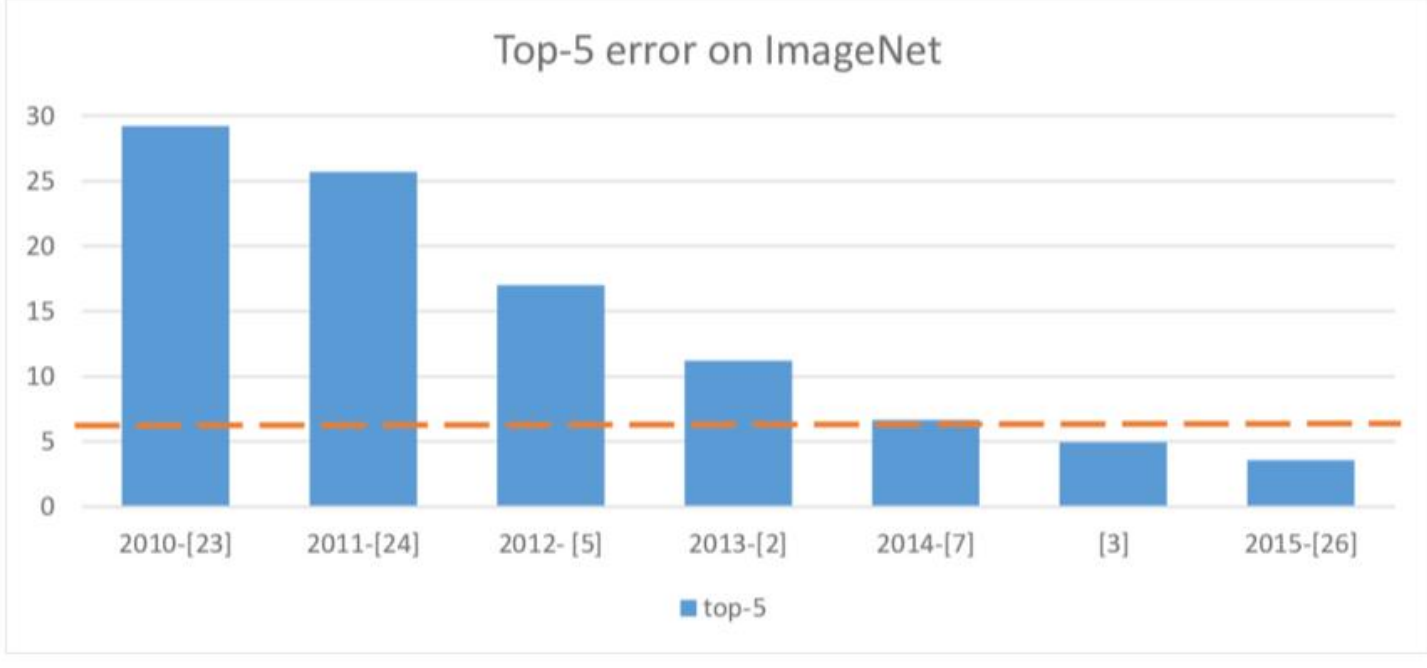
How does a Deep Learning Classifier work ?

We will present the outputs of a very high accuracy classifer for the ImageNet dataset

ImageNet:

- Large database of hand-labelled images -14MM images, 22K classes
    - High level categories (http://image-net.org/about-stats)
- Annual competition
    - no Deep Learning prior to 2012
    - drove innovation in Deep Learning post 2012
    - Training data: 1.2MM images
    - 200 dogs and cats !

Deep Learning Revolution



Chennupati: https://www.researchgate.net/figure/Shows-the-progress-of-classification-performance-top-5-error-on-Imagenet-dataset-over_fig27_312935261

Here is the classifier's response to a cat image:

High confidence.

How does the classifier "recognize" this as a "tiger cat" ?

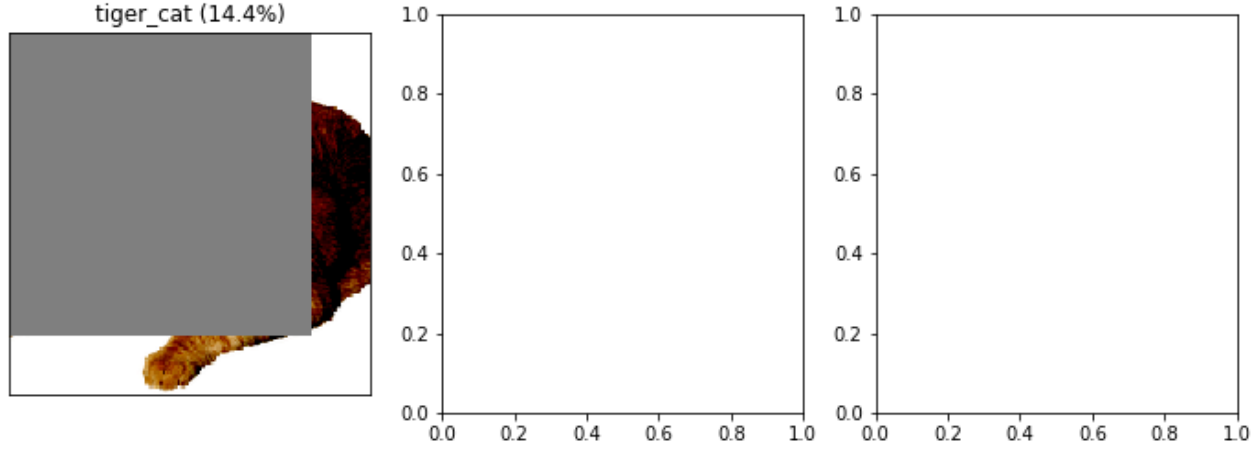
Maybe: by it's parts ?

