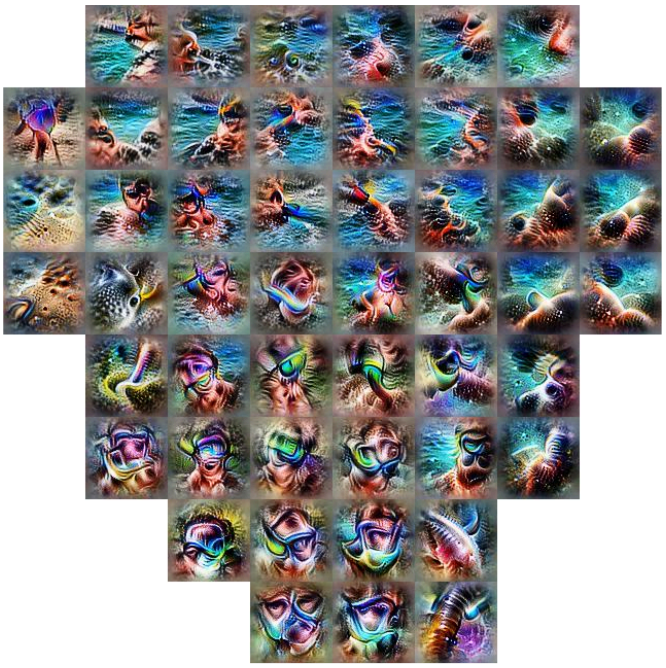
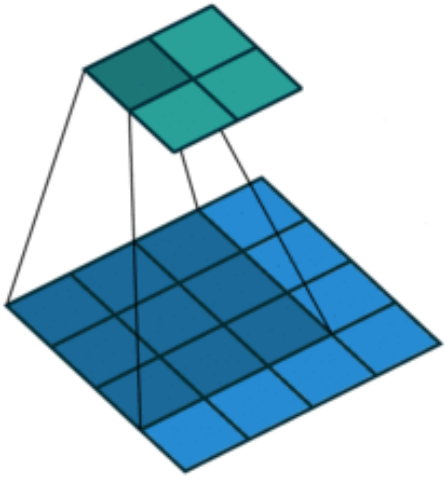
I ❤️ CNNs

