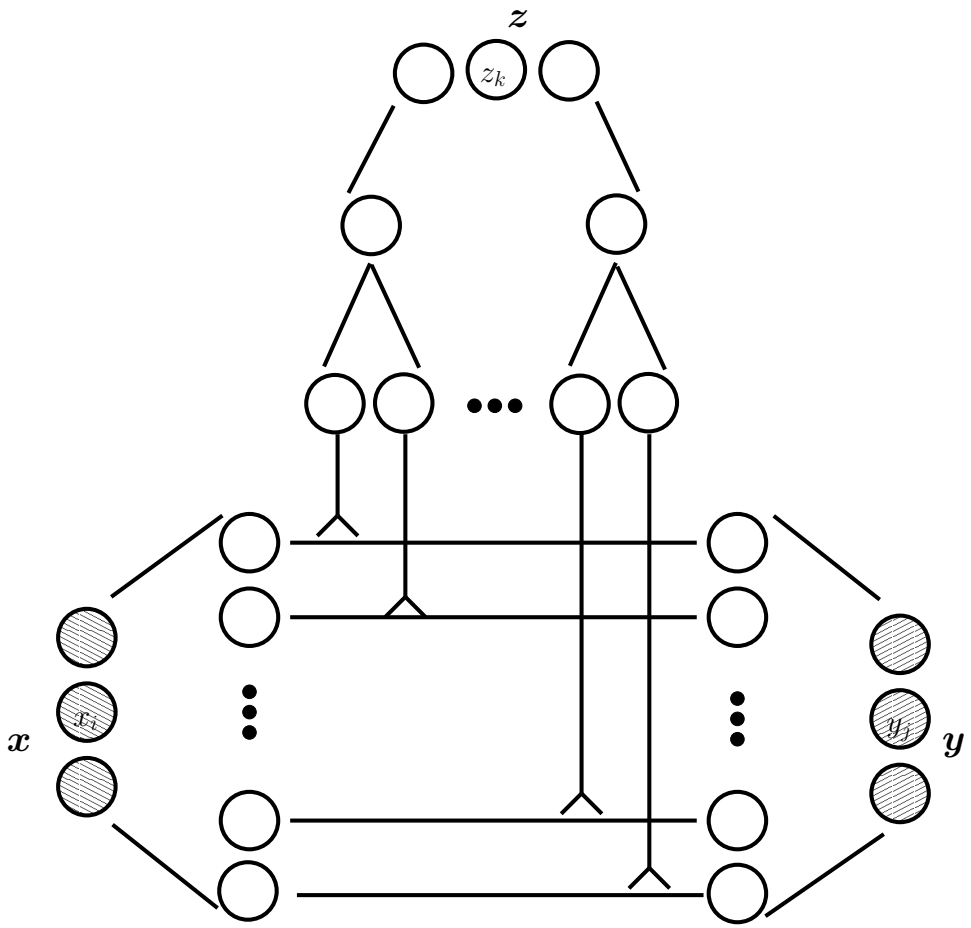
- The solution:
- **Let hiddens pool within *and* pool across subspaces.**
- This is exactly the factored bilinear model.

Summary: Learning relation-detectors

The cross-correlation model

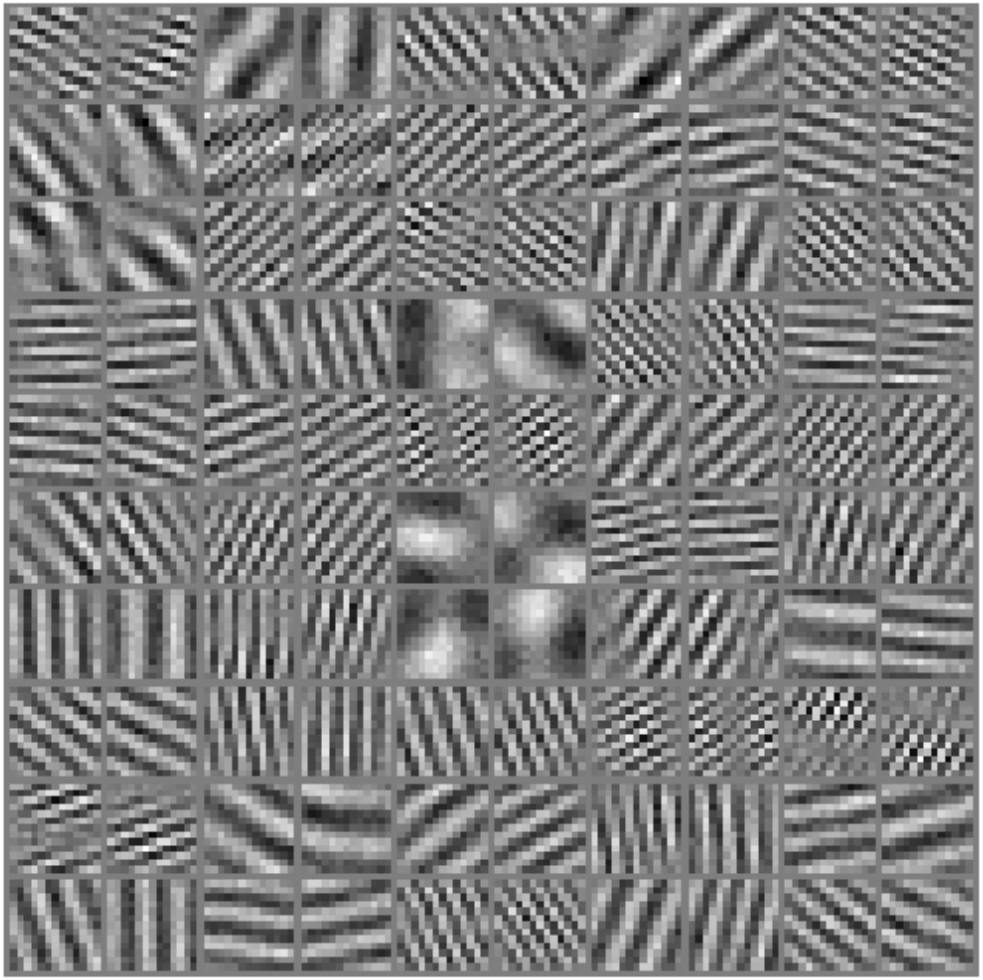
- A hidden variable that computes the sum over products of filter responses can detect rotations, θ, in an invariant subspace.
- To reconstruct the transformed output from the input image, it has to pool over multiple 2-dimensional subspaces.
- The population code of such hiddens is a good code for image transformations.
- Learning requires **contrast normalization** + **keeping the scales of filters roughly the same**!

Learning quadrature features

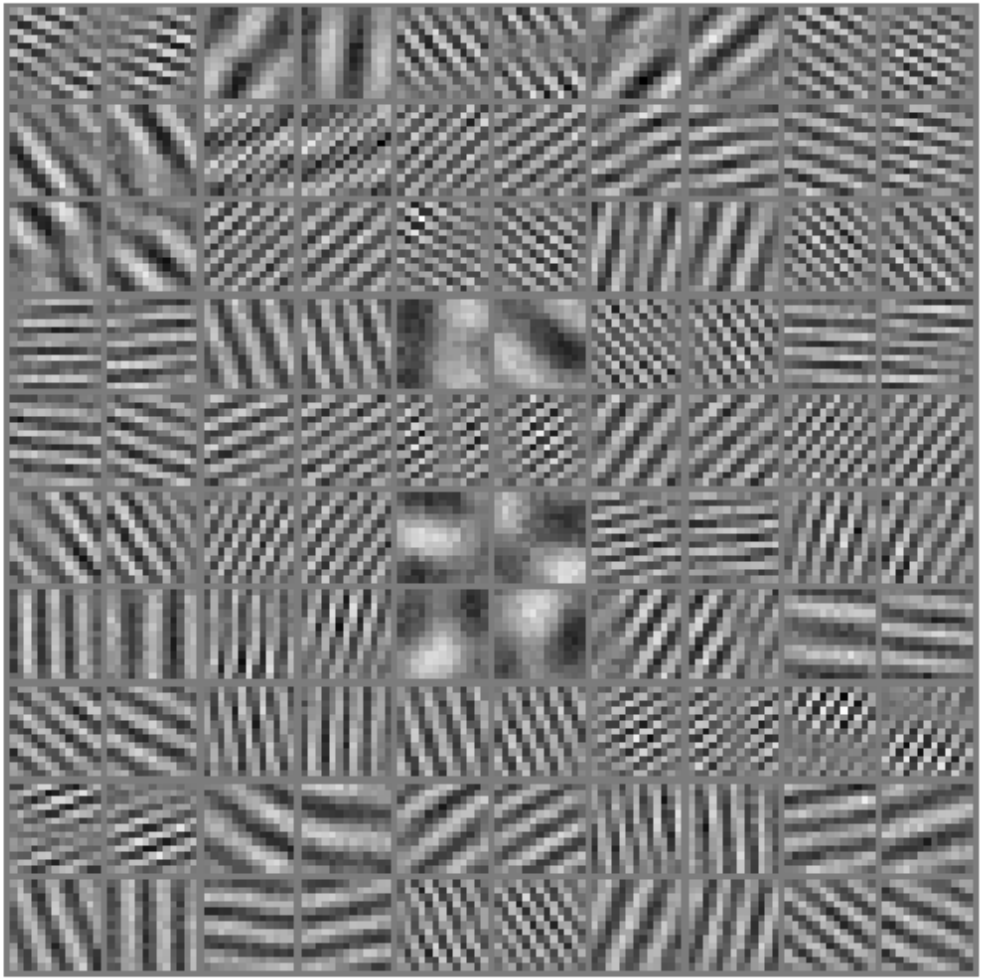


- We can see the quadrature features, if we outsource the across-subspace pooling into a separate layer.

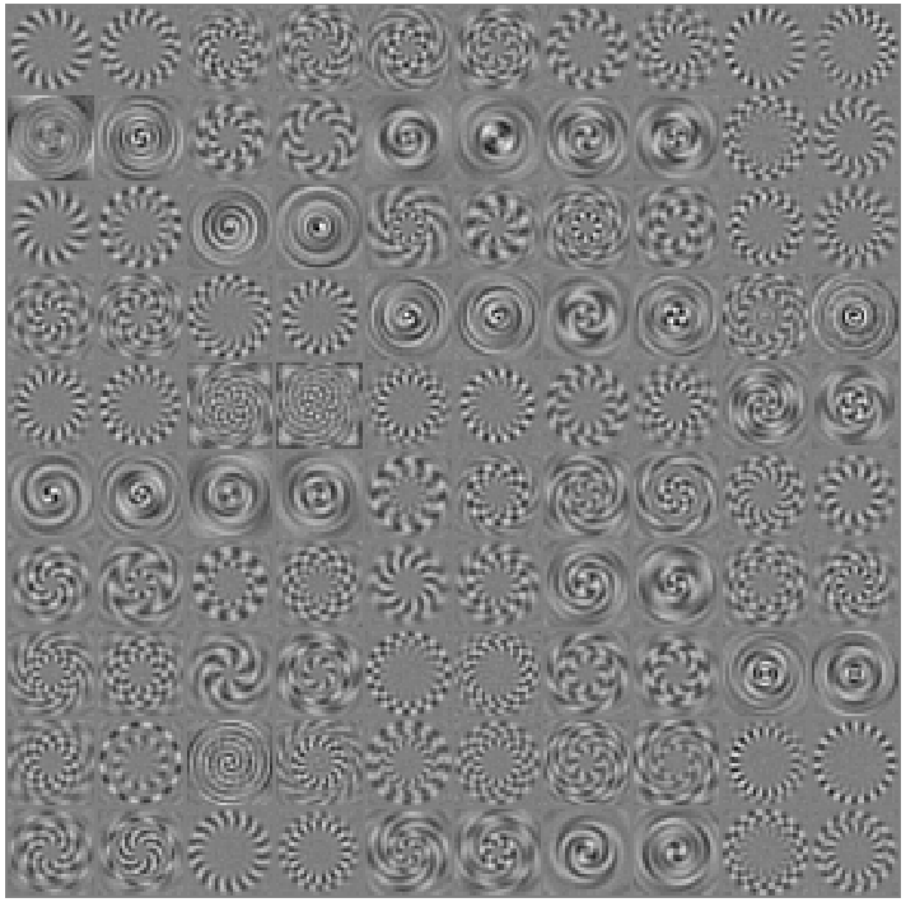
Learning quadrature Features



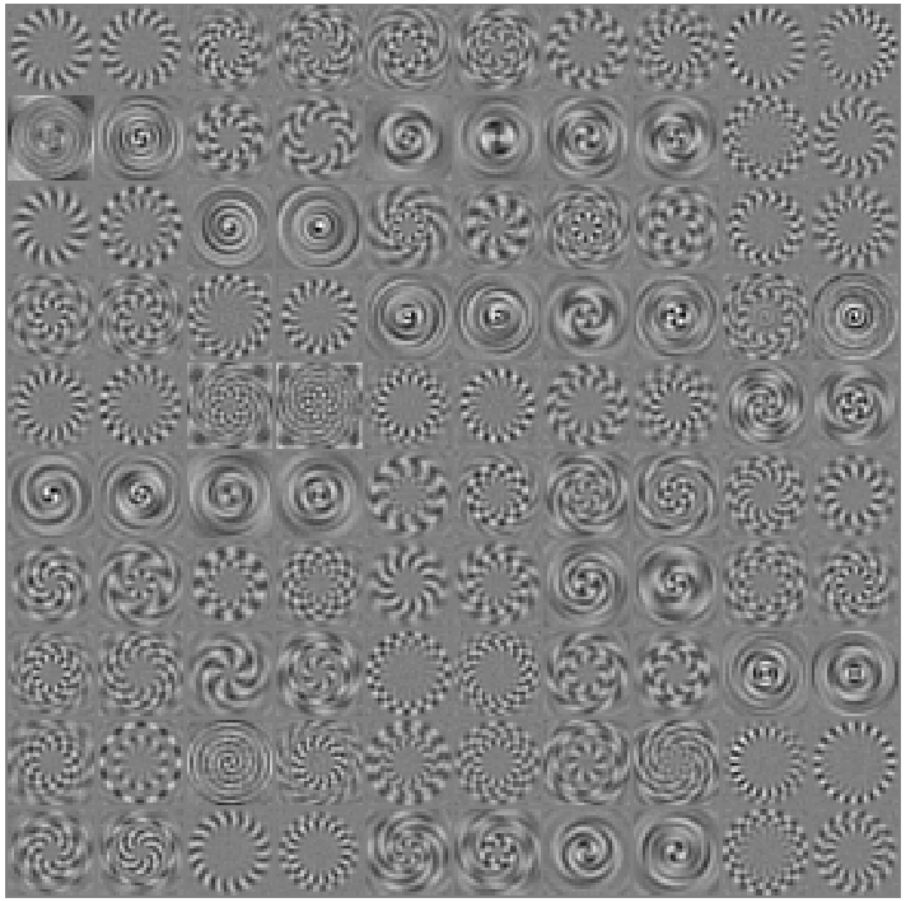
Learning quadrature Features



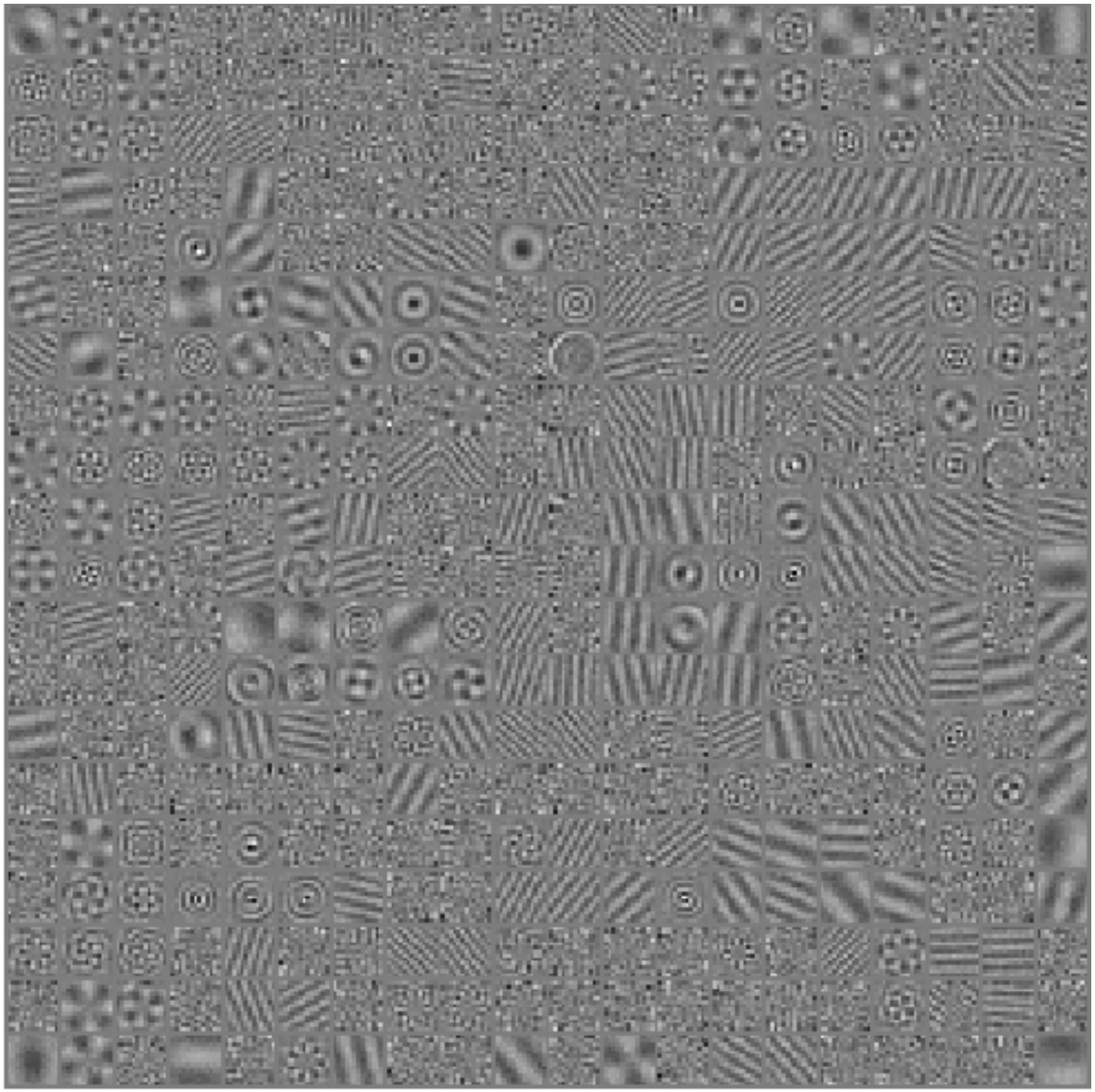
Rotation “quadrature” filters



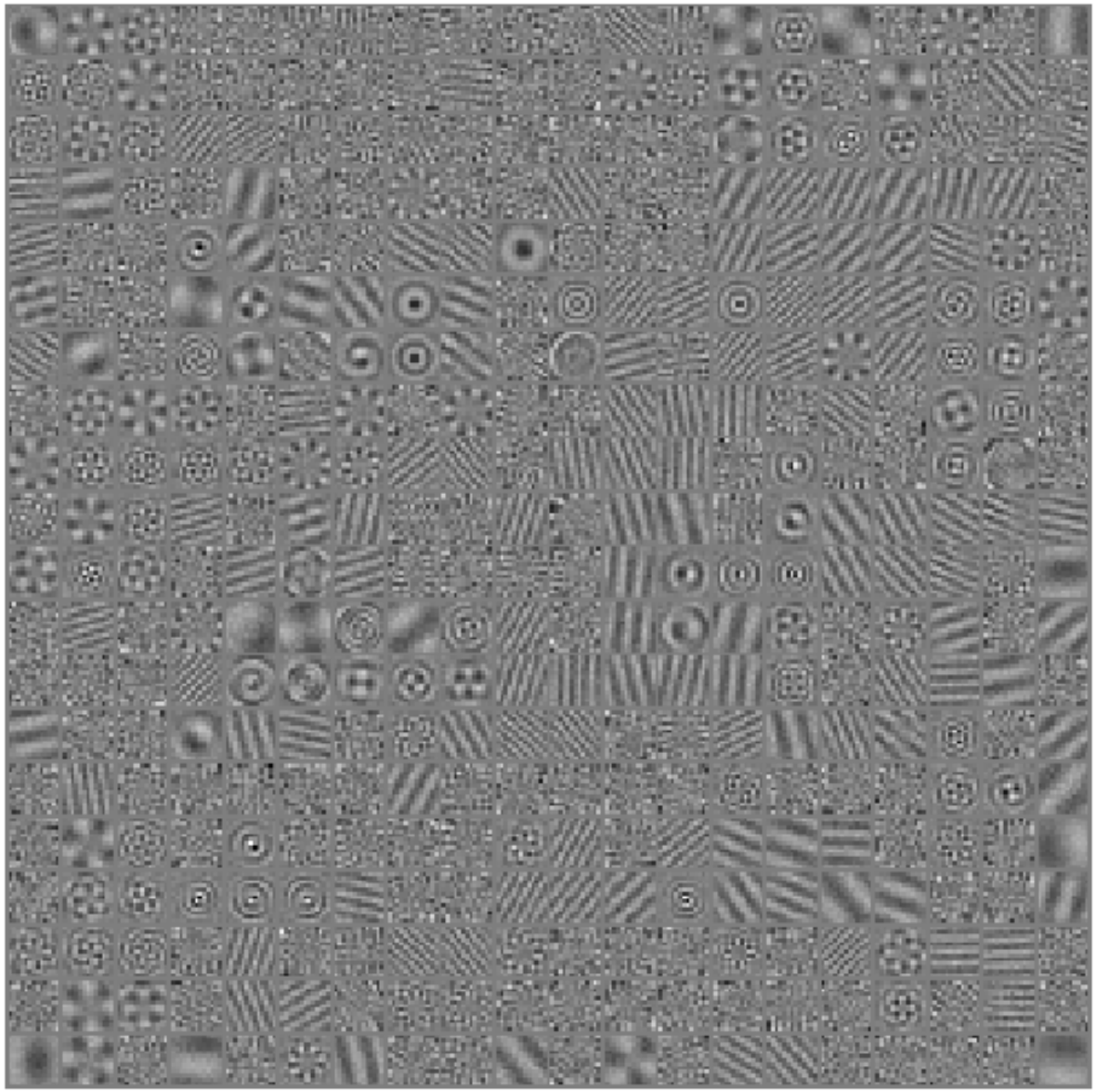
Rotation “quadrature” filters



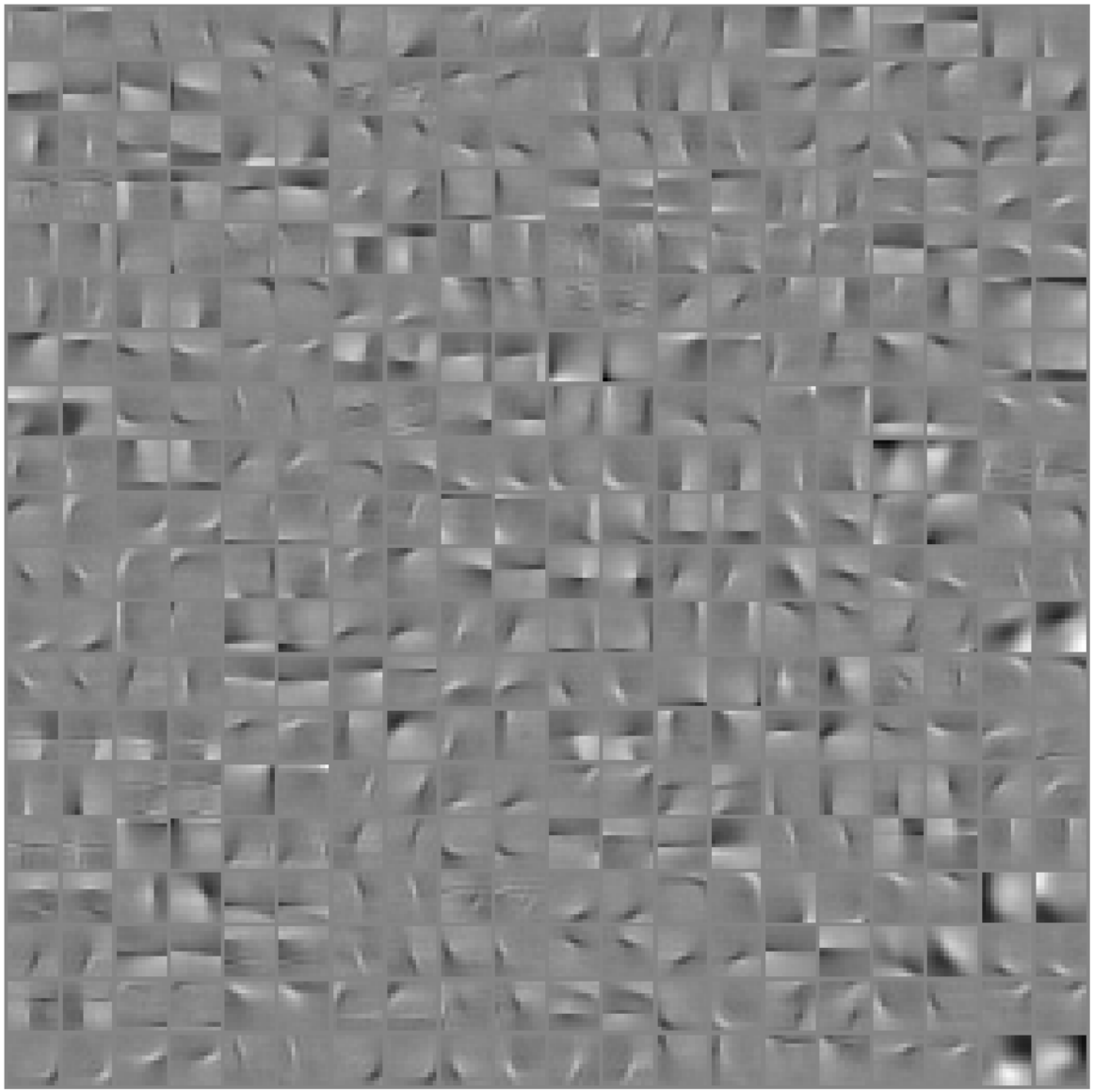
Mixed transformations



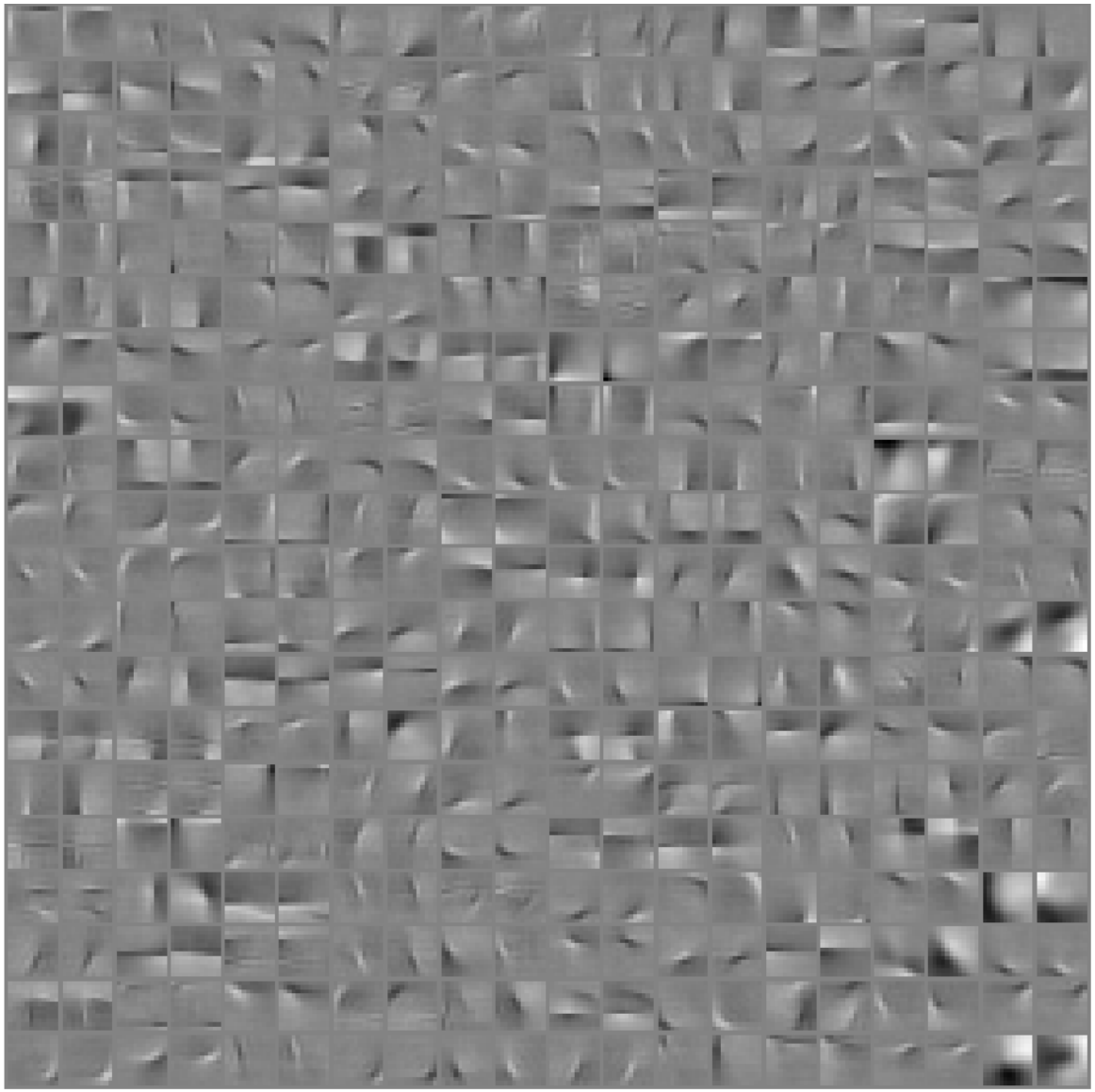
Mixed transformations



Quadrature features from natural video

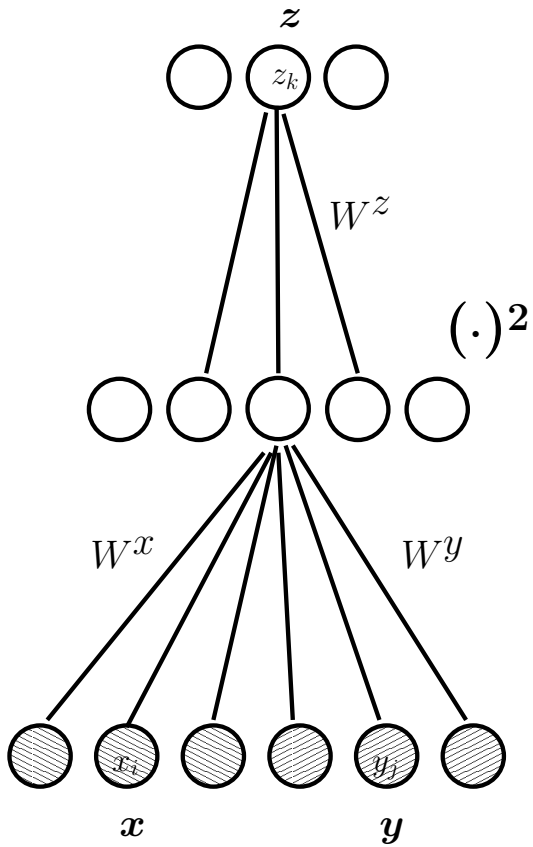


Quadrature features from natural video



Energy models

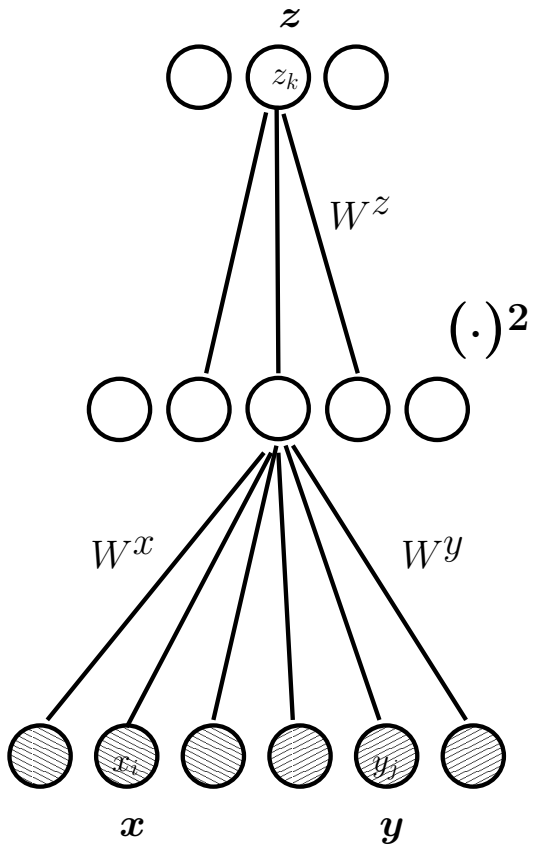
$$
\begin{aligned}
z_k &= \sum_f w_{fk} \left(\boldsymbol{u}_f^{\mathrm{T}} \boldsymbol{x} + \boldsymbol{v}_f^{\mathrm{T}} \boldsymbol{y} \right)^2 \\