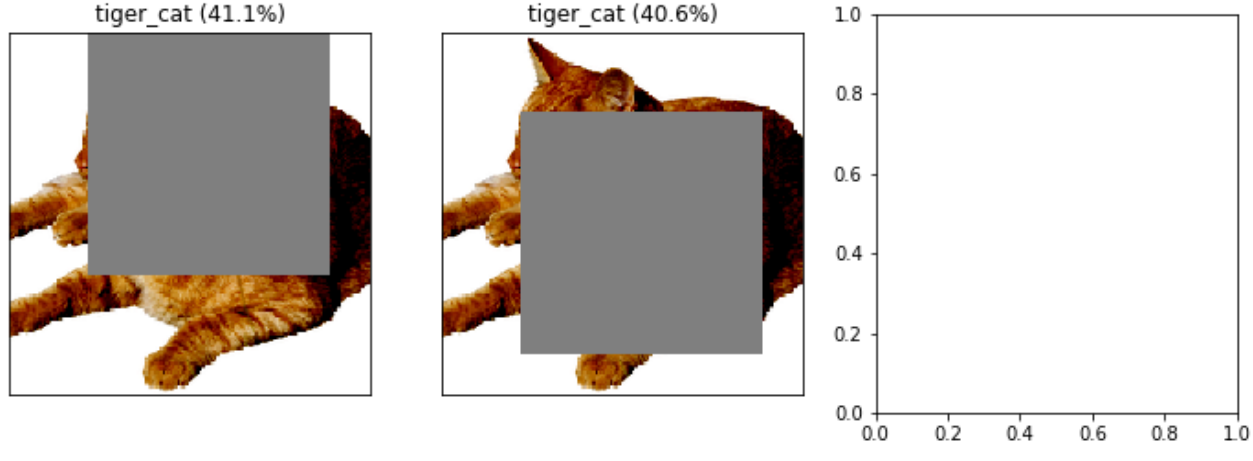
How does it work: Parts ?

Maybe parts, but certainly not arranged properly !

- CNN filter looking for presence/absence of features
- not necessarily location

How does it work: Parts ?

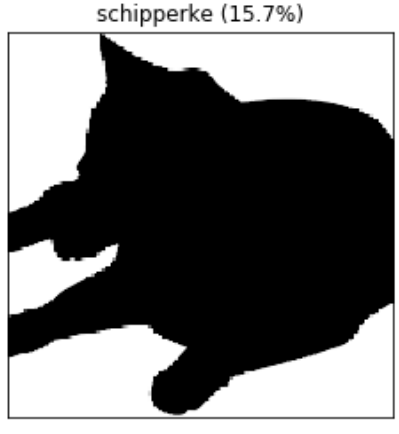




Probably not. Covering up (occluding) various parts still results in correct classification.

What about it's shape ?

How does it work: Shape ?



Probably not.

Maybe: texture ?

How does it work: Texture ?

tiger_cat (83.6%)

African_elephant (65.2%)

Perhaps it's the texture.

What is a feature map looking for ?

Up until now, our understanding of the workings of a NN has been limited

- each layer is a transformation
    - from representation ("synthetic features") given by output of layer $l-1$
    - to a new latent representation, the output of layer l
- for classification
    - the final layer is a logistic regression
        - pattern matching the features of penultimate layer

We now begin a quest to understand *what* these transformations are accomplishing.

Much of the presentaton is based on a very influential paper by Zeiler and Fergus (https://arxiv.org/abs/1311.2901)

- NYU PhD candidate and advisor !

