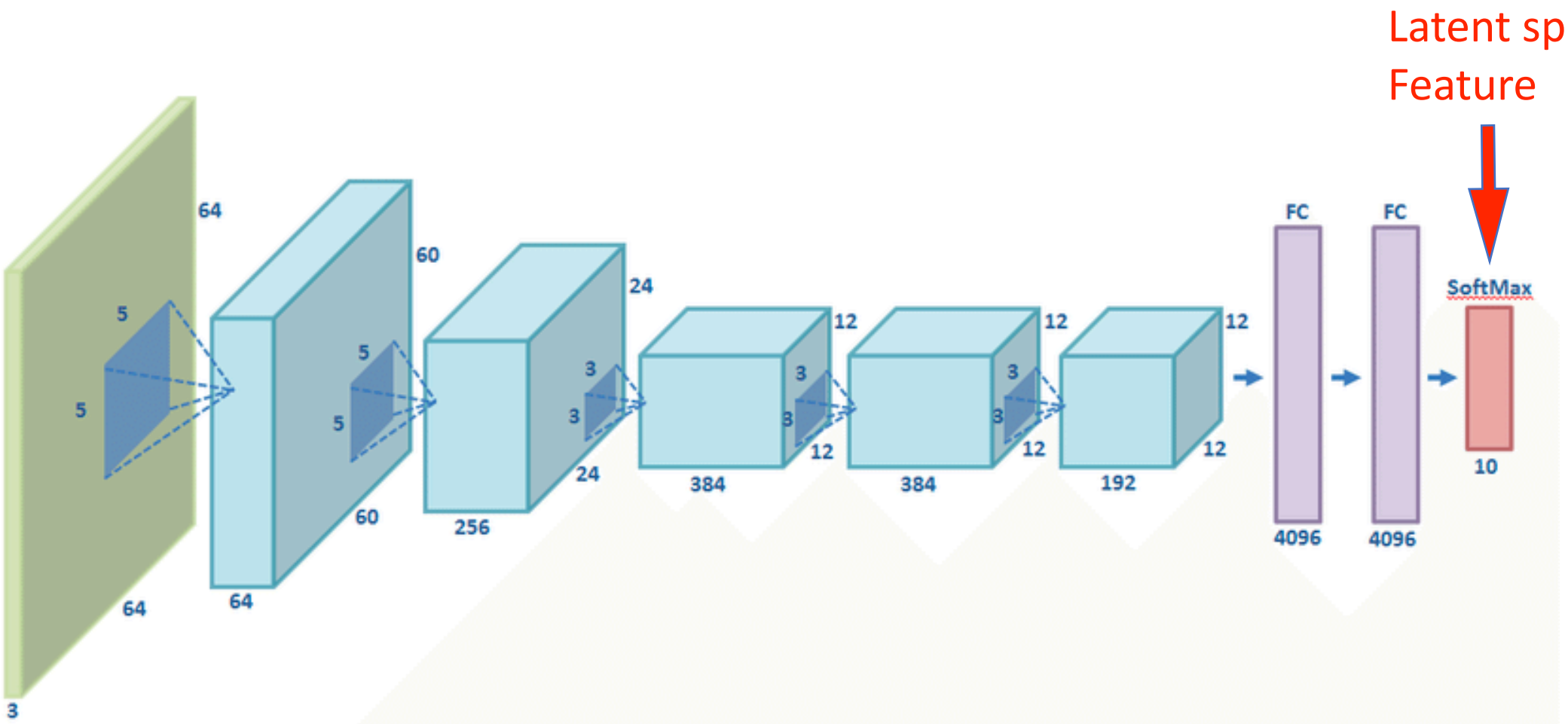
# Deep Forward Networks - Optimization, Regularization

## Thursday
## 8.00 – 8.45

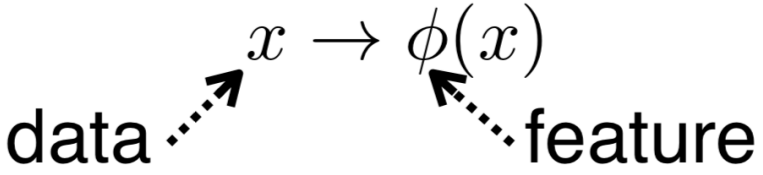
Qiyang Hu, Bern Winter School on Machine Learning 2021, Muerren
qiyang.hu@inf.unibe.ch

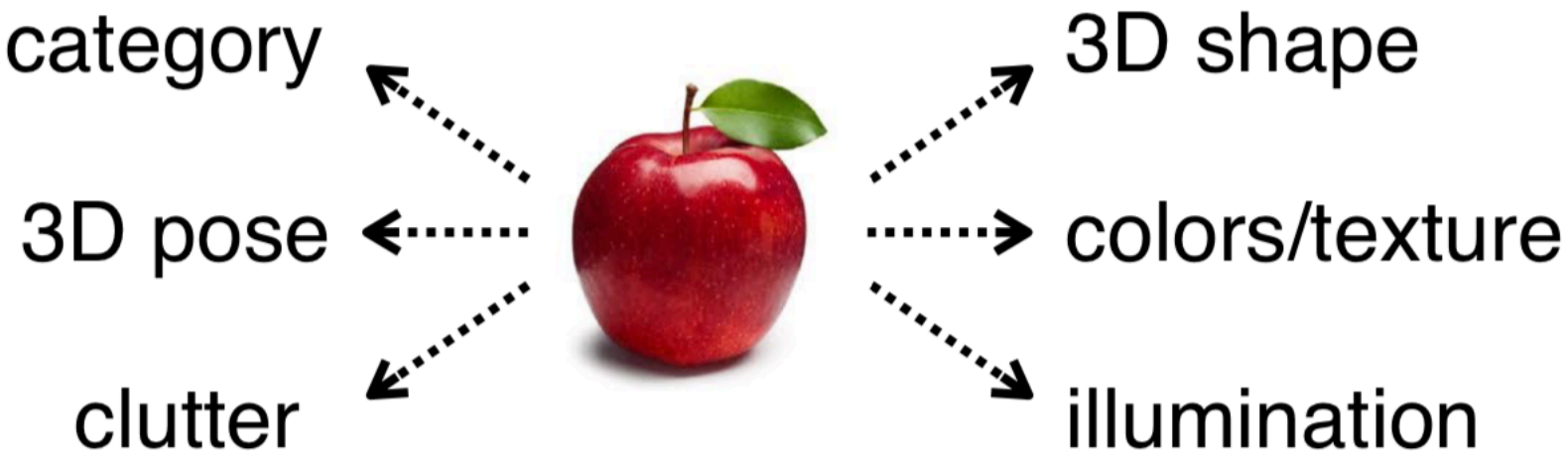
# Convolutional neural network



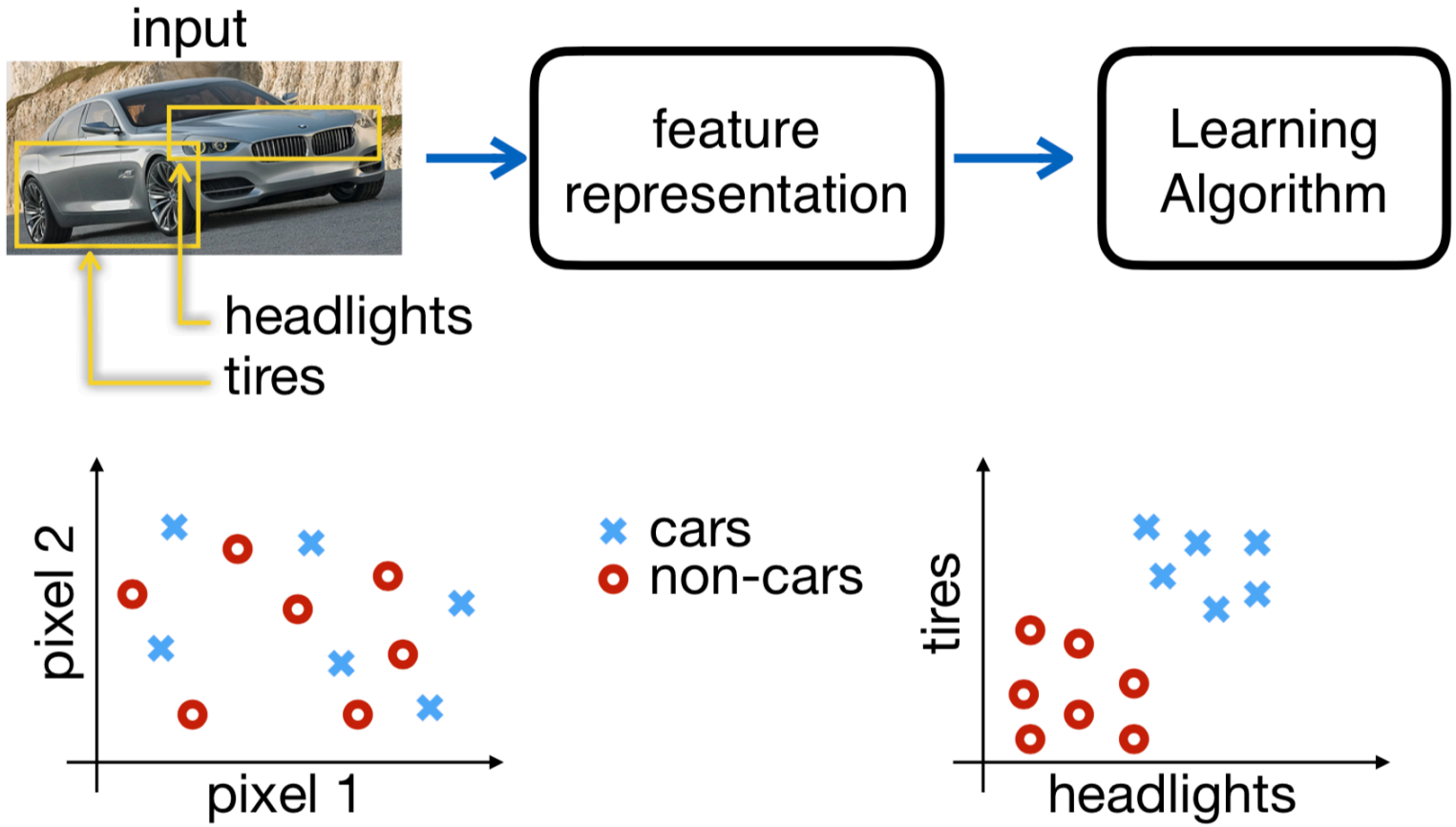
# Features

- Data often encoded into more focused relevant information (features or internal representation) to simplify the decision



