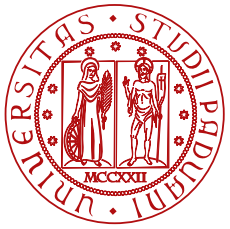

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Laurea in Ingegneria Elettronica

Transfer Learning: analisi di un codice Matlab

Laureando

Cimarosti Lorenzo

Numero di Matricola 2032371

Relatore

Giorgi Giada

Università di Padova

Anno Accademico
2024/2025

Sommario

Il transfer learning è una tecnica fondamentale del machine learning, che permette di riutilizzare un modello pre-addestrato adattandolo e personalizzandolo per nuovi scopi. In questo elaborato verranno presentate le principali metodologie e ambiti applicativi di tale processo. Successivamente tramite ambiente di lavoro Matlab, verrà analizzato nello specifico un'applicazione come classificatore di immagini partendo da una rete neurale esistente, mettendo in risalto i punti chiave per la realizzazione di tale processo. Durante lo sviluppo verranno approfondite le varie parti di codice del programma, trattando le metriche e i parametri più rilevanti da considerare per la progettazione della propria rete neurale; seguendo le indicazioni fornite in questa tesi sarà così possibile sviluppare un modello con buone performance pronto all'utilizzo.

Indice

Lista delle Figure **xi**

1 Il Transfer Learnig **1**
1.1 Come nasce il transfer learning 1
1.2 cos'è il transfer learning 2
1.3 Perchè viene utilizzato 2
1.4 campi di applicazione del transfer learning 2
1.5 Modelli di transfer learning 3

2 Macro blocchi **5**
2.1 Blocchi principali codice Matlab 5
2.1.1 Descrizione macroblocchi 6

3 Analisi codice Matlab **11**
3.1 Creazione imageDatastore 11
3.2 Preparazione rete neurale 13
3.3 Addestramento rete neurale 14
3.3.1 Overfitting 16
3.3.2 Underfitting 18
3.4 Test rete neurale 19
3.5 Classificazione nuova immagine 21

4 Conclusioni **23**

Bibliografia **25**

Lista delle Figure

2.1 Grafico reti CCN 7
2.2 Matrice di confusione 9

3.1 Parametri da modificare nella rete 13
3.2 Andamento Accuratezza durante la fase di addestramento 15
3.3 Andamento Perdita durante la fase di addestramento 16
3.4 Esempio matrice restituita da YTest 20
3.5 Matrice di confusione previsione/reale 21
3.6 Esito previsione nuova immagine 22

1

Il Transfer Learnig

1.1 Come nasce il transfer learning

La prima volta in cui si parlò di transfer learning fu nel 1976, quando gli scienziati Ante Fulgosi e Stevo Bozinovski divulgarono un primo articolo[1] che parlava dell'utilizzo di questa tecnica nei processi di costruzione delle reti neurali, analizzando anche un primo modello matematico. Più avanti nel 1981 fu pubblicato un rapporto sull'applicazione del transfer learning[2] per l'addestramento di una rete neurale, con un set di dati formati da immagini che rappresentavano caratteri utilizzati nei terminali dei computer. L'obiettivo del rapporto era dimostrare sperimentalmente come potesse avvenire l'apprendimento per trasferimento. Nonostante il transfer learning abbia visto i suoi fondamenti teorici già dagli anni Settanta, il suo utilizzo non fu immediato, infatti la sua diffusione è avvenuta in periodi più recenti, quando le risorse computazionali hanno consentito al deep learning il suo sviluppo.

1.2 COS'È IL TRANSFER LEARNING

Il transfer learning è una sotto-branca del machine learning e del deep learning che prevede il riutilizzo di modelli pre-addestrati su determinati dataset, sfruttando così la conoscenza acquisita per adattarli e personalizzarli a nuovi scopi.