KurwAI Articles 🦫

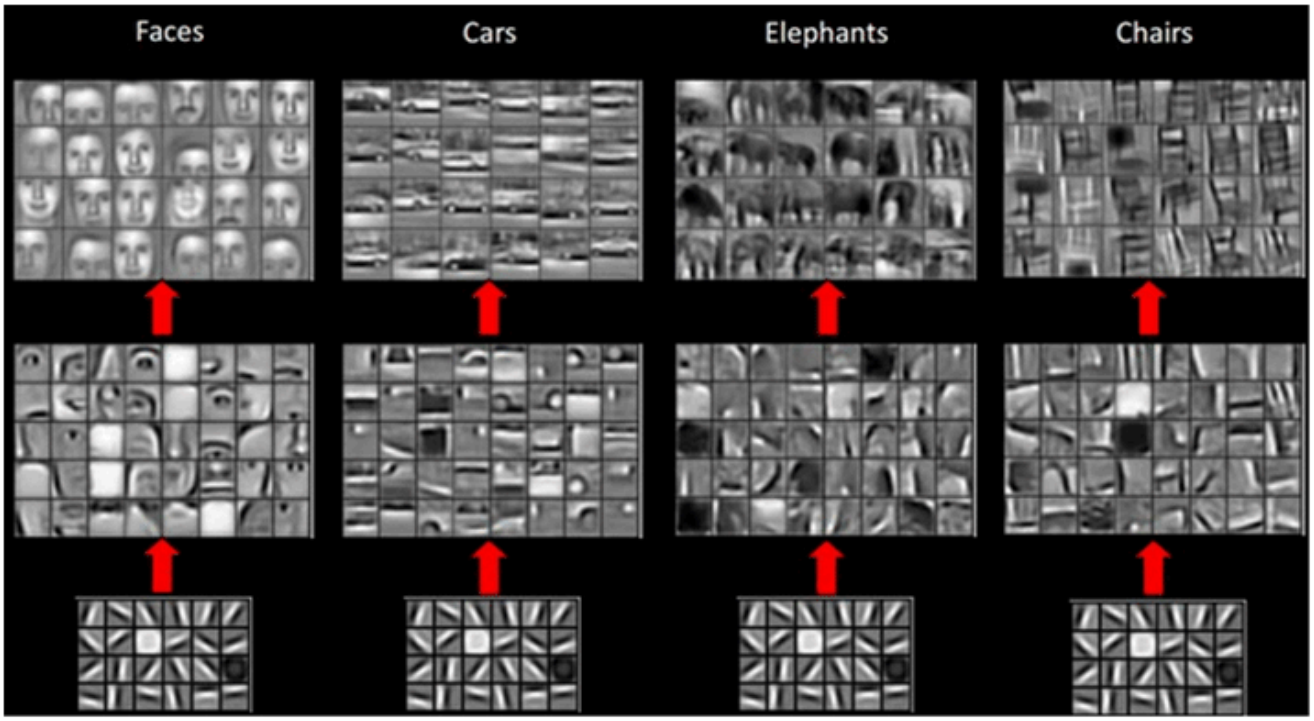


Casual CNN understanding

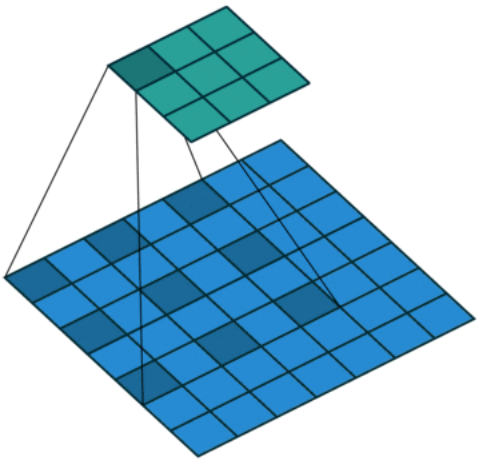
Typically CNNs are the next follow-up topics after Feed-Forward Neural Networks.
When student is entertained with tabular data and flattened MNIST tutors reveal magical world of CNNs, which feel completely different to Feed-Forward nets.
Instead of processing input entirely it process patches of image gliding special kernel matrix and calculating weighted sum of each patch.



We can think of it as of scanning over image for some patterns. On early layers we search for edges, on last layers - for objects.
This progression comes from **receptive field** expressive power increase deeper into the network. "Understanding Deep Learning" book (awesome book btw) revealed it to me.
I haven't thought of it before and couldn't understand optimism of "pattern complexity growth".



The thing is result of Convolutional Layer is a tensor, where **each scalar is a weighted sum of group of pixels**. In this way neural network pattern complexity gradually increases.
This idea is quite important and we'll see it later.
Convolutional layers have many hyperparameters, but the one, which attracted my attention was **Dilation rate**. Bigger kernels capture bigger features, but require more parameters as a result.
The idea is simple - to intersperse kernel with zeros. Dilation rate is a number of zeros between parameters - 1.



Dilation Rate 2

We Lose finer details, but the good thing is we don't need them in capturing bigger pattern.

Convolutional Neural Networks hide a lot of interesting and exciting things, but sadly it's easy to miss them.
Most of the beginners (including me back then) are excited to build a couple of classifiers on Cifar-10 and move on to something else (like Generative Models), but there are awesome things to capture **inside a CNN!**

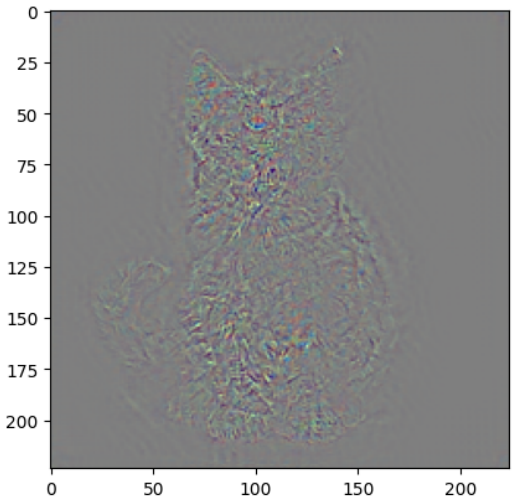
Inside a CNN

We trust CNNs because of their good performance, because of the theoretical basis we have (this gliding feature extractor actually sounds promising and tackles many of Feed-Forward net image processing problems)
However, we don't really know what's there, how CNN sees the world (and does it even see in traditional sense).
To unpack this mystery and interpret CNNs researchers found many useful methods, the most exciting of which we'll see today!
For deeper understanding of particular technique I recommend to watch the lecture from CS231n course on YouTube.
Also, All the code is available in my repo: https://github.com/Cheslaff/KurwAI-archive/tree/main/Guides

