**Università Degli Studi di Salerno**

**Dipartimento di informatica**

*Corso di Laurea Magistrale Cloud Computing*



**Unione Dei Progetti Di:**

* Compressione Percettivo Semantica Dell’Immagine Usando Reti Neurale Di Convoluzione

* Compressione Di Immagini Composite

|  |  |
| --- | --- |
| **Docente**  Prof. Bruno Carpentieri  Prof. Raffaele Pizzolante | **Gruppo: Negan**  Capaldo Giuseppe 0522500498  Donadio Raffaele 0522500423 |

Indice

[1. Introduzione 5](#_Toc506741210)

[2. Evoluzione 7](#_Toc506741211)

[3. Simulazione 13](#_Toc506741212)

[3. Testing 16](#_Toc506741213)

[4. Conclusioni 19](#_Toc506741214)

# Introduzione

I segnali analogici (musica, voce, tv e film) sono più difficili da memorizzare e da trasmettere su lunghe distanze a differenza dei digitali. Lo svantaggio principale che riguarda le versioni digitali dei segnali è il fatto che richiedono più bits. Per ridurre i costi di memorizzazione e trasmissione, si adoperano tecniche di compressione, il cui scopo è minimizzare il numero di bits necessari per ottenere una versione accettabile del segnale sorgente originale.

Però tutte le tecniche adoperate negli ultimi tempi sono state tecniche che non hanno tenuto conto di cosa bisognasse comprimere; questo ha portato all’immissione di troppi errori all’interno dell’immagine che si traduce in una pessima qualità visiva.

Il documento che ci è stato consegnato e che abbiamo messo sotto esame è il paper Semantic Perceptual Image Compression Using Deep Convolution Networks di Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo and James Storer della BrandeisUniversity. Nel suddetto si parla dell’utilizzo delle Reti Neurali Convoluzionali (CNN) e di come i recenti progressi nella potenza di calcolo, insieme alla disponibilità di ampi set di dati sull'addestramento, hanno aumentato l'interesse nell'applicazione di CNN’s di deep learning per affrontare il riconoscimento dell'immagine e le attività di elaborazione delle immagini.

Questo paper riporta anche il link di una sua prima implementazione già presente su github;

in cui vengono usati:

1. Tensorflow
2. Numpy
3. Pandas
4. Python PIL
5. PythonSKimage

Inoltre vengono suggeriti:

1. Imagemagick (per operazioni di immagine più veloci)
2. VQMT (per ottenere metriche di confronto delle immagini)

Tutto questo progetto è fondato su Python e sulle CNN, ma purtroppo nessuno del gruppo ha avuto la possibilità di studiare l’ambito dell'intelligenza artificiale e quindi non avendo le giuste competenze per capirlo fino in fondo è stato preso in considerazione come una black-box.

Il progetto in questione si prefigge l’obiettivo di rendere jpeg/jpeg 2000 content-aware, ovvero pur applicando sempre lo stesso algoritmo per compressore e decompressore, si fa in modo di cambiare dinamicamente la quantizzazione della compressione appunto usando la CNN.

Infatti alla fine del paper in questione si spiega che l’unico punto su cui si può migliorare ciò che loro stesso hanno fatto è la CNN migliorando in tutti i suoi aspetti le performance.

Adesso invece prendiamo in considerazione il secondo progetto che ci è stato fornito dal professore:

il progetto è stato realizzato dai nostri colleghi Aiello Michele, Bisogno Giuseppe, Manzo Gerardo, Paolella Vincenzo nell’anno Accademico 2007/2008.

In questo secondo progetto si parla della compressione dell’immagine mediante la separazione in più livelli della stessa, facendo sì che si possano usare livelli di compressione o, addirittura, tipi di compressione diversi per ogni livello facendo sì che al riassemblamento del tutto non si perda in qualità ma solo in dimensione.

I livelli che loro generano sono 3:

Background, Foreground e Maschera.

Il Background è il livello dei pixel che indica oggetti di fondo a cui l’occhio umano presterà meno attenzione.

Il Foreground è il livello dei pixel che indica gli oggetti in primo piano, quelli a cui l’occhio umano presterà più attenzione.

La Maschera è il livello che ci permette di estrarre i livelli sopra-citati dall’immagine originale.

