Super-risoluzione per il riconoscimento di targhe

Obiettivo di questo progetto è la super-risoluzione, cioè come fare ad ottenere un'immagine ad alta risoluzione partendo da una versione a bassa risoluzione in modo tale da conservare i dettagli in misura superiore a ciò che si otterrebbe con una normale interpolazione. A tal fine userete il dataset License Plate costituito da più di 500 fotografie che inquadrano targhe di risoluzione 289×386 pixel. Per questo progetto userete una fully convolutional network, cioè una CNN che non prevede strati fully connected e ha il vantaggio di poter essere applicata ad immagini di dimensioni qualsiasi. Per la super-risoluzione, la rete dovrebbe fornire un'immagine più grande rispetto a quella di ingresso. Come architettura userete la DnCNN proposta in [1] formata da 17 strati convoluzionali. La DnCNN realizza il *Residual Learning*, quindi la rete non fornisce direttamente l'immagine in uscita, ma l'immagine di dettagli che deve essere sommata all'immagine di ingresso per ottenere quella finale.

In questo progetto i passi da seguire sono:

1. **Download dei dati**. Scaricate le 500 immagini dal sito del dataset: `http://www.zemris.fer.hr/projects/LicensePlates/english/images.html`

2. **Preparazione dei dati**. Per limitazioni di memoria la rete verrà addestrata su blocchi. Selezionate 400 immagini per il training-set, 50 immagini per la validation e le restanti per il test. Per il training e la validation estraete da ogni immagine dei blocchi di dimensione 96×96 pixel sovrapposti adottando una passo di 24 pixel. I blocchi così ottenuti costituiranno l'uscita desiderata, ridimensionateli di un fattore 1/2 per ottenere i dati a bassa risoluzione e poi usate un'interpolazione lineare per riportarli alle dimensioni originali. In questo modo avete ottenuto i dati di ingresso alla rete. Sottraete i dati di ingresso ai blocchi originali per ottenere i dati di uscita della rete.

3. **Architettura**. Definite l'architettura descritta in tabella 1 utilizzando un'inizializzazione dei pesi con matrici ortogonali settando il parametro `kernel_initializer='Orthogonal'`.

4. **Addestramento**. Per l'addestramento utilizzate l'ottimizzatore Adam: `keras.optimizers.Adam`, mentre per la loss function usate l'MSE: `keras.losses.mean_squared_error`. Utilizzate le prestazioni sul set di validation per selezionare i migliori valori per il learning-rate, il batch-size, il numero di epoche.

5. **Valutazione delle prestazioni**. Valutate le prestazioni in termini di MSE e PSNR sulle immagini di test. Inoltre, provate ad addestrare la rete usando contemporaneamente diversi fattori di ridimensionamento.

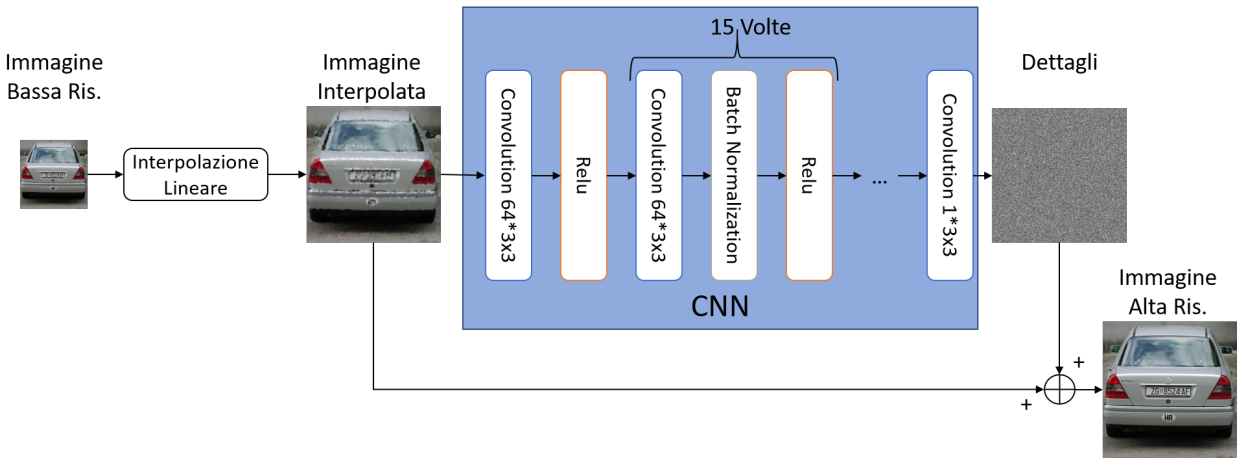

Figure 1: Architettura DnCNN

Tipo	Dim. Spaziale	Num. Feat.
Convolution+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution+BatchNorm+ReLU	3×3	64
Convolution	3×3	3

Table 1: Architettura della CNN.

References

[1] K. Zhang, et al. "Beyond a Gaussian Denoiser: Residual Learningof Deep CNN for Image Denoising" IEEE Transactions on Image Processing, 2017.