UNIVERSITÀ
degli STUDI
di CATANIA

Dipartimento di Ingegneria Elettrica, Elettronica e Informatica

Cdl Magistrale in Ingegneria Informatica

Elaborato di Internet of Things Based Smart System

Implementazione dell'algoritmo K-means per la compressione delle immagini acquisite da smart camera

Docenti:

Prof.Ing. Vincenzo Catania

Prof.Ing. Maurizio Palesi

Studenti:

Giovanni Fausto

Alessia Rondinella

Sommario

1. Introduzione

Con questa breve introduzione andremo a descrivere come abbiamo deciso di affrontare e suddividere il nostro elaborato.

L'elaborato sarà diviso in diverse parti dove verrà spiegato nel dettaglio cosa abbiamo fatto e le tecniche usate, dapprima ci sarà una veloce introduzione con le motivazioni che ci hanno spinto a fare questo elaborato, poi ci sarà una veloce spiegazione di quello che è l'approximate computing, l'algoritmo Kmeans, la compressione dei dati e delle immagini, per poi arrivare all'implementazione vera e propria e alle conclusioni che siamo riusciti a trarre dai vari test fatti durante l'elaborato e dai dati che abbiamo raccolto.

2. Obiettivi

L’obiettivo che si vuole conseguire con il presente elaborato è quello di sviluppare una camera smart che acquisisce e trasmette immagini. Abbiamo pensato come ambito applicativo una camera che viene posizionata in un luogo abbastanza vincolato e difficile da raggiungere, quindi con scarse risorse per quanto riguarda banda di trasmissione e alimentazione del nostro dispositivo. Quindi può essere ad esempio usata in una foresta per poter monitorare quello che è lo stato di certi alberi o la presenza di particolari animali.

Per fare ciò useremo una delle tecniche trattate a lezione per quanto riguarda l’approximate computing, grazie a questo riusciremo ad avere immagini che vengono elaborate prima di essere trasmesse.

Le immagini che sono trasmesse non sono immagini normali ma vengono prima processate tramite l’uso dell’algoritmo Kmeans che servirà per eseguire una quantizzazione delle immagini e quindi una riduzione del numero di colori, ma verrà spiegato più nel dettaglio successivamente.

L’obbiettivo che ci poniamo è quello di riuscire a ridurre la dimensione dell’immagine, e quindi comprimerla, mantenendo comunque una buona qualità di quest’ultima.

Abbiamo deciso di affrontare questo problema in quanto uno degli indici maggiori di consumo nei dispositivi sono le trasmissioni, soprattutto nei dispositivi che riguardano l’IOT e che spesso sono molto limitati nelle dimensioni e anche nei dispositivi che le alimentano, di solito batterie appunto molto piccole.

Quello che ci proponiamo di fare è ridurre la dimensione dell’immagine per poter diminuire il tempo di trasmissione e/o il data rate di trasmissione, in quanto variando questi due parametri possiamo avere enormi vantaggi per la durata del nostro dispositivo.

Inoltre, vogliamo fare in modo di poter diminuire lo spazio di archiviazione nei dispositivi per renderli ancora più piccoli, in quanto le dimensioni delle immagini sono ridotte, e quindi potremmo diminuire lo spazio di

archiviazione o lasciarlo invariato e quindi aumentare il numero di immagini salvati, questo non vale solo per il nodo che si trova all'edge della rete ma anche per un possibile server al quale inviamo le nostre immagini dopo la compressione.

Per far questo terremo in considerazioni diversi parametri, come primo andremo a valutare il tempo impiegato per poter eseguire l'algoritmo Kmeans al variare del numero di cluster e il numero delle iterazioni.

Poi valuteremo la dimensione dell'immagine, considerando sia l'immagine compressa che l'immagine ricostruita attraverso l'immagine compressa (successivamente verrà spiegato nel dettaglio questo aspetto).

Infine, valuteremo la qualità delle immagini ottenute tramite due parametri fondamentali quando si parla di compressione e qualità delle immagini che sono PSNR e SSIM, quello che andranno a considerare queste metriche è la somiglianza dell'immagine finale con quella di partenza.

Tutti questi dati che saranno ottenuti verranno riportati con i relativi grafici per poter semplificare maggiormente la comprensione dei risultati ottenuti.

Inseriamo inoltre l’immagine che abbiamo usato come campione per provare l’algoritmo Kmeans. Quest’immagine è di dimensione 1200X700 pixel per un peso di circa 1MB. Durante la relazione saranno inserite diverse immagini per poter fare un confronto diretto con l’originale.

Figure 1 Immagine originale 1200x700

3. Approximate Computing

Con l'aumentare del numero di informazioni scambiate e nodi dell'Internet of Things, il consumo energetico sta diventando sempre più un argomento centrale, poiché questo continua ad aumentare sempre di più, nonostante i continui progressi nelle tecnologie dei semiconduttori e l'uso sempre più frequente di tecniche di progettazione efficienti dal punto di vista energetico. Si cerca infatti oggi, di trovare dei metodi per abbattere questo impatto energetico.

Nel corso degli anni i nostri sistemi di calcolo si sono basati sulla precisione dei dati, nel cercare di avere una massima precisione da ogni calcolo per singolo step. Tuttavia, con l'avvento dei Big Data e dell'Internet of Things, sentiamo l'esigenza di trovare una soluzione alternativa al calcolo tradizionale, che comporti un consumo energetico decisamente inferiore.

Un approccio innovativo è quello di utilizzare le metodologie dell'Approximate Computing, il cui obiettivo mira a non trovare necessariamente la precisione del risultato di calcolo in ogni singolo step, che ci dà un risultato ottimale, ma a volte è sufficiente rilassare il calcolo per trovare un'approssimazione per ottenere un risultato, impreciso ma accurato, che viene comunque considerato un risultato accettabile.

L'Approximate Computing viene utilizzato per applicazioni in cui un risultato approssimato è sufficiente per il suo scopo. Rinunciamo quindi alla soluzione unica, perfetta, trovata step by step, in modo da snellire il processo di calcolo, riuscendo ad ottenere un risultato accurato riducendo il consumo energetico ed ottenendo prestazioni elevate e scalabili.

L'Approximate Computing mette a disposizione numerose tecniche che ci permettono di migliorare l'efficienza energetica e migliorare le performance di calcolo, al costo di non ottenere un risultato "perfetto". A livello hardware possiamo utilizzare dei circuiti meno accurati ma efficienti dal punto di vista energetico, oppure ridurre la tensione di alimentazione, mentre a livello software possiamo ignorare determinati calcoli o ignorare determinati accessi alla memoria.

Esempi di queste tecniche software sono il Loop Perforation, tecnica che ci permette di eliminare alcuni calcoli richiesti dall'applicazione, saltando alcune iterazioni. Saltare alcuni loop infatti ci permette di rendere il codice più veloce e più efficiente. Un'altra tecnica, che viene utilizzata in questo elaborato, è quella dell'algoritmo K-means, che permette di clusterizzare i dati simili, consentendoci di avere una perdita di accuratezza che fornisce un risparmio energetico significativo rispetto alla soluzione completamente accurata.

3.1 K-means clustering

Il K-means clustering è un metodo di clustering partizionale attraverso il quale i dati vengono divisi in un insieme di K gruppi, chiamati cluster, dove con K viene indicato il numero di cluster prestabilito che si intende creare. L'obiettivo è quello di determinare i K gruppi di dati generati da distribuzioni gaussiane.

Ogni cluster viene identificato mediante un centroide o punto medio. L'idea alla base dell'algoritmo è quella di definire i cluster in modo da minimizzare la varianza intra-cluster, che è la somma dei quadrati delle distanze tra gli elementi di un cluster e il relativo centroide.

L'algoritmo K-means clustering è iterativo e tenta di partizionare il set di dati in K sottogruppi distinti in cui ogni punto appartiene a un solo gruppo. Cerca di rendere i punti di dati all'interno del cluster il più simili possibili, mantenendo allo stesso tempo i cluster diversi il più lontano possibile. Minore è la varianza intra-cluster, più omogenei saranno i dati che si trovano all'interno dello stesso cluster.

La prima fase dell'algoritmo consiste nel decidere un numero K di partizioni che si vogliono creare e inizializzare i centroidi scegliendo in maniera casuale dal set di dati. Successivamente si passa alla fase di assegnazione, che consiste nell'assegnare ogni punto dati, tra quelli rimanenti, ad un cluster il cui centroide è più vicino ad esso (considerando la distanza euclidea). A questa segue la fase di aggiornamento dei centroidi, fase in cui vengono calcolati i nuovi centroidi di ciascun cluster scegliendo il valore medio dei punti dati. Una

volta calcolati tutti i centroidi, tutti i rimanenti punti dati vengono nuovamente assegnati basandosi sui centroidi ricalcolati.

Queste fasi vengono eseguite in maniera iterativa fino a quando o l'algoritmo converge, il che significa che i cluster ottenuti nella corrente iterazione corrispondono a quelli ottenuti dalla iterazione precedente, oppure viene raggiunto il numero massimo di iterazioni indicato.

Giusto per dare un'idea dei cluster dei punti dati, di seguito viene inserita un'immagine per rappresentare i punti dati, prima e dopo il clustering di K-means.

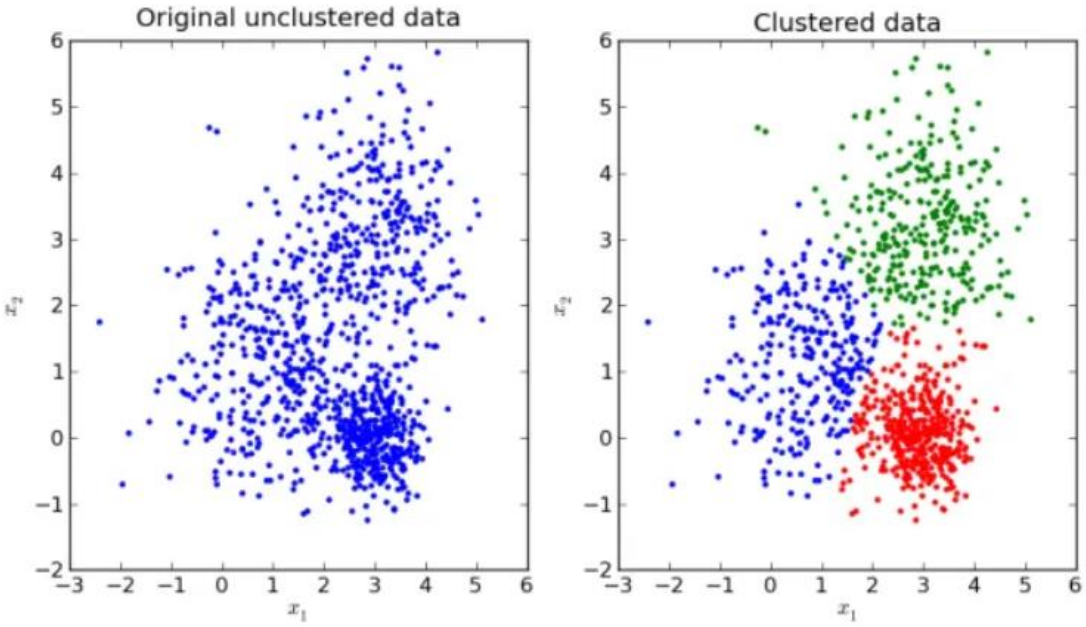


Figure 2 Esempio di K-means clustering con K=3

Il problema può essere formulato nel seguente modo:

Consideriamo di avere *N* oggetti diversi:

$$X = (x_1, \ldots, x_N)$$

dove ciascun oggetto x_i viene definito dai valori, modellizzati come vettori in uno spazio i-dimensionale.

Definiamo una partizione *P* di *X* come:

$$P_k = (p_1, \ldots, p_k)$$

che soddisfa tali proprietà:

- $\bigcup_{1}^{K} P_i = X$:l'unione dei cluster deve contenere tutti gli oggetti;
- $Pi \cap Pj = \varnothing,\ i \neq j$: ogni oggetto può appartenere a un solo cluster;
- $\varnothing \subset P_i \subset X$: almeno un oggetto deve appartenere ad un cluster e nessun cluster può contenere tutti gli oggetti.

Inoltre, deve essere valido che $1 < K < N$, poiché non ha senso avere un solo cluster o avere un numero di cluster pari al numero degli oggetti.

Una partizione viene rappresentata attraverso una matrice $U \in \mathbb{N}^{KxN}$, dove l'elemento $u_{i,j}$ indica l'appartenenza dell'oggetto j al cluster i.

Definiamo con $C = C_1, C_2, \ldots, C_K$ l'insieme dei K centroidi.

Possiamo definire la distanza euclidea tra gli elementi di un cluster e il relativo centroide:

$$V(U,C) = \sum_{i=1}^{K} \sum_{X_j \in P_i} ||X_j - C_i||^2$$

e di questa calcoliamo il minimo seguendo la procedura iterativa definita precedentemente:

1. Partiamo con U_v e C_v scelti in maniera casuale;
2. Calcoliamo U_n che minimizza $V(U,C_v)$
3. Calcoliamo C_n che minimizza $V(U_v,C)$
4. Ripetiamo fino alla convergenza dell'algoritmo.

È stato dimostrato che questo algoritmo converge molto rapidamente, cioè che il numero di iterazioni è generalmente minore del numero dei punti dati e, grazie alla sua velocità, è possibile applicarlo più volte in

modo da poter scegliere la soluzione più accurata tra le soluzioni prodotte.

Più avanti verrà illustrato un esempio di utilizzo dell'algoritmo K-means clustering per la compressione delle immagini.

4. Compressione dei dati

Con il termine compressione dei dati, si fa riferimento alla tecnica di elaborazione dati che, per mezzo di opportuni algoritmi, permette la riduzione della quantità bit necessari per la rappresentazione in forma digitale dell'informazione.

La compressione dei dati nasce dal bisogno di ridurre le dimensioni di un file, con la conseguente riduzione dello spazio necessario per la sua memorizzazione e riduzione della larghezza di banda necessaria per la sua trasmissione digitale.

Oggi più che mai è infatti necessario porre un limite alle dimensioni dei dati, a causa dell'enorme quantità di dati che dobbiamo gestire, che portano numerosi problemi, sia in termini di memoria nei calcolatori, sia in termini di tempo speso durante le operazioni di invio e ricezione dei dati e sulla qualità dei segnali.

Esistono diverse tecniche di compressione dei dati che ci permettono di ottenere una rappresentazione dei dati più compatta che ci consente di consumare meno risorse per la sua trasmissione ed elaborazione, a scapito di una inevitabile perdita dell'informazione.

Lo scopo della compressione è quello di fornire una rappresentazione dei dati con un numero ridotto di bit, senza che il significato degli stessi venga cambiato. La compressione dei dati può comportare anche un elevato consumo di potenza necessaria per le operazioni di compressione e decompressione.

Nello specifico, esistono due tipi di compressioni dati:

- Compressione con perdita
- Compressione senza perdita

4.1 Compressione con perdita

Si definisce compressione con perdita, chiamata anche compressione dati lossy, un qualsiasi algoritmo di compressione dati che permette di ottenere, tramite la decodifica della codifica dell'input, non i dati iniziali, ma bensì una loro approssimazione.

Quindi questi algoritmi portano alla perdita di parte dell'informazione originale poiché eliminano le informazioni che vengono ritenute inutili, poco rilevanti, mantenendo invece le informazioni considerate essenziali.

Una caratteristica di questo tipo di compressione è che le informazioni, una volta perse, non saranno più recuperabili, sono utili quindi nei casi in cui non è necessario esattamente l'input fornito, ma ci si accontenta di una sua ricostruzione fedele.

Se si provasse a decomprimere un file compresso con metodo lossy, la copia decompressa ottenuta sarà peggiore dell'originale, ma comunque sarà abbastanza simile da non comportare perdita di informazioni rilevanti.

Un esempio di applicazione di tali algoritmi si può avere nel caso della compressione di immagini, in cui tramite compressione, possiamo ottenere un'immagine che si differenzia dall'immagine originale in qualche pixel, in modo che l'occhio umano non si accorga della differenza. Viene inoltre fortemente utilizzata nei servizi streaming e su Internet.

4.2 Compressione senza perdita

Si definisce compressione dei dati senza perdita, chiamata anche compressione dati lossless, un qualsiasi algoritmo che ci permette di ottenere, tramite la decodifica della codifica dell'input, esattamente i dati di partenza.

Questa classe di algoritmi di compressione ci permettono di ridurre la grandezza di un file, senza che ci sia perdita di alcuna parte dell'informazione originale durante sia nella fase di compressione, che nella successiva fase di decompressione dei dati stessi.

I due file, quello originale e quello risultante dal processo di compressione, saranno identici dal punto di vista visivo, quello che cambierà invece sarà il loro impatto sulla memoria richiesta per l'archiviazione degli stessi.

Questi algoritmi vengono utilizzati in tutte quelle situazioni dove l'integrità del contenuto è fondamentale, come nel caso della compressione dei testi, poiché in questi casi, se dovessimo perdere anche solo un bit dell'informazione iniziale, questi potrebbero diventare incomprensibili. Vengono utilizzati inoltre, per i programmi, ma anche per le immagini anche se in realtà in questo ultimo caso non sono molto efficienti in quanto l'occhio umano non riesce a percepire piccole variazioni di colore tra pixel adiacenti, quindi un algoritmo di tipo lossy è preferito per le immagini.

Gli algoritmi lossy però non ci danno l'assoluta certezza che a seguito del processo di compressione il dato diminuisca di dimensione, esistono infatti particolari dati che non ridurranno la loro dimensione a seguito del processamento dell'algoritmo, questo risultato è stato dimostrato matematicamente.

5. Compressione delle immagini

5.1 Introduzione alle immagini digitali

Dal punto di vista fisico, un'immagine digitale è rappresentata da un vettore 2-D di elementi, che vengono detti pixel, in cui abbiamo le due coordinate che ci rappresentano, rispettivamente, la larghezza di un'immagine e l'altezza. Questi pixel vengono disposti in una matrice per formare l'immagine, ogni pixel può usare uno o più bit per definire un colore per quel determinato punto, maggiore è il numero di bit che verranno usati per rappresentare ogni pixel, maggiore sarà la precisione dell'immagine, precisione che determina inoltre il numero di colori che l'immagine può avere.

Possiamo avere diverse tipologie di immagini:

- Nelle immagini in bianco e nero, i colori disponibili sono solo due, bianco o nero, quindi la quantità di informazione necessaria per la rappresentazione è di 1 bit per pixel, 0=bianco, 1=nero, per cui nella posizione (i,j) in matrice, avremo come valore 0 oppure 1.
- Nelle immagini in scala di grigio, i colori disponibili sono più di due, corrispondenti a diverse "percentuali" di nero, per cui la quantità di informazione necessaria per la loro rappresentazione è di 8 bit per pixel, che equivale ad avere 256 valori, per cui nella posizione (i,j) ci sarà un valore compreso tra 0=bianco e 255=nero.
- Nelle immagini a colori invece, un colore visibile all'occhio umano, può essere descritto da una composizione di luce dei tre colori primari, rossa, verde o blu, RGB, per cui un determinato colore verrà descritto da una combinazione di questi tre colori primari.
  Associamo così un determinato numero per ciascun livello di intensità, per riuscire ad avere una rappresentazione digitale di un colore.
  In questo caso, per ogni colore, utilizziamo 8 bit, poiché in totale i colori sono tre, avremmo 24 bit per la rappresentazione. In questo caso in posizione (i,j) avremo una terna (x, y, z) con x, y, z che possono assumere valori compresi tra 0 e 255.

5.2 Compressione di immagini digitali

Sotto il nome di compressione delle immagini, vengono raggruppate tutte quelle tecniche di elaborazione digitale delle immagini ed algoritmi che appartengono alle tecniche di compressione dati, tecniche che vengono utilizzate per ridurre la dimensione in byte delle immagini digitali.
Le prime tecniche di compressione, poiché i primi computer avevano una potenza di calcolo limitata, erano molto semplici così si potevano riuscire a ottenere dei tempi di decompressione che fossero accettabili per i computer di quel tempo. Esempi di queste tecniche, che ci permettevano di non avere perdita dei dati, prevedevano la memorizzazione di zone di colore, uguali nell'immagine, tramite delle stringhe che memorizzavano il numero di punti da colorare con lo stesso colore. Questo era possibile poiché inizialmente le immagini avevano un numero limitato di colori e quindi queste tecniche traevano vantaggio da questo limite.
Tra i formati lossless abbiamo: png, tga, tiff, gif.
Successivamente, con il continuo crescere della potenza di calcolo dei computer, siamo oggi in grado di gestire immagini con milioni di colori. Quindi l'assunto del numero ridotto dei colori falliva, così vennero sviluppate delle nuove tecniche di compressione, che, rispetto alle precedenti, erano a perdita di informazione. Tra i vari metodi, il più diffuso divenne il jpeg.

Nella nostra applicazione, la cui implementazione viene descritta nel dettaglio nel successivo capitolo, useremo una tecnica di compressione per la quale riusciremo a comprimere un'immagine usando la quantizzazione del colore, sebbene la compressione risultante risulti in perdita di informazione.

6. Aspetti implementativi

Il nostro problema di compressione delle immagini si pone l'obiettivo di usare il K-means clustering per raggruppare i colori simili in un numero K di cluster.

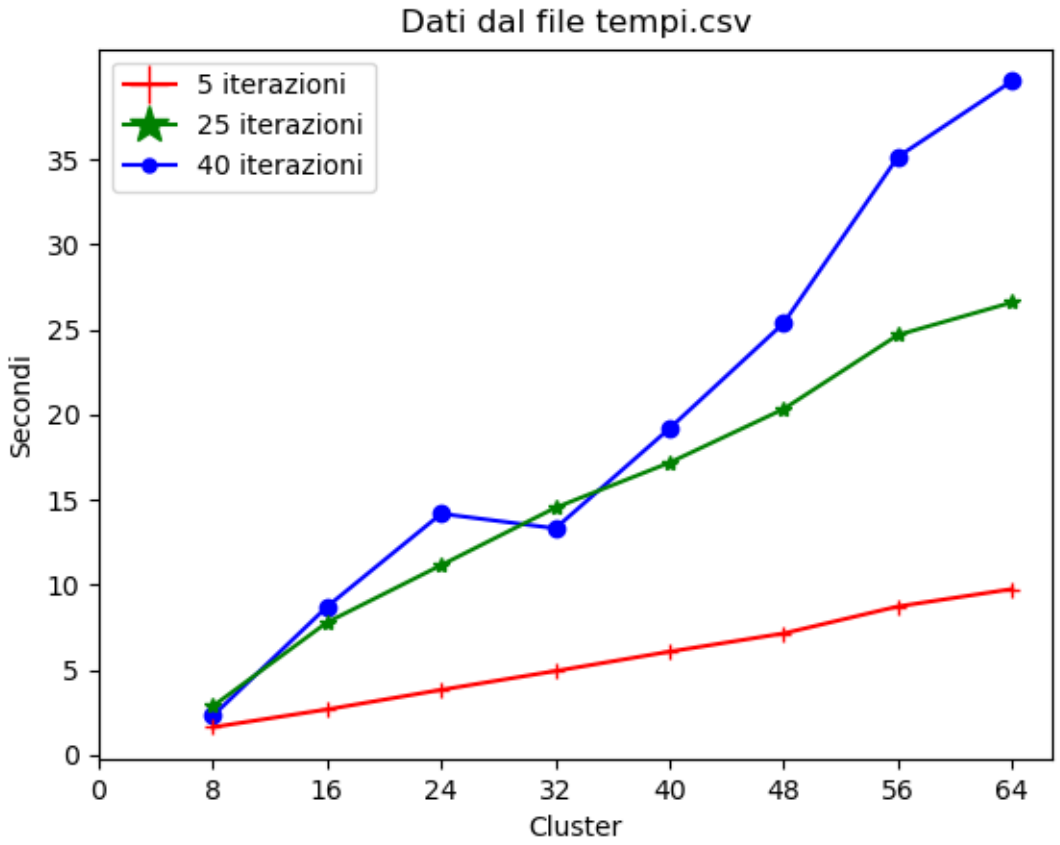
Per la realizzazione dell'elaborato abbiamo usato come linguaggio di programmazione Python. Per usare l'algoritmo Kmeans abbiamo usato la libreria sklearn.cluster dalla quale abbiamo importato per l'appunto Kmeans. Grazie a questa funzione abbiamo potuto impostare il numero di cluster che volevamo usare e il numero di iterazioni per calcolare i centroidi, in modo tale che abbiamo potuto fare diversi test al variare di questi due parametri.

Più precisamente quello che abbiamo consiste in diversi passaggi:

1. Per prima cosa abbiamo selezionato un'immagine di riferimento, che verrà anche usata come confronto per le immagini finali;
2. Calcolare Kmeans in base ai cluster e alle iterazioni selezionate, e quindi trovare i centroidi all'interno dell'immagine campione (i centroidi contengono una media dei colori vicini);
3. A questo punto si passa al salvataggio di due file, uno contiene una nuova immagine compressa dove ogni pixel non è altro che l'etichetta che fa riferimento a un cluster, e il secondo file le etichette con il relativo colore rgb che successivamente verrà usato per ricostruire l'immagine;

Come metrica in questa prima fase abbiamo considerato il tempo di computazione dell'algoritmo Kmeans, quindi per avere un buon riferimento abbiamo fatto diversi test dove abbiamo variato il numero di cluster da un minimo di 8 a un massimo di 64 a step di 8, e il numero di iterazioni da un minimo di 5 a un massimo di 50 a step di 5, per un totale di 80 prove.

Riportiamo il grafico dell'andamento dei tempi al variare di questi due valori:



Come si può capire dal grafico e come ci aspettavamo al variare del numero di cluster il tempo di computazione aumenta, anche se teniamo il numero di cluster fisso e aumentiamo le iterazioni il tempo aumenta. Come si vede però può capitare che il numero di iterazioni è maggiore ma si verifica un tempo di computazione minore, questo capita perché a volte l'algoritmo che calcola i centroidi converge prima del numero massimo di iterazioni selezionato (se le iterazioni sono molte sotto le 35 iterazioni difficilmente capita).

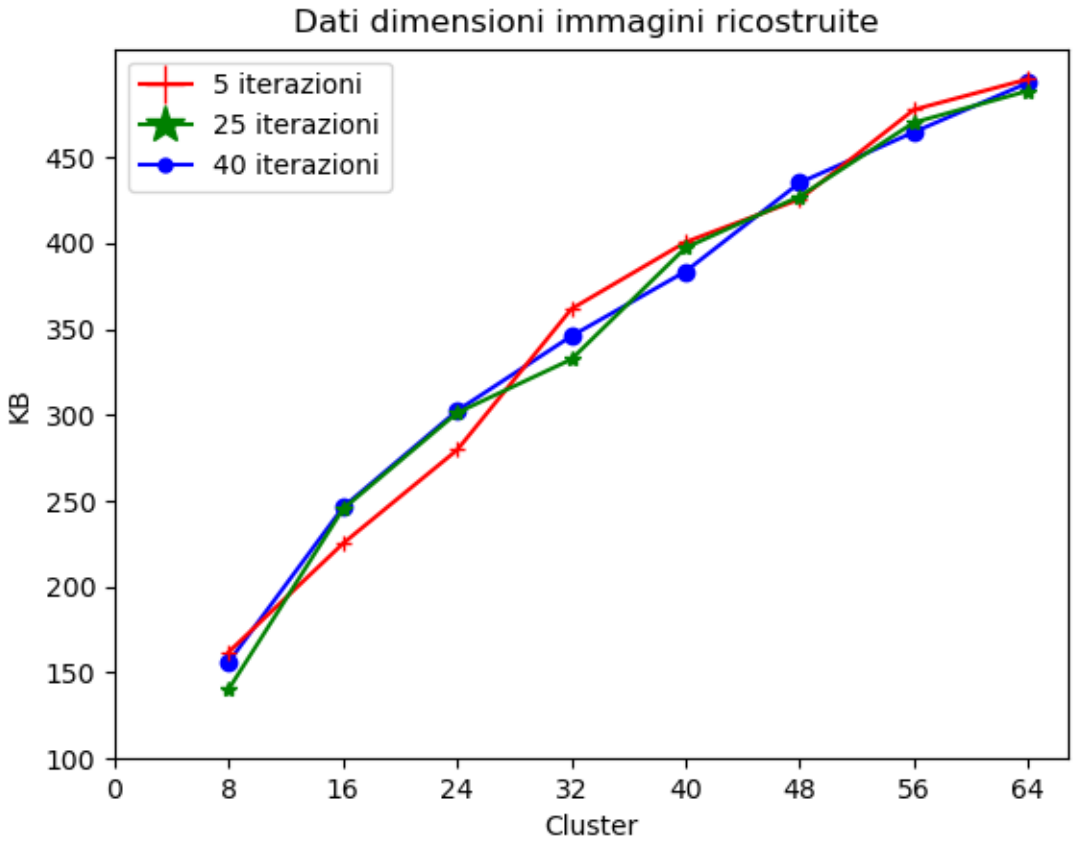
Successivamente a questa fase abbiamo ricostruito le immagini partendo dai due file salvati in precedenza che sono il codebook e l'immagine che contiene al posto del pixel rgb l'etichetta del cluster. L'algoritmo non fa altro che creare una nuova immagine dove nella posizione (i,j) inserisce un colore rgb. Questo colore rgb lo ricava dal valore che è presente nella posizione (i,j) dell'immagine che contiene le etichette, infatti dall'etichetta

che è presente nella posizione (i.j) prende il corrispettivo valore rgb che è presente nel codebook. Ad esempio, se sto ricostruendo l'immagine e sono al pixel in posizione (20,50) vado nel corrispettivo pixel (20,50) dell'immagine con le etichette, in quella posizione troverò un valore che andrà da 0 al numero di cluster massimo che ho selezionato, ad esempio in quella posizione c'è 20, quindi controllo nella posizione 20 il colore presente che in questo caso sarà rgb e lo metto nell'immagine che sto ricostruendo.

Questa fase di ricostruzione si presta bene anche a un'altra tecnica di approximate computing che sarebbe il loop perforation, in quanto durante la costruzione è possibile saltare qualche esecuzione per poterla ricostruire più velocemente, ma questo aspetto non è stato trattato in questo elaborato ma si presta per implementazioni future.

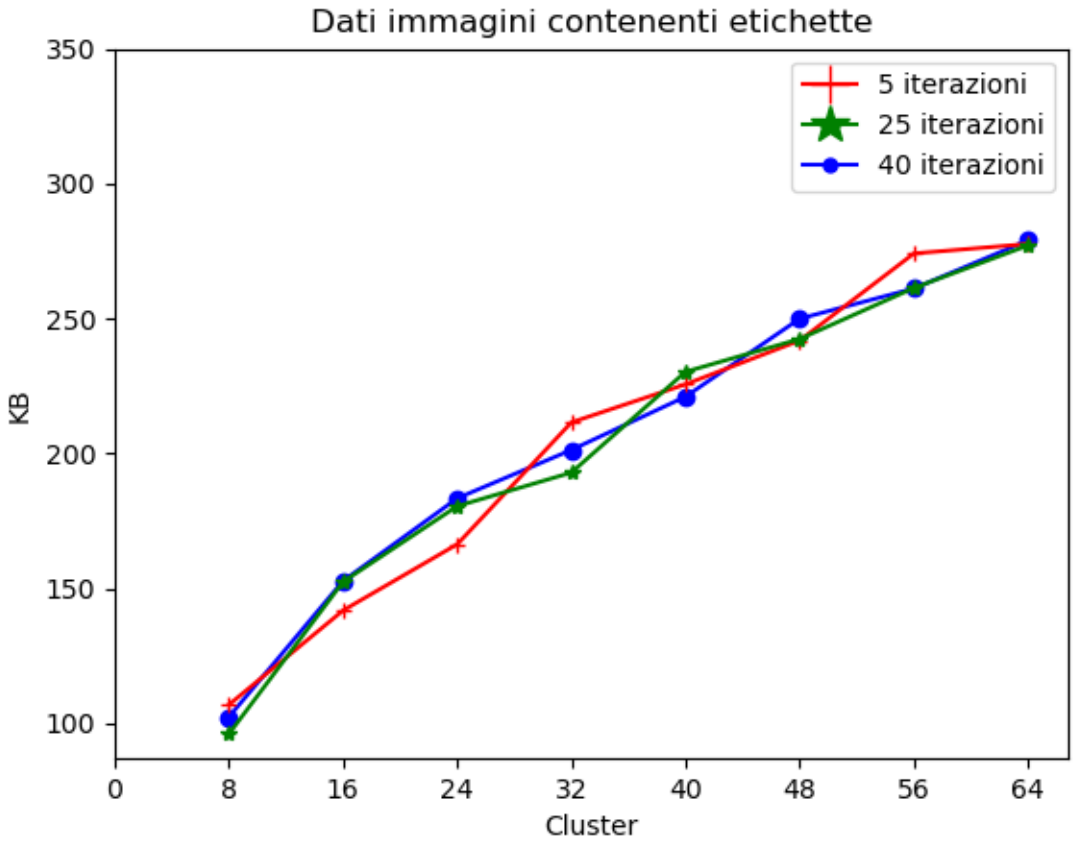
Abbiamo quindi ricostruito tutte le immagini delle quali in precedenza abbiamo calcolato i centroidi, quindi in totale 80 immagini.

Di queste come prima metrica abbiamo considerato la dimensione, non solo quella dell'immagine ricostruita ma anche la dimensione dell'immagine contenente solo le etichette. Non ci siamo concentrati sulla dimensione del file codebook che contiene i colori rgb in quanto ha sempre delle dimensioni irrisorie in confronto a quella delle immagini.



Come possiamo notare il parametro che influenza maggiormente la dimensione dell'immagine è sicuramente il numero dei cluster che consideriamo, anche perché più cluster ci sono e più colori avremo all'interno dell'immagine.

Per quanto riguarda le immagini che contengono le etichette le dimensioni sono riportate in questo grafico:



Anche in questo caso la dimensione è influenzata dal numero di cluster.

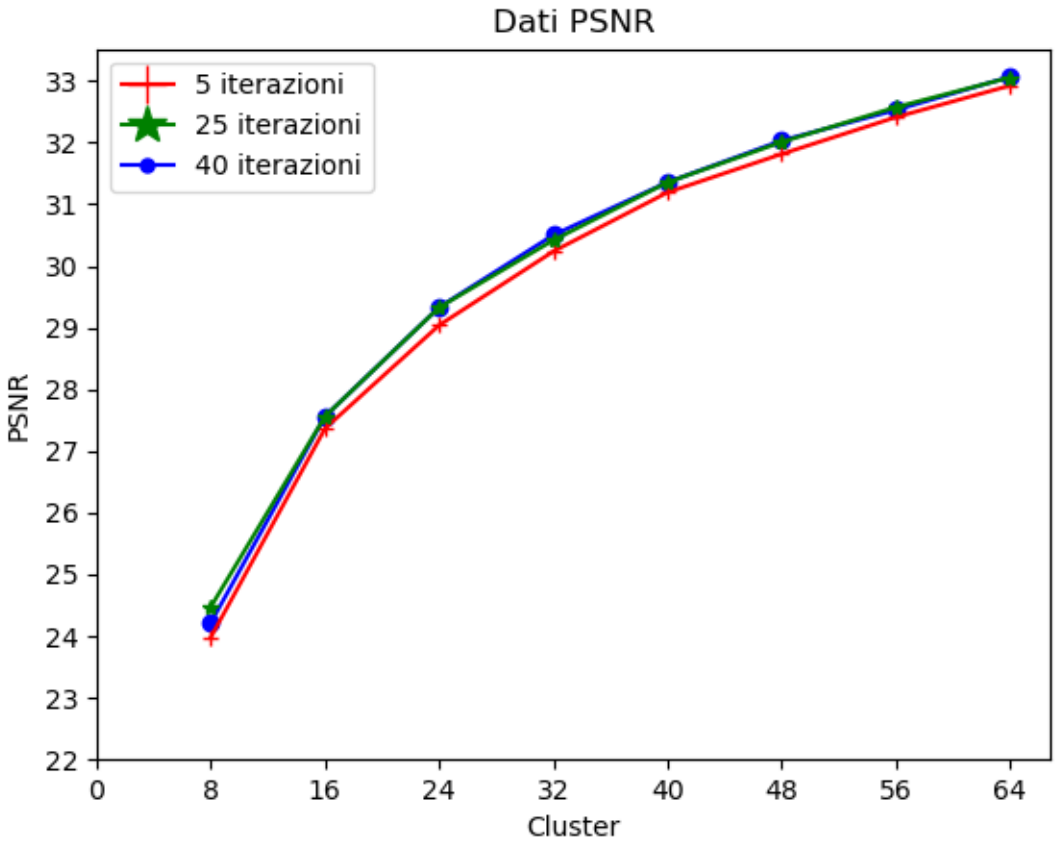
Se confrontiamo i due grafici notiamo che l'immagine più grande pesa quasi 500KB per quanto riguarda le immagini ricostruite, mentre per le immagini con le etichette meno di 300Kb. Questo è un dato importante perché in possiamo direttamente trasmettere l'immagine più leggera per consumare meno energia in trasmissione e per evitare di ricostruire direttamente nel nodo l'immagine, inoltre così anche il server sarà meno carico perché deve conservare immagini con dimensioni ridotte, e possiamo anche ricostruire l'immagine on demand quando serve e non per forza tenerla sempre ricostruita.

L'immagine di partenza era poco più di 1MB quindi notiamo che l'immagine migliore che otteniamo è comunque un terzo della dimensione originale, questo si trasforma in un minor consumo in trasmissione e maggiore spazio di archiviazione, in quanto nello stesso spazio possiamo salvare il triplo delle immagini.

Oltre a queste metriche che sono utile per quanto riguarda i consumi di trasmissione e processamento abbiamo usato anche delle metriche per poter giudicare la qualità dell'immagine.

Più precisamente abbiamo usato PSNR (Peak signal-to-noise ratio) : è una misura adottata per valutare la qualità di una immagine compressa rispetto all'originale. Questo indice di qualità delle immagini è definito come il rapporto tra la massima potenza di un segnale e la potenza di rumore che può invalidare la fedeltà della sua rappresentazione compressa. Poiché molti segnali hanno una gamma dinamica molto ampia, il PSNR è solitamente espresso in termini di scala logaritmica di decibel. Maggiore è il valore del PSNR maggiore è la "somiglianza" con l'immagine originale, nel senso che si "avvicina" maggiormente ad essa da un punto di vista percettivo umano. Tipici valori di PSNR variano da 20 a 40. Il PSNR non è una misura assoluta, nel senso che viene usato per valutare e confrontare due metodi di compressione. Un incremento di 0,25 dB viene in genere considerato una ottimizzazione significativa del metodo di compressione, apprezzabile dal punto di vista percettivo umano.

Riportiamo i dati relativi al PSNR delle immagini che abbiamo ricostruito rispetto a quella originale.

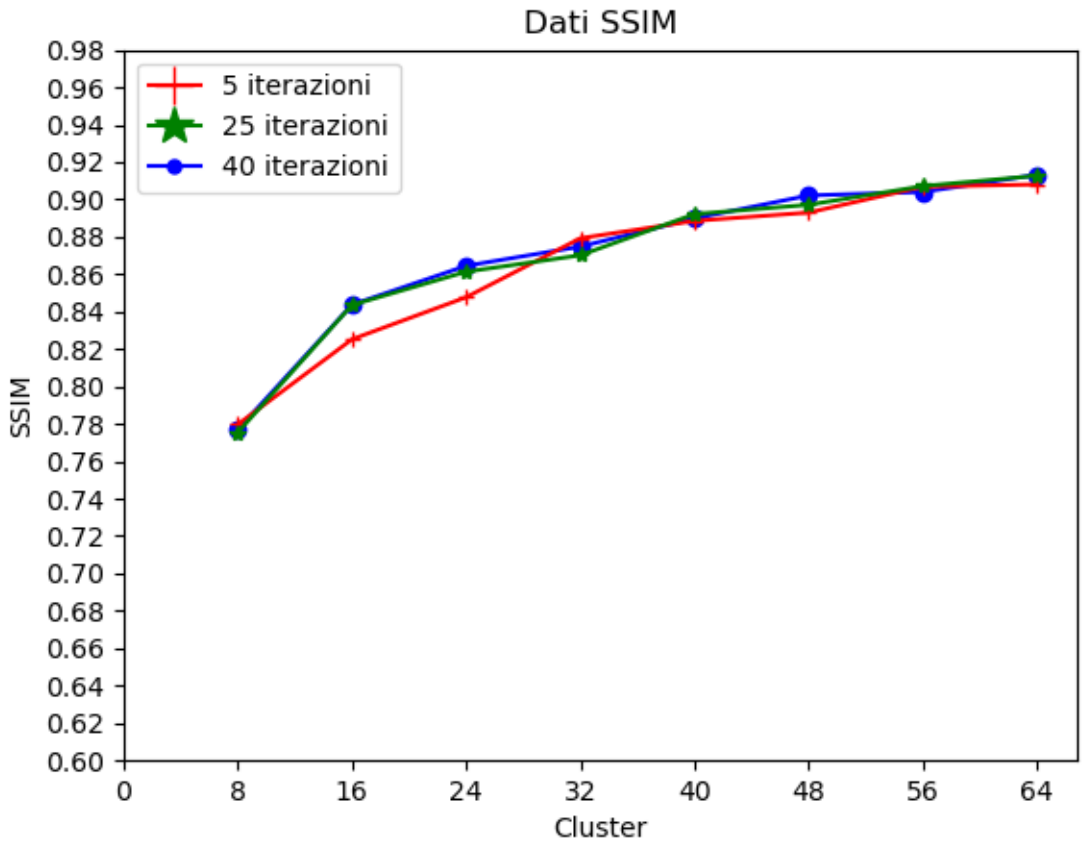


Notiamo ancora una volta che il valore che maggiormente incide sulle immagini è il numero dei cluster, infatti abbiamo valori più alti di PSNR quando il numero di cluster aumenta, queste immagini con PSNR più altro sono anche quelle che occupano più spazio e che quindi hanno una dimensione maggiore.

La seconda metrica che abbiamo considerato per controllare la qualità delle immagini è il SSIM (indice di somiglianza strutturale). SSIM viene utilizzato per misurare la somiglianza tra due immagini. L'indice SSIM è una metrica di riferimento completa ; in altre parole, la misurazione o la previsione della qualità dell'immagine si basa su un'immagine iniziale non compressa o priva di distorsione come riferimento. È una metrica considerata migliore rispetto a PSNR.

Questa metrica a differenza di PSNR varia da un minimo di 0 a un massimo di 1.

Anche di quest'ultima metrica riportiamo i risultati ottenuti:



Come si può notare abbiamo risultati simili a PSNR infatti anche qua all'aumentare del numero dei cluster i SSIM migliora.

Per fare un confronto riportiamo alcune immagini che abbiamo ricostruito durante i vari test per paragonarle con l'originale. Sono state ottenute tutte e tre considerando 50 iterazioni, ma fanno riferimento rispettivamente a 8 24 e 64 cluster.

Figure 3 Immagine ricostruita da 8 cluster

Figure 4 Immagine ricostruita da 24 cluster

Figure 5 Immagine ricostruita da 64 cluster

Come si nota quest'ultima è quella più definita, sebbene abbia perso molte informazioni sul colore rispetto all'immagine iniziale di riferimento, ed è anche quella che ha una dimensione maggiore.

7. Comunicazione

Il principale problema di oggi, quando si progetta una determinata applicazione, è sicuramente quello di cercare di ridurre il costo richiesto dal calcolo computazionale, che riguarda i tempi impiegati per l'esecuzione e il numero di accessi al disco, ma sta crescendo oggi sempre più l'interesse nel cercare di ridurre il costo legato alla comunicazione, che viene invece misurato in termini di numero di messaggi che possono essere scambiati sul canale di comunicazione

Il costo della comunicazione è quindi una misura fondamentale poiché il sistema di comunicazione oggi costituisce spesso il collo di bottiglia delle applicazioni e quindi impatta direttamente sul consumo di energia, sull'utilizzo della larghezza di banda della rete e sul tempo di esecuzione complessivo.

In questo elaborato proponiamo un metodo per il monitoraggio di zone difficilmente raggiungibili, come foreste, attraverso l'uso di una camera che acquisisce, elabora e trasmette immagini verso un server remoto.

Quello che supponiamo di fare, a livello di comunicazione di queste immagini, è quello di cercare di trasferire dati a bassa potenza, con alta velocità e su lunghe distanze.

Oggi le low-power wide-area network (LPWAN) sono un tipo di rete per telecomunicazioni wireless per consentire comunicazioni a lungo raggio a un basso bitrate tra oggetti, come sensori, che operano utilizzando una batteria. Queste sono complementari alle tecnologie wireless a corto raggio come Wi-Fi e Bluetooth Low Energy (BLE), per applicazioni IoT su scala urbana.

Esempi di recenti tecnologie LPWAN che vengono utilizzate nelle applicazioni IoT per ridurre il consumo di potenza sono Long Range (LoRa), NarrowBand IoT (NB-IoT) e Sigfox.

Quello che supponiamo di fare, a livello di comunicazione di queste immagini, è quello di trasferire i dati della camera sul livello fisico basato sul protocollo LoRa. Con questo modello, le immagini che richiedono un'alta velocità di trasferimento dei dati, vengono prima compresse, in

modo da ridurre la dimensione fisica delle stesse, e poi inviate sotto forma di pacchetto dati tramite il livello fisico LoRa o Sigfox.

LoRa e Sigfox si prestano bene per il nostro progetto in quanto sono ormai tecnologie consolidate, ed adatte al trasferimento di piccole quantità di dati con un raggio molto lungo e con un consumo di potenza molto basso rispetto ad altre tecnologie.

Se ad esempio facciamo un confronto con un wifi normale trasferirà i dati molto velocemente ma ci sarà un consumo elevato di energia e non raggiunge distanze molto elevate.

7.1 LoRa

La tecnologia LoRa consente una varietà di applicazioni IoT intelligenti volte a risolvere sfide come la gestione dell'energia, la riduzione delle risorse naturali, il controllo dell'inquinamento, l'efficienza delle infrastrutture, la prevenzione delle catastrofi e altro ancora.

LoRa utilizza bande di radiofrequenza sub-gigahertz libere come 433 MHz, 868 MHz (Europa) e 915 MHz (Nord America). LoRa consente trasmissioni a lungo raggio (oltre 10 km nelle zone rurali, 3–5 km in zone fortemente urbanizzate) a basso consumo energetico. Facendo riferimento ai livelli ISO/OSI, la tecnologia è presentata in due parti: LoRa, lo strato fisico e LoRaWAN (Long Range Wide Area Network), gli strati superiori.

LoRa e LoRaWAN consentono la connettività a lungo raggio per i dispositivi Internet of Things (IoT) in diversi tipi di settori.

LoRaWAN definisce il protocollo di comunicazione e l'architettura del sistema per la rete, mentre il livello fisico LoRa permette il collegamento di comunicazione a lungo raggio. LoRaWAN è anche responsabile della gestione delle frequenze di comunicazione, della velocità dei dati e dell'alimentazione per tutti i dispositivi.

Il punto forte di LoRa è il fatto di poter decidere a quale velocità trasferire i dati, quindi se ad esempio dobbiamo trasferire i dati relativi ai centroidi che hanno dimensioni molto ridotte (circa 1KB) possiamo usare anche il

bitrate più basso e consumare poco, mentre per trasferire l'immagine con le etichette possiamo incrementare il bitrate.

Riportiamo la tabella con i bitrate che è possibile scegliere.

DataRate	Modulation	SF	BW	bit/s
0	LoRa	12	125	250
1	LoRa	11	125	440
2	LoRa	10	125	980
3	LoRa	9	125	1'760
4	LoRa	8	125	3'125
5	LoRa	7	125	5'470
6	LoRa	7	250	11'000
7	FSK 50 kbps			50'000

Possiamo quindi decidere di inviare l'immagine contenente le etichette ad una velocità di 50 kbps e dato che il dato che vogliamo trasferire è varia da circa 100 KB ad un massimo di circa 300 KB il tempo di trasmissione varierà da circa 15 secondi ad un massimo di 45 secondi.

Questo è un ottimo risultato in quanto per trasferire l'immagine di partenza che era superiore ad 1 MB ci sarebbe voluto almeno il triplo del tempo.

Ovviamente grazie alla grande varietà di data rate che mette a disposizione LoRa possiamo scegliere quello che meglio si presta alla nostra applicazione.

7.2 Sigfox

SIGFOX è una rete di telecomunicazione dedicata all' Internet delle cose. Si tratta di una rete LPWA (Low Power Wide Area). La comunicazione viene sempre avviata dal dispositivo. La rete SIGFOX opera su frequenze sub-GHz, sulle bande ISM: 868MHz in Europa. Con un link budget di 162dB, SIGFOX consente comunicazioni a lungo raggio, con una copertura maggiore rispetto al GSM. Il numero massimo di messaggi

che possono essere inviati ogni giorno è di 140, per un totale di 12byte per messaggio.

Questo rende questa tecnologia complementare a LoRa per questo progetto, poiché ha un consumo molto ridotto, anche se non riesce a trasferire una gran quantità di dati giornalmente, quindi è ottimo per poter trasferire il file relativo ai centroidi in quanto ha dimensioni davvero ridotte.

8. Conclusioni e sviluppi futuri

Con questo progetto ci siamo posti l'obiettivo di trovare un modo efficiente e sicuro per poter trasferire dati (in questo caso immagini compresse) cercando di mantenere basso il consumo di potenza, e per questo abbiamo usato delle tecniche di approximate computing, abbiamo pertanto usato l'algoritmo Kmeans per poter comprimere le immagini, cercando di variare il più possibile il numero di cluster e iterazioni per ottenere risultati molto diversi in base alle situazioni.

Grazie alla compressione delle immagini infatti il dato che andremo a trasferire avrà dimensioni molto ridotte e pertanto il tempo di trasmissione sarà molto inferiore e in più si presta egregiamente a molte tecnologie di trasmissione low power destinate proprio per l'ambito IOT.

Non solo grazie ai vari test effettuati abbiamo riscontrato, come mostrato nei grafici precedenti, che la qualità delle immagini resta comunque elevata, sempre in relazione ai numeri di cluster che andiamo a considerare.

Per quanto riguarda i dati relativi ai consumi ci siamo basati effettivamente sui tempi di trasmissione in quanto non avendo un dispositivo sul quale testare la nostra applicazione risultava complicato. Abbiamo anche considerato diverse fonti trovate in rete per considerare quindi i datarate delle varie tecnologie per la comunicazione in ambito IOT, e quindi calcolare i vari tempi di trasmissione.

Poi sempre in maniera teorica abbiamo notato che in base al datarate i tempi di trasmissione aumentavano o diminuivano, anche in base alla qualità dell'immagine che volevamo trasmettere. Soprattutto rispetto all'immagine originale di partenza che era molto grande, infatti i tempi per trasmetterla erano persino il triplo rispetto ad un'immagine compressa.

Per quanto riguarda gli sviluppi futuri, una possibilità potrebbe essere quella di estendere la nostra applicazione utilizzando un dispositivo di trasmissione reale con la quale poter effettuare vari test sul campo, allo scopo di raccogliere dei dati dall'ambiente, in modo da poter analizzare

le metriche relative alla trasmissione, magari anche in contesti ambientali differenti.

In questo elaborato inoltre non sono stati valutati i consumi energetici da parte di queste tecnologie. Il motivo di base è che non si dispone degli strumenti specifici che misurano l'impatto della trasmissione sulle batterie dei dispositivi, in ottica di sviluppi futuri, si può pensare di raccogliere anche le metriche relative al consumo energetico, confrontando infine le varie tecnologie, in termini di tempi di trasmissione e consumi energetici, al fine di determinare quali sono le configurazioni ottimali dei dispositivi con lo scopo di ottenere la massima efficienza che questi protocolli possono offrirci per la trasmissione.