Titolo progetto : Rete Neurale

Nome : Leonardo
Cognome : Lucà
Matricola : 542416

Descrizione:

Verrà implementata una rete neurale che contiene una lista di layer, dove ogni layer contiene una lista di neuroni. Ogni neurone contiene un valore di attivazione , e due liste di pesi, una che contiene i pesi in input ed un’altra che contiene i pesi in output.

Inoltre per testare la rete neurale è’ stato creato l’ADT ImageRecognizer che implementa una rete neurale capace di riconoscere delle immagini di numeri scritti a mano.

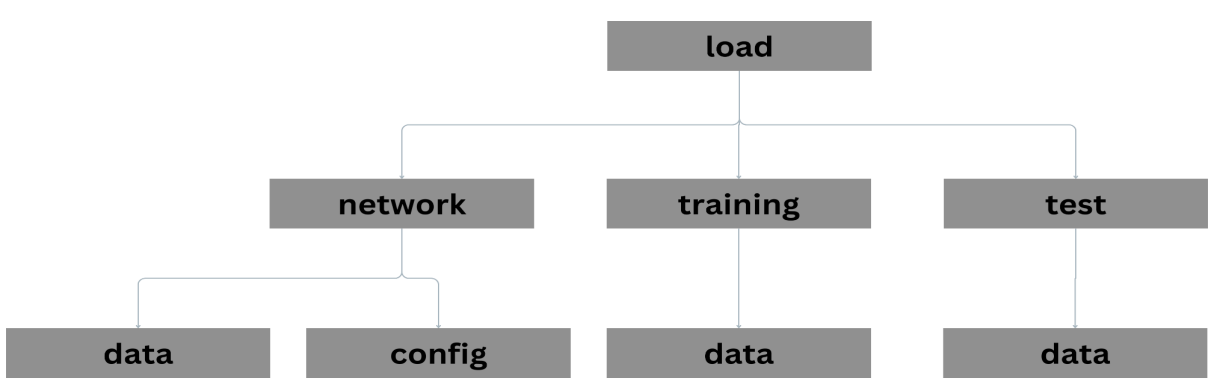
Descrizione dell’interfaccia:

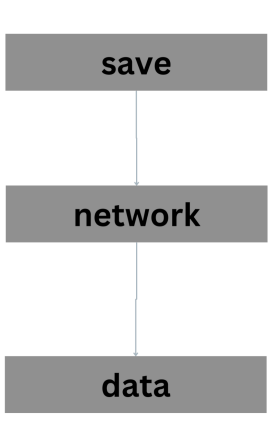
Per poter configurare la rete neurale verrà realizzata un’interfaccia impostata in modo da prendere in input dei comandi e di associare ad ogni comando una operazione da eseguire sulla rete neurale.
Ogni comando è formato da tre tag, un tag primario che denota la funzione del comando(load , save , show e config) e due altri comandi, separati da un trattino alto (-) che specificano esattamente a cosa viene applicato il comando , come nell’esempio :

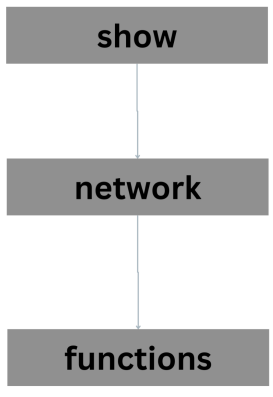
```
-->load network-config netConfig.txt
```

