Overview

We have seen that Neural Networks are capable of performing many tasks very well.

One surprising aspect of this

- we have not *directed* the Neural Network on how to achieve the task
- the task is achieved by minimization of a Loss Function

We have seen that it the *potential* to be a Universal Function approximatator

- implementing the function defined implicitly
- by the empirical distribution of input/output pairs
- represented by the labeled training dataset

But is there any way to gain insight into what is happening within the layers of a Neural Network ?

That is

- given the many synthetic features created by the Neural Network
- can we discover/interpret what is the meaning of a particular feature ?

That is the topic of this module.

Interpretation: The first layer

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

- they involve the dot product of an input and some weights
- matches inputs $\mathbf{x}$ against weights (pattern) $\mathbf{w}$

So we can understand the feature

- if we understand the pattern

Inputs with only a feature dimension

For examples that have only feature dimensions

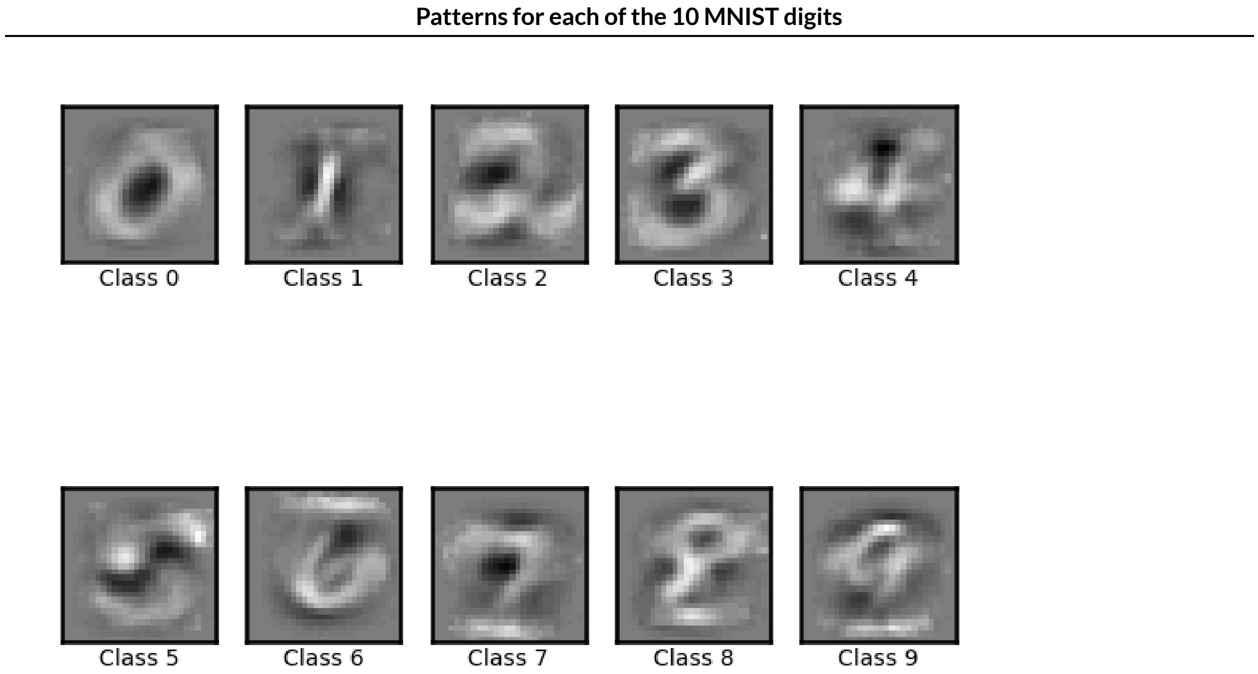
- the pattern is just a vector of feature values
- of length equal to the length of the input example

A Dense Layer has a pattern

- that exactly identifies the "ideal" input (highest dot product)

Recall the 10 patterns from our simple Logistic Regression Classifier for the 10 MNIST digits

- these are the "idealized" digits



Inputs with non-feature dimensions as well as a feature dimension

But we also allow examples to have a "shape"

- *non-feature* dimensions

For example

- an image has 2 non-feature dimensions: row and column
- in addition to a feature dimensions: e.g., 3 features: Red, Green, Blue

Recall our terminology when dealing with examples having $N \geq 1$ non-feature dimensions

- an element is a vector with only a feature dimension
- we can index an element by a vector of length N in
$$[1:d_1] \times [1:d_2] \times \ldots [1:d_N]$$
- an index identifies a specific *location* in the non-feature dimensions

The patterns are $(N+1)$ dimensional

- one feature dimensions of length n, which is also the number of input features
- N feature dimensions
    - each of length f
    - which is smaller than the length of the corresponding non-feature dimension of the input example

The output of the match with a single pattern is a *feature map*

- N-dimensional: matches the lengths of the input non-feature dimensions
- a measure of the strength of the pattern's match with the sub-region centered at each location

