IMAGE CLASSIFICATION BY NEURAL NETWORK OF A NIfTI IMAGE

Francesco Vaccaro

Giuseppe Spartera

Introduzione al progetto

Si vuole effettuare una segmentazione ibrida, utilizzando una rete neurale “convenzionale”, di immagini TAC con formato **NIfTI**. Le immagini utilizzate sono presenti nel database TCIA (https://wiki.cancerimagingarchive.net/display/Public/CTORG%3A+CT+volumes+with+multiple+organ+segmentations). Questo set di immagini è composto da delle immagini di TAC e le corrispettive immagini segmentate attraverso una rete neurale convoluzionale. In questo progetto si cerca di riprodurre i risultati ottenuti utilizzando però una rete “convenzionale”, prelevando dall’immagine dei quadrati di pixel 5x5 e classificarli in base alla classificazione del pixel centrale. Se ad esempio il pixel centrale è classificato come polmone o osso, tutto il quadrato sarà classificato come tale. A questo scopo è stato scelto come linguaggio di programmazione Python, data la grande disponibilità di librerie. La scelta è anche dovuta al fatto che un’eventuale implementazione di una rete neurale in Python è consigliabile rispetto ad un programma come Matlab. Lo scopo del progetto è la classificazione di immagini TC attraverso l’utilizzo di una rete neurale. Per questa classificazione è necessario elaborare un algoritmo che ci permetta di estrarre piccoli campioni dall’immagine per addestrare la rete. Questi campioni vengono esportati in un file di testo (o CSV) per inserirli nella rete. La struttura della rete è presa da progetti simili già disponibili in letteratura scientifica.

Introduzione al formato NIfTI

Una varietà di tecnologie di neuroimaging consente di studiare la struttura e la funzione del cervello umano intatto con un'invasione minima, offrendo un'enorme opportunità per comprendere meglio il cervello umano, sia in condizioni di salute che disordinate. Il neuroimaging fornisce una prospettiva cruciale per le neuroscienze umane di base e cliniche. Gli strumenti informatici sono fondamentali in tutte le fasi del neuroimaging, poiché consentono agli scienziati di controllare strumenti di imaging altamente sofisticati e di dare un senso alle grandi quantità di dati complessi da essi generati. Ad esempio, nella risonanza magnetica funzionale (fMRI), l'informatica viene utilizzata per: progettare e implementare il modo in cui gli strumenti di imaging catturano i segnali dal cervello, guidare i compiti comportamentali utilizzati per sondare particolari sistemi cerebrali, ricostruire i segnali risultanti in una rappresentazione tridimensionale del cervello, correggere e sopprimere il rumore, analizzare statisticamente i dati e visualizzare i risultati. Questo elenco non è completo e, naturalmente, l'informatica viene utilizzata anche per archiviare, interrogare, recuperare, condividere e confrontare i dati una volta raccolti.

La Neuroimaging Informatics Technology Initiative (NIfTI)

La Neuroimaging Informatics Technology Initiative (NIfTI) ha lo scopo di lavorare con le comunità di utenti e sviluppatori di strumenti per rispondere a queste esigenze. Pertanto, l'obiettivo principale di NIfTI è fornire servizi, formazione e ricerca coordinati e mirati per accelerare lo sviluppo e migliorare l'utilità degli strumenti informatici relativi al neuroimaging. L'Istituto Nazionale di Salute

Mentale e l'Istituto Nazionale di Malattie Neurologiche e Ictus sono sponsor congiunti di questa iniziativa.

L'obiettivo iniziale di NIfTI sarà sugli strumenti utilizzati nella fMRI. Ci sono diversi motivi per concentrarsi su quest'area. Innanzitutto, un'area sostanziale circoscritta come la fMRI offre un banco di prova limitato per NIfTI, che rappresenta un nuovo approccio per affrontare un problema a livello di comunità. In secondo luogo, l'fMRI è in rapida crescita e il miglioramento degli strumenti informatici avrà probabilmente ampi benefici per le neuroscienze. In terzo luogo, esiste un piccolo numero di strumenti informatici ampiamente utilizzati utilizzati da molti nella comunità di ricerca fMRI, quindi ci sono maggiori punti in comune relativi agli strumenti in quest'area rispetto ad altre modalità di neuroimaging, aumentando così il rapporto costi/benefici. Se NIfTI si dimostra utile nell'affrontare questioni informatiche nella comunità di ricerca fMRI, può essere ampliato per affrontare problemi simili in altre aree del neuroimaging.

Obiettivi di NIfTI

- Miglioramento degli strumenti informatici esistenti ampiamente utilizzati nella ricerca sul neuroimaging
- Diffusione di strumenti informatici di neuroimaging e informazioni su di essi
- Approcci basati sulla comunità per risolvere problemi comuni, come la mancanza di interoperabilità di strumenti e dati
- Attività di formazione uniche e opportunità di sviluppo della carriera di ricerca per coloro che fanno parte delle comunità di utenti e sviluppatori di strumenti
- Ricerca e sviluppo della prossima generazione di strumenti informatici di neuroimaging

Principi operativi di NIfTI

- Guida attraverso una comunicazione stretta e continua con le comunità di utenti e sviluppatori di strumenti
- I diritti di proprietà intellettuale e il credito per gli strumenti rimangono con gli autori originali
- Coordinamento all'interno di NIfTI e tra altri programmi e attività correlati
- Meccanismo ottimale da utilizzare per ogni attività, attingendo a risorse sia intramurali che extramurali
- Coinvolgimento di più istituti presso NIH

Significato di NIfTI

- Strumenti informatici di neuroimaging più utili e utilizzabili
- Risorsa unica per strumenti informatici di neuroimaging e informazioni su di essi
- Ambiente per facilitare la convergenza su soluzioni comuni a problemi diffusi

- Massimizzare le opportunità scientifiche dalla ricerca sul neuroimaging

Introduzione alla libreria Nibabel

Questo pacchetto fornisce accesso in lettura +/- scrittura ad alcuni comuni formati di file medici e di neuroimaging, tra cui: ANALYZE (normale, SPM99, SPM2 e successivi), GIFTI , NIfTI1 , NIfTI2 , CIFTI-2 , MINC1 , MINC2 , AFNI BRIK/HEAD , MGH e ECAT , nonché Philips PAR/REC. Possiamo leggere e scrivere file di geometria, annotazione e morfometria di FreeSurfer. C'è un supporto molto limitato per DICOM. NiBabel è il successore di PyNIfTI .

Le varie classi di formato immagine danno accesso completo o selettivo alle informazioni di intestazione (meta) e l'accesso ai dati immagine è reso disponibile tramite gli array NumPy. NiBabel supporta una raccolta sempre crescente di formati di file di neuroimaging. Ogni formato di file ha le sue caratteristiche e peculiarità che devono essere curate per ottenere il massimo da esso. A tal fine, NiBabel offre sia un accesso di alto livello indipendente dal formato alle neuroimmagini, sia un'API con vari livelli di accesso specifico al formato a tutte le informazioni disponibili in un particolare formato di file. Le immagini Nibabel sono degli oggetti costituiti da un array contenente i dati dell'immagine propria, una matrice 4x4 chiamata matrice affine che mappa le coordinate dell'array in uno spazio di coordinate globali RAS+, e i metadati sottoforma di intestazione. Grazie a questa libreria è possibile ottenere i dati dell'immagine in formato array Numpy. L'array affine descrive la posizione dei dati dell'immagine in uno spazio di riferimento. Si ricorda che un voxel è un pixel con volume. Ogni pixel nell'immagine in scala di grigi della fetta rappresenta anche un voxel, perché questa immagine 2D rappresenta una fetta dell'immagine 3D con un certo spessore. L'array 3D è quindi anche un array voxel. Come per qualsiasi array, possiamo selezionare valori particolari tramite l'indicizzazione. Se dall'immagine 3D prendiamo una sezione 2D, ogni pixel nell'immagine in scala di grigi rappresenta anche un voxel perché l'immagine 2D è una fetta dell'immagine 3D con un certo spessore. Un array 3D è anche un array voxel. Le coordinate voxel non ci dicono quasi nulla sulla provenienza dei dati in termini di posizione dello scanner. Ad esempio, supponiamo di avere la coordinata voxel (26, 30, 16). Senza ulteriori informazioni non abbiamo idea se questa posizione dei voxel sia a sinistra o a destra del cervello, o provenga dalla sinistra o dalla destra dello scanner. Si tiene quindi traccia della relazione delle coordinate dei voxel con uno spazio di riferimento. In particolare, l'array affine memorizza la relazione tra le coordinate voxel nell'array di dati dell'immagine e le coordinate nello spazio di riferimento.

Lo spazio di riferimento del soggetto scanner

Cosa significa "spazio" nella frase "spazio di riferimento"? Lo spazio è definito da un insieme ordinato di assi. Per il nostro mondo spaziale 3D, è un insieme di 3 assi indipendenti.

Possiamo decidere quale spazio vogliamo utilizzare, scegliendo questi assi. Dobbiamo scegliere l'origine degli assi, la loro direzione e le loro unità.

Per cominciare, definiamo un insieme di tre *assi dello scanner* ortogonali.

Gli assi scanner

- L'origine degli assi è nell'isocentro del magnete. Questa è la coordinata (0, 0, 0) nel nostro spazio di riferimento. Tutti e tre gli assi passano per l'isocentro.
- Le unità per tutti e tre gli assi sono millimetri.
- Immagina un osservatore in piedi dietro lo scanner che guarda attraverso il foro del magnete verso l'estremità del letto dello scanner. Immagina una linea che viaggia verso l'osservatore attraverso il centro del foro del magnete, parallela al letto, con il punto zero nell'isocentro del magnete e valori positivi più vicini all'osservatore. Questa linea si chiama asse del *foro* dello *scanner*.
- Traccia una linea che viaggia dal pavimento della sala scanner attraverso l'isocentro del magnete verso il soffitto, ad angolo retto rispetto all'asse del foro dello scanner. 0 è all'isocentro e i valori positivi sono verso il soffitto. Questo si chiama asse *scanner-pavimento/soffitto*.
- Traccia una linea ad angolo retto rispetto alle altre due linee, viaggiando dalla sinistra dell'osservatore, parallela al pavimento, e attraverso l'isocentro del magnete alla destra dell'osservatore. 0 è all'isocentro e i valori positivi sono a destra. Questa si chiama asse *scanner-sinistra/destra*.

Se facciamo in modo che gli assi abbiano un ordine (scanner sinistra-destra; scanner pavimento-soffitto; scanner foro) allora abbiamo un insieme ordinato di 3 assi e quindi la definizione di uno *spazio* 3D. Chiamate il primo asse asse “X”, il secondo “Y” e il terzo “Z”. Una coordinata (x,y,z)=(10,−5,−3) in questo spazio si riferisce al punto nello spazio 10 mm a destra (dell'osservatore fittizio) dell'isocentro, 5 mm verso il pavimento dall'isocentro e 3 mm verso la base del letto dello scanner. Questo spazio di riferimento è talvolta noto come "scanner XYZ". Era lo spazio di riferimento standard per il predecessore di DICOM, chiamato ACR/NEMA 2.0.

Dallo scanner al soggetto

Se il soggetto è disteso nella posizione abituale per una scansione cerebrale, a faccia in su e con la testa per prima nello scanner, lo scanner sinistro/destro è anche l'asse sinistro-destro della testa del soggetto, il pavimento/soffitto dello scanner è la parte posteriore- l'asse anteriore della testa e il foro dello scanner è l'asse inferiore-superiore della testa.

A volte il soggetto non giace nella posizione standard. Ad esempio, il soggetto potrebbe trovarsi disteso con il viso rivolto a destra (in termini di asse scanner-sinistra/destra). In tal caso lo “scanner XYZ” non ci darà informazioni del sinistro e del destro del soggetto, ma solo dello scanner sinistro e destro. Potremmo preferire sapere dove siamo in termini di sinistra e destra del soggetto. Per affrontare questo problema, la maggior parte degli spazi di riferimento utilizza sistemi di coordinate dello scanner centrati sul soggetto o sul paziente. In questi sistemi, gli assi sono ancora gli assi scanner appena descritti, ma l'ordine e la direzione degli assi derivano dalla posizione del soggetto. Il sistema di coordinate dello scanner centrato sul soggetto più comune nel neuroimaging è chiamato "scanner RAS" (destro, anteriore, superiore). Qui gli assi dello scanner vengono riordinati e capovolti in modo che il primo asse sia l'asse dello scanner più vicino all'asse da sinistra a destra del soggetto, il secondo sia l'asse dello scanner più vicino all'asse antero-posteriore del soggetto e il terzo è l'asse dello scanner più vicino all'asse inferiore-superiore del soggetto. Ad esempio, se il soggetto era disteso volto a destra nello scanner, il primo asse (X) del sistema di riferimento sarebbe scanner-pavimento/soffitto, ma invertito in modo che i valori positivi siano verso il pavimento. Questo asse va da sinistra a destra nel soggetto, con valori positivi a destra. Il secondo asse (Y) sarebbe scanner-sinistra/destra (posteriore-anteriore nel soggetto) e l'asse Z sarebbe scanner-bore (inferiore-superiore).

Denominazione degli spazi di riferimento

La lettura dei nomi degli spazi di riferimento può creare confusione a causa dei diversi significati che gli autori usano per gli stessi termini, come "sinistra" e "destra".

Si usail termine "RAS" per indicare che gli assi sono (in termini di soggetto): da sinistra a destra; posteriore ad anteriore; e inferiore e superiore, rispettivamente. Sebbene sia comune chiamare questa convenzione "RAS", non è del tutto universale, perché alcuni usano "R", "A" e "S" in "RAS" per significare che gli assi *iniziano* a destra, anteriore, superiore del soggetto, anziché *terminare* a destra, anteriore, superiore. In altre parole, userebbero "RAS" per riferirsi a un sistema di coordinate che chiameremmo "LPI". Per sicurezza, si chiama la nostra interpretazione della convenzione RAS "RAS+", nel senso che destro, anteriore, superiore sono tutti valori positivi su questi assi.

Alcune persone usano anche "destro" per indicare il lato destro quando un osservatore guarda la parte anteriore dello scanner, dal piede al letto dello scanner. Sfortunatamente, questo significa che devi leggere attentamente le definizioni del sistema di coordinate se non hai familiarità con una particolare convenzione.

Le coordinate voxel sono nello spazio voxel

Anche le coordinate dei voxel sono in uno spazio. In questo caso lo spazio è definito dai tre assi voxel (primo asse, secondo asse, terzo asse), dove 0, 0, 0 è il centro del primo voxel nell'array e le unità sugli assi sono voxel. Le coordinate *voxel* sono quindi definite in uno spazio di riferimento chiamato *spazio voxel*.

La matrice affine come una trasformazione tra spazi

Abbiamo coordinate voxel (nello spazio voxel). Vogliamo ottenere le coordinate RAS+ dello scanner corrispondenti alle coordinate dei voxel. Abbiamo bisogno di una *trasformazione di coordinate* per portarci dalle coordinate voxel alle coordinate RAS+ dello scanner.

In generale, abbiamo alcune coordinate spaziali voxel (i,j,k), e vogliamo generare la coordinata spaziale di riferimento (x,y,z).
Immagina di averlo risolto e di avere una funzione di trasformazione delle coordinate f che accetta una coordinata voxel e restituisce una coordinata nello spazio di riferimento:

$$(x,y,z)=f(i,j,k)$$

'f' accetta una coordinata nello spazio di *input* e restituisce una coordinata nello spazio di *output*. Nel nostro caso lo spazio di input è lo spazio voxel e lo spazio di output è lo scanner RAS+. In teoria 'f' potrebbe essere una complicata funzione non lineare, ma in pratica sappiamo che lo scanner raccoglie i dati su una griglia regolare. Ciò significa che la relazione tra(i,j,k) e (x,y,z)è lineare (in realtà *affine*) e può essere codificato con trasformazioni lineari (in realtà affine) comprendenti traslazioni, rotazioni e zoom.
Il ridimensionamento (zoom) in tre dimensioni può essere rappresentato da una matrice diagonale 3 per 3. Ecco come ingrandire la prima dimensione di p, la seconda di q e la terza di r unità:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} pi \\ qj \\ rk \end{bmatrix} = \begin{bmatrix} p & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & r \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

Una rotazione in tre dimensioni può essere rappresentata come una *matrice di rotazione* 3 per 3. Ad esempio, ecco una rotazione di θ radianti attorno al terzo asse dell'array:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

Questa è una rotazione di ϕ radianti attorno al secondo asse dell'array:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

Una rotazione di γ radianti attorno al primo asse dell'array:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

Le matrici di zoom e rotazione possono essere combinate mediante moltiplicazione di matrici.

Ecco un ridimensionamento di p, q, r unità seguite da una rotazione di θ radianti attorno al terzo asse seguito da una rotazione di ϕ radianti attorno al secondo asse:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & r \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

Questo si può anche scrivere:

$$M = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & r \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = M \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

Questo potrebbe essere ovvio perché la moltiplicazione matriciale è il risultato dell'applicazione di ciascuna trasformazione a turno sulle coordinate prodotte dalla trasformazione precedente. Combinare le trasformazioni in un'unica matrice MM funziona perché la moltiplicazione matriciale è associativa - ABCD=(ABC) D.

Una traslazione in tre dimensioni può essere rappresentata come un vettore di lunghezza 3 da sommare alla coordinata di lunghezza 3. Ad esempio, una traduzione di ‘a’ unità sul primo asse, ‘b’ sul secondo e ‘c’ sul terzo potrebbe essere scritto come:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Possiamo scrivere la nostra funzione f come una combinazione di moltiplicazione della matrice 3 per 3 rotazione / matrice di zoom M seguito dall'aggiunta di un vettore di traslazione 3 per 1 (a,b,c)

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = M \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Potremmo registrare i parametri necessari per f come matrice 3 per 3, M e il vettore 3 per 1 (a,b,c)(a,b,c).
In effetti, l' *array affine di* immagini 4 per 4 include esattamente queste informazioni. Se mi,j è il valore nella riga i colonna j di matrice M, quindi la matrice affine all'immagine A è:

$$A = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & a \\ m_{2,1} & m_{2,2} & m_{2,3} & b \\ m_{3,1} & m_{3,2} & m_{3,3} & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perché la riga in più di [0,0,0,1]? Abbiamo bisogno di questa riga perché abbiamo riformulato la combinazione di rotazioni/zoom e traslazioni come una trasformazione in *coordinate omogenee*. Questo è un trucco che ci permette di mettere la parte di traslazione nella stessa matrice delle rotazioni/zoom, in modo che sia le traslazioni che le rotazioni/zoom possano essere applicate per moltiplicazione di matrici. Per farlo funzionare, dobbiamo aggiungere un ulteriore 1 ai nostri vettori di coordinate di input e output:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & a \\ m_{2,1} & m_{2,2} & m_{2,3} & b \\ m_{3,1} & m_{3,2} & m_{3,3} & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix}$$

Ciò si traduce nella stessa trasformazione dell'applicazione M e (a,b,c) separatamente. Un vantaggio delle trasformazioni di codifica, in questo modo, è che possiamo combinare due insiemi di [rotazioni, zoom, traslazioni] mediante moltiplicazione matriciale delle due matrici affini corrispondenti. In pratica, sebbene sia comune combinare trasformazioni 3D utilizzando matrici affini 4 per 4, di solito *applichiamo* le trasformazioni scomponendo la matrice affine nella sua componente M matrice e (a,b,c) vettore e facendo:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = M \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Finché l'ultima riga della matrice 4 x 4 è [0,0,0,1], l'applicazione delle trasformazioni in questo modo equivale matematicamente all'utilizzo del modulo 4 x 4 completo, senza l'inconveniente di aggiungere l'1 in più ai nostri vettori di input e output.

L'inverso dell'affine fornisce la mappatura dallo scanner al voxel

Gli array affini che abbiamo descritto finora hanno un'altra piacevole proprietà: di solito sono invertibili. Come sapete, l'inverso di una matrice A è la matrice A^{-1} tale che $I=A^{-1}A$, dove I è la matrice identità. Si può scrivere:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = A \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix}$$

$$A^{-1} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = A^{-1}A \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} = A^{-1} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Ciò significa che l'inverso della matrice affine fornisce la trasformazione dalle coordinate RAS+ dello scanner alle coordinate voxel nei dati dell'immagine.

Ora immaginiamo di avere un array affine A per someones_epi.nii.gz e array affine B per someones_anatomy.nii.gz. A fornisce la mappatura dai voxel nell'array di dati dell'immagine someones_epi.nii.gz ai millimetri nello scanner RAS+. B fornisce la mappatura dai voxel nell'array di dati dell'immagine someones_anatomy.nii.gz allo *stesso* scanner RAS+. Ora diciamo che abbiamo una particolare coordinata voxel(i,j,k) nell'array di dati di someones_epi.nii.gz, e vogliamo trovare il voxel someones_anatomy.nii.gz che si trova nella stessa posizione spaziale. Chiama questa coordinata voxel corrispondente(i',j',k'). Per prima cosa

applichiamo la trasformata da someones_epi.nii.gz voxel allo scanner RAS+ (A) per poi applicare la trasformazione dallo scanner RAS+ ai voxel in someones_anatomy.nii.gz(B^{-1}):

$$\begin{bmatrix} i' \\ j' \\ k' \\ 1 \end{bmatrix} = B^{-1}A \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix}$$

Le immagini Nibabel utilizzano sempre le coordinate di output RAS+, indipendentemente dalle coordinate di output preferite del formato sottostante. Ad esempio, convertiamo le affini per le immagini DICOM in coordinate RAS+ in uscita invece delle coordinate LPS+. Abbiamo scelto questa convenzione perché è la più popolare nel neuroimaging; ad esempio, è lo standard utilizzato dai formati NIfTI e MINC .

Nibabel non applica un particolare spazio RAS+. Ad esempio, le immagini NIfTI contengono codici che specificano se le mappe sono affini allo scanner o MNI o allo spazio Talairach RAS+. Per il momento, bisogna consultare le specifiche di ciascun formato per trovare a quale spazio RAS+ è associato l'affine.

Metodi e risultati ottenuti

Si inizia con scrivere un software in python che permetta di tagliare l'immagine in francobollini 5x5 e ogni francobollo classificarlo in base al pixel centrale (osso, polmone, ecc.). Nell'immagine segmentata si ha che ogni tessuto ha un livello classificazione. Si hanno quindi 7 livelli di classificazione (0-6). Si inizia scrivendo lo script in python che permette di caricare l'immagine da segmentare e quella già segmentata scaricate dal database TCIA, e di estrarre da esse gli array di dati dalle immagini. Il formato delle immagini è spiegato nella parte introduttiva. Un oggetto immagine nibabel è l'associazione di tre cose: un array N-D contenente i dati dell'immagine; una matrice (4x4) affine che mappa le coordinate dell'array alle coordinate in uno spazio di coordinate mondiali RAS+ (sistemi di coordinate e affini); metadati dell'immagine sotto forma di intestazione. Ciò che si vuole estrarre è soltanto l'array contenente i dati delle immagini e inserirlo in una matrice numpy. Il formato delle immagini con estensione .nii.gz si riferisce al NIfTI descritto nell'introduzione. Si importano quindi le librerie nibabel, os, numpy e random, che è utilizzata in seguito.

```
import os
import random
import numpy as np
import nibabel as nib
tac_image = os.path.join("images", 'volume-55.nii.gz') #immagine da segmentare
label = os.path.join("images", 'labels-55.nii.gz') #immagine segmentata
np.set_printoptions(precision=2, suppress=True)
img = nib.load(tac_image)
data = img.get_fdata()          #questo metodo restituisce un array numpy che
                                 #contiene i soli dati dell'immagine
img_label = nib.load(label)
label_array = img_label.get_fdata()
```

Una volta fatto ciò, andiamo a scrivere il codice che ci permette di selezionare le dimensioni del quadrato dei pixel da estrarre:

```
print('insert x value (not a pair value)')
x = int(input())  #lato del quadrato; per semplicità si potrebbe già impostare 5
                   #come dimensione
if x % 2 == 0:
    print('pair value not admitted')
    exit()
print('insert y value (not a pair value)')
y = int(input()) #lato del quadrato
if y % 2 == 0:
    print('pair value not admitted')
    exit()
#print('insert the index of the slice on z axis to extract')
#z = 0
x_i = (x - 2)
x_f = (x + 3)
y_i = (y - 2)
```

Si sono impostati i limiti superiori ed inferiori dei cicli for per l'estrazione dei francobolli. Per evitare i problemi ai contorni dell'immagine, scegliamo per semplicità di non considerare quei pixel che ci farebbero uscire dai bordi con l'acquisizione del quadrato:

```
y_f = (y + 3)
l = ((x-1)/2)+1
i_sup=data.shape[0]-l #limiti sup e inf per i problemi di bordo
i_inf=(x-1)/2
j_inf=(y-1)/2
m = ((y-1)/2)+1
```

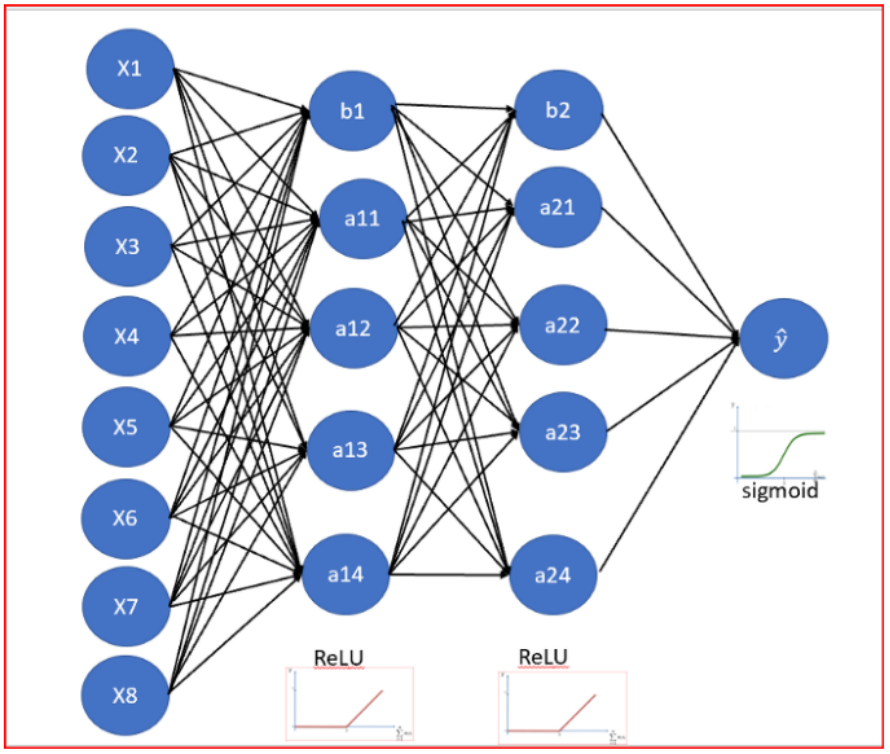
I quadrati estratti sotto forma di array, per essere esportati, vengono convertiti in modo da avere tutti gli elementi su una riga e tante colonne quanti sono gli elementi del quadrato. Prima di fare ciò si aggiunge a questo array come ultimo elemento il pixel centrale dell'immagine segmentata. Gli elementi di ogni riga sono separati da una virgola, in modo da poter essere interpretati dalla rete neurale. Con il seguente codice apriamo un file di testo esterno, dove inseriamo gli elementi dei quadratini(francobollini) estratti dall'immagine su ogni riga, separati da ','. Si sceglie di estrarre circa 30000 francobollini in modo da avere un adeguato data set. L'ordine di estrazione dei quadratini è casuale su tutto il volume dell'immagine.

```
with open('training_set_prova.txt', 'w') as outfile: #metodo per scrivere
                                    #stringhe su righe separate
for k in range(0,400):
    z = random.randint(0, k_sup)
    contatore = 1
    for i in range(0,500):
        x_c=random.randint(i_inf, i_sup)  #pixel centrale
        y_c=random.randint(j_inf, j_sup)  #pixel centrale
        x_i=int((x_c-((x-1)/2)))
        x_f=int((x_c+(((x-1)/2)+1)))
        y_i=int((y_c-((y-1)/2)))
        y_f=int((y_c+(((y-1)/2)+1)))
        square = data[x_i:x_f, y_i:y_f, z] #estrazione del quadratino fissato z
        label_px = label_array[x_c, y_c, z] #estrazione pixel centrale
```

```
                                        #dall'immagine segmentata
if label_px != 0 and contatore % 25 != 0: #codice per estrarre uno 0
                                            #ogni 25 iterazioni
    q = square
    q = np.append(square, label_px) #aggiunge il pixel
                                    #dell'immagine segmentata
    q = q.reshape(1, ((x * y) + 1)) #imposta le dimensioni volute
    np.savetxt(outfile, q, delimiter=',', fmt='%-7.2f')
elif contatore % 25 == 0:
    r = square
    r = np.append(square, label_px)
    r = r.reshape(1, ((x * y) + 1))
    np.savetxt(outfile, r, delimiter=',', fmt='%-7.2f')
contatore += 1
```

L'ultima parte del codice corrisponde al controllo dei pixel estratti per bilanciare il numero dei pixel con livello zero che viene estratto nell'immagine segmentata. Si prende un pixel a livello 0 ogni 25 cicli. Questo perché l'immagine segmentata risulta avere una quantità sproporzionata di pixel con livello zero.

La rete neurale che si utilizza per la classificazione è ispirata a una rete creata per un'applicazione simile(https://medium.datadriveninvestor.com/building-neural-network-using-keras-for-classification-3a3656c726c1).



Questa è l'architettura della rete neurale di partenza che abbiamo adattato alla nostra applicazione.

Siccome questa rete è ideata per un problema con 8 input e un output è necessario effettuare alcuni cambiamenti nella sua struttura. La nostra rete neurale creata ha in ingresso 25 elementi e 1

uscita ed è costituita da un primo livello nascosto con 512 nodi, un secondo livello nascosto con 256 nodi, e un livello di uscita con 6 nodi. ReLu sarà la funzione di attivazione per i livelli nascosti. Poiché si tratta di un problema di classificazione multipla, utilizzeremo Softmax come funzione di attivazione. Kernel è la matrice dei pesi. L'inizializzazione del kernel definisce il modo per impostare i pesi casuali iniziali dei livelli Keras. L'inizializzatore normale casuale genera tensori con una distribuzione normale. Per la distribuzione uniforme, possiamo usare gli inizializzatori uniformi casuali. Keras fornisce inizializzatori multipli sia per kernel o pesi, sia per unità di bias. Viene utilizzato Colab per implementare la rete. Innanzi tutto, si carica il data set nell'ambiente virtuale di Colab. Per prima cosa importiamo le librerie di base pandas e numpy insieme alle librerie di visualizzazione dei dati matplotlib e seaborn.

```
import tensorflow
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split
from keras import Sequential
from keras.layers import Dense
import tensorflow as tf
import keras.utils
from numpy import array
```

Ora leggiamo il file e carichiamo i dati in un dataset DataFrame

```
data = pd.read_csv('data_set.txt', sep=",")
```

Per comprendere meglio i dati, esaminiamo i dettagli del set di dati. Dobbiamo capire le colonne e il tipo di dati associati a ciascuna colonna.

```
print(data.head(10))
```

dobbiamo controllare che tipo di dati abbiamo nel set di dati.

```
print(data.describe(include='all'))
```

Ora che comprendiamo i dati, creiamo le funzionalità di input e le variabili di destinazione e prepariamo i dati per l'immissione nella nostra rete neurale mediante la pre elaborazione dei dati.

```
X= data.iloc[:,0:25]
y= data.iloc[:,25]
```

Poiché le nostre caratteristiche di input sono su scale diverse, dobbiamo standardizzare l'input.

```
X = tf.keras.utils.normalize(X,axis=1)
```

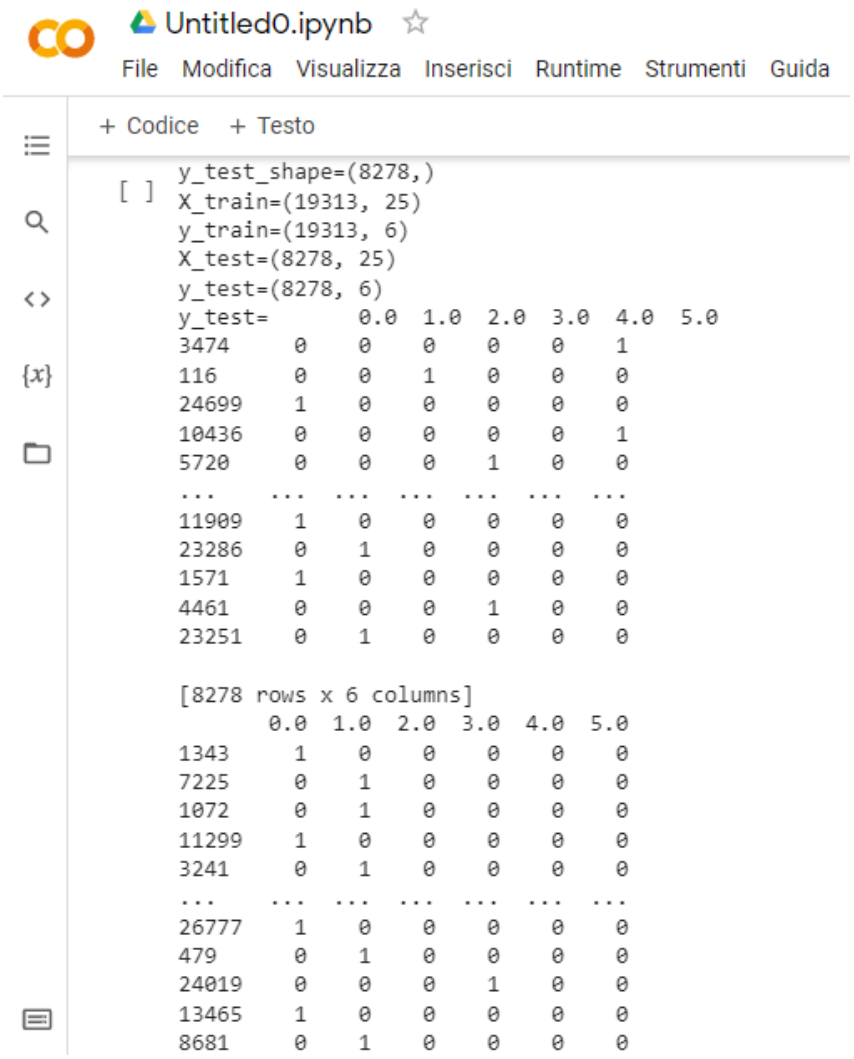
Ora dividiamo le funzionalità di input e le variabili di destinazione in set di dati di addestramento e set di dati di test. il nostro set di dati di prova sarà il 30% dell'intero set di dati.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
print(f'y_test={y_test}')
print(f'X_test={X_test}')
print(f'y_test_shape={y_test.shape}')
```

A volte nei set di dati, incontriamo colonne che contengono caratteristiche categoriali (valori stringa), ad esempio il parametro Gender avrà parametri categoriali come Male , Female . Queste etichette non hanno un ordine di preferenza specifico e inoltre poiché i dati sono etichette di stringhe, il modello di apprendimento automatico non può funzionare su tali dati. Un approccio per risolvere questo problema può essere la codifica delle etichette in cui assegneremo un valore numerico a queste etichette, ad esempio Maschio e Femmina mappati su 0 e 1 . Ma questo può aggiungere bias nel nostro modello poiché inizierà a dare una preferenza maggiore al parametro Female come 1>0 e idealmente entrambe le etichette sono ugualmente importanti nel set di dati. Per affrontare questo tipo di problema si utilizza la tecnica One Hot Encoding. Nel nostro esempio la situazione è analoga e le etichette sono i livelli da 0 a 6 dell'immagine classificata. Il metodo pandas.get_dummies() viene utilizzato per la manipolazione dei dati. Converte i dati categoriali in variabili fittizie o indicatori.

```
y_test = pd.get_dummies(y_test)
y_train = pd.get_dummies(y_train)
print(f'X_train={X_train.shape}')
print(f'y_train={y_train.shape}')
print(f'X_test={X_test.shape}')
print(f'y_test={y_test.shape}')

print(f'y_test={y_test}')
print(y_train)
```

Il risultato di tale manipolazione è il seguente:



Abbiamo pre-elaborato i dati e ora siamo pronti per costruire la rete neurale.
Usiamo keras per costruire la nostra rete neurale. Ci sono due tipi principali di modelli disponibili in Keras: Sequential e Model. Usiamo il modello Sequential per costruire la nostra rete neurale.
Usiamo la libreria Dense per creare livelli di input, nascosti e output di una rete neurale.

```
from keras.models import Sequential

classifier = Sequential()

#First Hidden Layer
classifier.add(Dense(512, activation='relu', kernel_initializer='random_normal',
 input_dim=25))

classifier.add(Dense(256, activation='relu', kernel_initializer='random_normal')
)

#Output Layer
classifier.add(Dense(6, activation='softmax', kernel_initializer='random_normal'
))
```

Una volta creati i diversi livelli, compiliamo la rete neurale. Poiché si tratta di un problema di classificazione multipla, utilizziamo categorical_crossentropy per calcolare la funzione di perdita tra l'output effettivo e l'output previsto. Per ottimizzare la nostra rete neurale utilizziamo Adam. Adam sta per stima adattativa del momento. Adam è una combinazione di RMSProp + Momentum. Momentum tiene conto dei gradienti passati per appianare la discesa del gradiente. Usiamo l'accuratezza come metrica per misurare le prestazioni del modello.

```
classifier.compile(optimizer ='adam',loss='categorical_crossentropy', metrics =[
'accuracy'])
```

Ora adattiamo i dati di addestramento al modello che abbiamo creato. usiamo un batch_size di 15. Ciò implica che usiamo 15 campioni per aggiornamento del gradiente. Iteriamo più di 150 epoche per addestrare il modello. Un'epoca è un interazione sull'intero set di dati.

```
classifier.fit(X_train,y_train, batch_size=15, epochs=150)
```

Dopo 100 epoche otteniamo una precisione di circa l'80%:

```
Epoch 148/150
1288/1288 [==============================] - 4s 3ms/step - loss: 0.3961 - accuracy: 0.8274
Epoch 149/150
1288/1288 [==============================] - 4s 3ms/step - loss: 0.3912 - accuracy: 0.8320
Epoch 150/150
1288/1288 [==============================] - 4s 3ms/step - loss: 0.3848 - accuracy: 0.8342
<keras.callbacks.History at 0x7fa1058438d0>
```

Possiamo anche valutare il valore di perdita e i valori delle metriche per il modello in modalità test utilizzando la funzione di valutazione.

```
eval_model=classifier.evaluate(X_train, y_train)
eval_model
```

```
604/604 [==============================] - 1s 2ms/step - loss: 0.3621 - accuracy: 0.8418
[0.362142413854599, 0.8417646288871765]
```

Ora prevediamo l'output per il nostro set di dati di test. Se la previsione è maggiore di 0,5, l'uscita è 1 altrimenti l'uscita è 0.

```
y_pred=classifier.predict(X1)
y_pred =(y_pred>0.5)
```

Controlliamo ora l'accuratezza sul set di dati di prova.

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test.values.argmax(axis=1), y_pred.argmax(axis=1))
print(cm)
```

```
[[1646   84   65   76   18  201]
 [ 355 1293    2    0  102  240]
 [  72    1  115    0    0   37]
 [ 653    0    0  597    0    0]
 [ 109   60    0    0  126   82]
 [ 337  192   44    1   63 1707]]
```

Data la diagonalità della Confusion matrix possiamo dire che la rete riproduce l’uscita desiderata con una buona probabilità.