1 Convolution Neural Network step-by-step

1.1 Pooling Layer

1.1.1 Local Down-Pooling Layers

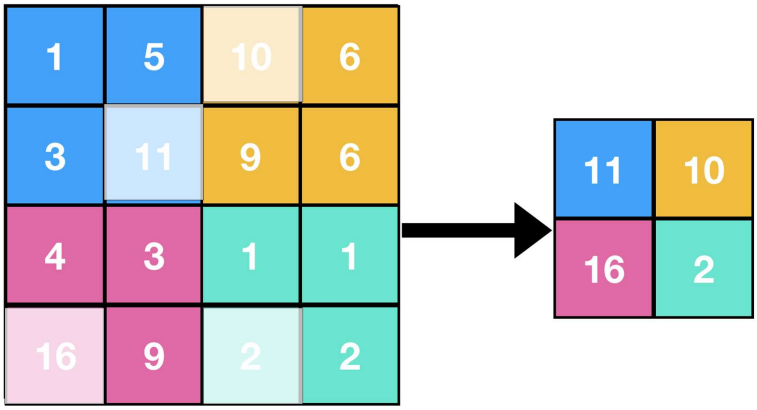
The main idea of the **pooling (down-sample)** layer is to reduce the feature map dimension without loose of information.

This idea is based on the assumption that each feature of the feature map lay in a group of several neighbor pixels.

Thus we can reduce image size by selection on of the pixel group.

The most popular pooling technique is the **max-pooling**.

> *The main idea of maxpooling is that most intensive (most bright) pixel contain the most amount of information.*
> The Max Pooling also performs a Noise Suppression (due to assumption that then more intensity then less sensitive noise, artifacts and other random effect influence).
> In other words maxpooling allow to make model more invariant (robust) to certain distortions.



The advantages of pooling

- enhance transformation invariance and robust
- reduce parameters
- faster computation
- avoid overfiting

The effect of max-pooling size reduction can be expressed as

$$W^{out} = (W^{in} - F)/S + 1;$$
$$H^{out} = (H^{in} - F)/S + 1;$$
$$D^{out} = D^{in},$$

where:

- W^{out}, H^{out} and D^{out} are the dimensions of output;
- W^{in}, H^{in} and D^{in} are the dimensions of input;
- F and S are the kernel/filter size F and stride S.

It can be distinguished a several types of pooling layers, such as

- **Max Pooling**:

  maximum value of each kernel

  $$\tilde{r} = \max[r_{ij}],$$

  where

  - $\tilde{r}$ is the output of pooling layer.
  - r_{ij} is the input region.

  Most popular type of pooling.

  Other variants are

  Median Pooling

  $$\tilde{r} = \text{sort}[r_{ij}][\text{center value}].$$

  k-Max Pooling: the k-most intensive values of each kernel are captured

  $$\tilde{r} = \text{sort}[r_{ij}][:k].$$

  Min Pooling:

  $$\tilde{r} = \min r_{ij},$$

- **Average Pooling(or mean-pooling)**:

  The average value of the kernel is calculated,

  $$\tilde{r} = \frac{1}{size(r_{ij})} \sum_{ij} r_{ij},$$

less robust then maximum pooling, but also more sensitive for features.

Other variations of Average Pooling are:

L2 Pooling : the L2 or the Frobenius norm is calculated as,

$$\tilde{r} = \frac{1}{size(r_{ij})}\sqrt{\sum_{ij} r_{ij}^2},$$

Lp norm (Generalized Mean Pooling, GeM) pooling is calculated as,

$$\tilde{r} = \frac{1}{size(r_{ij})}(\sum_{ij} r_{ij}^p)^{1/p},$$

less robust then maximum pooling, but also more sensitive for features (for inf or high degree same as max pooling).

- **Mixed Pooling**:

  the combination of a several types

  for L1+max case

$$\tilde{r} = \lambda \max r_{ij} + \frac{1-\lambda}{size(r_{ij})}\sum_{ij} r_{ij},$$

  where λ is the some hyperparameter.

- **Soft Pooling** (stochastic pooling):

$$\tilde{r} = \sum_{ij} r_{ij} p_{ij}$$

where $p_{ij} = e^{r_{ij}} / \sum_{ij} e^{r_{ij}}$.

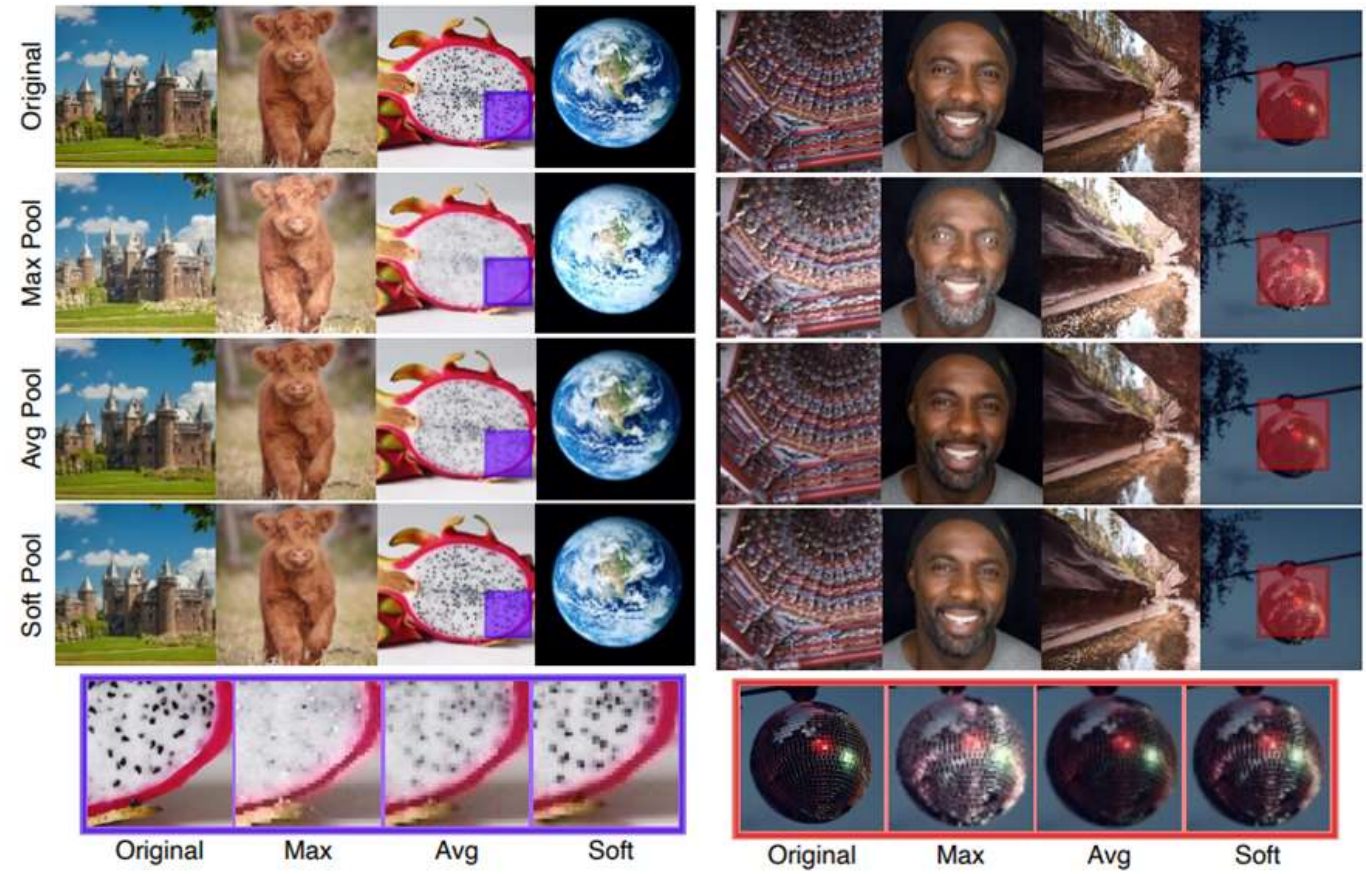
Other variant is **random pooling**, where is random value of multinomial distribution p_{ij} is taken.

