Analisi Comparativa di Architetture Deep per la SuperRisoluzione

Progetto per i Corsi di Deep e Machine Learning

Nome & Cognome : **Francesco Recchioni** Matricola: **523594**

Link Progetto Github: **https://github.com/FraRec/DeepLearningProject/tree/FraRec-patch-1**

Intro

La Super Resolution è uno degli ambiti più studiati nel contesto del Deep Learning.
Ogni anno escono tantissimi Articoli Scientifici i quali studiano nuove tecniche e architetture, che hanno come obiettivo l'aumento delle performance e il miglioramento dei risultati dell'anno precedente.

Oggi si è arrivati a costruire delle Architetture molto complesse che nascono dalla consapevolezza di ciò che ha funzionato nelle architetture passate e di ciò che invece è stato scartato nel tempo.
Compiendo un'analisi riferita all'evoluzione delle tecniche pubblicate durante gli anni, si possono trovare determinate scelte ricorrenti che contribuiscono all'ottenimento di migliori performance.
Alcune di queste scelte sono: l'uso della Perceptual Loss, Residual Learning, usare differenti tipi di ReLU, Sub-Pixel CNN, Dense Block, etc...

Lo studio fatto in questo progetto ha come obiettivo quello di applicare su "vecchie architetture" alcune di queste modifiche, per comprendere se sia possibile ottenere un miglioramento delle performance, senza però allontanarsi troppo dalla loro versione originale.
Le Architetture affrontate saranno, in ordine:

1. **SRCNN** ---- (Super-Resolution Convolutional Neural Networks)
2. **VDSR** ----- (Very Deep Super-Resolution)
3. **EDRN** ----- (Enhanced Deep Residual Networks)
4. **RDN** ------ (Residual Dense Net)
5. **SRResNet**
6. **SRGAN**

