

# Chapter 9: Advanced Deep Learning for Computer Vision

Matthew Kehoe

## 9.4: Interpreting what convnets learn

May 31, 2023

# Overview

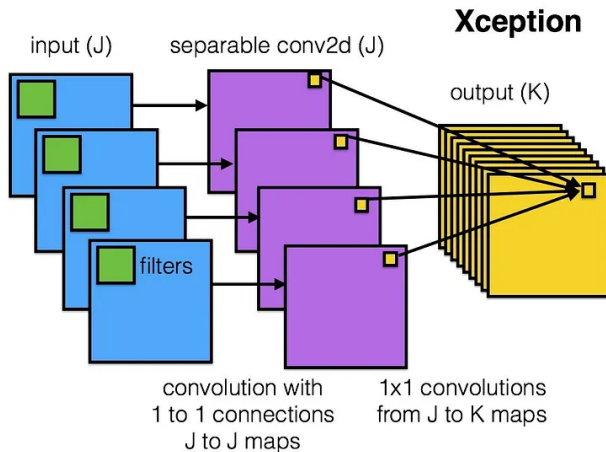
- 1 Introduction
- 2 9.4.1: Visualizing intermediate activations
- 3 9.4.2: Visualizing convnet filters
- 4 9.4.3: Visualizing heatmaps of class activation
- 5 Other Visualization Techniques (from [CS231](#))

# General Information

- 1 Google Research Machine Learning: [Link](#).
- 2 Fork the Github repository [here](#).
- 3 Using Github with Google Colab: [Link](#).
- 4 Course at Stanford: [Link](#).
- 5 Upgrade your memory on Google Colab: [Link](#). Doesn't work anymore.
- 6 **Convnet**: A convolutional neural network (CNN or convnet) is a subset of [machine learning](#). It is one of the various types of artificial [neural networks](#) which are used for different applications and data types. A CNN is a kind of network architecture for [deep learning](#) algorithms and is specifically used for [image recognition](#) and tasks that involve the processing of [pixel](#) data.

# The Xception Model

- Xception stands for the extreme version of Inception. With a modified depthwise separable convolution, it is even better than Inception-v3. It was developed at Google: [Summary](#) and [Paper](#).



# Problems with Computer Vision Applications

- A fundamental problem when building a computer vision application is that of **interpretability**: why did your classifier think a particular image contained a bike, when all you can see is a sedan?
- This is especially relevant to use cases where deep learning is used to **complement human expertise**, such as in **medical imaging** use cases.
- It's often said that deep learning models are “**black boxes**”: they learn representations that are difficult to extract and present in a human-readable form.
- Although this is partially true for certain types of deep learning models, it's definitely not true for convnets.

## Section 9.4 Summary

The representations learned by convnets are highly amenable to [visualization](#), in large part because they're representations of visual concepts. Techniques for conceptualizing these representations include

- 1 [Visualizing intermediate convnet outputs \(intermediate activations\)](#) — Useful for understanding how successive convnet layers transform their input, and for getting a first idea of individual convnet filters.
- 2 [Visualizing convnet filters](#) — Useful for understanding precisely what visual pattern or concept each filter in a convnet is receptive to.
- 3 [Visualizing heatmaps of class activation in an image](#) — Useful for understanding which parts of an image were identified as belonging to a given class, thus allowing you to localize objects in images.

# Visualizing intermediate activations

- Display the values returned by various convolution and pooling layers in a model, given a certain input (the **output of a layer** is often called its **activation**, the output of the activation function).
- This gives a view into how an input is decomposed into the different filters learned by the network.
- We want to visualize feature maps with **three dimensions**: width, height, and depth (channels). Each **channel** encodes relatively **independent features**, so the proper way to visualize these feature maps is by independently plotting the contents of every channel as a 2D image.

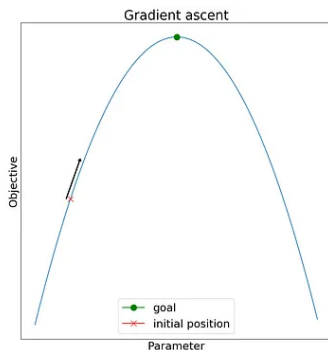
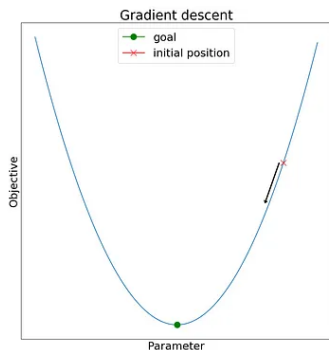
## Model: Cats-versus-dogs classification problem in §8.2

- We will start by loading the model we saw in Section 8.2.
- I will do this by expanding what François Chollet wrote in Google Colab.



# Gradient Ascent and Gradient Descent

- To find a local minimum of a function using **gradient descent**, one takes steps proportional to the **negative of the gradient** (or of the approximate gradient) of the function at the current point.
- If instead one takes steps proportional to the **positive of the gradient**, one approaches a local maximum of that function; the procedure is then known as **gradient ascent**.