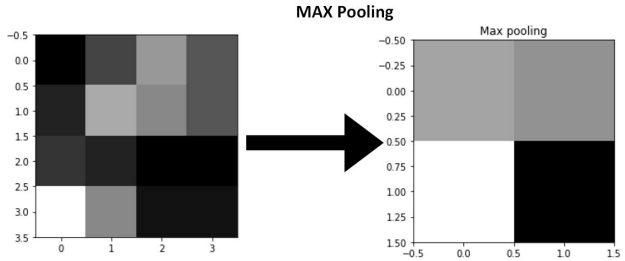
- **Local Importance Pooling (adaptive pooling)**:

$$\tilde{r} = \frac{1}{size(r_{ij})} \sum_{ij} r_{ij} w_{ij},$$

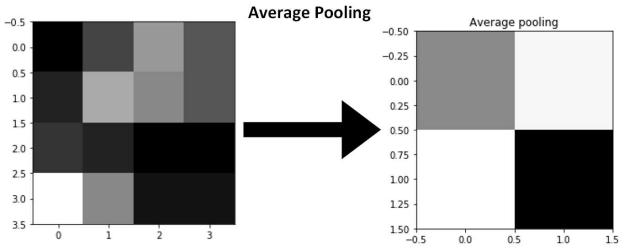
where $w_{ij} = e^{G(r_{ij})} / \sum_{ij} e^{G(r_{ij})}$,

and $G(r_{ij})$ is the pretrained Multilayer Perceptron (or other neural network), need for most important feature extraction.



