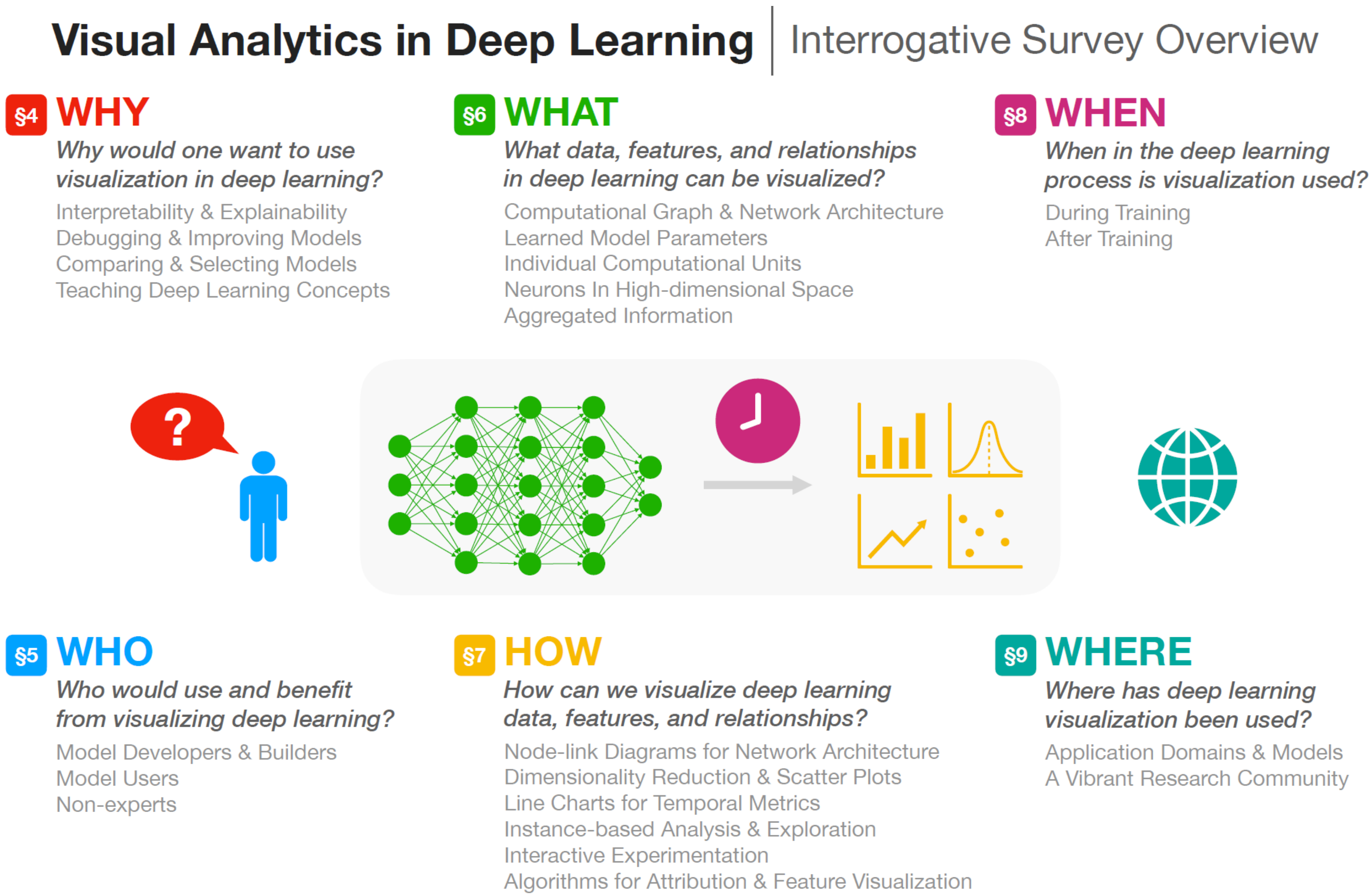
Machine learning for vision and multimedia
(01URPOV)

Lab 04 – Visualizing deep neural networks
Lia Morra

2024 – 2025

Politecnico di Torino

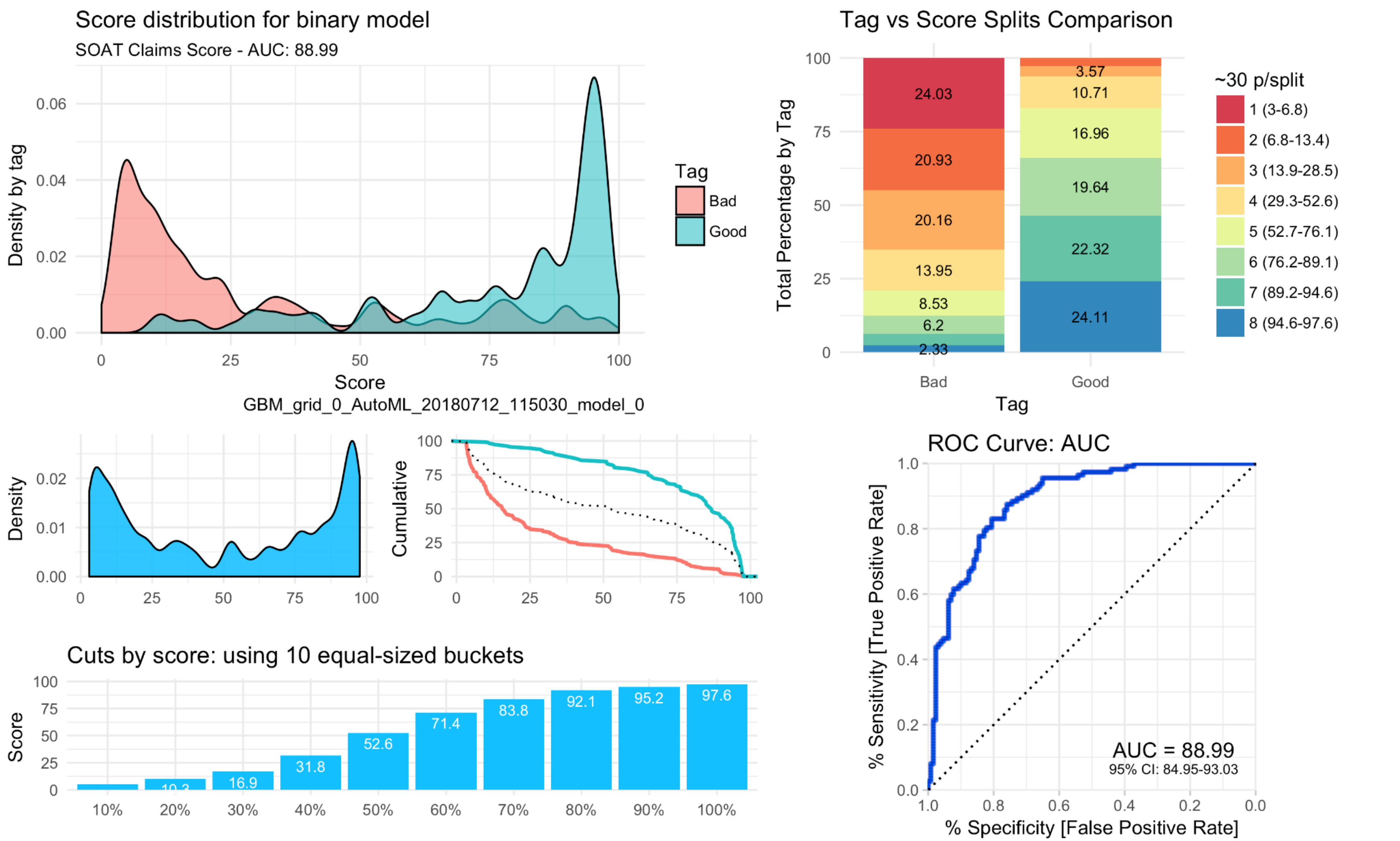
Visual analysis in deep learning



[Hohman, Fred, et al. 2018 "Visual analytics in deep learning: An interrogative survey for the next frontiers."]

MODEL PERFORMANCE

Performance: binary classifiers

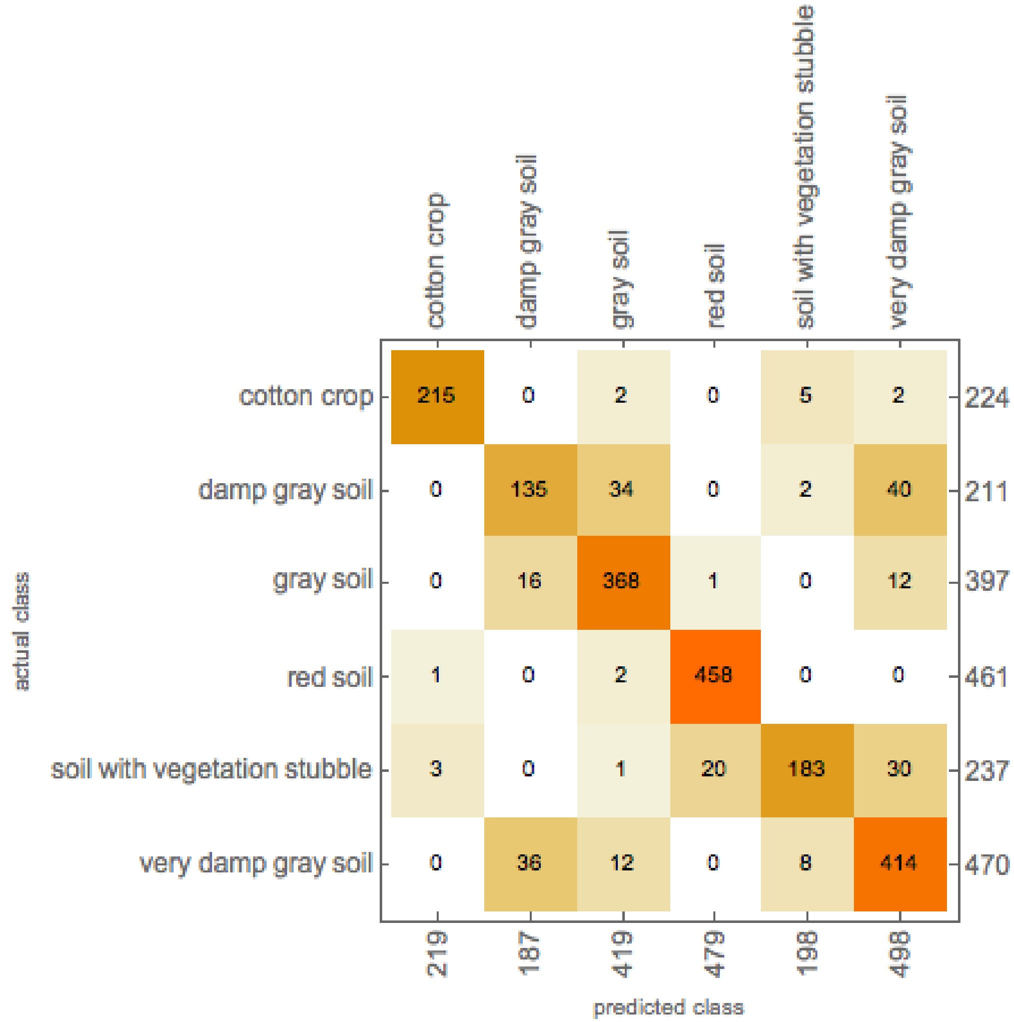


[Source: https://datascienceplus.com/machine-learning-results-one-plot-to-rule-them-all/]

Confusion matrix

Compare predicted and actual class

Color code for more immediate interpretation



MODEL TRAINING

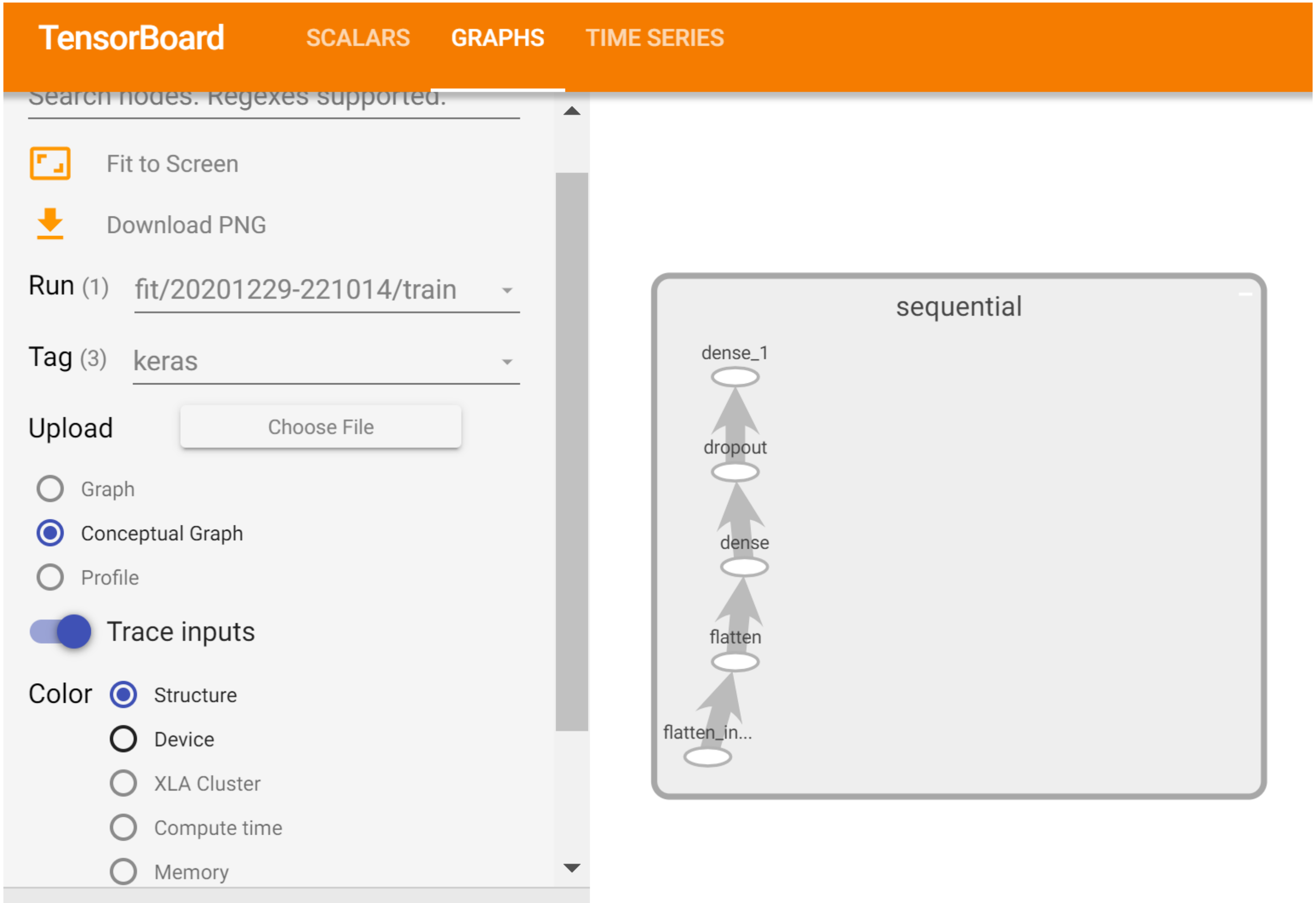
Tensorboard

- Default TF & Keras visualization tool
- Mostly used for managing and debugging training
- Different types of visualization including
  - **Graph dashboard**: Model graph, at conceptual and op-level
  - **Scalar dashboard**: tracking and visualizing loss and metrics, at batch and epoch level, during time
  - **Distributions and histograms dashboard**: Histograms of weights, biases, activations, gradients and how they change during training
  - **Tensorboard projector**: projecting embeddings on a lower dimensional space