So the pattern

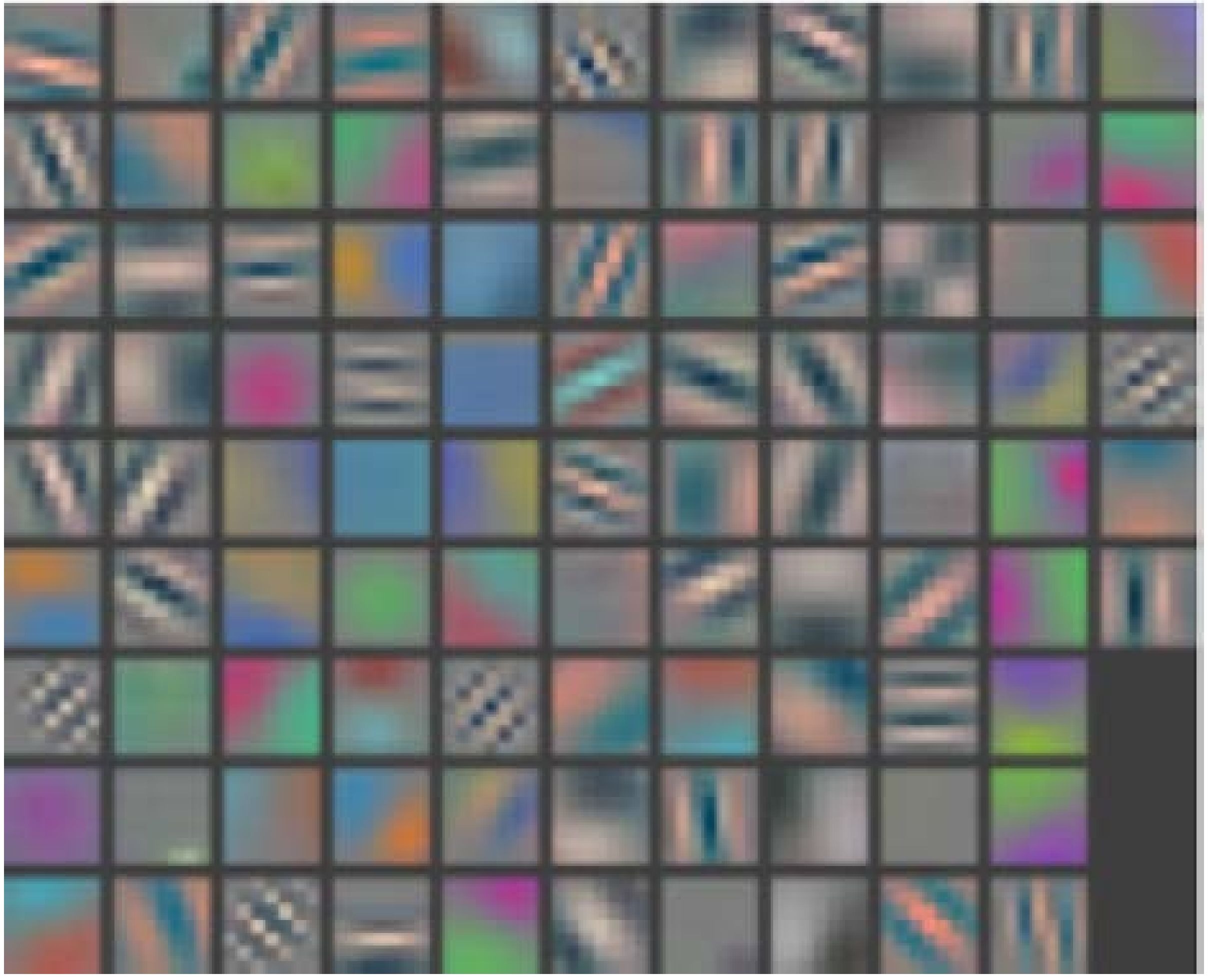
- identifies an "ideal" sub-region in the input example

To illustrate

- we show the patterns a CNN layer appearing in layer 1 of a NN
- there are $n_{(1)} = 96$ patterns
- each pattern is $(7 \times 7 \times n_{(0)})$
    - $n_{(0)} = 3$ are the number of input channels

Each square is a kernel.

Layer 1 kernels



Attribution: https://arxiv.org/pdf/1311.2901.pdf (https://arxiv.org/pdf/1311.2901.pdf)

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.

Beyond the first layer

Examining weights beyond the first layer presents difficulties

- the patterns are matched against outputs of layer $l > 0$
- we only know what the features are for layer 0
    - visually recognizable

So we can identify a pattern but can't assign a meaning to the inputs that are being matched.

We will have to come up with ways of interpreting synthetic features

- that do no involve interpreting the patterns

Probing

One way to gain insight is by *probing*

- choose one feature somewhere in the Neural Network
- try to discover Layer 0 inputs
- that causes this feature to assume high (positive or negative) values

We call the values produced at a feature in response to inputs the feature's *activations*.

To eliminate ambiguity, we will write

$$\mathbf{y}_{(l),k}\big|_{\mathbf{y}_{(0)}=\mathbf{x}^{(\mathbf{i})}}$$

to denote the activation when the Layer 0 input is $\mathbf{x}^{(\mathbf{i})}$

If

- we identify a property $\mathcal{P}$ common to all the inputs resulting in High values
- we can interpret the feature as being a detector for $\mathcal{P}$

The common property may not be easy to discern

- semantics: meaning
- rather than surface: appearance

For example

- there may be a neuron in some layer
- that acts as a "smile detector"
    - triggering only on inputs containing humans that are smiling