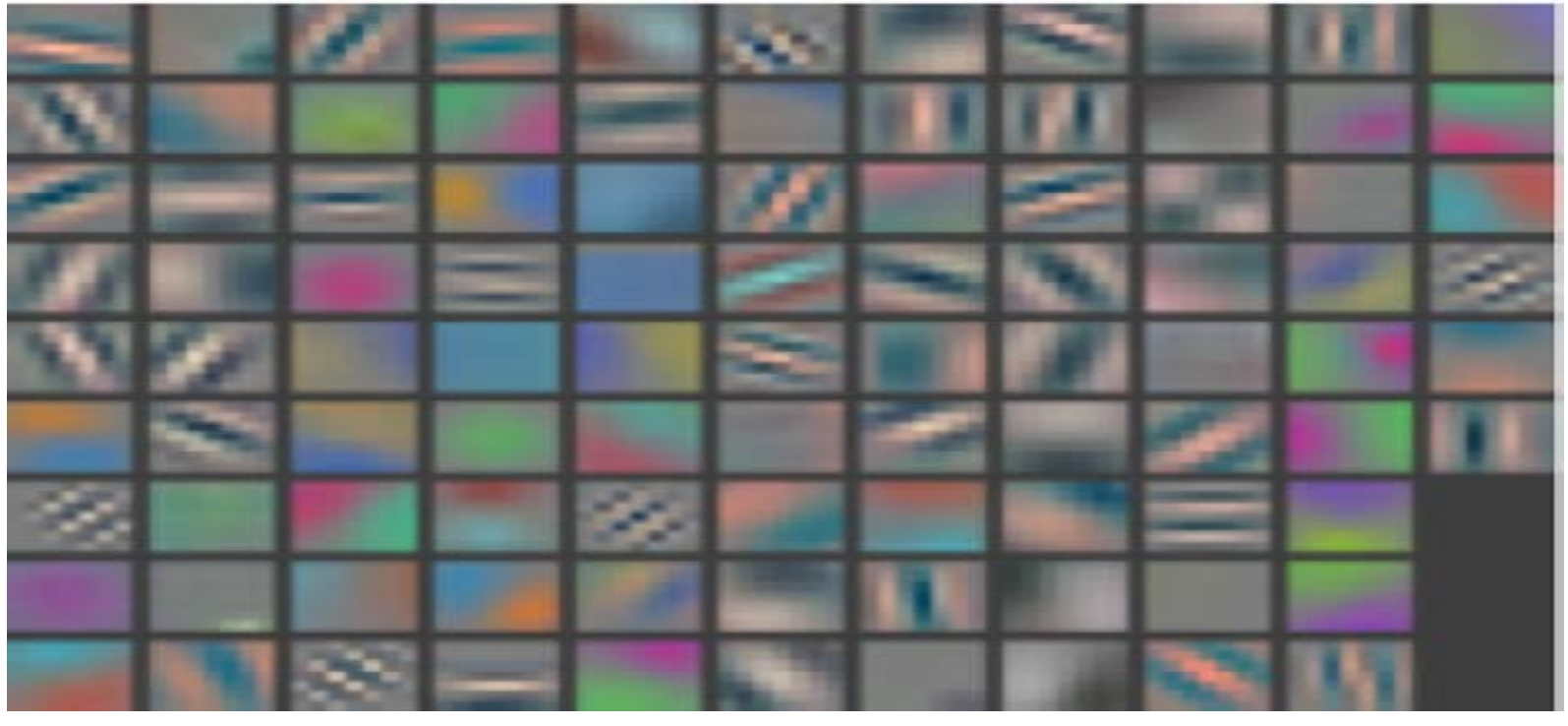
The first layer

It is relatively easy to understand the transformation of the first layer, as its inputs are from our problem domain, which makes them interpretable.

For a Dense (FC) layer, we can view the weights as the pattern in the input domain being searched for.

This applies as well to the filters in the first convolutional layer.

Layer 1 filters



- Each square in the grid represents values of a single filter ("template") in Convolutional Layer 1
- The templates seem to represent simple geometric shapes
    - lines in various orientations
    - colors
    - shading

Beyond the first layer

Interepretation of filter/weights beyond the first layer is difficult:

- layer l takes features from layer $l-1$
- which are synthesized and not necessarily interpretable

What we can hope to do

- somehow map the representation created by layer $l > 1$ into the inputs (layer 0 output)

The methods fall into two classes

- input dependent
- input independent

Interpretation of activations

Recall that the output of each layer is a transformed representation of the input.

So input $\mathbf{x}^{(\mathbf{i})}$ gets transformed to $\mathbf{y}_{(l)}^{(\mathbf{i})}$ at layer l.

We will give several methods to try to discern the meaning of $\mathbf{y}_{(l)}$.

