Basics of Convolutional Neural Networks

Olivier Le Meur
olemeur@irisa.fr

IRISA - University of Rennes 1, France

ECOLE SUPERIEURE D'INGENIEURS DE RENNES

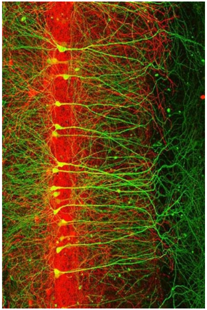
October 5, 2020

Outline

Outline

The artificial neuron

- The human brain contains around 80 billion neurons.
  - Mouse≈75 million neurons;
  - Cat≈1 billion neurons;
  - Chimpanzee≈7 billion neurons.
- A neuron is a nerve cell that is the basic building block of the nervous system.
- Neurons are specialized to transmit information throughout the body.

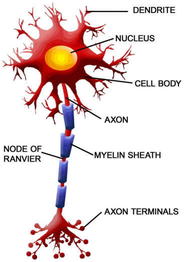


Courtesy of Erik Bloss, Janelia Research Campus

The artificial neuron

There are three basic parts of a neuron: the dendrites, the cell body, and the axon.

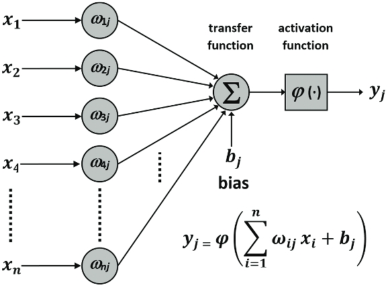
- the dentrites receive information from sensory receptors or other neurons.
- the cell body processes incoming information.
- the axon: each neuron has one axon that transmit the information to the following cell.



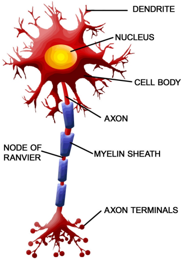
From http://www.interactive-biology.com/3247/the-neuron-external-structure-and-classif

The artificial neuron

A common scheme of a single neuron (perceptron (McCulloch and Pitts, 1943, Rosenblatt, 1958)):



Adapted from (Álvarez et al., 2017)



From http://www.interactive-biology.com/3247/the-neuron-external-structure-and-classif

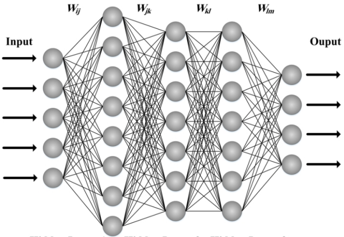
The basic model for a neuron j, defined for a generic input $\boldsymbol{x} \in \mathcal{R}^n$:

- performs the weighted linear activation, $\boldsymbol{w}_i \in \mathcal{R}^n$;
- use an activation function φ, for simulating the firing rate of the cell (e.g. sigmoid function, hyperbolic tangent function).

The artificial neuron

From a perceptron to a neural network:

- One perceptron outputs one decision;
- For multiple decisions (e.g. digit classification), stack as many outputs as the possible outcomes into a layer $\Rightarrow$ Neural Network;
- Use one layer as input to the next layer (Multi-layer perceptron).



Adapted from (Hosseini and Samanipour, 2015)

Humm, a number of weights to train...

Note that a neural network without an activation function boils down to a simple linear regression model.

The artificial neuron

A few words on backprogation algorithm:

- Measure the prediction error or loss z, error between the actual data and the prediction $\Rightarrow$ loss function;
- Optimize weights to reduce loss $\Rightarrow$ partial derivative of the loss w.r.t the weights;
- Backpropagate the loss, layer by layer, until all neuron weights have been improved (non-convex optimization by gradient descent):

$$\left(\boldsymbol{w}_i\right)^{t+1} = \left(\boldsymbol{w}_i\right)^{t} - \eta \frac{\partial z}{\partial \left(\boldsymbol{w}_i\right)^{t}} \tag{1}$$

  where $\boldsymbol{w}_i$ represents the weights of the i^{th} layer, η the learning rate (small positive value) and t the time index.
- Repeat until convergence

The artificial neuron

Limitations of deep neural networks at that time:

- Lack of processing power (1958-1998), no GPU...
- Lack of data, no super big annotated datasets
- Limited performance due to the limited training ability (processing power and data), models do not generalize well.

After a long AI winter, from 1998-2006, the deep neural networks come back with an amazing success.

Outline

The big picture

A family of parametric, non-linear and hierarchical representation learning functions, which are massively optimized with batch/stochastic/mini-batch gradient descent to encode domain knowledge, i.e. domain invariances, stationarity.

$$\hat{y}_L\left(x;\theta_{1,\ldots,L}\right) = h_L\left(h_{L-1}\left(\ldots h_1\left(x;\theta_1\right),\theta_{L-1}\right),\theta_L\right) \tag{2}$$

- x, input; θ_l, parameters for layer l, $\hat{y}_l = h_l(x,\theta_l)$, a (non)linear function.

Given training corpus $\{X, Y\}$, find optimal parameters to minimize the loss:

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\subseteq(X,Y)} \Phi\left(y;\hat{y}_L\left(x;\theta_{1,\ldots,L}\right)\right) \tag{3}$$

with Φ the chosen loss function.

Adapted from *Introduction to deep learning and neural networks*, UVA deep learning course - Efstrations GAVVE.

End-to-end learning

Given training corpus $\{X, Y\}$, find optimal parameters to minimize the loss:

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y) \subseteq (X,Y)} \Phi\left(y; \hat{y}_L\left(x; \theta_{1,\dots,L}\right)\right) \tag{4}$$

- A pipeline of successive modules
- Each module's output is the input for the next module
- Modules produce features of higher and higher abstractions
- Features are also learned from data!
  - hand-crafted feature extraction are no more required, such as SIFT, SURF, HoG....
  - they are very compact and specific for the task at hand
  - time spent for designing features now spent for designing architectures!

Adapted from *Introduction to deep learning and neural networks*, UVA deep learning course - Efstrations GAVVE.

Different types of deep neural network

- Deep (5-20 layers) vs Shallow (1-2 layers) neural network;
- Supervised vs Unsupervised:
  - Unsupervised learning infers a function that describes the structure of unlabeled data ($\Rightarrow$ Autoencoders, Deep Belief Nets, Generative Adversarial Networks, Self-organizing map);
  - Supervised learning.
    Given a bunch of input data X and labels Y, we are learning a function $f : X \rightarrow Y$ that maps X (e.g. images) to Y (e.g. class label). The function will be able to predict Y from novel input data with a certain accuracy if the training process converged.
    - Convolution Neural Network, appropriate for visual data
    - Recurrent Neural Network, appropriate for text, sound, series

Outline

What is a 2D discrete convolution in image processing? (1/3)

- Let $I : \Omega \subset \mathcal{R}^2 \rightarrow \mathcal{R}^m$ an input image;
- Let $\overline{I} : \Omega \subset \mathcal{R}^2 \rightarrow \mathcal{R}^n$ the transformed image.

Our goal is to fill in each location of $\overline{I}$ with a weighted sum of the pixel values from the locations surrounding the corresponding location in the image, using the same set of weights each time.

- Shift-invariant = the value of the output depends on the image neighbourhood, rather than the position of the neighbourhood;
- Linear = the output for the sum of two images is the same as the sum of the outputs obtained for the images separately. An operator T is linear if:
  - $T(f+g) = T(f) + T(g)$, $\forall f, g$;
  - $T(\alpha f) = \alpha T(f)$, $\forall f$, scalars α.

Any linear shift-invariant operation can be represented by convolution.

What is a 2D discrete convolution in image processing? (2/3)

2D discrete convolution

The convolution of a 2D filter K of size $2N+1 \times 2N+1$ $([-N,N] \times [-N,N])$ with an image I:

$$\overline{I}(i,j) = \sum_{l=-N}^{N} \sum_{p=-N}^{N} K(l,p) I(i-l, j-p) \tag{5}$$

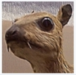
We denote convolution as $\overline{I} = K * I$.

- K is called the filter, kernel or mask.
- $K(0,0)$ is aligned with $I(i,j)$.

The output pixel's value is determined as a weighted sum of input pixel values.

What is a 2D discrete convolution in image processing? (3/3)

Depending on the kernel values, we can get different results:

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

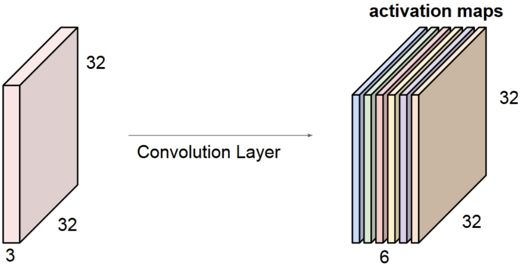
Convolution Layer

The **convolution operator** aims to extract features from the input image. It preserves the spatial relationship between pixels by learning image features using small chunk of input data (as a neuron would do in our visual cortex).

- Input: a 2D map
- Output: Convolved Feature or Activation Map or the Feature Map
- Parameters: a $N \times N$ kernel or filter (the same across all locations)
  - Example:
    1000×1000 images, 100 convolution filters, kernel size 10×10 $\Rightarrow 10 * 10 * 100 = 10k$ parameters to learn...
- Filters always extend the full depth of the input volume:
  - Example:
    $32 \times 32 \times 3$ images with $5 \times 5 \times 3 \Rightarrow$ 75 parameters to learn ($+1$ for the bias).

Here are the hyper-parameters:

- The depth: the number of filters we use for the convolution operation.
  Increasing the depth $\Rightarrow$ more feature maps are extracted.
  - Example:
    $32 \times 32 \times 3$ images with 6 filters $5 \times 5 \times 3 \Rightarrow 6 \times (75+1)$ parameters to learn.



with stride=1, pad=2

Adapted from Standford course http://cs231n.stanford.edu

Basic building operators of CNNs

Convolution layer (3/6)

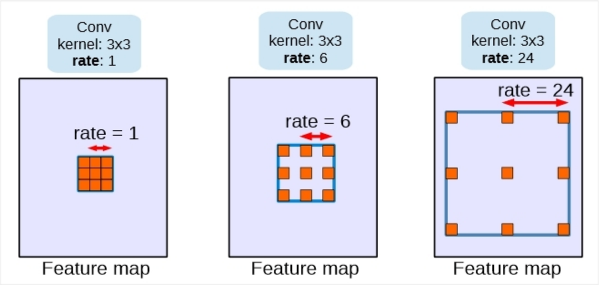
Here are the hyper-parameters:

- The stride: the stride is the number of pixels by which we slide our filter matrix over the input matrix.
  - stride=1, no decimation
  - stride=2, decimation of 2...
- Padding: padding pads the input volume around the border (zero padding).
  - if stride=1, we can pad the volume with $\frac{N-1}{2}$ to be sure to keep the same output resolution as the input one (N is the size of the convolutional kernel)
- Causal or not: a convolution is called causal if the filter output does not depend on future inputs (e.g. audio (Van Den Oord et al., 2016)).

Basic building operators of CNNs
Convolution layer (4/6)

➟ Dilatation rate (A trous convolution):

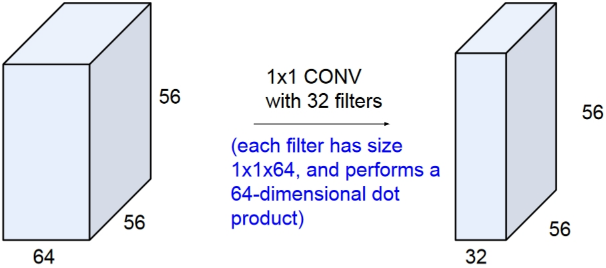
- Used to expand the receptive field without loss of resolution or coverage (Yu and Koltun, 2015)
- Multi-scale information without losing resolution (stride=1!!)



Extracted from (Chen et al., 2017)

➡ the particular case of the 1×1 convolution:

- use to reduce the dimension of the input volume (not the spatial dimension!)
- a 1×1 convolution with one layer produces only one layer in output, no matter the number of layer in input.



Adapted from Standford course http://cs231n.stanford.edu

➡ 3D convolution (e.g. spatial convolution over volumes):

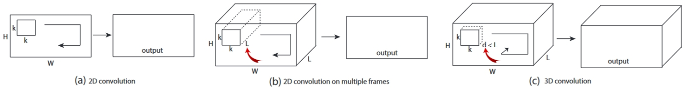


Figure 1. **2D and 3D convolution operations**. a) Applying 2D convolution on an image results in an image. b) Applying 2D convolution on a video volume (multiple frames as multiple channels) also results in an image. c) Applying 3D convolution on a video volume results in another volume, preserving temporal information of the input signal.

Adapted from (Tran et al., 2015)

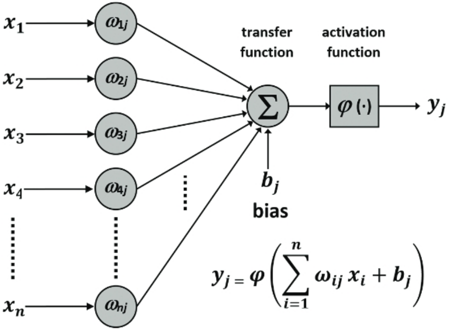
- We can specify the strides of the convolution along each spatial dimension (spatial ($\times 2$), temporal);
- The kernel size is defined by the depth, height and width of the 3D convolution window.

➡ In (Tran et al., 2015), they showed that the C3D network can model appearance and motion information simultaneously!!

➡ Video saliency (Ding and Fang, 2017), audio-visual saliency (Tavakoli et al., 2019), trajectory, motion...

Activation layer

The **activation operator** aims to simulate the firing rate of the cell.



Adapted from (Álvarez et al., 2017)

Basic building operators of CNNs
Activation layer(2/7)

- Sigmoid: $\varphi(x) = \frac{1}{1+e^{-x}}$
- Tanh: $\varphi(x) = tanh(x)$
- Relu (Krizhevsky et al., 2012): $\varphi(x) = \max(0, x)$
- Leaky-Relu (Maas et al., 2013):

$$\varphi(x) = \max(0.01 \times x, x)$$

- PRelu (Parametric Rectifier) (He et al., 2015):

$$\varphi(x) = \max(\alpha \times x, x)$$

- ELU (Exponential Linear Units) (Clevert et al., 2015):

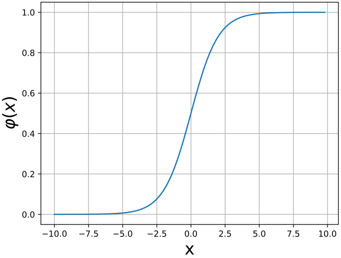
$$\varphi(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha\left(exp(x) - 1\right) & \text{if } x \leq 0. \end{cases}$$