## Other Visualization Techniques

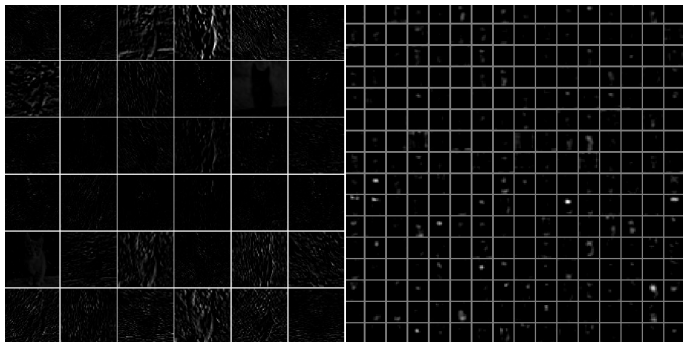
Several approaches for visualizing and understanding CNNs have been developed in the literature as a response to the [common criticism](#) that Neural Networks are [not interpretable](#). These include:

- Visualizing the activations and first-layer-weights
- Retrieving images that maximally activate a neuron
- Embedding the codes with t-SNE
- Occluding parts of the image
- Visualizing the data gradient and friends
- and much more

# Visualizing the activations

- **Layer activations:** one of the most straight-forward visualization techniques is to **show the activations** of the network during the **forward pass**.
- In **ReLU** networks, activations start out looking **blobby and dense**. As the training progresses, activations usually become more sparse and localized.
- With this visualization, it is easy to see where some **activation maps** are all **zero**, which can indicate the presence of **dead filters** and show **high learning rates**.

## Visualizing the activations (continued)

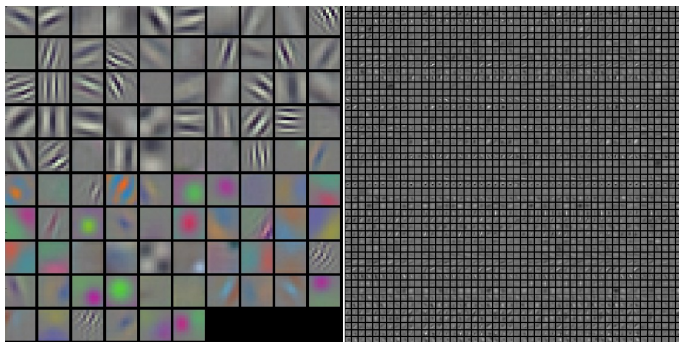


Typical-looking **activations** on the first CONV layer (left), and the 5th CONV layer (right) of a trained **AlexNet** looking at a picture of a **cat**. Every box shows an activation map corresponding to some filter. Notice that the **activations are sparse** (most values are zero, as shown in black) and are mostly **local**.

## Visualizing the first-layer weights

- **Conv/FC filters:** The second common strategy is to **visualize** the **weights**.
- These are usually **most interpretable** on the **first CONV layer** which is looking directly at the raw pixel data, but it is possible to also show the filter weights deeper in the network.
- The weights are useful to visualize because **well-trained networks** usually display **nice and smooth** filters without any noisy patterns.
- **Noisy patterns** can be an indicator of a network that hasn't been trained for **long enough**, or possibly a very **low regularization** strength that may have led to **overfitting**.

## Visualizing the first-layer weights (continued)

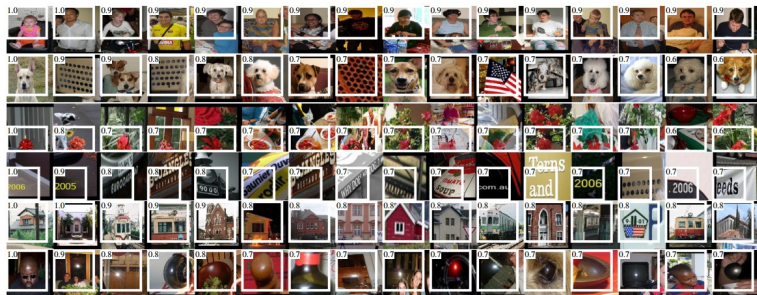


Typical-looking filters on the first CONV layer (left), and the 2nd CONV layer (right) of a trained [AlexNet](#). Notice that the [first-layer weights](#) are very [nice and smooth](#), indicating a nicely converged network. The 2nd CONV layer weights are not as interpretable, but it is apparent that they are still smooth, well-formed, and absent of noisy patterns.

## Retrieving images that maximally activate a neuron

- Another visualization technique is to take a **large dataset of images** and **feed them** into the **network** while keeping track of which images **maximally** activate a **neuron**.
- We can then visualize the images to get an **understanding** of what the **neuron** is looking for in its receptive field.
- This visualization technique (among others) is shown in **Rich feature hierarchies for accurate object detection and semantic segmentation** by Ross Girshick et al.

