Fast Neural Solvers

Patrick Pérez

valeo.ai

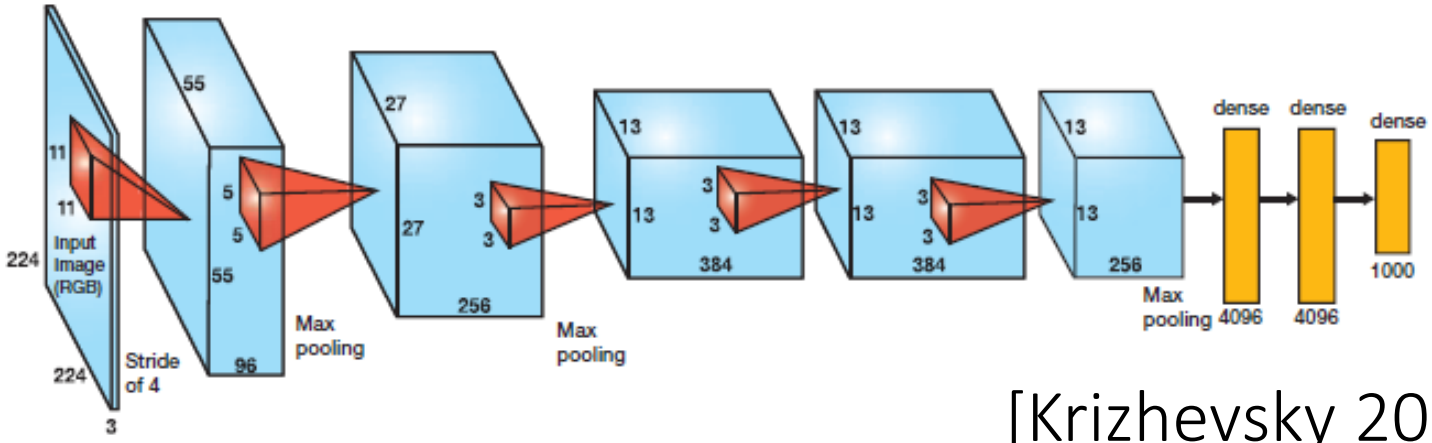
Horizon Maths: AI, ENS Paris

23 November 2018

[Tompson 2017]

Deep neural networks

Better performance on ML tasks, at high cost



[Krizhevsky 2012]

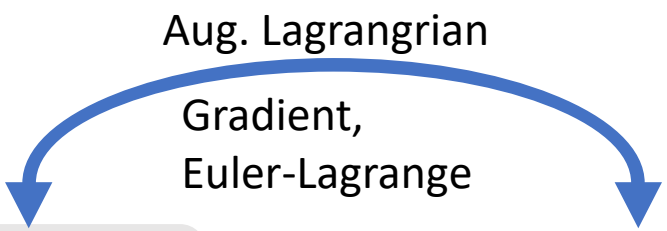
Fast approximate solution of ML and non-ML problems

Optimization problems, inverse problems, PDEs, ODEs

Setting the stage

Original problem

- Optimization or equation



$$\hat{\mathbf{y}}(\mathbf{x}) \in \arg\min_{\mathbf{y}} E(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) \qquad \hat{\mathbf{y}}(\mathbf{x}) \text{ s.t. } F(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = 0$$

- Input is not arbitrary: $\mathbf{x} \sim \mathbb{P}_X$

Classic solvers, *iterative* and *generic*

$$\text{Solver}(\mathbf{x}) = \text{Iter}^{\infty}(\mathbf{x}; \mathbf{y}^0) \approx \hat{\mathbf{y}}(\mathbf{x}), \ \forall \mathbf{x}$$

- Gradient, proximal, fixed point
- PDEs, ODEs solvers

Problem families

Physics

- Single physics: solid, elastic, fluid, condensed, EM, quantum, chemical
- Multi-physics: meteorology, medicine, cosmology, engineering

Inverse problems

- Signal enhancement, reconstruction and transformation
- Compressed sensing and sparse coding
- Inverse rendering

Model fitting

- Polynomial regression
- Factorization methods (dict. learning, NMF, graph spectral methods, etc.)
- Metric learning

Inverse problems

High level formulation

- Forward process (known or assumed) to “invert”

$$\mathbf{y} \rightarrow \mathbf{x} = g(\mathbf{y})$$

- Prior (known or assumed) to well-pose inversion

$$\forall \mathbf{x},\ \hat{\mathbf{y}}(\mathbf{x}) \in \arg\min_{\mathbf{y}} \left(\text{Loss}(\mathbf{x}, g(\mathbf{y})) + \text{Prior}(\mathbf{y}) \right)$$

Classic inverse problems

Signal enhancement and completion

- Forward process: degradation and/or masking
- Prior: spatial/temporal regularity

$$\min_{\mathbf{y}} \left(\|\mathbf{x} - g(\mathbf{y})\|_2^2 + \mu \|\nabla \mathbf{y}\|_p^q \right) \text{ s.t. } \mathbf{y}|_D = \mathbf{y}^*|_D$$

Signal recovery

- Linear forward process: atom composition, random measurements
- Prior: sparsity

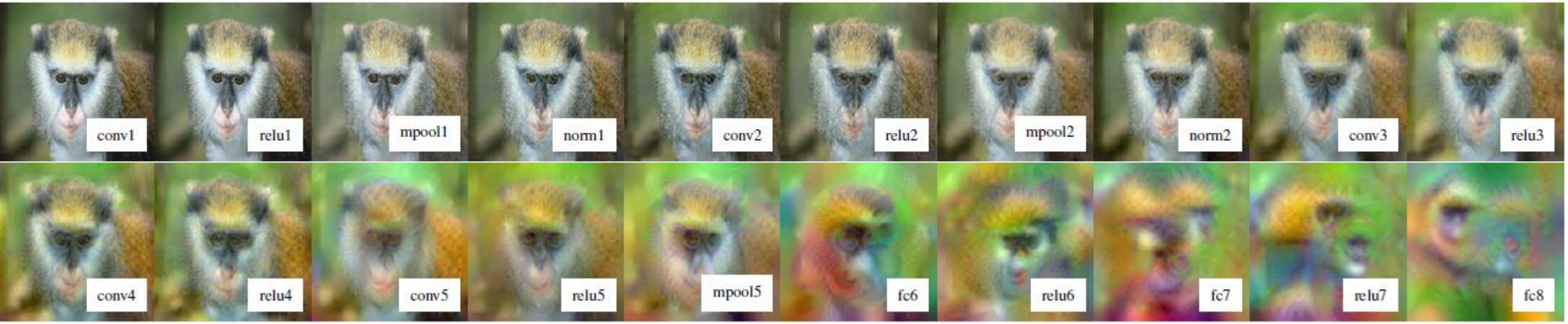
$$\min_{\mathbf{y}} \left(\|\mathbf{x} - A\mathbf{y}\|_2^2 + \mu \|\mathbf{y}\|_1 \right)$$

Neural inverse problems

Working in deep representation space

- Forward process: deep features extraction
- Visualization, editing

$$\min_{\mathbf{y}} \left(\|\mathbf{x} - \phi_{\ell_0}(\mathbf{y})\|_{\mathrm{F}}^2 + \mu \|\nabla \mathbf{y}\|_1 \right)$$



[Mahendran 2015]

Neural inverse problems

Working in deep representation space

- Forward process: deep features extraction
- Visualization, editing

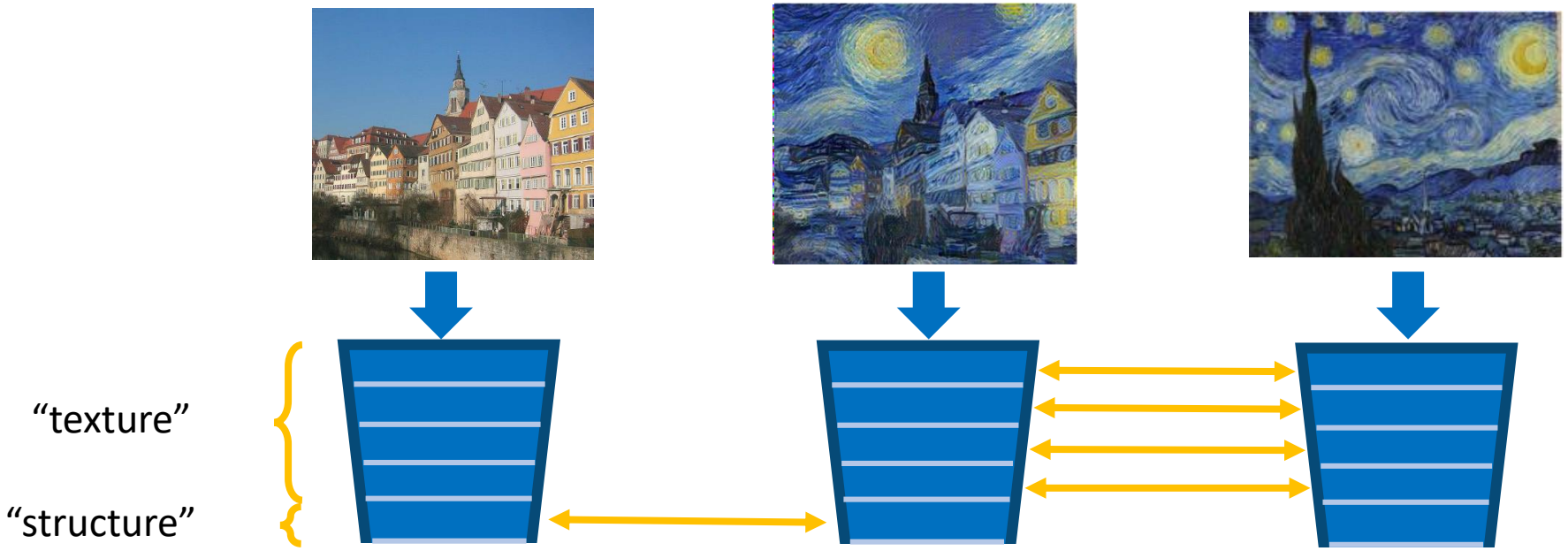
$$\min_{\mathbf{y}} \left(\|\mathbf{x} - \phi_{\ell_0}(\mathbf{y})\|_{\mathsf{F}}^2 + \mu \|\nabla \mathbf{y}\|_1 \right)$$

- “Style transfer”

$$\min_{\mathbf{y}} \left(\|\mathbf{x} - \phi_{\ell_0}(\mathbf{y})\|_{\mathsf{F}}^2 + \lambda \sum_{\ell \in \mathcal{L}_{\mathsf{sty}}} \|G_\ell - \phi_\ell(\mathbf{y})^\top \phi_\ell(\mathbf{y})\|_{\mathsf{F}}^2 \right)$$

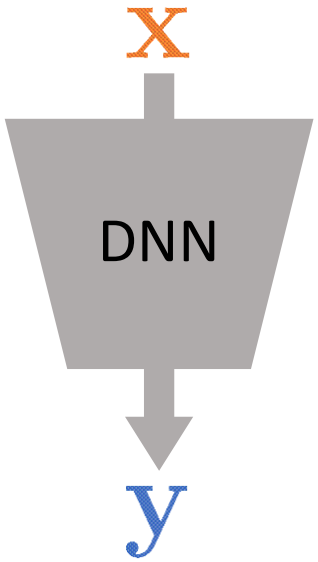
“Artistic” Style transfer

retain structure, imitate texture



[Gatys 2015-2016]

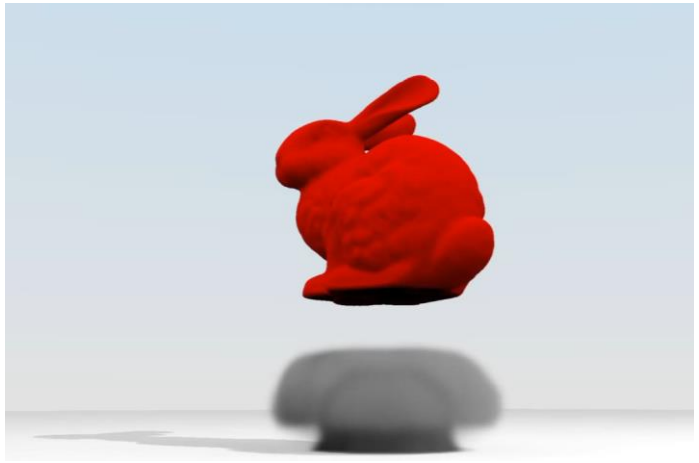
Neural solver



Train DNN to regress solution *for plausible inputs*

$$\mathrm{DNN}(\mathbf{x}; \mathcal{W}) \approx \hat{\mathbf{y}}(\mathbf{x}), \ \forall \mathbf{x} \sim \mathbb{P}$$

- Fixed complexity, fast (on GPU), differentiable
- Architecture? Training?



speed X 50

speed X 100

speed X 500

Architectures

Your favorite DNN

- Exploit popular architectures, possibly pre-trained
- In vision: VGG, ResNet, fully convolutional encoder-decoder, U-net…

Unrolling

- Mimic (possibly loosely) structure of iterative solver
- Each [non-linear ∘ linear] iteration becomes a neural layer
- Fixed, smaller number of “iterations”
- But, way more freedom exploited through training

Training

Fully-supervised – Using Solver, reconstruction loss

$$\mathbf{x}^{(n)} \sim \mathbb{P}_X, \ \min_{\mathcal{W}} \sum_{n=1}^{N} \|\mathsf{Solver}(\mathbf{x}^{(n)}) - \mathsf{DNN}(\mathbf{x}^{(n)}; \mathcal{W})\|_2^2$$

Self-supervised – For inverse problems, reconstruction loss

$$\mathbf{y}^{(n)} \sim \mathbb{P}_Y, \ \min_{\mathcal{W}} \sum_{n=1}^{N} \|\mathbf{y}^{(n)} - \mathsf{DNN}(g(\mathbf{y}^{(n)}); \mathcal{W})\|_2^2$$

Unsupervised – Using objective function as training loss

$$\mathbf{x}^{(n)} \sim \mathbb{P}_X, \ \min_{\mathcal{W}} \sum_{n=1}^{N} E(\mathbf{x}^{(n)}, \mathsf{DNN}(\mathbf{x}^{(n)}; \mathcal{W}); \boldsymbol{\theta})$$

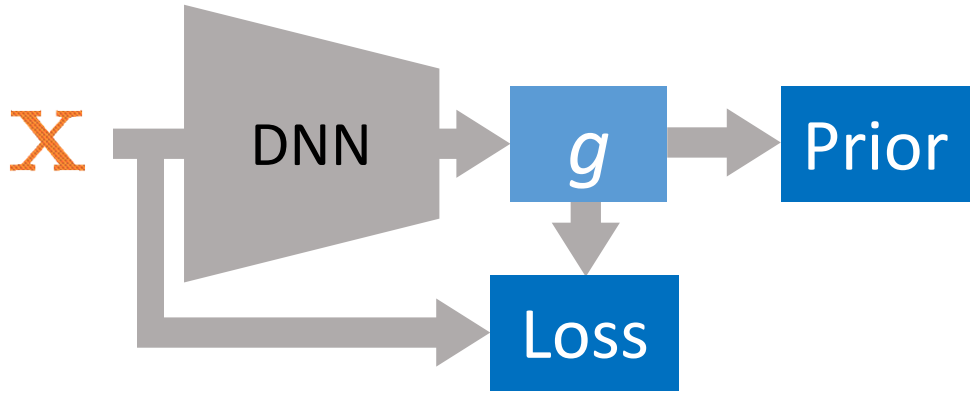
Unsupervised training

Unsupervised – Using objective function as training loss

$$\mathbf{x}^{(n)} \sim \mathbb{P}_X, \ \min_{\mathcal{W}} \sum_{n=1}^{N} E(\mathbf{x}^{(n)}, \mathrm{DNN}(\mathbf{x}^{(n)}, \mathcal{W}); \boldsymbol{\theta})$$

Inverse problem version

$$\min_{\mathcal{W}} \sum_{n=1}^{N} \mathrm{Loss}\Big(\mathbf{x}^{(n)}, g \circ \mathrm{DNN}(\mathbf{x}^{(n)}, \mathcal{W})\Big) + \mathrm{Prior}\Big(\mathrm{DNN}(\mathbf{x}^{(n)}, \mathcal{W})\Big)$$



Unrolling example

Sparse coding: LISTA [Gregor 2010]

Iterative solver: Iterative Soft Thersholding Alg. (ISTA)

$$\mathrm{Iter}(\mathbf{y}) = \sigma_{2\lambda\alpha}\Big((\mathrm{Id} - \alpha A^\top A)\mathbf{y} + \alpha A^\top \mathbf{x}\Big)$$

Learned neural layer (residual block and skip connection)

$$f_k(\mathbf{y}) = \sigma\Big((\mathrm{Id} - V_k)\mathbf{y} + W_k^\top \mathbf{x}\Big),\ \mathcal{W} = \{(W_k, V_k)\}_k$$

- Fully supervised training
- The deeper, the better approximation
- Modest speed-up

Some unsupervised neural solvers

Incompressible fluid simulation: Tompson *et al.* 2017

Multi-scale 3D ConvNet

Personalized 3D face model: Tewari *et al.* 2017

Encoder-decoder with differentiable rendering layer

Artistic style transfer: Ulyanov *et al.* 2016, Johnson *et al.* 2016

Encoder-decoder with “perceptual loss”

Flexible style transfer: Puy & Pérez 2018

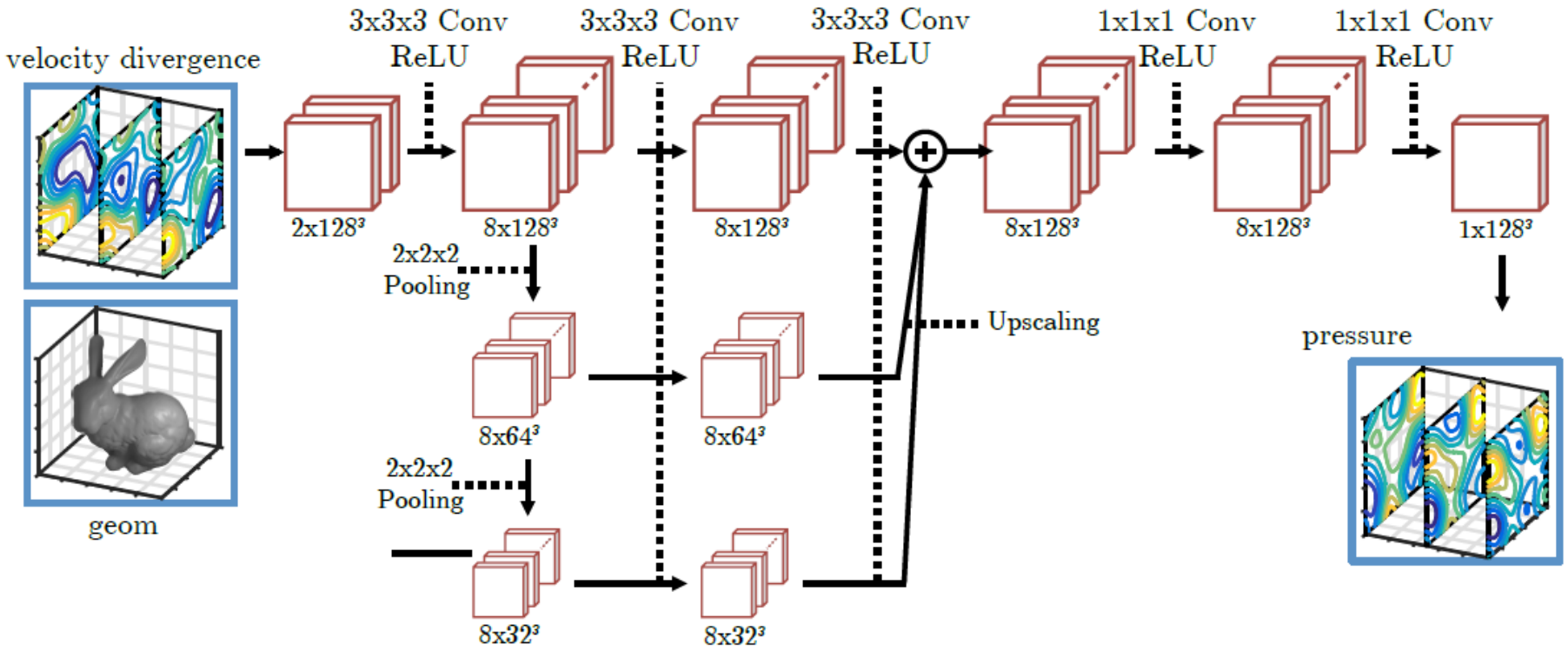
Unrolling descent, run time flexibility

Incompressible fluid simulation [Tompson 2017]

- Replaces Poisson equation on 3D pressure field in Euler advection scheme

$$\mathrm{DNN}(\nabla \cdot \mathbf{u}_t^*, \mathbf{g}_{t-1}; \mathcal{W}) \approx \mathbf{p}_t$$

- Multi-resolution custom ConvNet, unsupervised, loss on velocity div.
- Speed-up w.r.t. PCG: x50



Face pose/appearance estimation

[Tewari 2017]

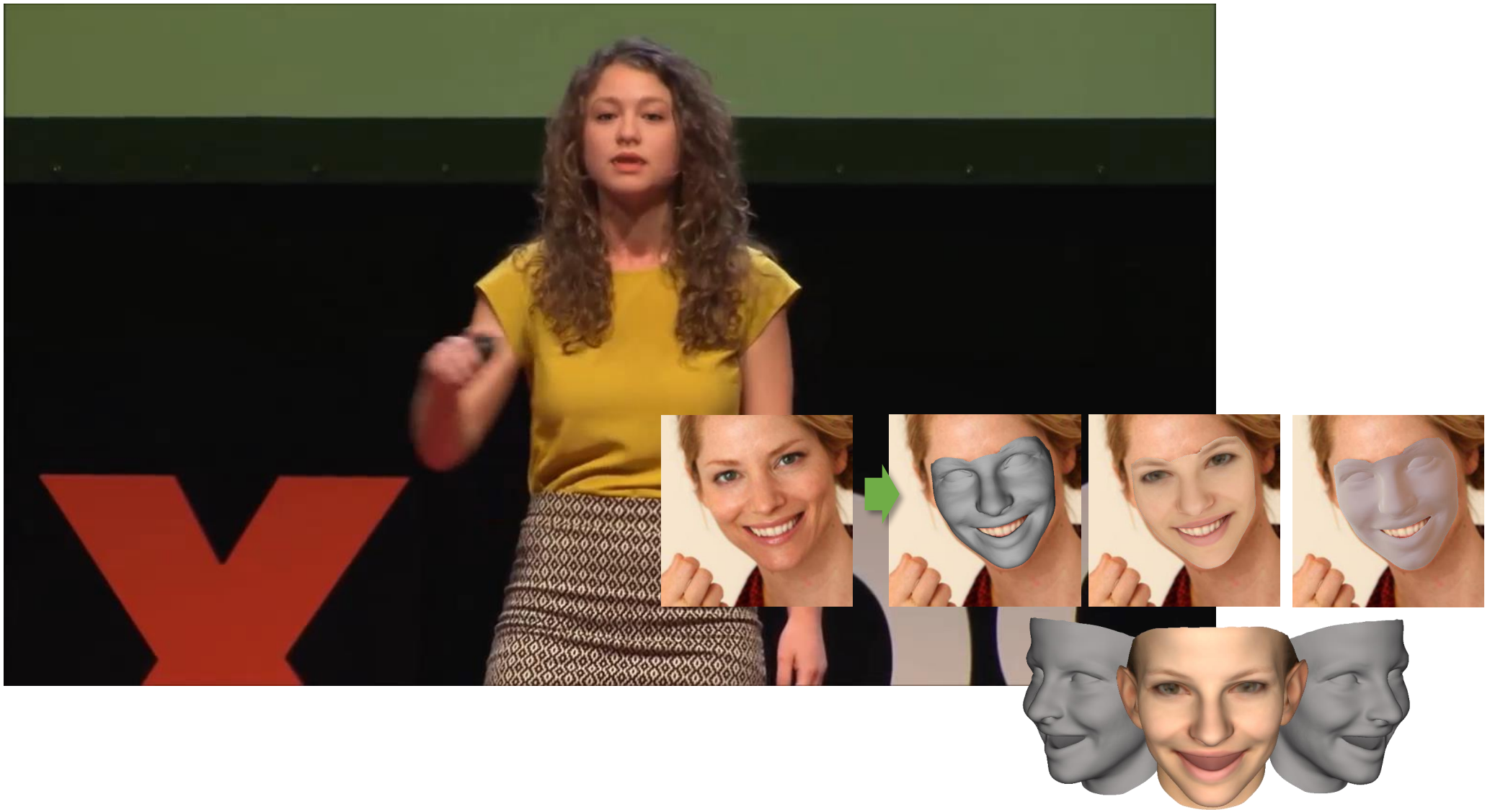
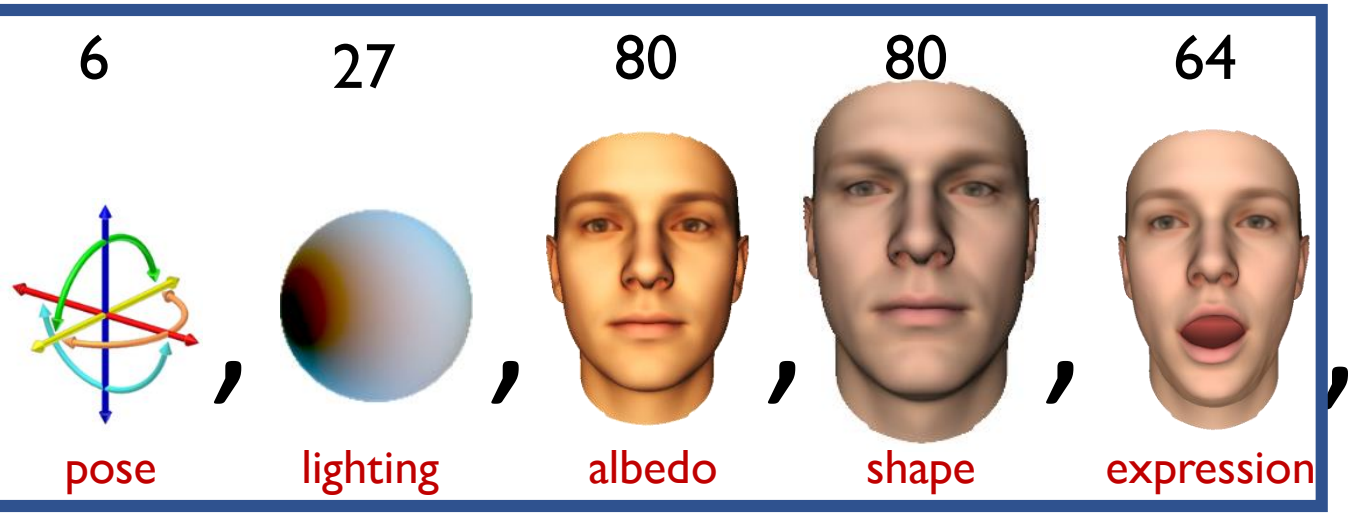


Photo-real face rendering

$$y = ($$  $$)$$

[Garrido 2017-2018]

Photo-real face rendering

$$\mathbf{y} = ($$  $$)$$

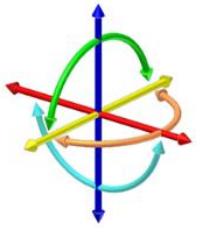
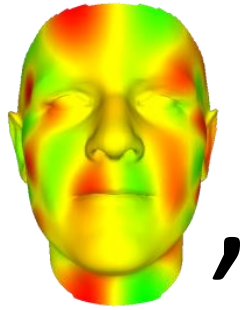
6	27	80	80	64	240	400k
pose	lighting	albedo	shape	expression	correctives	details

$$\mathbf{y} \in \mathbb{R}^{257} \rightarrow g(\mathbf{y}) =$$ 

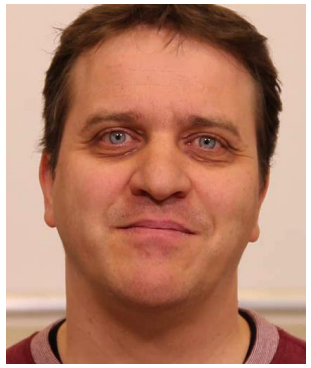
[Garrido 2017-2018]

Invert rendering

to obtain animatable personnalized 3D rig

$$\mathbf{y} = ($$ 6  pose , 27  lighting , 80  albedo , 80  shape , 64  expression , 240  correctives , 400k  details $$)$$

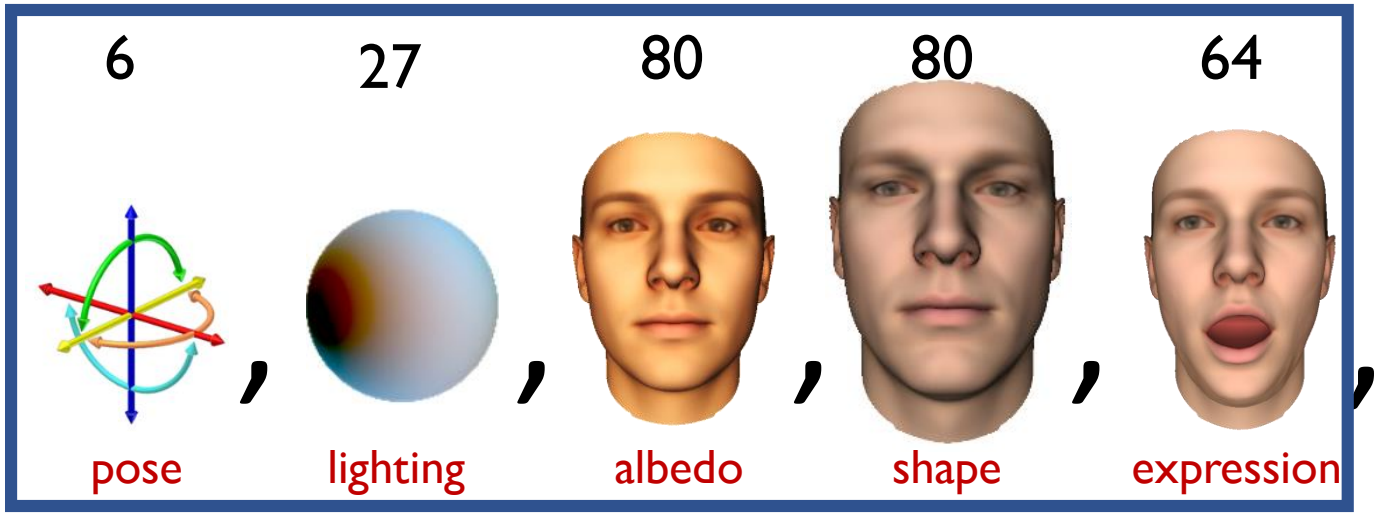
$$\mathbf{x} = \quad \rightarrow \mathbf{y} \in \mathbb{R}^{257}?$$



[Garrido 2017-2018]

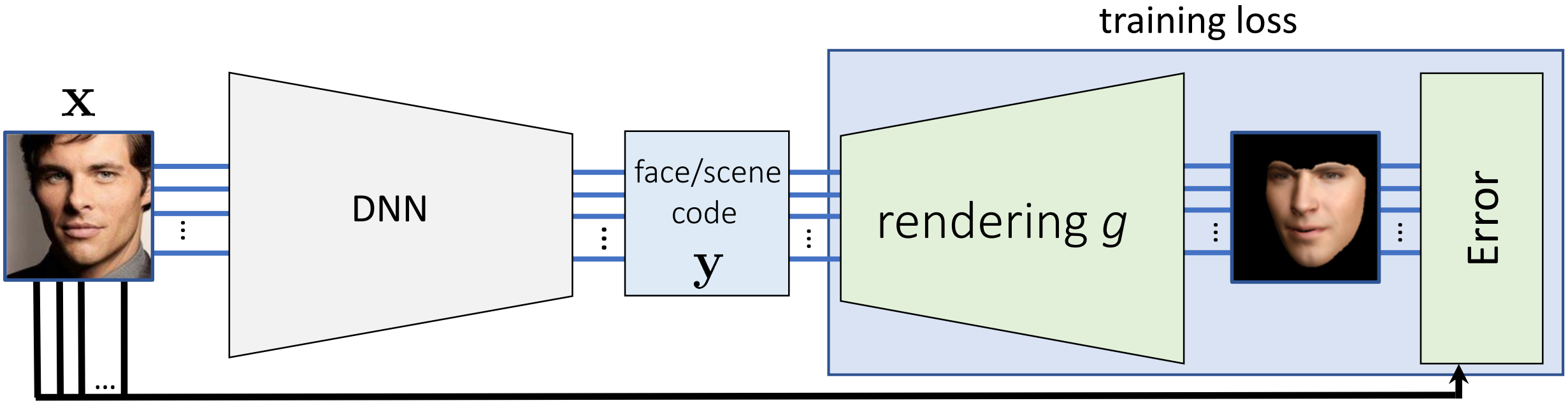
Invert rendering

To obtain animatable personnalized 3D rig

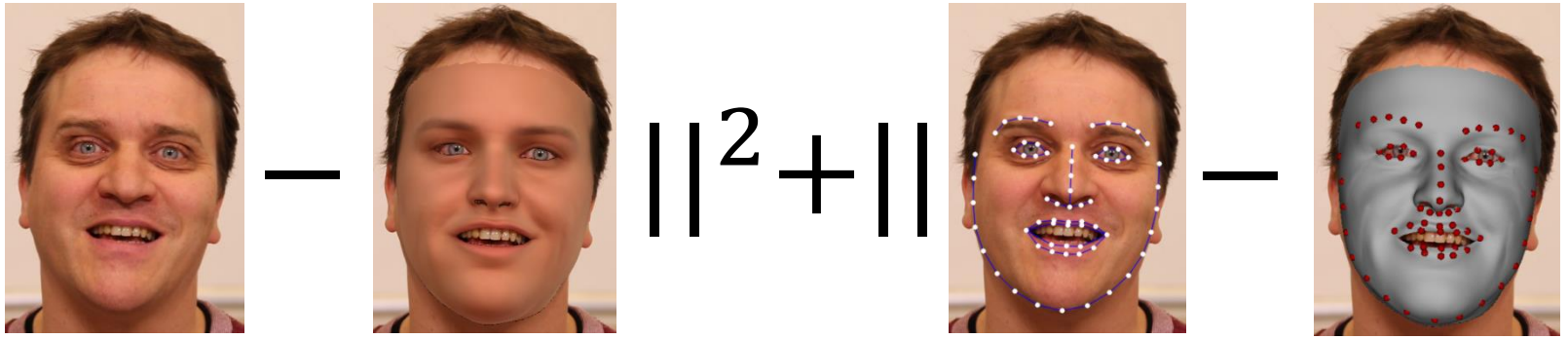
$$y = ($$  $$)$$

[Garrido 2017-2018]

Fast DNN solver



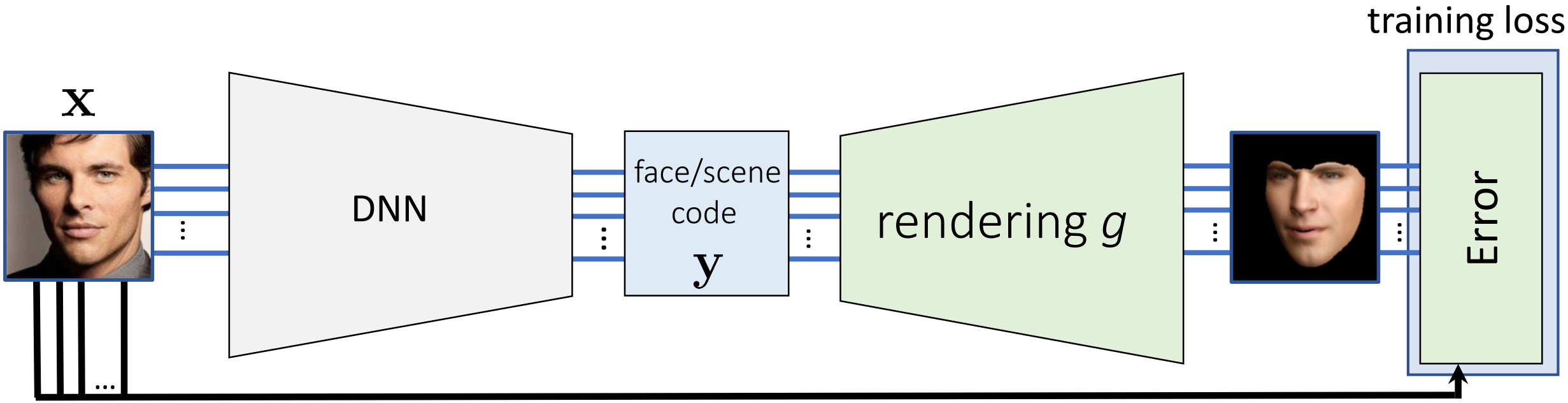
$$\text{Loss} = \| \quad - \quad \|^2 + \| \quad - \quad \|^2$$



observed rendered 2D landmarks correspondences

[Tewari 2017-2018]

Auto-encoder view



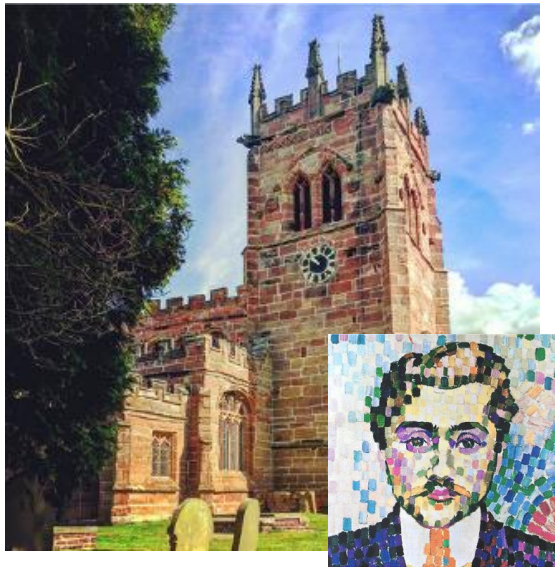
$$\text{Loss} = \| \text{observed} - \text{rendered} \|^2 + \| \text{2D landmarks} - \text{correspondences} \|^2$$

[Tewari 2017-2018]

Fast artistic style transfer

[Ulyanov 2016, Johnson 2016] and successors

- Convolutional encoder-decoder architectures
- Unsupervised training *for specified paintings*

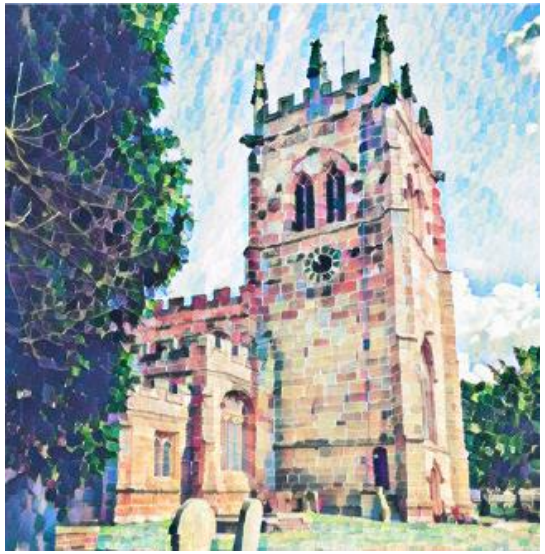


[Ulyanov 2017]

Fast artistic style transfer

[Ulyanov 2016, Johnson 2016] and successors

- Convolutional encoder-decoder architectures
- Unsupervised training *for specified paintings*

[Ulyanov 2017]

Fast flexible style transfer [Puy 2018]

Unrolling (part of) gradient descent

$$E(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \phi_{\ell_0}(\mathbf{y})\|_{\mathsf{F}}^2 + \mu\|\nabla\mathbf{y}\|_1 + \sum_{\ell \in \mathcal{L}_{\mathsf{sty}}} \lambda_\ell \|G_\ell - \phi_\ell(\mathbf{y})^\top \phi_\ell(\mathbf{y})\|_{\mathsf{F}}^2$$

- One layer mimics one step $\mathbf{y} \leftarrow \mathbf{y} - \alpha \nabla E_{\mathsf{sty}}(\mathbf{x}, \mathbf{y})$

$$\mathbf{y}_k = \mathbf{y}_{k-1} - f_k\Big(\mathbf{x}, \mathbf{y}_{k-1}; \mathcal{W}_k, \{\lambda_\ell, G_\ell\}_{\ell \in \mathcal{L}_{\mathsf{sty}}}\Big)$$

modifiable at *run time*

- Unsupervised training

Runtime restructuring

Choose style, mix styles, tune stylization intensity or scale

Runtime restructuring

Add new regularizers via proximal operator, e.g. for photorealism

$$\mathbf{y}_k = \mathbf{y}_{k-1} - \mathrm{Prox}_{\Omega}\left[f_k(\mathbf{x}, \mathbf{y}_{k-1})\right]$$

Runtime restructuring

Add new regularizers via proximal operator, e.g. for photorealism

$$\mathbf{y}_k = \mathbf{y}_{k-1} - \mathrm{Prox}_{\Omega}\left[f_k(\mathbf{x}, \mathbf{y}_{k-1})\right]$$

Runtime restructuring

Add new regularizers via proximal operator, e.g. for photorealism

$$\mathbf{y}_k = \mathbf{y}_{k-1} - \mathrm{Prox}_{\Omega}\left[f_k(\mathbf{x}, \mathbf{y}_{k-1})\right]$$

Conclusion

Neural solvers

- fast, specialized, possibly unsupervised, flexible, differentiable
- can go beyond original model by learning

Other problems, other fields?

- adversarial perturbation
- optimal transport
- PDEs with boundary conditions
- matrix: norms, spectrum, decompositions
- cryptography

Well, how accurate is it? A Study of Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations

N. Thuerey, K. Weissenow, H. Mehrotra, N. Mainali, L. Prantl, Xiangyu Hu

Technical University of Munich

Abstract

With this study we investigate the accuracy of deep learning models for the inference of Reynolds-Averaged Navier-Stokes solutions. We focus on a modernized U-net architecture, and evaluate a large number of trained neural networks with respect to their accuracy for the calculation of pressure and velocity distributions. In particular, we illustrate how training data size and the number of weights influence the accuracy of the solutions. With our best models we arrive at a mean relative pressure and velocity error of less than 3% across a range of previously unseen airfoil shapes. In addition all source code is publicly available in order to ensure reproducibility and to provide a starting point for researchers interested in deep learning methods for physics problems. While this work focuses on RANS solutions, the neural network architecture and learning setup are very generic, and applicable to a wide range of PDE boundary value problems on Cartesian grids.

1 Introduction

that the trained models yield a very high computational performance "out-of-the-box".

A second closely connected goal of our work is to provide a public test bed and evaluation platform for deep learning methods in the context of computational fluid dynamics (CFD). Both code and training data are publicly available at `https://github.com/thunil/Deep-Flow-Prediction` [TMM+18], and are kept as simple as possible to allow for quick adoption for experiments and further studies. As learning task we focus on the inference of Eulerian field functions, more specifically solutions of flow problems on Cartesian grids in terms of velocity and pressure distributions. Deep learning as a tool makes sense in this setting, as the functions were are interested in, i.e. velocity and pressure, are well represented on Cartesian grids, and convolutional layers, as a particularly powerful component of current deep learning methods, are especially well suited for such grids.

Deep learning and the Schrödinger equation

K. Mills*
University of Ontario Institute of Technology

M. Spanner
National Research Council of Canada

I. Tamblyn†
University of Ontario Institute of Technology & National Research Council of Canada

(Dated: November 6, 2017)

We have trained a deep (convolutional) neural network to predict the ground-state energy of an electron in four classes of confining two-dimensional electrostatic potentials. On randomly generated potentials, for which there is no analytic form for either the potential or the ground-state energy, the model was able to predict the ground-state energy to within chemical accuracy, with a median absolute error of 1.49 mHa. We also investigate the performance of the model in predicting other quantities such as the kinetic energy and the first excited-state energy.