- Swish (Ramachandran et al., 2017)(seems to be the best now):

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

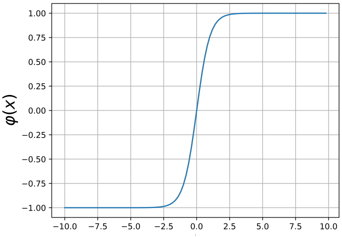
Basic building operators of CNNs
Activation layer(3/7)

- Sigmoid: $\varphi(x) = \frac{1}{1+e^{-x}}$



- Output numbers in the range $[0, 1]$
- ✗ Vanishing gradients, i.e. kills gradients when saturated
- ✗ Outputs are not zero-centered
- ✗ Exp() is computationally expensive

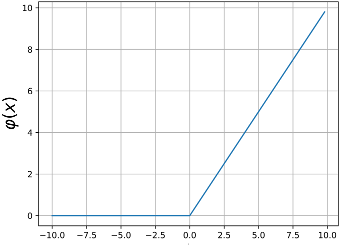
- tanh: $\varphi(x) = tanh(x)$



- Output numbers in the range $[0, 1]$
- ✗ Vanishing gradients, i.e. kills gradients when saturated
- ✓ Outputs are zero-centered

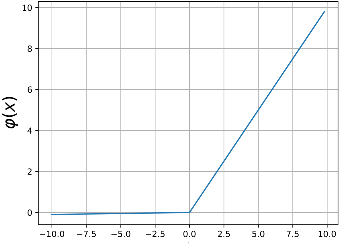
Basic building operators of CNNs
Activation layer(5/7)

- Relu: $\varphi(x) = \max(0, x)$



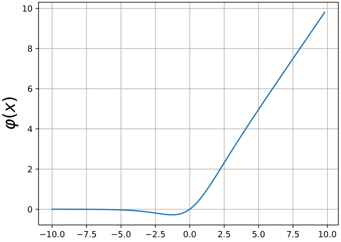
- ✓ No saturation for $x > 0$
- ✓ Very simple, and computationally efficient
- ✓ Converge faster than sigmoid and tanh
- ✗ No zero-centered

- Leaky Relu: $\varphi(x) = \max(0.01 \times x, x)$



- ✓ No saturation for $x > 0$ and small positive slope, when $x \leq 0$
- ✓ Very simple, and computationally efficient
- ✓ Converge faster than sigmoid and tanh
- ✗ No zero-centered

- Swish: $\varphi(x) = \frac{x}{1+e^{-x}}$



- A subtle mixture between sigmoid, and leaky-Relu

Pooling Layer

The **pooling operator** aims to map a subregion of the input into a single number in order to reduce the size of the representation (to speed up the computation) and to make features detection more robust.

Two types of pooling operators are widely used:

- max pooling maps a subregion to its maximum value;
- average pooling maps a subregion to its maximum value

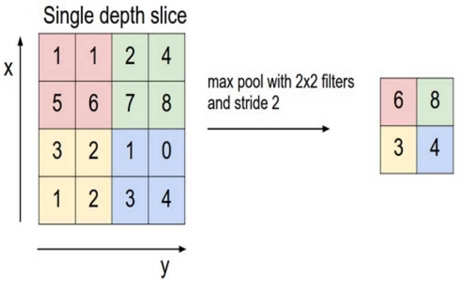
```
MaxPooling2D(pool_size=(2, 2), strides=None,
    padding='valid', data_format=None)
```

- global average pooling.

No parameters to learn!!

Here are the hyper-parameters:

- **Kernel size**: the size of the subregion of the input that will be mapped to a single value;
- **The stride**: same as the convolutional layer.



Max pooling is the most used: if a specific feature is in the original input volume, there will be a high activation value, the max pooling can catch it!

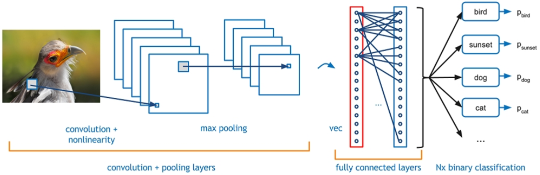
- 2D Global Average Pooling:
  - It consists in taking an average of every incoming feature map;
  - It is therefore independent of the size of the input image;
  - Reduce the number of parameters (cf. fully connected).

For example, with a $15 \times 15 \times 8$ incoming tensor of feature maps, we take the average of each 15×15 matrix slice, giving an 8 dimensional vector.

- Same concept for 2D Global Max Pooling.

Fully-Connected Layer

In a fully connected layer, each neuron is connected to every neuron in the previous layer, and each connection has it's own weight. This is a totally general purpose connection pattern and makes no assumptions about the features in the data. It's also very expensive in terms of memory (weights) and computation (connections).

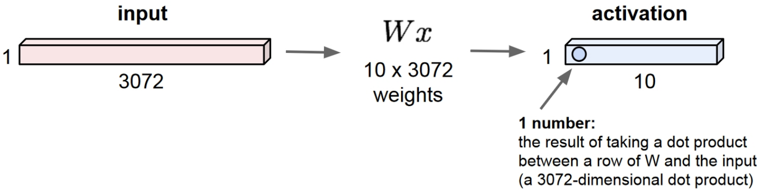


From https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

can hence be computed with a matrix multiplication

Basic building operators of CNNs
Fully-Connected Layer (2/3)

For instance, if we have an input image $32 \times 32 \times 3$, and a fully-connected layer of 10 outputs:



- Each neuron looks at the full input volume.
- There is no feature extraction!!

Extracted from http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

```
model = Sequential()
# Dense(64) is a fully-connected layer with 64
    hidden units.
# in the first layer, you must specify the expected
    input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64,activation='relu',input_dim=20))
```

```
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Loss functions (1/1)

The loss function is a method for evaluating how well your algorithm models your datasets:

- if the prediction is wrong, the loss function will output a high number;
- if the prediction is correct, the loss function will output a low number.

Loss functions for classification/regression and for dense prediction

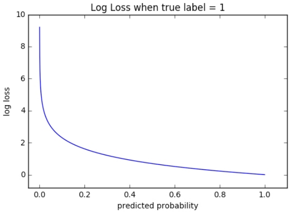
Loss functions for classification / regression prediction (1/5)

- Mean Squared Error, or L_2 loss function:

$$\mathcal{L}(y, \hat{y})_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where, y_i and $\hat{y}_i$ correspond to the actual value and the predicted value of the i^{th} observation, respectively. N is the number of observation.

- Cross-entropy or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.



http://wiki.fast.ai/index.php/Log_Loss

Loss functions for classification / regression prediction (2/5)

- Cross-entropy or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.
  - In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as:

$$\mathcal{L}(y,\hat{y})_{BCE} = -\left(y\log\hat{y} + (1-y)\log(1-\hat{y})\right) \tag{6}$$

    where, y is 0 or 1, indicating the class, and $\hat{y}$ is the predicted class.

Example:

- if the class to predict is $y=1$, and the prediction is $\hat{y}=1/4$, the loss value is $-\log 1/4 = 2\log 2$.
- if the class to predict is $y=1$ and the prediction is $\hat{y}=1/8$, the loss value is $-\log 1/8 = 3\log 2$.
- if the class to predict is $y=0$ and the prediction is $\hat{y}=1/4$, the loss value is $-\log 3/4$.

Loss functions for classification / regression prediction (3/5)

- Cross-entropy or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.
  - In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as:

$$\mathcal{L}(y, \hat{y})_{BCE} = -(y \log \hat{y} + (1-y)\log(1-\hat{y})) \tag{7}$$

Pyhton code example:

```
#-------------------------
# yHat is the prediction
# y is the label (0,1)
#-------------------------
def CrossEntropy(yHat, y):
    if y == 1:
        return -log(yHat)
    else:
        return -log(1 - yHat)
```

Loss functions for classification / regression prediction (4/5)

➟ Cross-entropy or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.

- When the number of classes M is superior to 2 (i.e. multiclass classification), cross-entropy can be calculated as the sum of log loss values for each class:

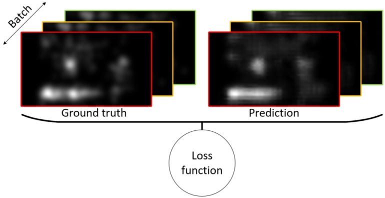
$$\mathcal{L}(y, \hat{y})_{BCE} = -\sum_{c=1}^{M} y_c \log \hat{y_c} \tag{8}$$

where, c indicates the index of classes ($c \in \{1, ..., M\}$) (for one observation).

Loss functions for classification / regression prediction (5/5)


```python
def cross_entropy(predictions, targets, epsilon=1e-12):
    """
    Computes cross entropy between targets and
        predictions.
    Input: predictions (N, k) ndarray
           targets (N, k) ndarray
    Returns: scalar
    """
    predictions = np.clip(predictions, epsilon, 1. -
        epsilon)
    N = predictions.shape[0]
    return -np.sum(targets*np.log(predictions))/N

targets = np.array([[0,0,0,1], [0,0,0,1]])
predictions = np.array([[0.25,0.25,0.25,0.25],
                        [0.01,0.01,0.01,0.96]])

#Correct answer 0.71355817782
x = cross_entropy(predictions, targets)
```

Loss functions for dense prediction (1/4)



- Loss function $\mathcal{L}(S, \hat{S})$ for a dense prediction between S and $\hat{S}$ map

- Taxonomy of loss functions:
  - Pixel-based loss functions
  - Probability distribution-based loss functions
  - Task-dependent loss functions (e.g. saliency metrics)

Loss functions for dense prediction (2/4)

➡ Pixel-based loss functions ($S, \hat{S} \in [0,1]$):

$$\mathcal{L}(S,\hat{S})_{MSE} = \frac{1}{N}\sum_{j=1}^{N}(S_j - \hat{S}_j)^2$$

(He et al., 2018)

$$\mathcal{L}(S,\hat{S})_{EAD} = \frac{1}{N}\sum_{j=1}^{N}\Big(exp(|S_j - \hat{S}_j|) - 1\Big)$$

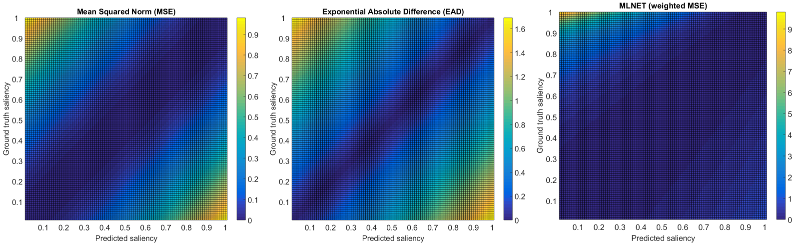
(Cornia et al., 2016)

$$\mathcal{L}(S,\hat{S})_{MLNET} = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{\alpha - S_j}(S_j - \hat{S}_j)^2, \alpha = 1.1$$

MSE: Mean Squared Error; EAD: Exponential Absolute Difference; MLNET: Weighted MSE

Loss functions for dense prediction (3/4)

➟ Pixel-based loss functions ($S, \hat{S} \in [0, 1]$):



From left to right: MSE, EAD, MLNET

MSE: Mean Squared Error; EAD: Exponential Absolute Difference; MLNET: Weighted MSE

Loss functions for dense prediction (4/4)

- Probability distribution-based loss functions ($\sum_i S_i = \sum_i \hat{S}_i = 1$):

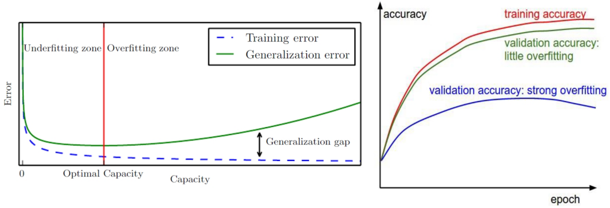
$$\mathcal{L}(S, \hat{S})_{Bhat} = -ln\left(\sum_{j=1}^{M} \sqrt{S_j \hat{S}_j}\right) \quad (9)$$

$$\mathcal{L}(S, \hat{S})_{KL} = \sum_{j=1}^{M} S_j log\left(\frac{S_j}{\hat{S}_j}\right) \quad (10)$$

Bhat: Bhattacharyya distance; KL: Kullback-Leibler divergence.

Training (1/3)

- Overfitting (network size, amount of data, gap between training and test performance (generalization))



Adapted from M. Tekalp, tutorial EUSIPCO 2018, *Deep Learning for image and video processing*.

To prevent overfitting:

- Weight-decay (L_1 decay, L_2 decay)
- Drop out

When the data set is too small:

- Pre-training on generic datasets;
- Data augmentation (Random crop, horizontal/vertical flip, rotations, synthetic data generation).

Training (2/3)

➠ Kernel initializers:

- Zeros, Ones, Constant
- Random Normal, Random Uniform: initialization with a normal (μ, σ and seed) / uniform distribution ($minval$, $maxval$ and seed);
- Le Cun Uniform (LeCun et al., 2012): initialization from a uniform distribution within $[-limit, limit]$ with $limit = \sqrt{\frac{3}{N}}$, N is the number of input channels of the layer.
- glorot_normal (Glorot and Bengio, 2010): initialization from a normal distribution centered on 0 with $\sigma = \sqrt{\frac{2}{N+M}}$, M is the number of output channels of the layer.

Many variants!

But all you need is a good init (Mishkin and Matas, 2015).

Not convinced by these initializers, make our own initializer!

Training (3/3)

Not convinced by these initializers, make our own initializer!

```
from keras import backend as K
def my_init(shape, dtype=None):
    return K.random_normal(shape, dtype=dtype)
model.add(Dense(64, kernel_initializer=my_init))
```

Or

Pretrained your network with synthetic or similar data (e.g. data augmentation), and fine-tuned it with real data...

Outline

Visual Geometry Group (VGG) network (1/6)

➠ CNN for image classification (Simonyan and Zisserman, 2014):

- Given an input image, VGG network aims to find object name in the image
- It can detect up to 1000 different objects
- It takes input image of size $224 \times 224 \times 3$ (RGB image)

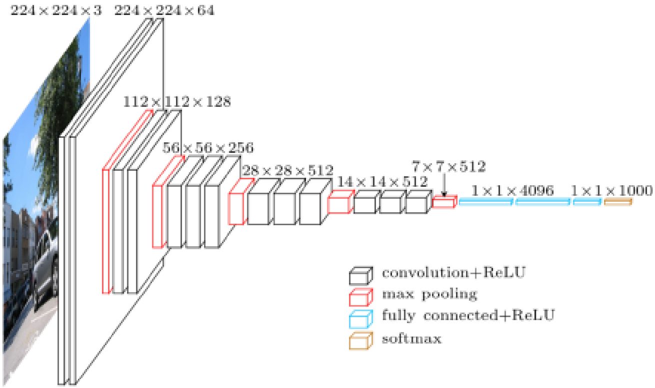
Built using:

- Convolutions layers (used only 3×3 size)
- Max pooling layers (used only 2×2 size)
- Fully connected layers at end
- Total 16 layers
- Trained with Imagenet, $\approx$ 16 Million images,1000 classes (Deng et al., 2009)

Hummm, 138 millions of parameters.

Visual Geometry Group (VGG) network (2/6)

➟ CNN for image classification:



Architecture of VGG16

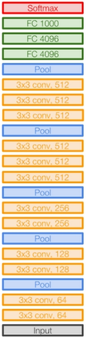
Visual Geometry Group (VGG) network (3/6)

VGG16 vs VGG19: the 16 and 19 stand for the number of weight layers in the network. VGG19 just has 3 more conv3 layers.

Visual Geometry Group (VGG) network (4/6)

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000



VGG16

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters

- Memory decreases with depth (most memory is in the first conv layers);
- Number of parameters increases with depth (most parameters are in FC).

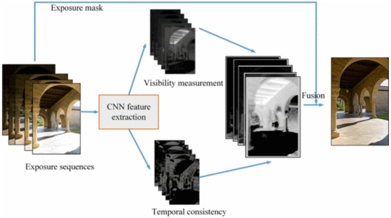
VGG-like convnet

```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD

# Generate dummy data
x_train = np.random.random((100, 100, 100, 3))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)), num_classes=10)
x_test = np.random.random((20, 100, 100, 3))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(20, 1)), num_classes=10)

model = Sequential()
# input: 100x100 images with 3 channels -> (100, 100, 3) tensors.
# this applies 32 convolution filters of size 3x3 each.
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd)

model.fit(x_train, y_train, batch_size=32, epochs=10)
score = model.evaluate(x_test, y_test, batch_size=32)
```

Visual Geometry Group (VGG) network (6/6)

➟ A number of applications with the deep features:

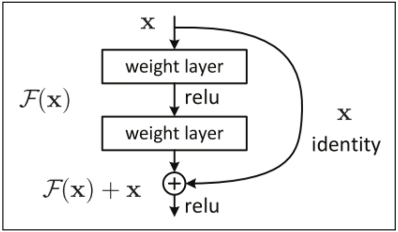
- Multi-Exposure Fusion with CNN features (Li and Zhang, 2018):



- Deep Features to Classify Skin Lesions (Kawahara et al., 2016).
- Image retrieval (Babenko and Lempitsky, 2015).
- Image saliency (Cornia et al., 2016, Kümmerer et al., 2014, 2016).

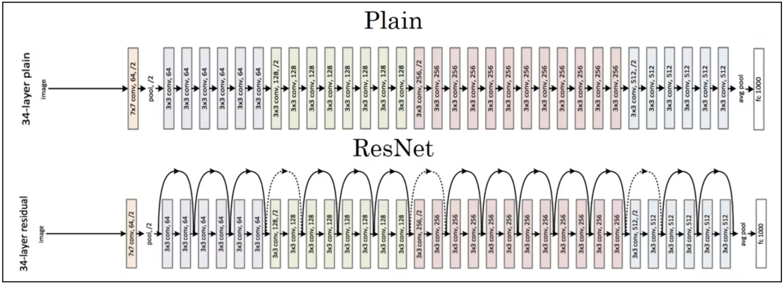
ResNet (1/2)

- Going deeper and deeper, but increasing network depth does not work by simply stacking layers together:
  - vanishing gradient problem;
  - too small gradient $\Rightarrow$ performance saturation.
- ResNet (He et al., 2016) (> 29000 citations...):
  - Skip connections or short cut connections;
  - Identity function, adding new layers do not hurt the ability to train the network.



ResNet (2/2)

➟ ResNet (He et al., 2016):



Wonderful explanations in 7 minutes:
https://www.youtube.com/watch?v=ZILIbUvp5lk

MobileNet (1/2)

➡ MobileNet (Howard et al., 2017):

- efficient models for mobile and embedded vision applications;
- light weight deep neural networks;
- main novelty is based on a depthwise Separable Convolutions=depthwise + pointwise convolution.

If we assume an image with 3 channels and a convolution kernel of 5×5 size, an image $M \times N$ and K outputs:

- For a classic 2d convolution: we actually do $5 \times 5 \times 3 \times M \times N \times K = 75 \times M \times N \times K$ multiplications.
- Depthwise convolution: Instead of 1 kernel, we use 3 kernels of shape $5 \times 5 \times 3 \times M \times N$.
- Pointwise Convolution: To get the final map, we use 1D convolution of size $1 \times 1 \times 3$ to mix together the different channels. $3 \times M \times N \times K$ multiplications.

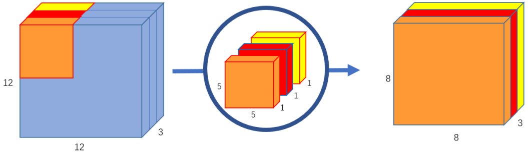
The number of multiplication significantly decreases!!

$$5 \times 5 \times 3 \times M \times N \times K \text{ » } 5 \times 5 \times 3 \times M \times N + 3 \times M \times N \times K$$

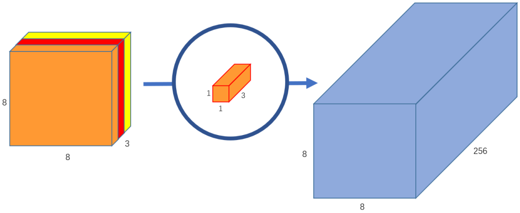
MobileNet (2/2)

- MobileNet (Howard et al., 2017):

Depthwise convolution



Pointwise convolution with 256 kernels



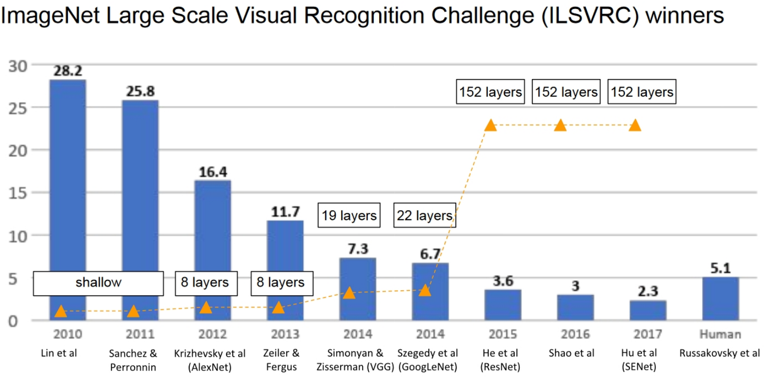
From https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

ILSVRC (1/2)

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale since 2010.

- Object localization for 1000 categories.
- Object detection for 200 fully labeled categories.
- Object detection from video for 30 fully labeled categories.

ILSVRC (2/2)



- 2010-2014: Shallow and deeper network
- 2012: Winner = CNN-based network
- 2014: Depth Revolution

References

Daniel Álvarez, Ana Cerezo-Hernández, Graciela López-Muñiz, Tania Álvaro-De Castro, Tomás Ruiz-Albi, Roberto Hornero, and Félix del Campo. Usefulness of artificial neural networks in the diagnosis and treatment of sleep apnea-hypopnea syndrome. In *Sleep Apnea-Recent Updates*. InTech, 2017.

Artem Babenko and Victor Lempitsky. Aggregating local deep features for image retrieval. In *Proceedings of the IEEE international conference on computer vision*, pages 1269–1277, 2015.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Marcella Cornia, Lorenzo Baraldi, Giuseppe Serra, and Rita Cucchiara. A Deep Multi-Level Network for Saliency Prediction. In *International Conference on Pattern Recognition (ICPR)*, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

Guanqun Ding and Yuming Fang. Video saliency detection by 3d convolutional neural networks. In *International Forum on Digital TV and Wireless Multimedia Communications*, pages 245–254. Springer, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Sen He, Nicolas Pugeault, Yang Mi, and Ali Borji. What catches the eye? visualizing and understanding deep saliency models. *arXiv preprint arXiv:1803.05753*, 2018.

Seyed Hamid Hosseini and Mahdi Samanipour. Prediction of final concentrate grade using artificial neural networks from gol-e-gohar iron ore plant. *American Journal of Mining and Metallurgy*, 3(3):58–62, 2015.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Jeremy Kawahara, Aicha BenTaieb, and Ghassan Hamarneh. Deep features to classify skin lesions. In *Biomedical Imaging (ISBI), 2016 IEEE 13th International Symposium on*, pages 1397–1400. IEEE, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Matthias Kümmerer, Lucas Theis, and Matthias Bethge. Deep gaze i: Boosting saliency prediction with feature maps trained on imagenet. *arXiv preprint arXiv:1411.1045*, 2014.

Matthias Kümmerer, Thomas SA Wallis, and Matthias Bethge. Deepgaze ii: Reading fixations from deep features trained on object recognition. *arXiv preprint arXiv:1610.01563*, 2016.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Hui Li and Lei Zhang. Multi-exposure fusion with cnn features. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 1723–1727. IEEE, 2018.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

Hamed R Tavakoli, Ali Borji, Esa Rahtu, and Juho Kannala. Dave: A deep audio-visual embedding for dynamic saliency prediction. *arXiv preprint arXiv:1905.10693*, 2019.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.