# Evoluzione

Inizialmente siamo partiti con l’utilizzare il progetto dei colleghi, per separare l’immagine nei 3 livelli che la compongono e quindi usiamo una compressione molto forte sul livello di Background e una compressione lieve sul Foreground così da ottenere, alla ricomposizione dell’immagine, un ottimo valore di compressione senza però perderne in qualità. O almeno queste erano le nostre premesse. Per prima cosa ci siamo scontrati con l’insufficienza dei commenti aggiunti al codice, poi abbiamo avuto difficoltà con la natura non modulare del progetto. Andando avanti, vedendo che i primi risultati non rispecchiavano le aspettative abbiamo capito che i colleghi invertono il Foreground dell’immagine con il Background. Poi nonostante queste imperfezioni siamo riusciti a far funzionare il progetto anche perché di fatto i nostri colleghi usano degli eseguibili già pronti per procedere alla compressione. Ma i risultati ancora non rispecchiavano le nostre aspettative ; il Compression Ratio non era confrontabile con la controparte di JPEG2000 puro. A questo punto confrontandoci anche con il professore sulla natura fallimentare di tutti questi tentativi ,abbiamo deciso di cambiare algoritmo di segmentazione passando da suddivisione in Foreground e Background ad algoritmi del tipo superpixel. Inoltre abbiamo creato partendo dalle basi un progetto basato su linguaggio Python che inizialmente ci ha fatto evincere, poiché anche in questo caso non abbiamo ottenuto risultati positivi che l’eseguibile .jar , che la compressione non era perfettamente funzionante perché i suoi valori di PSNR erano sospetti. Confrontandoli con un applicativo che ci permetteva in egual modo di ottenere da una immagine bitmap una compressa ci siamo resi conto che non funzionava bene quindi a questo punto abbiamo integrato JPEG sul progetto utilizzando l’implementazione di Python. Anche perché implementazioni di JPEG2000 al momento non erano reperibili. Quindi abbiamo creato un progetto la cui idea di base consiste nel prendere in input un’immagine e applicarvi un algoritmo di segmentazione, ottenendo un’immagine segmentata secondo la logica del singolo algoritmo di segmentazione.

Una volta ottenuto i frammenti si procede con la selezione dei frammenti più rilevanti visivamente parlando.

**Data la seguente immagine:**

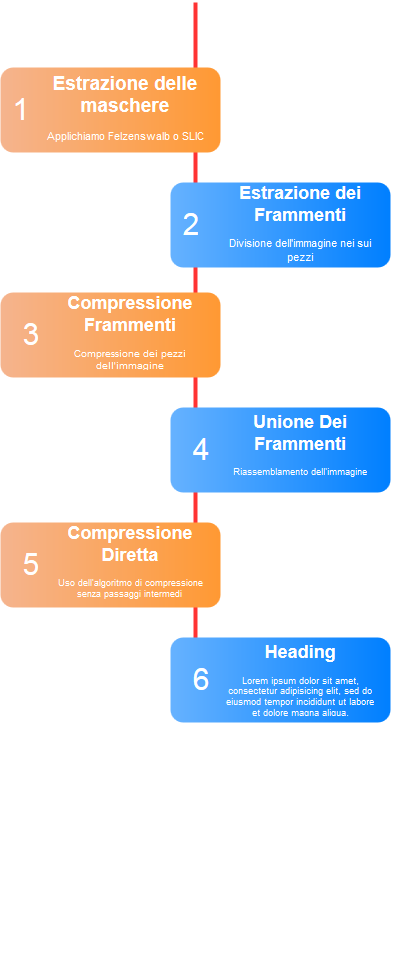


I frammenti più rilevanti potrebbero essere i frammenti che contengono il viso dell’astronauta e la tuta spaziale.

Una volta identificati i frammenti più importanti si procede con la compressione dei singoli , usando una qualità di compressione maggiore per i frammenti importanti e minore altrimenti.

Finita la Fase di compressione si fa un merge di tutti i frammenti per riottenere l’immagine completa.

**Il processo può essere rappresentato tramite il seguente modello:**



1. **Estrazione delle maschere**

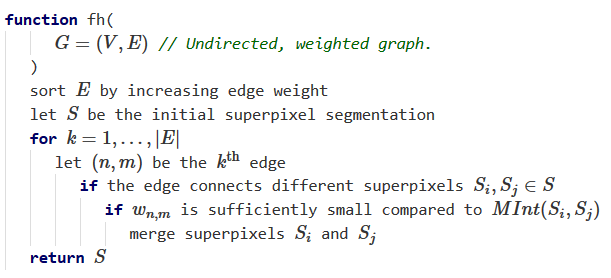
L’estrazione delle maschere avviene mediante l’algoritmo di segmentazione.

Per la segmentazione abbiamo usato la tecnica del superpixel precisamente ci siamo concentrati sulle varianti Felzenswalb & Huttenlocher e SLIC.

La variante Felzenswalb funziona considerata un'immagine e il grafico con nodi corrispondenti a pixel e bordi tra pixel adiacenti. Ad ogni bordo viene assegnato un peso . Quindi, per i sottoinsiemidefiniamo per essere il minimum spanning tree di ***A*** e

.

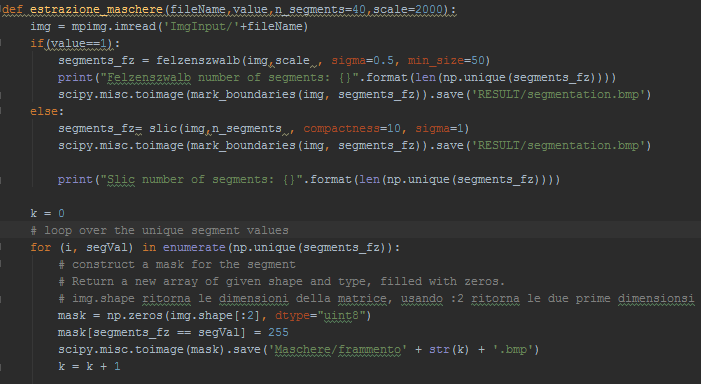
Dove τ è un parametro di soglia eè chiamato la differenza interna minima tra i componenti *A* e *B*. Partendo da una prima segmentazione di superpixel in cui ogni pixel forma il proprio superpixel, l'algoritmo elabora tutti i bordi in ordine di aumento del peso del bordo. Ogni volta che un bordo collega due superpixel diversi, questi vengono uniti se il peso del bordo è piccolo rispetto alla differenza interna minima.



Mentre l'approccio SLIC esegue un clustering locale di pixel nello spazio 5-D definito dai valori L, a, b dello spazio colore CIELAB e delle coordinate x, y dei pixel. Ha una diversa misurazione della distanza che consente compattezza e regolarità nelle forme dei superpixel e può essere utilizzata su immagini in scala di grigi e immagini a colori. SLIC genera superpixel raggruppando i pixel in base alla somiglianza e alla vicinanza dei colori nel piano dell'immagine. Per il clustering viene utilizzato uno spazio **[labxy]** a5 dimensioni.Lo spazio colore CIELAB è considerato uniforme perpetuamente per piccole distanze di colore.Non è consigliabile utilizzare semplicemente la distanza euclidea nello spazio 5D e quindi gli autori hanno introdotto una nuova misura di distanza che considera la dimensione dei superpixel.Si inizia campionando K centri di cluster regolarmente distanziati e spostandoli in posizioni di seme corrispondenti alla posizione di gradiente più bassa in un quartiere 3 × 3.Questo è fatto per evitare di metterli a un bordo e ridurre le possibilità di scegliere un pixel rumoroso.I gradienti di immagine sono calcolati come

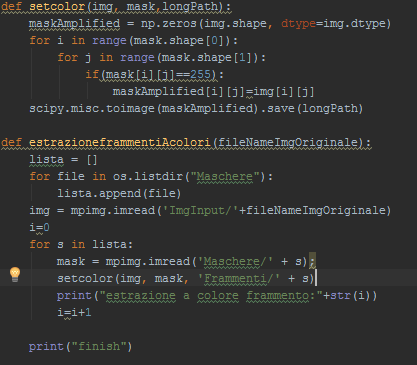
dove è il vettore lab corrispondente al pixel in posizione e è la norma L2. Questo tiene conto sia delle informazioni sul colore che sull'intensità. Ogni pixel dell'immagine è associato al centro del cluster più vicino la cui area di ricerca si sovrappone a questo pixel. Dopo che tutti i pixel sono associati al centro del cluster più vicino, un nuovo centro viene calcolato come vettore lab. Medio di tutti i pixel appartenenti al cluster. Al termine di questo processo, possono rimanere alcune etichette vaganti, ovvero pochi pixel in prossimità di un segmento più grande con la stessa etichetta ma non collegato ad esso. Mette in atto la connettività nell'ultimo passo dell'algoritmo rielaborando i segmenti disgiunti con le etichette del cluster più grande vicino.

**Ecco lo script python per l’estrazione delle maschere**



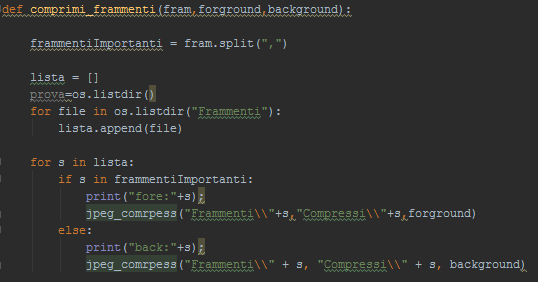
1. **Estrazione dei frammenti**

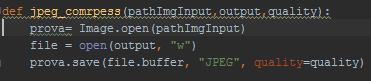
A partire dall’immagine di input e dalla maschera estraiamo i pezzi dell’immagine così da ottenere i frammenti di immagine a colore e in formato bitmap in modo da poter scegliere quali comprimere di più e quali di meno e lo facciamo con una funzione che richiama l’altra così da ottenere un codice più leggibile



1. **Compressione dei frammenti**

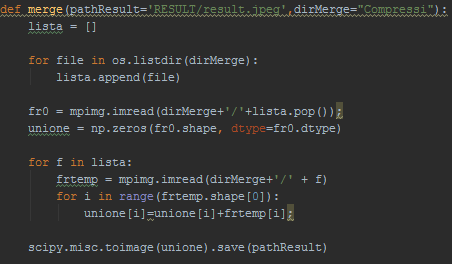
In questa fase si comprimono i frammenti utilizzando JPEG come algoritmo di compressione. Inoltre scelti i frammenti importanti si usa un valore di qualità diverso per quei frammenti.





1. **Unione dei frammenti**

In questa fase ricombiniamo tutti i frammenti compressi in un’unica immagine

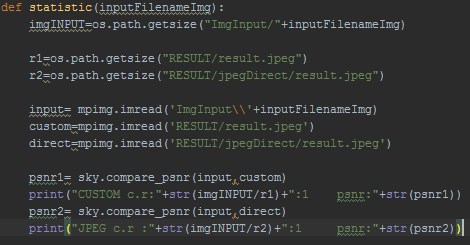


1. **Compressione diretta dell’immagine di input:**

Per ottenere un confronto con l’applicazione del semplice JPEG senza tutti i passaggi sopra-citati chiamiamo la funzione jpeg\_compress con i dovuti parametri.

1. **Calcolo dei Compression Ratio e dei PSNR:**

Inoltre per ottenere dei dati di confronto abbiamo creato una funzione che ci estrae alcuni parametri di confronto qualità e compressione.



Finalmente abbiamo iniziato a ottenere dei risultati più comparabili tra Compression Ratio e PSNR. Infatti a questo punto ci siamo concentrati sul tirare delle conclusioni su questo approccio confrontandolo con JPEG e ottenendo delle statistiche accettabili. Ma confrontandoci di nuovo con i professori non riuscivamo a capire il perché di alcune anomalie. A Parità di Compression Ratio JPEG puro ci risulta che la scomposizione e ricomposizione dell’immagine ha un PSNR inferiore. Inoltre PSNR nonostante tutti gli sforzi non riesce mai ad essere molto alto e quindi questo ci fa sospettare che la tecnica in generale della segmentazione sia fallimentare.

Nonostante queste considerazioni possiamo però dire che mettendo degli opportuni parametri comunque riusciamo ad ottenere degli ottimi Compression Ratio e nonostante il PSNR sia basso la qualità visiva dell’immagine è apprezzabile.

Inoltre con tutte queste operazioni sui pezzi dell’immagine ci siamo resti conto che sui pixel più chiari viene introdotto una quantità di errore rilevante poiché molti di essi variano di colore tendendo all’azzurro.

Con il Professor Pizzolante abbiamo capito che c’era un problema di riapplicazione dell’algoritmo jpeg oltre all’introduzione di una notevole quantità di padding sui frammenti ottenuti da qui ci siamo spinti verso la vera soluzione di questi problemi. Cosa di cui parleremo nel prossimo capitolo.

# Simulazione

Tutto il progetto è incentrato sulla compressione dinamica delle immagini e, come già stato detto, non avendo le competenze di CNN, con l’approvazione del professore, abbiamo optato per un artifizio.

# Testing

|  |
| --- |
|  |
| **Image Segmentation**   |  |  | | --- | --- | | **SLIC** | **Felzenswalb** | |  |  |   **Most Important Segment Seleceted**   |  |  | | --- | --- | | **SLIC** | **Felzenswalb** | |  |  |  |  |  |  | | --- | --- | --- | | **Originale** | **JPEG** | **Finale nostro** | |  |  |  | |  | |  | | --- | | **Quality Compression** | | 70 |  |  |  | | --- | --- | | **Compression Ratio** | **PSNR** | | 31,68:1 | 32.6 | | |  |  | | --- | --- | | **Value Of Fragment Important** | **Value Of Fragment Not**  **Important** | | 75 | 65 |  |  |  | | --- | --- | | **Compression Ratio** | **PSNR** | | 26.53:1 | 32.4 | | 32.61:1 |  | | |
| **Examples Local Analysis**   |  |  |  | | --- | --- | --- | | **frammento** | **Jpeg PSNR** | **Custom PSNR** | | Frammento 0,0 | 33.30 | 33.70 | | Frammento 0.1 | 37.27 | 37.60 | | Frammento 1,0 | 30.38 | 30.78 | | Frammento 1,2 | 30.95 | 31.25 | |

# Conclusioni