Upscaling with Deep Convolutional Networks and Muxout Layers

Pablo Navarrete Michelini, Lijie Zhang, Jianmin He
BOE Technology Group Co., Ltd., Software Lab, Internet of Things and Artificial Intelligence Institute
BOE

Abstract

Previous work has shown the advantage of using convolutional networks to enhance standard image upscalers. Our method incorporates the upscaler within the network using a so–called *Muxout Layer* that increases the size of image features by combining them in groups. The system structure is motivated by an interpretation of convolutional networks as adaptive filters and classic interpolation theory.

Introduction

Super–resolution (SR) methods upscale low–resolution images using prior information to get superior image quality. **Deep Learning** is one of the leading alternatives for example–based single image SR [1, 2]. We note that:

- Typical Deep Learning applications (e.g., classification) **reduce the dimensionality** of features from input to output (e.g. using *max–pooling*).
- Super-resolution needs to **increase the dimensionality** of features from input to output.
- **Previous work used traditional upscalers** (e.g., bicubic) and then enhance the result with convolutional networks [1, 2] (Figure 1a).
- **Previous work used shallow networks** (about 3–4 layers) to get best result.

Contributions

1. **Introduction of a new layer** that increases the dimensionality of image features.
2. **Provide a simple interpretation** of the system based on standard upscaler theory.
3. **Use deep networks** (more than 10 layers) to improve state–of–art results.
4. **Show ability of unsupervised learning objects** by using images of text and upscale from an almost unreadable resolution.

Muxout Layer

A **Muxout Layer** with parameters $S_x, S_y \in \mathbb{N}^+$ takes image features in an input tensor X with standard 4–D components (batch, feature, row and colum), and outputs a 4–D tensor Y with S_xS_y times less features, and each feature with S_x (S_y) times more horizontal (vertical) pixels. We use the Numpy/Theano notation for fancy indexing: $x[start : end : step] = [x_{start}, x_{start+step}, \ldots, x_{end-1}]$, with omitted indexes meaning $start = 0$, end equal to last element, and $step = 1$.

MUXOUT LAYER $M_{S_x \times S_y}$:

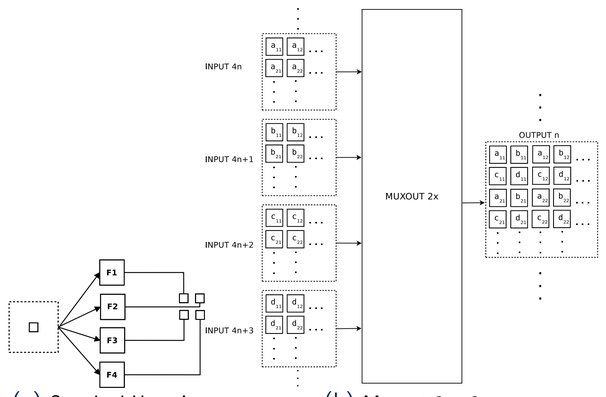
Input X, *4–D tensor,* F *features* ;
Output Y, *4–D tensor,* $F/(S_xS_y)$ *features* ;
for f ***in*** $0 : F$ **:**
  $k \leftarrow 0$;
  for n ***in*** $0 : S_x$ **:**
    for m ***in*** $0 : S_y$ **:**
      $Y[:, f, n :: S_x, m :: S_y] \leftarrow X[:, S_xS_yf + k, :, :]$;
      $k \leftarrow k + 1$;

Training

- Use stochastic gradient descent SGD. Muxout layers do not add parameters to optimize.
- For each input image x, we consider training examples $y_i, i = 1, \ldots, C$, after each convolutional network:

$$L(x, y_1, \ldots, y_C) = \sum_{l=1}^{C} \lambda_l \, MSE(x, y_l) + L_{reg}$$

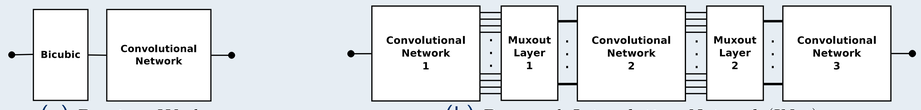
- Start with $\lambda_l = 0, \forall l > 1$, to train only first CN.
- End with $\lambda_l = 0, \forall l < C$, to train the whole network.



(a) Standard Upscaler (b) Muxout 2×2

Figure 3: (a) A standard $2\times$ Upscaler using 4 filters (convolutions). (b) Muxout Layer with $S_x = S_y = 2$.

Learning to Upscale English Text



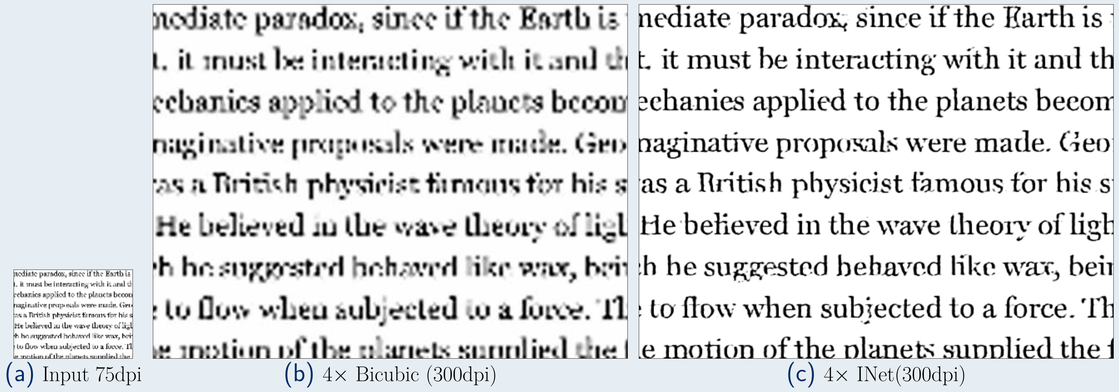
(a) Previous Work (b) Proposed *Intepolation Network* (INet)

Figure 1: Structure of previous work solutions(a) compared to the proposed solution using Muxout layers (b).