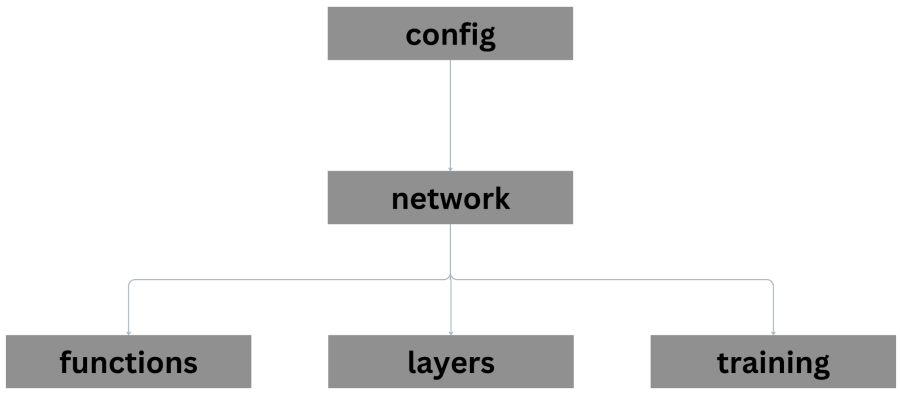
Come si può vedere dalla foto, attraverso il comando : “load network-config (config file)” è anche possibile caricare un file contenente già dei comandi che corrispondono ad una particolare configurazione della rete neurale.

Di seguito verranno riportati dei grafici ad albero che rappresentano tutte le combinazioni relative ad i comandi l**oad , save , sehow e config.**









Descrizione degli ADT :

ADT *List*
implementa una lista generica,doppiamente concatenata , con puntatori sia a testa che a coda e con inserimento in testa.

```c
//nodo della lsita doppiamente concatenata
typedef struct Node{

    void* info;//contenuto informativo generico
    struct Node* next;//handle al nodo successivo
    struct Node* prev;//handle al nodo precedente

}Node;

//getters e setters per Node---------------
void* getInfo(Node* node);
void setInfo(Node* node,void* info);

Node* getNext(Node* node);
void setNext(Node* node,Node* next);

Node* getPrev(Node* node);
void setPrev(Node* node,Node* prev);
//----------------------------------------



//getters e setters per Node---------------
typedef void(*freeFunction)(Node*);//funzione di deallocazione
typedef void(*saveFunction)(Node* node,FILE* file);


typedef struct List{

    unsigned int size;//dimensione logica della lista

    size_t typeSize;//spazio occupato dal contenuto informativo di ogni singolo nodo

    freeFunction freeFunc;//puntatore a funzione per eliminare lo spazio di un singolo nodo

    saveFunction saveFunc;//puntatore a funzione per salvare una lista in un file binario

    Node* head;//handle alla testa della lista

    Node* tail;//handle alla coda della lista

}List;

typedef void(*loadFunction)(List* list,short int dim,FILE* file);/*load list si trova fuori dalla struct list, poichè
nel file binario non viene salvata la funzione per caricare la lista*/-
```

Continuazione ADT List.

```c
List* createList(size_t typeSize,freeFunction freeFunc);//crea un nuovo ADT di tipo List
void freeList(List* list);//libera lo spazio allocato dalla lista
void freePrimitive(Node* node);//libera lo spazio di un qualsiasi dato di tipo primitivo(char,int,fload,double e rispettivi array) non allocato nell'heap

int add(List* list,void* element);//aggiunge un nuovo elemento in testa alla lista
int isEmpty(List* list);//ritorna 1 se la lista è vuota , altrimenti -1

void loopH(List* list,void(*func)(Node* node));//scorre la lista dalla testa alla coda e per ogni elemento esegue una funzione
void loopT(List* list,void(*func)(Node* node));//loopa la lista dalla coda alla testa e per ogni elemento esegue una funzione


//getter e setter-------------------------------------------------------

Node* getHead(List* list);//ritorna la testa della lista
void setHead(List* list,Node* head);

Node* getTail(List* list);//ritorna la coda della lista
void setTail(List* list,Node* tail);

unsigned int getSize(List* list);
void setSize(List* list,unsigned int size);

size_t getTypeSize(List* list);
void setTypeSize(List* list,size_t typeSize);

freeFunction getFreeFunction(List* list);
void setFreeFunction(List* list,freeFunction freeFunc);

saveFunction getSaveFunction(List* list);
void setSaveFunction(List* list,saveFunction saveFunc);
//-----------------------------------------------------------------------------


void printList(List* list,void(*printFunc)(Node* node));//metodo generico per stampare una lista
int saveList(List* list,FILE* file);
List* loadList(FILE* file,loadFunction loadFunc,freeFunction freeFunc);
void setSaveFunc(List* list,saveFunction saveFunc);

void loadPrimitive(List* list,short int dim,FILE* file);


//---WRAPPER FUNCTIONS----
void freeLists(Node* node);//libera lo spazio allocato da una lista di liste
void saveLists(Node* node,FILE* file);//salva in un file binario una lista di liste

double getDouble(Node* node);//ritorna l'info di un nodo della lista effettuando un cast a double
void setDouble(Node* node,double info);//setta il valore di un nodo della lista effettuando un cast a double
void printDouble(Node* node);//..
void saveDouble(Node* node,FILE* file);

float getFloat(Node* node);//....
void setFloat(Node* node,float info);//...

int* toIntArray(List* list);//trasforma una lista di interi in un array di interi
int getInt(Node* node);//...
void setInt(Node* node,int info);//...

char* getString(Node* node);//ritorna una stringa
char* getStringAt(List* list,unsigned int position);//ritorna una stringa nella posizione desiderata
void printString(Node* node);//stampa una stringa
```