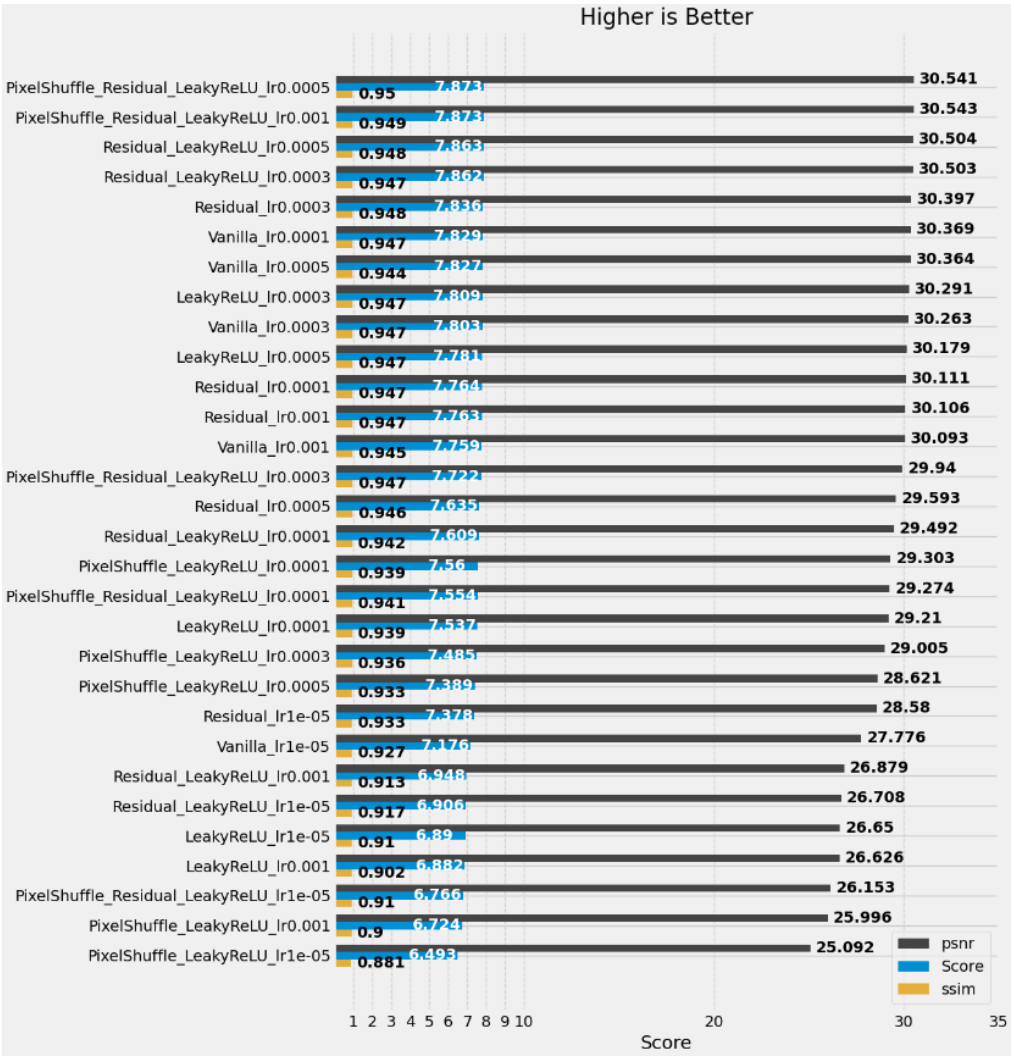
Struttura del Progetto

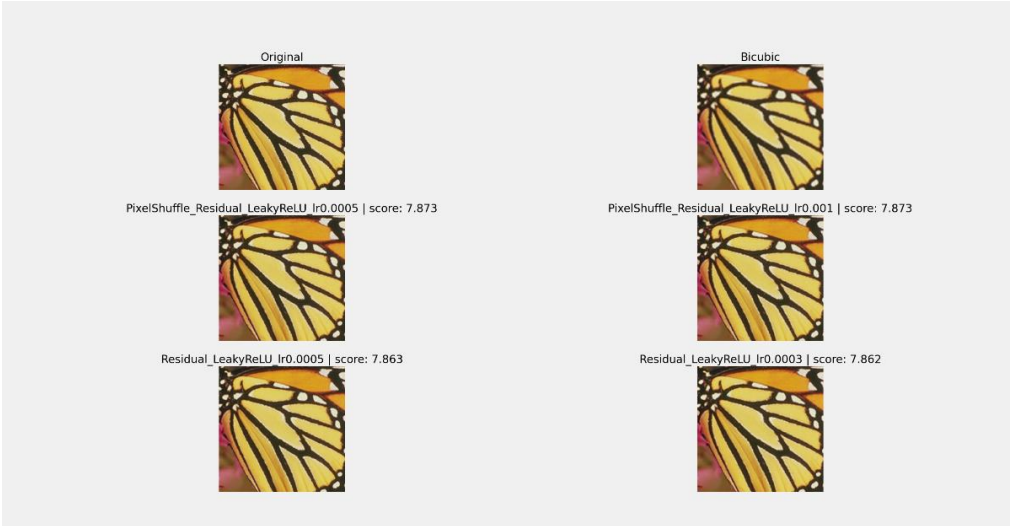
L'iter seguito per lo studio di ogni Architettura è iniziato con la ricerca e l'analisi di una sua implementazione sul web. Successivamente tale implementazione è stata modificata e addestrata nuovamente da zero per un determinato numero di epoche e utilizzando vari Learning Rates.
Si è pensato ad implementare solo modifiche che non andassero a snaturare troppo la versione originale dell'architettura, ma che permettessero di ottenere un miglioramento delle Performance.

Ad ogni cambiamento apportato all'architettura iniziale, sono stati memorizzati e conservati una moltitudine di dati.
In particolare durante l'addestramento ci si è affidati a dei callbacks per registrare tramite Tensorboard l'andamento dei parametri, della loss e delle metriche di valutazione. Poi, sempre per ogni modifica, è stata conservata una summary e un relativo plot del modello.

I file notebook presenti nel progetto rappresentano un esempio di come sono state addestrate le architetture. Tuttavia, il vero addestramento è stato effettuato utilizzando Google Colab e i relativi strumenti messi a disposizione dal servizio.
Infine, per rendere più agevole la fase di test, sono stati creati degli Script in Python chiamati: "ModelsEvaluation*NomeArchitettura*".
Essi sfruttano delle Classi create ad hoc per effettuare la valutazione di tutti i modelli modificati a partire da una determinata architettura. Dopo aver definito un immagine di test, vengono comparati i risultati creando una classifica basata sulle migliori performance.

