



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Cognitive Computing System applicato allo studio del Glioblastoma

Docenti:

Prof. Paolo Maresca

Autori:

Francesca Lettieri - M63000956

Angelo Russo - M63001016

Guido Maria Secondulfo - M63000927

Indice

1	Introduzione	2
1.1	Traccia	2
2	Architettura	2
2.1	Tecnologie utilizzate	2
2.1.1	E-utilities	2
2.1.2	IBM Cloud Db2	2
2.1.3	GitHub	2
2.1.4	IBM Cloud Foundry	2
2.1.5	IBM Watson Machine Learning	3
2.1.6	IBM Watson Knowledge Studio	3
2.1.7	IBM Watson Discovery	3
2.1.8	fastai	3
2.1.9	IBM Cloud API	3
2.1.10	Jupyter	4
3	Implementazione	4
3.1	Flusso di esecuzione del sistema globale	4
4	Apprendimento sugli articoli medici tramite NLP	5
4.1	Flusso di esecuzione del sistema di retrieval degli articoli	5
4.1.1	E-utilities	5
4.1.2	Query di ricerca	6
4.1.3	GitHub e IBM Cloud Foundry	8
4.1.4	IBM Cloud Db2	8
4.1.5	Funzioni implementate	13
4.2	Flusso di esecuzione del sistema di apprendimento sugli articoli	15
4.3	IBM Watson Knowledge Studio	15
4.3.1	Import degli Articoli	15
4.3.2	Creazione del Modello Entità-Relazione	15
5	Apprendimento sulle Risonanze Magnetiche tramite Visual Recognition	22
5.1	Flusso di esecuzione del sistema di apprendimento sulle MRI	22
5.2	CNN con la libreria <i>fastai</i>	22
5.2.1	Implementazione	22
5.2.2	Risultati ottenuti	24
5.3	Visual Recognition di Watson Studio	27
5.3.1	Implementazione	27
5.3.2	Risultati ottenuti - Primo Ciclo	30
5.3.3	Risultati ottenuti - Secondo Ciclo	32
5.3.4	Risultati ottenuti - Terzo Ciclo	33
6	Riferimenti	35

1 Introduzione

1.1 Traccia

Il progetto da noi sviluppato si articola in due percorsi distinti:

1. Retrieval da PubMed di articoli inerenti al Glioblastoma Multiforme (GBM), creazione di un database atto a contenere le informazioni rilevanti ed, infine, creazione di un modello di NLP (Natural Language Processing) al fine di realizzare un sistema di apprendimento in grado di realizzare una standard meta-analysis accurata;
2. Acquisizione delle risonanze magnetiche (MRI) di pazienti affetti da GBM e costruzione di un modello di apprendimento su di esse.

2 Architettura

2.1 Tecnologie utilizzate

2.1.1 E-utilities

Le E-utilities (Entrez Programming Utilities) costituiscono un insieme di 8 programmi server-side che forniscono un'interfaccia stabile per il sistema di database e query Entrez. Le E-utilities utilizzano una sintassi a URL fisso che traduce un insieme di parametri di input nei valori necessari ai componenti software del NCBI per ricercare e recuperare i dati richiesti.

2.1.2 IBM Cloud Db2

IBM Db2 on Cloud è un database cloud transazionale progettato per offrire prestazioni elevate. Esso assicura la scalabilità orizzontale e verticale per soddisfare le richieste di business.

2.1.3 GitHub

Git è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando, sviluppato per gestire anche progetti di elevate dimensioni con elevate velocità ed efficienza.

2.1.4 IBM Cloud Foundry

Cloud Foundry è una PaaS (platform-as-a-service) open source su IBM Cloud che consente di implementare e scalare le app senza gestire i server. È stato da noi utilizzato al fine di poter eseguire uno script in linguaggio Python all'interno dell'ambiente IBM Cloud.

2.1.5 IBM Watson Machine Learning

IBM Watson Machine Learning aiuta i data scientist e gli sviluppatori ad accelerare l'implementazione di AI e machine-learning. Con i suoi modelli estensibili, aperti, Watson Machine Learning aiuta le aziende a semplificare e sfruttare l'AI in scala su qualsiasi cloud.

2.1.6 IBM Watson Knowledge Studio

IBM Watson consente l'apprendimento del linguaggio del dominio con modelli personalizzati che identificano le entità e le relazioni specifiche del settore in testo non strutturato. I modelli possono essere creati in un ambiente collaborativo progettato sia per gli sviluppatori che per gli esperti di dominio, senza il bisogno di scrivere codice, e possono essere utilizzati in IBM Watson Discovery, IBM Watson Natural Language Understanding e IBM Watson Explorer.

2.1.7 IBM Watson Discovery

Watson Discovery è una tecnologia pluripremiata per la ricerca aziendale, basata sull'AI, in grado di abbattere i silos di dati e ottenere risposte specifiche alle tue domande, mentre analizza le tendenze e le relazioni nascoste nei dati aziendali. Watson Discovery applica le più recenti innovazioni nel machine learning, comprese le funzionalità di elaborazione del linguaggio naturale, ed è semplice da addestrare sul linguaggio del tuo dominio. Watson Discovery può essere implementato su qualsiasi ambiente cloud o on-premise.

2.1.8 fastai

fastai è una libreria di deep learning che fornisce agli operatori componenti di alto livello in grado di fornire rapidamente e facilmente risultati di ultima generazione in domini di deep learning standard, e fornisce ai ricercatori con componenti di basso livello che possono essere mescolati e abbinati per costruire nuovi approcci. fastai mira a fare entrambe le cose senza compromessi sostanziali in facilità d'uso, flessibilità, o prestazioni. Questo è possibile grazie ad un'architettura a strati, che esprime schemi comuni di base di molte tecniche di deep learning e di elaborazione dati in termini di astrazioni disaccoppiate. Queste astrazioni possono essere espresse in modo conciso e chiaro sfruttando il dinamismo del linguaggio Python sottostante e la flessibilità della libreria Pytorch.

2.1.9 IBM Cloud API

Le IBM Cloud API sono delle API che consentono di garantire l'accesso ai servizi IBM Cloud ai propri servizi ed alle proprie applicazioni che lo necessitano.

2.1.10 Jupyter

Il Progetto Jupyter è un'organizzazione creata per sviluppare software open-source, e supportare ambienti di esecuzione in decine di lingue. JupyterLab offre un'interfaccia per la gestione di documenti di diverso formato, per attività di text editing e attività da terminale.

3 Implementazione

3.1 Flusso di esecuzione del sistema globale

Il diagramma di flusso rappresentante l'elaborazione effettuata è il seguente:

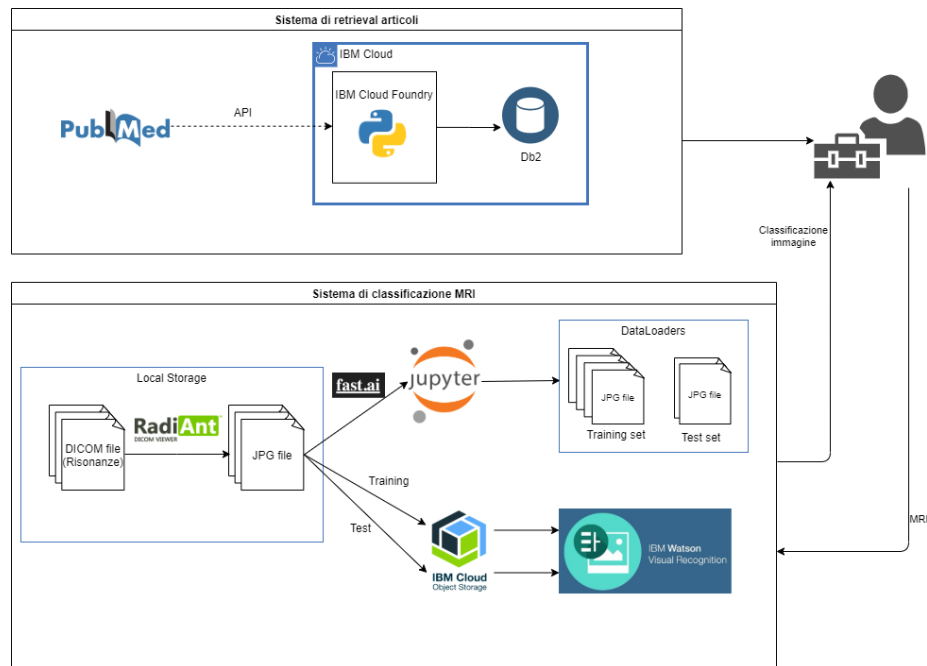


Figura 1: Diagramma di flusso dell'esecuzione

Al fine di agevolare la trattazione, affronteremo i problemi separatamente.

4 Apprendimento sugli articoli medici tramite NLP

4.1 Flusso di esecuzione del sistema di retrieval degli articoli

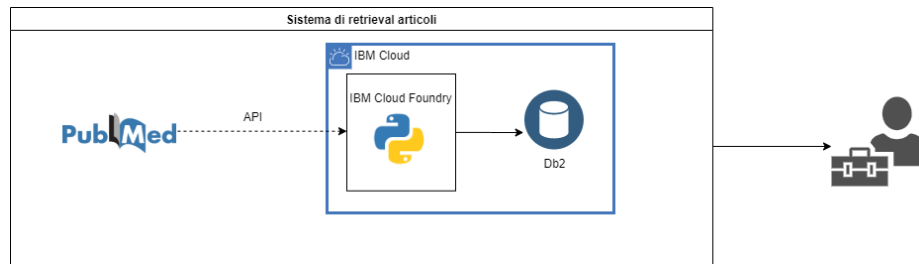


Figura 2: Diagramma di flusso dell'esecuzione

Innanzitutto è stato effettuato il download degli articoli desiderati da PubMed. Dopo una prima elaborazione di tali documenti, essi sono stati memorizzati in un database relazionale all'interno dell'ambiente IBM Cloud.

4.1.1 E-utilities

In seguito ad un confronto con gli esperti di dominio, abbiamo ritenuto opportuno effettuare un retrieval di articoli provenienti unicamente da PubMed, in quanto source affidabile e attendibile.

A tal fine è stato sviluppato uno script in linguaggio Python (*pubmed.py*) che, sfruttando le E-utilities, è in grado di effettuare il retrieval degli articoli desiderati da PubMed stesso.

In particolare, la funzione *getPubMedIDArticles* recupera gli ID degli articoli desiderati da PubMed, in base ai termini specificati. L'implementazione di tale funzione è la seguente:

```
def getPubMedIDArticles(query):
    test = "esearch.fcgi?db=pubmed"
    term = "&term="+query #query della ricerca
    maxArts = "&retmax=100000" #Numero massimo di articoli
    recuperati dalla ricerca
    link = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/
    /%22+test+term+maxArts

    s = urllib.request.urlopen(link)
    contents = s.read()
    file = open("export_idArticles.xml", 'wb')
    file.write(contents)
    file.close()
```

4.1.2 Query di ricerca

La query effettuata è stata elaborata dagli esperti di dominio. Essa espone quali siano i termini da ricercare all'interno di tutti i campi di un articolo, inclusi i MeSH Terms ¹ associati all'articolo stesso.

I criteri di inclusione per la selezione degli articoli sono i seguenti:

- Studi randomizzati di confronto fra trattamenti;
- Fase II o III (compresi studi Fase I/II purchè contengano risultati adeguati della Fase II);
- OS sopravvivenza globale (overall survival) e PFS sopravvivenza libera da progressione (progression free survival) e ORR (Overall Response Rate) Tasso di risposta complessiva;
- Solo Glioblastoma multiforme (GBM) o High Grade Glioma in ripresa dopo pregressa chirurgia e terapia adiuvante o in progressione.

I criteri di esclusione per la selezione degli articoli sono, invece, i seguenti:

- Tumori pediatrici;
- Studi non randomizzati;
- Studi a singolo braccio;
- Studi di fase I;
- GBM in trattamento adiuvante;
- Review/metanalisi;
- Studi preclinici;
- Studi di farmacoeconomia;
- Studi di confronto senza analisi statistiche.

La formulazione risultante della query è, dunque, la seguente:

¹Medical Subject Headings (acronimo: MeSH) è un enorme vocabolario controllato (o sistema di metadati) ideato con l'obiettivo di indicizzare la letteratura scientifica in ambito biomedico. Il tesoro è stato creato dalla National Library of Medicine (NLM) degli Stati Uniti, che è responsabile anche della sua gestione. Il MeSH viene adoperato per l'indicizzazione degli articoli delle oltre 5000 riviste mediche presenti nel database bibliografico Medline/PubMed e nel catalogo dei libri della NLM. Il vocabolario può essere consultato e scaricato gratuitamente da tutti gli utenti di internet. La stampa dell'edizione cartacea è stata abbandonata nel 2007.

(glioblastoma OR glioblastoma multiforme OR anaplastic astocytoma OR malignant glioma OR high grade glioma OR high grade astrocytoma) AND (diffuse OR multiple OR unresectable OR advanced OR recurrent OR inoperable OR disseminated OR progressive OR relapsed) AND (clinical outcome OR OS OR PFS OR ORR) AND (chemotherapy OR target Therapy OR immunotherapy OR cyclophosphamide OR bevacizumab OR temozolomide OR irinotecan OR carmustine OR lomustine OR nitrosourea OR BCNU OR CCNU OR carboplatin OR cisplatin OR platinum OR PCV OR etoposide OR VP-16 OR cilengitide OR alternating electric field therapy OR TTF OR tumor treating fields OR procarbazine OR vincristine OR sunitinib OR lapatinib OR pazopanib OR temsirolimus OR everolimus OR fotemustine OR sagopilone OR Bortezomib OR Vorinostat OR enzastaurin OR Vandetanib OR veliparib OR dasatinib OR ultrafractionated RT OR nivolumab OR Regorafenib OR pembrolizumab OR ipilimumab OR radiosurgery OR ypo-fractionated radiation OR anlotinib OR Cesium-131 brachytherapy OR gefitinib OR sorafenib OR dacomitinib OR pidilizumab OR ivosidenib OR Interstitial brachytherapy OR erlotinib OR depatuxizumab mafodotin OR durvalumab OR onartuzumab OR rituximab OR Disulfiram OR reirradiation OR capmatinib OR buparlisib OR drug delivery technology OR convection-enhanced delivery OR bulk flow OR intra-arterial chemotherapy OR nanotechnology-based controlled delivery OR intraoperative polymer implants OR dagnetic cationic microsphere OR biodegradable polymer wafers OR lipid-coated microbubbles OR magnetic hyperthermia OR photodynamic therapy)

Figura 3: Query originale

Tale query è stata, quindi, da noi raffinata al fine di poter effettuare il retrieval degli articoli che soddisfacessero i criteri richiesti.

L'operazione di raffinamento è stata dettata dalla risposta delle API di PubMed a determinati termini (es. "ypo-fractioned", "dagnetic") che facevano parte della query iniziale; tali termini, omessi in fase di raffinamento, non risultavano presenti all'interno del thesaurus (i.e. il MeSH).

(glioblastoma+OR+glioblastoma+multiforme+OR+anaplastic+astocytoma+OR+malignant+glioma+OR+high+grade+glioma+OR+high+grade+astrocytoma)+AND+(diffuse+OR+multiple+OR+unresectable+OR+advanced+OR+recurrent+OR+inoperable+OR+disseminated+OR+progressive+OR+relapsed)+AND+(clinical+outcome+OR+OS+OR+PFS+OR+ORR)+AND+(chemotherapy+OR+target+Therapy+OR+immunotherapy+OR+cyclophosphamide+OR+bevacizumab+OR+temozolomide+OR+irinotecan+OR+carmustine+OR+lomustine+OR+nitrosourea+OR+BCNU+OR+CCNU+OR+carboplatin+OR+cisplatin+OR+platinum+OR+PCV+OR+etoposide+OR+VP-16+OR+cilengitide+OR+alternating+electric+field+therapy+OR+TTF+OR+tumor+treating+fields+OR+procarbazine+OR+vincristine+OR+sunitinib+OR+lapatinib+OR+pazopanib+OR+temsirolimus+OR+everolimus+OR+fotemustine+OR+sagopilone+OR+Bortezomib+OR+Vorinostat+OR+enzastaurin+OR+Vandetanib+OR+veliparib+OR+dasatinib+OR+ultrafractionated+RT+OR+nivolumab+OR+Regorafenib+OR+pembrolizumab+OR+ipilimumab+OR+radiosurgery+OR+ypo-fractionated+radiation+OR+anlotinib+OR+Cesium-131+brachytherapy+OR+gefitinib+OR+sorafenib+OR+dacomitinib+OR+pidilizumab+OR+ivosidenib+OR+Interstitial+brachytherapy+OR+erlotinib+OR+depatuxizumab+mafodotin+OR+durvalumab+OR+onartuzumab+OR+rituximab+OR+Disulfiram+OR+reirradiation+OR+capmatinib+OR+buparlisib+OR+drug+delivery+tecnology+OR+convection-enhanced+delivery+OR++bulk+flow+OR+intra-arterial+chemotherapy+OR+nanotechnology-based+controlled+delivery+OR+intraoperative+polymer+implants+OR+dagnetic+cationic+microsphere++OR+biodegradable+polymer+wafers+OR+lipid-coated+microbubbles+OR+magnetic+hyperthermia+OR+photodynamic+therapy)

Figura 4: Query Raffinata

In entrambi i casi, il numero di articoli recuperati è pari a 1'318, per questo l'eliminazione dei termini è risultata ininfluente per quanto riguarda il risultato finale ottenuto. Un ulteriore filtraggio degli articoli recuperati è stato fatto in base alla data di pubblicazione, secondo quanto richiesto e consigliato dagli esperti di dominio: escludendo gli articoli pubblicati prima del 1990, il numero finale di articoli recuperati è pari a 1'313.

4.1.3 GitHub e IBM Cloud Foundry

Per poter importare gli articoli in ambiente IBM Cloud, innanzitutto è stato creato un repository su GitHub all'interno del quale è stato memorizzato il file *pubmed.py*.

In particolare, sono state effettuate le operazioni di **add**, **commit** e **push** del file all'interno del repository attraverso il prompt dei comandi.

4.1.4 IBM Cloud Db2

Innanzitutto è stata istanziata una versione lite del Database Db2. Utilizzando le credenziali di accesso fornite dal sistema e le API IBM_DB, è stato possibile accedere al Database attraverso una funzione all'interno del file *pubmed.py*.

Per la modellazione del database, volendo tener traccia di un'unica entità, gli articoli, è stato realizzato il seguente diagramma ER²:

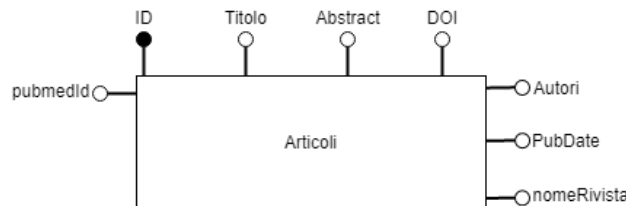


Figura 5: Diagramma ER

Abbiamo ipotizzato di memorizzare, per ogni articolo:

- DOI (Digital Object Identifier) - quando disponibile;
- ID dell'articolo nel database PubMed;
- Titolo;

²In informatica, nell'ambito della progettazione dei database, il modello entity-relationship (detto anche modello E-R; in italiano: modello entità-associazione o modello entità-relazione) è un modello teorico per la rappresentazione concettuale e grafica dei dati a un alto livello di astrazione, formalizzato dal prof. Peter Chen nel 1976.

Il modello entità-associazione viene spesso utilizzato nella prima fase della progettazione di una base di dati, nella quale è necessario tradurre le informazioni risultanti dall'analisi di un determinato dominio in uno schema concettuale, chiamato schema E-R (schema entità-associazione) o diagramma E-R (diagramma entità-associazione).

- Lista di autori - quando disponibile;
- Data di pubblicazione - quando disponibile;
- Rivista di pubblicazione - quando disponibile;
- Abstract - quando disponibile.

Osserviamo che, durante l'attività di recupero dalle API Pubmed, il DOI era assente in alcuni articoli, il che lo rendeva inadeguato al ruolo di chiave primaria dell'entità. Per questo motivo, abbiamo utilizzato un ulteriore identificativo numerico con auto-incremento come chiave primaria.

Quindi, è stato possibile creare una tabella ARTICOLI attraverso la seguente query:

```
CREATE TABLE articoli(
  id INTEGER NOT NULL PRIMARY KEY GENERATED ALWAYS AS
    IDENTITY (START WITH 1 INCREMENT BY 1) ,
  pubmedId INTEGER UNIQUE NOT NULL,
  doi VARCHAR(80) ,
  titolo VARCHAR(1000) ,
  autori VARCHAR(5000) ,
  dataPubblicazione DATE,
  abstract VARCHAR(10000) ,
  nomeRivista VARCHAR (1000)"
)
```

La connessione al database è realizzata attraverso la funzione *connectionDB2*, mentre tale connessione viene chiusa attraverso la funzione *endConnectionDB2*. L'implementazione di entrambe le funzioni è riportata di seguito:

```
def connectionDB2():
    conn_info = ""DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox
      -yp-lon02-13.
      services.eu-gb.bluemix.net;PORT=50000;PROTOCOL=TCPIP;
      UID=xxd26106;PWD=s5bzk6p^3s94hght;""
    conn = ibm_db.connect(conn_info, "", "")
    return conn

def endConnectionDB2(conn):
    ibm_db.close(conn)
```

Si noti che le informazioni relative al Database, quali nome dello stesso, l'Host-name, il porto, il protocollo, l'username e la password, sono strettamente legate all'account IBM Cloud che ha istanziato il database. In caso di migrazione del database, o di creazione di uno nuovo, è necessario modificare l'intera variabile *conn_info* con le rispettive Credenziali di Servizio generate sul portale IBM Cloud.

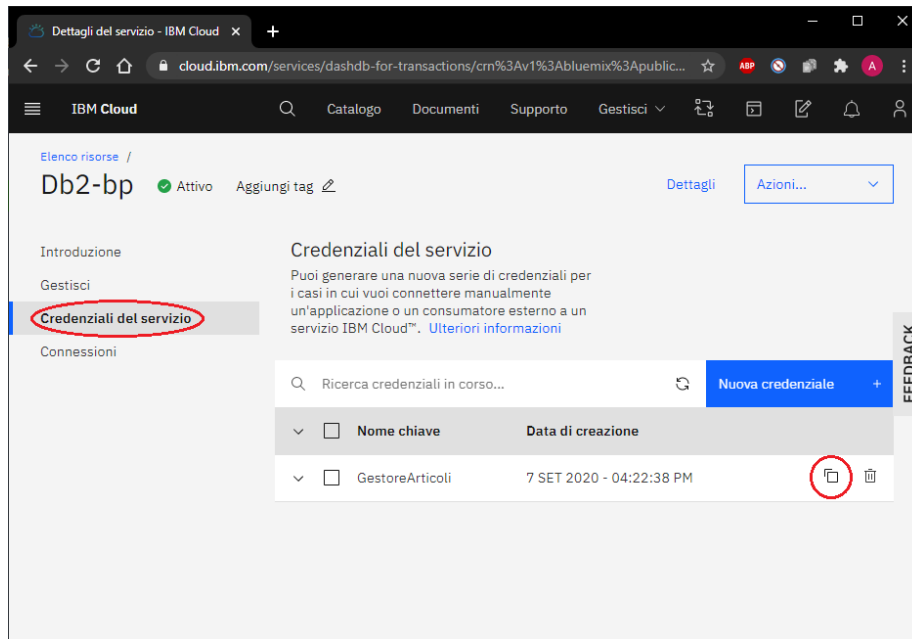


Figura 6: Schermata DB2 per ottenere le credenziali del servizio

Infine, il popolamento del database può essere effettuato attraverso due diverse funzioni. La prima, *insertArticleDB2*, inserisce un singolo articolo all'interno del database attraverso la seguente query sql:

```
INSERT INTO articoli (pubmedId, doi, titolo , autori ,
                     dataPubblicazione, abstract, nomeRivista)

VALUES (valore_pubmedId , valore_doi , valore_titolo ,
         valore_autori ,
         valore_dataPubblicazione , valore_abstract ,
         valore_nomeRivista);
```

La seconda, *insertArrayDB2*, è quella che è stata utilizzata per realizzare il primo popolamento del database. Essa riceve in ingresso un array di articoli, andando ad inserire ognuno di essi nel database stesso attraverso la stessa query vista precedentemente.

In entrambi i casi, si è resa, quindi, necessaria la definizione di una classe Articolo, la cui implementazione è riportata di seguito:

```
class Articolo():
    def __init__(self, titolo , autori , pubDate, pubMedId,
               doi, abstract, journalTitle):
        self.titolo = titolo
        self.autori = autori
        self.abstract = abstract
```

```

        self.doi = doi
        self.pubMedId = pubMedId
        self.pubDate = pubDate
        self.journalTitle = journalTitle
    def __str__(self):
        return "PUBMED ID: %s\nDOI: %s\nTitolo: %s\
            nAutori: %s\nData di Pubblicazione: %s\
            nAbstract: %s\nJournal Title: %s\n\n" % (self.
            pubMedId, self.doi, self.titolo, ', '.join(
            self.autori), self.pubDate, self.abstract,
            self.journalTitle)

```

L'implementazione delle funzioni sopra menzionate è la seguente:

```

def insertArticleDB2 (articolo):
    abstract_temp = " "
    titolo_temp = " "
    jtitle_temp = " "
    autori_insert = ', '.join(articolo.autori)
    if (autori_insert != None):
        autori_insert = autori_insert.replace(" ", " ")

    if (articolo.titolo != None):
        titolo_temp = articolo.titolo.replace(" ", " ")

    if (articolo.abstract != None):
        abstract_temp = articolo.abstract.replace(" ", " ")

    if (articolo.journalTitle != None):
        jtitle_temp = articolo.journalTitle.replace(" ", " ")

    insert = ibm_db.exec_immediate(conn, "INSERT INTO
    articoli (pubmedId, doi, titolo, autori,
    dataPubblicazione, abstract, nomeRivista) VALUES
    ('"+articolo.doi+"', '"+titolo_temp+"', '"+
    autori_insert+"', '"+ articolo.pubDate+"', '"+
    abstract_temp+"', '"+jtitle_temp+"')")

    print("[ "+str(i)+" ] - Inserito\n") #DEBUG dell'
        articolo inserito

def insertArrayDB2 (articoli):
    for articolo in articoli:
        abstract_temp = " "
        titolo_temp = " "

```

```

jtitle_temp = " "

autori_insert = ', '.join(articolo.autori)
if (autori_insert != None):
    autori_insert = autori_insert.replace(" ", "
    ")

if (articolo.titolo != None):
    titolo_temp = articolo.titolo.replace(" ", "
    ")

if (articolo.abstract != None):
    abstract_temp = articolo.abstract.replace(" ", "
    ")

if (articolo.journalTitle != None):
    jtitle_temp = articolo.journalTitle.replace("
    ", " ")

insert = ibm_db.exec_immediate(conn, "INSERT INTO
    articoli (pubmedId, doi, titolo, autori,
    dataPubblicazione, abstract, nomeRivista)
    VALUES ('"+articolo.doi+"', '"+titolo_temp+"',
    '"+ autori_insert+"', '"+ articolo.pubDate+"
    ', '"+abstract_temp+"', '"+jtitle_temp+"')")

print("[ "+str(i)+" ] - Inserito\n") #DEBUG dell'
articolo inserito

```

Inizialmente, si era pensato di mantenere in memoria le informazioni degli articoli recuperati, per poi caricarle tutte insieme sul database: questo metodo però è risultato poco efficiente a causa della grande mole di articoli da caricare. Tale risultato è stato ottenuto stressando il sistema tramite una query generica di prova, che cercasse gli articoli che contenessero in qualsiasi campo (sia esso Titolo, MeSH o Abstract) il termine "glioblastoma". Questa query di prova ha prodotto come risultato ben 41'364 articoli e pubblicazioni, rallentando di molto il processo di recupero, il mantenimento temporaneo in memoria e infine l'inserimento nel Database degli stessi. È risultato molto più conveniente, piuttosto, gestire grosse moli di articoli recuperandoli e inserendoli immediatamente nel database, in modo da non perdere l'eventuale cluster di articoli in caso di disconnessioni causate dal server di PubMed (problema che, più volte, è stato riscontrato durante il testing dello script). Conviene utilizzare, quindi, la funzione *insertArrayDB2* quando vengono recuperati quantità minime di articoli (nell'ordine massimo delle centinaia), come nel caso della query utilizzata ed esposta nel sottoparagrafo 3.1.2, dove la query ha restituito 1'318 articoli come esito della ricerca.

4.1.5 Funzioni implementate

Dopo aver ottenuto la lista contenente gli ID degli articoli, la funzione *pubMedIDConverter* recupera le informazioni ad essi associate. L'implementazione di tale funzione è la seguente:

```
def PubMedIdConverter(articoli):
    data_file = 'export.xml'
    tree = ET.parse(data_file)
    root = tree.getroot()
    id_converter = "https://eutils.ncbi.nlm.nih.gov/
        entrez/eutils/efetch.fcgi?db=pubmed&id="
    i=0

    for child in root.findall("./IdList/Id"):
        i=i+1
        id = child.text
        id_converter1 = id_converter+id+"&retmode=xml"

        xml = urllib.request.urlopen(id_converter1).read()
        xml_tree = ET.fromstring(xml)    # Root

        for child2 in xml_tree.findall("PubmedArticle"):
            temp_title = ""
            temp_abstract = ""
            temp_author = ""
            temp_doi = "Mancante"
            temp_pubdate = ""
            autori_vector = []

            for child2_0 in child2.findall("
                MedlineCitation/Article"):
                for child2_JournalTitle in child2_0.
                    findall("Journal/Title"):
                        temp_jtitle = child2_JournalTitle.
                            text
                for child2_1 in child2_0.findall("
                    ArticleTitle"):
                        temp_title = child2_1.text
                for child2_2 in child2_0.findall("
                    Abstract/AbstractText"):
                        temp_abstract = child2_2.text
                for child2_3 in child2_0.findall("
                    AuthorList/Author"):
                        #La presente soluzione e' concepita
                            nel caso si voglia tenere traccia
```

```

        solo di iniziali del nome e
        cognome (per intero)
path_cognome = child2_3.find("
    LastName")
path_iniziali = child2_3.find("
    Initials")
if (path_cognome != None):
    if (path_iniziali != None):
        temp_author = child2_3.find("
            LastName").text + " " +
            path_iniziali.text + "."
    else:
        temp_author = child2_3.find("
            LastName").text
    autori_vector.append(temp_author)
for child2_4 in child2.findall("PubmedData"):
    for child2_5 in child2_4.findall("History
        /PubMedPubDate/[@PubStatus='entrez']"):
        :
        temp_pubdate = child2_5[0].text+"-"+
            child2_5[1].text+"-"+child2_5[2].
            text
    childFinale = child2_4.find("
        ArticleIdList/ArticleId/[@IdType='doi
        ']")
    if (childFinale != None):
        temp_doi = childFinale.text
        print("[ "+str(i)+" ] - ", temp_doi) #
            Debug dell'articolo recuperato

articolo = Articolo (temp_title ,
    autori_vector , temp_pubdate , temp_pubmedid
    , temp_doi ,temp_abstract , temp_jtitle)
articoli.append(articolo)
return articoli

```

4.2 Flusso di esecuzione del sistema di apprendimento sugli articoli

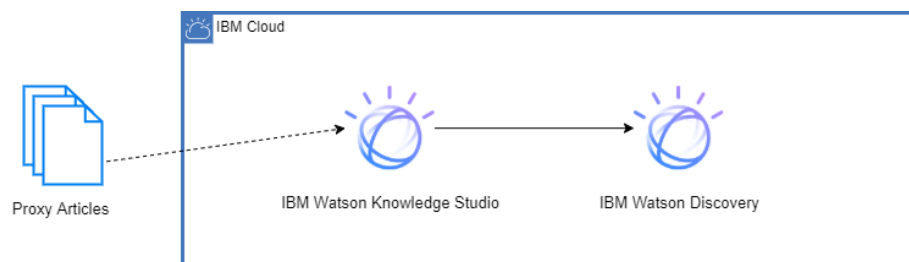


Figura 7: Diagramma di flusso dell'esecuzione

Osserviamo innanzitutto che all'interno di IBM Watson Knowledge Studio vengono memorizzati direttamente gli articoli completi. In particolare, momentaneamente gli articoli sono stati scaricati manualmente sfruttando i Proxy di diverse università campane, quali l'Università degli Studi di Napoli Federico II, l'Università degli Studi della Campania Luigi Vanvitelli e l'Università degli Studi di Salerno.

4.3 IBM Watson Knowledge Studio

4.3.1 Import degli Articoli

Come anticipato, inizialmente è stato utilizzato un set costituito da soli 8 articoli scaricati manualmente dalle rispettive riviste, attraverso i DOI corrispondenti memorizzati all'interno del database DB2.

In particolare, nella seguente immagine è possibile osservare gli articoli memorizzati all'interno di IBM Watson Knowledge Studio:

Name	Documents
All	7
quinn2009.pdf_set	1
wick2016.pdf_set	1
nabors2011.pdf_set	1
grossmann2012-2.pdf_set	1
beauchesne2010.pdf_set	1
ahluwalia2010.pdf_set	1
wakabayashi2018.pdf_set	1
Items per page: 10 ▾	1-8 of 8 items

Figura 8: Primo set di articoli caricati su IBM Watson Knowledge Studio

4.3.2 Creazione del Modello Entità-Relazione

Per realizzare il modello di NLP, innanzitutto, è stato necessario definire le Entità e le Relazioni che dovranno essere annotate all'interno degli articoli.

A seguito di un primo confronto con gli esperti di dominio, essi hanno fornito un primo insieme di entità e relazioni sufficienti per effettuare un'operazione di pre-annotation³. In particolare, in seguito a tale primo colloquio, sono state definite le seguenti relazioni e le seguenti entità (evidenziate in grassetto), con rispettivi sinonimi, abbreviazioni e termini appartenenti a quella data entità:

³Un pre-annotator è un dizionario di Knowledge Studio, un modello Rule-Based oppure un modello di machine learning che può essere eseguito per annotare automaticamente un testo.

Entità

DISEASE

- Molecular classification
- Neoplasm
- Malignancy
- Malignant glioma
- Tumor
 - Oligodendroglioma
 - Anaplastic glioma
 - HGG
 - GBM
 - Glioblastoma
 - High grade glioma
 - WHO grade IV
- Residual
- Recurrence
- Relapse

TRIAL

- Clinical trial registry
- Phase I
- Phase II
- Phase III
- Study design
- Open-label
- Exclusion criteria
- Inclusion criteria

THERAPY

- Chemotherapy
 - Alkylating agent
 - Nimustine
 - Temozolomide
 - Carboplatin
 - Interferon
 - Anti-VEGF
 - Bevacizumab
 - Sorafenib
 - HIV protease inhibitors
 - Ritonavir
 - Lopinavir
 - microtubule- destabilizing agent
 - Verubulin
 - convection-enhanced delivery
- Radiotherapy
 - Radiation therapy
 - Irradiation
 - Fractioned radiation therapy
 - Unfractioned radiation therapy
- Surgery
 - Biopsy
 - Inoperable
 - Operable
 - Gross total tumor resection
 - Complete tumor resection
 - Partial resection
 - Residual

RESPONSE TO THERAPY

- Dosage

Toxicity
Efficacy
Tolerance
Safety
Adverse reaction
Response assessment
Progression
Regression
Partial response
Stable disease
Pseudoprogression
Pseudoresponse
Survival
Overall survival
Progression free survival
Karnofsky status

Relazioni

Disease - therapy
Disease - trial
Trial - therapy
Therapy - response to therapy

Sinonimi e abbreviazioni

Neoplasm - malignancy - tumor
Glioblastoma - GBM - High grade glioma - HGG
Therapy - treatment
HIV protease inhibitors - HIVPI
Radiotherapy - irradiation - radiation therapy - RT
Gross total tumor resection - Complete tumor resection
Tolerance - tolerability
Overall Survival - OS
Progression free survival - PFS
Karnofsky status - KS
Temozolomide - TMZ
Nimustine - INN
Interferon - IFN
convection-enhanced delivery - CED

A partire dalle informazioni forniteci dagli esperti di dominio, è stato possibile definire il seguente Modello Entità-Relazione:

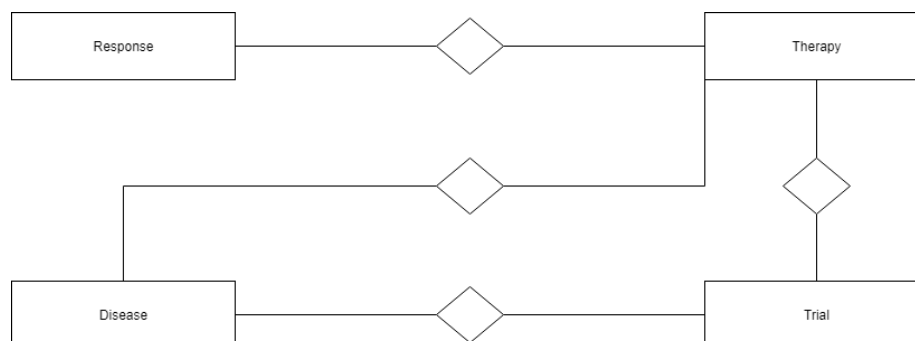


Figura 9: Modello Entità-Relazione

Definito il modello, è stato definito un Dizionario tale che associasse i termini specificati nell'elenco ricevuto dagli esperti di dominio alle rispettive entità.

Create Dictionary

Disease 13

Trial 8

Therapy 25

Response 17

Disease

Language: English | 13 entries

Entity type: Disease

Rule class: None

Add Entry Upload

Download

Enter text to filter

<input type="checkbox"/>	Lemma	Surface Forms	Part of Speech	Action
<input type="checkbox"/>	Molecular classification	Molecular classification	Noun	Edit Delete
<input type="checkbox"/>	WHO grade IV	WHO grade IV	Noun	Edit Delete
<input type="checkbox"/>	Anaplastic glioma	Anaplastic glioma	Noun	Edit Delete
<input type="checkbox"/>	Oligodendroglioma	Oligodendroglioma	Noun	Edit Delete
<input type="checkbox"/>	Relapse	Relapse	Noun	Edit Delete
<input type="checkbox"/>	Recurrence	Recurrence	Noun	Edit Delete
<input type="checkbox"/>	Residual	Residual	Noun	Edit Delete
<input type="checkbox"/>	Tumor	Tumor, Neoplasm, Malignancy	Noun	Edit Delete
<input type="checkbox"/>	GBM	Glioblastoma, GBM, High grade glioma, HGG	Noun	Edit Delete
<input type="checkbox"/>	Tumor	Tumor	Noun	Edit Delete

Figura 10: Dizionario definito

Una volta prodotto il Dizionario, è stata processata la pre-annotazione dei testi precedentemente caricati.

Di seguito è possibile osservare alcuni esempi di risultati ottenuti sugli 8 testi:

8 Progressive disease (PD) at 10 weeks was reconfirmed at 18 weeks to distinguish pseudoprogression.

9 **Progression**-free survival (**PFS**), excluding cases of confirmed pseudoprogression, was assessed (post-hoc/exploratory).

10 **Tumor** progression patterns were determined at each disease assessment/PD (prespecified/exploratory).

11 Results.

12 Of patients with PD in the bevacizumab and placebo arms, 143/354 (40.4%) and 155/387 (40.1%), respectively, had PD due to contrast-enhancing lesions, and 51/354 (14.4%) and 53/387 (13.7%) had PD due to nonenhancing lesions.

13 Of all patients in the bevacizumab arm (n¼ 458), 2.2% had confirmed pseudoprogression versus 9.3% in the placebo arm (n¼ 463).

14 Baseline characteristics did not differ between patients with/without pseudoprogression (including for MGMT status).

15 Excluding confirmed pseudoprogression, **PFS** (hazard ratio: 0.65, 95% CI: 0.56–0.75; P , .0001, bevacizumab vs placebo) was comparable to the intent- to-treat population.

16 At PD, most patients had the same tumor focus (local/multifocal, .84%) and infiltrative profile (.88%) as at baseline; no shift to a diffuse or multifocal phenotype was observed.

17 Conclusions.

18 **Pseudoprogression** complicated progression assessment in a small but relevant number of patients but had negligible impact on **PFS**.

19 **Bevacizumab** did not appear to adversely impact tumor progression patterns.

20 Keywords: glioblastoma, pseudoprogression, response, tumor pattern.

21 **Glioblastoma** is an aggressive and invasive brain malignancy with poor prognosis.1 Standard-of-care therapy for newly diag- nosed glioblastoma comprises concurrent radiotherapy and temozolomide (**RT**, **TMZ**), followed by adjuvant **TMZ**.2 In the ran- domized,

Type	Subtype	Role
Disease		
Response		
Therapy		
Trial		

Figura 11: Esempio di annotazione

quinn2009.pdf

1 556Neuro-Oncology■JULY 2006 Neuro-Oncology This phase I clinical trial conducted with patients who had recurrent or progressive malignant glioma (MG) was designed to determine the maximum tolerated dose (MTD) and toxicity of three different 5-day dosing regimens of temozolomide (**TMZ**) in combination with O6-benzylguanine (O6-BG).

2 Both **TMZ** and O6-BG were administered on days 1–5 of a 28-day treatment cycle.

3 A bolus infusion of O6-BG was administered at 120 mg/m2 over 1 h on days 1, 3, and 5, along with a continuous infusion of O6-BG at 30 mg/m2/day.

4 **TMZ** was admin- istered at the end of the first bolus infusion of O6-BG and then every 24 h for 5 days during the continuous infusion of O6-BG.

5 Patients were accrued to one of three 5-day dosing regimens of **TMZ**.

6 Twenty-nine patients were enrolled into this study.

7 The dose-limiting toxicities (DLTs) were grade 4 neutropenia, leukopenia, and thrombocytopenia.

8 The MTD for **TMZ** for the three different 5-day dosing schedules was determined as fol- lows: schedule 1, 200 mg/m2 on day 1 and 50 mg/m2/day on days 2–5; schedule 2, 50 mg/m2/day on days 1–5; and schedule 3, 50 mg/m2/day on days 1–5 while receiving pegfilgrastim.

9 Thus, the 5-day **TMZ** dosing schedule that **Phase I** trial of temozolomide plus O6-benzylguanine 5-day regimen with recurrent malignant glioma Jennifer A. Quinn, Sara Xiaoyin Jiang, David A. Reardon, Annick Desjardins, James J. Vredenburgh, Jeremy N. Rich, Sridharan Gururangan, Allan H. Friedman, Darell D. Bigner, John H. Sampson, Roger E. McLendon, James E. Herndon, Jr., Amy Walker, and Henry S. Friedman Departments of **Surgery** (J.A.Q., S.X.J., D.A.R., A.D., J.J.V., J.N.R., S.G., A.H.F., D.D.B., J.H.S.,

Type	Subtype	Role
Disease		
Response		
Therapy		
Trial		

Figura 12: Esempio di annotazione

5 Apprendimento sulle Risonanze Magnetiche tramite Visual Recognition

5.1 Flusso di esecuzione del sistema di apprendimento sulle MRI

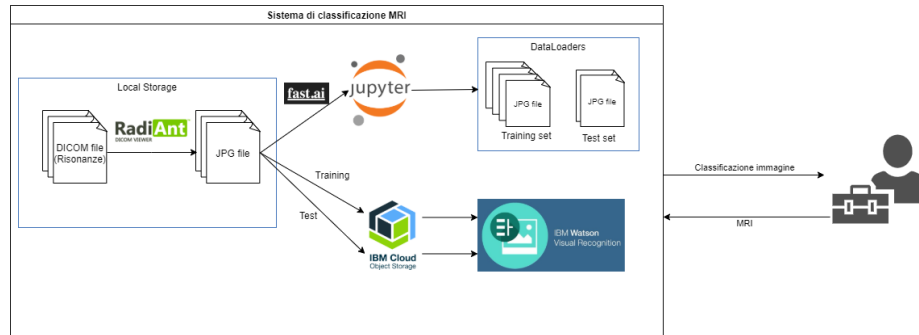


Figura 14: Diagramma di flusso dell'esecuzione

In un primo momento, i file DICOM sono stati sottoposti ad una fase di pre-processing: essi sono stati aperti tramite il tool *RadiAnt DICOM Viewer* e convertiti in formato JPG: laddove fosse necessario, successivamente alla conversione in JPG i file sono stati anche ridimensionati ad una dimensione pari a 256x256 pixel.

Dopo la fase di pre-processing, le risonanze sono state utilizzate per la realizzazione di due reti differenti:

1. una CNN implementata nella libreria *fastai*;
2. il tool di Visual Recognition di Watson Studio.

5.2 CNN con la libreria *fastai*

5.2.1 Implementazione

I file DICOM forniteci dagli esperti di dominio sono stati etichettati dagli stessi esperti, che hanno distinto due diverse classi di immagini. In particolare, ci è stato fornito un insieme di 691 immagini, divise in due diverse classi. La prima è la classe GBM che contiene 489 immagini di glioblastomi multiformi, mentre la seconda è la classe MET che contiene 202 immagini di metastasi.

Innanzitutto, i file DICOM etichettati dagli esperti di dominio sono stati suddivisi nei due insiemi di Training e Validation. In particolare, per realizzare la suddivisione è stata sfruttata la tecnica dell'Holdout⁴, per cui il Training Set contiene l'80% dei file, mentre il Validation Set contiene il 20% rimanente.

⁴La tecnica dell'Holdout prevede di suddividere un dataset in modo tale che 2/3 di esso siano usati come Training Set e il restante 1/3 viene usato come Test Set.

Ai file DICOM deve essere associato un file **.csv** in cui si specifica l'etichetta associata ad ognuno dei file. Specificando il path in cui è presente il file di etichettamento, il dataset viene caricato utilizzando la classe built-in *DataBlock*:

```
gbm = DataBlock(blocks=(ImageBlock, CategoryBlock),
                 get_x=lambda x: gbm_source/f"{x[0]}" ,
                 get_y=lambda x: x[1] ,
                 batch_tfms=aug_transforms(size=300))
```

- *gbm*: variabile che conterrà l'intero DataBlock, ovvero l'intero dataset delle risonanze;
- *ImageBlock*: funzione che specifica cosa dovranno contenere i blocchi;
- *gbm_source*: variabile precedentemente dichiarata contenente il Path del file di labeling delle risonanze.

Caricato il dataset, si generano il Training Set ed il Validation Set che verranno utilizzati per l'addestramento del sistema e la valutazione della sua accuracy. Ciò viene effettuato tramite la funzione *dataloaders* presente all'interno della classe DataBlock:

```
dls = gbm.dataloaders(df.values, num_workers=0, bs = int
                      (len(df.values)*0.80), drop_last=True)
```

- *df.values*: carica i valori delle etichette;
- *num_workers*: indica il numero di sottoprocessi per il caricamento dei dati. Il valore di default 0 indica che viene effettuato dal main process;
- *bs*: indica il dimensionamento dei due Set tramite una suddivisione in **batch**;
- *drop_last*: elimina eventuali batch generati in eccesso non contenenti elementi del dataset.

È possibile visualizzare il caricamento effettuato sui due set:

```
dls.train.show_batch(max_n=100)
```

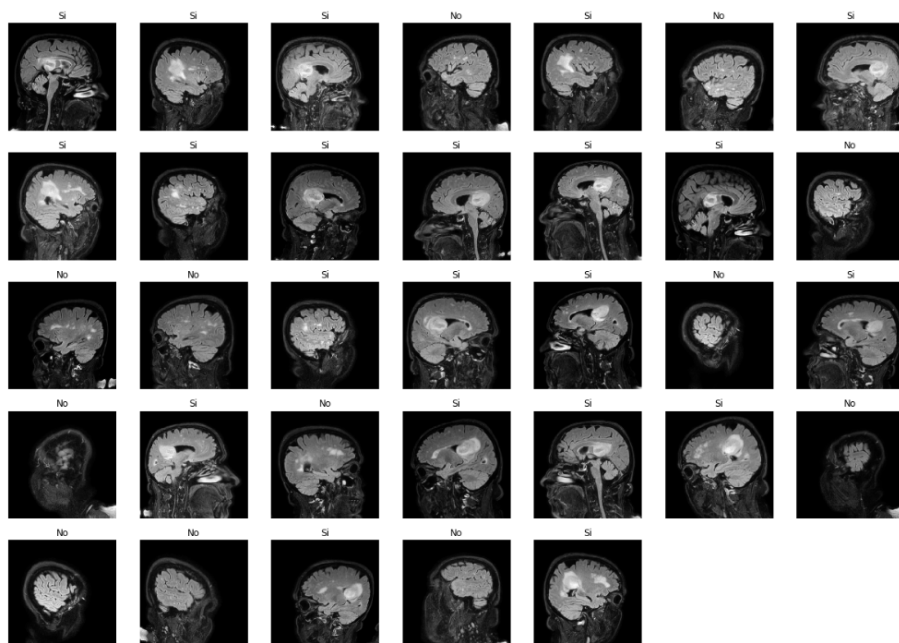



Figura 15: Visualizzazione del Training Set

```
dls.valid.show_batch(max_n=100)
```

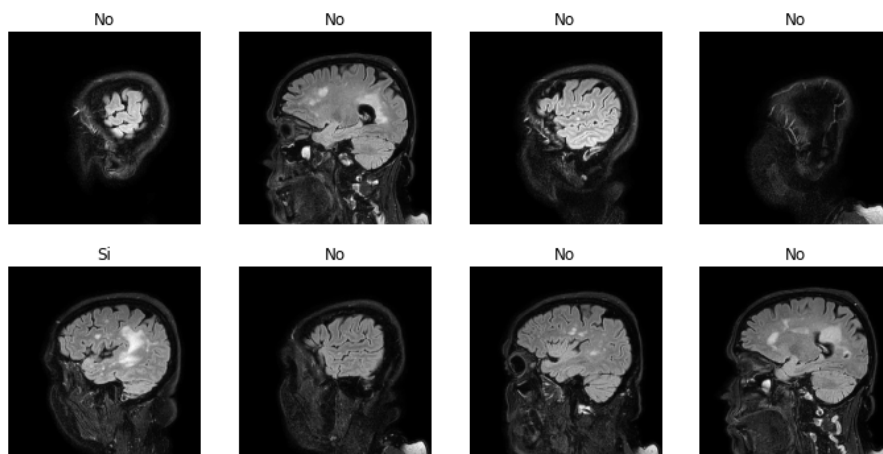


Figura 16: Visualizzazione del Validation Set

5.2.2 Risultati ottenuti

Generati Training e Validation Sets, è stato effettuato l'addestramento del classificatore, che ha impiegato un tempo pari a 6'03".

Dopodiché, il classificatore ottenuto è stato valutato tramite le metriche di **Accuracy**, **Precision** e **Recall**, ripetendo i cicli di addestramento della CNN fino a quando non raggiungesse un valore di accuracy pari all'83%. Tale valore di soglia è stato scelto in modo tale da evitare il verificarsi del fenomeno dell'overfitting per valori maggiori di accuracy.

Un esempio di addestramento della CNN è osservabile nella figura sottostante:

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.251160	0.729759	0.611111	0.611111	0.611111	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.346079	0.673051	0.666667	0.666667	0.666667	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.293452	0.638695	0.666667	0.666667	0.666667	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.172186	0.592265	0.666667	0.666667	0.666667	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.128820	0.543484	0.666667	0.666667	0.666667	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.085976	0.493860	0.722222	0.722222	0.722222	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.102275	0.446594	0.777778	0.777778	0.777778	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.115888	0.397800	0.777778	0.777778	0.777778	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.095570	0.346179	0.777778	0.777778	0.777778	00:00

epoch	train_loss	valid_loss	accuracy	precision_score	recall_score	time
0	0.075943	0.300927	0.833333	0.833333	0.833333	00:00

Numero di cicli di apprendimento: 34

Figura 17: Esempio di addestramento della CNN

Il risultato della classificazione ottenuto è, quindi, il seguente:

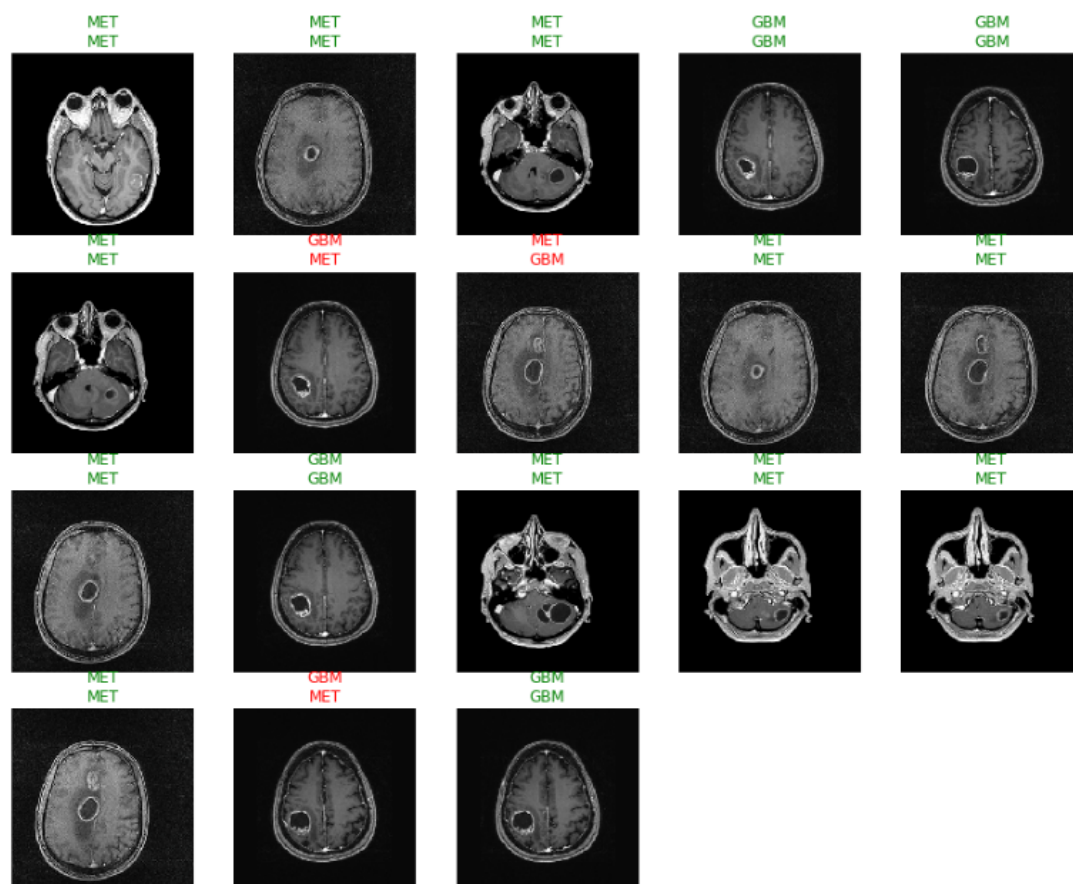


Figura 18: Classificazione effettuata dalla CNN

5.3 Visual Recognition di Watson Studio

L'addestramento del classificatore attraverso Watson Studio è stato addestrato attraverso diversi cicli di affinamento che analizzeremo nel dettaglio di seguito:

1. Primo Ciclo: il classificatore è stato addestrato fornendogli in ingresso un Training set costituito da un totale di 691 immagini, divise in due diverse classi. La prima è la classe GBM che contiene 489 immagini di glioblastomi multiforini, mentre la seconda è la classe MET che contiene 202 immagini di metastasi. Osserviamo che, in pratica il Training Set adoperato per realizzare il primo ciclo di addestramento è, in pratica, lo stesso Training Set utilizzato per addestrare il classificatore con **fastai**;
2. Secondo Ciclo: il classificatore è stato addestrato fornendogli in ingresso un Training set costituito da un totale di 1'838 immagini, divise in due diverse classi ed un set di immagini definite come Negative. In particolare, le classi GBM e MET sono rimaste invariate rispetto al primo ciclo di addestramento, mentre le 1'147 immagini Negative sono immagini che non appartengono a nessuna di esse;
3. Terzo Ciclo: il classificatore è stato addestrato fornendogli in ingresso un Training set costituito da un totale di 2'103 immagini, divise in due diverse classi ed un set di immagini definite come Negative. In particolare, le immagini definite come Negative sono rimaste invariate, mentre sono state aggiunte 221 immagini classificate come GBM e 44 immagini classificate, invece, come MET.

5.3.1 Implementazione

Innanzitutto, una prima parte di dati etichettati forniti dagli esperti di dominio è stata utilizzata come Training Set con lo scopo di far apprendere il modello. Dopodiché, un secondo insieme di dati etichettati dagli esperti di dominio è stato utilizzato come Test Set al fine di poter valutare la metriche di **Accuracy**, **Precision**, **Recall** ed **F1-score**.

In particolare, è stato realizzato un notebook in linguaggio Python che, sfruttando le IBM Cloud API, ci ha consentito di utilizzare il classificatore precedentemente realizzato per effettuare una classificazione del Test Set.

In primo luogo è stata generata un'istanza di **VisualRecognitionV3**, che consente di realizzare il collegamento con l'istanza di Visual Recognition addestrata all'interno di Watson Studio. Ciò è stato realizzato attraverso la funzione *createVRinstance(parameters)*, che riceve in ingresso un insieme di parametri specificati all'interno di un file JSON. In particolare, i campi specificati all'interno del file sono i seguenti:

- url;
- apikey;
- vr_id: id del classificatore da noi istruito;

- `test_csv_file`: file csv contenente i nomi delle immagini appartenenti al Test Set e le rispettive classificazioni;
- `results_csv_file`: file csv in cui verranno memorizzati i risultati;
- `confmatrix_csv_file`: file csv in cui verrà memorizzata la Confusion Matrix (Matrice di Confusione⁵).

Le informazioni contenute all'interno del file JSON vengono memorizzate all'interno di un oggetto di tipo `Parameters`:

```
def readParameters(parmsPath):
    #Lettura dei parametri utili da un file json
    vrParmsFile = parmsPath
    parms = ''
    with open(vrParmsFile) as parmFile:
        parms = json.load(parmFile)

    parameters = Parameters(parms['url'], parms['apikey'],
                           parms['vr_id'], parms['test_csv_file'], parms['results_csv_file'],
                           parms['confmatrix_csv_file'], parms['version'])
    json.dumps(parms)

    return parameters
```

I parametri appena acquisiti, come anticipato, vengono utilizzati per la creazione di un'istanza di Visual Recognition, ovvero un'istanza della classe **VisualRecognitionV3**, definita all'interno della libreria **ibm_watson**. L'implementazione della funzione *createVRinstance(parameters)* è riportata di seguito:

```
def createVRinstance(parameters):
    #Creazione di una istanza di Visual Recognition
    authenticator = IAMAuthenticator(parameters.apiKey)
    visualRecognition = VisualRecognitionV3(
        version = parameters.version,
        authenticator = authenticator
    )
    visualRecognition.set_service_url(parameters.url)

    return visualRecognition
```

Dopo la generazione dell'istanza della classe **VisualRecognitionV3**, è possibile effettuare la classificazione delle immagini contenute all'interno del Test

⁵Nell'ambito dell'Intelligenza artificiale, la matrice di confusione, detta anche tabella di errata classificazione, restituisce una rappresentazione dell'accuratezza di classificazione statistica. Ogni colonna della matrice rappresenta i valori predetti, mentre ogni riga rappresenta i valori reali. L'elemento sulla riga *i* e sulla colonna *j* è il numero di casi in cui il classificatore ha classificato la classe "vera" *i* come classe *j*. Attraverso questa matrice è osservabile se vi è "confusione" nella classificazione di diverse classi.

Set. A tal fine, innanzitutto è stata definita la funzione *getVRresponse* che va a classificare una singola immagine restituendo una lista delle classi associate all'immagine stessa, ordinata in base al valore di Confidence (score). La funzione riceve in ingresso 3 parametri, quali l'istanza di visual recognition, l'id del classificatore ed il path dell'immagine da classificare. L'implementazione della funzione è riportata di seguito:

```
def getVRresponse(vr_instance , classifierID , image_path):
    with open(image_path , 'rb') as images_file:
        image_results = vr_instance.classify(
            images_file = images_file ,
            threshold='0.6' ,
            classifier_ids=classifierID ).get_result()

    #Ogni singola immagine viene classificata attraverso
    il classificatore da noi istruito
    classList = []
    for classifier in image_results['images'][0]['
        classifiers']:
        if classifier['classifier_id'] == parameters.vrId
            :
            classList = classifier['classes']
        break

    #Le classi vengono ordinate in base allo score
    sorted_classList = sorted(classList , key=lambda k: k.
        get('score' , 0) , reverse=True)
    return sorted_classList
```

Sfruttando la funzione appena definita, è stato possibile definire un'ulteriore funzione *batchVR*, che consente di classificare un batch di immagini restituendo, per ciascuna di esse, i risultati ottenuti. In particolare, i path delle immagini appartenenti al batch da classificare vengono specificati all'interno di un file **csv**, insieme alla classe di appartenenza dell'immagine stessa. La funzione riceve in ingresso 3 parametri, quali l'istanza di visual recognition, l'id del classificatore ed il path del file csv. L'implementazione della funzione è riportata di seguito:

```
def batchVR(vr_instance , classifierID , csvfile):
    testClasses=[]
    vrPredictClasses=[]
    vr_predict_confidence=[]
    images=[]
    i=0
    with open(csvfile , 'r') as csvfile:
        csvReader=csv.DictReader(csvfile)
        for row in csvReader:
            testClasses.append(row['class'])
            vr_response = getVRresponse(vr_instance ,
                classifierID ,row['image'])
```

```

vrPredictClasses.append(vr_response[0]['class
'])
vr_predict_confidence.append(vr_response[0]['
score'])
images.append(row['image'])
i = i+1
if(i%250 == 0):
    print("")
    print("Processed ", i, " records")
if(i%10 == 0):
    sys.stdout.write('.')
print("")
print("Finished processing ", i, " records")
return testClasses, vrPredictClasses,
vrPredictConfidence, images

```

5.3.2 Risultati ottenuti - Primo Ciclo

Il tempo impiegato per effettuare il Training del classificatore dato il Training Set descritto precedentemente (che consta di 691 immagini) è pari a 36'17".

Le metriche di **Accuracy**, **Precision**, **Recall** ed **F1-score** sono state stimate attraverso l'utilizzo delle funzioni *accuracy_score* e *classification_report*, entrambe definite all'interno della libreria **sklearn**.

Di seguito sono riportati i risultati ottenuti:

```

accuracy = accuracy_score(testClasses, vrPredictClasses)
print('Classification Accuracy: ', accuracy)

```

Classification Accuracy: 0.6452599388379205

Figura 19: Accuracy della classificazione ottenuta

```

classification_report(testClasses, vrPredictClasses,
labels=labels)

```

	precision	recall	f1-score	support
GBM	0.71	0.80	0.76	223
MET	0.42	0.31	0.36	104
accuracy			0.65	327
macro avg	0.57	0.56	0.56	327
weighted avg	0.62	0.65	0.63	327

Figura 20: Valori di Precision, Recall e F1-Score

Inoltre, è stata generata e memorizzata all'interno di un file **csv** la matrice di confusione attraverso la funzione *confusion_matrix*, anch'essa definita all'interno della libreria **sklearn**:

```
labels=list(set(testClasses))
vrConfusionMatrix = confusion_matrix(testClasses,
                                     vrPredictClasses, labels)
confmatrix2csv(vrConfusionMatrix, labels, parameters.
               confMatrixCsvFile)
```

La matrice di confusione ottenuta è, quindi, la seguente:

```
df_cm = pd.DataFrame(vrConfMatrix, range(2), range(2))
sn.set(font_scale=1.4). # Label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16})
# Font size
plt.show()
```

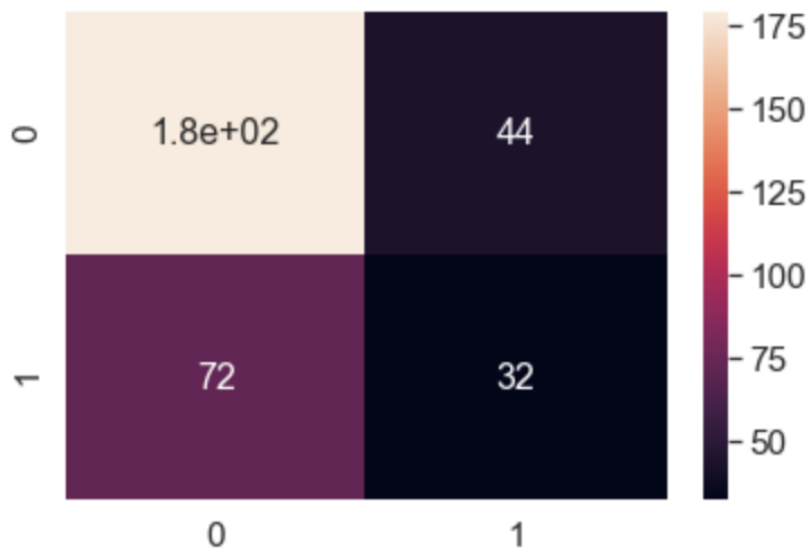


Figura 21: Matrice di Confusione

Infine, tale matrice è stata memorizzata all'interno di un file **csv** attraverso la funzione *confmatrix2csv*, la cui implementazione è riportata di seguito:

```
def confmatrix2csv(conf_matrix, labels, csvfile):
    with open(csvfile, 'w') as csvfile:
        csvWriter = csv.writer(csvfile)
        row=list(labels)
        row.insert(0,"")
        csvWriter.writerow(row)
```



```

for i in range(conf_matrix.shape[0]):
    row=list(conf_matrix[i])
    row.insert(0,labels[i])
    csvWriter.writerow(row)

```

5.3.3 Risultati ottenuti - Secondo Ciclo

Il tempo impiegato per effettuare il Training del classificatore dato il Training Set descritto precedentemente (che consta di 1'838 immagini) è pari a 1h56'59". Le metriche di **Accuracy**, **Precision**, **Recall** ed **F1-score** sono state stimate attraverso l'utilizzo delle funzioni *accuracy_score* e *classification_report*, entrambe definite all'interno della libreria **sklearn**.

Di seguito sono riportati i risultati ottenuti:

```

accuracy = accuracy_score(testClasses, vrPredictClasses)
print('Classification Accuracy: ', accuracy)

```

Classification Accuracy: 0.5192307692307693

Figura 22: Accuracy della classificazione ottenuta

```

classification_report(testClasses, vrPredictClasses,
    labels=labels)

```

	precision	recall	f1-score	support
GBM	1.00	0.52	0.68	52
micro avg	1.00	0.52	0.68	52
macro avg	1.00	0.52	0.68	52
weighted avg	1.00	0.52	0.68	52

Figura 23: Valori di Precision, Recall e F1-Score

Osserviamo, quindi, che c'è stato un calo dell'accuracy del classificatore di circa il 20% rispetto al primo ciclo di addestramento. Tale diminuzione è una conseguenza del fatto che il classificatore non sia più stato in grado di riconoscere le metastasi (immagini appartenenti alla classe MET). Infatti, tutte le immagini classificate come MET appartenenti al Test Set sono state erroneamente riconosciute come Negative.

Inoltre, è stata generata e memorizzata all'interno di un file **csv** la matrice di confusione attraverso la funzione *confusion_matrix*, anch'essa definita all'interno della libreria **sklearn**:

```

labels=list(set(testClasses))
vrConfusionMatrix = confusion_matrix(testClasses,
    vrPredictClasses, labels)
confmatrix2csv(vrConfusionMatrix, labels, parameters.
    confMatrixCsvFile)

```

La matrice di confusione ottenuta è, quindi, la seguente:

```

df_cm = pd.DataFrame(vrConfMatrix, range(2), range(2))
sn.set(font_scale=1.4). # Label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16})
    # Font size
plt.show()

```

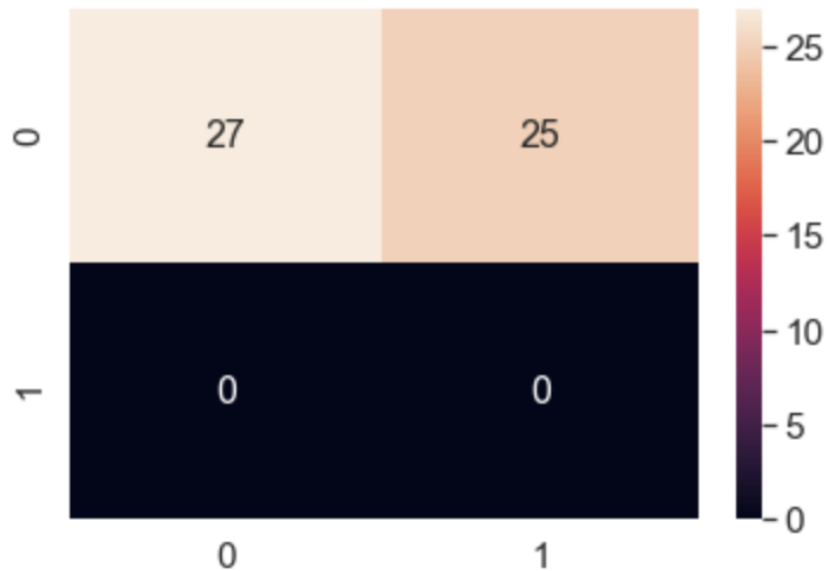


Figura 24: Matrice di Confusione

Dalla matrice è possibile osservare come, appunto, il classificatore non abbia classificato affatto le immagini appartenenti alla classe MET che, invece, sono state riconosciute tutte come Negative.

Infine, tale matrice è stata memorizzata all'interno di un file **csv** attraverso la funzione *confmatrix2csv*.

5.3.4 Risultati ottenuti - Terzo Ciclo

Il tempo impiegato per effettuare il Training del classificatore dato il Training Set descritto precedentemente (che consta di 2'103 immagini) è pari a ????.

Le metriche di **Accuracy**, **Precision**, **Recall** ed **F1-score** sono state stimate

attraverso l'utilizzo delle funzioni *accuracy_score* e *classification_report*, entrambe definite all'interno della libreria **sklearn**.

Di seguito sono riportati i risultati ottenuti:

```
accuracy = accuracy_score(testClasses , vrPredictClasses)
print('Classification Accuracy: ', accuracy)
```

```
classification_report(testClasses , vrPredictClasses ,
                      labels=labels)
```

Inoltre, è stata generata e memorizzata all'interno di un file **csv** la matrice di confusione attraverso la funzione *confusion_matrix*, anch'essa definita all'interno della libreria **sklearn**:

```
labels=list(set(testClasses))
vrConfusionMatrix = confusion_matrix(testClasses ,
                                     vrPredictClasses , labels)
confmatrix2csv(vrConfusionMatrix , labels , parameters.
               confMatrixCsvFile)
```

La matrice di confusione ottenuta è, quindi, la seguente:

```
df_cm = pd.DataFrame(vrConfMatrix , range(2) , range(2))
sn.set(font_scale=1.4). # Label size
sn.heatmap(df_cm , annot=True , annot_kws={"size": 16})
    # Font size
plt.show()
```

Infine, tale matrice è stata memorizzata all'interno di un file **csv** attraverso la funzione *confmatrix2csv*.

6 Riferimenti

- E-utilities: <https://www.ncbi.nlm.nih.gov/books/NBK25501/>;
- IBM Cloud Db2: <https://www.ibm.com/it-it/cloud/db2-on-cloud>;
- IBM Cloud Foundry: <https://www.ibm.com/it-it/cloud/cloud-foundry>;
- GitHub: <https://github.com>;
- MeSH: https://it.wikipedia.org/wiki/Medical_Subject_Headings;
- IBM Watson Machine Learning: <https://www.ibm.com/it-it/cloud/machine-learning>;
- IBM Watson Knowledge Studio: <https://www.ibm.com/it-it/cloud/watson-knowledge-studio>;
- IBM Watson Discovery: <https://www.ibm.com/it-it/cloud/watson-discovery>;
- fastai: <https://www.fast.ai>;
- IBM Cloud API: <https://cloud.ibm.com/apidocs/visual-recognition/visual-recognition-v3>;
- Jupyter: <https://jupyter.org>.