Maxout activation function and its variants

1) Introduction

In this report I will show motivation, the main concept and the results about possible application of a new activation function : maxout unit (MU). This new function promise benefits from a computation and quality point of view respect to other classical activation functions, like ReLu, adopted very often in many deep learning applications. Our test bed will be the Convolutional Neural Network (CNN) , in particular the Single Image Super Resolution (SISR) framework : we want to upscale a low resolution image in its high resolution version with the least error, respect the given high resolution image reference. Must be rememered that respect to Super Resolution(SR) problem applied to videos which use also time information, in Sigle Image Super Resolution we use only spatial information, and that lack of information make the problem a little harder. To achieve this task I test the network with Relu, maxout activation function and its variants.
Finally I compare the work's results and compare them in function of computational cost and respective performance measures like PSNR.
In the next section I will introduce Maxout activation function and their variants.

2) Maxout units

2.1) Maxout

The concept of maxout units is very simple : after the convolution step we want to obtain high nonlinearity among LR and HR images. In fact the simple use of max function provide nonlinearity. The main idea of the maxout is to divide the feature maps into two parts along channels and then compute the element-wise maximum among these two blocks.
In formula :

$$a^l = max(\mathbf{x}^l_1, \mathbf{x}^l_2)$$

In the program:

```
def maxout(self,X):
    x1, x2 = tf.split(X, 2, axis=-1)
    a = tf.math.maximum(x1,x2)
    return a
```

The particularity of this function is that after the activation, we have halved the number of parameters, unlike the ReLu function which has the number of parameters invariant.
Moreover this idea overcome the problems that can arise with Relu, for example with narrow or deep network too many values fall into negative and become zeros.
After the application of maxout the number of channel is halved.
Starting from this concept many other variants of the Maxout functions are been developed with a different set up, but preserving similar properties.

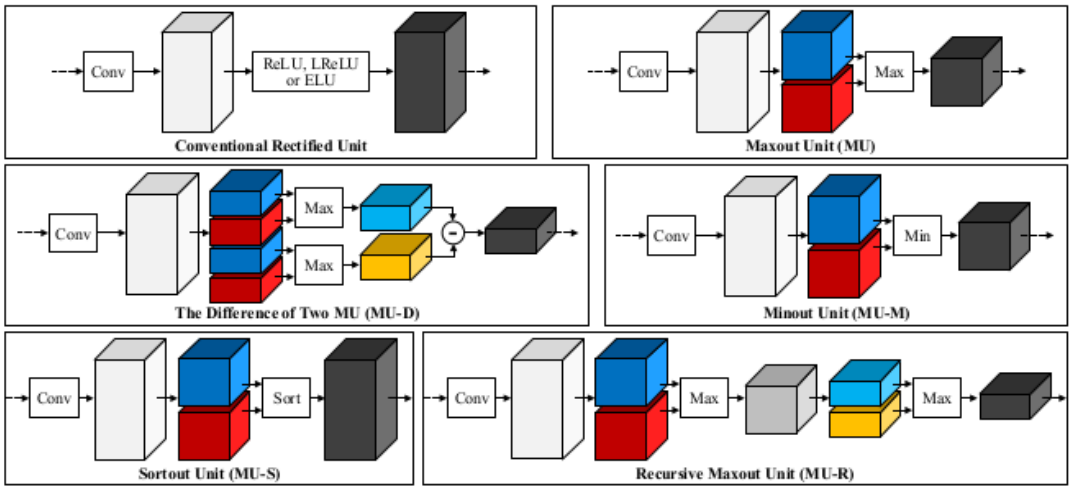


Fig.1 : Change in weight dimension for different activation function

2.2) Difference of two maxout

Here we introduce the difference of two maxout. Once divided input feature maps in four equal part, first we apply maxout to both first and second pairs, then we subtract one to another.
The idea behind this unit is that any continous piece-wise linear function can be expressed as a difference of two convex piece-wise linear function. After the application of the difference of maxout the dimension of input feature maps is a quarter of the initial dimension.
In formula:

$$a^l = max(x^l_1, x^l_2) - max(x^l_3, x^l_4)$$

In the program :

```
def diff_maxout(self,X):
    x1, x2, x3, x4 = tf.split(X, 4, axis=-1)
    y1 = tf.math.maximum(x1,x2)
    y2 = tf.math.maximum(x3,x4)
    a = tf.math.subtract(y1,y2)
    return a
```