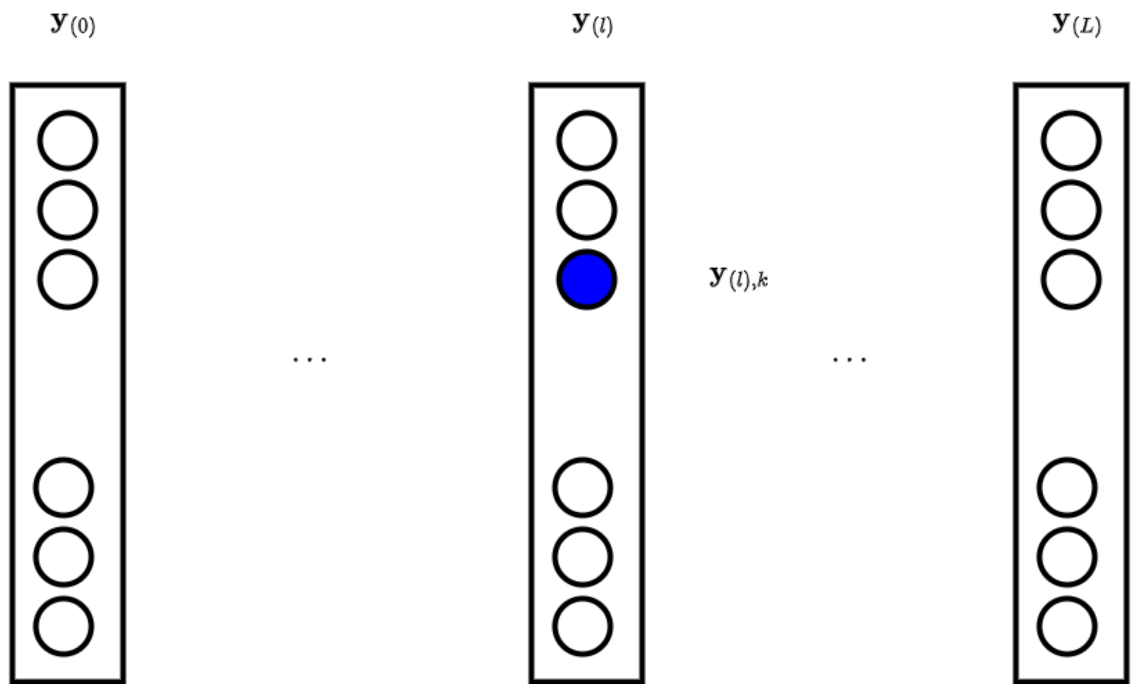
To be more precise:

Given a multi-layer Sequential Neural Network

- choose one feature at some layer to probe: $\mathbf{y}_{(l),k}$

We are interested in the output values (called *activations*) of this feature.

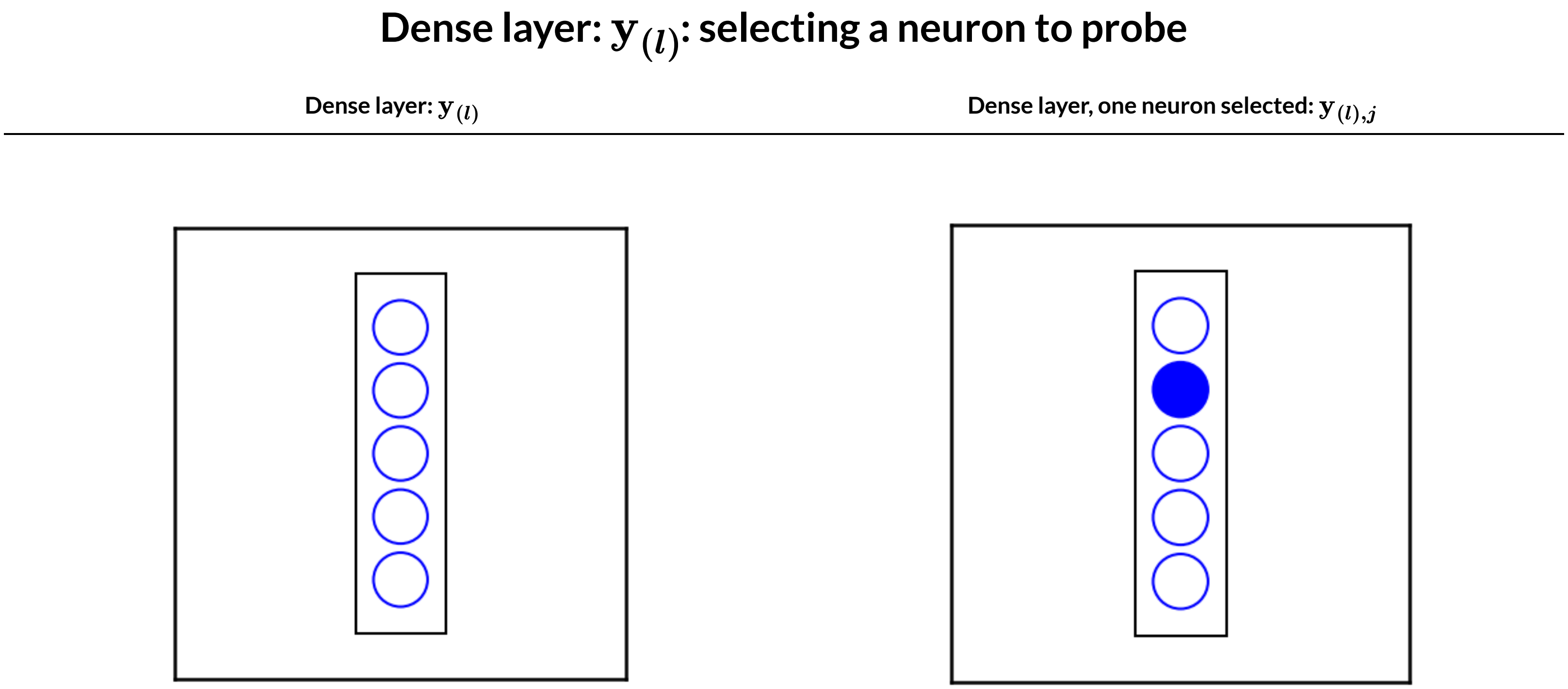
Interpreting neuron for feature k in layer l



When the layer l output has only feature-dimensions

- the selected feature is a scalar

for instance, a `Dense` layer:



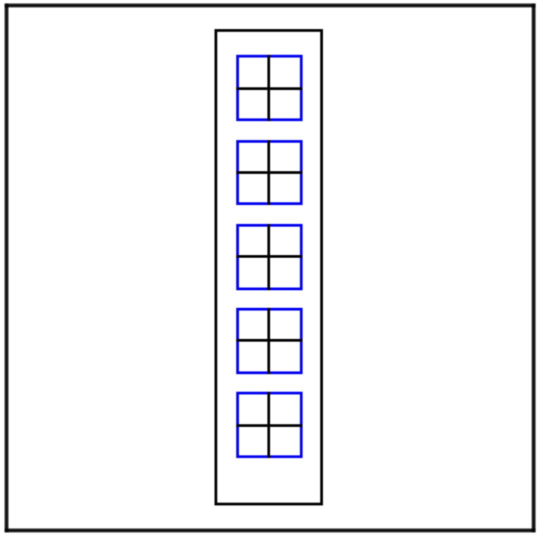
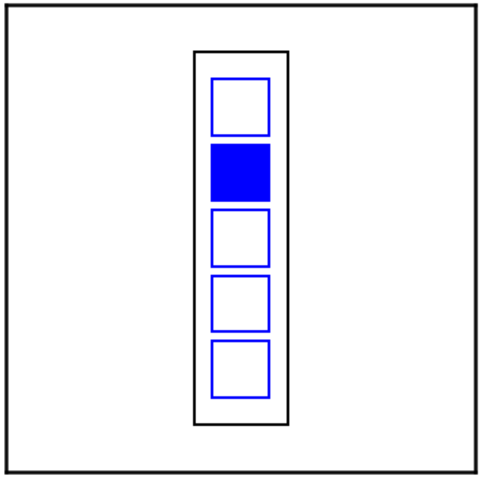
But when layer l has $N \ge 1$ non-feature dimensions

- the selected feature is really a *feature map*
- with dimensions matching the non-feature dimensions of the layer input
$$(d_1 \times d_2 \times \ldots d_N)$$

So there are $\prod_{i=1}^N d_i$ values (one per location) in the feature map

- rather than a single scalar value
- as in the case of layer outputs with only a feature dimension

Convolutional layer: $\mathbf{y}_{(l)}$: selecting a feature map to probe

Layer w/non-feature dimensions: $\mathbf{y}_{(l)}$	Layer w/non-feature dimensions, one element selected: $\mathbf{y}_{(l),j}$
	

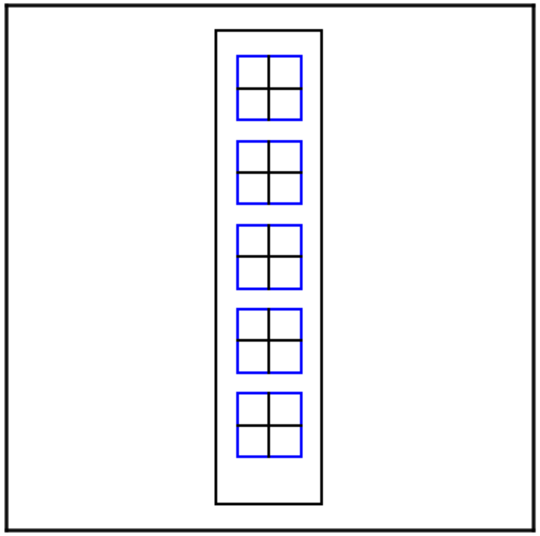
In such a case

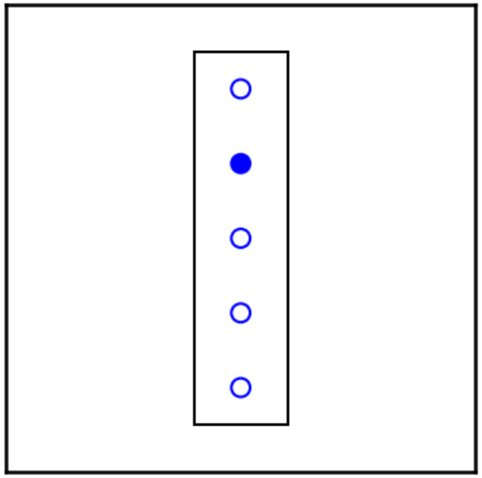
- we reduce each feature map (with non-feature dimensions)
- to a scalar
- using a Pooling operation to eliminate the non-feature dimensions
    - for example: Global Max Pooling

Convolutional layer: $\mathbf{y}_{(l)}$: selecting a feature map to probe

Global Pooling

Layer w/non-feature dimensions: $\mathbf{y}_{(l)}$	Layer w/non-feature dimensions, pooled, one element selected: $\mathbf{y}_{(l),j}$





Thus, Probing

- examines the activation of a feature
- where the activation is represented
- by a single scalar value

Maximally Activating Examples

This method identifies

- a subset S of the training examples
$$S \subset \mathbf{X}$$
- that produces high activations for the selected feature $\mathbf{y}_{(l),k}$

Hence, this method is called *Maximally Activating Examples*

The method is quite simple

- pass each input example $\mathbf{x^{(i)}}$ to the network
- measure the resulting activation of the selected feature

$$\mathbf{y}_{(l),k}\Big|_{\mathbf{y}_{(0)}=\mathbf{x^{(i)}}}$$

- rank the m resulting activations

$$\{i_1, \ldots, i_m\}$$

- Classify
  - the K highest (positive) magnitude activations as High
  - the K highest (negative) magnitude activations as Low

i	$\mathbf{y}_{(l),k}$	class
1	7.1	
2	-100.2	Low
3	- 6.3	

$\vdots$ 234 | 1000.4 | High $\vdots$ m | 45.6 |

Then the K Maximally Activating examples for $\mathbf{y}_{(l),k}$ are defined as

- the K examples with highest rank (classified as High)

$$\mathrm{MaxAct}_{(l),k,K} = \{\mathbf{x}^{(i_1)}, \ldots, \mathbf{x}^{(i_N)}\}$$

We then try

- via Intuition, Experiment
- to identify the property $\mathcal{P}$
- that is unique among $\mathbf{X}$ to the examples in $\mathrm{MaxAct}_{(l),k,K} = \{\mathbf{x}^{(i_1)}, \ldots, \mathbf{x}^{(i_N)}\}$

Probing the Classifier Head

Applying the Maximally Activating Examples technique to the head layer L is particularly useful