PCA

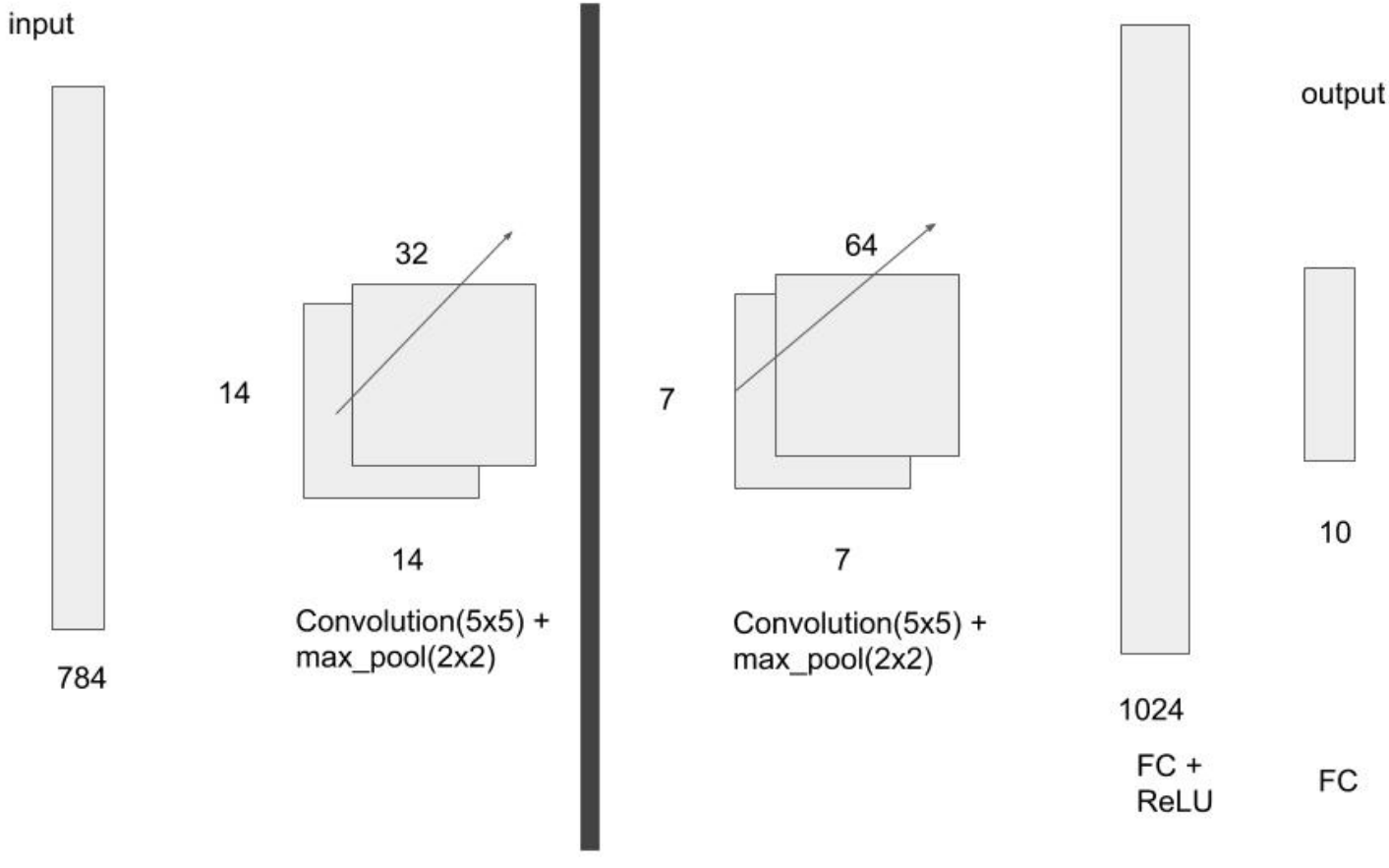
- Feed all n examples in $\mathbf{X}$ into the NN
    - layer l representation of $\mathbf{x^{(i)}}$: $\mathbf{y}_{(l)}^{\mathbf{(i)}}$
- Compute PCA of the collection of representations $[\mathbf{y}_{(l)}^{\mathbf{(i)}} | 1 \leq i \leq n]$ _ Is there some property p such that
    - Project example i onto the first few PC's
    - label the projected point with $p(\mathbf{x^{(i)}})$
    - are clusters formed with similar values of property p ?

Here is a Convolutional Neural Network applied to MNIST digit classification.

MNIST CNN

PCA

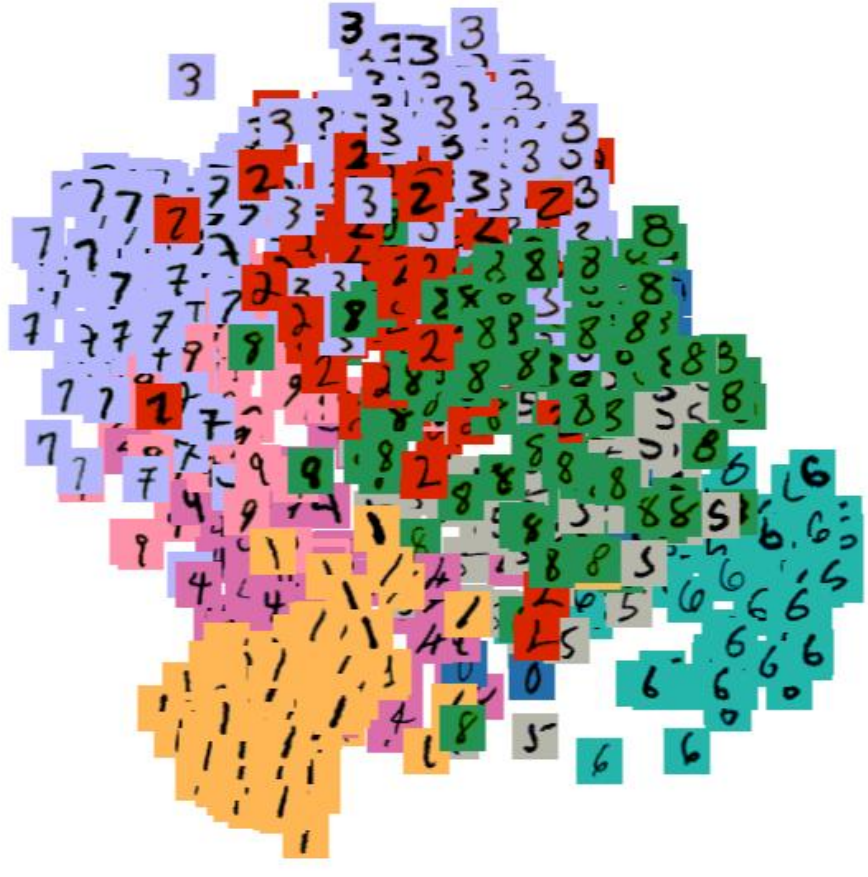
- Interpret the intermediate representation (what is the transformed feature ?)



