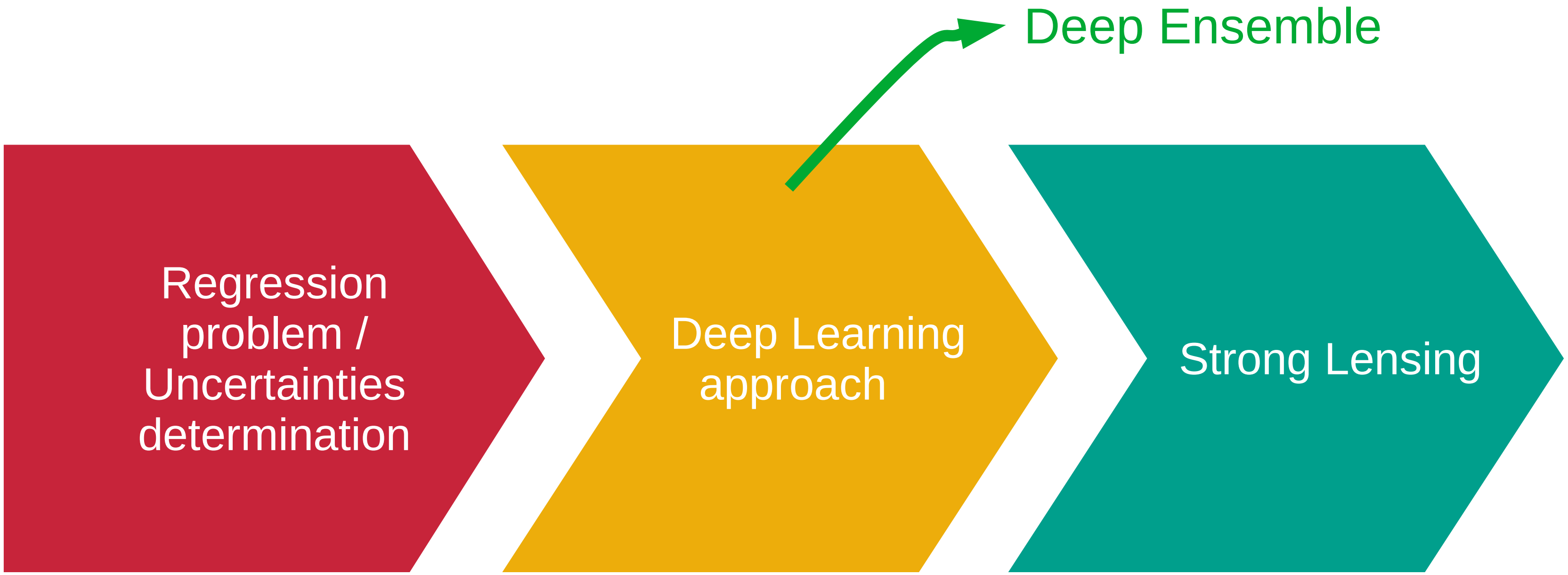
Uncertainties determination of gravitational lens

