Chapter 1

Visualization

1.1 t-SNE visualization

Embed high-dimensional points so that locally, pairwise distances are conserved i.e. similar things end up in similar places. Dissimilar things end up wherever. Pronounced "tisni".

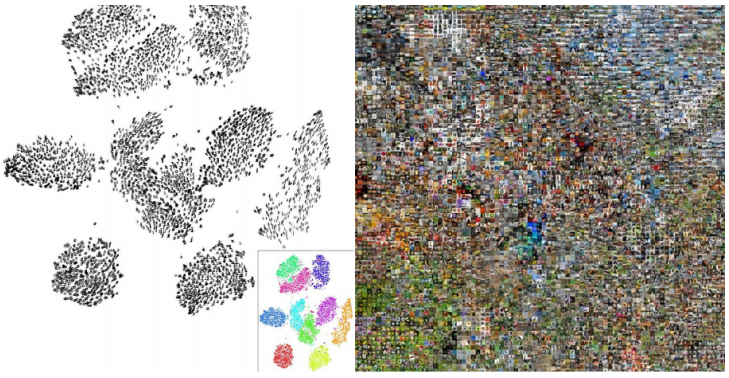

FIGURE 1.1: **Left**: Example embedding of MNIST digits (0-9) in 2D, **Right**: CIFAR-10 representation warped in a square shape

More CIFAR-10 examples here

1.2 Occlusions

The idea is to create a hit map of the output probability of the CNN of the image in the left corresponding to the correct label as you slide an occlusion across the image.

This helps you understand what are the regions which contains the most valuable features, for the CNN, in the image.

For example, in the first row, we can observe that the probability of the left image being a Pomeranian decreases when we cover its face.

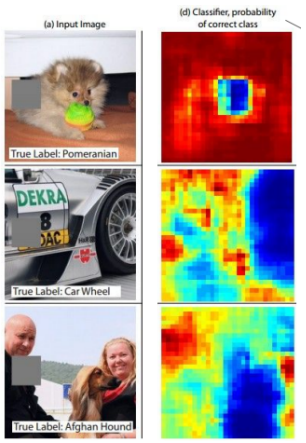

FIGURE 1.2: Visualize with occlusions

1.3 Visualizing Activations

How can we compute the gradient of any arbitrary neuron in the network w.r.t. the image?

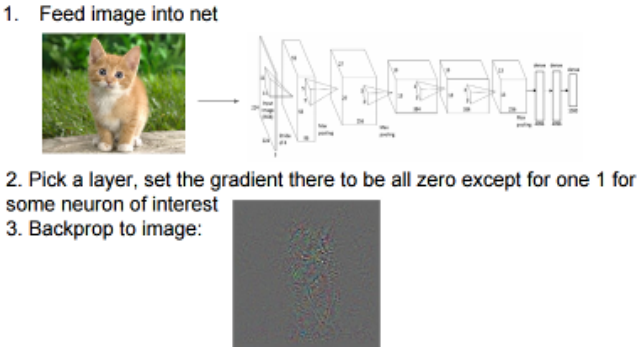

FIGURE 1.3: Visualize with occlusions

The output image gradients is not that good to understand what is going on. We will now see two strategies to obtain "better looking" gradients: Deconv approaches and

1.4 Deconv approaches

We are trying to see what parts of the input image are exciting a specific neuron. The idea to get a better looking gradient image is only to take into account the positive gradients. In this way we remove the noise caused by mixing positive and negative gradients.

The ReLU layer only backprop the gradient of the activations that where positive at the forward pass.

With Deconv, we must hack this behavior and change it so the ReLU layers only backprop positive gradients and do not care if the forward activations where positive or negative.