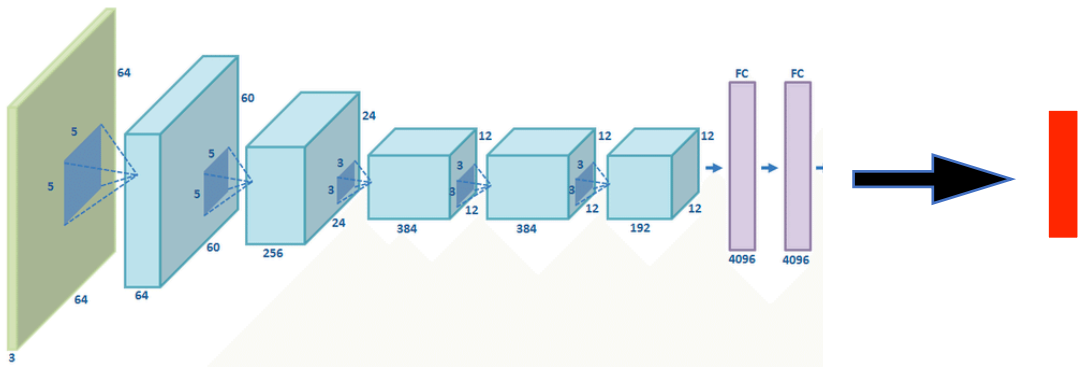
# Features Example :Image classification



# Convolutional neural network

Input

Latent space/
Feature

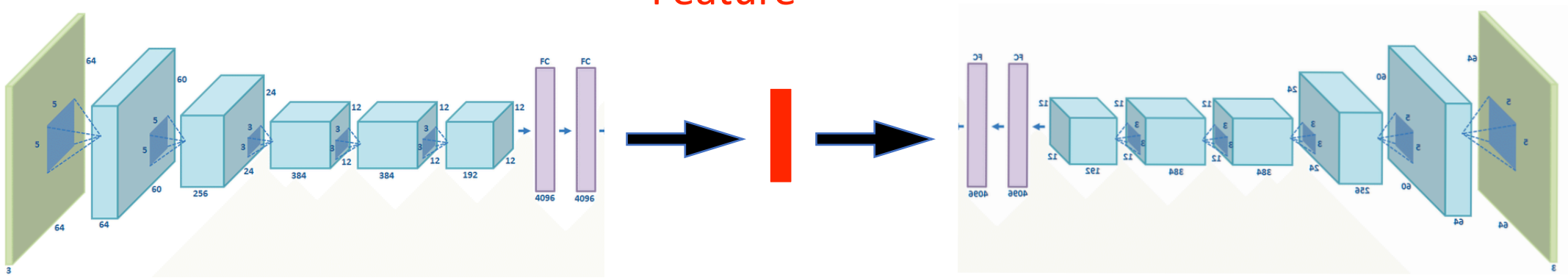


# AutoEncoder

Input

Latent space/
Feature

Reconstruction

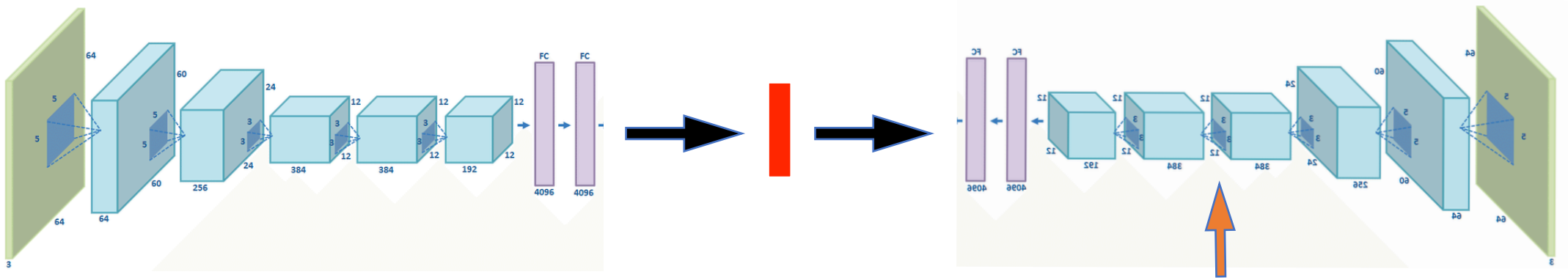


# AutoEncoder

Input

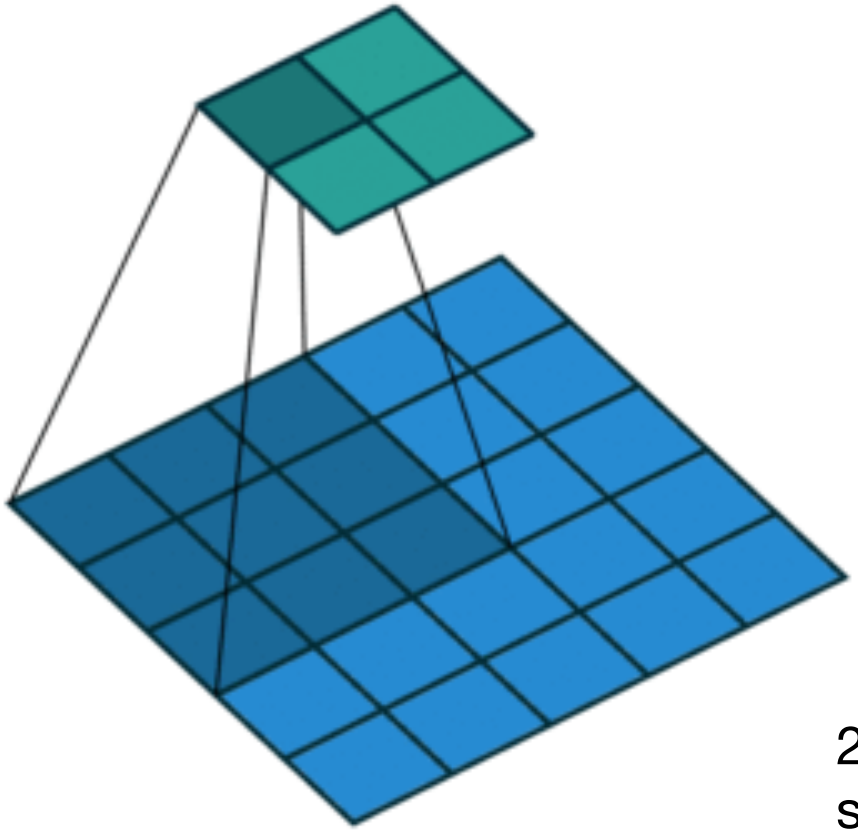
Latent space/
Feature

Reconstruction




Deconvolution
Transposed deconvolution/
Fractionally strided convolution

# AutoEncoder



2D convolution with no padding, stride of 2 and kernel of 3

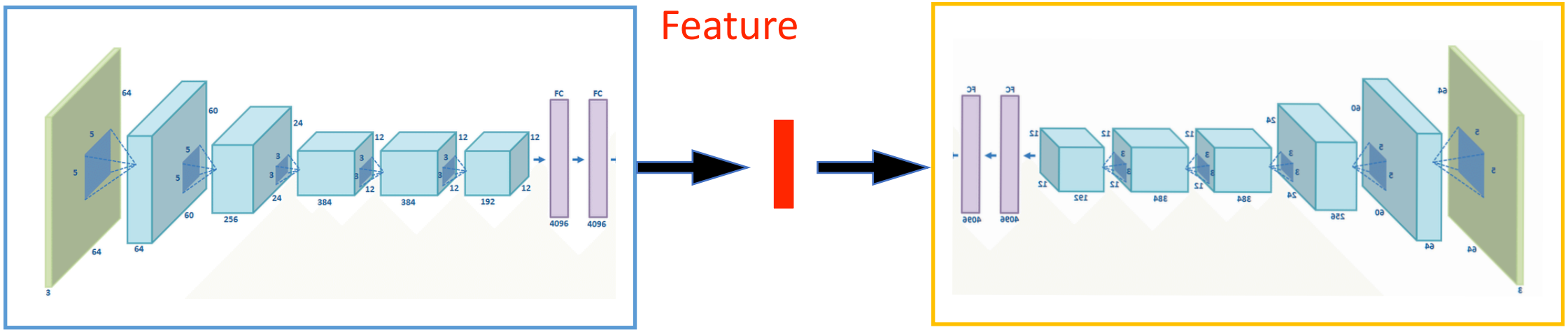
Animation from: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