Strong Lensing meets Deep Ensemble

David Camarena

The landscape



Regression and uncertainties

Input Data

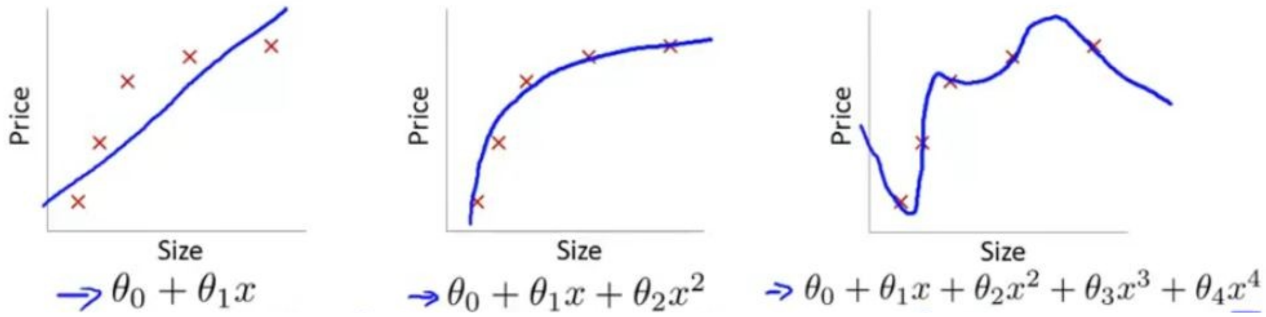
↓

Modeling

↓

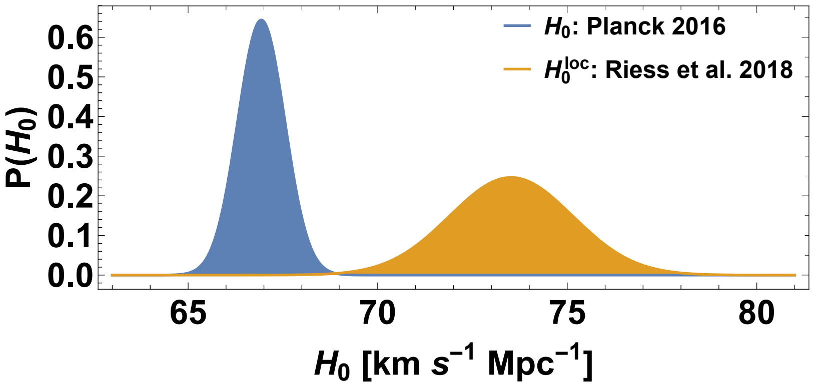
Prediction

Example: Linear regression (housing prices)



Uncertanties are critical!

Uncertainties can change the way we understand the universe!



Subscribe | Latest Issues | SCIENTIFIC AMERICAN 175 | Cart 0 | Sign In | Stay Informed

CORONAVIRUS THE SCIENCES MIND HEALTH TECH SUSTAINABILITY VIDEO PODCASTS OPINION PUBLICATIONS

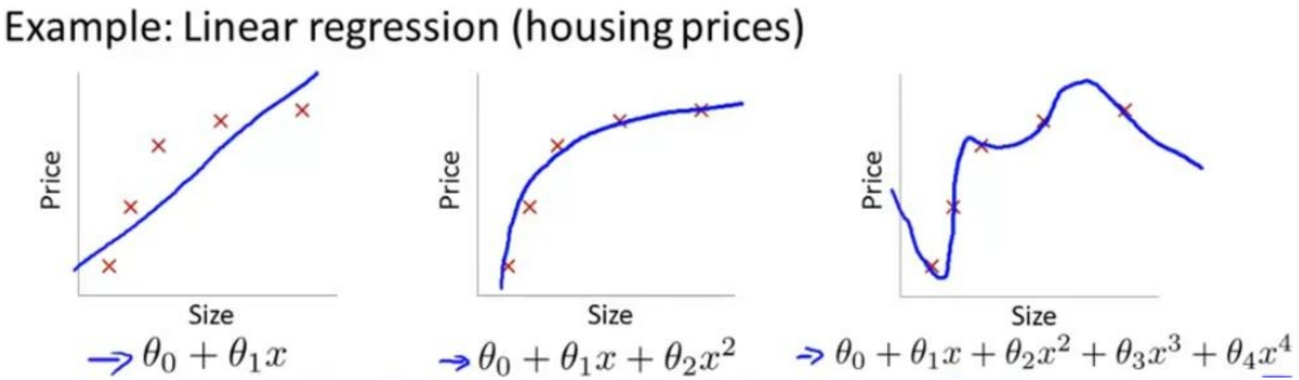
SPACE

How a Dispute over a Single Number Became a Cosmological Crisis

Two divergent measurements of how fast the universe is expanding cannot both be right. Something must give—but what?