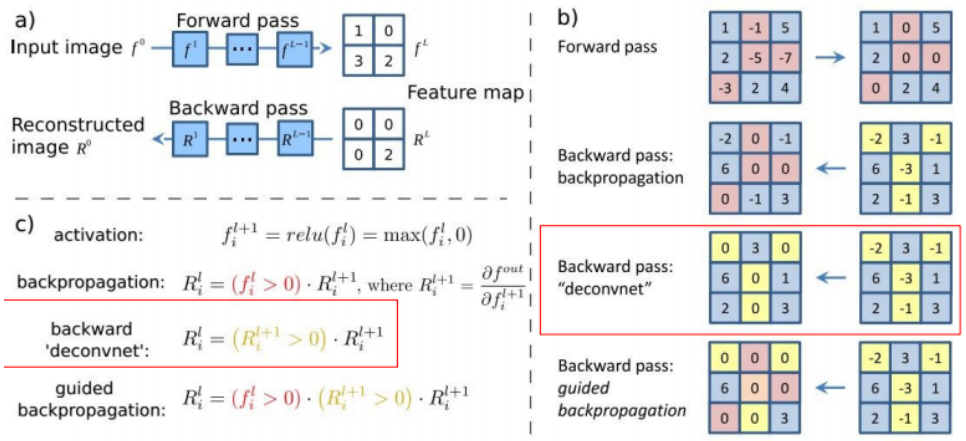

FIGURE 1.4: Deconv approaches

Now we can show what excites a specific neuron. The figures below show a grid of neurons at each layer. For each neuron we can see the image that arrived to that neuron and what parts of this image excited the neuron. So the deeper the neuron the bigger the area of the original image it observes. Also, it can be seen that the deeper the neuron the higher feature representation.

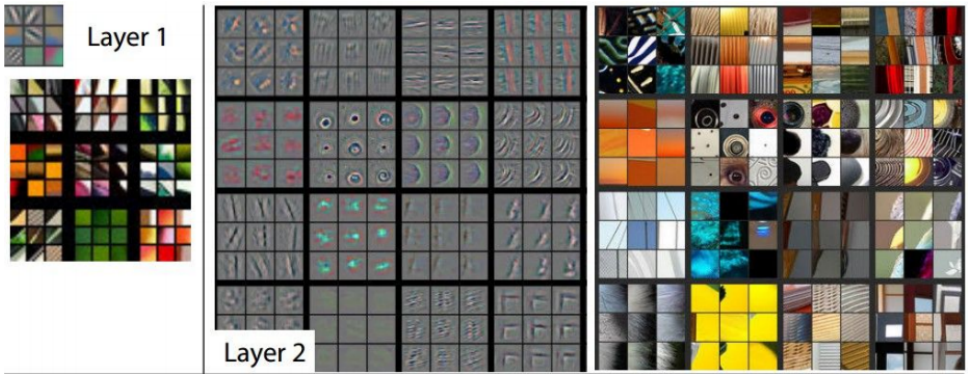

FIGURE 1.5: Deconv approaches layers 1 and 2

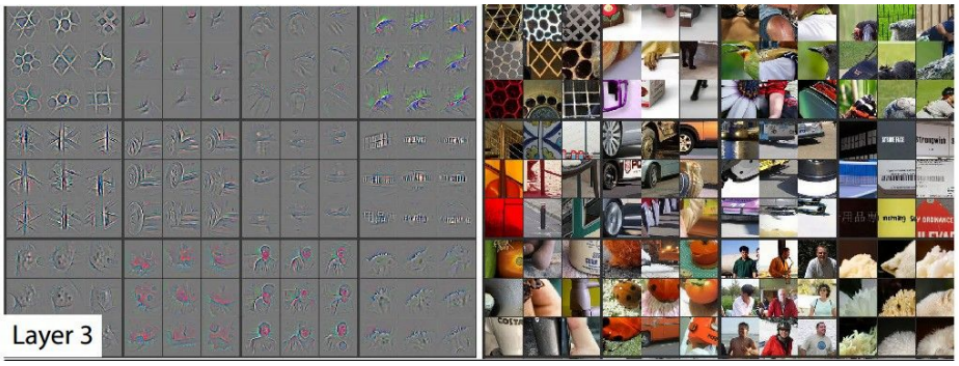

FIGURE 1.6: Deconv approaches layer 3

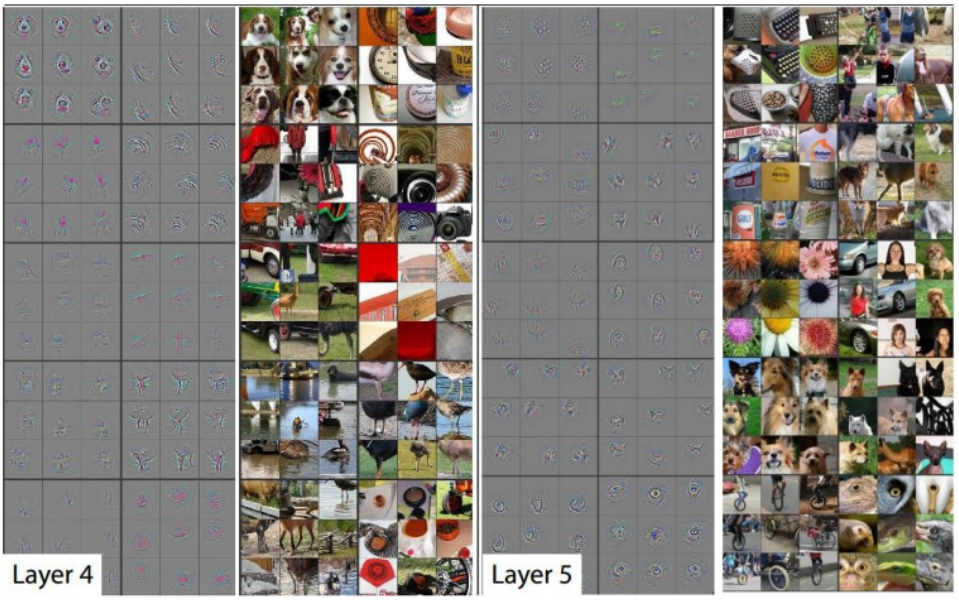

FIGURE 1.7: Deconv approaches layers 4 and 5

1.5 Optimization to image

How can we find an image that maximizes some class score?

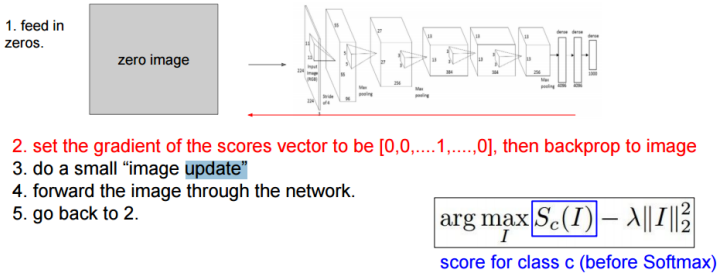

FIGURE 1.8: Optimize an image to maximize the score of a specific class

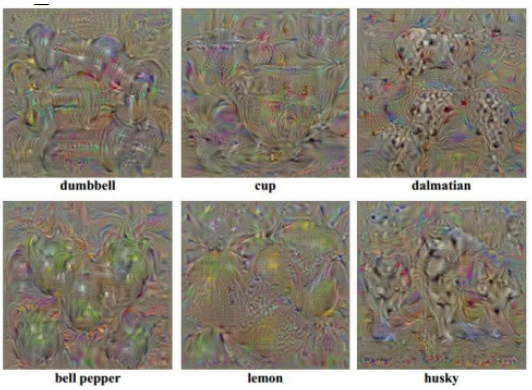

FIGURE 1.9: Optimize an image to maximize the score of a specific class output

1.6 Visualizing Data Gradients

1. Feed an input image
2. Set the gradient of the scores vector be [0,0,..,0,1,0,...0] where 1 is the ground-truth label class of the image
3. Backprop to the image
4. For each pixel of the image store the max gradient value across all the channels
5. Plot the gradients

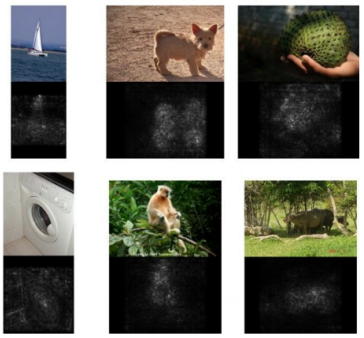

FIGURE 1.10: Data gradients output

The interpretation of the results are that the staff that is black is not influencing the score of the image. So you can wiggle them and the score will not change. So at the end it is telling you the are of the image that is influencing the network.