1.3 PERCHÈ VIENE UTILIZZATO

Tramite il transfer learning è possibile ottimizzare in maniera sensibile le risorse già disponibili; infatti, non è necessario creare ed istruire un nuovo modello di deep learning da zero ogni volta che viene richiesta una nuova applicazione, ma è possibile riutilizzare un modello di rete neurale già istruito in precedenza per un nuovo dataset. Per fare ciò bisogna modificare solo gli ultimi layer, in quanto la modifica dei layer di input non influenza la logica sottostante e ne consente l'adattamento a molteplici scopi, sfruttando così solo gli elementi necessari ai propri obiettivi. Inoltre, nell'ambito del deep learning spesso è necessario utilizzare set di dati generalisti, rendendo così quasi naturale in una fase successiva personalizzare la rete per soddisfare compiti specifici.

1.4 CAMPI DI APPLICAZIONE DEL TRANSFER LEARNING

Come già detto in precedenza, grazie al transfer learning i data scientist hanno potuto sfruttare modelli già usati per riadattarli a compiti simili; gli ambiti in cui sono più frequentemente utilizzati algoritmi di transfer learning sono elencati di seguito.

Computer vision/riconoscimento delle immagini: alcune reti neurali possono risolvere compiti di "image related" potendo riconoscere anche dettagli all'interno delle immagini. I layer di input contengono la logica necessaria per gli obiettivi di riconoscimento delle immagini, come ad esempio riconoscere bordi e texture, mentre i layer di output sono modificabili in quanto servono per riconoscere oggetti specifici, potendo così adattare la rete ai vari scopi. Quindi il transfer learning viene utilizzato in compiti come il riconoscimento dell'immagine (image recognition) e la rimozione

del rumore (noise reduction) oppure per il rilevamento di oggetti (object detection).

Natural language processing/riconoscimento della voce: consiste nell'utilizzo di modelli pre-addestrati in grado di comprendere le strutture del linguaggio, consentendo ai computer di comunicare con il linguaggio umano, così da risolvere problemi come: la traduzione automatica dei testi, il completamento automatico, la risposta alle domande degli utenti ed elaborare il discorso umano in forma scritta.

Classificazione documentale/riconoscimento del testo: i modelli per il riconoscimento del linguaggio vengono utilizzati anche per applicazioni nell'ambito della classificazione documentale, che gestiscono in maniera automatica i metadati dei singoli file per classificarli e predisporli in maniera efficiente.

Medico: sempre nell'ambito della computer vision il transfer learning è presente anche in ambito medicale, infatti viene utilizzato per applicazioni di diagnostica rendendo possibile rilevare eventuali infezioni, cellule maligne o alterazioni cardiache; imparando così a leggere ogni tipologia di referto medico, dai vetrini di laboratorio, ai tracciati, fino ai raggi X.

1.5 Modelli di transfer learning

Esistono differenti tipologie di transfer learning definite dai vari algoritmi di deep learning utilizzati, le principali sono elencate di seguito.

transfer learning induttivo: i domini di origine e destinazione sono gli stessi, ovvero viene usato lo stesso tipo di dati associandoli a differenti compiti. Gli algoritmi utilizzano le caratteristiche presenti nei domini di origine per provare ad adattarli, così da risolvere problemi di destinazione. Solitamente l'apprendimento induttivo può essere diviso in ulteriori due categorie: sequenziale, il modello si addestra su un primo compito per poi utilizzare le conoscenze apprese per risolvere compiti successivi; multitasking, dove il modello viene addestrato simultaneamente su più compiti correlati.

transfer learning trasduttivo: i domini di origine e destinazione sono differenti anche se simili tra loro. La differenza è che non tutti i dati etichettati nel

dominio di origine coincidono con quelli presenti nella destinazione, il trasferimento tra domini diversi viene considerato come adattamento del dominio.

transfer learning non supervisionato: simile alla tipologia induttiva, non sono presenti necessariamente dati etichettati nei domini di origine e destinazione, anche se questi domini sono simili, i compiti da eseguire sono diversi.

2

Macro blocchi

Di seguito verranno illustrate brevemente le operazioni fondamentali per applicare la procedura di transfer learning ad una rete neurale tramite Matlab, tale processo fa riferimento all'esempio presente su Matlab di un riconoscitore di immagini.