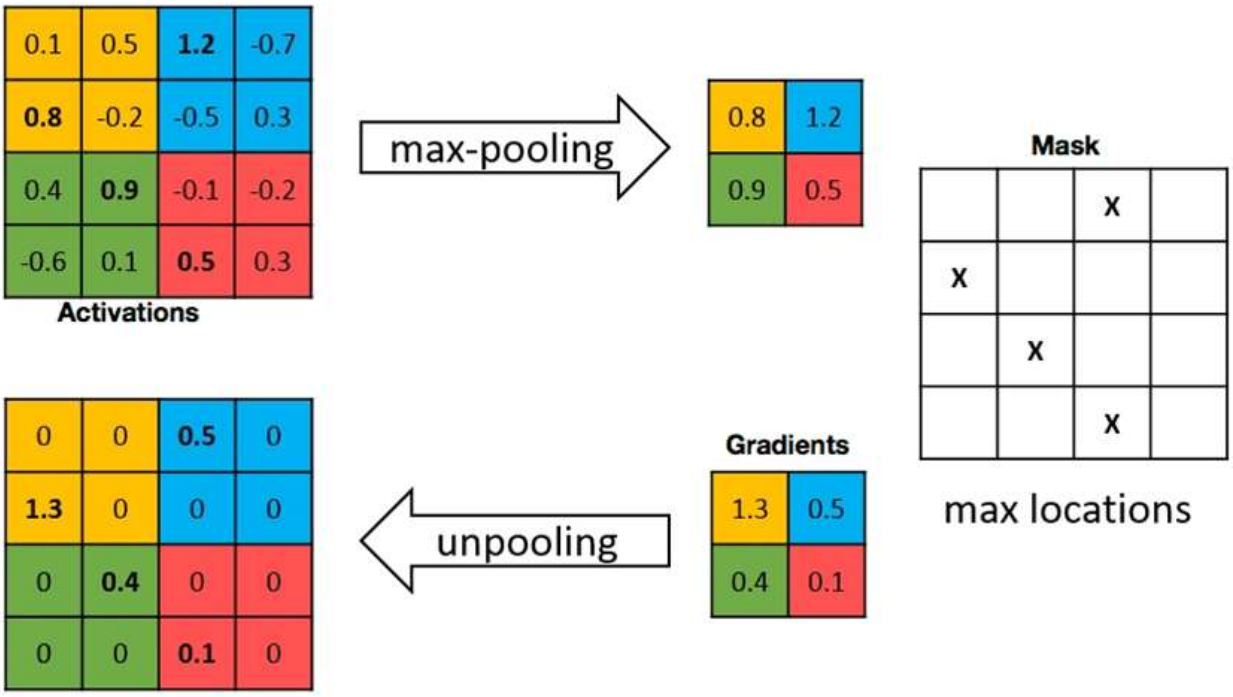


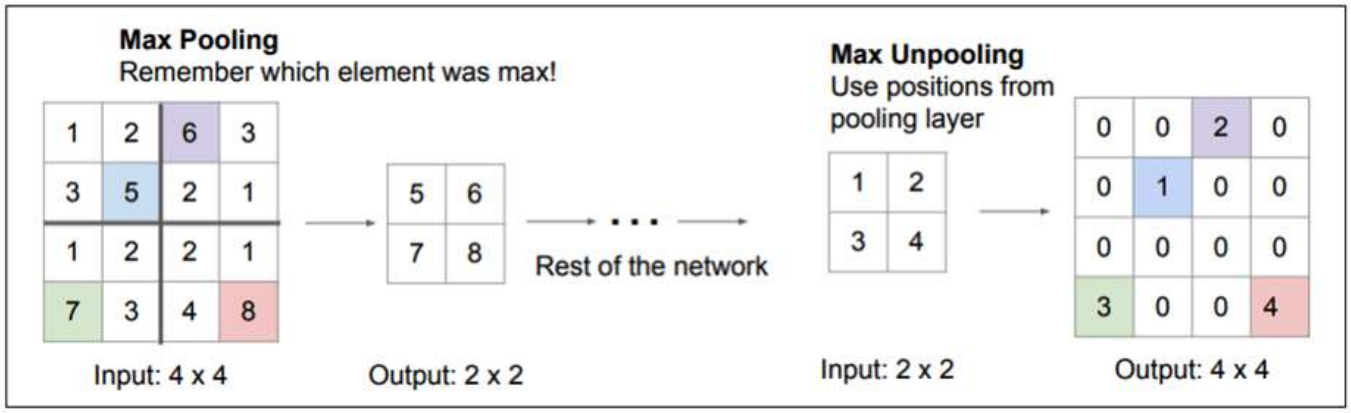
Note - The greater the value, the brighter the block.



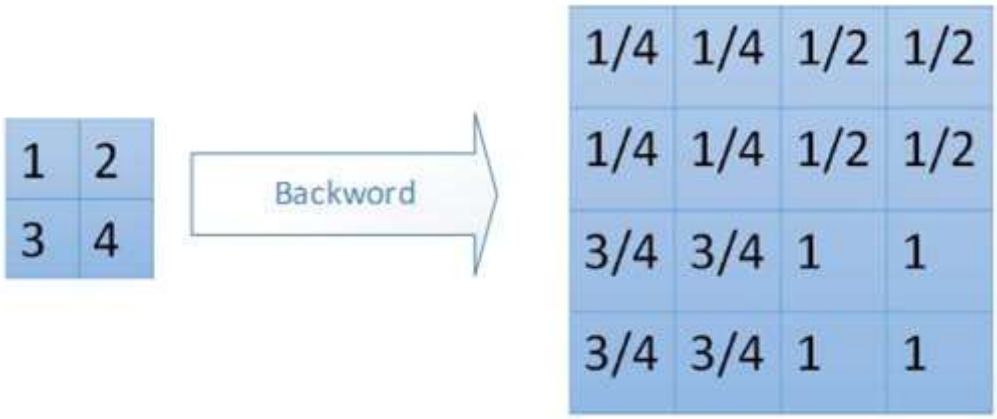
Note - The greater the value, the brighter the block.

Clarifications to the forward and backward-propagation through the max-pooling.

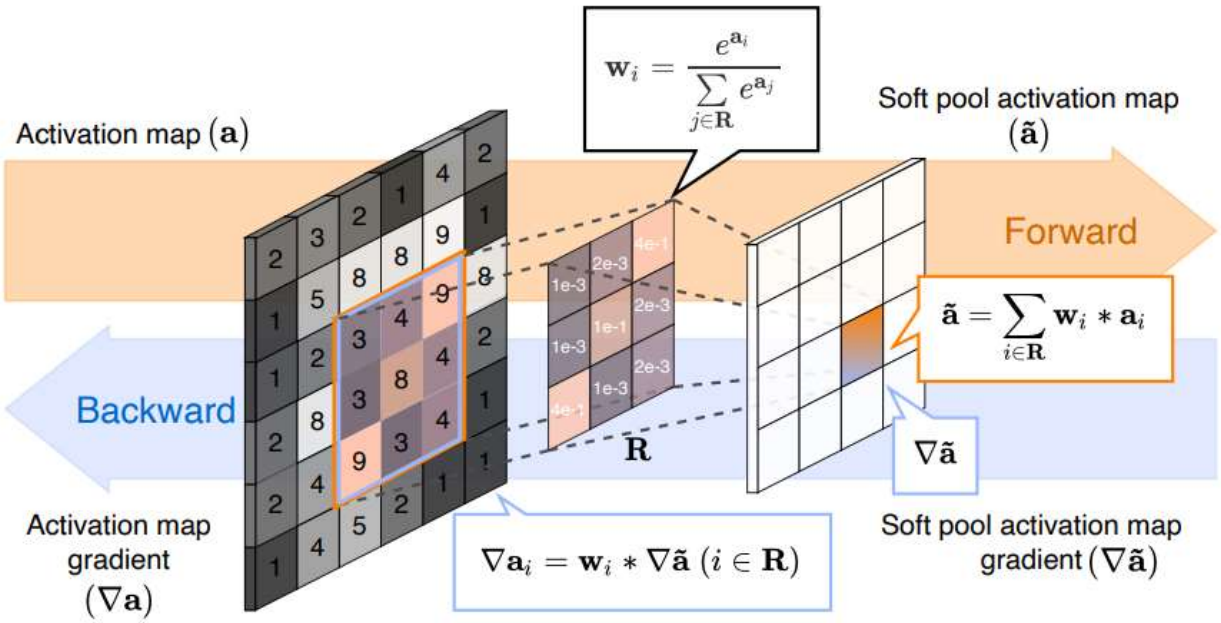




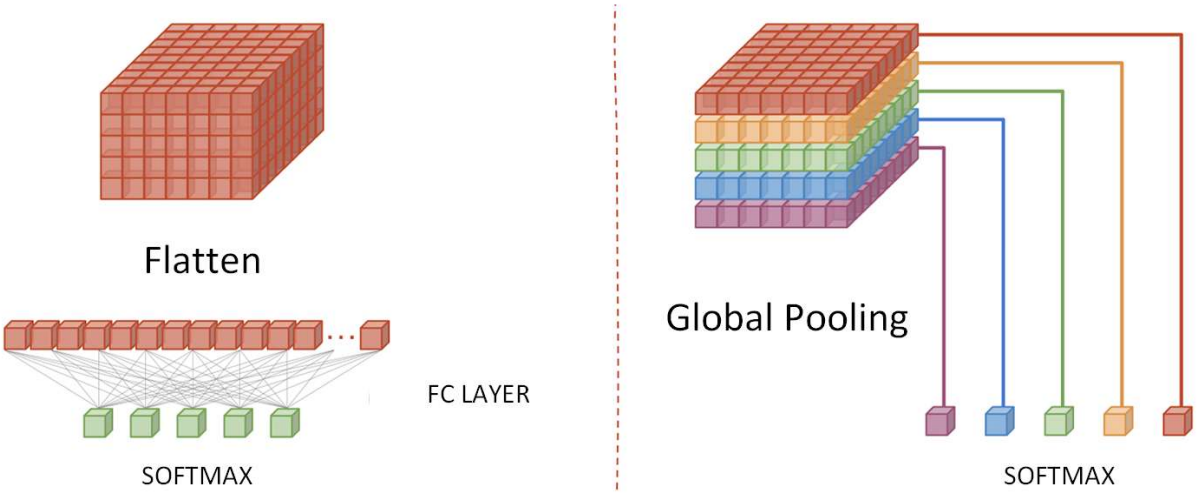
Clarifications to the backward-propagation through the average-pooling.



Commentaries to the forward and backward-propagation through the Soft Pooling.



1.1.2 Global Pooling Layers



The conventional way to transform feature maps to the **flatten** vector (flattening) is to using `flatten` (or `ravel`, `vect` and e.t.c.) operation which is consists in :

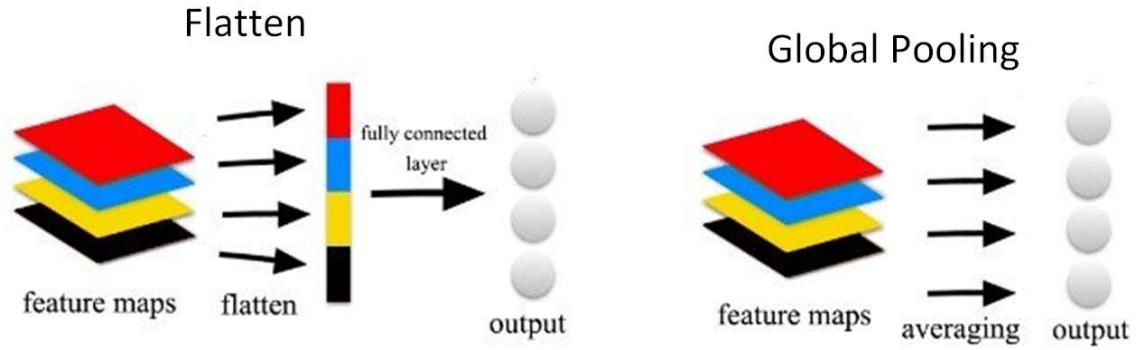
- split each 2d feature map to vectors such as each of columns or rows,
- concatenate vectors for one 2d feature map,
- concatenate vectors for all feature map to the big size vector.

The main drawbacks of this approach are:

- a lot of parameters (for the farther full-connected layer),
- full-connected layer size depends on the number of inputs

  thus you need to use only images with predefined size.

As the alternative to the flattening layer a global pooling layers can be considered. The main advantage of the replacement of flatten to Global Down-Pooling is reducing the requirement to the input images size to such that each MAP will be at least one pixel (size(MAP) ≥ 1.



There can be distinguished the following Global Pooling types as:

- **Global Average Pooling (GAP)**.

  For each 2d feature map:

$$\tilde{r}_c = \frac{1}{\text{size(MAP)}} \sum_{ij \in \text{MAP}} MAP_c,$$

where c is the channel of feature maps (MAP_c).

This technique is one of the most popular.

- **Global Max-Pooling**.

  For each 2d feature map:

  $$\tilde{r}_c = \max(MAP_c),$$

  not-popular due to bed in back-propagation.

- **Global Covariance Pooling**.

  For each 2d feature map:

  $$\tilde{r}_c = \frac{1}{n} \sum_{i=1}^{n} (X_c \cdot X_c^T),$$

  where $X_c = [x_{c1}, \ldots, x_{cn}] = \text{vec}(\text{MAP}_c)$, n is vec size.

  In more complex case use

  $$\tilde{R} = [\tilde{r}_1, \ldots, \tilde{r}_c]^T = U \Lambda^\alpha V$$

  , where $U \Lambda V$ is the **SVD** decomposition of R, and Λ is the singular values matrix, as a rule $\alpha = 0.5$

  It is also possible to use transformation $\Lambda \rightarrow log(\Lambda)$,

  and to cut Λ such as: $sort[\Lambda][:k]$ for reducing near-zero values.

- **Spatial Pyramid Pooling (SPP)**.

  For each 2d feature map:

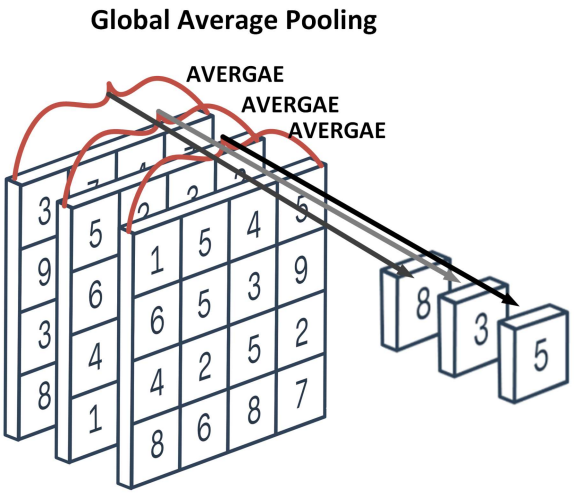
  $$\tilde{r}_c = vec\{\tilde{r}_1, \ldots, \tilde{r}_n\},$$

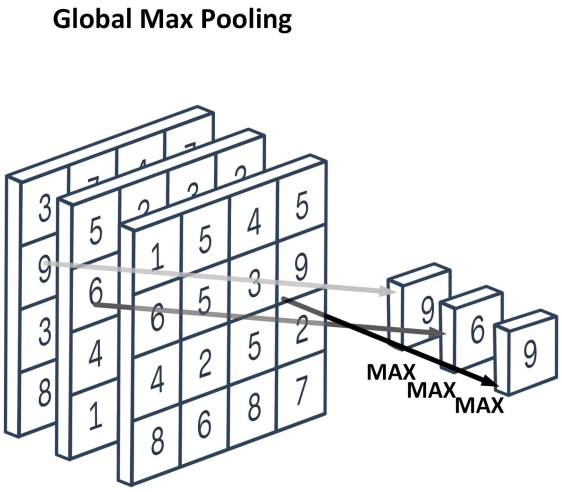
  where n is the required number of outputs; $\tilde{r}_1$ is the maxpooling output of the region size MAP_c/n i.e.

  - divide MAP_c into n regions
  - obtain $\tilde{r}_i$ as maxpooling over region i.
  - concatenate all pooled values into output vector.

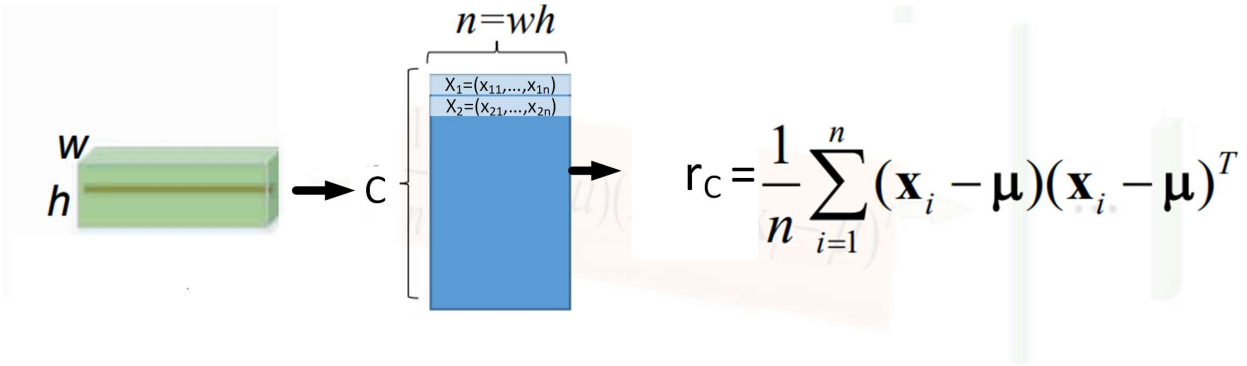
  Worthy replacement of Global maxpooling and flatten layers.

Note Actually, each type of conventional can be extended to the Global Layer Pooling or Spatial Pyramid Pooling.

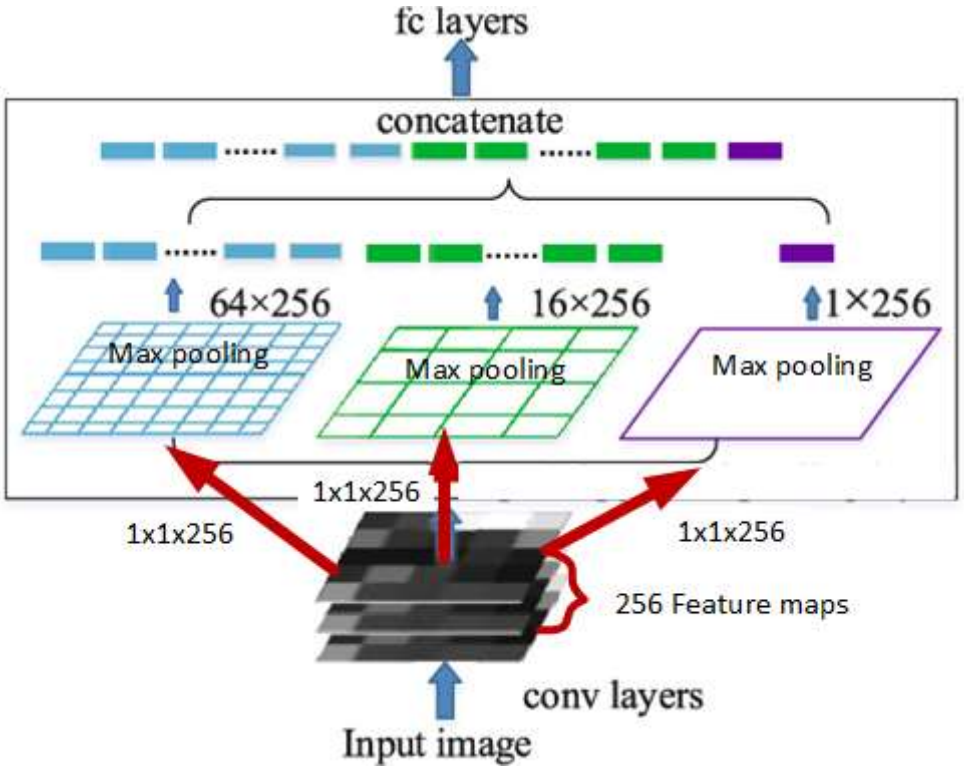




Global Covariance Pooling



Spatial Pyramid Pooling (SPP)



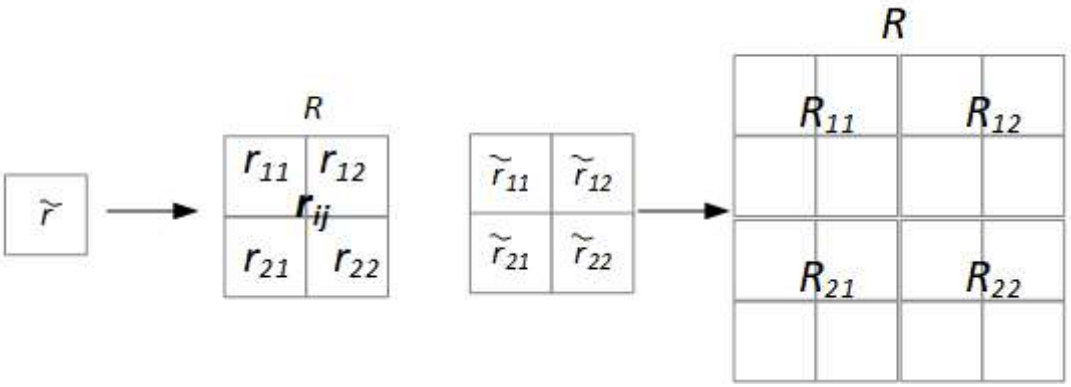
1.1.3 Upsampling Layer

Beside the pooling (or downsampling layer) in some cases it is need to make inverse operation - **upsampling**.

There are exist a lot of upsamplig techniques.

In the essence Upsampling consists in the task of mapping one input value $\tilde{r}$ into the region R

(or its elements r_{ij} such as $r_{ij} \in R$).



The tree main ways to upsampling are:

- **Transposed Convolution** ;
- **High-resolution Convolution**;
- **Classical up-sampling (pooling up-sampling, Interpolation up-sampling).**

Classical up-sampling (pooling upsampling).

- **Nearest Neighbors** Interpolation - fill full upsampled region with the same values as input.

$$r_{ij} = \tilde{r} \text{ for } ij \in R,$$

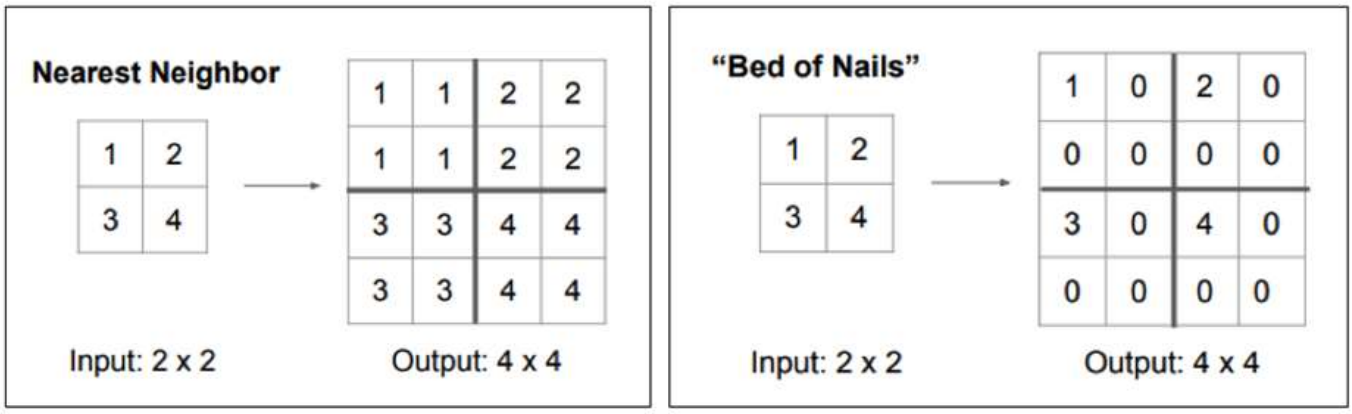
  where:

  - r_{ij} are the output elements of the upsampled region R;
  - $\tilde{r}$ is the input element, $\tilde{r} \rightarrow r_{ij} \in R$;

- **"Bed Of Nails"** - fill only first element of output region with input value.

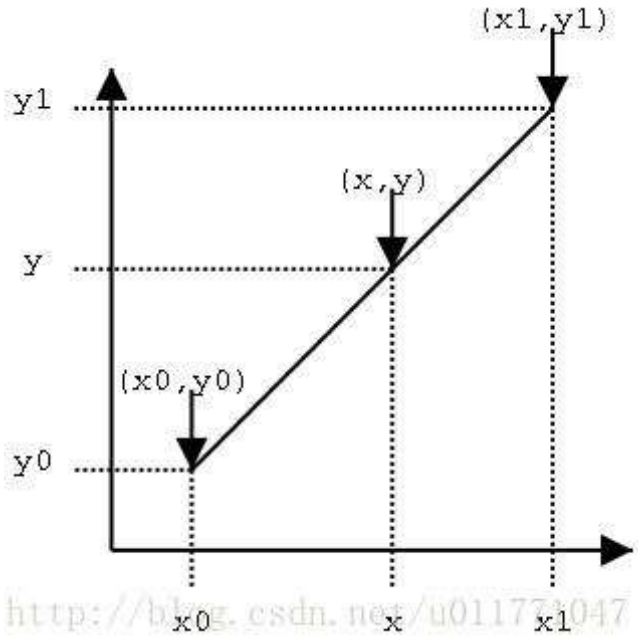
$$r_{ij} = \tilde{r} \text{ if } i = 0 \;\; j = 0 \in R; \;\; 0 \text{ otherwise.}$$

- **Bilinear Interpolation** - interpolate values using bilinear equation for 4 nearest points.
- **Bicubic Interpolation** - interpolate values using bicubick equation for 16 nearest points.
- **Low-pass filtration Interpolation** (**Lanczos interpolation**)- interpolate using "Bed Of Nails" and Low-Pass filtration.



The **Bilinear Interpolation** is the most popular way for upsampling.

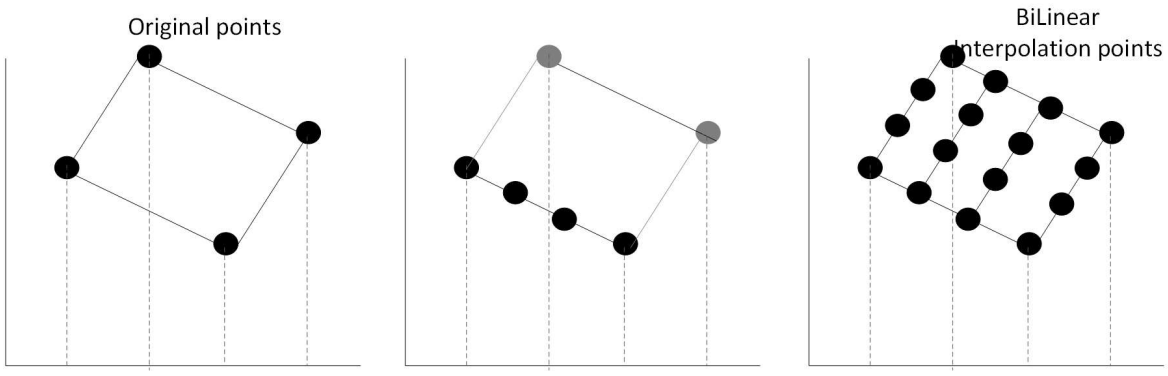
For understanding the bi-linear Interpolation let's first considering the linear case