# AutoEncoder

Input

Latent space/
Feature

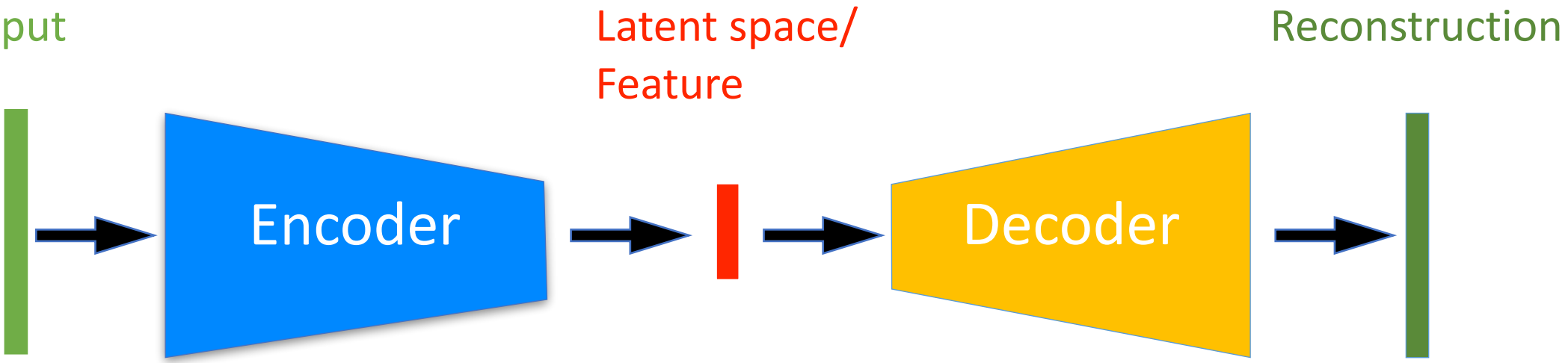
Reconstruction



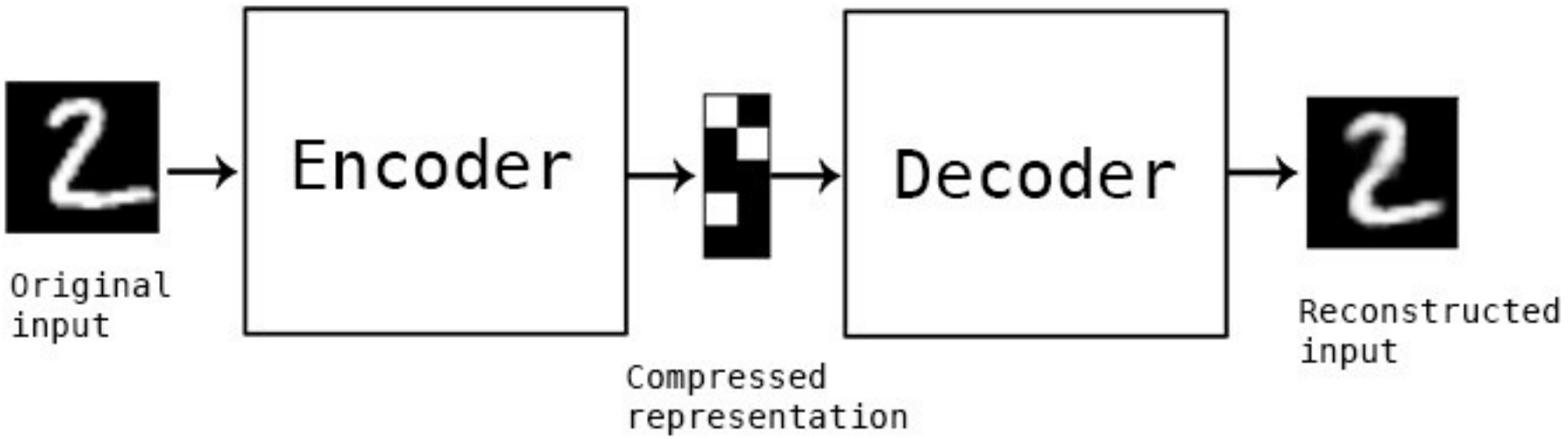

Encoder

Decoder

# AutoEncoder

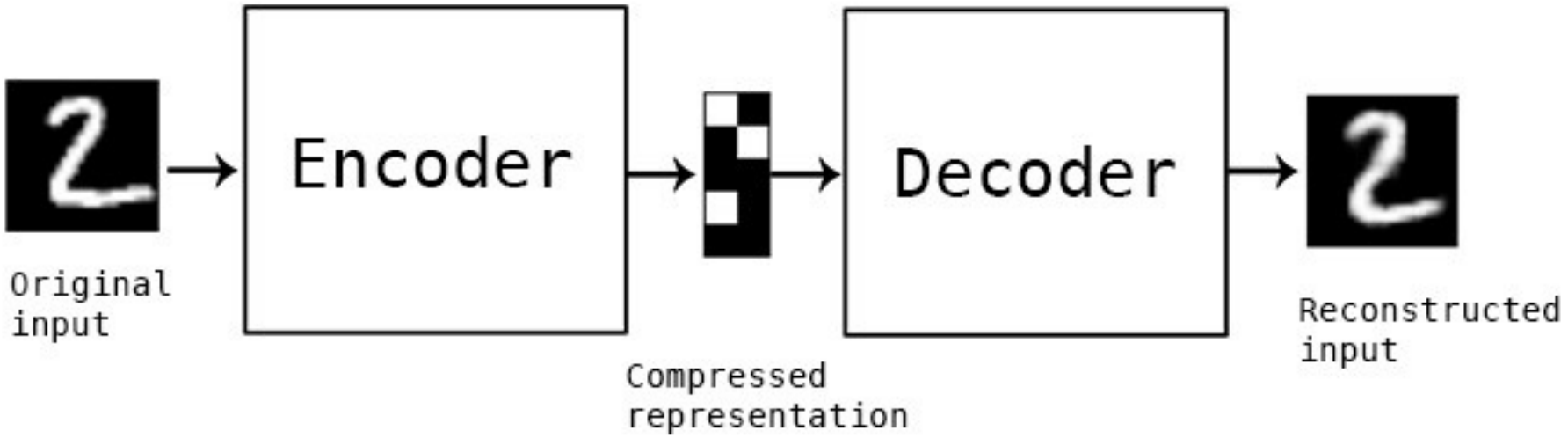


# AutoEncoder



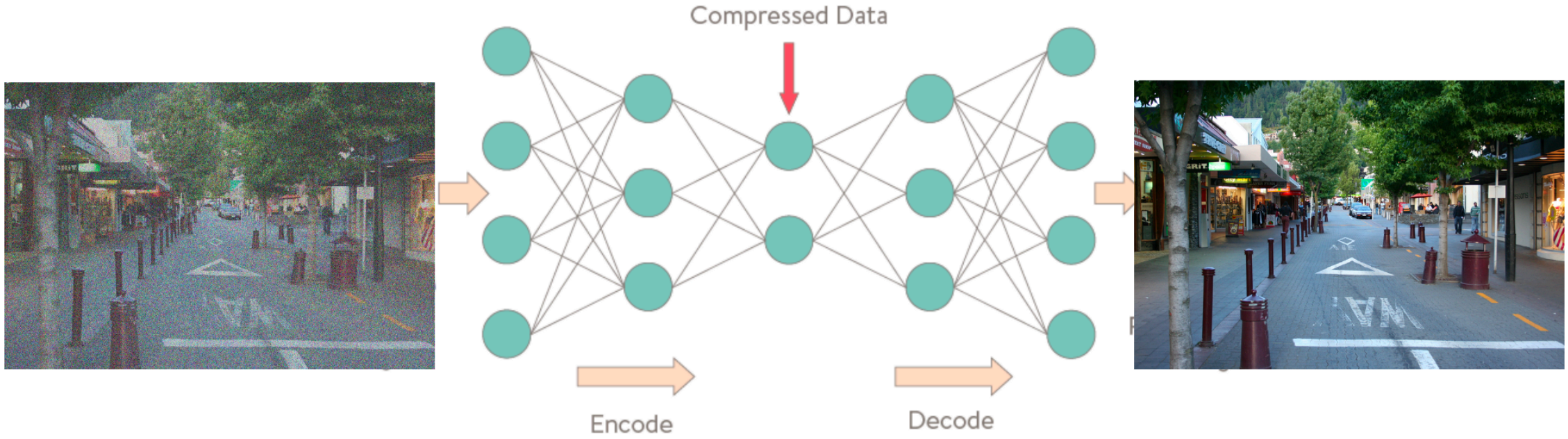
# AutoEncoder

Unsupervised

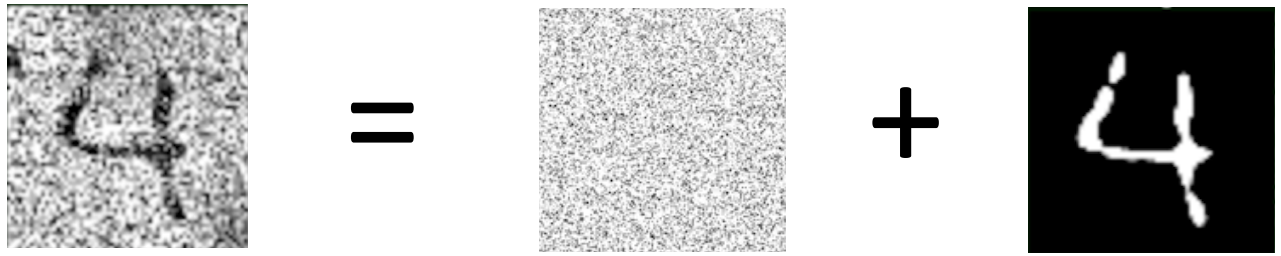


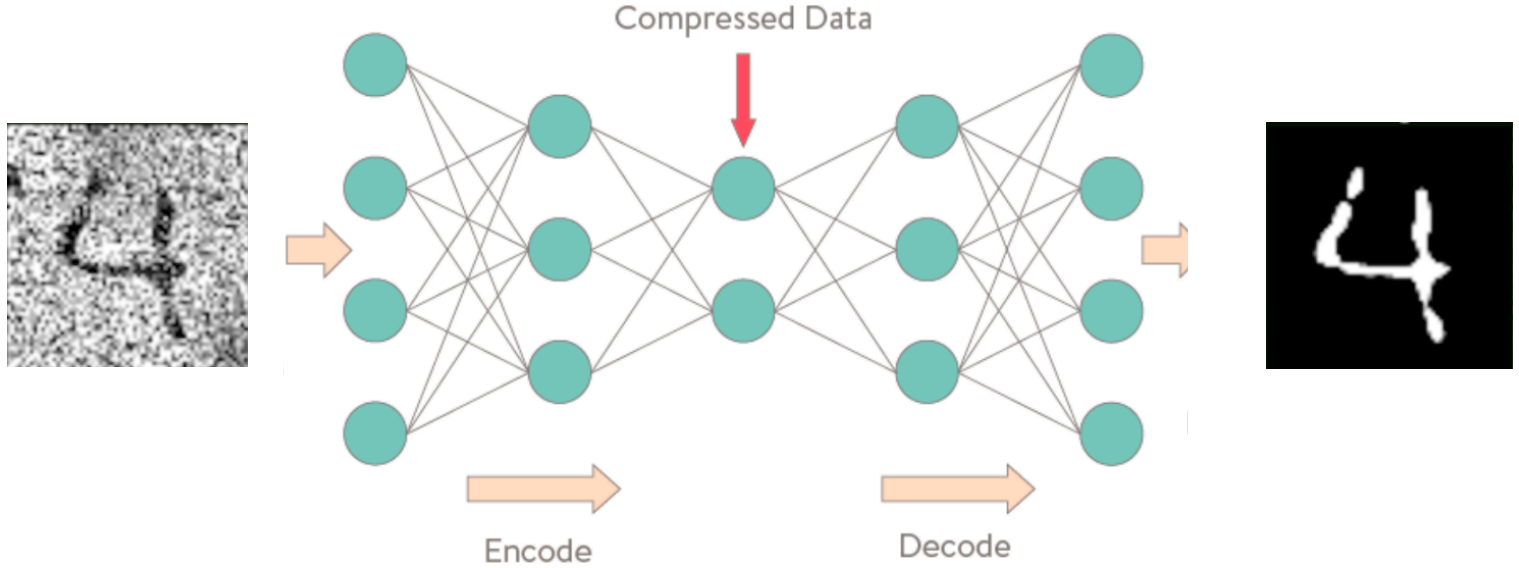
# AutoEncoder: Denoise

One very practical application of autoencoders is to denoise images



# AutoEncoder: Denoise






# Performance

- Bias reduction techniques
- Variance reduction techniques

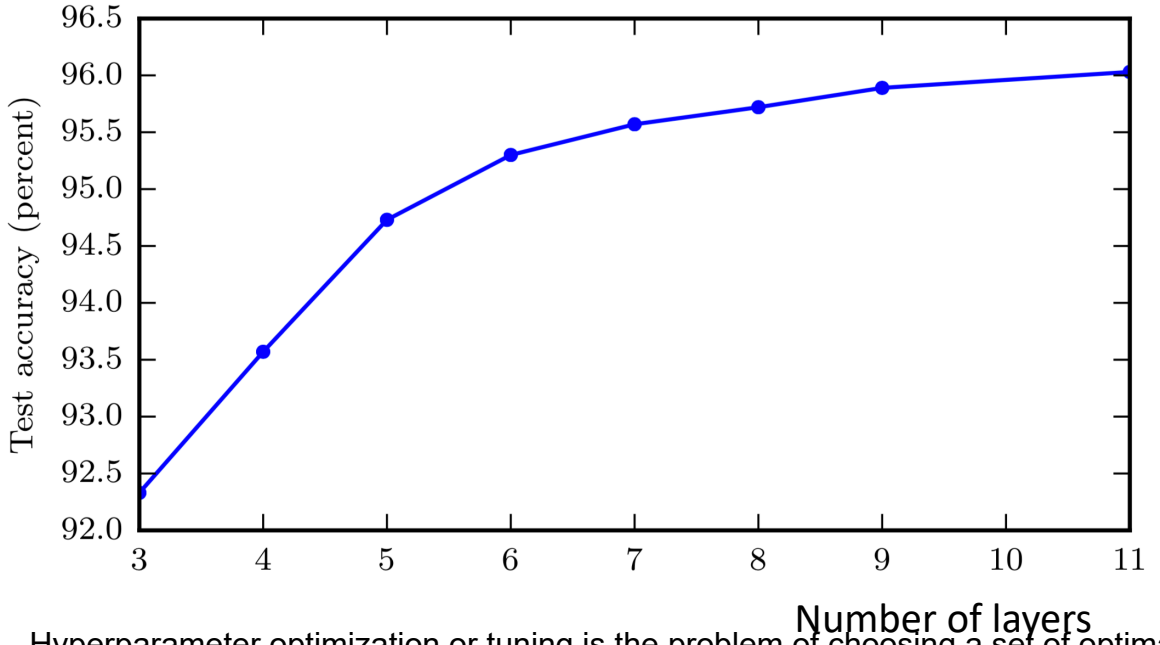