And a projection of the representaion produced by the first Convolutional Layer onto the first 2 PC's

MNIST CNN Conv1 PCA

PCA: MNIST Deep Classifier (post conv1)



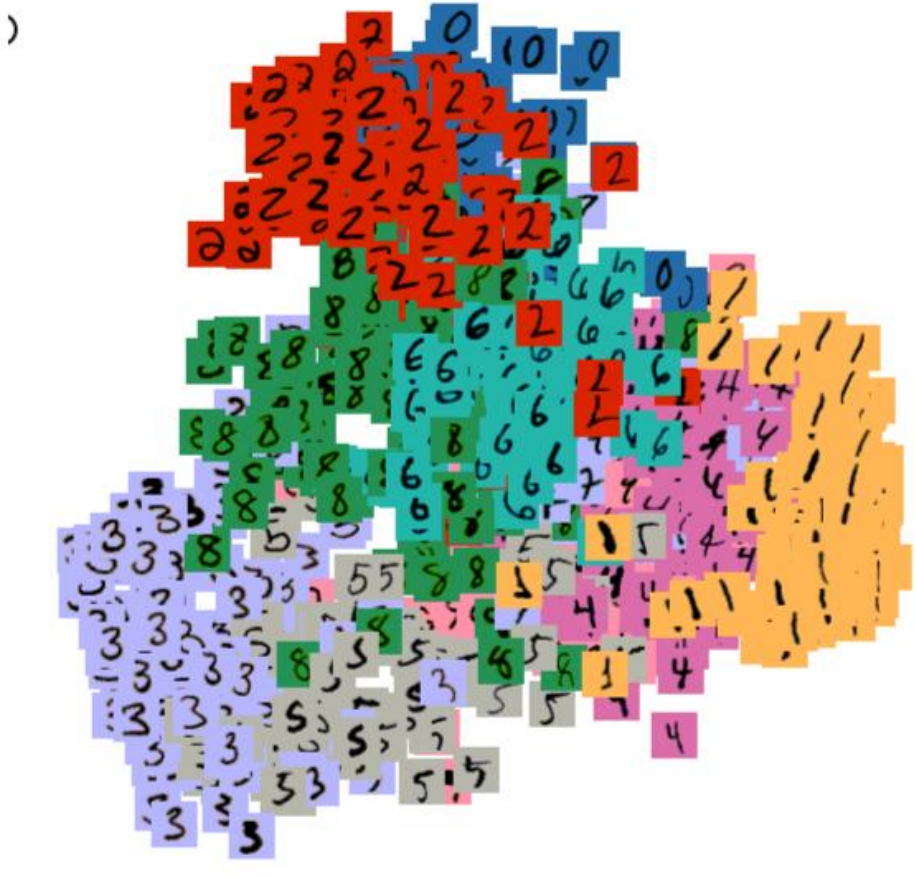
The property we are postulating is useful because "similar" digits are clustered together

- Is the layer recognizing features that group digits ?
- Left to right: strong vertical ("1", "7") to less vertical ?
- Bottom to top: digits *without* "curved tops" to those with tops ?

Let's examine the representation after the second Convolutional Layer.

MNIST CNN Conv1 PCA

PCA: MNIST Deep Classifier (Post conv2)



Interpretation: What is the role of a single neuron/single feature map ?

Rather than intepreting $\mathbf{y}_{(l)}$ in its entierty, perhaps we can discern the meaning of a single element j of $\mathbf{y}_{(l)}$.

For a given layer l, the layer output $\mathbf{y}_{(l)}$ consists of many features.

Can we discern what role feature $\mathbf{y}_{(l),j}$ plays ?

Note If $\mathbf{y}_{(l)}$ is of dimension $n_{(l)} > 2$ then $\mathbf{y}_{(l),j}$ denotes a single *feature map* spanning a multi-dimensional space.

So when we refer to the value of $\mathbf{y}_{(l),j}$, we mean a summary (e.g., max, average) of all values in the feature map.

Maximally Activating Examples

- Feed all n examples in $\mathbf{X}$ into the NN
- Measure the response $\mathbf{y}_{(l),j}$
- Do the examples with largest/smallest responses share some common property p ?

If so, then perhaps $\mathbf{y}_{(l),j}$ encodes a feature measuring the strength of p

Let

- $\mathbf{y}^{(\mathbf{i})}_{(l),j}$ denote the response of feature $\mathbf{y}_{(l),j}$ to input $\mathbf{x}^{(\mathbf{i})}$.
- $[i_1, i_2, \ldots, i_n]$ permutation of $[1, \ldots, n]$ that sorts the responses $[\mathbf{y}^{(\mathbf{i})}_{(l),j} | 1 \leq i \leq n]$
    - $\mathbf{y}^{(i_1)}_{(l),j} \leq \mathbf{y}^{(i_2)}_{(l),j} \cdots \leq \mathbf{y}^{(i_n)}_{(l),j}$
- Is $p\big(\mathbf{x}^{(i^k)}\big)$ true for all $k > T$ for some index $T \leq n$?

