Deep Learning in CFD

Ondřej Bublík

NTIS - New Technologies for the Information Society
Faculty of Applied Sciences, University of West Bohemia

December 2022

Motivation

- Full models:
  - solve the equations exactly
  - usually computationally expensive
- Surrogate (approximate) models:
  - similar behaviour to the original model
  - computationally cheaper
- Neural network:
  - highly nonlinear function with free parameters
  - the parameters are set to minimize the loss function
  - hight speed of evaluation
  - can be used as both cases: as a full or surrogate model

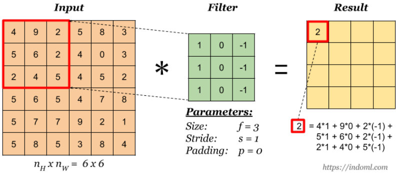
Neural network libraries

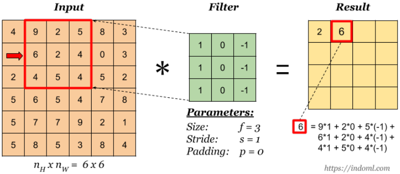
- Keras + Tensorflow:
  - excellent high-level API
  - easy to learn with a simple way to build new architectures
  - highly parallel pipelines with great scalability
  - support for GPU, CPU and TPU
  - trained models could by exported and used by different programming languages
- PyTorch:
  - easy to learn
  - developed natively in Python
  - support for GPU and CPU

Convolution neural network

Convolution layer - convolution operation

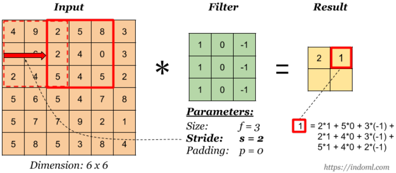
- convolution kernels (filters) slide along input features and provide responses known as feature maps
- shift (space) invariant

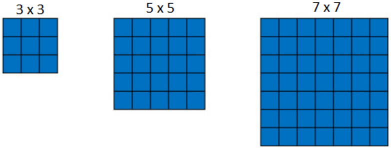


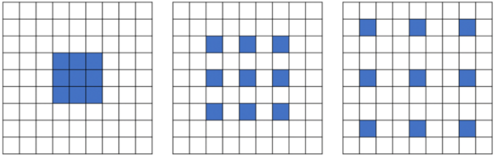


Convolution layer - stride, kernel size, dilation

- stride governs how many cells the filter is moved in the input to calculate the next cell in the result
- larger kernel size leads to better results, but the number of unknowns increases







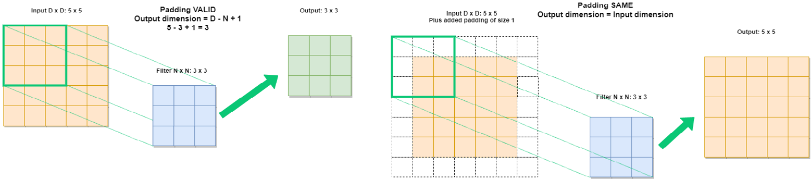
Kernel size: 3X3
Dilation rate: 1

Kernel size: 3X3
Dilation rate: 2

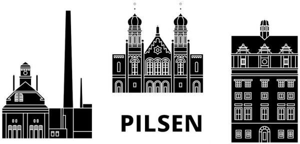
Kernel size: 3X3
Dilation rate: 3

Convolution layer - padding

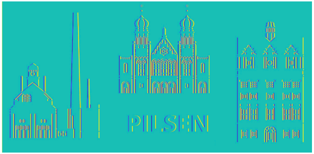
- valid - the dimension of the outgoing feature map is reduced by the kernel size
- same - output feature map has the same dimensions



Convolution layer - examples



$$\begin{bmatrix} 0, 0, 0 \\ 1, 0, -1 \\ 0, 0, 0 \end{bmatrix}$$

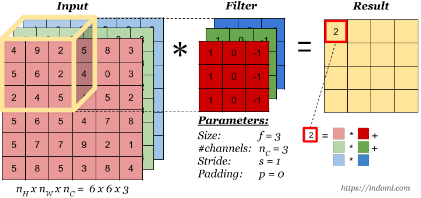


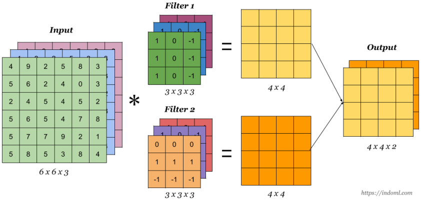
$$\begin{bmatrix} 0, 1, 0 \\ 0, 0, 0 \\ 0, -1, 0 \end{bmatrix}$$

$$\frac{1}{9}\begin{bmatrix} 1, 1, 1 \\ 1, 1, 1 \\ 1, 1, 1 \end{bmatrix}$$

Convolution layer - volume operation

- when the input has more than one channels, the filter should have matching number of channels
- to calculate one output cell, convolution is performed on each matching channel, and the results are add together

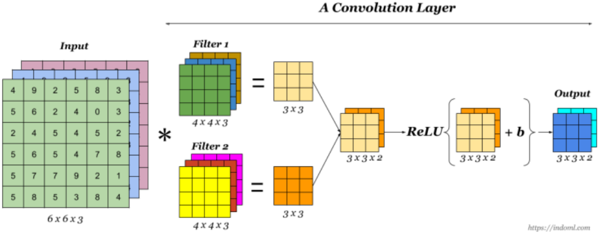




Convolution layer

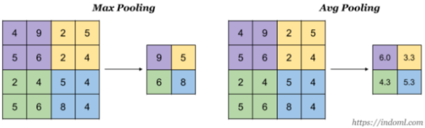
```
tensorflow.keras.layers.Conv2D(filters, frame, activation, padding)
```

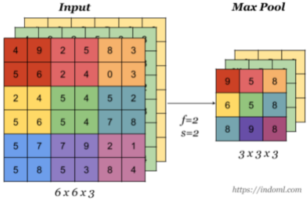
- a bias is added
- activation function such is applied



```
tf.keras.layers.MaxPooling2D(pool_size, strides, padding)

tf.keras.layers.AveragePooling2D(pool_size, strides, padding)
```

- reduce dimensions
- max pooling - get max number
- average pooling - get average number

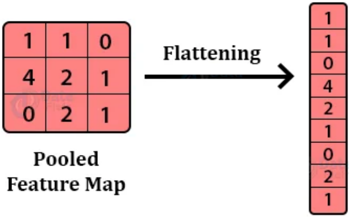




Flatten layer

```
tf.keras.layers.Flatten()
```

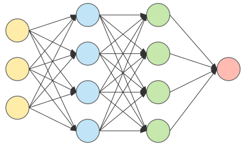
- used to convert the data into 1D arrays to create a single feature vector
- forward the data to a fully connected layer



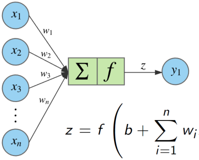
Dense layer

```
tf.keras.layers.Dense(units, activation)
```

- fully connected layers connect every neuron in one layer to every neuron in another layer
- the flattened matrix goes through a fully connected layer to classify the images



Neuron:



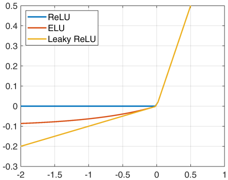
$$z = f\left(b + \sum_{i=1}^{n} w_i x_i\right)$$

Activation functions:

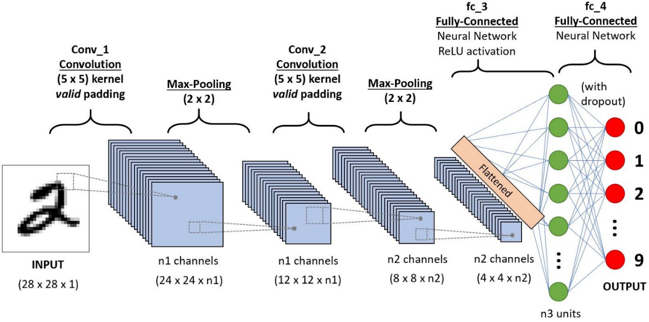
$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha\left(e^x - 1\right), & \text{otherwise} \end{cases}$$

$$\text{Leaky ReLU}(x) = \max(0.1x, x)$$



- used for image/object recognition and classification
- convolutional layer reduces the high dimensionality of images without losing its information

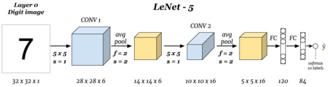


Convolution neural network - training

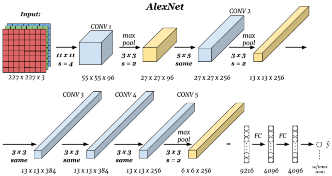
- The training sample has two part:
  - Input: matrix representing image
  - Output: probability vector $[0, 0, \ldots, 1, \ldots, 0]$
- Three sets of samples need to be prepared:
  - training set - used for training
  - validation set - used for error monitoring
  - test set - used for testing
- The loss function is usually defined as a **mean square error** between the predicted and desired output
- The **gradient descent method** is used for loss function minimization
- Various optimizers can be used to get better learning rate: RMSprop, Adam, SGD, ...

Convolution neural network - examples

- handwritten digit recognition
- from *Gradient-Based Learning Applied to Document Recognition* paper by Y. Lecun, L. Bottou, Y. Bengio and P. Haffner (1998)

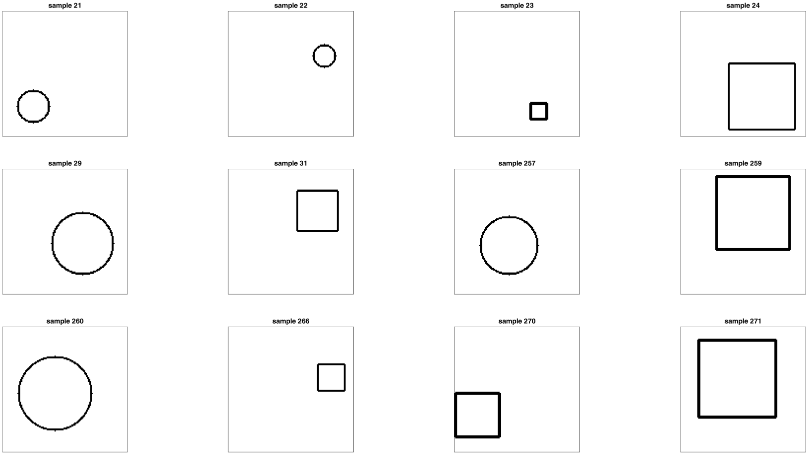


- image recognition
- from *ImageNet Classification with Deep Convolutional Neural Networks* paper by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever (2012)



Convolution neural network - shape classificatory

- image resolution: 128x128
- 1000 random circle/square samples



Convolution neural network - shape classificatory

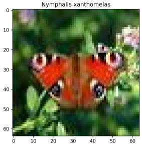
```python
def cnnModel(Input):
    pooling_frame = (2, 2)
    frame = (3, 3)
    act = 'relu'
    n = 8
    deep = 5
    solverDeep = 3
    nOutput = 2

    # decoder
    layer = Input
    for i in range(deep): # 128 x 128 (1)-> 64 x 64 (2)-> 32 x 32 (3)-> 16 x 16 (4)-> 8 x 8 (5)-> 4 x 4 (6)-> 2 x 2
        layer = Conv2D((i+1) * n, frame, activation=act, padding='same')(layer)
        layer = MaxPooling2D(pool_size=pooling_frame)(layer)

    decoded = Flatten()(layer)

    # classificatory
    s = decoded
    for i in range(solverDeep):
        s = Dense(deep * n, activation=act)(s)

    output = Dense(nOutput, activation='sigmoid')(s)

    return output
```

Convolution neural network - Example

- trained model was exported to the javascript
- simple node web server was created
- the image is generated on the frontend
- the prediction is realized using the node in the backend
- the result is send back to the frontend

Convolution neural network - butterfly classification

- Input: coloured picture 64 x 64 pixels
- Output: probability vector
- Considered butterflies:
  - Babočka admirál - Vanessa atalanta
  - Babočka bílé c - Polygonia c album
  - Babočka bodláková - Vanessa cardui
  - Babočka jilmová - Nymphalis polychloros
  - Babočka kopřivová - Aglais urticae
  - Babočka osiková - Nymphalis antiopa
  - Babočka paví oko - Inachis io
  - Babočka síťkovaná - Araschnia levana
  - Babočka vrbová - Nymphalis xanthomelas



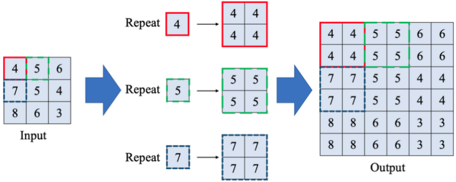
Training set

Encoder-decoder and U-Net

Up sampling layer

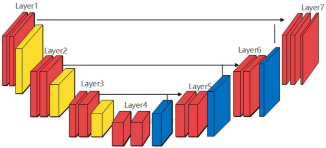
`tf.keras.layers.UpSampling2D(size)`

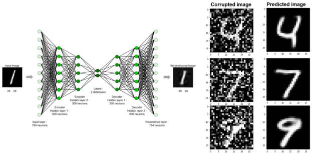
- increases the dimensions

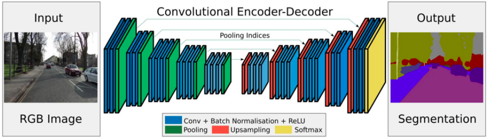


Encoder-decoder, Autoencoder and U-Net

- The output has the same character as the input
- Encoder-decoder:
  - image recognition, detection, and segmentation
- U-net:
  - is Encoder-decoder with skip connection
- Autoencoder:
  - encoder-decoder with unsupervised learning
  - is trained to copy its input to its output
  - used for image denoising, and anomaly detection



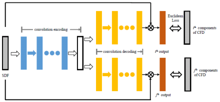


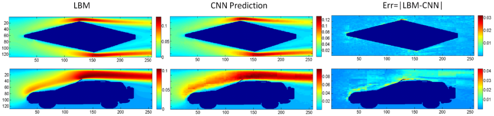


Prediction of steady flow field around airfoil

Neural Network in CFD - Convolution Neural Network

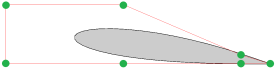
- **Guo, X., Li, W., Iorio, F. Convolutional neural networks for steady flow approximation (2016) Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 13-17-August-2016, pp. 481-490.** (419 citation)
- Convolution Neural Network was trained on the set of lattice Boltzmann simulations to produce the solution according to the boundary information



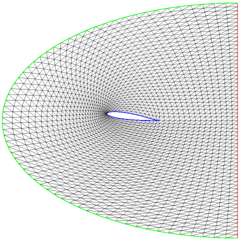


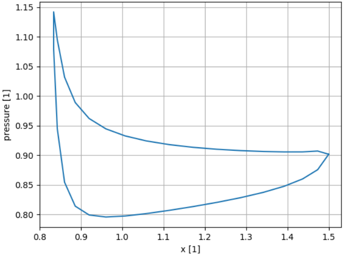
Problem setup

- Inviscid fluid flow around airfoil, angle of attack $\alpha = 0$, Mach number $M_\infty = 0.4$
- Structured C-mesh with 64 x 32 points, generated by elliptic mesh generator
- Airfoil shape is described using the Bezier curve with 8 control points



- First and last points are fixed on the airfoil tail
- Set of 1866 airfoils for various control points positions was created



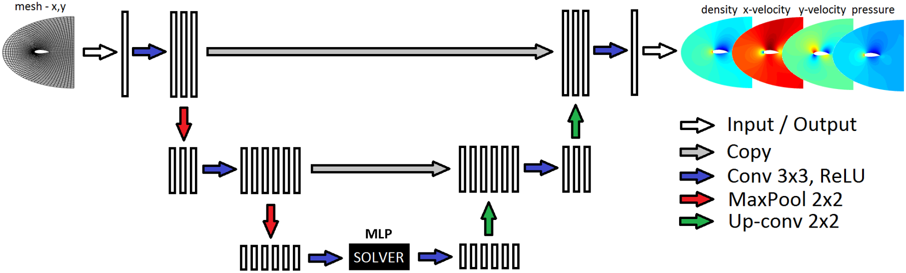


Lift coefficient

$$c_L = \oint_\Gamma p\, n_y \,\mathrm{d}l$$

Convolution Neural Network Architecture

- **Input**: C-mesh with 64 x 32 points
- **Output**: flow field (ϱ, p, u_x, u_y)
- 106 324 trainable parameters
- Trained on the set of 1866 airfoils
- Keras and TensorFlow libraries, python interface

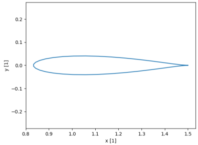


Convolution Neural Network - keras model

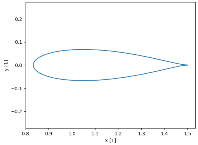
```
def u_net(self, input):
    nSpec, n1, n2, dim = np.shape(input)
    nOutput = self.nOutput

    # hyperparameters
    poolFrame = (2, 2)
    width = 1
    frame = (1 + 2 * width, 1 + 2 * width)
    actFun = 'relu'
    actFunOut = 'linear'
    nDown = 12
    nUp = 12
    deep = int(np.log(min(n1, n2)) / np.log(2))

    # encoder
    layer = input
    conv = [None] * deep   # store layers for concatenation
    for i in range(1, deep):
        conv[i - 1] = Conv2D(i * nDown, frame, activation=actFun, padding='valid', trainable=self.trainableEncoder)(layer)
        layer = MaxPooling2D(pool_size=poolFrame)(conv[i - 1])

    encoded = Conv2D(deep * nDown, frame, activation=actFun, padding='valid', trainable=self.trainableEncoder)(layer)

    # decoder
    layer = encoded
    for i in range(deep - 1, 0, -1):
        layer = UpSampling2D(poolFrame)(layer)
        conc = Conv2D(i * nUp, frame, activation=actFun, padding='valid')(layer)
        layer = concatenate([conc, conv[i - 1]])

    decoded = Conv2D(nOutput, frame, activation=actFunOut, padding='valid')(layer)

    return decoded
```

Results - Tests NACA Airfoils

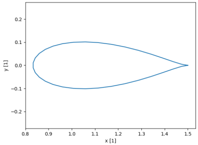
0012



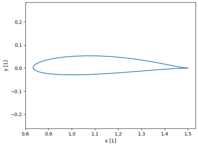
0020



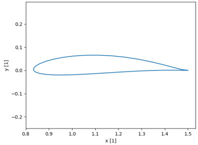
0030



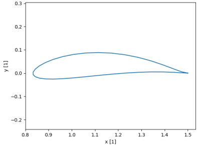
2412



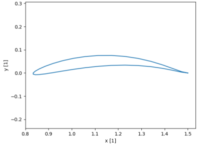
4412



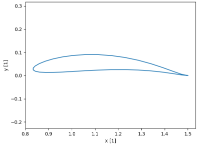
6615



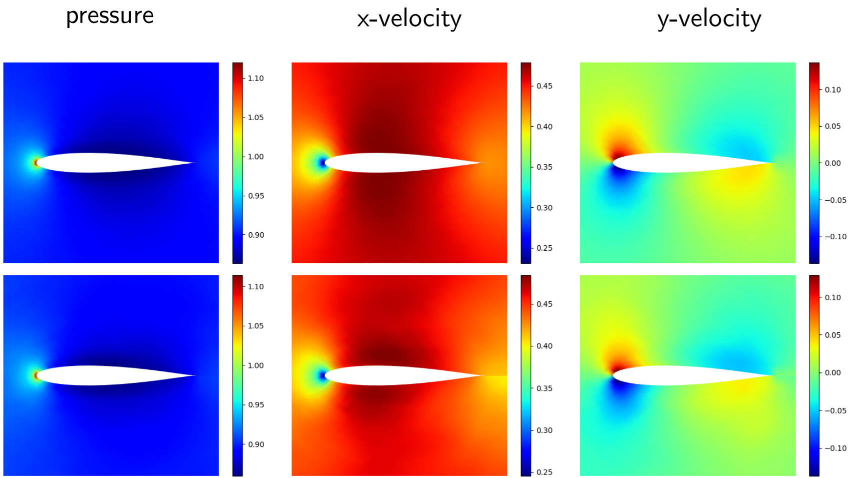
8607



9210

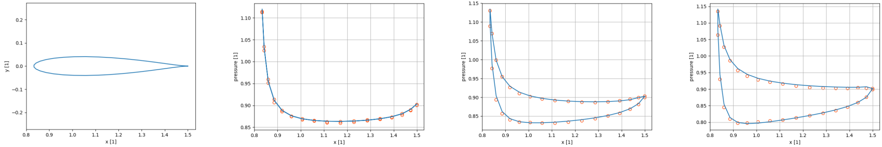


Results - Flow Field (top DNN, bottom CFD)

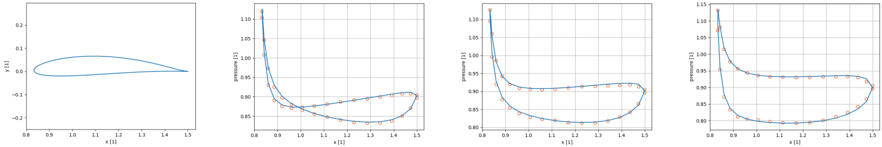


Results - Pressure

0012 | $\alpha = 0°$ | $\alpha = 5°$ | $\alpha = 10°$



4412 | $\alpha = 0°$ | $\alpha = 5°$ | $\alpha = 10°$



Results - Errors

Absolute error $\left|c_L^{CFD} - c_L^{CNN}\right|$ x10^3

α\airfoil	0012	0020	0030	2412	4412	6615	8607	9210
0	0.88	1.52	0.87	0.67	0.34	2.38	1.99	2.26
5	0.93	0.85	0.69	1.57	0.13	3.50	0.96	1.39
10	1.97	0.57	2.68	2.08	2.02	5.37	0.59	0.31
20	4.82	2.93	3.82	4.67	3.98	4.25	4.44	1.72

Relative error $\frac{\left|c_L^{CFD} - c_L^{CNN}\right|}{\left|c_L^{CFD}\right|}$x100

α\airfoil	0012	0020	0030	2412	4412	6615	8607	9210
0	-	-	-	**7.00**	1.78	**9.71**	**5.91**	**5.08**
5	2.57	2.63	2.60	3.45	0.24	**6.12**	2.07	4.76
10	2.89	0.93	**5.30**	2.71	2.39	**6.29**	0.75	0.56
20	4.51	3.00	4.54	4.11	3.32	3.52	3.88	1.92

Conclusion

- Structured mesh with 64 x 32 = 2048 cells
- The test set of 1866 NACA airfoils

CFD solution:

- DG method (FlowPro)
- First order of spatial accuracy
- Total CPU time of 1866 airfoils: **4.5hour**
- CPU time of one solution: **8.7s**
- (CPU time for the second order solution: 26s)

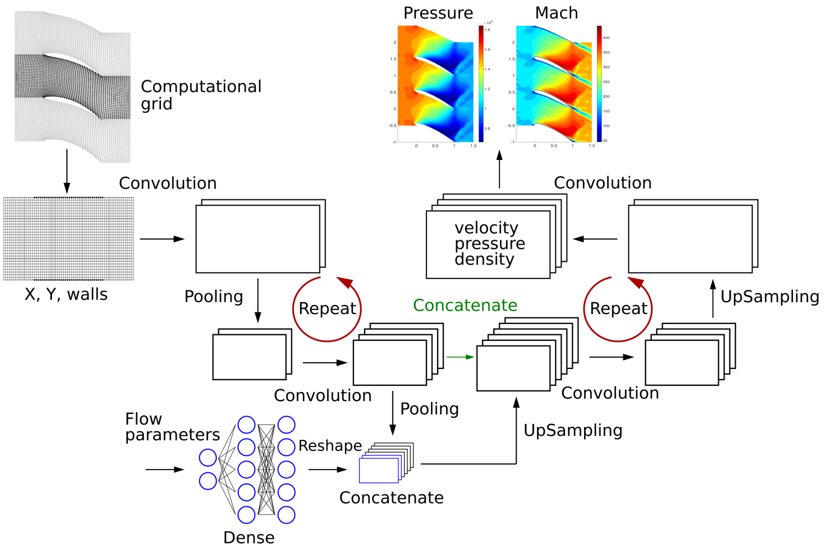
CNN solution:

- Total CPU time of 1866 airfoils: **10.7s**
- CPU time of one solution: **0.0057s**

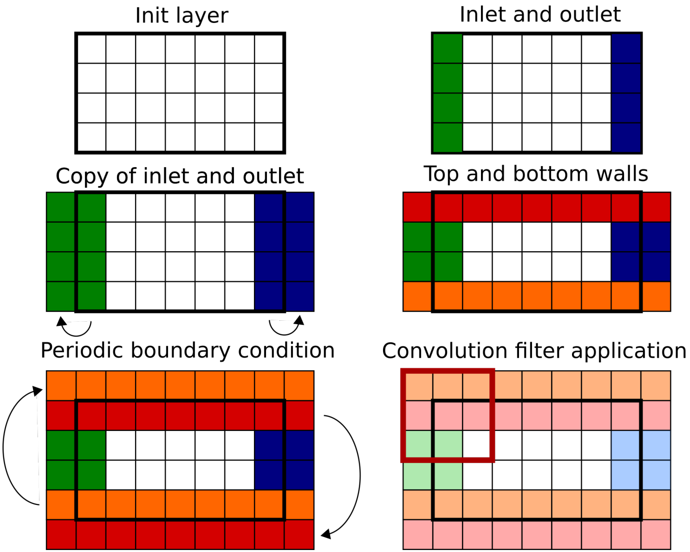
- The convolution neural network provides **1500 times faster** solution than classical CFD solver. (possible 4500 times faster than second order solution)

Prediction of steady flow field in cascade, parametrization

Main Neural Network Architecture - U-net

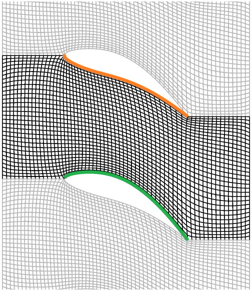


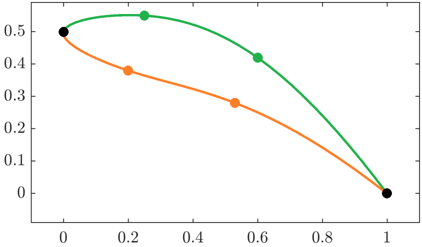
Main Neural Network Architecture - Convolution With Periodic Padding



Problem Setup

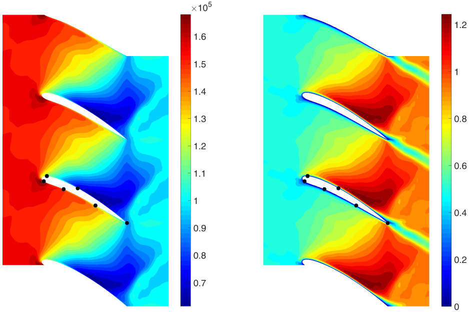
- Laminar fluid flow in blade cascade, angle of attack $\alpha = 10^\circ$
- Mach numbers $Ma = 0.9$
- Reynolds numbers $Re = 10000$
- Structured grid with 64 x 32 points, generated by elliptic mesh generator
- Periodic boundary condition
- Blade shape is described by cubic spline with 6 control points





Neural Network - Summary

- **Input tensor** $[n_{spec}, n_1, n_2, 3]$, $(X, Y, walls)$ grid coordinates and wall markers ($n_1 = 64 \times 32 = n_2$ points)
- **Output tensor** $[n_{spec}, n_1, n_2, 4]$: flow field $(u_x,\ u_y,\ p,\ \varrho)$
- 402 928 trainable parameters
- Trained on the set of 136 random airfoils
- Keras and TensorFlow libraries, Python interface



Convolution Neural Network - keras model

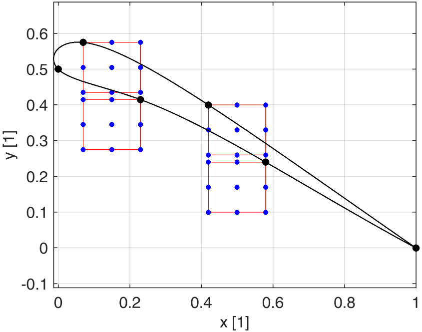
```
def u_net(self, input):
    nSpec, n1, n2, dim = np.shape(input)
    nOutput = self.nOutput

    # hyperparameters
    poolFrame = (2, 2)
    width = 1
    frame = (1 + 2 * width, 1 + 2 * width)
    actFun = 'relu'
    actFunOut = 'linear'
    nDown = 12
    nUp = 12
    deep = int(np.log(min(n1, n2)) / np.log(2))

    # encoder
    layer = input
    conv = [None] * deep    # store layers for concatenation
    for i in range(1, deep):
        conv[i - 1] = Conv2D(i * nDown, frame, activation=actFun, padding='valid', trainable=self.trainableEncoder)(
            self.addPeriodicPadding(layer, width))
        layer = MaxPooling2D(pool_size=poolFrame)(conv[i - 1])

    encoded = Conv2D(deep * nDown, frame, activation=actFun, padding='valid', trainable=self.trainableEncoder)(
        self.addPeriodicPadding(layer, width))

    # decoder
    layer = encoded
    for i in range(deep - 1, 0, -1):
        layer = UpSampling2D(poolFrame)(layer)
        conc = Conv2D(i * nUp, frame, activation=actFun, padding='valid')(self.addPeriodicPadding(layer, width))
        layer = concatenate([conc, conv[i - 1]])

    decoded = Conv2D(nOutput, frame, activation=actFunOut, padding='valid')(self.addPeriodicPadding(layer, width))

    return decoded
```

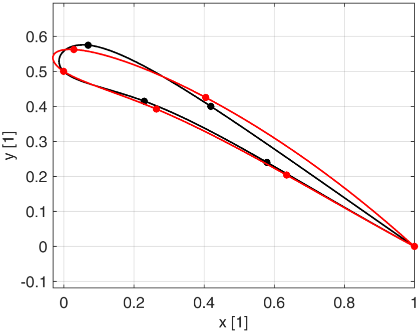
Convolution Neural Network - periodic padding function

```python
def addPeriodicPadding(self, T, width):
    # rows
    bottomRow = tf.slice(T, (0, 0, 0, 0), (tf.shape(T)[0], width, T.shape[2], T.shape[3]))
    topRow = tf.slice(T, (0, T.shape[1]-width, 0, 0), (tf.shape(T)[0], width, T.shape[2], T.shape[3]))
    T = tf.concat((bottomRow, T, topRow), axis=1)

    # colls
    leftCol = tf.slice(T, (0, 0, 0, 0), (tf.shape(T)[0], T.shape[1], width, T.shape[3]))
    rightCol = tf.slice(T, (0, 0, T.shape[2]-width, 0), (tf.shape(T)[0], T.shape[1], width, T.shape[3]))
    T = tf.concat((rightCol, T, leftCol), axis=2)

    return T
```

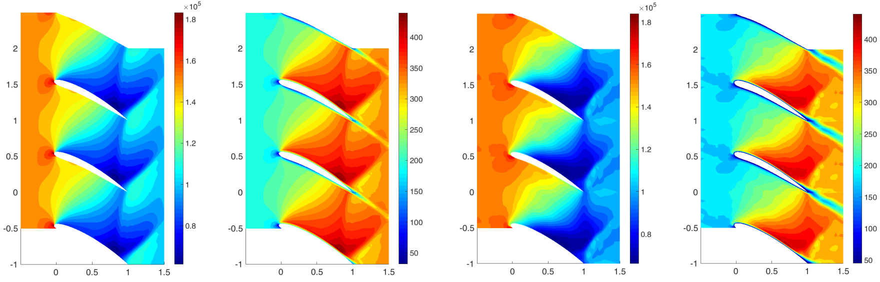
Application - Blade Optimization

- Blade profile optimization for the inlet Mach number $M = 0.95$
- Target functional: $\max(f(\boldsymbol{x})), f(\boldsymbol{x}) = \frac{c_L(\boldsymbol{x})}{1+c_D(\boldsymbol{x})}, \quad c_L = \oint_\Gamma p\, n_y, \quad c_D = \oint_\Gamma p\, n_x$
- Algorithm of optimization:
  - First step roughly search the state space - $9^4 = 6561$ combinations of control points - **13.3s of CPU time**
  - Second step perform 100 steps of gradient descent method - **32s of CPU time**





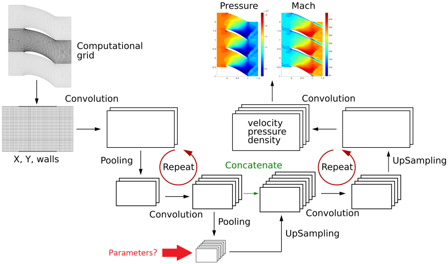
Comparison of flow fields for optimal blade

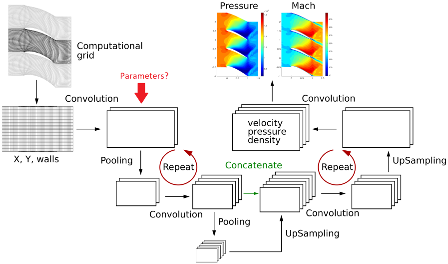


FlowPro (CFD software)

Neural network prediction

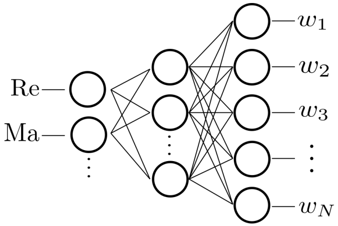
- How to include parameters in the neural network?
- In general, the sooner is the better





Hyper Neural Network

- Used for parametrization of a main network
- Main network is trained for all combinations of flow parameters and resulting weights are stored
- Map flow parameters into main neural network weights
- Dense neural network - one hidden layer

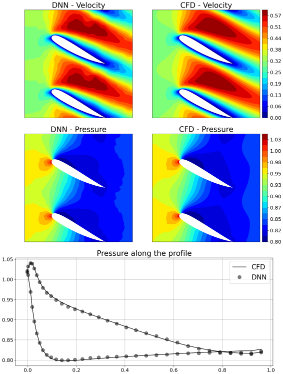


Hyper Neural Network - Single Parameter Re

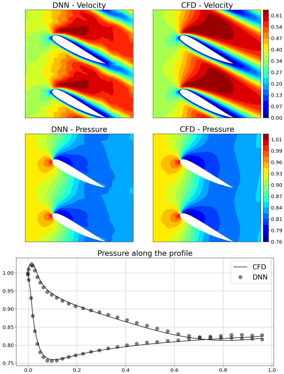
name	symbol	value
heat capacity ratio	κ	1.4
Training Reynolds numbers	Re	100, 500, 1000
Prandtl number	Pr	0.72
pressure ratio	p_{out}/p_{in_0}	0.843
angle of attack	α	15°

Re	Drag average err [%]	Drag SD	Lift average err [%]	Lift SD
100	1.9	1.2	1.1	0.5
250	**3.6**	**3.7**	**4.1**	**1.3**
500	3.9	1.4	2.4	1.4
750	**3.3**	**2.3**	**2.7**	**2.0**
1000	2.9	1.7	3.0	2.3

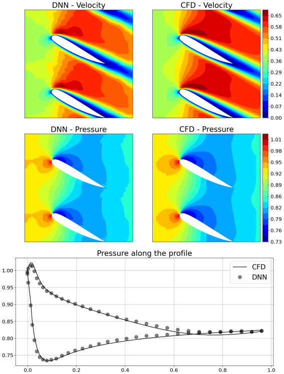
Hyper Neural Network - Single Parameter Re



$Re = 100$

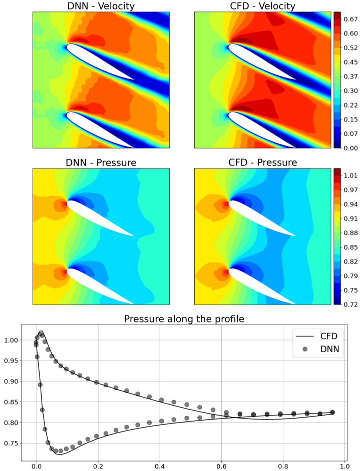


$Re = 250$

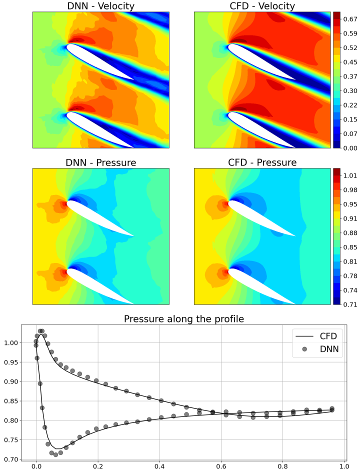


$Re = 500$

Hyper Neural Network - Single Parameter Re



$Re = 750$

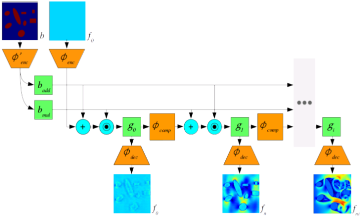


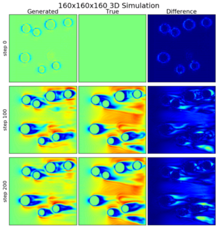
$Re = 1000$

Prediction of unsteady flow field

Neural Network in CFD - Convolution Neural Network

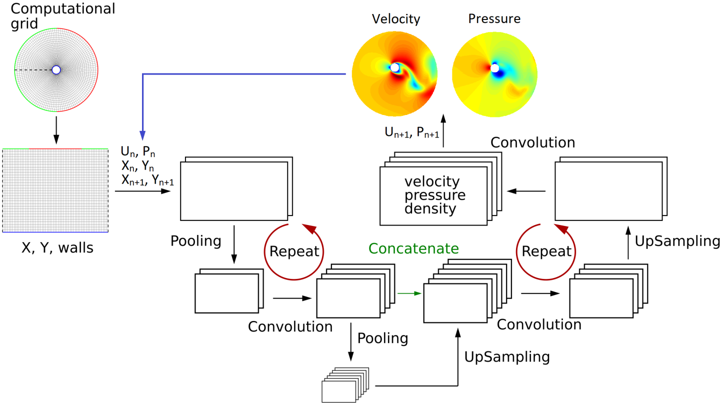
- **Hennigh, O., Lat-Net: Compressing Lattice Boltzmann Flow Simulations using Deep Neural Networks. (2017) arXiv e-prints arXiv:1705.09036** (61 citation)
- Time dependent solution compared with lattice Boltzmann simulation



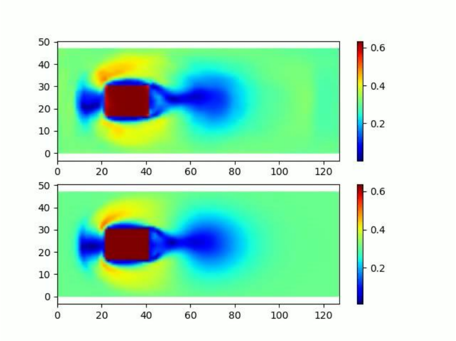


Neural network architecture

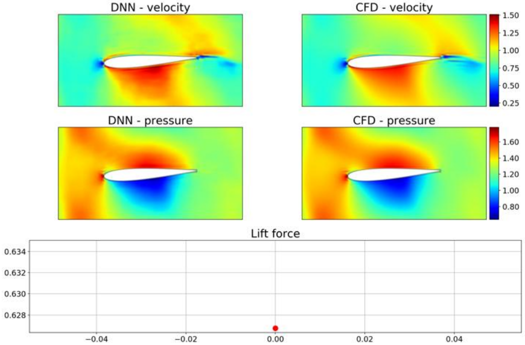
- The architecture is same as in the case of prediction steady flow field
- The solution of n time level is added as another input
- If the mesh is moving, the points coordinates in $n+1$ time level are also added as another input
- The outpus is the solution at $n+1$ time level



Unsteady flow field prediction



Flow field prediction with moving mesh



Vortex induced vibrations

- Structure equation of motion:

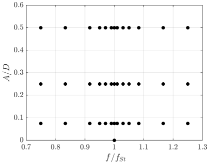
$$\ddot{y} + 2\,\zeta\,\omega_n\,\dot{y} + \omega_n^2\,y = \frac{L}{m}$$

- Lift force:

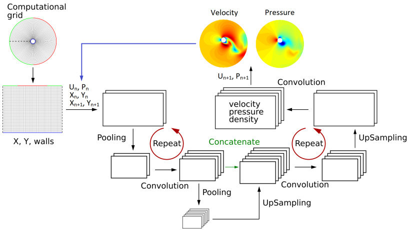
$$L = \oint_{\Gamma} (\sigma_{xx}\,n_x + \sigma_{yx}\,n_y)\;dS$$

- Parameters:
  - Damping ratio: $\zeta = \frac{c}{2m\omega_n}$
  - Stiffness: $k = m\,\omega_n^2$
  - Mass: $m = 10$
  - Damping: $c = 0.25$
  - Natural frequency: $f_n = \frac{\omega_n}{2\pi} = f_{St}$
  - Strouhal frequency: $f_{St} = \frac{\mu}{\varrho_\infty L^2} 0.212(\mathrm{Re} - 21.2)$

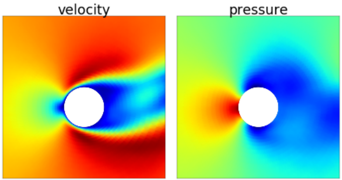
- Convolution neural network predict unsteady flow-field with moving boundary
- Training frequencies and amplitudes:

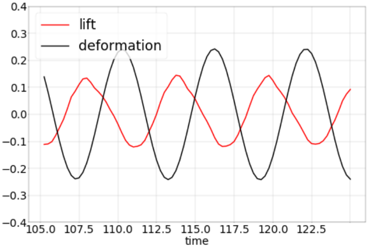


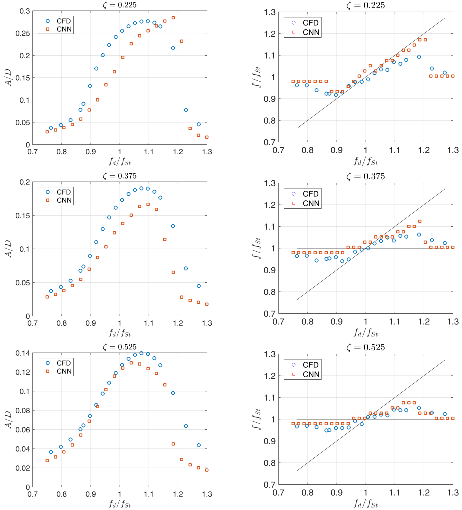
- Convolution neural network architecture:



CNN predicted unsteady flow field







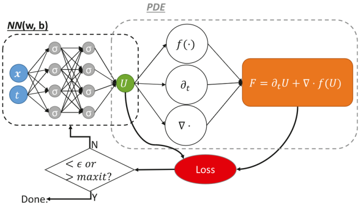
Observation

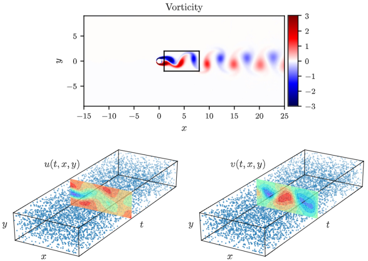
- The output values must be scaled to a range of around ± 1
  - for example: velocity range $[0, 400]$ ms^{-1} or pressure range $[8e^5, 1e^6]$ Pa
  - it is advantageous to consider the equations in dimensionless form
- If more outputs are present, their scales must be comparable
  - for example: dimensionless velocity range $[0, 1]$ is not comparable with dimensionless pressure range $[0.85, 1]$
  - either the data must be rescaled or the weights in the loss function must be taken into account
- The input values scale must be comparable
  - for example: Mach number $[0.1, 1]$ is not comparable with Reynolds number $[100, 1e6]$
  - instead of the real value the logarithm is taken $log_{10}(Re) = [2, 6]$

Physic Informed Neural Network

Neural Network in CFD - Physic Informed Neural Network

- **Raissi M., Perdikaris P., Karniadakis G., Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, 2019, 378, pp. 686â€“707** (2148 citation)
- Fully connected deep neural network is used for the solution approximation
- The loss function is computed according to PDE





Motivation

- Consider neural network as a solution function
- Use PDE in classical or weak form, together with boundary condition for evaluation of loss function $\Rightarrow$ no need of any train data

Motivation

- Consider neural network as a solution function

- Use PDE in classical or weak form, together with boundary condition for evaluation of loss function $\Rightarrow$ no need of any train data

Boundary value problem:

$$\mathcal{L}\left(\boldsymbol{u}, \frac{\partial \boldsymbol{u}}{\partial x_i}, \frac{\partial^2 \boldsymbol{u}}{\partial x_i \partial x_j}, \ldots\right) = \mathbf{0}, \qquad \boldsymbol{x} \in \Omega$$

$$\boldsymbol{u} = \boldsymbol{u}_0, \quad \boldsymbol{x} \in \Gamma_D$$

$$\frac{\partial \boldsymbol{u}}{\partial x_i} n_i = \boldsymbol{c}, \qquad \boldsymbol{x} \in \Gamma_N$$

$$\vdots$$

Motivation

- Consider neural network as a solution function
- Use PDE in classical or weak form, together with boundary condition for evaluation of loss function $\Rightarrow$ no need of any train data

Boundary value problem:

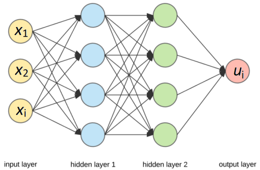
$$\mathcal{L}\left(\boldsymbol{u}, \frac{\partial \boldsymbol{u}}{\partial x_i}, \frac{\partial^2 \boldsymbol{u}}{\partial x_i \partial x_j}, \dots\right) = \mathbf{0}, \qquad \boldsymbol{x} \in \Omega$$

$$\boldsymbol{u} = \boldsymbol{u}_0, \qquad \boldsymbol{x} \in \Gamma_D$$

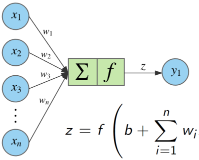
$$\frac{\partial \boldsymbol{u}}{\partial x_i} n_i = \boldsymbol{c}, \qquad \boldsymbol{x} \in \Gamma_N$$

$$\vdots$$

Dense neural network:



Neuron:



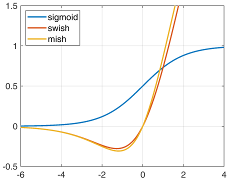
$$z = f\left(b + \sum_{i=1}^{n} w_i x_i\right)$$

Activation functions:

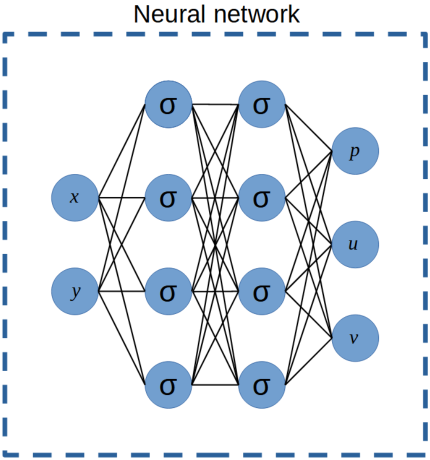
$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\text{swish}(x) = \frac{x}{1 + e^{-x}}$$

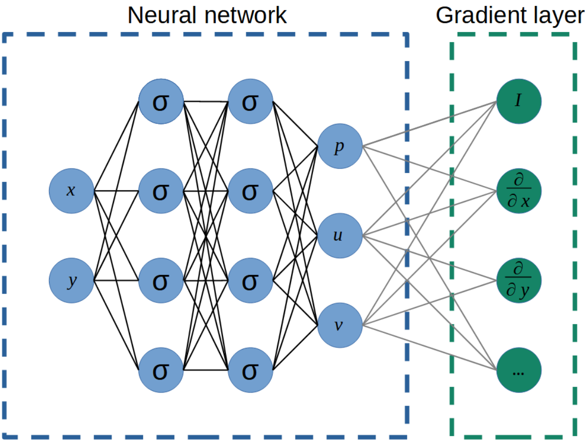
$$\text{mish}(x) = x \tanh(1 + e^{x})$$



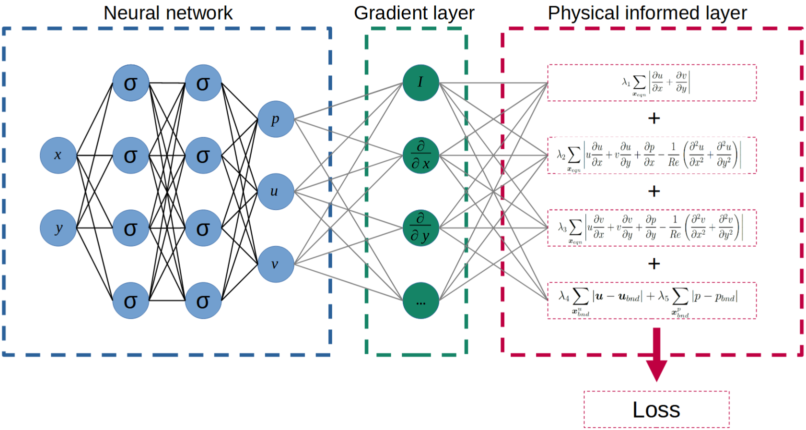
Neural network architecture - dense neural network



Neural network architecture - gradient layer



Neural network architecture - PINN



PINN layers in code

Gradient layer

```python
x, y = [ xy[..., i, tf.newaxis] for i in range(xy.shape[-1]) ]
with tf.GradientTape(persistent=True) as gg:
    gg.watch(x)
    gg.watch(y)
    with tf.GradientTape(persistent=True) as g:
        g.watch(x)
        g.watch(y)
        out = self.model(tf.concat([x, y], axis=-1))
        u = out[..., 0, tf.newaxis]
        v = out[..., 1, tf.newaxis]
        p = out[..., 2, tf.newaxis]
    u_x = g.batch_jacobian(u, x)[..., 0]
    u_y = g.batch_jacobian(u, y)[..., 0]
    v_x = g.batch_jacobian(v, x)[..., 0]
    v_y = g.batch_jacobian(v, y)[..., 0]
    p_x = g.batch_jacobian(p, x)[..., 0]
    p_y = g.batch_jacobian(p, y)[..., 0]
    del g

u_xx = gg.batch_jacobian(u_x, x)[..., 0]
u_yy = gg.batch_jacobian(u_y, y)[..., 0]
v_xx = gg.batch_jacobian(v_x, x)[..., 0]
v_yy = gg.batch_jacobian(v_y, y)[..., 0]
del gg

p_grads = p, p_x, p_y
u_grads = u, u_x, u_y, u_xx, u_yy
v_grads = v, v_x, v_y, v_xx, v_yy

return u_grads, v_grads, p_grads
```

PINN layer

```python
# compute gradients relative to equation
u_grads, v_grads, p_grads = self.grads(xy_eqn)
p, p_x, p_y = p_grads
u, u_x, u_y, u_xx, u_yy = u_grads
v, v_x, v_y, v_xx, v_yy = v_grads

# compute equation loss
continuity = tf.square(u_x + v_y)
u_eqn = tf.square(u*u_x + v*u_y + p_x/self.rho - self.nu*(u_xx + u_yy))
v_eqn = tf.square(u*v_x + v*v_y + p_y/self.rho - self.nu*(v_xx + v_yy))
```

Problem setup

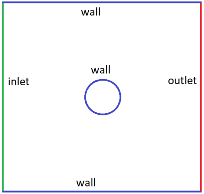
- Equations:

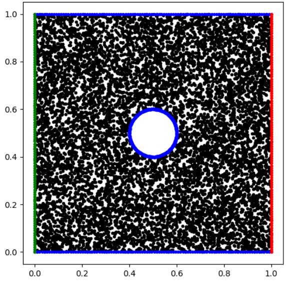
$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0,$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} = \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$

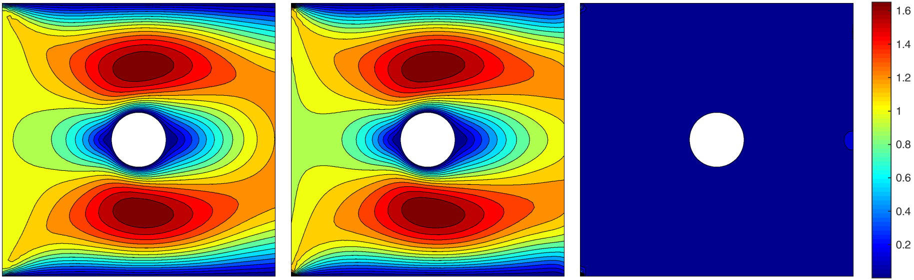
$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial v} = \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

- Boundary conditions:
  - inlet: $u = 1$, $v = 0$
  - outlet: $p = 0$
  - wall: $u = 0$, $v = 0$
- Control points:
  - 10000 equations points
  - 100 inlet points
  - 100 outlet points
  - 700 wall points



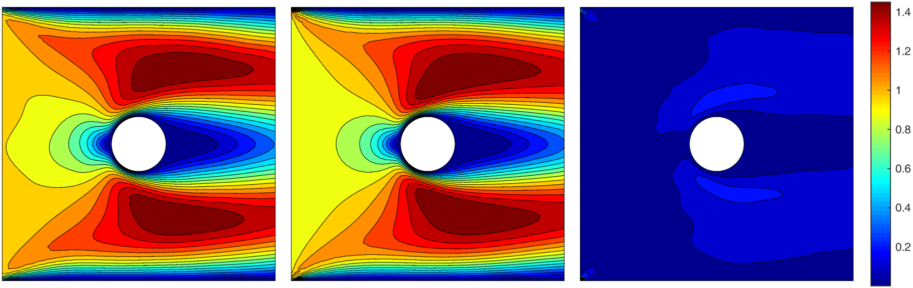


Results - $Re = 10$



PINN CFD difference

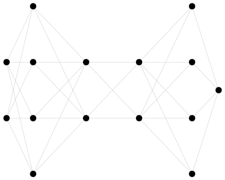
Results - $Re = 100$

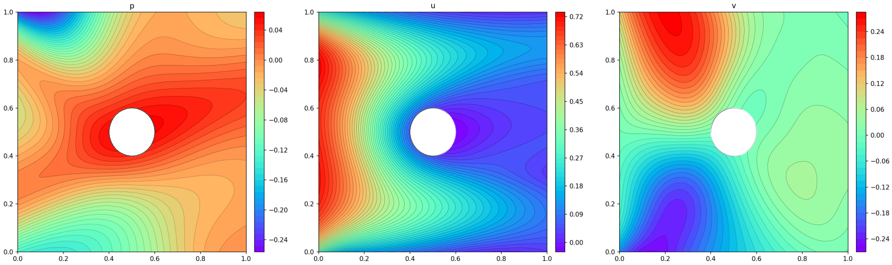


PINN CFD difference

Convergence

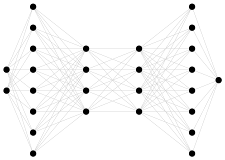
- Dense net [4,2,2,4]
- 28 unknowns
- Error 2.173

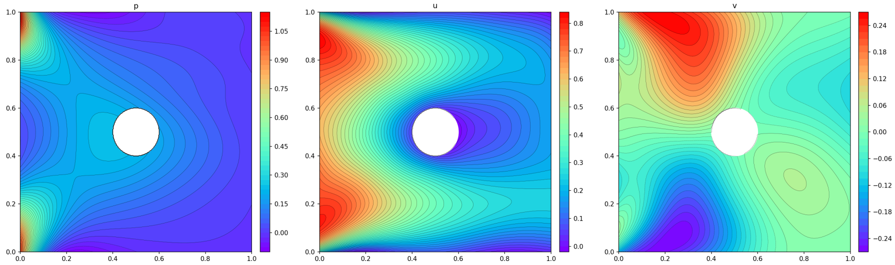




Convergence

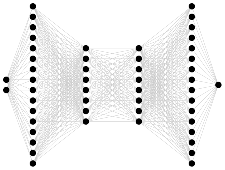
- Dense net [8,4,4,8]
- 96 unknowns
- Error 2.064

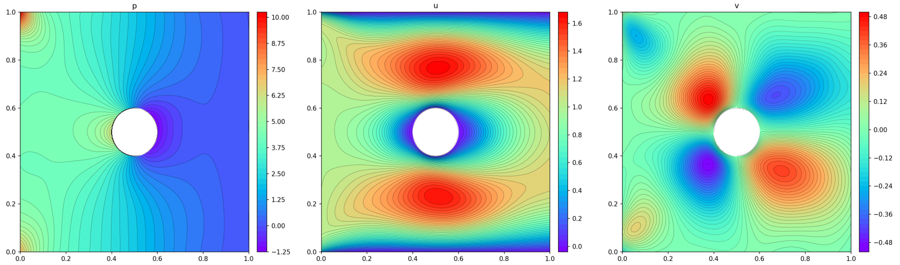




Convergence

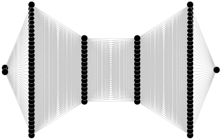
- Dense net [16,8,8,16]
- 352 unknowns
- Error 0.140

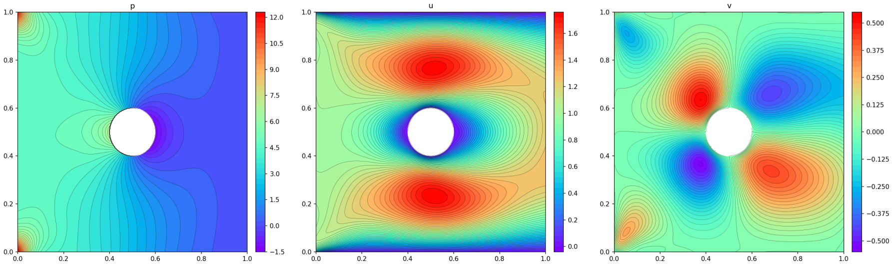




Convergence

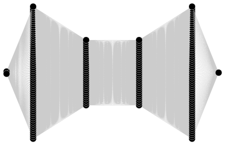
- Dense net [32,16,16,32]
- 1344 unknowns
- Error 0.051

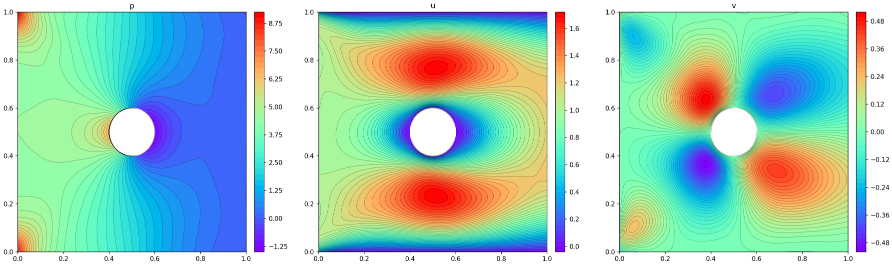




Convergence

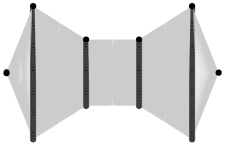
- Dense net [64,32,32,64]
- 5248 unknowns
- Error 0.036

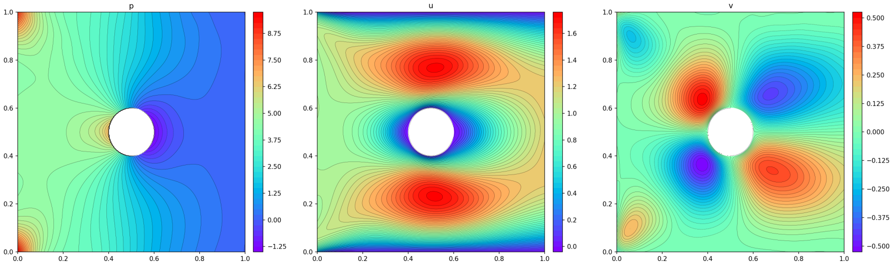




Convergence

- Dense net [128,64,64,128]
- 20736 unknowns
- Error - referential solution





Conclusion

- The loss function is the key to creating a good model.
- In physical modelling, the construction of the loss function can be made with the help of:
  - partial differential equations
  - conservation laws (integral form)
  - constitutive laws
  - ...
- If the loss function does not depend on the training data, the neural network can be said to be a full model.
- The next goal will be to bring physics to the loss function of a convolutional network.