High bias is an erroneous assumption from the learning algorithm that misses relevant relations in the data --> underfitting.

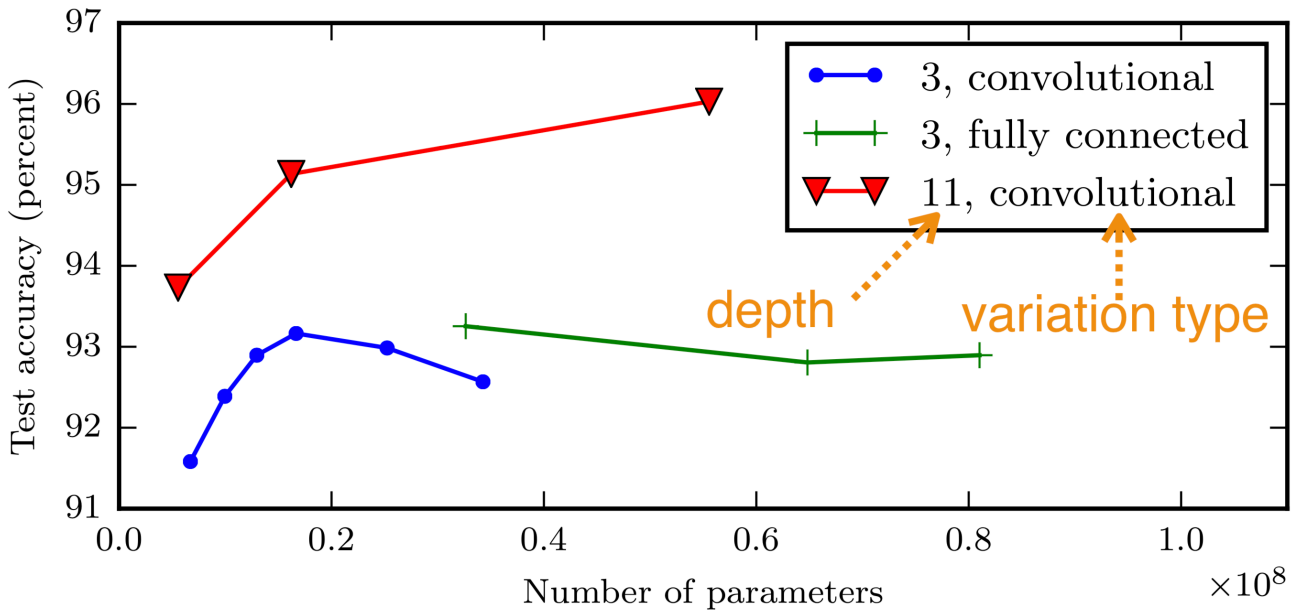

# Bias Reduction techniques

- Hyperparameter tuning
- Model tuning
- Optimization algorithm

# Hyperparameters : number of hidden layers/ units

- To go deeper helps generalization
- *better to have many simple layers than few highly complex ones*

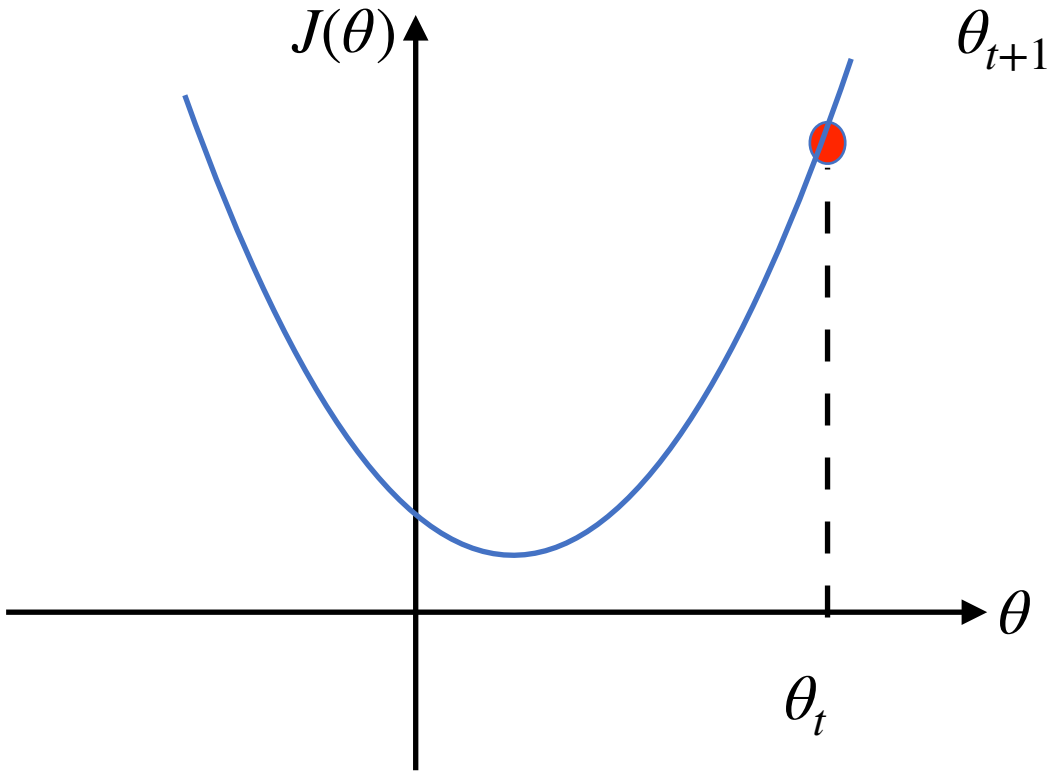



Hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.
The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns.
These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem.
Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data

