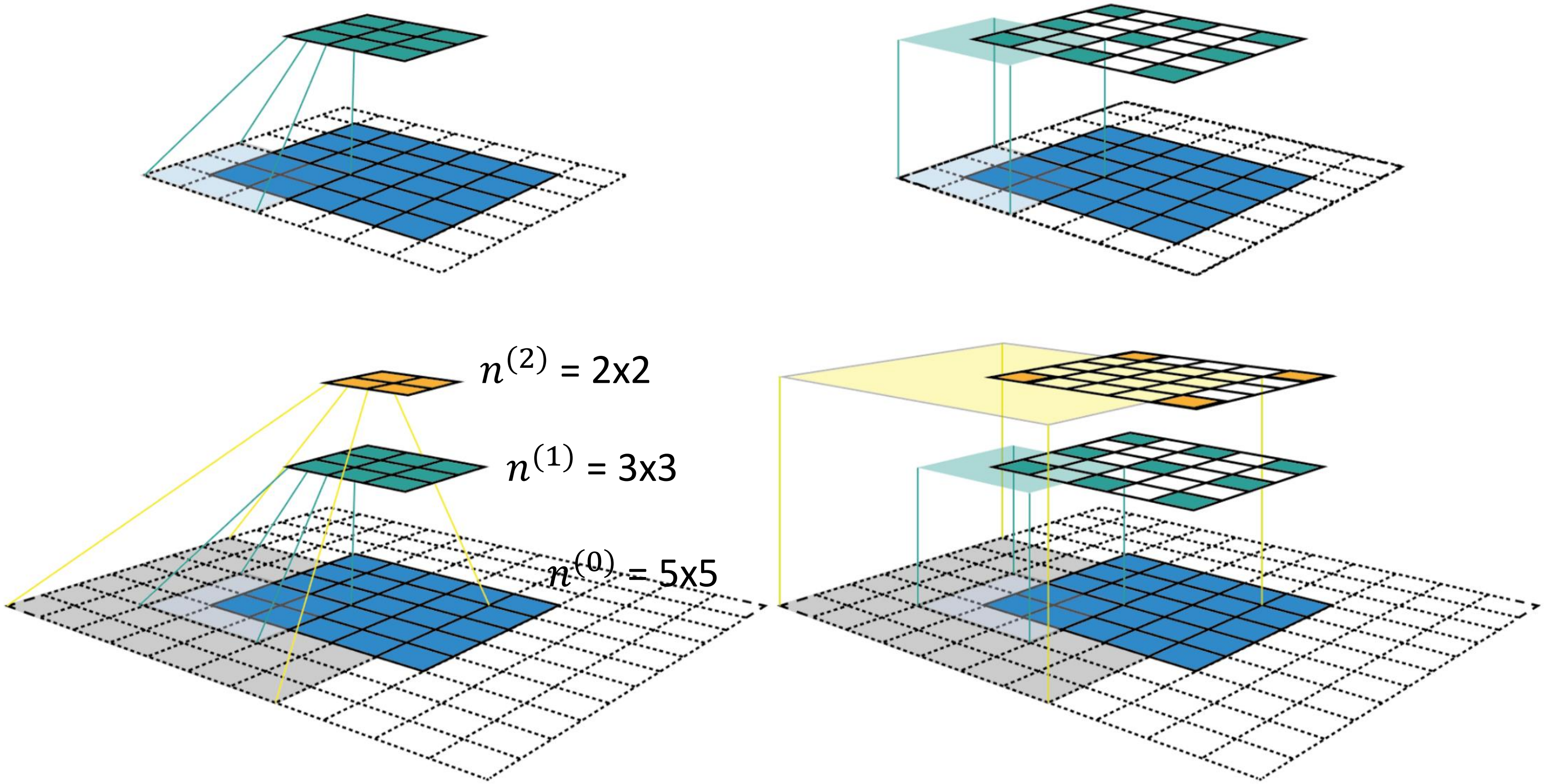
Convolutional neural networks

Receptive field



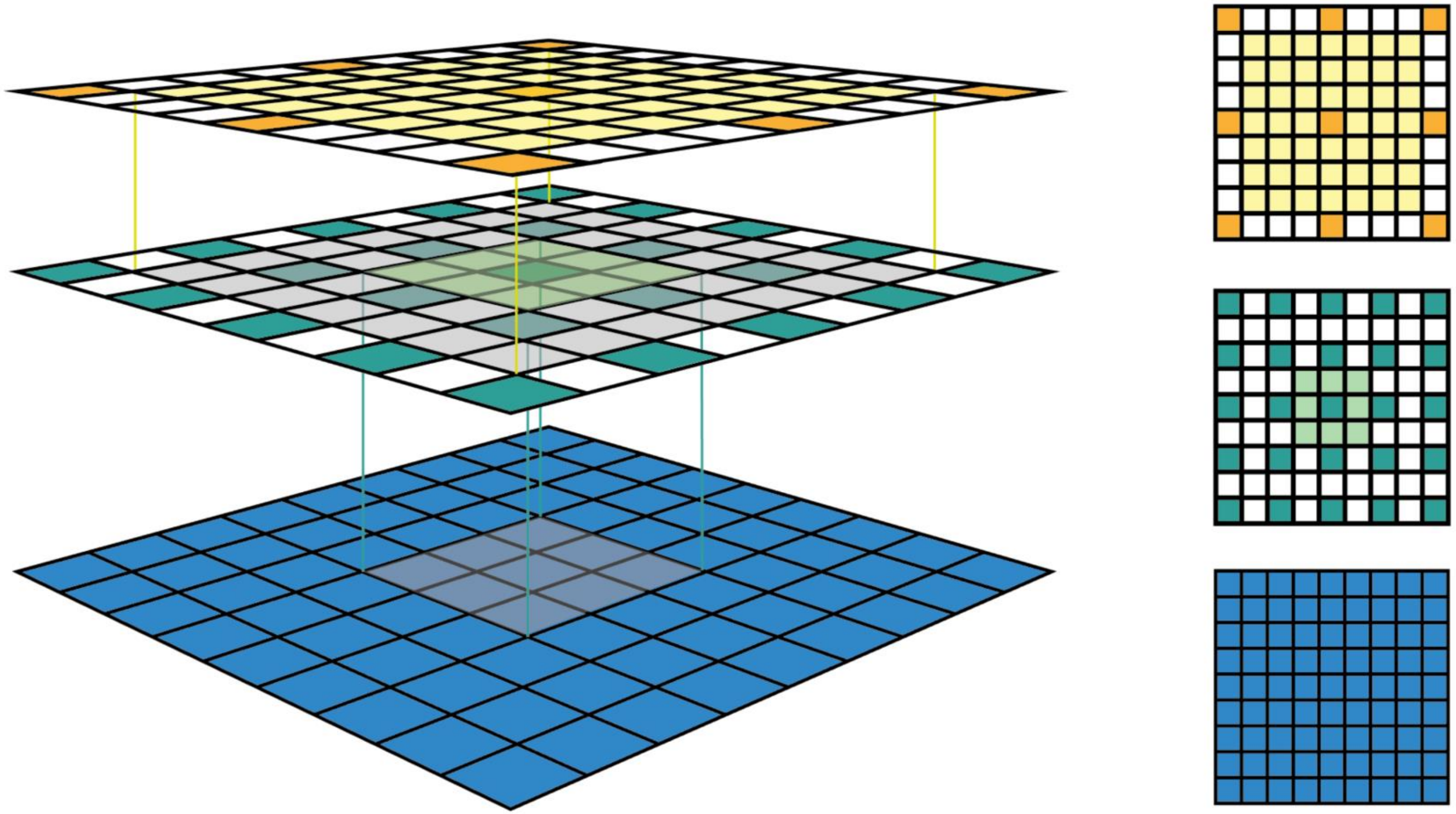
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

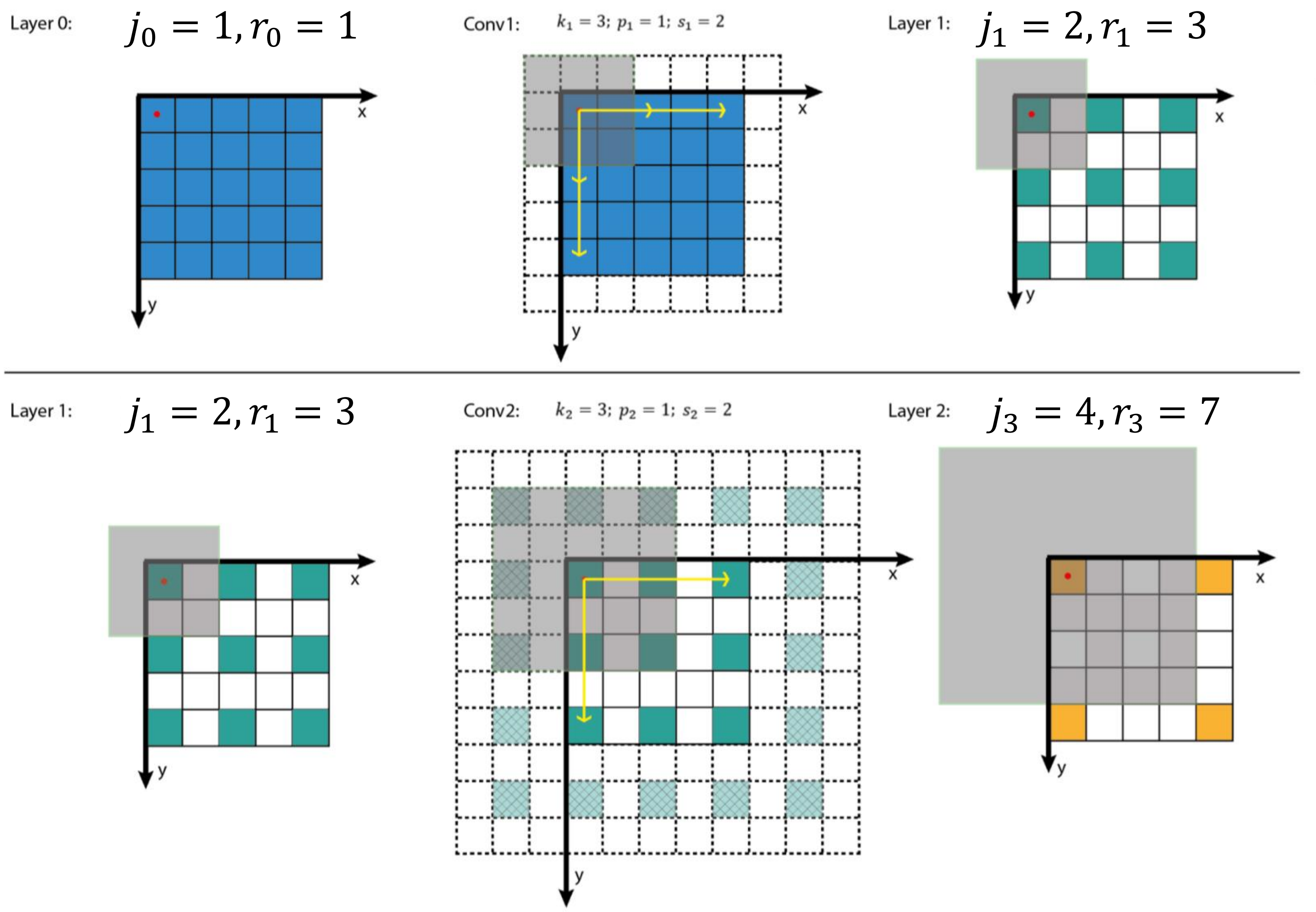
kernel size k = 3x3
padding size p = 1x1
stride s = 2x2

https://blog.mlreview.com/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

Receptive field



Receptive field



n : number of features
r : receptive field size
j : jump (distance between two consecutive features)
$start$: center coordinate of the first feature

k : convolution kernel size
p : convolution padding size
s : convolution stride size

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

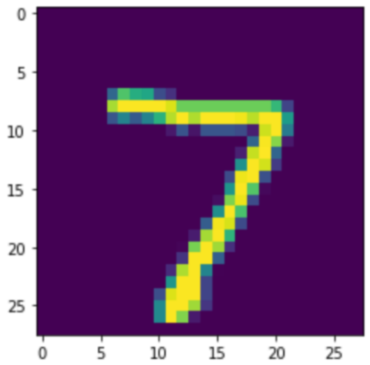
$$j_{out} = j_{in} * s$$

$$r_{out} = r_{in} + (k-1) * j_{in}$$

Receptive field

```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
reshape_1 (Reshape)          (None, 28, 28, 1)         0
________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 4)         40
________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 4)         0
________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 8)         296
________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 8)           0
________________________________________________________________
flatten_1 (Flatten)          (None, 392)               0
________________________________________________________________
dense_2 (Dense)              (None, 32)                12576
________________________________________________________________
dense_3 (Dense)              (None, 10)                330
================================================================
Total params: 13,242
Trainable params: 13,242
Non-trainable params: 0
________________________________________________________________
```

$$j_0 = 1, r_0 = 1$$
$$j_1 = 1, r_1 = 3$$
$$j_2 = 2, r_2 = 4$$
$$j_3 = 2, r_3 = 8$$
$$j_4 = 4, r_4 = 10$$