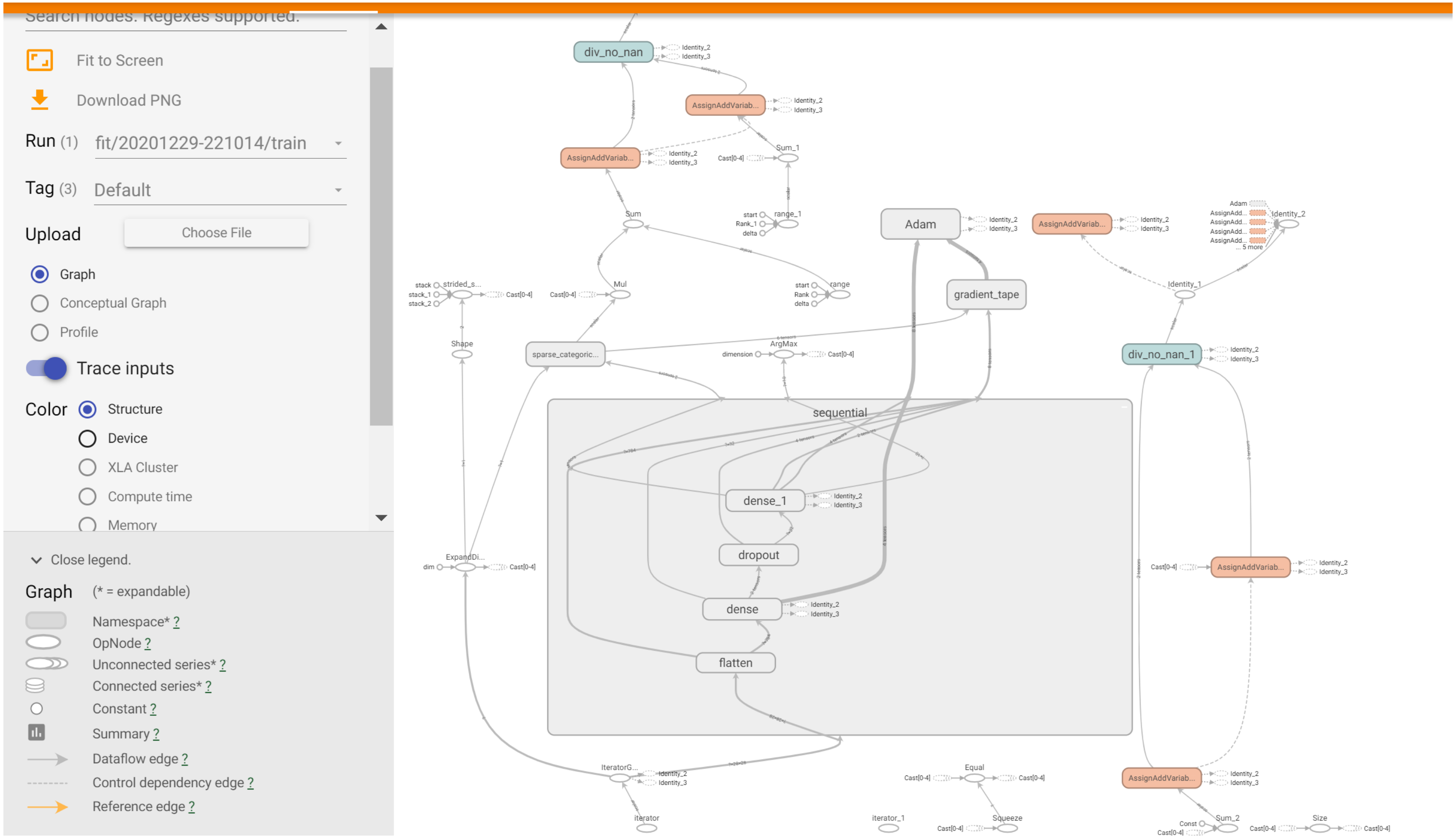
Using tensorboard

- Tensorboard relies on log files that are created and updated during training
  - In Keras, this can be obtained using the tf.keras.callbacks.Tensorboard callback
- The Tensorboard application is launched separately from the training process
  - The interface can be opened using any web browser
  - Tensorboard can be stopped or launched independently of the training process
  - Integration with Colab is available

Conceptual graph

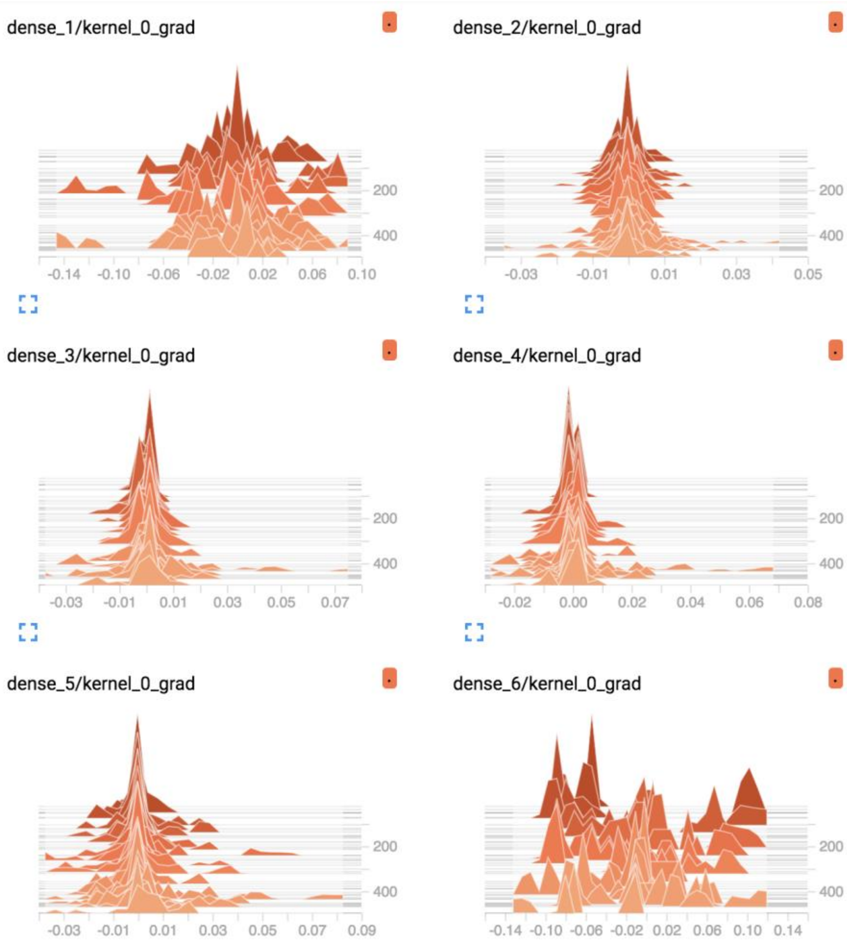


Op-level graph

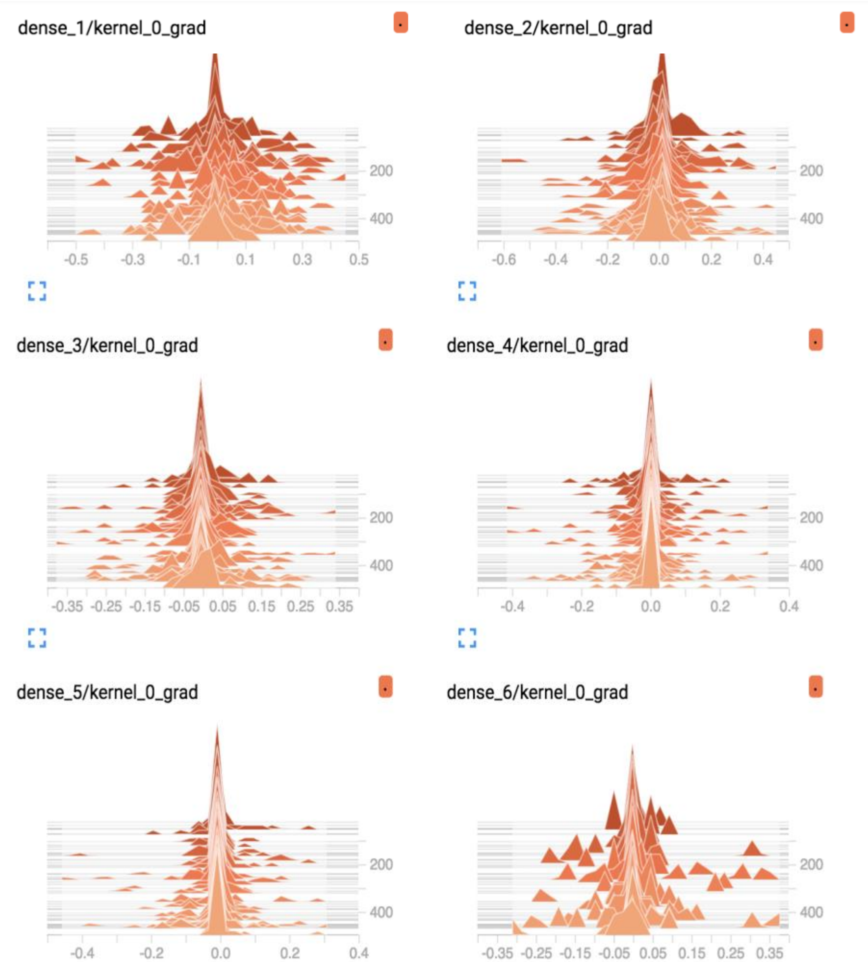


Distributions

MLP with tanh activations



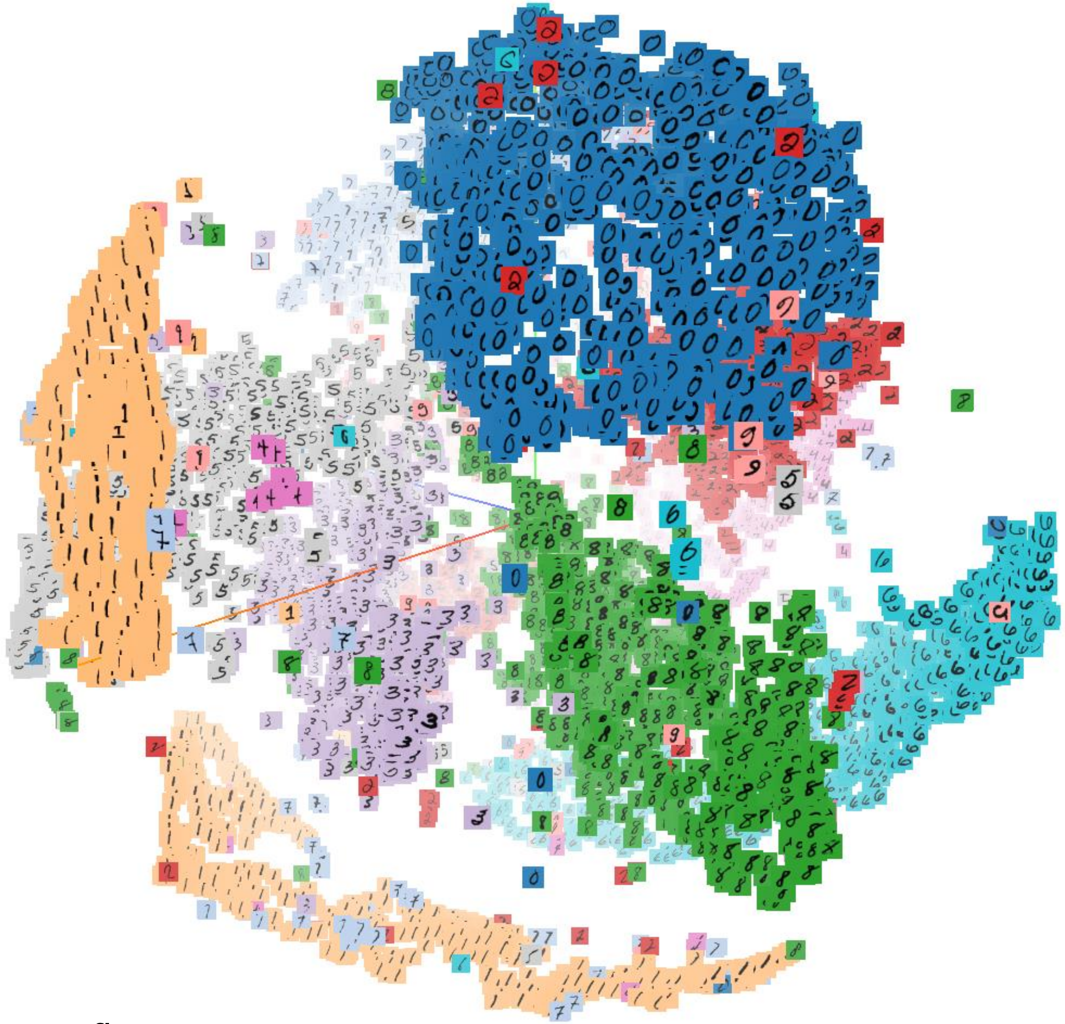
MLP with RELU activations



[https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function]