For a Classifier Head:

$$\mathbf{y}_{(L),k}\big|_{\mathbf{y}_{(0)}=\mathbf{x}^{(\mathbf{i})}}$$

- is the probability (or pre-probability "logit")
- that example $\mathbf{x}^{(\mathbf{i})}$ is in Class k

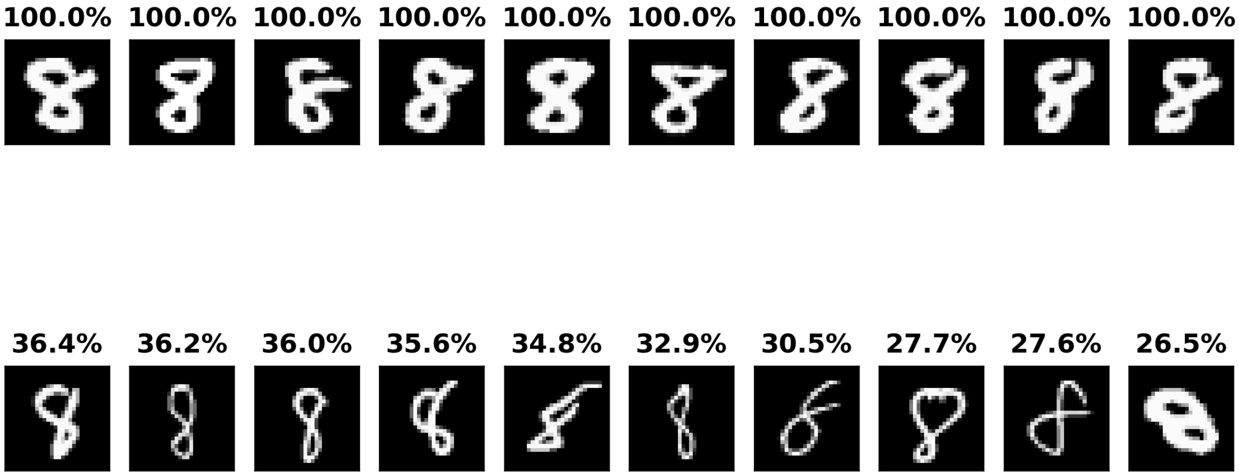
We can use Maximally Activation examples on a Head feature

- to identify inputs
- that are most/least confidently classified as being in Class k

Here we apply the technique to a restricted subset $\mathbf{X}' \subset \mathbf{X}$ of input images of digits that have label "8"

$$\mathbf{X}' = \{ \mathbf{x^{(i)}} \mid \mathbf{y^{(i)}} = 8 \text{ where } 1 \le i \le m \}$$

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.

- when the inputs have non-feature dimensions
- Global Pooling compresses *all* the locations to a single scalar
- losing information about the sub-region having the property

There is a simple technique called *Occlusion*

- that enables us to find a sub-region of a *particular input* $\mathbf{x}^{(\mathbf{i})}$
- that is responsible for the activation

$$\mathbf{y}_{(L),k}\Big|_{\mathbf{y}_{(0)}=\mathbf{x}^{(\mathbf{i})}}$$

It is similar in concept to Convolution applied to Layer 0 (the example)

In Convolution, we take a filter

- with N non-feature dimensions
- each of length f
- and $n_{(0)}$ features

and compute the dot product of the filter with the sub-region of $\mathbf{x}^{(\mathbf{i})}$ centered at each location.

- resulting in a feature map with identical non-feature dimensions as the input
$$d_1 \times d_2 \times \ldots d_N$$
- measuring the strength of the match of the filter and sub-region at each location
    - for each filter/kernel in the Convolutional layer

In Occlusion

- the sub-region of $\mathbf{x^{(i)}}$ centered at each location
- has all its values changed to an extreme value
    - equivalent to "hiding" the sub-region

Rather than computing the dot product at each location, Occlusion produces

- a feature map (*Occlusion Sensitivity map*) with identical non-feature dimensions as the input
- measuring *the change in* the probability $\mathbf{y}_{(L),k}$
    - from un-occluded $\mathbf{y}_{(L),k}\big|_{\mathbf{y}_{(0)}=\mathbf{x^{(i)}}}$
    - to the value of $\mathbf{y}_{(L),k}$ when the location is the center of the occluded region

It is the sensitivity of $\mathbf{y}_{(L),k}\big|_{\mathbf{y}_{(0)}=\mathbf{x^{(i)}}}$ to being occluded at each location.

For inputs with non-feature dimensions

$$d_1 \times d_2 \times \ldots d_N$$

the Occlusion sensitivity Map has the same non-feature dimensions

- just like Convolution with a single kernel/filter

Thus the non-feature dimensions of the input and the sensitivity map are identical.

Below is an example for an image with label: Afghan Hound.

It would seem that this feature recognizes faces.

- activation drops (blue = cold) when the faces are occluded

Occlusion Sensitivity Map

Input image	Activation of one filter at layer 5

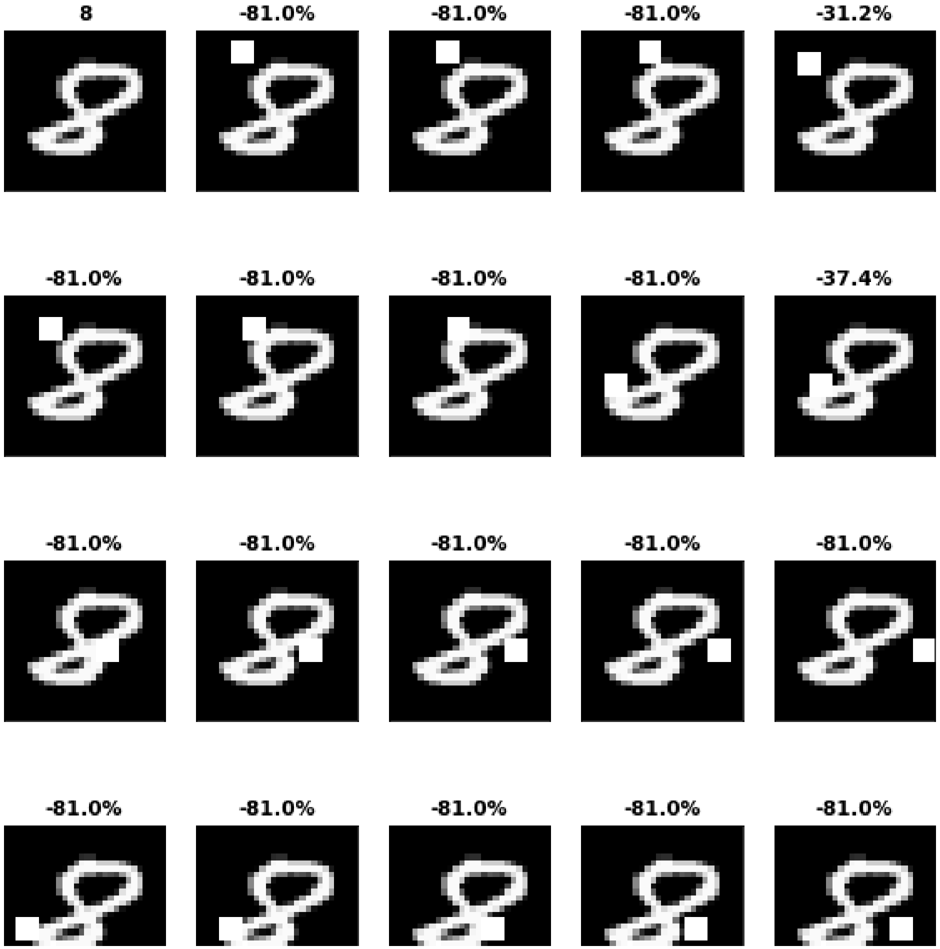
Attribution: [https://arxiv.org/pdf/1311.2901.pdf#page=7 (https://arxiv.org/pdf/1311.2901.pdf#page=7)](https://arxiv.org/pdf/1311.2901.pdf#page=7)

Occlusion Experiment 1: Head Layer logit on MNIST digit Classification

The following figure shows

- some of the occluded locations in the feature map
- of a particular example $\mathbf{x}^{(\mathbf{i})}$ representing digit "8"
- with the proportional change in $\mathbf{y}_{(L),8}$ indicated at the top of the occluded input
- for a NN performing MNIST digit classification

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%

Not what we expected !

The mere presence of the square changes the classification probability greatly

- even when we are not occluding what we believed to be the most important sub-regions of $\mathbf{x}^{(\mathbf{i})}$

- the ”pinched waist” of the 8.

This suggest that the NN performs Classification

- in a way different than what we might have directed to using a Procedural Program
- perhaps extreme locations
    - are used to recognize other digits
    - so the "bright" occlusion mask confuses the Classifier

We might want to use Data Augmentation to correct the Classifier

- adding noise to inputs, preserving the label
- to immunize the Classifier from bright spots at extreme locations

Occlusion Experiment 2: How does an ImageNet Classifier work

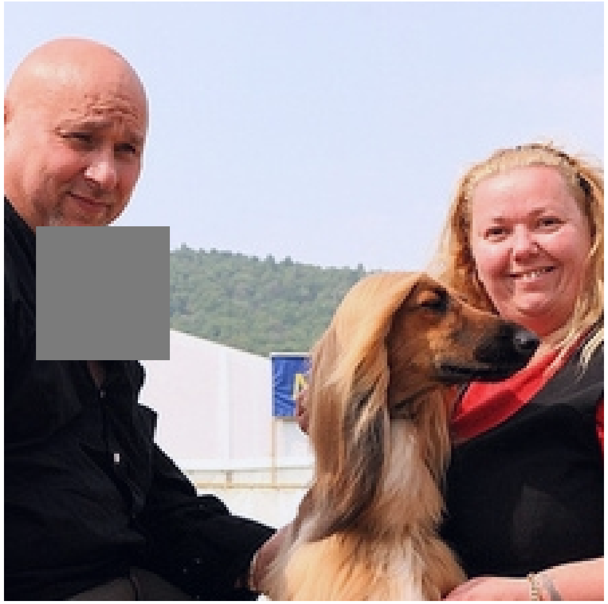
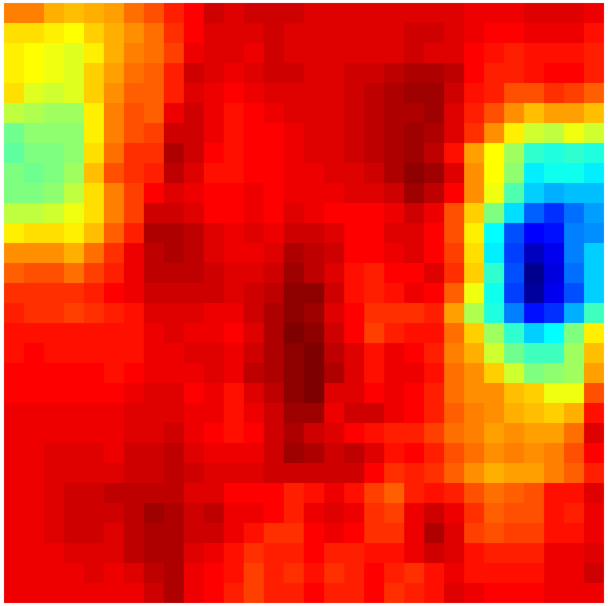
ImageNet was a competition (important historically in the evolution of Neural Networks)

- classification of images
- from among 1000 different classes
    - 200 different types of dogs and cats !