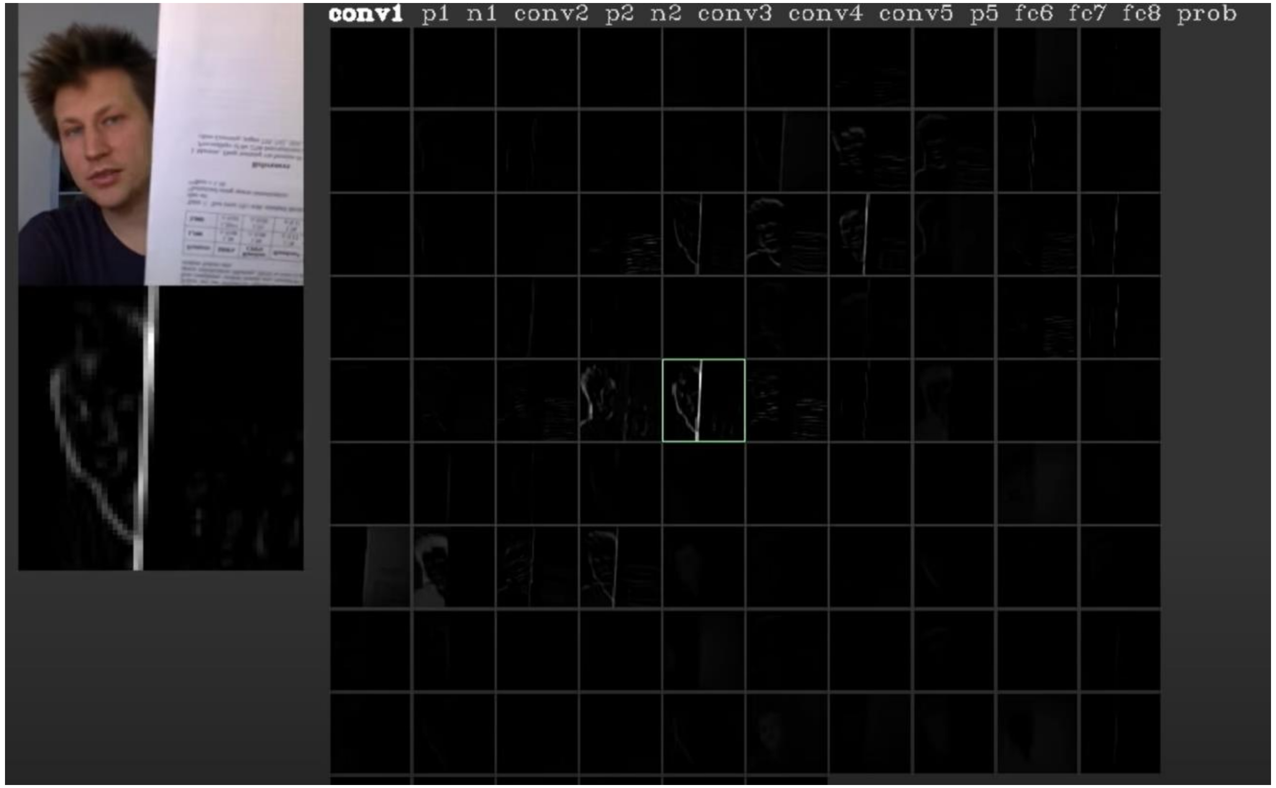
n : number of features
r : receptive field size
j : jump (distance between two consecutive features)
$start$: center coordinate of the first feature

k : convolution kernel size
p : convolution padding size
s : convolution stride size

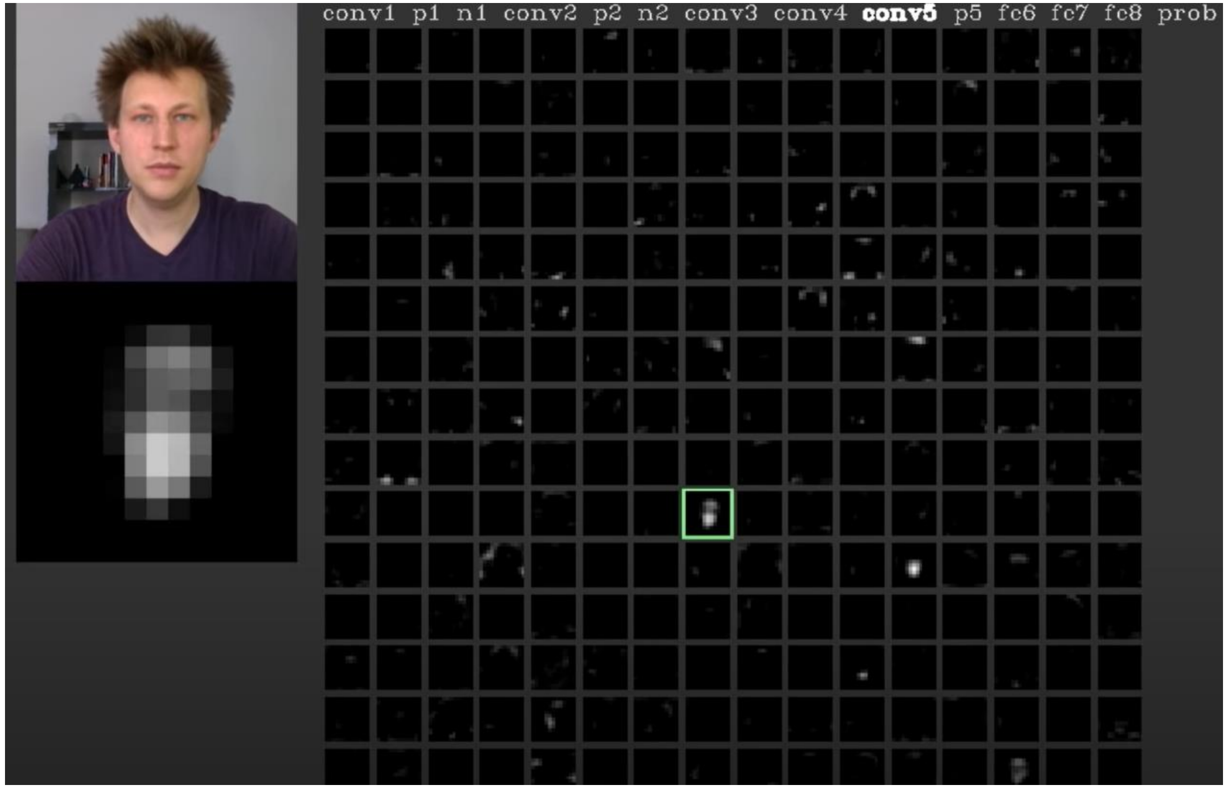
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$
$$j_{out} = j_{in} * s$$
$$r_{out} = r_{in} + (k - 1) * j_{in}$$

Shallow vs deep conv filters

First conv filters learns features that can be characterized locally (e.g. edges)



Deeper conv filters learn abstract concepts that are related to larger, global patterns



http://yosinski.com/deepvis

Assessing model performance

	Reality: covid	Reality: no covid
Prediction: positive	True Positive (TP)	False Positive (FP)
Prediction: negative	False Negative (FN)	True Negative (TN)

$$\text{Accuracy} = \frac{\text{\# of correct predictions}}{\text{Toatl \# of predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{\# of correct positive predictions}}{\text{\# of positive predictions}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{\# of correct positive predictions}}{\text{\# of positive realities}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

or sensitivity

$$\text{Specificity} = \frac{\text{T}N}{\text{TN} + FP}$$

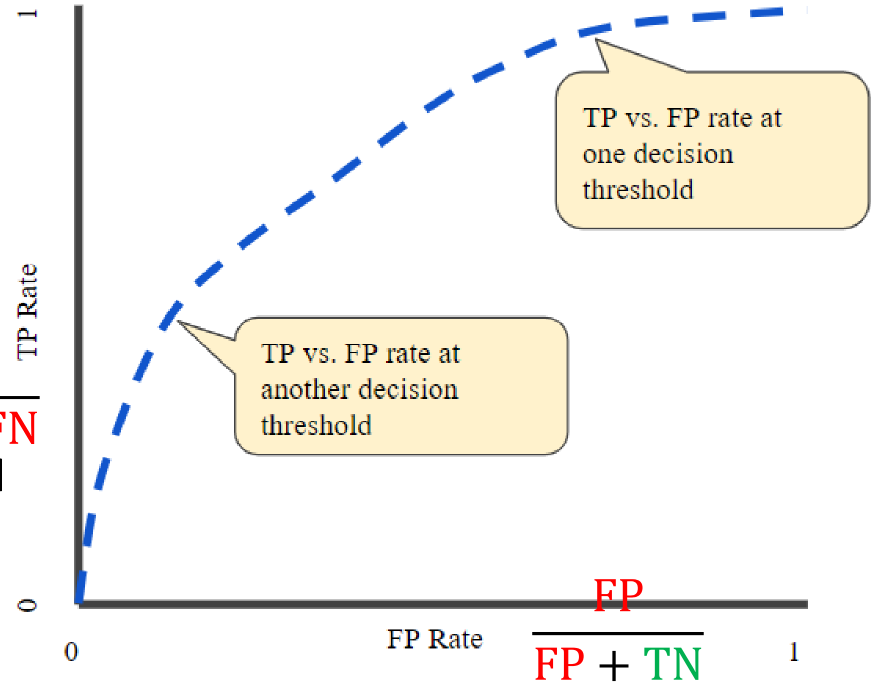
AUC = the probability that the model ranks an actual positive example more highly than an actual negative example

https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

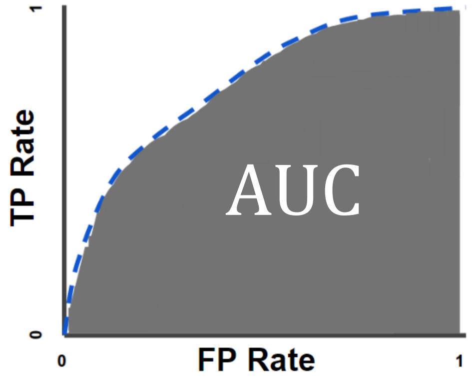
ROC curve (receiver operating characteristic curve)

$$\frac{\text{TP}}{\text{TP} + \text{FN}}$$

recall

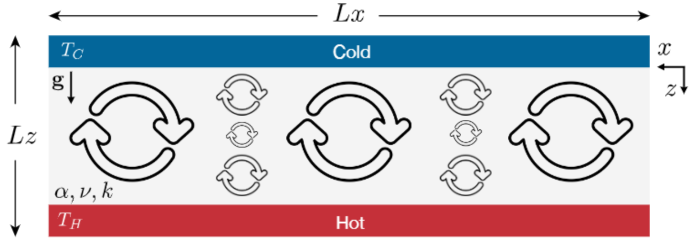


$$\frac{\text{FP}}{\text{FP} + \text{TN}}$$

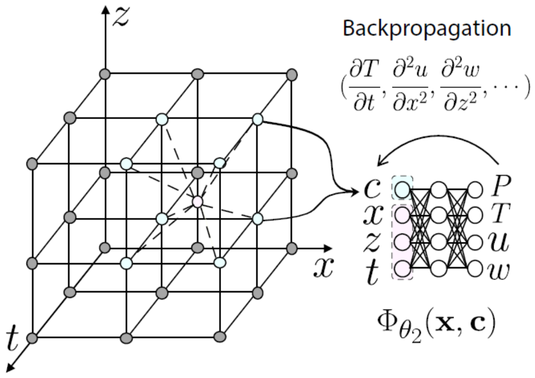


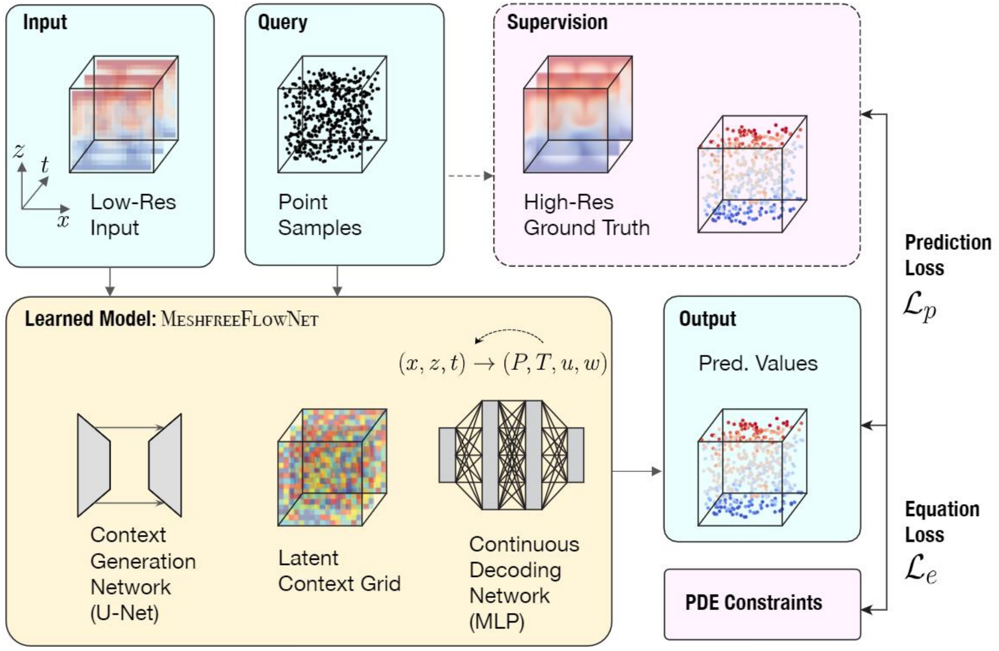
Jiang et al (2020)

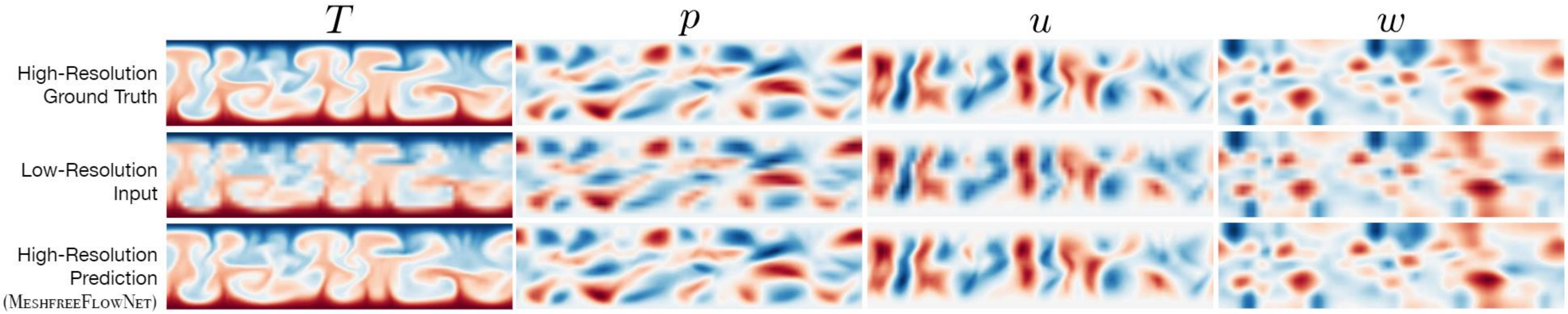
CNN for super-resolution



$$\nabla \cdot \mathbf{u} = 0\,,$$
$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - T\hat{z} - R^* \nabla^2 \mathbf{u} = 0\,,$$
$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T - P^* \nabla^2 T = 0\,,$$

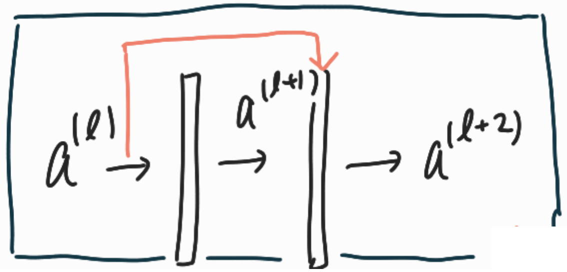






ResNet

Residual block :



short cut / skip connections

$\vec{a}^{(l)} \longrightarrow$ linear $\longrightarrow$ ReLU $\longrightarrow$ linear $\longrightarrow$ ReLU $\longrightarrow \vec{a}^{(l+2)}$

σ

$\vec{z}^{(l+1)} = \vec{W}^{(l+1)} \vec{a}^{(l)} + \vec{b}^{(l+1)}$ $\quad \vec{a}^{(l+1)} = \sigma(\vec{z}^{(l+1)})$ $\quad \vec{z}^{(l+2)} = \vec{W}^{(l+2)} \vec{a}^{(l+1)} + \vec{b}^{(l+2)}$ $\quad \cancel{\vec{a}^{(l+2)} = \sigma(\vec{z}^{(l+2)})}$

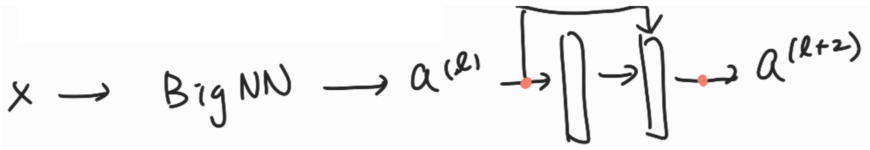
main path

$\vec{a}^{(l+2)} = \sigma(\vec{z}^{(l+2)} + \vec{a}^{(l)})$

The "short cuts" pass info deeper into the NN.