MODEL FEATURE SPACE

Embeddings

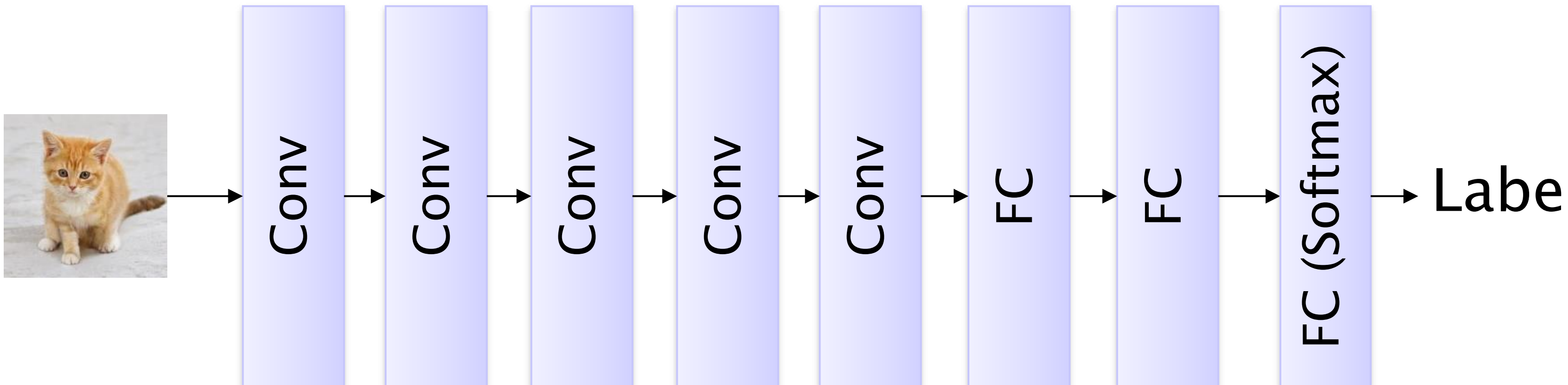


https://projector.tensorflow.org/

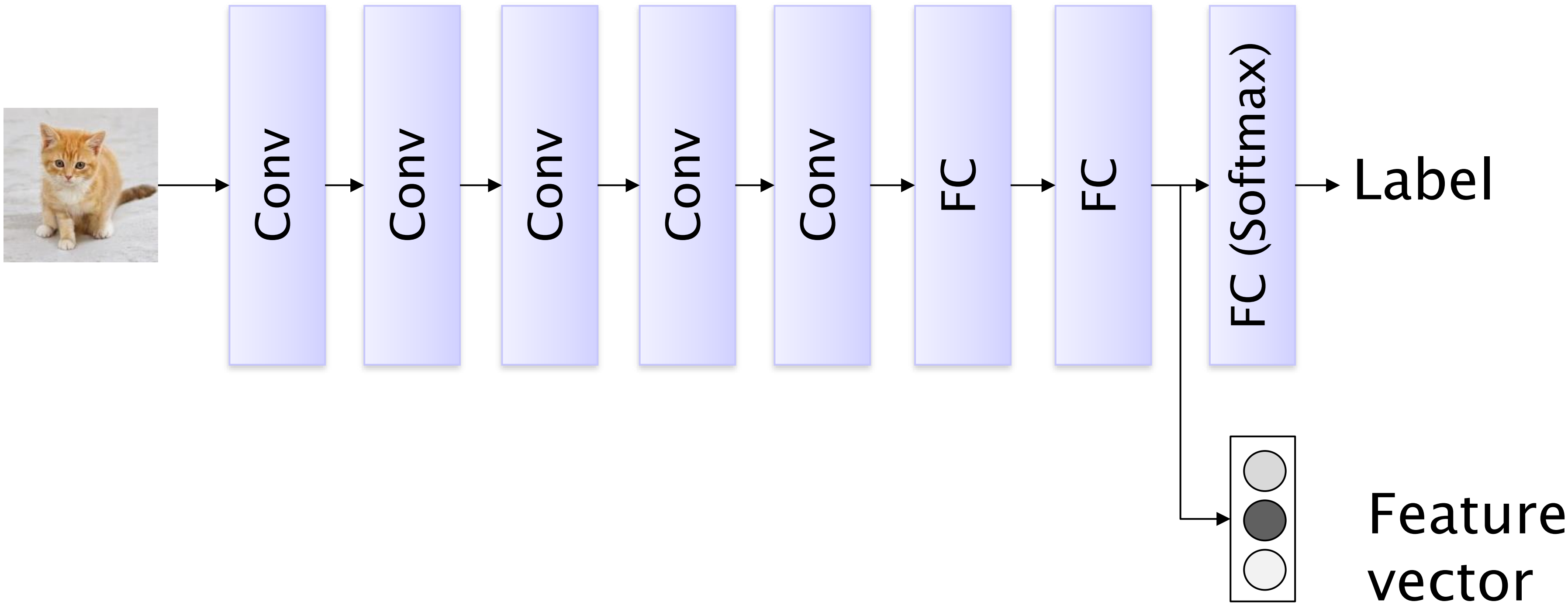
Visualizing embeddings

- Deep neural networks learn to map input data (e.g. images) into a lower-dimensional feature space
- Visualizing this feature space is important to understand its properties
  - Are samples from the same class nicely clustered?
  - Which examples are similar for the network?
- Dimensionality reduction techniques are needed to project the embedding space in 2 or 3 dimensions
- Available techniques include:
  - Principal Component Analysis (PCA)
  - t-Student Stochastic Neighbor Embeddings (tSNE)
  - Uniform Manifold Approximation and Projection (UMAP)

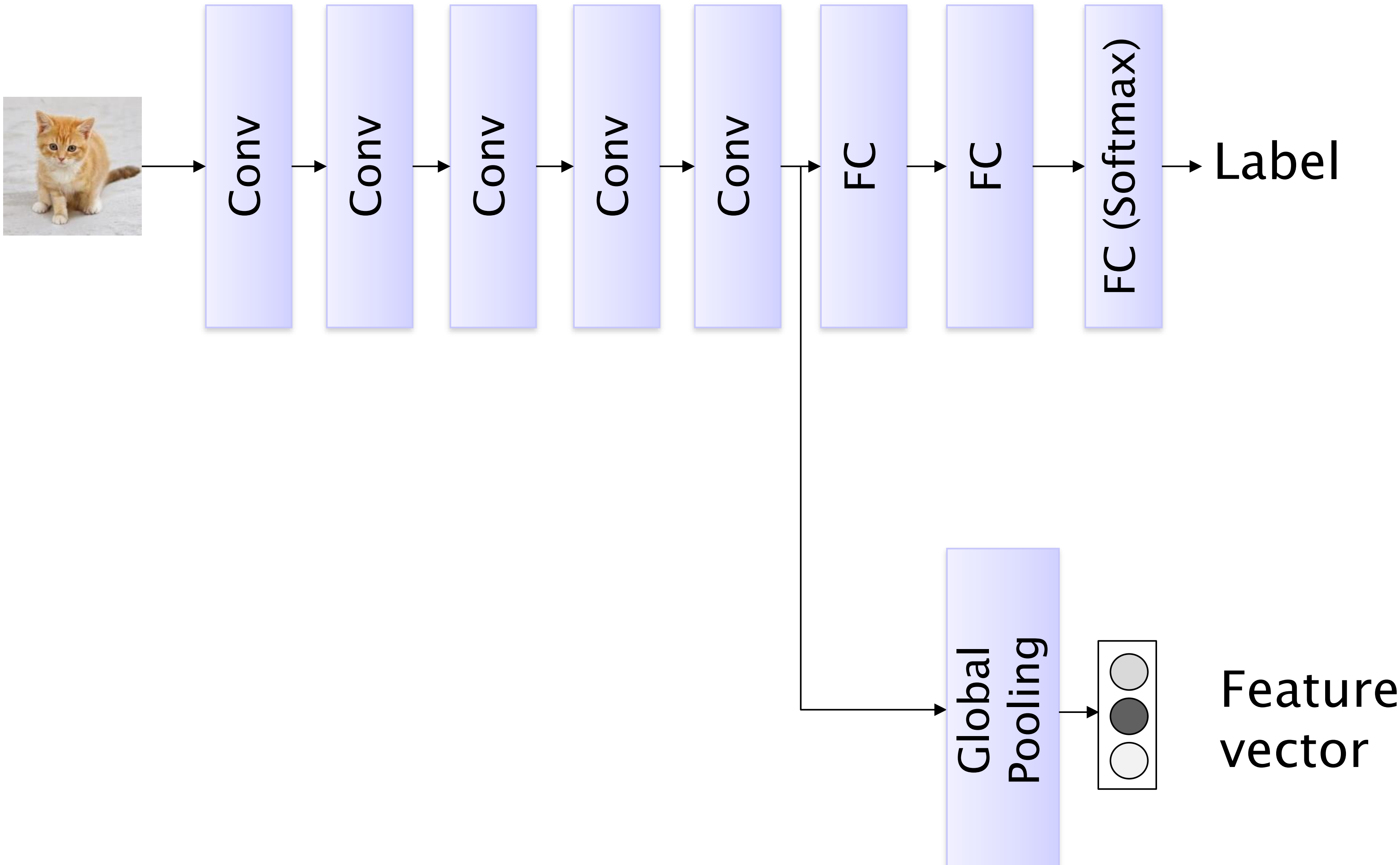
Extracting embeddings



Extracting embeddings



Extracting embeddings



tSNE embedding algorithm

1. (Optional) Use PCA to reduce the input dimensions into a smaller number
2. Construct a probability distribution over pairs of original high dimensional records: high probability ≈ low distance
3. Define a similarity probability distribution of the points in the low-dimensional embedding
4. Minimize the distance between the two distributions using gradient descent method

tSNE embedding algorithm

- High-dim space:
  - Gaussian distribution
  - Focus on local neighbors
  - Depends on perplexity parameter (number of expected neighbours per point)
  - Perplexity typically between 5 and 50

$$p_{j|i} = \frac{\exp(-\left|x_i - x_j\right|/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\left|x_i - x_k\right|/2\sigma_i^2)}$$

x_i, x_j, x_k represent the original samples (training set)

tSNE embedding algorithm

- High-dim space:
  - Gaussian distribution
  - Focus on local neighbors
  - Depends on perplexity parameter

$$p_{j|i} = \frac{\exp(-\left|x_i - x_j\right|/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\left|x_i - x_k\right|/2\sigma_i^2)}$$

- Low-dim space:
  - Student-t distribution
  - Heavy tails compensates for lower dimensionality

y_i, y_j, y_k represent the projections on low-dim space (output)

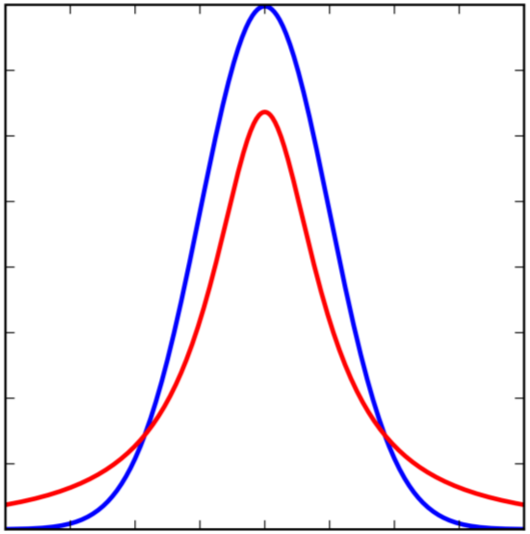
$$q_{ij} = \frac{\mathrm{f}(\left|y_i - y_j\right|)}{\sum_{k\neq i}\mathrm{f}(\left|y_i - y_k\right|)} \quad \text{with } f(z) = \frac{1}{1+z^2}$$

tSNE embedding algorithm

- High-dim space:
  - Gaussian distribution
  - Focus on local neighbors
  - Depends on perplexity parameter

- Low-dim space:
  - Student-t distribution
  - Heavy tails compensates for lower dimensionality

$$p_{j|i} = \frac{\exp(-|x_i - x_j|/2\sigma_i^2)}{\sum_{k\neq i} \exp(-|x_i - x_k|/2\sigma_i^2)}$$

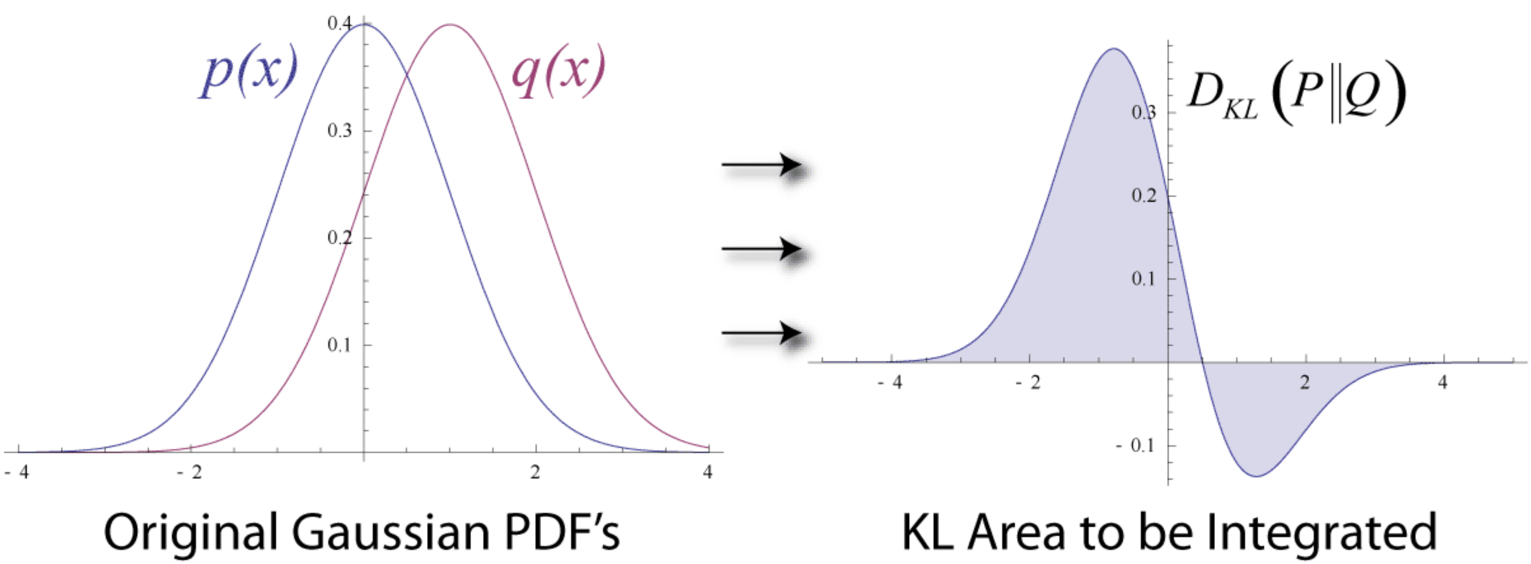


$$q_{ij} = \frac{\mathrm{f}(|y_i - y_j|)}{\sum_{k\neq i} \mathrm{f}(|y_i - y_k|)} \text{ with } f(z) = \frac{1}{1+z^2}$$