# Retrieving images that maximally activate a neuron (ctd)



Maximally activating images for some POOL5 (5th pool layer) neurons of an [AlexNet](#). The [activation values](#) and the receptive field of the particular neuron are shown in [white](#). (In particular, note that the POOL5 neurons are a function of a relatively large portion of the input image). It can be seen that some neurons are responsive to [upper bodies](#), [text](#), or [specular highlights](#).



## Retrieving images that maximally activate a neuron (ctd)

- One problem with this approach is that **ReLU neurons** do not necessarily have any **semantic meaning** by themselves.
- Rather, it is more appropriate to think of **multiple ReLU neurons** as the **basis vectors** of some space that represents in image patches. In other words, the visualization is **showing** the **patches** at the **edge of the cloud** of representations, along the (arbitrary) axes that correspond to the filter weights.
- Further analysis was performed by Szegedy et al. in **Intriguing properties of neural networks**, where they perform a similar visualization along arbitrary directions in the representation space.

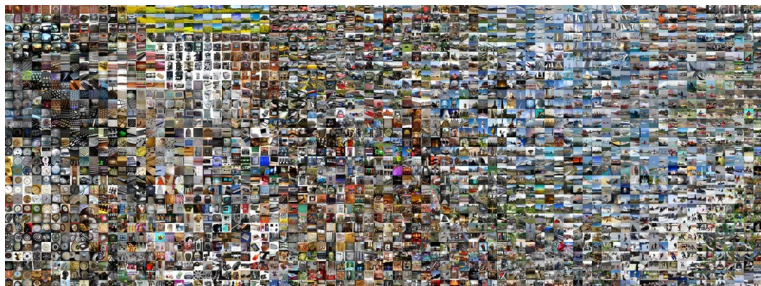
## Embedding the codes with t-SNE

- Convnets can be interpreted as **gradually transforming** the **images** into a representation in which the classes are separable by a **linear classifier**.
- We can get a rough idea about the topology of this space by **embedding images** into **two dimensions** so that their low-dimensional representation has approximately equal distance as the high-dimensional representation.
- There are many embedding methods which embed high-dimensional vectors in a low-dimensional space while **preserving** the **pairwise distance** between the points. One of the most popular methods is **t-Distributed Stochastic Neighbor Embedding (t-SNE)**.

# Embedding the codes with t-SNE

- To produce an embedding, we can take a **set of images** and use the **ConvNet** to **extract** the **CNN codes** (e.g. in AlexNet the 4096-dimensional vector right before the classifier, and crucially, including the ReLU non-linearity).
- We can then **plug** these into **t-SNE** and get **2-dimensional vector** for **each image**. The corresponding images can then be visualized in a grid.

## Embedding the codes with t-SNE (continued)

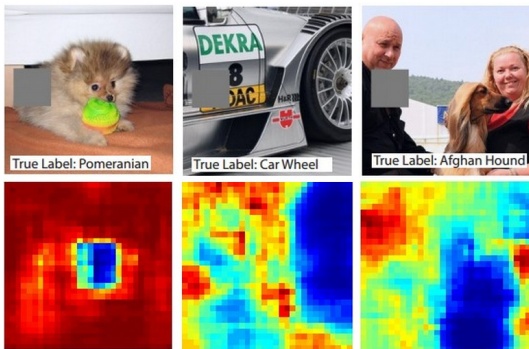


t-SNE [embedding](#) of a set of images based on their [CNN codes](#). Images that are [nearby](#) each other are also [close](#) in the [CNN representation space](#), which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based. For more details on how this visualization was produced see [the t-SNE visualization of CNN codes](#).

## Occluding parts of the image

- Suppose a ConvNet **classifies** an image as a **dog**. How can we be sure that it is **actually picking** a dog as oppose to some **contextual clues** from the **background** of the image?
- One way of investigating which part of the image the ConvNet predicts is by **plotting** the **probability** of the **class of interest** (e.g., dog class) as a function of the position of an occluder object.
- That is, we **iterate** over regions of the image, set a **patch** of the image to be all **zero**, and look at the **probability** of the class.
- We can then **visualize** the **probability** as a 2-dimensional **heat map**.

## Occluding parts of the image (continued)



Three input images (top). Notice that the **occluder region** is shown in **grey**. As we slide the occluder over the image we record the **probability** of the **correct class** and then visualize it as a heatmap (shown below each image). For instance, in the left-most image we see that the probability of Pomeranian plummets when the occluder covers the face of the dog.

# Visualizing the data gradient and friends

- Data Gradient: [Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.](#)
- DeconvNet: [Visualizing and Understanding Convolutional Networks.](#)
- Guided Backpropagation: [Striving for Simplicity: The All Convolutional Net.](#)

# Other Techniques

- Reconstructing original images based on CNN Codes: [Understanding Deep Image Representations by Inverting Them](#).
- How much spatial information is preserved? [Do ConvNets Learn Correspondence?](#) (tldr; yes)
- Plotting performance as a function of image attributes: [ImageNet Large Scale Visual Recognition Challenge](#).
- Fooling ConvNets: [Explaining and Harnessing Adversarial Examples](#).
- Comparing ConvNets to Human labelers: [What I learned from competing against a ConvNet on ImageNet](#).