By Richard Panek on March 1, 2020

Bayesian approach

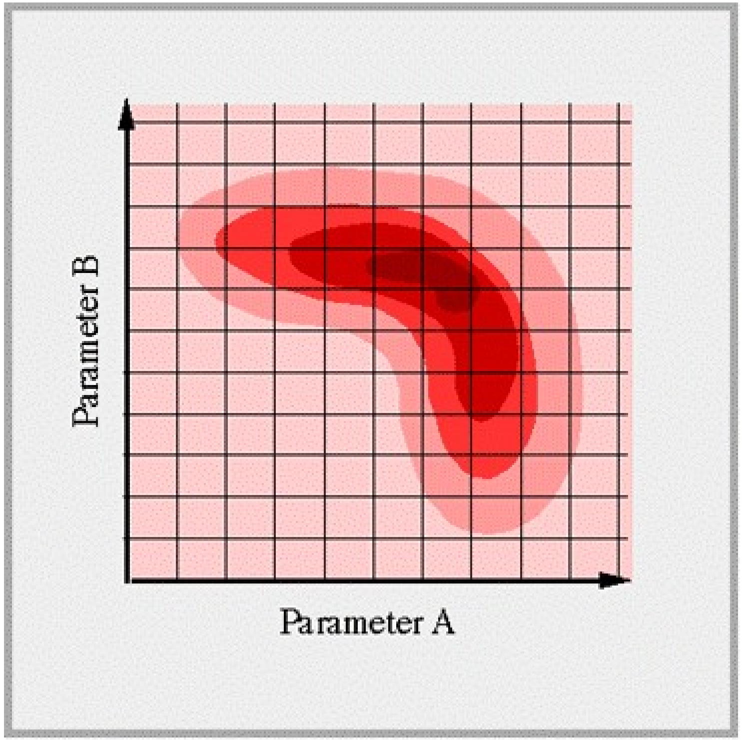


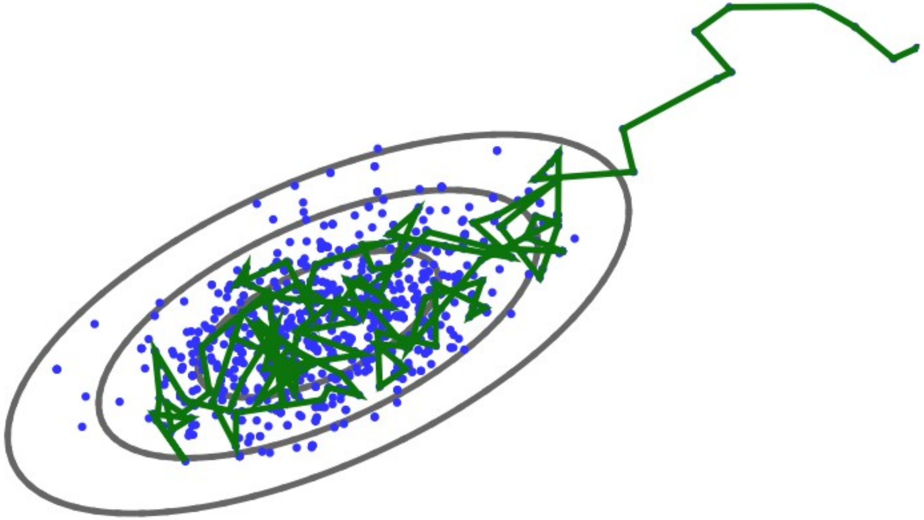
$$\mathcal{P}(\theta|D) = \frac{p(\theta)\,\mathcal{L}(D|\theta)}{p(D)}$$

D : data
θ : model parameters
| : conditional parameters

p : prior
L : likelihood
P : posterior

Exploring the parameter space





Bayesian Deep Learning

$$p(\boldsymbol{\theta}|D) = \frac{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}'} \propto p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta}).$$

$$\Big\downarrow \; p(\theta)$$

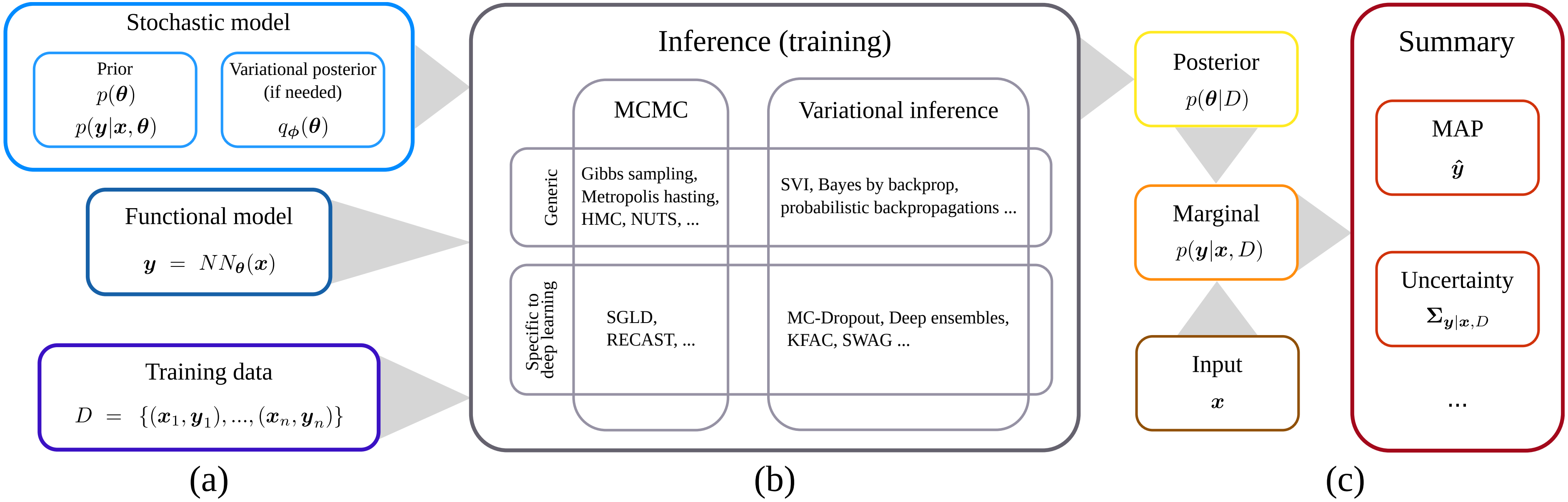
$$p(\boldsymbol{y}|\boldsymbol{x}, D) = \int_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}')p(\boldsymbol{\theta}'|D)d\boldsymbol{\theta}'.$$

Predictions

θ : NN parameters
Dx : Train features
Dy: Train label

X : Valid fearures
Y : Valid labels.

Bayesian Deep Learning



https://arxiv.org/abs/2007.06823 Hands-on Bayesian Neural Networks -- a Tutorial for Deep Learning Users

Deep Ensembles

- Ensemble with N networks.
- Model parameters, θ ,are randomly init.
- Each network is trained using randomic mini-batch.
- Each network have two output.
- A proper LOSS is defined.
- Final ensemble outputs are combined.

https://arxiv.org/abs/1612.01474   Simple and scalable predictive uncertaintyestimation using deep ensembles

Deep Ensembles

Outputs are:

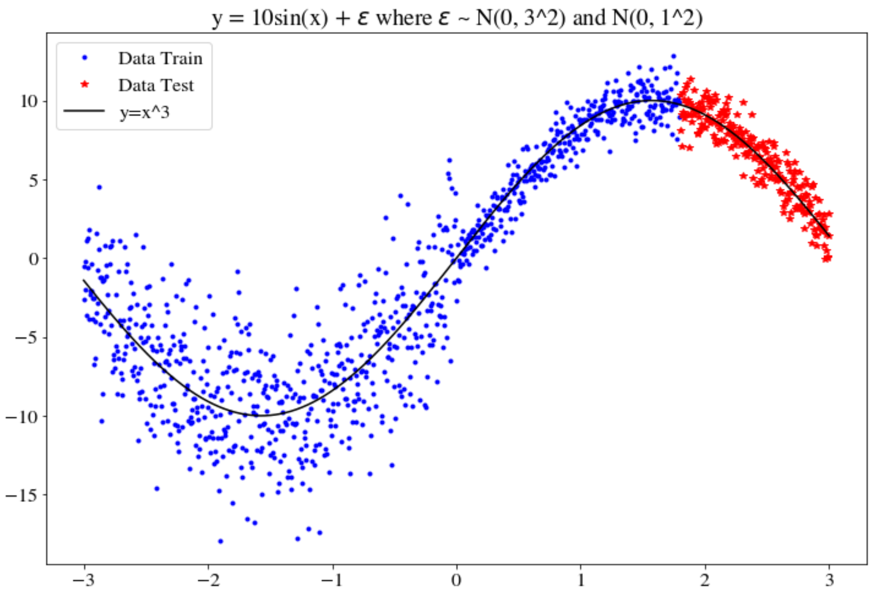
$$\mu_\theta(x) \qquad \sigma^2_\theta(x)$$

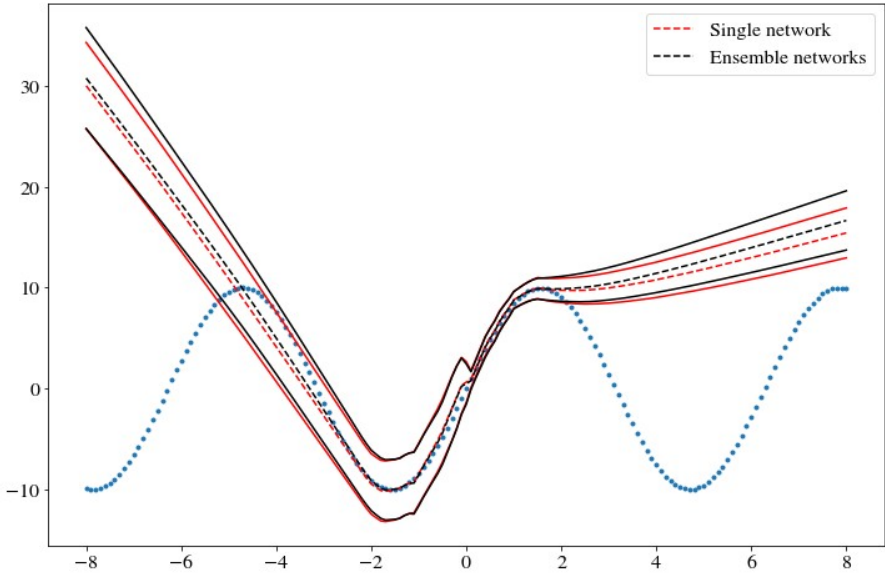
Regression

Proper Loss (Negative Log Likelihood)