2.1 Blocchi principali codice Matlab

- Creazione datastore
- Caricamento rete pre-addestrata
- Modifica rete pre-addestrata con parametri
- Addestramento nuova rete
- Test della rete
- Nuovo test con nuove immagini

2.1.1 Descrizione macroblocchi

Creazione datastore

Come primo passo è necessario caricare su Matlab la cartella dove sono presenti le immagini che verranno utilizzate per istruire la nuova rete neurale, in seguito si crea un Datastore di immagini dove verranno inserite e memorizzate le immagini campione. Successivamente dal Datastore si estraggono i due parametri necessari alla rete neurale: i nomi e il numero delle classi di oggetti.

Caricamento rete preaddestrata

Tramite il tool "deepNetworkDesigner" viene caricata una rete neurale pre-addestrata che in seguito verrà modificata, sono presenti tantissimi modelli pre-addestrati tra cui scegliere, ognuno con differenti caratteristiche. Per selezionare il modello più opportuno per il proprio scopo, è necessario considerare gli aspetti positivi e negativi di ciascuno di essi basandosi su:

Velocità delle previsioni: è un parametro che indica il tempo necessario per calcolare un output a partire da un input, varia in base alle dimensioni del modello e alla sua architettura, inoltre è influenzata da fattori come dimensioni di batch e hardware.

Dimensioni: l'importanza di questo parametro dipende da dove e come si ha intenzione di distribuirlo, se il modello sarà utilizzato su un target a risorse limitate, bisognerà considerare le dimensioni della rete.

Precisione: un modello con alte prestazioni su specifici dati probabilmente funzionerà bene con nuovi dataset simili. Ma se si riaddestra un modello con bassa precisione su un nuovo dataset diverso dall'originale, non implica necessariamente che le sue prestazioni saranno scarse, al contrario, si potrebbero riscontrare dei miglioramenti.

In figura 2.1 è presente un grafico a dispersione che confronta le prestazioni delle varie reti neurali convoluzionali per il riconoscimento delle immagini. L'asse orizzontale indica il tempo di predizione relativo, con valori maggiori verso destra che corrispondono a modelli con velocità inferiore. L'asse verticale rappresenta l'accuratezza espressa in percentuale, una posizione più elevata corrisponde a una maggiore precisione nella classificazione. Inoltre la dimensione

delle bolle rappresenta la complessità della rete, dimensioni maggiori indicano modelli più pesanti che richiedono più memoria e risorse computazionali.