Questa classifica può essere visualizzata attraverso delle immagini di cui viene eseguito il Plot alla fine dell'esecuzione di un dei file "ModelsEvaluation*NomeArchitettura*".
Le Classi create ad hoc si chiamano: Iperparameters, ModelFolder, EvaluateAllModels, e si trovano nella cartella utils. *(eccetto che per quelle relative alla SRCNN, in quanto la struttura del modello ha richiesto delle modifiche a parte, e quindi degli script specifici)*





(Esempio immagini Output dello script ModelsEvaluationSRCNN)

Architetture Studiate:

Ogni Architettura è stata addestrata per un numero di Epoche arbitrario, la cui scelta dipendeva sia dalla velocità di addestramento, sia dalla volontà di ottenere dei risultati il più efficienti possibile.

SRCNN

La SRCNN è stata la prima architettura ad essere studiata, infatti è anche la più semplice tra tutte quelle presenti.
Essendo composta da soli 3-layer convoluzionali [1], a causa della grandezza del Dataset, non è in grado di entrare in overfitting, anche se addestrata per un numero di epoche molto grande. Nel caso di questo progetto ci si è limitati ad un addestramento di 50 epoche. Le modifiche effettuate per cercare di ottenere migliori performance sono svariate, tuttavia analizzando i risultati relativi agli Score finali è possibile notare come la migliore performance sia stata raggiunta dall'architettura:

- **PixelShuffle_Residual_LeakyReLU**
    - Utilizzo della Funzione di Attivazione "LeakyReLU" al posto della ReLU base.
    - Skip Connection che permette addizionare l'output dei layer convoluzionali con l'input della rete dopo il Resizing.
    - Sostituzione dell'ultimo layer di convoluzione base con uno di Upscale basato sul PixelShuffle.

Tale architettura infatti riesce a migliorare la versione originale della SRCNN [2] (*definita Vanilla nel Progetto*) e al contempo diminuire ulteriormente il numero di parametri addestrabili della rete, riuscendo a renderla, se vogliamo, ancora più semplice.

➔ *Vanilla ------------------------- Total params: 109827 (429.01 KB)*
➔ *PixelShuffle_Residual_LeakyReLU -- Total params: 81123 (316.89 KB)*

VDSR

Nel caso della VDSR, purtroppo, non si è riusciti a migliorare le performance della versione originale (se non attraverso un cambio del Learning Rate).
Questo perché l'architettura di partenza, essendo composta solo da un susseguirsi di layer convoluzionali [3], non lasciava molto spazio a possibili modifiche senza andare a snaturare troppo la sua natura di base. Per questa motivazione ci si è limitati a cambiare solo la funzione di attivazione, analizzando le relative conseguenze.
Per via della lentezza nel riuscire ad addestrare un'architettura di questo tipo, ci si è limitati ad un periodo di 10 epoche. Anche questa caratteristica ha giocato un ruolo importante nella decisione di non addentrarsi troppo in uno suo studio più approfondito.

EDSR

La scelta di studiare questo tipo di architettura nasce dal modo in cui è costruita [4]. Infatti, tale rete presenta, come componente principale del suo scheletro, dei blocchi chiamati ResBlock. Questi blocchi sono costituiti da layer di varia natura. Possiamo trovare layer convolutivi, batch norm, 1x1, etc…
L'idea di base è che cambiando il modo in cui sono costruiti questi ResBlock, possiamo ottenere performance diverse, senza essere costretti a cambiare anche la loro disposizione nella Rete (od i loro input e output).
In questo modo possiamo rimanere vicini all'architettura originale e analizzare i risultati delle modifiche, trovando una configurazione dei blocchi che riesca a migliorare le performance.
Un ulteriore motivo del perché si è scelto di studiare questa Rete risiede nel fatto che questi blocchi vengono usati anche nella SRResNet.

Di conseguenza, trovando una buona configurazione dei ResBlock attraverso la EDSR, possiamo riutilizzarla per tentare di migliorare anche la SRResNet.
In questo modo possiamo acquisire una direzione ed un contesto su quali modifiche possono funzionare e quali no.
Effettivamente, come si può notare nel Progetto, si è riusciti ad ottenere più di una configurazione dei ResBlock che migliora le performance della EDSR rispetto alla sua versione originale.