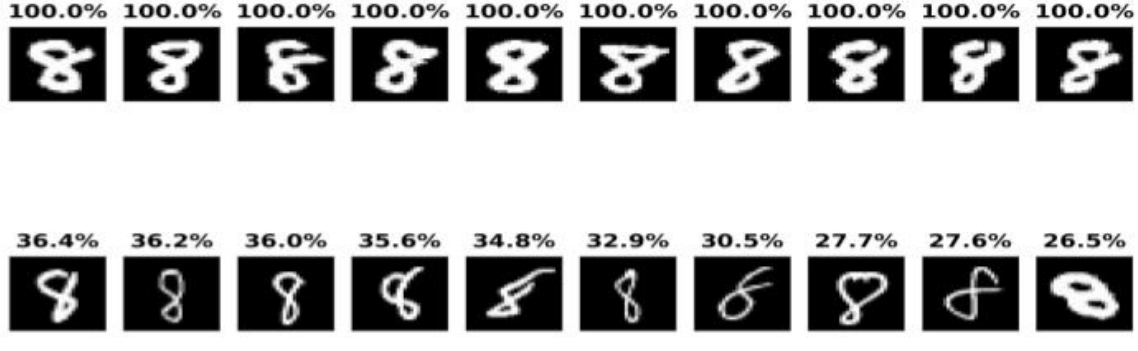
MNIST CNN maximally activating 8's

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

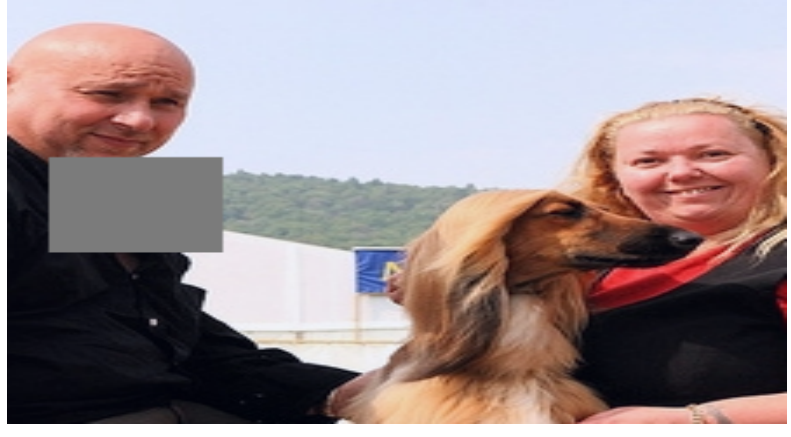
Not what we expected !

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

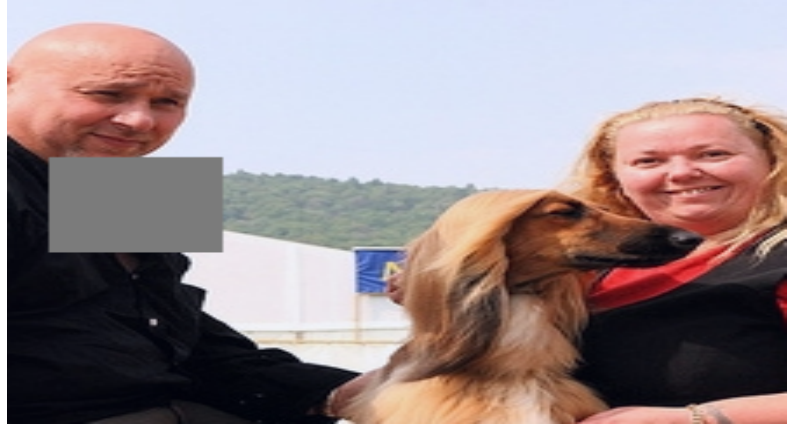
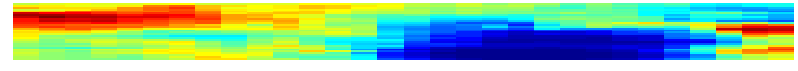
Here is the change in response of a single feature map in layer 5 of an image classifier (Zeiler and Fergus).

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"
	

- When the dog is masked, logits drop a lot (get colder: blue)
- When the two faces are masked, the logits increased (get hotter: red)
    - perhaps the faces were competing with the dog for possible classification ?

This is super interesting because "Face" is *not* in the set of target classes for the training set!

This suggests that the CNN

- found it useful to recognize faces
- even though Face is not an output class
- perhaps because Face is *correlated with* some other target class

The authors also found this to be true for Text.

- not an output class
- but strongly correlated with Book, which is a target class

Gradient Ascent: Inverting a Neural Network

Our use of NN thus far has been to find weights $\mathbf{W}$

- that maximize correctly predicting $\hat{y} = \mathbf{y^{(i)}}$ given input $\mathbf{x^{(i)}}$.

$$\mathbf{W} = \underset{\mathbf{W}}{\operatorname{argmin}} \mathcal{L}$$

where $\mathcal{L}$ is a measure of the "distance" between correct prediction $\mathbf{y^{(i)}}$ and actual prediction $\hat{\mathbf{y}}$.

- typically MSE for regression
- cross entropy for classification

We solved for the loss minimizing $\mathbf{W}$ by Gradient Descent on the loss function

- Find the gradient of the loss with respect to weights

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$$

- update weights $\mathbf{W}$ in the negative direction of the gradient

Let's suppose our trained NN classifies inputs $\mathbf{x}$ among classes in set C.

Let $\mathbf{W}$ be the weights that we obtained.

Suppose we *freeze* $\mathbf{W}$ and present the NN with the following input/target pair

$$(\mathbf{x}^{(i')}, \mathbf{y}^{(i')}) = (\mathbf{x}^{(0)}, \mathbf{y}^{(0)})$$

where

- $\mathbf{x}^{(0)}$ is random noise, or all zero
- $\mathbf{y}^{(0)} \in C$

What does the following derivative do ?

