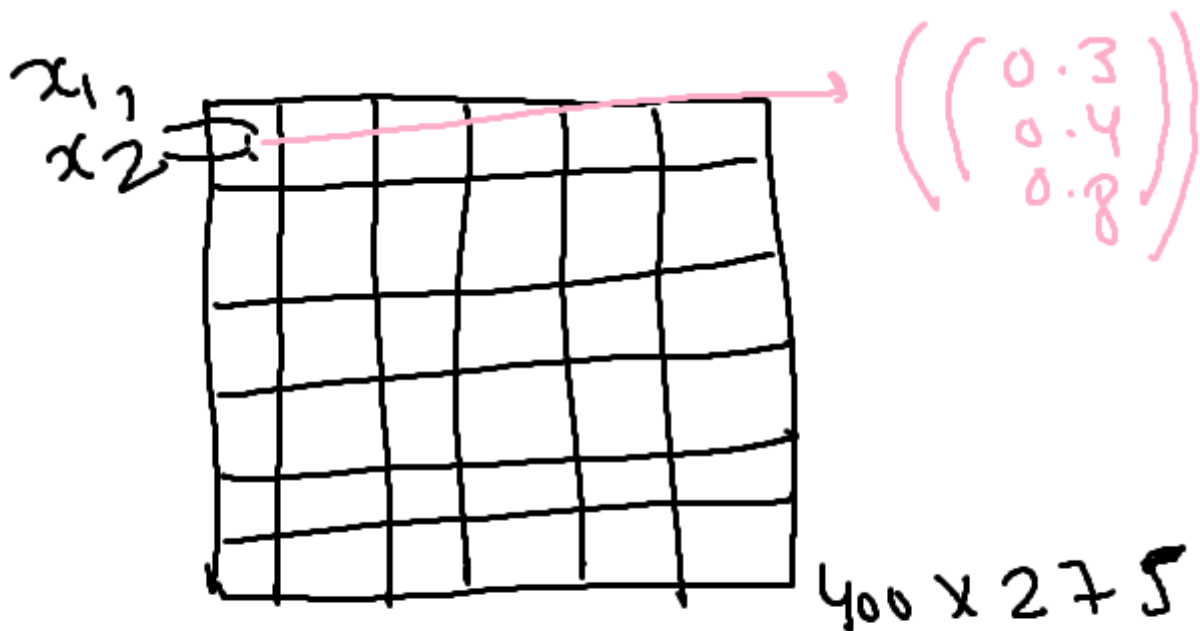


Using periodic activation functions for implicit representation



Let's say this is a 400 * 275 resolution image. which means that for channel of each RGB (red, green, blue) color it has a grid of 400 * 275 pixels. So, for this image there is a 2D space coordinate (R^2) [x,y coordinates] and 3D RGB Space.

*For training the neural network a function let's say

$$\Phi$$

Φ , is learned which can map coordinates of the image to the RGB values. *

Implicit Representation

Neural network needs to be trained per image for the implicit representation. If we get the trained neural network parameters it is equivalent to get the images itself. So, the entire dataset is the single image only.

Advantages

We get the continuous representation of the image, which means we get the continuous mapping of pixels location to pixel values.

This is the property which is differentiable which helps in not only learning the image but learning its gradient as well.

SINE ACTIVATION FUNCTION

Instead of other popular activation functions the SINE function $\sin(x)$ is used with the same architecture of 5 layer neural network with using a slightly different approach for initialization.

Introduction

In this paper , the signals are represented as functions mapping from their co-ordinates to thier values and they did it very well using the new models called **SIRENs**

SIRENs are the neural network that use sine waves as their non-linearity.

Equation 1

$$F(\mathbf{x}, \Phi, \nabla_{\mathbf{x}}\Phi, \nabla_{\mathbf{x}}^2\Phi, \dots) = 0, \quad \Phi : \mathbf{x} \mapsto \Phi(\mathbf{x}).$$

Let's understand this equation :

$$Input = x$$

$$Output = \phi$$

Equation 2

$$\phi : x \mapsto \phi(x)$$

The above function (equation2) maps input to output, and this is what we will call a neural or a implicit representation.

x
could be a datapoint and it is mapping it to a label

$$\phi(x)$$

In case of images :

x (input) could be a latent vector , and then we have a neural network

$$\phi(x)$$

mapping the latent vector to an image

generally , the latent vector is considered as the continous representation , but here the function

$$\phi(x)$$

itself is the continuous representation of the image.

In case of images , we use to learn the function

$$\phi$$

from the data such that if i give the one particular vector it will give the one particular image. then plug another vector to get the another image and the function always remains the same.