RDN

La RDN è stata studiata per gli stessi motivi della EDSR. Tuttavia, in questo caso al posto dei ResBlock, troviamo i DenseBlock [5]. Questi blocchi si differenziano soprattutto per via dell'utilizzo massiccio di SkipConnections che viene fatto.
L'implementazione della Rete è stata fatta quasi completamente in autonomia a causa della mancanza di riferimenti sul web da cui trarre ispirazione.
Come nella VDSR, non sono state effettuate grosse modifiche, ad eccezione del cambio di Funzione di Attivazione, poiché si è voluto lasciare invariati i DenseBlock originali, così da poterli successivamente riutilizzare nello studio della SRResNet.

SRResNet

L'SRResNet è sicuramente l'architettura più importante studiata in questo progetto [6], assieme alla SRGAN, di cui in realtà fa parte svolgendo il ruolo di Generatore.
Per via della complessità di questa Rete si è optato per un addestramento ridotto a 25 epoche, ma aiutato dal Transfer Learning.
Il processo di addestramento infatti è stato svolto in maniera leggermente diversa rispetto a quello visto nelle architetture precedenti. Prima è stata costruita la Rete Originale, poi sono stati caricati pesi pre-addestrati attraverso un processo descritto nel Progetto GitHub preso come riferimento [7].
In seguito, la prima metà della Rete è stata congelata e sono state effettuate modifiche solo sulla seconda metà.
In questo modo i risultati relativi agli score sono ovviamente più simili tra di loro, ma si è comunque riusciti a migliorare le performance. In particolare i risultati migliori sono stati raggiunti attraverso la Rete modificata:

- **DenseBlock**
  Nella seconda metà della Rete sono stati utilizzati i DenseBlock visti nella RDN, al posto degli originali ResBlock.

(Uno dei motivi che hanno portato a utilizzare questi DenseBlock è stato lo studio del paper relativo alla ESRGAN, ovvero alla versione enhanced della SRGAN. In queto paper infatti, per ottenere migliori performance, veniva implementata proprio questa tipologia di blocchi.)

SRGAN

Per via del lungo tempo di addestramento necessario a completare una singola epoca, la SRGAN è stata addestrata per solo 5. L'addestramento è stato effettuato attraverso l'utilizzo di un custom loop originale [7] [8], nato attraverso l'unione di varie implementazioni viste sul web.
L'architettura originale è composta da un Generatore ed un Discriminatore, quindi per cercare di ottenere migliori performance, si potrebbe pensare di applicare modifiche ad entrambi. Tuttavia, nei limiti di questo progetto, ci si è concentrati solo sul Generatore, il quale ricordiamo essere interpretato da una SRResNet.
In particolare, a causa della grande quantità di risorse necessaria per l'addestramento della Rete, al posto di effettuare modifiche al Generatore senza una direzione precisa, si è optato per usare la migliore versione

della SRResNet trovata in precedenza, ovvero quella che sfruttava i DenseBlock. Effettivamente questa versione della Rete ottiene delle performance migliori rispetto a quelle dell'originale.

Conclusioni

In conclusione possiamo dire che, in effetti, è possibile trovare delle piccole modifiche con cui alterare le architetture di SR più vecchie, per ottenere delle performance migliori, senza però snaturare la loro struttura originale.
Tuttavia, va ricordato come le performance delle architetture di SR, molto spesso, dipende dal tipo e dalle caratteristiche possedute dall'immagine che gli si chiede di upscalare. Quindi usando la stessa architettura, con alcune immagini potremo ottenere ottimi risultati, mentre con altre no.

Riferimenti

[1] C. C. L. Chao Dong, «Image Super-Resolution Using Deep Convolutional Networks».

[2] MaokeAI, «SRCNN-keras,» [Online]. Available: https://github.com/MaokeAI/SRCNN-keras/tree/master.

[3] J. K. L. a. K. M. L. Jiwon Kim, «Accurate Image Super-Resolution Using Very Deep Convolutional Networks».

[4] S. S. H. K. S. N. K. M. L. Bee Lim, «Enhanced Deep Residual Networks for Single Image Super-Resolution».

[5] Y. Zhang, «Residual Dense Network for Image Super-Resolution».

[6] L. T. F. H. J. C. A. C. Christian Ledig, «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial».

[7] jlaihong, «image-super-resolution,» [Online]. Available: https://github.com/jlaihong/image-super-resolution.

[8] A. Expedition, «Image Super Resolution: SRResNet and SRGAN TensorFlow 2 implementation and model intuition,» [Online]. Available: https://www.youtube.com/watch?v=FwvTsx_dxn8&list=WL&index=19.