We train a system with 12 convolutional layers and 2 Muxout layers.

$$1 \xrightarrow[9\times19]{} 64 \xrightarrow[9\times19]{} 64 \xrightarrow[9\times19]{} 64 \xrightarrow[9\times19]{} 64 \xrightarrow[M_{2\times2}]{} 16 \xrightarrow[5\times5]{} 16 \xrightarrow[5\times5]{} 16 \xrightarrow[5\times5]{} 16 \xrightarrow[5\times5]{} 16 \xrightarrow[M_{2\times2}]{} 4 \xrightarrow[5\times5]{} 4 \xrightarrow[5\times5]{} 4 \xrightarrow[5\times5]{} 4 \xrightarrow[5\times5]{} 4 \xrightarrow[1\times1]{} 1$$

Fiter shapes of 9×19 in first convolutional layers are chosen to extend the horizontal scope of the system and capture words. Learning is **unsupervised** in the sense that all information is given as image pixels.



(a) Input 75dpi (b) $4\times$ Bicubic (300dpi) (c) $4\times$ INet(300dpi)

Figure 2: Our system performance after training with images of English text.

Interpretation

A standard interpolator (Figure 3a) is equivalent to a *Muxout Layer* (Figure 3b) using fixed filters instead of a convolutional network (CN) in the input layer.
A CN with parameters W (filters) and b (biases) is a non–linear system due to the activation functions (e.g. *ReLU*, sigmoid, etc.). Using *ReLU*'s in a CN can be interpreted as using switches to turn on and off different sets of filters. The function of bias parameters b is to select which filters to turn on/off. Then, the net effect of a CN is mostly given by the response of active filters, equivalent to a single filter that changes with the input. The whole system then can be interpreted as an interpolation method with adaptive filters, which motivates the name of *Interpolation Network* (INet).

- For example images, in Figure 1b ($C = 3$) we used:
  - Output y_3: original text image at 300dpi.
  - Muxout layer y_2: LPF images at 300dpi.
  - Muxout layer y_1: downsample images at 150dpi.
  - Input x: downsample images at 75dpi.
- We found that introducing a bias in filter initialization can accelerate convergence for deep networks. We propose 2 alternatives:

1. $W_{i,j} = \underbrace{\delta_{c_x,c_y}/n_{in}}_{\substack{nearest\ neighbor \\ interpolator}} + \underbrace{\alpha\, U(-1,1)/\sqrt{n_{in}}}_{Standard(Xavier,et.al.)}$

2. $W_{i,j} = \underbrace{1/(H\ W\ n_{in})}_{area\ interpolator} + \underbrace{\alpha\, U(-1,1)/\sqrt{n_{in}}}_{Standard(Xavier,et.al.)}$

Table 1: PSNR Comparison using 14 natural images

Average St.Dev	BICUBIC	SC [3]	SRCNN [1]	INet
$2\times$	30.14 3.76	32.07 4.27	32.18 4.19	32.71 3.98
$3\times$	27.43 3.57	28.34 3.79	28.95 4.00	30.01 4.12
$4\times$	25.90 3.43	26.46 3.53	27.12 3.72	28.33 3.85

Analysis

- **For text upscaling**, INet is not provided text data, only images (unsupervised learning). The system is ambiguous when it makes mistakes like: "E" or "B", "li" or "h", "ec" or "cc".
- **For natural images**, Table 1 shows how our method performs similar to [3, 1, 2] for $2\times$ upscaler, and performs better in $3\times$ and $4\times$ upscalers.
- **Performance** is comparable to state–of–art [1]. Upscale $330 \times 536 \longrightarrow 1320 \times 2144$ takes $0.2[sec]$ on NVidia GTX770. Our implementation uses: Theano GIT + CuDNN v4rc + CUDA v7.5.

References

[1] C. Dong, C.C. Loy, K. He, and X. Tang.
Learning a deep convolutional network for image super–resolution.
In D. Fleet et al., editor, *Proceedings of ECCV*, volume 8692 of *LNCS*, pages 184–199. Springer, September 2014.

[2] C. Dong, C.C. Loy, K. He, and X. Tang.
Image super-resolution using deep convolutional networks.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015, DOI 10.1109/TPAMI.2015.2439281,.

[3] Jianchao Yang, J. Wright, T.S. Huang, and Yi Ma.
Image super–resolution via sparse representation.
Image Processing, IEEE Transactions on, 19(11):2861–2873, Nov 2010.

Contact Information

- Web: http://www.boe.com.cn
- Email: pnavarre@boe.com.cn
- Phone: +86 186 0115 5054