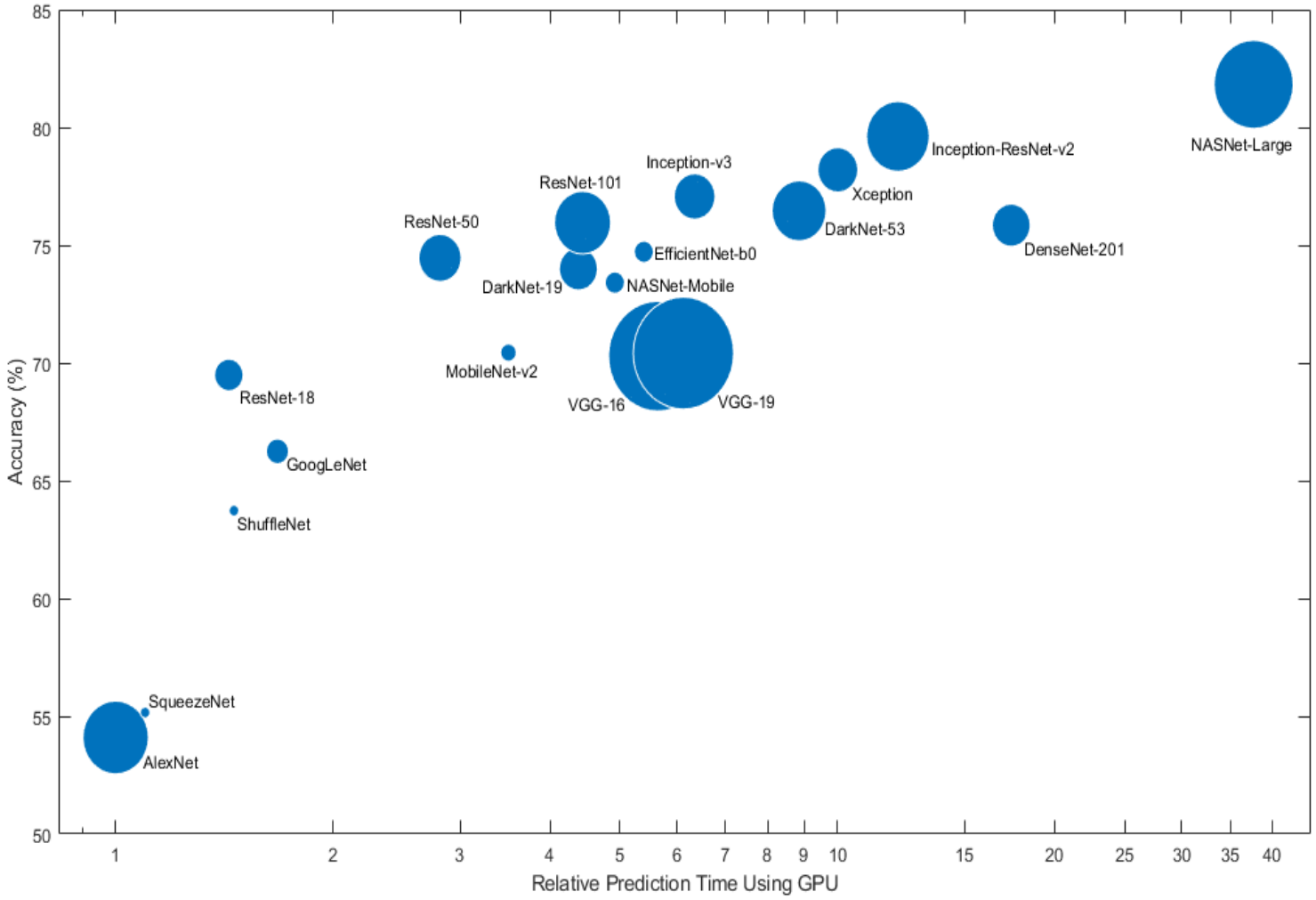

Figura 2.1: Grafico reti CCN

Per questo esempio si utilizza la SqueezeNet (si poteva selezionare qualsiasi altra rete presente tramite Matlab). Si tratta di una rete convoluzionale, solitamente utilizzata per attività di classificazione di immagini e riconoscimento degli oggetti. Una rete convoluzionale è composta da un layer di input, un layer di output e molti layer nascosti nel mezzo, i quali eseguono elaborazioni sui dati al fine di apprendere le feature specifiche dei dati stessi.

Modifica rete pre-addestrata con parametri

Per modificare la rete pre-addestrata è necessario modificare l'ultimo livello apprendibile e il livello di classificazione del modello. Il layer finale viene modificato con i parametri estratti dal datastore. Il modello SqueezeNet è realizzato per classificare 1000 classi di oggetti, tale parametro è espresso nel "convolution2dLayer" dove verrà modificato con il nuovo numero di classi da analizzare. È inoltre possibile definire il passo e le velocità di apprendimento per questo livello. In seguito a questa modifica, bisognerà analizzare la rete per verificare che non siano presenti errori o avvisi dovuti alle modifiche apportate e successivamente esportarla.

Addestramento nuova rete

Dopo aver modificato la rete, vengono specificate le opzioni di addestramento. Per la scelta di tali specifiche si utilizza l'applicazione Experiment Manager. Le opzioni di addestramento presenti sono le seguenti:

- discesa stocastica del gradiente con momento/Stochastic Gradient Descent with Momentum (SGDM)
- propagazione quadratica media/Root Mean Square Propagation (RMSProp)
- stima adattiva del momento/Adaptive Moment Estimation (Adam)
- memoria limitata BroydenFletcherGoldfarbShanno/Limited-memory BroydenFletcherGoldfarbShanno (L-BFGS)

Ognuno di questi algoritmi serve per velocizzare e migliorare la stabilità durante la fase di addestramento, per questo modello verrà utilizzata la stima adattiva del momento. Siccome l'obiettivo è la classificazione, si utilizzerà la cross-entropy loss. Successivamente avverrà l'addestramento della rete neurale per renderla in grado di classificare le categorie di oggetti presenti nel datastore.

Test della rete con imageDatastore

Dopo che la rete è stata addestrata è necessario effettuare una fase di test, dove si utilizzano immagini campione che non sono state utilizzate dal modello durante l'addestramento. Questo processo andrà a verificare se la rete risponde bene a nuove immagini, cioè se riesce effettivamente a distinguere le categorie di oggetti su cui è stata addestrata, determinando così le sue performance e se il modello è valido. Per valutare le performance di una rete si fa riferimento a queste metriche:

Accuratezza: indica quante volte il modello ha correttamente classificato un oggetto rispetto al totale

$$accuratezza = \frac{numero\ risposte\ corrette}{numero\ risposte\ totali} \tag{2.1}$$

Bisogna prestare attenzione al senso di questo parametro, infatti a seconda del contesto applicato può risultare fuorviante. Se ad esempio si utilizza un dataset sbilanciato, dove le classi sono distribuite in maniera impari, con una classe che compone l'85% del dataset e il modello classifica erroneamente ogni dato sempre a quella classe, l'accuratezza risulterà l'85% anche se il modello è tutt'altro che performante.

Precisione: Per comprendere questo parametro si fa riferimento a una matrice di confusione(figura 2.2), la quale determina il grado d'errore del modello confrontando i valori previsti con quelli effettivi per un set di dati.

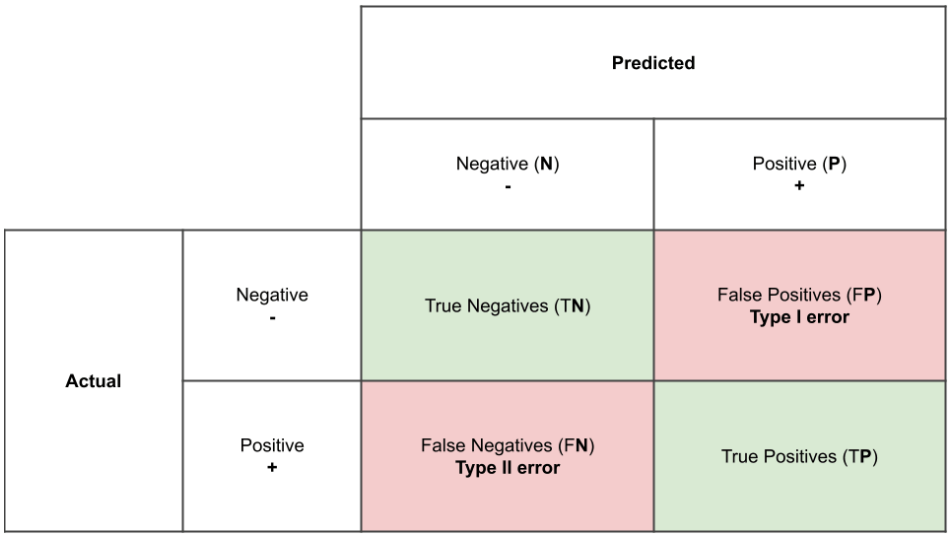

Figura 2.2: Matrice di confusione

Di seguito sono riportati i significati dei possibili valori:

Veri positivi, TP: è stata correttamente classificata la presenza di un dato in una classe

Falsi positivi, FP: è stata classificata la presenza di un dato in una classe anche se non presente in questa

Veri negativi, TN: predice correttamente la non presenza di un dato in una classe

Falsi negativi, FN: non riconosce la presenza di un dato in una classe

Può essere definito come

$$precisione = \frac{TP}{TP + FP} \tag{2.2}$$

indica quanto spesso il modello è corretto quando classifica una classe come positiva, è chiamata anche specificità perchè indica quanto è sensibile a riconoscere i dati.

Richiamo (Recall): può essere definito come

$$richiamo = \frac{TP}{TP + FN} \tag{2.3}$$

all'aumentare del richiamo, il nostro modello diventa sempre meno preciso e classifica anche classi negative come positive.

Punteggio F1 (F1-Score): combina precisione e richiamo in una sola metrica. Questa è la media armonica di precisione e richiamo.

$$accuratezza = 2 \times \frac{precisione \times richiamo}{precisione + richiamo} \tag{2.4}$$

Log loss: indica la differenza tra le probabilità delle previsioni del modello e le probabilità della realtà osservata, stima la probabilità che un dato abbia una classe positiva.

$$LogLoss = -\frac{1}{n}\sum[y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})] \tag{2.5}$$

n indica il numero di esempi nel dataset, y_i è il valore di riferimento e $\hat{y_i}$ sta per la predizione fornita dal modello.

