

Advanced Deep Learning techniques

ERA

Jiří Materna



Machine
Learning
College

About me

- Ph.D. in Natural Language Processing and Artificial Intelligence at Masaryk University
- 10 years at seznam.cz (last 8 years as Head Of Research)
- Founder and co-organizer of ML Prague
- Mentor at StartupYard
- ML Freelancer and consultant

Outline

- Neural Network architectures
- Optimizers and their evolution
- Loss functions and their properties
- Initialization of weights in neural networks
- Normalization and Regularization in neural networks
- Functional model definition in Keras
- Semi-supervised learning
- AutoML approaches
- ML Explainability

Neural Network architectures design



Neural Network design best practices

- ★ Start from simple architectures
- ★ Get inspiration from architectures for similar problems
- ★ Change one parameter only and then validate

Most common architectures

Feed forward network

Convolutional network

Recurrent network

Autoencoder network and Restricted Boltzmann Machines

Transformer network

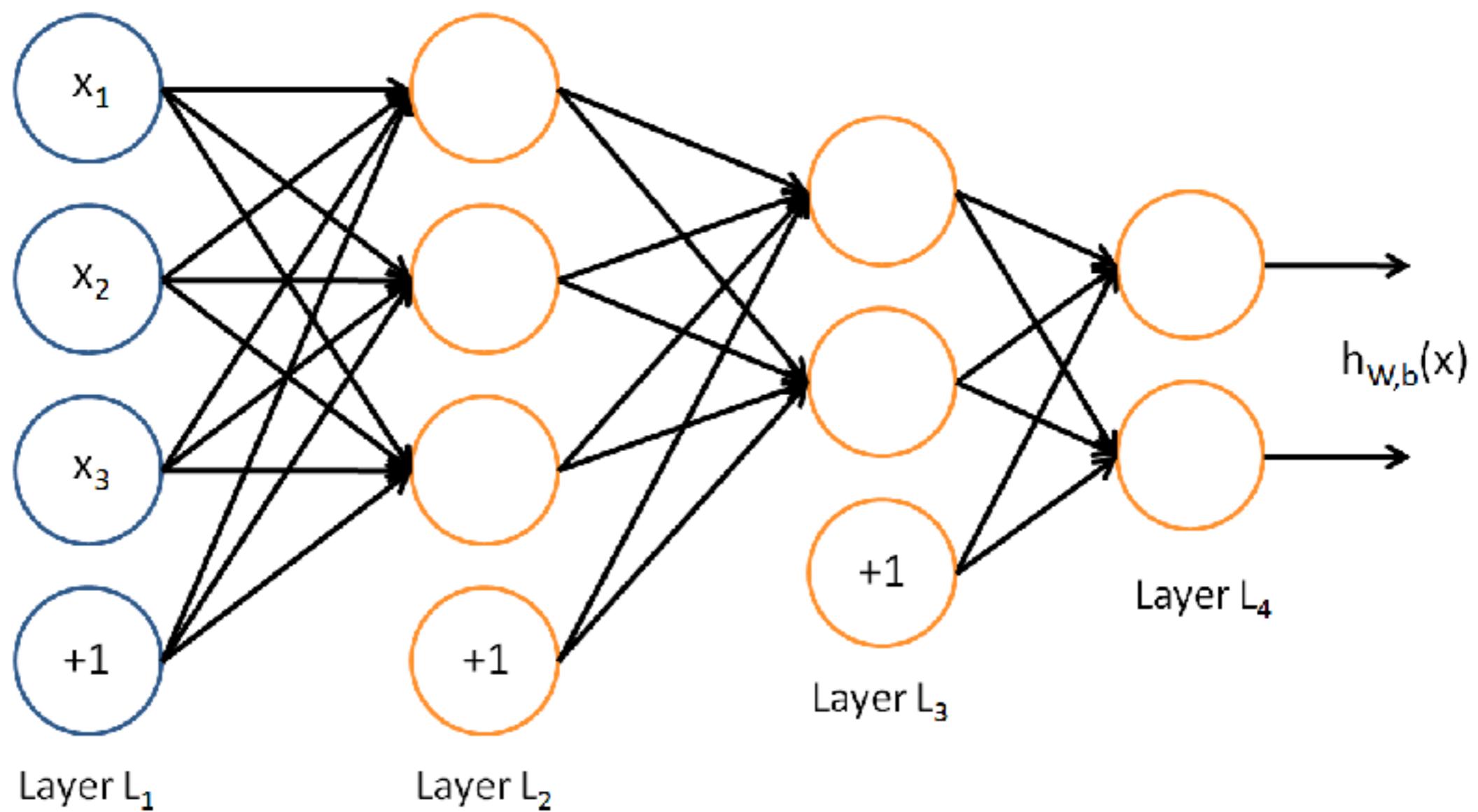
Yolo

U-Net

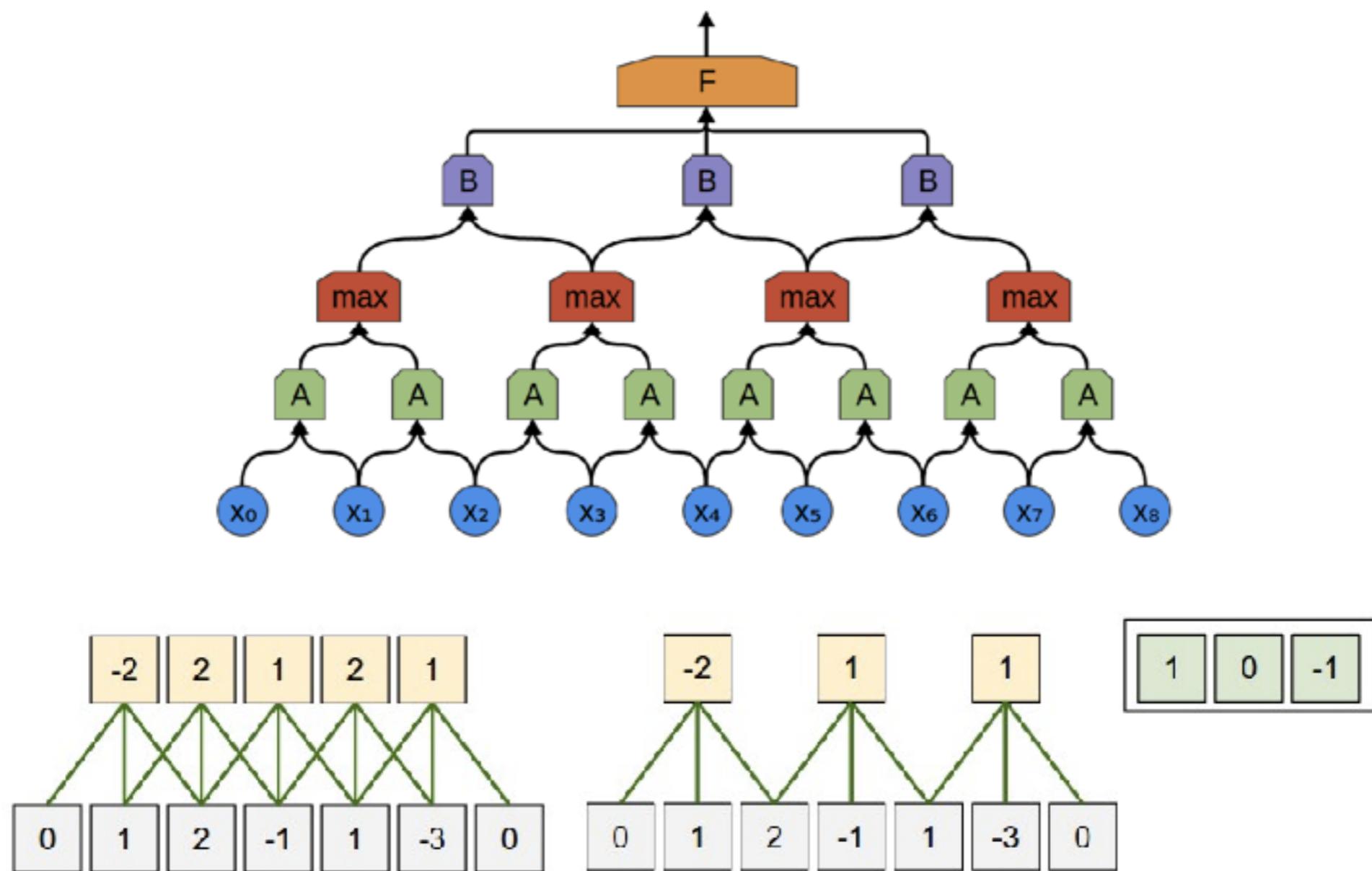
Generative adversarial network

Diffusion models

Feed Forward Network

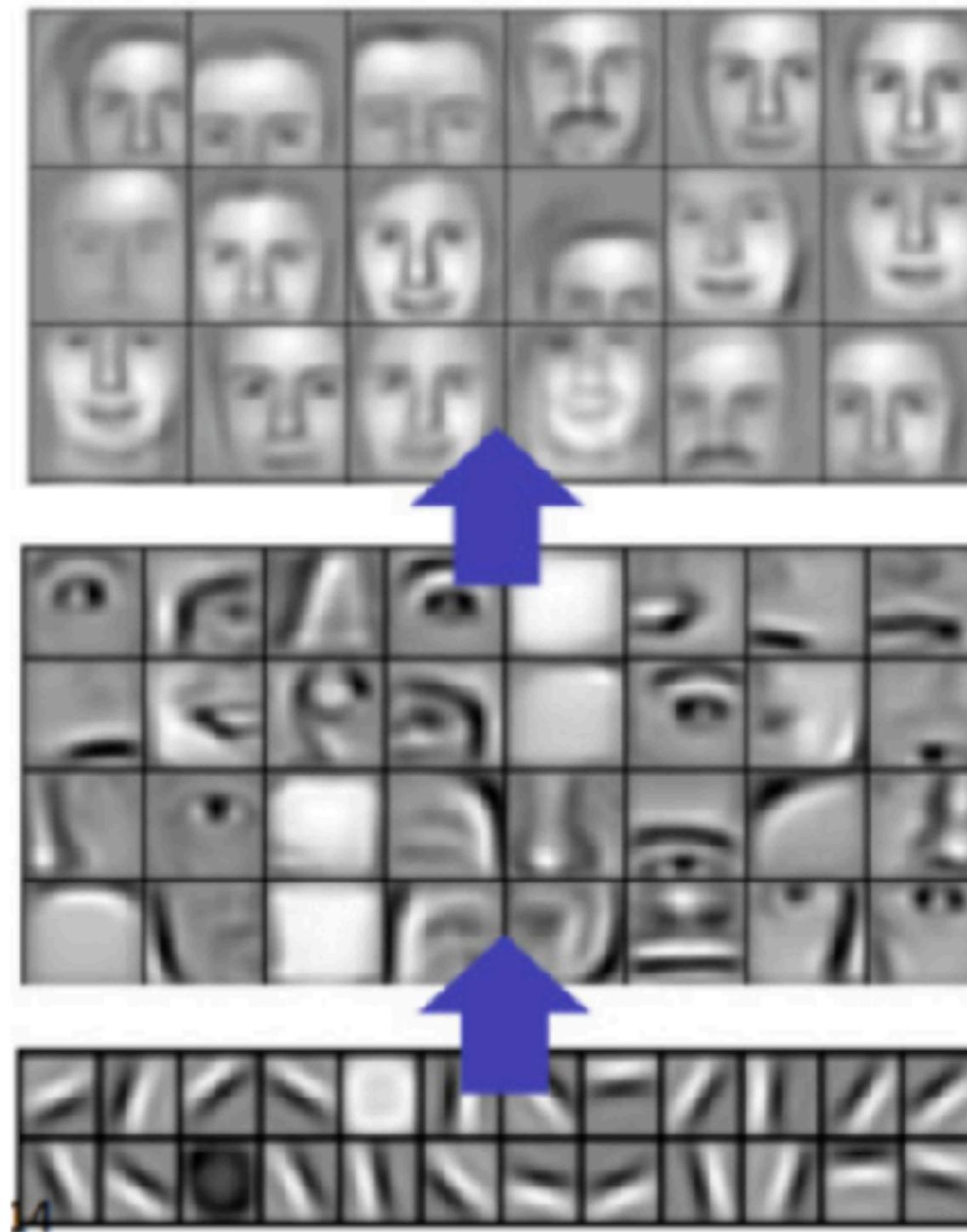


Convolution



Source: <https://www.tensorflow.org>

Weights visualization



Layer 3

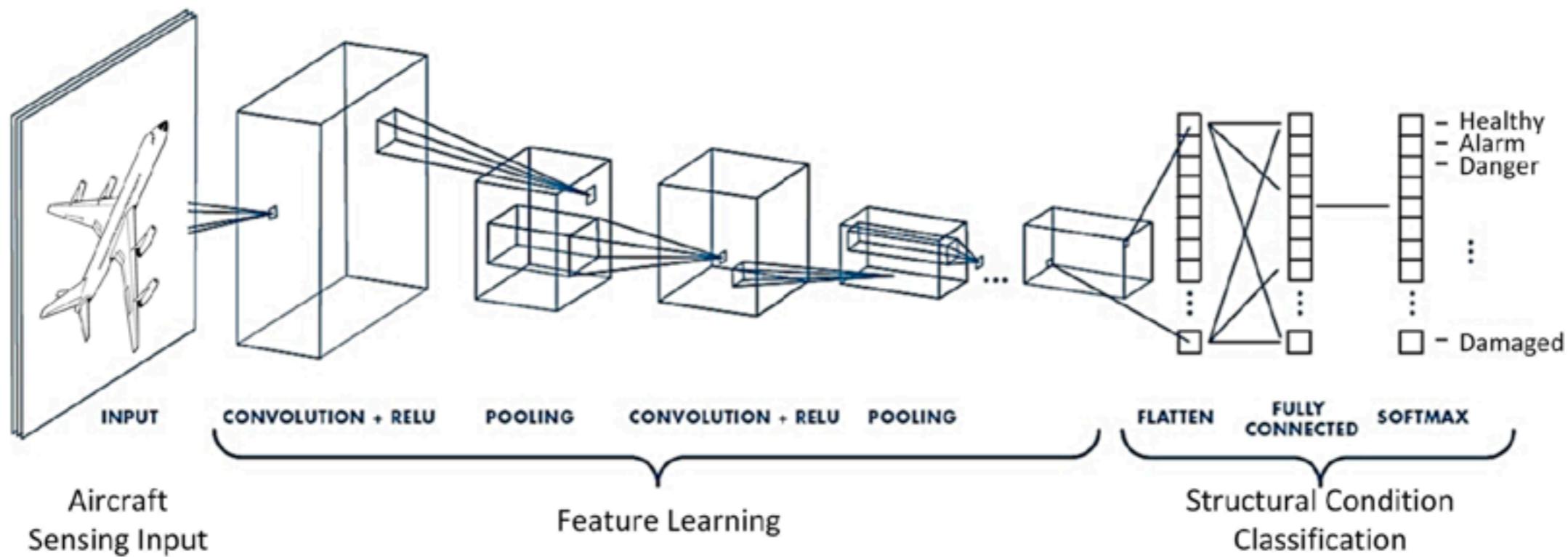
Layer 2

Layer 1

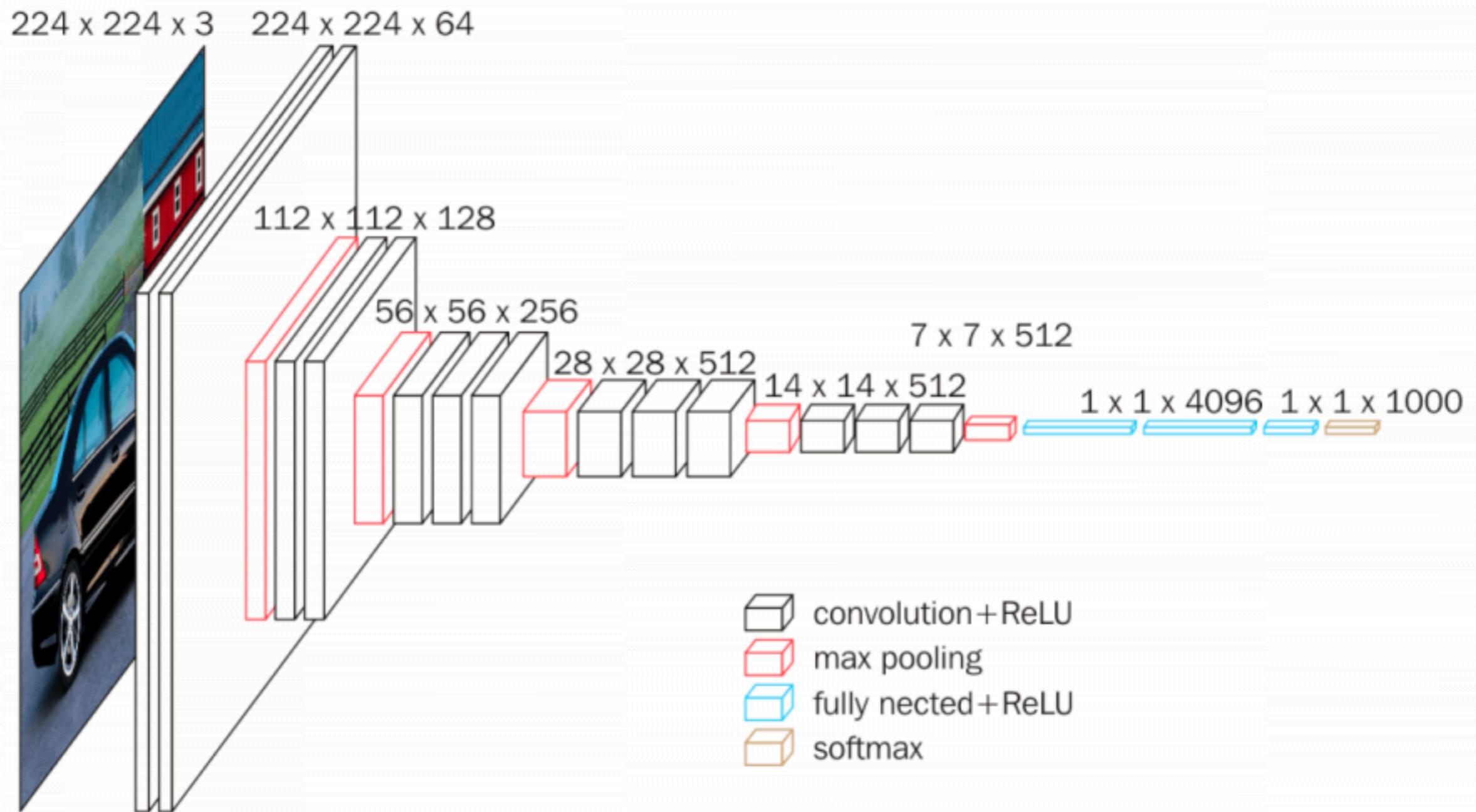
MNIST Classification

[01-MNIST-classification.ipynb](#)

Convolutional Neural Network

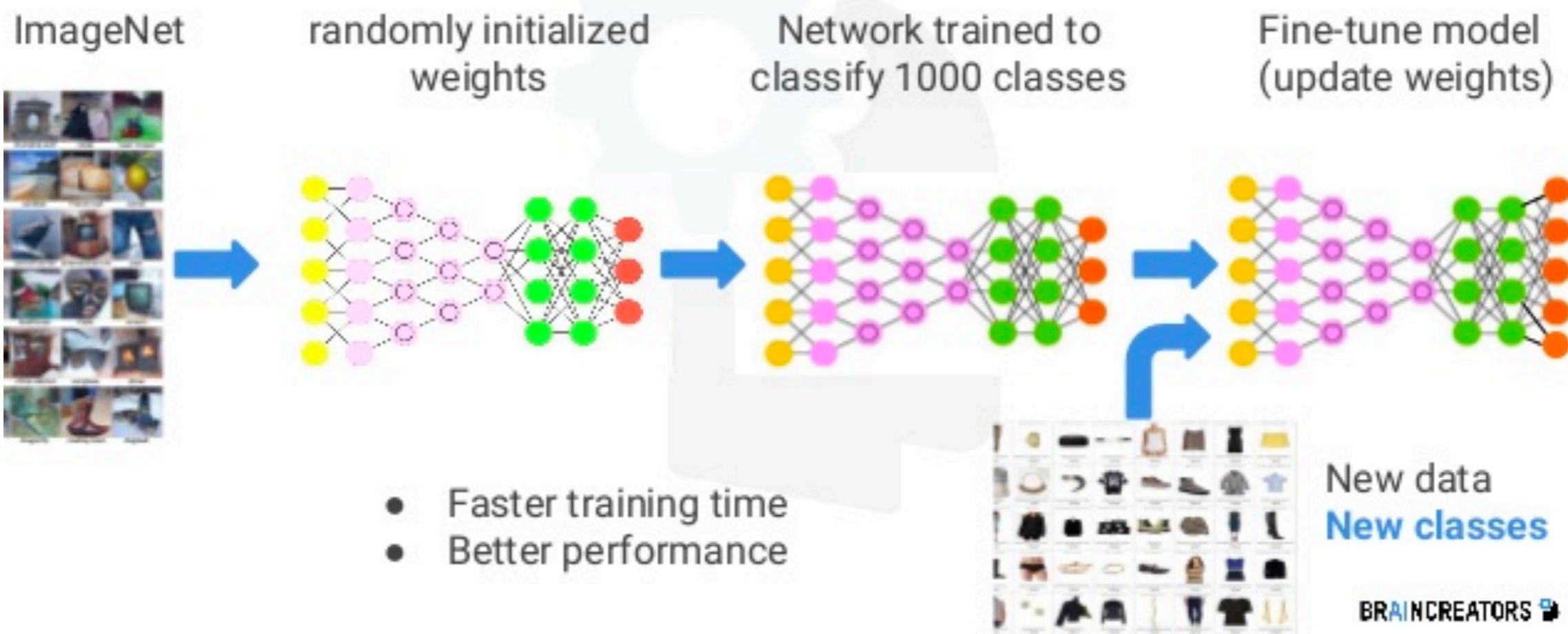


VGG 16



Finetuning

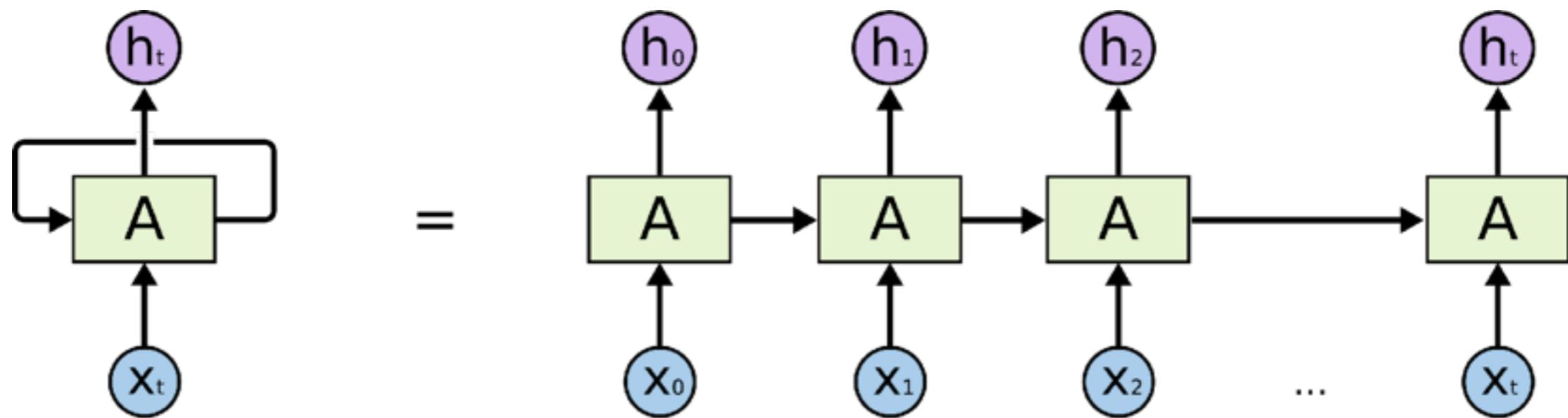
Transfer Learning



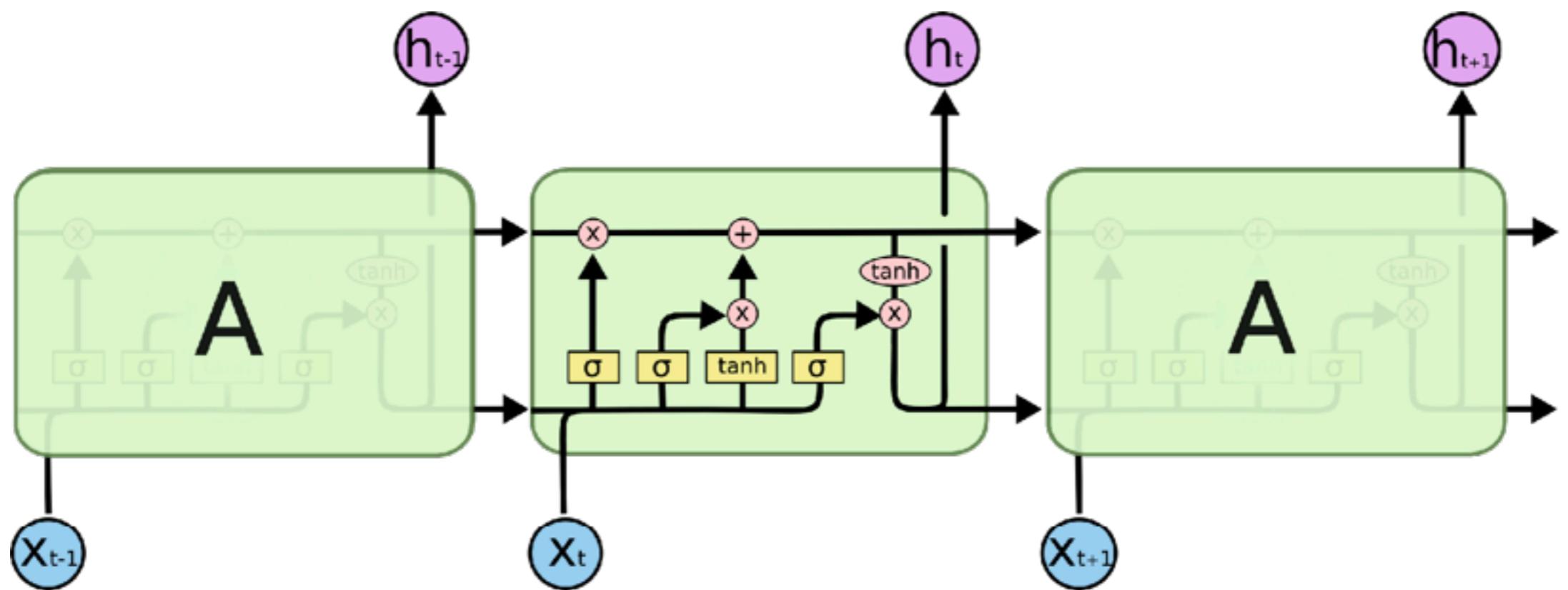
Transfer learning example

02-Transfer_learning.ipynb

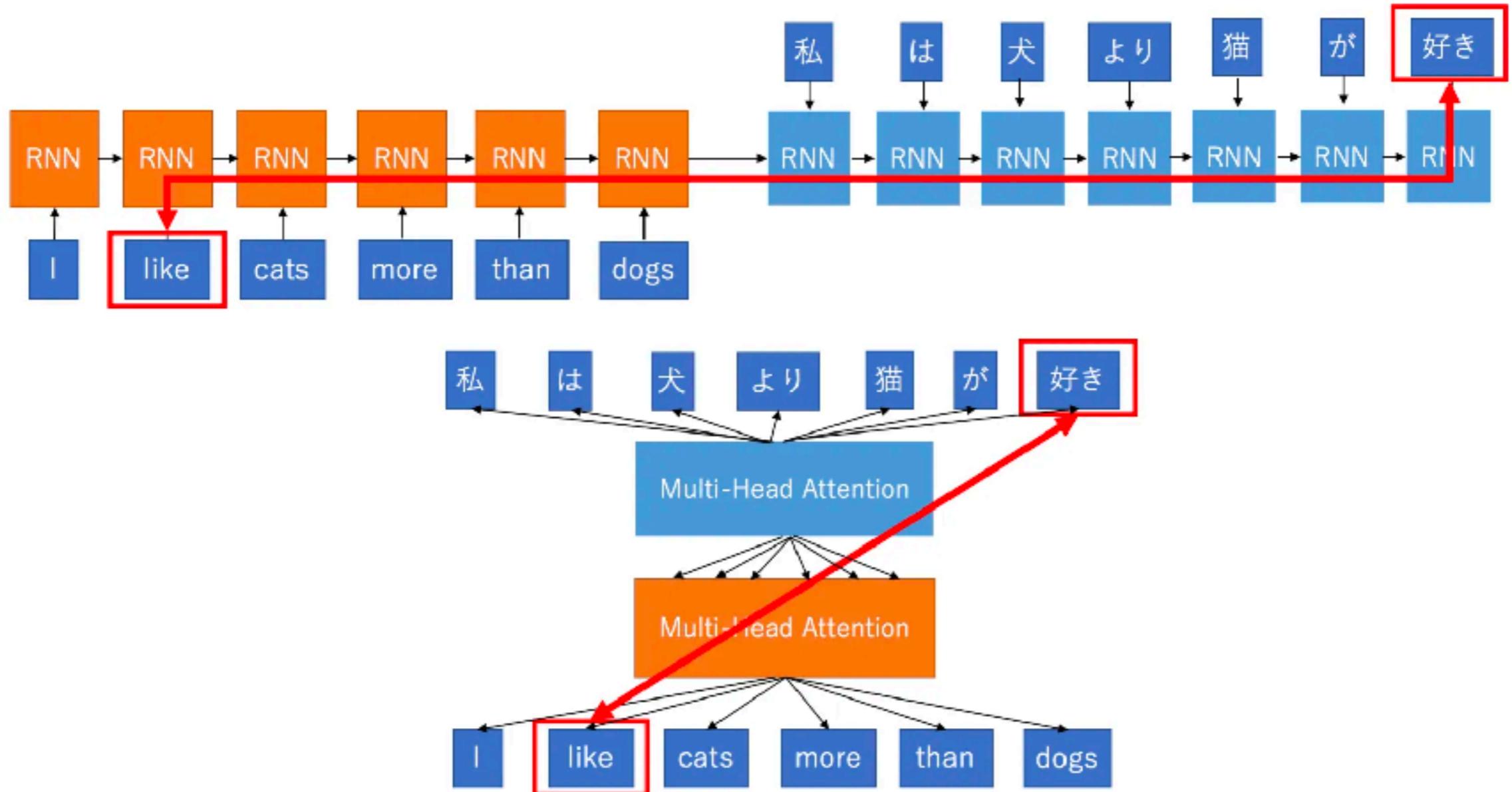
Recurrent Neural Network



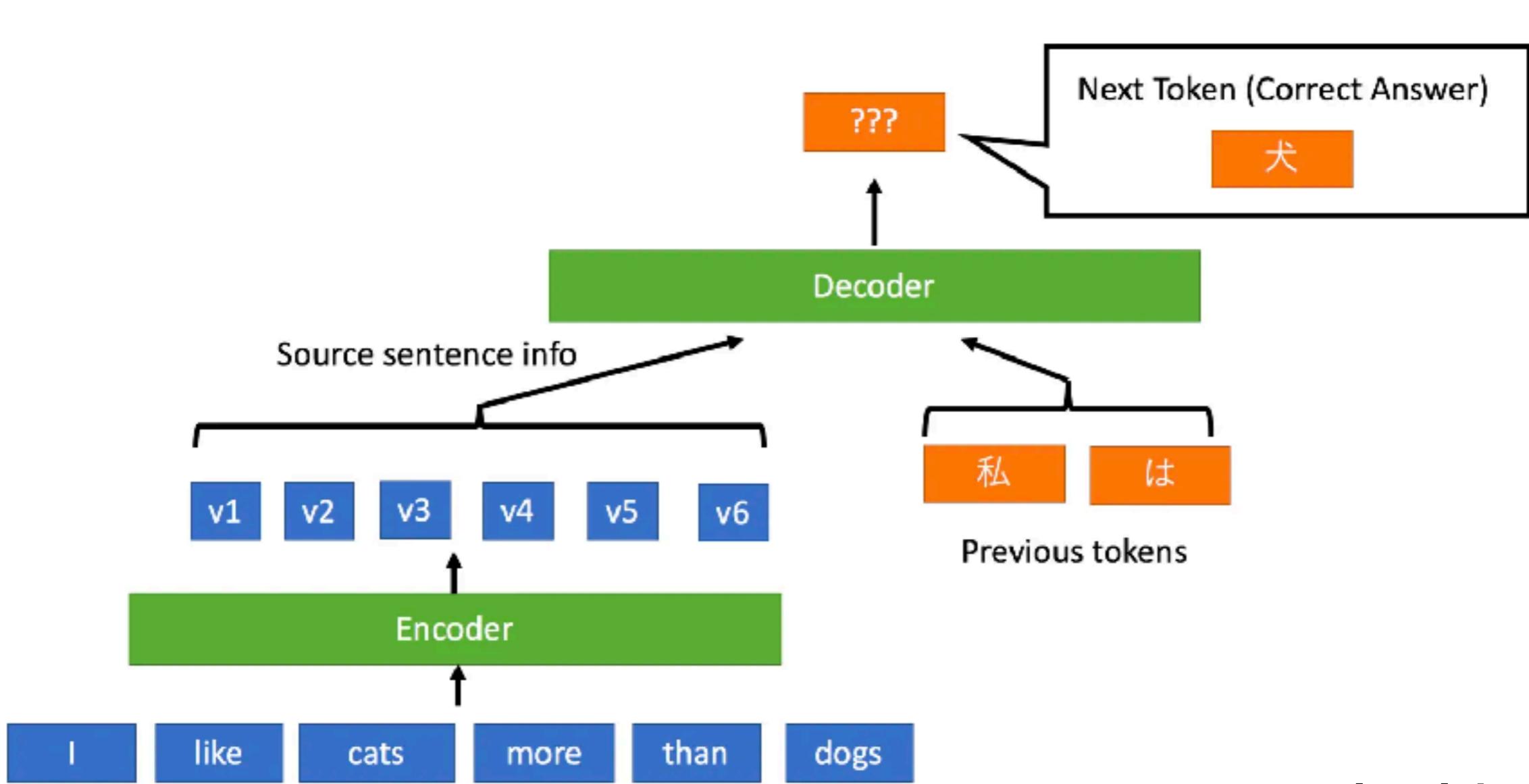
Long Short Term Memory



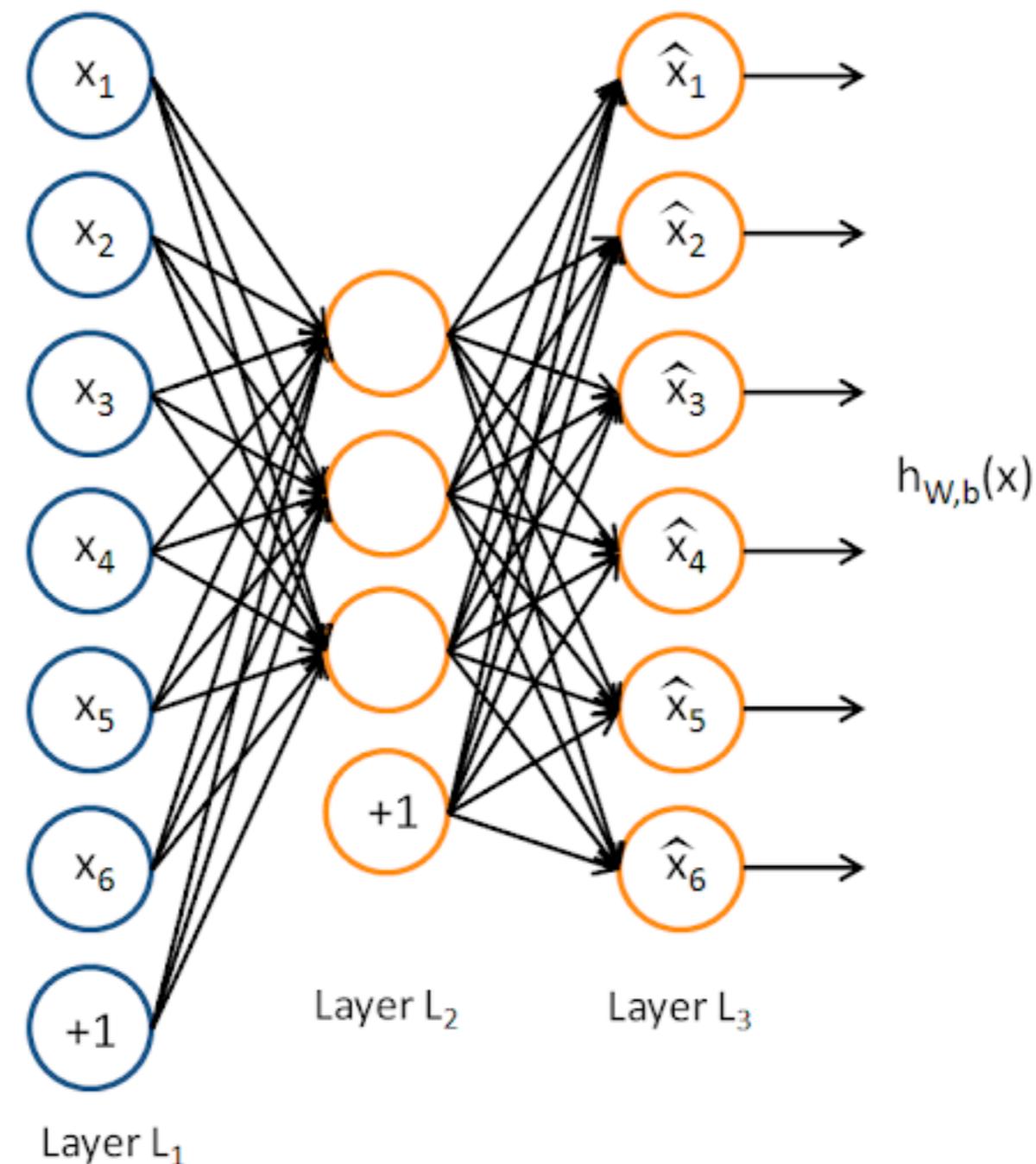
Transformer



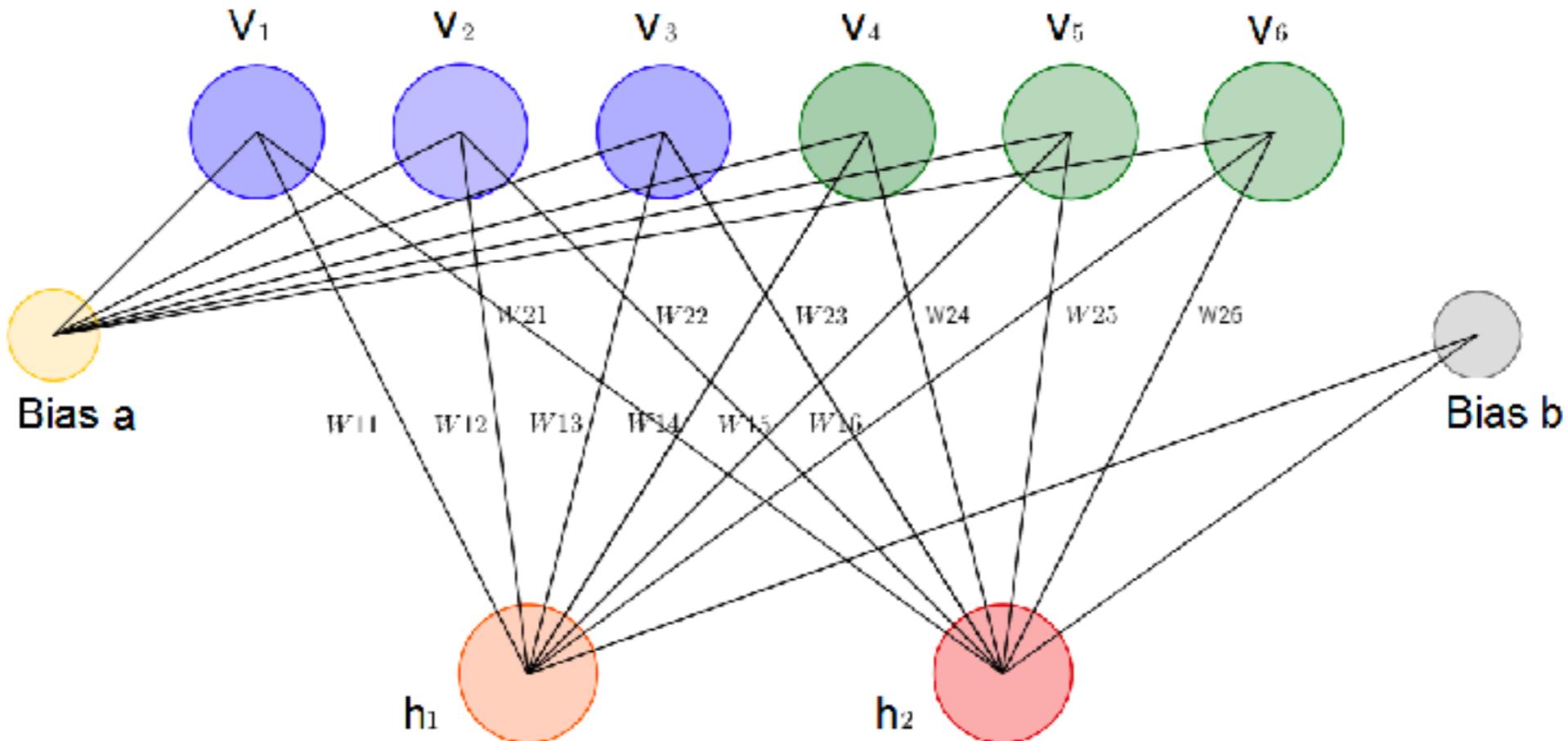
Machine Translation with Transformers



Autoencoder Network



Restricted Boltzmann Machine



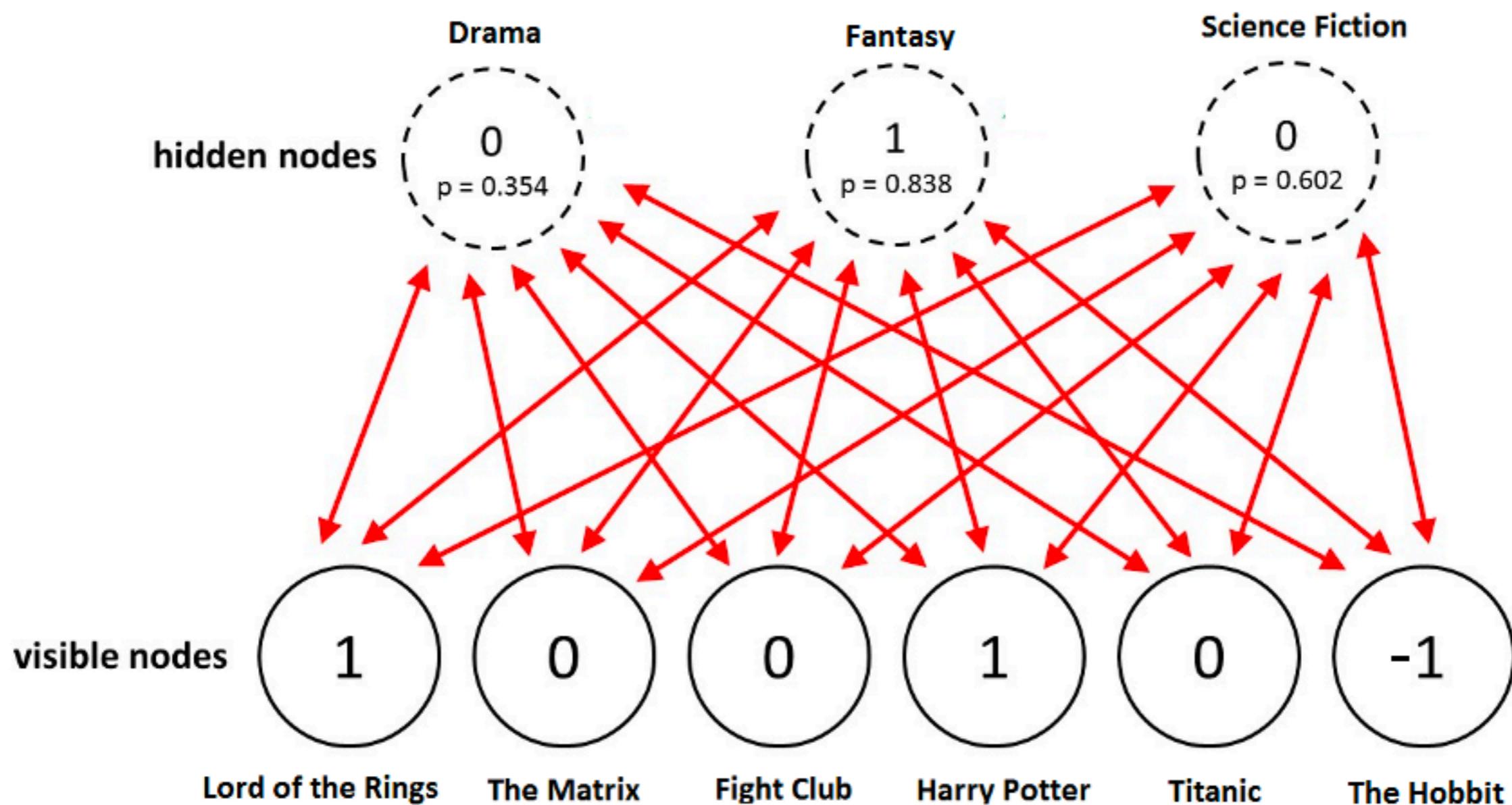
$$E(\mathbf{v}, \mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

Gibbs sampling + contrastive divergence

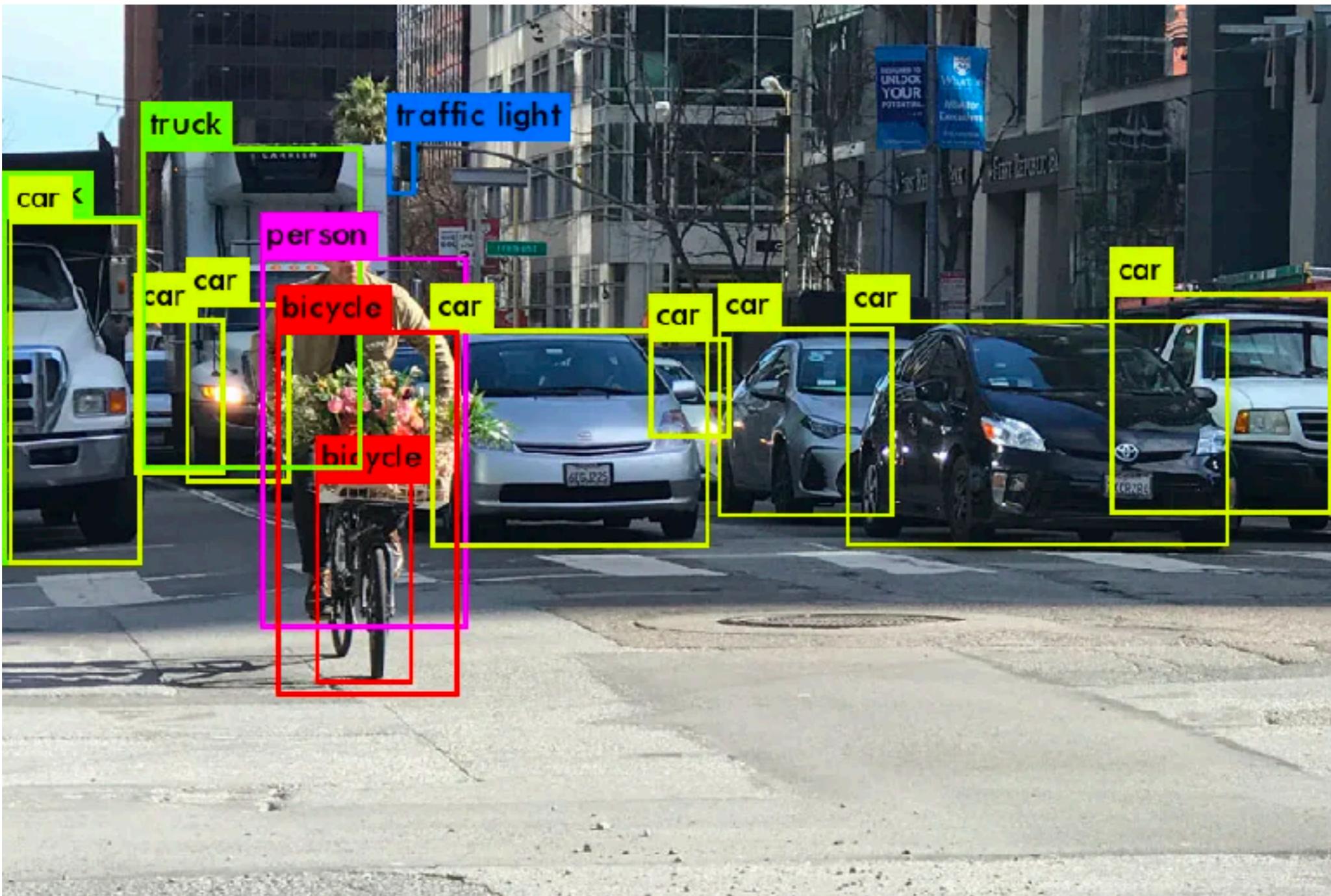
$$p(h_j = 1 | \mathbf{v}) = \frac{1}{1 + e^{(-(b_j + W_j v_i))}} = \sigma(b_j + \sum_i v_i w_{ij})$$

$$p(v_i = 1 | \mathbf{h}) = \frac{1}{1 + e^{(-(a_i + W_i h_j))}} = \sigma(a_i + \sum_j h_j w_{ij})$$

Collaborative filtering with Restricted Boltzmann Machine



Object detection



YOLO v1 (You Only Look Once)

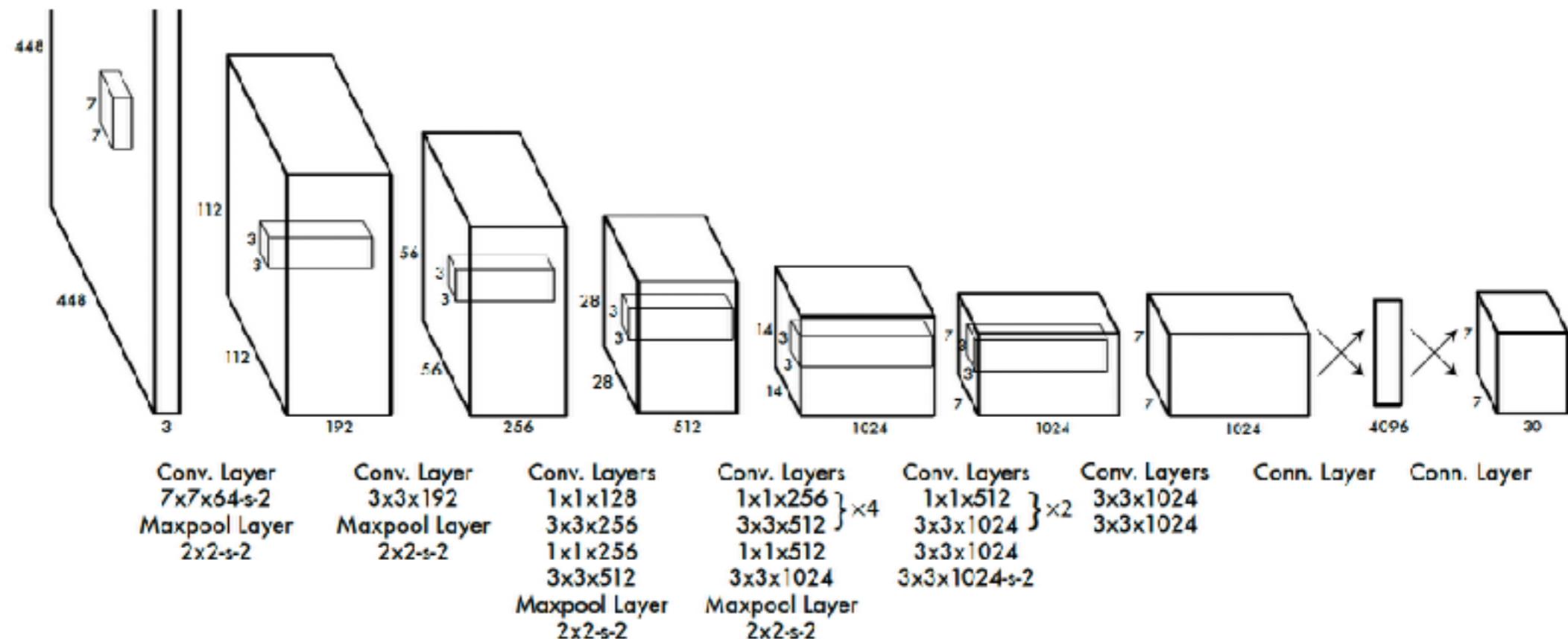


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

YOLO (You Only Look Once)

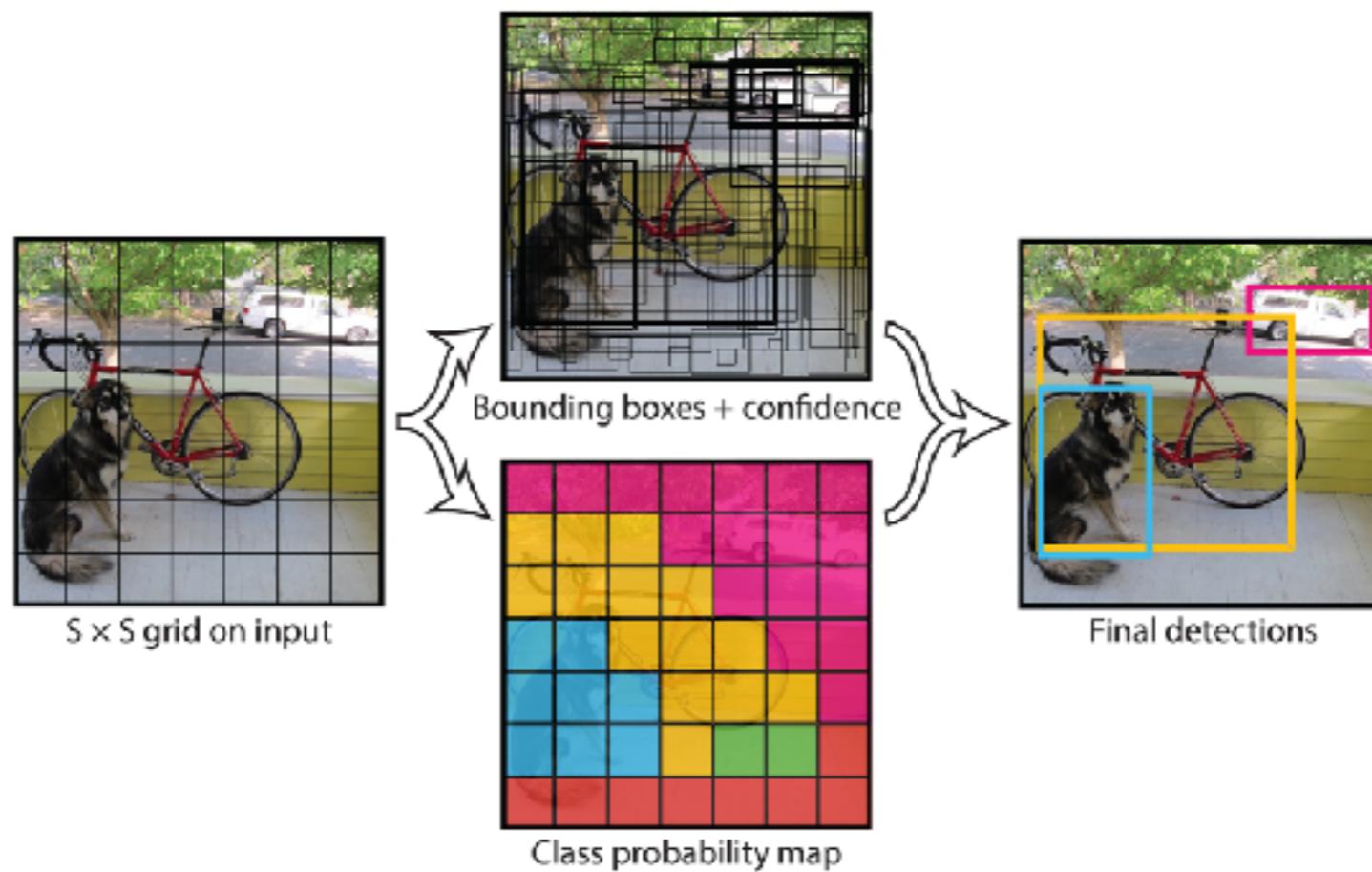


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

YOLO v8 usage

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8n.yaml") # build a new model from scratch
model = YOLO("yolov8n.pt") # load a pretrained model (recommended for training)

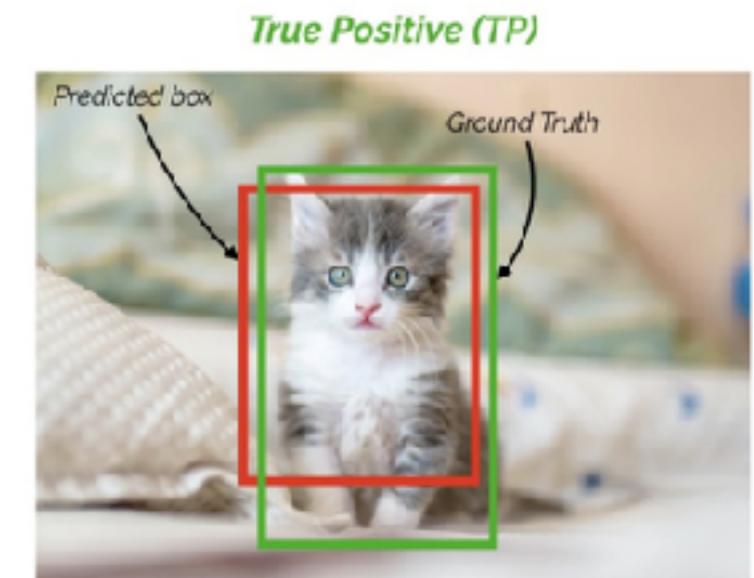
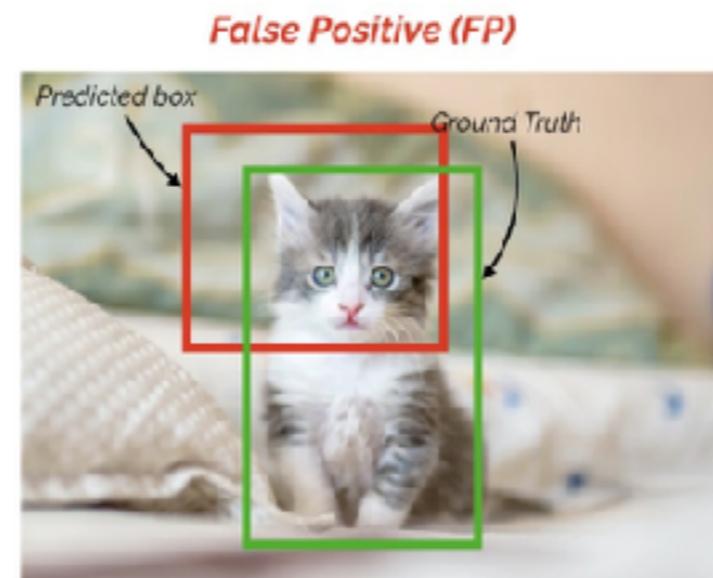
# Use the model
model.train(data="coco128.yaml", epochs=3) # train the model
metrics = model.val() # evaluate model performance on the validation set
results = model("https://ultralytics.com/images/bus.jpg") # predict on an image
path = model.export(format="onnx") # export the model to ONNX format
```

Object detection metrics

IoU

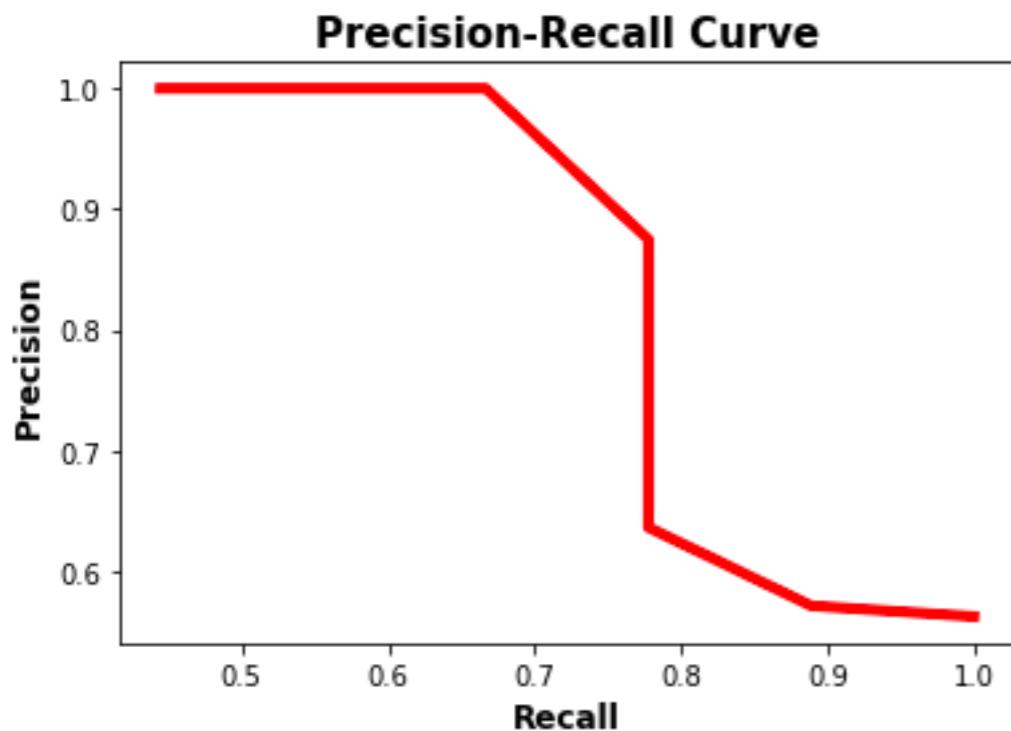
If IoU threshold = 0.5

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



Object detection metrics

mAP



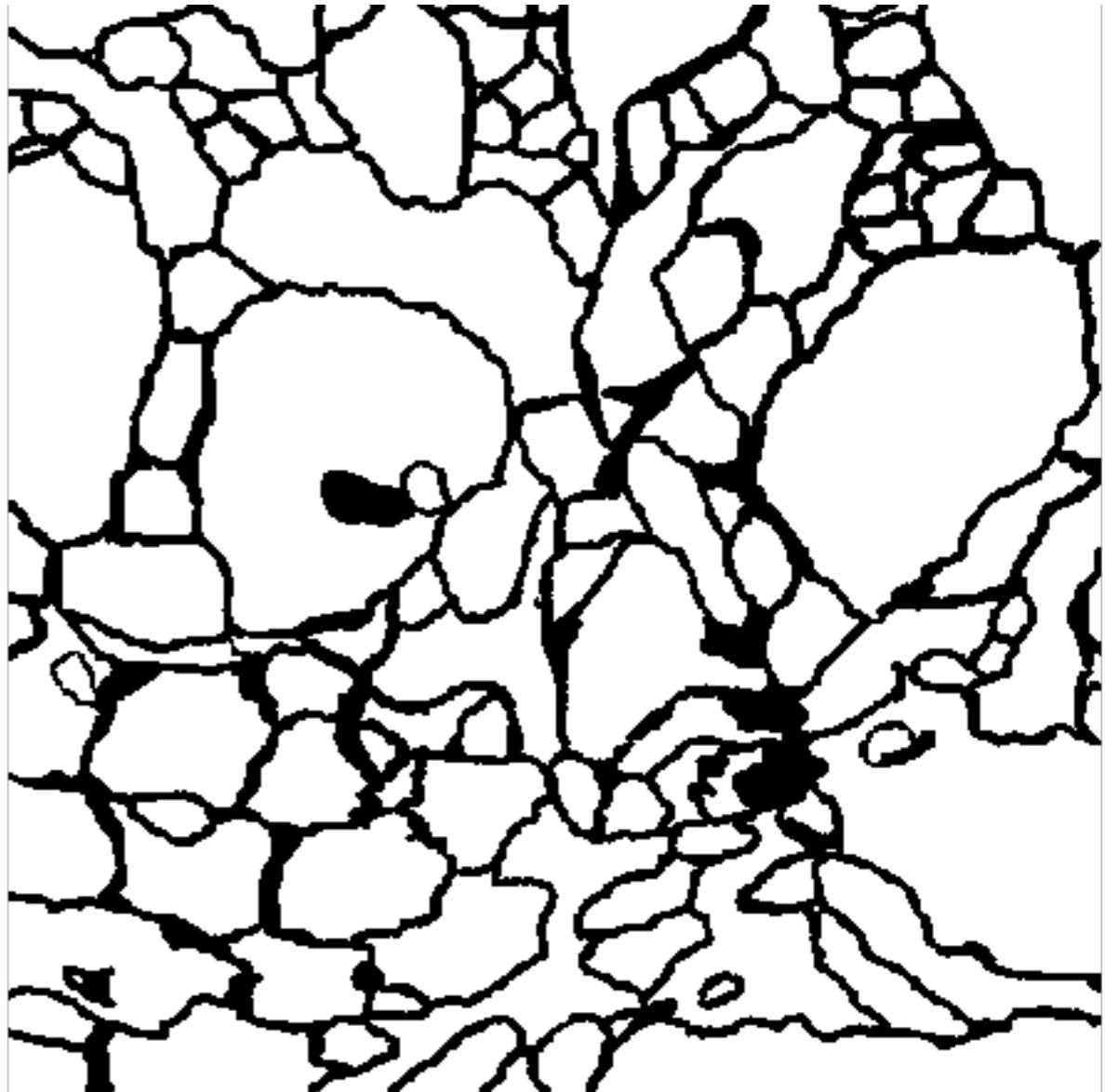
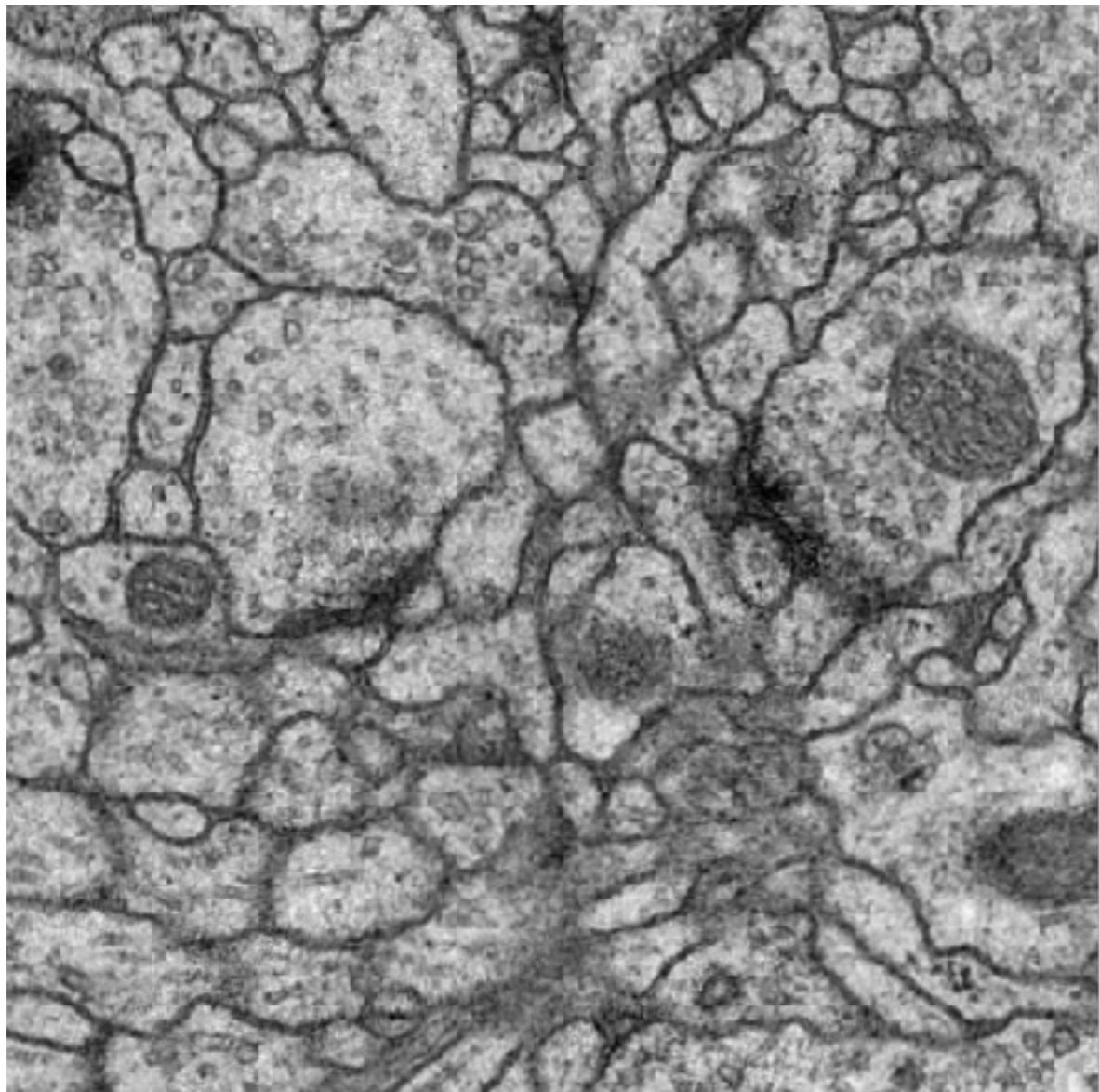
$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k + 1)] * Precisions(k)$$

Recalls(n) = 0, Precisions(n) = 1
n = Number of thresholds.

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

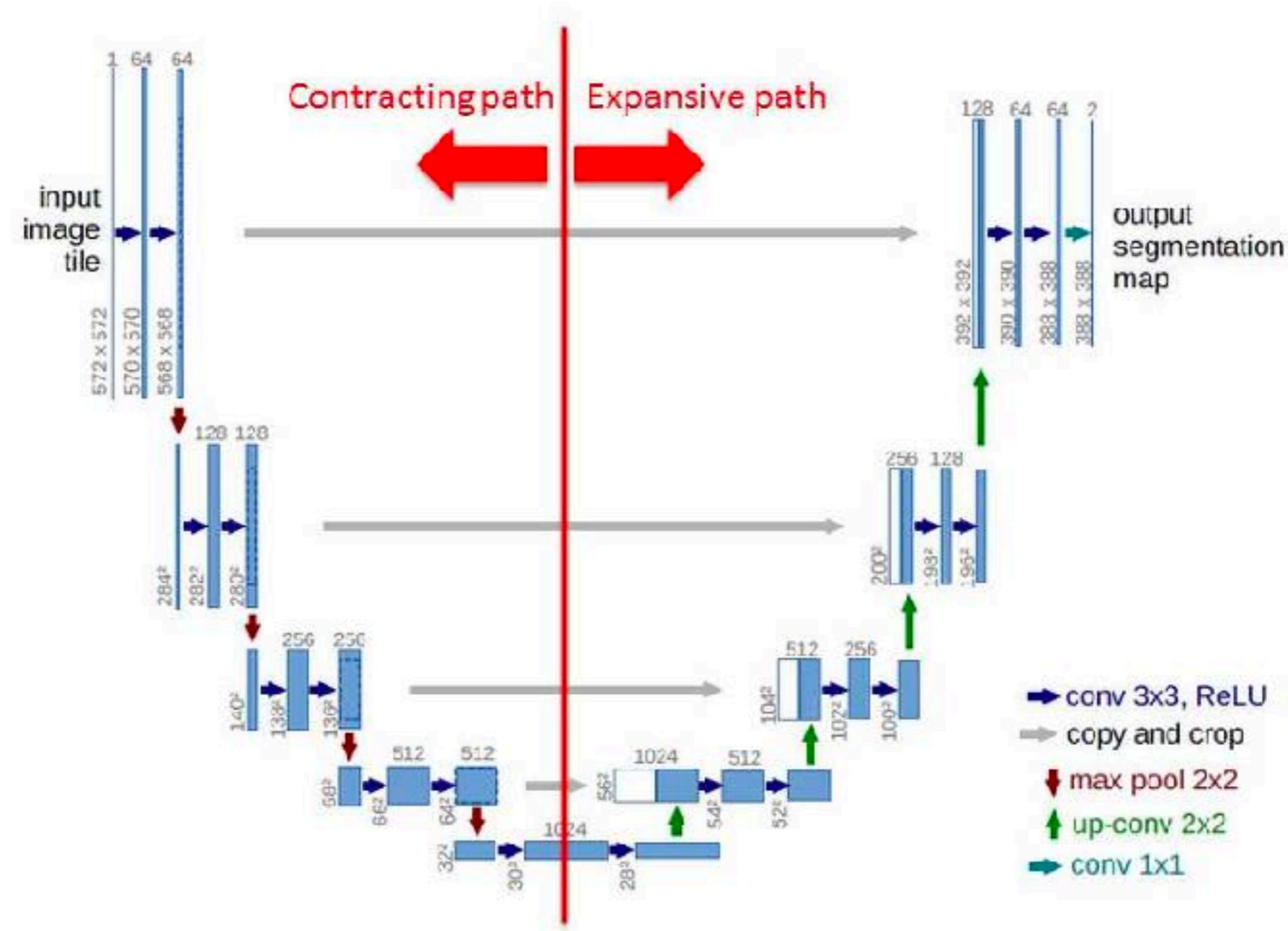
AP_k = the AP of class k
n = the number of classes

Image segmentation



U-Net

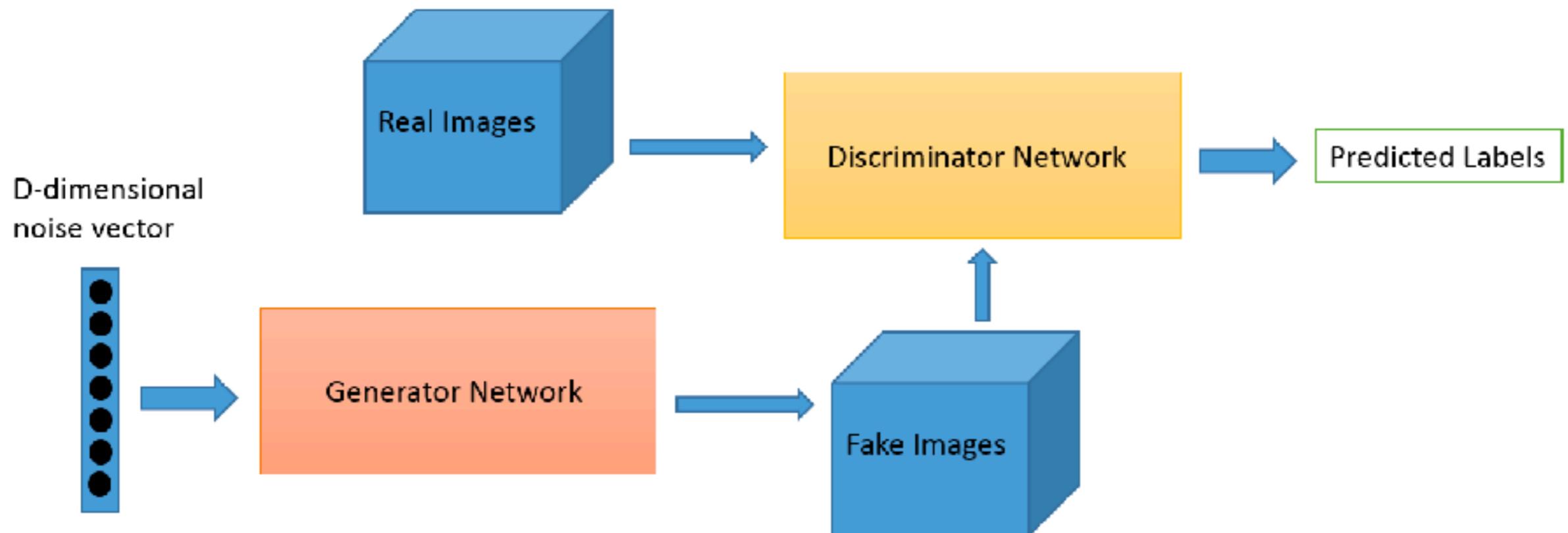
Network Architecture



U-Net segmentation example

03-Segmentation.ipynb

Generative Adversarial Networks



Superresolution

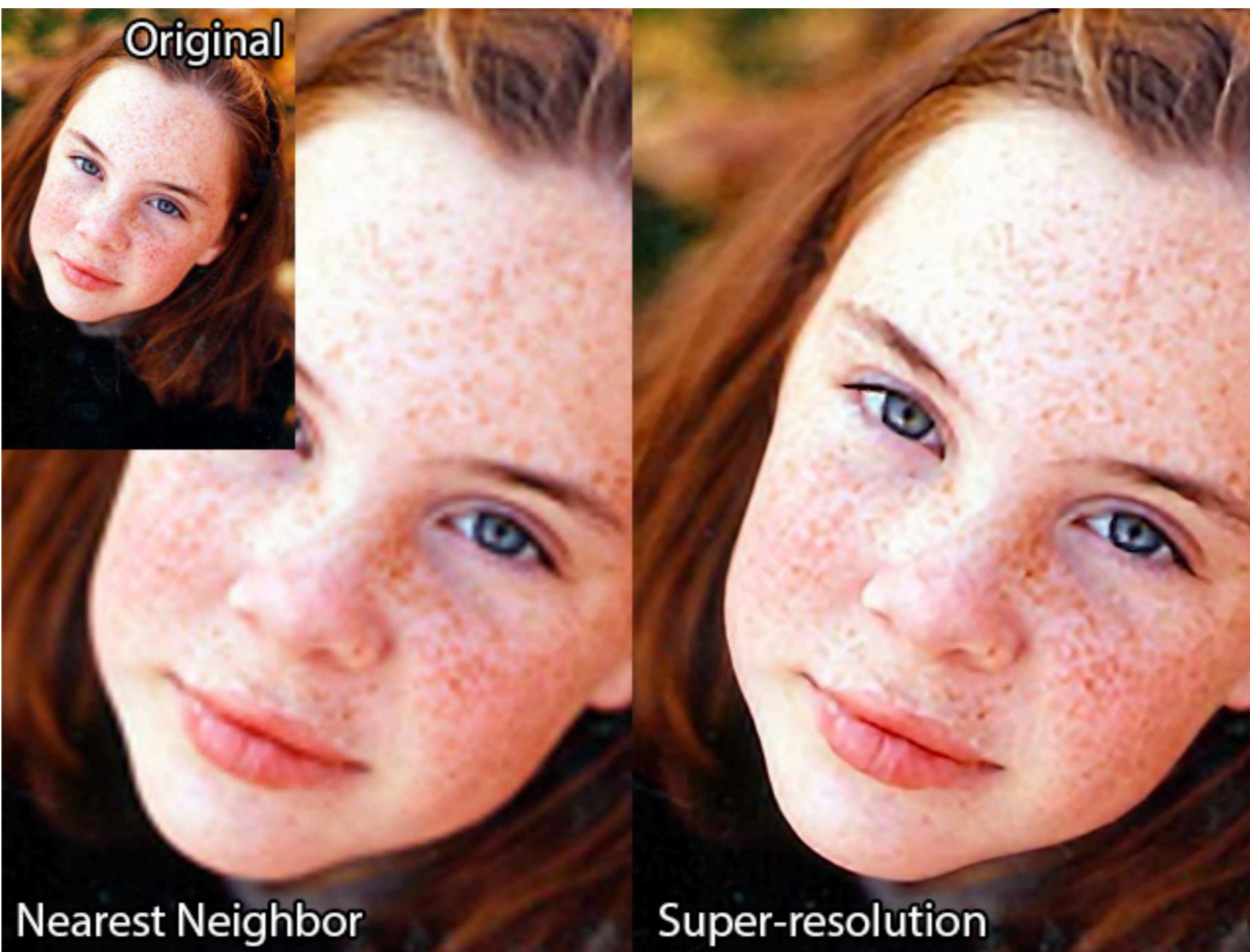
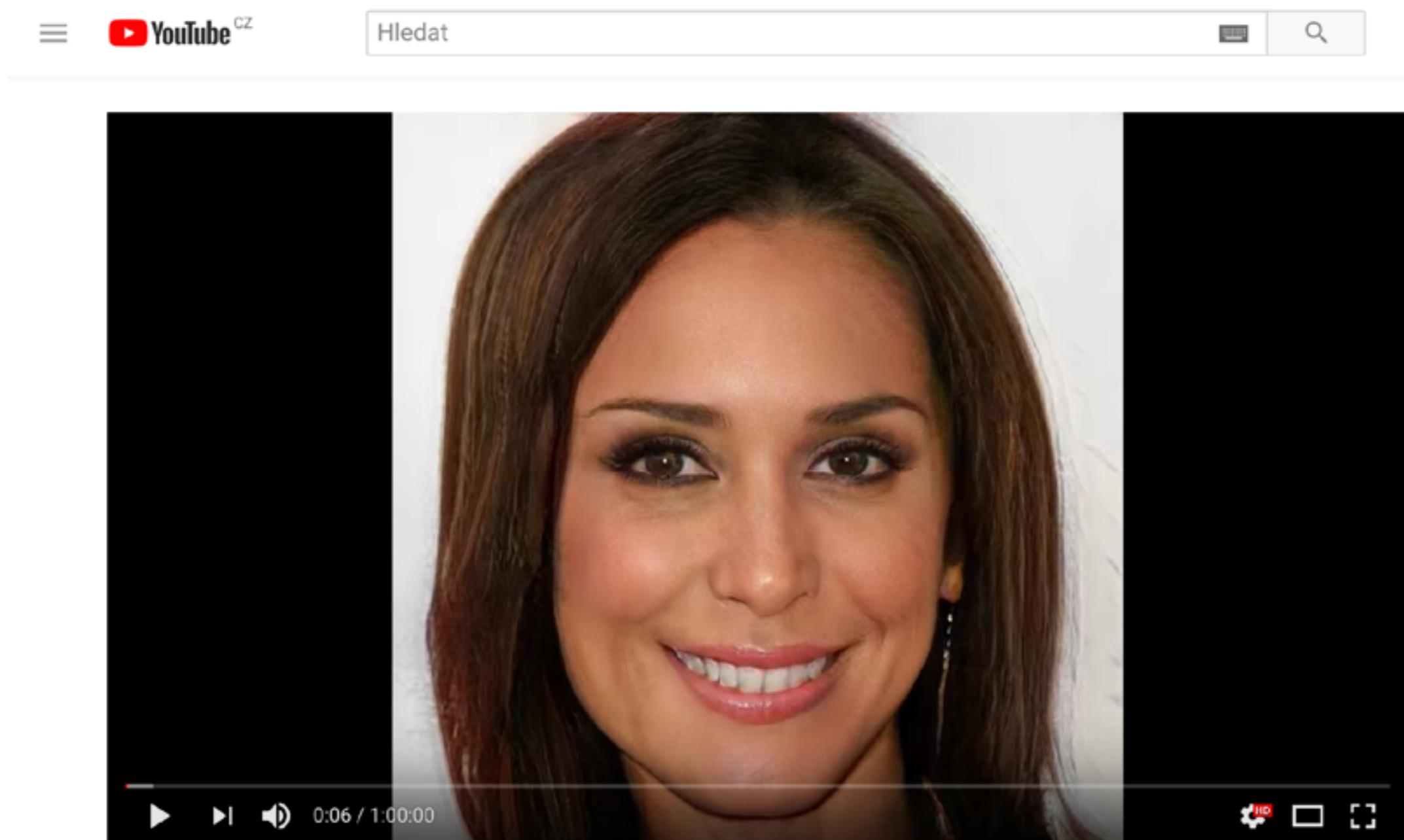


Image synthesis



One hour of imaginary celebrities

95 832 zhlédnutí

TO SE MI LÍBÍ NELÍBÍ SE SDÍLET ...

Which one is fake?

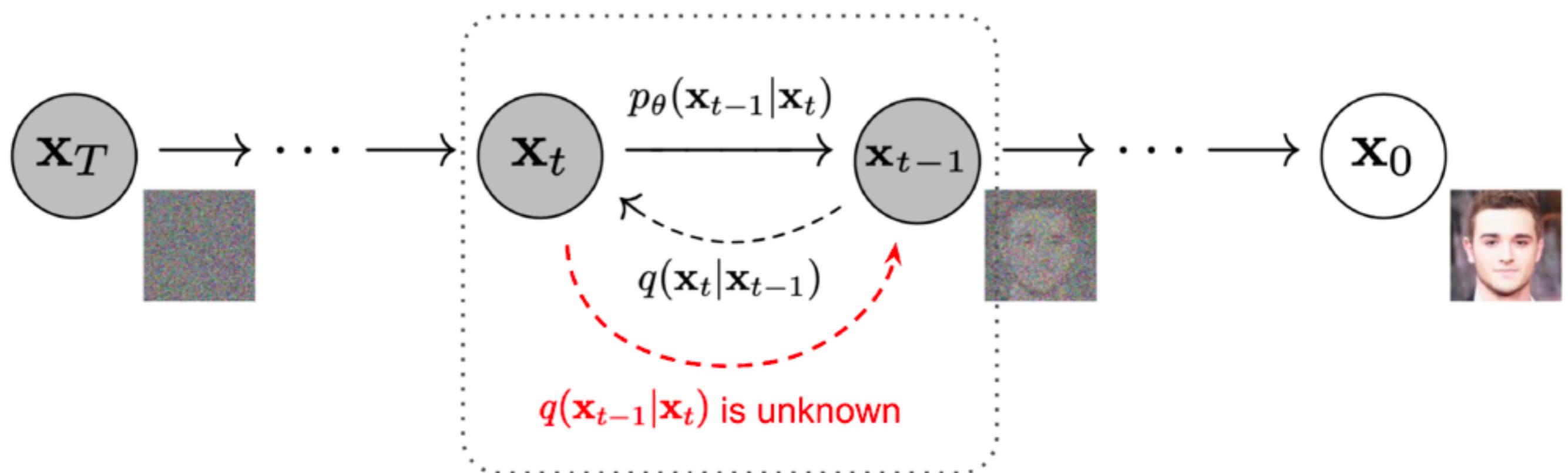


Image manipulation

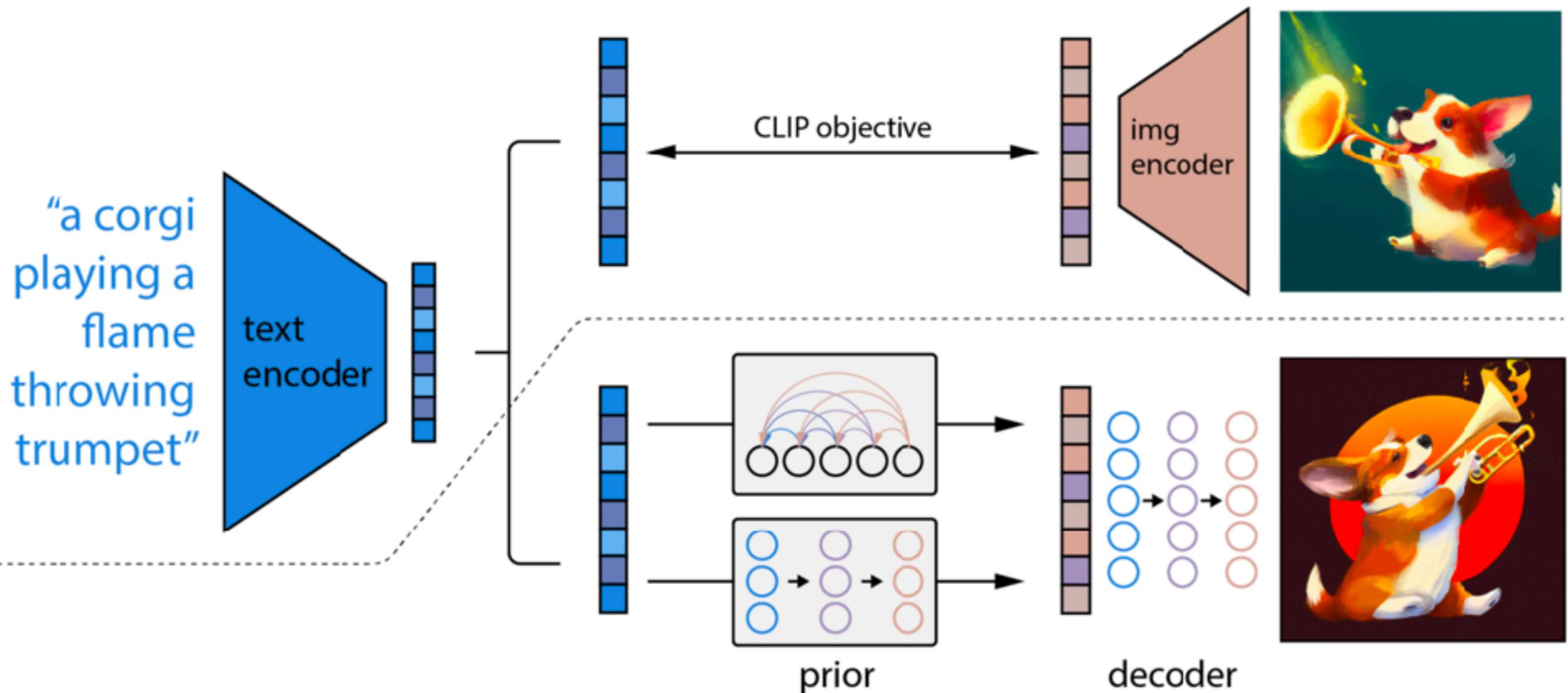


Diffusion models

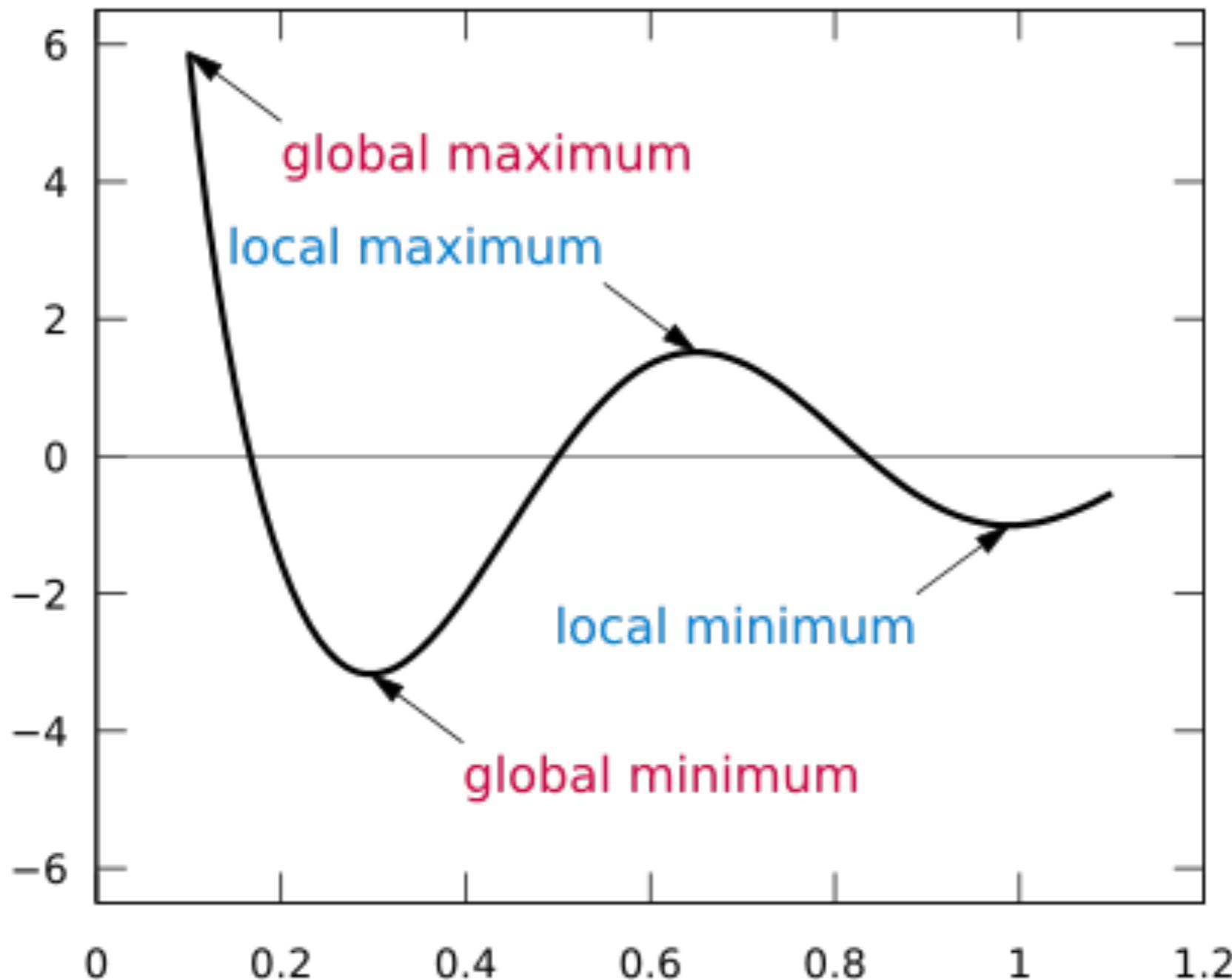
Use variational lower bound



Conditioned generation using Diffusion models

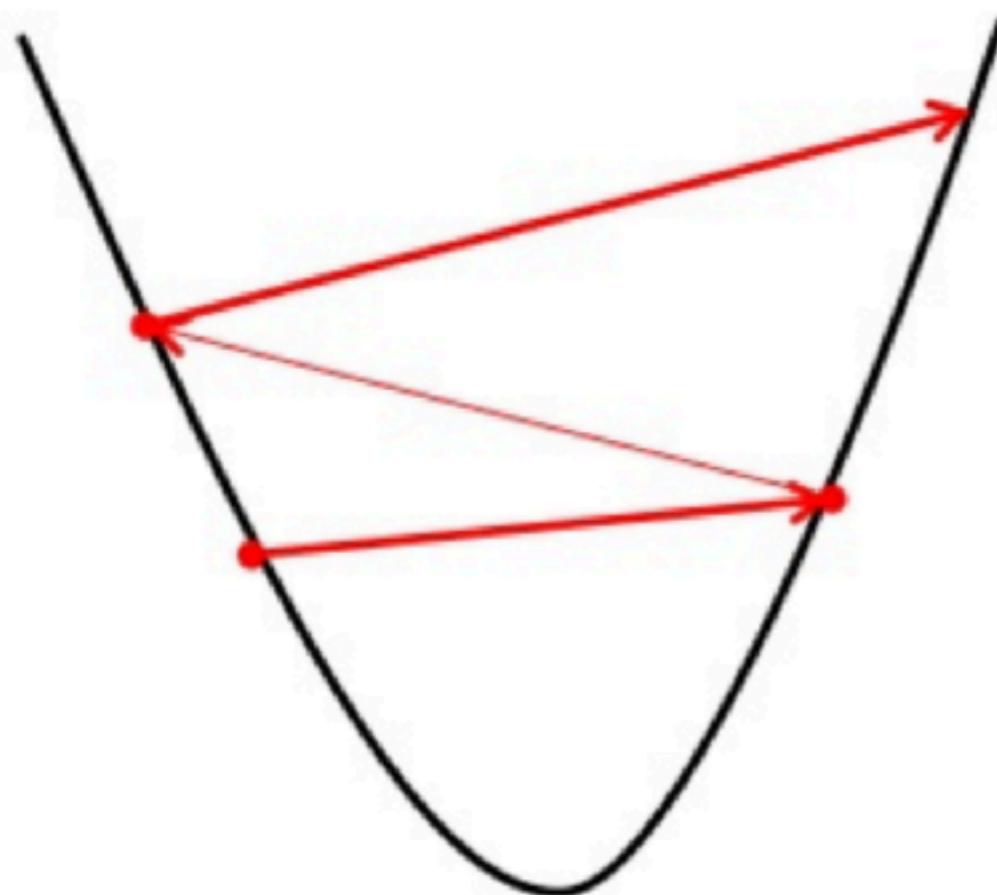


Parameter optimization strategies

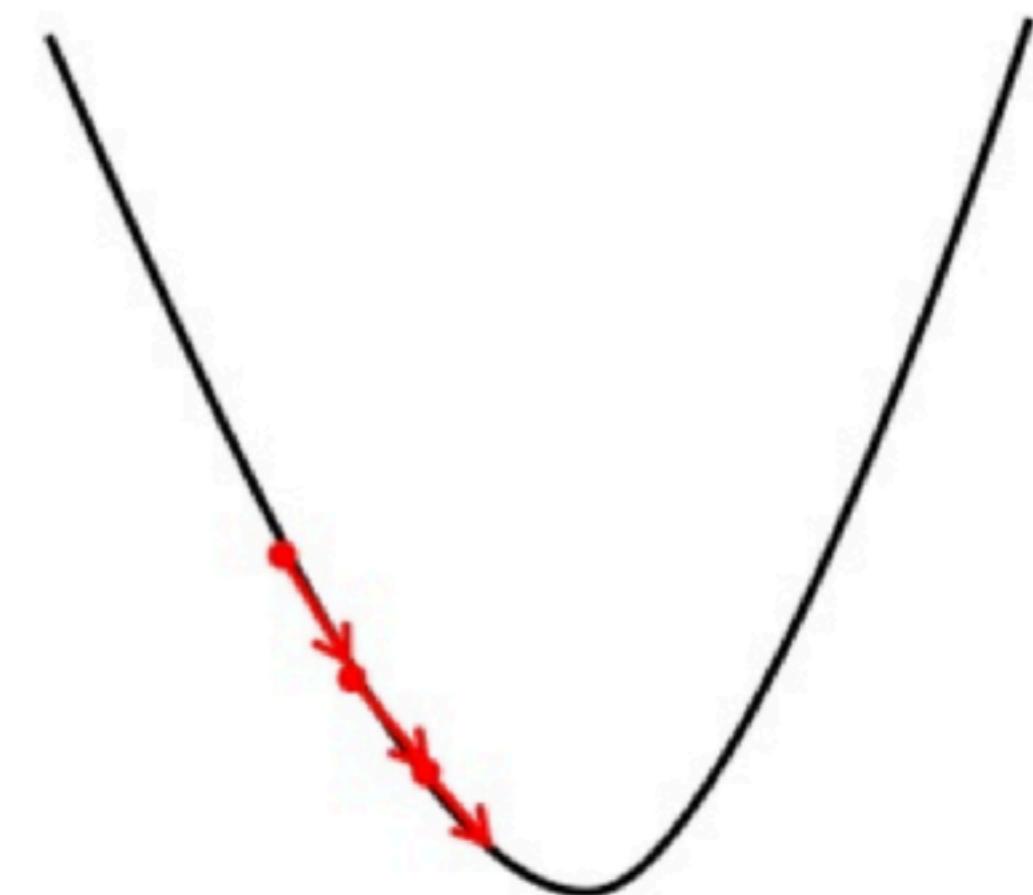


Learning rate tuning

Big learning rate



Small learning rate



Gradient Descent Variants

(Batch) Gradient Descent

$$w_{t+1} = w_t - \lambda \frac{\partial e(X, y)}{\partial w_t}$$

Stochastic Gradient Descent

$$w_{t+1} = w_t - \lambda \frac{\partial e(X^i, y^i)}{\partial w_t}$$

Mini-Batch Gradient Descent

$$w_{t+1} = w_t - \lambda \frac{\partial e(X^{(i,i+n)}, y^{(i,i+n)})}{\partial w_t}$$

Momentum and Nesterov Accelerated Gradient

$$v_t = \gamma v_{t-1} + \lambda \frac{\partial e(w_t)}{\partial w_t}$$

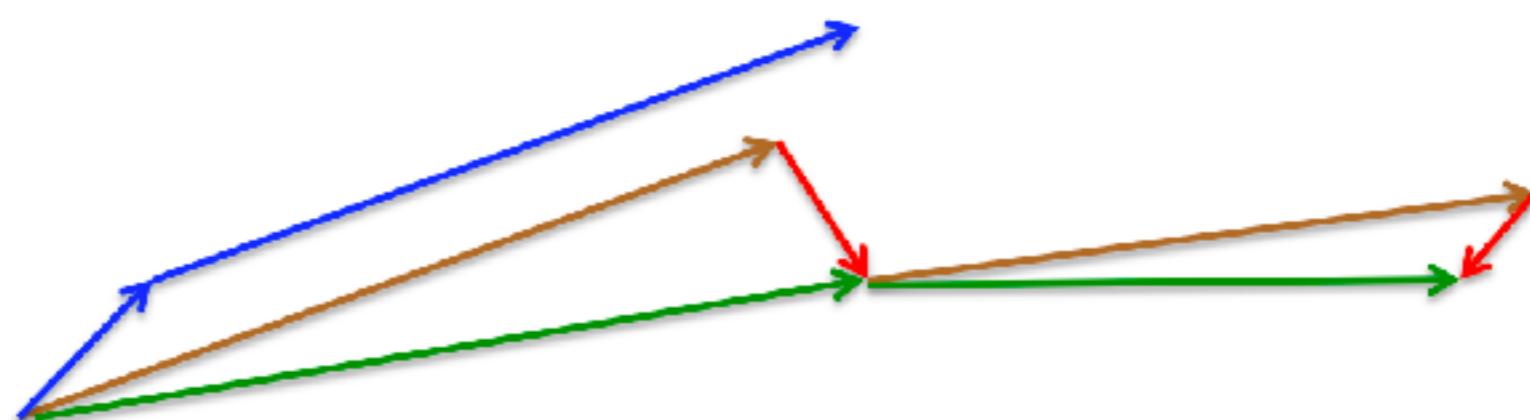
Naive momentum

$$w_{t+1} = w_t - v_t$$

$$v_t = \gamma v_{t-1} + \lambda \frac{\partial e(w_t - \gamma v_{t-1})}{\partial w_t}$$

Nesterov Accelerated Gradient

$$w_{t+1} = w_t - v_t$$



Adaptive Gradient Algorithms

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\sum_{i=1}^t g_i^2 + \epsilon}} g_t$$

Adagrad

$$g_i = \frac{\partial e(w_i)}{\partial w_i}$$

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\mathbb{E}[g^2]_t - \epsilon}} g_t^2$$

RMSProp

$$\mathbb{E}[g^2]_t = \gamma \mathbb{E}[g^2]_{t-1} + (1 - \gamma) g_t^2$$

Adam and Nadam

Adam

Combination of RMSProp with momentum

Nadam

Combination of RMSProp with Nesterov momentum

Loss functions for deep learning

Mean Squared Error

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

Mean Absolute Error

$$MSE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

Cross Entropy (Negative Log Likelihood)

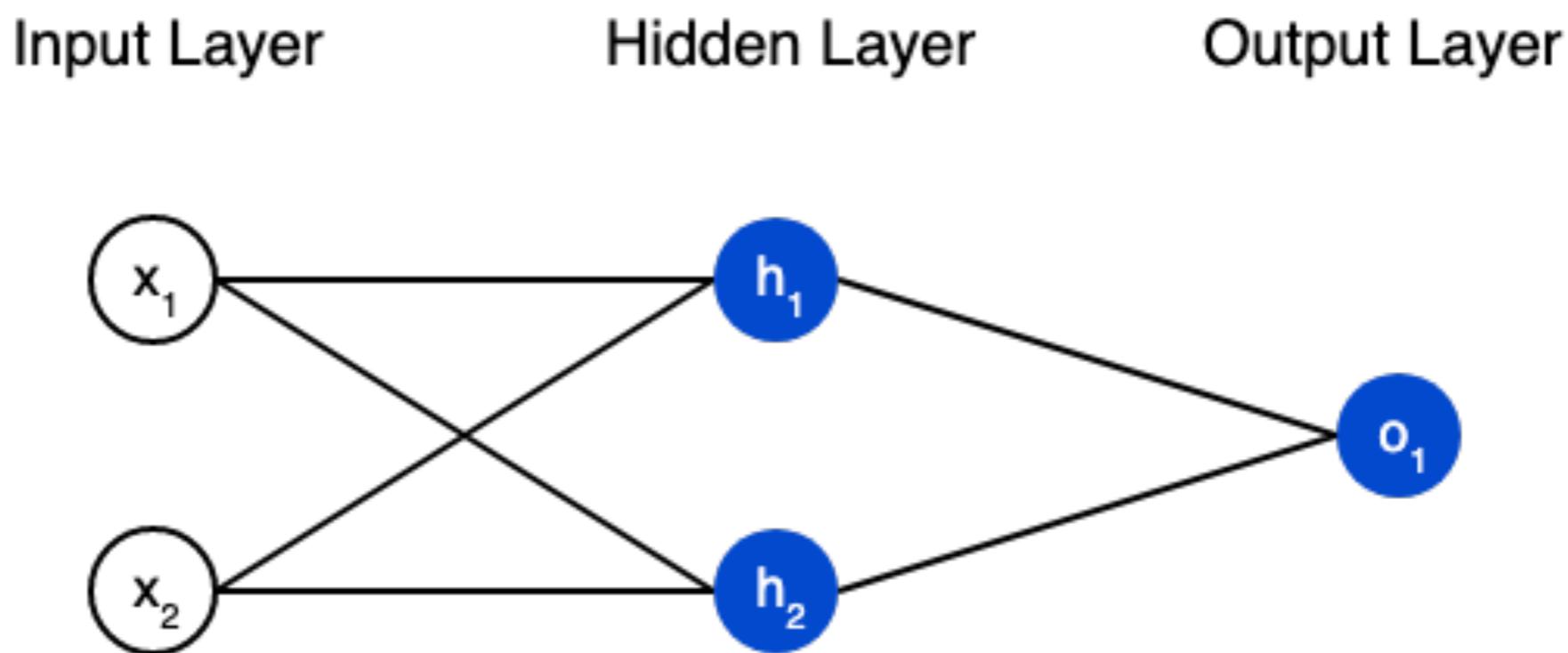
Categorical Cross Entropy
$$CCE = -\frac{\sum_{i=1}^n \sum_{j=1}^c y_{i,j} \log(\hat{y}_{i,j})}{n}$$

Binary Cross Entropy

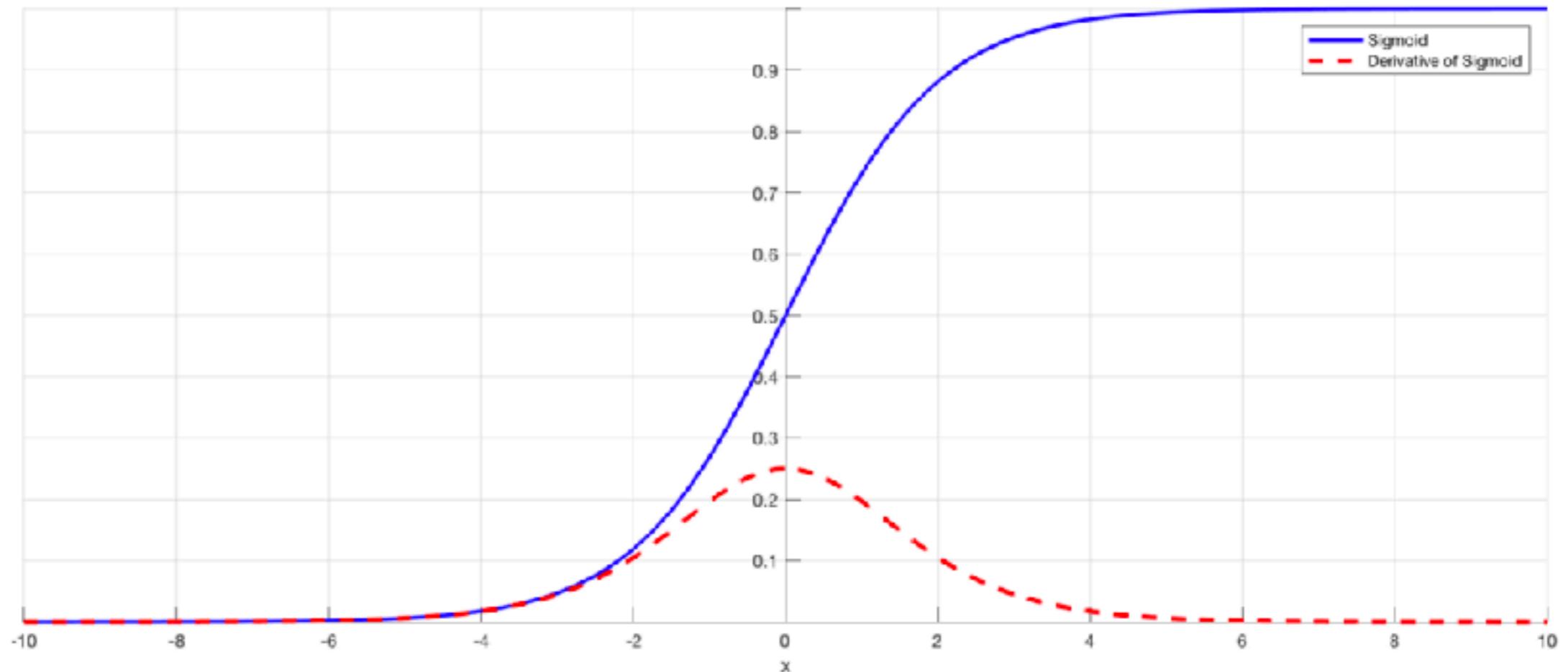
$$BCE = -\frac{\sum_{i=1}^n [y_{i,j} \log(\hat{y}_{i,j}) + (1 - y_{i,j}) \log(1 - \hat{y}_{i,j})]}{n}$$

Weight initialization

Zero or constant initialization



Too low or too high initialization



Xavier and He initializers

1. The mean of the activations should be zero
2. The variance of the activations should stay the same across every layer

**Xavier (Glorot) initialization
for tanh**

$$\mathbf{W}^l \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{1}{n^{l-1}})$$

$$b^l = 0$$

**He (Kaiming) initialization
for relu**

$$\mathbf{W}^l \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{2}{n^{l-1}})$$

$$b^l = 0$$

Experiment with various initializations for a deep network

[04-Regression-nn-assignment.ipynb](#)

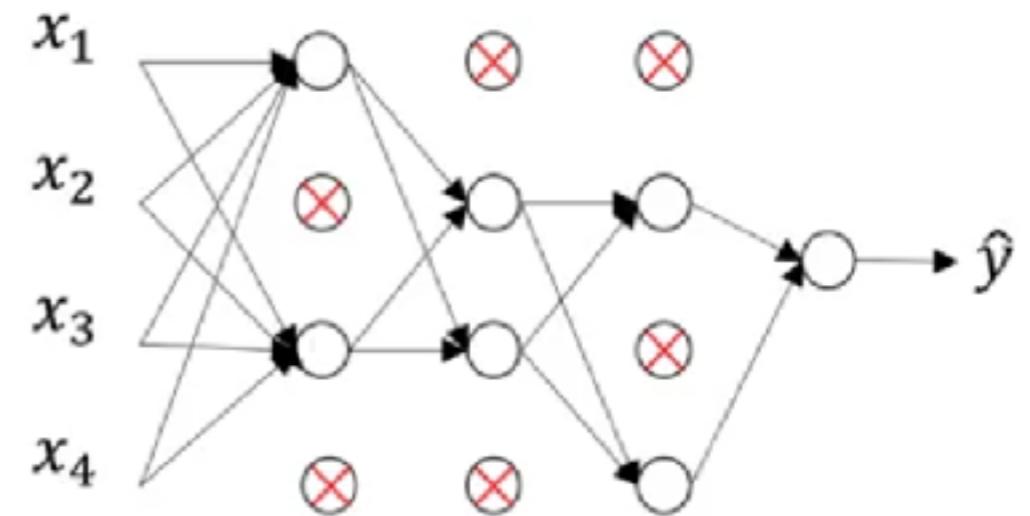
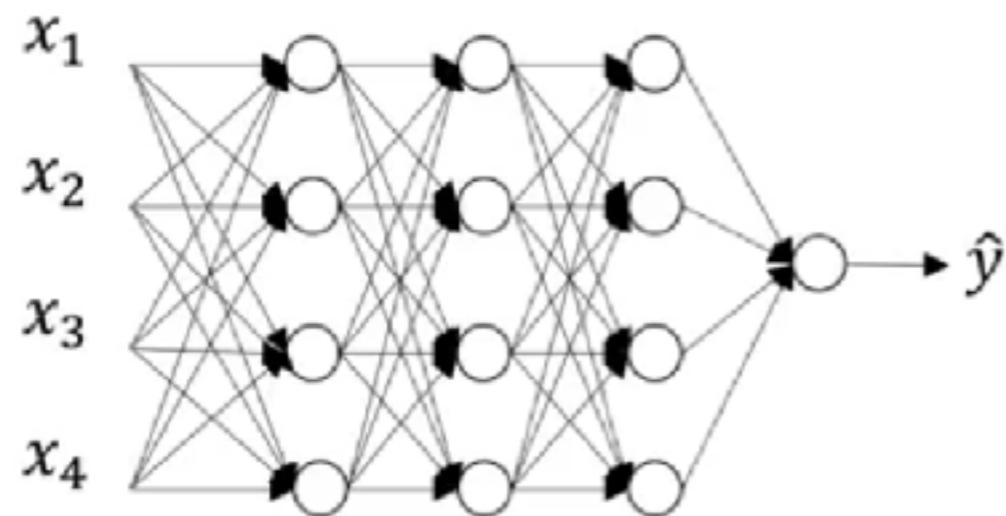
L2 Regularization in deep learning

$$cost(w^1, b^1, \dots, w^L, b^L) = \frac{1}{n} \sum_{i=1}^n Loss(y_i, \hat{y}_i) + \frac{\lambda}{2n} \sum_{l=1}^L \|w^l\|_F^2$$

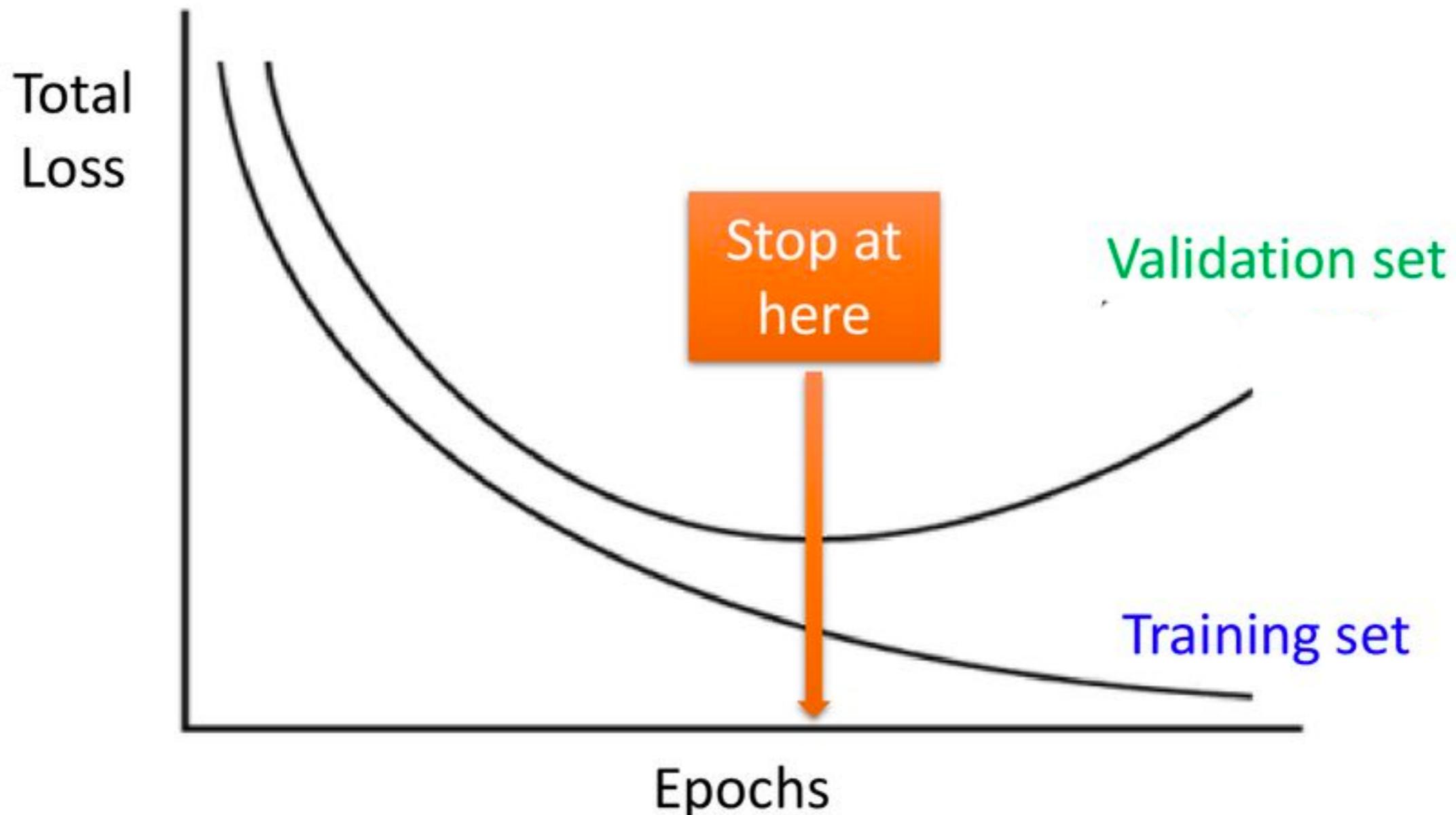
Frobenius norm

$$\|w\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^m |w_{i,j}|^2}$$

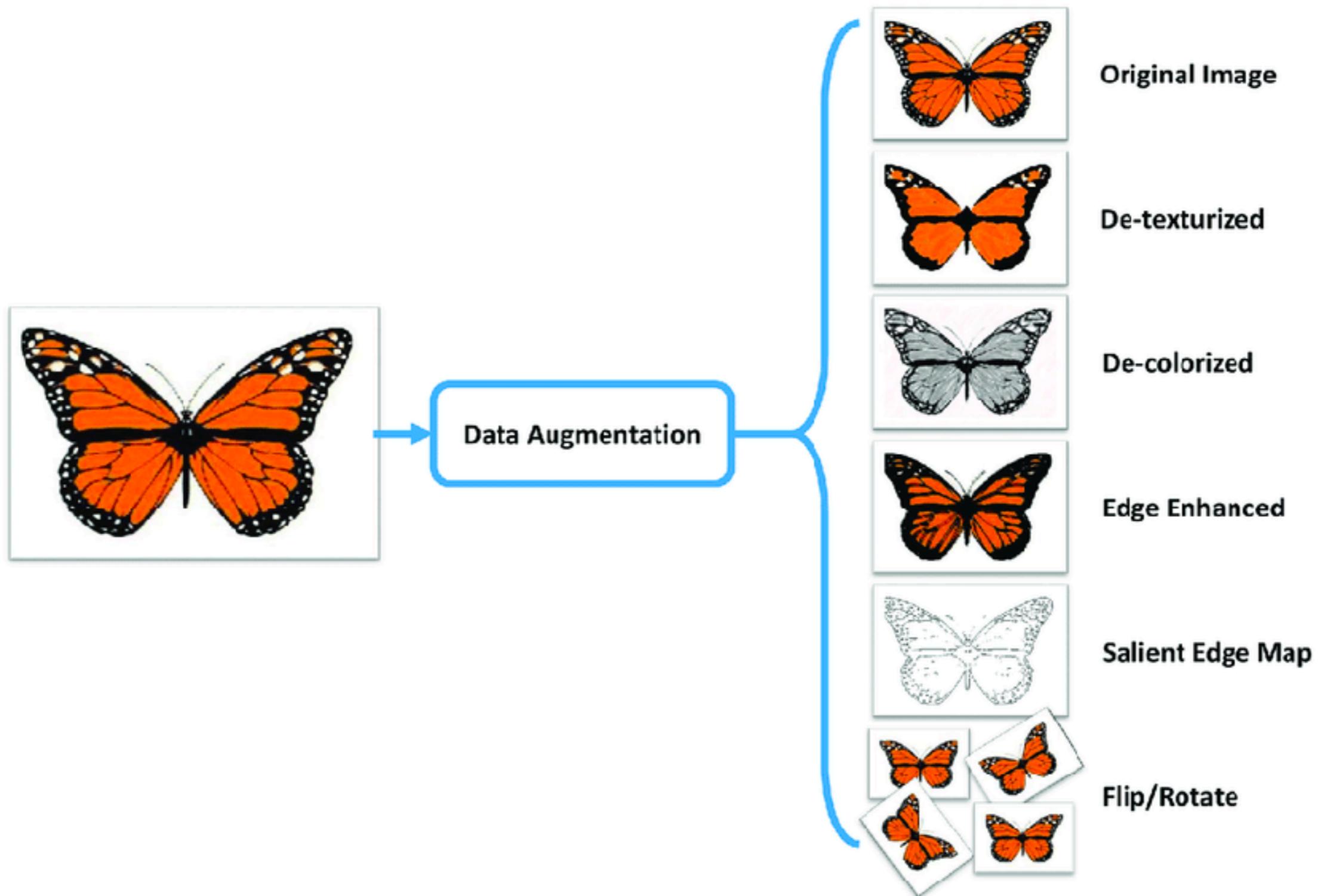
Dropout



Early stopping



Data augmentation



Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

Output: $\{y_i \equiv \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

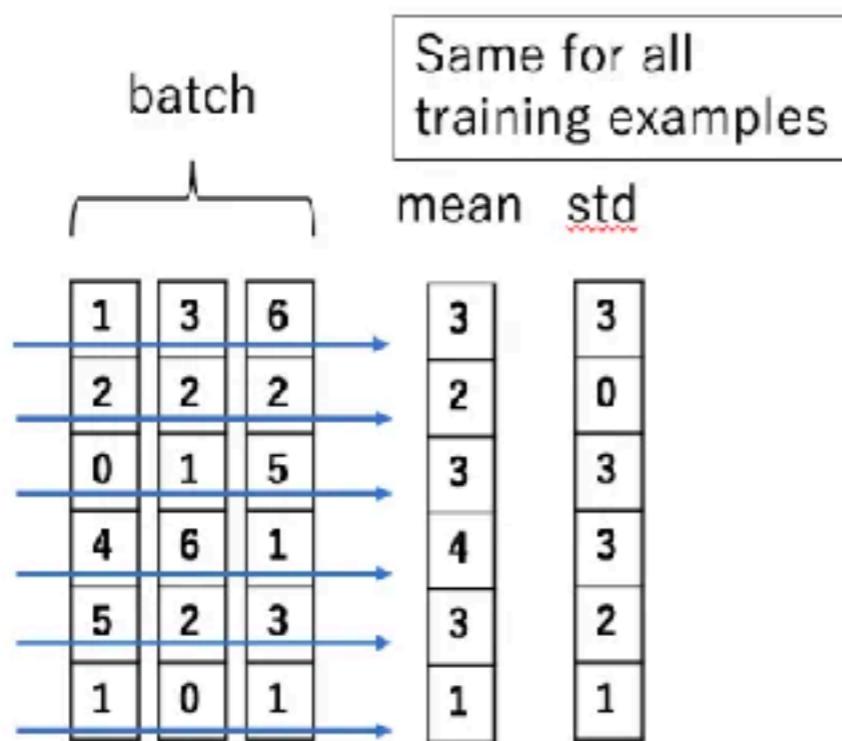
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

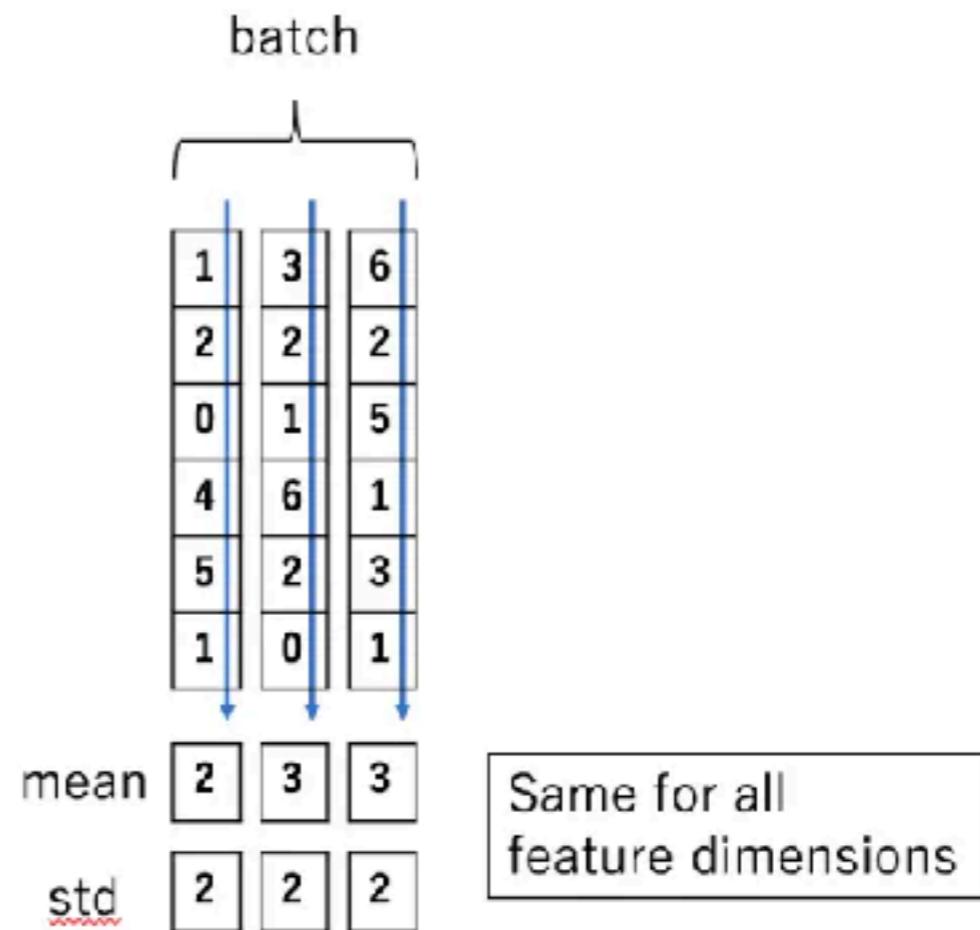
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Layer normalization

Batch Normalization



Layer Normalization



Functional API in Keras

```
1 # Sequential model
2 from keras.models import Sequential
3 from keras.layers import Dense
4
5 model = Sequential()
6 model.add(10, input_shape=(10,), activation='relu')
7 model.add(Dense(20, activation='relu'))
8 model.add(Dense(10, activation='relu'))
9 model.add(Dense(1, activation='sigmoid'))
```

```
1 # Functional model
2 from keras.models import Model
3 from keras.layers import Input, Dense
4
5 visible = Input(shape=(10,))
6 hidden1 = Dense(10, activation='relu')(visible)
7 hidden2 = Dense(20, activation='relu')(hidden1)
8 hidden3 = Dense(10, activation='relu')(hidden2)
9 output = Dense(1, activation='sigmoid')(hidden3)
10 model = Model(inputs=visible, outputs=output)
```

Shared Input

```
1 # Shared Input Layer
2 from keras.utils import plot_model
3 from keras.models import Model
4 from keras.layers import Input, Dense, Flatten
5 from keras.layers.convolutional import Conv2D
6 from keras.layers.pooling import MaxPooling2D
7 from keras.layers.merge import concatenate
8 # input layer
9 visible = Input(shape=(64,64,1))
10 # first feature extractor
11 conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
12 pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
13 flat1 = Flatten()(pool1)
14 # second feature extractor
15 conv2 = Conv2D(16, kernel_size=8, activation='relu')(visible)
16 pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
17 flat2 = Flatten()(pool2)
18 # merge feature extractors
19 merge = concatenate([flat1, flat2])
20 # interpretation layer
21 hidden1 = Dense(10, activation='relu')(merge)
22 # prediction output
23 output = Dense(1, activation='sigmoid')(hidden1)
24 model = Model(inputs=visible, outputs=output)
```

Multiple inputs (outputs)

```
1 # Multiple Inputs
2 from keras.utils import plot_model
3 from keras.models import Model
4 from keras.layers import Input
5 from keras.layers import Dense
6 from keras.layers import Flatten
7 from keras.layers.convolutional import Conv2D
8 from keras.layers.pooling import MaxPooling2D
9 from keras.layers.merge import concatenate
10 # first input model
11 visible1 = Input(shape=(64,64,1))
12 conv11 = Conv2D(32, kernel_size=4, activation='relu')(visible1)
13 pool11 = MaxPooling2D(pool_size=(2, 2))(conv11)
14 conv12 = Conv2D(16, kernel_size=4, activation='relu')(pool11)
15 pool12 = MaxPooling2D(pool_size=(2, 2))(conv12)
16 flat1 = Flatten()(pool12)
17 # second input model
18 visible2 = Input(shape=(32,32,3))
19 conv21 = Conv2D(32, kernel_size=4, activation='relu')(visible2)
20 pool21 = MaxPooling2D(pool_size=(2, 2))(conv21)
21 conv22 = Conv2D(16, kernel_size=4, activation='relu')(pool21)
22 pool22 = MaxPooling2D(pool_size=(2, 2))(conv22)
23 flat2 = Flatten()(pool22)
24 # merge input models
25 merge = concatenate([flat1, flat2])
26 # interpretation model
27 hidden1 = Dense(10, activation='relu')(merge)
28 hidden2 = Dense(10, activation='relu')(hidden1)
29 output = Dense(1, activation='sigmoid')(hidden2)
30 model = Model(inputs=[visible1, visible2], outputs=output)
```

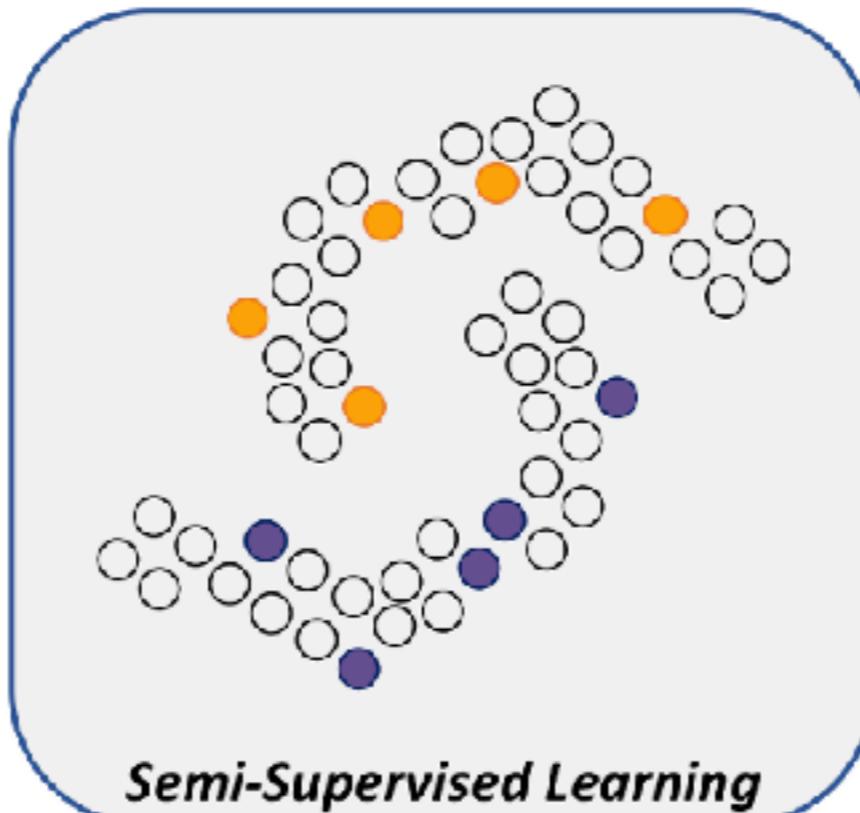
Practical example on regularization and normalization

[**05-Normalization-and-regularization-assignment.ipynb**](#)

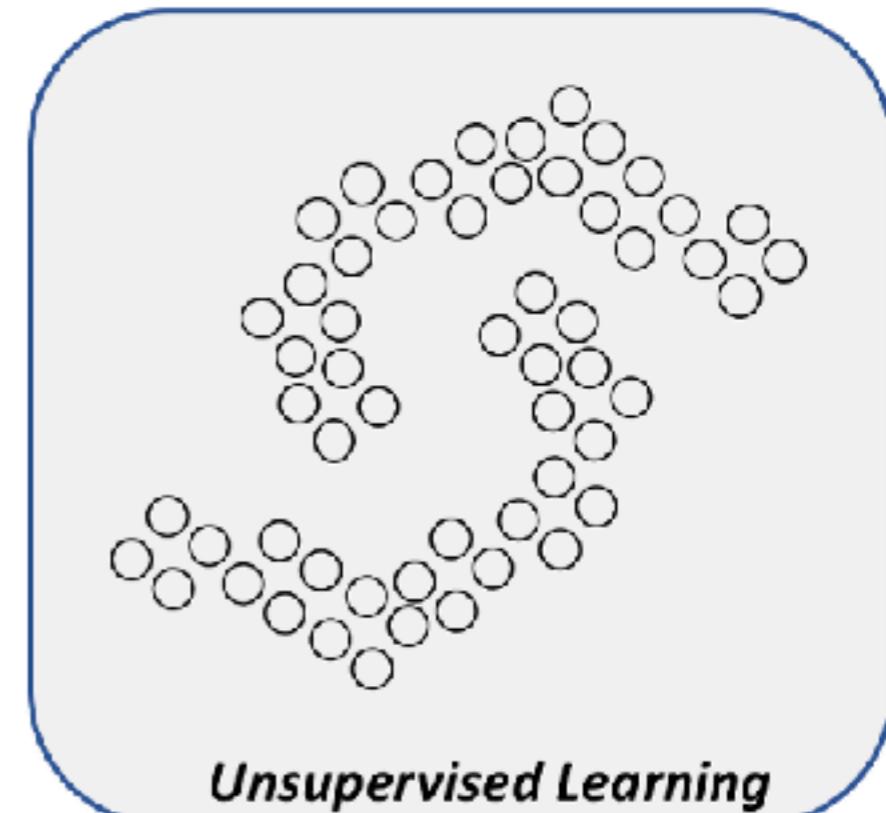
Semi-supervised learning



Supervised Learning

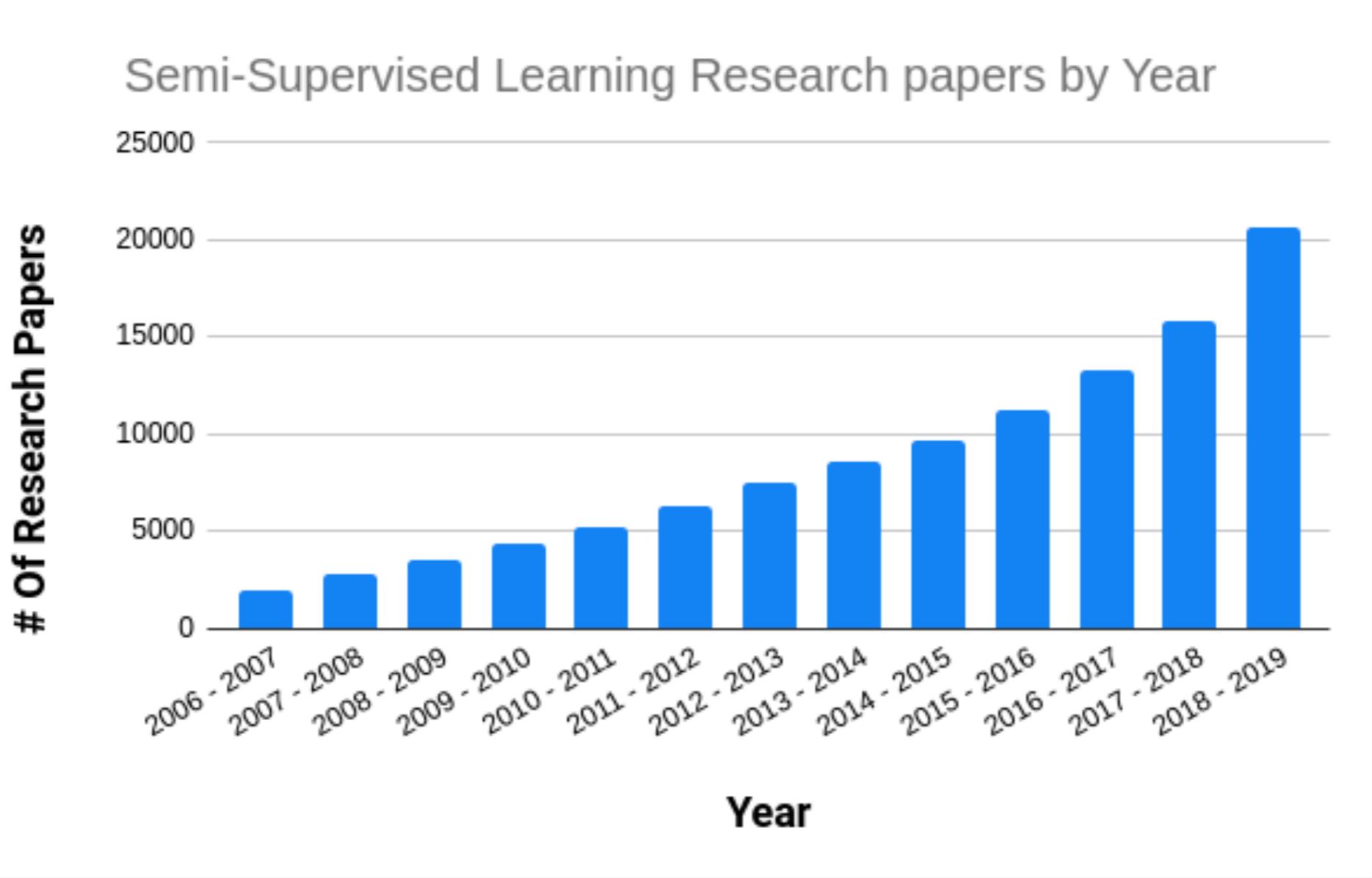


Semi-Supervised Learning

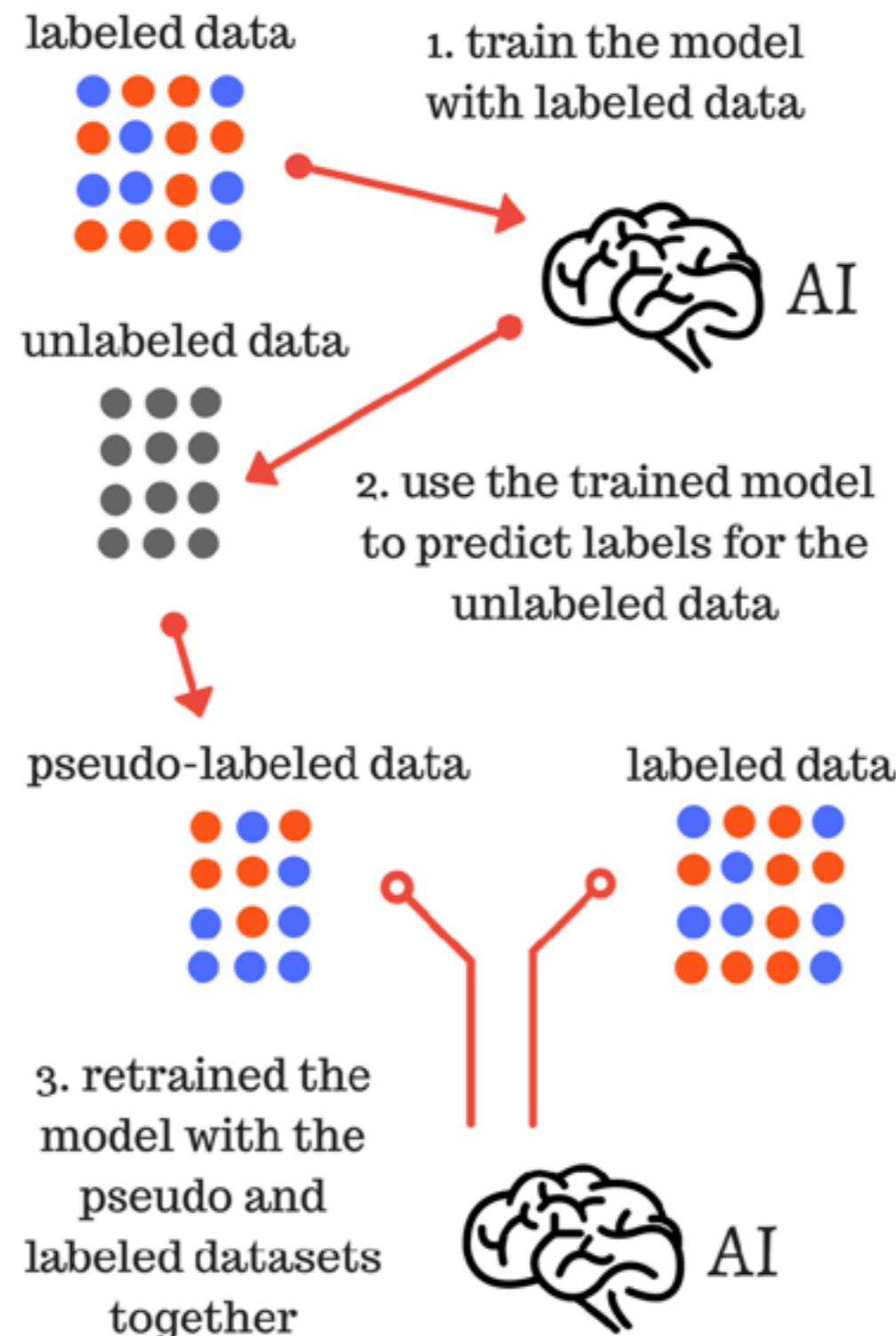


Unsupervised Learning

Small Data is the new Big Data

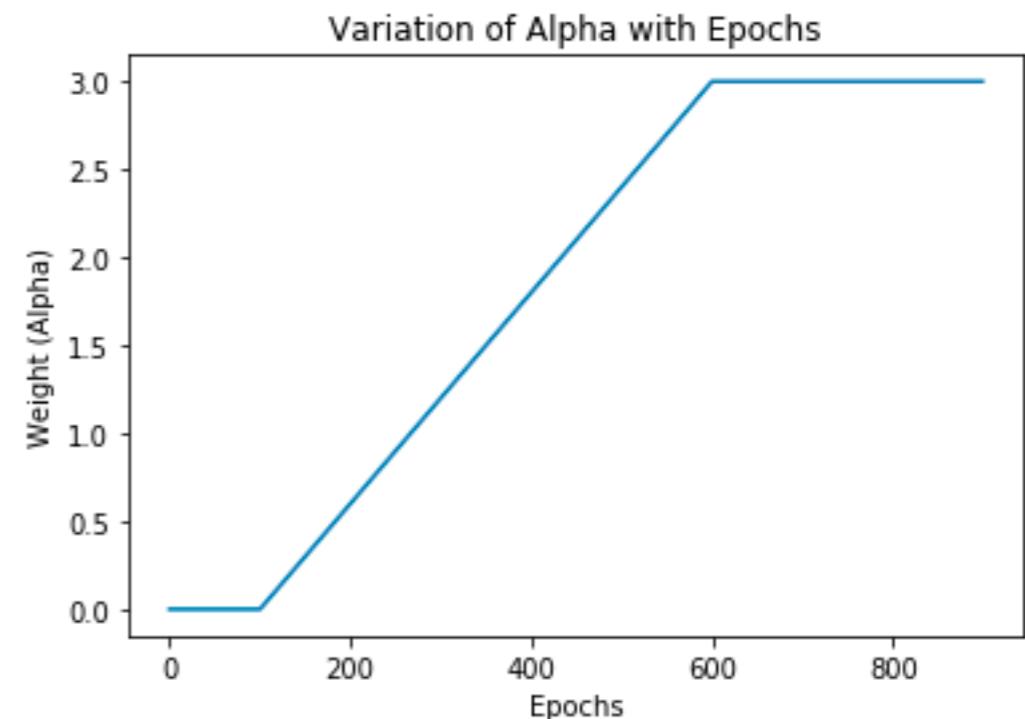


Pseudo-labeling

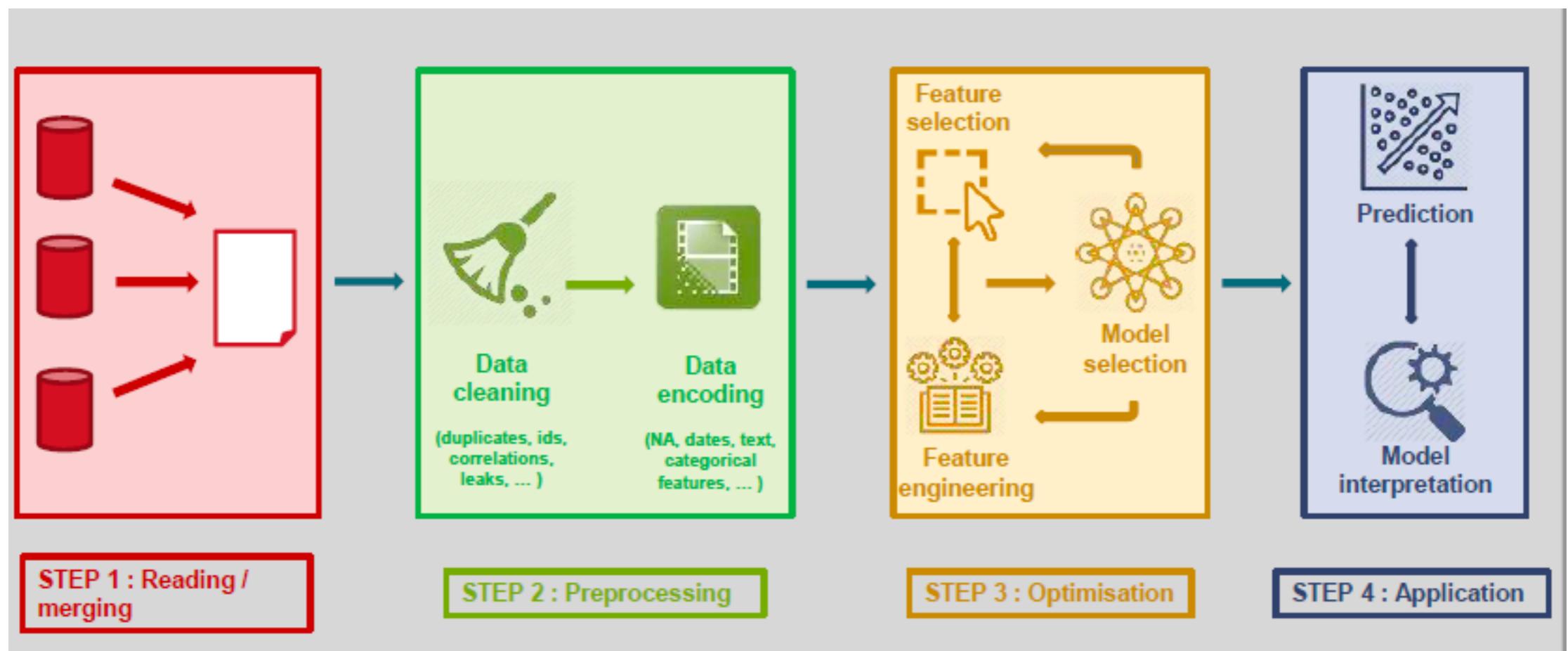


Pseudo-labeling with neural networks

- ★ Train model on a batch of labeled data
- ★ Use the trained model to predict labels on a batch of unlabeled data
- ★ Use the predicted labels to calculate the loss on unlabeled data
- ★ Combine labeled loss with α -weighted unlabeled loss and backpropagate
- ★ Repeat



AutoML approaches



AutoML frameworks



AutoGluon

Google's AutoML



HYPEROPT



MLBox,
Machine Learning Box

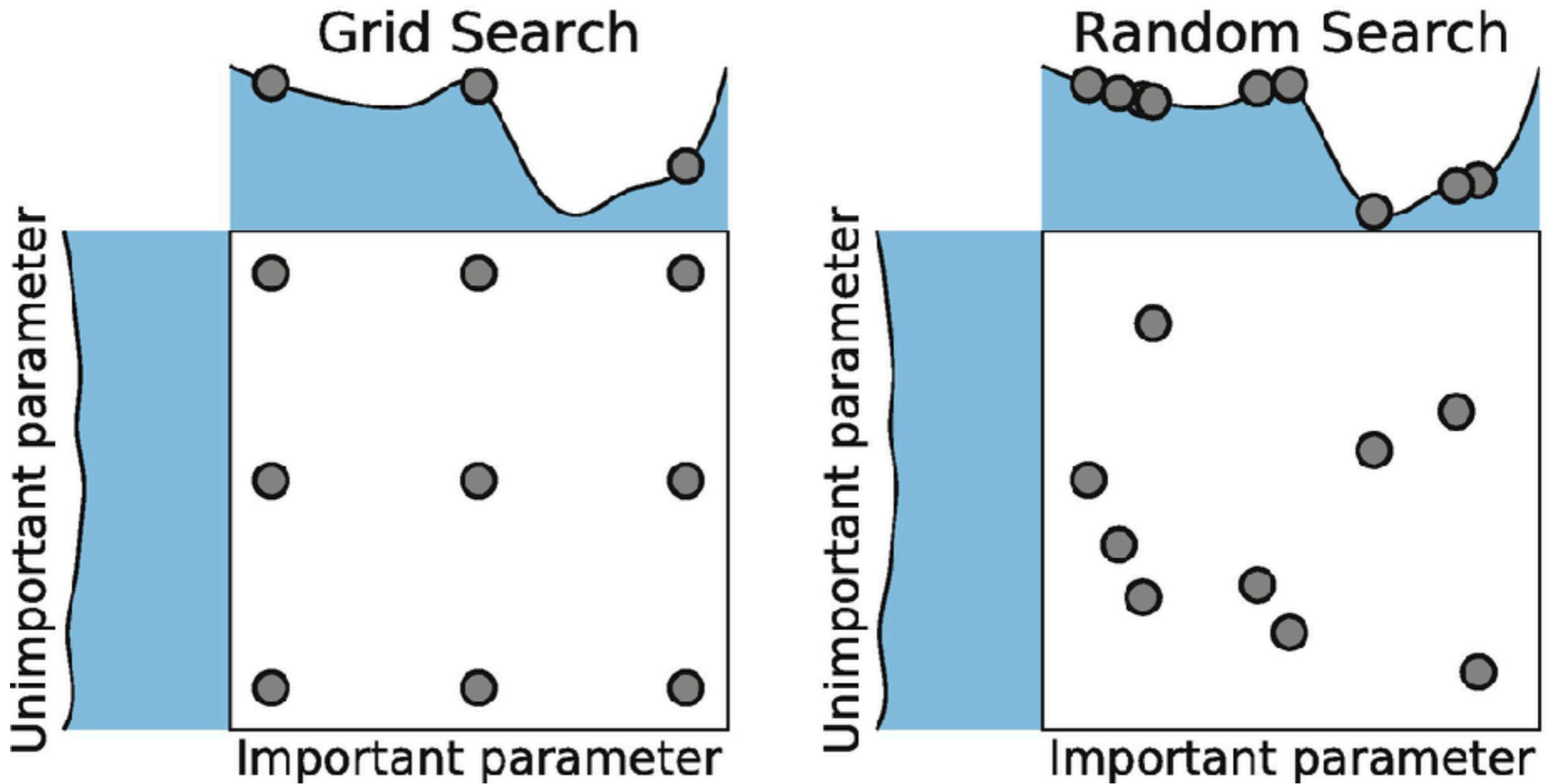


Hyper-parameters tuning

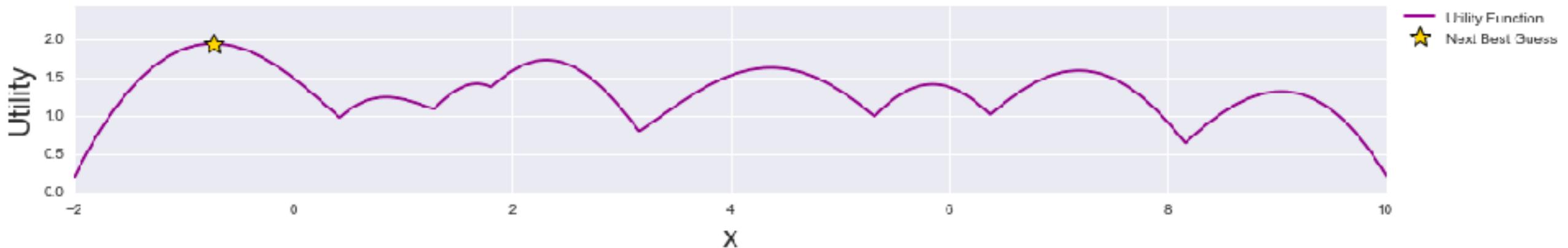
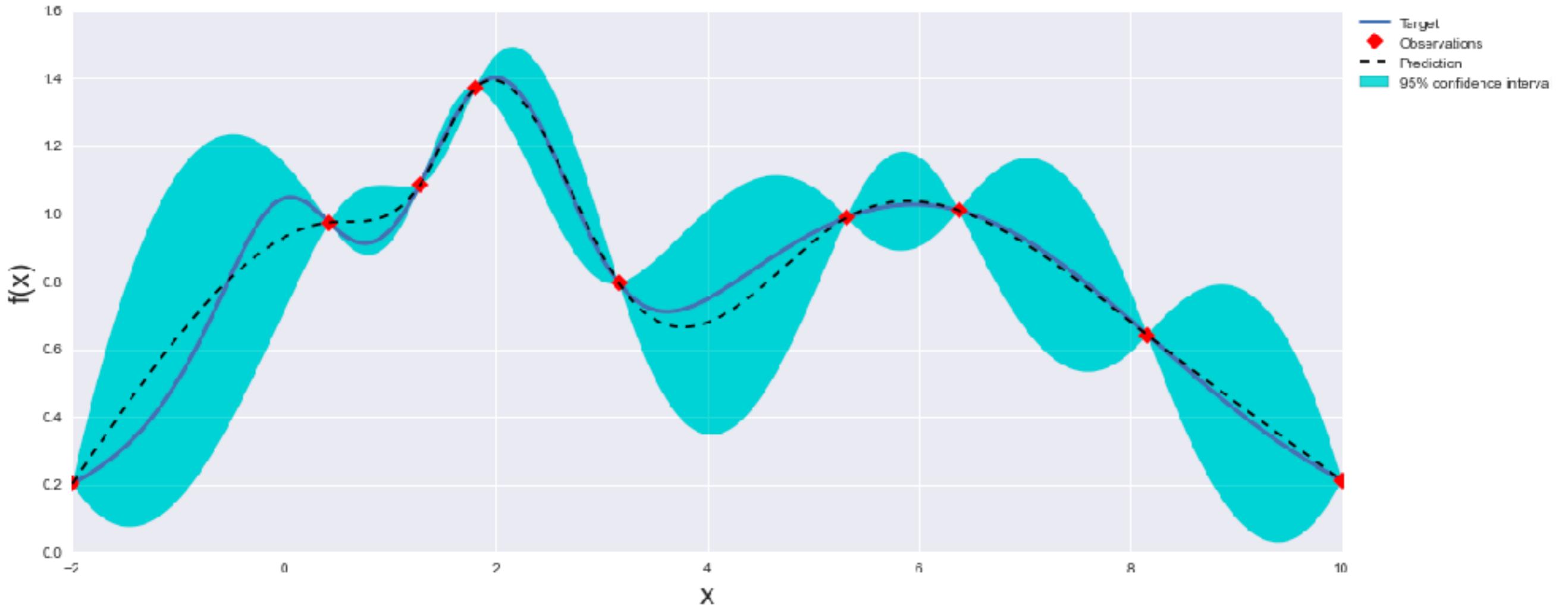
Parameters - learnable weights of the neural network

Hyper-parameters - parameters set before training (learning rate, momentum, dropout rate, etc.)

Grid search vs random search



Bayesian optimization using Gaussian processes



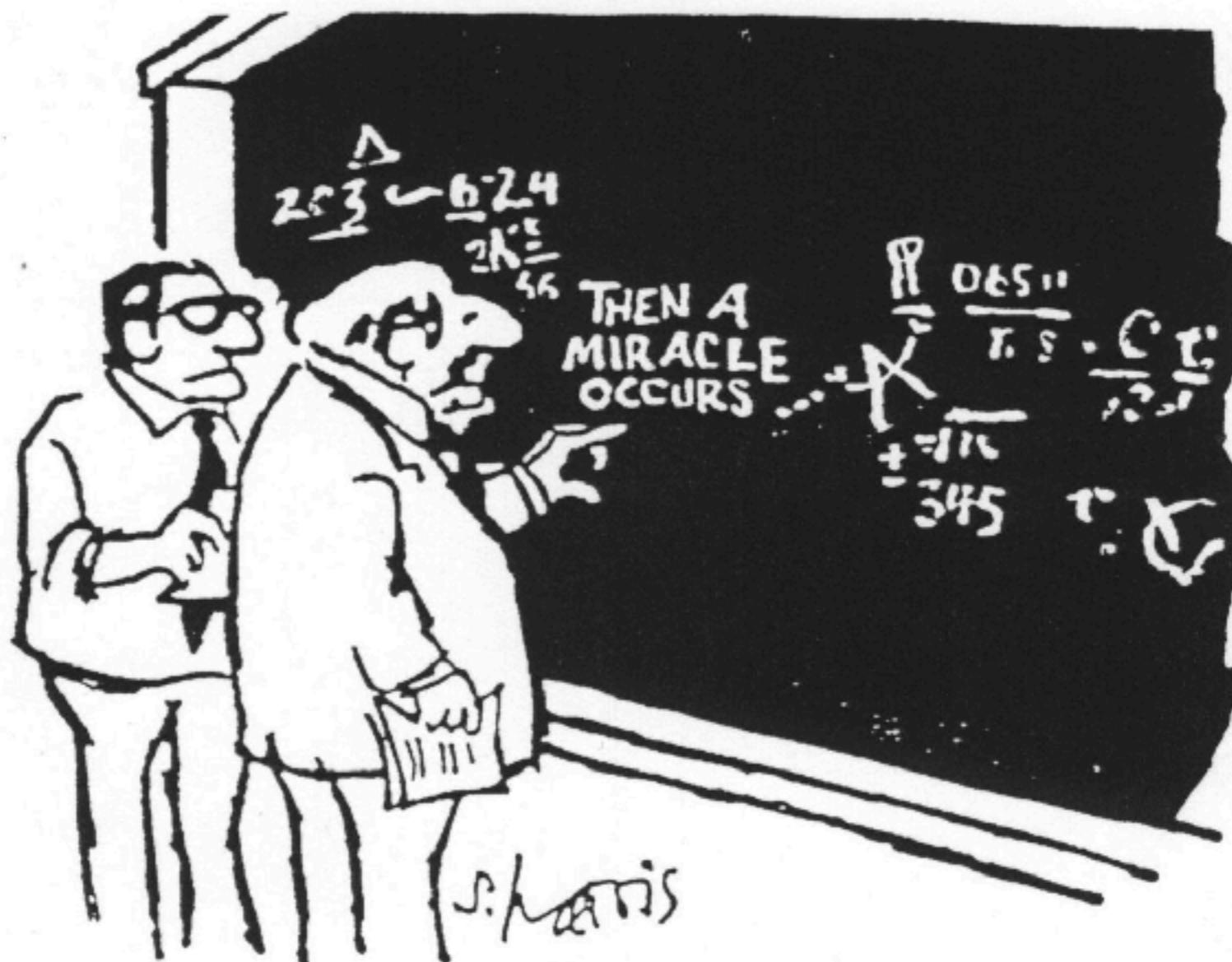
Neural Architecture Search (NAS)



AutoML with AutoKeras

06-Autokeras.ipynb

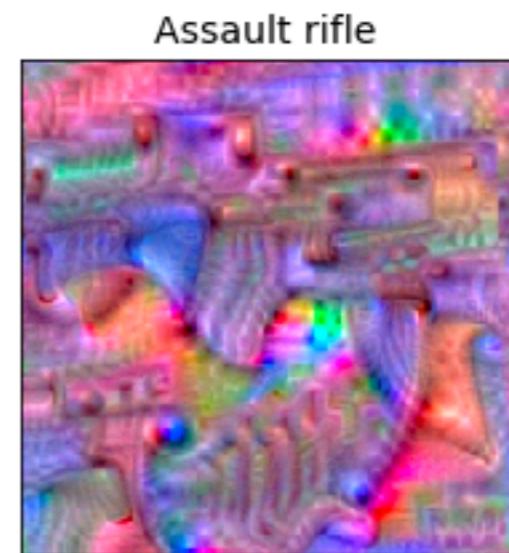
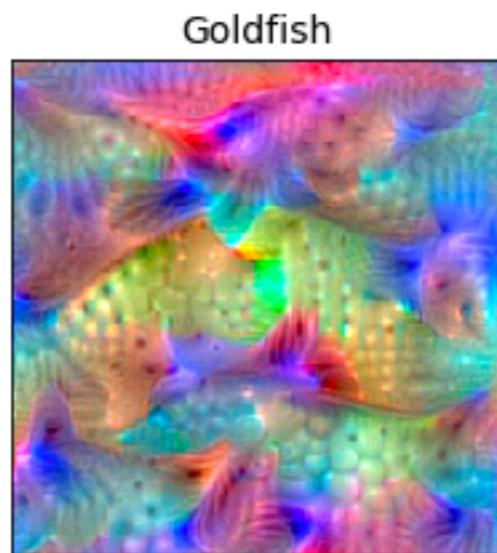
Neural network explainability



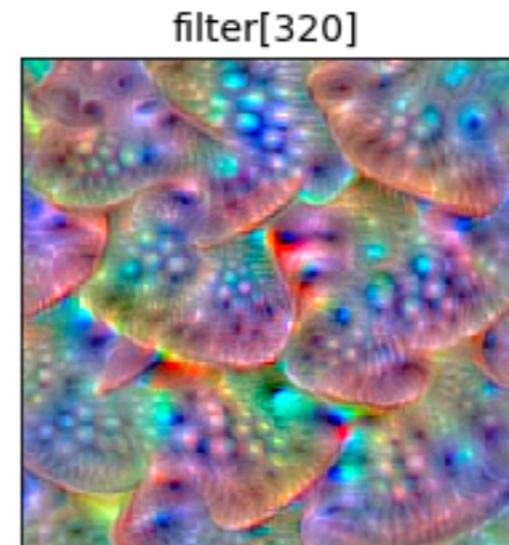
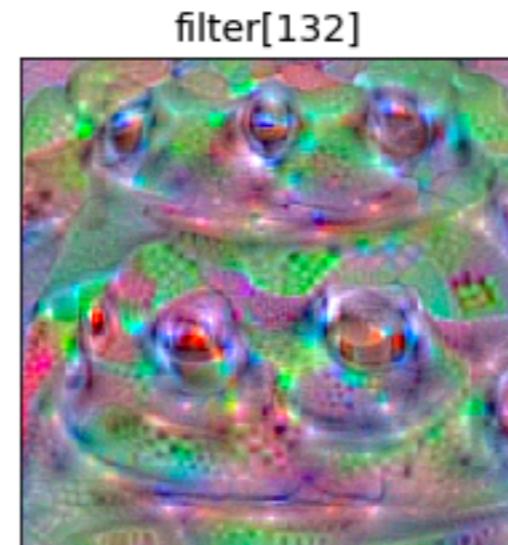
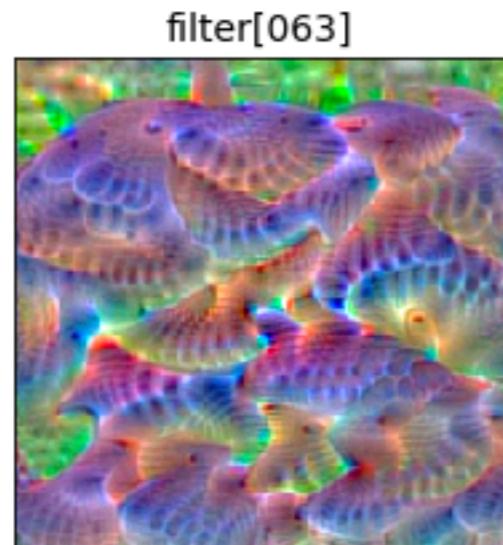
I think you should be a little
more specific, here in Step 2

Activation Maximization

Visualized output classification Layer



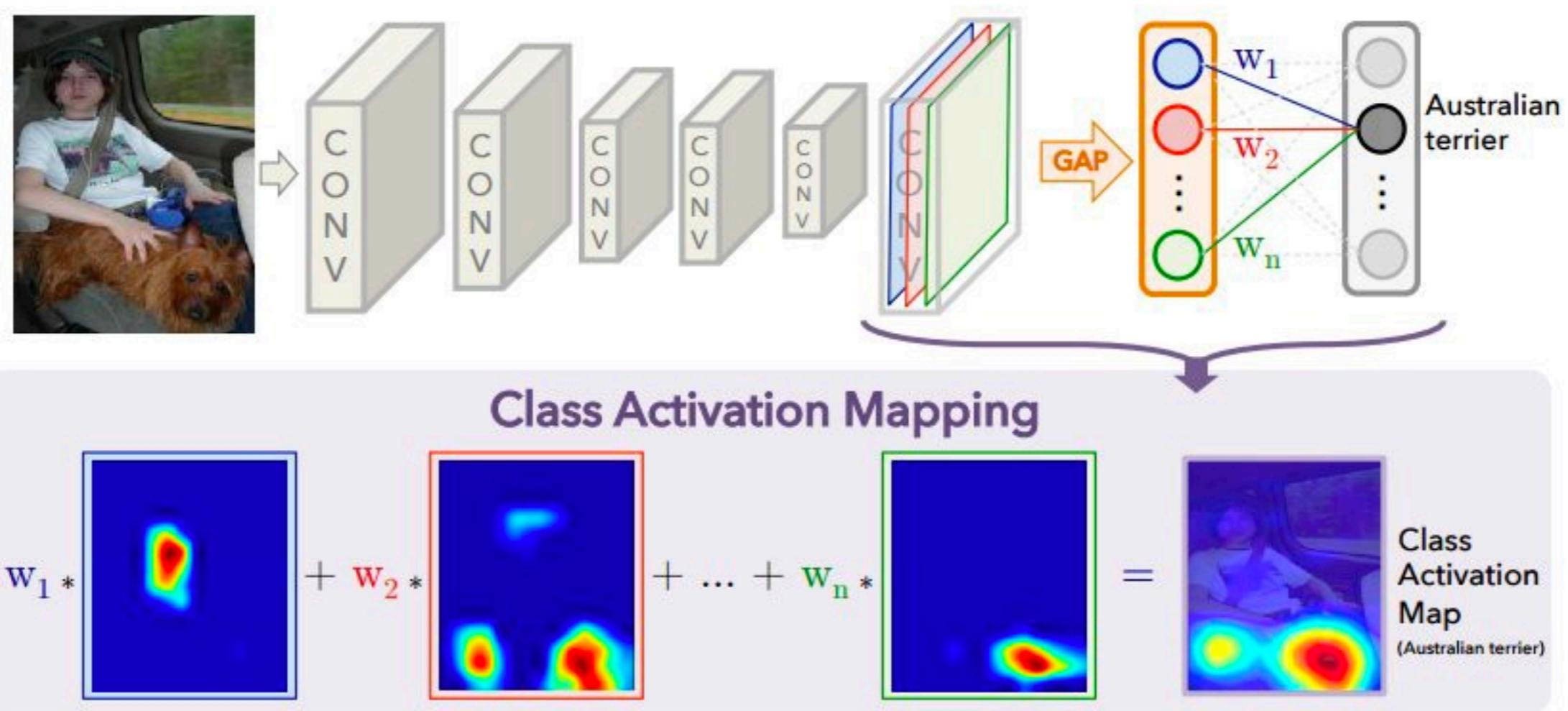
Visualized hidden convolutional layers



Grad-CAM heat maps



CAM heat maps



Convolutional Neural Networks Explainability in Keras

07-Explainability.ipynb

Thank you for your attention

e-mail: jiri@mlguru.com

Web: www.mlguru.com

Twitter: @JiriMaterna

Facebook: <https://www.facebook.com/maternajiri>

LinkedIn: <https://www.linkedin.com/in/jirimaterna/>