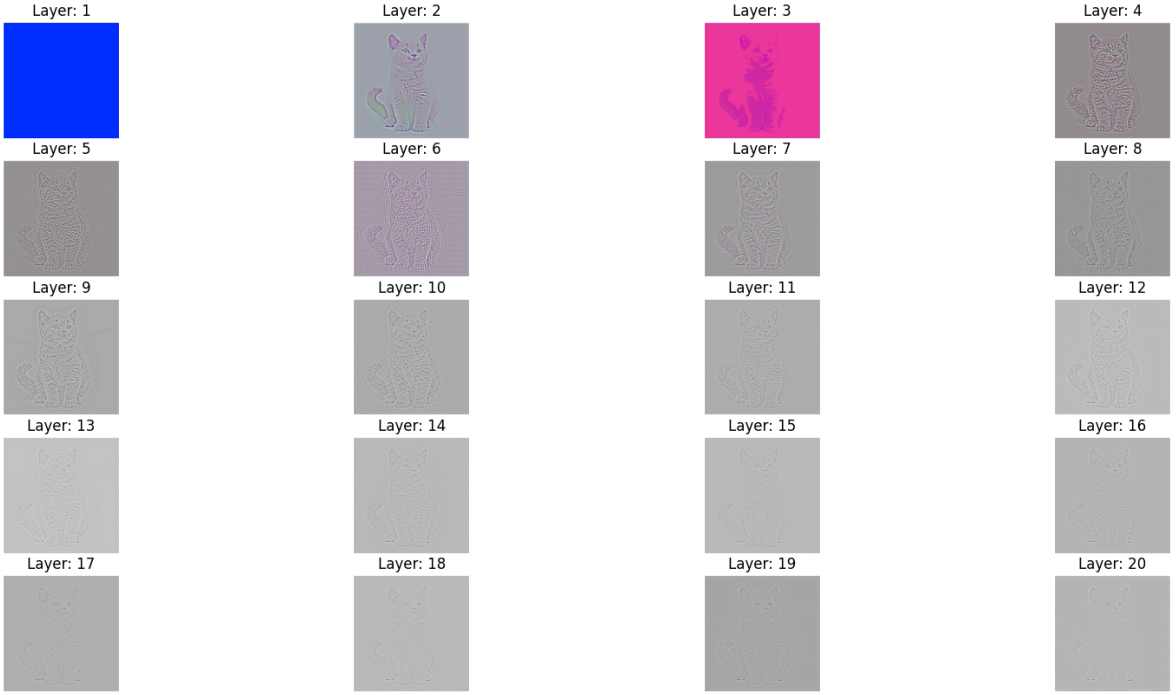
Deconvolution.

This is a good method to start with. It's expressive enough, simple and entertaining.
Given image:

We'd like to know what parts of it are important for particular neuron/layer.
To do it we could do forward propagation up to this layer and backpropagate from there.
Here's an example for the random neuron from the last layer of ResNet50 trained on ImageNet.