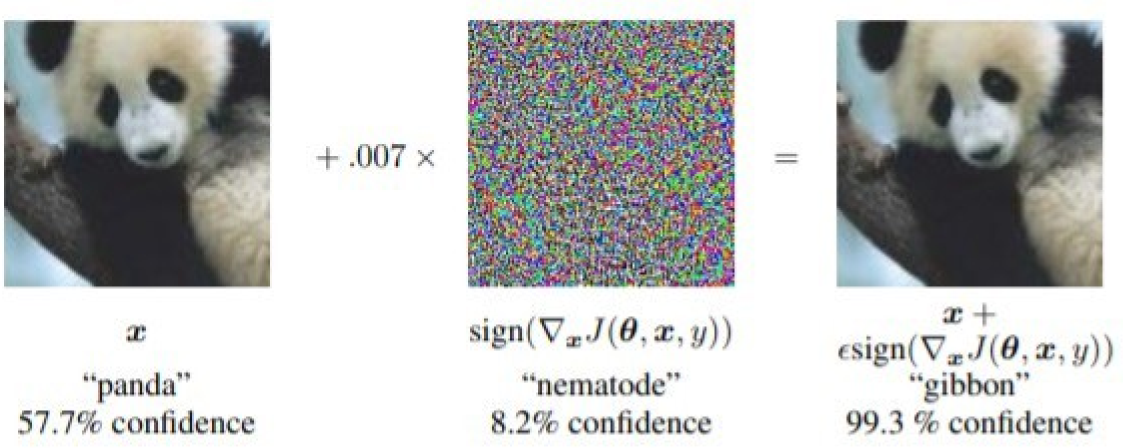
$$-\log p_\theta(y_n|\mathbf{x}_n) = \frac{\log \sigma^2_\theta(\mathbf{x})}{2} + \frac{\left(y - \mu_\theta(\mathbf{x})\right)^2}{2\sigma^2_\theta(\mathbf{x})} + \text{constant}.$$