&= 2 \sum_f w_{fk} \left(\boldsymbol{u}_f^{\mathrm{T}} \boldsymbol{x} \right) \left(\boldsymbol{v}_f^{\mathrm{T}} \boldsymbol{y} \right) \\
&+ \sum_f w_{fk} \left(\boldsymbol{u}_f^{\mathrm{T}} \boldsymbol{x} \right)^2 + \sum_f w_{fk} \left(\boldsymbol{v}_f^{\mathrm{T}} \boldsymbol{y} \right)^2
\end{aligned}
$$



- When we apply energy models to the **concatenation of two images**, we add square terms in inference.
- This may make the rotation detectors more conservative. Otherwise inference is the same!

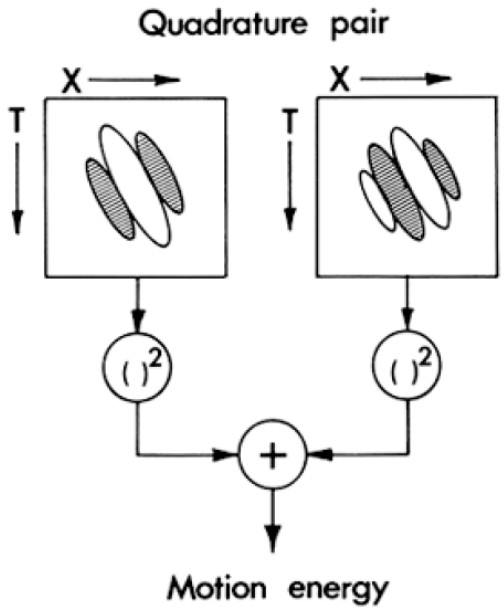
Energy models

$$
\begin{aligned}
z_k &= \sum_f w_{fk} \left(\boldsymbol{u}_f^{\mathrm{T}} \boldsymbol{x} + \boldsymbol{v}_f^{\mathrm{T}} \boldsymbol{y} \right)^2 \\
&= 2 \sum_f w_{fk} \left(\boldsymbol{u}_f^{\mathrm{T}} \boldsymbol{x} \right) \left(\boldsymbol{v}_f^{\mathrm{T}} \boldsymbol{y} \right) \\
&+ \sum_f w_{fk} \left(\boldsymbol{u}_f^{\mathrm{T}} \boldsymbol{x} \right)^2 + \sum_f w_{fk} \left(\boldsymbol{v}_f^{\mathrm{T}} \boldsymbol{y} \right)^2
\end{aligned}
$$



- When we apply energy models to the **concatenation of two images**, we add square terms in inference.
- This may make the rotation detectors more conservative. Otherwise inference is the same!

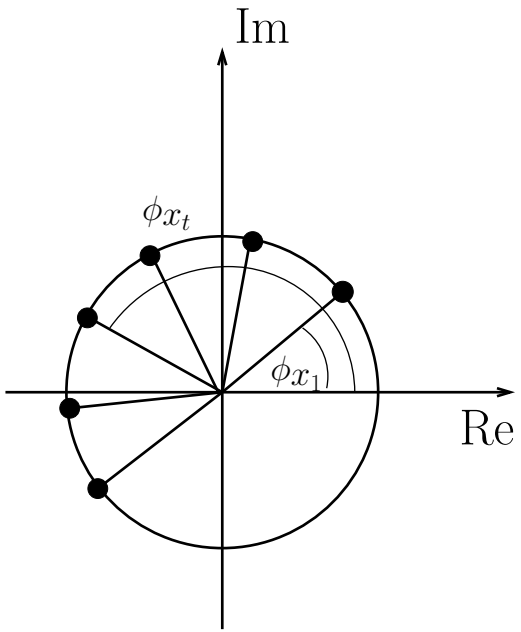
Energy models



The energy model

- (Adelson and Bergen, 1985): Motion
- (Ozhawa, DeAngelis, Freeman; 1990): Disparity
- Equivalence to cross-correlation: See, for example, (Fleet et al.; 1994).

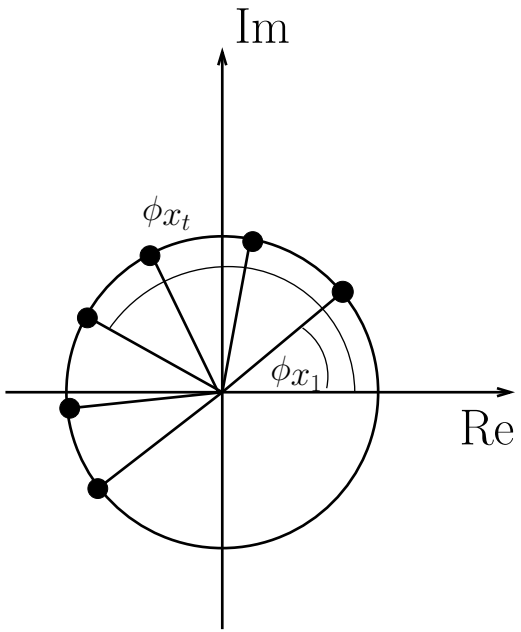
Learning energy models on movies



- What happens when we train energy models on movies?
- Hiddens receive all pairs of products between filters applied to frames.
- So they detect the **repeated application of the same eigenvalue**:

$$\left(\sum_s \boldsymbol{v}^{s\mathrm{T}}\boldsymbol{x}_s\right)^2 = \sum_s \left(\boldsymbol{v}^{s\mathrm{T}}\boldsymbol{x}_s\right)^2 + \sum_{st} \left(\boldsymbol{v}^{s\mathrm{T}}\boldsymbol{x}_s\right)\cdot\left(\boldsymbol{v}^{t\mathrm{T}}\boldsymbol{x}_t\right)$$

Learning energy models on movies



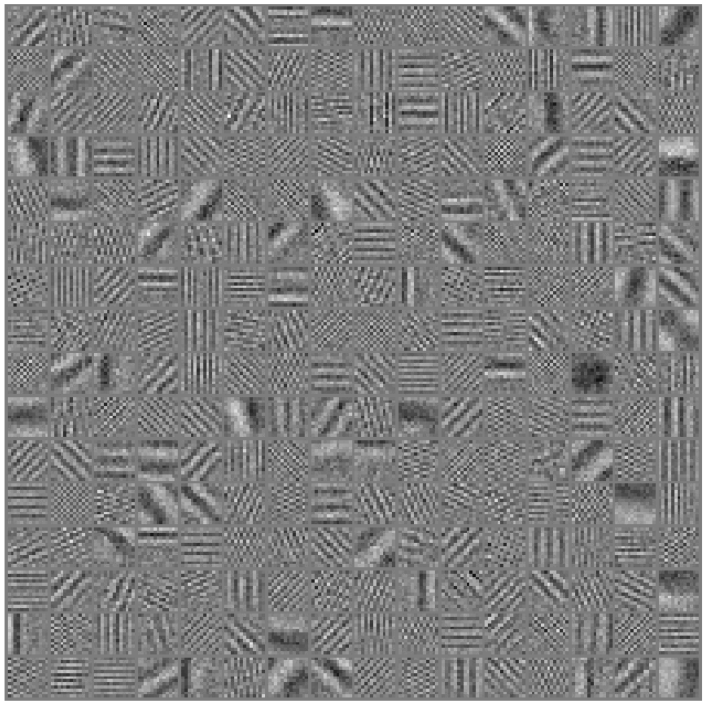
- What happens when we train energy models on movies?
- Hiddens receive all pairs of products between filters applied to frames.
- So they detect the **repeated application of the same eigenvalue**:

$$\left(\sum_s \boldsymbol{v}^{s\mathrm{T}} \boldsymbol{x}_s\right)^2 = \sum_s \left(\boldsymbol{v}^{s\mathrm{T}} \boldsymbol{x}_s\right)^2 + \sum_{st} \left(\boldsymbol{v}^{s\mathrm{T}} \boldsymbol{x}_s\right) \cdot \left(\boldsymbol{v}^{t\mathrm{T}} \boldsymbol{x}_t\right)$$

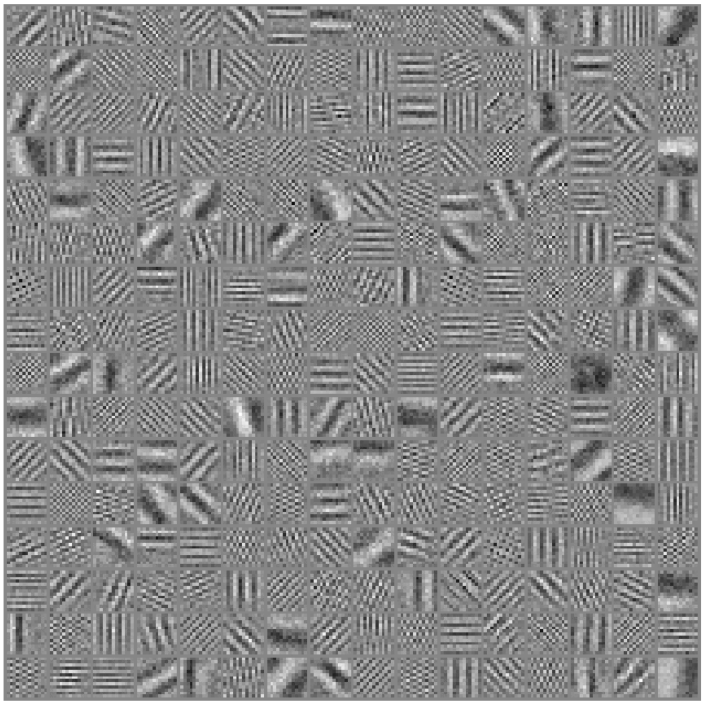
Training energy models via gating

- We can train a cross-correlation model via the energy mechanism.
- But we can do the opposite, too:
- Plug in *the same* data left and right and tie left and right filters.
- So we don't have to use ISA or PoT to train energy models.

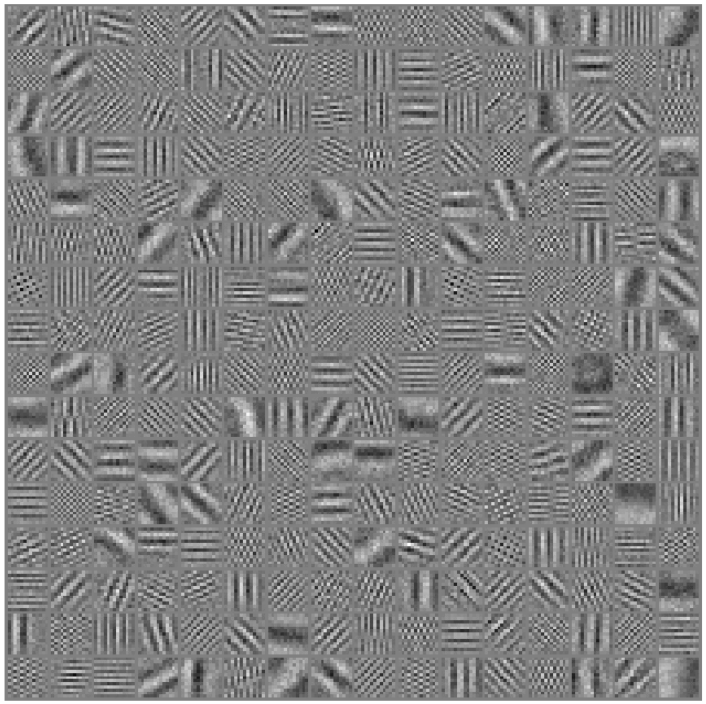
A covariance encoder trained on movies



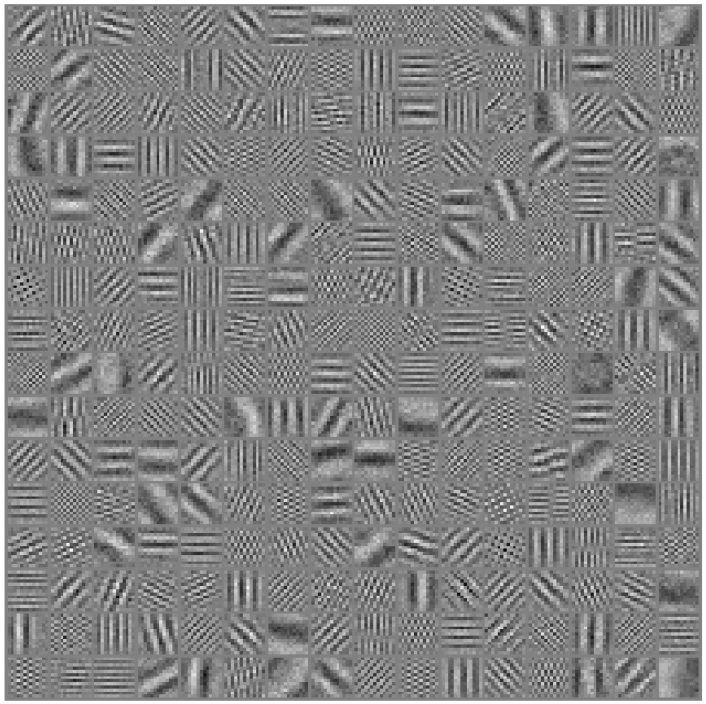
A covariance encoder trained on movies



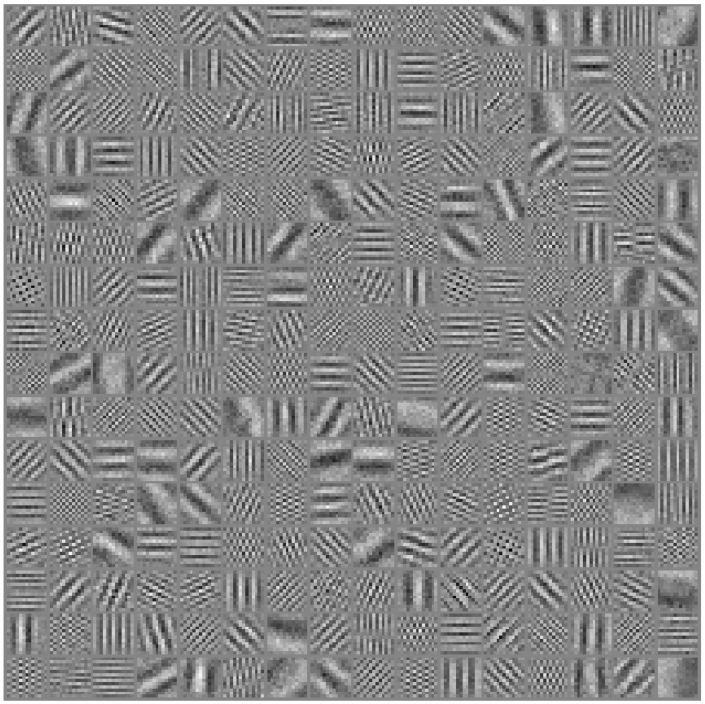
A covariance encoder trained on movies



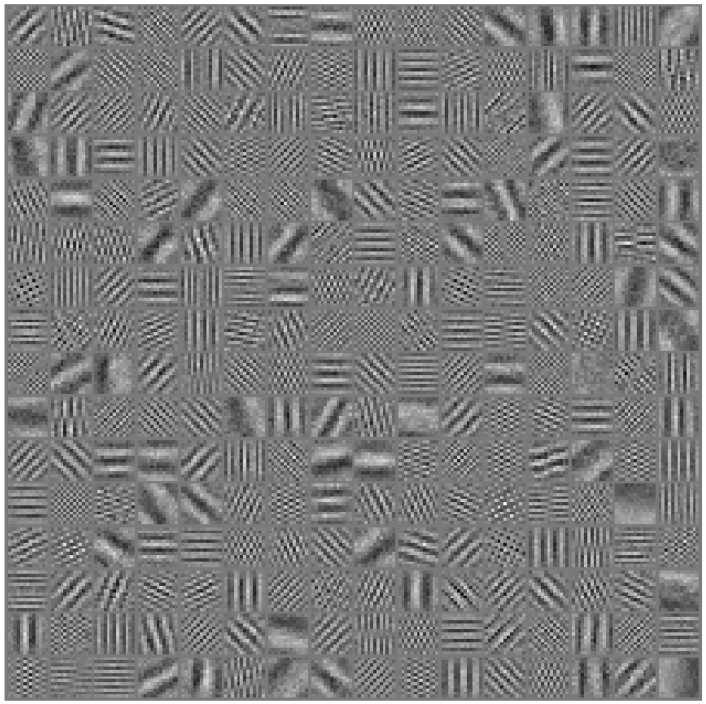
A covariance encoder trained on movies



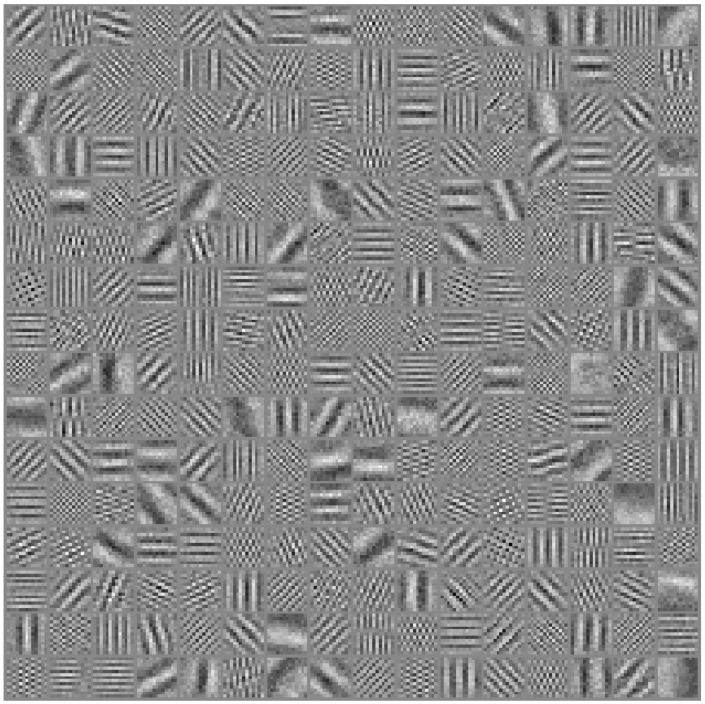
A covariance encoder trained on movies



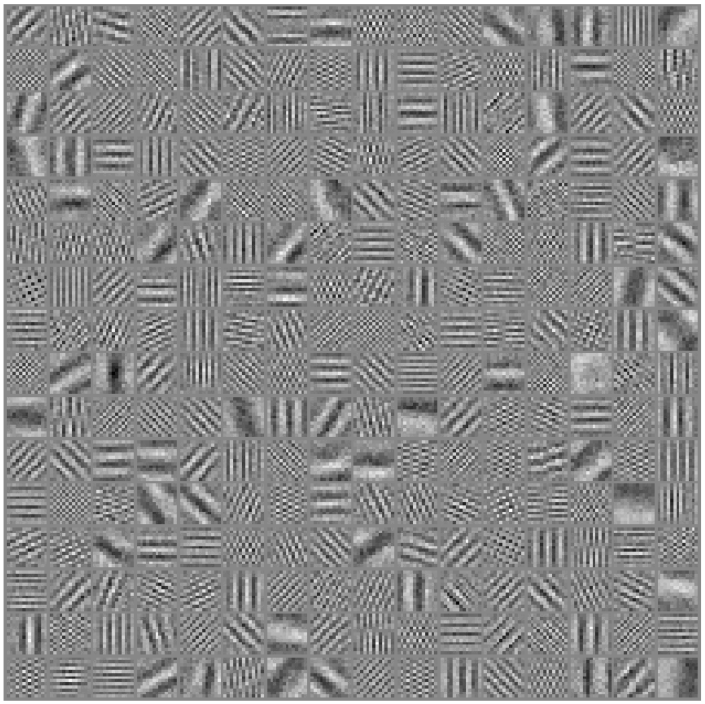
A covariance encoder trained on movies



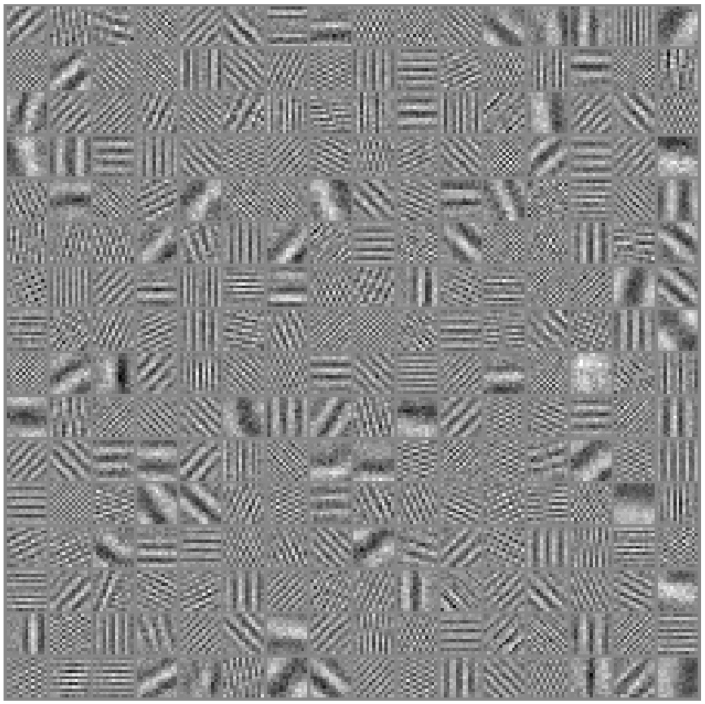
A covariance encoder trained on movies



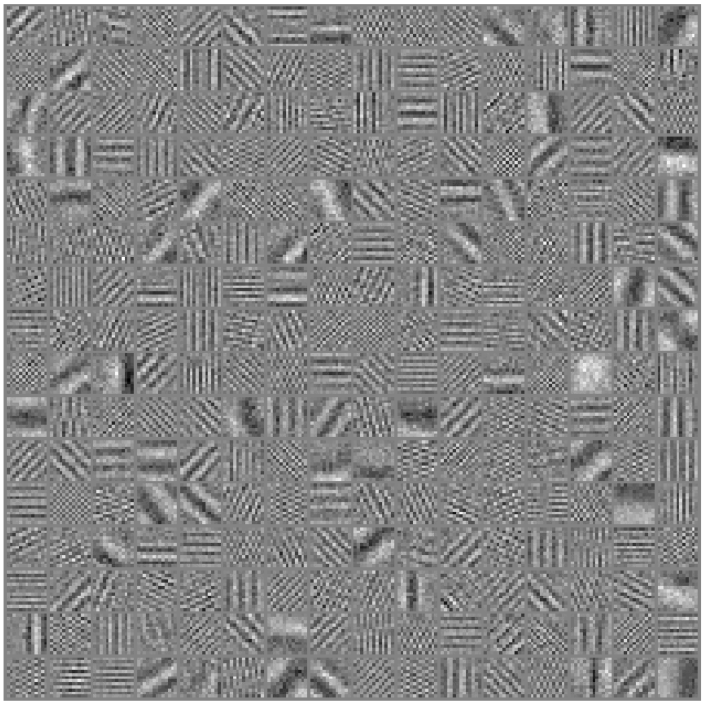
A covariance encoder trained on movies



A covariance encoder trained on movies



A covariance encoder trained on movies



Learning cross-correlation and energy models

Take-home message, factored model

To learn about transformation, let hidden units pool over products of filter responses (gated feature learning) or pool over squares of sums of filter responses (energy model).

A bag of tricks

Tricks for learning:

- Normalize filters during learning, so they *grow slowly*, and they *grow together*: Normalize with a running average of the average filter norms.
- Connect top-level hiddens *locally* to the factors.
- Probably even better: make them *locally overlapping* ("Topographic ICA").
- *DC-centering and contrast-normalization* for each patch.
- Plus: *Whiten the data* before learning, using PCA or ZCA.
- Fast learning: large data-sets essential (use GPU's...).

A bag of tricks

Tricks for learning:

- Normalize filters during learning, so they *grow slowly*, and they *grow together*: Normalize with a running average of the average filter norms.
- Connect top-level hiddens *locally* to the factors.
- Probably even better: make them *locally overlapping* ("Topographic ICA").
- *DC-centering and contrast-normalization* for each patch.
- Plus: *Whiten the data* before learning, using PCA or ZCA.
- Fast learning: large data-sets essential (use GPU's...).

A bag of tricks

Tricks for learning:

- Normalize filters during learning, so they *grow slowly*, and they *grow together*: Normalize with a running average of the average filter norms.
- Connect top-level hiddens *locally* to the factors.
- Probably even better: make them *locally overlapping* ("Topographic ICA").
- *DC-centering and contrast-normalization* for each patch.
- Plus: *Whiten the data* before learning, using PCA or ZCA.
- Fast learning: large data-sets essential (use GPU's...).

A bag of tricks

Tricks for learning:

- Normalize filters during learning, so they *grow slowly*, and they *grow together*: Normalize with a running average of the average filter norms.
- Connect top-level hiddens *locally* to the factors.
- Probably even better: make them *locally overlapping* ("Topographic ICA").
- *DC-centering and contrast-normalization* for each patch.
- Plus: *Whiten the data* before learning, using PCA or ZCA.
- Fast learning: large data-sets essential (use GPU's...).

A bag of tricks

Tricks for learning:

- Normalize filters during learning, so they *grow slowly*, and they *grow together*: Normalize with a running average of the average filter norms.
- Connect top-level hiddens *locally* to the factors.
- Probably even better: make them *locally overlapping* ("Topographic ICA").
- *DC-centering and contrast-normalization* for each patch.
- Plus: *Whiten the data* before learning, using PCA or ZCA.
- Fast learning: large data-sets essential (use GPU's...).

A bag of tricks

Tricks for learning:

- Normalize filters during learning, so they *grow slowly*, and they *grow together*: Normalize with a running average of the average filter norms.
- Connect top-level hiddens *locally* to the factors.
- Probably even better: make them *locally overlapping* ("Topographic ICA").
- *DC-centering and contrast-normalization* for each patch.
- Plus: *Whiten the data* before learning, using PCA or ZCA.
- Fast learning: large data-sets essential (use GPU's...).