Documentazione Progetto Ingegneria della Conoscenza

Componenti gruppo:

- Tria Stefano – 679571
- Scagliusi Francesco – 687845
- Lionetti Tommaso – 676669

Link Repository Github: https://github.com/stefanotria/ing_con

Idea:

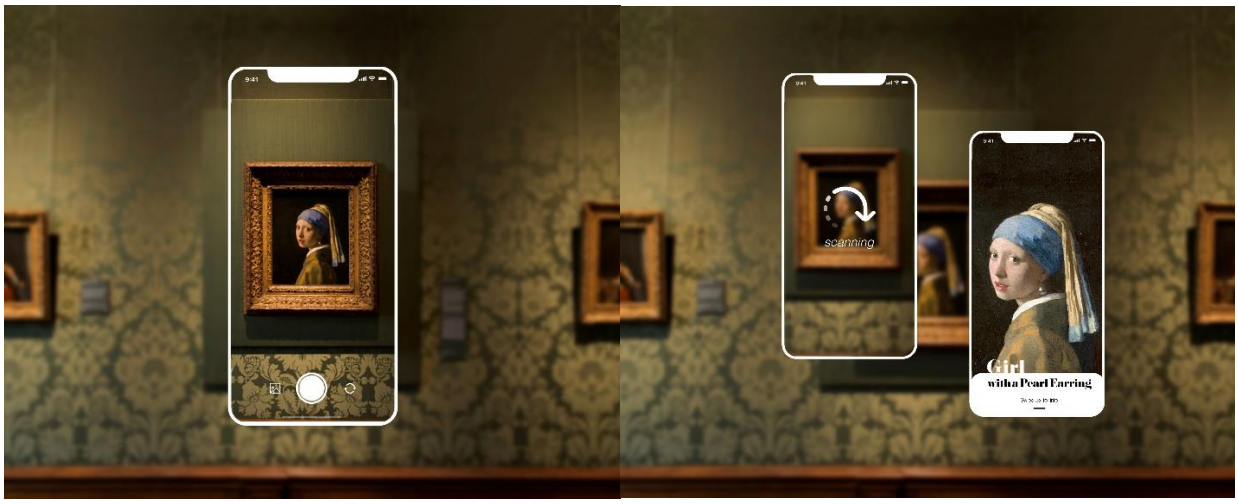
La nostra intenzione era di realizzare un sistema basato su conoscenza che a partire da un'immagine di un quadro, riconoscesse il quadro trattato e, mediante l'utilizzo dei Linked Open Data, restituisse delle informazioni sullo stesso.

Successivamente abbiamo deciso di implementare un sistema che, sulla base delle caratteristiche estrapolate dal quadro riconosciuto, fornisse all'utente consigli su possibili quadri ritenuti d'interesse. Abbiamo deciso di utilizzare Python come linguaggio per implementare il nostro progetto.

Manuale utente:

Una volta avviato il programma, verrà chiesto all'utente di selezionare un'immagine da riconoscere, che potrà scegliere tra quelle presenti nella cartella "Immagini di test" della repository. Il programma visualizzerà mediante un'interfaccia grafica le informazioni trovate e, mediante gli appositi button, l'utente potrà accedere ai quadri raccomandati.

Mockup:



Funzionamento:

Il programma è stato scomposto in alcune principali funzioni:

-Riconoscimento dei quadri

-Formulazione delle query

-Individuazione di possibili quadri d'interesse

Riconoscimento dei quadri:

Per il riconoscimento delle immagini abbiamo utilizzato TensorFlow, una Libreria software open source per l'apprendimento automatico, che fornisce moduli utili nella realizzazione di algoritmi per diversi tipi di compiti percettivi e di comprensione del linguaggio.

Come fase preliminare abbiamo dunque caricato il dataset contenente le immagini dei quadri suddivisi in train e test set mediante la classe dataset.py la quale, oltre al caricamento, si occupa di formattare le immagini in tensori a virgola mobile opportunamente pre-elaborati prima di allenare la rete neurale, ovvero:

- decodifica il contenuto delle immagini e lo converte nel formato a griglia appropriato secondo il loro contenuto RGB
- successivamente le converte in tensori a virgola mobile
- ridimensiona i tensori da valori compresi tra 0 e 255 a valori compresi tra 0 e 1

Un'immagine è memorizzata come una serie di matrici, ognuna avente una dimensione M x N rappresentante larghezza ed altezza; il numero di matrici necessarie a rappresentare una determinata immagine è definita come la profondità o depth della stessa e corrisponde al numero di canali colore che essa possiede. Per esempio, un'immagine RGB contiene 3 canali, uno per rosso, verde e blu, mentre se si ha un'immagine in scala di grigi si necessita di un solo canale per la rappresentazione del file.

Il contenuto delle matrici è invece un insieme di valori che rappresentano per ogni pixel la predominanza cromatica di quel particolare canale. All'interno di una rete neurale è preferibile mantenere l'intervallo di valori ristretto, quindi si applica un procedimento detto normalizzazione che consiste nel dividere ogni pixel dell'immagine per un valore, riducendone così il range. Per ottenere un intervallo di valori che vada da 0 ad 1 si può dividere il tensore per 255.

Tali operazioni vengono svolte mediante la classe ImageDataGenerator di TensorFlow utilizzata nel metodo generateTrainData(self) e generateTestData(self) che elaborano immagini rispettivamente del trainset e testset dove appunto, dopo aver definito i generatori per le immagini di addestramento, attraverso la chiamata al metodo flow_from_directory carica le immagini dal disco, applica la formattazione e ridimensionamento nelle dimensioni specificate.

CREAZIONE DEL MODELLO

Successivamente si passa alla costruzione del modello di apprendimento della rete neurale delegata alla classe recognition.py. Tale modello è composto da tre blocchi di convoluzione con uno strato massimo di pool in ciascuno di essi. Per attuare il processo di convoluzione si fa scorrere un certo filtro sul tensore mediante la chiamata a Conv2D() all'interno del metodo defineModel(). Un filtro è un insieme di pesi rappresentati sotto forma di un tensore di dimensione m x n e la cui profondità è la medesima dell'immagine di input. Durante questo processo, i valori del filtro vengono moltiplicati ai pixel values originali, ed infine sommati in un singolo valore scalare che rappresenta l'output di quel passo di convoluzione. Questi processi di convoluzione svolgono il ruolo di identificare una certa caratteristica di ciò che si sta osservando come linee, curve o angoli. Dunque gli strati preliminari tendono a codificare features molto semplici, mentre in profondità, partendo da quelle informazioni, la rete le ricompone per carpire informazioni più complesse come volti interi o oggetti. Dopo ogni layer convoluzionale viene effettuata un'operazione di Pooling. Ciò consiste nel creare una versione ridimensionata dell'output dello strato precedente in modo da ottenere una rappresentazione più compatta delle feature iniziali attraverso il metodo MaxPoling2D().

Alla fine di una rete convoluzionale è necessario ridurre la dimensionalità dell'output degli strati convoluzionali in modo da avere in uscita un vettore. Ciò si ottiene richiamando il metodo Flatten() che prende il tensore uscente da uno strato convoluzionale e lo ri-distribuisce come un vettore. Il risultato finale viene usato come input per una rete neurale 'classica' in cui vengono utilizzati strati completamente interconnessi di neuroni che creano la rappresentazione finale dell'input. Infine all'ultimo strato viene applicata una funzione "relu" di attivazione che appunto restituisce un tensore i cui valori sono ridimensionati in un range da 0 a 1.

COMPILAZIONE DEL MODELLO:

Prima di addestrare il modello, è necessario configurare il processo di apprendimento, che viene eseguito tramite la chiamata a compile() nel metodo compileModel(). Riceve tre argomenti:

- Un ottimizzatore "adam"
- Una funzione di perdita "categorical_crossentropy", rappresenta l'obiettivo che il modello proverà a minimizzare.
- Una metrica impostata su "accuracy", ovvero una funzione utilizzata per giudicare le prestazioni del modello

ADDESTRAMENTO DEL MODELLO:

L'addestramento è la fase in cui una rete neurale impara a svolgere il suo compito. In questa fase è necessario fornire degli esempi dai quali la rete deve imparare; questi sono coppie input/output e servono a far vedere alla rete come deve reagire di fronte a specifici dati in ingresso. La rete impara a partire da tali esempi e perciò si parla di apprendimento supervisionato. Più in dettaglio, una rete è un modello matematico il cui output è regolato da molti parametri (i pesi dei segnali ricevuti da ogni neurone). In fase di addestramento si vanno a tarare progressivamente questi

parametri in modo che sempre di più l'output della rete a fronte di un certo input si avvicini a quello desiderato.

Questo procedimento è svolto nel metodo fitModel() che addestra il modello per un numero fissato di epoche, ovvero di iterazioni su un set di dati. Dunque oltre a specificare il set da cui prendere le immagini per addestrare il modello, tiene conto anche del batch size che indica il numero di campioni per ogni epoca ed infine le informazioni sul criterio di miglioramento della predizione e dell'arresto dell'addestramento stesso mediante validation_data che specifica i dati su cui valutare la perdita e le eventuali metriche del modello alla fine di ogni epoca. È importante notare che il modello non verrà addestrato su questi dati, ma vengono utilizzati esclusivamente per calcolare una stima degli errori commessi e permettere al sistema di reiterare l'apprendimento laddove tali errori non sono al di sotto di una soglia stabilita. Mentre validation_steps si occupa di indicare il numero totale di lotti di campioni da convalidare prima di arrestarsi.

SALVARE IL MODELLO APPRESO:

Dato che i modelli di apprendimento possono richiedere molto tempo per essere generati ed addestrati, è possibile salvarli in modo da riutilizzarli in prossime predizioni, stando attenti però a prevedere il ricalcolo del modello stesso nel caso in cui il dataset delle immagini venga modificato o ampliato. Per questo motivo è stato implementato un metodo saveModel() e loadModel() per il salvataggio su disco e il caricamento del modello.

Il file che verrà generato per salvare il modello conterrà:

- l'architettura del modello

- i valori dei pesi del modello appresi durante l'allenamento

- la configurazione dell'allenamento del modello, dunque i parametri optimizer, metrics e loss usati nella compilazione del modello

FARE PREDIZIONI:

Sono stati implementati due metodi(predictImages() e predictImage(image)) per effettuare la predizione di immagini date in input; il primo si occupa di fare predizioni su un insieme di immagini del testset, utilizzate appunto per testare l'efficacia del sistema. Il secondo invece è quello che risponde alle esigenze del nostro programma, ovvero prevede un'immagine in input (che si assume essere il quadro che l'utente ha fotografato) sulla quale fare predizione per capire di quale quadro si tratti. Quindi utilizzando il modello creato o caricato dal disco, si richiama il metodo della classe keras predict() al quale viene passata l'immagine in input opportunamente pre-elaborata per renderla compatibile con il formato delle immagini utilizzate nell'allenamento del modello stesso. In base al valore restituito si cerca in un file csv la corrispettiva label contenente il nome del quadro riconosciuto che sarà fornito in output.

Formulazione delle query:

Una volta che il nostro programma ha riconosciuto con successo il quadro dall'immagine selezionata sul disco, vengono effettuate delle interrogazioni ad una base di conoscenza (KB) per ottenere le varie informazioni di cui abbiamo bisogno sul quadro stesso.

Per ottenere queste informazioni, abbiamo bisogno di una KB (contenente i dati strutturati) e un metodo per interrogarla. Come KB, ci serviamo sia di WikiData sia di DBpedia. Su WikiData otteniamo informazioni quali Autore, Data di realizzazione dell'opera, Museo, Movimento Artistico, Genere e Dimensioni; mentre su DBpedia otteniamo la descrizione del quadro.

Per effettuare le interrogazioni ci serviamo di SPARQL, cioè un linguaggio per interrogazioni di KB modellate in RDF attraverso endpoint. Infatti, le informazioni vengono organizzate in Linked Open Data, cioè dati strutturati atti ad essere collegati fra loro. I dati presenti nelle KB vengono codificati in RDF.

Tutto questo viene effettuato dalla classe Query presente nel file queryManager.py.

Per poter effettuare le varie interrogazioni, in Python, importiamo il pacchetto SPARQLWrapper, cioè il pacchetto che ci aiuta a creare le query in SPARQL, ad eseguirle e a convertire il risultato in un formato più gestibile (nel nostro caso, il risultato ci arriva sotto forma di JSON).

Per poter effettuare un'interrogazione, viene creato un oggetto di classe Query che al suo interno contiene i metodi per ottenere le informazioni necessarie. La query viene creata concatenando le diverse parti che richiedono i diversi campi. Infatti troviamo i metodi getAuthor, getDate, getMuseum, getMovement, getGenre, e getDimension per poter ottenere le informazioni da WikiData e il metodo getInfo per ottenere la descrizione da DBpedia.

Una volta creata la query, è il metodo runQuery che si occupa di eseguirla e da cui vengono estrapolati i valori dei campi richiesti. Un'importanza particolare riveste il metodo setQuery, che setta la query, setta il formato di ritorno del risultato e inoltre controlla che non ci siano stati errori (nelle nostre prove, infatti, abbiamo riscontrato più volte l'errore HTTP 429 cioè l'errore fornito da WikiData 'Too Many Requests'. Per poterlo risolvere abbiamo gestito l'eccezione sollevata dall'errore ed eseguito nuovamente la query per un massimo di 5 volte. Per gestire l'errore utilizziamo la libreria urllib di Python).

Individuazione possibili quadri d'interesse:

Per individuare i quadri che possano interessare all'ultente abbiamo deciso di prendere in considerazione le feature: Author, Museum, Genre e Movement.

Si è quindi deciso di confrontare le feature del quadro riconosciuto con le feature di un insieme di quadri presenti nella stessa nazione.

Come fase preliminare, all'interno della classe 'collection.py', mediante le funzioni della libreria pandas, vengono scritte in un file csv tutte le informazioni sui quadri presenti nella stessa nazione del quadro riconosciuto. Le informazioni vengono estrapolate grazie alle funzioni 'createCollection' e 'getContent' della classe 'queryManager.py'.

Successivamente all'interno della classe 'neighbors.py', dopo avere letto le informazioni presenti nel file 'collection.csv', dal metodo 'setWeights' vengono confrontate le feature del quadro riconosciuto con quelle dei quadri presenti nel file. Ove viene trovata una corrispondenza viene avvalorata la feature corrispondente all'interno del vettore che rappresenta il quadro.

Nel metodo 'getDistance', mediante la similarità del coseno, viene calcolata la vicinanza tra il vettore che rappresenta l'istanza del quadro riconosciuto e i vettori dei quadri con feature comuni. A questo score abbiamo sommato, per ottenere un altro fattore di confronto, la similarità calcolata tra i vettori tf-idf rappresentanti la descrizione testuale del contenuto di ogni quadro.

Questa funzione viene implementata nella classe 'tfidf.py' grazie all'utilizzo della libreria 'sklearn' che fornisce un vettorizzatore tf-idf preconfigurato che calcola il punteggio tf-idf per la descrizione del contenuto di ciascun quadro, parola per parola, con il metodo 'TfidfVectorizer'. Gli score di ogni parola, in relazione a ciascun documento o quadro nel nostro caso, vengono salvati nella matrice 'tfidf_matrix'.

Anche in questo caso abbiamo calcolato la similarità del coseno di ogni vettore con ogni altro elemento presente nella 'tfidf_matrix', organizzando i risultati in una struttura 'results' in base allo score ottenuto e utilizzando l'uri del quadro come identficativo.

A questo punto gli score e i loro identificativi vengono passati al metodo 'getNeighbors' della classe 'neighbors.py', che sommando sia i punteggi tf-idf che quelli ottenuti dal metodo 'getDistance' ritorna i 'k' quadri più simili.

Nell'interfaccia grafica vengono inoltre stampate le feature comuni con i quadri raccomandati in modo da giustificarne la somiglianza.