ADT *Neuron*
Implementa un singolo neurone.

```c
typedef struct Neuron{

    double activation;//valore di attivazione del neurone/
    double z;//valore del neurone prima della funzione di attivazione
    double error;//valore di errore del neurone o gradiente generale del neurone(per la backpropagation)
    double bias;//bias del neurone
    List* inputWeights;//lista di pesi in input del neurone
    List* outputWeights;//lista di pesi in output del neurone

}Neuron;

//crea un nuvo neurone assegnando bias e attivazione e creando una lista di pesi vuota
Neuron* createNeuron(double activation,double bias);
void deleteNeuron(Neuron* neuron);//libera lo spazio allocato da un neurone

int initInputWeights(Neuron* neuron,int prevLayerSize);/*inizializza la lista di pesi ad un valore scelto
inoltre ritona 1 se tutto è andato a buon fine, altrimenti ritorna -1*/
void initOutputWeights(List* layers);

void freeInputWeights(Node* node);//libera lo spazio allocato dalle liste di pesi
void freeOutputWeights(Node* node);

void printNeuron(Node* node);//stampa un neurone
```

Continuazione ADT *Neuron*

```c
//getters e setters
double getActivation(Neuron* neuron);//ritorna il valore di attivazione
void setActivation(Neuron* neuron,double newActivation);//setta l'attivazione

double getBias(Neuron* neuron);//ritorna il bias
void setBias(Neuron* neuron,double newBias);//setta il bias

List* getInputWeights(Neuron* neuron);//ritorna la lista di pesi
List* getOutputWeights(Neuron* neuron);

void setOutputWeights(Neuron* neuron,List* newWeights);
void setInputWeights(Neuron* neuron,List* newWeights);//setta i pesi

double getZ(Neuron* neuron);
void setZ(Neuron* neuron,double newZ);

double getError(Neuron* neuron);
void setError(Neuron* neuron,double newError);

void updateError(Neuron* neuron,double newError);

double getInWeightValue(Node* weight);//ritorna il contenuto informativo di un nodo nella lista di pesi(double)
void setInWeightValue(Node* weight,double newValue);//setta il contenuto informativo di un nodo nella lista di pesi

double getOutWeightValue(Node* weight);
void setOutWeightValue(Node* weight,double newValue);

void printOutWeight(Node* node);
void printInWeight(Node* node);
```

ADT *Layer*
Implementa un singolo layer della rete neurale come una lista di Neuron.

```c
typedef List* Layer;

void saveLayer(Node* node,FILE* file);
void loadLayer(List* list,short int dim,FILE* file);
void freeLayer(Node* node);//dealloca un singolo layer(che contiene neuroni)

void printLayer(Node* node);//stampa un singolo layer

Neuron* getLayerInfo(Node* layerNode);//ritorna il contenuto informativo di un nodo di un singolo layer
void setLayerInfo(Node* layerNode,Neuron* newInfo);//setta il contenuto informativo di un nodo di un singolo layer
```

ADT *Layers*
Implementa una lista di Layer, che rappresenta il cuore della rete neurale.

```c
//tipo di dato Layers che rappresenta una lista di liste di neuroni
typedef List* Layers;

//metodo per allocare lo spazio e inizializzare i tutti i layer della rete neurale
Layers createLayers(double** errros,unsigned int nLayers,unsigned int nInputs,unsigned int nOutputs,unsigned int nHiddens,double defValue);
Layers createCustomLayers(double** errors,unsigned int nLayers,unsigned int* layersData,double defValue);

void saveLayers(Node* node,FILE* file);
void loadLayers(List* list,short int dim,FILE* file);
void initLayers(double** errors,Layers layers);//funzione utilizzata per inizializzare i layers dopo il
//cariamento da file
void freeLayers(Node* node);//dealloca la lista di layer
void printLayers(Layers layers);//stampa la lista di layer



List* getLayersInfo(Node* layer);//ritorna il contenuto informativo di un nodo della lista di layer
void setLayersInfo(Node* layer,List* newInfo);//setta il contenuto informativo di un nodo della lista di layer

void forwardPassLayers(NetFunctions functions,Node* node);
```

ADT *NeuralNet*
Implementa una rete neurale.

```c
typedef enum{//enumeratore per identificare le due modalità di training (mini batch ed online)
    MINI_BATCH,
    ONLINE
}Mode;

typedef void(*ChangeError)(Neuron* neuron,double newValue);/*variabile utilizzata per
cambiare l'operazione da effettuare sull'errore di ciascun neurone ovvero update o set, viene effettuato
un update se si sta utilizzando lo stochastic-gradient descent che prevede l'utilizzo di un mini batch,
altrimenti se si utilizza il metodo classico online,allora si setta un nuovo errore volta per volta*/


//tipo di dato che definisce la rete neurale
typedef struct NeuralNet{

    unsigned int numberOfNeurons;//numero totale di neuroni

    Layers layers; //lista di layer

    NetFunctions functions; //funzioni della rete neurale

    double learnRate; //learning rate per implementare l'algoritmo Gradient Descent

    double error;//risultato di della cost function,nonchè l'errore tra il risultato aspettato e quello ottenuto

    double** errors;/*memorizza tutti gli errori di ogni neurone, in modo da aggiornarli
                    facilmente quando viene implementato il mini-batch stochastic gradient descent
                    */
}NeuralNet;
```

Continuazione ADT *NeuralNet*

```
int createNet(NeuralNet** net);//ritorna 1 se va a buon fine altrimenti , -1
int initDefNet(NeuralNet* net,unsigned int nLayers,unsigned int nInputs,unsigned int nOutputs,unsigned int nHiddens,double defValue);
int initCustomNet(NeuralNet* net,unsigned int nLayers,unsigned int* layersData,double defValue);
int fromFileNet(NeuralNet* net,Layers layers,double learnRate,unsigned int numberOfNeurons,NetFunctions functions);


void deleteNet(NeuralNet* net);//libera lo spazio allocato dalla rete neurale
List* getInputLayer(NeuralNet* net);
List* getOutputLayer(NeuralNet* net);
void printNet(NeuralNet* net);/*stampa il contenuto della rete neurale nel formato :
ogni layer : {
                a1.50(Attivazione del neurone) b1.59(Bias del neurone) {.....(lista di pesi)...}
                a...
                a..
        }
*/
List* forwardPass(NeuralNet* net,List* input);/*utilizzando lo stato corrente della rete neurale,
(quindi mantenendo attivazioni,pesi e bias), viene "dato in pasto" un input alla rete neurale e
attraverso l'algoritmo di attivazione viene ritornato l'handle
all'output layer della rete neurale.
*/
void calcHiddenLayerError(Node* scanNeurons,Node* prevLayer,NetFunctions functions,ChangeError errorFunc);
void calcOutLayerError(Node* scanNeurons,Node* scanValues,NetFunctions functions,ChangeError errorFunc);

void backpropagate(NeuralNet* net,List* expValues,Mode mode);
void updateWeights(NeuralNet* net);

void printOutputLayer(NeuralNet* net);
//funzioni helper che non fanno parte dell'interfaccia vera e propria , utilizzate per rendere il codice più modulare
double calcError(NetFunctions functions,List* layer,List* expValue);//calcola la media degli errori, expValue deve essere una lista di double

//getters e setters--------------

unsigned int getNumberOfNeurons(NeuralNet* net);
void setNumberOfNeurons(NeuralNet* net,unsigned int nn);

Layers getLayers(NeuralNet* net);
void setLayers(NeuralNet* net,Layers layers);

NetFunctions getNetFunctions(NeuralNet* net);
void setNetFunctions(NeuralNet* net,NetFunctions functions);

double getLearnRate(NeuralNet* net);
void setLearnRate(NeuralNet* net,double learnRate);

double getNetError(NeuralNet* net);
void setNetError(NeuralNet* net,double error);

double** getErrors(NeuralNet* net);
double* getErrorsAt(NeuralNet* net,unsigned int index);
void setErrors(NeuralNet* net,double** errors);
```

ADT *ImageRecognizer*

Implementa una rete neurale che può essere allenata o con lo *stochastic gradient-descent* che prevede l'utilizzo di un *mini batch,* oppure può anche essere allenata con il metodo classico , online.

```
#define NUMTEST 10000
#define NUMTRAIN 60000

typedef enum{
    true,
    false,
    error
}boolean;

typedef struct{
    TrainData trainData;//dati di training

    Image** testImages;//immagini per effettuare il testing
    Image** trainImages;//immagini per effettuare il training

    NeuralNet* net;//rete neurale

    unsigned int batchSize;//batch size se si utilizza il STGD

    unsigned int numberOfLayers;//numero di layer per effettuare l'inizializzazione della rete neurale

    boolean customConf;//valore booleano per sapere che tipo di configurazione si sceglie (custom o default)

    unsigned int* layerConf;//array contenente la configurazione dei layer

    Mode trainingMode;//modalità di training
}ImageRecognizer;

//getters e setters---------

TrainData getTrainData(ImageRecognizer* imgRec);
void setTrainData(ImageRecognizer* imgRec,TrainData trainData);

Image** getTestImages(ImageRecognizer* imgRec);
void setTestImages(ImageRecognizer* imgRec,Image** testImages);

Image** getTrainImages(ImageRecognizer* imgRec);
void setTrainImages(ImageRecognizer* imgRec,Image** trainImages);

NeuralNet* getNet(ImageRecognizer* imgRec);
void setNet(ImageRecognizer* imgRec,NeuralNet* net);

unsigned int getBatchSize(ImageRecognizer* imgRec);
void setBatchSize(ImageRecognizer* imgRec,unsigned int batchSize);

unsigned int getNumLayers(ImageRecognizer* imgRec);
void setNumLayers(ImageRecognizer* imgRec,unsigned int numLayers);

boolean getCustomConf(ImageRecognizer* imgRec);
void setCustomConf(ImageRecognizer* imgRec,boolean customConf);

unsigned int* getLayerConf(ImageRecognizer* imgRec);
void setLayerConf(ImageRecognizer* imgRec,unsigned int* layerConf);

Mode getTrainMode(ImageRecognizer* imgRec);
void setTrainMode(ImageRecognizer* imgRec,Mode trainMode);
```

Continuazione ADT ImageRecognizer

```c
int createImageRecognizer(ImageRecognizer** imgRec);//ritorna 1 se tutto va a buon fine, -1 altrimenti
int initImageRecognizer(ImageRecognizer* imgRec);//ritorna 1 se va bene e -1 altrimenti

int saveImageRecognizer(ImageRecognizer* imgRec,char* filePath);//ritorna 1 se va tutto bene, -1 se il file non è trovato e -2 se la rete neurale non è stata allocata
int loadImageRecognizer(ImageRecognizer* imgRec,char* filePath);

int trainImageOnline(ImageRecognizer* imgRec);//ritorna -1 se qualcosa va storto , -2 se viene chiamata la funzione senza aver prima inzializzato la rete neurale
int trainImageMiniBatch(ImageRecognizer* imgRec);

boolean isPredictionCorrect(unsigned int expected,List* response);//verifica se l'output è uguale alla label
unsigned int predict(List* response);//ritorna l'output della rete neurale

int testImageRecognizer(ImageRecognizer* imgRec);//effettua il testing della rete neurale

void freeImageRecognizer(ImageRecognizer* imgRec);//libera lo spazio allocato dalla rete neurale
```