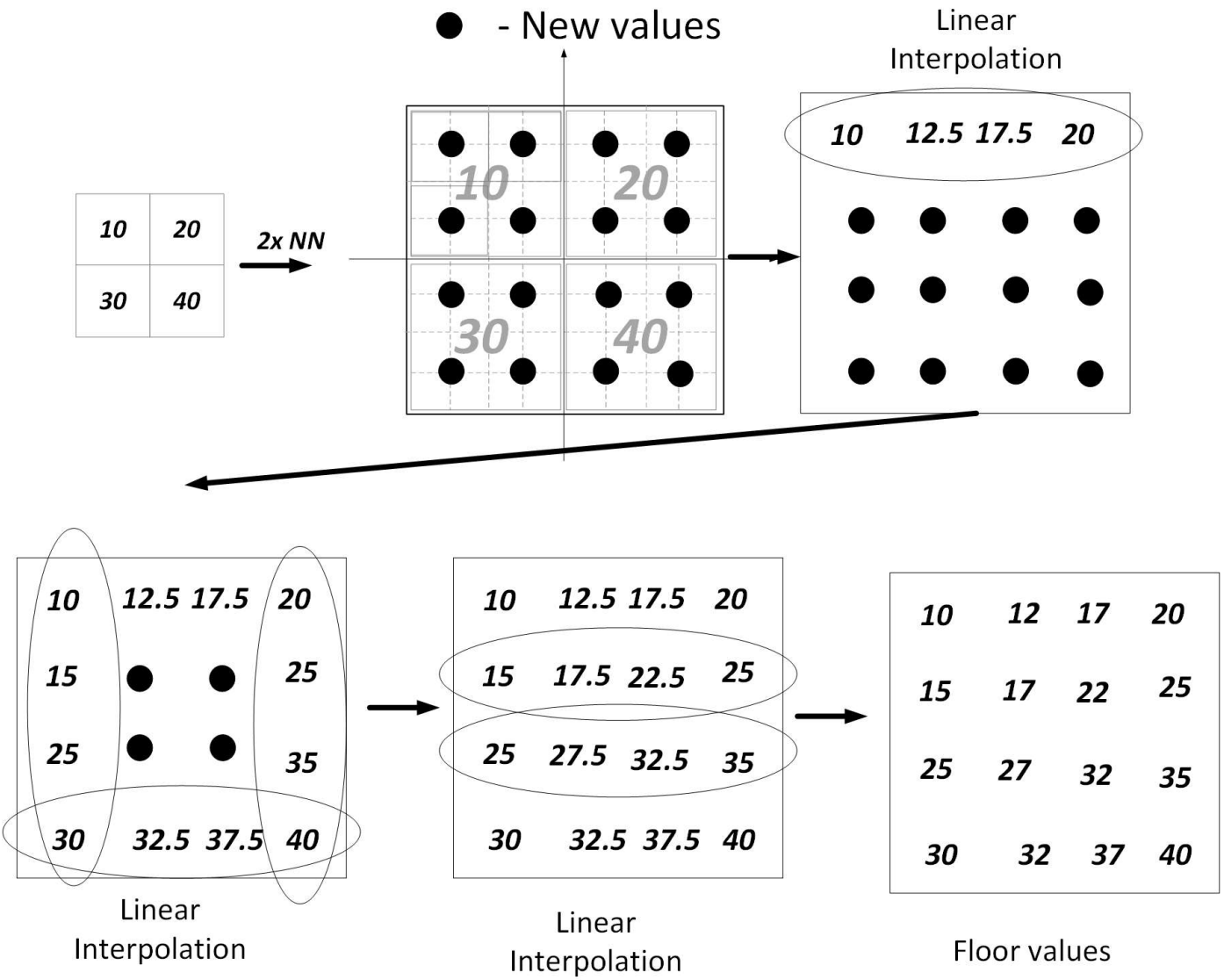


The bilinear case is the 2-dimensional interpolation:

The idea is to obtain new value as linear interpolation of the points values which we have.

- Represent region as 2d plot.
- use linear interpolation for edge horizontal axis.
- use linear interpolation for edge vertical axis.
- use linear interpolation for intermediate values using obtained vertical values.





The Bilinear interpolation can be obtained analytically:

for New point P=(x,y) and :
Old values points:

$$Q_{11} = (x_1, y_1) \quad Q_{12} = (x_2, y_1)$$
$$Q_{21} = (x_2, y_1) \quad Q_{22} = (x_2, y_2)$$

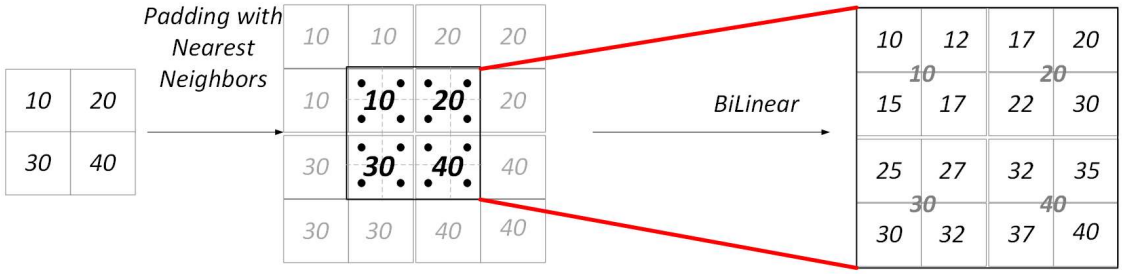
calculate new value: $f(P)$

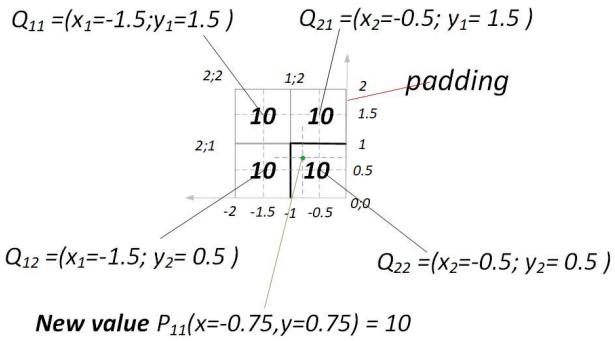
$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$f(P) = \frac{y_2 - x}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

where: R_1, R_2 auxiliary points.





New value $P_{11}(x=-0.75,y=0.75) = 10$

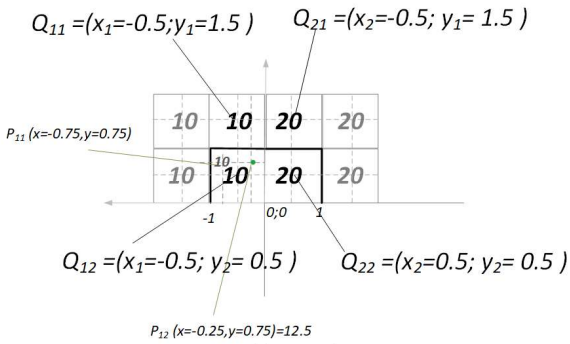
New value P_{11}

$P_{11}=(x=-0.75,y=0.75)$ ->

$f(R_1) = 10(-1.5+0.75)/(-1.5+0.5)+10(-0.75+0.5)/(-1.5+0.5) = 10$

$f(R_2) = 10(-1.5+0.75)/(-1.5+0.5)+10(-0.75+0.5)/(-1.5+0.5) = 10$

$F(P_{11})=10(1.5-0.75)/(1.5-0.5)+10(0.75-0.5)/(1.5-0.5) = 10$



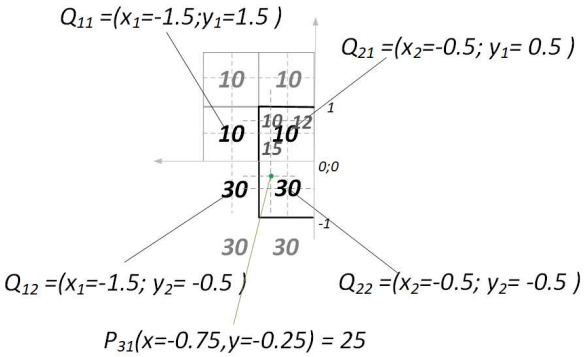
New value

$P_{12}=(x=-0.25,y=0.75)$ ->

$f(R_1) = 10(0.5+0.25)/(0.5+0.5)+20(-0.25+0.5)/(0.5+0.5) = 12.5$

$f(R_2) = 10(0.5+0.25)/(0.5+0.5)+20(-0.25+0.5)/(0.5+0.5) = 12.5$

$F(P)=12.5(0.5-0.75)/(0.5-1.5)+12.5(0.75-1.5)/(0.5-1.5) = 12.5$



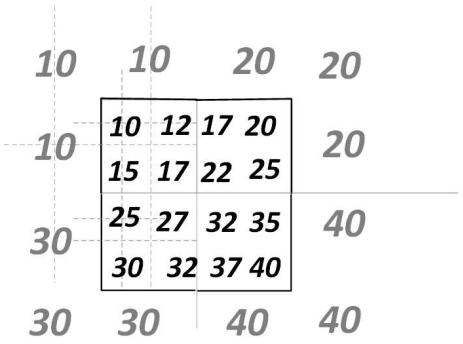
$P_{31}(x=-0.75,y=-0.25) = 25$

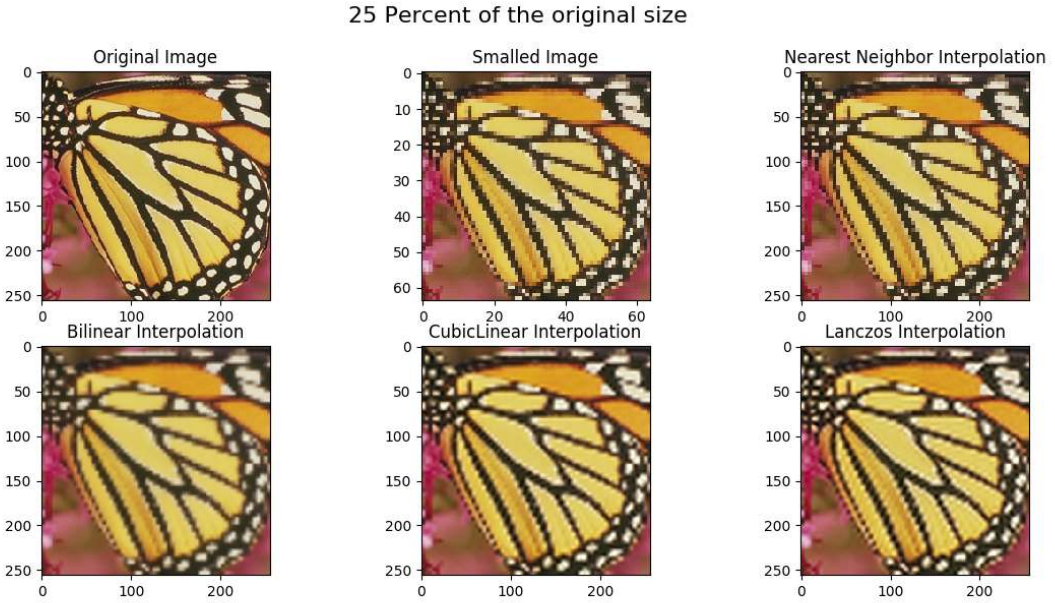
$P_{31}=(x=-0.75,y=-0.25)$ ->

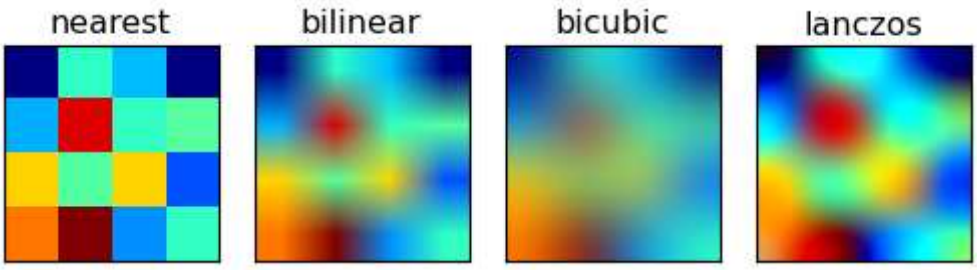
$f(R_1) = 10(-0.5+0.75)/(-1.5+0.5)+10(-0.75+1.5)/(-1.5+0.5) = 10$

$f(R_2) = 30(-0.5+0.75)/(-1.5+0.5)+10(-0.75+1.5)/(-1.5+0.5) = 30$

$F(P_{31})=10(-0.5+0.25)/(-0.5-0.5)+30(-0.25-0.5)/(-0.5-0.5) = 25$




25 Percent of the original size
Original Image
Smalled Image
Nearest Neighbor Interpolation
Bilinear Interpolation
CubicLinear Interpolation
Lanczos Interpolation


nearest
bilinear
bicubic
lanczos

In [ ]: