Basic methods for Interpretation

We begin our study of Interpretability by presenting simple techniques.

Our discussion will be specialized to Neural Networks

- Consisting of multiple Convolutional Layers

The reason for this specialization is two-fold

- They are extremely common for task involving images (something that humans can easily interpret)
- The ability of a Convolutional Layer to preserve spatial dimensions
- Across Layers
- Means its easy to relate features at layer l back to the same spatial location in the input

Let's do a quick refresher on the important concepts and notation of Convolutional Layers.

CNN refresher (notation)

(We review concepts from the lecture on Convolutional Neural Networks (CNN)

A *feature map* for layer l

- Is the value of a *single* feature at layer l
- At *each* spatial location

An element of a feature map is the value of the feature at a single spatial location.

Here are the feature maps for two layers

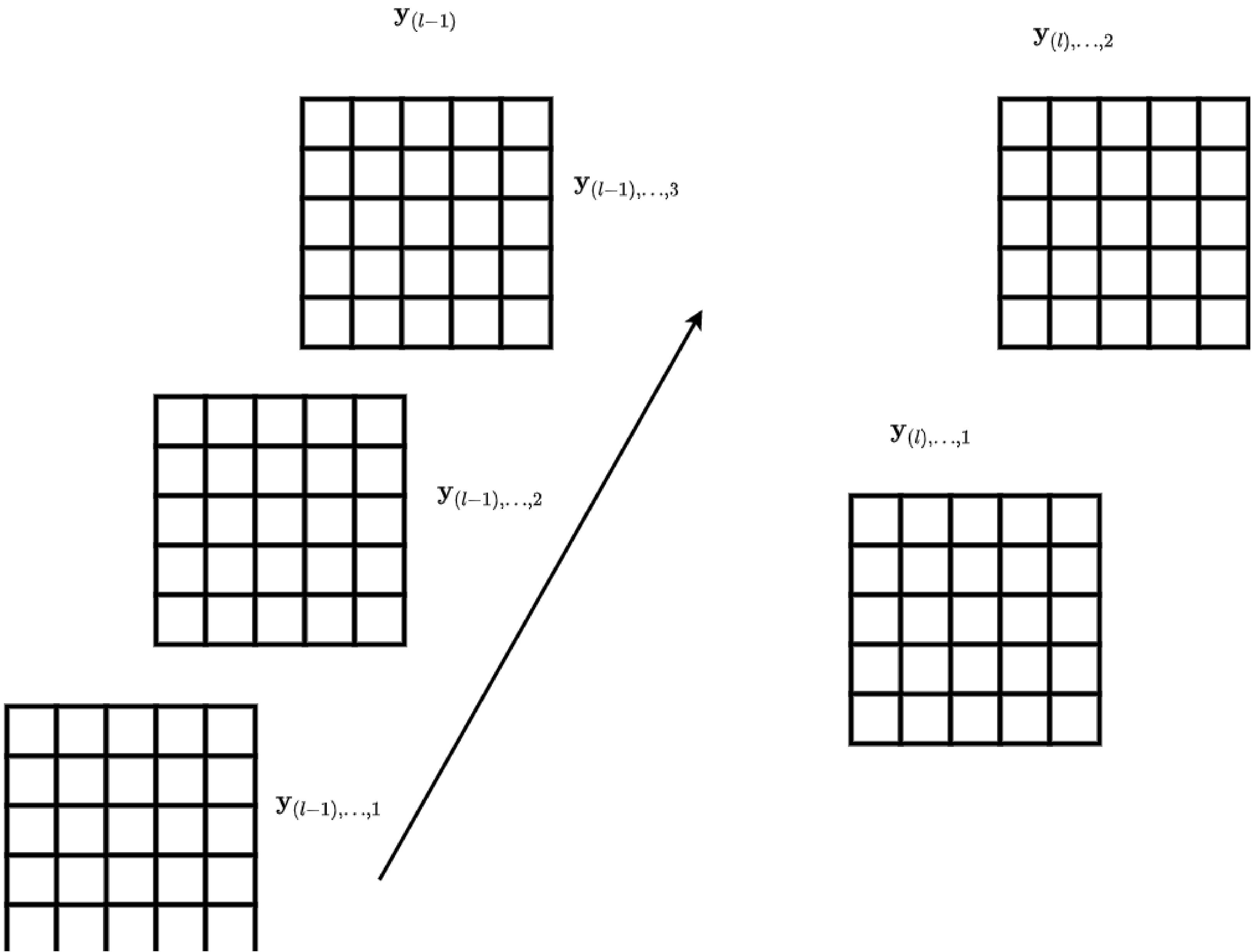
- Layer $(l-1)$ has three feature maps $\mathbf{y}_{(l-1),\ldots,k}$ for features $1 \leq k \leq 3$
- Layer (l) has two feature maps $\mathbf{y}_{(l),\ldots,k}$ for features $1 \leq k \leq 2$

Aside: Notation reminder

The feature/channel dimension

- Appears *last* in the subscripted list of indices (Channel Last convention)
- The ellipses $(\ldots)$ signify the variable number of *spatial* dimensions
- Thus feature k of layer l is denoted $\mathbf{y}_{(l),\ldots,k}$

Feature maps


$\mathbf{y}_{(l-1)}$
$\mathbf{y}_{(l),\ldots,2}$
$\mathbf{y}_{(l-1),\ldots,3}$
$\mathbf{y}_{(l-1),\ldots,2}$
$\mathbf{y}_{(l),\ldots,1}$
$\mathbf{y}_{(l-1),\ldots,1}$

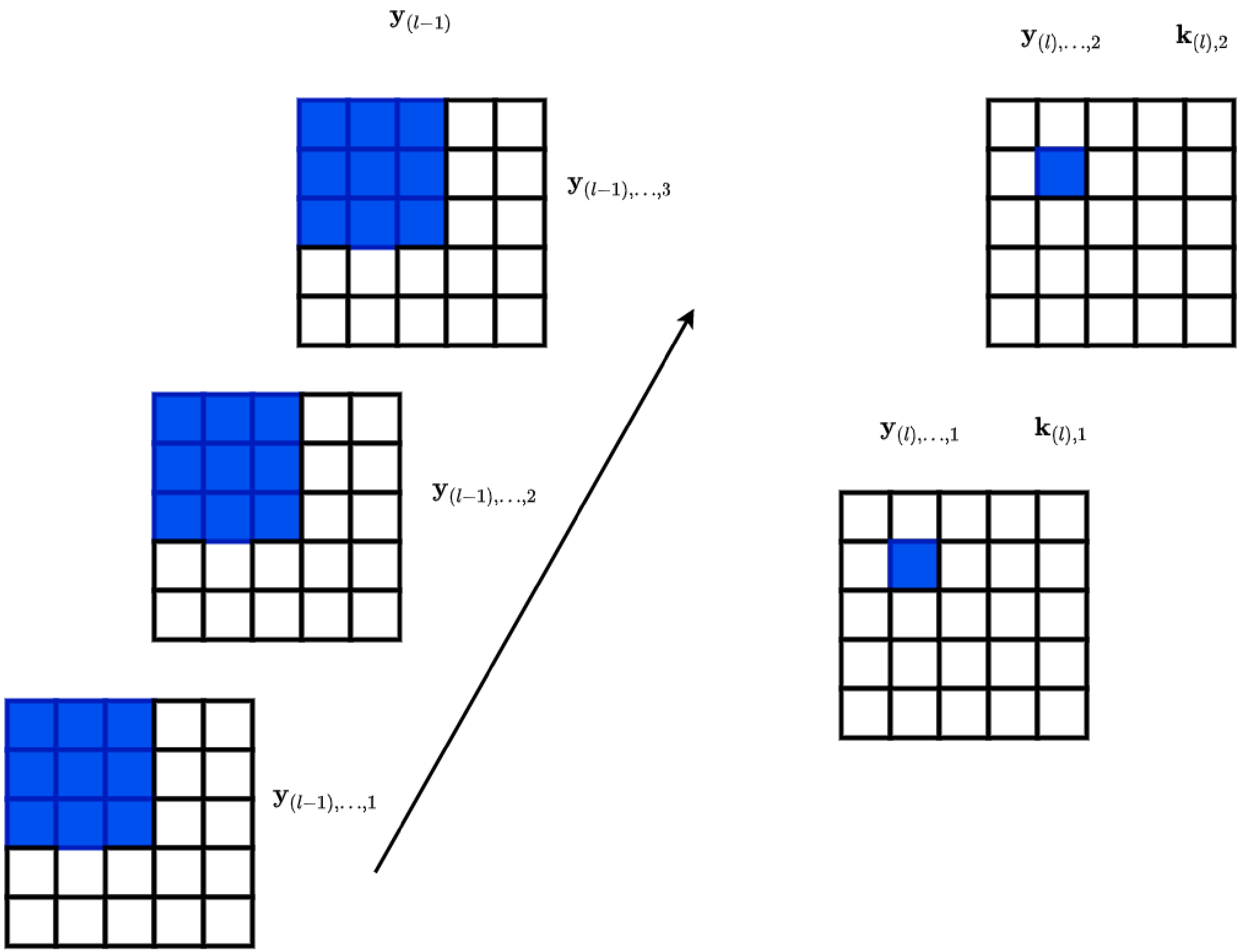
Each feature map k of layer l

- Was created by applying a $(f_{(l)} \times f_{(l)} \times n_{(l-1)})$ convolutional kernel $\mathbf{k}_{(l),k}$
- To layer $(l-1)$ output $\mathbf{y}_{(l-1)}$

We "slide the kernel" over all spatial locations of $\mathbf{y}_{(l-1)}$

- The Convolutional Layer l
- Preserves the spatial dimension
- But changes the number of features from $n_{(l-1)}$ to $n_{(l)}$

Two layer l feature maps, same spatial location but different output features


$\mathbf{y}_{(l-1)}$
$\mathbf{y}_{(l-1),\ldots,3}$
$\mathbf{y}_{(l-1),\ldots,2}$
$\mathbf{y}_{(l-1),\ldots,1}$
$\mathbf{y}_{(l),\ldots,2}$
$\mathbf{k}_{(l),2}$
$\mathbf{y}_{(l),\ldots,1}$
$\mathbf{k}_{(l),1}$

Since a Convolutional layer l

- Preserves the spatial dimension of its input (layer $(l-1)$ output
- Assuming full padding
- We can directly relate the spatial location of each feature map
- To a spatial location of layer 0, the input

The question we seek to answer:

- Can we describe (interpret) the feature being recognized in a single feature map of layer l ?

Much of our presentation is based on a very influential paper by [Zeiler and Fergus (https://arxiv.org/abs/1311.2901)](https://arxiv.org/abs/1311.2901)

- NYU PhD candidate and advisor !

Interpretation: The first layer

It is relatively easy to understand the features created by the first layer

Since feature map k is the result of a dot-product (convolution)

- And the dot product is performing a pattern match
- Of the pattern given by kernel $\mathbf{k}_{(1),k}$
- Against a region of the input
- We can interpret layer 1 as trying to create synthetic features identified by the pattern

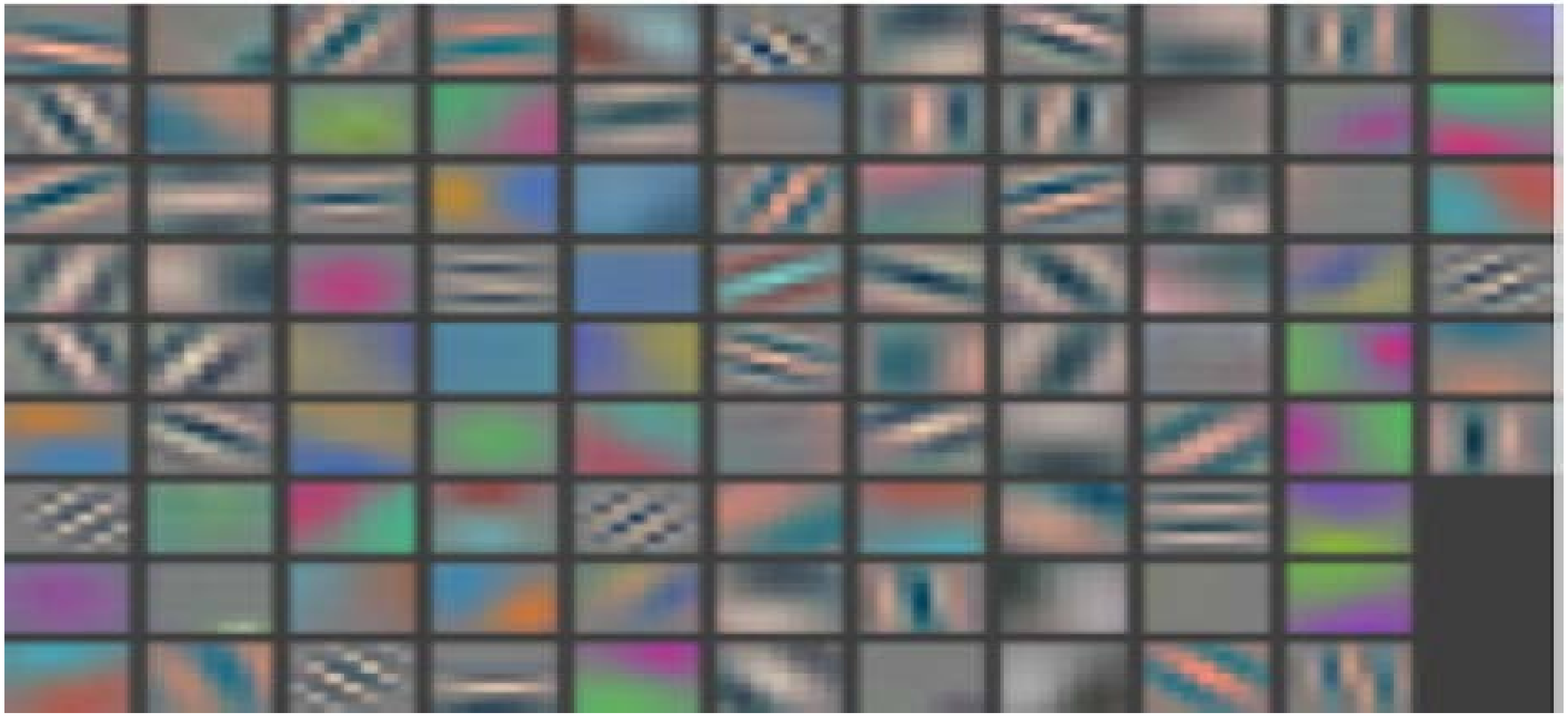
So all we have to do is examine each kernel to see the pattern for feature k !

Here is a visualization of the kernels from the Zeiler and Fergus paper

- For 96 individual features
- Being computed by the first layer (1)
- Using a $(7 \times 7 \times n_{(0)})$ kernel
    - $n_{(0)}$ are the number of input channels

Each square is a kernel, whose spatial dimensions are (7×7) and depth $n_{(0)} = 3$

Layer 1 kernels



The "patterns" being recognized by these kernels seem to represent

- Lines, in various orientations
- Colors
- Shading

We interpret Layer 1 as trying to construct synthetic features representing these simple concepts.

So feature map k of layer 1 can be interpreted as

- Identifying the presence/absence of pattern $\mathbf{k}_{(1),k}$ in input $\mathbf{x}$
- At each spatial location of the input

Layer 1 Kernel example From Figure 2

There are kernels looking for "checkered" patterns

- At row 7, columns 1 and 5

Note that examining layer 1 kernels

- Is *input independent*
- Does not depend on the value of any example $\mathbf{x}^{(\mathbf{i})}$

Beyond the first layer: Clustering examples

We could try to interpret the kernels of layer $(l > 1)$ but this will be difficult

- Layer l's inputs ($\mathbf{y}_{(l-1)}$) are *synthetic features*, rather than actual inputs
- Unless we understand the synthetic features of the earlier layers
- We won't be able to interpret the pattern that layer l is matching

What we can hope to do

- Somehow map the representation created by layer $(l > 1)$ back to the inputs (layer 0 output)

We will present several *input dependent* methods

- Depend on a particular input example $\mathbf{x^{(i)}}$

So the interpretation is only as good as the set of input examples we use.

The methods will find *clusters* of examples

- That produce a similar feature map
- For map k at layer l

If we can identify a property that is common to all examples in the cluster

- We can interpret feature map k of layer l as implementing the feature

"Is the property present in the input ?"

Maximally Activating Examples

Recall that the *feature map* k of layer l

- Is matching a pattern (the kernel for k)
- At each index idx in the spatial dimension
- Thus, $\mathbf{y}^{(\mathbf{i})}_{(l),\text{idx},k}$ is the intensity of the feature being present at spatial location idx of input $\mathbf{x}^{(\mathbf{i})}$

The problem: there are lots of locations in the spatial dimensions.

Rather than examining all locations, we we can *summarize* whether the feature exists *anywhere* in example i.