Terminata la fase di test, se le performance raggiunte dal modello soddisfano i requisiti, la rete è pronta per l'utilizzo, in questo caso come classificatore di immagini; ogni qualvolta verrà inserita una nuova immagine come input questa sarà etichettata tra le classi presenti.

3

Analisi codice Matlab

Di seguito verrà analizzato nel dettaglio il codice Matlab necessario per l'implementazione del transfer learning in una rete neurale utilizzata per riconoscere immagini di: cappelli, torce, cacciaviti, carte da gioco e cubi di Rubik.

3.1 Creazione imageDatastore

```
folderName = "MerchData";
unzip("MerchData.zip",folderName);
```

Estrae il contenuto del file zip e lo inserisce nella cartella associata al "folderName", se questa cartella non esiste Matlab la crea. Nella cartella "MerchData" sono contenute 5 sottocartelle che rappresentano le diverse categorie di oggetti da identificare, dove in ciascuna sono presenti 15 immagini per oggetto. Il formato delle immagini è di 227x227 pixel con 3 canali, corrispondenti a un'immagine a colori RGB.

```
imds = imageDatastore(folderName, ...//sorgente dei file
    IncludeSubfolders=true, ...//controllo sottocartelle
    LabelSource="foldernames"); ...//assegnazione
   etichette classi immagini
```

Crea un datastore di immagini dove sono presenti tutti gli elementi della cartella "MerchData", riconoscendo le estensioni di tipo: ".png", ".jpg", ".jpeg",

".tif", ".tiff" tramite funzione interna "imformats". È possibile con l'istruzione "FileExtensions" selezionare quali formati riconoscere (se non specificati vengono riconosciute tutte le estensioni di default), oppure includere tipi di file non presenti da "imformats", che in seguito devono essere gestiti correttamente nel codice. Tramite il comando "LabelSource=foldernames" si assegna alle varie immagini presenti nel file "MerchData" l'etichetta con il nome della sottocartella di cui fanno parte, così da dividere le immagini nelle 5 categorie di oggetti da identificare. Il motivo principale per cui si utilizza una struttura di tipo datastore è che la memoria al suo interno non viene caricata dal programma fino al momento dell'uso. Di conseguenza, è possibile memorizzare grandi raccolte di dati senza incorrere in problemi di memoria, consentendo una lettura efficiente durante l'addestramento della rete neurale.
Per verificare se le immagini sono state caricate correttamente nel datastore è possibile visualizzare alcuni elementi di esso tramite questo codice di esempio

```
numImages = numel(imds.Labels);  ...// numero elementi
   datastore "75"
idx = randperm(numImages,16);  ...// genera 16 numeri
   casuali dal numero di elementi totali
I = imtile(imds,Frames=idx); ...//mostra immagini del
   datastore, Frames seleziona solo le immagini che devono
    essere visualizzate
Figure ...// crea nuova finestra
imshow(I) ...// visualizza il contenuto
```

```
classNames = categories(imds.Labels);
numClasses = numel(classNames);
```

Identifica i nomi delle classi all'interno delle etichette del datastore tramite il comando "categories"; successivamente memorizza il loro numero nella variabile "numClasses".

```
[imdsTrain,imdsValidation,imdsTest] = splitEachLabel(imds
   ,0.7,0.15,0.15,"randomized");
```

Il datastore viene suddiviso in 3 diverse sezioni: imdsTrain (70%), imdsValidation(15%) e imdsTest(15%); in ogni sezione sono presenti immagini scelte in modo casuale che verranno utilizzate rispettivamente durante le fasi di: addestramento, validazione e verifica della rete neurale.

3.2 Preparazione rete neurale

Tramite il tool "DeepNetworkDesigner" viene selezionata e caricata la rete neurale "SqueezeNet" da riaddestrare, modificando l'ultimo livello convoluzionale della rete.