1.7 DeepDream

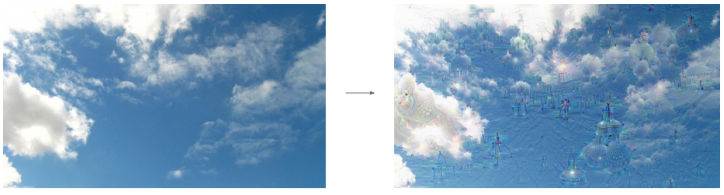

FIGURE 1.11: DeepDream output

DeepDream link

It normally hallucinate a lot of dogs because there are a lot of dog samples in ImageNet (because there are a lot of different dog classes).

At the end, the only thing DeepDream is doing is: given a layer immediately after a ReLU layer, it sets the gradients equal to the activation values. So if you iteratively update the image with the gradients, you are modifying the image to increase whatever it is exiting the selected layer.

```
def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
              jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst)  # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

DeepDream: set dx = x :)

jitter regularizer

"image update"

FIGURE 1.12: DeepDream code

In other words, DeepDream modifies the image in a way that "boosts all activations. This creates a feedback loop: e.g. any slightly detected dog face will be made more and more dog like over time.

1.8 DeepArt

Explained at cs231n lecture 9

1.9 Adversarial data - Fooling networks

It is very simple to fool a CNN. Images are super high dimensional objects (one dimension for each pixel). The training images are constrained to a small manifold and we are training ConvNets over it. The trained ConvNet works extremely well inside that manifold, but outside of it is complete random.

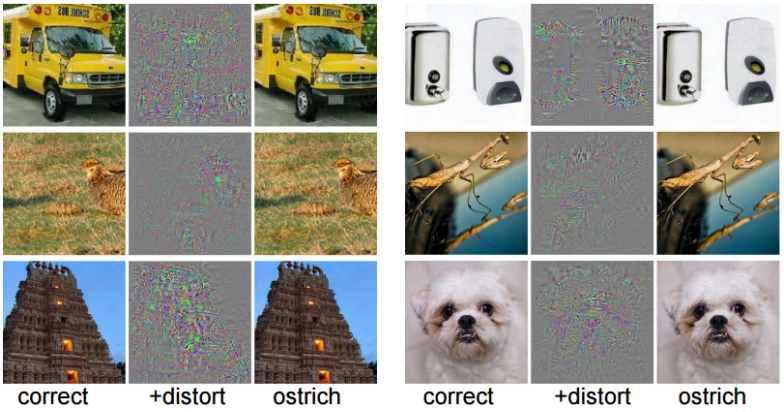

FIGURE 1.13: Adversarial data examples

The think is that during the forward pass we are applying dot product over all the dimensions of the data (one per pixel) so if we change the input a tiny bit but all in the correct way, the dot product output is going to change a lot.

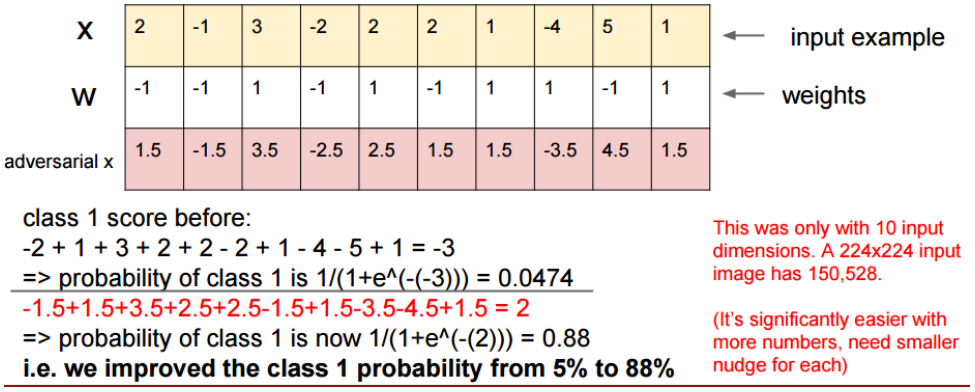

FIGURE 1.14: Adversarial data case example

This is not a problem of deep learning, its is a problem in learning in general. There are some ways of decreasing this effect. One is to train also with adversarial examples; it will be more robust to them but the classifier accuracy will decrease. Another one is to instead of classifying the hole image, classify chunks.