*trocar E1 – ba
*PT – ~~corrido~~ layout ...

Convolutional Neural Networks

Until now, we simply discarded each image's spatial structure by flattening them into one dimensional vectors, feeding the MLP parameters. Prefrably, we would leverage our prior knowledge that nearby pixels are related to each other.

Invariance

Not be overly concerned with the precise location of image.

C. NN suitable for computer vision:
1. Translation invariance
2. Localization principle

CNNs are special family of NN that contain convolutional layers.

Convolutional kernel or filter or simply the layer's weights that are often a learnable parameter.

Convolution

$$(f * g)(x) = \int f(z)\, g(x - z)$$

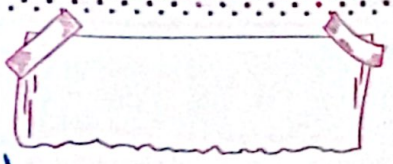
padding (outer), stride, pooling

input

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

kernel — output

My playlist today: #




LeNet

Consists of two parts:
(i) Convolutional encoder with two convolutional layers
(ii) Dense block with three fully connected layers

Basic units in each convolutional block:
- Convolutional layer
- Sigmoid activation function
- Average pooling operation

Note that while ReLU and max-pooling work better, these discoveries had not yet been made in the 1990s.

Each convolutional layer
- 5×5 kernel
- sigmoid activation function

These layers map spatially arranged inputs to a number of two dimensional feature maps, typically increasing the number of channels.

In order to pass output from the convolutional block to dense block, we must flatten each example in a minibatch

Deep Convolutional Neural Networks

Rather than training end-to-end (pixel to classification) systems, classical pipelines looked more like this:

1. Obtain an interesting dataset. In the early days, these datasets required expensive sensors (at the time, 1 megapixel images were state-of-art)

2. Preprocess the dataset with hand-crafted features based on some knowledge of optics, geometry, other analytic tools, and occasionally on the serendipitous discoveries of luckily graduate students

3. Feed the data through a standard set of feature extractors such as SIFT (Scale Invariant Feature Transform), the SURF (Speed up Robust Features) on any number of other hand-tuned pipeline

4. Dump the resulting representations into your favourite classifier, likely a linear model or a kernel method, to train a class

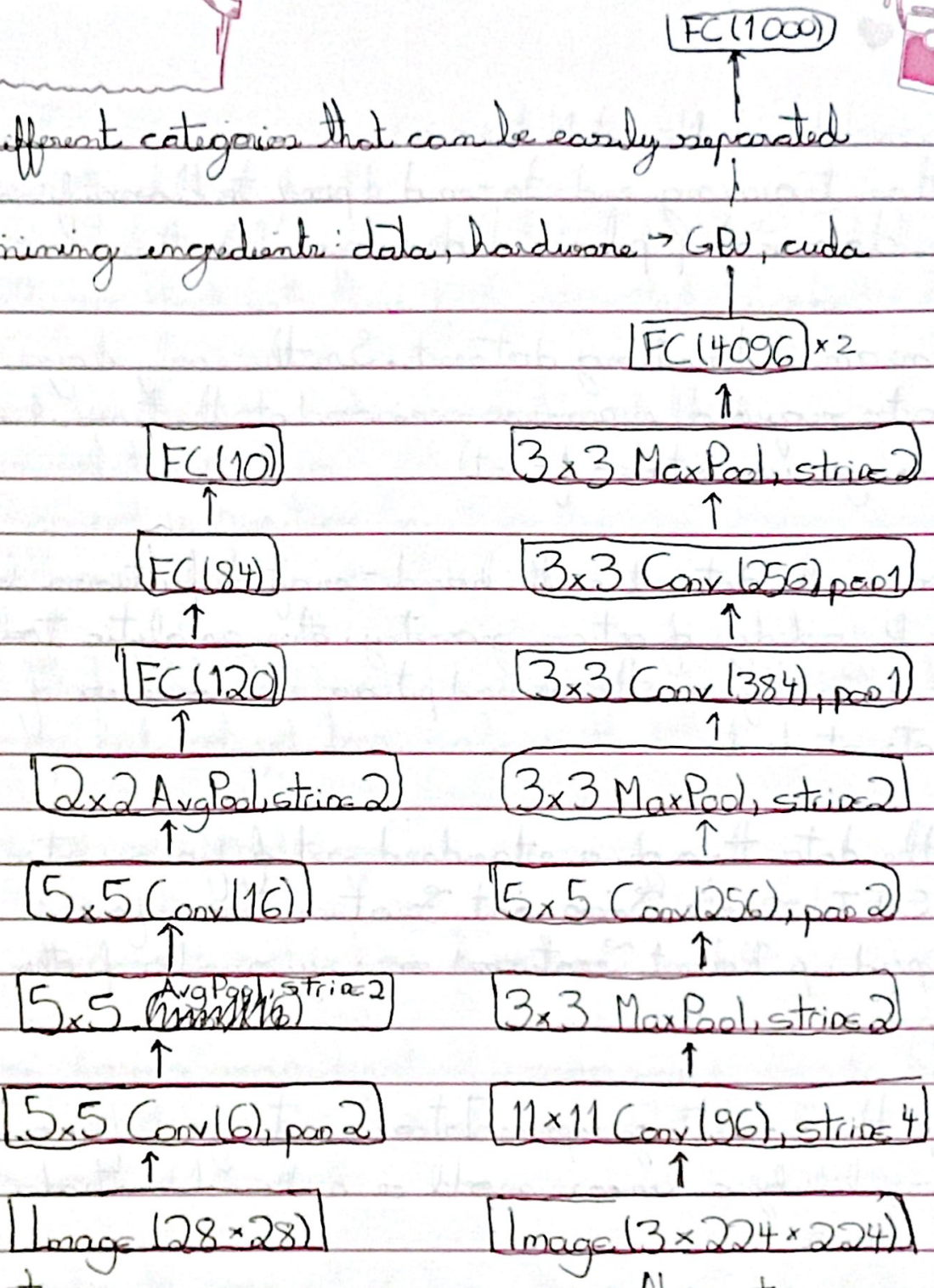
Features ought to be learned - Alexnet

Higher layers of the representation to represent layer structs: eyes, noses, Even larger, The final hidden state learns a compact representation of the image that summarizes its contents such that data belonging

My playlist today: #

to different categories that can be easily separated

Winning ingredients: data, hardware → GPU, cuda



Alexnet is much deeper than the comparatively small LeNet5. Second, Alexnet used ReLu instead of sigmoid as its activation function

Since most images in ImageNet are ten times higher and wider than MNIST images, objects in ImageNet data tend to occupy more pixels. Consequently, a larger convolution window is needed to capture the object. Moreover, AlexNet has ten times more convolution channels than LeNet.

AlexNet controls the model complexity of the full connected layer by dropout while LeNet only uses weight decay.

To augment the data even further, the training loop of AlexNet added a great deal of image augmentation, such as flipping, clipping, and color changes. This makes the model more robust and larger sample size effectively reduces overfitting.

Networks using blocks (VGG)

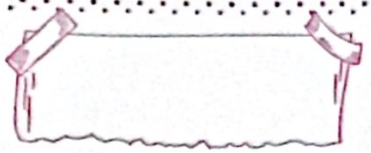
The idea of using blocks first emerged from the Visual Geometry Group at Oxford University, in their VGG network

VGG blocks

The basic building block of a classic CNN is a sequence of the following:
i) A convolutional layer with padding to maintain resolution
ii) A nonlinearity such as a ReLU
iii) A pooling layer such as max

One VGG block consists a sequence of convolutional layers, followed by a maximum layer for spatial down-sampling

My playlist today. #



FC (1000)

FC (4096)

FC (4096)

3x3 MaxPool, stride 2

3x3 Conv (384), pad 1

3x3 Conv (384), pad 1

3x3 MaxPool, stride 2

5x5 Conv (256), pad 2

3x3 Maxpool, stride 2

11x11 Conv (96), stride 4

Alexnet

2x2 MaxPool, stride 2

3x3 Conv, pad 1

3x3 Conv, pad 1

VGG block

FC (4096)

FC (4096)

FC (4096)

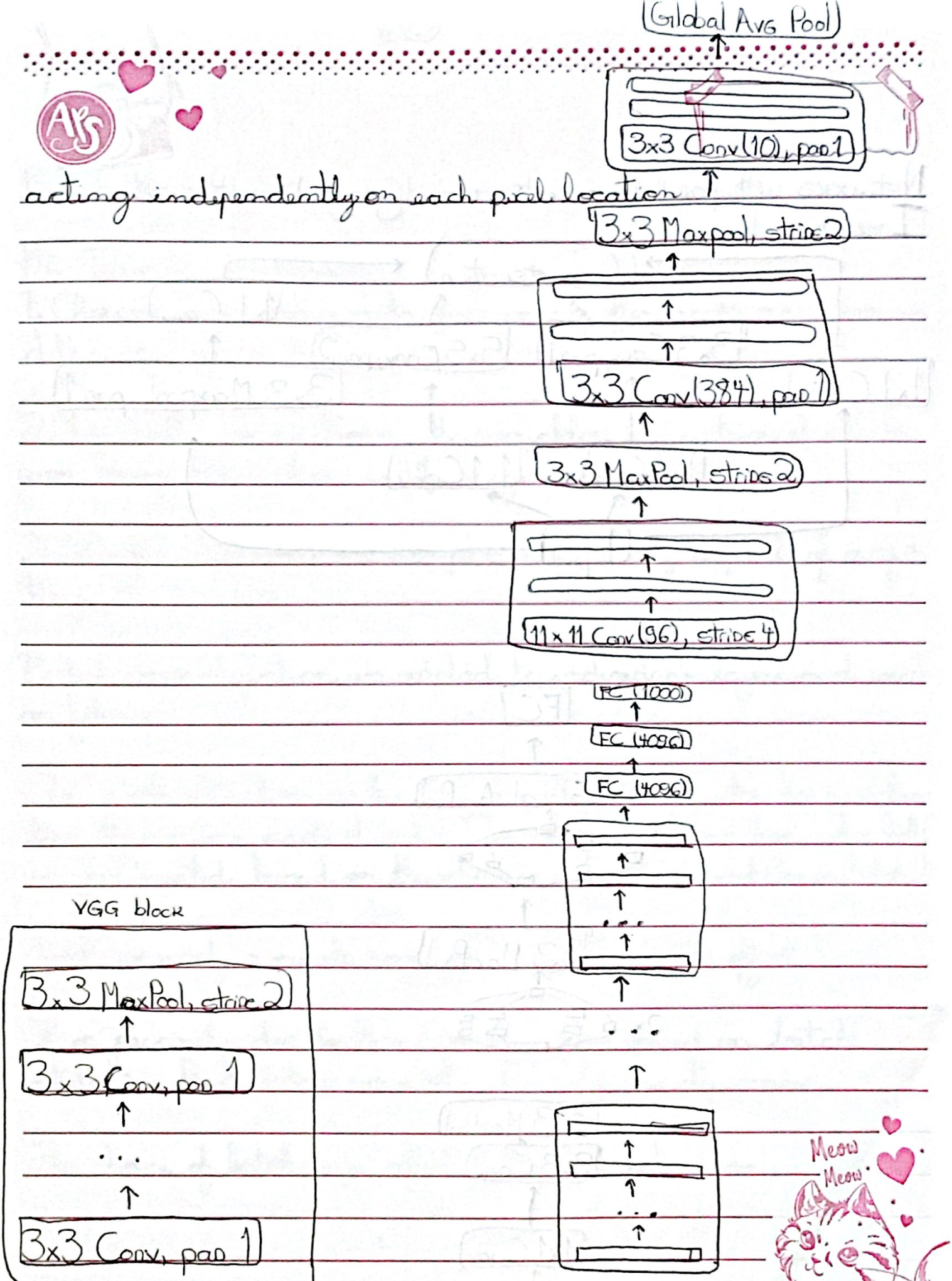
Network in Network (NiN)

Use an MLP on the channels for each pixel separately

NiN block

If we tie the weight across each spatial location, we could think of this as a 1×1 convolutional layer or as a fully-connected layer

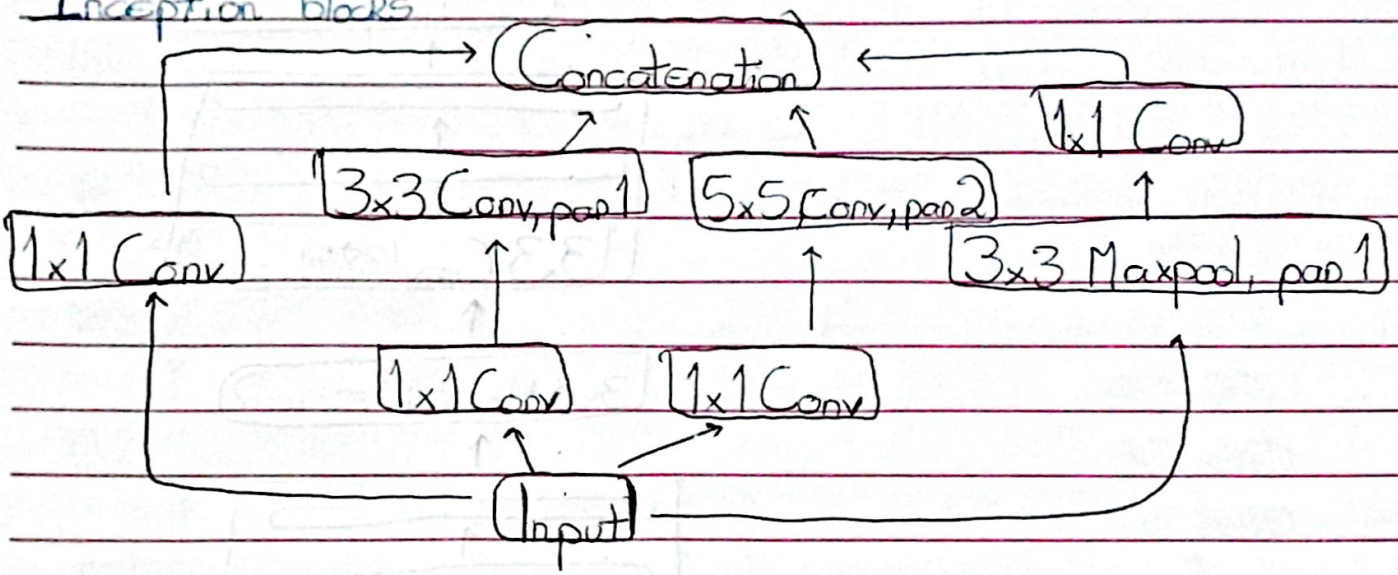
acting independently on each pixel location

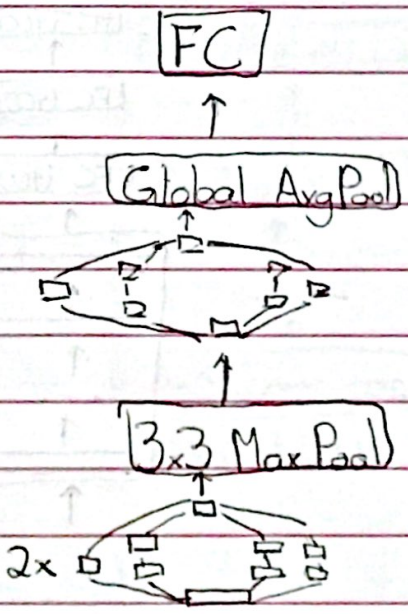


My playlist today: #

Networks with parallel concatenation (GoogLeNet)

Inception blocks





Batch Normalization

ML Training challenges

1. Choices regarding data processing often make an enormous difference in results.

2. Typically as we train, the variables in intermediate layers may take values with varying magnitude.

3. Deeper networks are complex and easily capable of overfitting.

Batch normalization is applied to individual layer and work as follows.

In each iteration, we first normalize the inputs by subtracting their mean and dividing by standard deviation, where both are estimated based on the statistics of the current minibatch.

Next, we apply a scale coefficient and a scale offset.

It is precisely due to this normalization based on batch statistics that batch normalization derives its name.

The choice of batch size affects batch normalization.

My playlist today. #

Formally, denoting by $x \in B$ an input to batch normalization (BN) that is from a minibatch B

$$BN(x) = \gamma \odot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta \qquad \hat{\mu}_B = \text{sample mean} \qquad \hat{\sigma}_B = \text{sample std}$$

$$\hat{\sigma}_B^2 = \frac{1}{|B|} \sum_{x \in B} (x - \hat{\mu}_B)^2 + \epsilon \qquad \hat{\mu}_B = \frac{\sum_{x \in B} x}{|B|}$$

Note that we add a small constant $\epsilon > 0$ to the variance estimate to ensure that we never attempt to divide by 0. Yes.

Batch normalization layers function differently in training mode (normalizing by minibatch statistics) and in prediction mode (normalizing by dataset statistics)

Residual Networks (ResNet)
Every additional layer should more easily contain the identity function as one of its elements. These considerations are

Densely Connected Networks (DenseNet)
Taylor expansion for functions

Given a point $x = 0$

$$f(0) + f'(0)x + \frac{f''(0)x^3}{3!} +$$

The key point is that decomposes functions into

$$f(x) = x + g(x)$$

My playlist today: #