[Zieler and Fergus (https://arxiv.org/pdf/1311.2901.pdf)](https://arxiv.org/pdf/1311.2901.pdf) have some interesting Occlusion results.

The Occlusion Sensitivity map we used as illustration above comes from this paper

- Interpretation of Layer 5 feature: face detector

Occlusion Sensitivity Map	
Input image	Activation of one filter at layer 5
	

Attribution: [https://arxiv.org/pdf/1311.2901.pdf#page=7 (https://arxiv.org/pdf/1311.2901.pdf#page=7)](https://arxiv.org/pdf/1311.2901.pdf#page=7)

The fact that we have discovered a "face detector" is interesting.

- Faces *are not* one of the 1000 possible labels
- Perhaps this non-label feature is necessary
    - to assist in creating features that *do identify* labels

For example

- there is evidence that many Classifiers have features that recognize Letter Characters (e.g., A-Z)
    - not one of the 1000 classes
- which may, in turn
    - help to identify "Book", which is of the 1000 classes

The results of probing

- the logit of the class "Afghan Hound"
    - the correct label for the input image
- is very interesting

Occluding the dog causes a big drop (blue: cold) in probability of correct classification

- as expected

But occluding each face *increases the probability* (red: hot) of correct classification !

- Perhaps the presence of a face is suggestive of an alternative class
    - removing the input signal for the alternative class results in a more confident prediction for the correct clas
- Even though "face" is not itself a class

Occlusion has helped us learn something unexpected about the workings of the Neural Network.

Saliency maps

Each location in the Occlusion Sensitivity map reflects

- a change in $\mathbf{y}_{(l),k}$
- give a fairly big change
    - occlusion replaces pixels with an extreme value
- in a region of $\mathbf{y}_{(0)}$

We can compute a more traditional sensitivity via the derivative

$$\frac{\partial \mathbf{y}_{(l),k}}{\partial \mathbf{y}_{(0)}} \Bigg|_{\mathbf{y}_{(0)}=\mathbf{x}^{(\mathbf{i})}}$$

Each location in this derivative (same non-feature dimensions as $\mathbf{y}_{(0)}$) reflects

- a change in $\mathbf{y}_{(l),k}$
- for an infinitesimal change
- in a single location in $\mathbf{y}_{(0)}$

This is called a *Saliency Map*

- when input has non-feature dimensions
- the Saliency Map has the same non-feature dimensions

$$d_1 \times d_2 \times \ldots d_N$$

Saliency Maps, when applied to a Head Layer logit k

- explains the influence of each location in the input $\mathbf{y}_{(0)}$
- on the classification of the input as being in class k

Hence, they are useful for explaining the output of a NN.

Understanding a non-head layer via Saliency Maps

Saliency Maps are also useful for explaining features in non-head layers.

Recall that the Saliency Map and Input have the same non-feature dimensions.

Saliency map for a shallow layer

Below are a collection of Saliency Maps for some feature in Layer 2 of an ImageNet Classifier.

- maps for 9 **different input examples**
    - the examples with largest activation in the feature map

All 9 examples appear to be eye-balls.

It would seem this Layer 2 feature is recognizing eye balls.

The diagram can be confusing

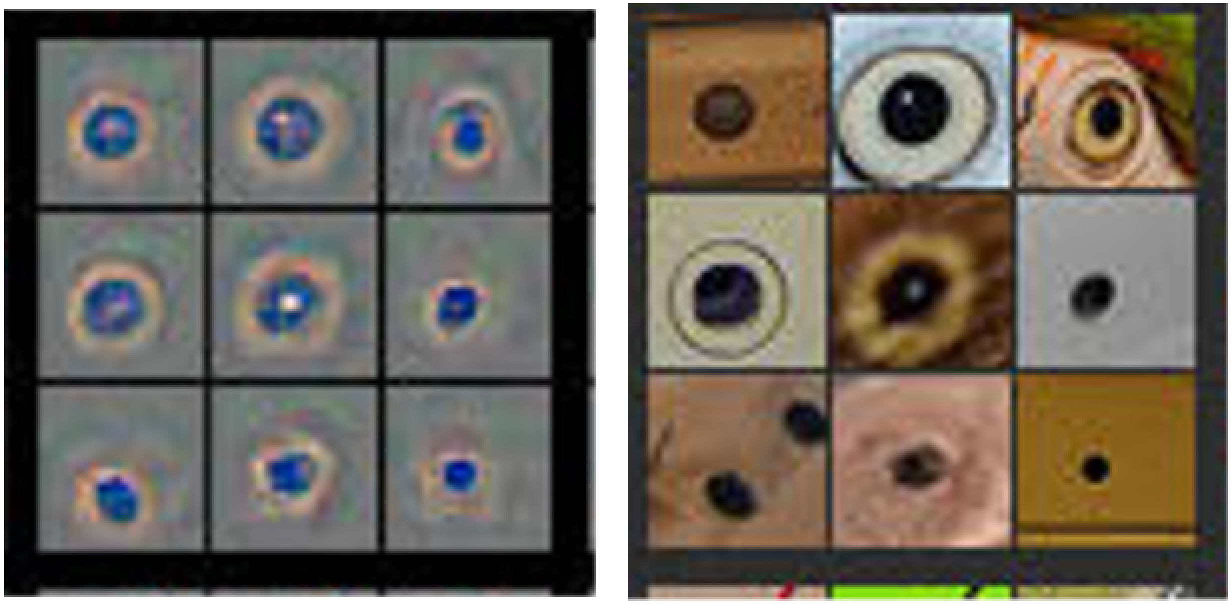
- they are for 9 **different input examples**
- the non-feature dimensions seem to be for a sub-region (a *patch*) of the input, rather than the entire input
    - just the eye, not the rest of the image

We will explain after presenting the diagram.