2.3) Minimum unit

This unit replace the max function of the maxout with the min function, namely it compute the element-wise minimum among two inputs.
The concept remain the same and naturally also with min fuction non linearity property is preserved.
In formula:

$$a^l = min(x^l_1, x^l_2)$$

In the program :

```
def minout_unit(self,X):
    x1, x2 = tf.split(X, 2, axis=-1)
    a = tf.math.minimum(x1,x2)
    return a
```

2.4) Sorting units

The main particularity of this function is that it mantains the dimension of input vector unchanged, like Relu function. It's a combination of the previous units combined so that the output size is the same as the input. The final concatenation is computed at channels level.
In formula:

$$a^l = concat(max(x^l_1, x^l_2), min(x^l_1, x^l_2))$$

In the program :

```
def sortout_unit(self,X):
    y1 = self.maxout(X)
    y2 = self.minout_unit(X)
    a = tf.concat([y1,y2],axis=-1)
    return a
```

2.5) Recursive maxout

The last unit is the recursive maxout unit. It's implement the previous maxout n times, each time with the output of the previous maxout function as input. Final dimension of the output channels depend on the number of the recursive calls n and it is equal to $1/2^n$.

In formula:

$$a^l = f(x_1^l, x_2^l)$$

In the program :

```
def recursive(self,X):
    y = self.maxout(X)
    y2 = self.maxout(y)
    a = self.maxout(y2)
    return a
```

3) Network

For testing the maxout and their variants functions and their properties about the Super Resolution framework I‘ve built a convolutional neural network and I’ve executed many training experimets with different activation functions. In the next sections I will show the network details and parameters.

3.1) Architecture

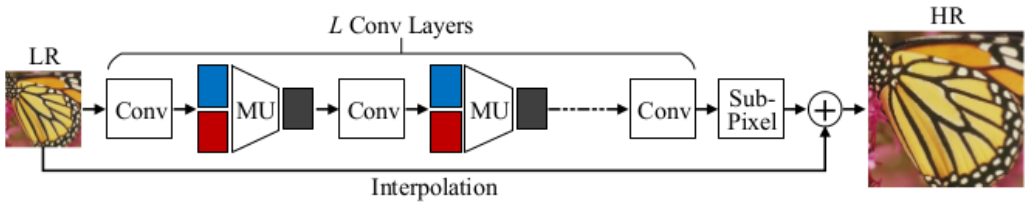


Fig. 1 : Network architecture example

The convolutional neural network used in the experiments is a 6-layers network. For each layer I computed convolution and then the results are passed to activation function to introduce high non linearity. The particularity of this network comes out at the end of the structure : in fact the outcomes of this layer are passed to an interpolation layer.
This layer allow to built up the final proposed upscaled image : the final variables are arranged together input pixel in order to compute the desired result. For example if we want upscale an image of order 2, in addition to the initial pixel we need also 3 new pixel computed by the network.
If we want upscaled an image of order 3 we need 8 new pixel, an image of order 4 need 15 new pixel and so on. So in order to compute this transformation I built a function named Interpolation which has the task to create the upscaled image starting from variables and low resolution input image. I’ve built two kind of Interpolation function to built images two and three times greater respect the input. Below I show a picture example:

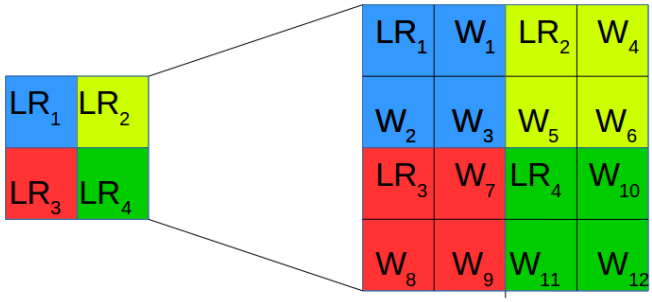


Fig.2 : An example of interpolation function mechanism

Subsequently I introduce the performance measures

3.2) Performance measures

To measure the outcomes and the results of the training in this work I've adopted the PSNR (Peak Signal-to-Noise Ratio) performance measure. This is a well know quality measure of compressed images respect to original : it is defined as the ratio between the maximum power of a signal (I) and the power of noise that can invalidate the fidelity of its compressed representation (the square root of mean square error among original and reconstructed image).
It is measured in decibel.

$$PSNR = 20 \cdot \log_{10}\left(\frac{MAX\{I\}}{\sqrt{MSE}}\right)$$

4) Experimental results

Once defined problem and the proposed solution, first I will show last details and parameters used in this work, then I will show finally obtained results of maxout and their variants activaction functions. I've performed many experiments to validate the performance and the advantages of maxout functions, also respect to well known Relu function.

4.1) Dataset

For this work I used a set of 100 high resolution images taken from the web. Their low resolution version(x2, x3) was been obtained with a bicubic interpolation function. Before given them into the network, as the work in the reference paper do, both low resolution and high resolution images are normalized among 0 and 1, and then low resolution ones are sutracted of 0.5 to have zero mean. The super resolution process is applied only on the Y-channel of YcbCr color space, then the chroma components of original high resolution images can be added at the end of the process.

The dimension of input images is 24x24 and depending on the upscale request, the output dimension is 48x48 for x2 upscaling, 72x72 for x3 upscaling and so on. Low and high resolution images pairs are cropped, from their original ones, in a random position every time start the training process, so for each training process are used different images. Moreover each 500 epochs the data are uploaded and refreshed to avoid overfitting. Naturally high resolution crop dimension depend on upscaling process. In order to avoid overfitting data augmentation methods are implemented : in fact some LR-HR pairs are also flipped and rotated. A separated set of images was used to test PSNR index .

4.2)Traning and parameters

All the networks were implemented using Tensorflow and were trained and tested both on Colab and on CPU intel i7. Each training process were trained for 5000 epochs (epoch duration depend on batch size which for all experiments was setted to 2). During the training the system minimize a loss function which is the PSNR index changed of sign (since we want maximize it) among reconstructed images and high original one. All the networks are trained using ADAM optimizator with an initial learning rate of 10^{-4} descreased at the end to 10^{-3}. The number of parameters for each activation test is controlled by adjusting the number of convolution filters. Depending on the type of activation function used the dimension of the network's weight change. Each activation function trasform input dimesion depending on its property : for example maxout function's output dimension is halved respect the input, relu and sorting unit maintains same dimension between input and output, recursive depend on the number of maxout inside it and so on. So weight's dimension change according to which activation function we want to used.
In the next section are showed the obtained results.

4.3)Results

Here for each activation function I'll show the achieved results in terms of PSNR tested on training set and on separated validation set. The test bed is the task to upscale a low resolution image of order x2. In addition to maxout and its variants (Difference of Maxout – MU-D, Minimum – MU-M, Sorting MU – S, Recursive MU-R), I've tested also Relu activation function in order to measure the effectiveness and the difference with the maxout and at the end maxout function for the upscaling process of order x3 (MU – 3).
Relu produces feature maps with many zeros whose number is not controllable, so in these experiment we also confirmed the better performance of maxout functions respect to classical activations.
In next tables at the end of the section of each activation function will be show the PSNR index plot during training (each point in the plot is the mean value of 50 epochs, so 100 points for 5000 epochs of training).

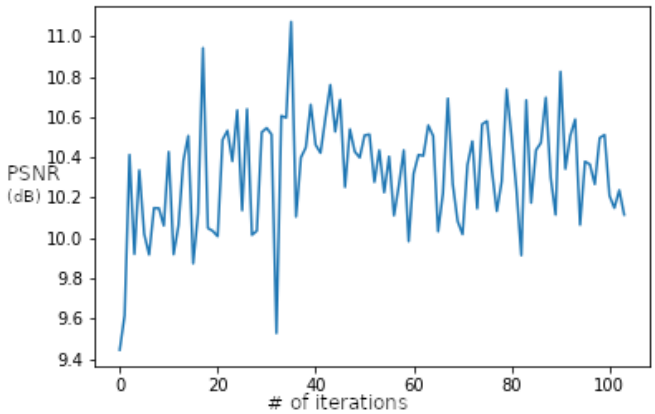
Activation function	Size of conv. filter	# of Parameters	Training PSNR mean	Training PSNR	Testing PSNR
Relu	3 x 3 x 12 x 12	5.7K	9.58	9.60	9.58
Maxout	3 x 3 x 8 x 16	5K	12.04	12.43	12.21
MU-D	3 x 3 x 6 x 24	5.5K	12.90	13.30	13.09
MU-M	3 x 3 x 8 x 16	5K	12.01	12.05	11.98
MU-S	3 x 3 x 12 x 12	5.6K	12.52	12.99	12.90
MU-R	3 x 3 x 6 x 24	5.5K	12.75	13.19	12.96
MU-3	3 x 3 x 8 x 16	5K	10.91	10.98	10.87

Table 1. The values of training and testing PSNR are taken from the network after $1.6 \cdot 10^6$ iterations. Test values have been carried out from a separated set of images.

In the next plots belows are showed the PSNR results measured in dB.

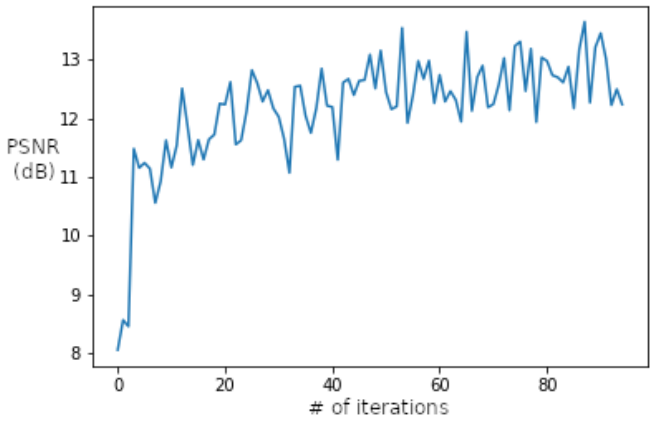
4.1) Relu

We first start to analyze the Relu activation function, since in this environment is the most used, in order to compare typical behaviour and results respect to maxout-type activation functions.
We can see that training results are very noisy : it reach quickly a good PSNR value and then try very slowly to optimize it.



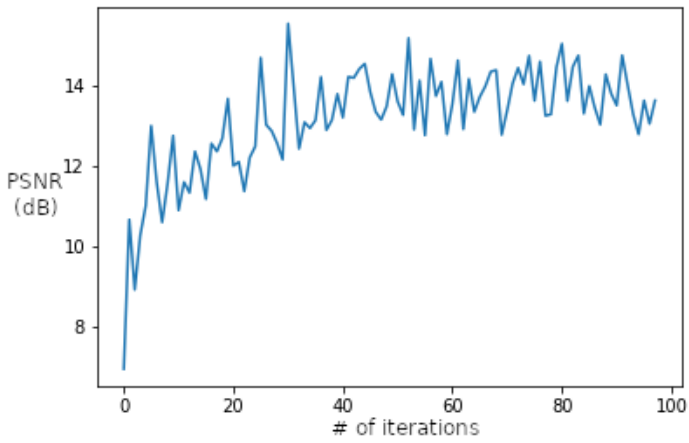
4.2) Maxout

Here I analyze the results obtained from the maxout activation function's experiments. It presents a more smooth learning during training, always considering the noisy environment. It reach values bigger respect to relu function using however less parameters, confirming so our assumptions.
In conclusion this function achieved very good results.



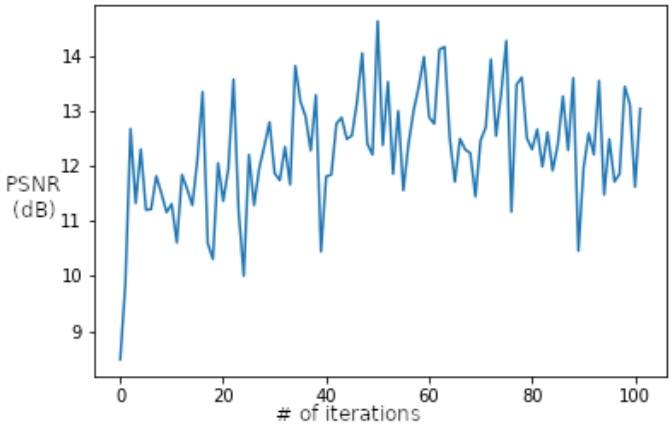
4.3) Difference of maxout

This variant of maxout achieve very good results. It uses few more parameters respect to other variants. Learning is a bit faster respect to maxout then it converge to an high final training mean of 13.30 dB.



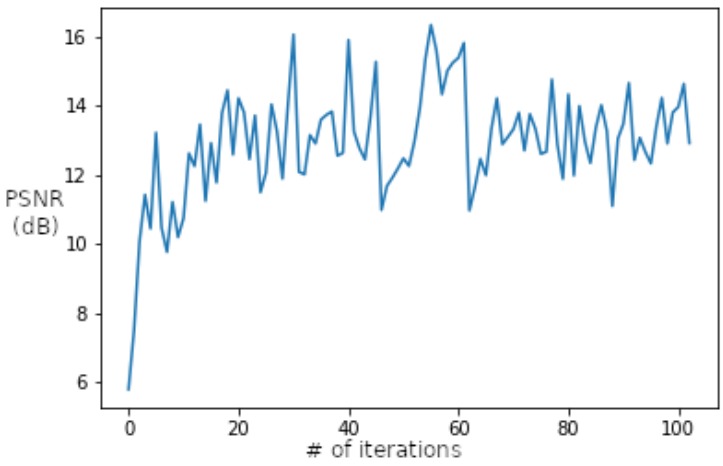
4.4) Minimum

Here I show miimun unit results. The PSNR plot is very less smooth, but however it reach a positive outcome.



4.5) Sorting

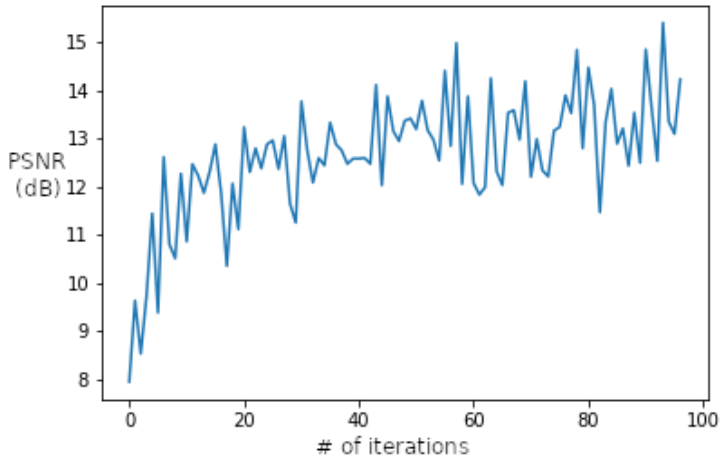
Also sorting variants present a behaviour similar to maxout functions. It reach very good results : in fact final training PSNR reach value of 12.99 dB



4.6) Recursive

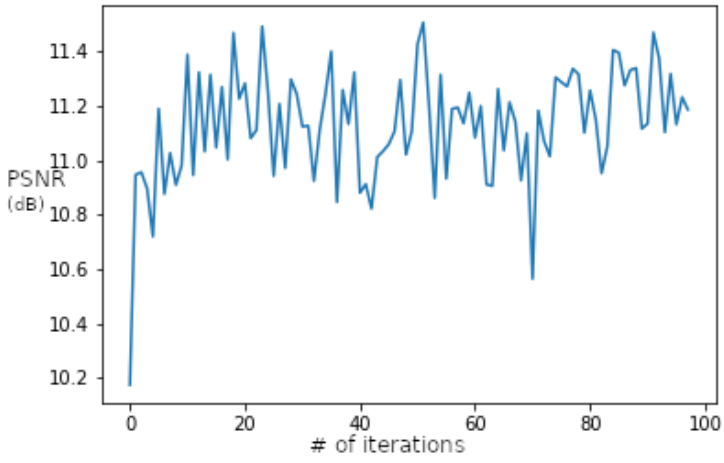
The number of weights of the network is similar to other variants. The recursive activation function achieve similar results of other functions using slightly more parameters than other original maxout. Here we used a recursive function which call maxout function only two times recursively.

Obviously the more the recursive calls are the more the output size decreases and we must be careful to not use too much calls in order to not loose a lot of information.



4.7) Maxout (x3)

Here we can see the results of maxout activation function applied in the work of upscaling 3 times a low resolution images. We can notice the fact that since the task is more complex the PSNR index Is smallest respect the corrispective upscaling process x2. The behaviour is still noisy. In the network the main change is in the number of feature maps in the last layer needed to compute final image. For example in upscaling x2 framework for each low resolution input pixel I need other 3 pixel (from the network) to build the upscaled images, and so in the last layer before interpolation I need 3 feature maps. Instead, in upscaling x3 images, for each low resolution input pixel I need 8 pixel from the network, and so 8 feature maps in the last layer plus one from original low resolution image to build the upscaled x3 high resolution image.
Here below the plot PSNR plot during training.



5) Conclusion

In conclusion in this work I've implemented maxout activation function and its variants applied to single image super resolution framework. For each activation function test I've changed the network's variable dimension to adapt it in function of input and output of all activation functions. I've also compared this variants respect to wide used activation function in CNN, Relu function, and I've show the best results achieved by maxout function and its variants depending on PSNR index and number of weight parameters. I 've tested all functions on upscaling x2 framework and I' ve tested maxout function also in upscaling x3 environment to test the efficacy of this function.

References

[1] Choid, M. Kim : Single Image Super-Resolution Using Lightweight CNN with Maxout Units