Politecnico di Torino

Dipartimento di Automatica e Informatica

Corso di Laurea Magistrale in Ingegneria Informatica

Progetto e sviluppo di un assistente virtuale in ambito Retail

---

Laureando:
Petrea Tancau

Relatore:
Prof. Elio Piccolo

Anno Accademico 2018/2019

Indice

Capitolo 1

Introduzione

Il lavoro presentato riguarda il progetto e lo sviluppo di un assistente virtuale in ambito retail, in particolare per prodotti di moda. Lo scopo è realizzare, per mezzo di algoritmi di intelligenza artificiale, un consulente di moda personalizzato con il quale interagire in linguaggio naturale ed avere assistenza per la ricerca e l'acquisto di oggetti. L'idea è quella di farne il consulente di fiducia per un amante dello shopping ed illustrare un modo innovativo di interagire con un sistema informatico per effettuare acquisti online.

Tradizionalmente si naviga sul sito web del commerciante, si visualizza il catalogo, si leggono le descrizioni ed una volta scelto un prodotto lo si aggiunge al carrello, mediante un apposito pulsante. Questa procedura risulta poco flessibile e spesso non vi è alcun aiuto da parte del sistema. Per esempio, se vogliamo impostare dei filtri per la ricerca, occorre cercare l'apposita sezione all'interno dell'interfaccia grafica che spesso non è intuitiva. Il catalogo prodotti è talmente vasto che si rischia di navigare per ore prima di trovare il prodotto che stiamo cercando. La ricerca sarebbe molto più semplice se il sistema informatico fosse in grado di comprendere la necessità delle persone attraverso il linguaggio oppure mediante contenuto multimediale, come per esempio il riconoscimento di un prodotto a partire da una fotografia.

Il progetto ha come obiettivo la realizzazione di un sistema informatico con il quale parlare come ad un essere umano, in questo caso come se fosse il commesso nel negozio. Ciò che si vuole ottenere è un assistente in grado di capire una persona che si esprime in linguaggio naturale e nella sua lingua d'uso. Il consulente è capace di ricordare cosa piace al cliente in modo da fornire una esperienza personalizzata imparando a conoscerne lo stile.

Ogni end-user ha dei gusti particolari e si vuole dare importanza a questo

aspetto: i prodotti vengono impostati in base al comportamento passato degli utenti ed il sistema sarà in grado di consigliare prodotti in maniera distinta per ogni persona. Il suggerimento sarà guidato dai dati e quindi il modo di raccomandare prodotti è oggettivo. Tradizionalmente si acquista dal proprio negozio di fiducia e, nel caso in cui si chiede un consiglio, ci si affida all'opinione del commesso. Il suggerimento che ci viene offerto è soggetto ai gusti personali della persona che ci assiste ed è quindi una tecnica soggettiva: non è completamente pensata per il cliente.
Il problema trattato è complesso e inizialmente non definito, si sono dovuti quindi stabilire i requisiti, progettare l'architettura ed infine implementare la soluzione. Parte del lavoro iniziale è stato rivolto alla ricerca di strumenti già esistenti sul mercato per la realizzazione di alcuni componenti, come per esempio la classificazione di immagini (Google Vision API, Face++, Watson Visual Recognition) e strumenti di sviluppo di modelli conversazionali (Google Actions, Amazon Alexa, IBM Watson Assistant).
Gli strumenti che consentono l'elaborazione del linguaggio naturale non forniscono un grafo di conversazione ma soltanto la tecnologia necessaria a riconoscere intenti ed estrarre entità, è necessario quindi costruire un modello di stati rappresentativi di un dialogo e risolvere le problematiche relative alla semantica ed al modo di parlare delle persone. Lo strumento scelto per lo sviluppo del modello di conversazione, Watson Assistant, nonostante sia risultato migliore rispetto alle alternative, non è perfetto: è' stato necessario definire un'architettura ad-hoc per il grafo conversazionale, basata su molteplici sotto-modelli, in modo da massimizzare l'accuratezza dello strumento, nella classificazione di testo.
Gli strumenti disponibili per il riconoscimento visivo permettono l'addestramento di reti neurali convoluzionali ma non risultano sufficientemente accurati per il caso d'uso, è stata sviluppata una soluzione ad-hoc per il riconoscimento di prodotti simili a partire dall'addestramento di reti neurali pre-addestrate su ImageNet, utilizzando il framework PyTorch per la programmazione.
Gli algoritmi di filtro-collaborativo, per l'implementazione di un motore di ricerca, sono disponibili in diverse implementazioni ma è necessario l'addestramento di un modello adatto al caso d'uso, con i parametri ottimizzati sul dataset di feedback, per ottenere una predizione della preferenza dell'utente. Come soluzione al problema di utenti non noti al sistema di raccomandazione, è stata proposta una tecnica basata su Nearest Neighbors.
I componenti del sistema sono eterogenei e sviluppati in diversi linguaggi di programmazione, principalmente Python e JavaScript. Per consentire la

comunicazione degli elementi, sono stati sviluppati servizi web per l'esposizione di RESTful API (prodotti simili, predizione del motore di raccomandazione).

Il sistema ha in definitiva l'obbiettivo di fornire:

- Migliore esperienza di acquisto: interagendo mediante conversazione tramite testo oppure voce è possibile visualizzare il catalogo, cercare prodotti e vederne le caratteristiche, gestire un carrello della spesa e completare un ordine.
- Accessibilità: l'assistente virtuale è in grado di operare sulle piattaforme di messaggistica che supportano i bot. Nell'implementazione attuale la scelta è di comunicare attraverso Telegram, che ha un supporto per mobile e desktop, e Google Assistant per l'accesso da tutti i dispositivi Google. Tuttavia per la modularità con la quale la soluzione è stata progettata è possibile integrare l'assistente con ulteriori canali di comunicazione senza modificare l'architettura del sistema.
- Ricerca di prodotti simili: quante volte ci si è trovati nella situazione che avremmo comprato immediatamente un prodotto se solo lo avessimo trovato con qualche piccola variante? L'assistente virtuale può prendere in input una immagine, capire le caratteristiche del prodotto e suggerire un capo d'abbigliamento simile.
- Raccomandazione: il consulente è capace di presentare prodotti in base alle preferenze dell'end-user. Basandosi sul feedback degli utenti e su algoritmi di Machine Learning è possibile fare una predizione per capire quali prodotti il cliente è maggiormente interessato a vedere, senza necessariamente proporglieli tutti.
- Supporto per domande più frequenti: se un cliente ha un dubbio dovrà soltanto parlare con il sistema.
- Personalizzazione: l'assistente può assumere un carattere femminile o maschile in base alla preferenza dell'utente. Se l'interazione avviene mediante voce il sistema sarà in grado di parlare con una voce adeguata in base a questa scelta. Nella fase iniziale di configurazione, l'assistente cerca di scoprire alcune caratteristiche dell'utente come età, sesso e nome.

- "Teniamoci in contatto": per i canali di comunicazione che consentono l'invio di un messaggio da parte dell'assistente in maniera attiva (ovvero senza che sia stato esplicitamente invocato), è possibile implementare logiche per riprendere una conversazione tenuta in sospeso. Per esempio se passa un lungo periodo di tempo dall'ultimo messaggio ricevuto, il consulente può chiedere in maniera attiva all'utente la posizione e consigliare dei prodotti in base al meteo. Questo è soltanto un esempio, una interazione simpatica può portare il cliente ad avere più fiducia verso il sistema.

Capitolo 2

Problematiche e componenti necessari

La tecnologia che ha permesso l'ascesa dei sistemi di dialogo è il Natural Language Processing. Gli agenti conversazionali rappresentano oggi il modo più intuitivo possibile con cui una persona può comunicare con un sistema informatico in quanto ci si interfaccia mediante linguaggio naturale. Per eseguire una operazione non è più necessario navigare sul web e fare click o seguire perfettamente delle procedure meccaniche ma è sufficiente esprimersi come si farebbe con un'altra persona. La tecnologia è avanzata molto negli anni recenti e sta dando ottimi risultati.
Il secondo fattore che ha portato al successo delle interfacce conversazionali è il fatto che possono operare su piattaforme di messaggistica che già si utilizzano giornalmente, non è necessario scaricare ed installare una nuova applicazione. Di fatto si può avere accesso a diversi servizi attraverso un'unica applicazione che ha una grafica familiare a tutti i possessori di smartphone come per esempio Facebook Messenger o Telegram.
Molti sistemi di dialogo sono stati realizzati in passato, dal semplice chatbot domanda e risposta ad agenti conversazionali ed assistenti virtuali come Amazon Alexa, Google Assistant, Windows Cortana e la Siri di Apple. Spesso i termini vengono utilizzati in modo interscambiabile ma cosa distingue un assistente virtuale da un chatbot e da un agente conversazionale?
Un chatbot è un sistema semplice, non dotato di memoria cognitiva, in grado di riconoscere comandi posti in modo imperativo o tramite linguaggio naturale, se è dotato di un modulo di Natural Language Processing. All'identificazione di uno dei comandi, il chatbot capisce ed esegue la corrispondente azione programmata; come per esempio: "Okay Google, svegliami tra 5 ore". Risultato: Google Assistant ha impostato una sveglia sul proprio te-

lefono cellulare.
Un agente conversazionale, dotato di memoria cognitiva, non si limita ad un dialogo domanda e risposta ma cerca di raccogliere più informazioni facendo conversazione in modo attivo: pone domande all'utente in modo da raffinare la decisione da prendere, dando quindi un livello di assistenza superiore rispetto ad un sistema di singolo scambio.
Un assistente virtuale è il passo successivo nei sistemi di dialogo in quanto l'idea è di dare all'utente un agente conversazionale che sia specializzato per i suoi bisogni, dando l'impressione che sia lì soltanto per lui e che lo conosca. Necessita quindi di una memorizzazione di contesto nettamente superiore rispetto ad un agente conversazionale poichè bisogna tenere traccia delle preferenze dell'interlocutore.
Agente conversazionale ed assistente virtuale sono spesso due facce della stessa moneta ed in genere si confondono i termini e vengono chiamati genericamente "chatbots" in quanto spesso vengono adoperati su piattaforme di chat online.

2.1 Componenti di un sistema conversazionale e problematiche nella progettazione di un modello

Un sistema conversazionale moderno deve essere in grado di capire l'utente che si esprime in modo naturale, occorrono quindi tecniche di processamento del linguaggio: sono necessari il riconoscimento di intenti, per individuare l'azione che l'utente vuole svolgere, e di entità, per capire quali sono i parametri dell'azione da compiere.

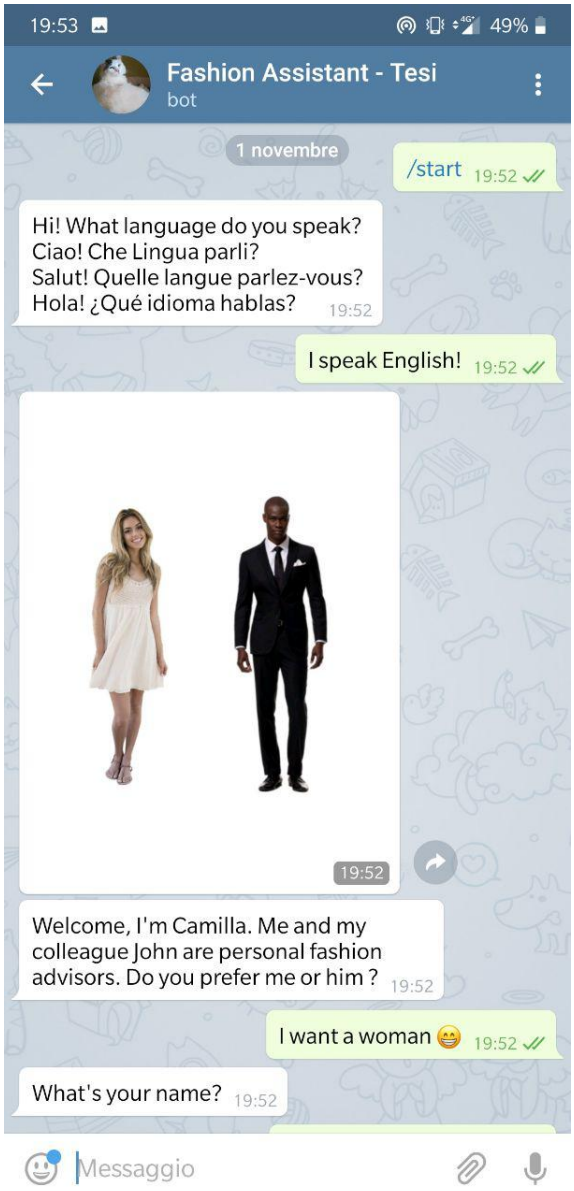
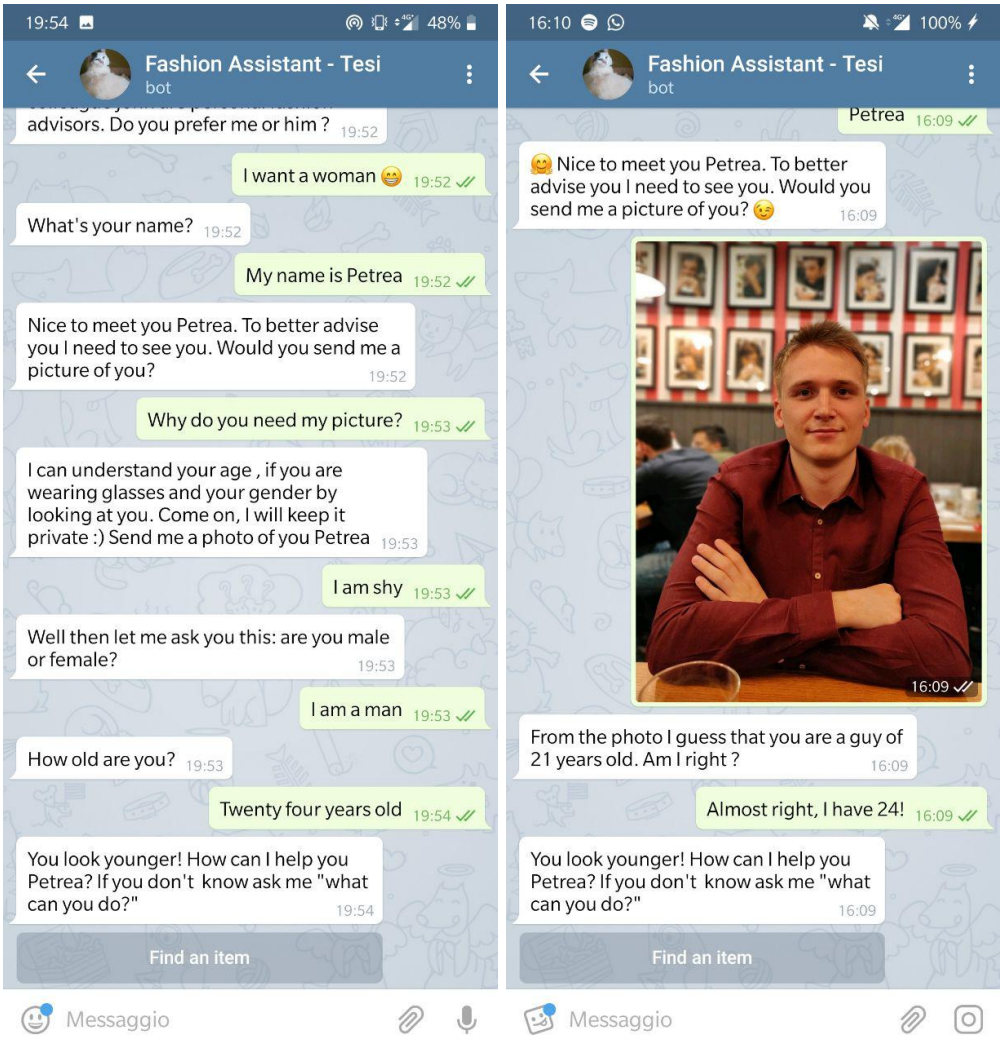


Figura 2.1: Primo dialogo

Parlare con qualcuno che si dimentica tutto quello che hai detto dopo ogni frase può risultare frustrante, per questo motivo bisogna che un sistema di dialogo moderno sia in grado di mantenere un contesto, ricordando cosa è stato detto prima e tenendo traccia delle informazioni importanti e necessarie per poter procedere oltre nel dialogo. I dati rilevanti dipendono ovviamente dal campo applicativo e dall'operazione che l'utente desidera svolgere.



(a) informazioni personali 1 (b) informazioni personali 2

Figura 2.2: Informazioni personali

Prendiamo come esempio l'inzio della conversazione con l'assistente che ho realizzato, come si può osservare in figura 2.1 l'agente inizia la conversazione chiedendo con quale lingua si desidera procedere; al momento della risposta memorizza la scelta dell'utente in modo da poter proseguire il dialogo nella lingua selezionata. Se l'utente non fornisce tale informazione il dialogo non può procedere oltre ma è necessario insistere affinchè venga espressa una preferenza in modo esplicito. Allo stesso modo procede nella configurazione iniziale della scelta dell'assistente e nella richiesta di informazioni personali come il nome e l'età. Tenendo conto delle risposte dell'interlocutore, figura 2.2, l'agente decide di richiedere le informazioni necessarie in maniera differente: per esempio, nel momento in cui l'assistente chiede una foto all'utente finale quest'ultimo può rifiutarsi di inviar-

la, tuttavia, per poter suggerire prodotti adatti al suo genere ed alla sua età il sistema necessita di queste informazioni e quindi l'assistente decide di procedere in maniera testuale.
Il componente fondamentale è il modello conversazionale e pone le maggiori difficoltà di progetto. Esso è un modello di stati che può essere rappresentato da un grafo diretto, l'agente in un qualunque punto della conversazione si trova in uno stato che è definito dalla conversazione passata e dalle variabili di contesto. Cosa succede quando l'utente invia un nuovo messaggio? L'assistente, mediante classificazione del messaggio, capisce l'intento ed estrae eventuali entità. Essendo context-aware è a conoscenza dello stato attuale e delle variabili di ambiente: sfruttando queste informazioni ed il nuovo input decide di portare la conversazione in uno stato piuttosto che in un altro in base ai vincoli imposti dal modello.
Le prime difficoltà che si presentano nella realizzazione dell'interfaccia conversazionale sono legate a modellare tutti i possibili stati che si potrebbero avere necessità di rappresentare. Nel modello ideale si cerca di anticipare le intenzioni e le conversazioni che un utente finale potrebbe sostenere con l'agente: più sono gli stati raggiungibili, ovvero modellati, e maggiormente la conversazione risulterà naturale e fluida. Sotto queste ipotesi la dimensione del modello aumenta e con essa anche la difficoltà di progettazione. Ogni conversazione può essere rappresentata da un cammino attraverso questo grafo, intuitivamente si può capire che più il cammino è flessibile, ovvero consente deviazioni, più il dialogo risulta naturale.
In pratica non si riescono a prevedere tutte le possibili interazioni con il sistema; è quindi buona norma un rilascio graduale dopo aver sottoposto il progetto all'attenzione di alcuni tester in modo da non realizzare completamente il dialogo secondo l'interazione con la propria personalità.
In ogni stato della conversazione lo spazio di possibili input è infinito, un modello conversazionale di successo cerca di limitare in modo inconsapevole all'utente ciò che egli può dire utilizzando una comunicazione ed un linguaggio comprensibile da chiunque. Per fare questo occorre anticipare la maggior parte delle domande che un utente potrebbe avere in mente, per ogni nodo del grafo.
La principale difficoltà che si presenta progettando una interfaccia conversazionale è soddisfare le aspettative degli utenti, i quali si aspettano che l'agente sia in grado di capire ogni richiesta anche quando questa è completamente fuori contesto.
Data l'eterogeneità degli utenti, sia culturale che caratteriale, è necessario cercare di creare metodi alternativi per ottenere informazioni perché una

strada forzata può risultare fastidiosa.
E' fondamentale monitorare costantemente l'utilizzo dell'assistente, leggendo le conversazioni con gli utenti reali e cercando di capire come e quali nodi aggiungere al grafo, quali esempi aggiungere al training set per il riconoscimento degli intenti e infine quali sinonimi per le entità. Di fatto un modello di successo dovrebbe essere allenato con gli esempi che un utente effettivamente andrà ad utilizzare per parlare con l'assistente.
Queste sono le principali problematiche nel progetto di un modello conversazionale: costruire interfacce di conversazione non rappresenta solo una sfida tecnologica ma anche linguistica e sociale.
Per concludere, il deployment dell'applicazione non è l'ultima fase del progetto ma esso va costantemente monitorato e migliorato attraverso un processo di apprendimento continuo.

2.2 Utilizzo su più canali

I motivi per operare su più di una piattaforma sono principalmente due:

- Raggiungere un maggior numero di utenti
- Dare la possibilità all'utente finale di interagire con il sistema mediante dispositivi e piattaforme eterogenee

Essere raggiungibili su molteplici applicazioni introduce il caso in cui uno stesso utente voglia, per comodità o necessità, interagire per mezzo di più di un canale di comunicazione.
Questa situazione è illustrata in figura 2.3: l'utente A interagisce con l'assistente per mezzo di Telegram e da un dispositivo che contiene Google Assistant (per esempio utilizzando Google Home mentre è in casa).
Quando questo utente interagisce da diverse applicazioni il sistema deve riconoscere che l'interlocutore è già noto all'assistente: occorre un meccanismo per legare insieme le informazioni provenienti da diversi canali. In questo modo, quando l'utente A ha già interagito con l'assistente su Telegram e vuole effettuare altre operazioni, in un successivo momento, dal suo dispositivo Google l'agente conversazionale sarà in grado di identificare univocamente l'utente ed in modo indipendente dalla piattaforma. Per esempio: A cerca ed aggiunge al carrello un prodotto per mezzo di Telegram, dopo un certo intervallo di tempo, invoca l'assistente attraverso Google Assistant e chiede di procedere con l'ordine.
Questo è possibile soltanto se viene implementato un meccanismo che lega

insieme in una unica utenza tutti i possibili account che l'utente usa sulle differenti applicazioni.

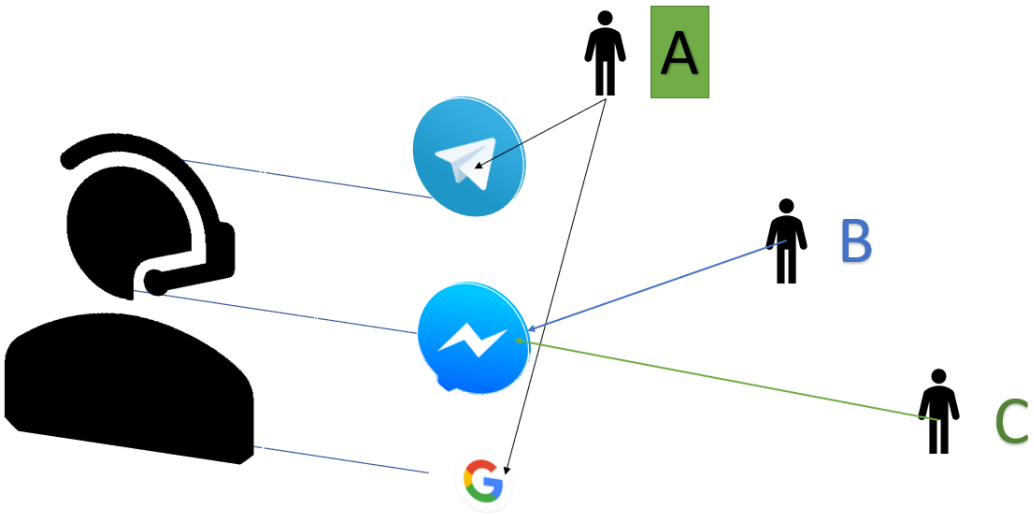


Figura 2.3: Canali utente

2.3 Integrazione

La descrizione del sistema verrà fornita nel capitolo a seguire, tuttavia i moduli che lo compongono sono molteplici. Per la natura dell'applicazione è utile poter seguire il flusso del messaggio attraverso le elaborazioni del sistema. A questo proposito lo strumento di sviluppo e di integrazione Node-RED è risultato il più adeguato nella fase di sviluppo di un prototipo.

2.3.1 Node - RED

Node - RED [1] è costruito sopra Node.js, un framework versione server-side di JavaScript maturo e pesantemente utilizzato che risulta quindi ampiamente testato ed affidabile.
Node-RED è uno strumento per lo sviluppo flow-based. Flow-based programming è un paradigma inventato da J. Paul Rodker Morrison: è un modo di pensare alla programmazione come dati, processi che elaborano quei dati e la rete che collega i processi insieme. Un insieme di processi possono essere legati per creare un flusso al fine di raggiungere un obiettivo, allo stesso modo i flussi possono essere raggruppati logicamente per raggiun-

gere un obiettivo superiore e così via. Dei dati entrano in ingresso, vengono elaborati da diversi processi ed un risultato è fornito: come una specie di catena di montaggio.
Node-RED fornisce una interfaccia visuale per la programmazione flow-based, è possibile definire dei cosiddetti nodi che di fatto sono un processo per il paradigma flow-based, dentro il nodo vi è del codice che elabora i dati in ingresso e ne restituiscono il risultato, quest'ultimo segue i collegamenti fino al prossimo nodo, il quale allo stesso modo esegue le sue elaborazioni e lo trasmette ulteriormente fino ad un nodo terminale.
La struttura di un nodo quindi è semplice, una porta di ingresso, del codice che elabora il dato ricevuto e una o più porte di uscita.
Il messaggio Node-RED rappresenta i dati che attraversano il flusso ed ha un formato di tipo JSON, il campo payload viene convenzionalmente utilizzato come dati da elaborare nel prossimo nodo del flusso, tuttavia è soltanto una convenzione e pertanto ogni nodo dovrebbe essere accompagnato da una documentazione che ne spiega il formato in input ed in output.
In figura 2.4 è possibile visualizzare i nodi , ovvero i processi che operano sui dati e la rete che li collega. Nell'insieme costituiscono il flusso, logico ed effettivo che un messaggio segue quando entra nel flusso. All'interno dei nodi arancioni è possibile scrivere codice JavaScript. Il nodo Viola è un nodo fornito dalla libreria node-red-node-watson che è un wrapper per le API del servizio IBM Watson Visual Recognition. Il nodo chiamato Face++ e esegue una richiesta HTTP e ne formatta la risposta in un formato JSON.

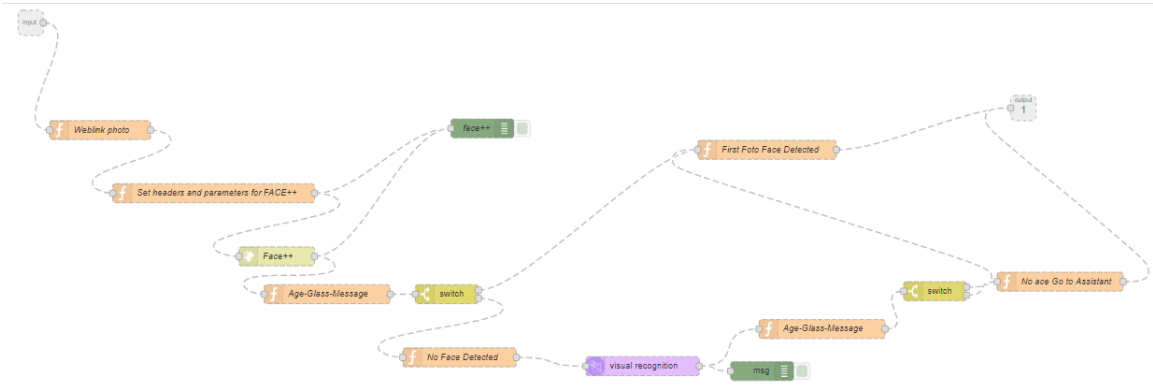


Figura 2.4: Flusso per determinare età e sesso dell'utente

2.3.2 Integrazione di servizi di terze parti

Con la crescita dei servizi in cloud vi sono moltissime funzionalità che possono essere aggiunte. L'assistente virtuale può fare affidamento su servizi di terze parti per l'implementazione di azioni come, per esempio, una previsione del meteo in base alla posizione dell'utente per la raccomandazione di un prodotto utile secondo le condizioni meteorologiche.
Una volta integrati, i servizi esterni faranno parte del sistema, influenzandone le performance ed il corretto funzionamento. Prendiamo per esempio che si debba scegliere tra un servizio che fornisce riconoscimento del volto che è gratuito ed un altro che è a pagamento. La scelta che chiunque preferisce a priori è quella gratuita, tuttavia bisogna considerare il carico di utenti che ci si aspetta, il tempo di risposta di tale servizio e la percentuale di chiamate API gestite con successo. Dal momento della richiesta all'API del riconoscimento del volto l'utente aspetta: se questo servizio è lento egli dovrà attendere indifferentemente da quanto il resto della applicazione sia performante; se il servizio è offline per qualche motivo anche la parte del sistema che ne fa affidamento risulterà inutilizzabile.
Bisogna quindi fare un compromesso sui servizi che vogliamo adottare nel nostro sistema, considerando di avere almeno un piano di riserva.
Per esempio, nell'assistente virtuale vi è il servizio offerto da Face++ che è gratuito come primo servizio che viene chiamato per la stima dell'età e del sesso dell'utente, tuttavia tramite test è stato osservato come il servizio risulti offline per periodi di tempo non trascurabili e quindi in caso di chiamata non riuscita il compito viene affidato al servizio IBM Visual Recognition che fornisce un modello addestrato per le medesime funzionalità. Tale servizio risulta affidabile ma è a pagamento per un utilizzo enterprise, ovvero offre un piano gratuito ma limitato nel numero di classificazioni che è possibile effettuare.
In figura 2.4 si può vedere questa strategia: il messaggio in input viene elaborato estraendo il link sorgente della fotografia (inviata nel primo nodo a sinistra) e prosegue nel nodo successivo che setta gli headers e i parametri per effettuare una richiesta HTTP POST, il messaggio ritorna con la risposta del servizio Face++ e viene elaborato per estrarne le informazioni rilevanti e, nel caso in cui Face++ ritorni un codice di errore, l'immagine viene sottoposta all'analisi del servizio IBM Visual recognition prima di uscire dal flusso.
I fattori da prendere in considerazione nella scelta di un servizio, offerto da diverse organizzazioni, comprendono non soltanto il prezzo ma an-

che le prestazioni, l'affidabilità e la facilità di integrazione con la propria applicazione.

2.4 Sistema di raccomandazione e problematiche

Insieme all'idea di dare un assistente virtuale personale nasce la necessità di un motore di raccomandazione. La domanda alla quale il motore deve fornire risposta è: quale prodotto suggerire ad uno specifico cliente?
L'obiettivo di un sistema raccomandazione è predirre la valutazione di un utente per uno specifico prodotto. Dato un insieme di prodotti che soddisfano una ricerca, il prodotto suggerito all'utente è quello per il quale la predizione presenta un valore più alto.
I requisiti sono la scalabilità e la velocità di predizione della soluzione, a questo proposito le tecniche di filtro collaborativo, ampiamente utilizzate da compagnie come Amazon, Netlflix e Google, risultano particolarmente adeguate.
Collaborative-Filtering usa l'aggregazione del comportamento/preferenze degli end-user per suggerire item rilevanti ad uno specifico utente. L'idea alla base del filtro collaborativo è che i suggerimenti migliori ci vengono forniti da persone con gusti simili ai nostri. L'ipotesi di questa tecnica è che se due utenti hanno la stessa preferenza per un prodotto allora è molto probabile che si troveranno d'accordo su un prodotto differente.

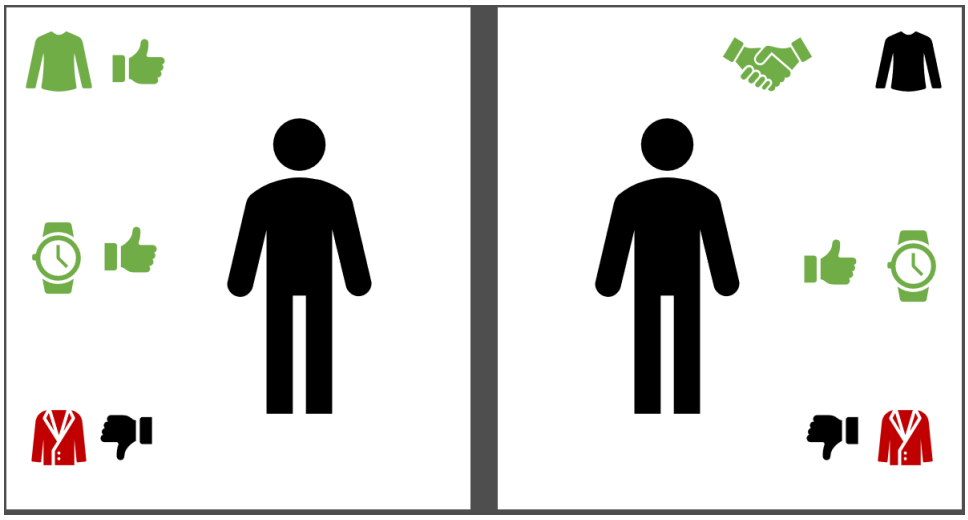


Figura 2.5: Filtro collaborativo - idea base

Per esempio, figura 2.5: Alice ha espresso preferenza positiva sull'item A, positiva sull'item B ed negativa sull'item C. Bob ha espresso anche lui preferenza positiva sull'item B e negativa sull'item C. Volendo predirre se Bob apprezza l'item A, si nota che egli ed Alice hanno le stesse preferenze per gli oggetti C e B , rispettivamente negativa e positiva. Quindi Alice e Bob hanno gusti simili e perciò è più probabile che Bob esprima un giudizio positivo sull'item A.
In questo esempio gli utenti sono solo due per illustrare l'idea alla base del filtro collaborativo, il successo di questa tecnica si basa sul fatto di avere molti utenti e molte preferenze espresse. Occorre quindi raccogliere l'opinione dei consumatori, più un utente è partecipe ed esprime le sue opinioni sugli oggetti e più il sistema risulterà accurato. Un utente che partecipa e fornisce feedback non soltanto aiuterà sè stesso, poichè ci saranno più dati che lo rappresentano, ma aiuterà anche tutti gli altri utenti partecipando alla predizione.
Il successo di un sistema di raccomandazione è influenzato prima di tutto dai dati, ovvero dai feedback raccolti dagli utenti: il motore è in grado di fornire predizioni accurate se la dimensionalità del training set e la qualità dei record a disposizione sono sufficienti, come avviene per la maggior parte degli algoritmi di Machine Learning. Quindi la problematica principale dal punto di vista dell'interfaccia conversazionale è il raccoglimento dei feedback: l'assistente virtuale deve persuadere l'utente ad esprimere un giudizio sugli oggetti che gli vengono proposti.
Il problema classico di un motore di raccomandazione è il suggerimento di prodotti per un nuovo utente che quindi non ha mai espresso una preferenza al momento dell'ultimo addestramento del modello di raccomandazione. Il problema è anche noto come cold-start problem.
Un ulteriore problema che ci si presenta dovendo implementare un sistema di raccomandazione è come gestire il re-training dell'allenamento, ovvero come e quando aggiornare il modello in modo da tenere in conto i nuovi utenti ed i nuovi feedback.

Capitolo 3

Descrizione ed architettura del sistema

Le scelte implementative, i servizi utilizzati e l'architettura del sistema vengono discusse in questo capitolo.
L'architettura logica è composta da quattro layer:

- *Channel Layer: è rappresentativo del canale di comunicazione, l'utente raggiunge l'assistente attraverso questo layer*
- *Input/Output Normalization Layer: l'input viene normalizzato per essere adattato al formato interno dell'assistente, l'output viene trasformato in base al tipo di messaggio e del canale di comunicazione.*
- *Cognitive Layer: cuore del sistema, contiene le componenti di intelligenza artificiale*
- *Persistence Layer: per rendere i dati persistenti*

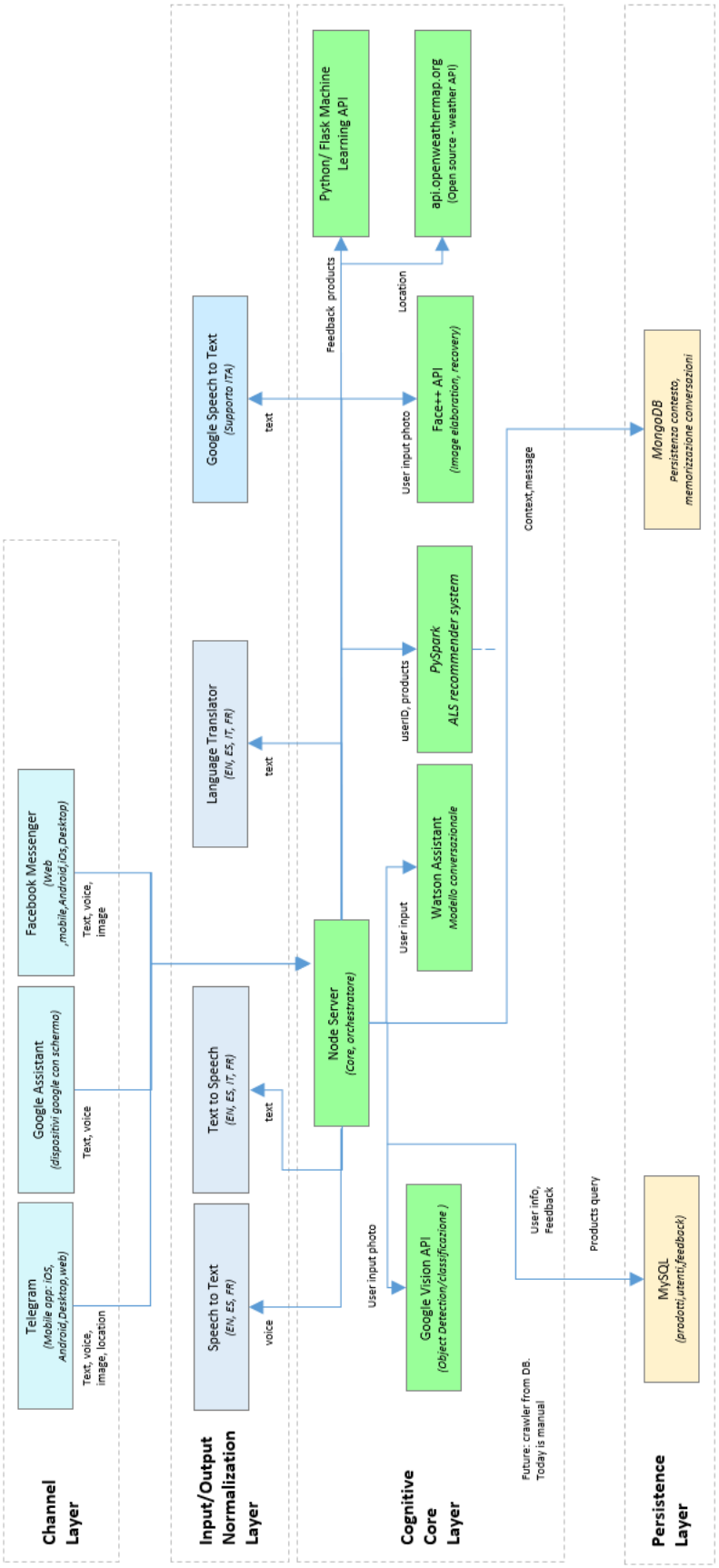


Figura 3.1: Architettura logica del Sistema

3.1 Tecnologie e strumenti utilizzati

3.1.1 IBM Watson Assistant

Watson Assistant [2] è lo strumento offerto da IBM nella piattaforma Bluemix, risulta particolarmente adeguato per lo sviluppo del modello di conversazione. Oltre alla possibilità di creare intenti ed entità, esso permette attraverso una interfaccia browser-based di creare il grafo degli stati. Lo strumento è inoltre in grado di interpretare un sottoinsieme del linguaggio di programmazione Spring Expression Language (spEL),simile al JavaScript. Questo consente di introdurre delle condizioni logiche complesse per entrare in un nodo del grafo. E' possibile, all'interno dello strumento, l'assegnazione di variabili ma la complessità di queste operazioni è limitata dall'esecuzione di una sola istruzione. Per questo motivo le operazioni complesse sono da gestire esternamente a Watson Assistant.

Input Watson Assistant prende in input una frase ed un contesto, inizialmente vuoto, che deve essere mantenuto esternamente al servizio. Il vantaggio di questa soluzione è poter applicare una logica esterna sulla base delle informazioni contenute nel contesto.

Contesto Il contesto è ciò che consente allo strumento di capire in quale stato ci si ritrova e quali sono le variabili note, come discusso nell'introduzione è di particolare importanza per la realizzazione di un agente conversazionale.

Intenti Un intento è l'intenzione o scopo che si cela dietro ad una frase, per esempio con la frase "I want to make a gift to my girlfriend, do you have decollete shoes?" si può capire che si vuole cercare un prodotto.
Gli intenti, in Watson Assistant, sono un insieme di frasi di esempio che esprimono lo stesso scopo. Più sono numerosi ed affini al modo in cui gli utenti finali esprimono l'intenzione e più il modello di classificazione sarà accurato.
Durante lo sviluppo del modello ho notato come la creazione di un intento richieda anche l'esistenza del complementare, anche se non utilizzato per muoversi nel grafo ma semplicemente per migliorare l'accuratezza di classificazione. Per esempio, volendo creare l'intento "SomethingEconomic" come set di esempi che un utente usa per esprimere l'intenzione di voler comprare qualcosa di economico, si possono usare frasi come:

- Affordable stuff
- I want something cheap
- Not too expensive
- cheap
- economic

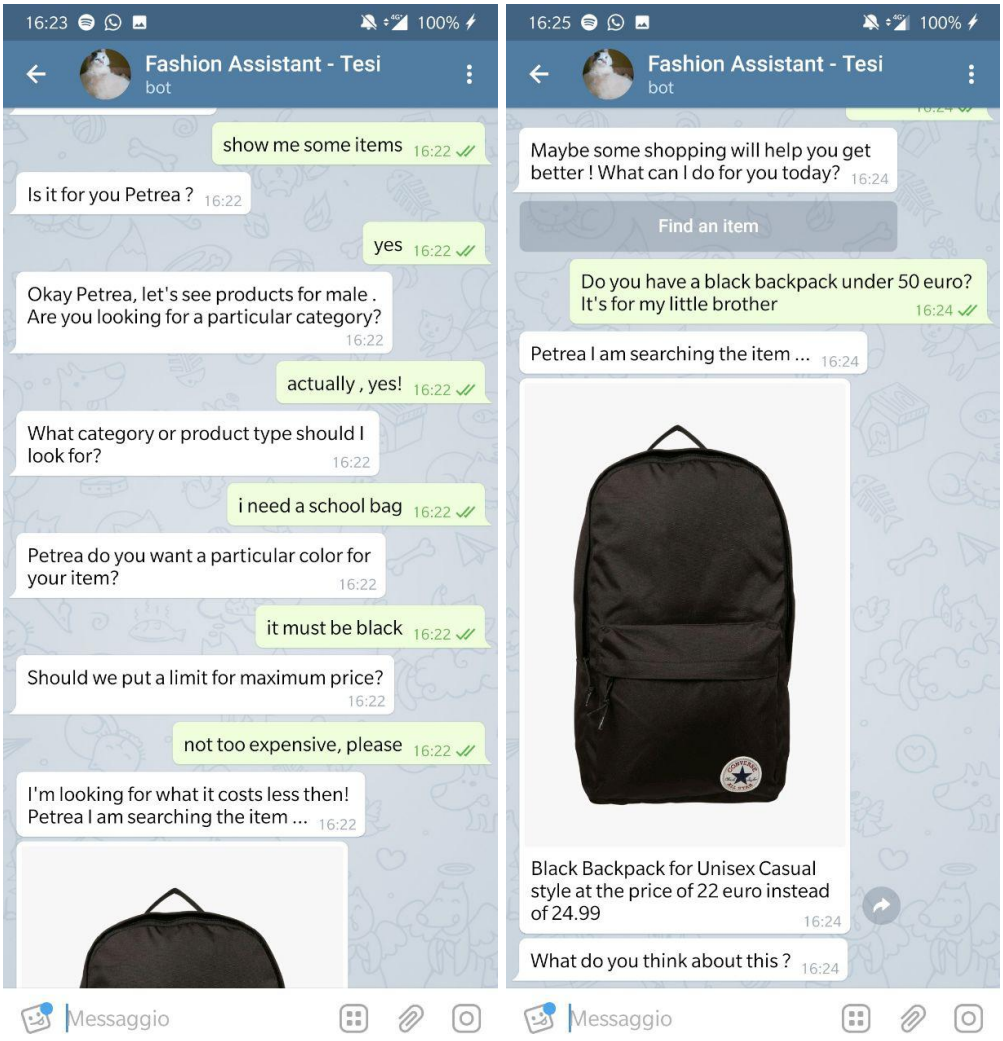
Tuttavia se un utente scrive "not economic", per similitudine, se non è presente l'intento che esprime la negazione esso verrà classificato come "SomethingEconomic" influenzando in maniera negativa la conversazione. Esempio: Not affordable stuff, expensive.
E' quindi buona pratica creare per ogni intento anche la versione negata.

Entità Le entità rappresentano le informazioni rilevanti che si vogliono estrarre dall'input. Esempi di entità sono in questo caso specifico i capi di moda, divisi in categorie e sotto-categorie, i colori, il genere, l'età. Per esempio nella frase "I want to buy jeans", la frase viene classificata come "Ready shopping", intento che ho creato per esprimere che un utente è pronto per cercare un prodotto, all'interno della frase però l'utente ha anche espresso un capo d'abbigliamento particolare che vuole cercare: dei "jeans". L'assistente è in grado di comprendere che l'utente vuole cercare dei pantaloni (categoria) di tipo jeans (sotto-categoria).
Le entità rappresentano un modo per ottenere informazioni utili per il processo, come nella ricerca dell'oggetto: l'utente può fornire una frase generica come "fammi vedere cosa vendi" oppure dettagliando "fammi vedere dei jeans neri sotto i 50 euro", nel secondo caso si possono estrarre informazioni in modo trasparente senza dover chiedere esplicitamente, dalla frase l'assistente estrae le seguenti entità:

- Colore : nero
- Categoria: pantaloni
- Tipologia: jeans
- Prezzo massimo: 50

Tali valori di entità vengono utilizzati per filtrare i risultati della ricerca dei prodotti. Nel caso in cui l'utente rimane generico è l'assistente a proporre in maniera attiva dei filtri. Come per esempio in figura 3.2.

Si può notare in figura 3.2 (b), ricerca diretta, la frase in input è ricca di informazioni e l'assistente ha già catturato tutte le informazioni che può utilizzare per fornire dei risultati specifici per l'utente proseguendo con la ricerca ed il suggerimento del prodotto.
In figura 3.2 (a) invece l'utente rimane sul generico, non vengono rilevate entità e l'assistente ne richiama l'attenzione chiedendo se desidera apportare i filtri di ricerca in maniera flessibile.



(a) Ricerca assistita (b) Ricerca diretta

Figura 3.2: Ricerca flessibile

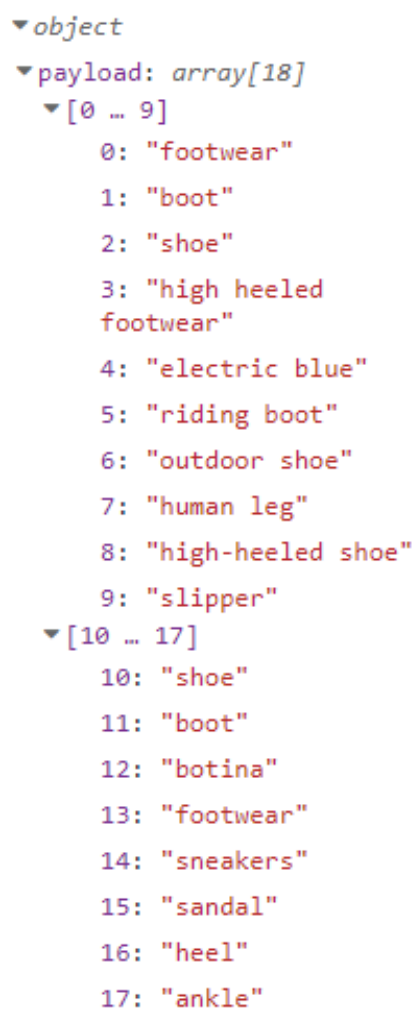
3.1.2 Speech to Text e Text to Speech

Il moduli di Speech to Text e Text to Speech permettono all'assistente virtuale di comunicare attraverso voce. Il servizio Speech To Text restituisce la trascrizione di un audio ed il testo prodotto può essere elaborato da Watson

Assistant, classificando l'intento, estranedo eventuali entità e muovendosi nel grafo conversazionale.
In maniera complementare il servizio Text to Speech fornisce una voce per l'assistente virtuale. Il testo in output dal modello di dialogo viene elaborato dal servizio per ottenere una sintesi dei caratteri, ottenendo un audio con voce umana naturale.
L'integrazione di questi servizi costituiscono una interfaccia vocale per l'assistente virtuale rendendolo quindi in grado di parlare e di "ascoltare". Il risultato che si ottiene è flessibilità e comodità per l'utente finale, in base alla situazione in cui ci si ritrova è possibile comunicare per via testuale oppure vocale. Questo strumento è necessario per i canali come Telegram e Facebook Messenger perché l'utente ha la possibilità di inviare un audio. Per piattaforme come Google Assistant ed Amazon Alexa il servizio è superfluo e non viene utilizzato perché viene fornito di default dal canale.

3.1.3 Google Vision API

Google Vision API[3] consente di analizzare il contentuto di una immagine, classificandola in una varietà di etichette tra cui categorie associate a prodotti di moda. E' la soluzione scelta per implementare una "interfaccia visiva" per l'assistente virtuale: data una immagine, tramite Vision API si ottengono le etichette che poi vengono elaborate dall'assistente per cercare di capire se è presente un prodotto di moda nella foto e consentire di ricercare un item simile data una immagine. Ciò che si vuole estrarre sono etichette relative alle categorie e sottocategorie dei prodotti presenti nel catalogo.
Il modello è generico, Google dichiara migliaia di etichette: occorre combinare le etichette proposte osservando la classificazione dei vari tipi di immagini. Un esempio è riportato in figura 3.3, a sinistra sono riportate le etichette per la foto inviata dall'utente in foto (b). Da (b) si nota che l'assistente virtuale riconosce che nella foto inviata vi sono delle scarpe stivaletto con il tacco. Analizziamo le etichette in figura (a): fra queste vi sono "boot" (stivale) e "high heeled footwear" (tacchi alti); insieme vengono combinate per formare l'etichetta ankleboots.
Mediante l'osservazione di diverse immagini è possibile estrarre un pattern ed adattare il modello generico di Google Vision API al caso moda: si costruiscono delle regole per ogni tipologia di prodotto del catalogo.
Oltre alle etichette di abbigliamento anche il colore è utile per filtrare gli item da proporre.



(a) Etichette ritornate da Vision API

(b) Risultato ottenuto nella conversazione

Figura 3.3: Integrazione Google Vision API

Il sottoflusso di classificazione dell'immagine è riportato in figura 3.4, per conformità con il formato richiesto dalle Vision API occorre passare l'immagine codificata in base 64. Per adempiere questo requisito occorre scaricare l'immagine tramite una richiesta HTTP GET all'url dell'immagine. Si converte in base 64 il buffer. Si impostano headers e payload per la richiesta, secondo quanto riportato nella documentazione di Google, e si esegue la chiamata all'endpoint di Vision API. Infine, si imposta una soglia di confidenza pari a 0.5 per l'etichettatura per filtrare i risultati con una confidenza bassa. Il risultato viene ritornato in formato JSON e nel nodo "rule the tags!" avviene l'elaborazione delle etichette come riportato nella sezione 5.1.

Una singola immagine può essere etichettata con molteplici classi: utilizzando la logica if-then-else si combinano le etichette. Data una serie di elementi riconosciuti nella foto, questi si combinano per formare una nuova etichetta. Se una regola ha un match, un nuovo risultato viene inserito nell'array image_classification, contenente tutte le classi trovate nell'immagine con relativo score.
Nella determinazione degli score viene valorizzata la combinazione poichè è più specifica (le singole etichette hanno score più alto rispetto ad una media dei due), così come per le etichette gerarchiche: shirt da confidenza maggiore rispetto ad elegant shirt ma quest'ultima è più specifica. Le regole codificate possono essere migliorate mediante ulteriori osservazioni.

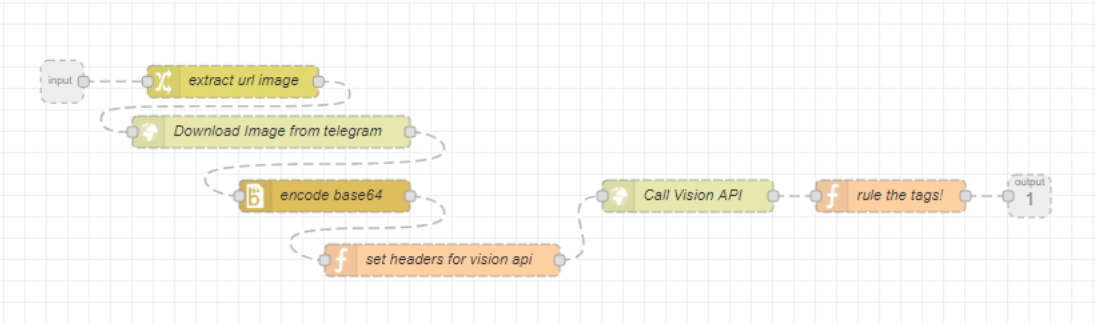


Figura 3.4: Sottoflusso classificazione immagine

3.1.4 Database Management System

MySQL E' un database management system di tipo relazionale. Veloce, affidabile ed open-source. E' fondamentale per rendere i dati di utenti e relativi feedback persistenti e per accomodare i prodotti disponibili nel catalogo. La scelta di utilizzare un database relazionale è motivata dalla presenza dei vincoli relazionali che possono essere utilizzati per mantenere consistenti i dati, requisito fondamentale per la qualità dei feedback prodotto-utente.
Lo schema riportato in figura 3.5 riporta le tabelle con gli attributi e le relazioni di chiavi primaria ed esterna.

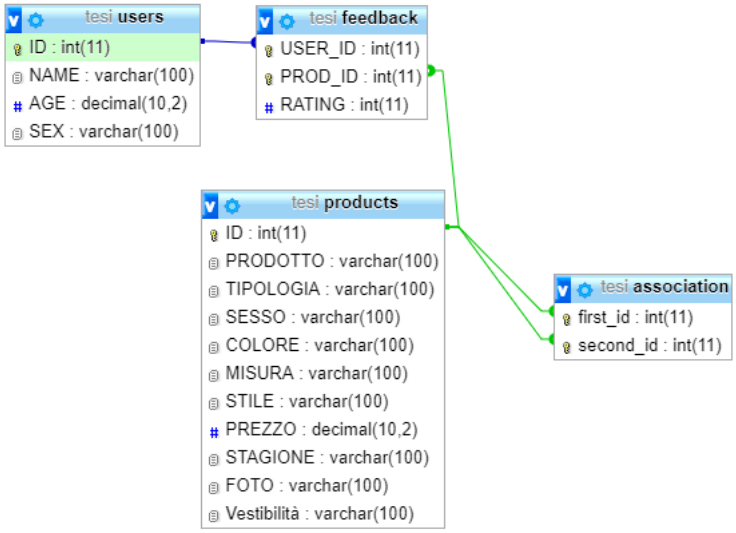


Figura 3.5: ER-schema tabelle necessarie per operare

MongoDB MongoDB è classificato come database NoSQL ovvero di tipo non relazionale ed orientato al documento con rappresentazione simile al formato JSON.
La struttura di un documento può essere differente rispetto agli altri documenti della collezione, in particolare i campi che ne compongono lo "schema" posso essere variabili ed assumere il valore di strutture dati complesse come un array oppure un altro oggetto JSON.
Queste sono le caratteristiche aggiuntive che si rendono necessarie: rispetto ad un database relazionale, MongoDB risulta particolarmente adatto

per la memorizzazione del contesto, delle informazioni relative alla singola conversazione ed infine per legare insieme gli identificativi forniti dalle piattaforme di comunicazione allo scopo di identificare univocamente l'utente, indipendentemente dal canale di comunicazione. Ogni informazione può essere ricercata tramite una chiave pari all'identificativo dell'utente. MongoDB ha una propria Application Programming Interface che consentono di effettuare operazioni atomiche sul database come inserimento,ricerca,cancellazione e modifica di documenti.

3.2 Channel Layer

Il channel layer è il mezzo attraverso il quale l'utente finale può comunicare con l'assistente virtuale.
Per operare su molteplici canali, in modo da raggiungere una quantità di utenti maggiore, l'approccio di progettazione utilizzato è di tipo modulare: il channel layer è incaricato della raccolta del messaggio originale proveniente dalla sorgente attuale del messaggio (per esempio Telegram). In questo modo è possibile aggiungere in modo incrementale dei canali senza alterare il resto del progetto.
I canali adottano diverse politiche sullo scambio di messaggi: alcune piattaforme permettono di inviare liberamente messaggi ed altre limitano la risposta ad un solo messaggio. L'assistente virtuale genera messaggi in vari punti del flusso, per ogni messaggio più di una risposta può essere generata: la struttura dati utilizzata è una coda FIFO dove un elemento è dato dal messaggio ed i relativi flag (usati per statistiche e suggerimenti da fornire all'utente). Alla generazione di un messaggio, lo si inserisce in coda e si procede nel flusso. In output, in base al canale, occorre unire i messaggi in un unico corpo (Google Assistant) oppure inviare separatamente (Telegram, Facebook Messenger).

3.2.1 Telegram

Per utilizzare telegram come canale di comunicazione, è necessario creare un bot. A questo proposito occorre contattare Botfather, "The one bot to rule them all" , all'indirizzo @BotFather ed inserire il comando /newbot.
Ai fini della applicazione è necessario ottenere un token, comando /token, in modo da poter effettuare le operazioni sul proprio bot attraverso le API di Telegram. Il token è viene utilizzato per l'autenticazione: solo chi lo conosce può gestire il bot, più specificamente solo chi ne è in possesso può leggere i messaggi inviati al bot ed inviare messaggi dal bot verso l'utente. E' vitale tenere il token segreto, altrimenti chiunque ne sia a conoscenza può operare al posto dell'assistente virtuale.
Ogni conversazione Telegram è caratterizzata da un ID. Questo campo può essere utilizzato per distinguere in maniera univoca un utente, consentendo di memorizzare distintamente le informazioni ad egli relative, come per esempio il contesto. Occorre osservare che questo identificativo non viene utilizzato per identificare il contesto relativo ad una conversazione ma solamente per identificare l'utente all'interno della piattaforma Telegram. Per

quanto riguarda le informazioni relative all'utente l'identificativo utilizzato come chiave primaria è il codice assegnato dall'assistante virtuale (vedere sezione 3.2.3).
Di seguito è riportato il payload di un messaggio tipico, il campo chatId è l'ID dell'utente.

```
"payload" = {chatId":125710886,
    "messageId":6639,
    "type":"message",
    "content":"I'm fine thank you"
    }
```

```
payload =  {
    "chatId":125710886,
    "messageId":6641,
    "type":"voice",
        "content":"AwADBAADcQQAAk8HUVGsLTo8rsXl5gI",
    "date":1537861212,
    "blob":true,
        "weblink":"https://api.telegram.org/file/
        bot676916098<token>/voice/file_297.oga"
    }
```

Il messaggio Telegram può essere di diversi tipi, per l'applicazione in questione è necessario gestire i formati:

- message: messaggio di tipo testuale
- photo: per la gestione dell'input di fotografie
- voice: necessario per rendere disponibile l'interazione mediante voce
- callback_query: per l'inserimento di tasti nella conversazione
- location: contiene informazioni sulla posizione dell'utente, utile per suggerimenti in base a dove si trova fisicamente
- pre_checkout_query : necessario per la gestione dei pagamenti
- succesful_payment: indica che una transazione è stata effettuata con successo, necessario per la gestione degli ordini

Per utilizzare Telegram[4] all'interno di Node-Red si possono seguire differenti approcci: utilizzare dei nodi rappresentativi delle operazioni che

vengono effettuare su Telegram oppure interfacciarsi attraverso chiamate api mediante l'utilizzo del nodo HTTP Request che è presente in Node-Red di default.
Il primo approccio è modulare, a tal proposito saranno necessari almeno due nodi: input ed output verso Telegram. La libreria piùdiffusa nella comunità è node-red-contrib-telegrambot, installabile attraverso il comando npm install node-red-contrib-telegrambot.
Sebbene sia la libreria più utilizzata, più di cinquantamila download mensili, è incompleta nella gestione dei vari tipi di messaggio. Nasce quindi la necessità di implementare funzionalità aggiuntive a quelle già presenti.
I nodi sono un wrapper per le REST API di Telegram, quindi un approccio alternativo è interfacciarsi attraverso la preparazione di una chiamata HTTP mediante un nodo javascript function dove si possono impostare gli headers ed il body del messaggio, in conformità con formato stabilito da Telegram, ed infine inoltrare la chiamata attraverso HTTP.
In questo progetto è stato utilizzato un approccio misto, per operazioni semplici è stato utilizzato il primo approccio, ovvero chiamate HTTP all'interno di node-red, ed è stata utilizzata la libreria con modifica dei nodi in modo da poter gestire i pagamenti, ovvero sono stata implementata la gestione dei messaggi di tipo succesful_payment e pre_checkout_query.
In particolare ogni canale sarà caratterizzato da un flusso di input ed uno di output, rispettivamente per ricevere ed inviare messaggi.

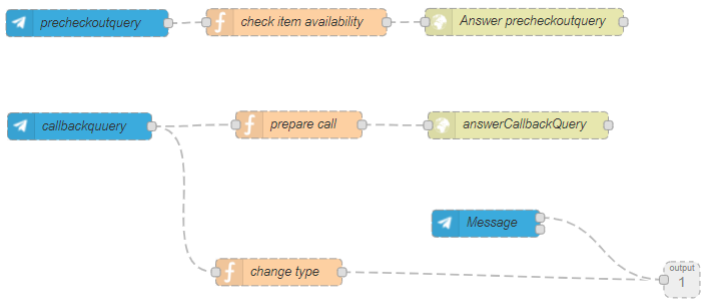


Figura 3.6: Sottoflusso input Telegram

In figura 3.6 è riportato il sottoflusso di input per il canale telegram, la maggior parte delle operazioni vengono svolte dai nodi azzurri.
Il nodo precheckoutquery è in ascolto per messaggi di tipo pre_checkout_query e ritorna un messaggio contenente informazioni relative ad una fattura emessa su telegram. La fattura è creata dall'assistente virtuale ed inviata

all'utente nel momento in cui quest'ultimo decide di voler procedere all'acquisto dei prodotti che ha nel carrello. Questa informazione viene allegata insieme al messaggio per identificare l'ordine ed implementare la logica di vendita che si vuole come per esempio può essere la disponibilità dei prodotti della fattura. A questo messaggio occorre rispondere con un messaggio answerPreCheckoutQuery della piattaforma Telegram, indicando nella riposta stessa l'id della pre_checkout_query originale e la riposta alla domanda "l'ordine è accettato?".
Per semplicità, nell'implementazione corrente, tutti gli ordini vengono accettati ma è dipendente dall'azienda stessa che fornisce il servizio di vendita dei prodotti. Normalmente si verifica la disponibilità, se questi sono disponibili la risposta sarà affermativa altrimenti negativa. Alcune società accettano l'ordine ed al più nel peggiore dei casi effettuano un rimborso se l'ordine non può essere spedito.
Il nodo callbackquery è necessario per la gestione di tasti presenti nella schermata della chat. I tasti vengono utilizzati per suggerire delle operazioni all'utente, di fatto vengono trattati come un normale messaggio testuale per cui il campo "type" viene settato a "message" ed inoltrato nel flusso principale.
Il nodo Message è in ascolto per tutti gli altri tipi di messaggio trattati dall'applicazione.
In figura 3.7 è rappresentato il flusso di processi che elaborano un messaggio in output dall'assistente virtuale verso un utente Telegram. Come prima operazione si filtrano i messaggi vuoti che possono essere utilizzati per avanzamento interno all'interno del modello conversazionale, per consentire di eseguire azioni interne che non sono visibili all'utente. Si procede con l'estrazione di un messaggio dalla coda FIFO e si impostano i suggerimenti nel sottoflusso "keyboard telegram" in base ai flag associati. Se occorre, il testo viene tradotto e/o sintetizzato in audio ed infine inviato verso all'utente Telegram.

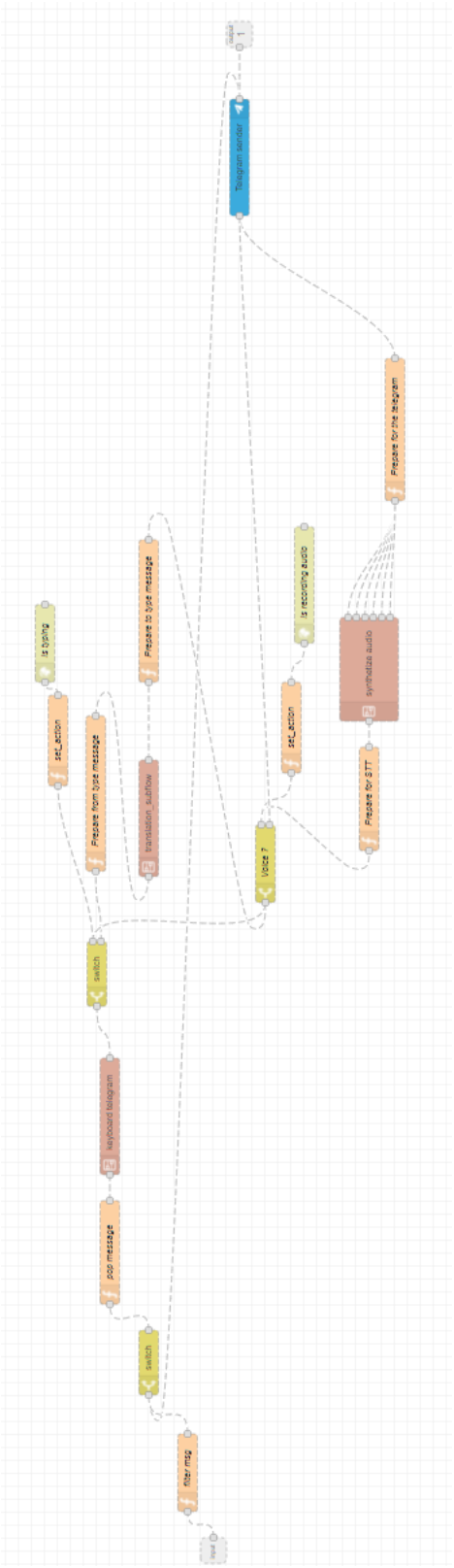


Figura 3.7: Sottoflusso output Telegram - Channel e Normalization Layers

3.2.2 Google Assistant

Il consulente viene reso disponibile sotto forma di "Action" per Google Assistant. A tale scopo, occorre creare un progetto Google Actions e definire un nuovo agente. Sebbene lo strumento di Google, DialogFlow, permetta di costruire al suo interno un modello conversazionale, l'intenzione è di utilizzarlo come canale di comunicazione: è necessario inoltrare tutte le richieste provenienti da Google Assistant al Fashion Advisor.
Per poter interagire con Google, occorre esporre un end-point HTTPS, avente un certificato valido, al quale il servizio effettua delle chiamate POST per fornire all'applicazione la richiesta originale dell'utente. Il modo per inoltrare le richieste è creare un intento, all'interno di DialogFlow, che si attiva per qualsiasi input ed abilitare l'inoltro della richiesta all'end-point sopra citato.
L'input proveniente da questo canale viene automaticamente riportato in testo, anche se l'interazione è vocale: la trasformazione è a carico di Google Assistant. Un esempio di richiesta è riportato in figura 3.8. In particolare ci interessano i campi queryResult.queryText che riporta la trascrizione ed il campo user che fornisce informazioni come la lingua ,l'ID dell'utente e le capacità del dispositivo utilizzato.

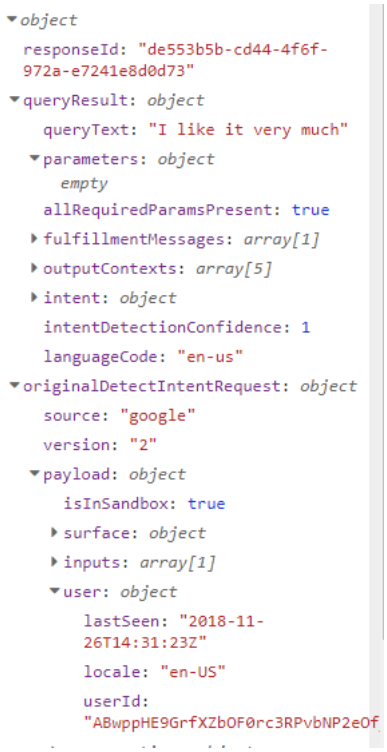


Figura 3.8: Google Input JSON

Poichè ad una chiamata è possibile fornire una sola risposta, in output occorre creare un unico messaggio: si unisce la coda FIFO in una unica entità. Le immagini vengono utilizzato per creare contenuto arricchito, come definito nella documentazione. Il testo viene impostato come testo video e speech to text per la riproduzione su dispositivi audio. La tastiera, in analogia con il processo di telegram, viene impostata come "suggestions".

Gli elementi utilizzati riproducono il risultato in figura 3.9, in addizione all'anteprima del prodotto viene aggiunto un link al file originale della foto per una visualizzazione più attenta. In basso vengono mostrati i suggerimenti. Il codice di esempio per ogni elemento è riportato nella sezione 5.2.

Ulteriori informazioni sul formato dei descrittori posso essere trovate sulla documentazione ufficiale [5].

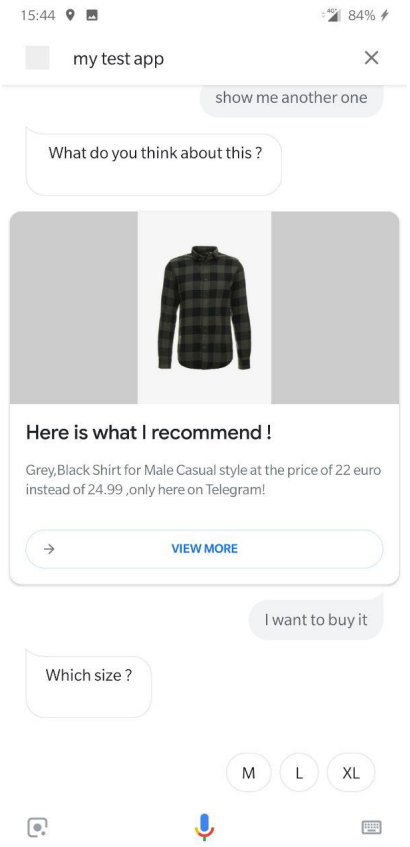


Figura 3.9: Google: elementi conversazione

3.2.3 Account-linking cross-canale

La soluzione al problema introdotto nella sezione 2.2 è utilizzare un account, fornito dall'assistente virtuale, che identifica l'utente in maniera univoca in modo indipendente dal canale di comunicazione.
All'interno del singolo canale di comunicazione, viene fornito, all'utente, un codice identificativo: per diversi canali l'utente avrà codici differenti, in particolare uno per ogni piattaforma.
La struttura dati utilizzata è la seguente:

```
1    "id":identificativo_utente, #obbligatorio,univoco,
2    "telegramID":chatID (canale Telegram),#univoco,
3    "googleID":chatID (canale Google),#univoco
4    "secret":password_hash_sale
```

Allo stesso modo, con l'integrazione di altre piattaforme, si aggiungo ulteriori identificativi che vengono associati con l'identificativo univoco fornito dall'assistente virtuale. Essendo una struttura flessibile, si utilizza mongoDB per la memorizzazione del mapping.

Per gestire l'autenticazione, a monte del modello conversazionale, si inserisce un flusso che:

- Verifica l'esistenza di un mapping fra id fornito dal canale ed id fornito dall'assistente
- Se il mapping esiste l'interazione procede in modo trasparente
- Altrimenti, all'utente viene chiesto di autenticarsi oppure registrarsi

Ne consegue che un utente è obbligato a concordare le informazioni di autenticazione al primo dialogo in assoluto, momento in cui gli viene assegnato un ID da parte dell'agente e viene memorizzato il mapping tra id-piattaforma ed ID (interno).
Successivamente se l'utente comunica per via della stessa applicazione il mapping viene individuato e la conversazione procede senza interruzioni. Nel momento in cui l'utente decide di voler parlare per mezzo di un altro canale supportato, l'assistente non trova una corrispondenza e gli chiede di autenticarsi. L'utente si autentica, il mapping con il nuovo canale viene memorizzato e l'interazione procede come per il primo canale utilizzato.

3.3 Normalization Layer

Il Channel Layer può accomodare canali eterogenei, ogni piattaforma di mesaggistica ha un formato proprio per la rappresentazione di diversi tipi di messaggio. Volendo implementare una architettura modulare, in grado di operare su un nuovo canale, senza modificare l'intero sistema, nasce la necessità di rendere l'input uniforme. Inoltre, tutti i messaggi vengono elaborati dal modello conversazionale che accetta soltanto un formato testuale e sentenze in lingua inglese [1].
Essendo il formato di un messaggio proprio del canale di comunicazione, una stessa informazione può avere un nome diverso rispetto alle altre piattaforme adoperate o può essere addirittura mancante. Occorre uniformare le informazioni ricavate dall'input con un formato unico dell'applicazione, in modo da poter centralizzare l'elaborazione delle comunicazioni di diversi canali nello stesso flusso, evitando ridondanze. In particolare, normalizzare i dati significa:

- Messaggi voce/audio: accertarsi che il messaggio ricevuto sia trasformato in testo, a questo proposito si utilizza un servizio di Speech-To-Text per ottenere una trascrizione dell'audio.
- Assicurarsi che il testo sia scritto nella stessa lingua con il quale è stato creato il modello, altrimenti gli intenti non potranno essere riconosciuti: traduzione del testo dalla lingua dell'utente alla lingua inglese.
- Settare i campi necessari per il proseguimento nel flusso, in conformità con il formato della applicazione[2].

Supporto multilingua Le lingue ufficiali dell'Europa attualmente sono 24: bulgaro, ceco, croato, danese, estone, finlandese, francese, greco, inglese, irlandese, italiano, lettone, lituano, maltese, olandese, polacco, portoghese, rumeno, slovacco, sloveno, spagnolo, svedese, tedesco e ungherese. [6]
Volendo implementare un supporto multilingua per aumentare i segmenti di clientela, ovvero rendere disponibile l'utilizzo dell'assistente virtuale a persone non di lingua inglese, le strategie che possono essere adoperate sono principalmente due.

[1] lingua utilizzata come linguaggio nativo per lo sviluppo del grafo
[2] per distinguere l'azione da svolgere in base al tipo di messaggio

Il primo approccio consiste nella replica del modello conversazionale per ogni lingua che si vuole implementare. Questa strategia richiede quindi, supponendo di voler dare supporto per le lingue ufficiali parlate in Europa, di creare ventiquattro modelli.
Questo approccio ha il vantaggio che l'accuratezza nel riconoscimento degli intenti e delle entità risulterà migliore ed il dialogo si presenterebbe più naturale. Gli svantaggi sono la necessità di ridondare il modello di conversazione in ventiquattro copie: si traduce in aumento dei costi del servizio e difficoltà nel mantenere il progetto. Una modifica al grafo principale comporta di conseguenza la necessità di modificare tutti gli altri, introducendo quindi il rischio di avere inconsistenze tra i vari modelli.
Il secondo approccio consiste nell'utilizzo di un traduttore, strategia adoperata nello sviluppo di questo assistente virtuale.
Una volta normalizzato il testo proveniente da un canale di comunicazione, in base alla preferenza espressa dall'utente durante la prima conversazione con l'assistente virtuale si elabora il testo per ottenere una traduzione dalla lingua scelta alla lingua inglese. In output occorre effettuare l'operazione inversa: la lingua sorgente è l'inglese e la lingua di destinazione è la lingua dell'utente.
Questa seconda soluzione soffre di diverse problematiche. Il problema principale è l'influenza della qualità della traduzione sulla classificazione di un intento, il servizio di traduzione non è accurato nel caso in cui nella frase ci sono errori ortografici o di sintassi. In parole semplici, se il testo introdotto nel traduttore non è sintatticamente ed ortograficamente corretto la traduzione che ne risulta non è accurato e di conseguenza, poichè viene inviato al servizio Watson Assistant, anche la classificazione di intenti ed il riconoscimento di entità.
Il vantaggio però è avere un unico modello conversazionale.
Seconda problematica è il costo del servizio di traduzione, sebbene vi sia presente un piano gratuito nel catalogo IBM Bluemix per il servizio Translator, volendo passare ad un utilizzo enterprise occorre un piano a pagamento. Occorre quindi tagliare i costi il più possibile, a questo proposito la strategia adoperata è tradurre tutto l'input dell'utente poichè con alta probabilità sarà diverso da persona a persona (memorizzare tutte le possibili frase porterebbe a problemi di memoria) e tradurre l'output dell'assistente virtuale solamente la prima volta. E' possibile utilizzare questo approccio poichè le risposte fornite dall'assistente virtuale sono limitate in numero.
Viste le imperfezioni del traduttore, in base al numero di persone che lavorano al progetto si dovrebbe espandere il modello gradualmente nello

sviluppo del grafo nelle varie lingue che si vogliono rendere disponibili. La prima strategia è da preferire in fase di rilascio.

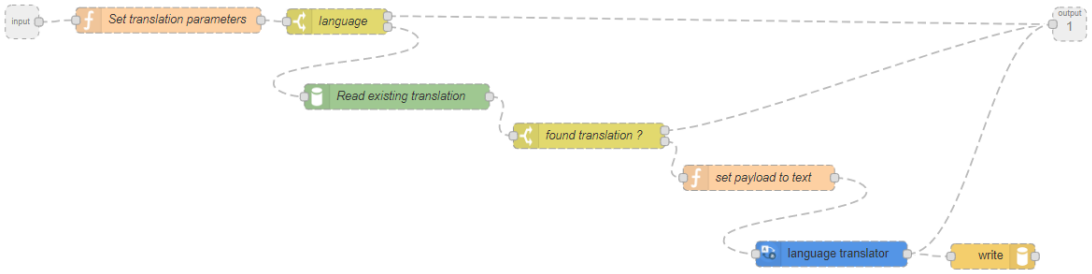


Figura 3.10: Sottoflusso di traduzione in output

In figura 3.10 è riportato il sottoflusso di traduzione. Il messaggio da parte dell'assistente virtuale viene inoltrato in fase di output al sottoflusso di traduzione, come prima operazione sui dati occorre estrarre la lingua di destinazione (memorizzata nel contesto). Se la lingua di destinazione è l'inglese allora il servizio di traduzione non è necessario ed il messaggio è dato direttamente in output. Altrimenti, come prima operazione si cerca se la traduzione è già stata eseguita in passato, mediante interrogazione di un key-value database in formato json, dove la chiave che identifica la traduzione stessa è la frase originale concatenata con il codice della lingua.
Se il risultato tornato è nullo allora la traduzione dello specifico testo non è ancora avvenuta in passato e quindi si preparano i parametri da passare al traduttore, infine si invia in output il messaggio tradotto ed in parallelo si memorizza la traduzione con il formato sopra citato.
Il flusso di traduzione in input è banale, occorre settare i parametri ed eseguire la traduzione con la lingua scelta dall'utente come sorgente e destinazione inglese.

3.4 Cognitive Layer

Il Cognitive Core Layer è al centro del sistema: contiene le componenti di intelligenza artificiale e la logica applicativa utile a simulare l'interazione umana tra il consulente di moda e il cliente.
Le componenti principali sono il modello conversazionale, il sistema di raccomandazione, i flussi applicativi che orchestrano le azioni e la ricerca di prodotti visivamente simili a partire da una immagine.

3.4.1 Architettura del modello conversazionale

Il modello conversazionale è composto da cinque sottomodelli, ovvero da cinque workspace di IBM Watson Assistant. Ognuno dei modelli è incaricato a gestire l'interazione pertinente ad una specifica fase del dialogo.

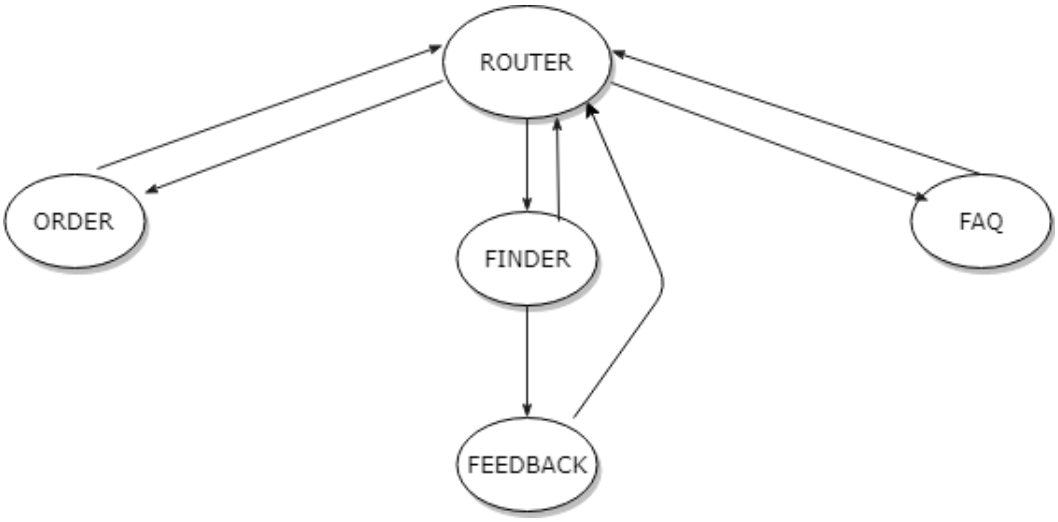


Figura 3.11: Architettura modello conversazionale

Motivazioni della implementazione mediante molteplici workspaces
Progettando un modello conversazionale si incontrano difficoltà tecniche nel riconoscimento di intenti, spesso è necessario poter esprimere con la stessa frase due intenti separati. Mantenendo un unico workspace, il classificatore di intenti non sarà in grado di distinguere tra due classi perché una frase di esempio deve appartenere soltanto ad un intento per evitare la generazione di conflitti nel training set. Per generare un conflitto non è strettamente necessario che due frasi, appartenenti a due diversi intenti, siano uguali ma è sufficiente siano simili. Un conflitto, totale o parziale, nel training set si traduce in un calo dell'accuratezza della classificazione.

Da queste osservazioni si deduce che per produrre un modello di classificazione più accurato è necessaria la separazione degli intenti dal punto di vista semantico.
Il grafo conversazionale cresce rapidamente di dimensioni con il numero di casi che si vogliono gestire: si ha difficoltà a gestire ed estendere modelli di grande dimensionalità nel numero di nodi. Per questo motivo un approccio "Divide et impera" risulta particolarmente adatto nella realizzazione di un assistente virtuale complesso. Con questo approccio modulare, i sottomodelli vengono combinati fra di loro ed è possibile generare nuovi cammini di dialogo senza replicare i nodi del grafo. Una soluzione che separa il problema in sotto-problemi più semplici non soltanto riduce la difficoltà di progettazione ma porta ad una maggiore accuratezza della classificazione: il risultato finale è un assistente virtuale in grado di capire meglio l'utente.
Una implementazione separata del modello presenta anche svantaggi, il principale è l'aumento della difficoltà di integrazione. I vari workspace devono comunicare fra di loro come riportato dalle frecce direzionali in figura 3.11. Se mantenendo un workspace unico l'interazione poteva essere fatta direttamente all'interno del modello mediante un salto da uno stato ad un altro, con l'implementazione multipla occorre gestirla esternamente. Da questo punto di vista può essere visto anche come un vantaggio poichè ci permette di inserire operazioni complesse tra un sotto modello ed un altro.
Un ulteriore svantaggio della soluzione è il mantenimento cinque entità di tipo contesto che sono differenti, una per workspace. Ad ogni chiamata REST API, al servizio IBM Watson Assistant, deve essere inclusa l'informazione del contesto, informazione rappresentativa della posizione attuale nel grafo. Tecnicamente in ogni istante per ogni workspace ci si trova in un nodo, se il dialogo non è attivo ci si ritrova all'origine del sotto-modello. Avere workspace separati significa dover mantenere molteplici contesti relativi ai singoli sotto modelli, i quali insieme formano il contesto generale dell'assistente virtuale. Le informazioni estratte in un sotto modello possono essere trasmesse ad un altro sotto modello tramite la gestione esterna delle variabili all'interno di questa struttura dati.
Le difficoltà nell'integrazione dei moduli sono risolvibili mediante un'attenta progettazione ed implementazione; L'accuratezza del modello nel riconoscimento di intenti ed entità è dipendente in modo diretto dall'insieme di esempi utilizzato, questa è la principale motivazione di questa scelta implementativa. Inoltre è stato il modo naturale di pensare al problema, prendiamo in considerazione l'interazione che può avvenire fisicamente in negozio:

- Si può essere approcciati da una persona che ci chiede da chi vogliamo essere assistiti (configurazione e router) e ci mette in contatto con il commesso.
- Il commesso ci chiede informazioni sul prodotto che stiamo cercando (finder)
- ci propone dei capi d'abbigliamento e cerca di capire se ci piacciono (feedback)
- infine, se scegliamo di acquistare, può indirizzarci verso la cassa dove si procede con il controllo dell'ordine e si provvede al pagamento (order).

Ogni singola fase può essere svolta da una persona differente: è l'idea alla base dell'architettura del modello conversazionale.

Router Il router è il modello principale, è il punto di inizio di una conversazione, il dialogo incaricato a svolgere è connesso alla configurazione dell'assistente virtuale e conversazione generale. E' composto da 101 nodi di dialogo di Watson Assistant ed è in grado di riconoscere 42 intenti e 24 entità.
Il compito principale del Router, oltre alla configurazione, è di indirizzare la conversazione verso il workspace più adatto in base all'azione che l'utente intende svolgere. Da qui il nome Router.
Per adempire a questo compito, il Router, deve poter riconoscere tutti gli intenti ad alto livello degli altri workspace. Per esempio, se un utente scrive nel messaggio un testo che è pertinente ad una Frequently Asked Question il Router deve essere in grado di capire che si vuole parlare con il workspace FAQ. Se l'input esprime l'intenzione di ricerca di un prodotto, per esempio "sto cercando una maglietta rossa che non costi troppo", il Router comprende che l'operazione richiesta è gestita dal workspace Finder. In modo simile per il workspace che gestisce il carrello, se la frase in input è per esempio "Vorrei controllare il carrello della spesa" il dialogo deve procedere attraverso il sotto modello Order.
Il modo più semplice per poter implementare questo meccanismo è duplicare gli intenti generali degli altri workspace, ovvero quelli che sono nel primo livello di profondità del grafo, all'interno del Router che riconoscendo uno di questi intenti imposterà una variabile di contesto, simbolicamente nominata switch, con la quale si può distinguere con chi avverrà la comunicazione ed inoltra il messaggio attraverso il flusso Node-RED al corrispetti-

vo workspace.
Ovviamente questo risulta in una doppia chiamata al servizio IBM Watson Assistant, ma si può limitare la ridondanza soltanto al primo momento in cui si riconosce l'intento corrispondente ad un altro workspace. Mediante l'ausilio della variabile switch ed un componente node-red che implementa lo switch di programmazione, impostato sulla medesima variabile e posto a monte del router si può saltare la chiamata al router che in alternativa agisce da elemento passivo lasciando scorrere la conversazione verso l'altro workspace e sprecando una chiamata API. La strategia è rappresentata dalla porzione di flusso node-red in figura 3.12, dove il percorso che viene seguito dal messaggio è evidenziato da colori differenti. Il Router, colore azzurro, comunica direttamente con FAQ, Order e Finder, per cui in uscita dall'elaborazione del Router uno switch è posto per indirizzare la conversazione verso il modello successivo da interpellare. Il prossimo messaggio che si riceve non andrà più verso il Router ma seguirà il percorso indicato dallo switch posto a monte. In modo analogo avviene l'interazione tra Finder e Feedback.

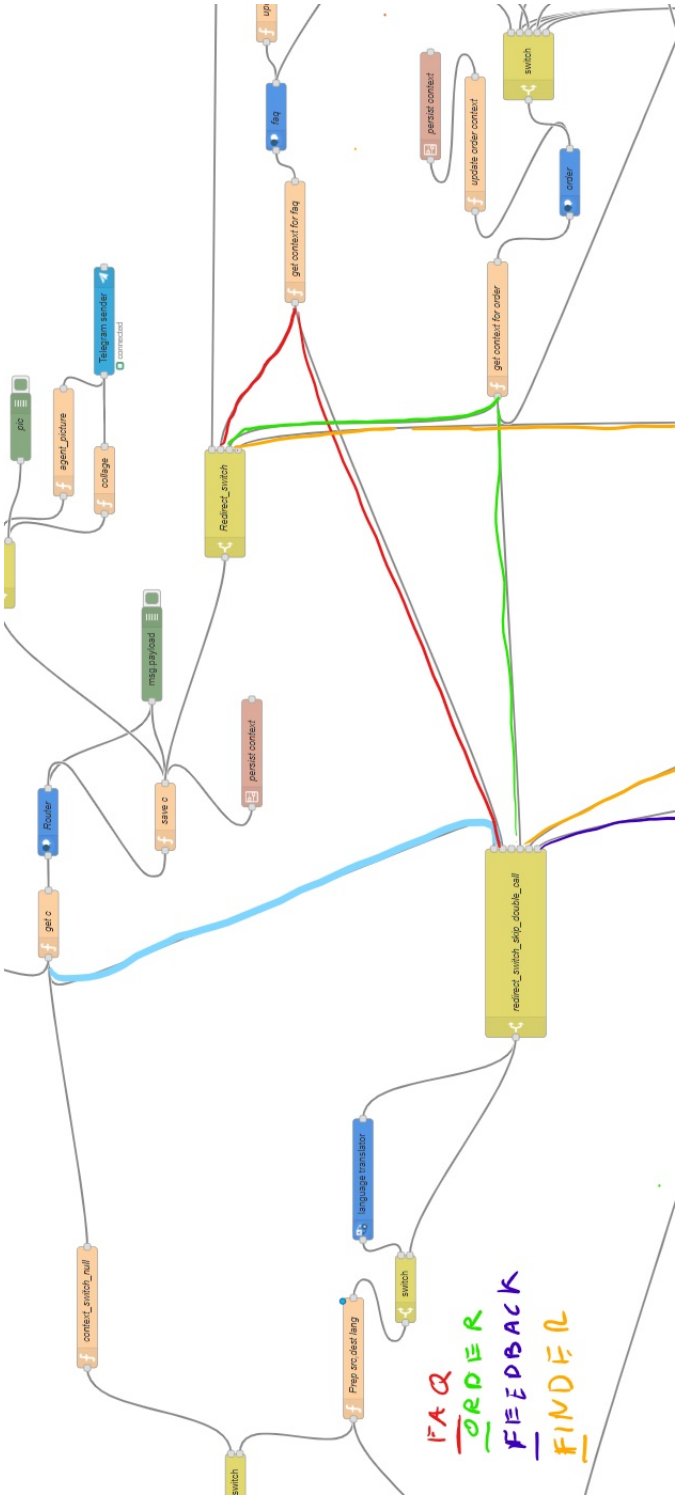


Figura 3.12: Strategia per saltare la doppia chiamata API al servizio Watson Assistant

Router - Azioni Il router è il primo modello con il quale l'utente interagisce. Essendo incaricato della configurazione, è in grado di capire e

riconoscere che un nuovo utente sta utilizzando l'assistente virtuale: l'azione principale che viene orchestrata dal router è l'inserimento di un nuovo utente nella base dati. Altre azioni possono essere implementate, come per esempio l'invio di una foto da parte dell'agente, come in figura 2.1.
Per implementare una azione si può utilizzare una variabile di contesto indicativa dell'operazione da svolgere. Nel flusso Node-RED la variabile può essere utilizzata per direzionare il processo su strade differenti. L'orchestrazione delle azioni da parte del router è riportata in figura 3.13: se nel contesto del router la variabile "action" è uguale al valore "add_user" un nuovo utente deve essere memorizzato nel database, se l'azione è "send_collage" l'assistente invia la foto in figura 2.1 ed infine se l'azione è "send_picture" l'agente invia una foto individuale dell'assistente scelto (John o Camilla). Lo switch R_ACTION differenzia le azioni da svolgere.
Ulteriori azioni possono essere aggiunte in maniera modulare con la crescita delle funzionalità del workspace router: si crea un sottoflusso che implementa l'azione, il router gestisce il dialogo e capisce quando questa va svolta ed in fine il flusso Node-Red orchestra il processo.

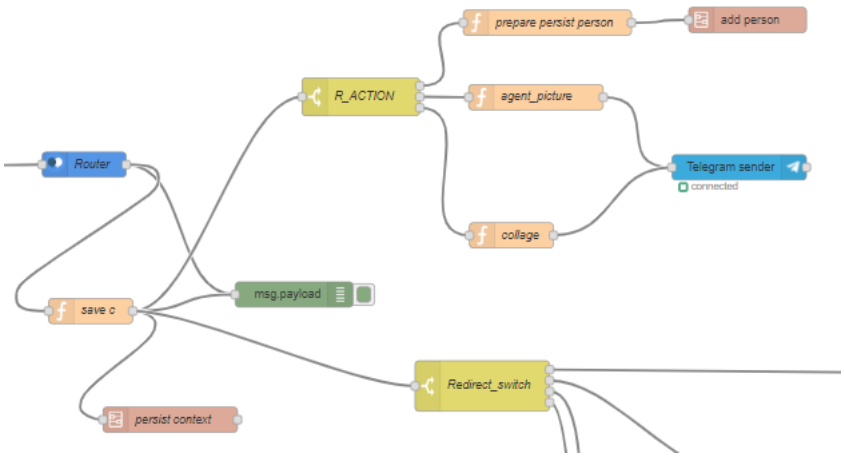


Figura 3.13: Porzione flusso - Azioni router

Finder Il workspace intitolato Finder è incaricato a gestire il dialogo che riguarda la ricerca di un prodotto, in particolare è in grado di interagire con l'utente allo scopo di raccogliere informazioni sul prodotto che l'end-user sta cercando.
Il Finder è in grado di riconoscere le entità inerenti ai prodotti di abbigliamento, organizzati con una struttura gerarchica a due livelli: categoria di prodotto e sottocategoria. In figura 3.14 sono riportate le entità presenti

nel workspace.

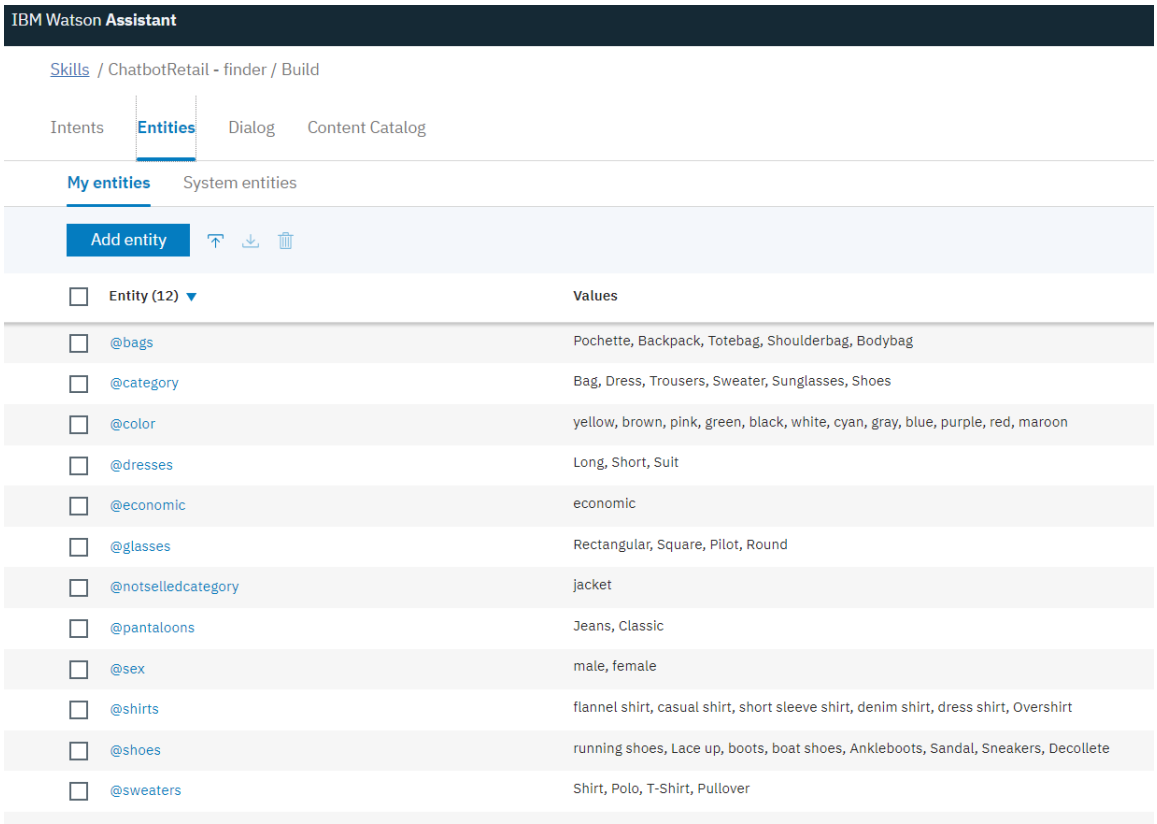


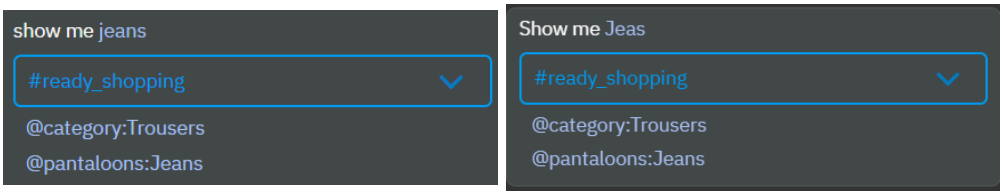
Figura 3.14: Entità Finder

Nella creazione di una entità la difficoltà che si incontra è determinare tutti i sinonimi con con i quali ci si può riferire ad un determinato prodotto. Per dare flessiblità all'utente, il riconoscimento di entità può essere fatto con l'opzione "fuzzy-matching". Abilitando questa opzione una entità verrà riconosciuta anche in presenza di errori ortografici e match parziali, per esempio se scriviamo "tshirt" invece che "T-Shirt" il valore verrà riconosciuto. Alcune entità hanno valore molto simile per esempio "T-shirt" e "shirt". L'abilitazione del fuzzy-matching può portare al riconoscimento di "shirt" come match parziale di "T-Shirt", un altro esempio è "skirt" che può essere facilmente confuso con "shirt".
In definitiva il fuzzy-matching è una funzionalità utile e può far sembrare l'agente conversazionale molto intelligente se trattato in modo adeguato: occorre gestire l'ambiguità che si crea per termini simili, possibilmente eliminandola del tutto. Il valore riconosciuto può essere prelevato in due modi: usando il match letterale, ovvero come è stato scritto nel testo, oppure il valore associato. L'assistente virtuale riconosce l'entità ed il valore che

viene salvato è utilizzato come campo per una ricerca all'interno della tabella prodotti all'interno di un database. Per questo motivo il fuzzy-matching crea problemi se si prelevano i valori letterali poichè non saremmo in grado di trovare all'interno di un database, mediante linguaggio query in sql, risultati che soddisfano i requisiti poichè non è previsto un match parziale anche nel linguaggio SQL ed in ogni caso è difficile replicare lo stesso meccanismo con il quale viene riconosciuto all'interno di Watson Assistant. Nonostante queste difficoltà, per maggiore flessibilità nei confronti degli utenti , si implementa la possibilità di riconsocere le entità in presenza di errori sintattici ed ortografici.
La strategia adottata consiste nell'organizzazione delle entità su due livelli differenti, una prima entità @categoria verrà utilizzata con fuzzy-matching per riconoscere il tipo di prodotto che si cerca ed il valore che si memorizza non sarà il testo letterale ma il valore associato.
L'entità *@category* contiene i valori delle categorie di prodotti vendute dall'assistente virtuale.

Entità categoria	
Valore	Sinonimi
Trousers	Jeans,Classic,chinos,pantaloons,denim trousers,suit pants,slack trousers,trouser
Sweater	T-shirt,Pullover,Polo,sweater,shirt,shirts
Sunglasses	Pilot,Square,Rectangular,Aviator
Bag	Backpack,Bodybag,Totebag,Pochette,Shoulderbag, hand bag,clutch,rucksack,knapsack,daypack,bookbag ,school bag,shoulder bag,carryall bag, holdall bag,shopper,shopping bag
Shoes	Ankleboots,Decollette,Sandal,Sneakers,Lace up,lace-up,lace,running shoes,training shoes,heels,high heels,casual shoes
Dress	Suit,Short,Long,midi dress,longuette,maxi dress,short dress,mini dress

Per ogni valore i sinonimi riportati sono tutte le sotto categorie appartenenti ad una specifica categoria. In questo modo, quando viene digitato all'interno della frase uno dei sinonimi associati alla categoria il valore riconosciuto sarà il valore stesso di categoria.



(a) Corretto (b) Fuzzy

Figura 3.15: Riconoscimento di categoria e sottocategoria

Come si può osservare in figura 3.15 (a), il valore riconoscito che verrà salvato per categoria è consistente con quanto memorizzato nelle entry della tabella prodotti. L'informazione di quale categoria è stata riconosciuta consente di abilitare il riconscimento della entità corrispondente alla medesima, abilitando il fuzzy matching in entrambe se non si presentano ambiguità con altri prodotti (b). La query che si può costruire in definitiva sarà "SELECT * FROM PRODUCTS WHERE Category = 'Trousers' AND Subcategory = 'Jeans'". Non utilizzando questa organizzazione a due livelli ma utilizzando il match letterale dei sinonimi della entità @categoria i casi sono due: se si abilita il fuzzy-matching il risultato letterale utilizzato per la costruzione della query SQL non sarà in grado di ritornare risultati, se non viene abilitato il fuzzy-matching allora l'assistente virtuale non sarà in grado di intendere che c'è stato un semplice errore ortografico nel testo.
Per i prodotti che possono introdurre ambiguità , esempio "shirt" e "skirt", non si può abilitare il fuzzy-match perché ne risulterebbe nel riconoscimento di entrambe le entità quando uno dei due input viene fornito. Un modo per dare comunque flessibilità su questi termini consiste nell' inserire nei sinonimi valori contenenti errori ortografici. Un esempio è riportato in figura 3.16.
In definitiva, il fuzzy-matching è una funzionalità interessante ma come già affermato sopra va gestito correttamente altrimenti ci potrebbe ritrovare nella situazione in cui l'utente esprime l'intenzione di voler vedere una gonna ("Do you have a nice skirt to recommend?") e l'assistente virtuale gli presenterebbe delle camicie perché "skirt" ha un match parziale alto rispetto a "shirt".

Entity values (4) ▼	Type	
Polo	Synonyms	polos, polo
Pullover	Synonyms	pullover, pullovers, pllover, pull over, pulover
Shirt	Synonyms	Shirts, shirt, shrt, dress shirt
T-Shirt	Synonyms	t-shirt, t-shirts, teeshirt, t shirt, t shiirts

Figura 3.16: Entità Sweaters, fuzzy match simulato

Finder - Indagini di mercato L'entità @notselledcategory è composta da un unico valore che ha come sinonimi tutte le possibili tipologie di capi di moda che attualmente non sono vendute dal fashion advisor, in questo modo si può comprendere comunque ciò che l'utente vuole cercare e tenere traccia di tutte le operazioni di ricerca che non coinvolgono soltanto prodotti che vengono venduti. Il riconoscimento della situazione in cui l'utente ricerca un item non disponibile consente di capire il desiderio verso quel determinato capo d'abbigliamento. Questo può tornare utile per indagini di mercato : capire quale è la domanda e decidere se offrirla.

Finder - Azioni L'unica azione che il finder è incaricato ad orchestrare è l'inzio della presentazione di un prodotto all'utente. Quando le informazioni relative alla ricerca sono state fornite si aziona il meccanismo di ricerca,raccomandazione e presentazione del prodotto.

Finder - Metriche Occorre stabilire delle metriche per la valutazione della qualità del modello, nel caso Finder la misura che è più adeguata è il rapporto tra ricerche iniziate e ricerche portate a termine.
Quando il ROUTER classifica l'intento destinato al finder, significa che una ricerca ha inizio. I = Numero di interazioni iniziate con il workspace Finder.
Nel momento in cui il Finder stabilisce l'azione di ricerca del prodotto, il dialogo con esso si è concluso con successo. F = Numero di ricerche nella tabella prodotti.

$Q = F/I$, metrica per la valutazione della qualità del sotto modello conversazionale Finder.

FAQ Incaricato di dare risposte alle domande più frequenti degli utenti, è il workspace più semplice. Una funzionalità da aggiungere è la possibilita di connettere lo user con un agente umano in caso in cui non sia soddisfatto della risposta automatica. Volendo si può introdurre un dialogo per cercare di approfondire ogni problematica, questo aspetto però e fortemente dipendente dalle politiche amministrative aziendali e per questo motivo non è stato implementato in questo progetto.

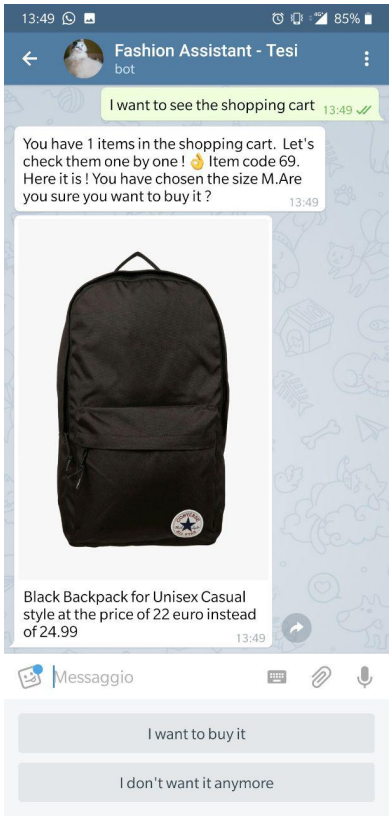
ORDER Il workspace assume il ruolo di cassa ed incaricato a gestire un carrello della spesa. Il dialogo viene sviluppato allo scopo di richiedere conferma all'utente sui prodotti che ha aggiunto al carrello.
Nel suo contesto vi è traccia degli item selezionati; in particolare l'ID e la taglia scelta per ognuno di essi. In generale occorre memorizzare la scelta di ogni parametro variabile di un prodotto: se il prodotto è disponibile in più colori il cliente può scegliere il suo preferito: ID,Taglia e Colore definiscono univocamente la configurazione desiderata. Per semplicità, i prodotti disponibili nel dataset sono di un unico colore per cui nell'implementazione corrente l'unico parametro variabile è la taglia. Le entità necessarie sono quindi l'entità taglia che può essere letterale oppure numerica in accordo con le taglie degli indumenti. A questo proposito si possono usare Regular Expressions per il riconoscimento del pattern letterale e numero in modo da limitare il numero di valori accettabili all'interno del workspace ORDER e realizzare un primo filtro, ulteriori controlli sulla disponibilità della taglia selezionata dal cliente vengono effettuati esternamente dall'orchestratore.

Entity values (2) ▼	Type	
size_literal	Patterns	(^|\s|\.)([x|X]{0,2}[s|l|S|L]|[m|M])(\s|$|\?|\.)
size_number	Patterns	(^|\s|\?)([3-5][0-9])(\s|$|\?|\.)

Figura 3.17: Entità size, regular expressions

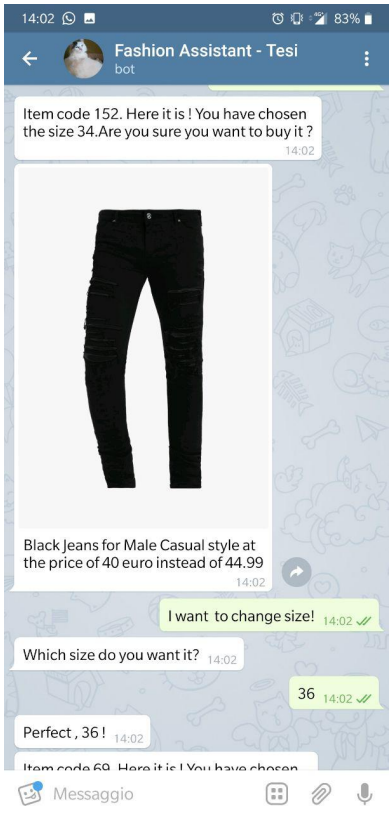
Utilizzando i pattern in figura 3.17 si è in grado di riconoscere tutte le taglie letterali più comuni all'interno di una frase in una generica posizione. Per le taglie numeriche un primo filtro viene posto sul range accettabile, in questo caso vengono da 30 a 59. Il valore memorizzato nel contesto è il match.
Non tutte le taglie riconoscibili sono accettabili per uno specifico item: dato un itemID è necessario leggere l'informazione delle taglie disponibili allo

scopo di validarne la scelta e può essere utilizzata per dare un suggerimento all'utente. Se la taglia comunicata non è disponibile l'assistente virtuale informa l'utente. Alcune conversazioni sono riportate qui sotto, in particolare nella figura (c) è mostrata la possibilità di cambiare taglia in fase di conferma acquisto.



(a) Conferma acquisto

(b) Emissione fattura



(c) Cambio taglia

Figura 3.18: Gestione del carrello mediante conversazione

ORDER - Azioni Analizziamo la conversazione che è contenuta in figura 3.18:

- un oggetto viene mostrato con la configurazione scelta: action 'retrieve_item' per azionare il flusso che invia la foto e la descrizione dell'item all'utente.
- L'oggetto viene confermato: deve essere aggiunto alla fattura, action 'add_item_to_invoice' per azionare il flusso che preleva le informazioni relative alla configurazione scelta,la descrizione ed il prezzo del prodotto e lo aggiunge alla fattura.
- L'oggetto viene rifiutato: deve essere tolto dal carrello, gestito internamente al workspace in quanto consiste nella rimozione della entry nell'array contenuto nel contesto stesso.
- Un utente puo' aver sbagliato taglia: occorre mostrare le taglie disponibili e gestire la nuova scelta , action 'show_sizes' per azionare il flusso che dato l'itemID mostra le taglie disponibli ed aggiunge al contesto del workspace l'array di taglie accettate.
- Si procede con l'acquisto: occorre inviare la fattura preparata, action 'prepare_invoice'
- Non si procede con l'acquisto: è necessario cancellare la fattura preparata, action 'clear_invoice'

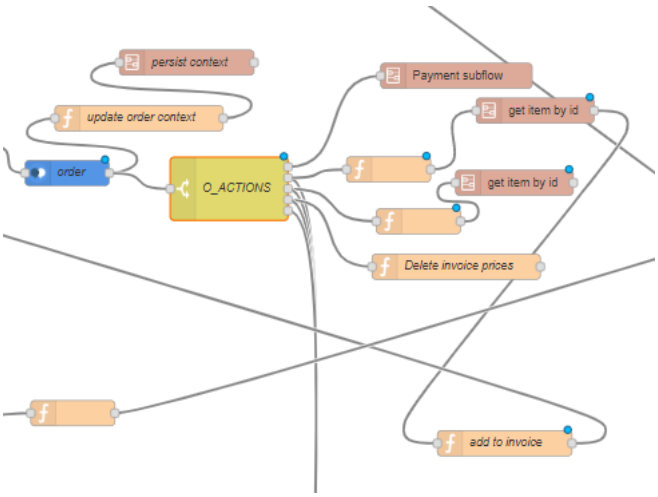


Figura 3.19: Porzione flusso - azioni order

Lo switch O_ACTIONS , in figura 3.19, basandosi sulla variabile action del contesto del sottomodello ORDER instrada il messaggio verso la relativa azione.

FEEDBACK Il workspace è incaricato a gestire il dialogo allo scopo di presentare, raccogliere l'opinione e gestire l'aggiunta al carrello (scelta configurazione) di prodotti.
I prodotti vengono presentati in maniera iterativa ed ordinati in base allo score dato dalla predizione del sistema di raccomandazione. Affinchè la raccomandazione sia efficace occorre avere il feedback dell'utente, per questo motivo vi è particolare enfasi nel chiedere all'utente quanto gli piace l'item proposto.
Visualizzato un prodotto, l'assistente virtuale chiede l'opinione e suggerisce dei possibili input che vengono attribuiti a tre livelli:

- 0: non mi piace (1StarIntent)
- 1: mi piace (2StarIntent)
- 2: mi piace moltissimo (3StarIntent)

Di fatto questi suggerimenti sono in corrispondenza con intenti e ne consegue che l'opinione dell'utente non viene vincolata ai singoli suggerimenti ma egli può esprimere in linguaggio naturale quello che effettivamente ne pensa. A questo proposito nel set di esempio degli intenti sono state aggiunge frasi che sono state reputate adeguate per ognuno dei tre livelli di rating.
Questo metodo da flessibilità e rende l'assistente virtuale più "umano", tuttavia viene dato il suggerimento per far intendere all'utente quale tipo di risposta ci si aspetta.
Sempre per flessibilità, lo user non è tenuto ad esprimere una opinione ed ha la possibilità di vedere un altro prodotto oppure aggiungere il prodotto al carrello direttamente. Queste possibilità non vengono mostrate come suggerimento per enfatizzare la necessità di feedback.
Se l'opinione data è positiva (livello 1 o 2) l'assistente esplicita la possibilità di comprare il prodotto attualmente proposto oppure continuare a vederne altri. Nel caso in cui l'opinione non sia positiva l'agente nella risposta omette la possibilità di comprare ma propone di vedere un altro prodotto. Questo per dare l'idea di un assistente amico che non pensa semplicemente a vendere.

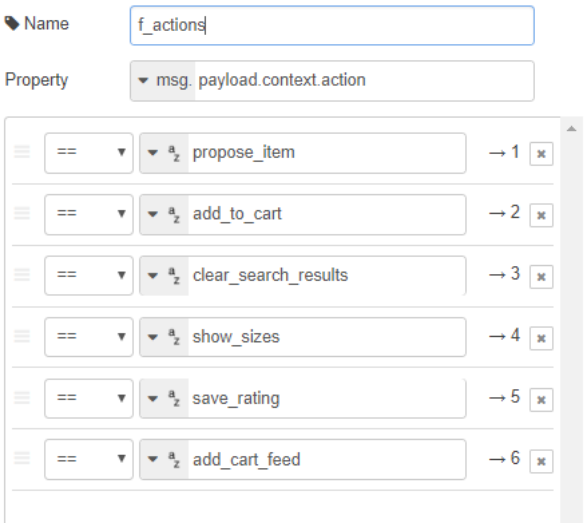


Figura 3.20: FEEDBACK - Azioni

FEEDBACK - Azioni In figura 3.20 sono riportate le azioni intercettate ed orchestrate dal workspace FEEDBACK.
L'azione "save_rating" viene riconosciuta nel momento in cui un prodotto è stato proposto e l'utente ha espresso una opinione : uno degli intenti 1StarIntent,2StarIntent,3StarIntent è stato riconosciuto. Il flusso viene indirizzato verso i nodi corrispondenti per aggiungere un record nella tabella FEEDBACK , in particolare verrà memorizzata una tupla (USER_ID,PROD_ID,RATING) dove la chiave primaria è data dalla coppia dei primi due che referenziano , rispettivamente, la tabella USERS e PRODUCTS. La query eseguita è parametrizzata :

```
query = "INSERT INTO feedback(USER_ID,PROD_ID,RATING) "
query+= " VALUES ("+msg.user+","+msg.itemID+","+msg.rate+")"
query+= "ON DUPLICATE KEY UPDATE RATING="+msg.rate
msg.topic = query
return msg;
```

Nel caso in cui la chiave primaria sia dupplicata il record viene aggiornato. Questa situazione si presenta quando un end-user ha già espresso un voto per uno specifico prodotto.
Memorizzato il feedback, il flusso ritorna al workspace e le alternative sono principalmente tre.
L'utente decide di vedere un altro item, l'azione impostata è "propose_item" ed il flusso viene deviato verso il sottoflusso incaricato a ricercare prodotti, in particolare la ricerca e la racomandazione è già stata effettuata e l'operazione effettuata è semplicemente quella di estrarre un nuovo item da

proporre. Da qui il procedimento è iterativo.
In alternativa l'utente può decidere di aggiungere al carrello il prodotto: occorre proporre e raccogliere la taglia del capo d'abbigliamento. A questo proposito, nel contesto viene impostata la variabile action al valore "show_sizes" che instrada il flusso per estrarre le taglie disponibili per questo prodotto e le imposta nel contesto del workspace FEEDBACK ed infine reindirizza il flusso al workspace stesso. Vengono mostrate le taglie disponibili sotto forma di suggerimento, un utente esprime la sua preferenza oppure se non è disponibile dichiara di non essere più interessato.
Se la taglia scelta è disponibile il prodotto viene aggiunto al carrello : action "add_to_cart". Inserendo quindi taglia ed identificativo del prodotto nel contesto del workspace order, incaricato di gestire il carrello. Viene inoltre cercato un abbinamento per il prodotto scelto, se presente viene proposto come combinazione, altrimenti si informa l'utente che l'oggetto è stato al carrello e la conversazione ritorna nello stato iniziale.
In fine un utente decide di smettere di visualizzare prodotti: occorre azzerare i risultati di ricerca , a questo proposito l'action "clear_search_results" viene utilizzata per instradare il flusso verso un nodo javascript che azzera le informazioni della ricerca ed imposta la variabile switch a null (per indicare che la conversazione ritorna al workspace ROUTER).
Le azioni discusse sono illustrate in figura 3.21

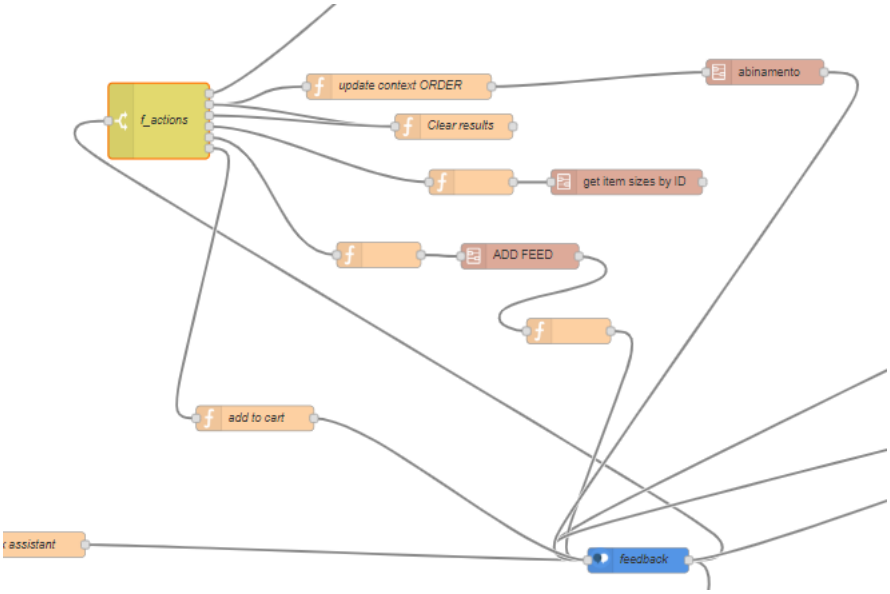


Figura 3.21: Sottoflusso azioni - FEEDBACK

3.4.2 Sistema di raccomandazione

Il metodo utilizzato è una tecnica di filtro collaborativo, in particolare la scelta allo stato dell'arte è l'utilizzo dell'algoritmo Alternating Least Squares con Weighted-Lambda-Regularization (ALS-WR), discusso nel parper "Large-Scale Parallel Collaborative Filtering for the Netflix Prize" [7].
L'algoritmo è parallelizzabile e scalabile, inoltre con l'aumentare del numero di iterazioni il modello il parametro lambda ci consente di regolarizzare il modello per evitare di avere grande overfitting. L'implementazione utilizzata è quella di Apache Spark, in linguaggio python.
Per poter interagire con il sistema di raccomandazione è necessario un meccanismo con il quale può essere richiamato dall'orchestratore. A questo proposito si utilizza Flask per la scrittura di API per la predizione: l'assistente virtuale dovrà semplicemente fare una chiamata http all'endpoint esposto dal sistema di raccomandazione per ottenere il risultato.
Il dataset utilizzato per allenare il modello è il contenuto della tabella FEEDBACK. Tutte le informazioni utili per la raccomandazione sono contenute in questa tabella: utenti, prodotti e rispettive preferenze. Per illustrare il caso d'uso, è stato generato un dataset random con 500 utenti pre-esistenti in modo da poter integrare il motore di raccomandazione con il resto del sistema.
Rimane il problema della predizione per i nuovi utenti: coloro i quali non hanno ancora espresso preferenze al momento dell'ultimo training del modello. Per natura un sistema di raccomandazione di filtro collaborativo necessità di avere feedback espressi dall'utente: l'assistente virtuale è incaricato a raccogliere delle opinioni obbligatorie.
Per studiare i nuovi utenti, l'agente chiede tre preferenze obbligatorie la prima volta che si ricerca un prodotto all'interno di una specifica categoria. Non sotto-categoria perché troppo specifico ed il catalogo prodotti è di piccole dimensioni. L'alternativa è richiedere delle opinioni in fase di configurazione su prodotti di tutte le categorie del catalogo ma questo approccio richiede un numero di feedback maggiore per una singola conversazione : un utente potrebbe non avere pazienza. La giustificazione tecnologica di questa scelta è data dalla scelta di implementazione della predizione per i nuovi utenti: si utilizza un modello di neirest neighbors utilizzando la libreria fornita da Scikit-Learn[8].
Il modello di nearest neighbors consente di calcolare la similarità di un utente rispetto agli altri. In particolare si utilizza un modello creato al momento della chiamata API corrispondente alla predizione per un nuovo

utente. E' necessario creare il modello live perché si hanno feedback per pochi prodotti, in particolare per ogni categoria si avrà i tre obbligatori. Ne consegue che il dataset utilizzato per la creazione di neirest neigbors è dato dalle tuple della tabella prodotti filtrate per PROD_ID in lista di ID per i quali si hanno i feedback , all'interno di una stessa categoria : al più tre feedback per ogni utente . Ne consegue un dataset di dimensioni ridotte che giustifica l'addestramento di un modello a run-time.
Se si utilizzasse l'intero dataset per il training del neirest neighbors si avrebbe il problema delle preferenze mancanti: su un numero elevato di prodotti del catalogo è altamente probabile che soltanto una piccola parte di questi siano stati valutati. Si potrebbe pensare di assegnare un valore di default per gli elementi mancanti ma questo ne conseguirebbe in un modello errato. perché verrebbero messe a confronto non soltanto le preferenze effettivamente espresse dall'utente nuovo ma anche quelle per le quali non si hanno dati.
Il sistema di raccomandazione è composto da due file sorgente, uno per l'addestramento del modello ALS-WR ed uno per la creazione del server in Flask.

Il codice "Addestramento del modello ALS", riportato al fondo della sezione, è utilizzato per il training del modello ALS, i parametri dell'algoritmo numberOfLatentFactors , n_iterations e lambda_c possono essere scelti mediante validazione dei parametri stessi tramite validazione: si divide il dataset in training e validation set e si inserisce un intervallo di valori per i tre parametri andando ad osservare il comportamento sul validation set: si cerca qual è la combinazione che produce il miglior compromesso tra errore quadratico medio (MSE) ,misura per valutare la qualità del modello, e tempi di training. Sono importanti i tempi di training perché è necessario addestrare nuovamente il sistema per accomodare i dati relativi ai feedback di nuovi utenti. La validazione è stata fatta per dimostrazione su un HP Spectre x360 13-ae060nz con processore Intel® Core™ i5-8250U, il codice eseguito è riportato in fondo alla sezione con il titolo "Validazione modello ALS".
I risultati ottenuti sono riportati nella tabella sottostante: diverse combinazioni riportano il valore minimo di errore quadratico medio (0.68), per scegliere i parametri migliori si osservano i tempi di training e si nota che sono paragonabili per tutte le combinazioni (con la dimensione del training set pari a circa 80 mila righe).La combinazione con numero di fattori a 20, iterazioni a 15 e lambda a 0.1 riporta un tempo di esecuzione leggermente inferiore rispetto alle altre. Tuttavia si preferisce , dato che que-

sta differenza è di 3s, la combinazione di 10 fattori latenti, 10 iterazioni e lambda pari a 0.1 per limitare la quantità di memoria RAM richiesta per la memorizzazione della matrice.

Factors	**Iterations**	**Lambda**	**Time [s]**	**MSE**
10	5	0.01	78.14	0.76
10	5	0.1	72.94	0.70
10	5	0.2	75.54	0.72
10	10	0.01	85.11	0.77
10	10	0.1	72.47	0.68
10	10	0.2	0.68	0.71
10	15	0.01	82.26	0.77
10	15	0.1	72.38	0.68
10	15	0.2	68.52	0.71
15	5	0.01	73.17	0.79
15	5	0.1	72.39	0.70
15	5	0.2	70.67	0.72
15	10	0.01	74.37	0.81
15	10	0.1	71.17	0.69
15	10	0.2	70.28	0.71
15	15	0.01	69.85	0.81
15	15	0.1	76.17	0.68
15	15	0.2	76.51	0.71
20	5	0.01	80.87	0.82
20	5	0.1	111.68	0.70
20	5	0.2	70.11	0.73
20	10	0.01	78.95	0.84
20	10	0.1	80.23	0.69
20	10	0.2	65.82	0.71
20	15	0.01	70.46	0.85
20	15	0.1	69.47	0.68
20	15	0.2	70.35	0.71

Tabella 3.1: Parameter tuning ALS model

Avendo un modello allenato non resta che esporre delle API per essere richiamabile dall'assistente. Le principali da esporre sono due: una per la predizione di utenti nuovi ed una per gli utenti che hanno già espresso preferenze e quindi i loro dati sono stati utilizzati per allenare il modello ALS. Si utilizzerà la stessa rotta con distinzione tra metodo GET e POST , GET per i "vecchi" utenti e POST per i nuovi. Si utilizza il metodo GET perché i dati trasportati non sono sensibili. In alternativa si può dividere in due rotte distinte entrambe con metodo POST. Oltre alle API per la predizione, una API per il training dall'esterno può essere implementata per esempio per aggornare il modello in base al numero di nuovi feedback ricevuti che solo l'assistente ed il database possono conoscere. La API alla route "/recommendation" definita per il metodi GET e POST è incaricata nel gestire la predizione. Nel metodo GET , per gli utenti esistenti, si effettua la predizione sul modello ALS-WR precedentemente allenato e caricato all'inizio dell'esecuzione del programma. Se l'utente non è presente nell'ALS allora un'eccezione viene lanciata nel momento in cui si chiama il metodo predict sul modello stesso: semplicemente torniamo come risposta un messaggio di errore per comunicare l'evento al chiamante in modo che possa riprovare sul metodo POST.
L'ID dell'utente e la lista di identificativi dei prodotti per i quali si vuole fare la predizione devono essere passati in forma url-encoded:
Esempio di chiamata GET: `http://127.0.0.1:5000/recommendation?user_id=123&item_ids=22,23,24`
Risposta:

```
[
    {
        "item_id": 22,
        "pred_rating": 1.906140154558606
    },
    {
        "item_id": 23,
        "pred_rating": 1.897904078375017
    },
    {
        "item_id": 24,
        "pred_rating": 1.0825413902651162
    }
]
```

Il metodo POST è incaricato della predizione per utenti nuovi, è richiesto come corpo della richiesta una lista di feedback. Dai ratings vengono selezionati i record corrispondenti solamente alla lista degli item per i quali si vuole la predizione per l'utente e si genera un modello di neirest neighbors su questo sottoinsieme. Si estraggono i tre neirest neighbors per limitare il problema degli outliers e si effettua la predizione per gli utenti più simili appena selezionati facendone la media ed utilizzandola come predizione per il nuovo utente.

Esempio di chiamata POST: http://127.0.0.1:5000/recommendation

```
{
  "user_id" : 53000,
  "feedback":[{
    "item_id":97,
    "rating":3
  },{
    "item_id":87,
    "rating":2
  },{
    "item_id":81,
    "rating":3
  }],
  "item_list":"1,2,3,4"
}
```

Risposta:

```
[ {
        "item_id": 1,
        "pred_rating": 1.3967569391355147
    },{
        "item_id": 2,
        "pred_rating": 1.937368345347492
    }, {
        "item_id": 3,
        "pred_rating": 1.9674239762974457
    },{
        "item_id": 4,
        "pred_rating": 2.7586887266491324
    }]
```

Il metodo POST alla route "/train" e' implementato per effettuare il training del modello dall'esterno, per autenticare la richiesta è utilizzato un segreto condiviso ed inoltre le richieste sono accettate solo se provenienti da localhost (i servizi sono sulla stessa macchina). Se si vuol utilizzare questo metodo e installare i servizi su macchine diverse è necessaria una forma di autenticazione più sicura, non basta il filtro sull'indirizzo IP per via dell'IP Spoofing. Ciò che avviene è l'avvio di un processo che esegue il codice "Addestramento del modello ALS" che produce un nuovo modello: occorre caricare nuovamente i dati di feedback ed il modello stesso.
Con l'aumentare delle dimensioni della tabella FEEDBACK la memoria necessaria per tenere i dati in memoria potrebbe non essere sufficiente, a questo proposito si potrebbe leggere i dati necessari dal database al momento della predizione.

3.4.3 Prodotti visivamente simili

In alternativa all'uso di Google Vision API, dato che non fornisce una indicazione di similarità tra due prodotti ma aiuta solamente a comprendere la tipologia del prodotto, viene illustrata in questa sotto sezione la soluzione progettata per il riconoscimento di prodotti che sono visivamente simili, applicata a due casi d'uso.

Primo caso d'uso Per il caso d'uso è stata scelta la categoria scarpe per l'implementazione di ricerca di prodotti simili a partire da una immagine ma l'approccio descritto è espandibile al resto dei prodotti.
Le fasi per la realizzazione della soluzione sono:

- Raccolta di immagini dal web per costruire il dataset di addestramento della Resnet34
- Transfer-learning rete Resnet34
- Costruzione modello Nearest Neighbor, usando i prodotti del catalogo, con l'uso della rete addestrata come feature-extractor
- Analisi dei colori dominanti mediante K-Means clustering
- Combinazione delle caratteristiche estratte dalla Resnet con l'analisi dei colori dominanti

Il processo di estrazione dei prodotti più simili è composto quattro stadi:

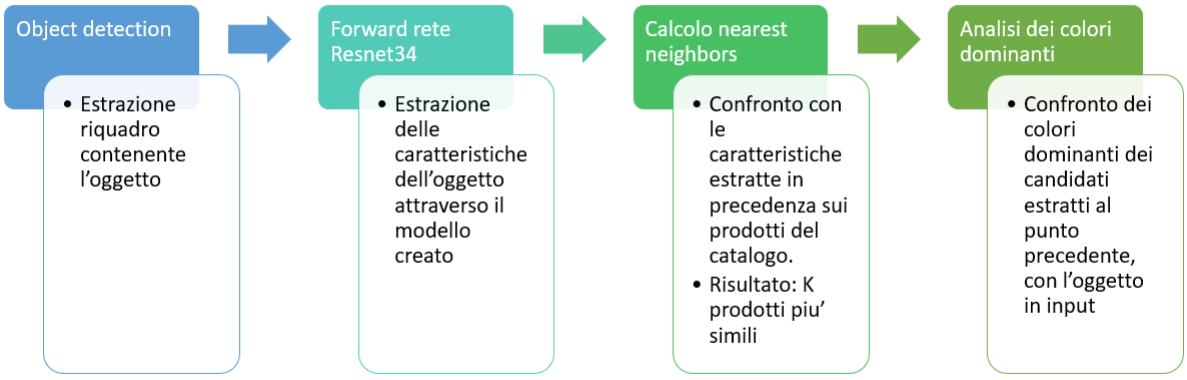


Figura 3.22: Processo prodotti simili

Transfer-learning Il Transfer-learning è una tecnica utilizzata in Deep Learning per migliorare le prestazioni. Invece che addestrare una rete da zero, viene inizializzata con i parametri della rete già addestrata su ImageNet[9]. Come è stato dimostrato dai ricercatori, questa tecnica consente di raggiungere accuratezze più alte quando non si ha a disposizione un database grande come ImageNet. I filtri che la rete ha imparato nei primi livelli convoluzionali e in quelli intermedi possono essere utilizzati per il riconoscimento di molteplici oggetti. Le convolutional neural networks (CNN) imparano le caratteristiche degli oggetti automaticamente ed in modo gerarchico, nei layer più prossimi all'output si hanno le features di alto livello. E' proprio qui che andiamo ad imparare nuovi valori per i filtri in modo da specializzare la rete, addestrata in passato su ImageNet, nel riconoscimento delle caratteristiche delle calzature. Questa tecnica è anche chiamata Fine-Tuning.
La necessità di addestrare la rete per imparare nuove caratteristiche ad alto livello è evidente dal questo risultato, figura 3.24, che è stato ottenuto utilizzando la rete Resnet34 originale addestrata su ImageNet come feature extractor. L'immagine di input è la prima in alto a sinistra, il resto sono in ordine la prima, seconda, terza, quarta e quinta più simile.

Figura 3.23: Prodotti simili senza fine tuning e senza analisi dei colori dominanti

Dataset Il dataset utilizzato per il Fine-Tuning della rete è stato costruito a partire dal web. E' stato adoperato un tool per la scannerizzazione automatica di immagini a partire da alcune parole chiave suggerite dalla ricerca di Google Immagini. In particolare sono state create cinque categorie di scarpe e si è diviso in training set e test set per valutare l'accuratezza del modello di classificazione:

Classe	**Training**	**Test**
Ankleboots	3555	481
Decollete	4334	497
Formal	3574	522
Sandal	4658	522
Sneakers	6161	630

Tabella 3.2: Dimensione dataset calzature

Transfer-learning Resnet34 La rete scelta per l'addestramento è la Resnet, nella versione 34. Gli esperimenti sono stati svolti su Google Colab

che pone a disposizione gratuitamente un ambiente accelerato da una GPU NVIDIA K80.
Con riferimento alla architettura riportata nella sezione 5.4, il Fully-Connected layer (fc) è stato cambiato per dare una dimensione dell'output pari al numero di classi nel trainig set, ovvero cinque, la rete originale ne prevede mille come le classi di ImageNet. I blocchi nominati layer1,layer2 e layer3 sono stati congelati per tenere inalterati i valori della rete preaddestrata sui loro parametri, l'ottimizzatore calcola i gradienti sugli ultimi layer: layer4 e Fully-Connected (fc).
In figura 3.24 sono riportati risultati che si ottengono dopo l'addestramento della rete con weight decay di 0.01, optimizer Adam[10] con learning rate pari a 0.0001. Alla epoca numero tre si ottiene l'accuratezza più alta e quindi questi parametri sono quelli salvati, nella successiva il modello inizia a soffrire di overfitting.

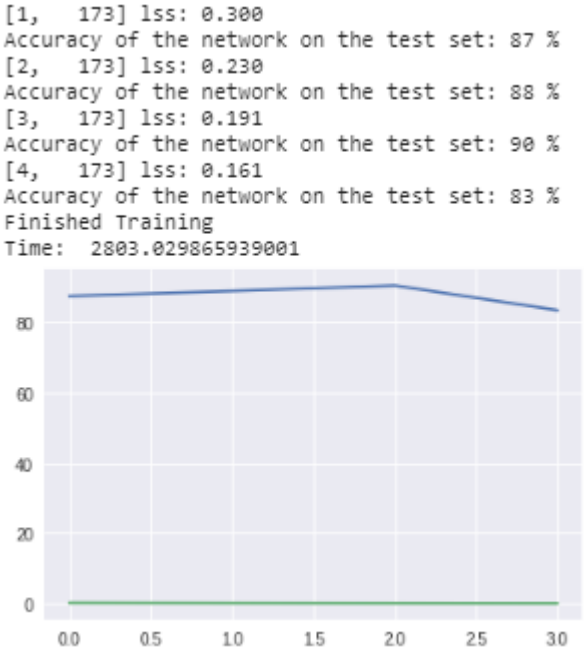


Figura 3.24: Risultati Transfer-Learning

Costruzione del modello Nearest Neighbor Utilizzando il modello derivato dal Transfer-Learning si estrae l'output del processo di forward in due punti, l'output finale che ne determina la classe e l'output dell'average pool prima del fully connected layer.
L'output finale viene utilizzato per ottenere la classe dell'oggetto e può essere utilizzato per la costruzione automatica di database per training futuri del modello oppure, se il numero di prodotti è elevato e si necessita di otti-

mizzare i tempi, per filtrare in base alla classe predetta e dividere il modello di Nearest Neighbor in cinque, uno per ogni classe. In questo modo si andrà a calcolare la misura di similarità soltanto all'interno della classe che è stata predetta.
L'output dell'avgpool contiene l'informazione delle caratteristiche estratte dalla rete, in particolare, come possiamo vedere dall'architettura sono 512. Queste ultime vengono utilizzate per trasformare il catalogo di N immagini in un dataset di N entry con ognuna 512 valori per calcolare un modello di Nearest Neighbors.

Analisi e calcolo della similarità dei colori dominanti L'analisi dei colori dominanti viene utilizza come informazione aggiuntiva al risultato prodotto dalla Resnet per cercare di favorire oggetti più simili dal punto di vista cromatico. In figura 3.25, la prima immagine a sinistra è l'input e le altre a seguire da sinistra verso destra solo la prima più simile, seconda, terza e quarta. La Resnet è stata in questo caso in grado di riconoscere e distinguere molto bene il tipo di calzatura e l'oggetto più simile calcolato è quello vero, ma la scarpa in posizione quattro è più simile all'input rispetto alla posizione due ed è ciò che si vuole correggere tramite l'analisi dei colori dominanti.

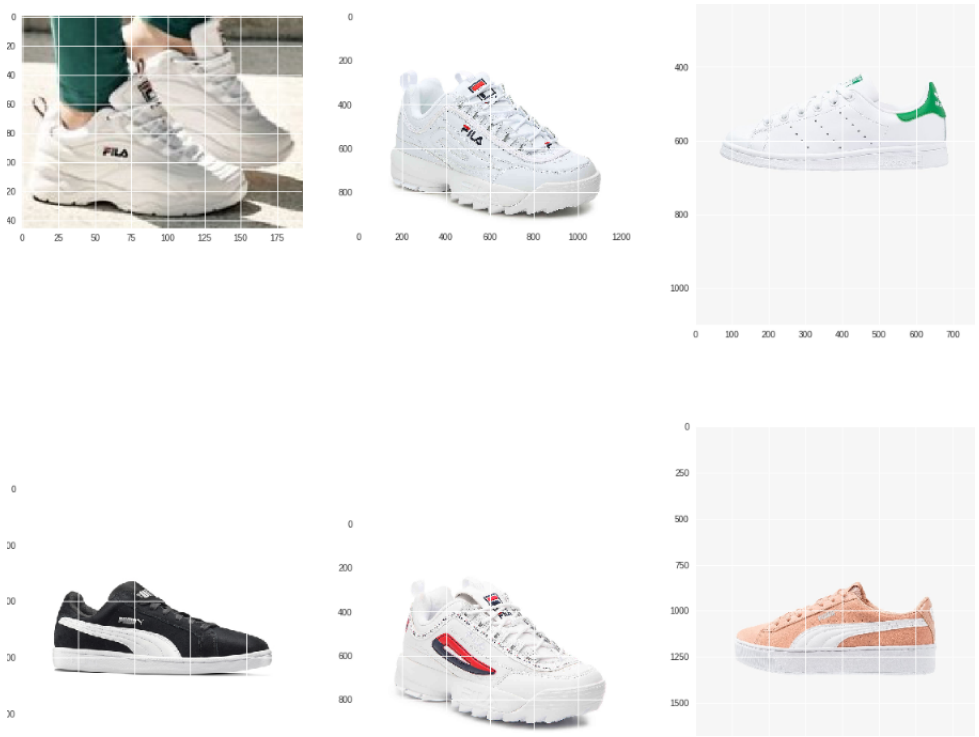


Figura 3.25: Prodotti simili con transfer-learning e senza analisi colori dominanti

Per l'analisi dei colori dominanti,la cui implementazione è riportata nella sezione 5.4.1, si utilizza una tecnica di clustering, in modo da formare dei clusters dei pixel. L'algoritmo utilizzato è K-Means con numero di clusters pari a cinque, determinato mediante diverse sperimentazioni.
Per ridurre un po' il rumore del background si ritaglia l'immagine a partire dal centro in proporzione 0.9 della scala originale, sono state provate diverse percentuali ed il rapporto 0.9 è risultato più efficace.
Dopo il center-crop il passo successivo è la riduzione della risoluzione dell'immagine, sperimentalmente una dimensione di 128x128 mantiene l'informazione significativa e velocizza notevolmente l'applicazione del clustering all'immagine.
Per clusterizzare i pixels si trasforma l'immagine da un array tridimensionale di forma NxMx3 in uno bidimensionale di forma (NxM)x3, in questo modo ogni riga è corrispondente ad una entry di tre valori R,G,B.
Oltre allo spazio RGB è stato sperimentato anche lo spazio YUV ed il secondo ha portato risultati migliori. Il secondo passo è quindi la trasformazione di ogni entry RGB in YUV.
A questo punto l'immagine è pronta per il clustering. Ne risultano N punti che sono i centroidi dei clusters: i colori dominanti. Per poter confrontare due immagini diverse è necessario definire una metrica, per il confronto tra i singoli centroidi si utilizza la distanza euclidea ma quale distanza prendere in considerazione nel caso in cui il primo colore dominante di una fotografia è il secondo colore dominante della fotografia di confronto ? Questo potrebbe essere il caso in cui nella prima immagine lo sfondo è meno presente rispetto alla seconda perché per esempio gli oggetti hanno forma diversa. Il confronto fra i colori dominanti di una immagine ed i centroidi dell'altra avviene in modo completo: si prende il primo centroide della immagine in input e si confronta con tutti i colori dominanti della seconda fotografia e ne si prende il minimo. Prendendo il minimo però potrebbe portare il rumore del background a determinare la misura di similarità in modo significativo. Per esempio nell'immagine in input, in alto a sinistra in figura 3.25, le ombre risultano tra i cinque colori dominanti che sono molto simili alla scarpa nera. Si introduce nella misura di similarità una penalità per quei colori dominanti che si trovano in posizione diversa, parametro regularizer della funzione computeDistance nel codice sotto riportato "Analisi dei colori dominanti". più i colori dominanti sono relativamente in posizioni differenti e più si applica un costo, in questo modo si favoriscono i colori dominanti che si ritrovano in posizioni più vicine, se due colori sono nella medesima posizione allora il valore di distanza è quello prodotto dalla distanza eucli-

dea. Se due colori sono in posizioni diverse, al valore di distanza euclidea si aggiunge un termine proporzionale alla differenza di posizione moltiplicato per una costante (regularizer) che ci permette di pesare questa penalità esternamente e determinare un valore ottimo mediante cross-validation.
Il risultato con analisi dei colori dominanti e senza regolarizzazione è il seguente:

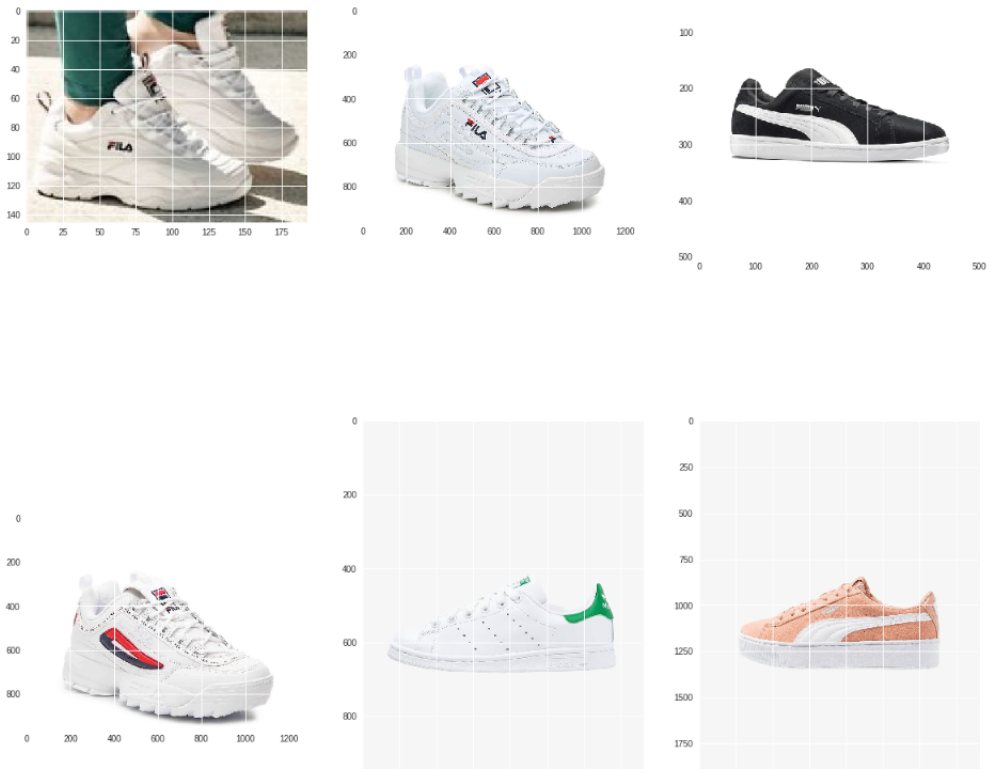


Figura 3.26: Prodotti simili con analisi colori dominanti e senza penalità

Applicando una penalità di 0.1 come valore del parametro regularizer:

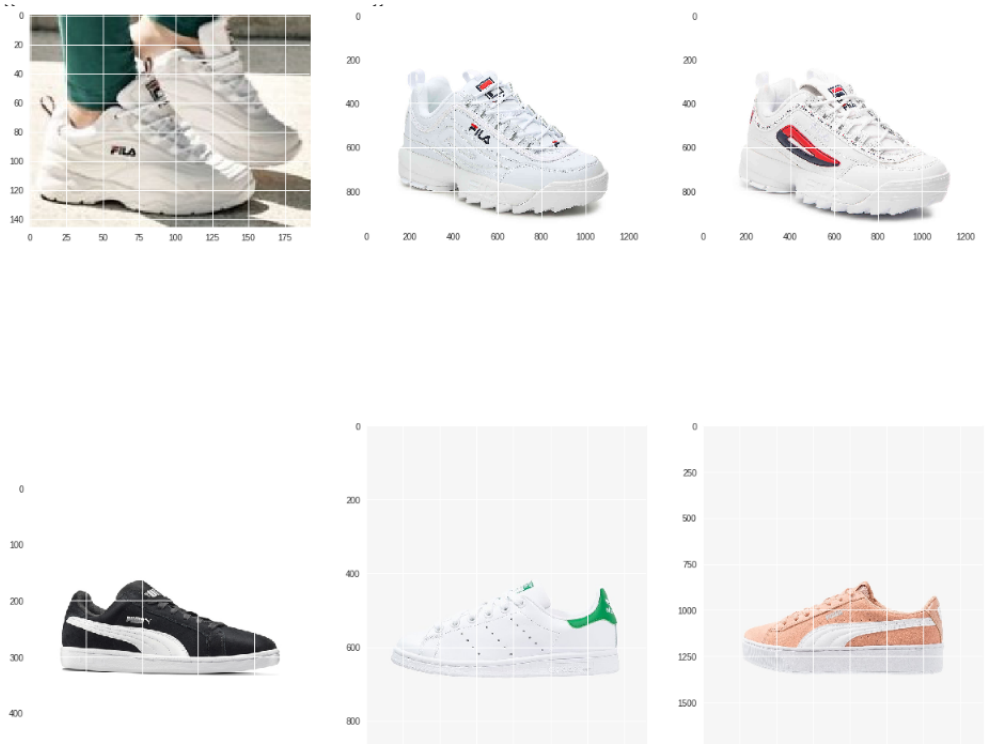


Figura 3.27: Prodotti simili con analisi colori dominanti e con penalità

Combinare l'analisi del colore con le distanze calcolate sulle features estratte dalla fine-tuned Resnet L'approccio utilizzato per mettere insieme queste due informazioni i passi seguiti sono i seguenti:

- Normalizzazione dei vettori di distanze calcolati, sia per il nearest neighbor, derivato dalle features estratte dalla fine-tuned Resnet, sia per l'analisi dei colori dominanti
- Con un parametro variabile, mediante test della soluzione, si determina in quale percentuale considerare l'informazione del colore

Considerare completamente il colore per andare a riordinare come gli oggetti sono riportati per similarità prodotta dal primo passo può condurre a risultati molto sbagliati e quindi andando a sommare le due distanze con lo stesso peso non è la strategia giusta. Per illustrare questo esempio si può considerare il caso in cui la prima informazione delle caratteristiche prodotte dalla Resnet siano simili per diversi oggetti e fra questi ve ne siano alcuni molto diversi ma che hanno lo stesso colore dell'oggetto in input. In questo caso, se si considera troppo l'informazione dei colori dominanti, un oggetto diverso ma con lo stesso colore viene portato in testa.
Il peso dell'informazione dei colori dominanti è quindi un parametro da

determinare mediante cross-validation. A seconda degli oggetti presi in considerazione, tramite osservazioni multiple si può determinare il valore ottimale.

Nella sezione 5.5.1 sono riportati alcuni esempi di query, direttamente dall'interazione con l'assistente virtuale.

Secondo caso d'uso L'uso di reti resnet preaddestrate si è rivelato performante anche nel riconoscimento di borse, l'architettura utilizzata è simile alla soluzione del primo caso d'uso. L'unico elemento che cambia è la rete neurale convoluzionale che viene sostituita dalle reti Resnet50 e Resnet152 senza transfer-learning.

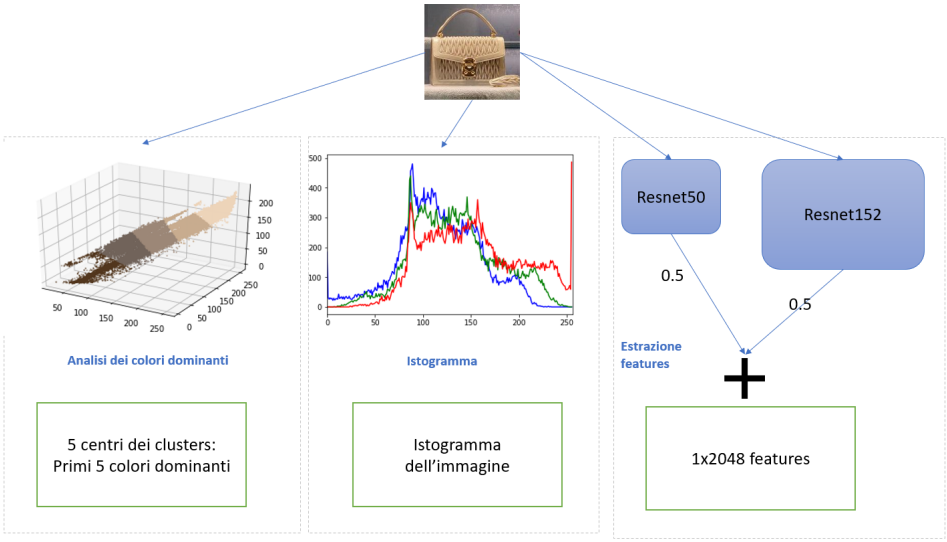


Figura 3.28: Estrazione delle caratteristiche del prodotto

Estrazione delle caratteristiche del prodotto L'estrazione delle caratteristiche dell'immagine in input è descritta in figura 3.28:

- L'immagine viene processata dalla Resnet50 e dalla Resnet152, l'output delle reti è di dimensione 2048 e viene sommato utilizzando peso 0.5 per ciascuna. I pesi sono stati determinati sperimentalmente.
- Viene calcolato l'istogramma dell'immagine che descrive la distribuzione dei pixels RGB, l'istogramma riportato è quello dell'immagine in input.
- Si applica il clustering per l'estrazione dei colori dominanti, l'immagine 3.28 rappresenta il clutering 3D dei pixels della fotografia in input dove ogni pixel è stato colorato con il colore del suo centroide (colore dominante).

Per ogni prodotto del catalogo si applica la procedura descritta ad ogni sua immagine e si memorizzano i risultati per evitare calcoli ridondanti.
Per ottenere la predizione occorre prima di tutto ripetere la stessa procedura applicata per l'estrazione delle caratteristiche delle immagini di confronto. Terminato il processo di estrazione si può procedere nel determinare i prodotti che sono più simili come in figura 3.29

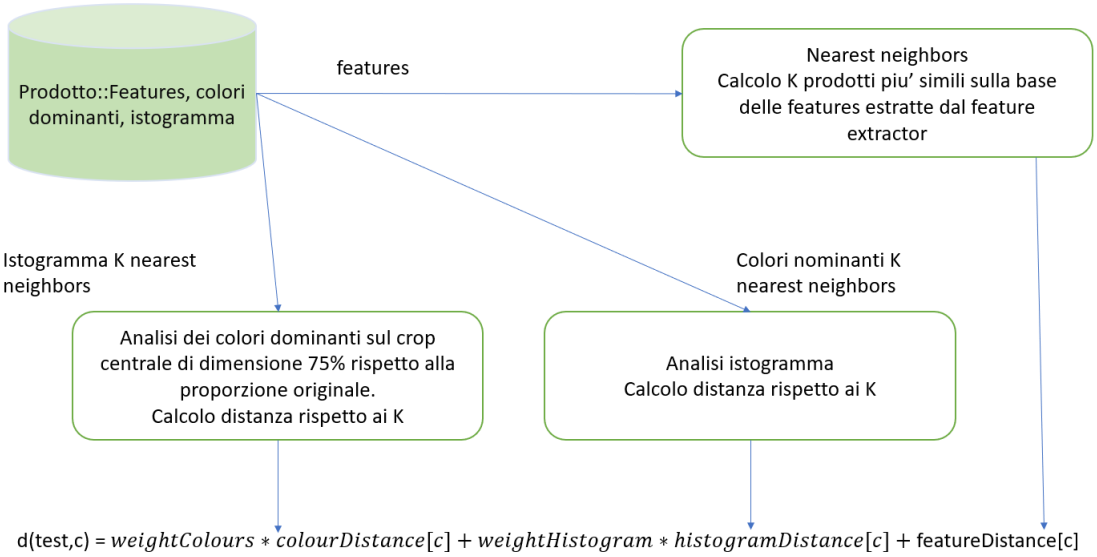


Figura 3.29: Calcolo distanza prodotti simili

I passi principali sono:

- Estrazione caratteristiche dell'immagine (resnet,clustering,istogramma).
- Calcolo dei K prodotti più simili mediante l'uso delle caratteristiche estratte dalle reti (features).
- Confronto dell'istogramma dell'immagine in input con l'istogramma di ogni candidato presente nel set di K candidati estratti al punto precedente. Si ottiene un array di distanze
- Calcolo di un array di dimensione K: un elemento per ogni candidato rappresentativo della distanza tra i colori dominanti
- Calcolo del valore finale di distanza con la formula riportata in figure 3.29, dove i pesi weightColours e weightHistogram sono utilizzati per limitare l'influenza delle informazioni dei colori dominanti e istogramma, rispettivamente. Questo viene fatto per non stravolgere il

risultato prodotto dalle reti neurali convoluzionali, le analisi del colore sono solamente un termine correttivo. Se una immagine è molto distante dal punto di vista delle reti allora questa immagine non deve poter raggiungere la prima posizione soltanto perché ha lo stesso colore del prodotto in input.

Per ogni modello di borsa sono disponibili quattro fotografie, un esempio è riportato in figura 3.30: tre vengono utilizzate per il training del modello ed una viene adoperata come test per calcolare gli indicatori di performance del modello. La figura di test è stata ritagliata da una fotografia più grande, dove la borsa è indossata da una modella. E' quindi diversa sia in termini di luce che di posizionamento rispetto alle fotografie utilizzate come confronto. Ogni modello di borsa è descritto da un codice, ogni fotografia contiene nel nome del file il codice del prodotto.

(a) Retro (b) Fronte

(c) Laterale (d) Test

Figura 3.30: Campione immagine training set ed immagine di test utilizzata per valutare la soluzione

Gli indicatori di performance sono relativi al riconoscimento del codice del prodotto: si ha un match positivo se il prodotto di test è lo stesso della predizione. E' necessario specificare che è diverso rispetto ad una classificazione: non è sufficiente determinare una classe ma individuare lo stesso oggetto ed è quindi un problema più difficile. Per esempio, se il prodotto in prima posizione è uguale al test allora si ha un match in prima posizione;

se il test è stato trovato in quarta posizione allora si ha un match nei primi cinque.

La soluzione è stata testata con lo scopo di calcolare i seguenti KPI:

- KPI_1 = Percentuale del numero di campioni riconosciuti in prima posizione
- KPI_2 = Percentuale del numero di campioni riconosciuti nelle prime tre posizioni
- KPI_3 = Percentuale del numero di campioni riconosciuti nelle prime cinque posizioni

In totale le borse sono 324 ed i risultati ottenuti su 324 foto di test, una per ogni prodotto, con la soluzione con pesi weightColour = 0.1, regularization = 0.1 e weightHistogram = 0.15 sono:

- KPI_1 = 50%
- KPI_2 = 73%
- KPI_3 = 80%

Alcuni esempi di query ,che riconoscono il prodotto nelle prime cinque posizioni, sono riportati nella sezione 5.5.2. Come possiamo osservare, visivamente i risultati hanno tutti delle caratteristiche in comune che danno una misura di similarità elevata con riferimento all'immagine di test.
Per avere una idea più significativa dei risultati prodotti, come sensazione percettiva di similarità è utile visualizzare il risultato prodotto per il test in figura 3.31. I prodotti scelti dall'algoritmo hanno tutti delle caratteristiche in comune: stesso pattern di design, scritte argentate ,catenella e stessa forma per le prime quattro posizioni. Dal punto di vista percettivo, è un buon risultato.

Test della vetrina

Figura 3.31: Prodotto non riconosciuto: visualizzazione dei risultati più simili

(a) Caso 7

(b) Caso 8

(c) Caso 9

(d) Caso 10

Figura 3.32: Casi d’uso test di validazione - 3

3.5 Persistence Layer

Il livello di persistenza è rappresentativo delle operazioni volte alla memorizzazione stabile delle informazioni. Fanno parte di questo modulo tutte le operazioni necessarie per l'interazione con il database, alcuni esempi: ricerca dei prodotti del catalogo, persistenza del contesto e memorizzazione delle conversazioni.

3.5.1 Persistenza del contesto

Come espresso nel capitolo due, il contesto dell'assistente rappresenta lo stato di una conversazione. Guasti, manutenzione ed aggiunta di nuove funzionalità possono portare ad un restart del server. Per riprendere un dialogo, in presenza di questo evento, è necessario che questi dati siano memorizzati in memoria di massa.
In generale è opportuno tenere traccia di ogni dato temporaneo che viene utilizzato per più di un singolo scambio domanda-risposta. Se ciò non viene implementato e nel mezzo del dialogo avviene un guasto al server, al riavvio, l'assistente non avrebbe più le informazioni necessarie per continuare l'interazione in modo naturale.
Poichè ogni utente è identificato da un ID univoco si utilizza una memorizzazione di tipo Chiave-Valore, con ID chiave e valore il contesto.

La struttura del contesto è definita nel seguente modo, dove ogni valore può essere un oggetto complesso:

```
"contesto":{
    "key":user,
    "router": contesto_router,
    "feedback":contesto_feedback,
    "order":contesto_order,
    "finder":contesto_finder,
    "faq":contesto_faq,
    "search_results": risultati_ricerca,
    "shopping_cart": carrello
    "invoice": fattura_temporanea
}
```

più specificamente gli oggetti di tipo contesto fanno riferimento alle specifiche definite nella documentazione di Watson Assistant[2]. Gli oggetti "search_results", "shopping_cart" e "invoice" sono definiti internamente al sistema. Il tipo "Invoice" deve racchiudere tutte le informazioni richieste dai canali di comunicazione dato che, per l'invio della fattura all'utente, è necessario rispettare il formato di canale definito nella documentazione ufficiale di ogni piattaforma.

```
  invoice = {
  "chat_id":chatId,
  "provider_token": provider_token,
  "start_parameter": start_parameter,
  "title": title,
  "description": description,
  "currency": currency,
  "payload": payload,
  "prices": [
        {"price" = {
           "label":descrizione_oggetto,
           "amount": prezzo_oggetto
        }
       }
  ],
  "need_shipping_address": true};
```

Invoice

L'oggetto "shopping_cart" contiene l'informazione relativa ai prodotti che sono attualmente nel carrello della spesa, l'assistente tiene traccia, per ogni prodotto aggiunto di un oggetto nel formato:

```
[
    "item":{
        "id":codice_prodotto,
        "price":prezzo_prodotto,
        "size":taglia_scelta
    }
]
```

Shopping cart

L'oggetto "search_results" viene impostato a seguito di una ricerca di prodotti, contiene al suo interno una lista di prodotti estratti dal database e la relativa informazione della posizione del prodotto che è stato presentato per ultimo al fine di poter scorrere la lista in modo ordinato.
Conclusa l'operazione che necessita di un'informazione temporanea, questa viene cancellata dal contesto per liberare memoria.
Definito un formato non resta che decidere quando aggiornare i valori nel database. La soluzione migliore è aggiornare i dati quando avviene un cambiamento: nuova ricerca, nuovo stato raggiunto nel modello conversazionale ed aggiunta di un record alla fattura sono degli esempi. Nell'implementazione questa operazione può essere eseguita in parallelo come avviene in figura 3.13.
Un messaggio viene ricevuto dal workspace Router: lo stato attuale e le variabili di contesto possono cambiare. Il messaggio prosegue nel flusso verso successive elaborazioni ed in parallelo avviene anche la memorizzazione stabile. In maniera analoga avviene la memorizzazione del contesto locale degli altri sotto-modelli del modello conversazionale e dei risultati della processo di ricerca con raccomandazione. Eseguendo le operazioni in maniera concorrente e non essendoci conflitti sulla risorsa, le prestazioni non subiscono variazioni rilevanti.

3.5.2 Memorizzazione delle conversazioni

Tenere traccia dei messaggi scambiati tra user ed assistente virtuale presenta molteplici vantaggi dal punto di vista dell'ingegnere e del business.
Innanzitutto è fondamentale per migliorare il modello conversazionale: osservando i dialoghi reali si può rilevare quali intenti ed entità sono scarsa-

mente riconosciuti ed agire di conseguenza, ampliando il training set con gli esempi effettivamente utilizzati.
Dal punto di vista del business si è in grado di ottenere una visione in grado di ottimizzare le campagne marketing e identificare nuovi modi per raggiungere ed ingaggiare i consumatori.
Osservare le conversazioni permette di monitorare le parole e le frasi da utilizzare per parlare lo stesso linguaggio dell'utente. Il consumatore vuole essere ascoltato e avere risposte concrete, espresse in un linguaggio a lui familiare, per le sue esigenze. Dal punto di vista del marketing, la visibilità del brand aumenterà, crescerà la fiducia tra cliente ed assistente e questo porterà ad un incremenento del numero dei clienti potenziali.
La struttura dati definita è relativa all'intero storico delle conversazioni per utente. Oltre ai messaggi scambiati, vi si aggiungono alcune statistiche per ogni evento rilevante, calcolate in modo incrementale quando si verifica l'evento.
La struttura dati è la seguente:

```
{
    "user":userID,
    "statistics":{
        "startSearch":numero_di_ricerche_iniziate,
        "endSearch":numero_di_ricerche_portate_a_termine,
        "seenProducts":numero_di_prodotti_visualizzati,
        "buyProducts":numero_di_prodotti_acquistati,
        "toCartProducts":
           numero_di_prodotti_inseriti_nel_carrello,

    },
    "conversation":[
            "payload":{
                "type": ENUM(SPEDITO,RICEVUTO),
                "content":text,
                "time":timestamp,
                "intents":[
                    {
                        "intent":intent_name,
                        "confidence":confidenza_intent
                    }
                ],
```

```
                "entities":[
                    "entity":entity_name,
                    "confidence":confidenza_entity
                    "position":[
                        "start":startIndex,
                        "end":endIndex
                    ],
                    "value":valore_entity
                ]
            }
    ]
}
```

La chiave è l'identificativo utente. La struttura "conversation" è una coda FIFO contenente elementi di tipo "payload", la naturale sequenza dei messaggi viene mantenuta: ogni messaggio viene inserito in coda nell'ordine in cui viene generato. Il campo "payload" rappresenta un singolo messaggio. Esso contiene l'informazione di chi ha inviato il messaggio , assistente o utente. Il campo timestamp è inserito per dare un contesto temporale a chi legge la conversazione. Le informazioni relative alla classificazione sono contenute negli array "intents" ed "entities" e ne riportano i valori e la confidenza.
L'ultimo campo è relativo alle statistiche che si possono calcolare incrementalmente dall'assistente virtuale. Nella struttura dati vi sono riportate degli esempi ed in modo modulare vi si possono aggiungere metriche di interesse.
L'approccio utilizzato permette di allegerire il compito della ricostruzione della conversazione e del calcolo delle statistiche: se i messaggi fossero memorizzati in modo piano, senza una struttura di questo tipo, al momento della composizione della chat sarebbe necessaria l'aggregazione di tutti i messaggi relativi al singolo utente. Tale operazione può risutare computazionalmente complessa.

Visualizzazione delle conversazioni Allo scopo sopra illustrato è stata sviluppata una interfaccia per allegerire l'analisi della conversazione. Nella dashboard in figura 3.33 ,selezionando l'utente desiderato si può visualizzare il dialogo con l'assistente e le relative informazioni ed in particolare le informazioni rugardanti gli intenti riconosciuti per una determinata frase in input. Nella conversazione in figura 3.33 si può notare come l'input utente

(a destra) corrispondente alla frase "What's available?" viene classificato come un intento di chiarimento ma con una confidenza di 0.69. Dal contesto in cui viene utilizzato questa frase è effettivamente un valido esempio per chiedere maggiori informazioni e di conseguenza l'azione da prendere è aggiungerlo al training set dell'intento.

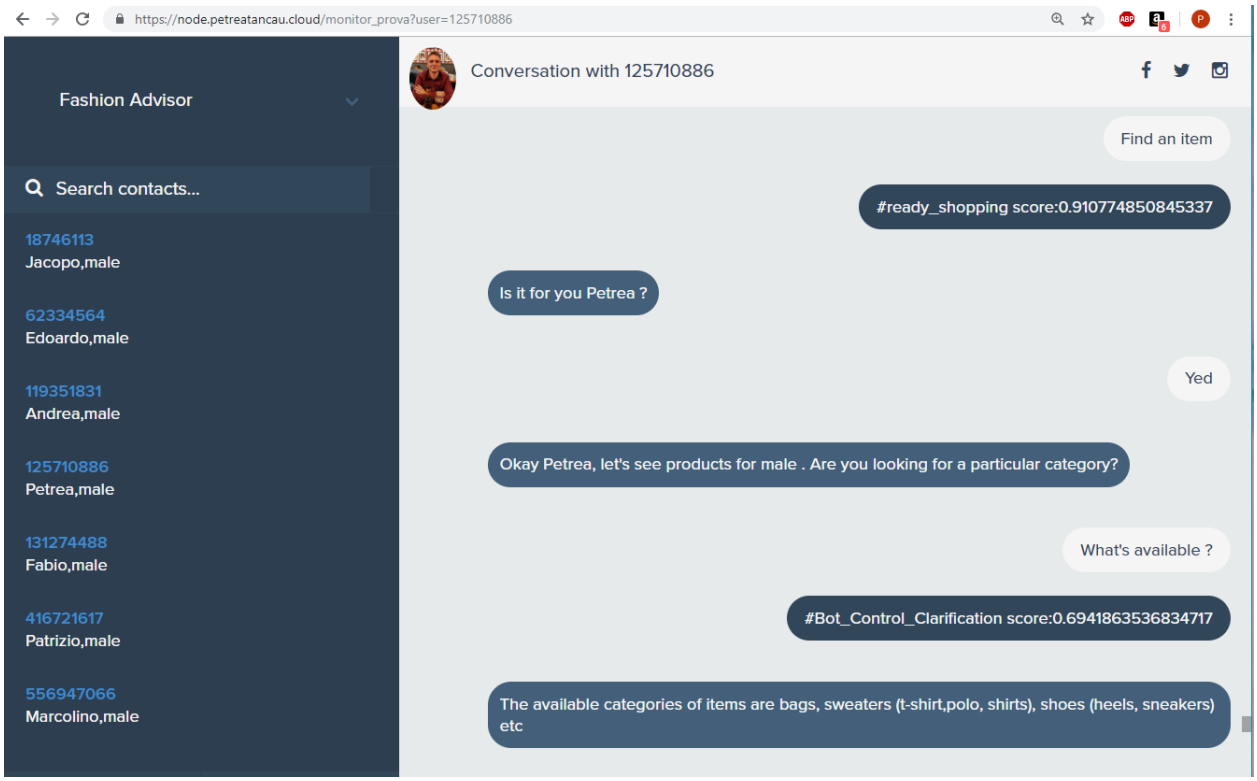


Figura 3.33: Dashboard - conversazioni utente

Capitolo 4

Deployment e considerazioni finali

4.1 Posizionamento dei servizi e tempo di risposta

I servizi utilizzati sono molteplici ed alcuni di questi sono di terze parti e disponibili in cloud. Per ottimizzare le prestazioni, ovvero minimizzare il tempo di risposta, bisogna limitare quanto più possibile il numero dei servizi esterni.
Il server Node-RED interagisce numerose volte con il database e con il sistema di raccomandazione. La scelta migliore è averli almeno nella stessa rete locale, in modo da minimizzare il tempo trascorso in internet.
Con l'aumentare del numero di utenti e di conseguenza dei dati, la singola macchina potrebbe non scalare adeguatamente. Almeno il training del sistema di raccomandazione dovrebbe essere eseguito su una macchina differente ed adeguata per tali operazioni. Poichè l'algoritmo ALS è parallelizzabile, l'addestramento può essere eseguito su un cluster ed il modello esportato sulla macchina che espone il server Flask contenente il motore di raccomandazione.
Le conversazioni possono essere memorizzate per profilazione, analisi e miglioramento del modello conversazionale. Con l'aumento del numero di utenti e di conseguenza dei messaggi scambiati, il disco potrebbe essere saturato. Per scalabilità nasce la necessità di memorizzare questi dati esternamente in una struttura adatta. In questo caso le prestazioni non sono da prendere in considerazione poichè tale operazione può essere svolta in parallelo e non influisce sul tempo di risposta dell'assistente virtuale.

Il database può essere diviso in tre servizi:

- uno di tipo relazionale per i dati relativi a feedback e catalogo prodotti. Il risultato delle query eseguite su questo servizio è necessario per procedere nel flusso e quindi è preferibile averlo nella stessa rete locale.
- Un database non relazionale nella stessa rete locale per la memorizzazione del contesto: sebbene l'aggiornamento è parallelizzato, l'operazione di lettura del contesto è necessaria per procedere nel dialogo. L'assistente virtuale può tenere in memoria RAM le conversazioni attualmente attive per limitare il numero di letture, questo richiede un'attenta gestione della memoria con un dimensionamento adeguato sulla base di una previsione del numero di utenti attivi in contemporanea.
- Un database non relazionale per la memorizzazione delle conversazioni, senza vincoli di posizionamento (anche in cloud)

4.2 Funzionalità e sviluppi futuri

L'analisi dell'utilizzo del sistema può portare allo sviluppo delle personalità di due o più agenti. Osservando l'interazione le due personalità possono essere adattate per un pubblico femminile o maschile e differenziando per età. Il primo sviluppo futuro consiste nell'**arricchimento delle abilità comunicative** dell'agente.
La specializzazione dell'assistente in ambito moda può essere ampliata con un catalogo reale di prodotti. Uno sviluppo futuro consiste nell'identificare e gestire la configurazione di un prodotto in base alla sua tipologia, gestendo non soltanto taglia ma anche vestibilità, colore e caratteristiche proprie di un particolare tipo di prodotto.
Sviluppo graduale del modello conversazionale in lingua nativa, limitando l'uso del traduttore.
Estensione del servizio di prodotti visivamente simili agli altri capi di moda, anche più di uno per fotografia di input.

Capitolo 5

Codice ed approfondimenti

5.1 Google Vision API combinazione dei tagging

```
1  tags = msg.payload.responses[0].labelAnnotations
2  web = msg.payload.responses[0].webDetection.webEntities
3  //list_colors contiene i colori semplici riconosciuti dall'assistente al
       momento
4  list_colors = ["black","brown","beige","grey","green","blue","red","white","
       pink","purple","yellow","silver","bronze","gold","denim"]
5  msg.color = []
6  msg.image_classification = []
7  var labels = []
8  var scores = []
9  msg.category = []
10 // estrazione etichette e score ritornate da "labels"
11 for(i=0;i<tags.length;i++)
12 {
13     if(tags[i].score > 0.4 && tags[i].description!= null && tags[i].
           description!=undefined){
14       labels.push(tags[i].description)
15       scores.push(tags[i].score)
16     }
17    for(j=0;j<list_colors.length;j++)
18     if(tags[i].description == list_colors[j])
19     {
20         msg.color.push(tags[i])
21     }
22 }
23 // estrazione etichette e score ritornate da web entities
24 for(i=0;i<web.length;i++)
25 {
```

```
26      if(web[i].score > 0.4 && web[i].description!= null && web[i].description
            !=undefined){
27        labels.push(web[i].description)
28        scores.push(web[i].score)
29      }
30     for(j=0;j<list_colors.length;j++)
31      if(web[i].description == list_colors[j])
32      {
33          msg.color.push(web[i])
34      }
35  }
36  //per semplicita le etichette vengono convertite in lowercase
37  labels = labels.map(x => x.toLowerCase())
38  /*a questo punto abbiamo nell'array labels tutte le etichette e nell'array
        scores la confidenza della etichetta associata (stessa corrispondenza
        per indice)*/
39  //ESEMPIO PILOT SUNGLASSES
40  if(labels.includes('sunglasses') && labels.includes('aviator sunglasses'))
41      {
42            result = {
43              "category" : "Sunglasses",
44              "subcateogry" : 'Pilot',
45              "display":"aviator sunglasses",
46              "score": (scores[labels.indexOf("sunglasses")]+scores[labels.
                    indexOf("aviator sunglasses")])
47          }
48          msg.image_classification.push(result)
49      }
50  //SHIRT
51  if(labels.includes('dress shirt'))
52  {
53          result = {
54              "category" : "Sweater",
55              "subcateogry" : 'Shirt',
56              "vestibilita" : 'Elegant',
57              "display":"dress shirt",
58              "score": scores[labels.indexOf("dress shirt")]
59          }
60          msg.image_classification.push(result)
61  }
62  else if(labels.includes('shirt'))
63  {
64          result = {
65              "category" : "Sweater",
66              "subcateogry" : 'Shirt',
67              "display":"shirt",
```

```
68              "score": scores[labels.indexOf("shirt")]
69          }
70          msg.image_classification.push(result)
71  }
72
73  //SNEAKERS -- CASO IN CUI MOLTEPLICI CLASSI RAPPRESENTANO LO STESSO PRODOTTO
74          if((labels.includes("sneakers") || labels.includes("walking shoe")
                || labels.includes("running shoe") || labels.includes("athletic
                shoe")) && !labels.includes('oxford shoe'))
75          {
76              sc = []
77              n=labels.indexOf("sneakers")
78              if(n>=0 && n<scores.length)
79              {
80                sc.push(scores[n])
81              }
82               n=labels.indexOf("walking shoe")
83               if(n>=0 && n<scores.length)
84              {
85                  sc.push(scores[n])
86              }
87              n=labels.indexOf("running shoe")
88              if(n>=0 && n<scores.length)
89              {
90                  sc.push(scores[n])
91              }
92               n=labels.indexOf("athletic shoe")
93                 if(n>=0 && n<scores.length)
94              {
95                  sc.push(scores[n])
96              }
97              result = {
98              "category" : "Shoes",
99              "subcateogry" : 'Sneakers', "display":"sneakers",
100             "score": Math.max(...sc)
101         }
102         msg.image_classification.push(result)
103         }
104         else if(labels.includes('oxford shoe'))
105         {
106              result = {
107             "category" : "Shoes",
108             "subcateogry" : 'Laceup',
109             "display":"formal shoes, it seems like oxford shoes to me",
110             "score": scores[labels.indexOf('oxford shoes')]
111             }
```

```
112         }
```

5.2 Google channel layer

```
1  //imposta headers
2  msg.headers={}
3  msg.headers["Content-Type"] = "application/json"
4  var suggestions;
5  TextToSpeech = "";
6  card = null;
7  //unisci coda fifo
8  for(i = 0 ; i< msg.output.length ; i++)
9  {
10     payload = msg.output[i].payload;
11     msg.flags = msg.output[i].flags;
12     context = global.get("context-"+msg.user)
13     //esempio di suggerimento
14     if(msg.flags=="show_sizes")
15     {
16         suggestions = [];
17         //aggiungi taglie al suggerimento
18         for(i=0;i<msg.sizes.length;i++)
19             suggestions.push({"title":msg.sizes[i]})
20     }
21     //testo o immagine
22     if(payload.type == "message" && payload.content != "" && payload.content
           != null && payload.content != undefined)
23             TextToSpeech+=payload.content+"\n";
24     else if(payload.type == "photo"){
25         card = {
26                   "title": "Here is what I recommend !",
27                   "formattedText": payload.caption
28                   "image": {
29                               "url": payload.content,
30                               "accessibilityText": "Image alternate text"
31                                 },
32                                "buttons": [
33                                     {
34                                         "title": "View more",
35                                         "openUrlAction": {
36                                             "url": payload.content
37                                         }
38                                     }
39                                 ],
40                              "imageDisplayOptions": "CROPPED"
41                             }
```

```
    }
}
//costruzione della risposta
msg.payload = {
  "payload": {
    "google": {
      "expectUserResponse": true,
      "richResponse": {
        "items": [
          {
            "simpleResponse": {
              "textToSpeech": TextToSpeech
            }

          }
        ]
      }
    }
  }
}
if(card != undefined && card != null)
    msg.payload.payload.google.richResponse.items.push({"basicCard":card})
if(suggestions != undefined && suggestions != null)
    msg.payload.payload.google.richResponse["suggestions"]=suggestions
//inoltra se e solo se il messaggio non e' vuoto
if(TextToSpeech != "" && TextToSpeech != "\n" || card != null )
return msg;
```

5.3 Moduli del sistema di raccomandazione

```
1  from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel,
       Rating
2  from pyspark import SparkContext
3  from pyspark.sql import SparkSession
4  import mysql.connector
5  import pandas as pd
6  sc = SparkContext()
7  spark = SparkSession(sc)
8  conn = mysql.connector.connect(
9    host="localhost",
10   user="***",
11   passwd="***",
12   database="***"
13 )
14 df = pd.read_sql("SELECT * FROM FEEDBACK",conn)
15 df['LABEL'] = df['RATING']
16 df.drop('RATING',axis=1,inplace=True)
17
18 conn.close()
19
20 ratings = spark.createDataFrame(df)
21 # convert into a sequence of Rating objects
22 ratings = ratings.rdd.map(lambda p: Rating(int(p[0]), int(p[1]),
23                                            int(p[2])))
24
25 train = ratings
26 numberOfLatentFactors = 10
27 n_iterations = 10
28 lambda_c = 0.1
29 model = ALS.train(ratings, numberOfLatentFactors, iterations=n_iterations)
30 import shutil
31 shutil.rmtree('./rc_model')
32 import sys
33 model.save(sc, "./rc_model")
```

Addestramento del modello ALS

```
train,test = ratings.randomSplit([0.7,0.3])

lambdas = [0.01,0.1,0.2]
iter=[5,10,15]
features = [10,15,20]
bestmse = 10
for l in lambdas:
  for i in iter:
    for f in features:
      start_time = time.time()
      model = ALS.train(train, f, iterations=i,nonnegative=True,lambda_=l,
          seed=42)
      # test the model
      model.productFeatures().cache()
      model.userFeatures().cache()
      x = test.map(lambda p: (p[0], p[1]))
      p = model.predictAll(x).map(lambda r: ((r[0], r[1]), r[2]))
      ratesAndPreds = test.map(lambda r: ((r[0], r[1]), r[2])).join(p)
      mse = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
      print("------------- Factors = %s - Iterations = %s - Lambda = %s
          -----------" % (f, i, l))
      print("------------- %s seconds ---------------" % (time.time()-
          start_time))
      print("------------- Train MSE: %s -------------" % mse)

      if mse < bestmse:
        bestmse  = mse
        bestL = l
        bestI = i
        bestF = f

print("-------------best mse MSE: %s -------------" % mse)
print("-- parameters: lambda %s----" % bestL)
print("-- parameters: num iter %s----" % bestI)
print("-- parameters:  number of latent factors %s----" % bestF)
```

"Validazione modello ALS"

```
1  from flask import Flask, jsonify, request
2  from flask_cors import CORS
3  import os
4  from sklearn.neighbors import NearestNeighbors
5  import pandas as pd
6  import mysql.connector
7  from pyspark import SparkContext
8  from pyspark.mllib.recommendation import MatrixFactorizationModel
9  import numpy as np
10
11
12 app = Flask(__name__)
13 CORS(app)
14 sc = SparkContext()
15 rec_model = MatrixFactorizationModel.load(sc, "/root/Recommender/rc_model_2"
       )
16 rec_model.productFeatures().cache()
17 rec_model.userFeatures().cache()
18
19
20 def read_data():
21   conn = mysql.connector.connect(
22     host="localhost",
23     user="root",
24     passwd="Federica2018.",
25     database="tesi"
26   )
27   df = pd.read_sql("SELECT * FROM feedback",conn)
28   conn.close()
29   return df
30
31 df = read_data()
32
33
34 @app.route("/train", methods=["POST"])
35 def train():
36     if request.method == "POST":
37           secret = request.get_json()['secret']
38           if secret == "VnuEH73A8W30zrb75Gt87I8Y-
                 Jy2fRJU6rBwjx9wfJq9a670jjk3D9XeZ308dfwu7":
39               os.system("python recommendation_save.py")
40               global rec_model
41               rec_model = MatrixFactorizationModel.load(sc, "/root/
                     Recommender/rc_model_2")
42               rec_model.productFeatures().cache()
43               rec_model.userFeatures().cache()
```

```
44              global df
45              df = read_data()
46              return "ok"
47          else:
48              return "unauthorized"
49
50
51  @app.route("/recommendation", methods=["GET", "POST"])
52  def recommendation():
53
54      if request.method == "GET":
55          user_id = request.args.get("user_id")
56          item_ids = request.args.get("item_ids")
57          user_id = int(user_id)
58          item_list = item_ids.split(",")
59          result = []
60          app.logger.info("Input data: user_id = %s, item_ids = %s" % (user_id
                , item_ids))
61          try:
62              for item in item_list:
63                  item_res = {}
64                  item_id = int(item)
65                  item_res["item_id"] = item_id
66                  rating = rec_model.predict(user_id, item_id)
67                  item_res["pred_rating"] = rating
68                  result.append(item_res)
69          except Exception:
70              error_msg = "Error!"
71              return jsonify(error_msg)
72          app.logger.info("Output data: %s" % result)
73          return jsonify(result)
74
75      elif request.method == "POST":
76          data = request.get_json()
77          ratings = pd.Series([])
78          feedback = data["feedback"]
79          train_items = []
80          for i in range(len(feedback)):
81              ratings[feedback[i]["item_id"]] = feedback[i]["rating"]
82              train_items.append(feedback[i]["item_id"])
83
84          ratings.sort_index(inplace=True)
85          app.logger.info(np.array(ratings))
86          train_items = np.array(train_items)
87          item_ids = data["item_list"]
88          item_list = item_ids.split(",")
```

```
89          global df
90          train_data=df[df['PROD_ID'].isin(train_items)]
91          train_data=train_data[train_data['USER_ID']!=data['user_id']]
92          users=train_data['USER_ID'].unique()
93          user_item2 = train_data.pivot_table(values='RATING', index="USER_ID"
                , columns="PROD_ID", aggfunc="max")
94          user_item2.fillna(value=0, inplace=True)
95
96          knn_model = NearestNeighbors(n_neighbors=2,metric='euclidean')
97          knn_model.fit(user_item2)
98          nn = knn_model.kneighbors([ratings])
99          similar_users_index = nn[1][0]
100
101         similar_users = [users[similar_users_index[0]],users[
                similar_users_index[1]]]
102         app.logger.info('similar users %s' % similar_users)
103         result = []
104         for item in item_list:
105             item_res = {}
106             item_res["item_id"] = item
107             item_res["pred_rating"] = 0
108             for similar in similar_users:
109                 rating = rec_model.predict(similar, item)
110                 item_res["pred_rating"] = item_res["pred_rating"] + rating
111             item_res["pred_rating"] = item_res["pred_rating"] / 2
112             result.append(item_res)
113         app.logger.info("Input data: %s" % data)
114         app.logger.info("Output data: %s" % result)
115         return jsonify(result)
116
117 if __name__ == "__main__":
118     app.run(host="0.0.0.0",port = 5000, debug=True)
```

"Flask REST/API motore di raccomandazione"

5.4 Moduli per l'implementazione dei prodotti visivamente simili

In questa sezione viene riportato il codice per l'implementazione della ricerca di prodotti visivamente simili. Per ogni immagine del catalogo prodotti, in accordo con quanto descritto nella sezione 3.4.3, viene calcolata una tupla contenente le features estratte dalle reti, istogramma dell'immagine, colori dominanti ed infine nome del file. Se cambia il catalogo prodotti oppure si arricchisce il set di immagini, l'unica modifica richiesta è quella di ricalcolare il file contenente queste informazioni. IL seguente codice consente quindi lo sviluppo graduale del riconoscimento di prodotti visivamente simili per differenti categorie. Un classificatore iniziale identifica la categoria del prodotto ed un server orchestra il processo di predizione: sceglie quali reti e quale file di features caricare.

5.4.1 Analisi dei colori dominanti

```
1  from sklearn.cluster import KMeans
2  import numpy as np
3  from PIL import Image
4  from scipy.spatial.distance import euclidean
5  from scipy.spatial import distance as dist
6  import cv2
7  import os
8
9  def RGB2YUV(rgb):
10     m = np.array([[0.29900, -0.16874,  0.50000],
11                   [0.58700, -0.33126, -0.41869],
12                   [0.11400, 0.50000, -0.08131]])
13     yuv = np.dot(rgb,m)
14     yuv[:,:,1:] += 128.0
15     return yuv
16
17 def compute_distance(v,u,regularizer):
18     sum = 0
19     for i in range(len(v)):
20         min = euclidean(v[i],u[0])
21         for j in range(1,(len(u))):
22             tmp = euclidean(v[i],u[j])*(1+(np.abs(i-j)*regularizer))
23             if tmp < min:
24                 min = tmp
25         sum = sum + min
26
```

```
27      return sum/len(v)
28
29  def extract_colors(input_img,n_clusters,max_iter,ratio):
30      #img = Image.open(input_img).convert("RGB").resize((224,224))
31      img = input_img
32      image = img.crop((img.width/2-img.width*ratio, img.height/2 - img.height
            *ratio, img.width/2 + img.width*ratio, img.height/2 + img.height*
            ratio))
33      w,h = (image.width, image.height)
34      image = np.array(image)
35      image = RGB2YUV(image).reshape((w*h, 3))
36
37      clusters = KMeans(n_clusters=n_clusters, max_iter=max_iter, random_state
            =42).fit(image)
38
39      return clusters.cluster_centers_.astype("int")
40
41
42  def compute_single_histogram(pil_image):
43      image = np.asarray(pil_image)
44      hist = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8],[0, 256, 0, 256,
             0, 256])
45      hist = cv2.normalize(hist, hist).flatten()
46      return hist
47
48
49  def compute_directory_histogram(directory):
50      index = []
51      for filename in os.listdir(directory):
52          pil_image = Image.open(os.path.join(directory,filename)).resize
                ((224,224))
53          index.append({"filename":filename,"histogram":
                compute_single_histogram(pil_image)})
54      return index
55
56
57  def compute_distance_histogram(index,test_hist):
58    results = {}
59    for (k, hist) in index.items():
60        d = dist.cityblock(test_hist, hist)
61        results[k] = d
62    return results
```

"Analisi dei colori dominanti"

5.4.2 Fine-tuning della rete Resnet34 preaddestrata su ImageNet

In questa sezione viene riportata la strategia di addestramento discussa nella sezione 3.4.3, prodotti visivamente simili caso d'uso calzature.

```
1  import torch
2  import torchvision
3  from torchvision import models
4  import torchvision.transforms as transforms
5  from torchvision.transforms import ToPILImage
6  import torch.optim as optim
7  import torch.nn as nn
8  import torch.nn.functional as F
9  import matplotlib.pyplot as plt
10 import numpy as np
11 import timeit
12
13 def test_accuracy(net, dataloader,train):
14       correct = 0
15       total = 0
16       net.eval()
17       with torch.no_grad():
18           for data in dataloader:
19               images, labels = data
20               images = images.cuda()
21               labels = labels.cuda()
22               outputs = net(images)
23               _, predicted = torch.max(outputs.data, 1)
24               total += labels.size(0)
25               correct += (predicted == labels).sum().item()
26       accuracy = 100 * correct / total
27       if train==True:
28           print('Accuracy of the network on the train set: %d %%' % (
29       accuracy))
30       else:
31           print('Accuracy of the network on the test set: %d %%' % (
32       accuracy))
33       return accuracy
34
35 def train_net(net,trainloader,testloader,n_epochs,n_loss_print):
36         accuracy_epoch = []
37         loss_epoch = []
38         start = timeit.default_timer()
39         for epoch in range(n_epochs):
40             net.train()
41             running_loss = 0.0
```

```
42              for i, data in enumerate(trainloader, 0):
43                  inputs, labels = data
44                  inputs = inputs.cuda()
45                  labels = labels.cuda()
46
47                  optimizer.zero_grad()
48
49                  outputs = net(inputs)
50                  loss = criterion(outputs, labels)
51                  loss.backward()
52                  optimizer.step()
53                  running_loss += loss.item()
54
55                  if i % n_loss_print == (n_loss_print -1):
56                      print('\r[%d, %5d] lss: %.3f' %
57                            (epoch + 1, i + 1, running_loss / n_loss_print))
58                      loss_epoch.append(running_loss/n_loss_print)
59                      running_loss = 0.0
60                      accuracy = test_accuracy(net,testloader,train=False)
61                      accuracy_epoch.append(accuracy)
62
63                      if accuracy >= 90:
64                        torch.save({
65                          'model_state_dict': net.state_dict(),
66                          'optimizer_state_dict': optimizer.state_dict(),
67
68                      }, "./model_"+str(epoch))
69
70          print('Finished Training')
71          stop = timeit.default_timer()
72          print('Time: ', stop - start)
73          #show loss and accuracy
74          plt.plot(accuracy_epoch)
75          #plt.plot(train_accuracy_epoch)
76          plt.plot(loss_epoch)
77
78
79  transform_train = transforms.Compose(
80      [
81
82          transforms.Resize((224,224)),
83          transforms.RandomHorizontalFlip(),
84          transforms.RandomRotation((15,15)),
85          transforms.ToTensor(),
86          transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
```

```
87      ])
```

"Importazione librerie e definizione delle funzioni necessarie per laddestramento della rete

A seguire, definizione della rete e del dataset. L'ultimo layer è stato cambiato per fornire in output un numero di classi pari a cinque.

```
1  trainset = torchvision.datasets.ImageFolder(root='./training_set', transform
       =transform_train)
2  trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
3                                            shuffle=True, num_workers=4,
                                                drop_last=True)
4  testset = torchvision.datasets.ImageFolder(root='./test_set', transform=
       transform_test)
5  testloader = torch.utils.data.DataLoader(testset, batch_size=128,
6                                           shuffle=False, num_workers=4,
                                               drop_last=True)
7
8  net = models.resnet34(pretrained=True)
9  net.fc = nn.Linear(512, 5)
10 net = net.cuda()
```

"Caricamento dataset e rete Resnet34"

Per addestrare solo gli ultimi livelli della rete, i parametri dei livelli antecedenti sono congelati (requires_grad = False). La funzione di Loss utilizzata è la funione softmax ed è il criterio della rete. All'optimizer scelto, Adam, non vengono dati tutti i parametri della rete ma soltanto quelli che vogliamo addestrare: ovvero i parametri con requires_grad = True.

```
1  #Layers 1,2 e 3 per l'addestramento dei livelli presenti nel layer 4.
2  for child in net.layer1:
3    for params in child.parameters():
4      params.requires_grad=False;
5
6  for child in net.layer2:
7    for params in child.parameters():
8      params.requires_grad=False;
9
10 for child in net.layer3:
11   for params in child.parameters():
12     params.requires_grad=False;
13
14
15 criterion = nn.CrossEntropyLoss().cuda() #softmax
16 optimizer = optim.Adam(filter(lambda p: p.requires_grad, net.parameters()),
       lr=0.0001,weight_decay=0.01) #convergenza migliore rispetto a SGD
```

```
17 train_net(net,trainloader,testloader,n_epochs,len(trainloader))
```

"Fine-tuning Resnet34"

5.4.3 Predizione prodotti visivamente simili

In questa sezione vengono riportati i moduli per la creazione del motore di ricerca per prodotti visivamente simili. Data una immagine, il servizio ritorna l'identificativo dei primi N prodotti più simili. L'architettura utilizzata si basa sulla descrizione riportata nella sezione 3.4.3, prodotti visivamente simili.

```
1  def remove_duplicates(item_list):
2      unique_list = []
3      for item in item_list:
4          if item not in unique_list:
5              unique_list.append(item)
6
7      return np.array(unique_list)
8
9  def build_knn_scaler(dataframe,K):
10     vals = dataframe['features'].values
11     twoDarray = []
12     for val in vals:
13         twoDarray.append(val)
14     features = np.array(twoDarray)
15     scaler = StandardScaler()
16     scaler.fit(features)
17     knn = NearestNeighbors(n_neighbors=K,p=2,algorithm='brute').fit(scaler.
           transform(features))
18     return knn, scaler
```

"utilità"

Il modulo sottostante è incaricato a definire le funzioni necessarie per il caricamento delle reti e per la predizione. Il codice è predisposto per essere utilizzato su GPU con supporto Cuda o CPU.

```
1  from PIL import Image
2  import numpy as np
3  import torch
4  import torch.nn as nn
5  import torchvision.transforms as transforms
6  from torchvision import models
7  import time
8  from sklearn.preprocessing import normalize
9
10 # Custom import
11 import lib.utils as utils
12 import lib.colors_utils as colors_utils
13
14
15 test_transforms = transforms.Compose([
16     transforms.Resize((224, 224)),
17     transforms.ToTensor(),
18     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
19 ])
20
21
22 if torch.cuda.is_available():
23     print('[INF] -- RUNNING ON CUDA GPU')
24 else:
25     print('[WARN] -- RUNNING ON CPU MAY RESULT IN SLOWER PERFORMANCE')
26
27
28 def hook(module, input, output):
29     global extracted_features
30     extracted_features.append(output.data.cpu().numpy())
31
32
33 def hook2(module, input, output):
34     global extracted_features2
35     extracted_features2.append(output.data.cpu().numpy())
36
37
38 def load_networks(model_name_list):
39     networks = []
40     for i in range(0, len(model_name_list)):
41         if model_name_list[i] == "resnet50":
42             net = models.resnet50(pretrained=True)
43             net.avgpool.register_forward_hook(hook)
```

```
44              networks.append(net)
45          elif model_name_list[i] == "resnet152":
46              net = models.resnet152(pretrained=True)
47              net.avgpool.register_forward_hook(hook2)
48              networks.append(net)
49          elif model_name_list[i] == "resnet34_shoes":
50              check_point=torch.load('/root/search-by-image-engine/shoesModel2
                    ',map_location='cpu')
51              net = models.resnet34(pretrained=False)
52              net.fc = nn.Linear(512, 5)
53              net.load_state_dict(check_point["model_state_dict"])
54              net.avgpool.register_forward_hook(hook)
55              networks.append(net)
56
57      return networks
58
59
60  def prepare_image(pil_img):
61      pil_img = pil_img.convert('RGB')
62      img = test_transforms(pil_img)
63      img = img.view(1, 3, 224, 224)
64      return img
65
66  def extract_features(image, nets):
67
68      global extracted_features
69      global extracted_features2
70      extracted_features = []
71      extracted_features2 = []
72
73
74      if torch.cuda.is_available():
75
76          if len(nets) == 2:
77              nets[0].cuda()
78              nets[1].cuda()
79              nets[0].eval()
80              nets[1].eval()
81
82              with torch.no_grad():
83                  img = prepare_image(image)
84                  output = nets[0](img.cuda())
85                  output = nets[1](img.cuda())
86
87              extracted_features_array = np.array(extracted_features)
88              extracted_features_array = extracted_features_array.reshape((
```

```
                extracted_features_array.shape[0], nets[0].fc.in_features))
89
90              extracted_features2_array = np.array(extracted_features2)
91              extracted_features2_array = extracted_features2_array.reshape((
                    extracted_features2_array.shape[0], nets[1].fc.in_features))
92
93              features = (extracted_features_array + extracted_features2_array
                    )/2
94
95              return features
96
97          else:
98              nets[0].cuda()
99              nets[0].eval()
100
101             with torch.no_grad():
102                 img = prepare_image(image)
103                 output[0] = nets[0](img.cuda())
104
105             extracted_features_array = np.array(extracted_features)
106             extracted_features_array = extracted_features_array.reshape((
                    extracted_features_array.shape[0], nets[0].fc.in_features))
107
108             return extracted_features_array
109
110     else:
111
112         if len(nets) == 2:
113
114             nets[0].eval()
115             nets[1].eval()
116
117             with torch.no_grad():
118                 img = prepare_image(image)
119                 output = nets[0](img)
120                 output = nets[1](img)
121
122             extracted_features_array = np.array(extracted_features)
123             extracted_features_array = extracted_features_array.reshape((
                    extracted_features_array.shape[0], nets[0].fc.in_features))
124             extracted_features2_array = np.array(extracted_features2)
125             extracted_features2_array = extracted_features2_array.reshape((
                    extracted_features2_array.shape[0], nets[1].fc.in_features))
126
127             features = (extracted_features_array + extracted_features2_array
                    )/2
```

```
128             print(features)
129             return features
130 
131         else:
132 
133             nets[0].eval()
134 
135             with torch.no_grad():
136                 img = prepare_image(image)
137                 output = nets[0](img)
138 
139             extracted_features_array = np.array(extracted_features)
140             extracted_features_array = extracted_features_array.reshape((
                    extracted_features_array.shape[0], nets[0].fc.in_features))
141 
142             return extracted_features_array
143 
144 
145 def show_similar_products(pil_img, df, networks, knn, scaler):
146 
147     X = extract_features(pil_img, networks)
148     scaled_X = scaler.transform(X)
149     distances, indices = knn.kneighbors(scaled_X)
150     print(distances,indices)
151     ids = df["filename"]
152     candidates = []
153     #nome file nel formato idProdotto_tipologia_numeroFotografia
154     for i in range(len(indices[0])):
155       candidates_tok = ids[indices[0][i]].split('_')
156       c = candidates_tok[0]
157       candidates.append(c)
158 
159     return distances, indices, candidates
160 
161 
162 def picture_analysis(input_img, df, networks, color_weight, reg_param,
        histogram_weight, knn, scaler,n_clusters,max_iter,ratio):
163     ids = df["filename"]
164 
165     pil_img = Image.open(input_img)
166     distances, indexes, candidates = show_similar_products(pil_img, df,
            networks, knn, scaler)
167     distances = normalize(distances)
168 
169     compare_with = []
170     index = {}
```

```
171     for neighbor in indexes[0]:
172         compare_with.append(df[df["filename"]==ids[neighbor]]["yuv_colors"].
                values[0])
173         index[ids[neighbor]] = df[df['filename']==ids[neighbor]]['histogram'
                ].values[0]
174 
175     compare_with = np.asarray(compare_with)
176 
177     input_img_colors = colors_utils.extract_colors(pil_img,n_clusters,
            max_iter,ratio)
178 
179     color_distances = []
180 
181     for i in range(len(compare_with)):
182         color_distances.append(colors_utils.compute_distance(
                input_img_colors, compare_with[i], reg_param))
183 
184 
185     color_distances = normalize([color_distances])
186 
187 
188     test_histogram = colors_utils.compute_single_histogram(pil_img)
189     results = colors_utils.compute_distance_histogram(index,test_histogram)
190     results = normalize([list(results.values())])
191 
192     finals = {}
193     for i in range(len(candidates)):
194         d = distances[0][i] + color_weight * color_distances[0][i] +
                histogram_weight * results[0][i]
195         finals[d]=candidates[i]
196 
197     final_candidates = utils.remove_duplicates(finals.values())
198 
199     finals = sorted(finals.items())
200 
201     results = {}
202     for fc in final_candidates:
203         for dist, filename in finals:
204             if fc == filename:
205                 results[dist] = fc
206                 break
207 
208     results = sorted(results.items())
209 
210     results_list = []
211     for i in results:
```

```
212         dict_item = {}
213         dict_item["distance"] = i[0]
214         dict_item["partNumber"] = i[1]
215 
216         results_list.append(dict_item)
217 
218     return results_list
```

"Predizione prodotti visivamente simili"

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding
```

```
            =(1, 1), bias=False)
32        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
33        (relu): ReLU(inplace)
34        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
35        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
36        (downsample): Sequential(
37          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
38          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
39        )
40      )
41      (1): BasicBlock(
42        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
43        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
44        (relu): ReLU(inplace)
45        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
46        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
47      )
48      (2): BasicBlock(
49        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
50        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
51        (relu): ReLU(inplace)
52        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
53        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
54      )
55      (3): BasicBlock(
56        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
57        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
58        (relu): ReLU(inplace)
59        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
60        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
```

```
61      )
62    )
63    (layer3): Sequential(
64      (0): BasicBlock(
65        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding
              =(1, 1), bias=False)
66        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
67        (relu): ReLU(inplace)
68        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
69        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
70        (downsample): Sequential(
71          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
72          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
73        )
74      )
75      (1): BasicBlock(
76        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
77        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
78        (relu): ReLU(inplace)
79        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
80        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
81      )
82      (2): BasicBlock(
83        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
84        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
85        (relu): ReLU(inplace)
86        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
87        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
88      )
89      (3): BasicBlock(
90        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
91        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
```

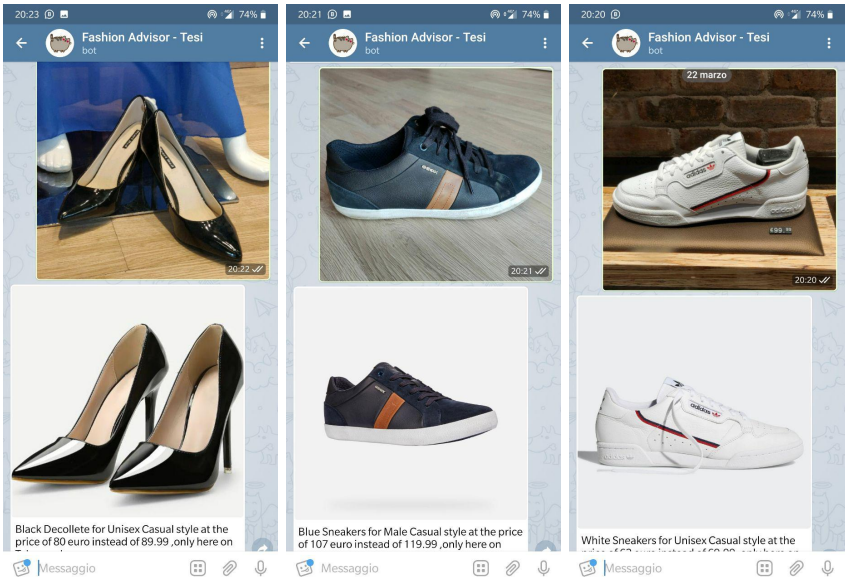
```
92        (relu): ReLU(inplace)
93        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
94        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
95      )
96      (4): BasicBlock(
97        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
98        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
99        (relu): ReLU(inplace)
100       (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
101       (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
102     )
103     (5): BasicBlock(
104       (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
105       (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
106       (relu): ReLU(inplace)
107       (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
108       (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
109     )
110   )
111   (layer4): Sequential(
112     (0): BasicBlock(
113       (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding
              =(1, 1), bias=False)
114       (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
115       (relu): ReLU(inplace)
116       (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
117       (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
118       (downsample): Sequential(
119         (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
120         (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
121       )
122     )
```

```
123     (1): BasicBlock(
124       (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
125       (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
126       (relu): ReLU(inplace)
127       (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
128       (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
129     )
130     (2): BasicBlock(
131       (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
132       (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
133       (relu): ReLU(inplace)
134       (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
135       (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
136     )
137   )
138   (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
139   (fc): Linear(in_features=512, out_features=5, bias=True)
140 )
```

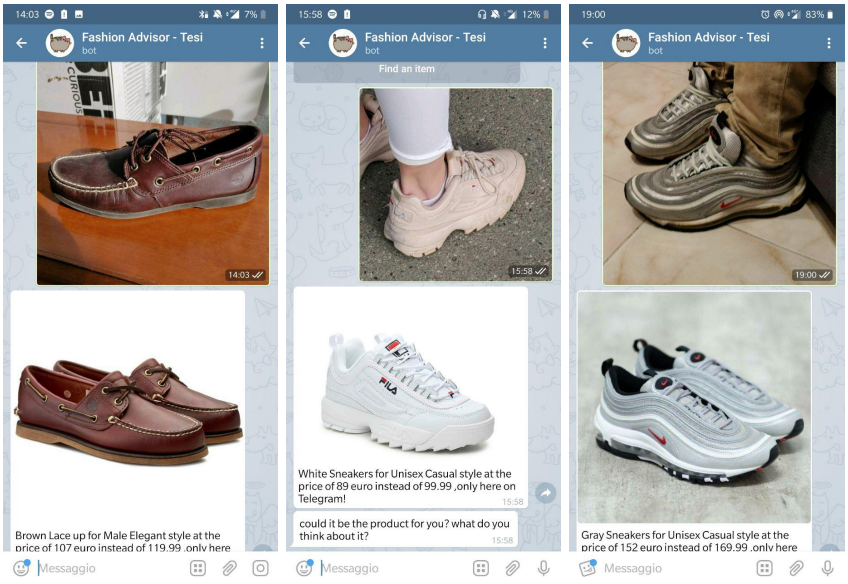
"Architettura Resnet34"

5.5 Prodotti visivamente simili: esempi

5.5.1 Scarpe: esempi con fotografie raccolte con il telefono cellulare

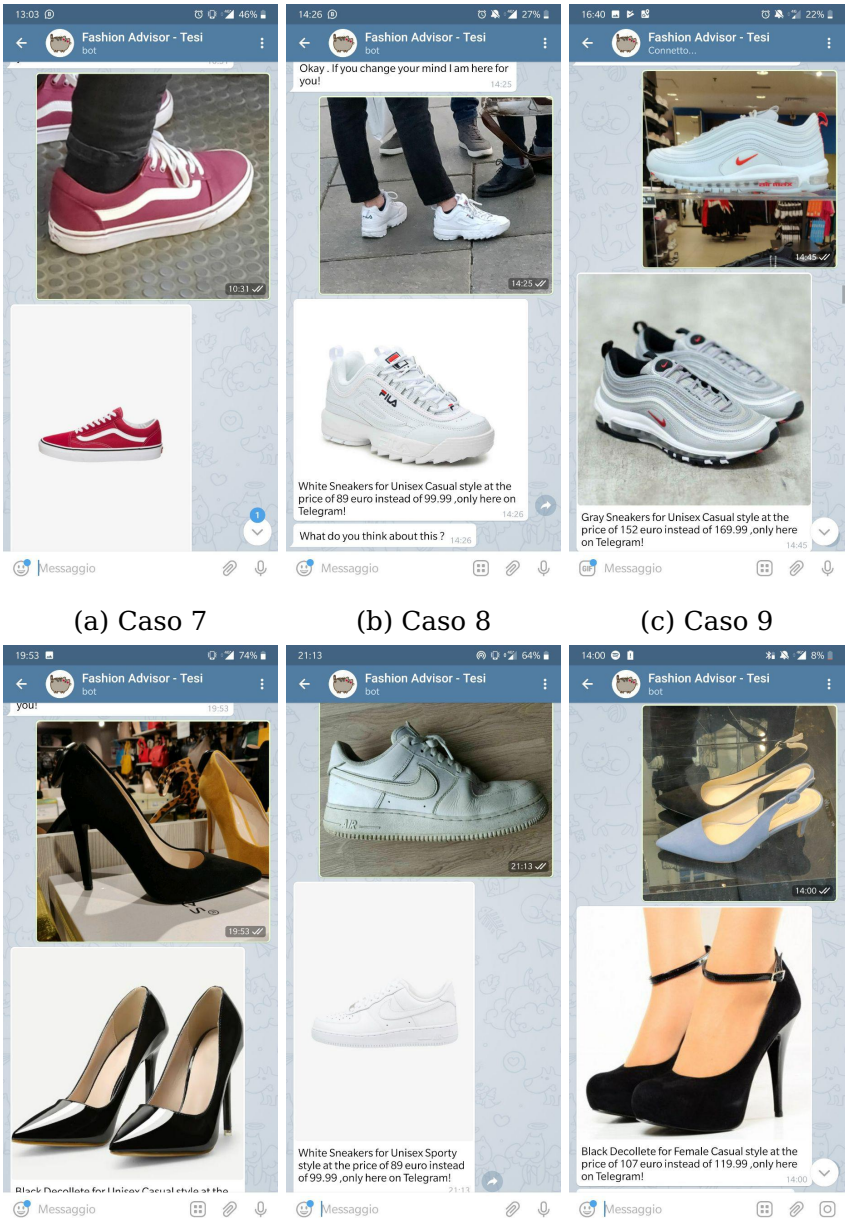


(a) Caso 1 (b) Caso 2 (c) Caso 3



(d) Caso 4 (e) Caso 5 (f) Caso 6

Figura 5.1: Scarpe: esempi di query - 1



(a) Caso 7 (b) Caso 8 (c) Caso 9

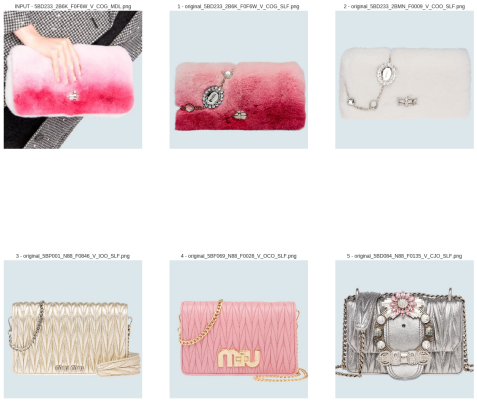
(d) Caso 10 (e) Caso 11 (f) Caso 12

Figura 5.2: Scarpe: esempi di query - 2

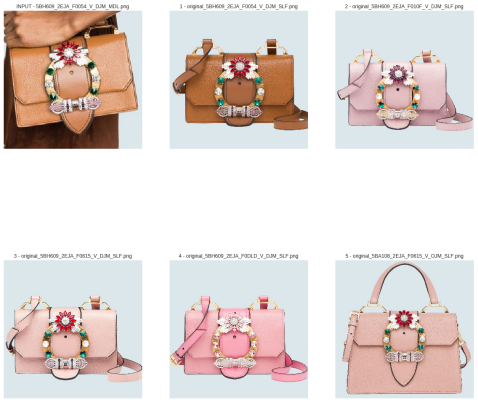
5.5.2 Borse: esempi estratti dal set di validazione

(a) Caso 1

(b) Caso 2

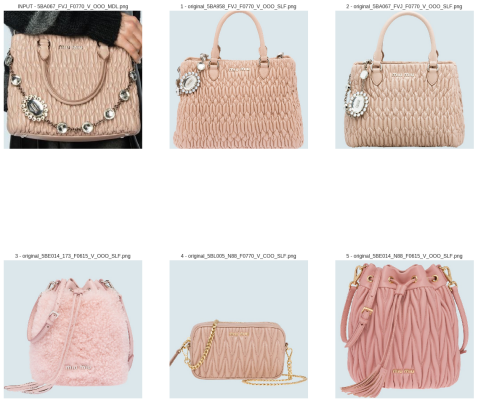


(c) Caso 3

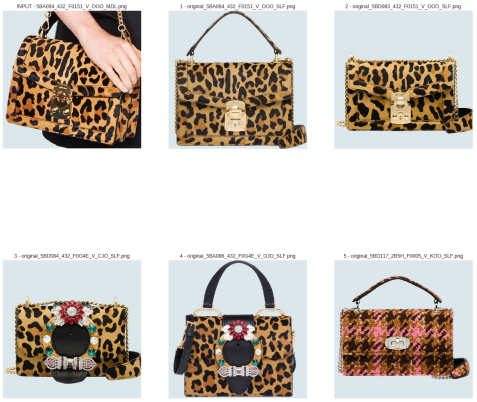


(d) Caso 4

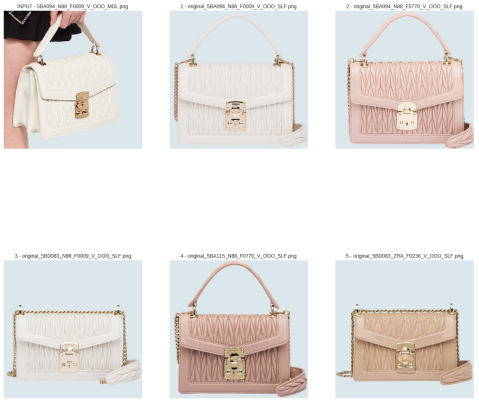
Figura 5.3: Casi d'uso test di validazione - 1



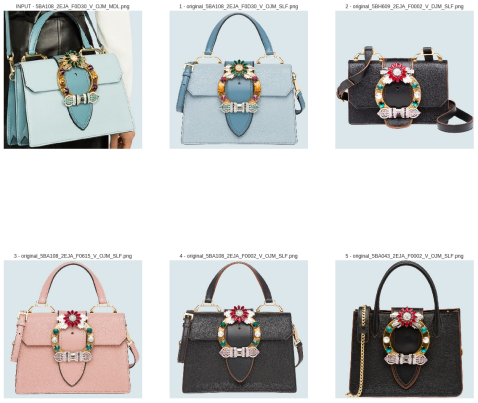
(a) Caso 5



(b) Caso 6



(c) Caso 7



(d) Caso 8

Figura 5.4: Casi d'uso test di validazione - 2

(a) Caso 9

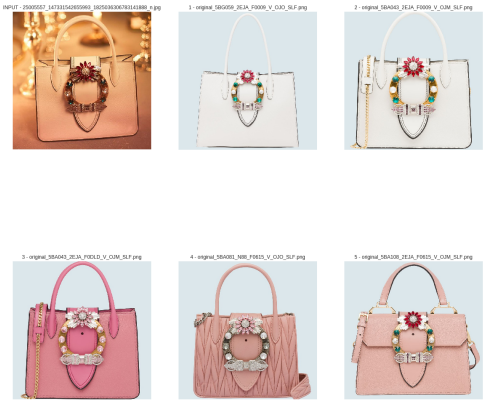
(b) Caso 10

(c) Caso 11

(d) Caso 12

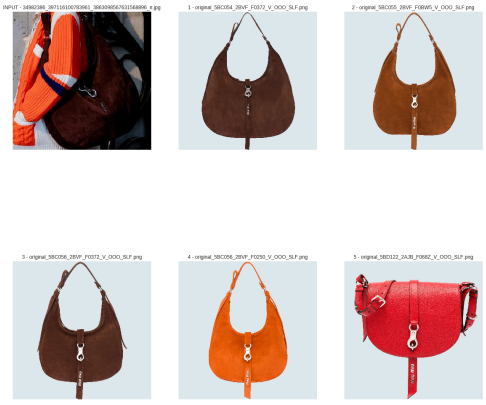
Figura 5.5: Casi d’uso test di validazione - 3

5.5.3 Borse: esempi estratti da fotografie di Instagram

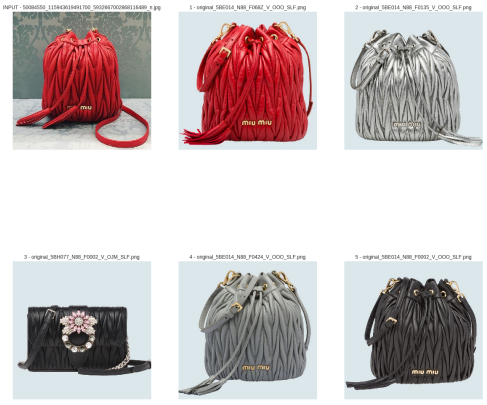


(a) Caso 1

(b) Caso 2



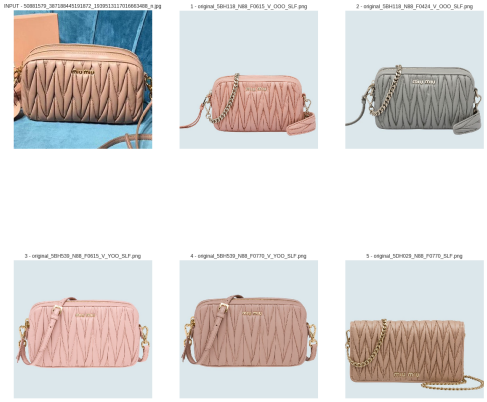
(c) Caso 3



(d) Caso 4

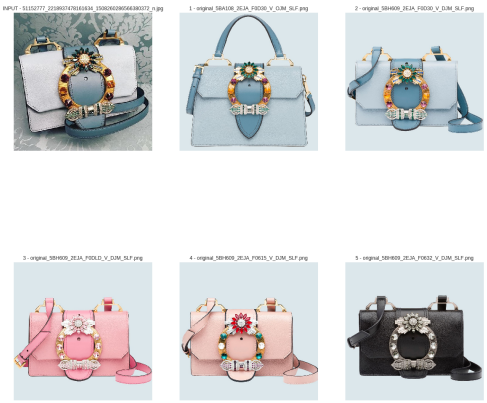
Figura 5.6: Casi d'uso Instagram - 1

(a) Caso 5



(b) Caso 6

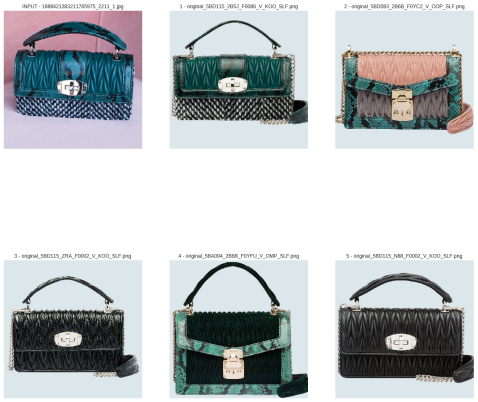
(c) Caso 7



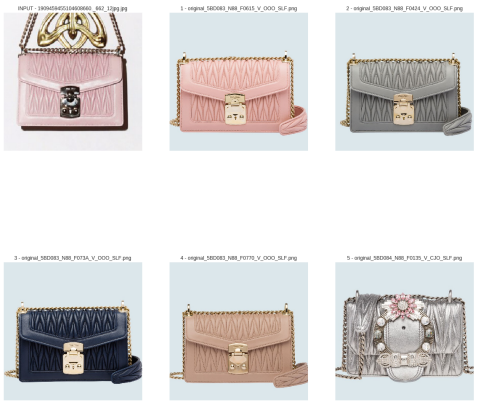
(d) Caso 8

Figura 5.7: Casi d’uso Instagram - 2

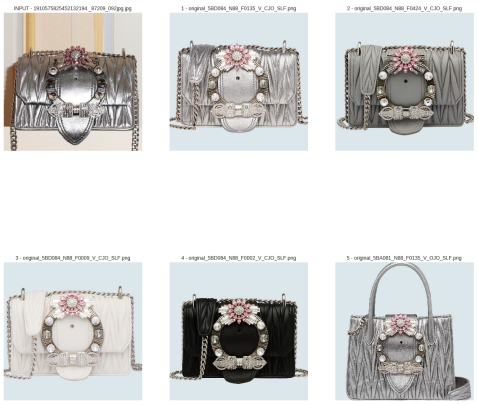
(a) Caso 9



(b) Caso 10



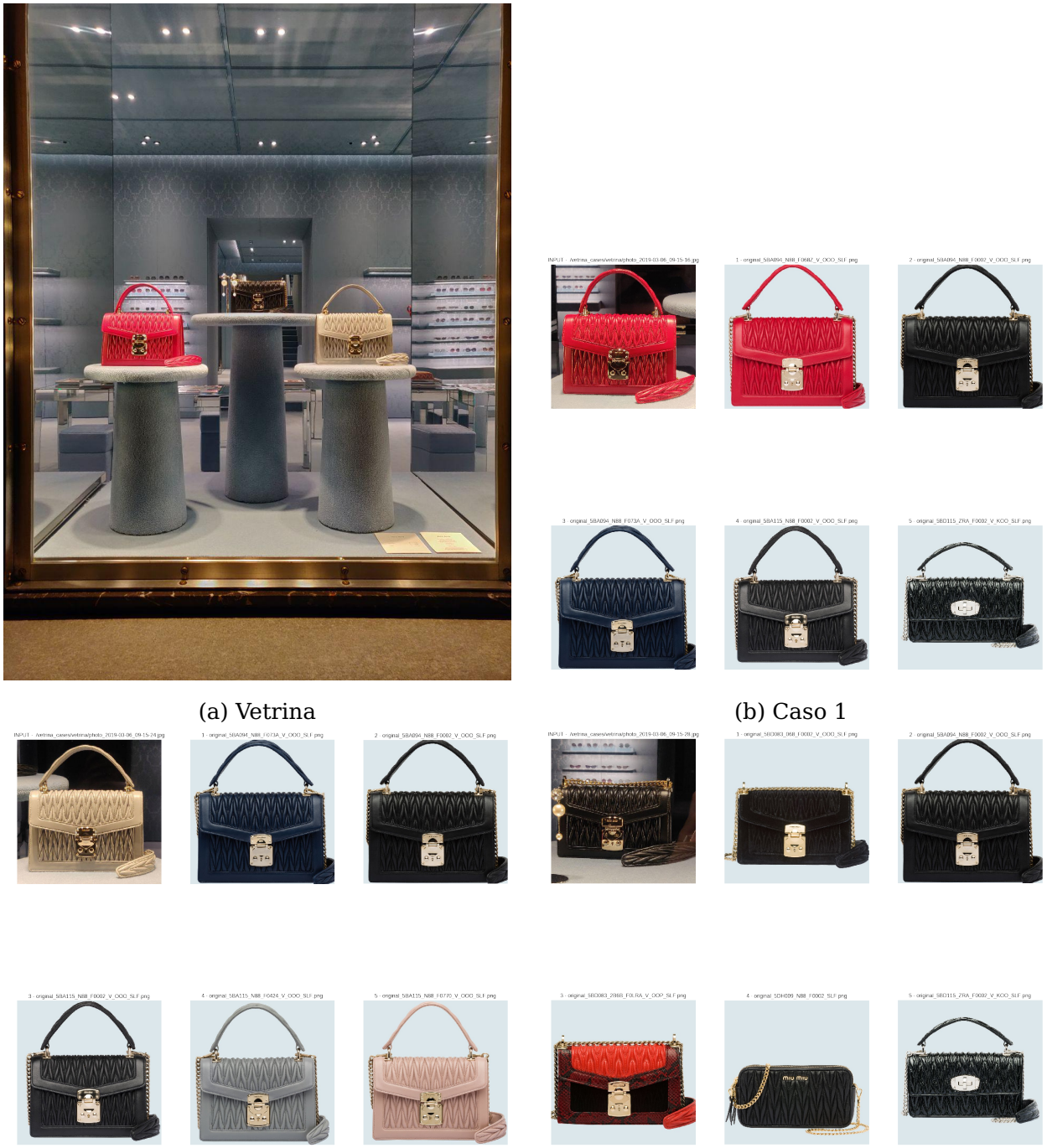
(c) Caso 11



(d) Caso 12

Figura 5.8: Casi d’uso test di validazione - 3

5.5.4 Borse:esempi estratti da vetrina



(a) Vetrina

(b) Caso 1

(c) Caso 2

(d) Caso 3

Figura 5.9: Casi d’uso da vetrina

Bibliografia

[1] Node-RED. (2018) Node-red: Documentation. [Online]. Available: https://nodered.org/docs/"

[2] IBM. (2018, Nov.) Ibm cloud watson assistant. [Online]. Available: https://console.bluemix.net/docs/services/assistant/index.html

[3] Google. (2019, Feb.) Google vision api reference. [Online]. Available: https://cloud.google.com/vision/docs/apis

[4] Telegram. (2018) Bots: An introduction for developers. [Online]. Available: https://core.telegram.org/bots

[5] Google. (2019, Feb.) Google actions. [Online]. Available: https://developers.google.com/actions/

[6] europa.eu. (2018) Multilinuismo - europa | unione europea. [Online]. Available: https://europa.eu/european-union/topics/multilingualism_it

[7] S. R. P. R. Zhou Y., Wilkinson D., "Large-scale parallel collaborative filtering for the netflix prize." 2008.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[10] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

Elenco delle figure

Elenco delle tabelle