# Hyperparameters : learning rate $\alpha$

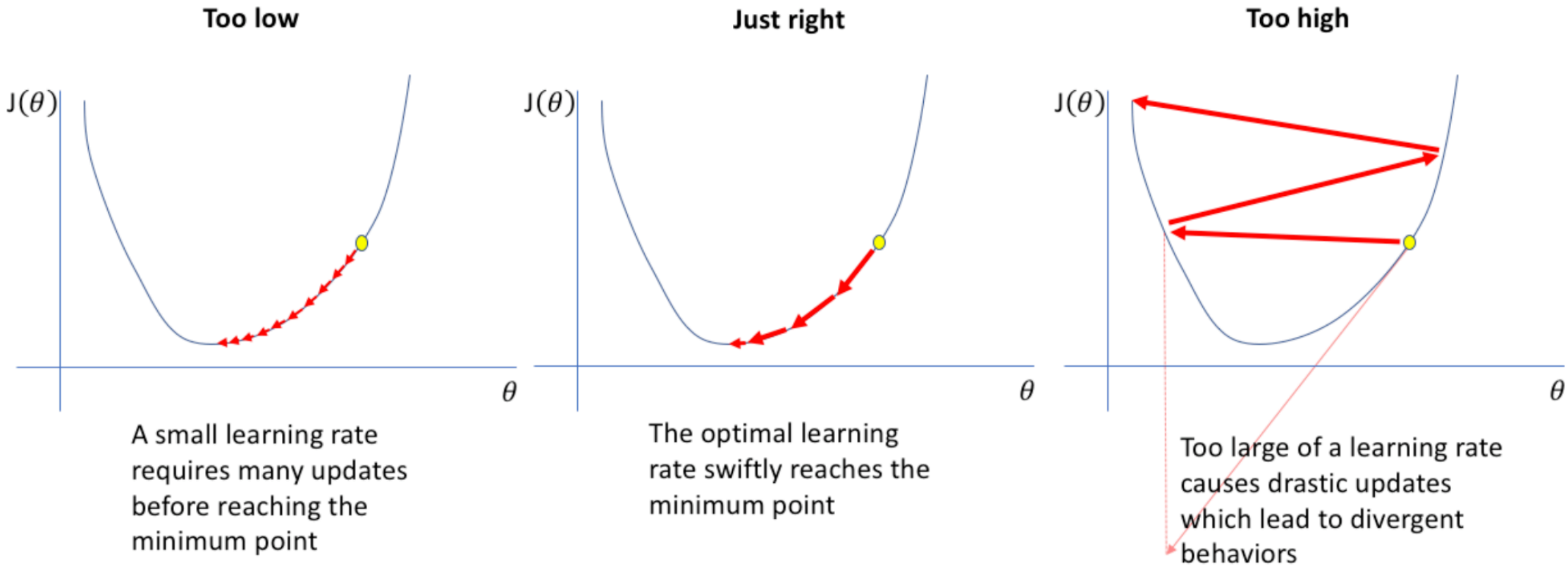
- Has a significant impact on the model performance, while being one of the most difficult parameters to set




$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

# Hyperparameters : learning rate $\alpha$

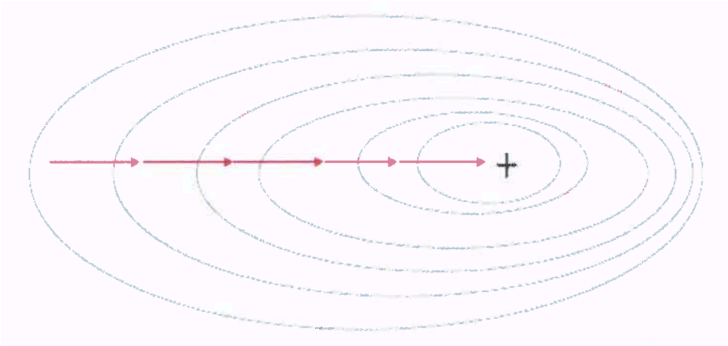
- Has a significant impact on the model performance, while being one of the most difficult parameters to set



# Hyperparameters : batch size

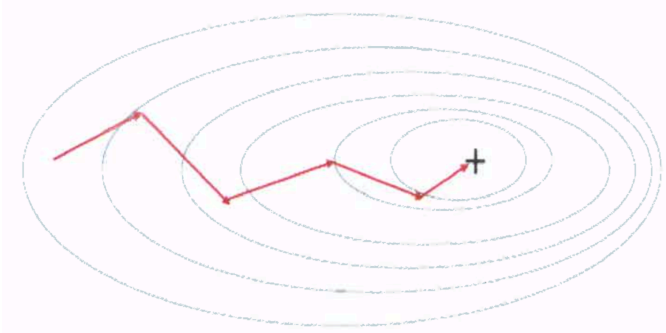
- At each iteration :

Gradient descent (GD)



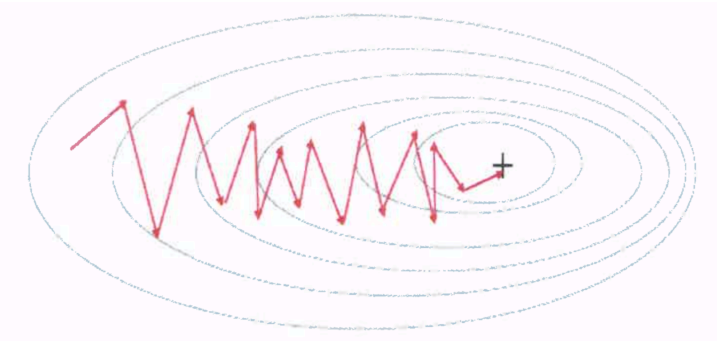
the *whole* training set

Mini-batch gradient descent



a *batch* of samples

Stochastic gradient descent (SGD)



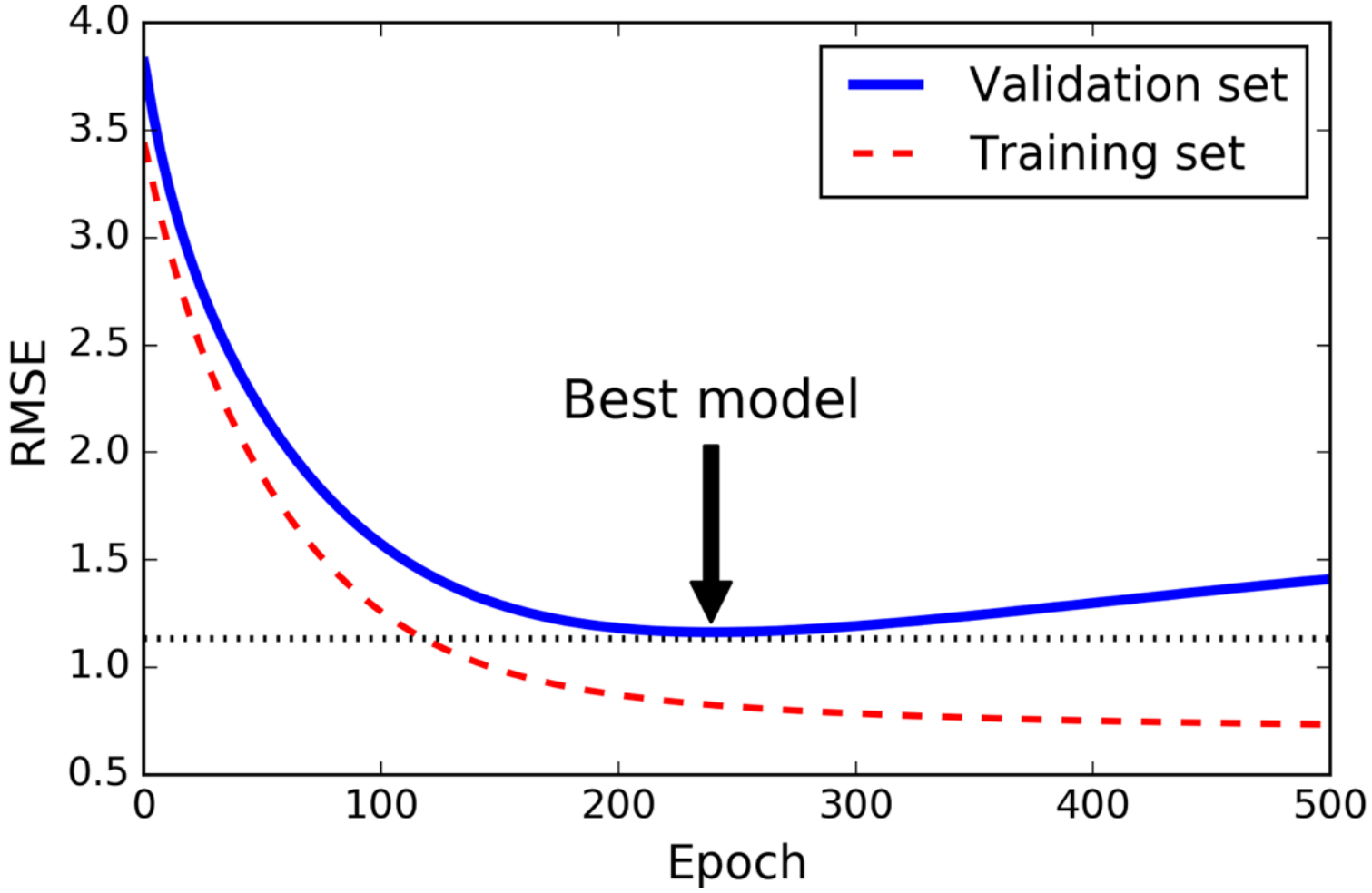
*1 sample*

- Batch size choice *typically 32,64,128,256,512*

# Hyperparameters : epochs

- Train *longer*

root mean square error



# Model : Weight initialization

- The initial parameters need to break the symmetry between different units.