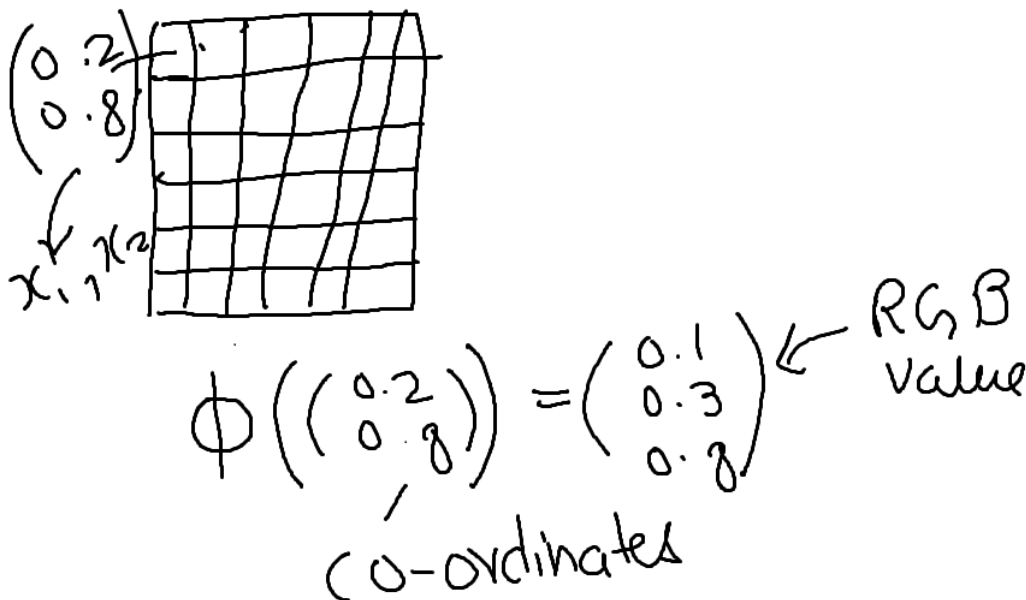
but, here it is one function per image, it's like the function is the image.

How a function can be the image ?

As , image is made up of pixels and it has a x and y co-ordinate.(x1,x2). And each pixel also has a color value which is 3D RGB color value. So image can be seen as a function

$$\phi$$

from co-ordinates to pixel values.



$$\phi\left(\begin{pmatrix} 0.2 \\ 0.8 \end{pmatrix}\right) = \begin{pmatrix} 0.1 \\ 0.3 \\ 0.8 \end{pmatrix}$$

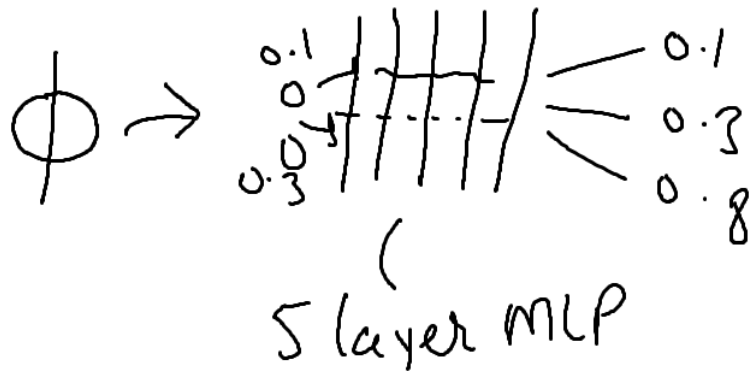
co-ordinates

RGB value

now goal is the make the function

$$\phi$$

a neural network which is 5 layer MLP which take the input (co-ordinates) which travels through 5 layer MPL which as the output gives the 3 RGB values. There will be one particular network per images which is trained to map input values (co-ordinates) to output values (RGB)



Why can't we have the image as a pixel values why do we need a function mapping co-ordinates to the pixels.

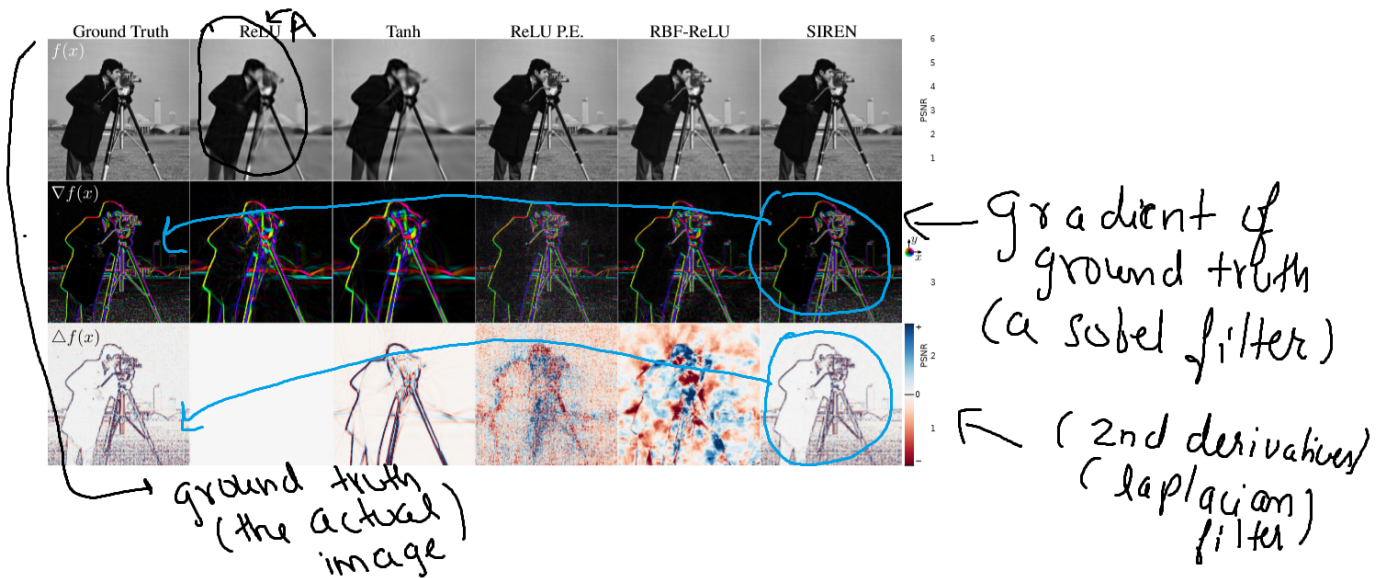
Advantage what we are getting here is the continuous representation, if we store the complete image we only know its value at each of the pixel location. but if we store image like this



we know its value at any continuous in-between location. the network can tell the pixel value at any location.

this turns out to be not really a machine learning problem but optimization problem, because all we need to do is to make the neural network match the input to output. there is not really a training or test set. Each pixel in the image is considered as a one data point. the way it is trained is by taking the different samples of mini-batch of

pixels.(same samples can be used many times). What at last we want is continuous representation of the image.