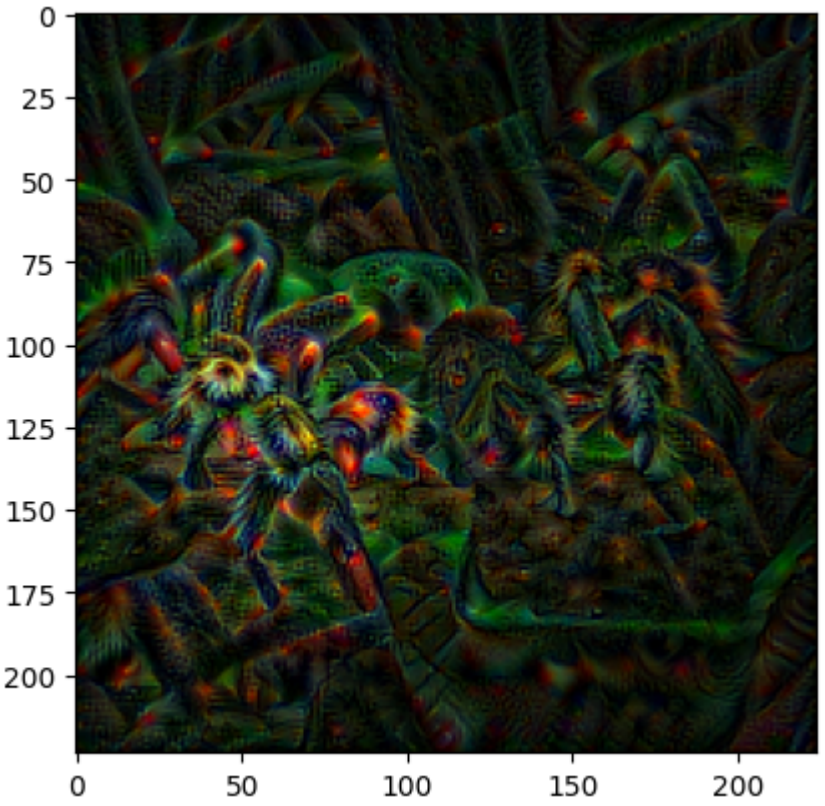
Oh this looks good! We removed all unimportant gradients and here's the beautiful result!
Here are the results of such visualization for different layers (tbh, not sure they're correct 😅)



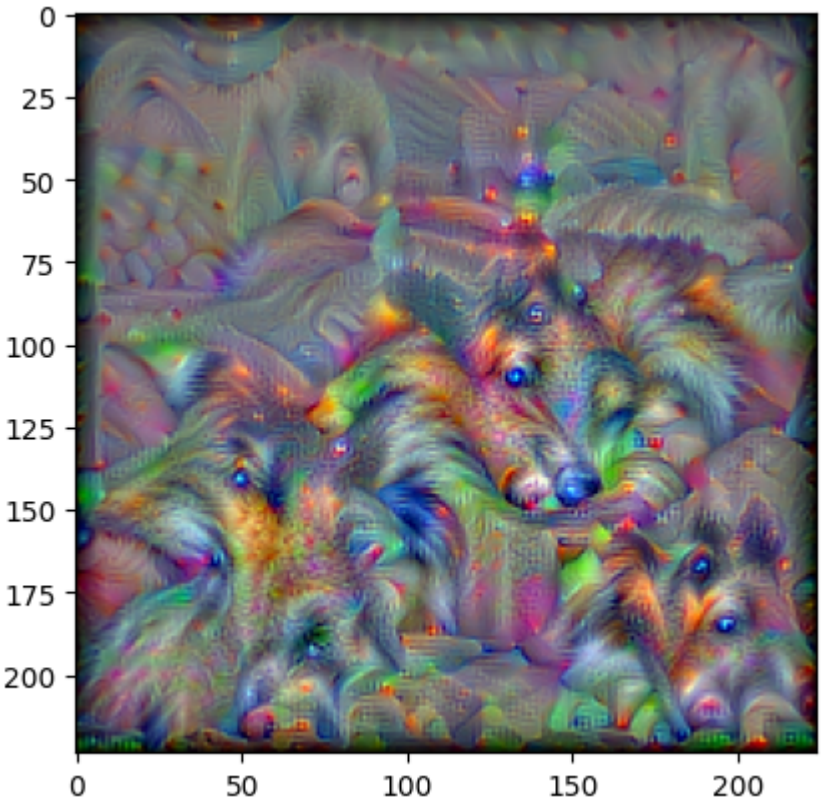
Visualizations Link.

Synthetic Input for neuron maximization

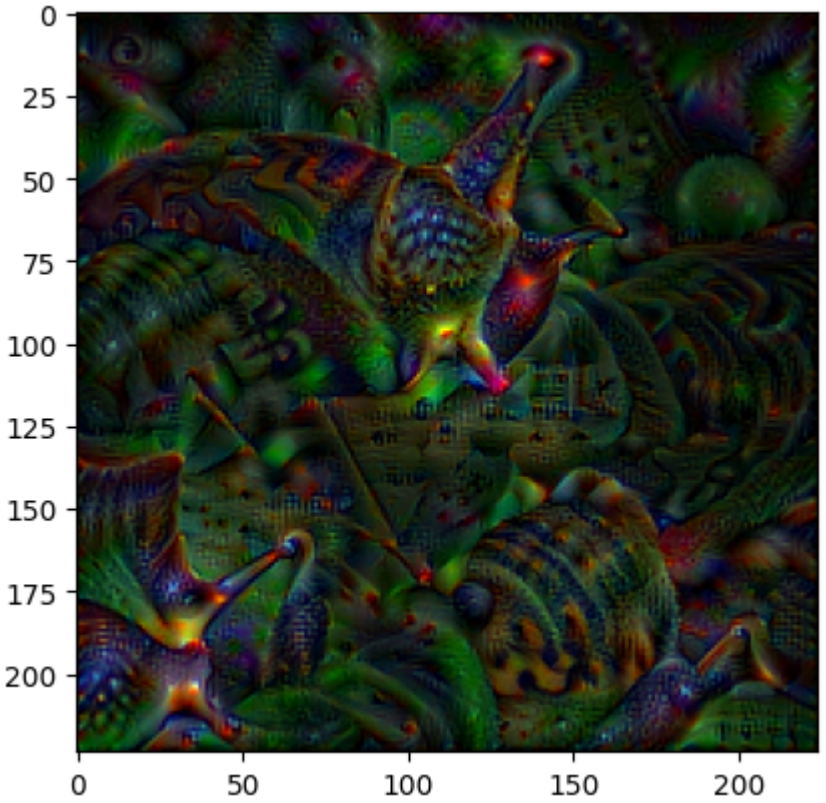
Another way is to start with random noise as input, require gradient for it and gradually iteratively update this image to maximize particular neuron or layer.
This one, to me personally, is more interesting visualization. Let's see some of the cool examples I achieved with it!



Tarantula class maximized from zeros (all black)



Collie class maximized from rand values



Snail class maximized from zeros (all black)

Looking at this now I have a feeling that I screwed up normalization, this is the reason for these weird colors (sorry).
Overall, this is an exciting visualization technique.
Once again, if you want to see the code and experiment yourself (it's quite easy), here's a notebook link on GitHub:
GitHub.
CoLab

Deep Dream

This is the trippiest technique of them all!
Deep Dream went viral somewhere in 2016 when released.
The idea is pretty simmilar to the previous algorithm, but goes beyond that.
Now we send a picture as input (whatever we want) and let network dream on the specified layer.
We apply .norm() on this layer and run backward from there.
Network changes the image in the way it wants to maximize this overall norm.
This one is insanely cool, because it takes into account input picture.
I really like how TensorFlow describe it in their tutorial:

> DeepDream is an experiment that visualizes the patterns learned by a neural network. Similar to when a child watches clouds and tries to interpret random shapes, DeepDream over-interprets and enhances the patterns it sees in an image.

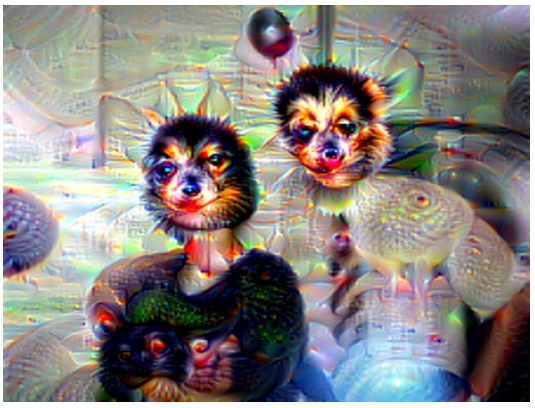
Sounds cool. Here's the image I passed as input:

And here's a simple deep dream (vanilla) result after 100 iterations.

Looks cool already, but we have a couple of problems with it:

- Same Granularity (same size of dreams all over the image)
- Low quality of the image
- We can't really control how sane our dream is and how diverce are the features.

Thankfully we can run it multiple times (multiple times) with growing image size (octave_scale) (code available)
Here is the result with 3 octaves and octave_scale of 1.4:



Here's another visualization, but I don't remember what hyperparameters I used for this one:

When playing with DeepDream you can see a lot of dog faces and fish parts appearing here and there (like here etc.)
It's due to ImageNet data has a lot of dog classes in it, so our neural network really likes dogs (so much that dreams about them 😅)
Here are the results for all layers in Network:
No Octaves:

Layer: 1

Layer: 2

Layer: 3

Layer: 4

Layer: 5

Layer: 6

Layer: 7

Layer: 8

Layer: 9

Layer: 10

Layer: 11

Layer: 12

Layer: 13

Layer: 14

Layer: 15

Layer: 16

There's one last experiment to run, it's now not about feature visualization (not directly)

Neural Style Transfer (NST)

Sadly not all of us are good artists, but let's see if our network is!
Neural Style Transfer takes 2 images as input:
content image - this is an image with object style of which we want to change
style image - this is an image with artstyle we want to apply to content image

We pass these 2 images through pretrained frozen CNN and take outputs from one particular layer (or from group of layers).

Then we clone content image and require gradient for it, to update it in training loop.
We want to preserve structure of the image (keep objects), but change style (make it simmilar to the style image, but only stylistically).
The first task is relativelly (MSE loss), but how to measure style simmilarity, how to extract style from the layer?
We'll extract them from Convolutional layers, each of which has the shape of (B, N, H, W) - batch (=1) x channels x height x width.

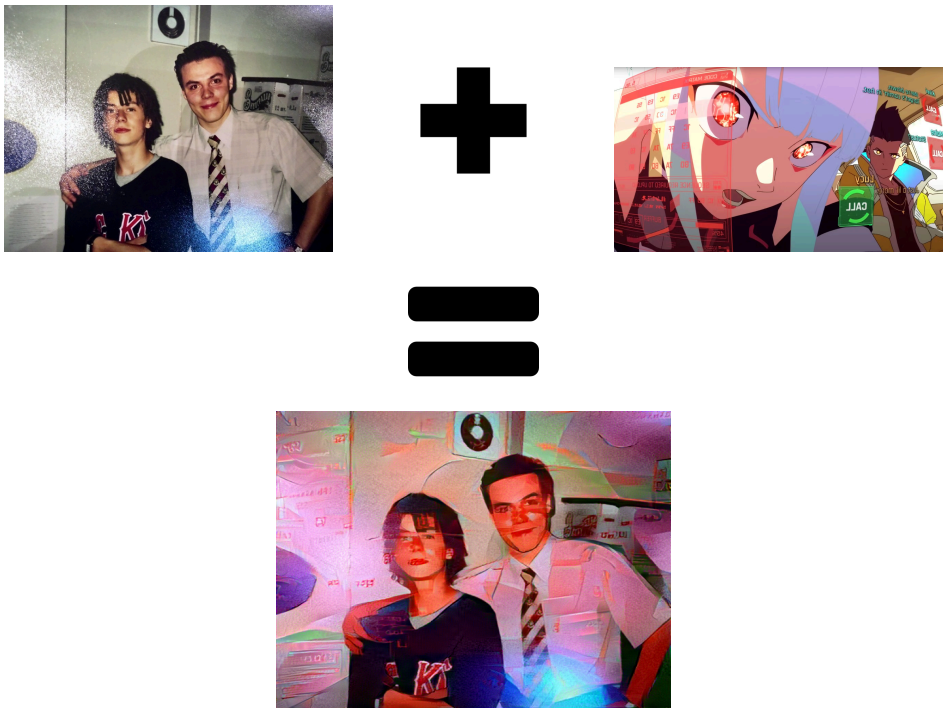
We'll transform it into matrix of shape $(B \times N, H \times W)$
Now important moment, we'll calculate gram matrix on this matrix F so that $G = F \cdot F^T$
We'll calculate same matrix for style image at this layer and calculate scaled MSE

$$\sum \frac{(G_{gen} - G_{style})^2}{(W \times N \times H \times W)}$$

Finally, we'll combine MSE (content loss) with Style Loss as a weighted sum (weights to prioritize style or content for flexibility).
Code is available here:
GitHub
CoLab

Here are awesome results I got!

And another one!



Why do I ❤️ CNNs?

For everything, for their efficiency, for their interpretability (as you could've seen in this article), for their stability and for the inspiration boost they give you when studying them!

Thanks For reading this article.
Links:

GitHub CNN Visualizations
CoLab CNN Visualizations

GitHub Style Transfer
CoLab Style Transfer

CS231n Visualizations overview Lecture by A. Karpathy

Don simon through the CNN verse!