For example, using "max" for summarization

- We can identify the *value* $\text{max}_{(l),k}^{(\mathbf{i})}$ of the strongest activation
- Without identifying its exact location

$$\text{max}_{(l),k}^{(\mathbf{i})} = \max_{\text{idx}} \mathbf{y}_{(l),\text{idx},k}^{(\mathbf{i})}$$

By sorting examples on $\max^{(\mathbf{i})}_{(l),k}$

- We identify a cluster of examples
- That are most identified with the feature

These examples with largest $\max^{(\mathbf{i})}_{(l),k}$ are the *Maximally Activating Examples* for feature k of layer l.

If we can identify a common property among the examples with largest $\mathbf{max}_{(l),k}^{(\mathbf{i})}$

- We can interpret feature map k of layer l as implementing the feature

"Is the property present in the input ?"

Formally

- Let $\mathrm{MaxAct}_{(l),k} = [i_1, \ldots, i_m]$ be the permutation of example indices, i.e., $[i|1 \leq i \leq m]$
- That sorts $\max_{(l),k}^{\mathbf{(i)}}$ in ascending order

$$\max_{(l),k}^{(i_1)} \geq \max_{(l),k}^{(i_2)} \geq \ldots \geq \max_{(l),k}^{(i_m)}\$$$

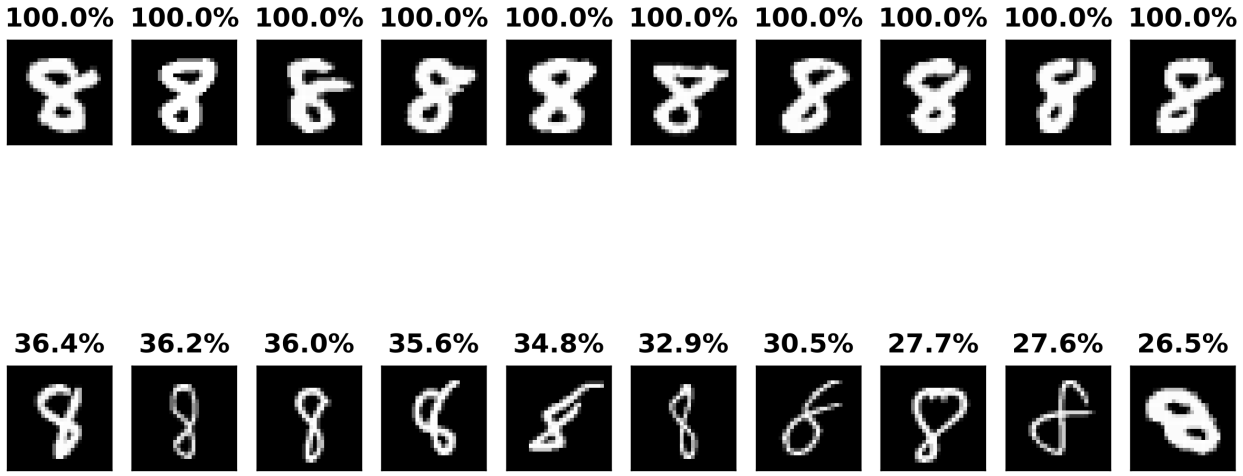
In this way we can try to interpret the feature map of each layer.

Applying this technique to layer L (the "head", a Classifier in the case of MNIST) is particularly useful

- We can identify the examples most/least strongly identified with the concept of each digit
- Because $\mathbf{y}_{(L),k}^{\mathbf{(i)}}$ is the *probability* that example i is digit $k \in \{0, \ldots, 9\}$

Here are examples of the digit "8" that maximally/minimally activate the classifier's "8" output $\mathbf{y}_{(L),8}$

MNIST CNN maximally activating 8's



Interesting ! Do we have a problem with certain 8's ?

Much lower probability when

- 8 is thin versus thick
- tilted left versus right

So although our goal was interpretation, this technique may be useful for Error Analysis as well.

Occlusion

Maximally activating inputs are very coarse: they identify concepts at the level of entire input.

But, it's reasonable to suspect that some elements of the input are more important to the concept than others.

In particular, a CNN has a "receptive field" which defines the input elements that contribute to the layer output.

Close to the input layer, the receptive field is narrow so its clear that the "features" being identified are small in span.

Occlusion is one way of identifying the elements of the input layer that most affect the latent representation.

We will describe this in terms of a 2D input, but we can generalize.

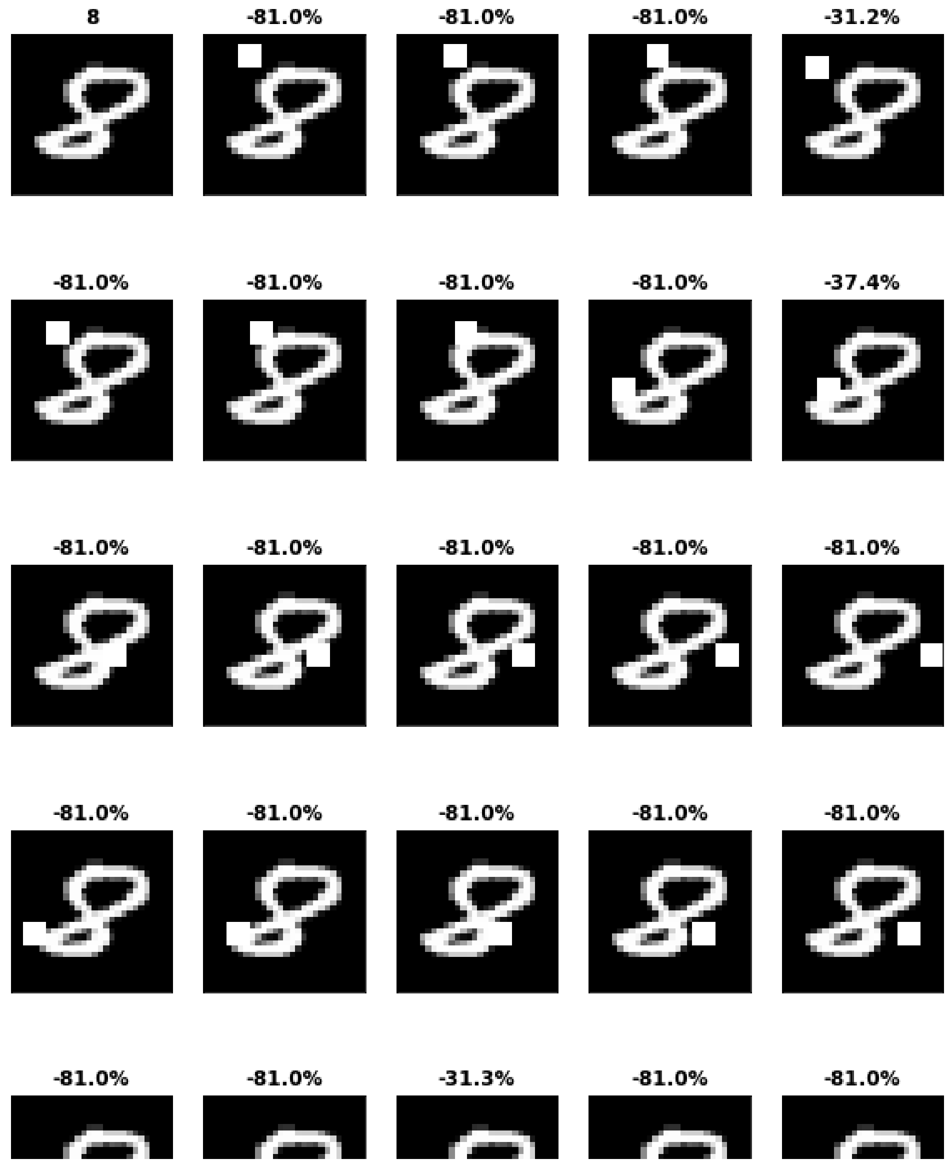
Let

- $\mathbf{y}^{(\mathbf{i})}_{(l),j}$ denote the response of feature $\mathbf{y}_{(l),j}$ to input $\mathbf{x}^{(\mathbf{i})}$.
- Place an occluding square over some portion of input $\mathbf{x}^{(\mathbf{i})}$ and measure the change in $\mathbf{y}_{(l),j}$
- Do this for each location in input $\mathbf{x}^{(\mathbf{i})}$ and create a "heat map" of changes in response $\mathbf{y}_{(l),j}$

Let's use occlusion to see how images of the digit "8" are recognized

- Perhaps: by the two "donut" holes and "pinched waist" ?

Occlusion: Relative decrease in probability of being "8"


8
-81.0%
-81.0%
-81.0%
-31.2%
-81.0%
-81.0%
-81.0%
-81.0%
-37.4%
-81.0%
-81.0%
-81.0%
-81.0%
-81.0%
-81.0%
-81.0%
-81.0%
-81.0%
-81.0%
-81.0%
-81.0%
-31.3%
-81.0%
-81.0%

Not what we expected !

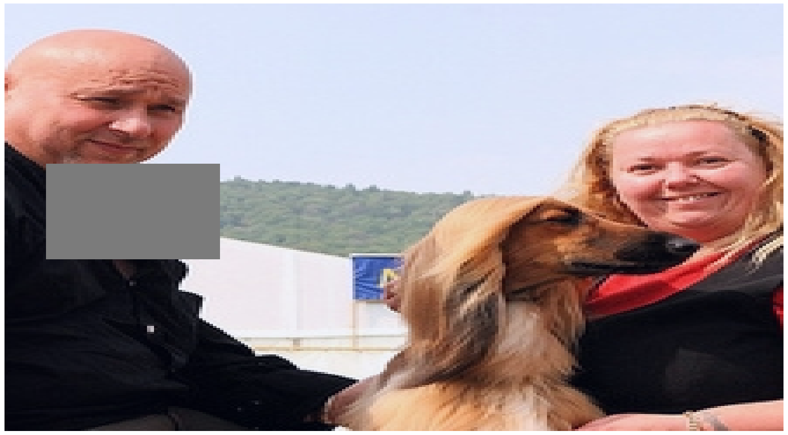
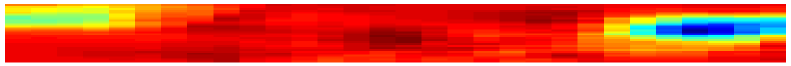
The mere presence of the square changes the classification probability greatly, even when we are not blocking the "waist" of the 8.

Here is the change in response of a single feature map in layer 5 of an image classifier (Zeiler and Fergus).

The chosen feature map is the one with the highest activation level in the layer.

You can see that it is responding to "faces".

- "Cold" colors: drop in intensity of activation

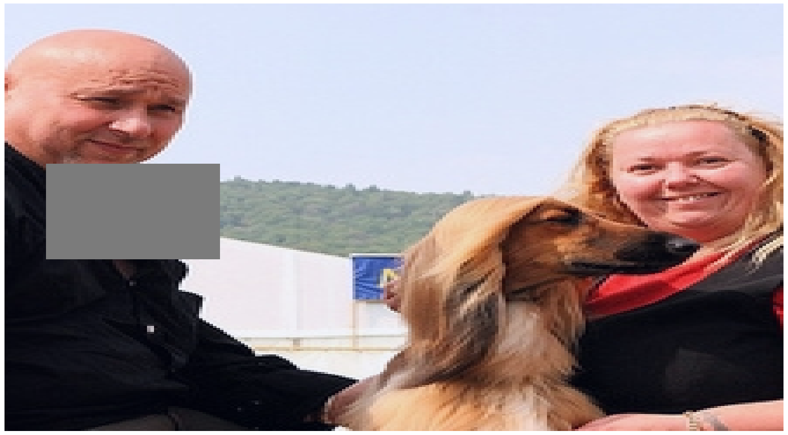
Input image	Activation of one filter at layer 5
	

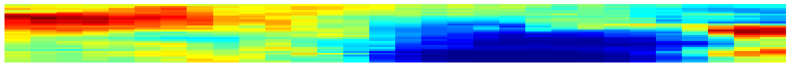
Zeiler and Fergus also measured the change in activation of $\mathbf{y}_{(L),j}^{(\mathbf{i})}$, the logit corresponding to the correct class ("Afghan Hound").

- "Hot" colors: increase in intensity of the "Afghan Hound" logit

Input image

Change in logit for "Afghan hound"





Conclusion

We began our quest for understanding how Neural Networks work with simple techniques.

The first technique

- Find clusters of example
- Created by a particular feature map
- Relate a human-observable common property of the cluster
- To the feature that the feature map is attempting to recognize

Whereas clustering identifies groups of examples, the second technique tries to find *sub-regions* of the examples

Occlusion measures the change in response of a feature map summary (or single neuron)

- When a sub-region of the input is visible
- Versus when it is not visible

The interpretation that arises is that the feature map is attempting to recognize a property in a narrow area.

So, beyond clustering, it is attempting to *localize* the spatial location of the feature.

```
In [4]: print("Done")
```

```
Done
```