UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Relazione Progetto - Ingegneria della Conoscenza

Urban Flora:
Deep Junk Classification e Ottimizzazione Percorsi di Raccolta

Studenti:
Marco Barnaba , Mat. 699506 , m.barnaba21@studenti.uniba.it
Francesco Coviello, Mat. 676707, f.coviello5@studenti.uniba.it
Gianluca Colasuonno, Mat. 678658, g.colasuonno14@studenti.uniba.it

Link del repository:
https://github.com/francesco-cov/UrbanFlora

Indice dei contenuti:

Introduzione

Il progetto svolto mira a realizzare quello che era stato definito da un progetto precedente, legato all'ambiente e alla tecnologia per Smart City. Le due componenti del progetto, quindi, seppur distaccate rappresentano elementi portanti dell'idea iniziale, illustrata di seguito.

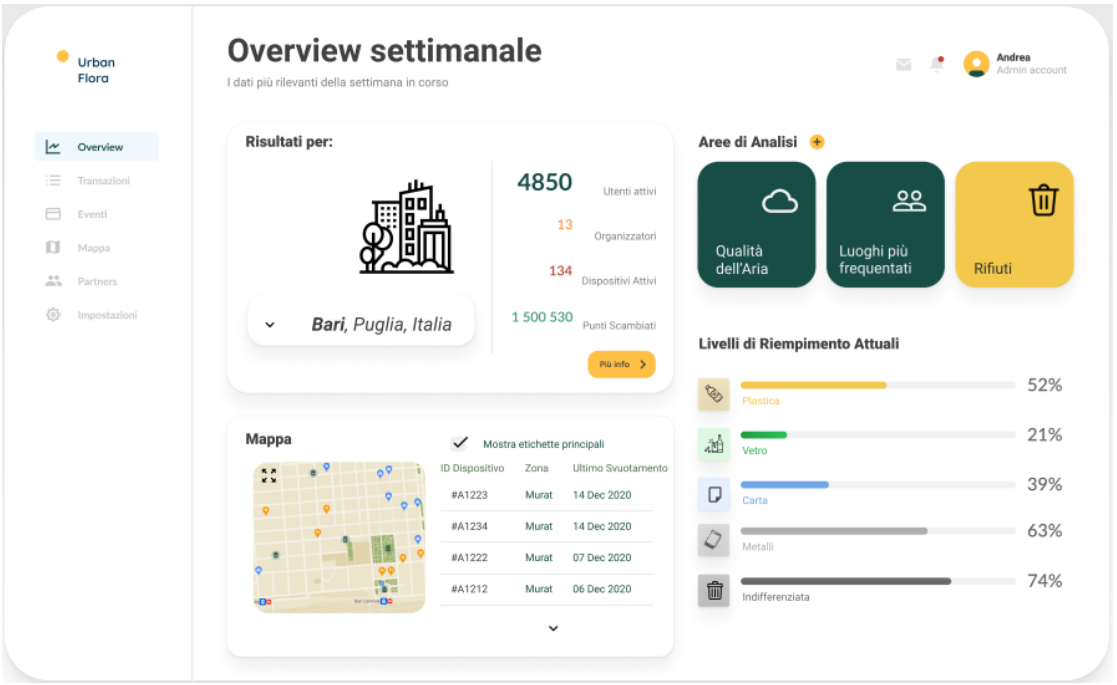
Tutti i particolari riguardo il lavoro svolto possono essere approfonditi consultando i **notebook Colab** allegati a questo documento.

Progetto precedente

Urban Flora è un progetto nato nell'ambito del Samsung Innovation Camp 2021. Agli studenti era stato chiesto di coniugare Internet of Things e Intelligenza Artificiale in un'unica idea.

UF propone una piattaforma che offre servizi sia ai suoi utenti, i cittadini, sia alle amministrazioni che decidono di installare i dispositivi in città.
I primi, attraverso la App Mobile possono interagire con gli Smart Objects sparsi in città o partecipare a eventi, guadagnando punti virtuali riscattabili in ricompense reali presso i negozi eco-friendly della città.

Le amministrazioni, invece, possono disporre di una Dashboard per il monitoraggio dei dati raccolti e per l'analisi degli stessi. I dati in questione provengono dai dispositivi Smart installati nella città, principalmente Cestini Smart dotati di sensori per monitorare la zona e riconoscere i rifiuti all'interno, separandoli correttamente.



Naturalmente i cestini Smart in questione hanno bisogno di un software per il riconoscimento dei rifiuti, prima che questi vengano smistati meccanicamente. Per questo ci siamo occupati di addestrare una Rete Neurale per la *classificazione di immagini* di rifiuti.

D’altro canto, l’eventuale disponibilità di dati come quelli raccolti periodicamente da dispositivi come questi consentirebbe di effettuare un’analisi degli stessi utile a migliorare la qualità dei servizi offerti ai cittadini per l’ente amministrativo. Uno dei modi, il principale, per sfruttare queste informazioni è quello di *ottimizzare la raccolta dei rifiuti* dai cestini, disponendo in tempo reale di informazioni riguardo al riempimento.

Deep Junk Classification

Strumenti utilizzati:

- Tensorflow, Keras
- Python, Google Colab con accelerazione GPU
- Trashnet Dataset

Il punto di partenza è stata una rete convoluzionale dall'architettura classica, definita attraverso il modello Sequenziale:

```
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Effettuando un primo training, risulta evidente l'overfitting della rete, che in *validation* ha prestazioni non in linea con il training.

Proviamo quindi a migliorarla:

- Aggiungiamo un layer di Data Augmentation per migliorare le prestazioni

- Rendiamo l'architettura più profonda, aggiungendo un layer convoluzionale.

- Aggiungiamo un layer di Dropout per ridurre l'overfitting

Il grafico ottenuto mostra risultati nettamente migliori.

A questo punto, la ricerca si è concentrata sulla migliore configurazione del modello, prendendo in considerazione:

- *Percentuale Dropout*, tra il 10% e il 30%
- *Ottimizzatore*, tra i principali forniti da Keras
- *Learning rate* per l' ottimizzatore più performante, tra 0,1 e 0,0001
- *Numero di unità* nell'ultimo layer Denso, tra 32 e 512

In particolare, con il Keras Tuner è stato utilizzato l'algoritmo HyperBand, che mantiene "in vita" incrementalmente i modelli più promettenti.

I valori ottenuti dalla ricerca, in base alla performance, sono i seguenti:

- **Dropout** 0.2
- **Ottimizzatore** Adamax
- **Learning** rate 0.001
- **Numero unità** nell'ultimo layer Denso 480

Ottenuti gli iperparametri migliori, alleniamo il modello per un numero più alto di epoche in modo da individuare il valore migliore anche per queste, in base alla *validation accuracy/loss.*

Di seguito i risultati del modello finale:

Come si evince dai grafici, il modello fornisce una precisione molto vicina a 0.8. Per quanto riguarda la loss, c'è ancora margine di miglioramento dal momento che oltre le 40 epoche smette di decrescere globalmente.

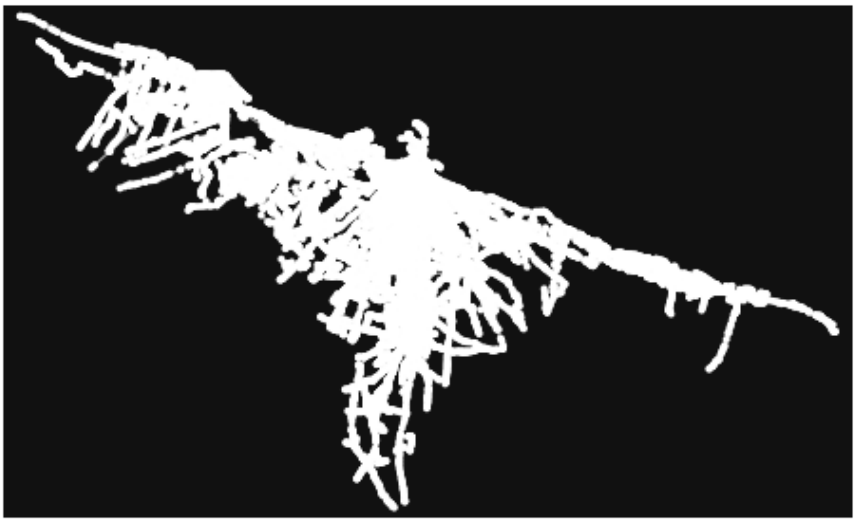
Ottimizzazione percorsi

Si è voluto implementare un modo per **ottimizzare i percorsi** dei veicoli di raccolta rifiuti che partono dal centro di raccolta, visualizzandoli su una mappa interattiva.
Il problema trattato per la realizzazione di questo task è il **Capacitated Vehicle Routing Problem** (**CVRP**): in questo problema dei veicoli con capacità di trasporto limitata devono raccogliere oggetti in più posizioni, ma senza mai eccedere la capacità dei veicoli.
L'ottimizzazione consiste dunque nell'usare il minor numero possibile di veicoli, fornendo per ognuno di essi un percorso diverso per la raccolta di cestini intelligenti, in base al loro riempimento.

Anche qui si è deciso di usare Python con Google Colab, utilizzando i seguenti moduli:

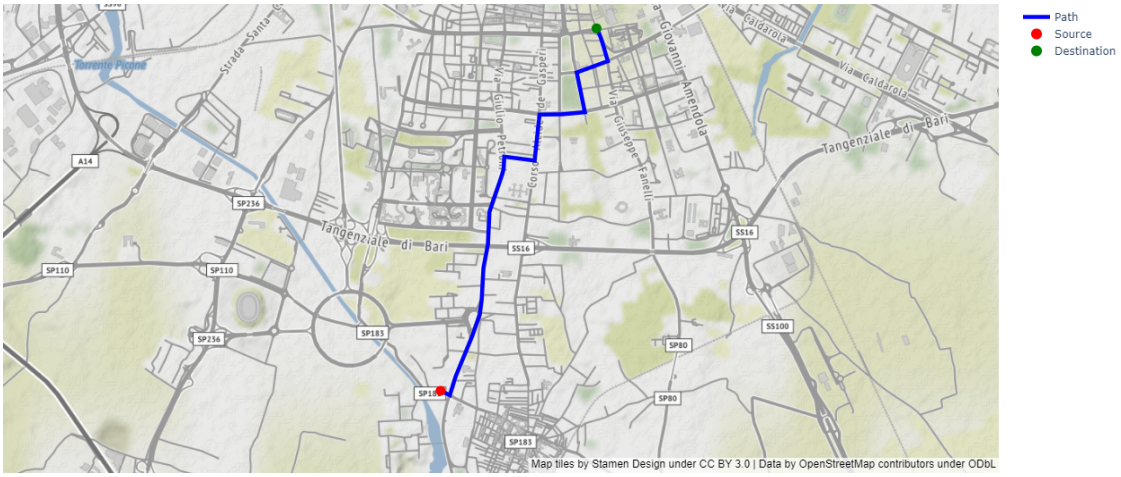
- ***NumPy***, per creare array, generare valori casuali con distribuzione normale e per calcolare la media di una lista
- ***Pandas***, per la gestione di dataframe
- ***Random***, per la generazione di numeri interi casuale
- ***Requests***, per il download di file online
- ***Pyrosm***, per la lettura di file con estensione .pbf
- ***OSMnx***, per modellare e gestire una grafo di una rete stradale con OpenStreetMap
- ***NetworkX***, per la gestione del grafo
- ***Plotly***, per la visualizzazione di nodi e archi stradali su una mappa interattiva
- ***OR-Tools***, per la risoluzione del CVRP

Per eseguire il task, innanzitutto serve inizializzare il grafo su cui lavorare. Viene scaricato da "Estratti OpenStreetMap Italia" un file con estensione .pbf. Questo sito è un servizio per estrarre dati sull'Italia presenti nel database di OpenStreetMap in più formati, aggiornati con frequenza giornaliera. Nel nostro caso effettuiamo il download dei dati relativi alla città di Bari.
Con un oggetto di classe OSM leggiamo il file .pbf e otteniamo nodi e archi relativi alle strade della città di Bari su cui si può guidare; da questi possiamo creare un grafo con NetworkX. Nel grafo generato gli archi rispettano i sensi di marcia reali.



Visualizzazione grafica del grafo della città di Bari

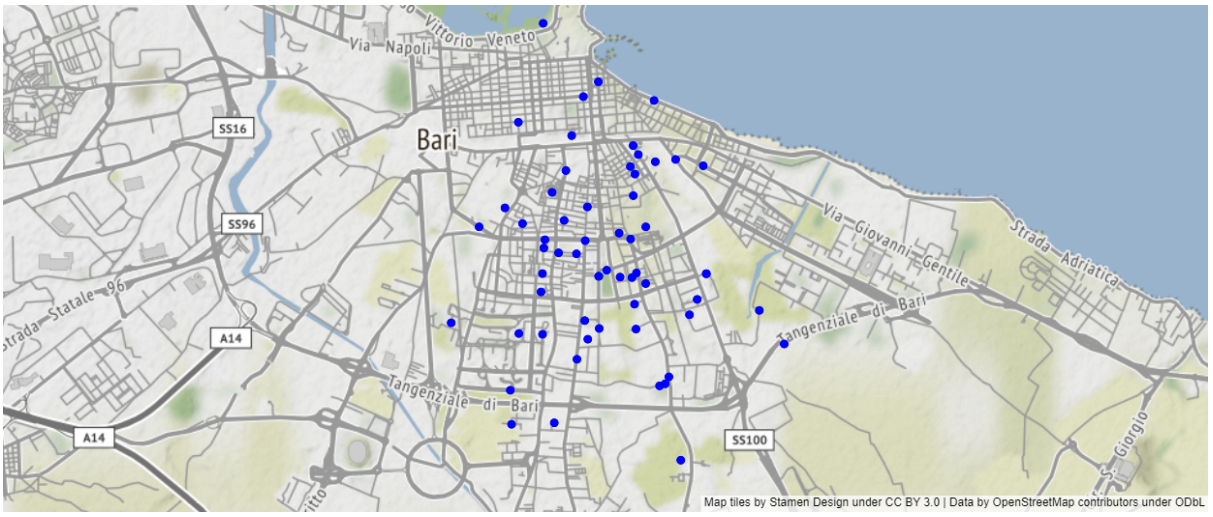
Per l'ottimizzazione del problema useremo l'algoritmo di Dijkstra del modulo NetworkX per trovare il percorso minimo tra due nodi. È stato scelto l'algoritmo di Dijkstra in quanto il più indicato per lo scopo.



Esempio ricerca percorso minimo con Dijkstra

Per risolvere il CVRP sfruttiamo il modulo OR-Tools di Google.
Il problema in input necessita di un numero di nodi da considerare (qui un centro di raccolta e i cestini), il numero massimo di veicoli disponibili e la capacità dei veicoli.
Per la nostra esecuzione abbiamo utilizzato 60 nodi e 6 veicoli con capacità massima di 75. La posizione dei nodi di cestini e centro di raccolta e il riempimento dei cestini sono stati generati in modo casuale. In particolare, la posizione dei nodi viene generata con distribuzione

normale attorno a un punto, questo per fare sì che i nodi siano quanto più possibile equamente distribuiti nel centro cittadino. Il punto centrale è prefissato con coordinate (41.107114,16.874238) per evitare di generare troppi nodi ai confini del grafo da noi considerato.



Esempio generazione nodi

Ottenuti i nodi, per ottenere un modello dei dati del problema, serve calcolare la matrice di distanza dei nodi: questa è una matrice quadrata contenente in ogni posizione la distanza tra due nodi. Questo valore come accennato in precedenza sarà ottenuto utilizzando l'algoritmo di Dijkstra.

Creato il modello di dati si passa alla risoluzione del problema, settando il routing model con distanze tra i nodi e riempimenti dei cestini, oltre a creare una dimensione per la capacità dei singoli veicoli.

Se viene trovata una soluzione con gli input inseriti, vengono stampati per ogni veicolo la distanza percorsa e il carico a fine percorso.
Qui in basso è riportato un esempio di stampa:

Vehicle 0:
Distance of the route: 0m
Load of the route: 0kg

Vehicle 1:
Distance of the route: 0m
Load of the route: 0kg

Vehicle 2:
Distance of the route: 15343m
Load of the route: 70kg

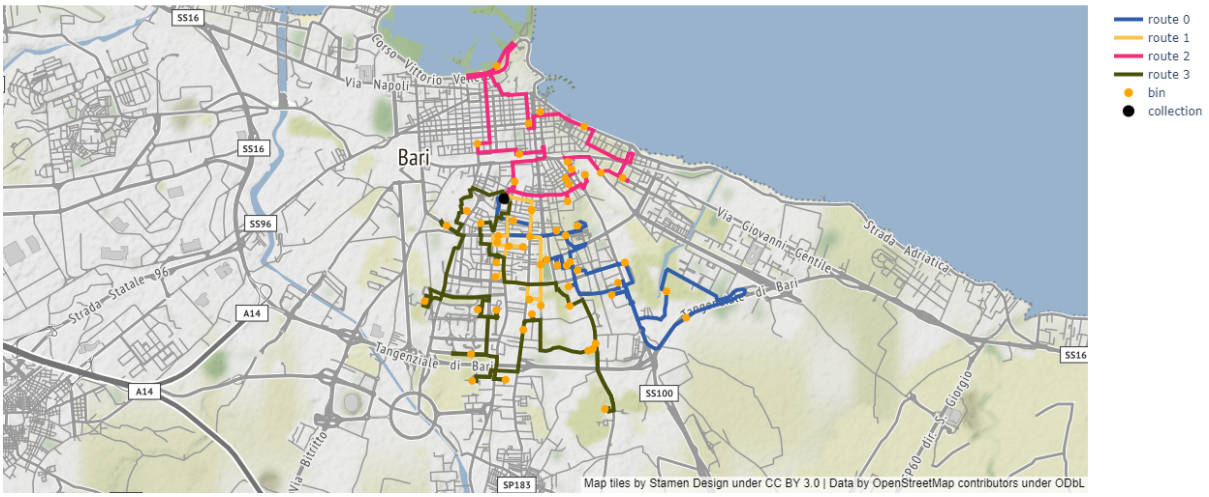
Vehicle 3:
Distance of the route: 5760m
Load of the route: 62kg

Vehicle 4:
Distance of the route: 13947m
Load of the route: 58kg

Vehicle 5:
Distance of the route: 19831m
Load of the route: 74kg

Come si nota dall'esempio, sui 6 veicoli a disposizione ne vengono utilizzati 4, massimizzando quanto possibile il loro carico.

Oltre alla stampa, vengono plottati su una mappa i percorsi ottenuti dalla risoluzione del problema con i nodi dei cestini e il centro di raccolta.



Esempio di risoluzione del CVRP

Cercando di risolvere il problema più volte ma spostando il centro di raccolta notiamo che il numero dei veicoli impiegati non varia, così come non varia la somma dei carichi di tutti i veicoli. Ciò che varia maggiormente è la distanza totale percorsa.

	Es.1 (route)	Es.2 (route)	Es.3 (route)	Es.4 (route)
Vehicle 0	0	0	0	0
Vehicle 1	0	0	0	0
Vehicle 2	15343	21925	22342	16730
Vehicle 3	5760	16184	14171	11676
Vehicle 4	13947	15133	12388	11864
Vehicle 5	19831	8765	7560	16702
SUM	54881	62007	56461	56972

Esempio di confronto su 4 routes con gli stessi cestini ma centri di raccolta differenti

Nell’esempio in alto notiamo come nel secondo tentativo la distanza totale percorsa sia nettamente maggiore rispetto agli altri esempi. Supponendo che ci sia più di un centro di raccolta, si può pensare a come questo sistema possa determinare quale centro sia più efficiente per la raccolta tenendo in considerazione anche la distanza totale percorsa dai veicoli.

Conclusioni

Per quanto riguarda il modello di **Junk Classification**, le prestazioni sono soddisfacenti ma c'è ancora margine di miglioramento.

In futuro, si potrebbe provare a migliorare il modello indagando in diverse direzioni:

- Architetture diverse, con numero e dimensione diversa per i layer
- Regolarizzazione dei pesi
- Utilità di side-information collezionabili attraverso i sensori dei cestini

Invece per quanto riguarda l'**Ottimizzazione dei percorsi** si può provare a creare un modello che tenga conto di più centri di raccolta e veicoli di diversa capacità.