As a first pass

- these are the 9 examples that stimulated the feature most strongly
    - hence, may be useful for interpreting what the feature is
- on the left is a saliency map for a sub-region (*patch*) of the input
- on the right is the corresponding patch

Saliency Maps and Corresponding Patches
Single Layer 2 Feature Map
On multiple input images



Layer 2 Feature Map (Row 10, col 3).

Attribution: https://arxiv.org/abs/1311.2901#page=4 [(https://arxiv.org/abs/1311.2901#page=4)](https://arxiv.org/abs/1311.2901#page=4)

Explaining why the diagram has "small" maps and patches

Why are the Saliency Maps and corresponding patches restricted to sub-regions of the input ?

- i.e., smaller than
  $d_1 \times d_2 \times \ldots d_N$

Recall that the multiple locations in the layer are reduced to a single value

- the max, when using Max Pooling for the summarization
- so the Saliency map is the change of a *single location* $\mathbf{y}_{(l),\text{idx},k}$ in $\mathbf{y}_{(l),k}$

$$\frac{\partial \mathbf{y}_{(l),\text{idx},k}}{\partial \mathbf{y}_{(0)}} \Big|_{\mathbf{y}_{(0)}=\mathbf{x}^{(\mathbf{i})}}$$

  - where idx the location of the *max*

In a NN with multiple CNN layers,

- the [*receptive field* (CNN_Receptive_Field.ipynb)](CNN_Receptive_Field.ipynb)
- is the input **sub-region** that affects a single location in a layer
- the dimensions of the sub-region grows with the depth (i.e., layer number l) of the layer

So, in a shallow layer (i.e., Layer 2 in our diagram)

- the receptive field for *any* location
- is less than the full input
    - very small: only slightly larger than f, the size of a side of the filter/kernel

Thus, the non-feature dimensions of the Saliency Map for a shallow layer (e.g., layer 2 in the diagram)

- is much smaller than
$$d_1 \times d_2 \times \ldots d_N$$
- because the receptive field for idx, the location of the max in layer l
- is small

Saliency map for a deeper layer

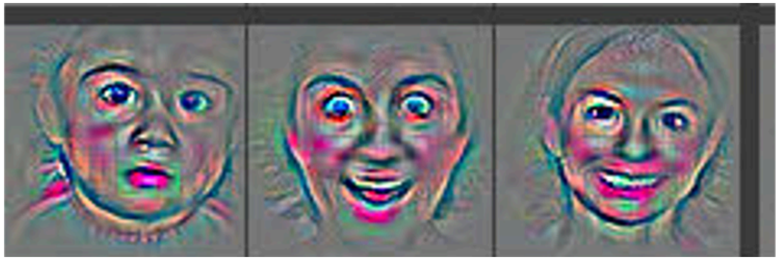
As we go deeper into the network

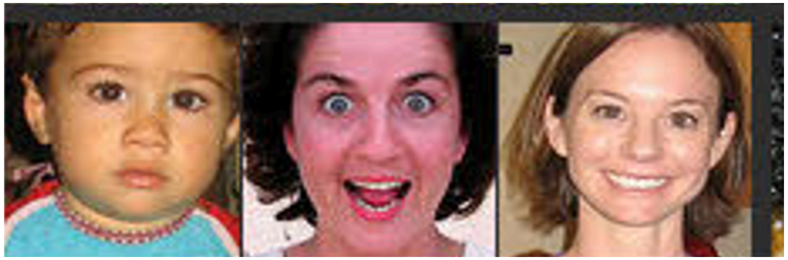
- the size of the receptive field grows in a NN with successive CNN layers
- the representations become more complex
    - perhaps because of the larger receptive field
    - perhaps just because they are combinations of more complex representations
        - their layer inputs

In Layer 5, the feature whose map we show

- may be recognizing "smiling faces"
    - note the high (red) sensitivity
    - to lips and cheeks

Saliency Maps and Corresponding Patches
Single Layer 5 Feature Map
On 9 Maximally Activating Input images





Computing the Saliency Map

Computing a Saliency Map is easy

- a simple variant of Back Propagation

Recall the definition of the Loss Gradient in Back Propagation

$$\mathcal{L}'_{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}}$$

and it's recursive update

$$\begin{aligned}
\mathcal{L}'_{(l-1)} &= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l-1)}} \\
&= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} \\
&= \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}
\end{aligned}$$

To compute Saliency Maps

- replace $\mathcal{L}$ with $\mathbf{y}_{(l),k}$
- so the "loss gradient" is now the "saliency gradient"

$$\mathcal{L}'_{(l')} = \frac{\partial \mathbf{y}_{(l),k}}{\partial \mathbf{y}_{(l')}}$$

  - we use the index l to denote the layer of the feature map
  - thus, we are forced to use l' in the subscript of $\mathcal{L}'$ to avoid conflict

Substituting $l' = 0$:

$$\mathcal{L}'_{(0)} = \frac{\partial \mathbf{y}_{(l),k}}{\partial \mathbf{y}_{(0)}}$$

we get the derivative defining the Saliency Map.

Guided Back Propagation

Our ultimate purpose is to try to *interpret* the meaning of a synthetic feature.

The "true" mathematical derivative of the Saliency Map

- is sometimes sacrificed
- in order to enhance the interpretability

[Zeiler and Fergus (https://arxiv.org/abs/1311.2901)](https://arxiv.org/abs/1311.2901) (and similar related papers) modify Back propagation

- In an attempt to get better intuition as to which input features most affect $\mathbf{y}_{(l),k}$
- For example: ignore the *sign* of the derivatives as they flow backwards
    - Look for strong positive or negative influences, not caring which

This is called *Guided Back propagation*.

Video: interactive interpretation of features

There is a nice video by [Yosinski (https://youtu.be/AgkfIQ4IGaM)](https://youtu.be/AgkfIQ4IGaM) which examines the behavior of a Neural Network's layers on video images rather than stills.

- using several of the techniques we describe

In [4]:

```
print("Done")
```

```
Done
```