Symmetry: When some machine learning models have weights all initialized to the same value, it can be difficult or impossible for the weights to differ as the model is trained.

The aim of weight initialization is to prevent layer activation outputs from exploding or vanishing during the course of a forward pass through a deep neural network. If either occurs, loss gradients will either be too large or too small to flow backwards beneficially, and the network will take longer to converge, if it is even able to do so at all.

# Model : Weight initialization

Random weights from Gaussian or Uniform distribution doesn't work very well. Better use Xavier weights.

- Use *random weights* from a Gaussian or Uniform distribution. Alternatively, use *Xavier weights*

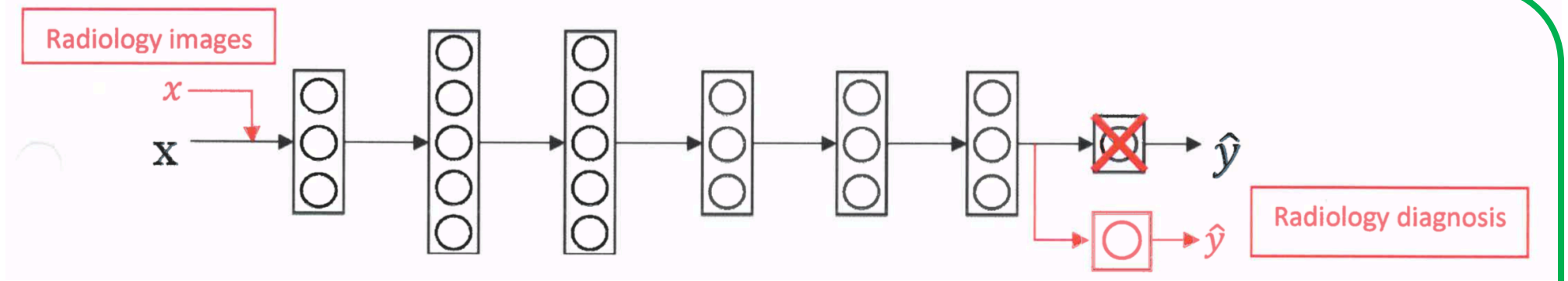
- Another strategy is to initialize weights by transferring weights learnt via an unsupervised learning method (method also called fine-tuning)

# Transfer Learning

- Use weights that have been previously trained for another task

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.[1] For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks.

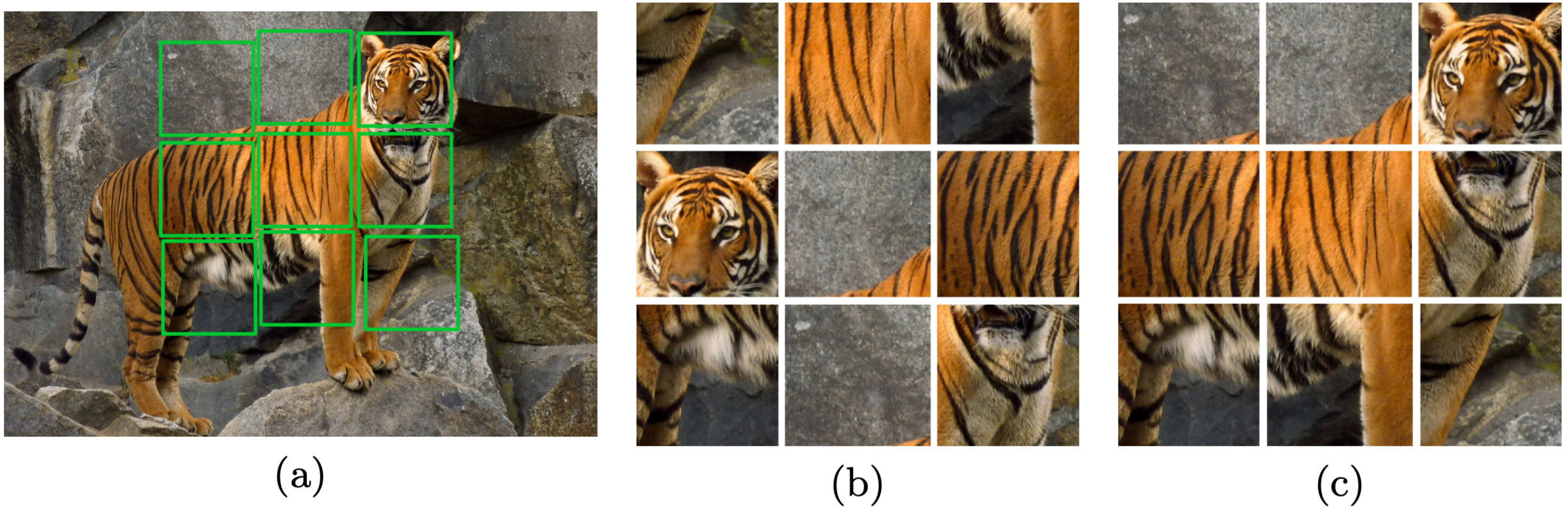
- **Use cases** :
  - Tasks A and B have the same input X
  - A lot more data for Task A than Task B
  - Low level features from Task A could be helpful for Task B




Pre-training : training on cat images

Fine-tuning : update the weights for radiology

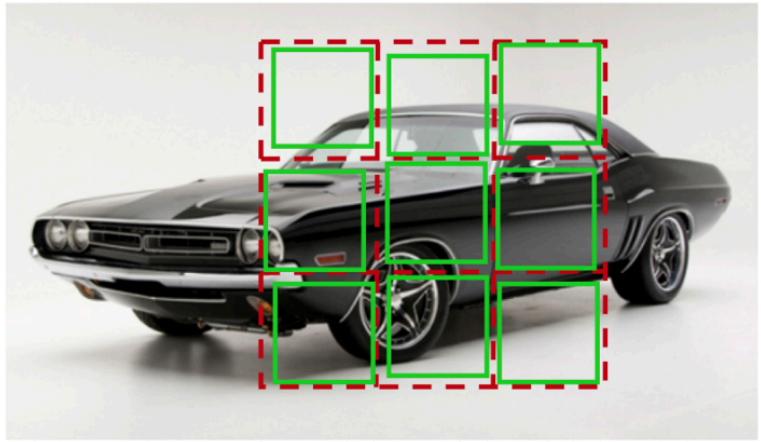
# Transfer Learning



(a) (b) (c)

Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles, M. Norrozi and P. Favaro, ECCV 2017