He et al (2015) : ResNet $\rightarrow$ allows you to train much deeper NN

ResNet

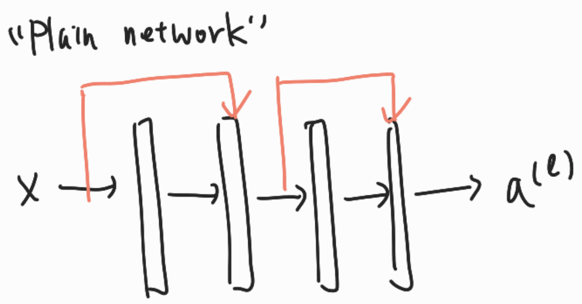


Use $\sigma(z) = \text{ReLU}$, outputs $a \geq 0$

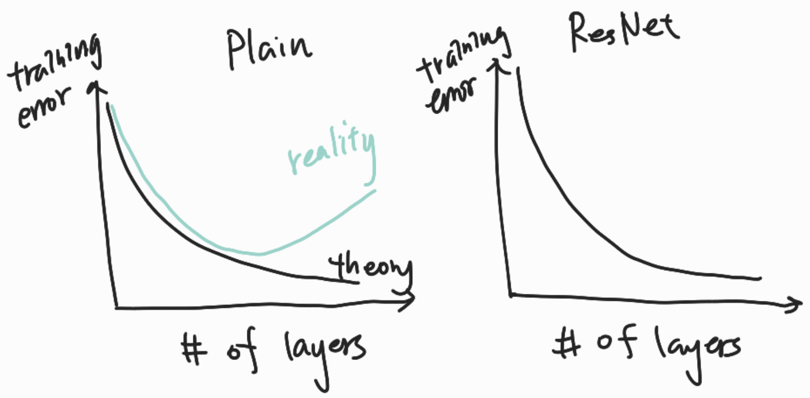
$$a^{(l+2)} = \sigma(\underbrace{z^{(l+2)} + a^{(l)}}_{\text{same dimension}}) = \sigma(\cancel{W^{(l+2)}} a^{(l+1)} + \cancel{b^{(l+2)}} + a^{(l)}) = \sigma(a^{(l)}) = a^{(l)}$$

if $W^{(l+2)} = b^{(l+2)} = 0$

It is easy for ResNet to learn identity functions!

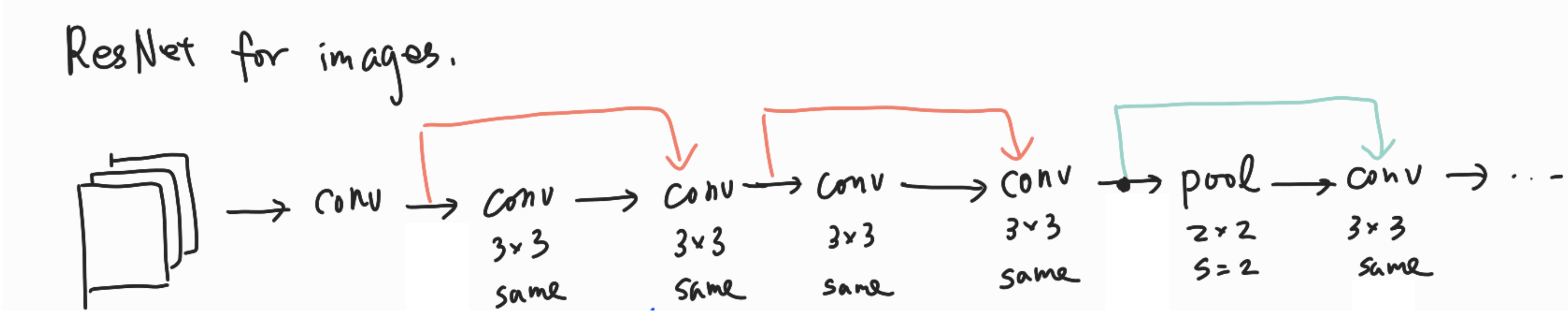


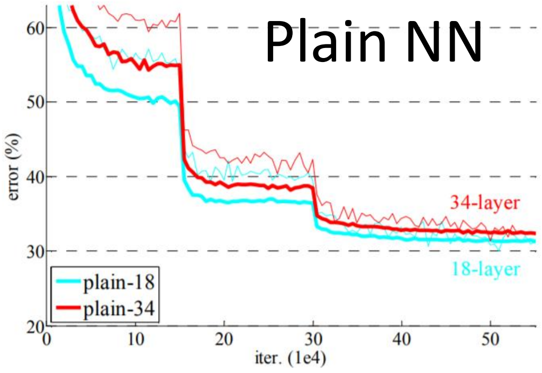
2 residual blocks

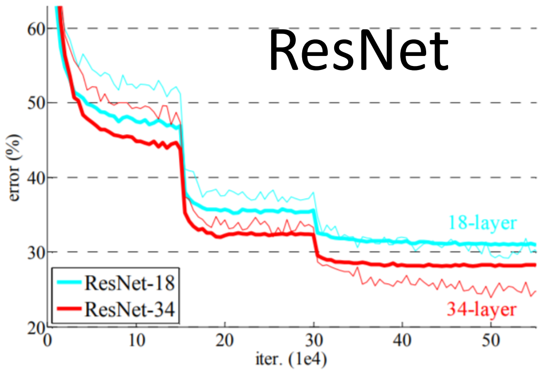


ResNet

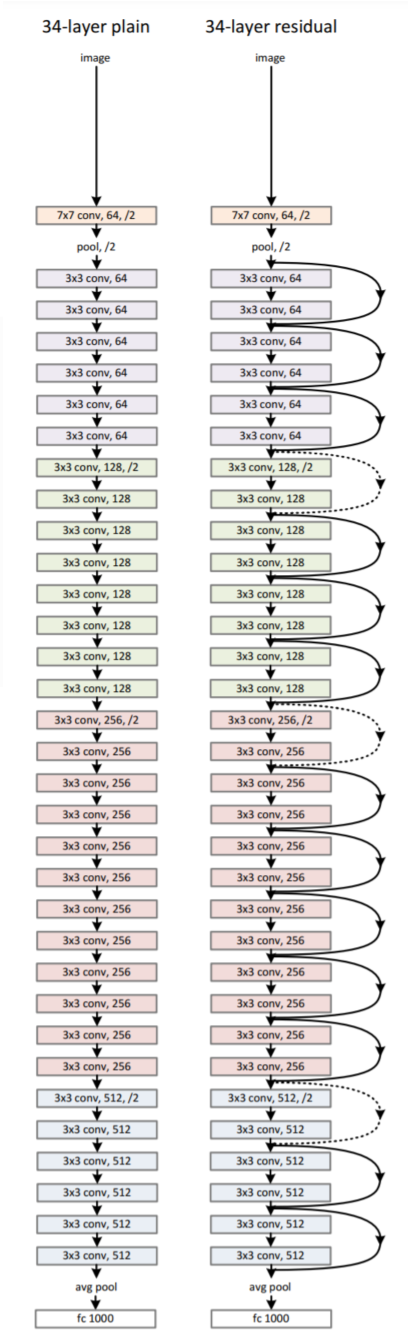
Deep residual learning for image recognition
K He, X Zhang, S Ren, J Sun - … and pattern **recognition**, 2016 - openaccess.thecvf.com
Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer …







Deep ResNet learns better than deep plain NN!



For plain NN, adding layers often make NN hard to learn the right parameters even for the identify function, making the result worse -> Use Resnet with “skip connections”

Neural Ordinary Differential Equations

Chen et al (2018), Neural ODE

Neural Ordinary Differential Equations

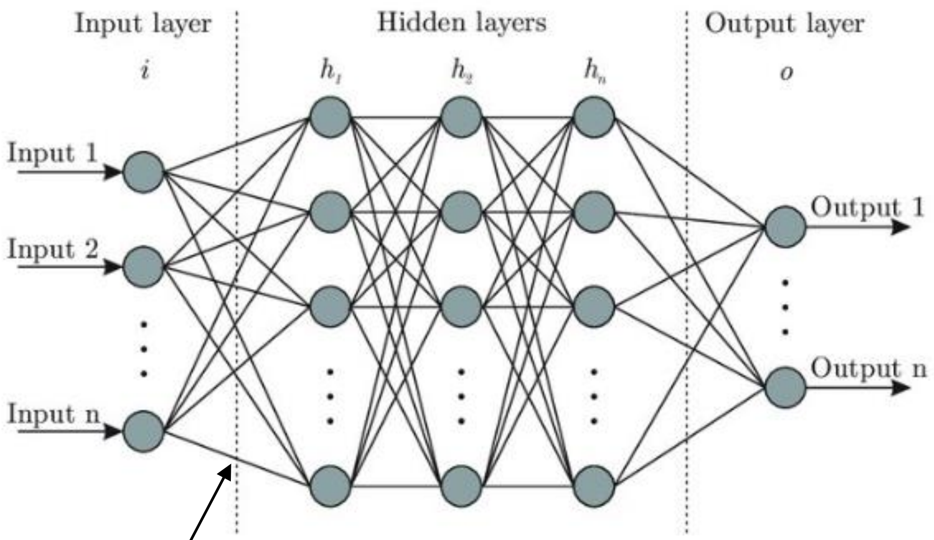
- A new family of deep neural network models

- Use Ordinary Differential Equations solvers (100+ years of development) to optimize NN

- More accurate predictions for time-series data with irregular time spacing (e.g. health-care data, financial data, decease transmission data)

Chen et al (2018), Neural ODE

FC Neural Net

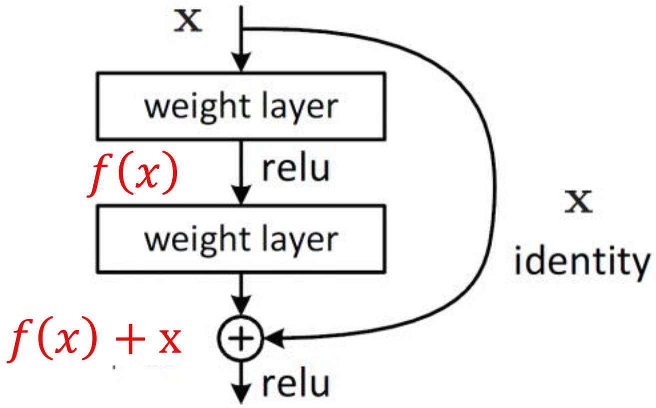


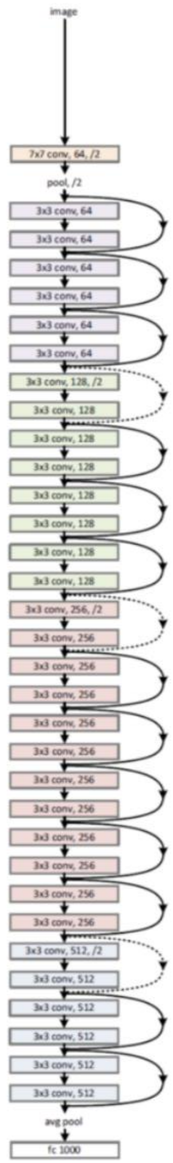
$f(x) = wx + b$

x1 = f1(x)
x2 = f2(x1)
x3 = f3(x2)
x4 = f4(x3)
x5 = f5(x4)
y = f6(x5)

ResNet

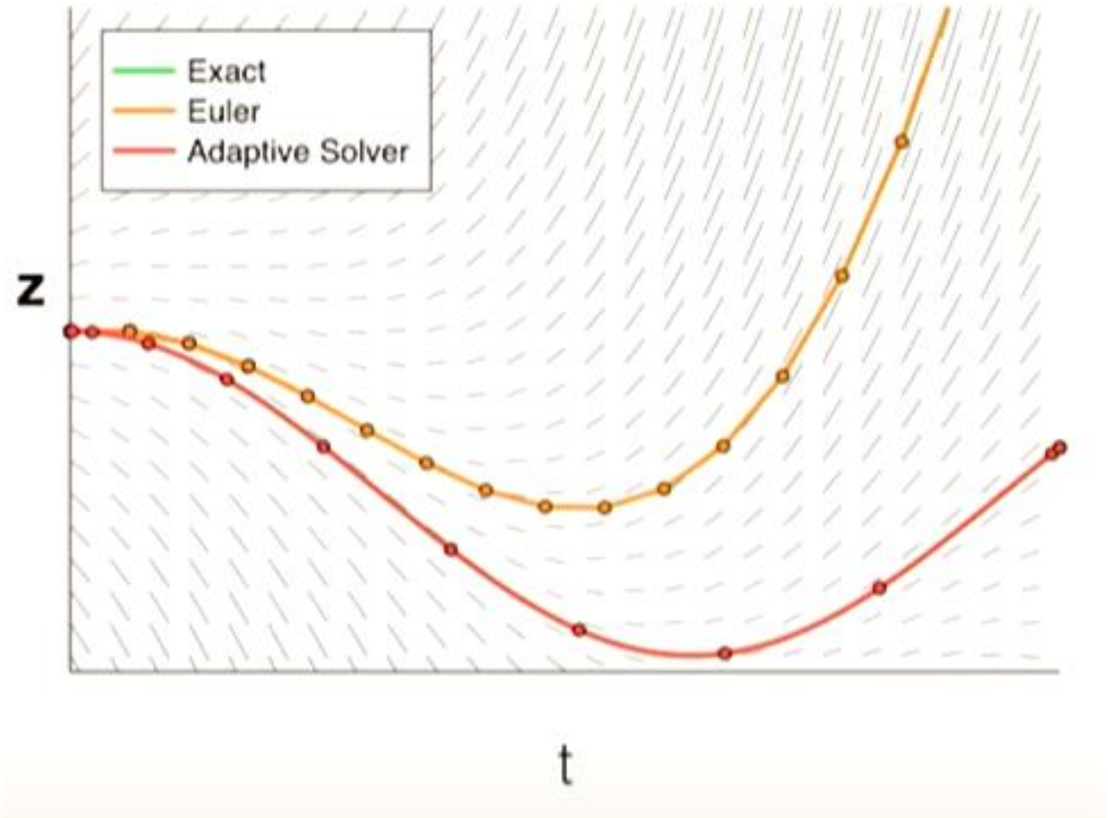
Residual block





x1 = f1(x)   + x
x2 = f2(x1) + x1
x3 = f3(x2) + x2
x4 = f4(x3) + x3
x5 = f5(x4) + x4
y   = f6(x5) + x5