Pragmatical Example





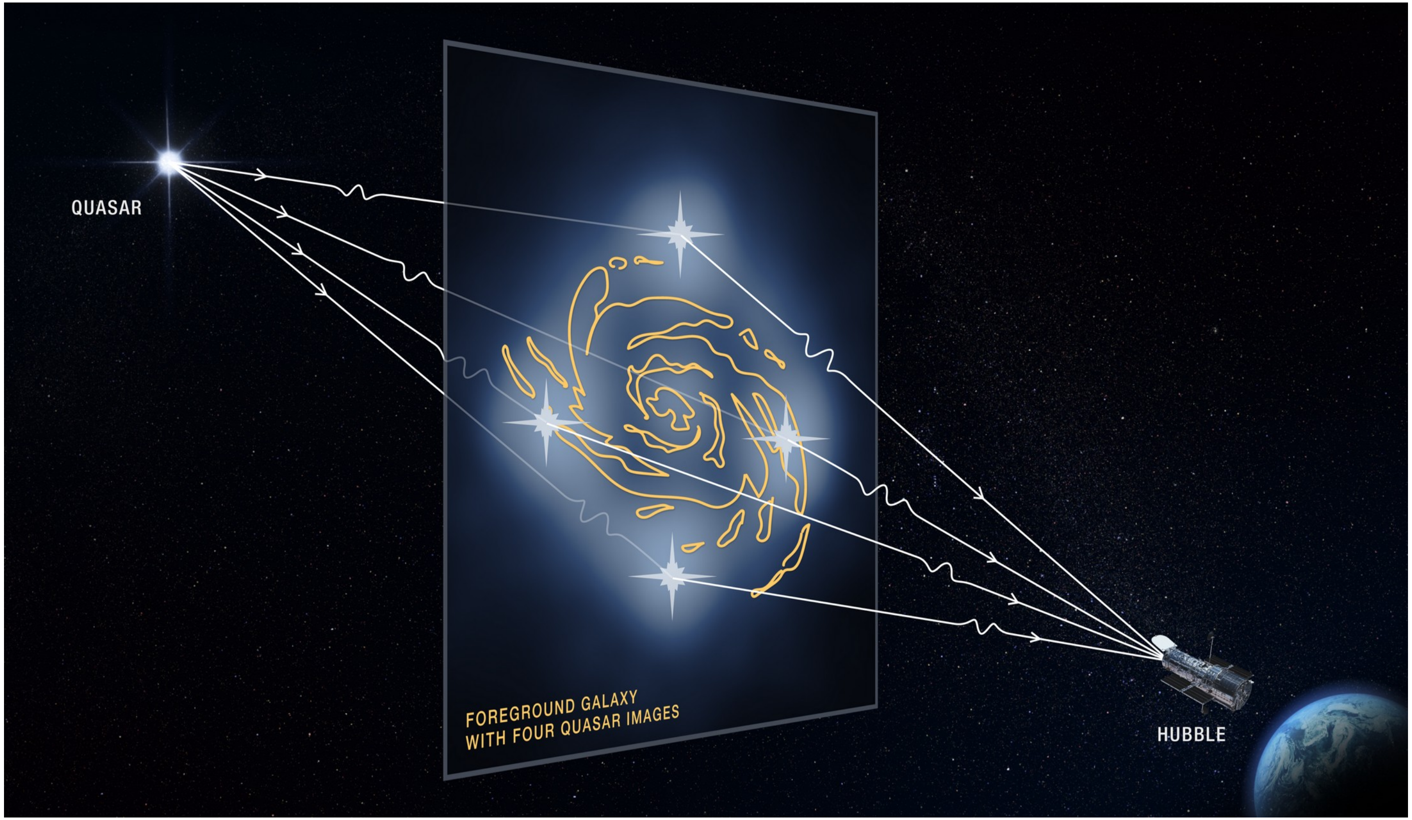
Adversarial training



Smooth the prediction
\+ Robust results

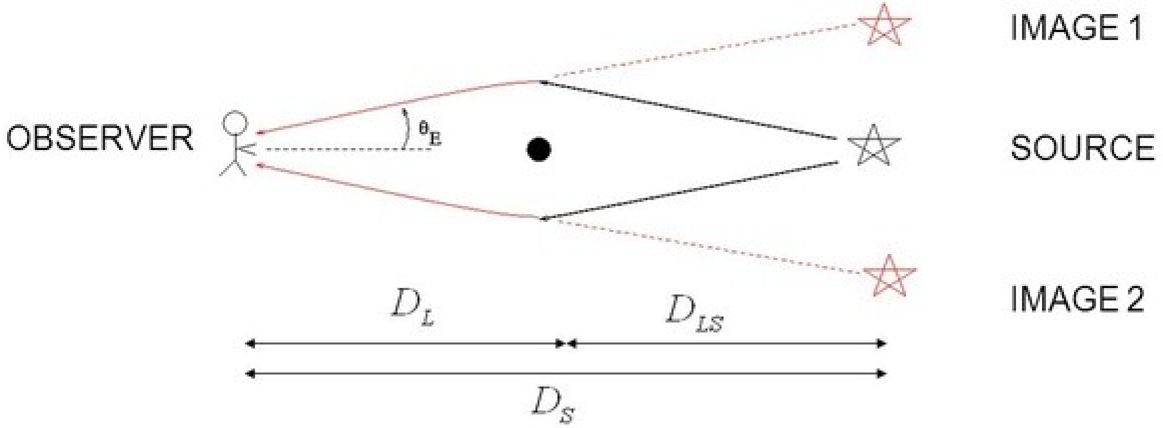
Deus dá as batalhas mais difíceis aos seus melhores soldados.

Strong Lensing

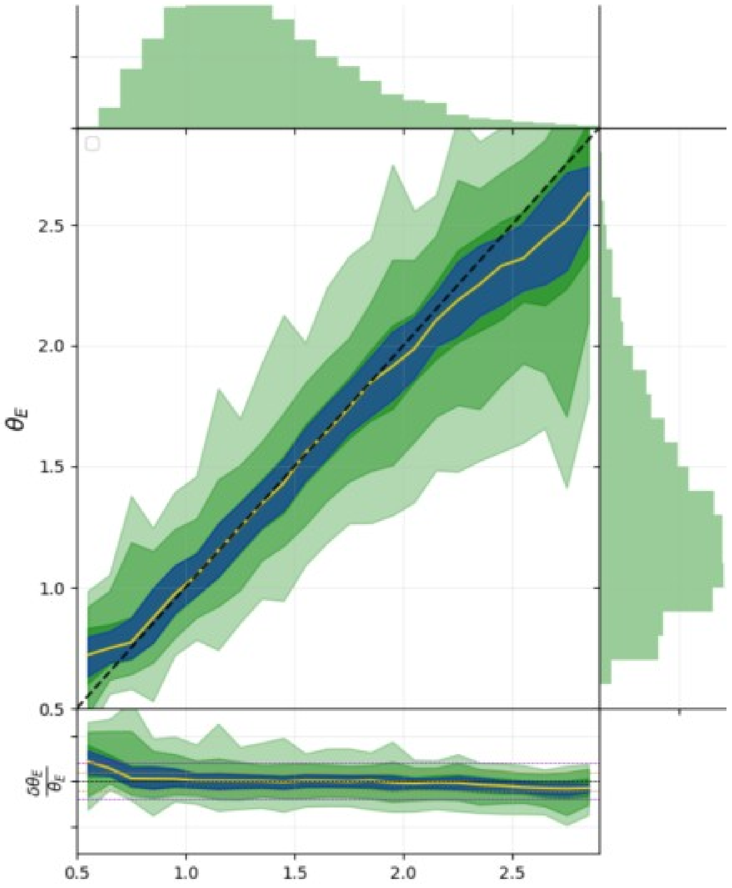


(Vainilla) Goals

https://arxiv.org/abs/1911.06341

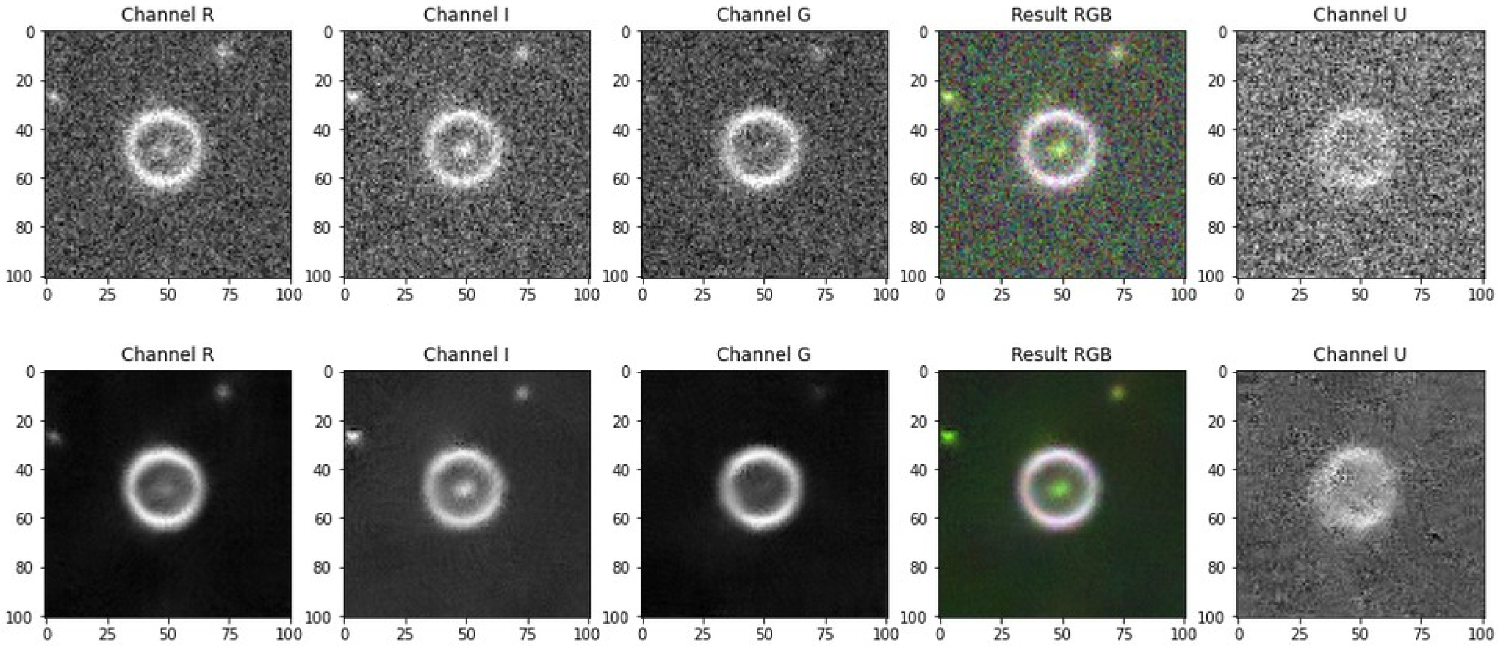


Einstein radius: $$R_E = \sqrt{\frac{4GM_{tot}D_{LS}}{c^2 D_L D_S}}$$



Data set

Lens Finding Challenge 1 (http://metcalf1.difa.unibo.it/blf-portal/gg_challenge.html)



Vainilla model

```
def get_network(network_name):
    init = tfki.GlorotNormal()
    input_1 = tfkl.Input(shape=(55, 55, 3),dtype='float32')
    conv_1 = tfkl.Conv2D(32, (3,3), activation='relu', kernel_initializer=init,
                         bias_initializer=init)(input_1) # , input_shape=(55, 55, 3)
    batchn_1 = tfkl.BatchNormalization()(conv_1)
    maxpool_1 = tfkl.MaxPool2D(pool_size=(2,2))(batchn_1)
    conv_2 = tfkl.Conv2D(16, (5,5), activation='relu', kernel_initializer=init,
                         bias_initializer=init)(maxpool_1)
    batchn_2 = tfkl.BatchNormalization()(conv_2)
    maxpool_2 = tfkl.MaxPool2D(pool_size=(2,2))(batchn_2)
    flat_1 = tfkl.Flatten()(maxpool_2)
    dense_1 = tfkl.Dense(64, activation='relu', kernel_initializer=init, bias_initializer=init)(flat_1)
    batchn_3 = tfkl.BatchNormalization()(dense_1)
    dense_2 = tfkl.Dense(32, activation='relu', kernel_initializer=init, bias_initializer=init)(batchn_3)
    batchn_4 = tfkl.BatchNormalization()(dense_2)
    out_mean = tfkl.Dense(1, kernel_initializer=init, bias_initializer=init)(batchn_4)
    out_mean_pos = tf.keras.backend.abs(out_mean)
    out_raw_var = tfkl.Dense(1, kernel_initializer=init, bias_initializer=init)(batchn_4)
    out_var = tf.keras.backend.log(1.0 + tf.exp(out_raw_var)) + 1e-3
    model = tf.keras.Model(name=network_name,inputs=[input_1], outputs=[out_mean_pos,out_var])
    return model
```

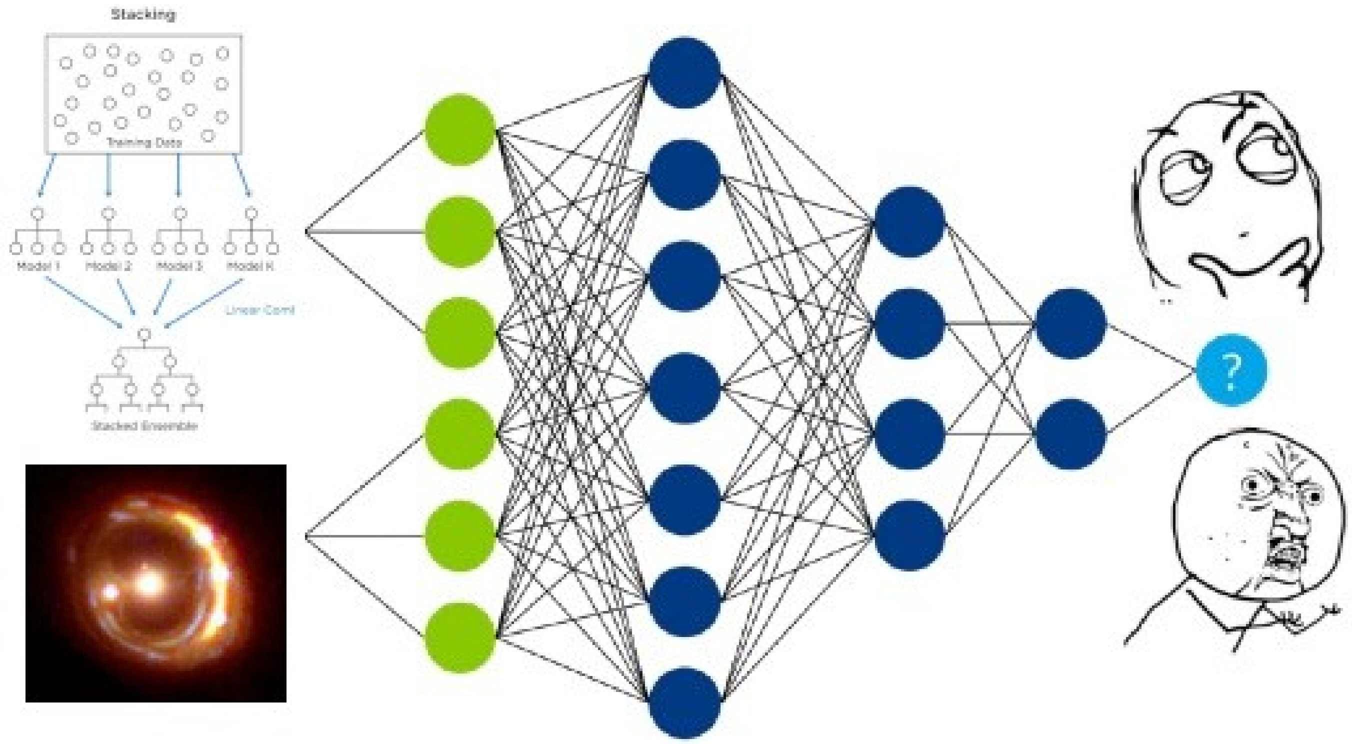
Current status

```
------------------------- Iteration: 25 -------------------------
Average Loss(NLL): [       nan 0.05070004 0.04717745 0.13318719 0.04487872]
Average Loss(NLL): [       nan 0.05642756 0.05009264 0.16188377 0.040849  ]
------------------------- Iteration: 50 -------------------------
Average Loss(NLL): [       nan        nan        nan 0.04954464        nan]
Average Loss(NLL): [       nan        nan        nan 0.06562837        nan]
------------------------- Iteration: 75 -------------------------
Average Loss(NLL): [nan nan nan nan nan]
Average Loss(NLL): [nan nan nan nan nan]
------------------------- Iteration: 100 -------------------------
Average Loss(NLL): [nan nan nan nan nan]
Average Loss(NLL): [nan nan nan nan nan]
------------------------- Iteration: 125 -------------------------
Average Loss(NLL): [nan nan nan nan nan]
Average Loss(NLL): [nan nan nan nan nan]
------------------------- Iteration: 150 -------------------------
Average Loss(NLL): [nan nan nan nan nan]
Average Loss(NLL): [nan nan nan nan nan]
```

Abstract and introduction

Methods and results

Obrigado!


Stacking
Training Data
Model 1
Model 2
Model 3
Model K
Linear Comb
Stacked Ensemble
?