ADT *TrainData*

Implementa un ADT necessario per registrare i dati di training, ed eventualmente effettuare valutazioni,come *l'history* di tutti gli errori della rete neurale comparato con ciascuna epoch.

```c
typedef struct {
    unsigned int nCorrect;//numero di previsioni corrette

    unsigned int epochs;//numero di scorrimenti del dataset

    double* errors;//array contenente l'history di tutti gli errori(per plottare il risultato)

    double meanError;//errore medio
}TrainData;


TrainData createTrainData(unsigned int epochs);//crea un nuovo adt TrainData
void freeTrainData(TrainData* trainData);

unsigned int getNCorrect(TrainData trainData);
void setNCorrect(TrainData* trainData,unsigned int nCorrect);
void updateNCorrect(TrainData*,unsigned int nUp);

unsigned int getEpochs(TrainData trainData);
void setEpochs(TrainData* trainData, unsigned int epochs);

double* getTrainErrors(TrainData trainData);
void setTrainErrors(TrainData* trainData,double* errors);
void setTrainErrorsAt(TrainData* trainData,double value,unsigned int index);

double getMeanError(TrainData trainData);
void setMeanError(TrainData* trainData,double meanError);
void updateMeanError(TrainData* trainData,double meanUp);

void saveTrainData(TrainData trainData);//salva in un file csv l'history di tutti gli errori con le epochs per plottar

double getAccuracy(TrainData trainData);//ritorna la accuracy della rete neurale
```

ADT *Image*

Definisce un tipo di dato per memorizzare dei dati presi da un file di immagini di numeri scritti a mano in formato csv e le rispettive label, ovvero cosa rappresentano le immagini.

```c
//ADT per memorizzare immagini di numeri interi lette da un file csv

#define MAXIMGS 60000//numero massimo di immagini da caricare

typedef struct{
    unsigned int label;//label, ovvero il valore che rappresenta quell'immagine
    List* data;//lista di dati dell'immagine, (valori interi da 0-255)
}Image;

Image* createImage();
void freeImage(Image* img);

Image** loadImages(const char* filePath);//carica in un array di puntatori ad immagini i dati presenti su un file csv

unsigned int getLabel(Image* image);
void setLabel(Image* image,unsigned int label);

List* getData(Image* image);
void setData(Image* image,List* data);
```

ADT functions
Definisce delle funzioni di costo e di attivazione da poter utilizzare nella rete neurale.

```c
typedef double(*CostFunction)(double currValue,double expValue);
typedef double(*ActFunction)(double value);

/*tipo di dato che contiene i puntatori a funzioni cost e squish, con le loro rispettive derivate prime*/
typedef struct NetFunctions{
    CostFunction costFunc;//cost function
    CostFunction costFuncPrime;//derivata prima della cost function
    ActFunction squishFunc;//squishification function
    ActFunction squishFuncPrime;//derivata prima della squishification function
}NetFunctions;

//getters e setters ---------------------------
NetFunctions getFunctions(CostFunction costFunc,CostFunction costFuncPrime,ActFunction squishFunc,ActFunction squishFuncPrime);

CostFunction getCostFunc(NetFunctions functions);
void setCostFunc(NetFunctions* functions,CostFunction costFunc);

CostFunction getCostFuncPrime(NetFunctions functions);
void setCostFuncPrime(NetFunctions* functions,CostFunction costFuncPrime);

ActFunction getSquishFunc(NetFunctions functions);
void setSquishFunc(NetFunctions* functions,ActFunction squishFunc);

ActFunction getSquishFuncPrime(NetFunctions functions);
void setSquishFuncPrime(NetFunctions* functions,ActFunction squishFuncPrime);
//---------------------------------

//funzioni disponibili--------------------
double linearActv(double value);
double linearActvPrime(double value);
double cost(double currValue,double expValue);
double costPrime(double currValue,double expValue);
double sigmoid(double value);
double relu(double value);
double sigmoidPrime(double value);
double logLoss(double pY,double y);
double logLossPrime(double pY,double y);
double reluPrime(double value);

double softMax(List* outLayer,double z);
double softMaxPrime(List* outLayer,double z);
```

interfaccia *ImageRecView.*
Definisce l'interfaccia del programma

```c
#define LIMIT 200//numero massimo di caratteri in input

#define FCOMMANDS 7//numero di comandi iniziali
#define SCOMMANDS 4//numero di comandi secondari
#define TCOMMANDS 5//numero di comandi terziari

#define NCOST 2//numero di cost functions disponibili
#define NACT 3//numero di activation funcions disponibili
#define NFUNCKEYS 2

//enumeratore per classificare ciascun comando disponibile
typedef enum {
    LOAD_N_CONFIG = 111,
    LOAD_TE_DATA = 231,
    LOAD_TR_DATA = 221,
    LOAD_N_DATA = 211,
    SAVE_N_DATA = 212,
    CONFIG_N_FUNC = 313,
    SHOW_N_FUNC = 314,
    CONFIG_N_LAYERS = 413,
    CONFIG_N_TRAIN = 513,
    LOAD_C_CONFIG = 141,
    TRAIN = 5,
    TEST = 6,
    ESC = 7
}Command;



void startView();//funzione principale da richiamare nel main

List* tokenize(char* input);//prende in input un array che rappresenta il comando e ritorna una lista di token
Command parse(List* tokens);//prende in input una lista di token e la analizza cercando di classificarla come uno dei comandi disponibili nell'enum
void executeCommand(Command command,ImageRecognizer* imgRec,List* tokens,boolean* esc);//esegue un comando

//------funzioni che vengono chiamate all'esecuzione di un determinato comando-----------------

boolean loadTestData(List* tokens,ImageRecognizer* imgRec);
boolean loadTrainData(List* tokens,ImageRecognizer* imgRec);

boolean configFunctions(List* tokens,ImageRecognizer* imgRec);
void showFunctions();//ok

boolean configNetLayers(List* tokens,ImageRecognizer* imgRec);
boolean configNetTraining(List* tokens,ImageRecognizer* imgRec);

void loadCurrentConfig(ImageRecognizer* imgRec);

void train(ImageRecognizer* imgRec);
void test(ImageRecognizer* imgRec);

void escProgram(ImageRecognizer* imgRec);

void loadNetworkConfig(List* tokens,ImageRecognizer* imgRec,boolean* esc);

boolean saveNetwork(List* tokens,ImageRecognizer* imgRec);

boolean loadNetwork(List* tokens,ImageRecognizer* imgRec);

//utility
int isNumber(char* string);//ritorna 1 se è un numero , -1 altrimenti
List* obtainAdvancedConfig(char* configString);/*prende in input una stringa per configurare in advanced : {1,2,3}, controlla se
```

Descrizione corrispondenza tra interfaccia utente e interfaccia dell'adt :

Nella seguente tabella viene rappresentata la corrispondenza tra i comandi del programma ,con rispettivi input se necessari, e le operazioni presenti nell'interfaccia I*mageRecView.*

comandi interfaccia	**input necessario**	**operazioni interfaccia ADT**
load network-config	percorso file	loadNetworkConfig
load test-data	percorso file	loadTestData
load train-data	percorso file	loadTrainData
load current-config	percorso file	loadCurrentConfig
load network-data	percorso file	loadNetwork
save network-data	percorso file	saveNetwork
config network-functions	Cost=(costo) Act=(attivazione)	configFunctions
config network-layers	config network-layers=simple numInputs=(..) numHiddens=(..) hiddenSize=(..) numOutputs=(...) config network-layers=advanced {num neuroni,num neuroni,.....,..}	configNetLayers
config network-training	epochs=(..) learningRate=(..) trainMode=(online/miniBatch(..))	configNetTraining
train	nessun input	train
test	nessun input	test
esc	nessun input	escProgram