Kullback–Leiber divergence

Measures how one probability distribution P differs from a second, reference probability distribution Q

$$KL(P \parallel Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx$$



tSNE embedding algorithm

- Minimize Kullback-Leiber divergence by gradient descent optimization

$$KL(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

- KL divergence is not symmetrical
  - large p ("similar" points in input space) modelled with small q ("dissimilar" points in output space) → big penalty
  - small p ("dissimilar" points in input space) modelled with large q ("similar" points in output space) → small penalty
  - preserves local (not global) structure: distance between clusters may not be meaningful

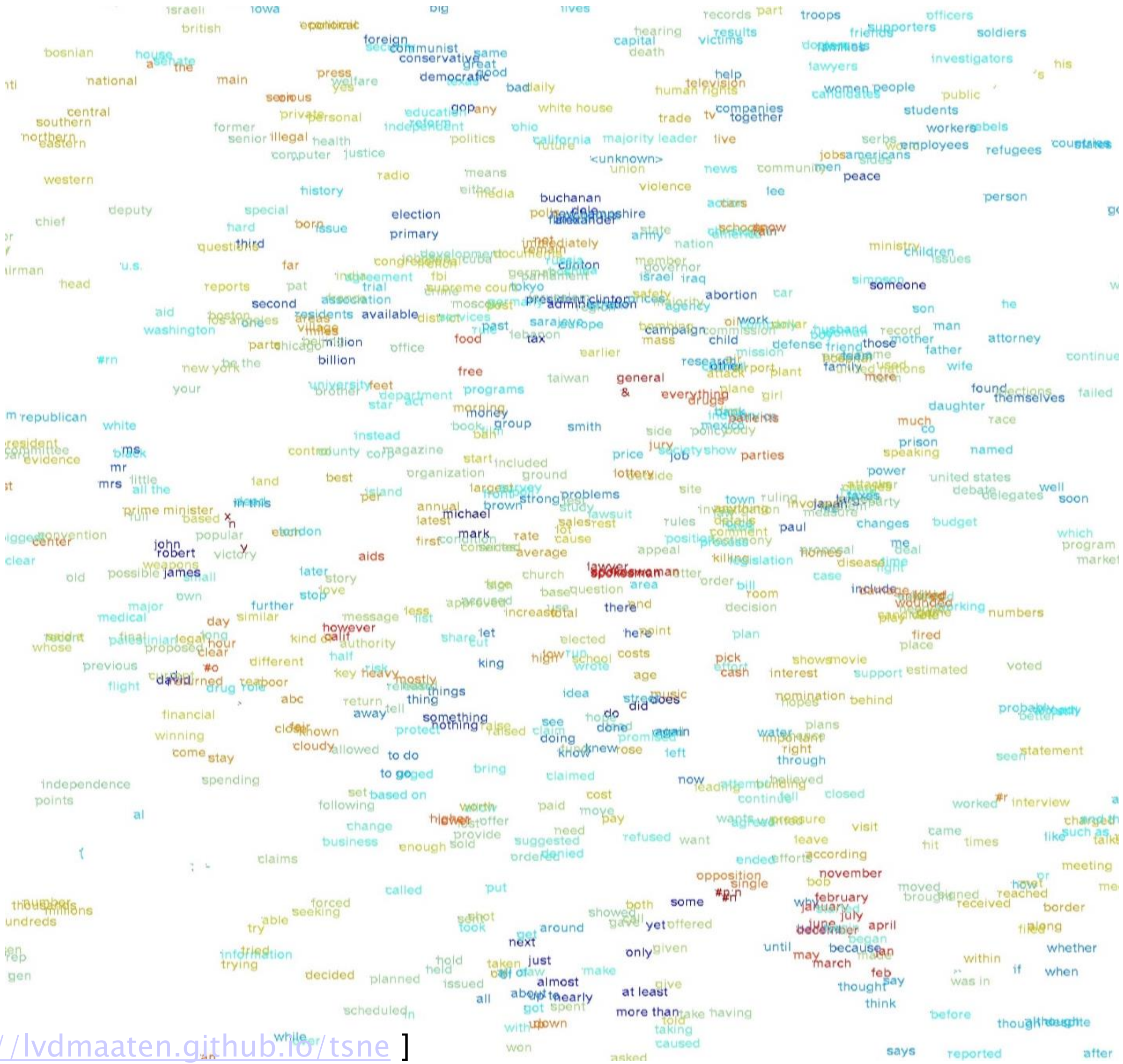
tSNE hyperparameters

- Perplexity:
  - balance attention between local and global aspects of the dataset
  - best guess about the number of close neighbors
  - good values between 5 and 50, but important to try different values in real parameters
  - must be lower than the number of input records
- Learning rate, training iterations:
  - train until convergence
  - beware that multiple iterations will results in different visualizations

tSNE example: Caltech101

[source: https://lvdmaaten.github.io/tsne]

tSNE example: word embeddings



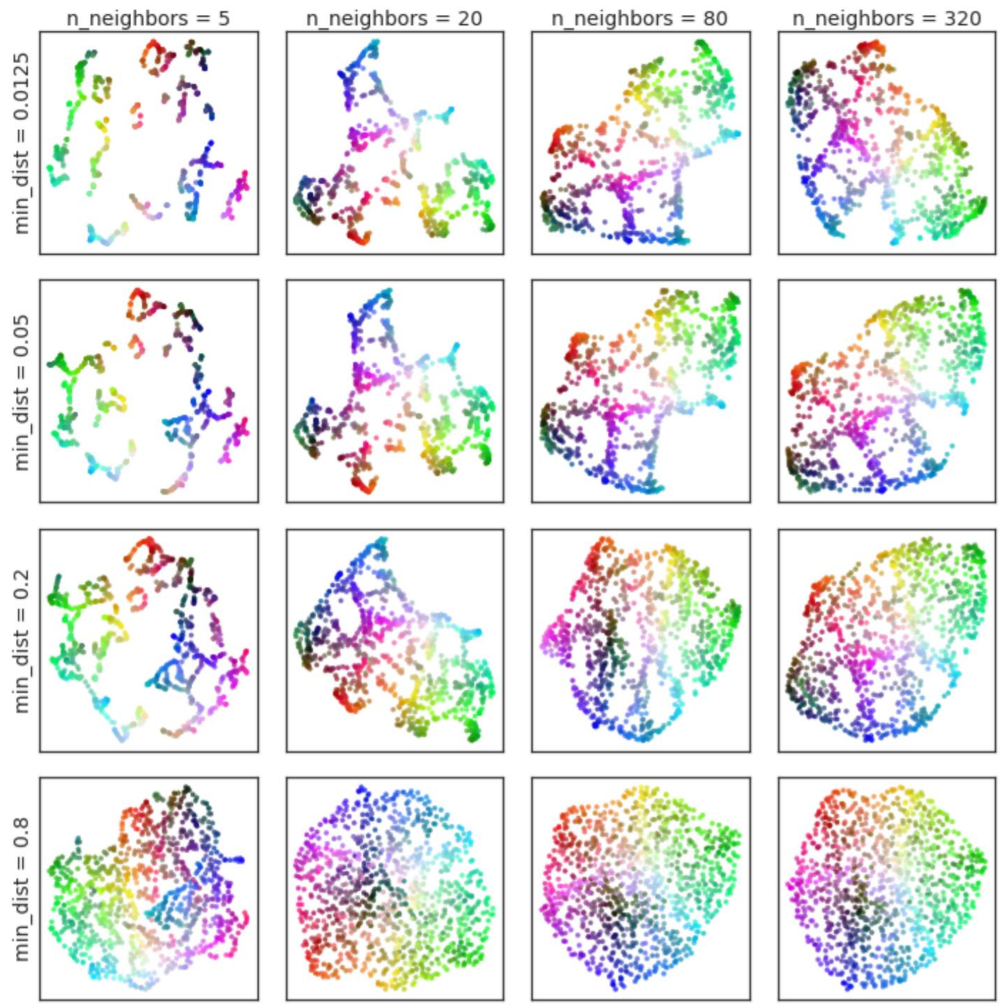
[source: https://lvdmaaten.github.io/tsne]

Interpreting embeddings

- Manifold learning algorithms assume that there is a structure in the data
- Highly non-linear embeddings can fit a structure to the noise of a dataset – similarly to humans!
- Care must be taken with small sample sizes of noisy data, or data with only large scale manifold structure
- Shape may not be always preserved.
  - See https://distill.pub/2016/misread-tsne/

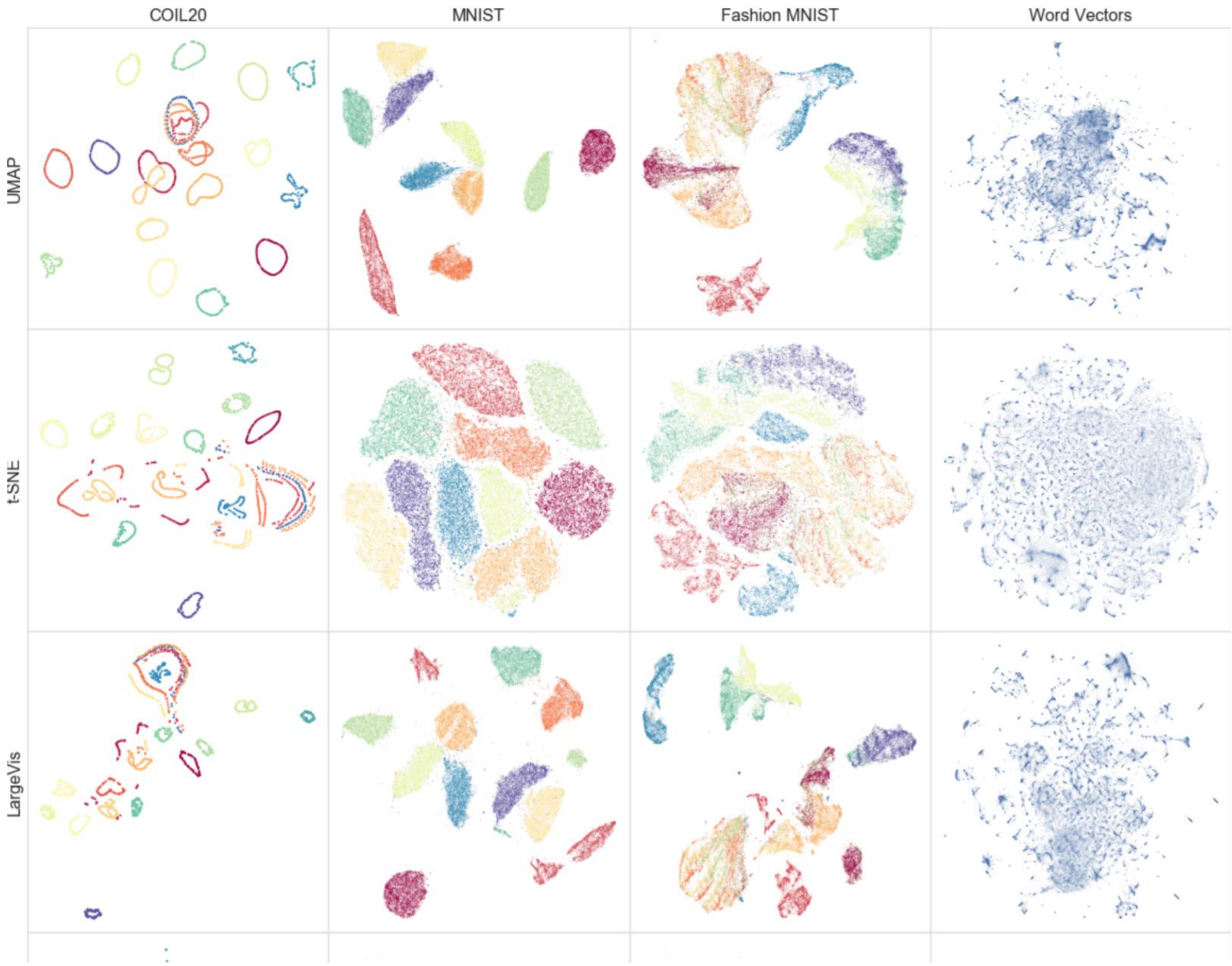
Interpreting embeddings

- Example: randomly sampled data in the RGB domain (3 to 2 dimensions)
- UMAP trained with different choices of hyper-parameters shows emerging spurious structures



[McInnes et al. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.]

Interpreting embeddings (II)

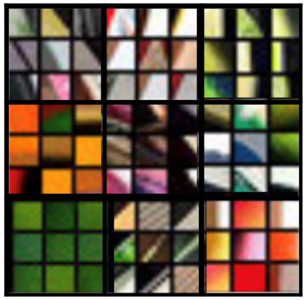


FEATURE VISUALIZATION

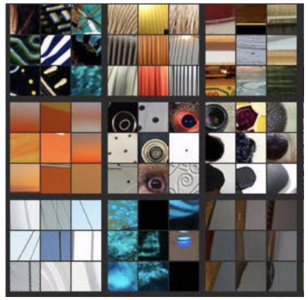
Visualizing features

- Visualizing the activations of individual neurons may be long and not very informative (networks are large and deep, activation maps difficult to interpret and sparse)
- There are fundamentally two ways of visualizing individual neurons
  - Finding the patterns that optimize its response
  - Finding images (or patches) that activate (high-level) neurons: can we see any pattern emerge?
- Visualizing features before/after training shows "repurposing" of individual neurons

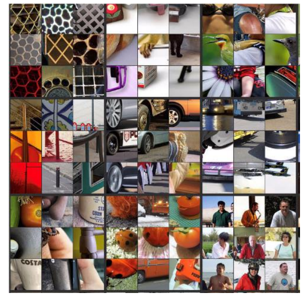
Visualize what network is learning



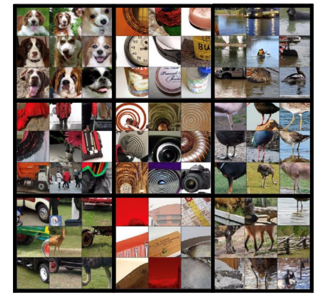
Layer 1



Layer 2



Layer 3

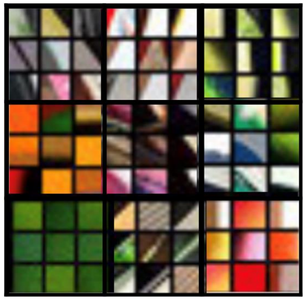


Layer 4

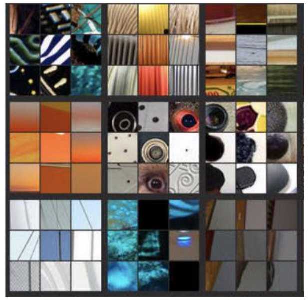
Layer 5

[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

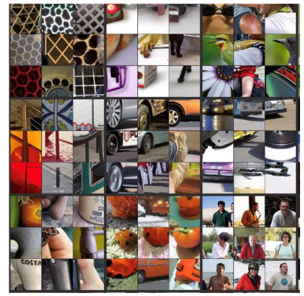
Visualize what network is learning



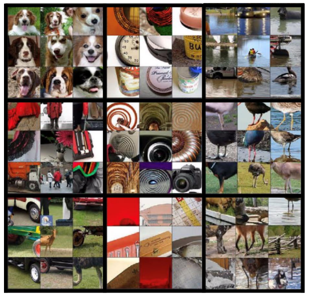
Layer 1



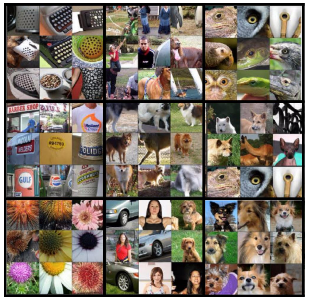
Layer 2



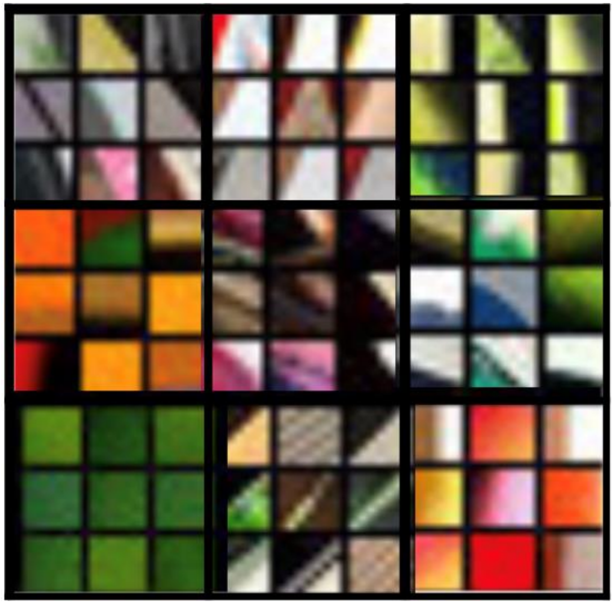
Layer 3



Layer 4

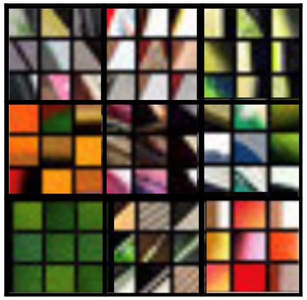


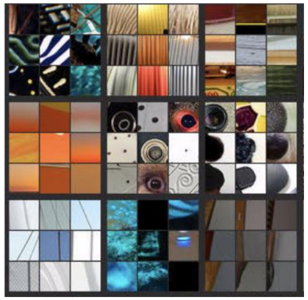
Layer 5



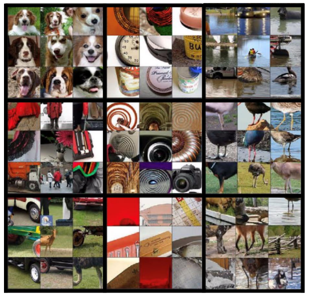
[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

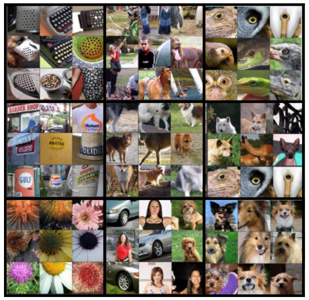
Visualize what network is learning

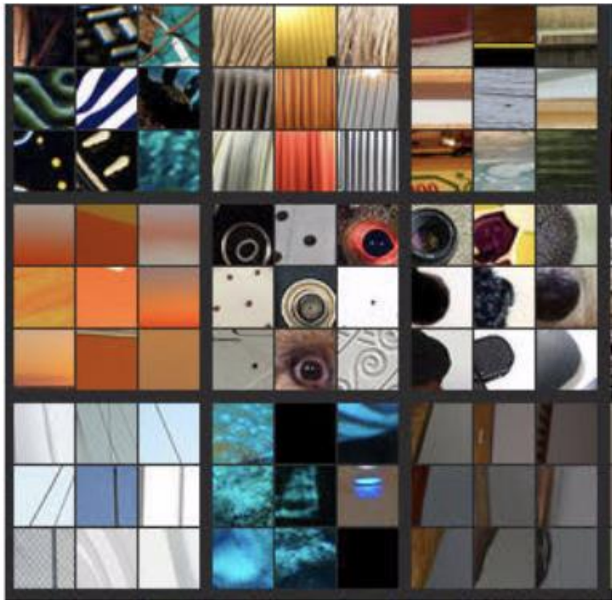

Layer 1


Layer 2


Layer 3

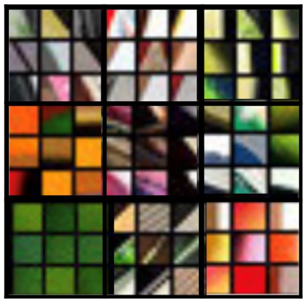

Layer 4


Layer 5

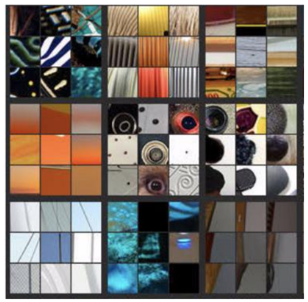


[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

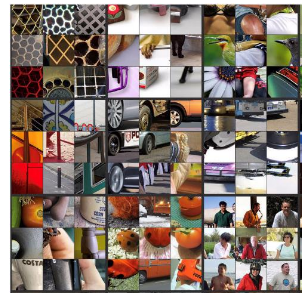
Visualize what network is learning



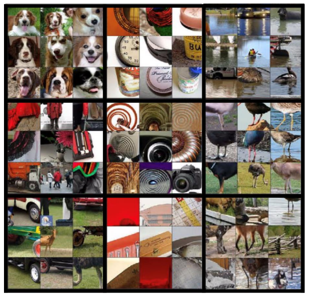
Layer 1



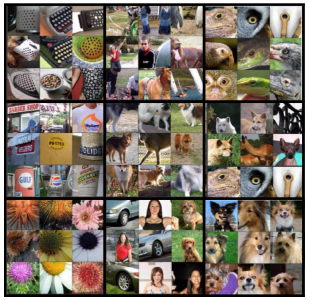
Layer 2



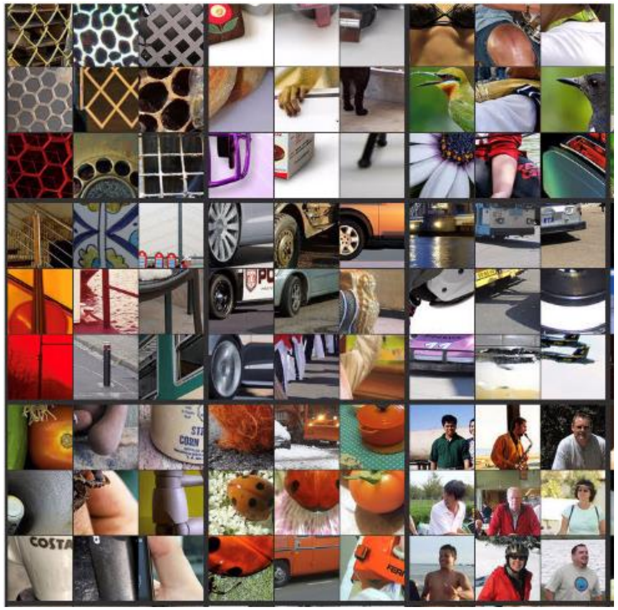
Layer 3



Layer 4

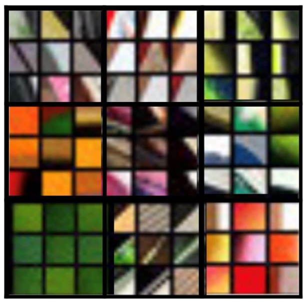


Layer 5

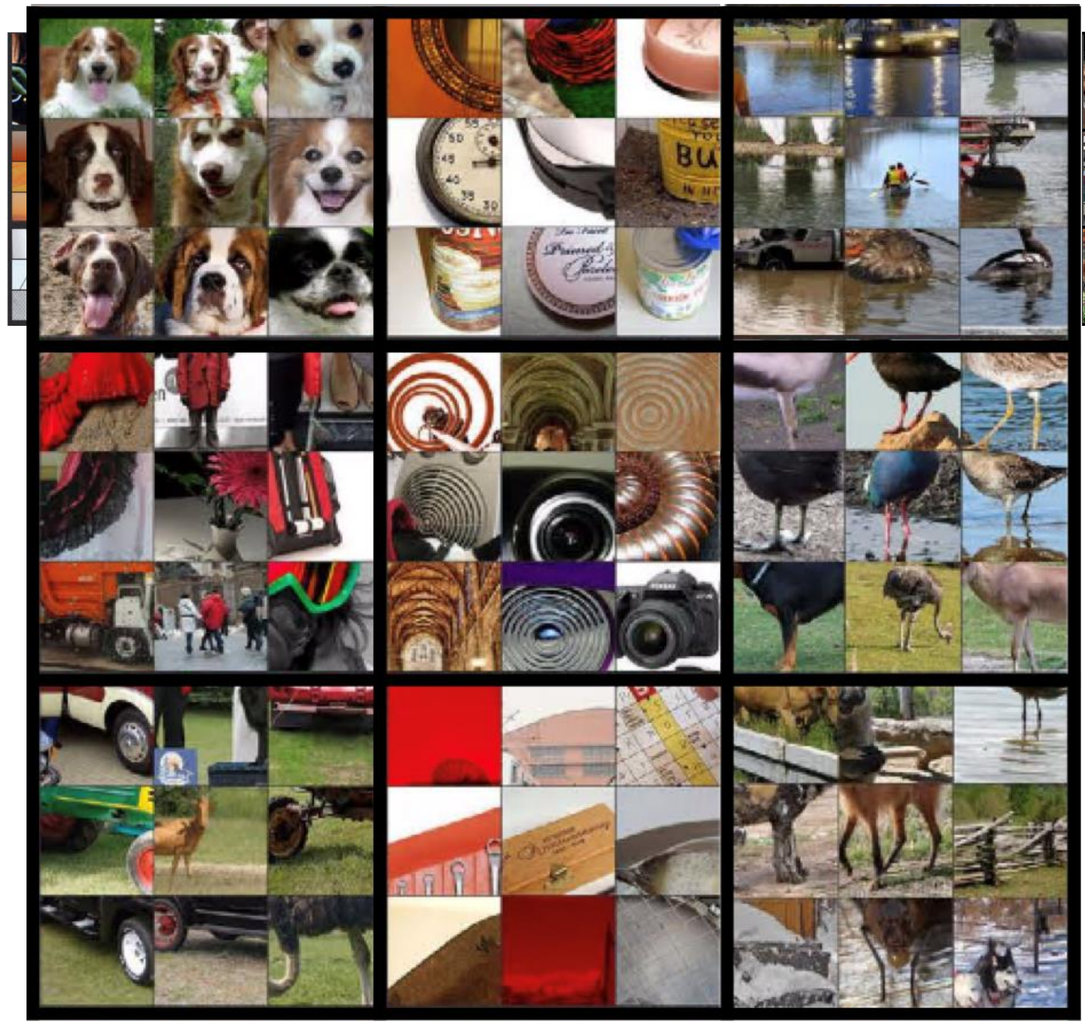


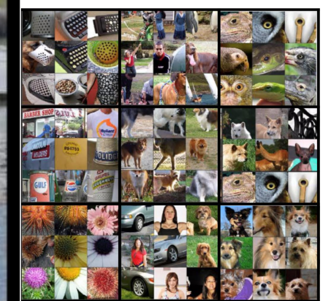
[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

Visualize what network is learning



Layer 1

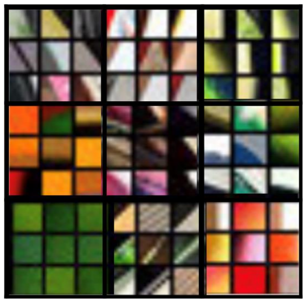




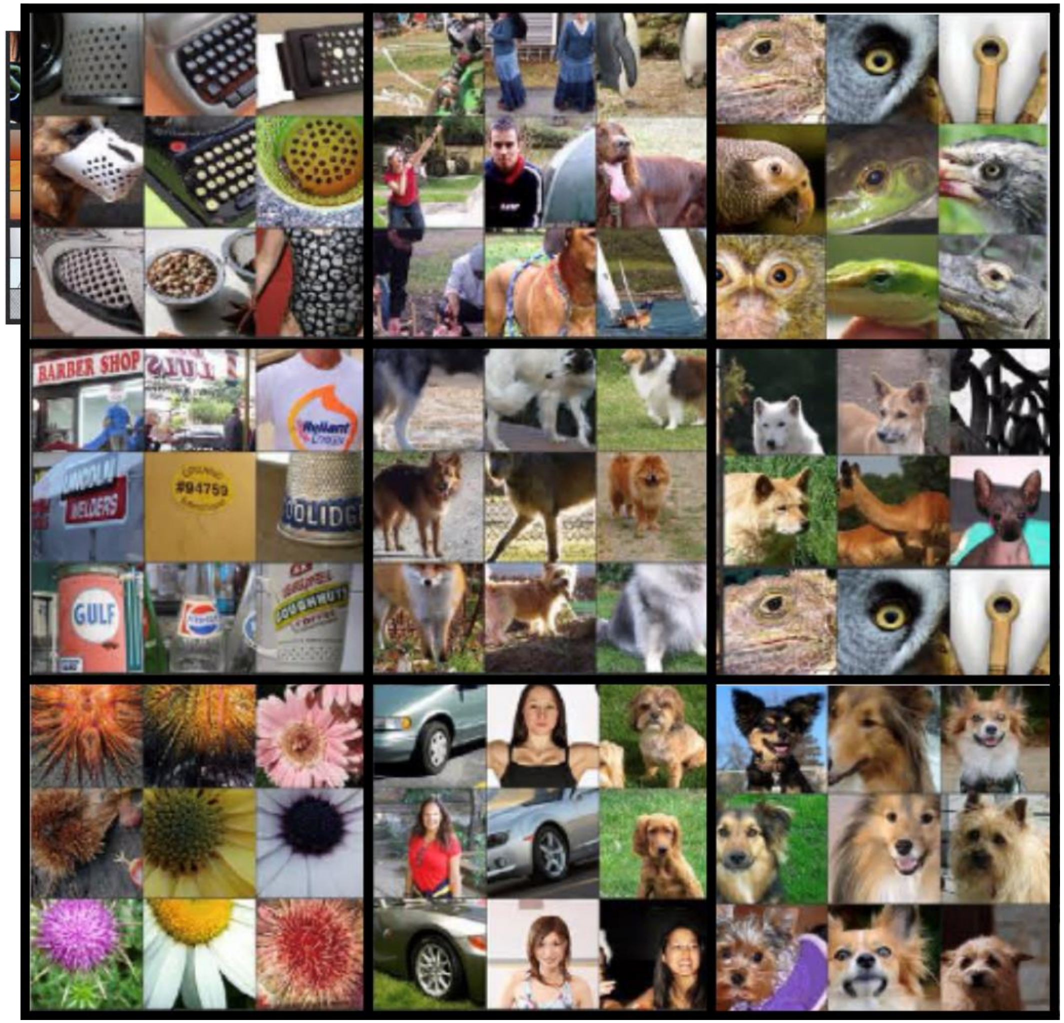
Layer 5

[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

Visualize what network is learning



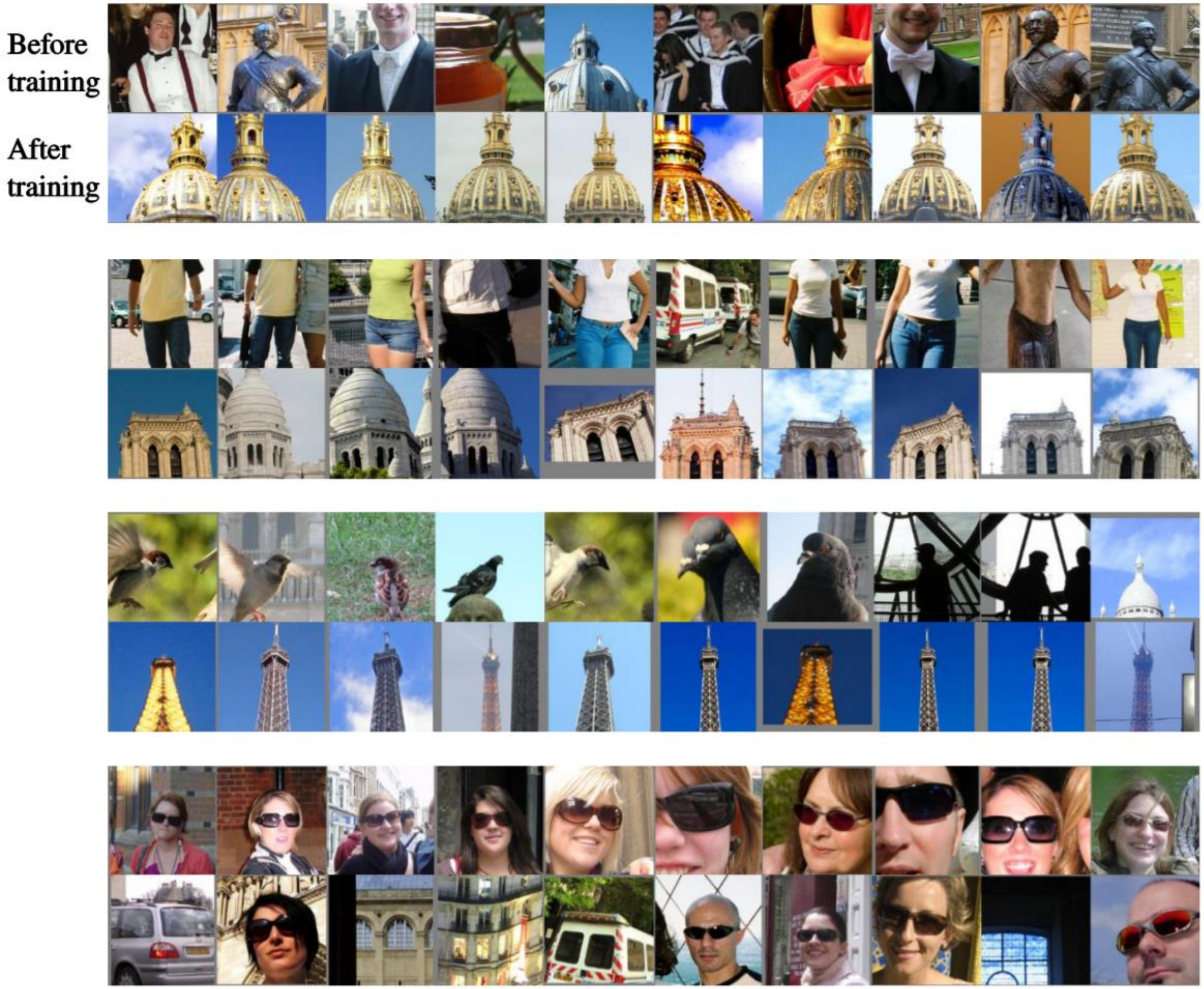
Layer 1



Layer 5

[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

Re–porpusing

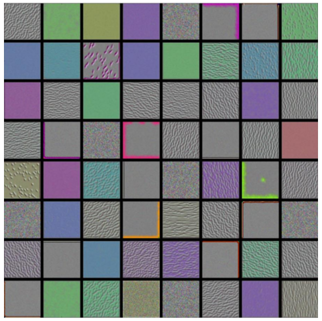


[Gordo, 2016, Deep Image Retrieval: Learning global representations for image search]

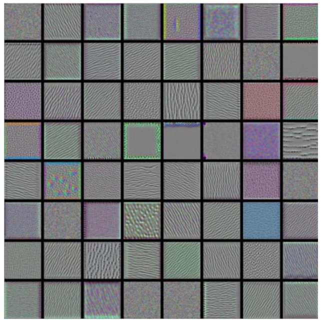
Visualization by gradient ascent

- Another easy way to inspect the filters learned by convolutional networks is to display the visual pattern that each filter is meant to respond to.
- This can be done with **gradient ascent** in input space: applying gradient descent to the value of the input image to maximize the response of a specific filter, starting from a random input image.
- The resulting input image will be one that the chosen filter is maximally responsive to.

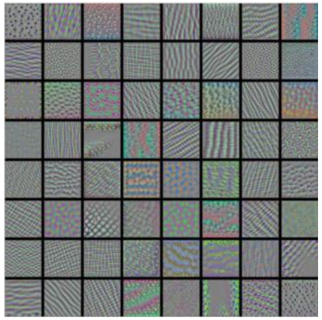
Gradient ascent



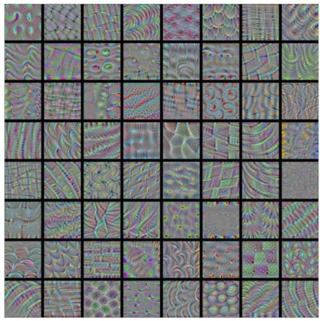
Layer 1



Layer 2

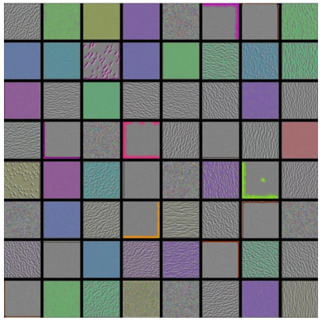
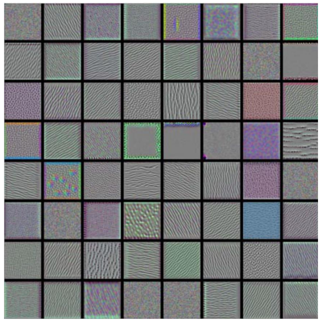
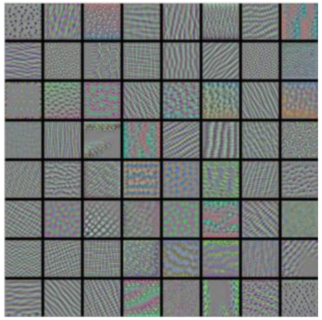
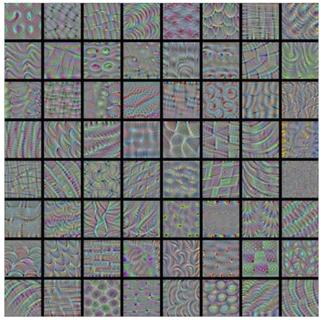


Layer 3

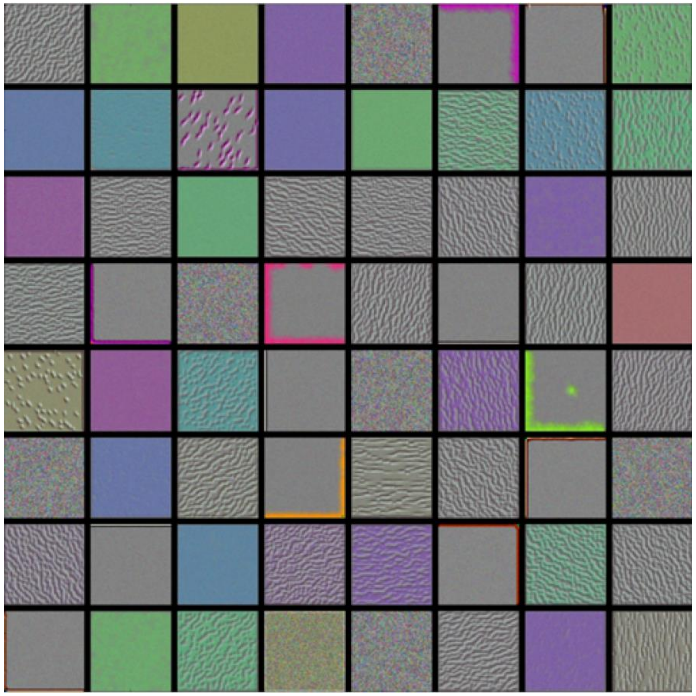


Layer 4

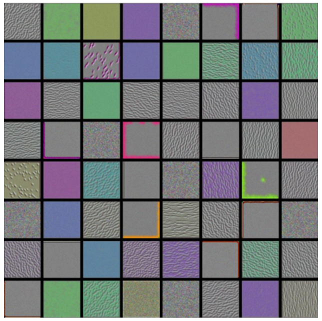
Gradient ascent

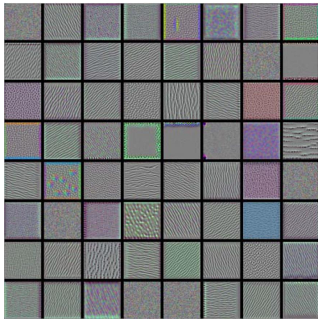
Layer 1 Layer 2 Layer 3 Layer 4



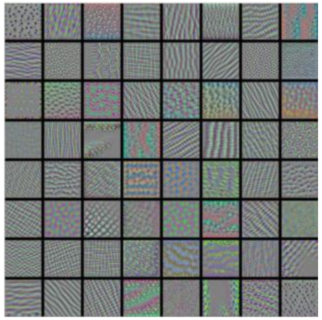
Gradient ascent



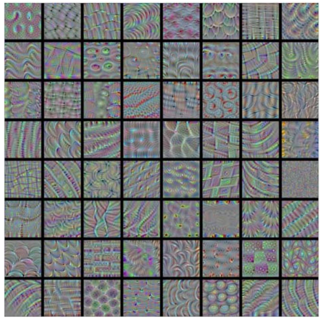
Layer 1



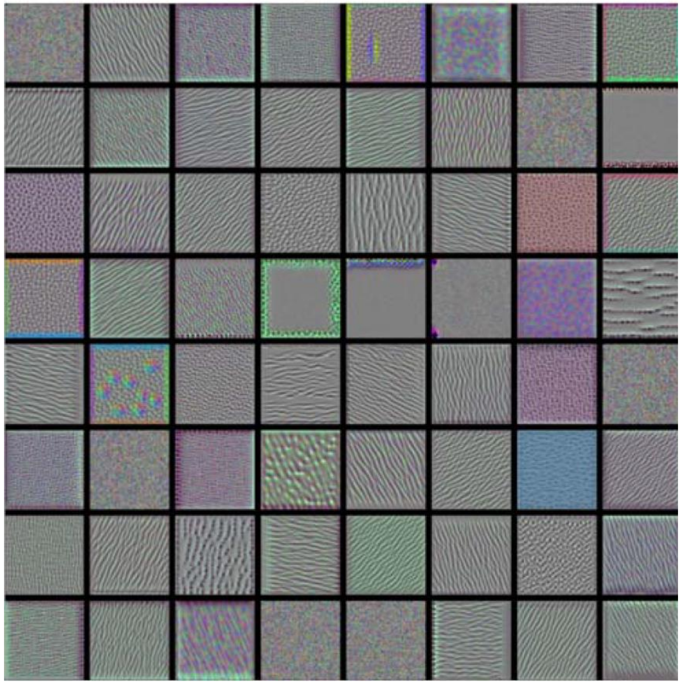
Layer 2



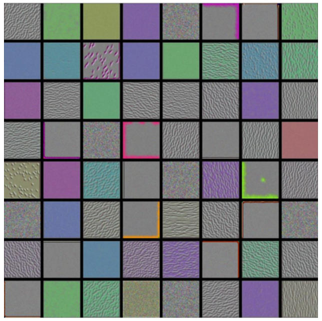
Layer 3



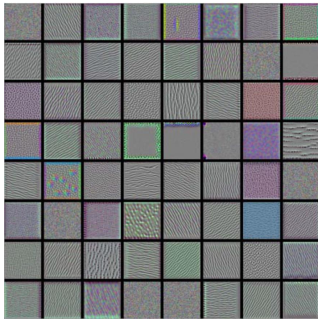
Layer 4



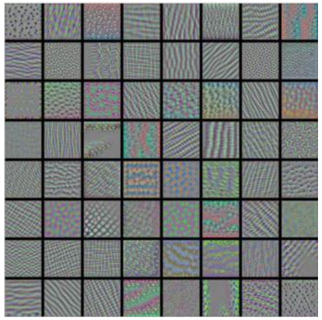
Gradient ascent



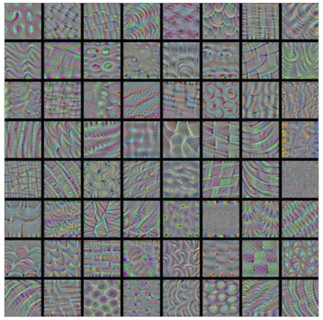
Layer 1



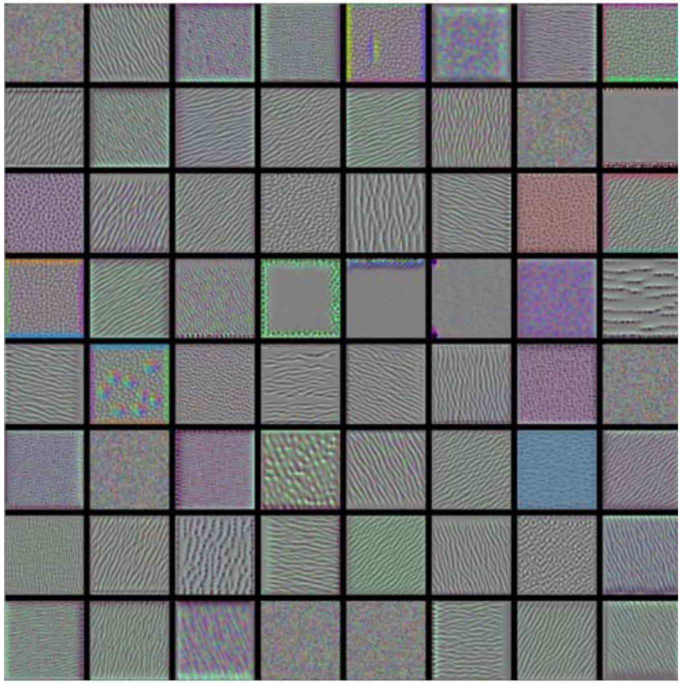
Layer 2



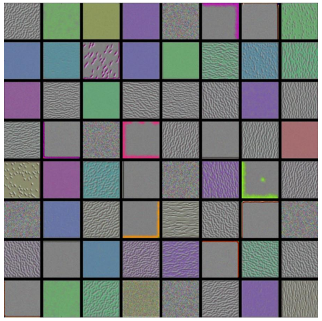
Layer 3



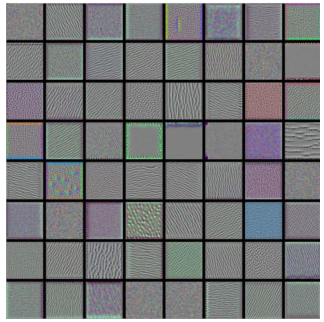
Layer 4



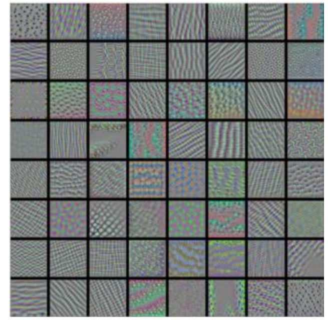
Gradient ascent



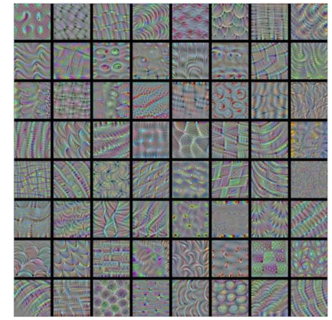
Layer 1



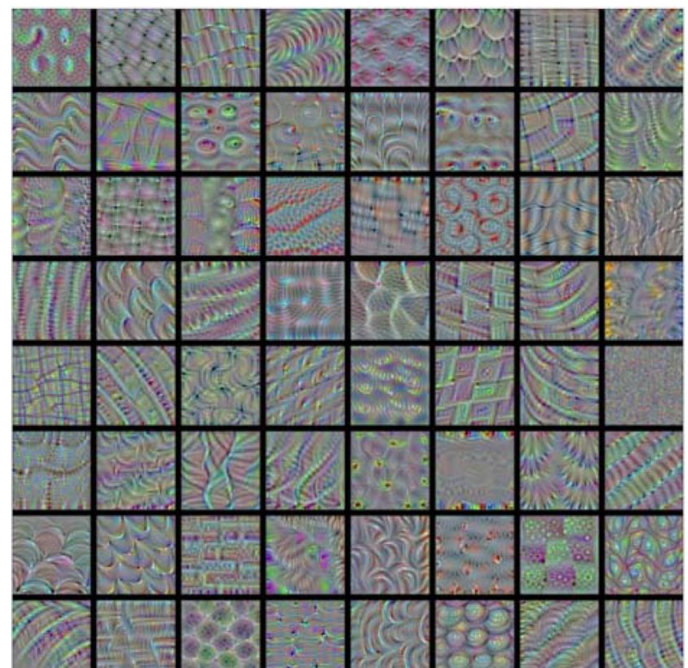
Layer 2



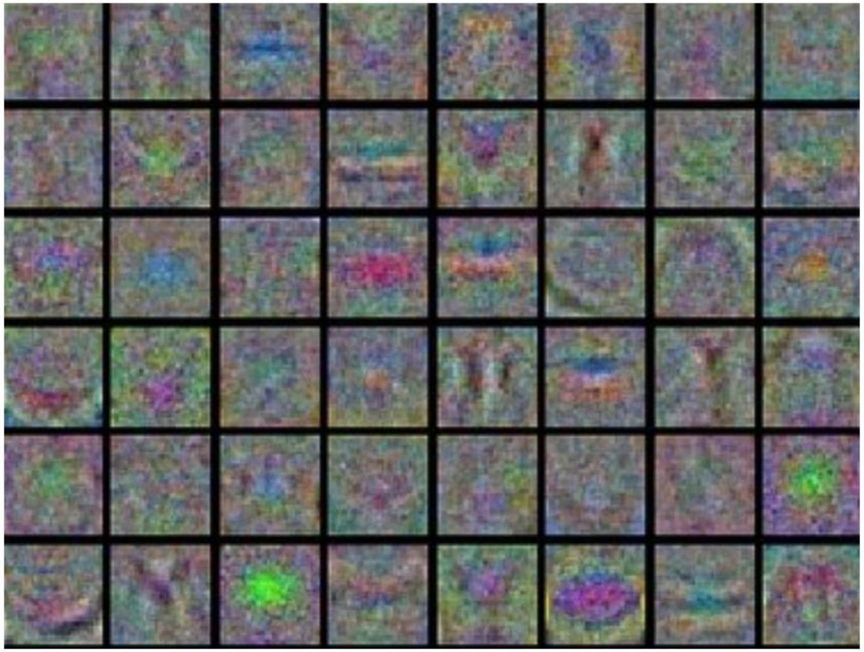
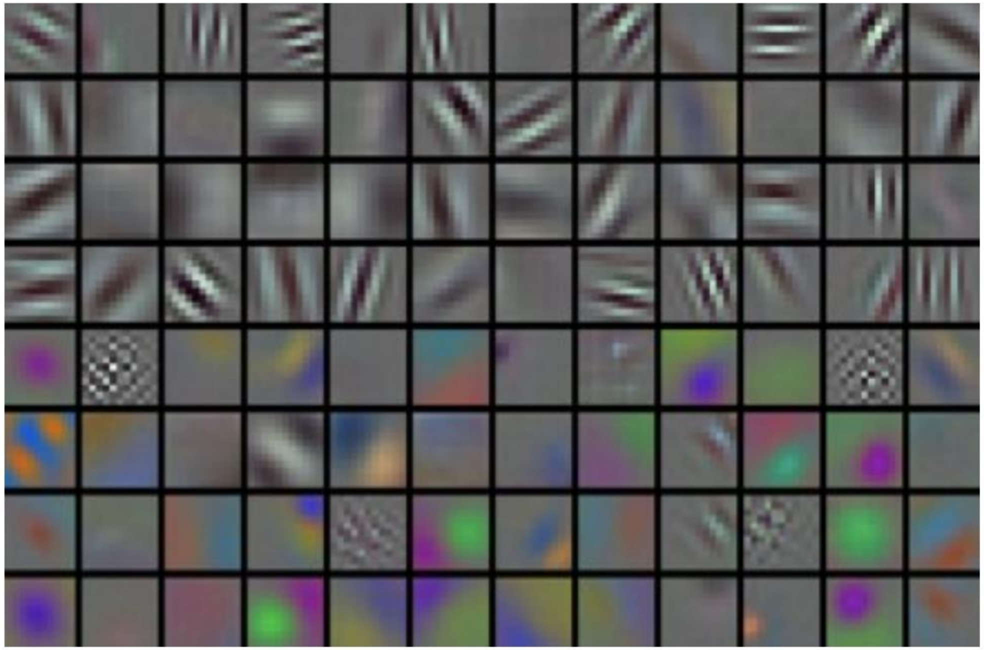
Layer 3



Layer 4



Visualize filter kernels

Filter kernels can be visualized directly as images
Noisy features (left) are usually the sign of poor training (low learning rate, low number of epochs)

[Source: https://cs231n.github.io/neural-networks-3/#vis]

MODEL EXPLANATION

Model explanation

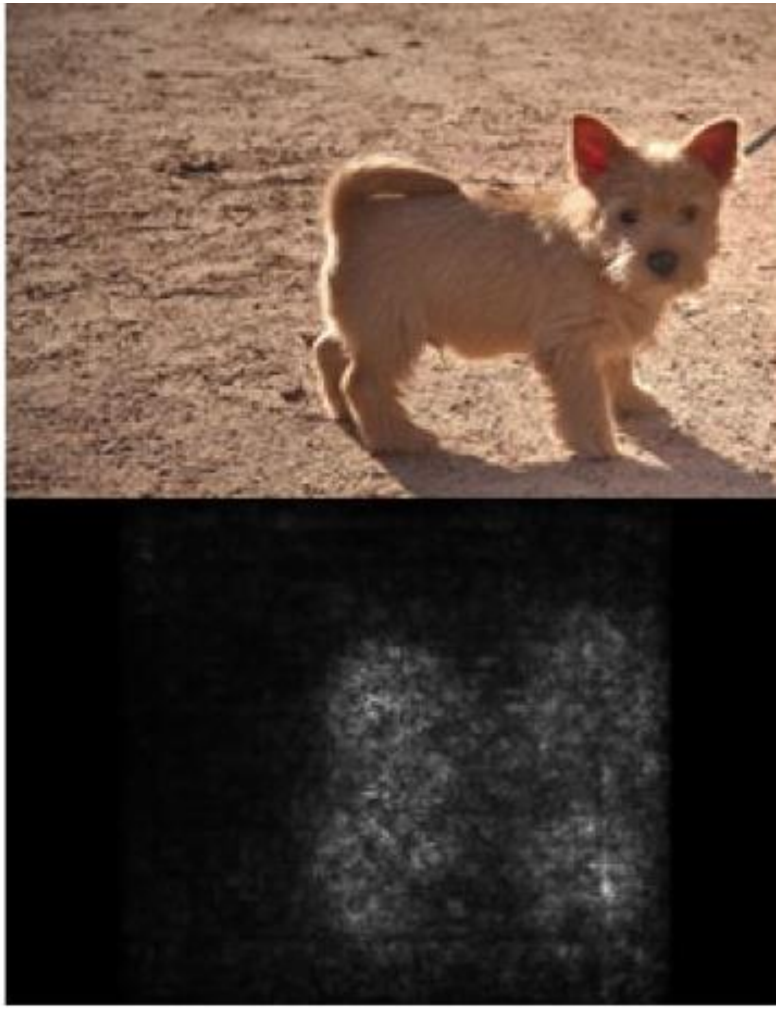
- “Explaining a prediction” → presenting a **qualitative** and possibly **quantitative** understanding of the **relationship** between the instance’s components (e.g. words in text, patches in an image) and the model’s prediction.
- Explanations are important to determine
  - Why and how a model is failing
  - Whether a model is correct “for the right reasons”
- Most techniques are relevance-based: which features activate a certain neuron/output? which features are most relevant for a certain classification?

Model explanation (III)

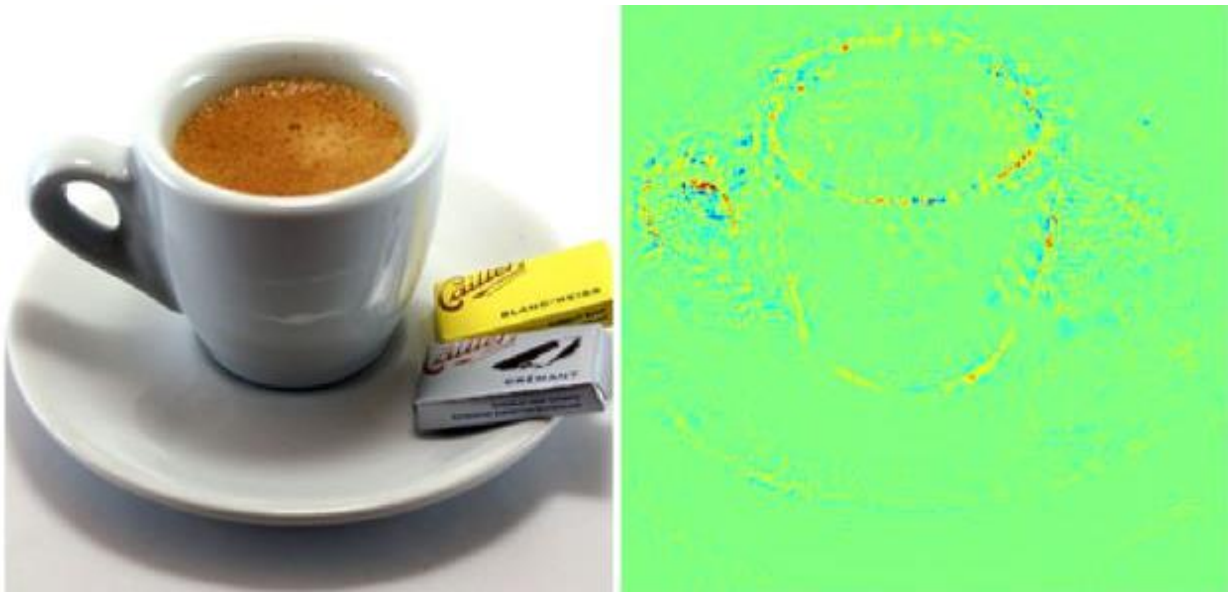
- Visualization is a key techniques for enhancing interpretability especially for CNNs
- Most techniques identify pixels which are mostly relevant to a given classification ("where" the network is looking)
- Different types of visualization techniques have been proposed
  - gradient-based
  - attribution-based
  - localization-based
  - perturbation-based methods

Gradient-based approaches

- Find the derivatives of class scores with respect to pixel intensities
  - Deconvolution
  - Guided Backpropagation
  - SmoothGrad
  - etc..



Attribution–based approaches



- For each pixel, measure its relevance to the score
  - LRP, DeepTaylor, PatternAttribution, LIME
- The key idea is to decompose the output in terms of contributions from its inputs
- Some techniques (e.g., LIME) areapplicable to other types of input features other then images

GRAD-CAM

- GRAD-CAM computes a heatmap which assess the importance of each pixel with respect to a given class c, with output score y^c
- Let A^k be the activation of the k^{th} neuron in a convolutional layer
- First, the importance of each neuron is computed

$$\alpha_c^k = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

GRAD-CAM

- GRAD-CAM computes a heatmap which assess the importance of each pixel with respect to a given class c, with output score y^c
- Let A^k be the activation of the k^{th} neuron in a convolutional layer
- First, the importance of each neuron is computed

Global Average Pooling

$$\alpha_c^k = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

GRAD-CAM

- GRAD-CAM computes a heatmap which assess the importance of each pixel with respect to a given class c, with output score y^c
- Let A^k be the activation of the k^{th} neuron in a convolutional layer
- First, the importance of each neuron is computed

Global Average Pooling

$$\alpha_c^k = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

Gradients of y^c w.r.t to filter k

GRAD-CAM

- GRAD-CAM computes a heatmap which assess the importance of each pixel with respect to a given class c, with output score y^c
- Let A^k be the activation of the k^{th} neuron in a convolutional layer
- First, the importance of each neuron is computed

Global Average Pooling

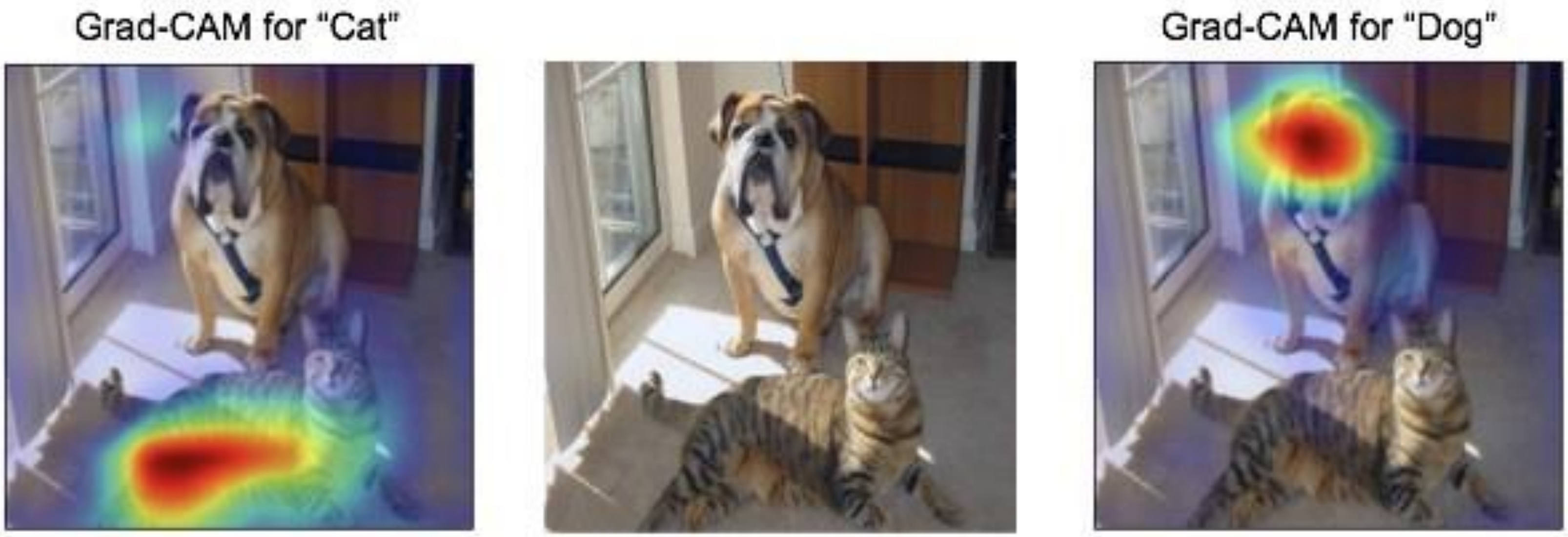
$$\alpha_c^k = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

Gradients of y^c w.r.t to filter k

- The heatmap weights activations by their importance

$$L^c = ReLU\left(\sum_k \alpha_c^k A^k\right)$$

GRAD-CAM Example (I)

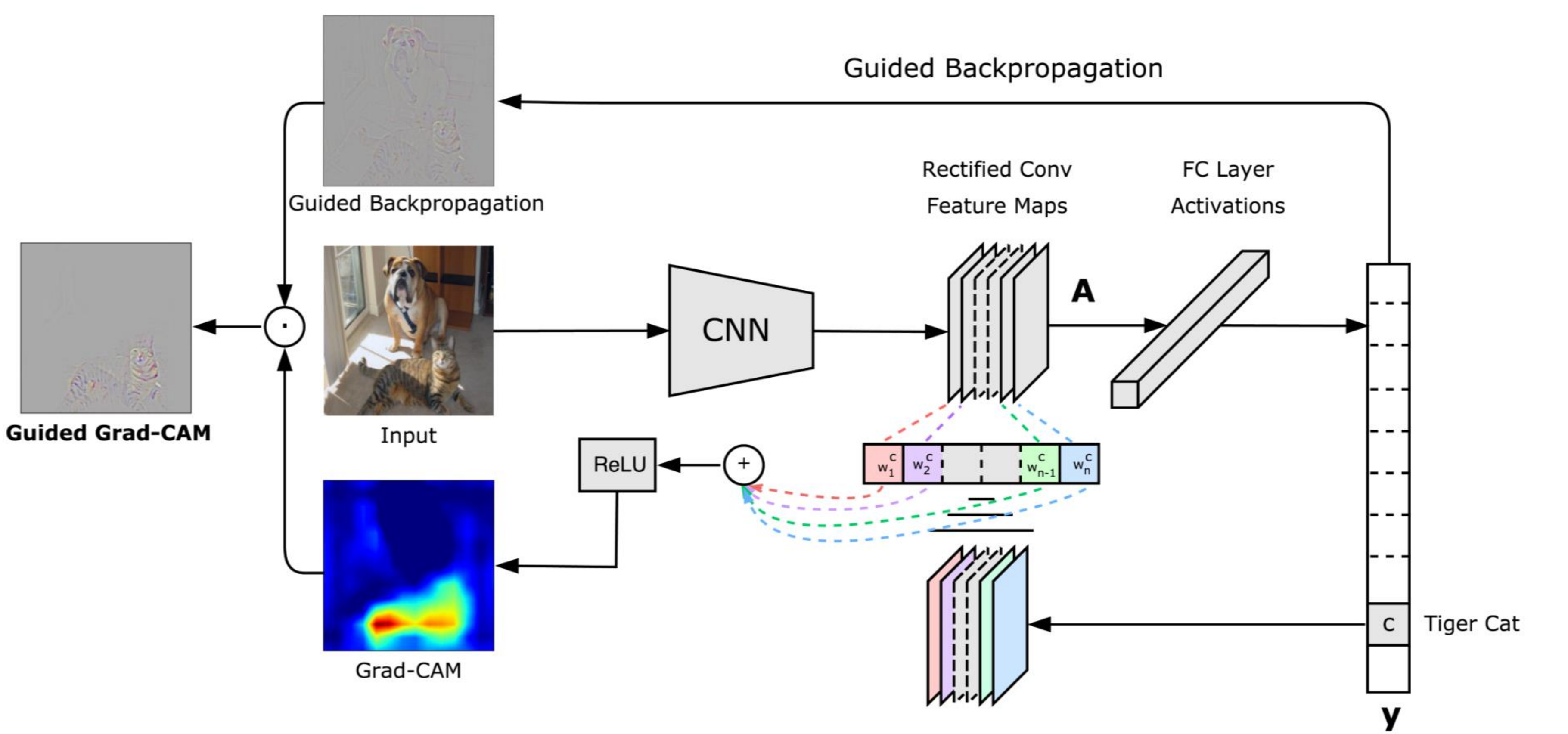


GRAD-CAM Example (II)

Visualization may identify biases in the trained model, then corrected by revising the training set

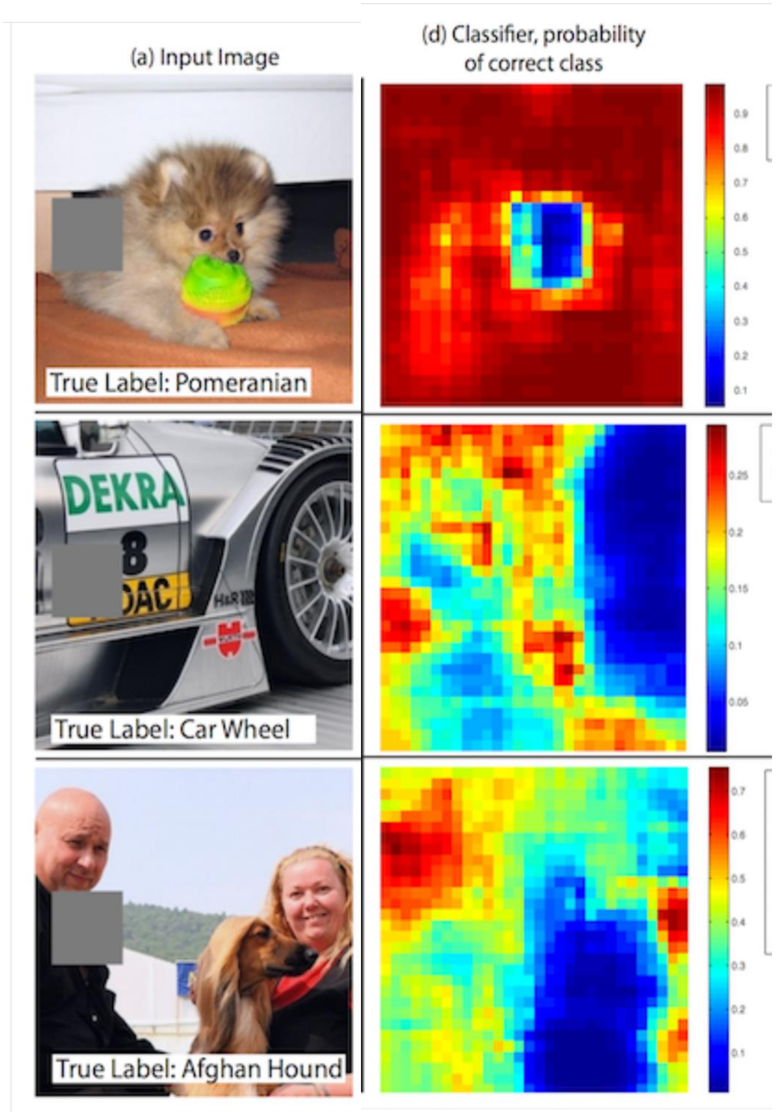
[Selvaraju, 2016, Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization]

Guided GRAD-CAM



Occlusion-based

- Modify the input image by masking patches (with varying patch size) and record the probability of correctly classifying the class
- When the prediction drops, the pixels are maximally relevant
- Simple to implement but computationally expensive



Summary

- While training:
  - Monitor the loss and metrics on training and validation set
  - Use Tensorboard to diagnose failure to convergence
- After training:
  - Use tSNE and filter visualization to get an overall intuition of what the model has learnt
  - Inspect instance-based explanations (e.g., GradCAM)
  - Do not focus only on cases that are classified correctly: deep neural networks are prone to taking shortcuts
  - Embedding visualization works well with any type of input
  - Some types of visualization are well suited to images, possibly text
- Check available notebooks →

Exercise

- For one of the networks trained in the previous labs (e.g., CNN trained on FashionMNIST):
  - Visualize the filters learnt using gradient ascent
  - Extract the embeddings and project them in 2 dimensions using t-SNE
  - Visualize explanations on 10 randomly selected cases using GRAD-CAM