Ordinary differential equations (ODE)



ODE: $\frac{dz}{dt} = \boxed{f(z(t), t)}$

known, e.g. $f = z^2, zt, t^3$

initial condition: $z(t = 0) = z_0$

what is $z(t) = ?$

- Simplest ODE solver (Euler's method) uses local slopes $f = dz/dt$ to project solution trajectories

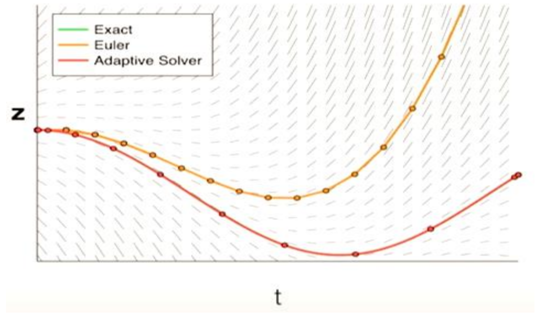
$$\frac{z(t + \Delta t) - z(t)}{\Delta t} = f(z, t)$$

$$\longrightarrow z(t + \Delta t) = z(t) + \Delta t f(z, t)$$

- Modern ODE solvers (Adaptive method) works very well to find solution trajectories!

Similarity between ResNet and ODE

ODE solver



f(z) determines how input z evolve with time

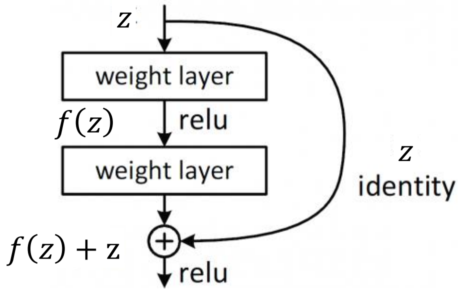
$$z(t+\Delta t) = z(t) + \Delta t f(z,t)$$

$$\frac{z(t+\Delta t) - z(t)}{\Delta t} = f(z,t)$$

When Δt approaches zero

$$\frac{dz}{dt} = f(z(t),t)$$

ResNet



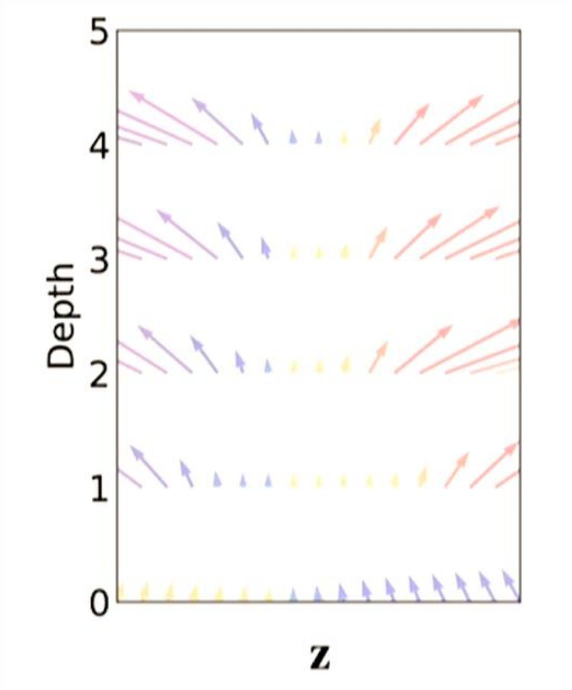
f(z) determines how input z evolve with layers

$$z_{k+1} = z_k + f(z_k)$$

- Time in ODE (continuous) = Depth of layer in ResNet (discrete)
- New NN layer in a ResNet depends on previous layer in a same fashion as ODE solution between two time-steps
- Each residual block can be replaced by ODEnet
  -> optimize ResNet with an ODE solver

Similarity between ResNet and ODE

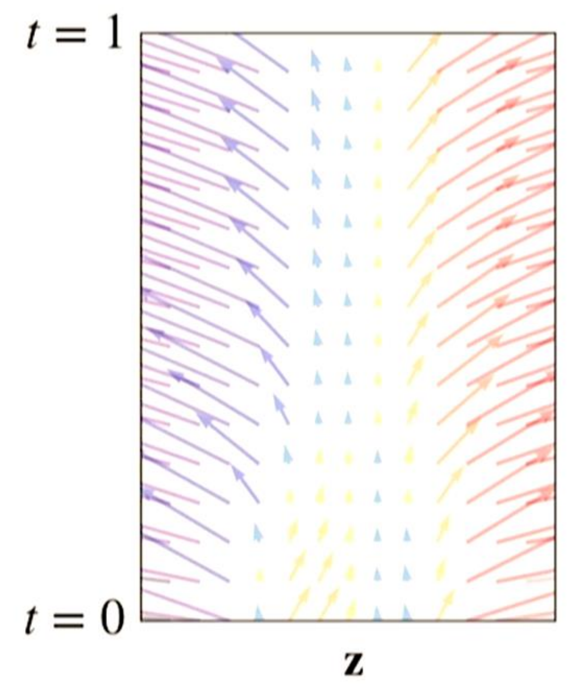
A **Residual network** defines a discrete sequence of finite transformations.




```
def f(z, t, θ):
  return nnet(z, θ[t])

def resnet(z):
  for t in [1:T]:
    z = z + f(z, t, θ)
  return z
```

$$z_{t_1} = \mathrm{f}(z_{t_0}) + z_{t_0}$$
$$z_{t_2} = \mathrm{f}(z_{t_1}) + z_{t_1}$$
$$\dots \dots$$
$$z_{t_N} = \mathrm{f}(z_{t_{N-1}}) + z_{t_{N-1}}$$

An **ODE network** defines a vector field, which continuously transforms the state.



ODENet

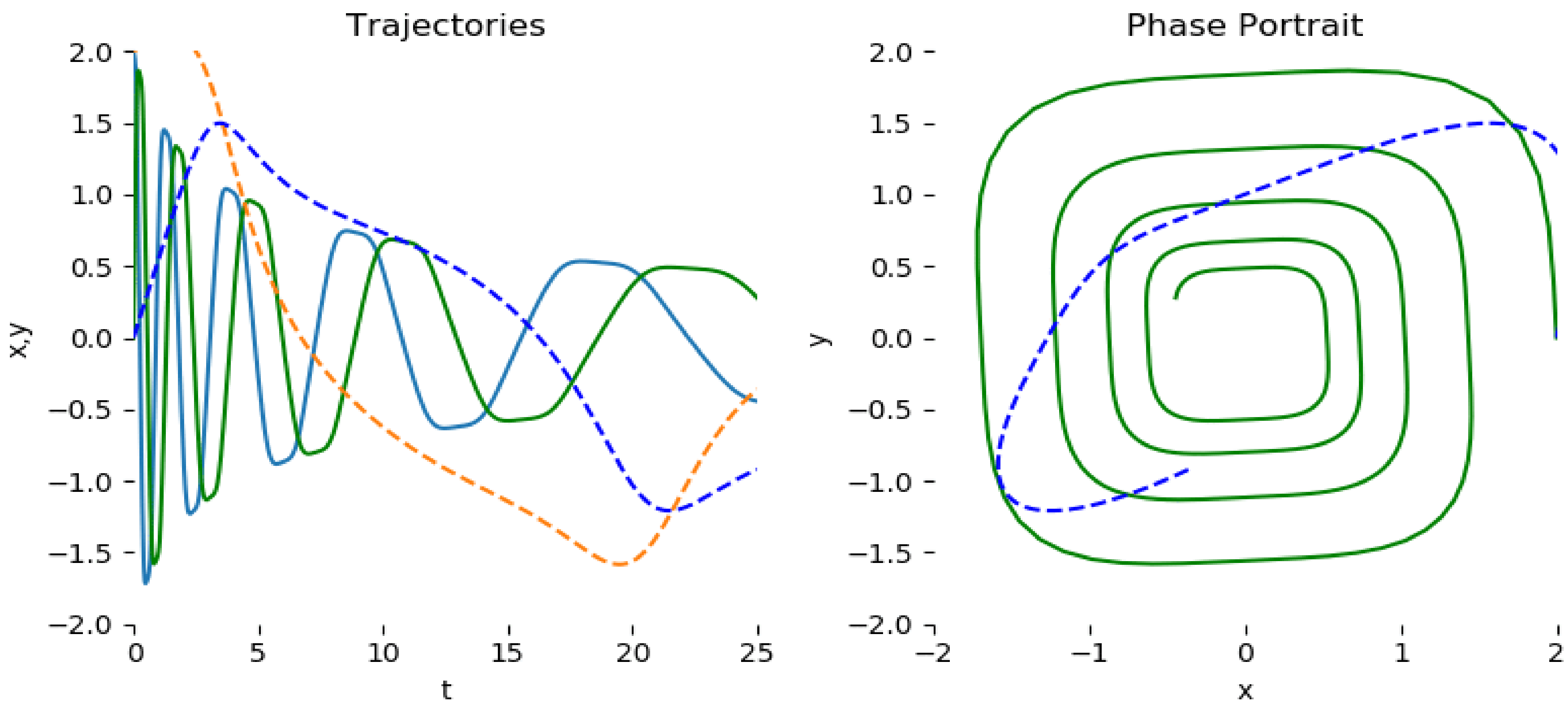
```
def f(z, t, θ):
  return nnet([z, t], θ)

def ODEnet(z, θ):
  return ODESolve(f, z, 0, 1, θ)
```

ODE solver

$$\frac{dz}{dt} = f(z(t), t)$$

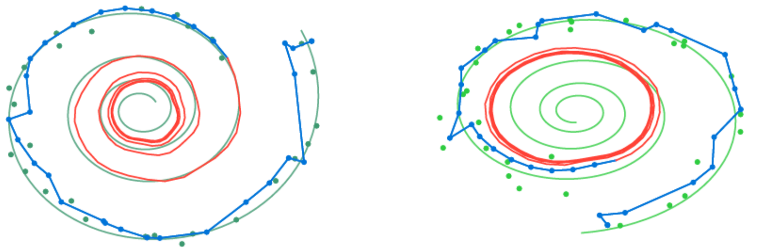
"The output of the network is computed using a blackbox differential equation solver."

Training (fitting) neural net f



Training (fitting) neural net f and extrapolation

Recurrent Neural Net



Neural ODE

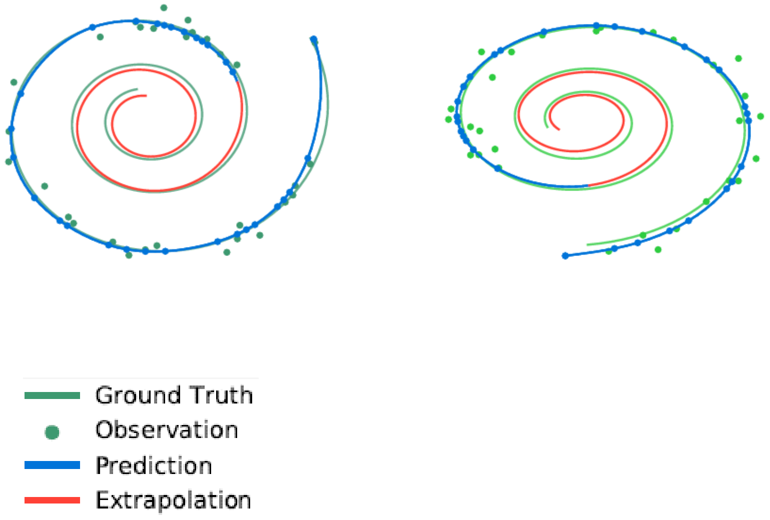
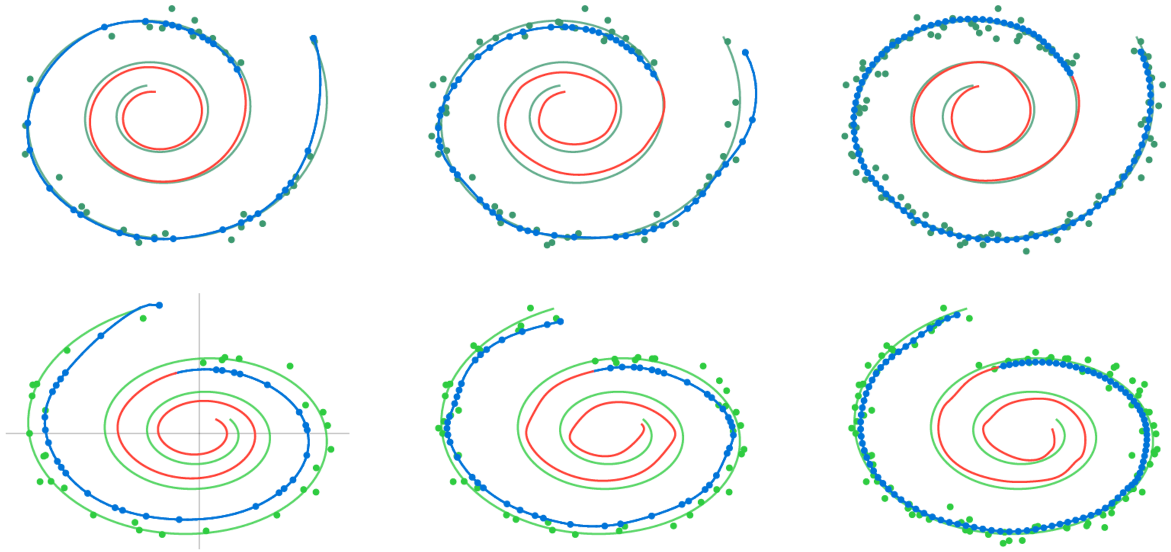


Table 2: Predictive RMSE on test set

# Observations	30/100	50/100	100/100
RNN	0.3937	0.3202	0.1813
Latent ODE	**0.1642**	**0.1502**	**0.1346**



(a) 30 time points (b) 50 time points (c) 100 time points

What's next?

- **PINN-** inverse problems, physics-informed interpolation
- **SINDY, PDE-FIND-** model discovery
- **CNN-** pattern recognition, super-resolution
- **ResNet**

- **Guest lecturer: Dr. Maike Sonnewald**

No Free Lunch: How ML can be used (or mis-used) to uncover dynamical regimes in the ocean and beyond.
Please bring your laptop

- Project presentations in two weeks! (11/30, 12/2)