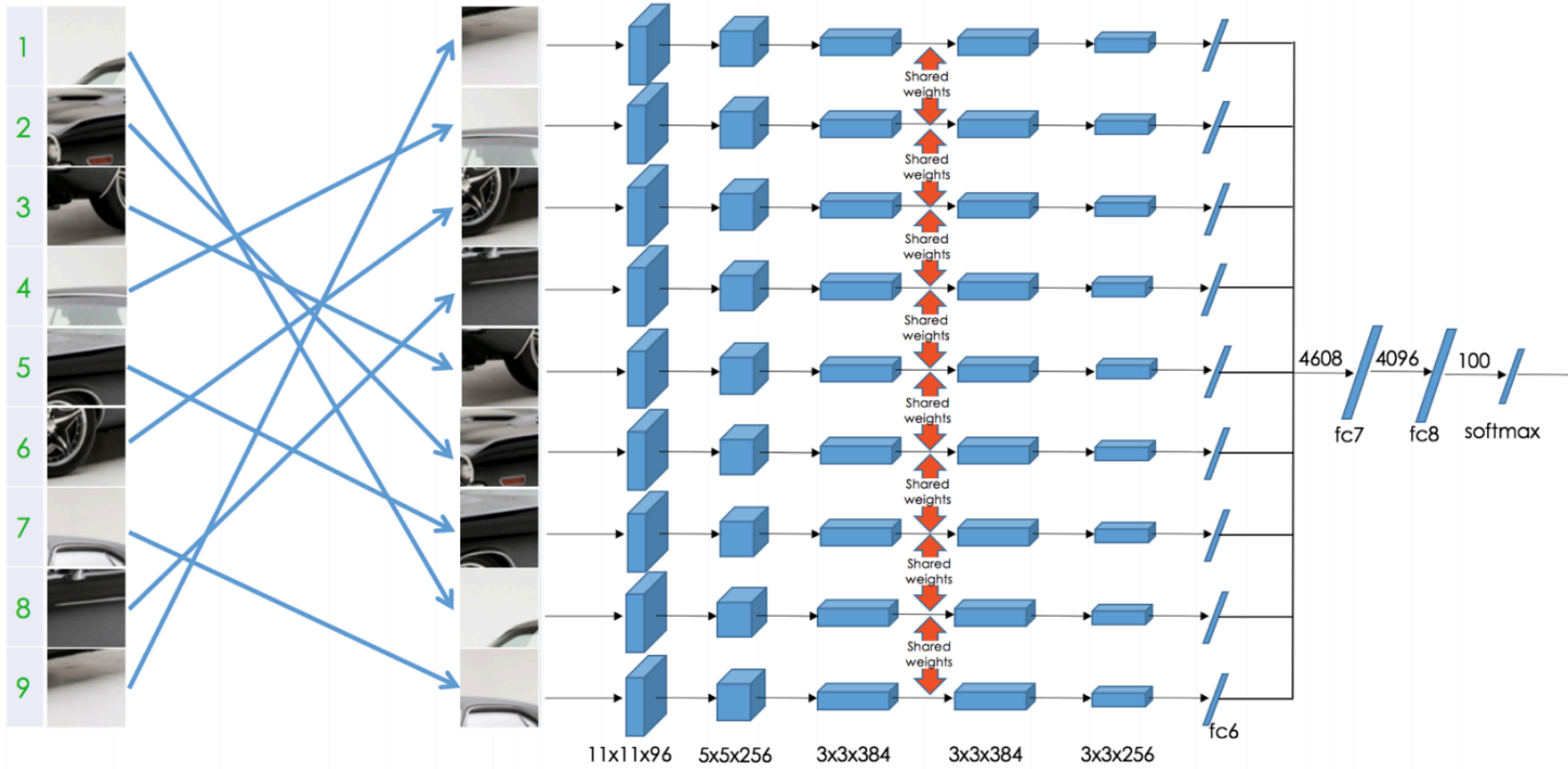
# Transfer Learning



Permutation Set

| index | permutation |
|---|---|
| | |
| 64 | 9,4,6,8,3,2,5,1,7 |
| | |
| | |

Reorder patches according to the selected permutation



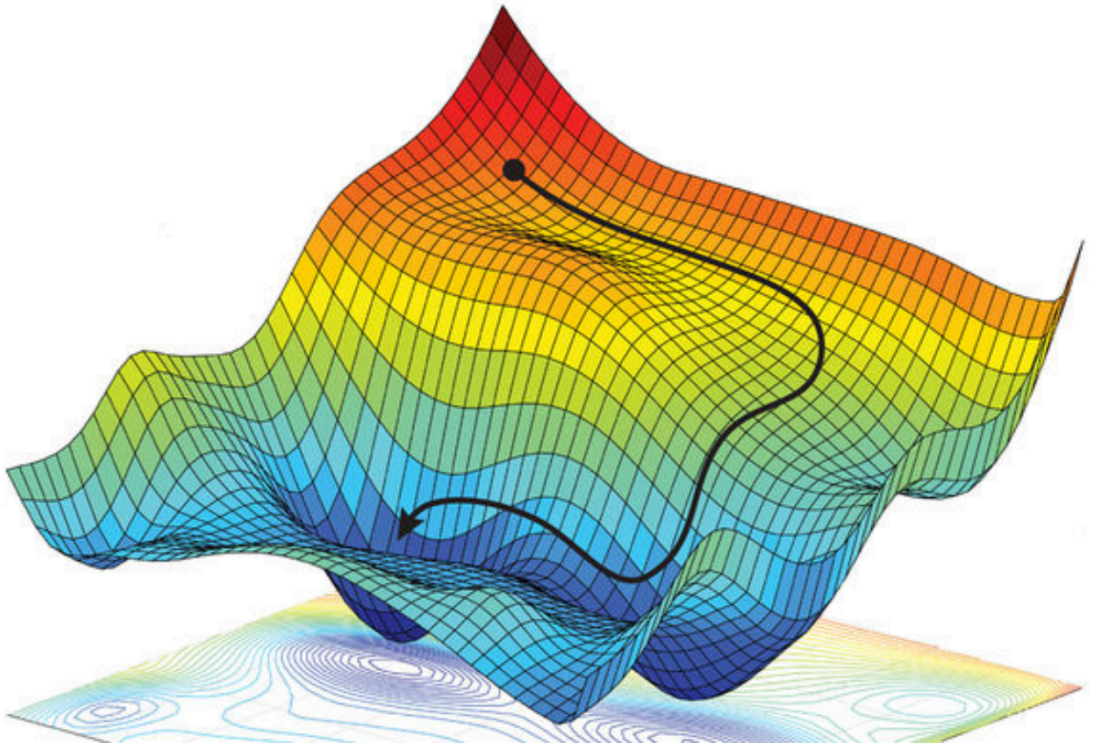

Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles, M. Norrozi and P. Favaro, ECCV 2017

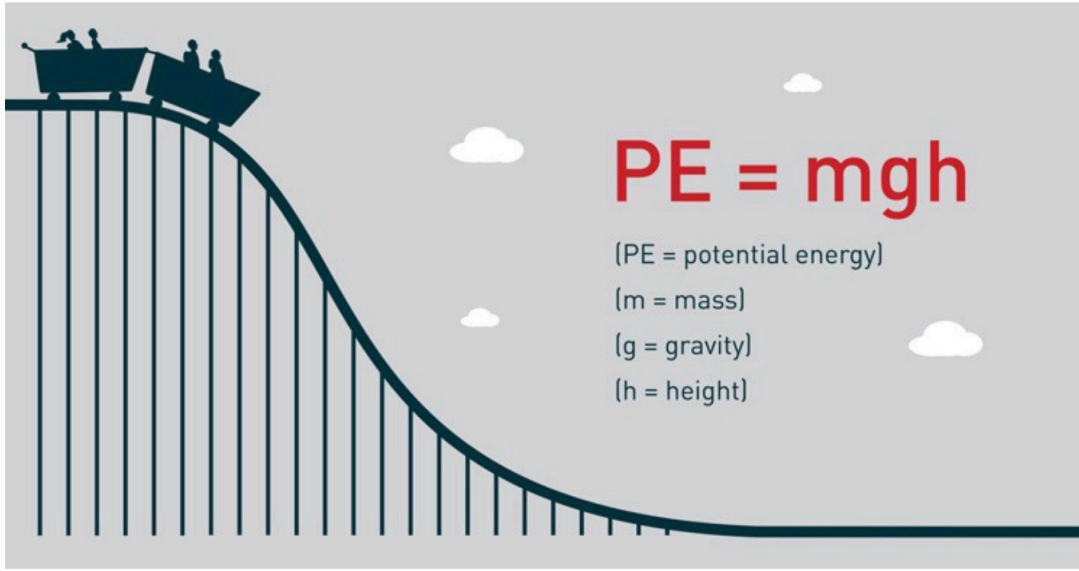
# Optimization algorithm

- No consensus on what algorithm performs best

- Most popular choices :
  - SGD (mini-batch gradient descent)
  - SGD + Momentum
  - RMSProp
  - RMSProp + Momentum
  - Adam

- Strategy : *pick one and get familiar* with the tuning





The variance is an error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).


# Variance Reduction techniques

- Bigger training set
- Regularization

# Regularization

- Different strategies :
  - Dataset (division, augmentation,…)
  - Model (dropout, L2-, …)
  - Training (early stopping)
- Use cases : if few data or if model has more than 50 layers (CNN)

# Regularization (*Dataset*) : Division

- Divide the data into a training, validation and test sets
  - Training set to define the optimal predictor
  - Validation set to choose the capacity
  - Test set to evaluate the performance

Validation set: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

Test set: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

# Regularization (*Dataset*) : Augmentation

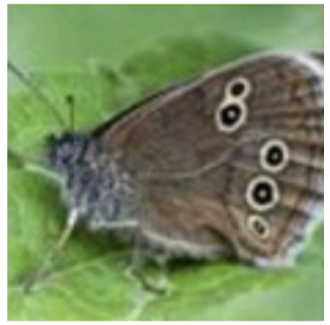
- Apply realistic transformations to data to create new synthetic samples, with same label

original

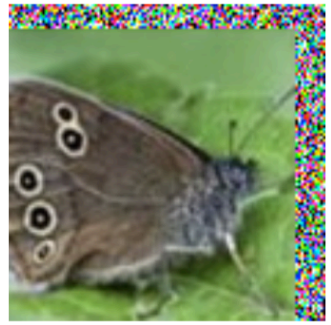
affine distortion

noise

elastic deformation



horizontal flip

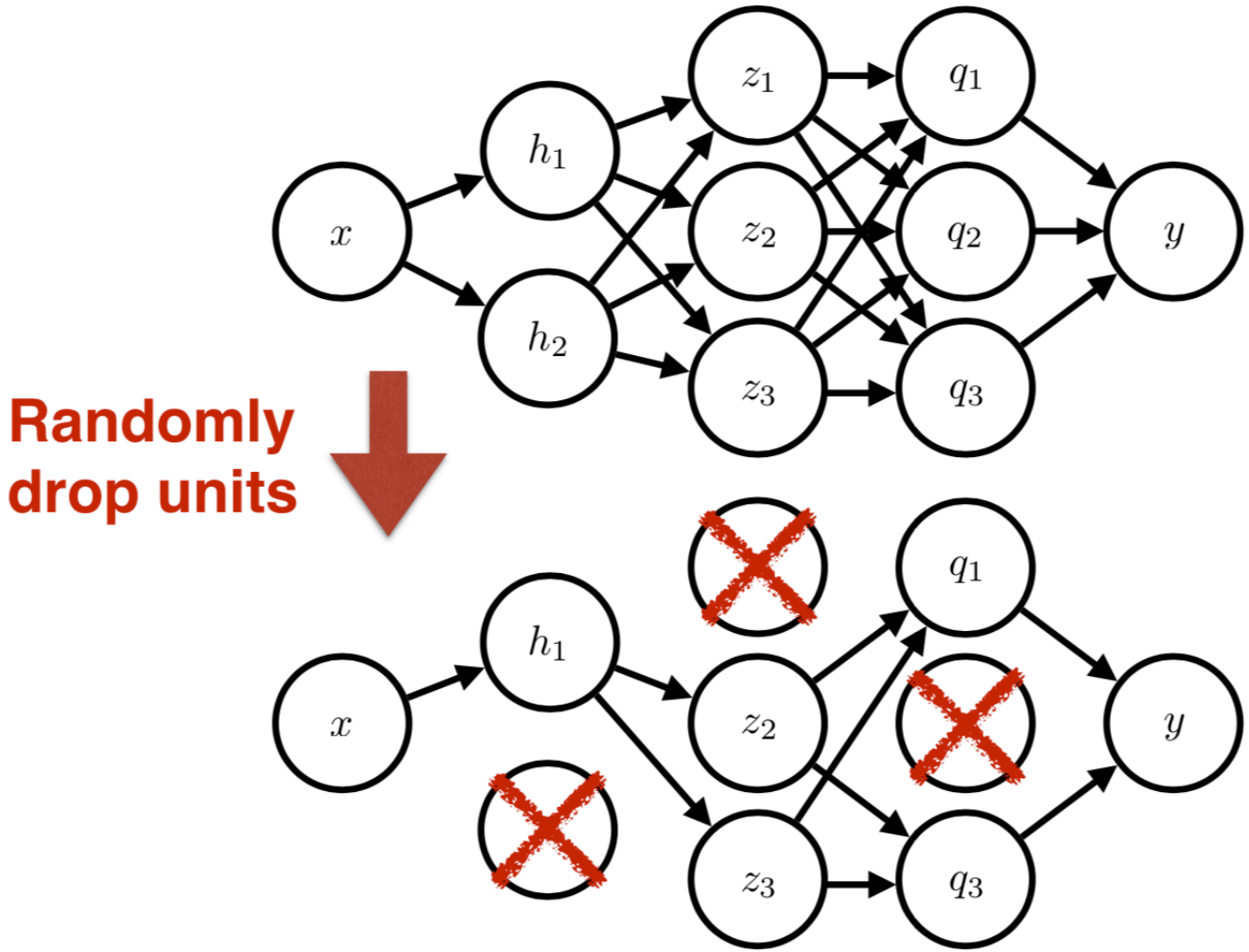


random translation

hue shift

- Process also called jittering

# Regularization (*Model*) : Dropout



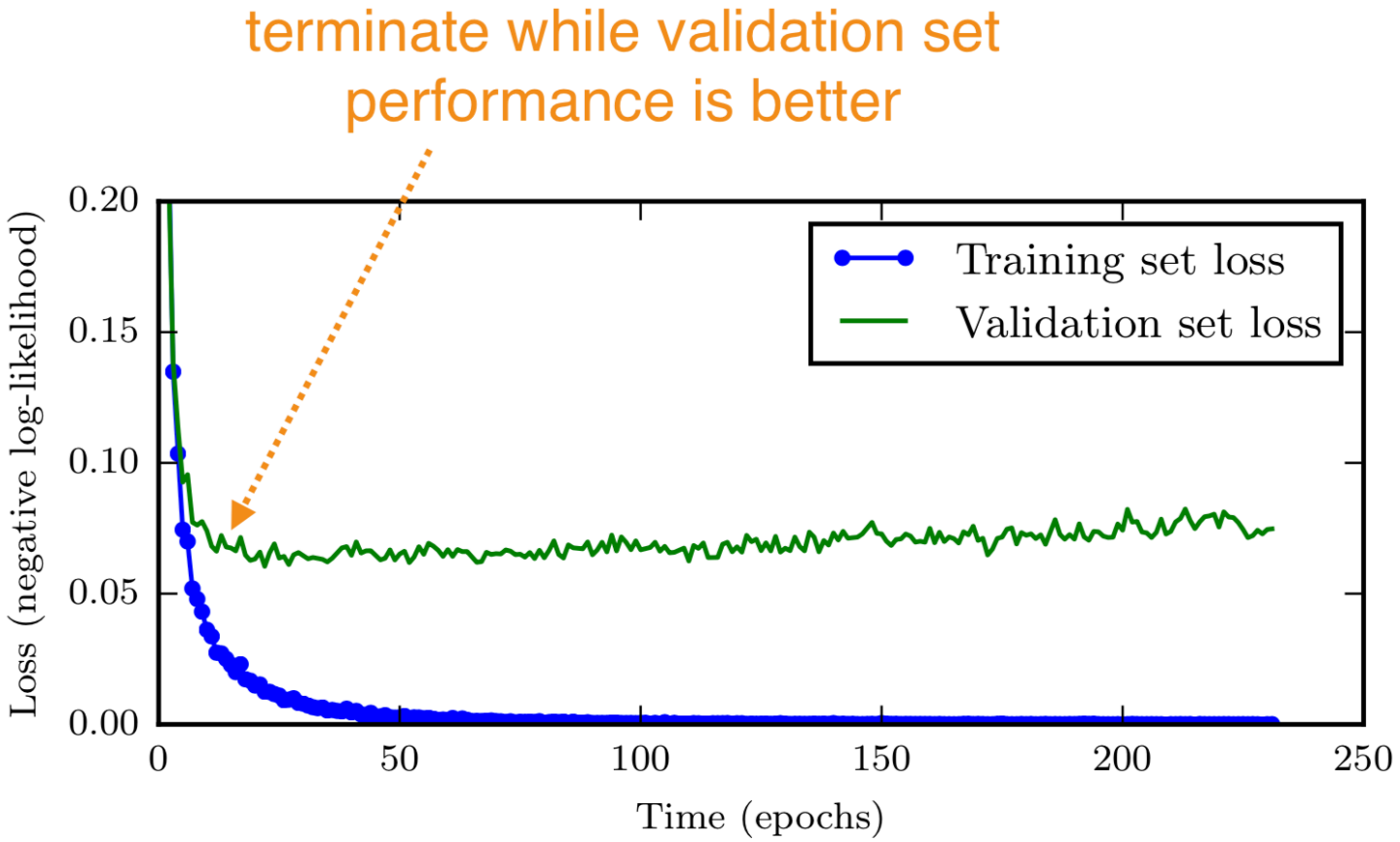

A single model can be used to simulate having a large number of different network architectures by randomly dropping out nodes during training. This is called dropout and offers a very computationally cheap and remarkably effective regularization method to reduce overfitting and improve generalization error in deep neural networks of all kinds.

A large network with more training and the use of a weight constraint are suggested when using dropout.

- Apply it both in forward and backward propagations
- *BUT* use it only in the *training phase !*

# Regularization (*Training*) : Early stopping

- Limit the number of iterations




*Stop the training when dev set error starts increasing again*

# Case

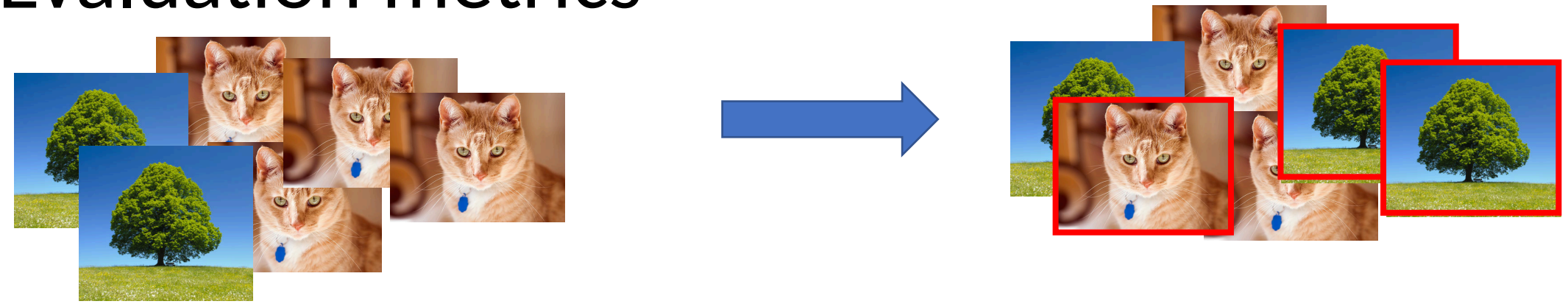
- You want to find cats in images

- Classification error (portion of wrong answers) used as an evaluation metric

| Algorithm | Classification error (%) |
|---|---|
| **A** | **3%** |
| **B** | **5%** |

➢ *Which one is best ?*

# Evaluation metrics



- Precision (p)

$$\text{Precision (\%)} = \frac{\textit{True positive}}{\textit{Number of predicted positive}} x\ 100 = \frac{\textit{True positive}}{(\textit{True positive}+\textit{False positive})}\ x\ 100$$

Precision is a good measure to determine, when the costs of False Positive is high (for example in healthcare).

$$\frac{2}{2+1} \times 100 = 66\%$$

- Recall (r)

$$\text{Recall (\%)} = \frac{\textit{True positive}}{\textit{Number of predicted actually positive}} x\ 100 = \frac{\textit{True positive}}{(\textit{True positive}+\textit{False negative})}\ x\ 100$$

Conversely, Recall is the best metric when there is a high cost of false negative.

$$\frac{2}{2+2} \times 100 = 50\%$$

# F1-score

- F1-score is a harmonic mean combining p and r

$$\text{F1-Score} = \frac{2}{\frac{1}{p}+\frac{1}{r}}$$

| | Precision | Recall | F1 Score | |
|---|---|---|---|---|
| Algo 1 → | 0.5 | 0.4 | 0.444 | ✓ |
| Algo 2 → | 0.7 | 0.1 | 0.175 | |
| Algo 3 → | 0.02 | 1.0 | 0.0392 | |

F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives)