Artificial Neural Networks
Part III

Beatrice Lazzerini

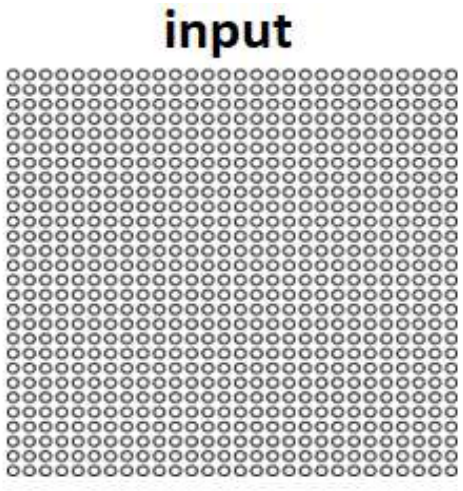
Department of Information Engineering
University of Pisa
ITALY

Convolutional Neural Network (CNN)

CNNs combine three architectural ideas:

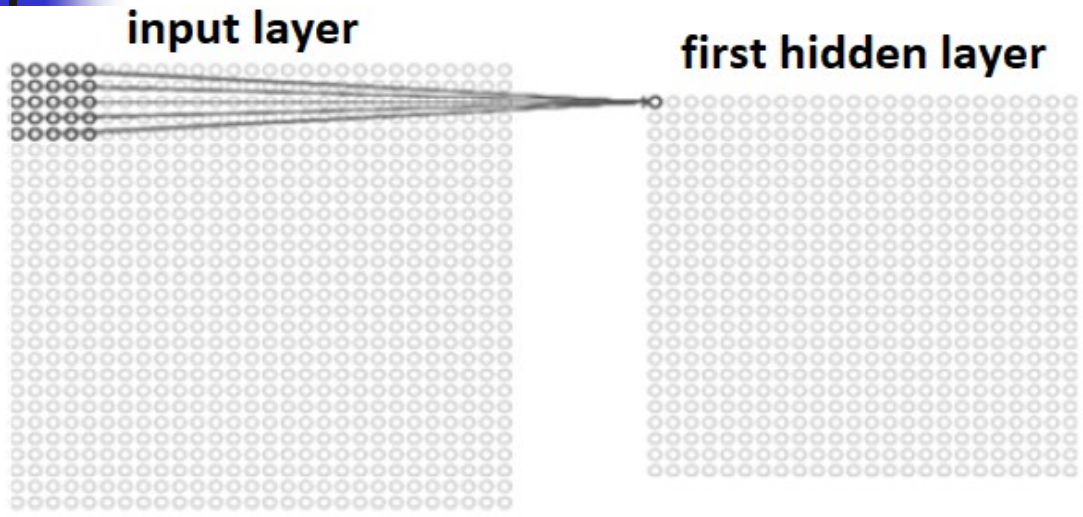
- local receptive fields
- shared weights and biases
- pooling

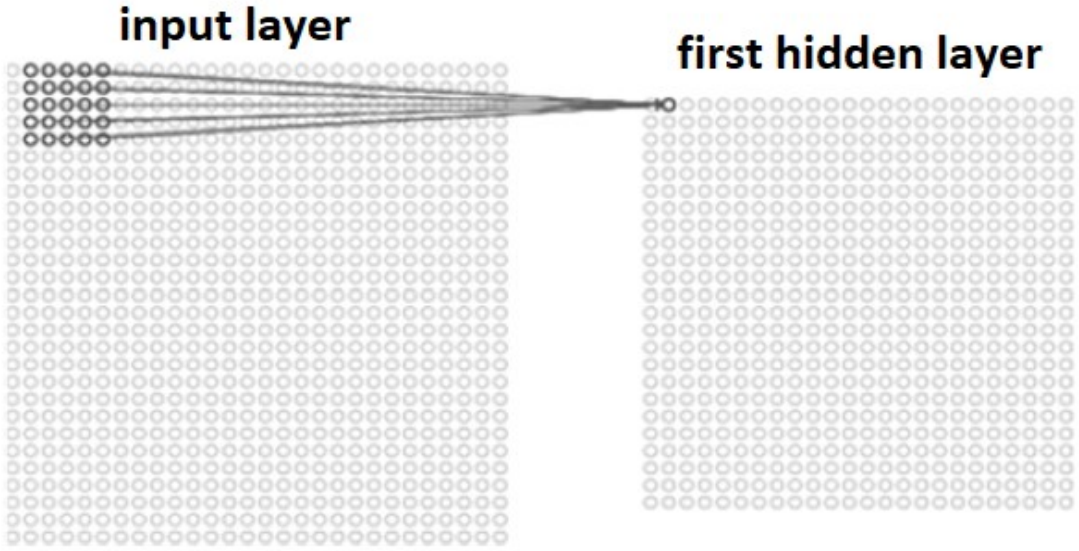
Example



- Input: square of neurons whose values are pixel intensities.

Local receptive fields

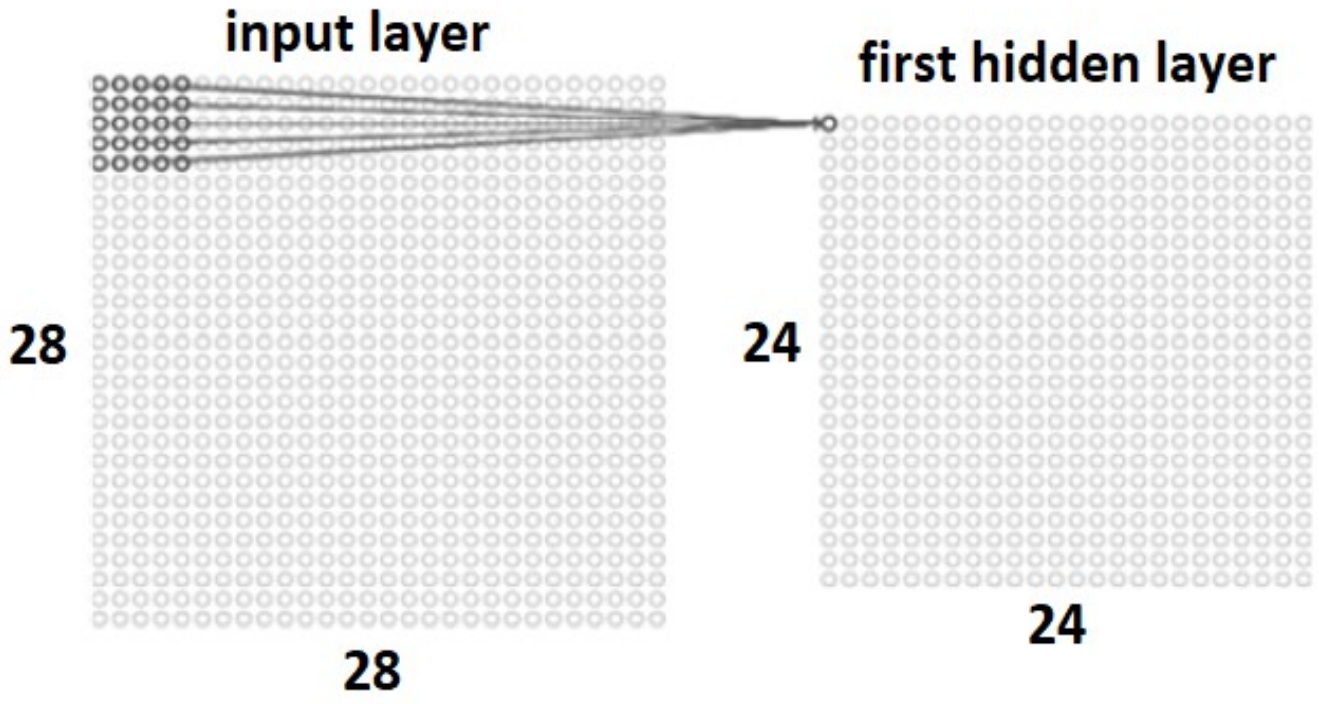




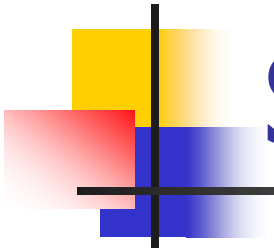
- Each neuron of the first hidden layer is connected to a small region, called *local receptive field* (e.g., 5x5) of the input space

  ➔ strong reduction in the number of connections.
- For each hidden neuron: 5x5 weights + 1 bias = 26 parameters.

- We slide the local receptive field (e.g., by one pixel) across the input image.



24x24 hidden neurons



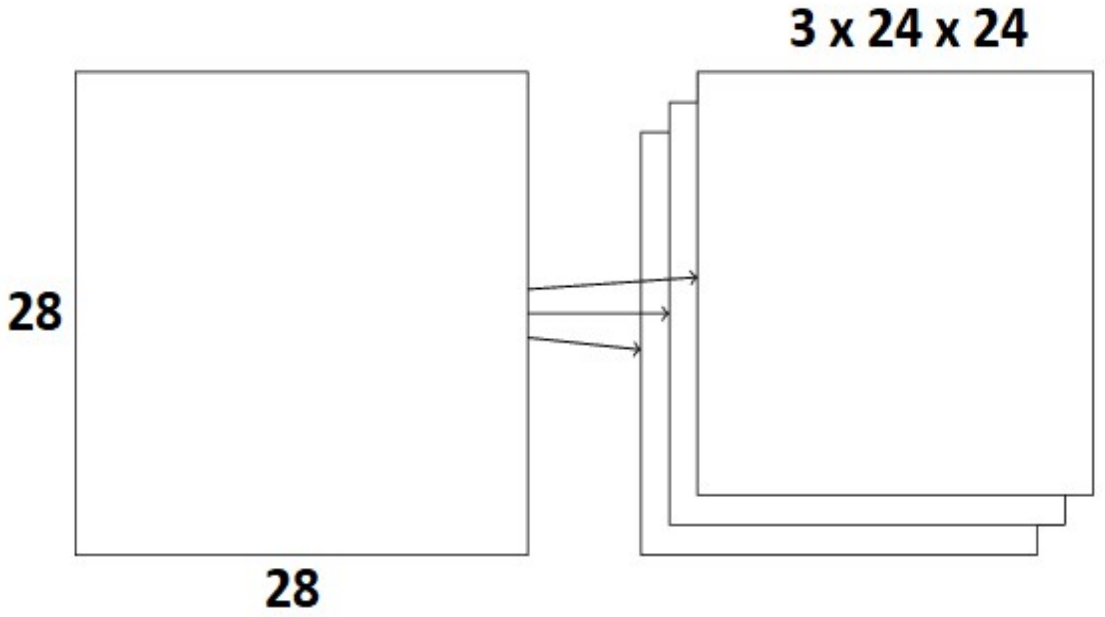
Shared weights and biases

- We use the *same weights* and the *same bias* for all hidden neurons. This means that all neurons in the first hidden layer detect the same feature, but in different positions of the input image

  ➔ significant reduction in the number of weights.

- Shared weights and bias define a *filter* (or *kernel*).
- The filter convolves the input image and forms a *feature map*.
- The layer is called *convolutional layer*.

In general, multiple feature maps are considered (e.g., 3):

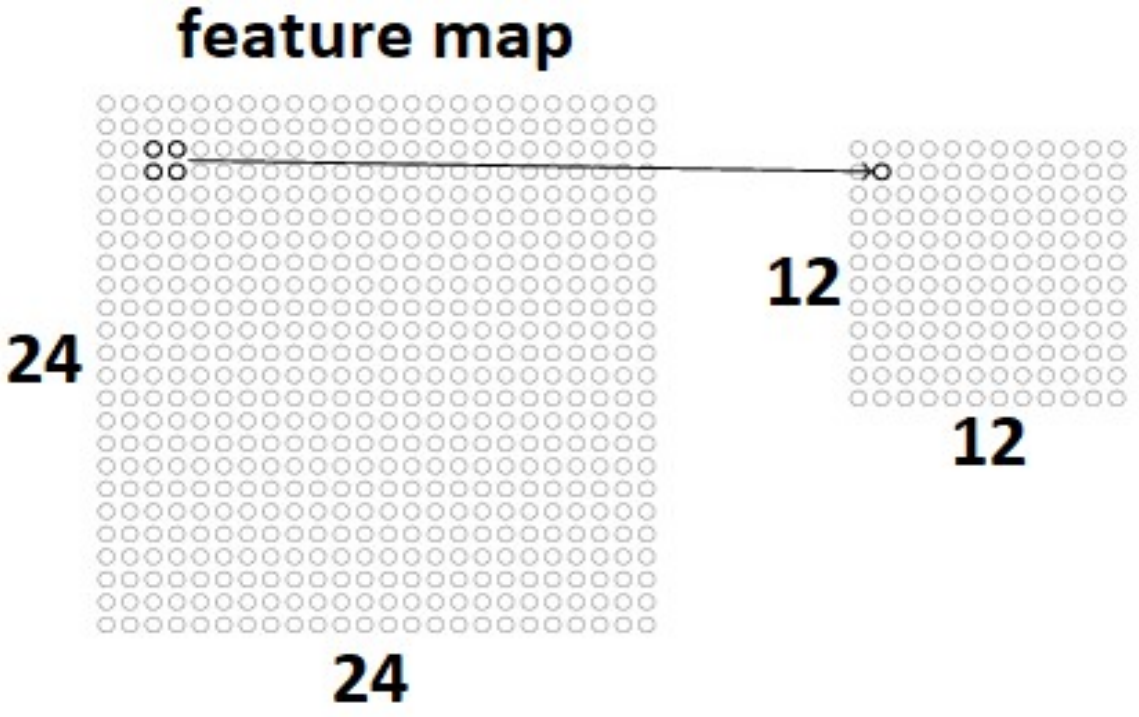


Each feature map is the result of a convolution that uses a different set of weights and a different bias. The number of feature maps equals the number of filters. In this case, the network can detect 3 different features, each of which can be detected in the whole image.

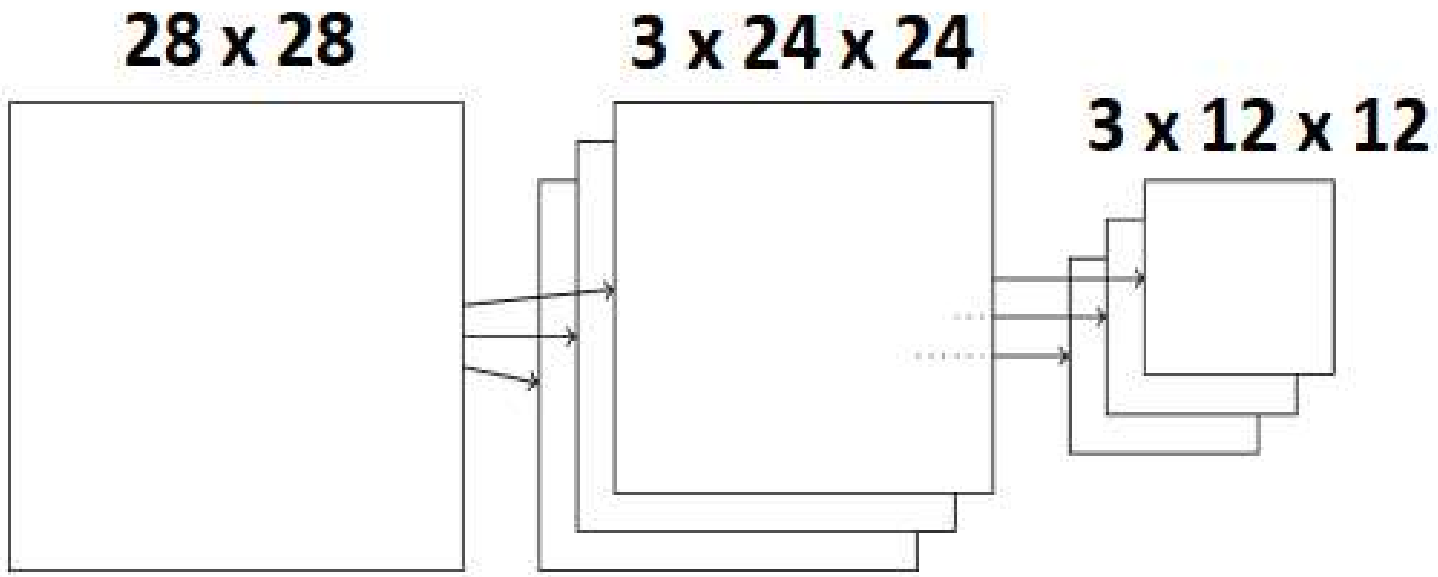
Pooling

A pooling layer is usually used immediately after a convolutional layer: for each feature map of the convolutional layer, the pooling layer produces a condensed feature map.

E.g., in max-pooling, a pooling unit provides the maximum activation in an input region (e.g., 2x2).

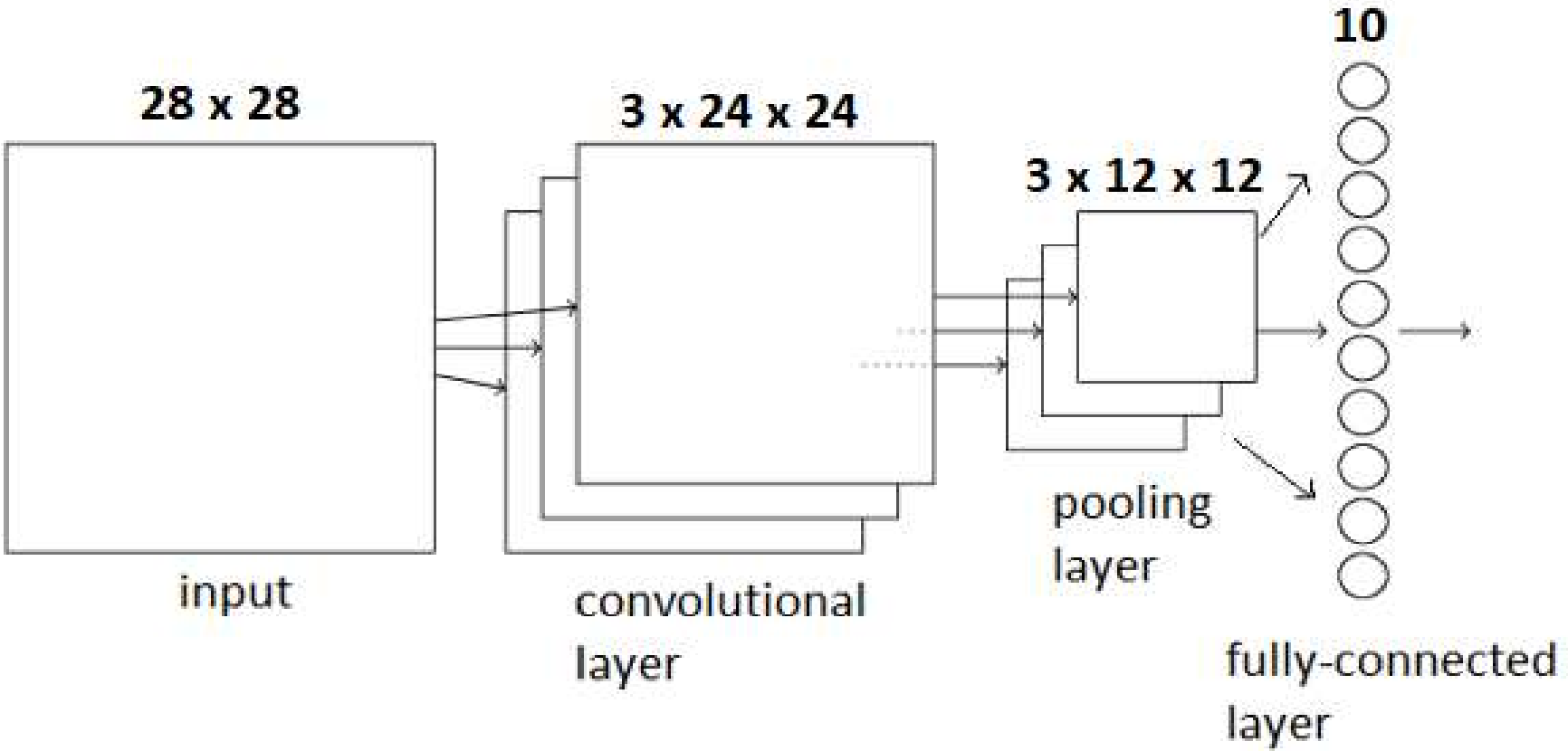


In the case of 3 feature maps, we apply max-pooling to each feature map separately:

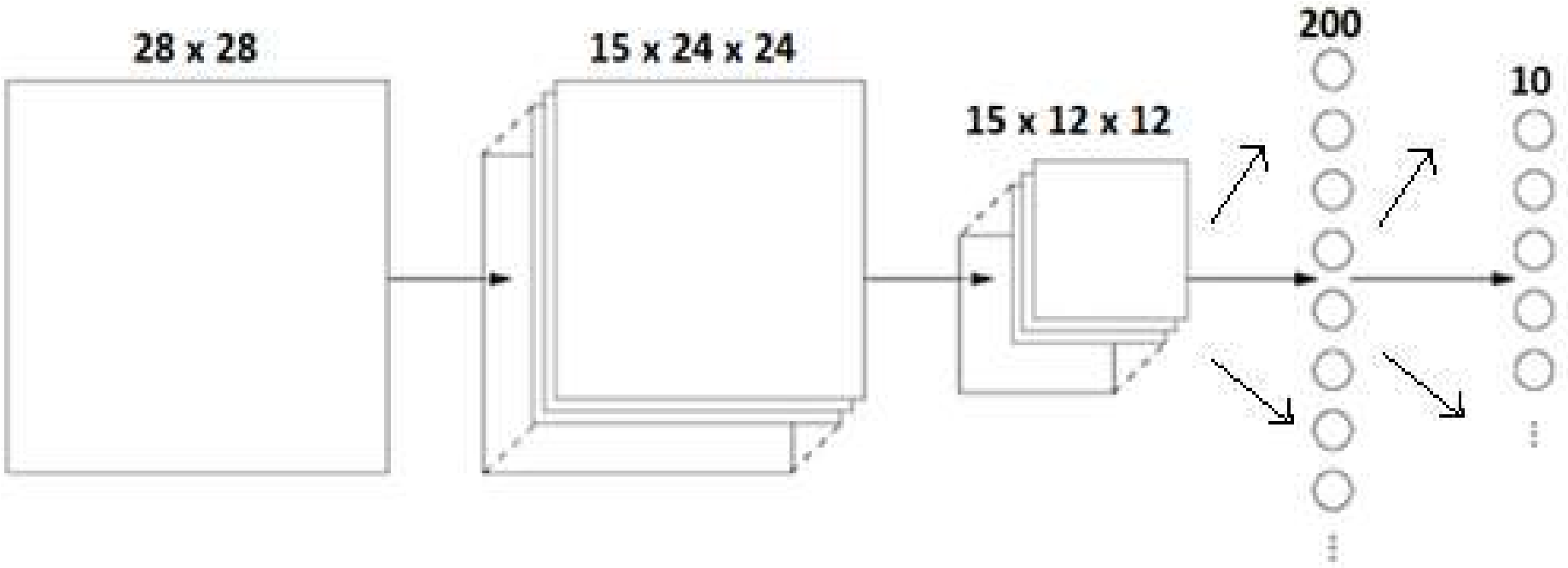


The complete CNN

Suppose we want to develop a 10-class classifier:

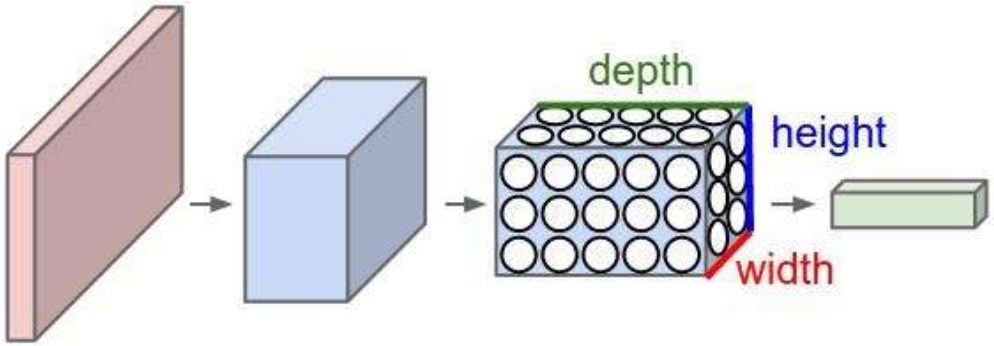


Another possibility:



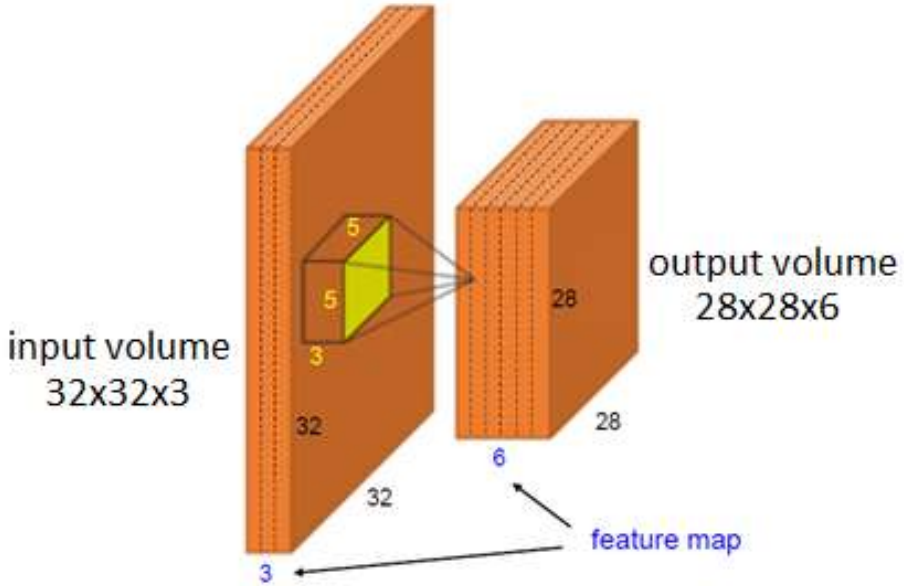
Architecture of a CNN

A CNN consists of a hierarchy of layers. The input layer is connected to the image pixels, the last layers are generally fully connected and implement an MLP classifier, the intermediate layers use local connections and shared weights.



width and *height* represent the spatial organization of the feature map, *depth* identifies the different feature maps.

For each pair of consecutive layers, there is an input volume and an output volume:



connections: (28x28x6) x (5x5x3) = 352,800

weights: 6 x (5x5x3 + 1) = 456

- **Activation function**

  The most used activation function is *Rectified Linear Unit* (*ReLU*): f(x)=max(0,x)

- **Stride**

  When a 3D filter slides over the input volume, instead of moving with a unitary step (i.e., one neuron at a time), a longer step (*stride*) can be used. This reduces the size of the feature maps, but there may be a slight penalty in accuracy.

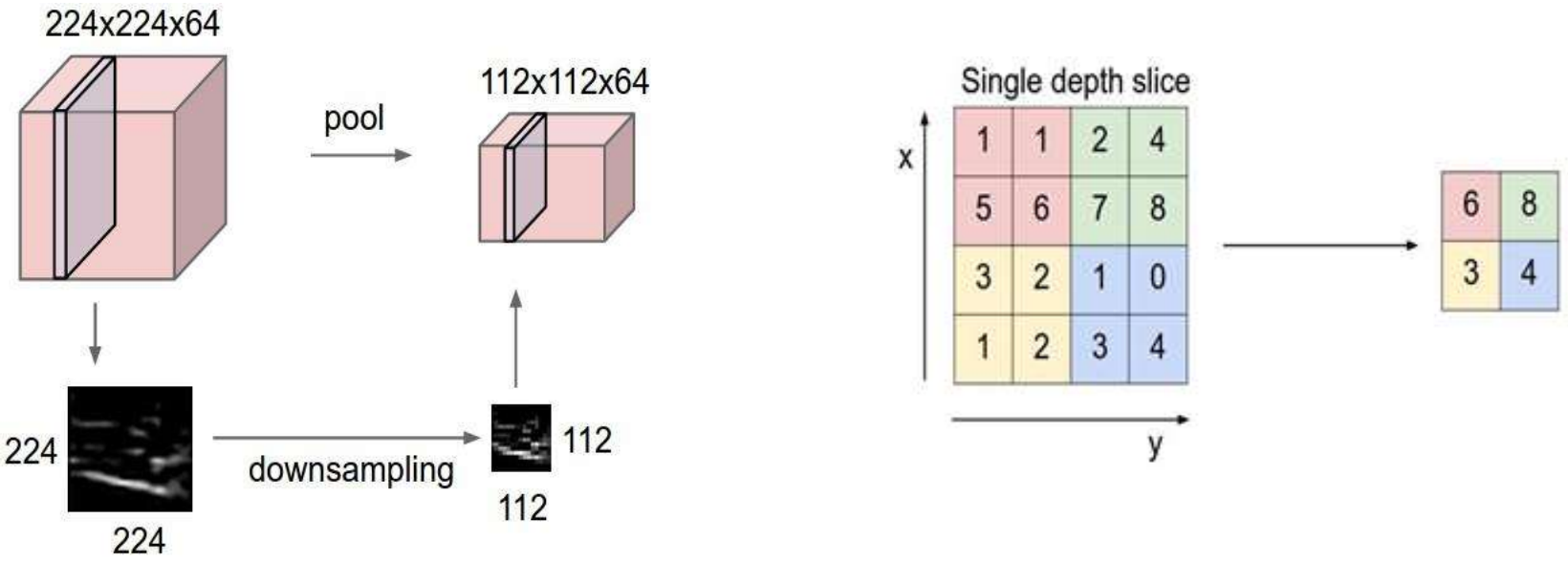
- **Padding**

  We can add a *border* (zero values) to the input volume.

- **Pooling**

  The most used aggregation operators are the maximum (*Max*) and the average (*Avg*).

Example: max-pooling with 2x2 filters and stride=2



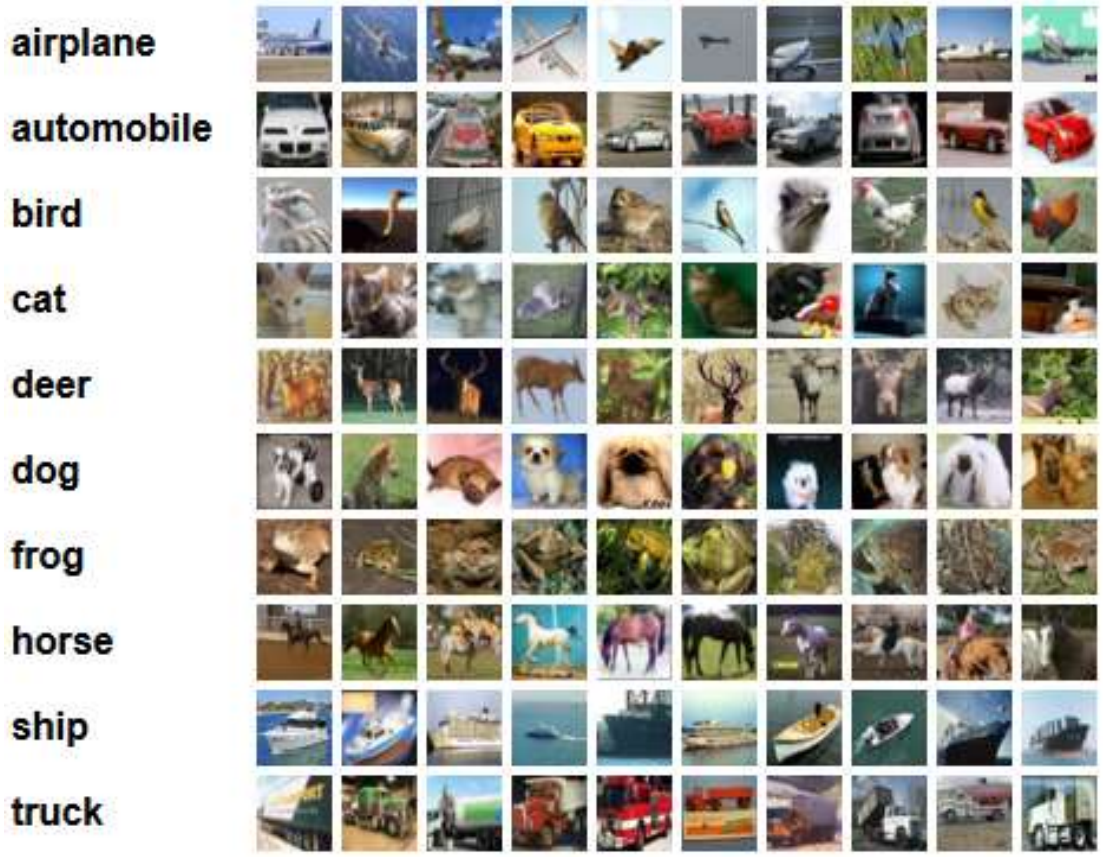
- **Fully connected layers**
  In a CNN, after a certain number of pairs (convolutional layer, pooling layer), one or more fully connected layers are used.

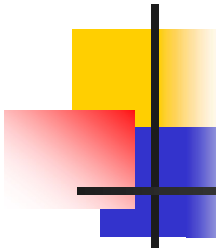
  The size of the last of these layers, which coincides with the output layer, is equal to the number of classes in which the inputs to the CNN are classified.

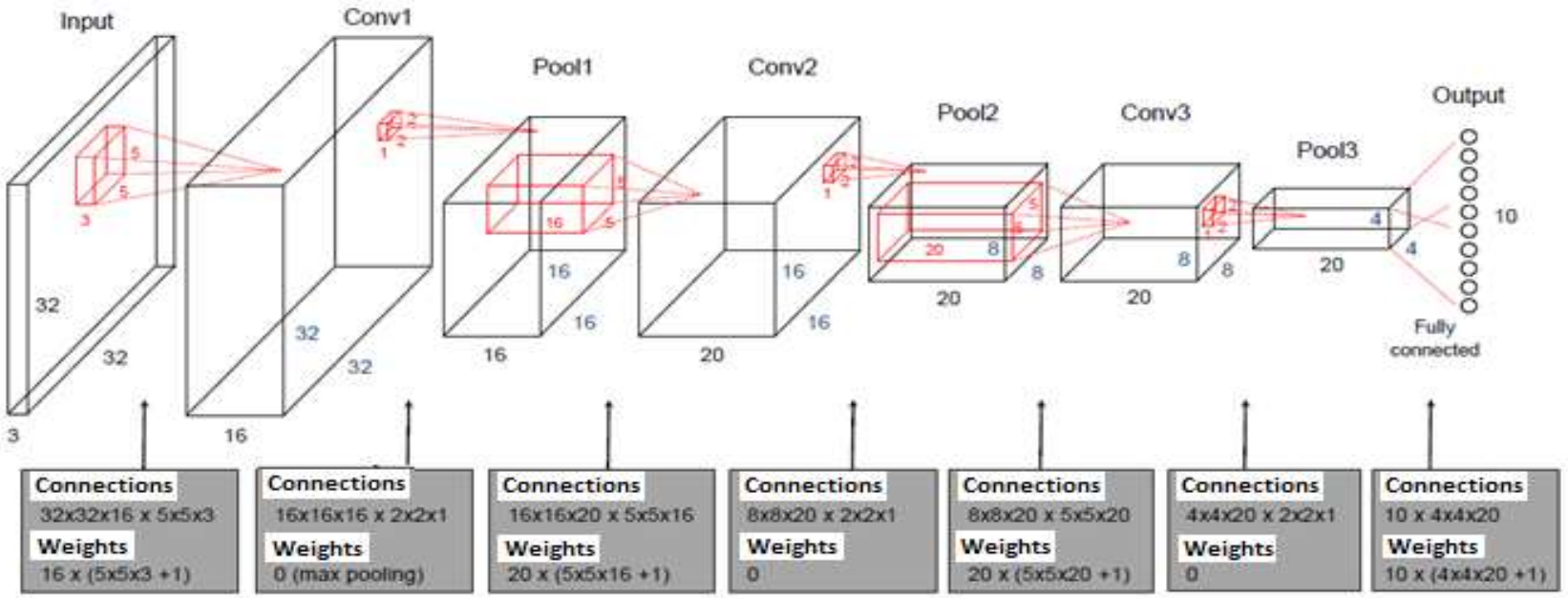
Example 1: Cifar-10

CIFAR-10 is a dataset of 60,000 32x32 color images, belonging to 10 classes (6,000 images per class). There are 50,000 training images and 10,000 test images.

Classes and 10 random images of each of them







Input: 32x32x3 RGB images
Conv1: filters: 5x5, FeatureMaps: 16, stride: 1, pad: 2, activation: ReLU
Pool1: type: Max, filters: 2x2, stride: 2
Conv2: filters: 5x5, FeatureMaps: 20, stride: 1, pad: 2, activation: ReLU
Pool2: type: Max, filters: 2x2, stride: 2
Conv3: filters: 5x5, FeatureMaps: 20, stride: 1, pad: 2, activation: ReLU
Pool3: type: Max, filters: 2x2, stride: 2
Output: Softmax, #classes: 10

Total neurons: 31,562 (including input layer)

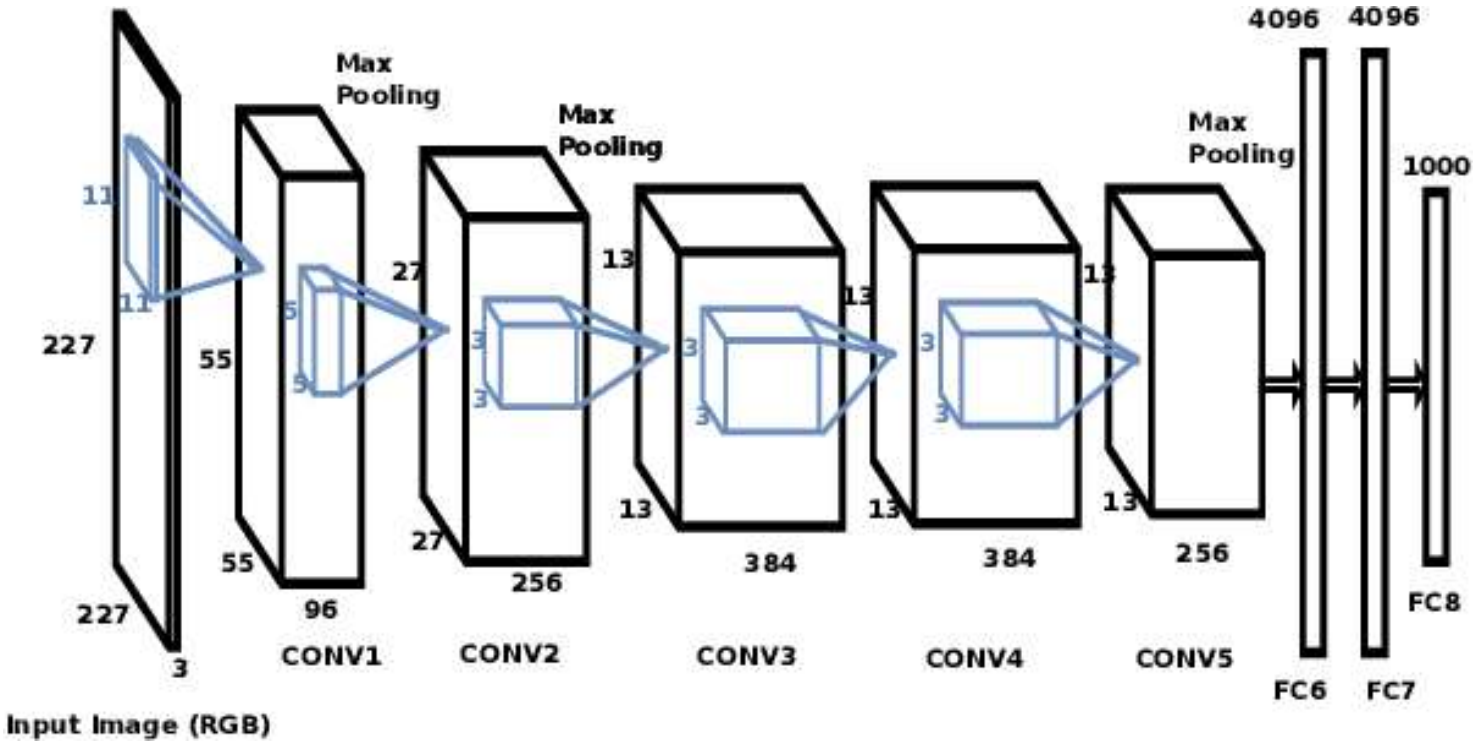
Total connections: 3,942,784

Total weights: 22,466 (including biases)

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Example 2: CaffeNet

CaffeNet is pre-trained on ImageNet (1000 classes and 1.2 million images).

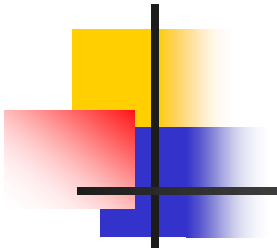


http://faratarjome.ir/u/media/shopping_files/store-EN-1520504599-9025.pdf

In practice

Implementing a CNN (from scratch) is possible but takes a lot of development time and resources. Fortunately, there are many software (often open-source) that allow you to operate on CNNs. Among the most used there are:

Caffe, Theano, Torch, TensorFlow (Google), Keras.



Transfer Learning

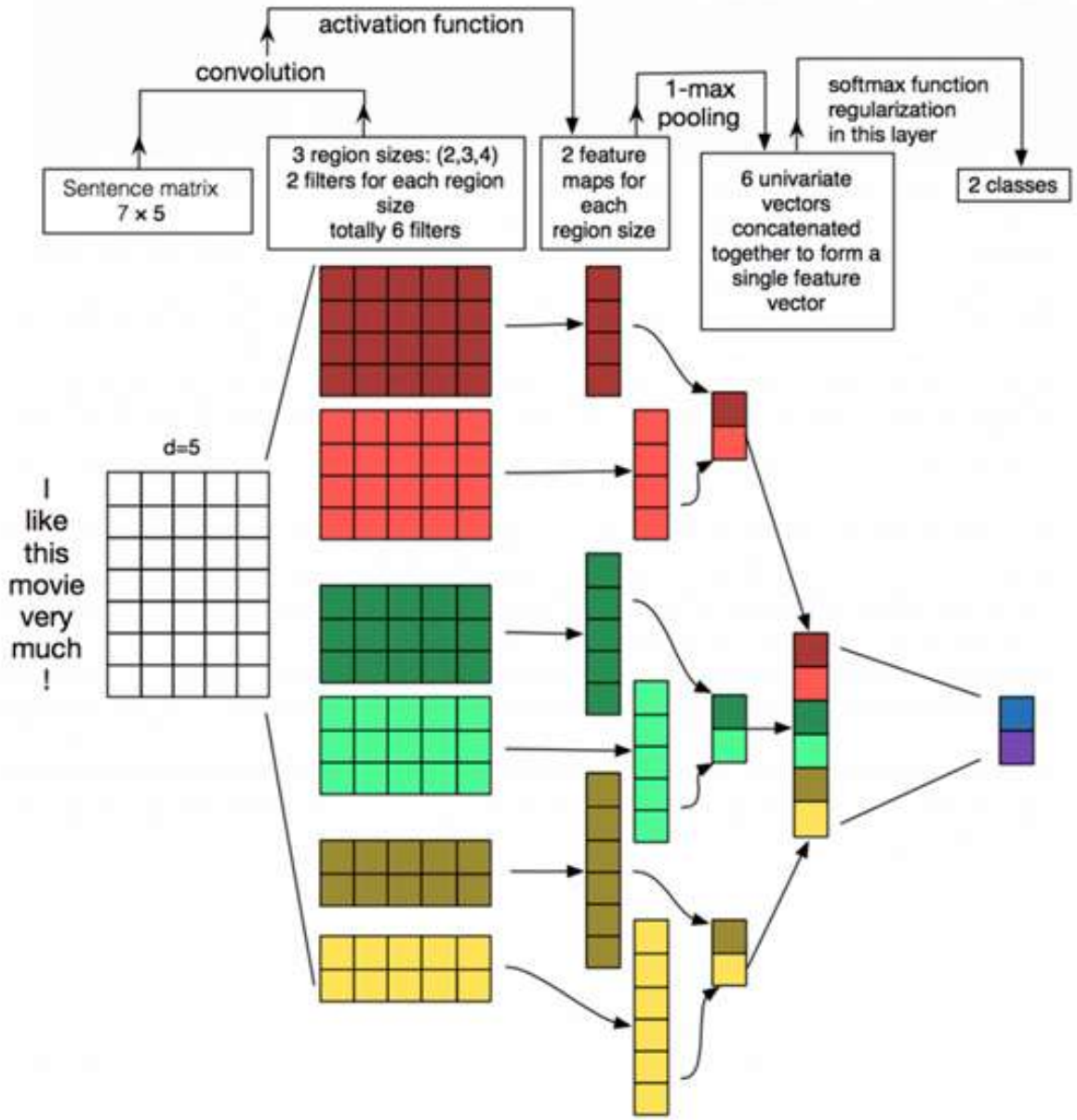
Training complex CNNs on large datasets can take days/weeks even when performed on GPUs.

Additionally, training a CNN on a new problem requires a labeled training set of considerable size (often unavailable). An alternative is *Transfer Learning*:

- Feature reuse: an existing (pre-trained) network is used as *feature extractor*. The features generated by the network during the forward step are extracted (at intermediate layers). These features are used to train an external classifier (e.g., SVM or softmax classifier).

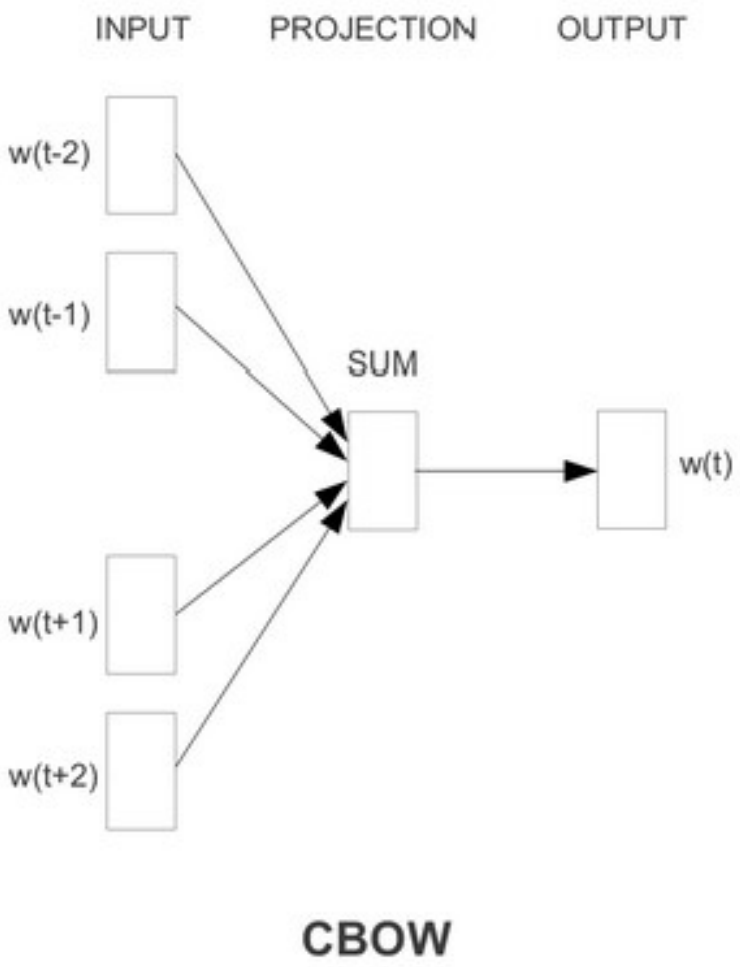
- Fine-Tuning: start with a network pretrained on a similar problem and
  - add your custom network on top of the already-trained base network;
  - freeze the base network;
  - train the newly added part;
  - unfreeze some of the top layers of the base network;
  - jointly train both these top layers and the newly added part.

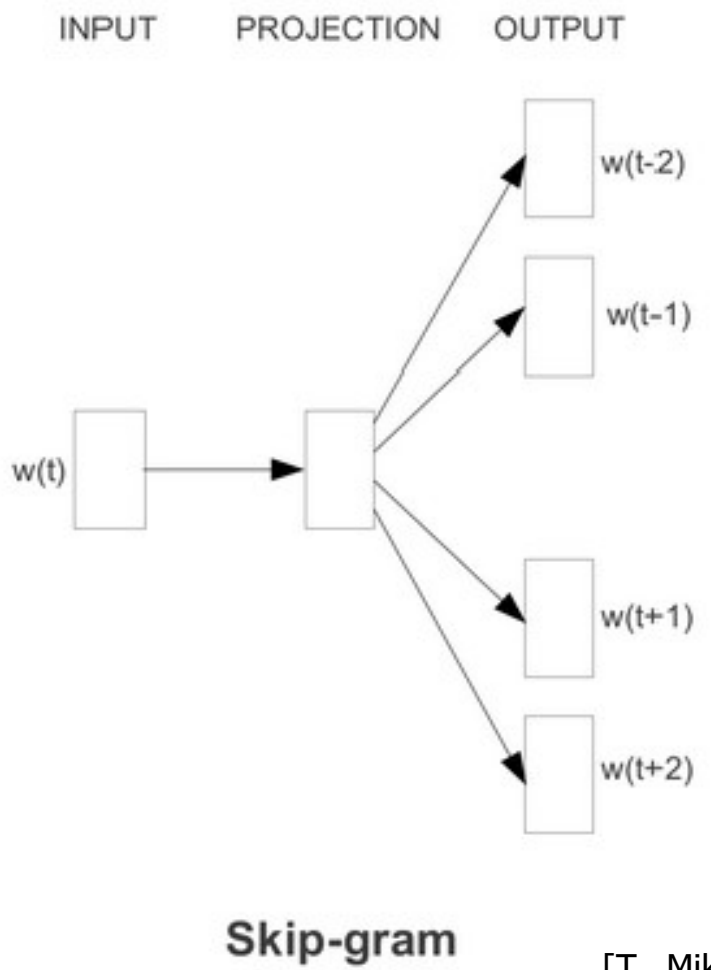
CNN for sentence classification



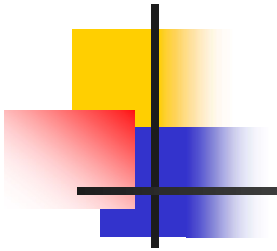
[Y. Zhang, B.C. Wallace]

Word2Vec





[T. Mikolov, K. Chen, G. Corrado, J. Dean]



Object Detection

Objective: to localize and classify objects in an image.

Two classes of algorithms:

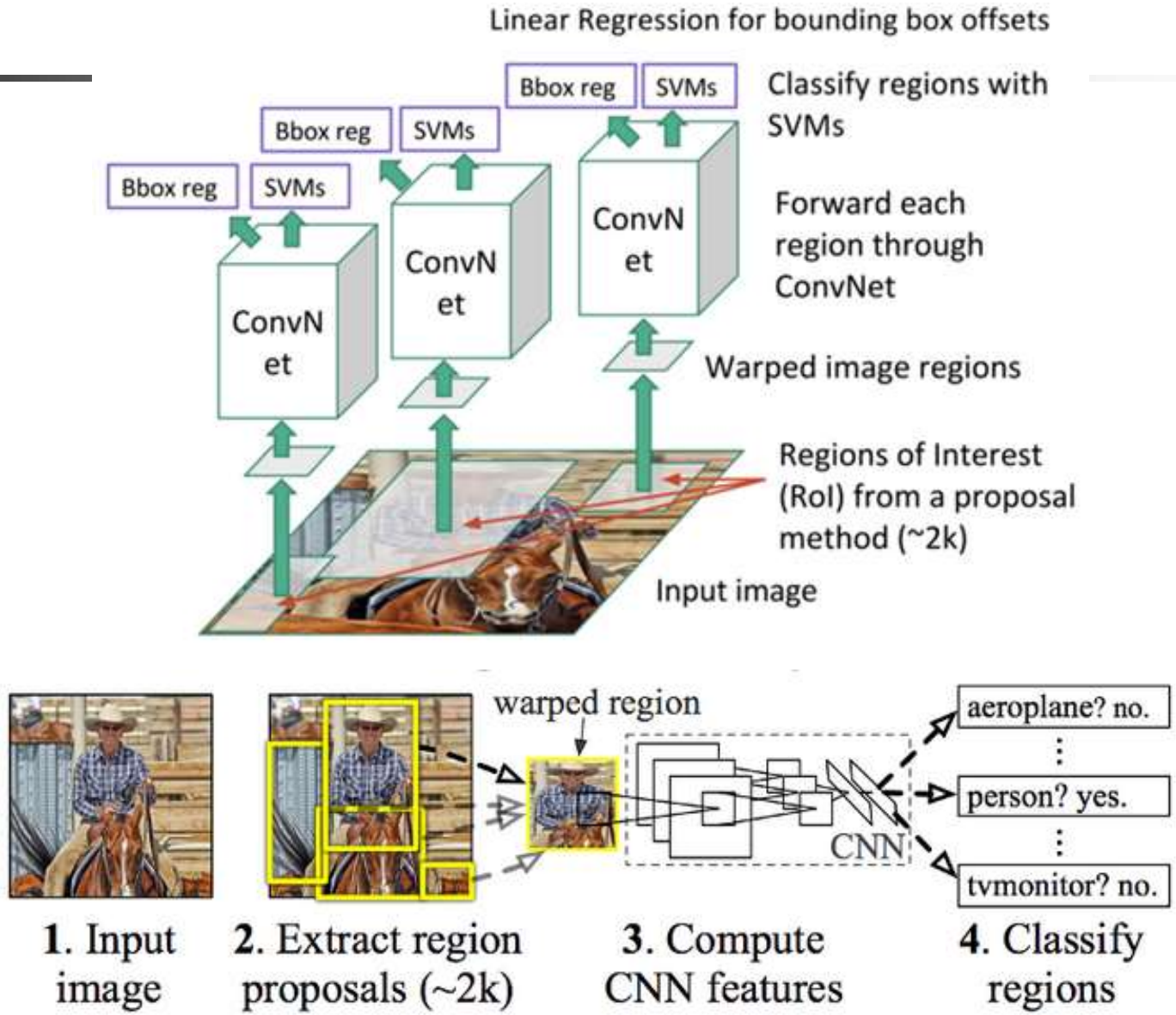
- two steps:
  1. Select interesting regions from the image
  2. Classify those regions

  R-CNN (Region-based CNN)
  Fast R-CNN
  Faster R-CNN

- one step:
  1. Predict classes and bounding boxes at the same time

  YOLO (You Only Look Once)
  SSD (Single Shot Detector)

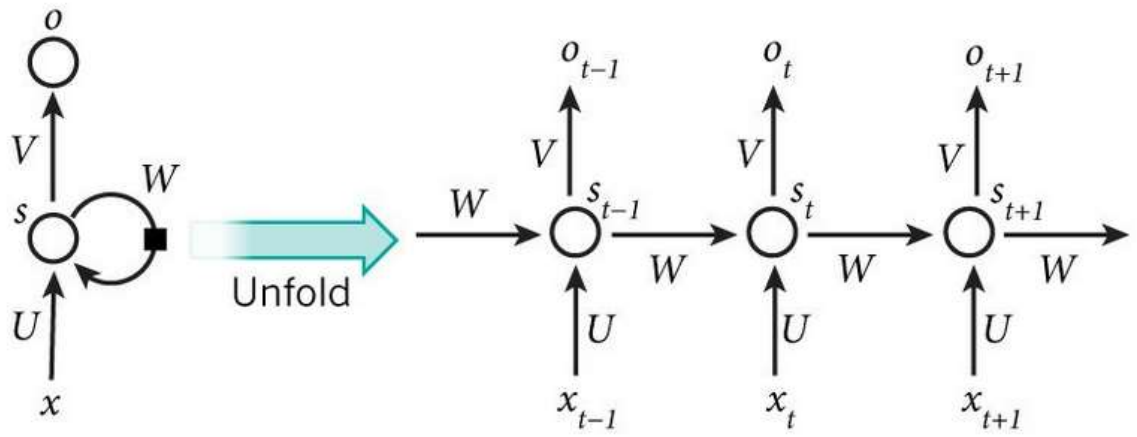
Region-based CNN (R-CNN)

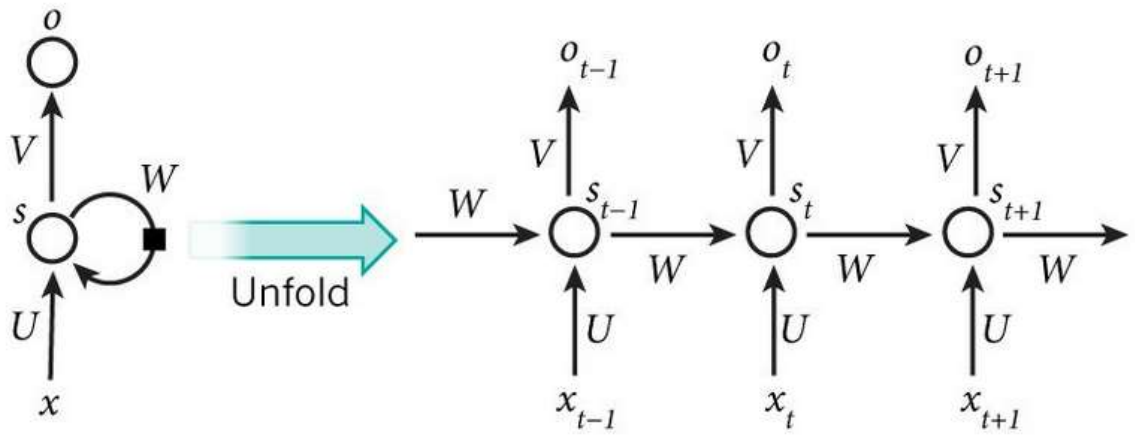


[R. Girshick]

Recurrent Neural Network (RNN)

- RNNs process an input *sequence* one element at a time and maintain a '*state vector*' in their hidden units.
- RNNs perform the same task for every element of a sequence.
- A typical RNN can be unrolled by representing the network for the complete sequence:





- x_t input at time step t
- s_t hidden state at time step t (it is the memory of the network)

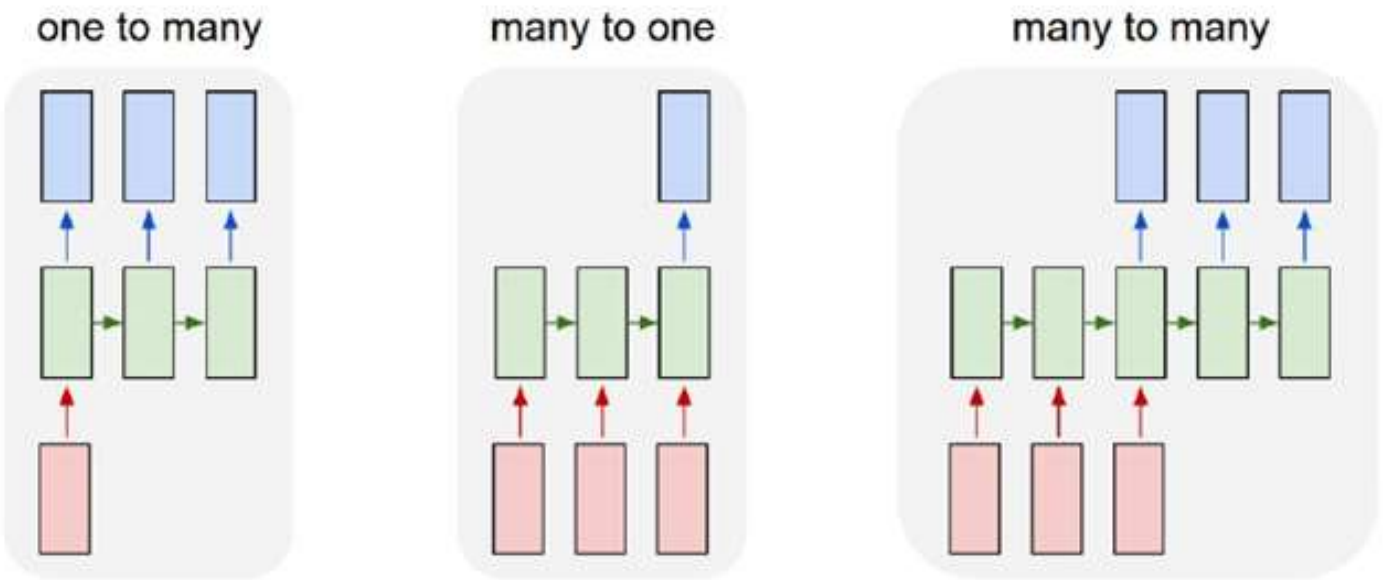
$$s_t = f\left(U\, x_t + W\, S_{t-1}\right)$$

- o_t output at time step t (it depends on the memory at time t).

$$o_t = f\left(V\, s_t\right)$$

- The RNN shares the same parameters (U, V, W) in all steps.

Applications of RNNs



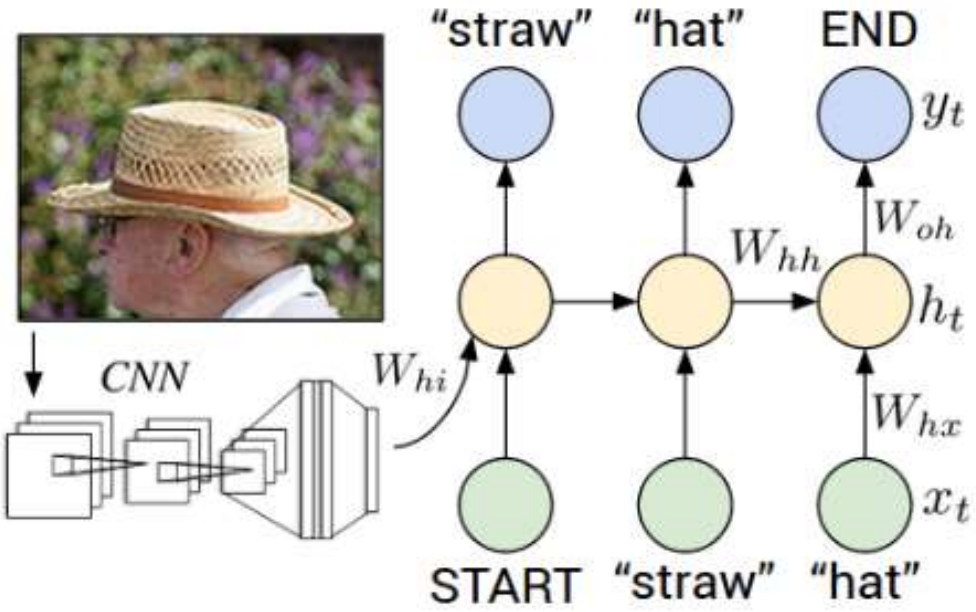
1 **2** **3**

1: image captioning (image → sequence of words)

2: Sentiment classification (sequence of words → sentiment)

3: Machine translation (sequence of words → sequence of words)

Image Captioning

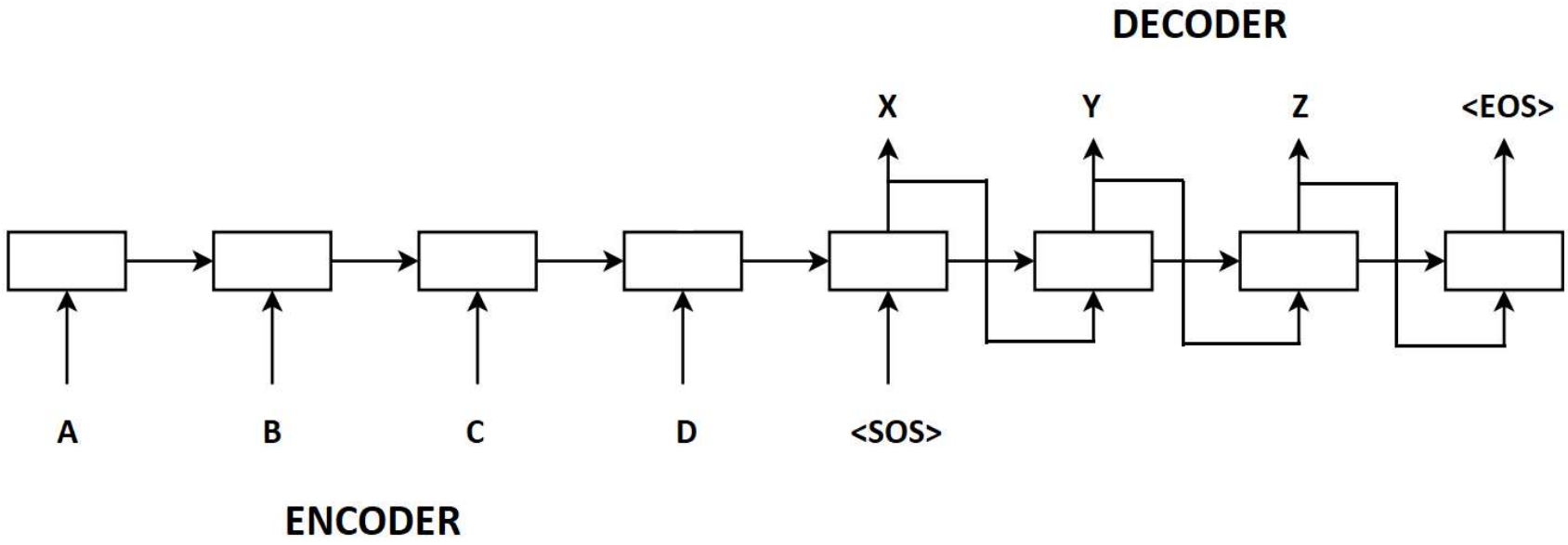


[A. Karpathy]

Encoder: A CNN extracts the features from the input image.

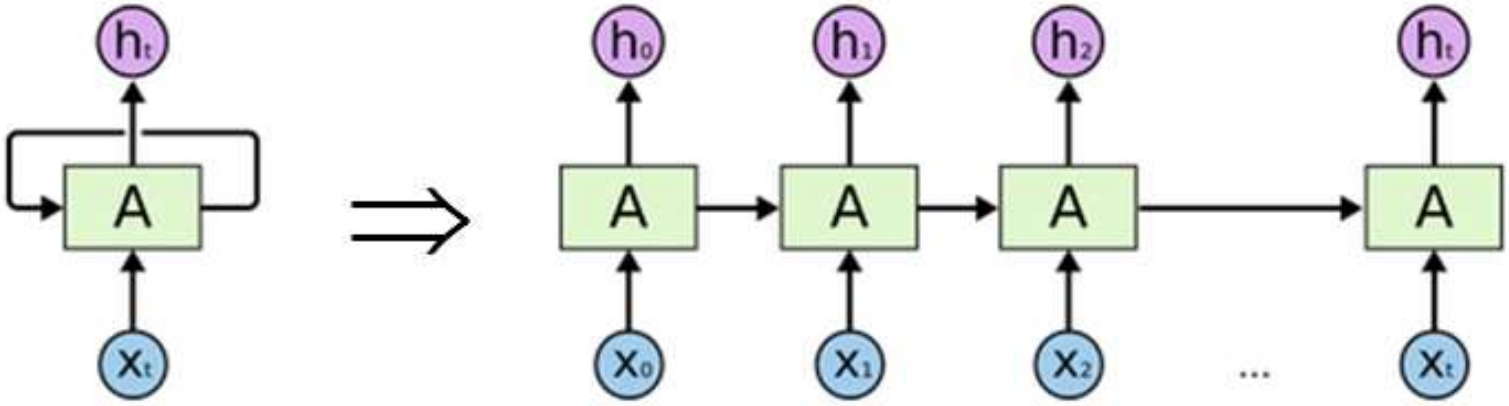
Decoder: An RNN translates the features into a natural sentence. More precisely, the RNN predicts the next word in a sentence. Note that we also insert <start> and <end> to signal the beginning and end of the caption.

Neural Machine Translation (NMT)



Training RNNs

- Unfold the network
  - Remove loops



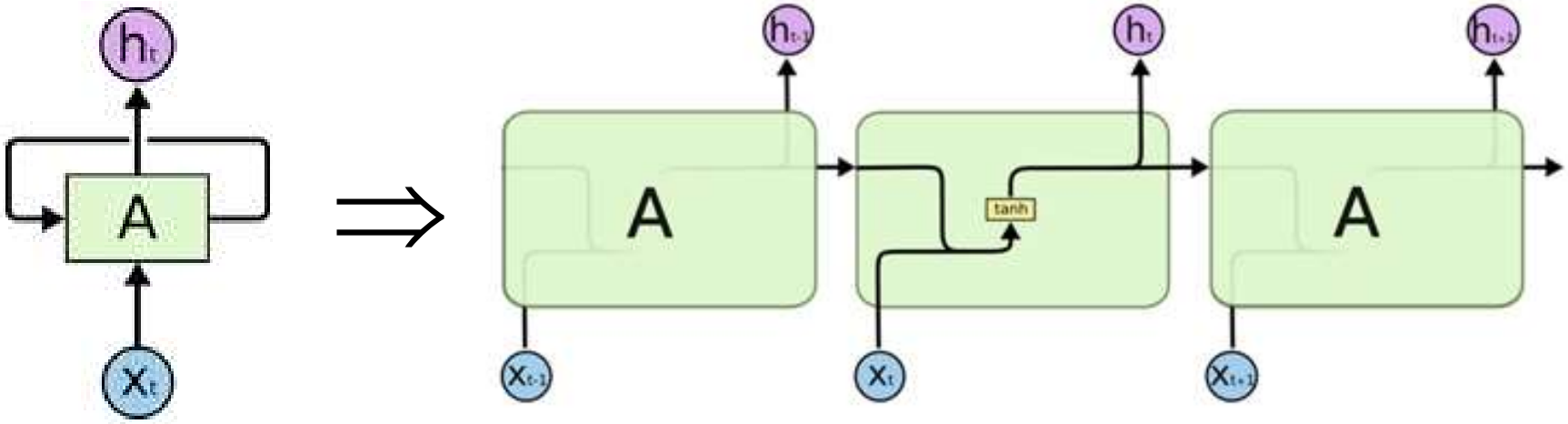
- Backpropagation Through Time (BPTT)

LSTM

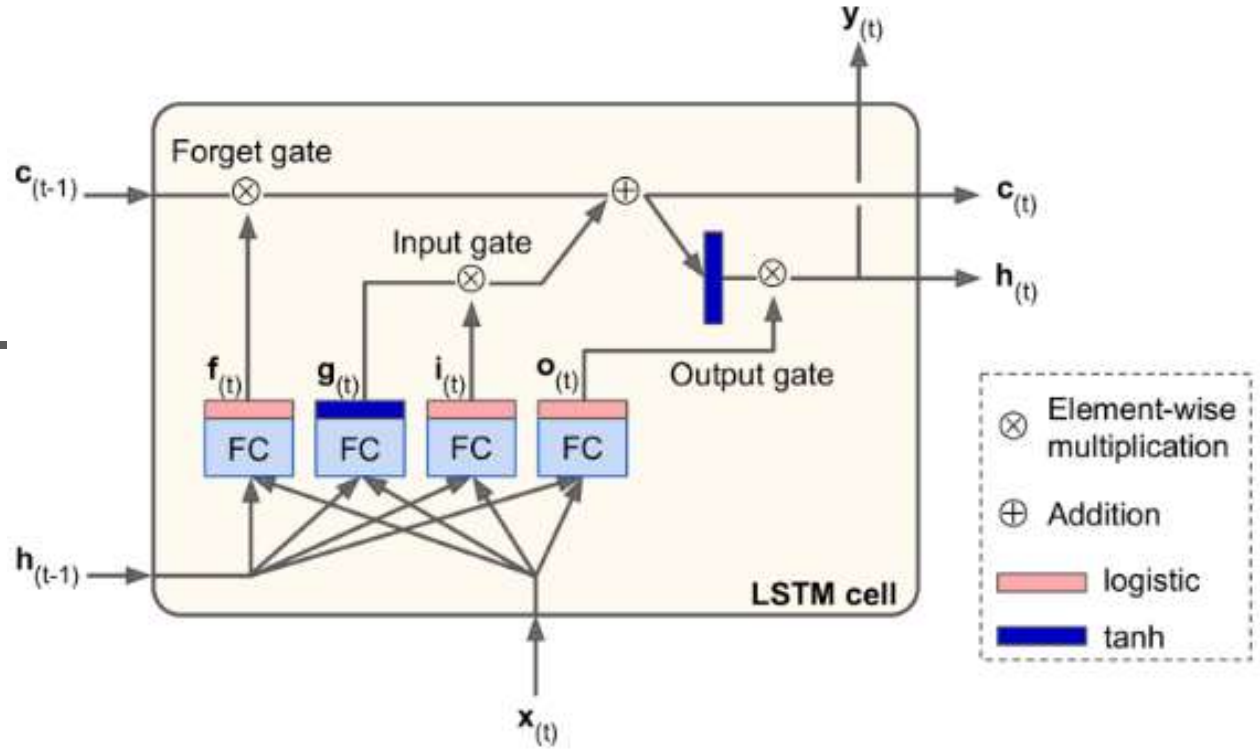
- The problem of *long-term dependencies*
  - as the gap between relevant information and where it is needed increases, RNNs become unable to learn how to connect information.

Long Short-Term Memory (**LSTM**) is a special type of RNN that can learn long-term dependencies. It can remember information for long periods of time.

RNNs have the form of a chain of repeating NN modules. In standard RNNs, the repeating module has a very simple structure, e.g., a single *tanh* layer:

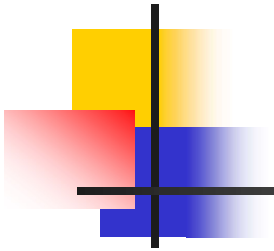


- In LSTMs, the repeating module has four, interacting fully connected layers.
- The LSTM can remove or add information to the cell state, by means of *gates*.
- The gates selectively let information through. They consist of a sigmoid neural net layer and an elementwise multiplication operation.
- The sigmoid layer outputs numbers between zero and one, deciding how much of each component is to be let through: zero means "let nothing through", one means "let everything through".
- An LSTM has three of these gates.



[A. Géron]

- The state of an LSTM cell is split into two vectors: $\mathbf{h}_{(t)}$ (short-term state) and $\mathbf{c}_{(t)}$ (long-term state).
- During training the network learns what to throw away from the long-term state, what to store in the long-term state, and what to read from it.
- The long-term state $\mathbf{c}_{(t-1)}$ passes through a *forget gate*, dropping some memories, then adds new memories selected by an *input gate*. The result $\mathbf{c}_{(t)}$ is sent out, without any further transformation. After the addition operation, $\mathbf{c}_{(t)}$ is copied and passed through the *tanh* function, then the result is filtered by the *output gate*. This produces the short-term state $\mathbf{h}_{(t)}$ (which is equal to the cell's output $\mathbf{y}_{(t)}$).



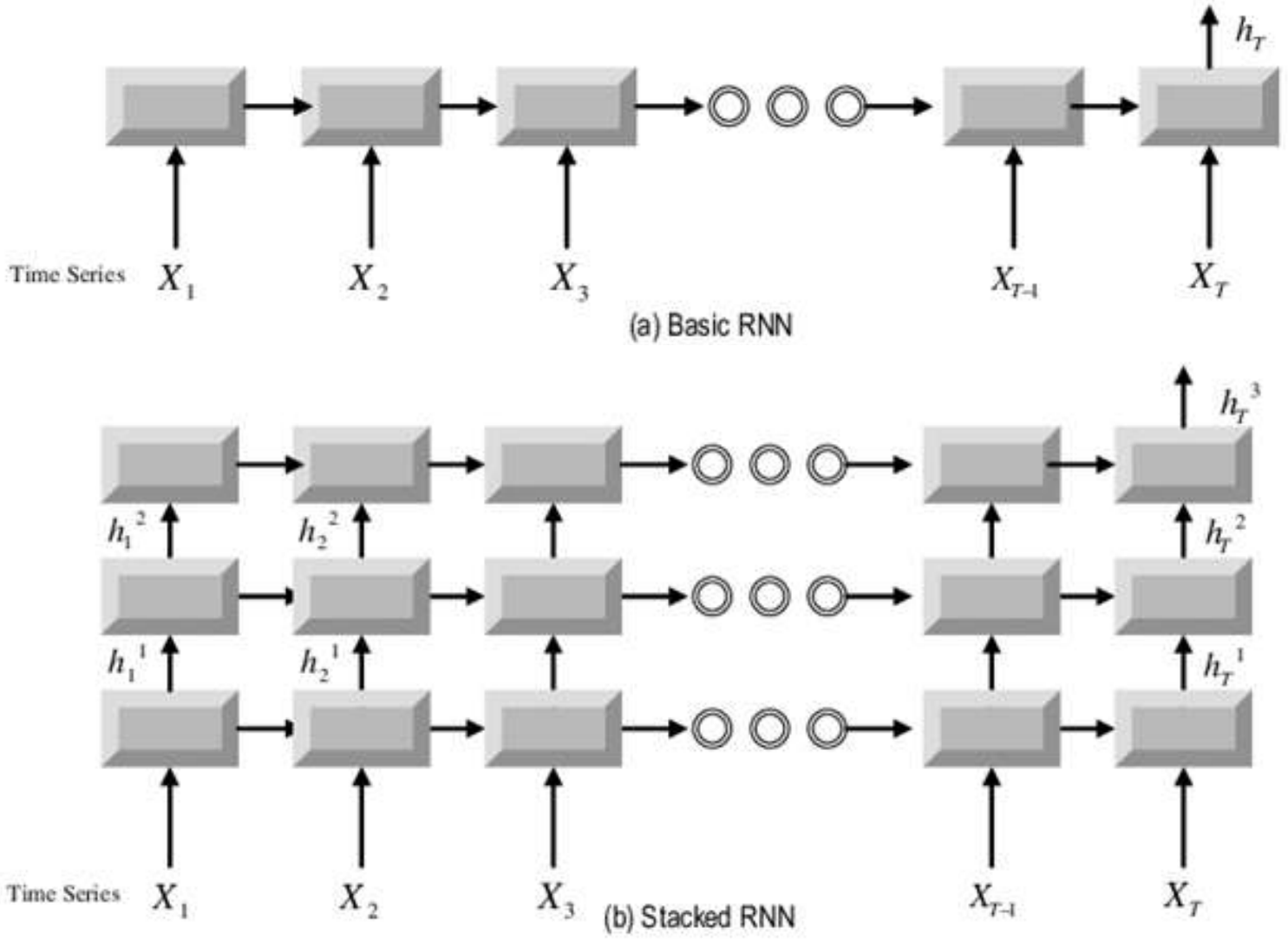
Summarizing: an LSTM cell can learn to recognize an important input, store it in the long-term state, learn to preserve it for as long as necessary, and learn to extract it whenever it is needed.

GRU

Gated Recurrent Unit (*GRU*) is a simplified version of LSTM

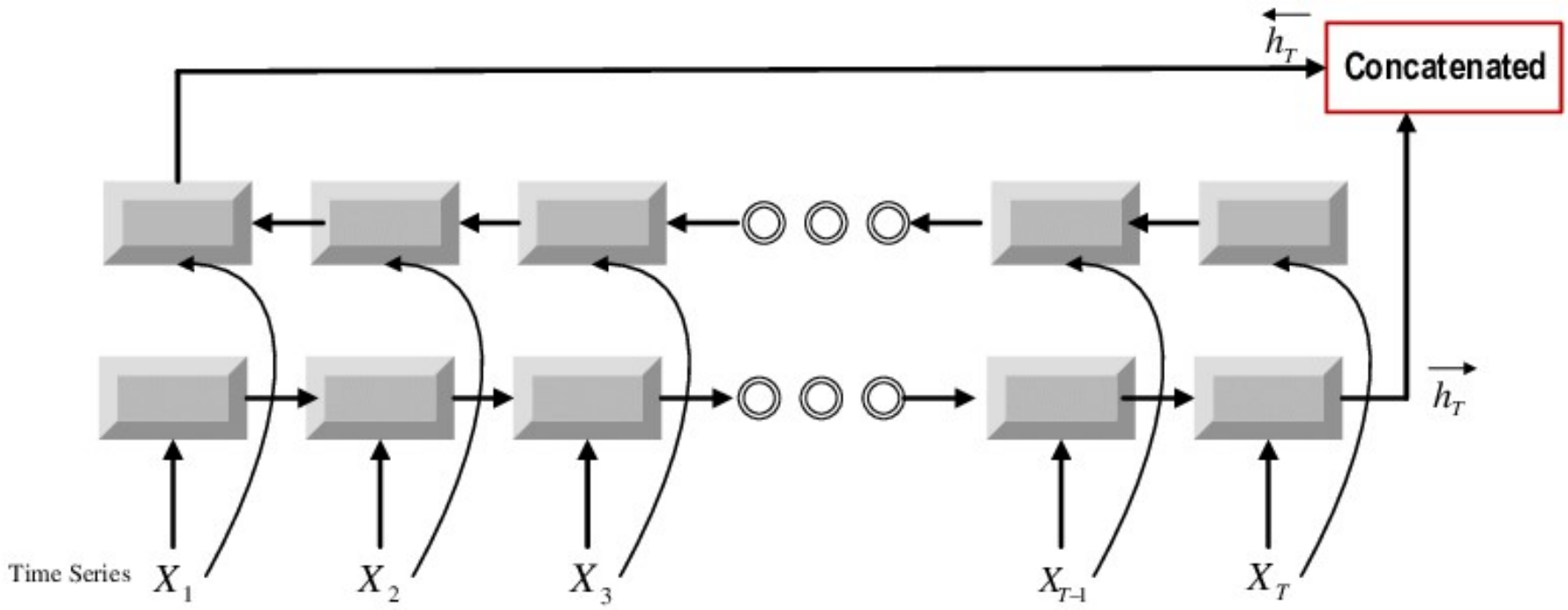
- one memory state
- two gates: update gate and reset gate

Stacked RNN



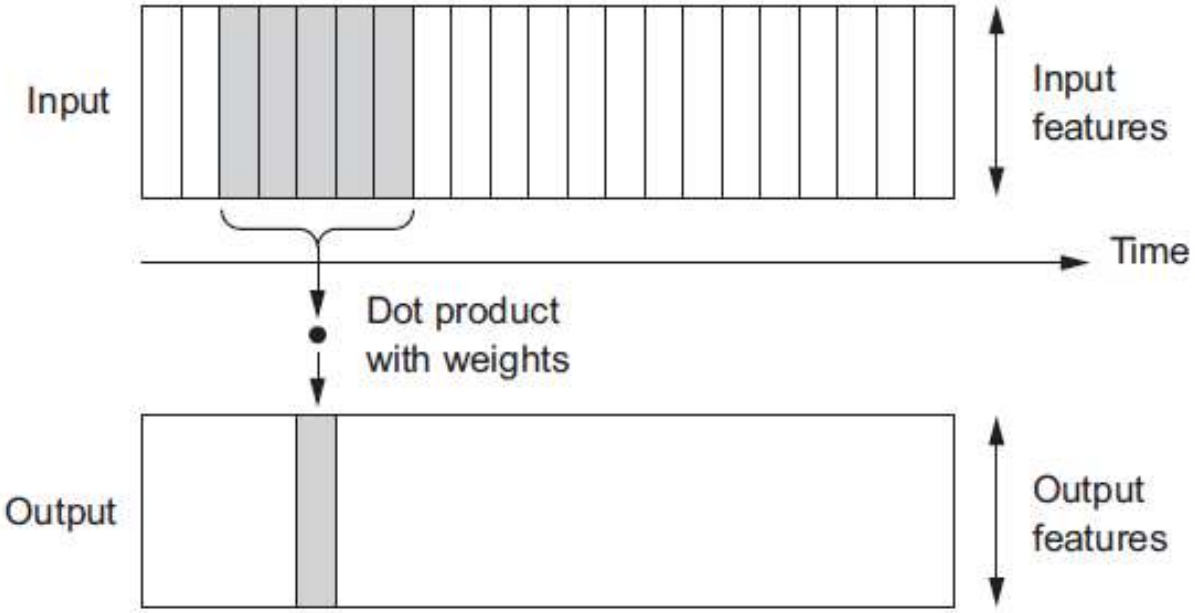
https://www.researchgate.net/figure/Illustrations-of-normal-RNN-stacked-RNN-and-bidirectional-RNN_fig7_311839720

Bidirectional RNN

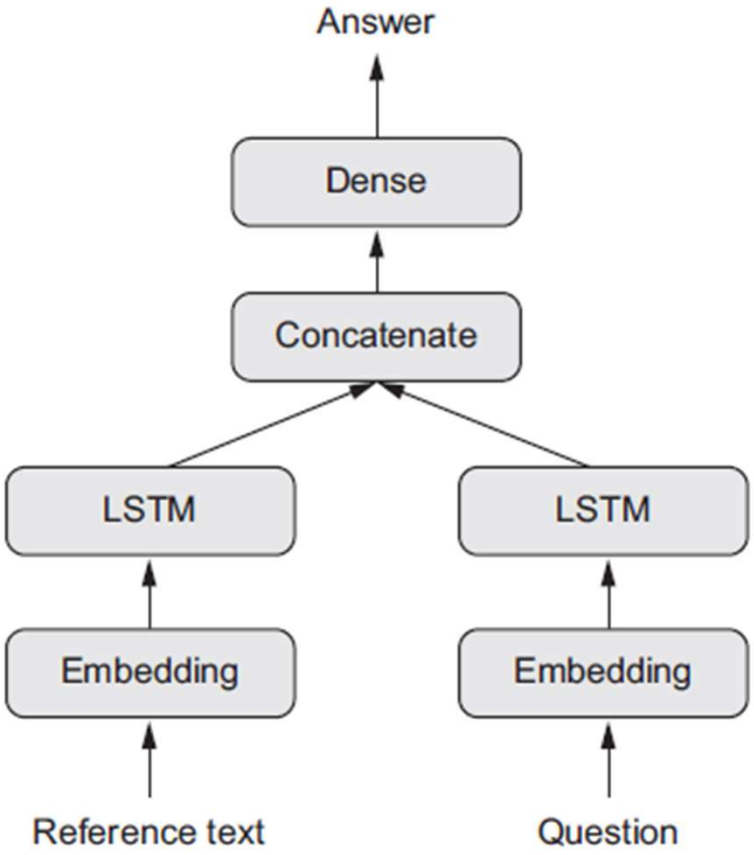


https://www.researchgate.net/figure/Illustrations-of-normal-RNN-stacked-RNN-and-bidirectional-RNN_fig7_311839720

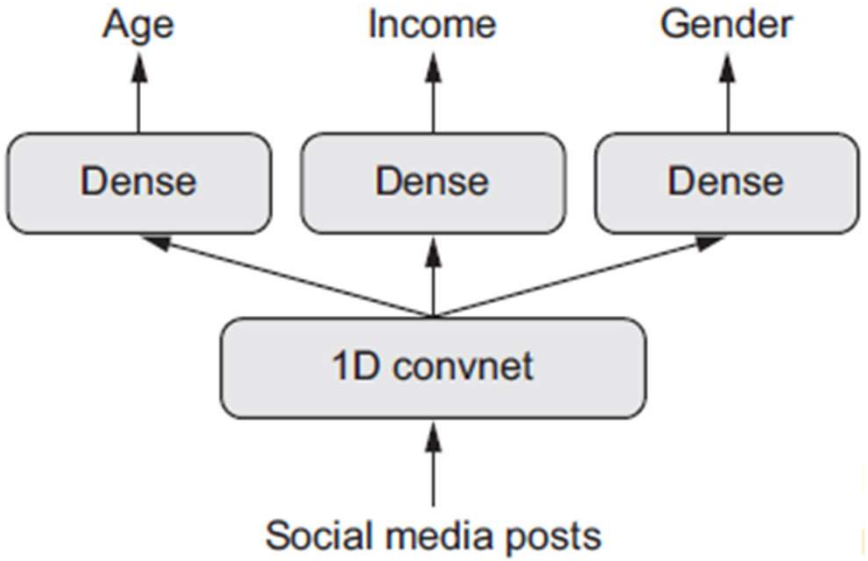
1D CNN


Input
Input features
Time
Dot product with weights
Output
Output features

Multi-input model
Multi-output model

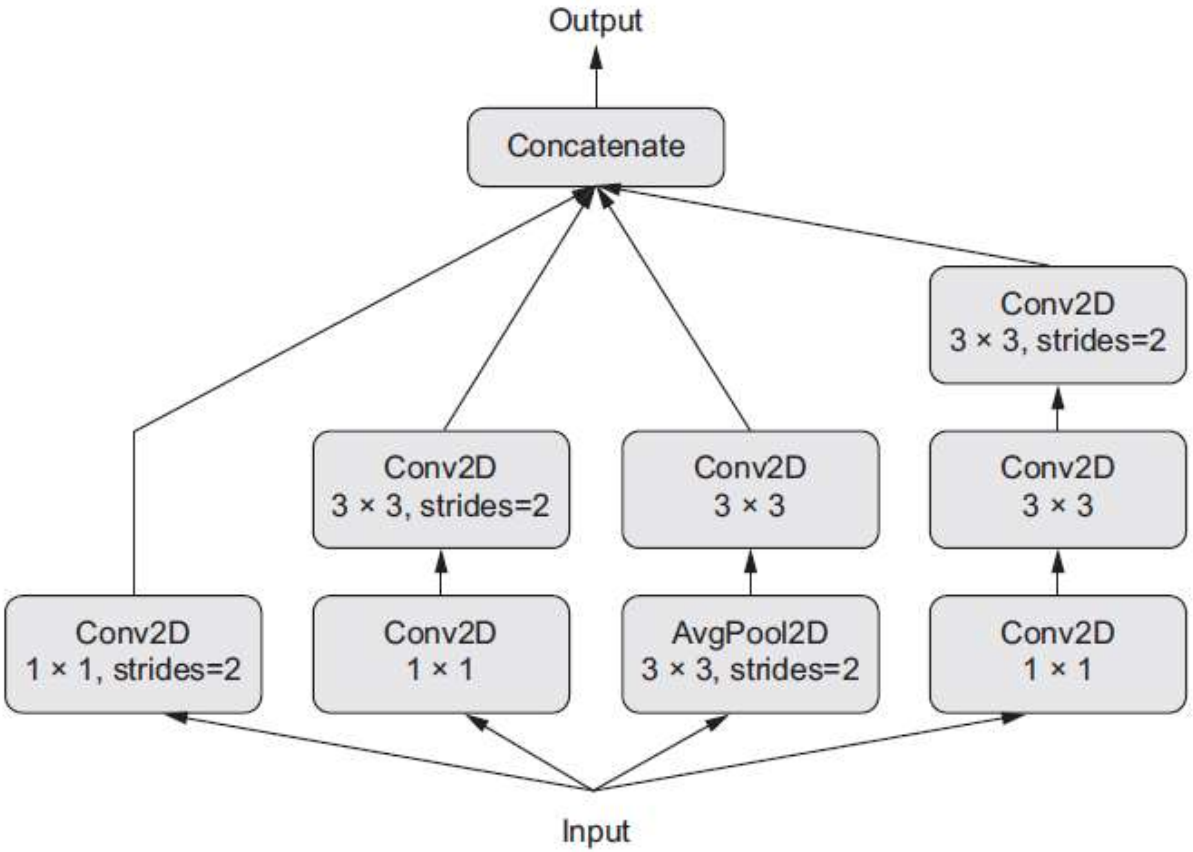


Multi-input model
(question-answering model)



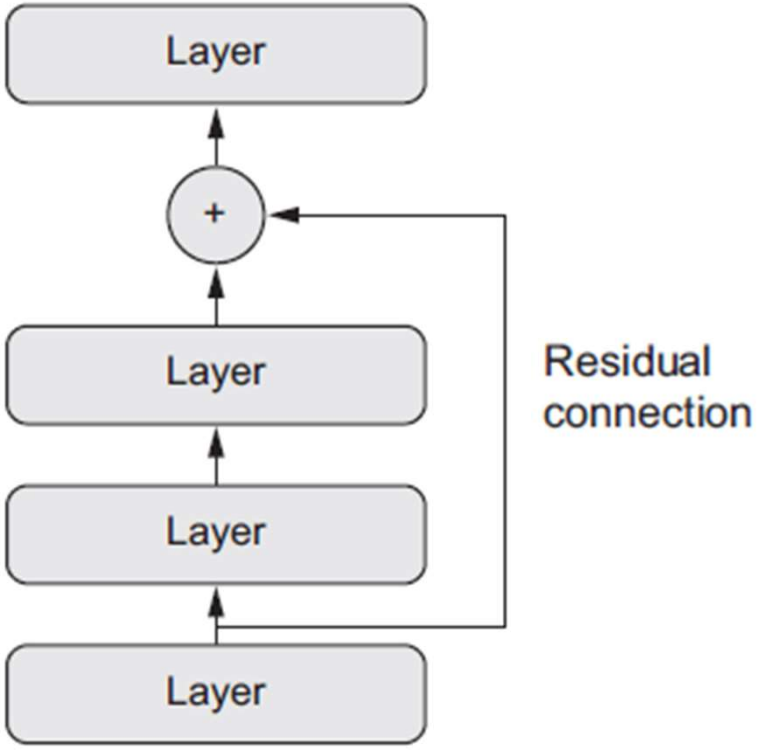
Multi-output model

Inception module



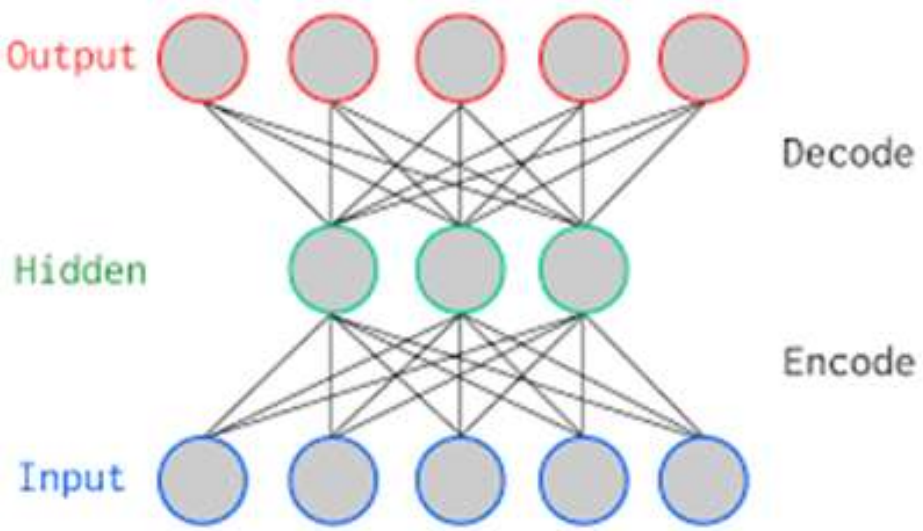
- The input to the inception module is processed by parallel convolutional branches.
- The features of the branches are concatenated.

Residual connection



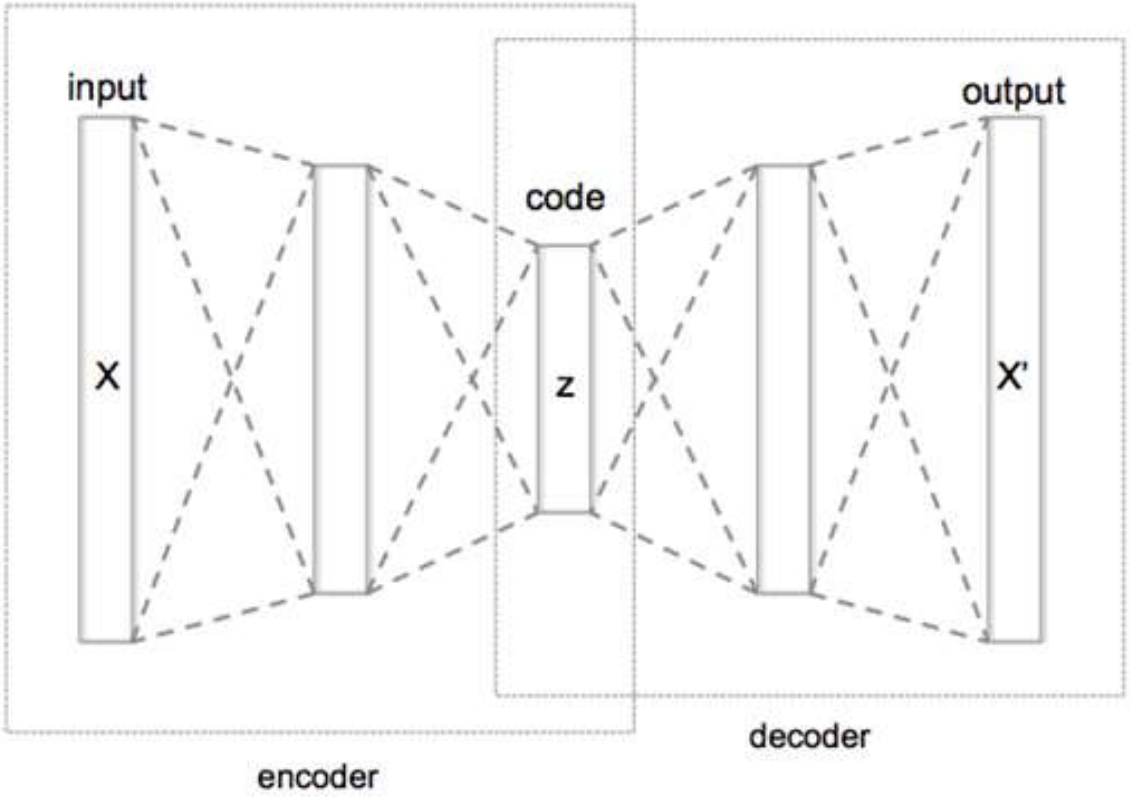
- The output from a previous layer is added to the input of a subsequent layer.

Classical autoencoder



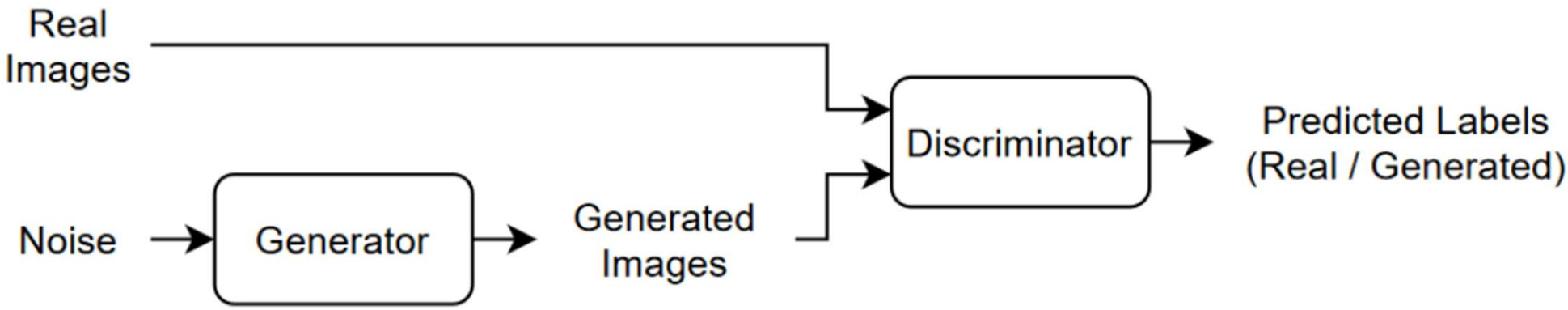
Decode

Encode



- A classical autoencoder learns a compressed representation of a data set.
- The training process minimizes the *reconstruction error*.

Generative Adversarial Network (GAN)



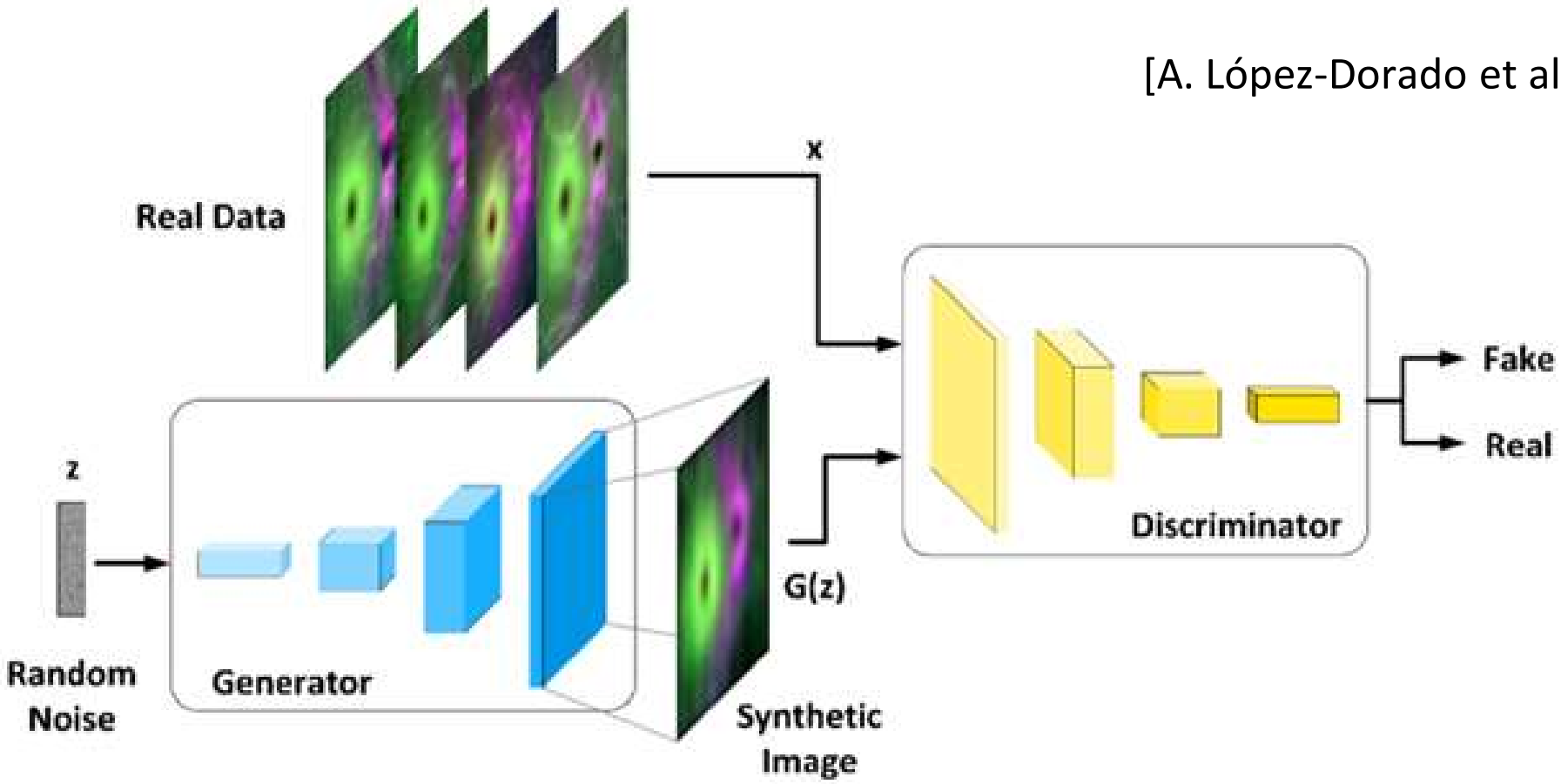
The *generator* receives a vector of random values and generates synthetic data with the same structure as the training data.

The *discriminator* is a classifier that learns to classify data as "real" or "generated" (false).

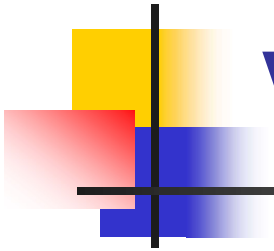
The *generator* is trained to deceive the discriminator, that is, to make the discriminator classify the generated data as real.

As the performance of the generator improves during training, the performance of the discriminator deteriorates because the discriminator cannot easily distinguish between real and false data.

[A. López-Dorado et al.]

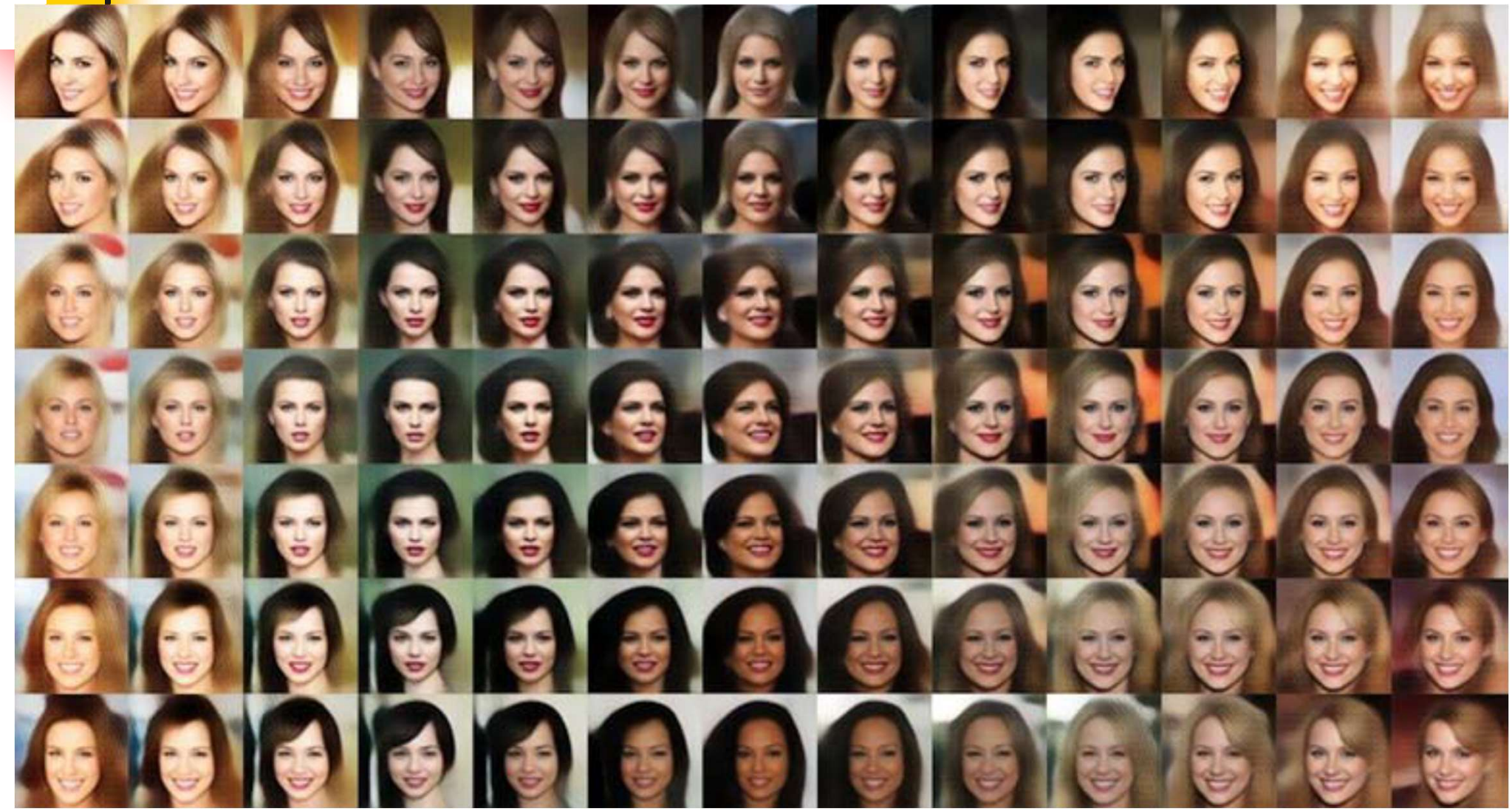


A few sample faces — all completely fake — created by ThisPersonDoesNotExist.com



Variational autoencoder (VAE)

- The encoder of a VAE returns a distribution over the latent space (mean and variance).
- The loss function is composed of both a *reconstruction term* (on the final layer) and a *regularization term* (on the latent layer).
- The latent space is continuous and highly structured.



A continuous space of faces generated by Tom White using VAEs