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

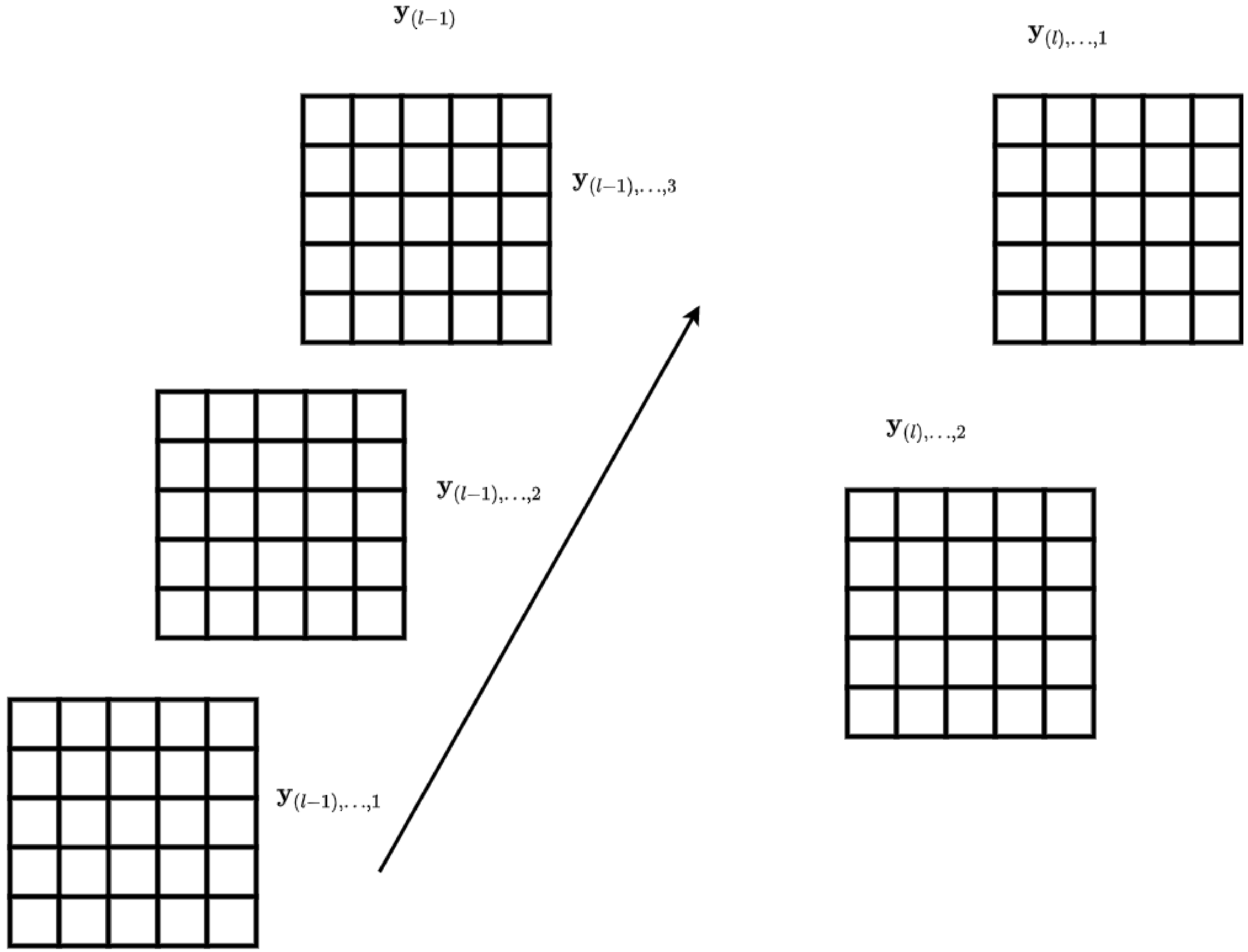
- Layer $(l-1)$ has three feature maps
  $$\mathbf{y}_{(l-1),\ldots,k} \text{ for features } 1 \le k \le 3$$
- Layer (l) has two feature maps
  $$\mathbf{y}_{(l),\ldots,k} \text{ for features } 1 \le k \le 2$$
  is the feature map for $\mathbf{y}_{(l),1}$, feature number 1 of layer l

Aside: Notation reminder

The feature/channel dimension

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

Feature maps


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

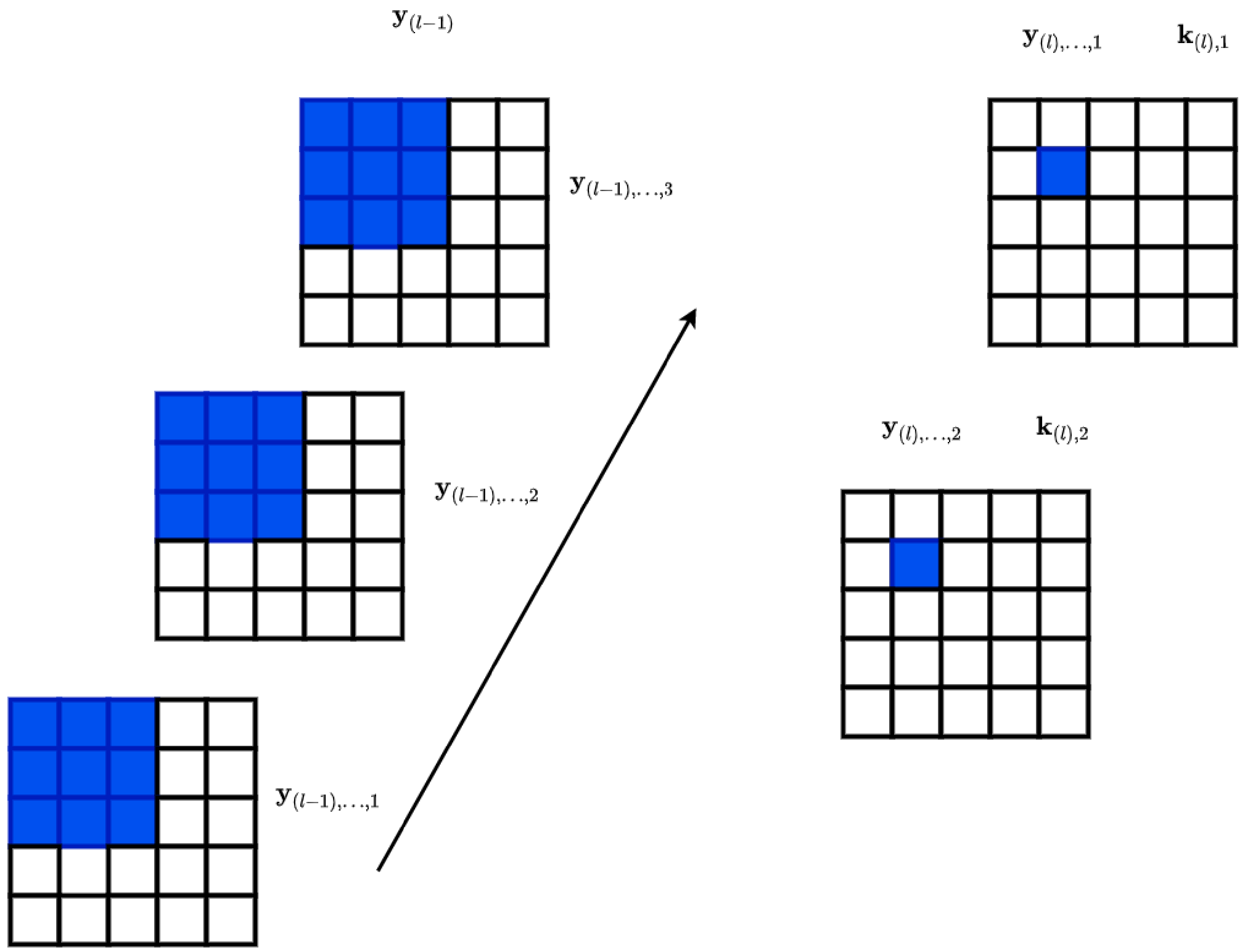
Each feature map k of layer l

- 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 $n_{(l)}$

Feature maps produced by Convolutional Layer l


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

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 presentaton is based on a very influential paper by Zeiler and Fergus (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

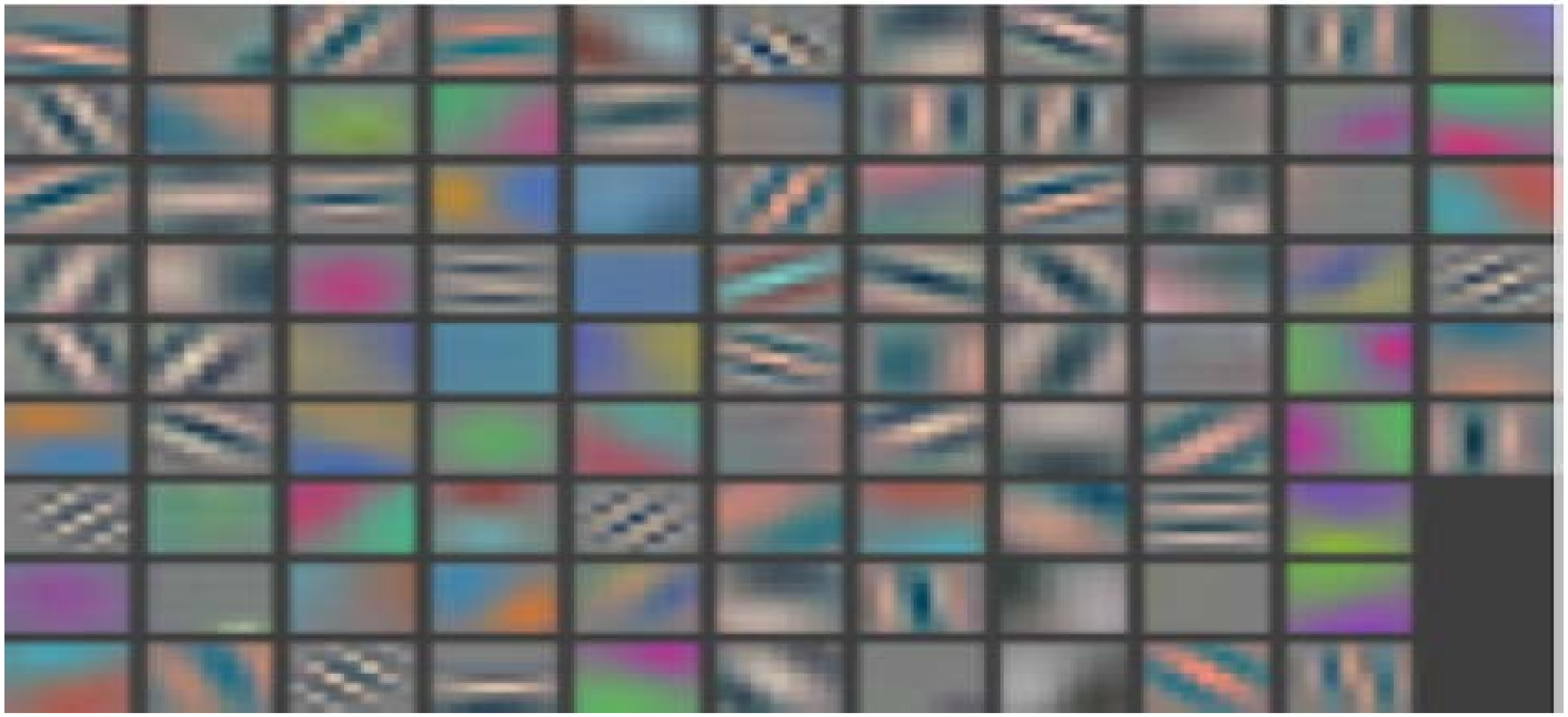
So all we have to do is examine each kernel to see the pattern for feature k !

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

- For 96 individual features
- Being computed by layer 1
- Using a $(7 \times 7 \times n_{(0)})$ kernel

Each square is a kernel, whose spatial dimensions are (7×7).

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 interpretted 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 methods that

- Are *input dependent*
- Are conditional on the value of a particular input example $\mathbf{x^{(i)}}$

The method will be to find *clusters* of examples

- That produce similar feature maps
- 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 ?"

```
In [ ]: The first problem is that a feature map is big!
        - One element for each spatial location
        - Makes it hard to compare feature maps for similarity
```

PCA of Feature Maps

It is hard to find clusters when objects are of high dimension

- With so many dimensions
- Any distance measurement tends to be large even for similar objects
- Because the number of *irrelevant* elements
- May be larger than the number of relevant elements

Consider a feature map $\mathbf{y}_{(l),\ldots,k}$ with spatial dimension (1000×1000)

- A typical image size
- Two examples have a dog in the center
- Surrounded by much different backgrounds

If the number of spatial locations in the background is large than the region containing the dog

- Then these two similar examples
- Have large distance
- Due to the different, but irrelevant, backgrounds

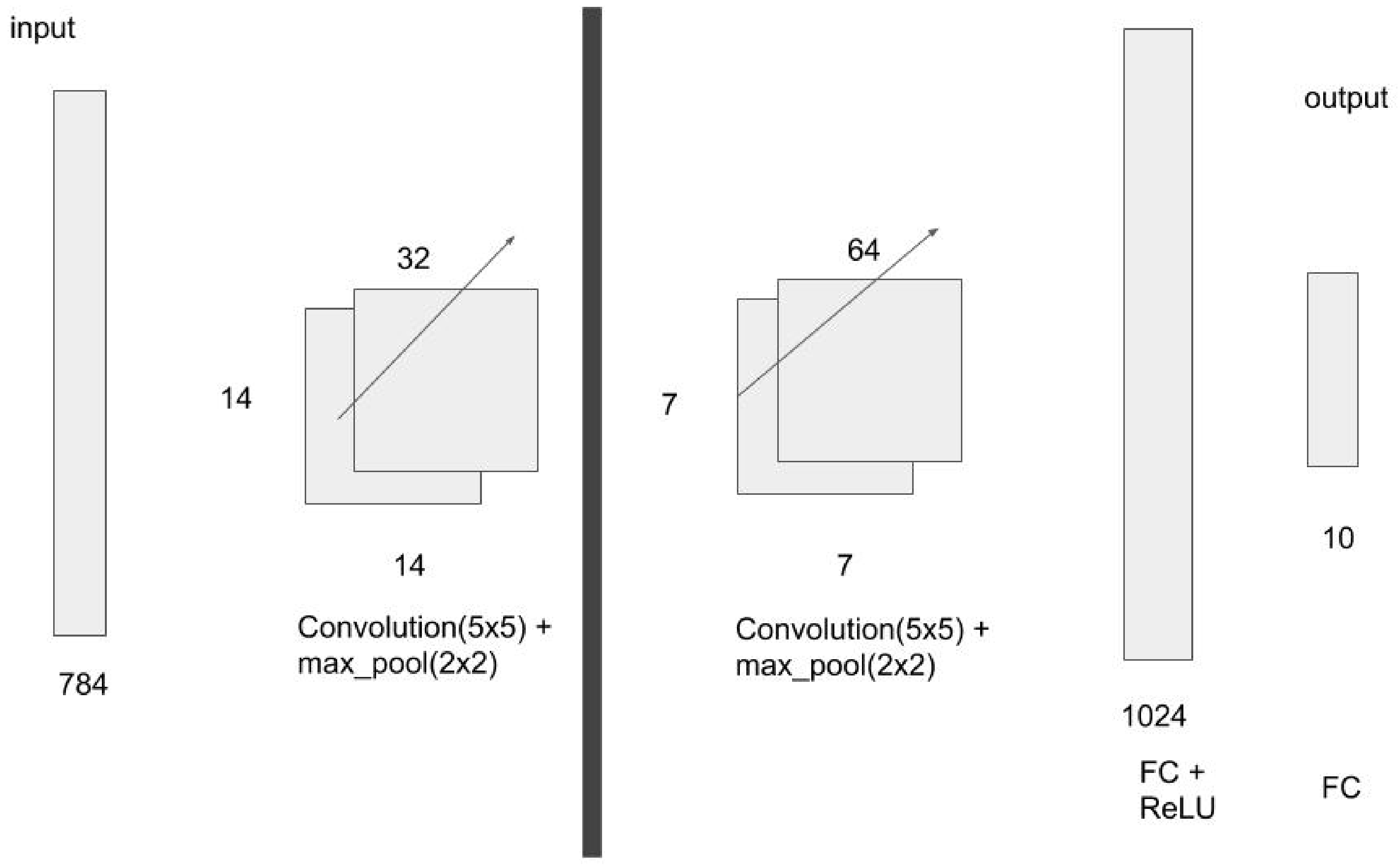
```
In [ ]: We can use *dimensionality* reduction techniques of Classical Machine Learning.

        One such technique is Principal Components Analysis
        - Find a small number of synthetic features
        - That express commonalities of many examples
        - Represent an example in a synthetic feature space
        - Of reduced dimensions

        In this case: we are reducing the number of spatial locations
```

Here is a two layer Neural Network that we built to classify digits in the MNIST dataset

MNIST CNN

PCA

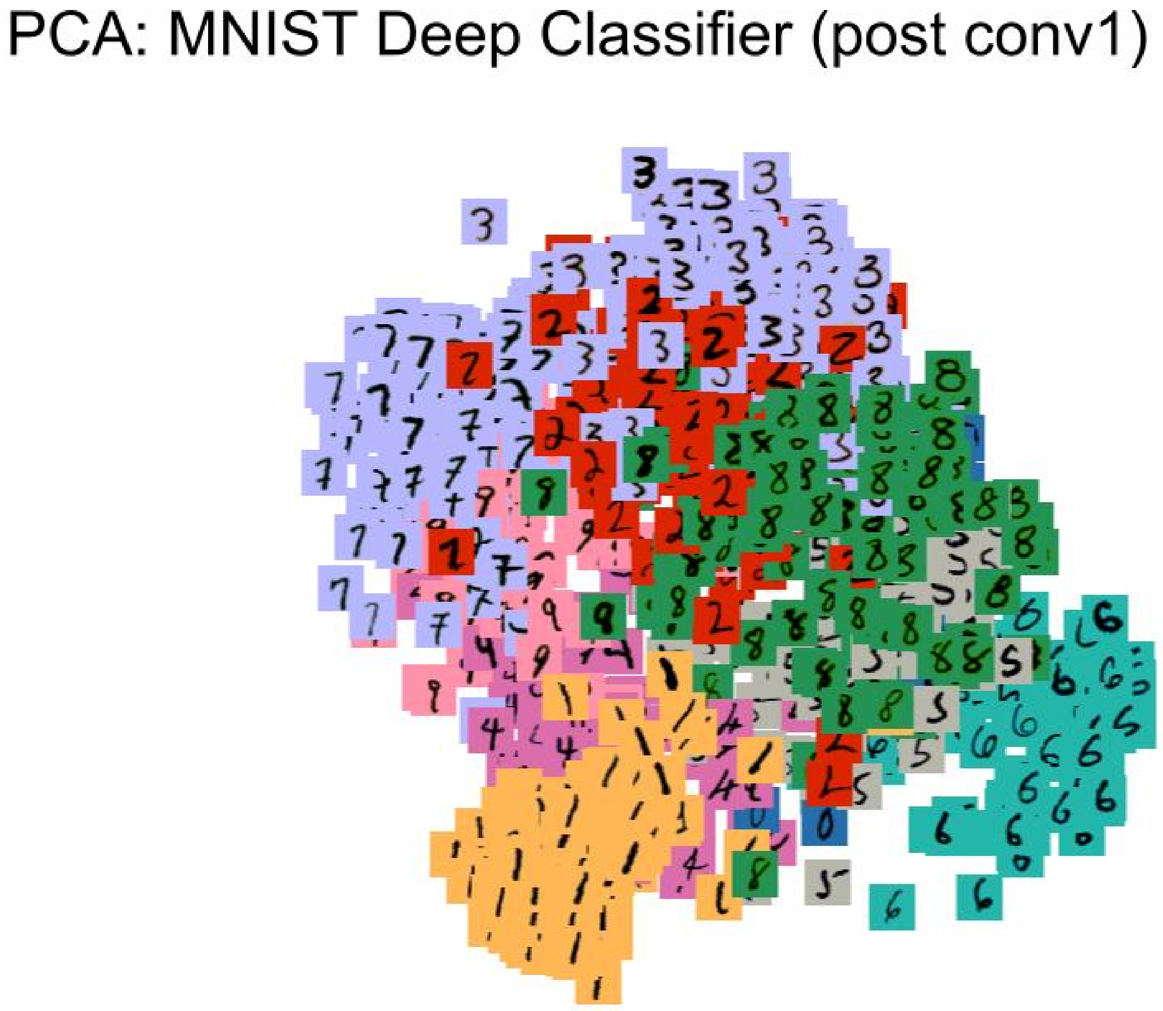
- Interpret the intermediate representation (what is the transformed feature ?)



We perform PCA on the representations produced by the first Convolutional Layer (dark vertical line)

- Plotting each example
- Using the two most important synthetic features (components) as coordinates in the plot

MNIST CNN Conv1 PCA



Clusters are starting to appear.

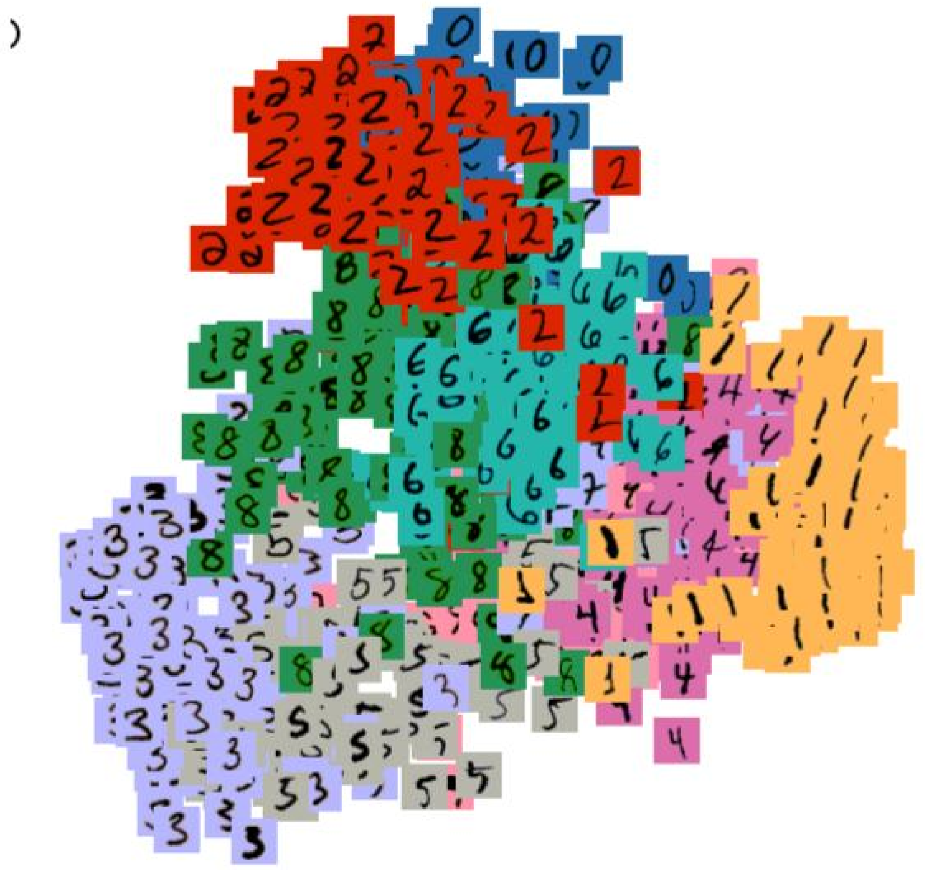
Do these clusters give us a clue as to the property that the layer is representing ?

- Left to right: strong vertical ("1", "7") to less vertical ?
- Bottom to top: digits *without* "curved tops" to those with tops ?

Let's perform the same analysis on the representations of the second Convolutional layer

MNIST CNN Conv1 PCA

PCA: MNIST Deep Classifier (Post conv2)



The clusters become "more pure".

So the deeper representation

- May be finding *combinations* of input features
- That cluster similar digits

So we might be able to intepret what the first two Convolutional Layers are representing

- Without necessarily understanding what the second layer is doing in isolation

Maximally Activating Examples

The goal remains

- Find clusters of examples
- That produce a similar feature map k at layer l

Our first attempt was to reduce the large spatial dimension of $\mathbf{y}_{(l)}, \ldots, k$ to something smaller.

We now try a more extreme approach.

The first difficulty we encounter for a given feature map k of layer l

- There are many spatial locations

We can reduce this complexity

- By *summarizing* feature map k with a single number
- For example: max or average (like MaxPooling layer)

This makes it easy to find examples with similar feature maps

- They have similar summary values

The method known as *Maximally Activating Examples*

- Finds the examples in a set $\mathbf{X}$
- With the *largest* summary values
- Recognizing "strong" features (max)

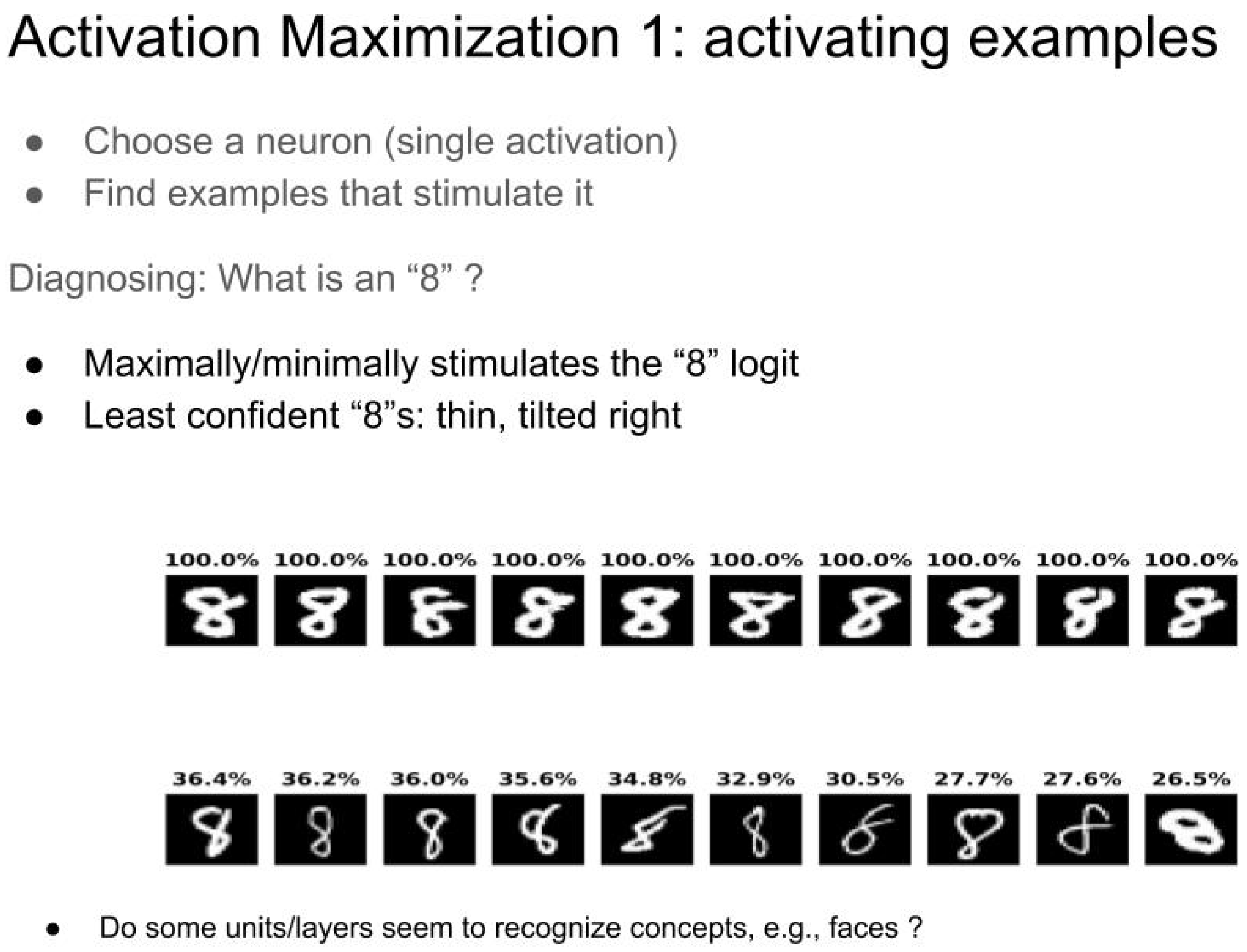
The method works as follows:

- For each image in a set $\mathbf{X}$
- Compute the summary value $s^{(\mathbf{i})}$ conditional on input $\mathbf{x}^{(\mathbf{i})}$
- Consider all the subset of $\mathbf{X}$ with the *largest absolute value* of $s^{(\mathbf{i})}$

Here's a way to use Maximally Activating Examples to improve your Neural Network

- We find the examples that give the strongest response
- To the *single neuron* in Layer L (the Classifier)
- That is responsible for producing the output "Example is an 8"
    - i.e., the logit that computes the score for the binary classifier "Is 8"

MNIST CNN maximally activating 8's

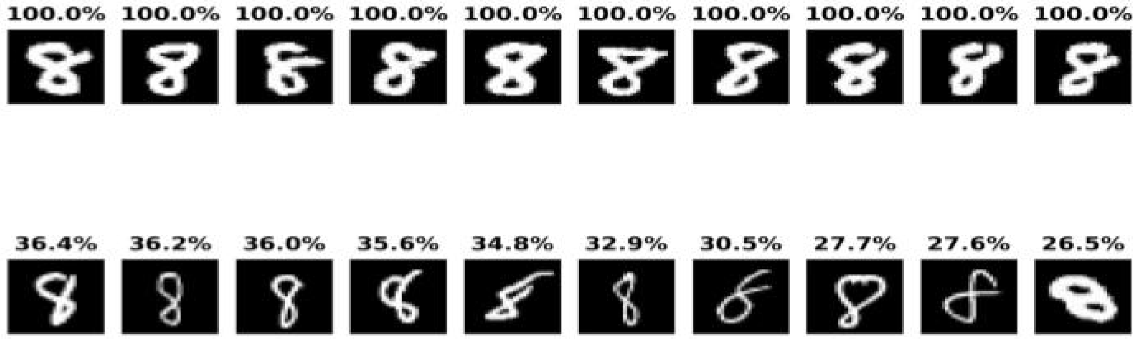


Activation Maximization 1: activating examples

- Choose a neuron (single activation)
- Find examples that stimulate it

Diagnosing: What is an “8” ?

- Maximally/minimally stimulates the “8” logit
- Least confident “8”s: thin, tilted right



- Do some units/layers seem to recognize concepts, e.g., faces ?

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

Much lower probability when

- 8 is thin versus thick
- tilted left versus right

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 particluar, 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

- $\mathbf{y}_{(l),j}^{(\mathbf{i})}$ denote the response of feature $\mathbf{y}_{(l),j}$ to input $\mathbf{x}^{(\mathbf{i})}$.
- Place an occulding 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 $\mathbf{y}_{(l),j}$

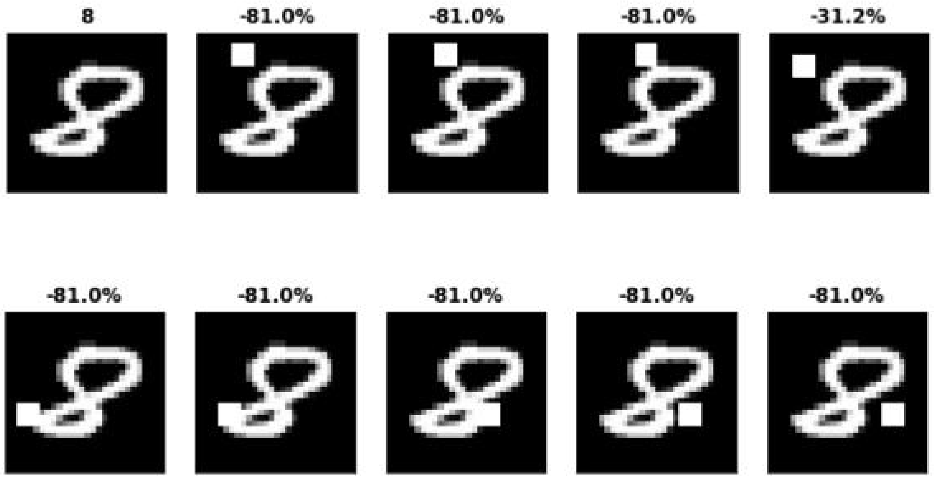
The number on top is the percent decrease in $\mathbf{y}_{(L),j}$, the logit for digit 8.

Occluding 8

Occlusion

Why am I a polar bear ?

- The snow ? The fur ? The face ?
  - Activation maximization: which images
  - Occlusion: which parts of an image



- Diagnostic: Is this a problem ?; Is it a problem only with “8” ?
  - Can get all 8’s wrong and still be high overall accuracy

Not what we expected !

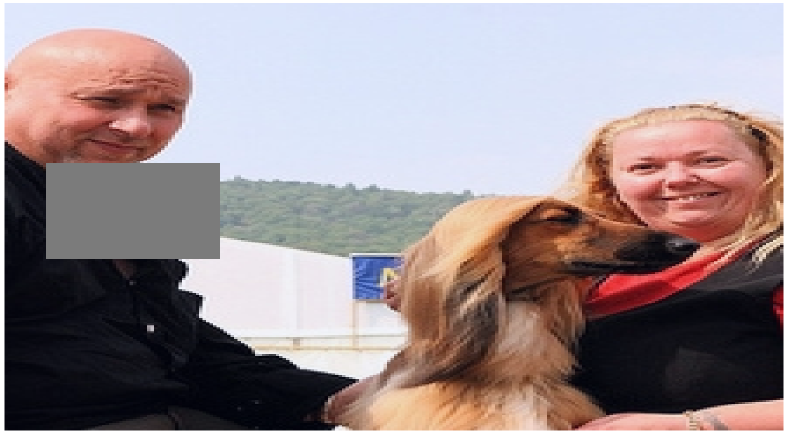
The mere presence of the square changes the classification probability greatly, even when we are not blocking the "waist" of the 8.

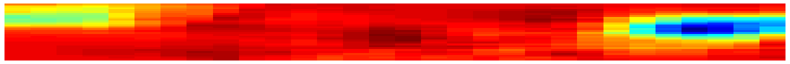
Here is the change in response of a single feature map in layer 5 of an image classifier (Zeiler and Fergus).

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

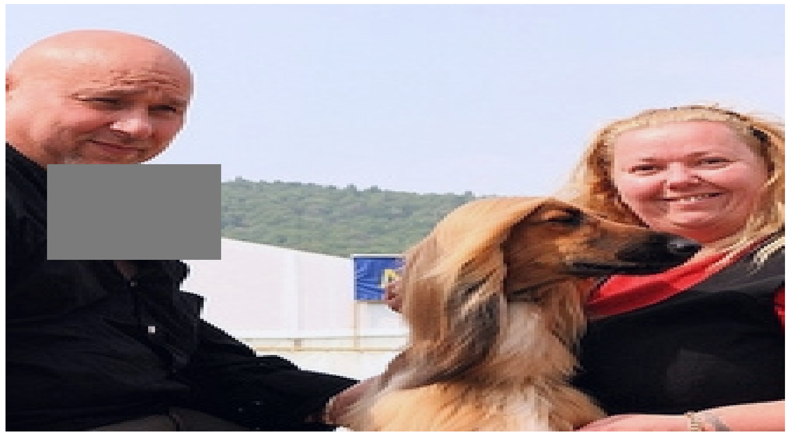
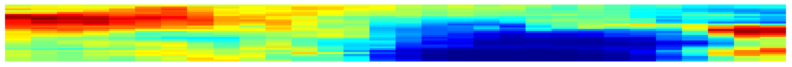
You can see that it is responding to "faces".

Input image **Activation of one filter at layer 5**





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

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
```