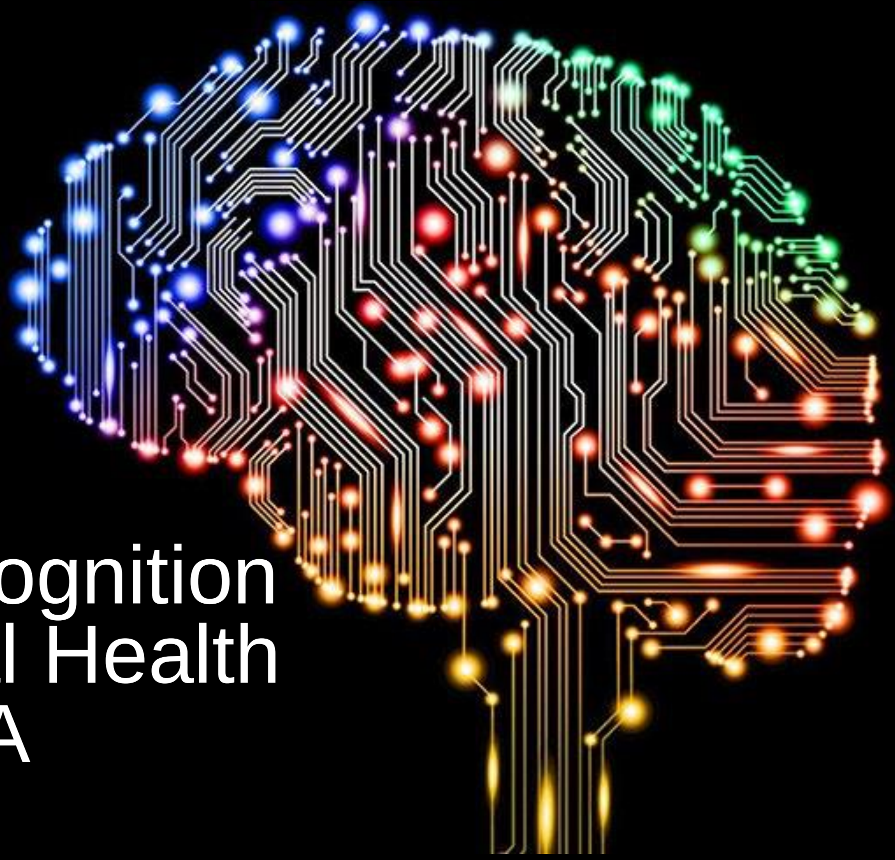
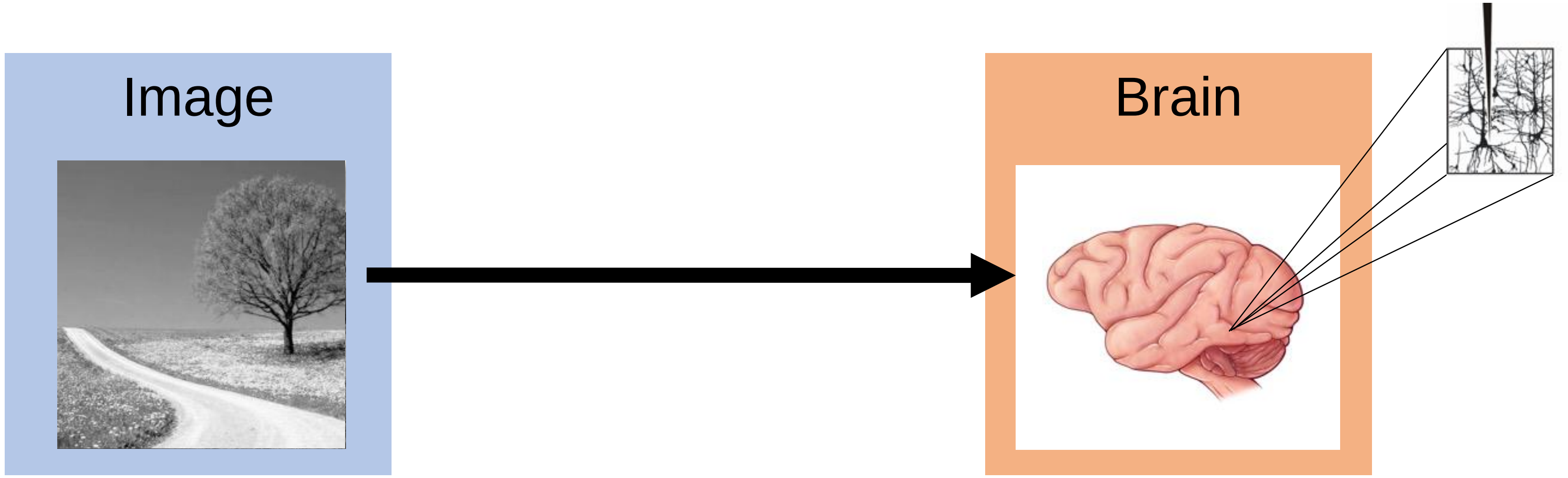
Comparing brains and DNNs: Methods and findings



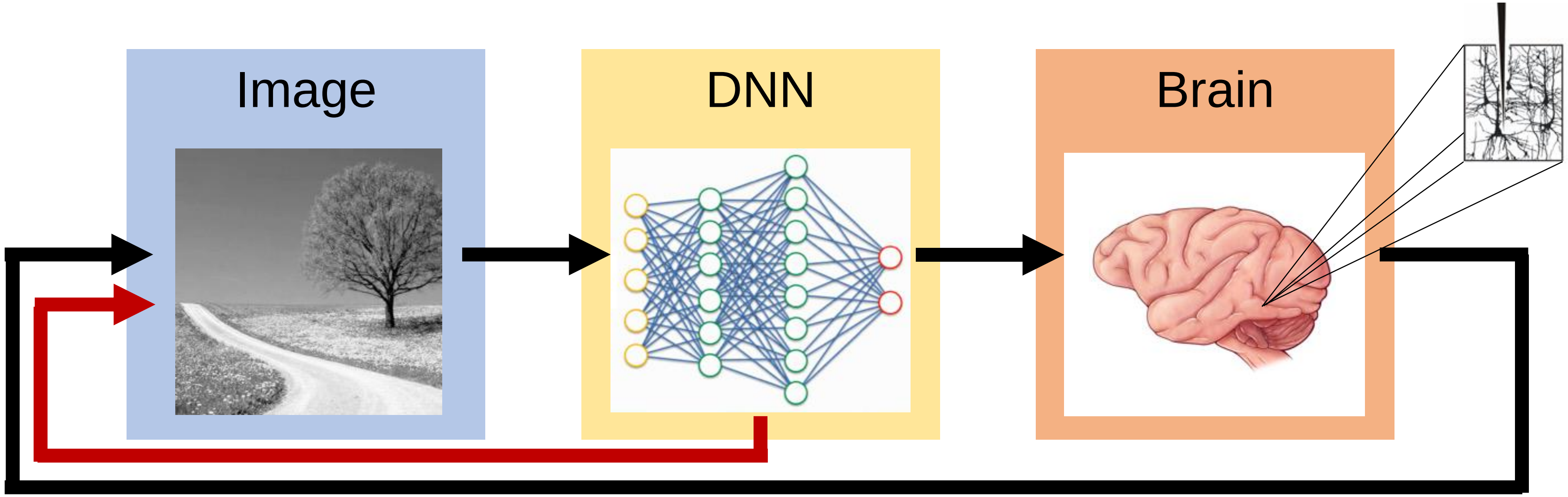
Martin Hebart

Laboratory of Brain and Cognition
National Institute of Mental Health
Bethesda, MD, USA

What information does a neuron represent?



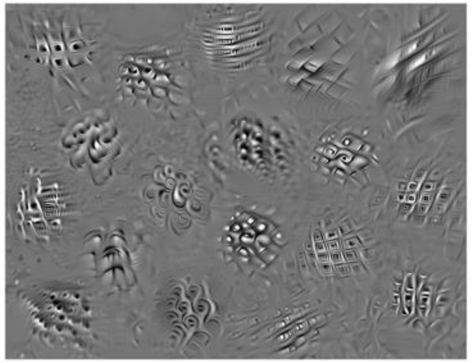
What information does a neuron represent?



Mouse V1

Walker et al, 2018, bioRxiv

Monkey V4



Bashivan et al, 2019, Science

Monkey IT

Ponce et al, 2019, Neuron

Overview

Comparing brains and DNNs: Overview

Methods and findings for comparing brains and DNNs

Practical considerations

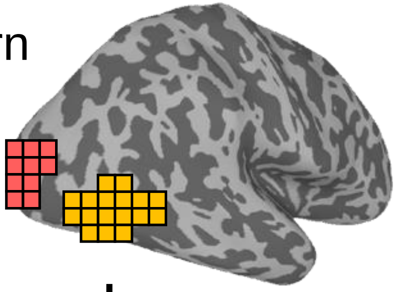
Disclaimer / comments

- Presentation offers only incomplete overview
- Focus on methods and results, less interpretation
- More human data, more similarity-based methods
- Strong focus on vision

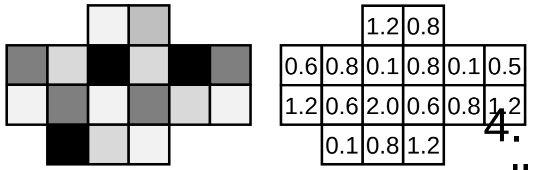
Comparing brains and DNNs: Overview

Brain (e.g. fMRI)

1. Identify pattern (e.g. region of interest)

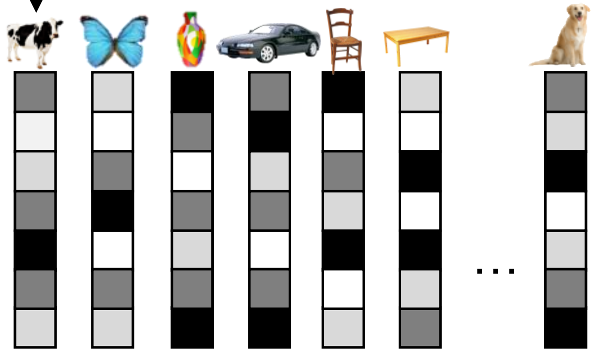


2. Extract activation estimate for condition



4. Get pattern for all conditions

3. Vectorize (i.e. flatten) pattern

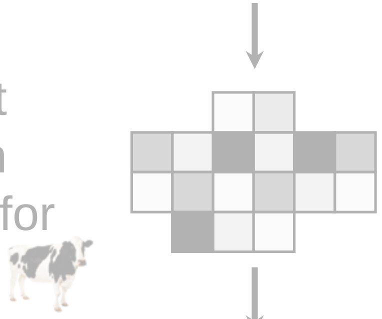


Comparing brains and DNNs: Overview

Brain (e.g. fMRI)

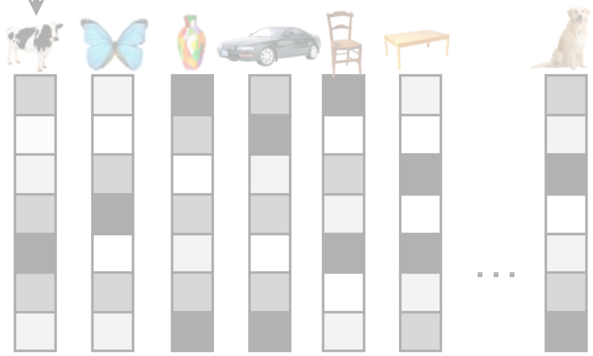
1. Identify pattern (e.g. region of interest)