In image above the output A, is the network which tried the ReLU non-linearity, as it seems missing the lots of higher definitions in the image.

2nd grid is 1st derivative of the image which is sobel filter which is basically a edge detector.

3rd grid is the second derivative which is laplacian of the image.

Normally, if the implicit representation models the signal very well it should model it's derivative also very well.

These SIRENs are designed to not only match the signals but also to match it's derivatives as well.

SIREN even though only trained on image itself but it can be seen that it's derivatives are very much inline with derivatives of original signal.(see blue color circles in the image).

Simply by matching the signal this SIREN architecture manages to also capture the derivatives of the signal which results in giving the better output.

What do the SIRENs do different to capture the signals well from other architectures ?

$$\Phi(\mathbf{x}) = \mathbf{W}_n (\phi_{n-1} \circ \phi_{n-2} \circ \dots \circ \phi_0)(\mathbf{x}) + \mathbf{b}_n,$$

network

final layer
(linear layer)

all these layers are following each other

A SIREN is a multi-layer perceptron, with a linear final layer consists of all layers following each other.

$$\mathbf{x}_i \mapsto \phi_i(\mathbf{x}_i) = \sin(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i).$$

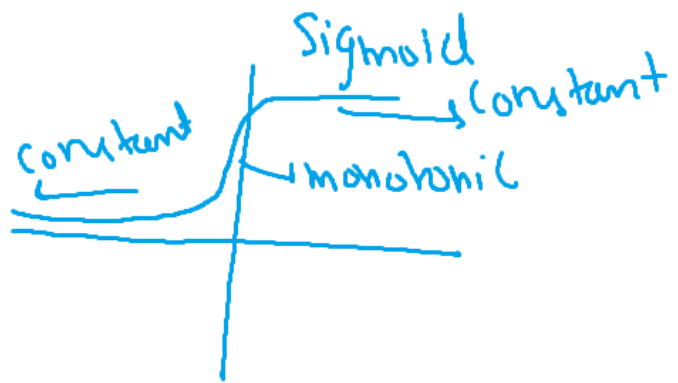
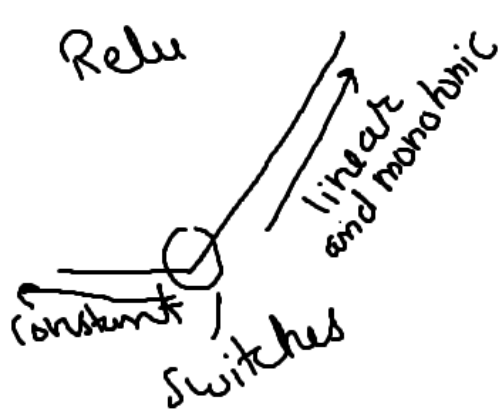
sine wave

weight input matrix

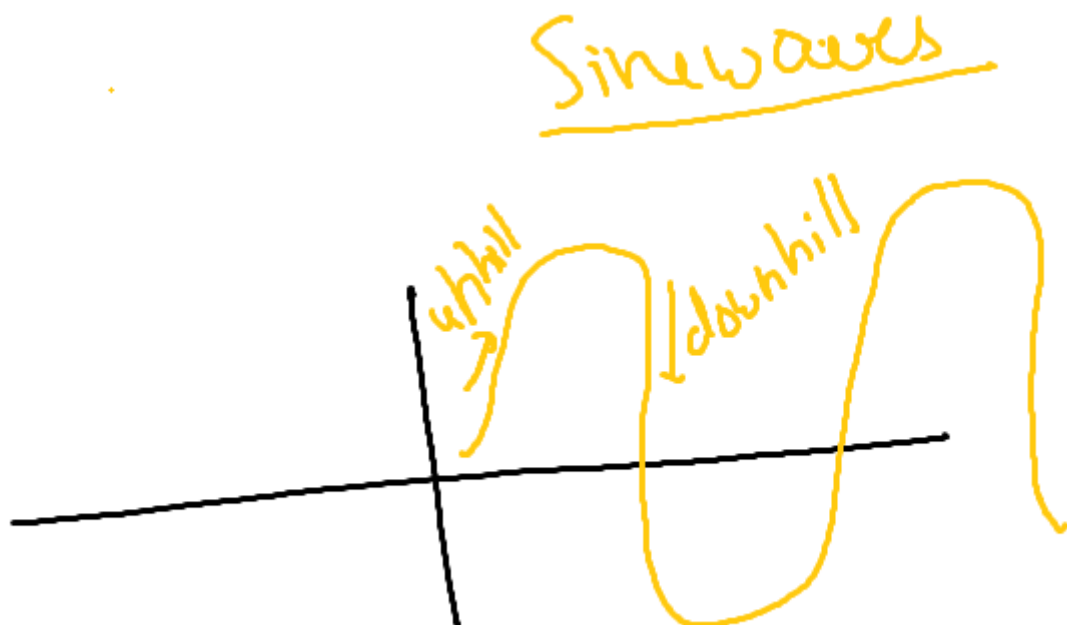
bias

Each layer is made up like shown in the above image.

we have a input, then it is multiplied by the weight matrix and then bias is added. And then it is put inside the sine wave. this sine wave is the only change from normal mlp. usually instead of sin we have ReLu or sigmoid



If you see the Relu or sigmoid both are constant upto some point then switches to monotonic behaviour. till now are are use to monotonic activation functions. where as the sine waves are really different .



it's not monotonic at all if we want to increase the function value at any point and takes a large step up the hill we end up down the hill again, but it turns out that these networks also have particularly some good properties for capturing the natural signals. how they overcame the of its disadvantage of being perodic by initiallizing it in a particular fasion.

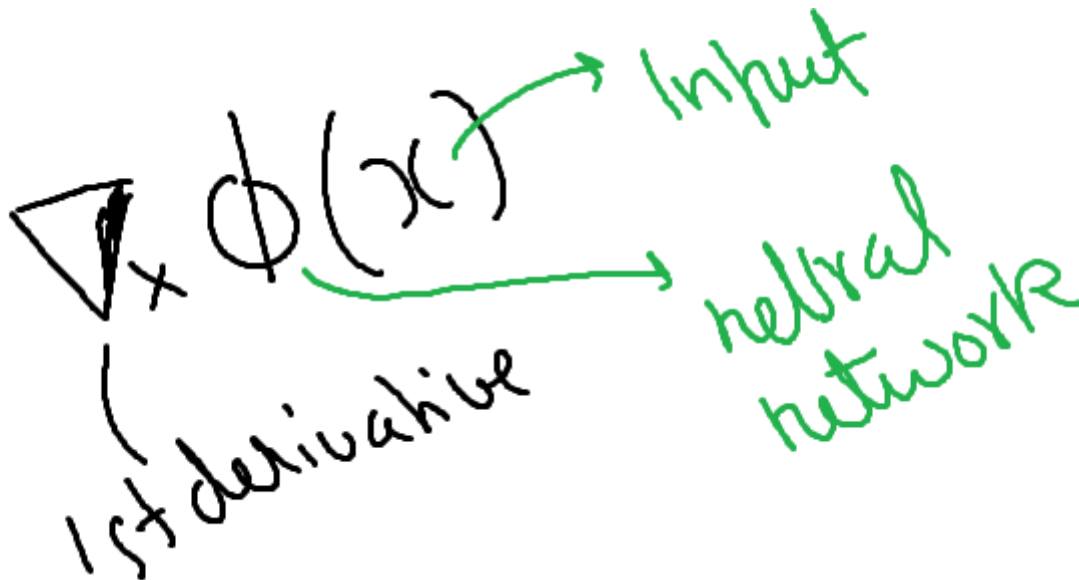
$$w_i \sim \mathcal{U}(-c/\sqrt{n}, c/\sqrt{n}), c \in \mathbb{R}$$

we need to sample the weight uniformly from the uniform distribution given above. they propose if we take the $c = 6$ then the input to each sine activation is distributed normally with standard deviation of 1.

So, what is about the derivatives used?

As, the network used has the sine waves in it, it's a neural network with sine waves as the non-linearity.

What will be first derivative of the neural network with respect to its input,



first derivative of a sine wave is a sine wave that is shifted which is also a cosine wave (phase shifted sine wave), next derivative is again a shifted sine wave and so on. So derivative of a SIREN is a SIREN and that does not work for any of the other non-linearity function.

for example the 2nd derivative for relu function is a constant zero function.

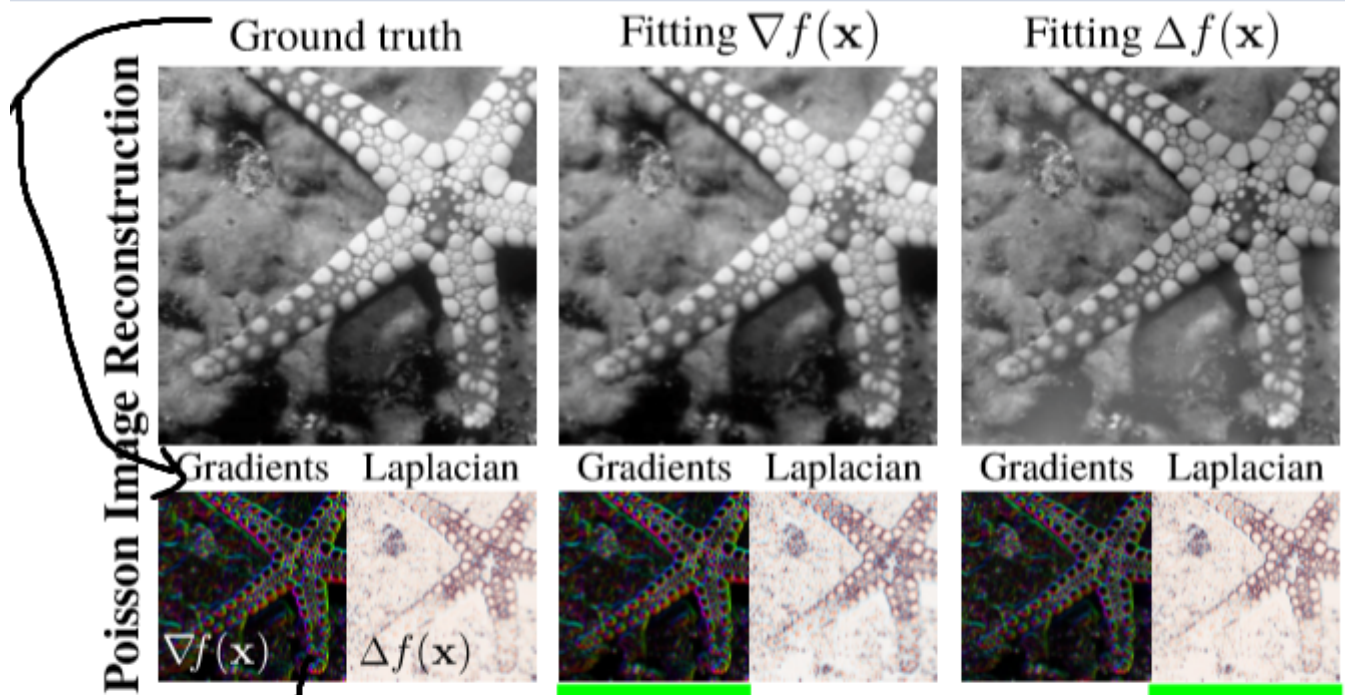
and since we want to match signals as well as their derivatives this property of siren become very useful.

How do we train a SIREN ?

Instead of matching the pixels value to its RGB value there is more can be done with SIREN, given that there derivatives are also SIREN.

$$F(\mathbf{x}, \Phi, \nabla_{\mathbf{x}} \Phi, \nabla_{\mathbf{x}}^2 \Phi, \dots) = 0, \quad \Phi : \mathbf{x} \mapsto \Phi(\mathbf{x}).$$

we can say that we can find the relation between the input and it's first derivative. and see what is coming out , that is something is shown in the image below.



after we put the gradient(second derivative) in SIREN , it will map the co-ordinates with RGB.

but the lost function isn't going to be mapping x,y to RGB but it will be depending on the gradient of that.

LOSS Function will be like :

Loss function

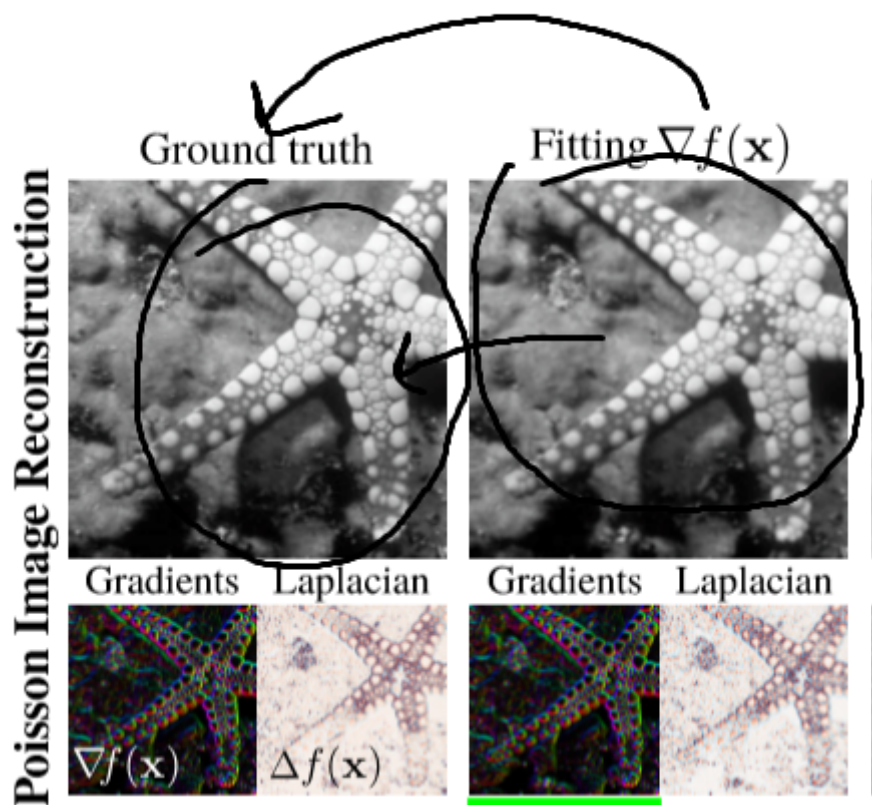
$$L = \|\nabla I - \nabla \phi\|$$

gradient of Image

gradient of pn matches $x, y \rightarrow RGB$

Loss function is subtracting the gradient of function matches co-ordinates to its RGB values from gradient of the image itself. So, now we are looking for a function whose gradient matches the gradient of the image.

After doing this it can be found that if the function is asked to reproduce the image from cycling over each co-ordinates and mapping it to the RGB values, it can be seen that just by matches the gradient it can matches to the image pretty well.



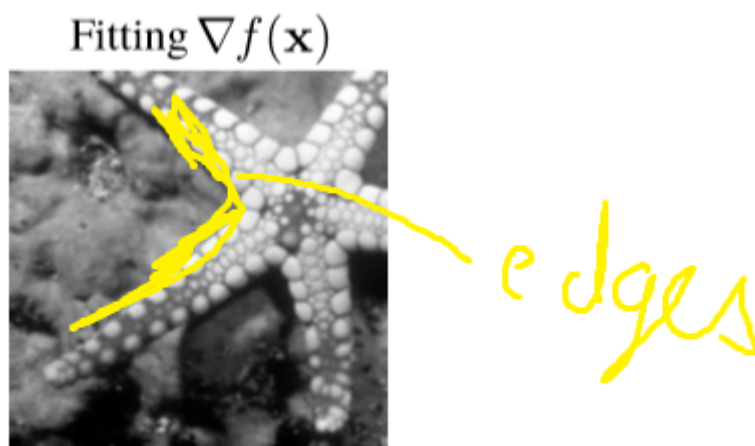
though it is not matching RGB from real image as the gradient is a grey scale image because gradient loses constant bias information.

if we have a derivative of function and want to find then function itself, we integrate and the solution is always the entire space of functions and we have to add a constant and we don't know what the constant really was because when we derive the function the constant drops.

$$\nabla f \rightarrow F = \int \nabla f dx + c$$

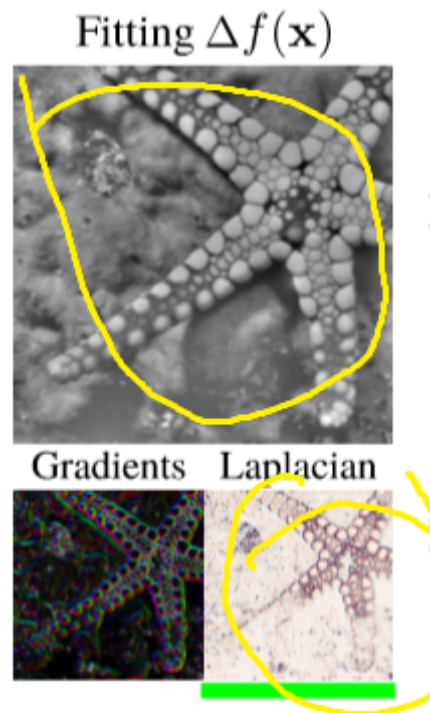
constant

So, we can expect that the image we are getting back representing the change around edges very well since, we are matching the gradient and gradient is basically the edge detector.



and it is matching the edges very well, the only difference we can see is only change in luminosity

the same thing is done to match the second derivatives, which is laplacian. Note: in Relu we don't even have a laplacian it is a constant. and still the outcome is very good.

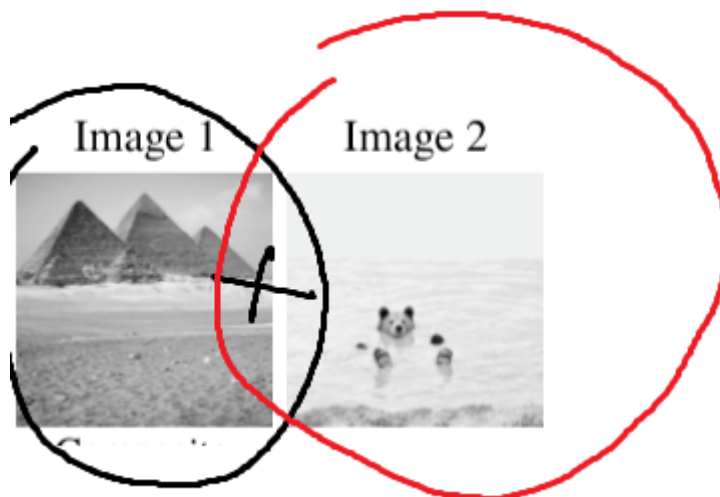


this is now missing the luminosity in the 0th and 1st derivatives but the reconstruction is really good.

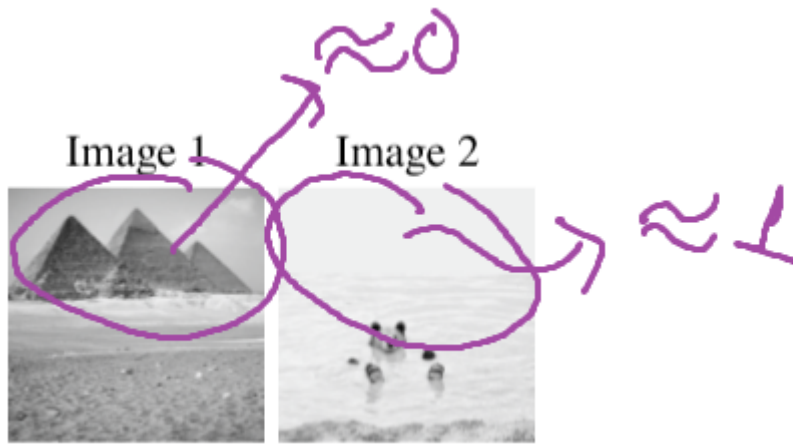
Notice the power of these networks as they are using just a image as whole datasets. we can consider the laplacian or sober filter as the dataset and the real image is the test sample.

Application 2

If we want to mix two images what we can do is linearly interpolate,



but that will not give good result because if you see in Image 2 there are lots of bright pixels which probably have value of 1, and in Image 1 there are dark pixels which will probably have values closer to zero



if we will simply add them together and divid by two , we will end up washing out the important pixels. With GAN's we can do this but we have to have a training set and everything.

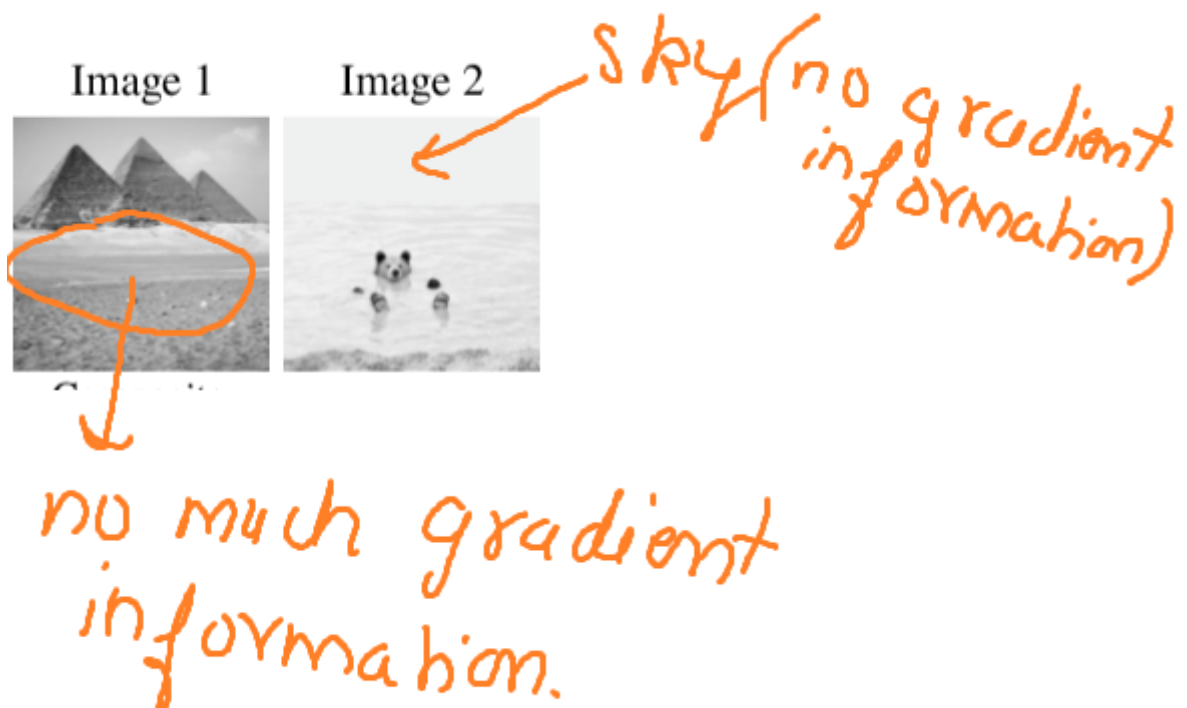
Here what we are doing is

Taking the gradient of image one.

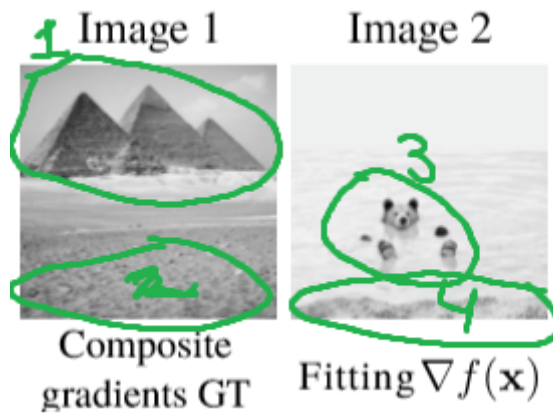
Taking the gradient of image two.

Then we will add the two gradient maps.

What it does ?

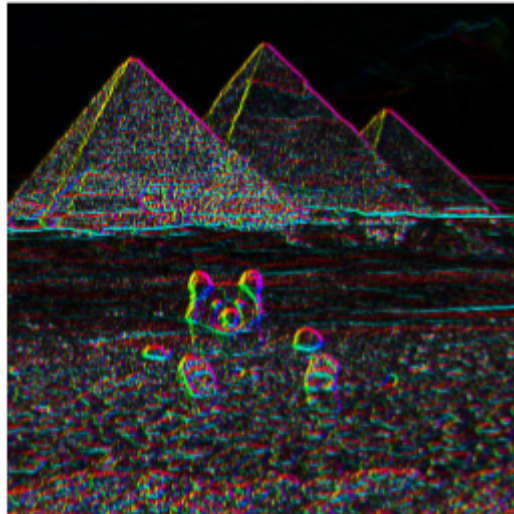


So, while mixing images it could be a good idea to mix their gradients as information in the image is what where the gradients are.



So gradient will carry these 4 portions (everything where signal is not flat) And if we fit our function such that the gradient of the function matches this mixed gradient.

Composite
gradients GT



adding the
gradient

the pixel information from image 1 and image 2 is added to this gradient above and then this gradient is fitted then the function is asked to output pixel value at each location which gives us this output.

Estimated composite image



Representing shapes with Signed Distance Function

To understand we need to go over the actual formulation of their loss function. it is very complicated stated, but basically what it means is :

$$\mathcal{L} = \int_{\Omega} \sum_{m=1}^M \mathbf{1}_{\Omega_m}(\mathbf{x}) \|\mathcal{C}_m(\mathbf{a}(\mathbf{x}), \Phi(\mathbf{x}), \nabla\Phi(\mathbf{x}), \dots)\| d\mathbf{x},$$

2nd, 3rd derivatives and so on.
 constraints
 anything depending on input itself
 then output of function
 then gradient of output of function

This loss function operates on the component \mathcal{C}_m which is a constraint, and constraint is a across $\mathbf{a}(\mathbf{x})$, basically it is just \mathbf{x} , anything depending on input itself then the output of function, the gradient of output of the function, then second derivative, third derivative and so on..

These SIRENs can fit anything that we can formulate as a set of constraints that related the input of function $\mathbf{a}(\mathbf{x})$ to its output or any of its derivative.

it has been already seen that if we fit an image our only constrain is that

$$\mathcal{C}_m(\mathbf{a}(\mathbf{x}), \Phi(\mathbf{x}), \nabla\Phi(\mathbf{x}), \dots) = 0.$$

these thing
 match?
 that the co-ordinates
 are mapped to RGB values

$a(x)$ should match to

$$\phi(x)$$

which is co-ordinates are mapped to the RGB values, then when we match the gradient we don't care about the output

$$\phi(x)$$

$$\mathcal{C}_m(\mathbf{a}(\mathbf{x}), \cancel{\Phi(\mathbf{x})}, \nabla \Phi(\mathbf{x}), \dots) = 0.$$

we only care about relation
between this and so on,

we only care about the relation between the input and first derivative and so on.

So the loss function is just over the entire signal space which is in this case over entire image, we want these constraints to be as small as possible.

Loss function

$$\mathcal{L} = \int_{\Omega} \sum_{m=1}^M 1_{\Omega_m}(\mathbf{x}) \|\mathcal{C}_m(\mathbf{a}(\mathbf{x}), \Phi(\mathbf{x}), \nabla \Phi(\mathbf{x}), \dots)\| d\mathbf{x},$$

over the entire signal space (image)

norm

we want these as small as possible

constraints are always formulate such that if they full filled , they are equal to zero.

for example the L2 loss between the RGB values between the true image and the RGB values that we fit the L2 loss will be a constraint , the more differentiable we make it the easier this network has fitting it, that why there is the norm symbol in equation.

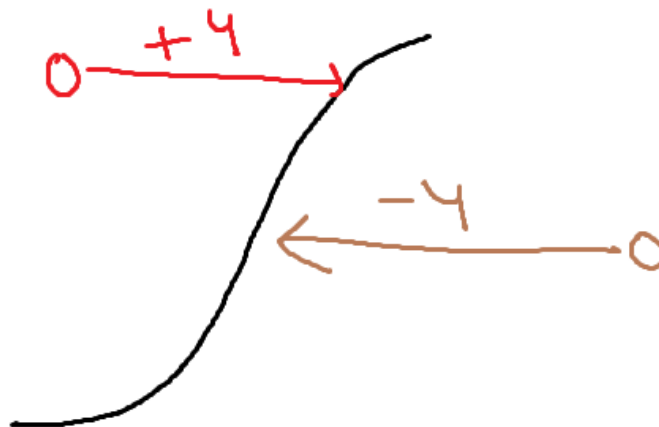
it simply says whatever we can formulate as a constraint on relating the inputs to the outputs or any of the derivatives of this implicit representation that is the loss function.

Representing Shapes with Signed Distance Functions

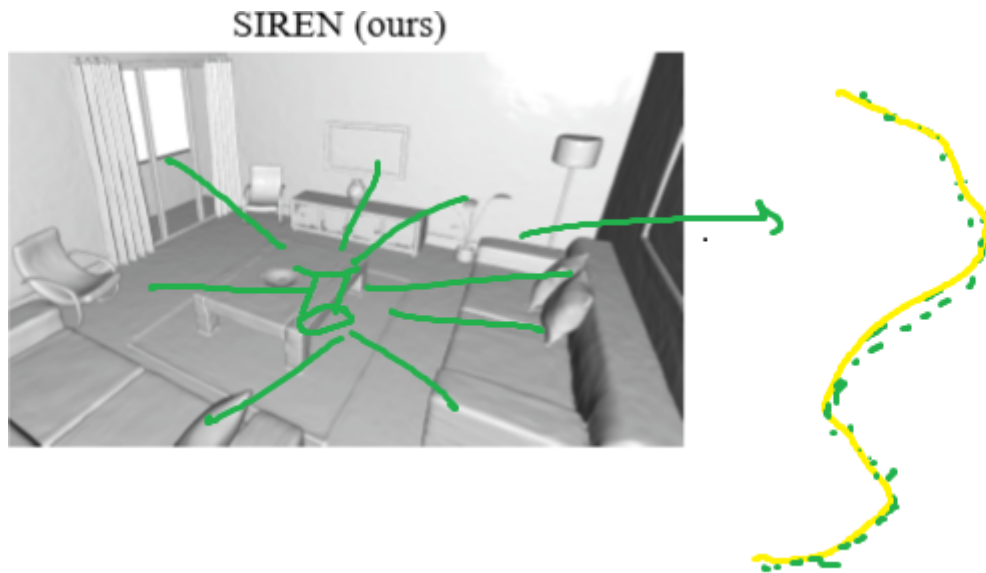
What is signed Distance function ?

it is nothing what distance of a certain data point from some boundary with sign associated with it.

This is useful in case of point clouds.



they have shown these room interior , which is point cloud of 3D scene. what it means that like we put a laser scanner in middle of the room , and it shoots lasers at random location and measures the distance , that's how we end up with a point cloud. then we should be able to connect those points to obtain the continuous shape.



And this what we going to try to do with SIRENs to go from point cloud to shape by training from implicit representation . We are going to train a neural network that can represent the shapes by co-ordinates to signed distance values.