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$$

This gradient (with respect to the input layer $\mathbf{x}$)

- shows us the direction in which to modify $\mathbf{x}$
- so that the NN will classify the modified input as $\mathbf{y}^{(0)}$

Our optimization becomes

$$\mathbf{x} = \underset{\mathbf{x}}{\operatorname{argmin}} \mathcal{L}^{(\mathbf{i})}$$
$$\text{subject to}$$
$$\hat{\mathbf{y}}^{(\mathbf{i})} = \mathbf{y}^{(0)}$$

That is, we are

- starting with $\mathbf{x} = \mathbf{x}^{(0)}$
- modifying $\mathbf{x}$
- to derive an input $\mathbf{x}$ that causes the NN to predict $\hat{y} = \mathbf{y}^{(0)}$ with highest probability

We are inverting the output.

We refer to this process as *Gradient Ascent* on the input.

Note that the final $\mathbf{x}$ is not unique

- it is conditioned on the initial $\mathbf{x}^{(0)}$

We will show how to abuse the NN by choice of $\mathbf{x}^{(0)}$

The inverted $\mathbf{x}$ can be thought of as *the canonical* example of an input with class $\mathbf{y}^{(0)}$

We can use this $\mathbf{x}$ as a way of "discovering" the input pattern that the NN is seeking to match when classifying an input with label $\mathbf{y}^{(0)}$.

Note that we don't have to invert ultimate output $\hat{y} = \mathbf{y}_{(L)}$.

We can invert any single activation in any layer $\mathbf{y}_{(l)}$.

Thus we are "discovering" what any individual unit in any layer is seeking to match.

What is a CNN looking for: Zeiler and Fergus

Input dependent methods

Saliency maps/guided back propagation/deconvolution

A method similar to occlusion is to just compute the derivative of $\mathbf{y}_{(l),w,h,c}$ with respect to each coordinate (w, h) in input $\mathbf{x}^{(\mathbf{i})}$:

$$\frac{\partial \mathbf{y}_{(l),w,h,c}}{\partial \mathbf{x}^{(\mathbf{i})}_{w,h}}$$

There are a group of closely related methods along these lines that were discovered independently.

The objective of each of these methods is to identify the elements of the input example that most affect an activation (or summary).

All the techniques use variants of back propagation on $\frac{\partial \mathbf{y}_{(l),w,h,c}}{\partial \mathbf{x}^{(\mathbf{i})}_{w,h}}$

Observe that input elements $\mathbf{x}^{(\mathbf{i})}_{w,h}$ that don't contribute to $\mathbf{y}_{(l),w,h,c}$ have zero derivatives.

For those input elements $\mathbf{x}^{(\mathbf{i})}_{w,h}$ that do contribute to $\mathbf{y}_{(l),w,h,c}$ the various methods compute slight variations of the true derivative. Some examples

- back-propagate only *positive* intermediate values
    - the "derivative" identifies only strong positive activations
- ignore the channel dimension by back propagating only the maximum (across channels) of the gradient
    - so identifies influential elements of the input but can't associate it with a partciluar concept ?

It is not so important to understand the details as the concept: identificaton of the elements of input $\mathbf{x}^{(\mathbf{i})}$ that most affect the feature map being probed.

Deconvolution and Maximally Activating Patches

The Zeiler and Fergus paper uses a form of guided back propagation called *de-convolution* (also know as Convolution Transpose). It's main features

- ignores signs of activations
    - so looks for "strong" signa, either positive or negative
    - it is able to go "backwards" through max-pooling layers by recording how the forward pass flowed in "switches"

What Deconvolution will do is find the sub-areas (called "patches"_ of the input that influence a feature map.

- Similar to Occlusion in that it identifes sub-areas of the input
- Different in that it starts at the feature map and works backwards
    - rather than starting at the input and working forwards.

For each layer beyond the first, Zeiler and Fergus show the 9 patches that maximally activate a feature map. That is

- a feature map is summarized by the largest absolute value
- we find the $N = 9$ inputs responsible for the largest summary of the chosen feature map
- we identify the sub-areas (patches) of these image

What we find is

- closest to the input, where the receptive field is smallest, primitive geometric features matched
    - lines
    - shades of color
- layer $(l+1)$ combines features of layer l into increasingly complex templates for matching
    - edges combined into corners; corners into squares
- at higher layers, "concepts" start to emerge
    - dog face
    - text
- as the layers get closer to the classifier "head", the templates become more specialized for the specific task
    - this has implications for Transfer Learning
        - too close to the classifier results in task-specific features that may not transfer as well as shallower features

From Figure 2. Best viewed under magnification. (we have intentionally scaled the right image to be larger in order to better see its elements)

Here are the filters for the first layer, and the patches that activate them.

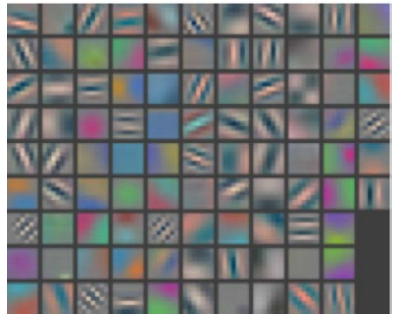
Note: for each filter, the 9 most activating patches are shown.

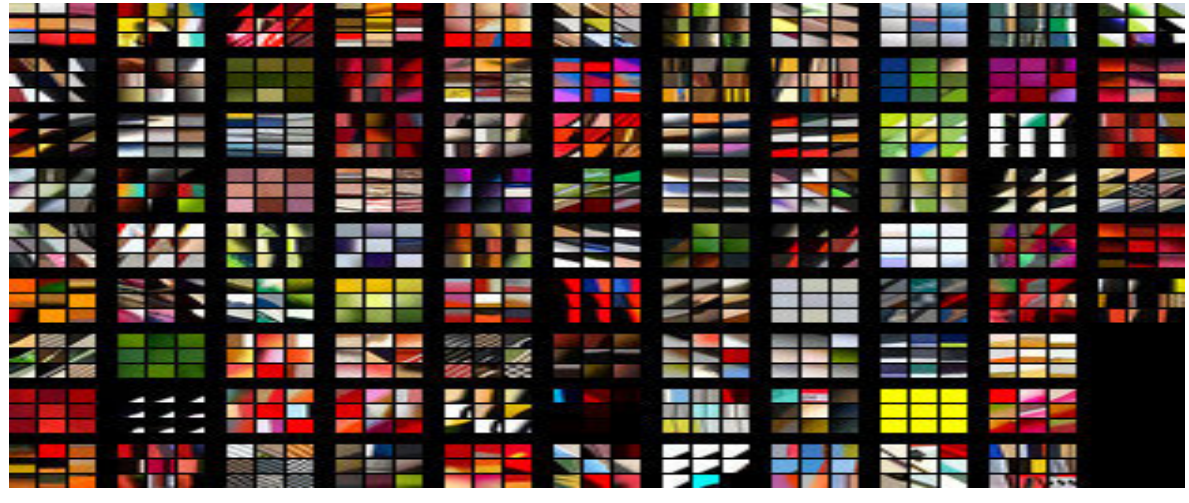
So each grid element on the left corresponds to a a grid element on the right organized as 3×3 patches.

For example, the filter at row 7, column 4 (on the left) seems to respond to checked patches (on the right).

Layer 1 filters

Filter **Strongly activating image patch for each filter**





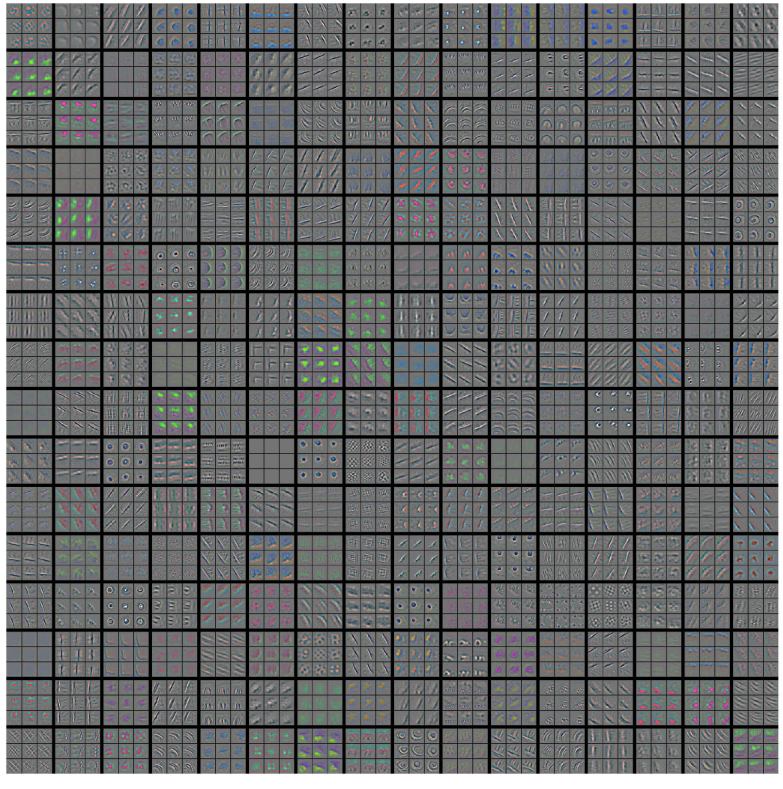
From Figure 2. Best viewed under magnification.

Layer 3.

Left: Each grid element are the top 9 activations (organized as a 3×3 grid) of some feature map, projected to input space.

Right: Each grid element is the patch of the activating image.

Layer 3 feature map projected to image, for multipe filters



Strongly activating image patch for each filter

Magnified view of layer 3 maximally activating patches.

Input independent methods

All of the preceding methods were specific to a particular input $\mathbf{x}^{(i)}$.

We now describe a method independent of the input.

The objective will be to construct a synthetic input $\mathbf{x}'$ that maximizes the value of the activation $\mathbf{y}_{(l),w,h,c}$ of a chosen feature map.

One can then try to interpret the activation $\mathbf{y}_{(l),w,h,c}$ (or a summary of the activations in a layer l) as attempting to match the synthetic input. Again, if the synthetic input is readily identifiable, we can attempt to infer that the layer is searching for this pattern.

Final word on gradient ascent, with constraints: Cool cost functions

Neural style transfer

Further exploration

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

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

```
Done
```