2. Extract activation estimate for condition

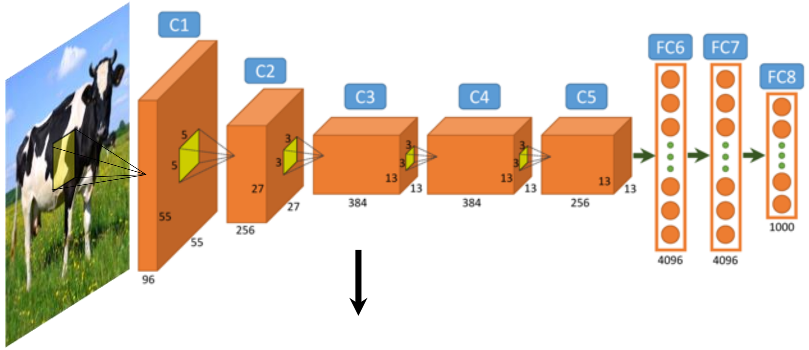


3. Vectorize (i.e. flatten) pattern

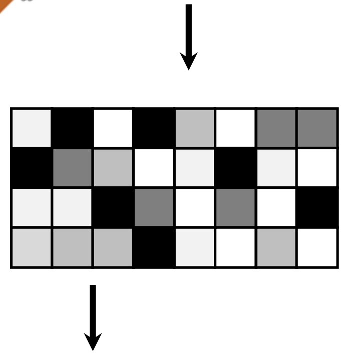
4. Get pattern for all conditions



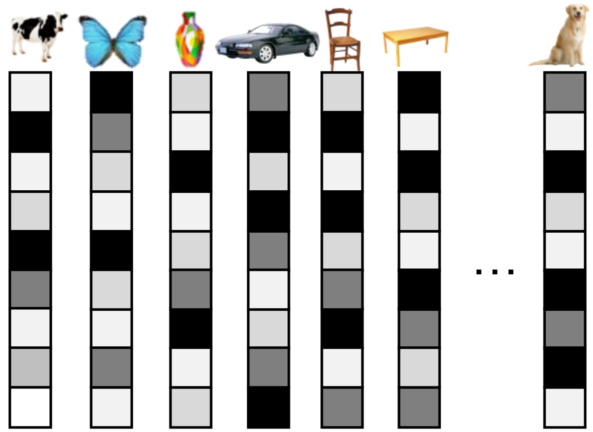
DNN



1. Choose DNN architecture and layer



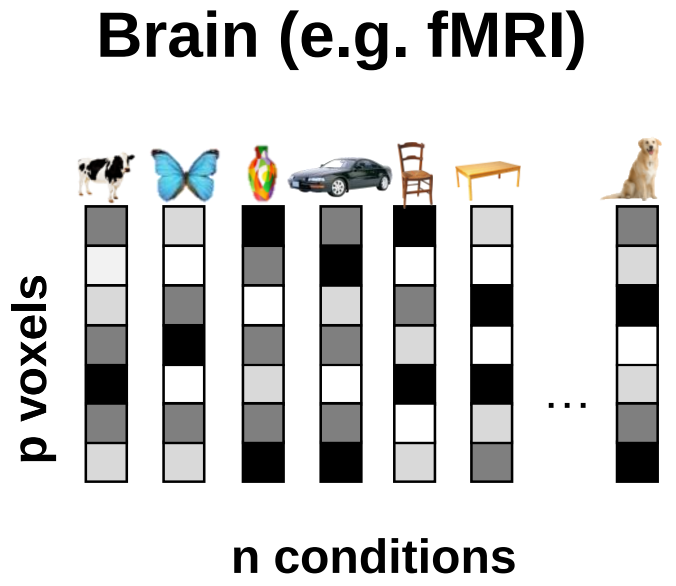
2. Push image through DNN and extract activation at layer

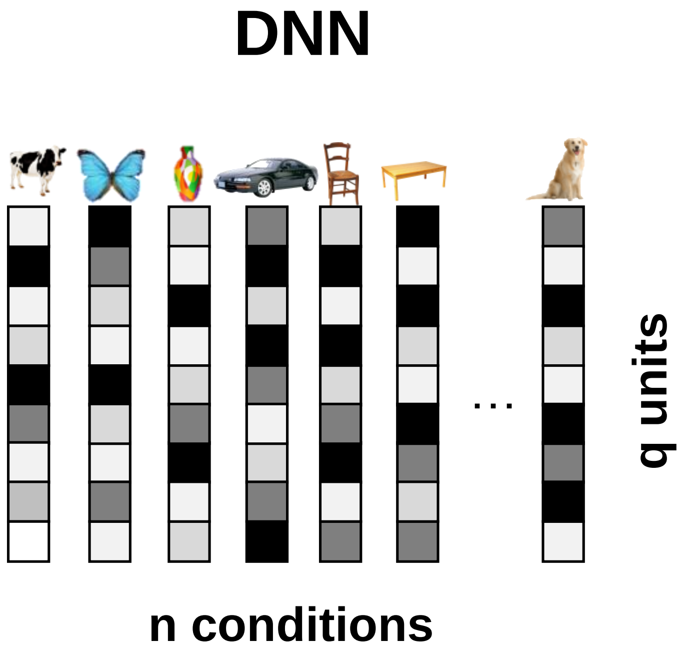


3. Vectorize (i.e. flatten) pattern

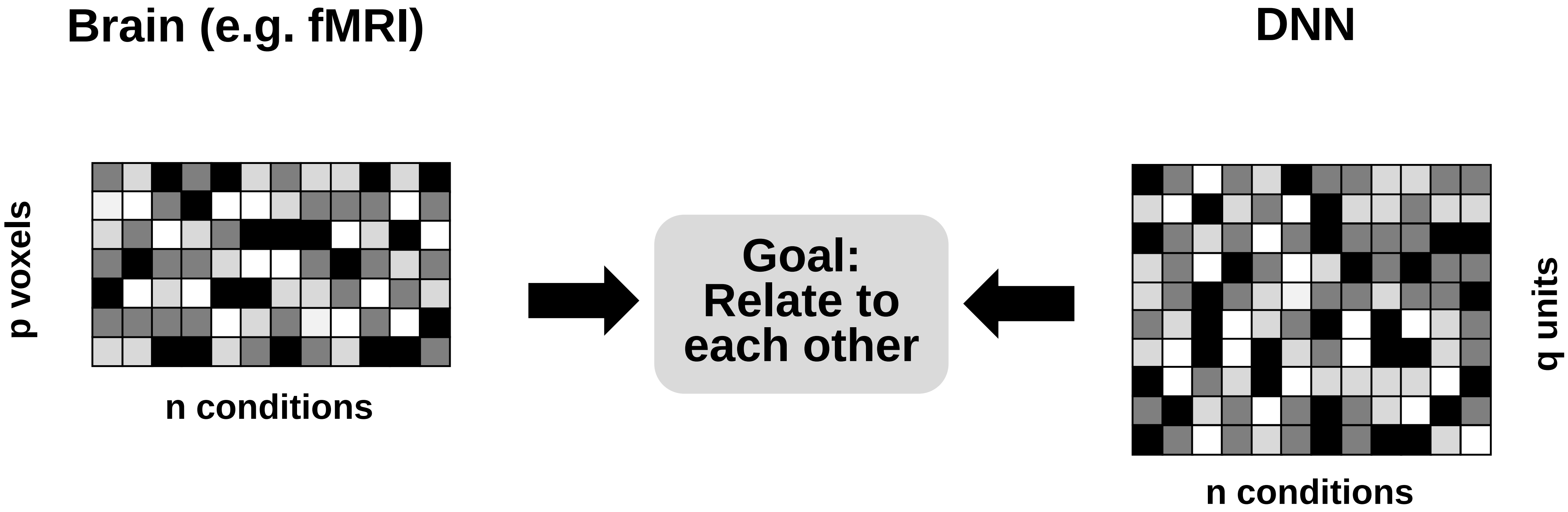
4. Get pattern for all conditions

Comparing brains and DNNs: Overview





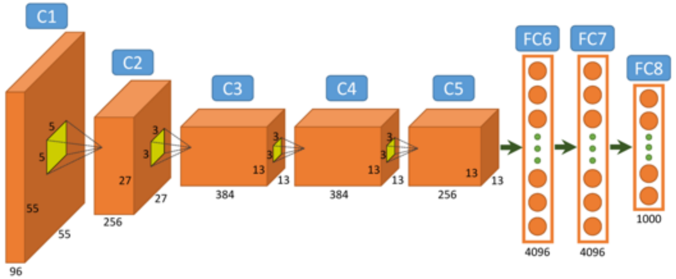
Comparing brains and DNNs: Overview



Overview of methods relating DNNs and brains

S: Stimuli

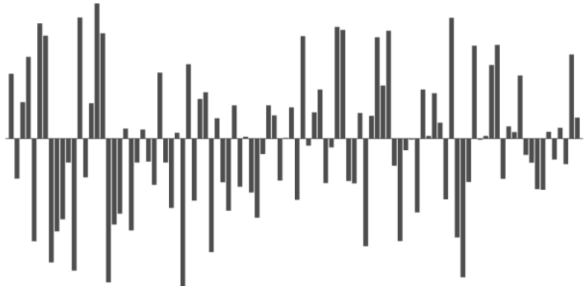
X = f(S)



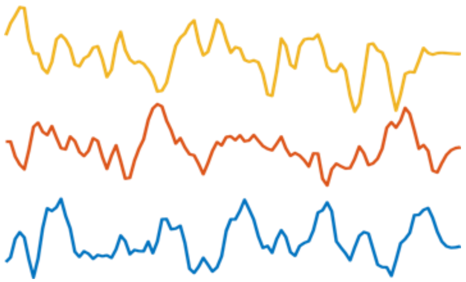
Encoding: g: X → Y

Decoding: h: Y → X

X: Model (stimulus feature representation)

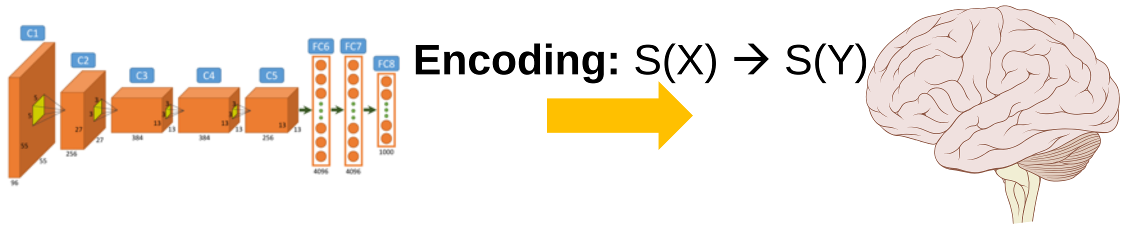


Y: Measurement (brain data)

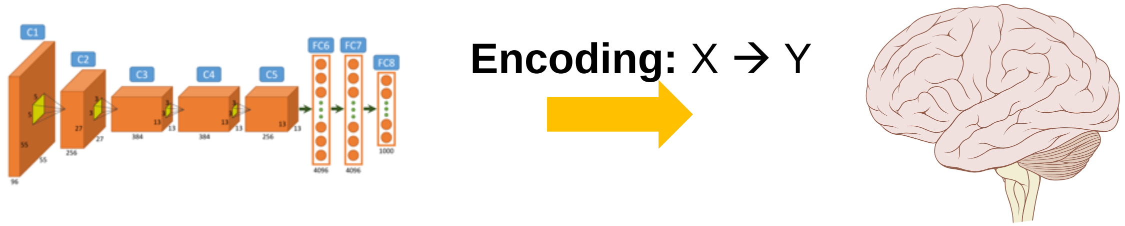


Overview of methods relating DNNs and brains

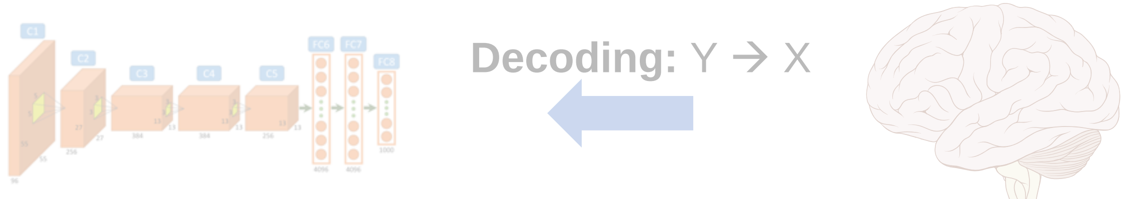
Similarity-based encoding methods (RSA)



Regression-based encoding methods

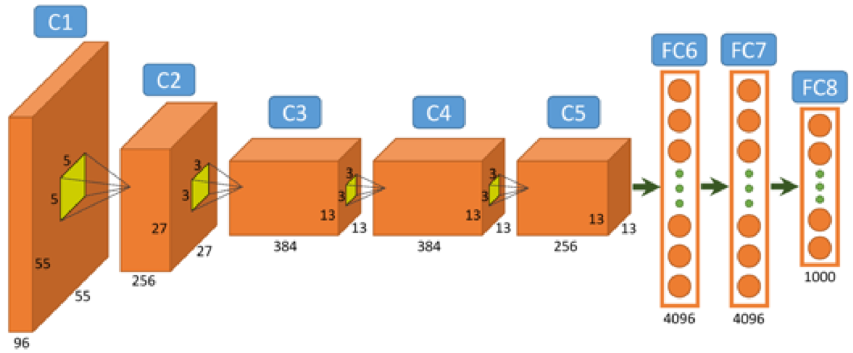


Regression- and classification-based decoding methods

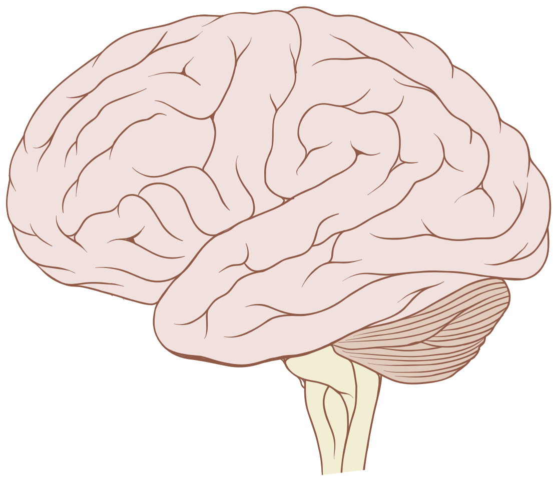


Horikawa & Kamitani, 2017, Nat Commun

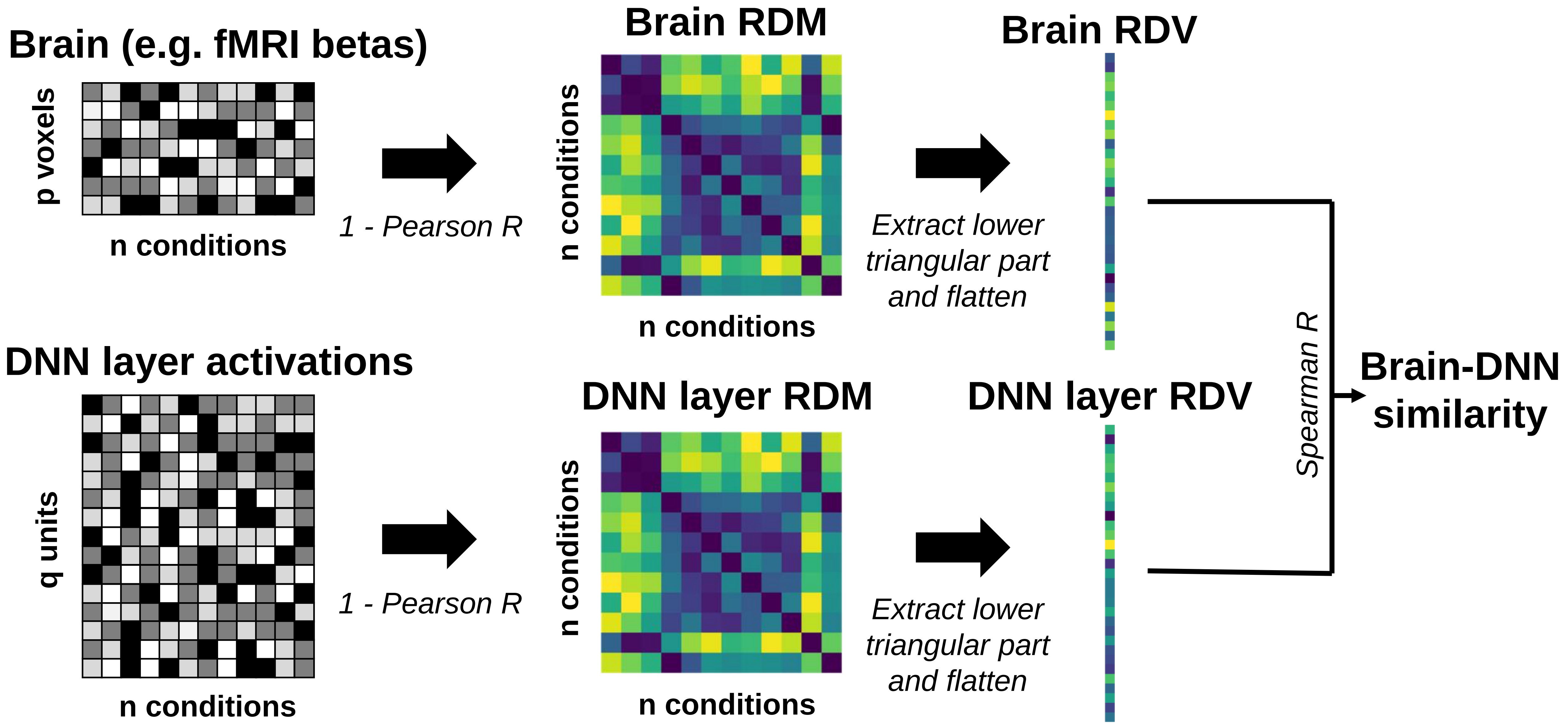
Similarity-based encoding methods



Encoding: S(X) → S(Y)

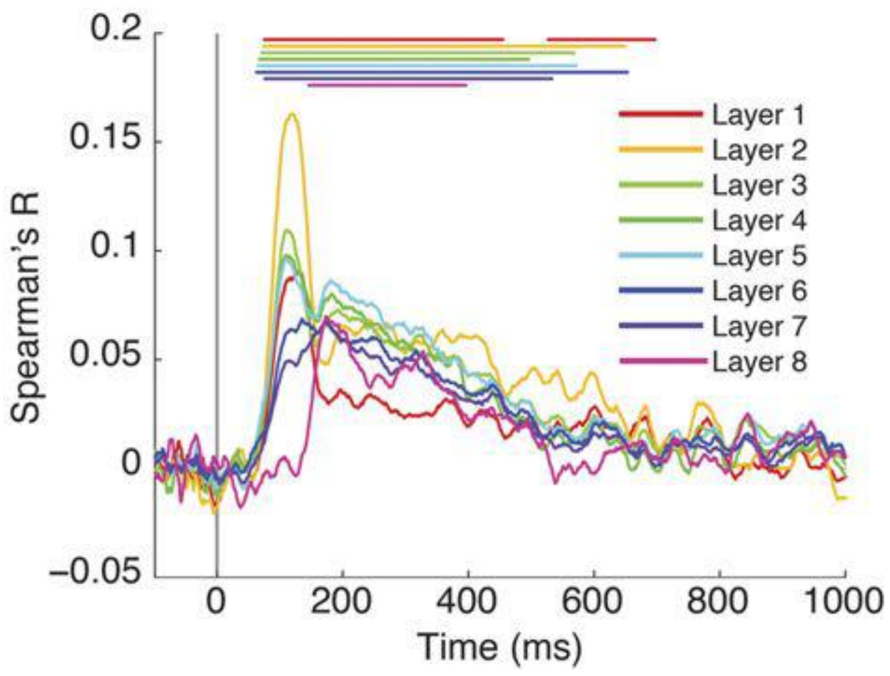


Vanilla representational similarity analysis

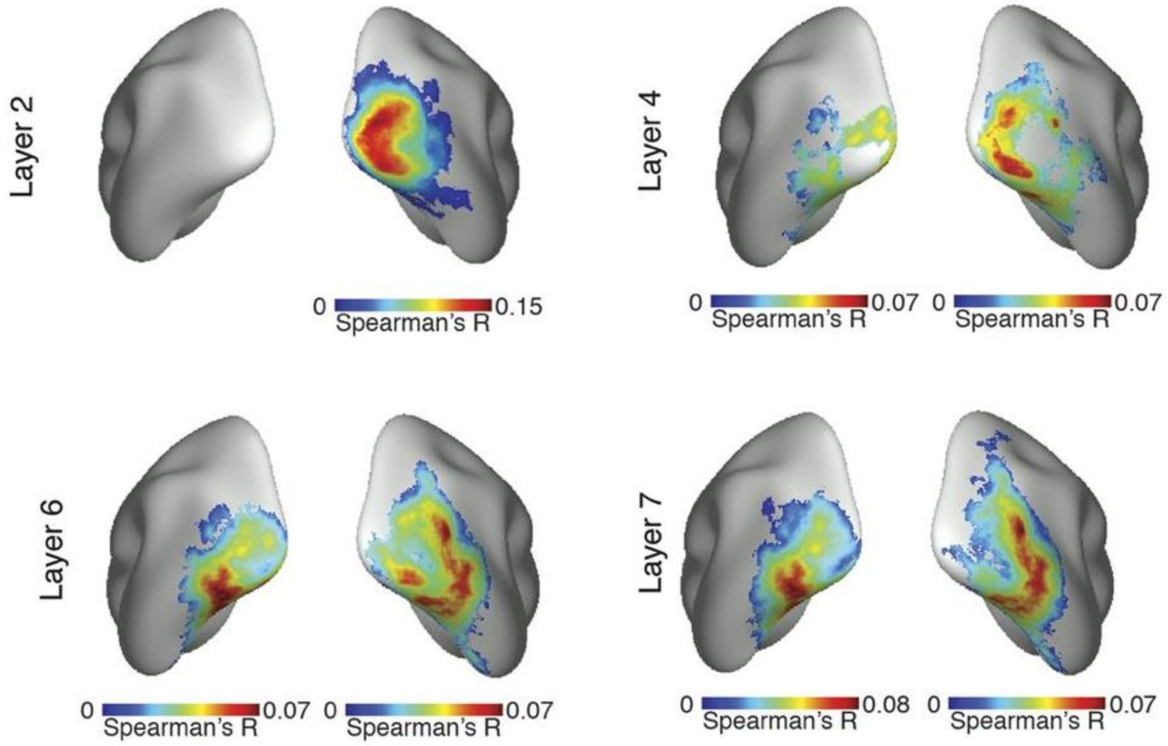


Results: Comparing DNN with MEG and fMRI

MEG (time-resolved)



fMRI (searchlight)



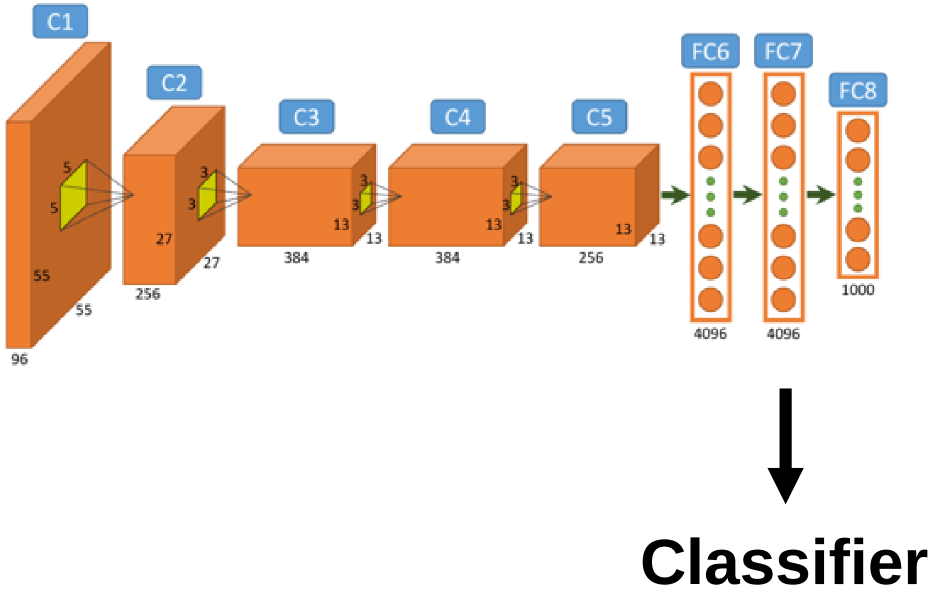
- 118 natural objects with background
- custom-trained AlexNet

Cichy, Khosla, Pantazis, Torralba & Oliva, 2016, Scientific Reports

Advanced RSA: remixing and reweighting

Remixing: Does the layer contain a representation of the category that can be linearly read out?

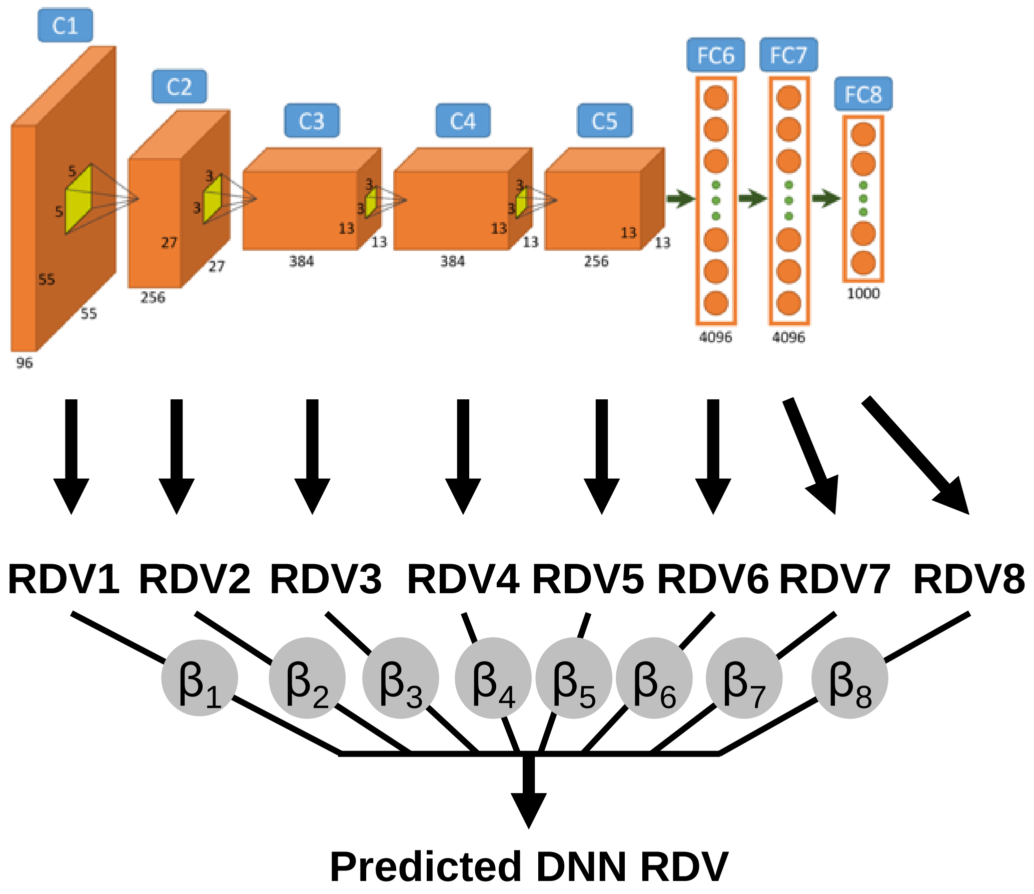
1. Train classifier on layer for relevant categories using new images (e.g. >10 / category)
2. Apply classifier to original images and take output of classifier (e.g. decision values)
3. Construct RDM from output



Advanced RSA: remixing and reweighting

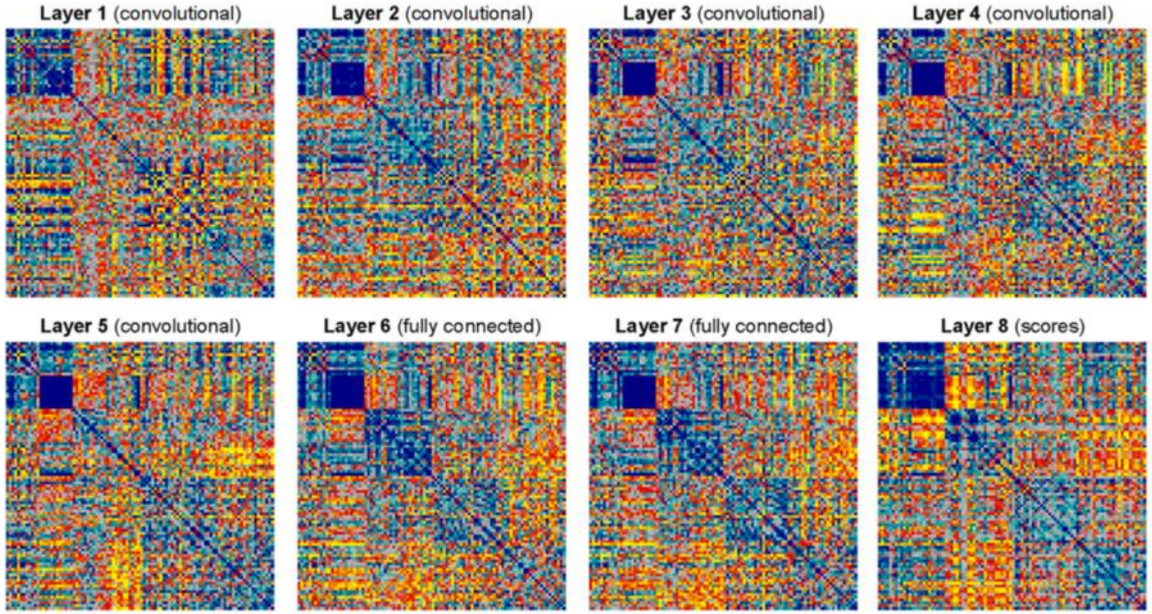
Reweighting: Can the measured brain representational geometry be explained as a linear combination of feature representations at different layers?

1. Create RDV for each layer
2. Carry-out cross-validated non-negative multiple regression
3. Compare predicted DNN RDV to measured brain RDV

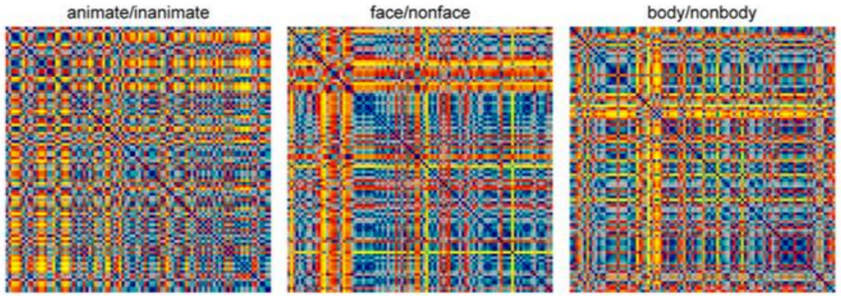


Results: Remixing & reweighting

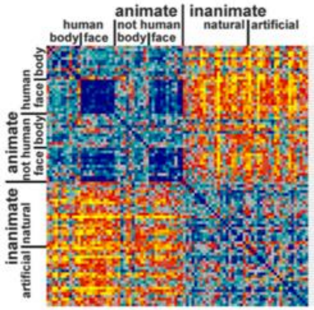
AlexNet, 92 objects



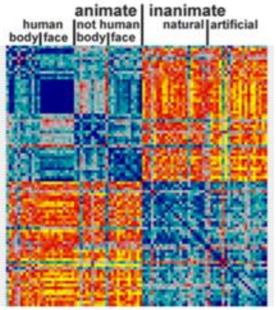
remixing



brain response



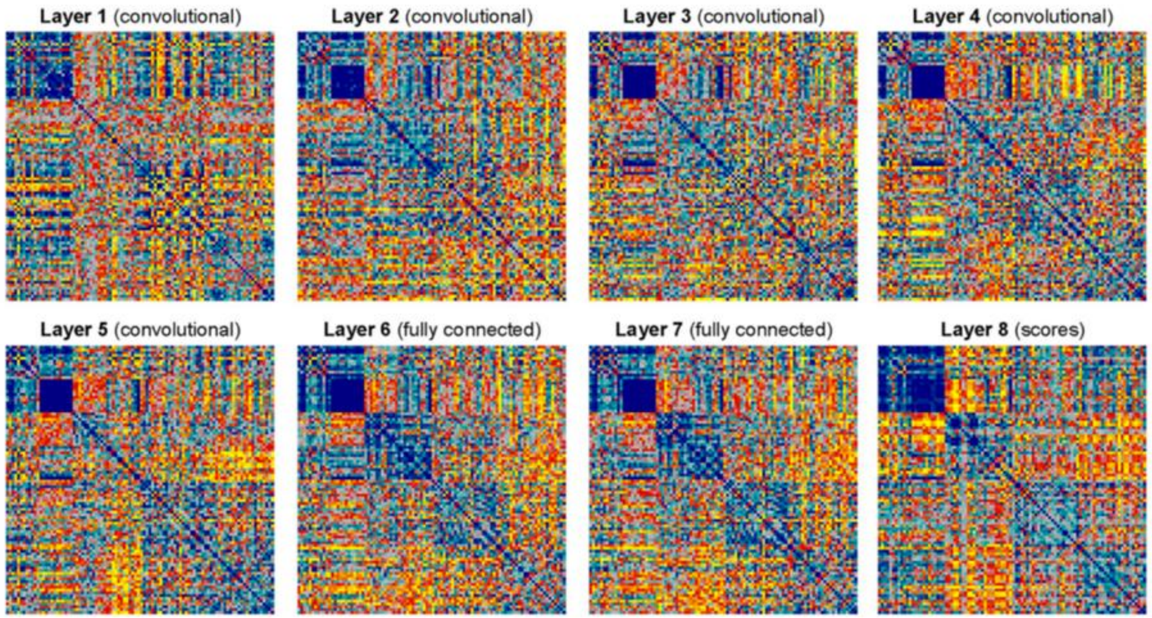
remixing plus reweighting



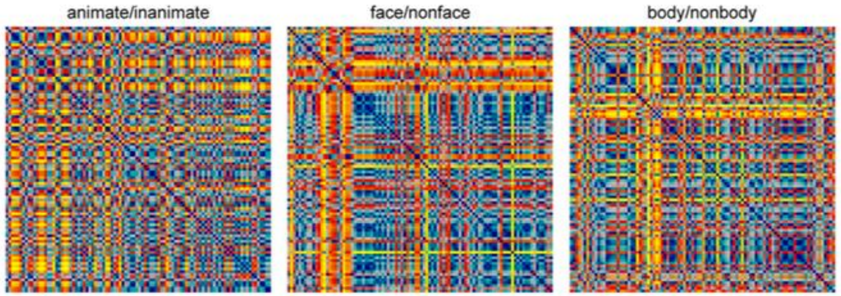
Khaligh-Razavi & Kriegeskorte, 2014, PLoS Comput Biol

Results: Remixing & reweighting

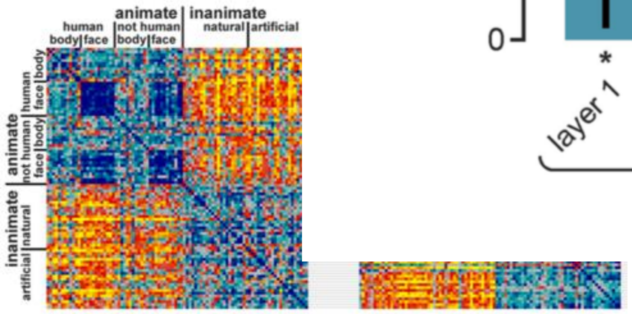
AlexNet, 92 objects



remixing

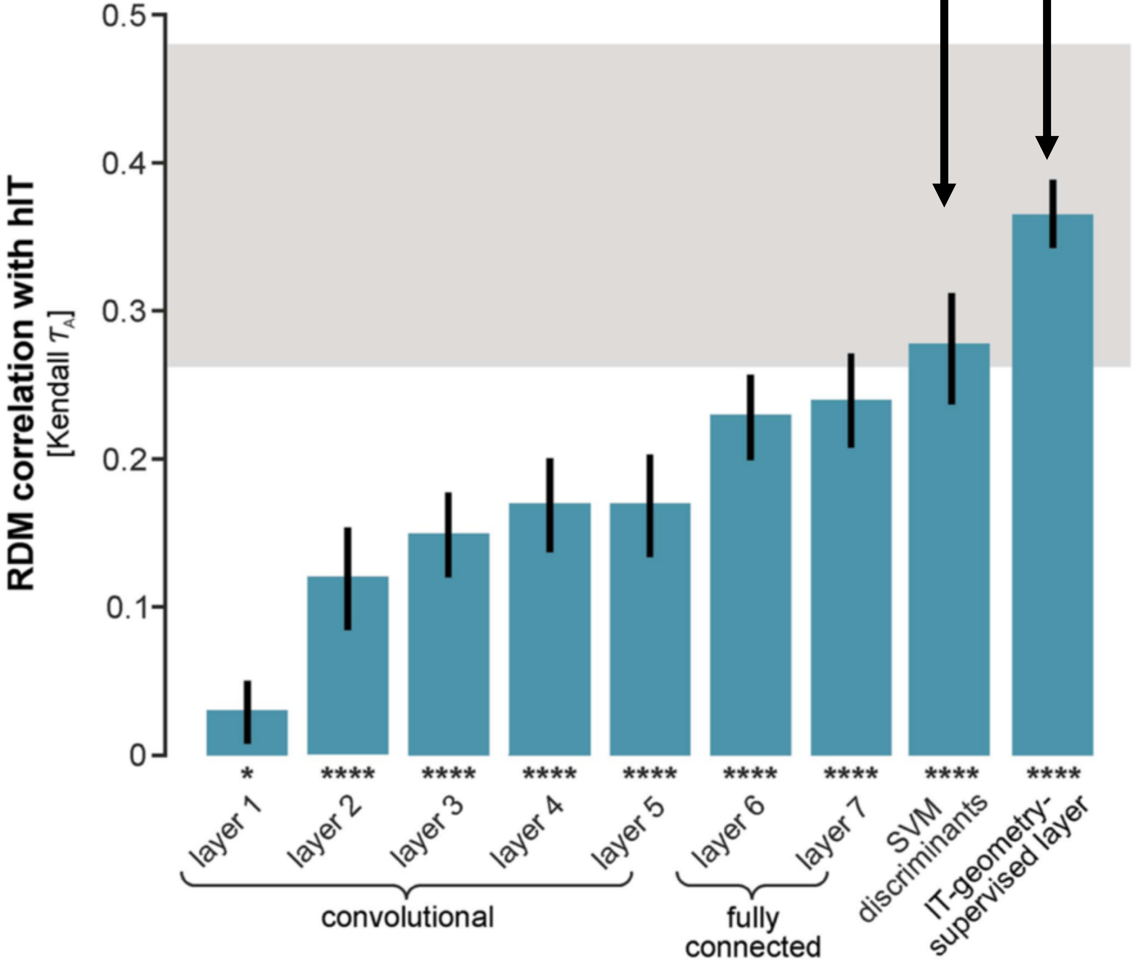


brain response



remixing plus reweighting

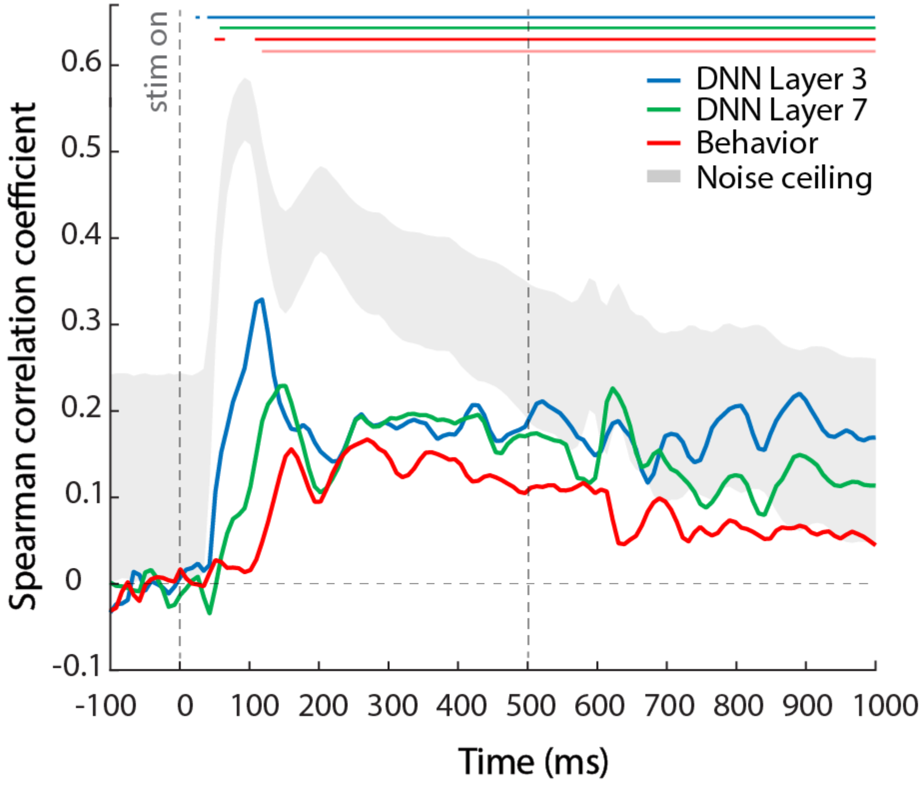
remixing

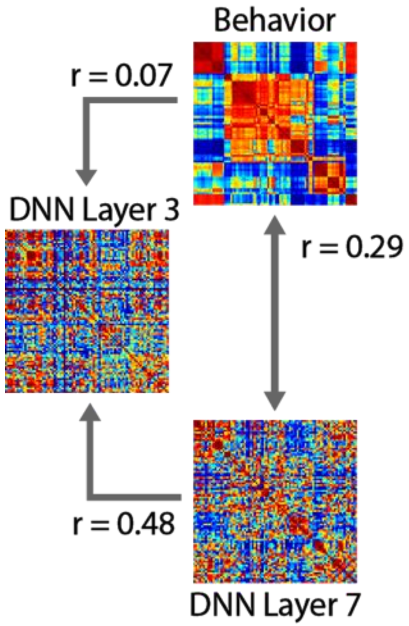


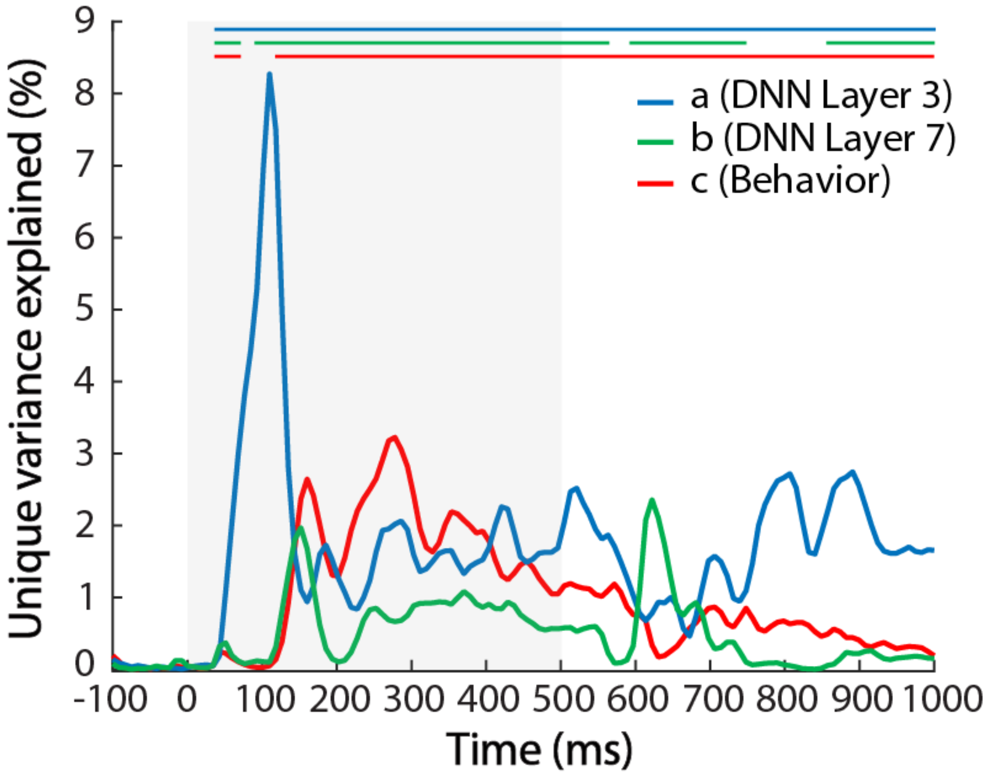
Khaligh-Razavi & Kriegeskorte, 2014, PLoS Comput Biol

Advanced RSA: variance partitioning to control for low-level features

Can we tease apart low-level and high-level representations?





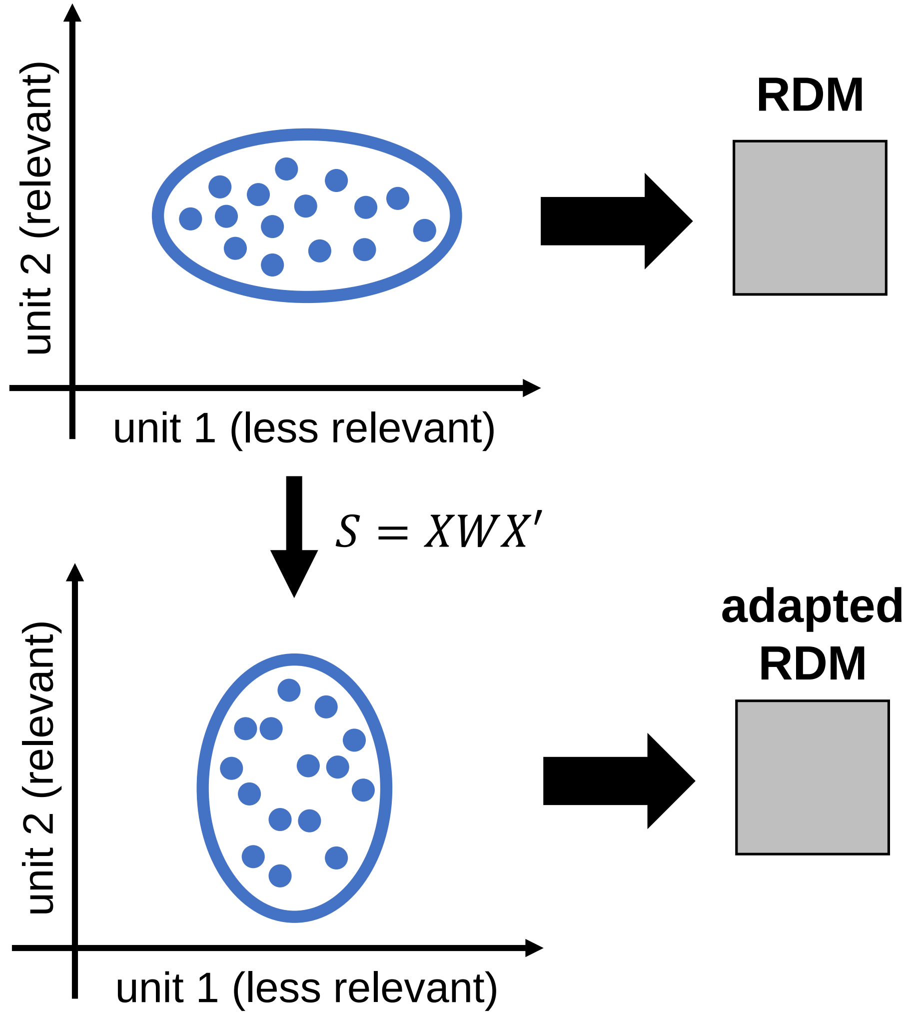


- 84 natural objects without background
- DNN: AlexNet

Bankson, Hebart*, Groen & Baker, 2018, Neuroimage*

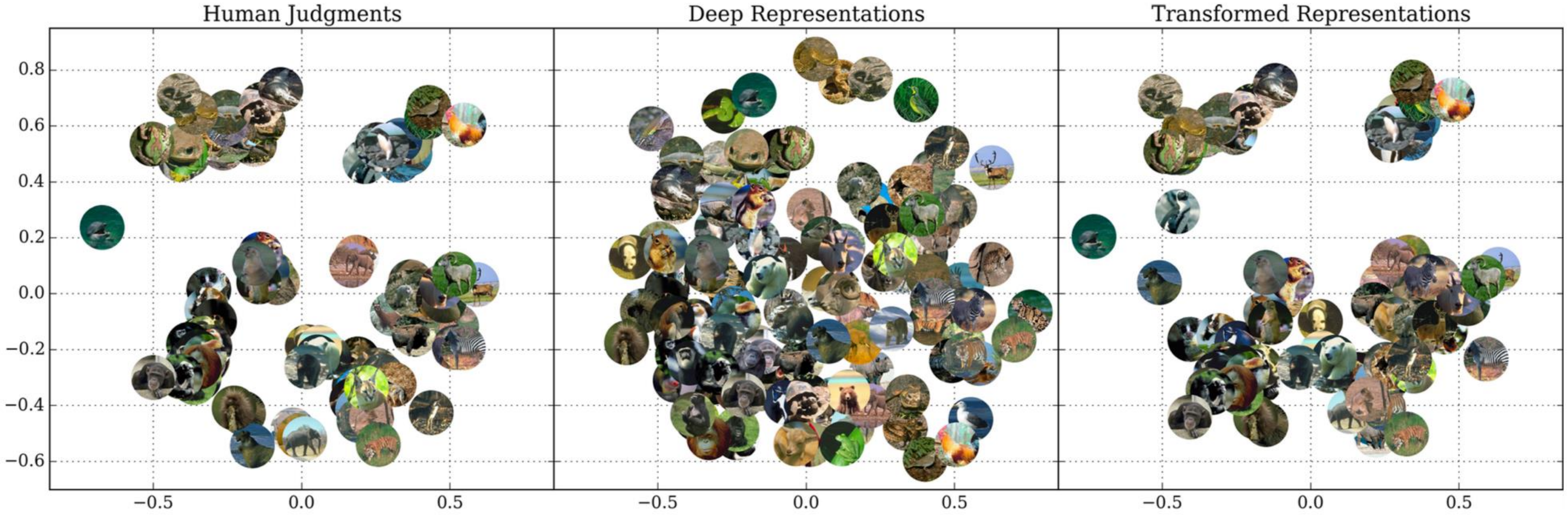
Optimal linear weighting of individual DNN units to maximize similarity

- In standard similarity analysis, all dimensions of the data (e.g. DNN units) contribute the same
- But: Some dimensions may matter more than others
- It is possible to optimize the weighting of each dimension to maximize the fit
- This can be done using cross-validated regression



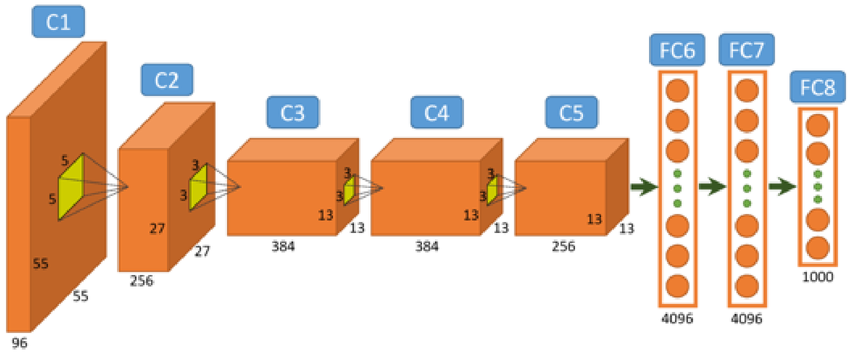
Peterson, Abbott & Griffiths, 2018, Cognitive Science

Optimal linear weighting of individual DNN units to maximize similarity

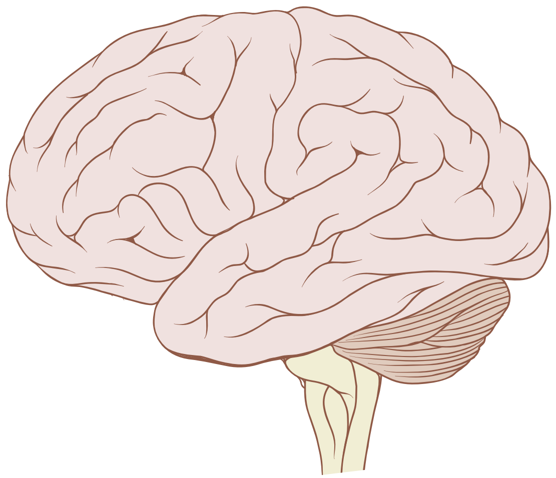


Peterson, Abbott & Griffiths, 2018, Cognitive Science

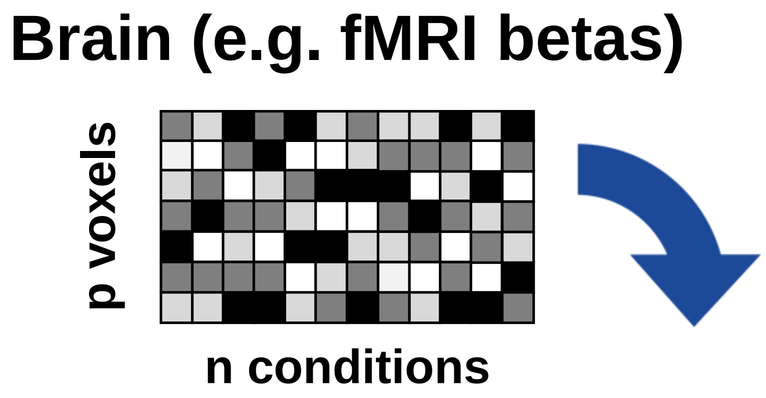
Regression-based encoding methods

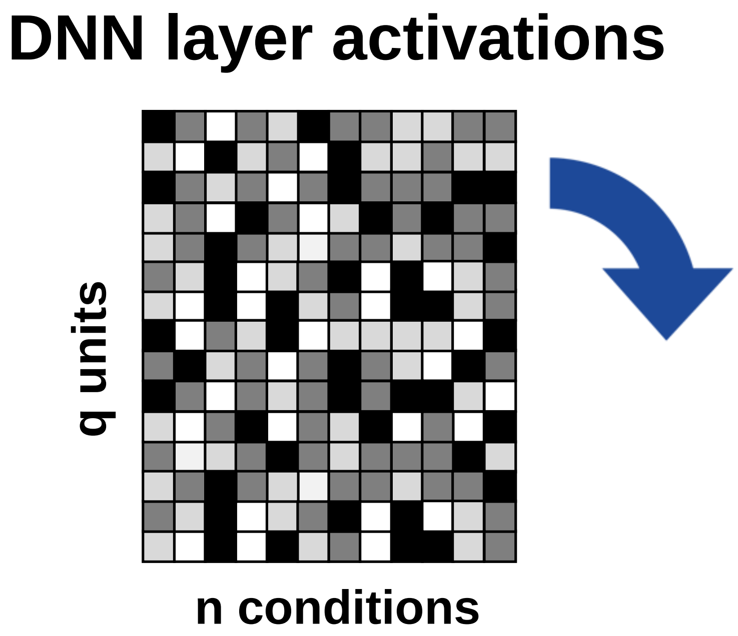


Encoding: X → Y



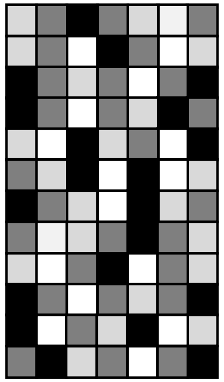
Simple multiple linear regression



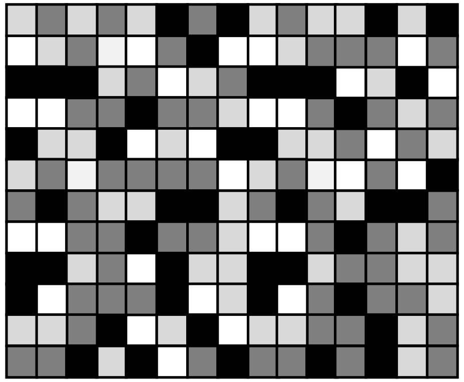


Simple multiple linear regression

Brain (e.g. fMRI betas)



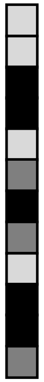
DNN layer activations



Simple multiple linear regression

Brain (e.g. fMRI betas)

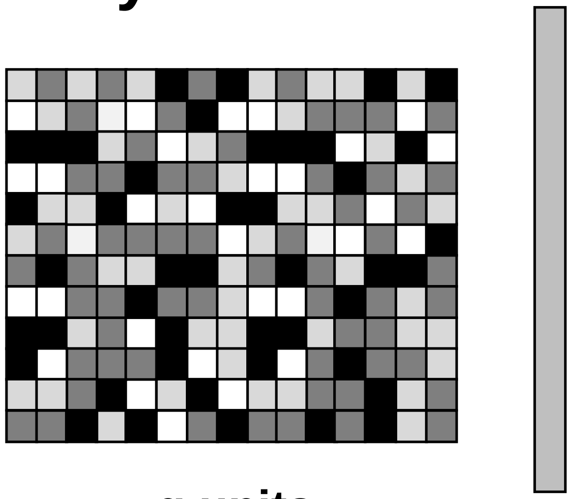
n conditions



voxel i

DNN layer activations

n conditions



q units

$$y \quad = \quad X \cdot \beta + \varepsilon$$

→ Repeat for each voxel (i.e. univariate method)

Simple multiple linear regression

Problem: Often more variables (*q* units) than measurements (*n* conditions)
→ no unique solution, unstable parameter estimates and overfitting

One solution: Regularization, i.e. adding constraints on the range of values β can take (e.g. Ridge regression, LASSO regression)

Another solution: Dimensionality reduction, i.e. projecting data to a subspace (e.g. Principal Component regression, Partial Least Squares)

Regularization in multiple linear regression

Formula for regression: $y = X ß + \varepsilon$

Constrains range of beta

Error minimized for OLS regression: $\sum (\mathrm{y} - Xß)^2$

Error minimized for ridge regression: $\sum (\mathrm{y} - Xß)^2 + \lambda_r ||ß||^2$

Error minimized for LASSO regression: $\sum (\mathrm{y} - Xß)^2 + \lambda_l ||ß||$

➡ Requires optimization of regularization parameter **λ** (e.g. using cross-validation)

➡ Advanced regularization: explicit assumptions on covariance matrix structure

Regularization in multiple linear regression

Formula for regression: $y = X\text{ß} + \varepsilon$

Constrains range of beta

Error minimized for OLS regression: $\sum (y - X\text{ß})^2$

Presence of many variables leads to potential for overfitting

Error minimized for ridge regression: $\sum (y - X\text{ß})^2 + \lambda_l \|\text{ß}\|^2$

→ quality of fit can be estimated using cross-validation (e.g. split-half or 90%-10% split)

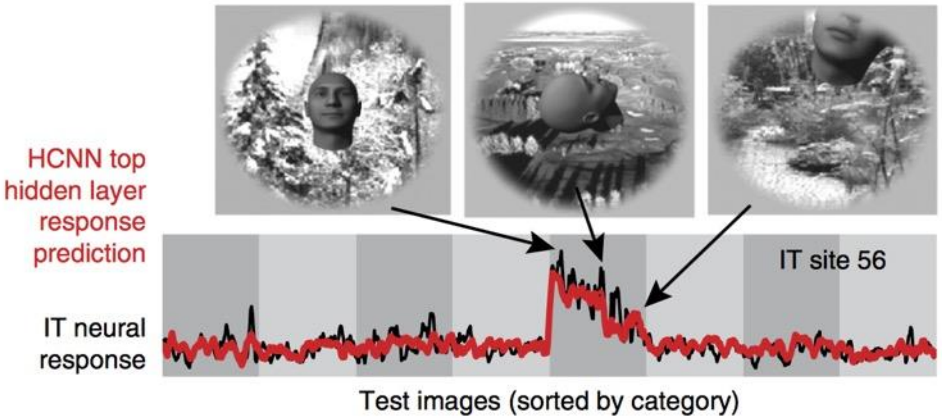
Error minimized for LASSO regression: $\sum (y - X\text{ß})^2 + \lambda_l \|\text{ß}\|$

➡ Requires optimization of regularization parameter **λ** (e.g. using cross-validation)

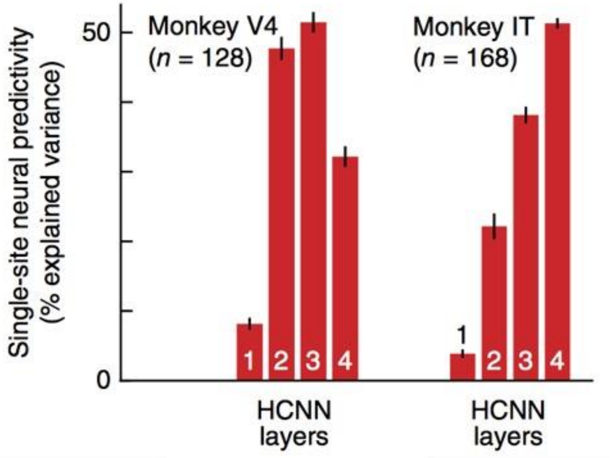
➡ Advanced regularization: explicit assumptions on covariance matrix structure

Results: Regression-based encoding methods

Monkey V4 and IT



- 5760 images of 64 objects (8 categories)
- custom DNN “HMO”

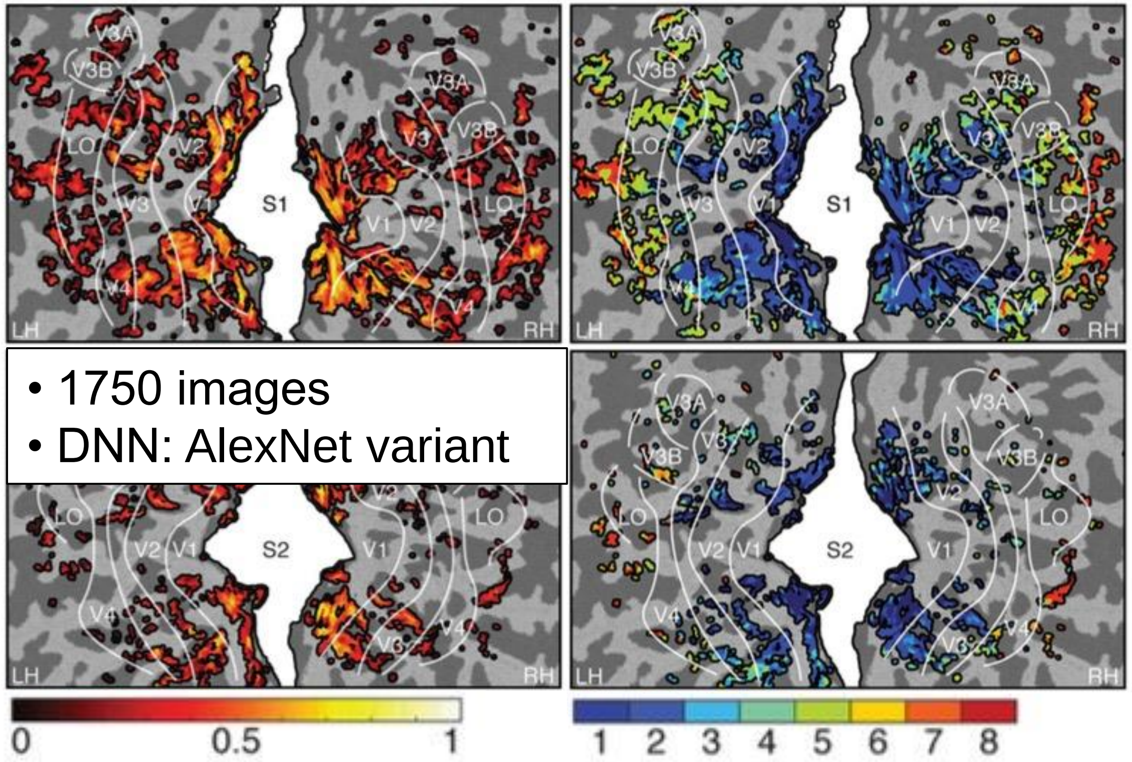


Yamins et al., 2014, PNAS

Human visual cortex

Voxelwise prediction

Most predictive layer



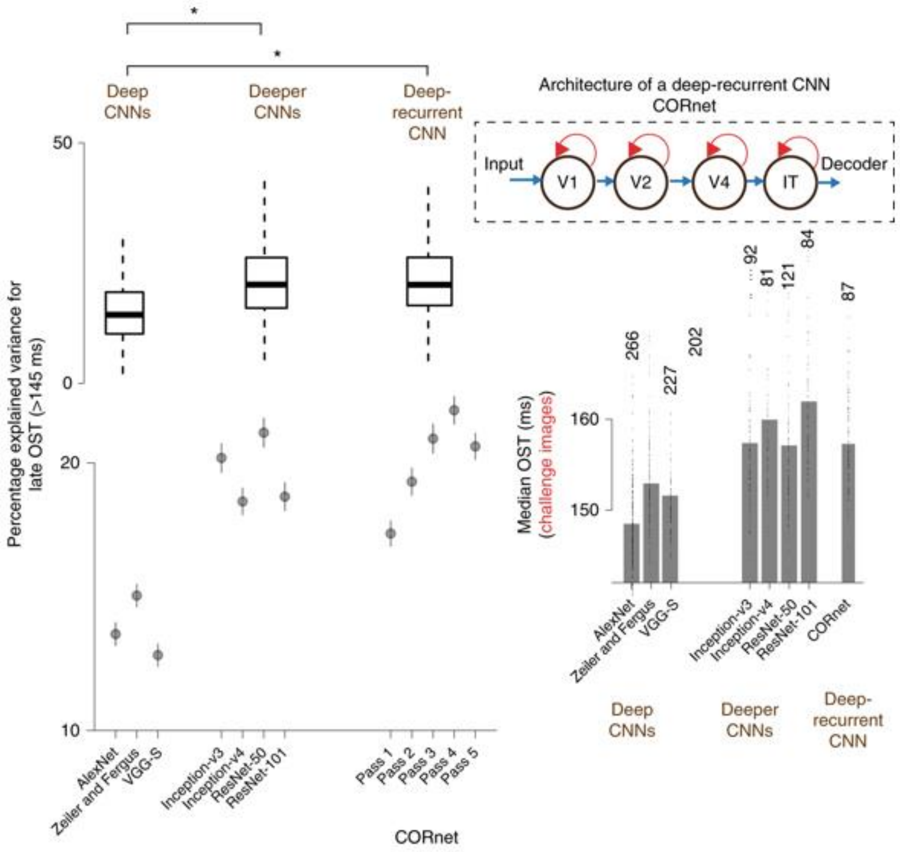
- 1750 images
- DNN: AlexNet variant

Güçlü & van Gerven, 2015, J Neurosci

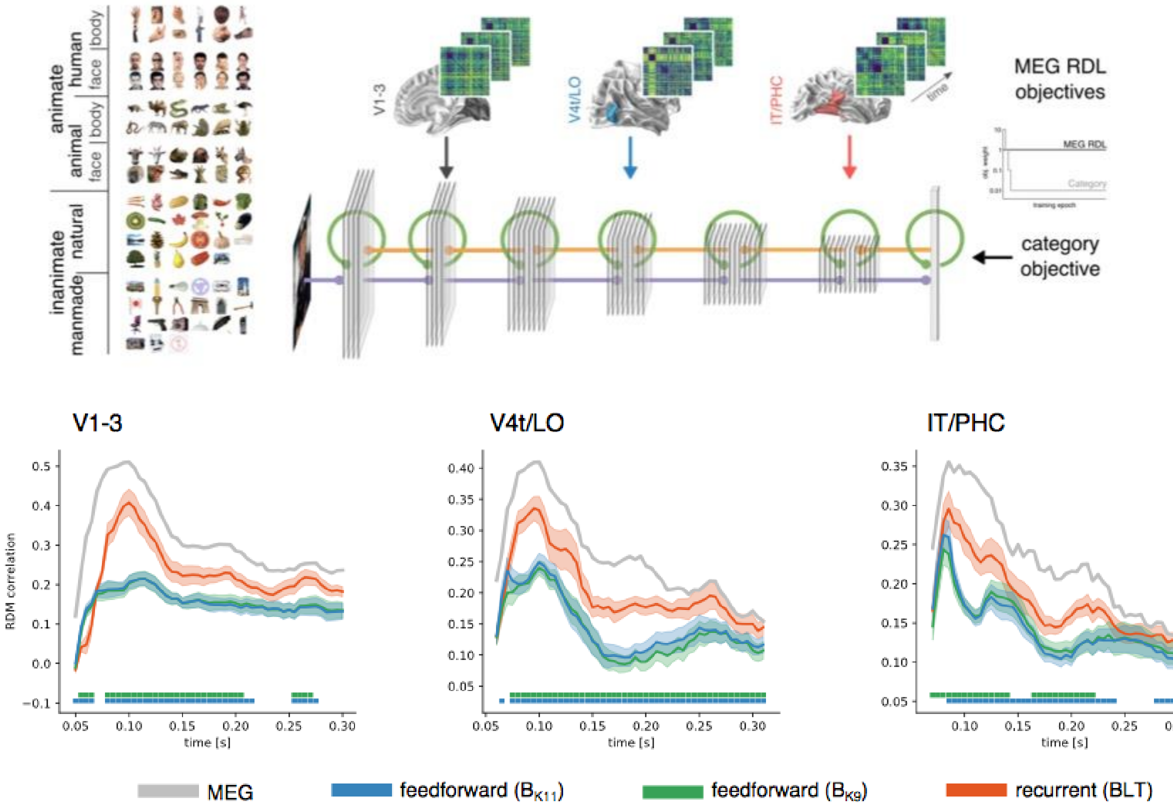
Building networks to model the brain

Recurrent models better capture core object recognition in ventral visual cortex

in both monkey recordings…



… and humans (MEG sources)



Kar et al., 2019, Nat Neurosci

Kietzmann, et al., 2018, bioRxiv

Practical considerations

Matlab users: Using MatConvNet

- Downloading pretrained models:
  http://www.vlfeat.org/matconvnet/pretrained/
- Quick guide to getting started:
  http://www.vlfeat.org/matconvnet/quick/
- Function for getting layer activations:
  http://martin-hebart.de/code/get_dnnres.m

Python users: Using Keras

- Keras is very easy, but classic TensorFlow or PyTorch also work
- Running images through pretrained models:
  https://engmrk.com/kerasapplication-pre-trained-model/
- Getting layer activations (still requires preprocessing images):
  https://github.com/philipperemy/keract

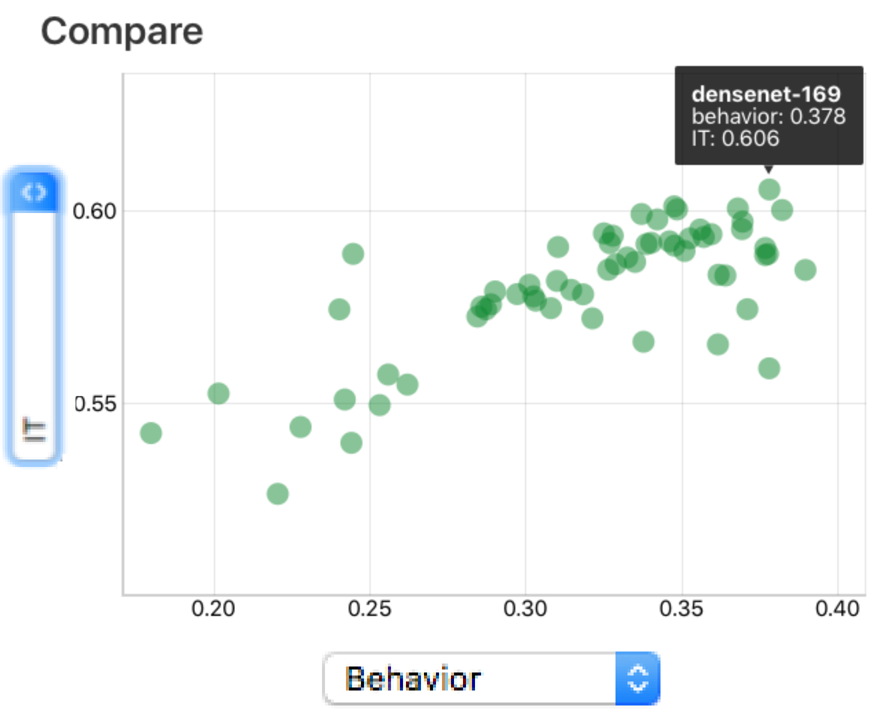
What architecture should we pick?

If goal is maximizing brain prediction:

- Pick network with most predictive layer(s)
- Brain score?

If goal is using plausible model:

- Very common / better understood architectures: AlexNet and VGG-16
- Other architectures (e.g. ResNet, DenseNet) less common



Schrimpf, Kubilius et al., 2018, bioRxiv

Which layers should we pick?

If goal is to maximize brain prediction

→ Try all layers

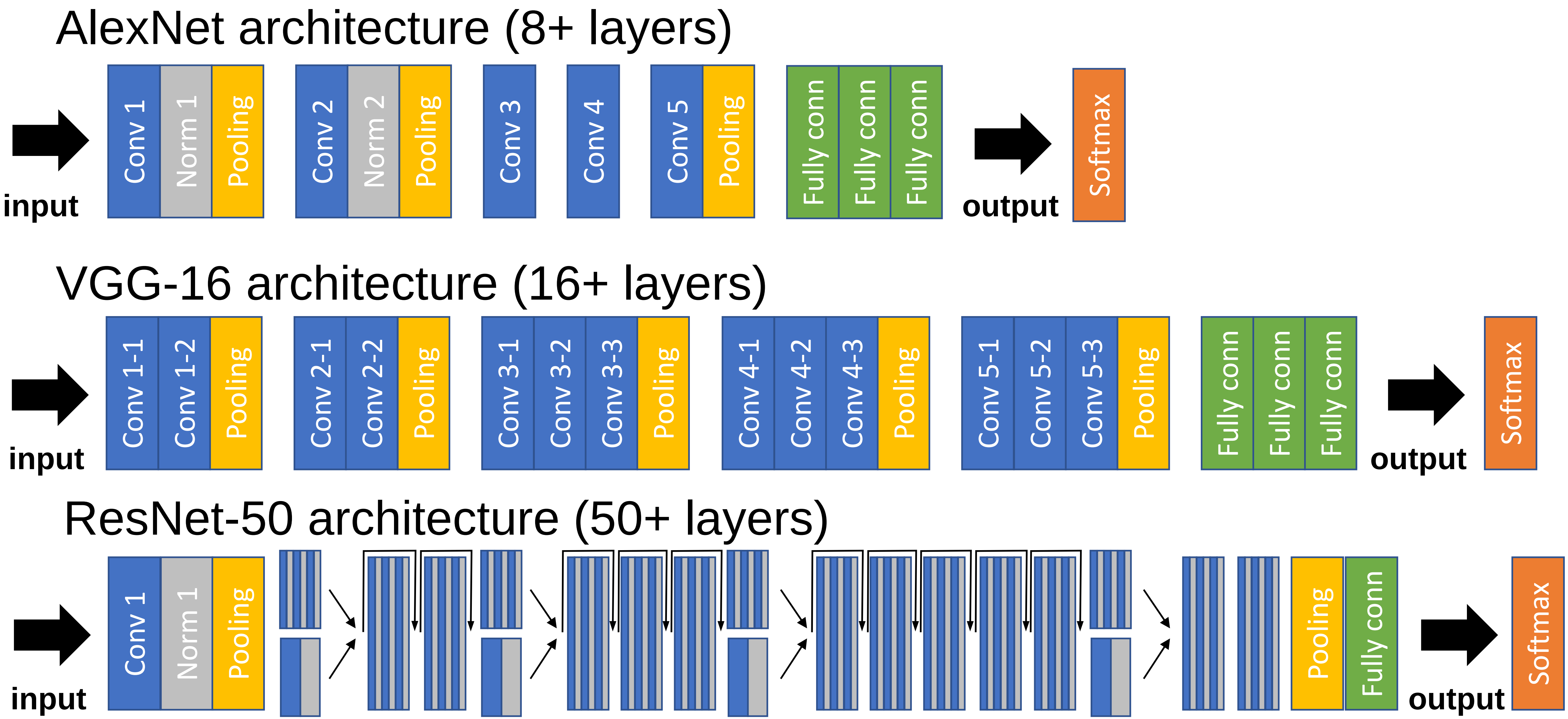
If goal is using entire DNN as model of brain

→ Try all or some layers

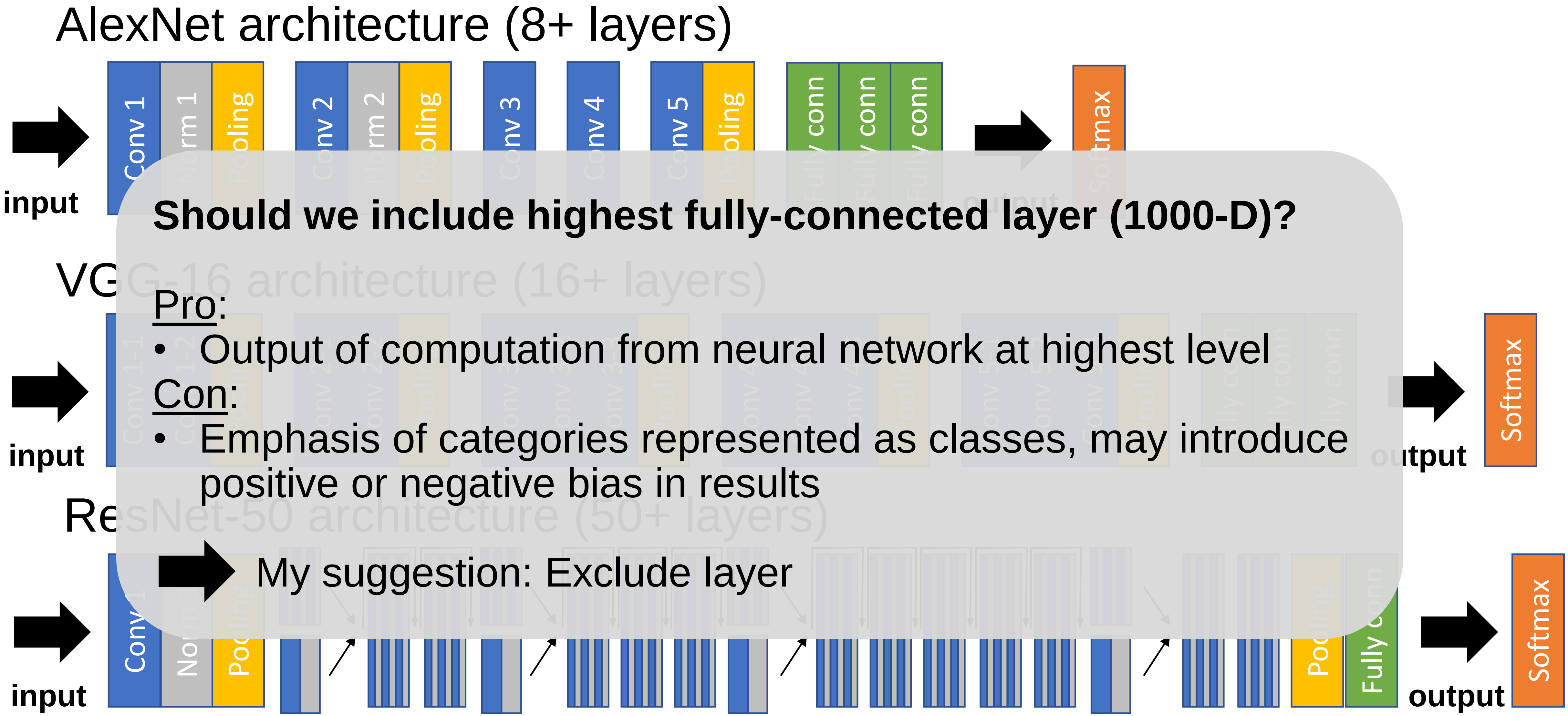
If goal is using plausible model where layer progression mirrors progression in brain: some layers

→ Pick plausible layers

Which layers should we pick?



Which layers should we pick?



Should we include highest fully-connected layer (1000-D)?

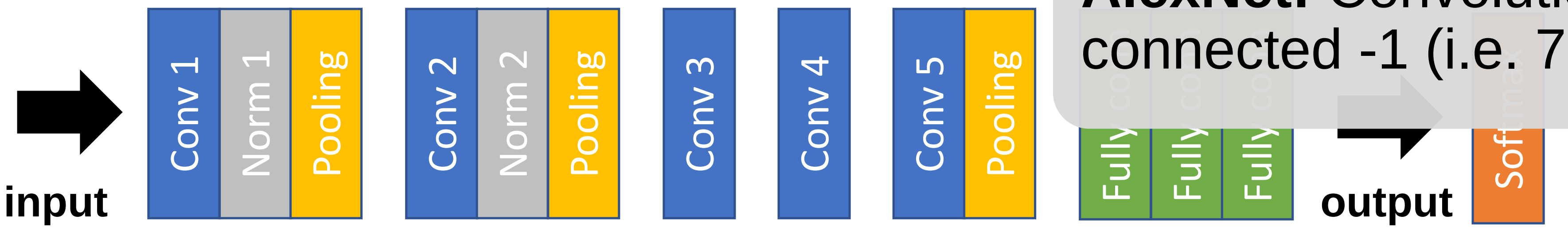
Pro:
- Output of computation from neural network at highest level

Con:
- Emphasis of categories represented as classes, may introduce positive or negative bias in results

➡ My suggestion: Exclude layer

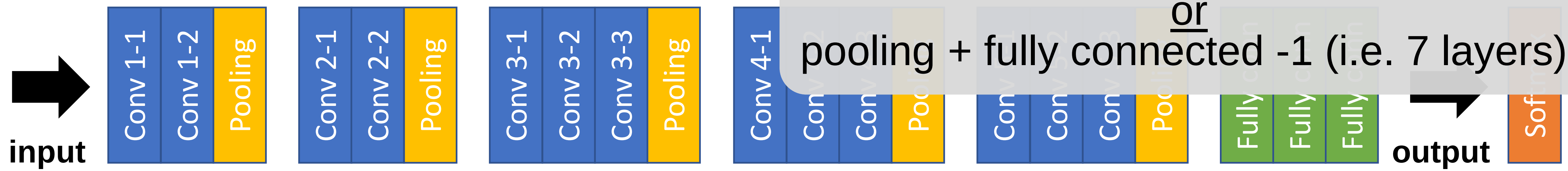
Which layers should we pick?

AlexNet architecture (8+ layers)



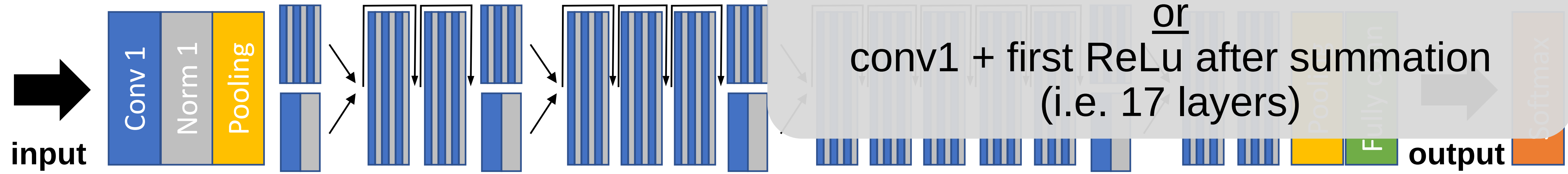
AlexNet: Convolutional and fully connected -1 (i.e. 7 layers)

VGG-16 architecture (16+ layers)



VGG-16: highest conv + fully conn - 1
or
pooling + fully connected -1 (i.e. 7 layers)

ResNet-50 architecture (50+ layers)



ResNet-50: conv1 + summation
or
conv1 + first ReLu after summation
(i.e. 17 layers)

Common preprocessing of images

Original image

1. Resize

2. Crop to square and keep 7/8th

3. Normalize (e.g. z-score or subtract mean image during training)

My advice:

- Run studies on participants / animals using square images
- Resize and crop images to correct size <u>before</u> running toolbox function → provides maximal control
- Make sure image normalization is implemented and correct

Reduction of model size

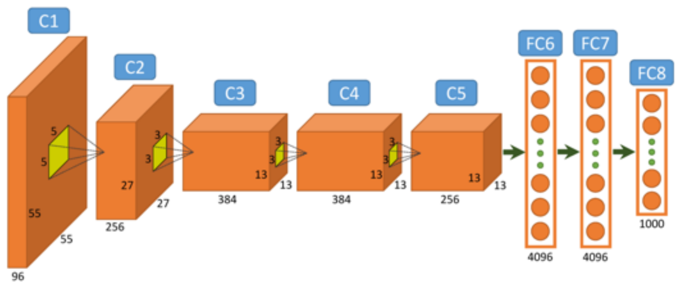
- Useful when predicting brain data from layers with many units
  - Makes more complex models possible at all
  - increases computational speed
  - can reduce overfitting
- Examples:
  - AlexNet Layer 1: 55×55×96 = 290,400 units
  - VGG-16 / ResNet Layer 1: 112×112×64 = 802,816 units

➡ Common approach: PCA compression

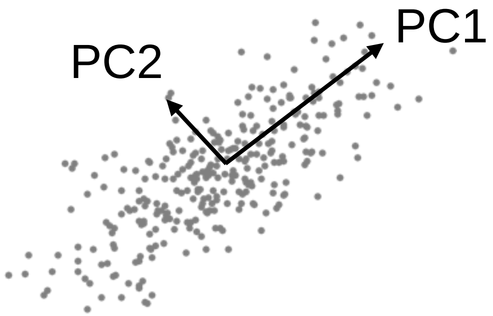
PCA compression of DNN layer

Step 1: Get ImageNet validation set of 50,000 images (possibly include test set of 150,000 images)

Step 2: Push images through network in batches, extract layer activation, flatten and store on hard drive



Step 3: Run incremental PCA or random projection (e.g. in scikit-learn), set number of PCs to a reasonable number (e.g. 1000)



Step 4: Save PCA model, push new images through network, extract layer activation, flatten and apply transformation from PCA

Take-home messages

Comparing brains and DNNs is easy, but what to do with it is harder

Common methods to map DNNs and brains are regression-based and similarity-based encoding methods

DNNs often treated only loosely as brain model (e.g. taking all layers to predict activity in V1)

Even older models (e.g. AlexNet) perform well and are still common