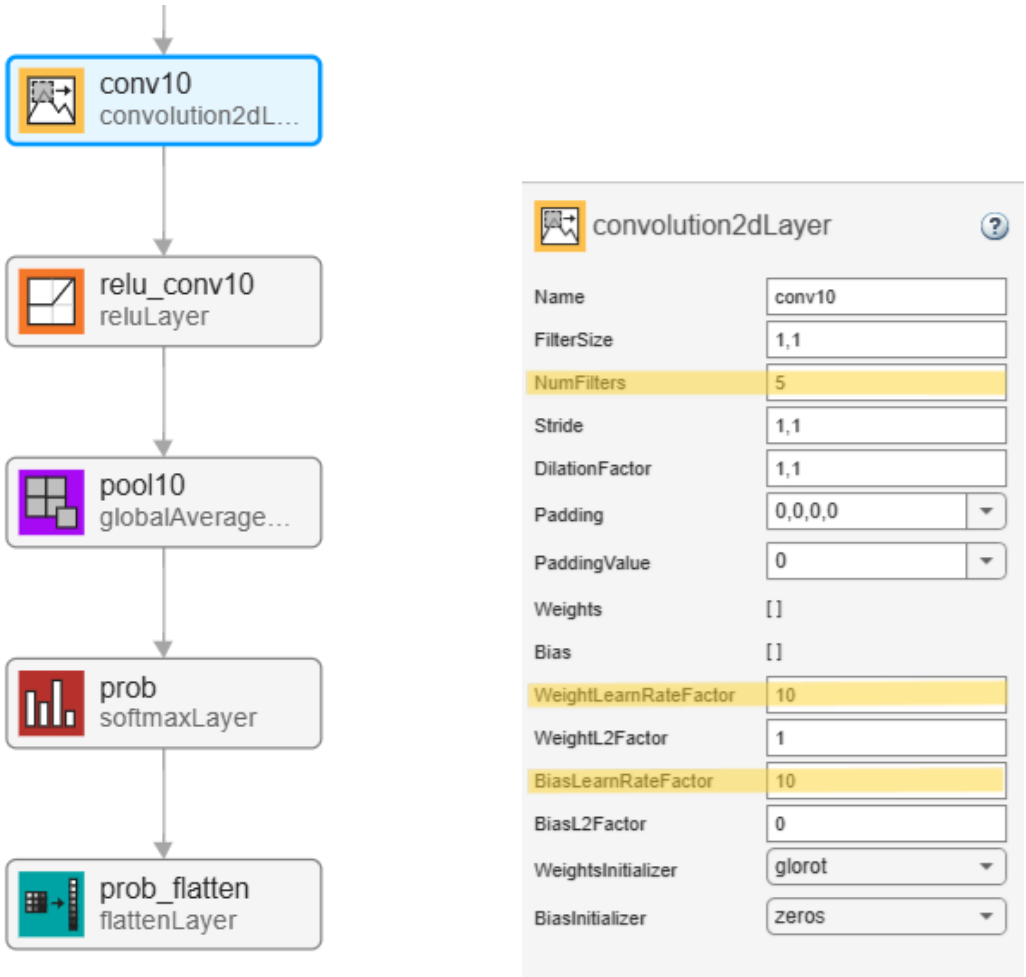

Figura 3.1: Parametri da modificare nella rete

Nel pannello Designer si seleziona l'ultimo livello convoluzionale "conv10" e si abilita lo sblocco del livello per poter modificare i parametri al suo interno

che dovranno essere adattati al nuovo problema di identificazione. Il parametro "NumFilters" dovrà essere impostato con il numero di classi di uscita, in questo caso pari a 5, modificando così il numero di kernel presenti nel layer, che individueranno i 5 pattern differenti da assocciare alle immagini legate alle classi oggetto presenti. Successivamente, impostare il parametro "WeigthLearnRateFActor" e "BiasLearnRateFactor"= 10, aumentando così il tasso di aggiornamento dei pesi e dei bias rispetto agli altri layer. Aumentare tali parametri in soli determinati livelli permette, durante la fase di addestramento, di reagire più velocemente agli errori, prestando attenzione a non incorrere in problemi di "overfitting", che verranno analizzati in seguito. Al termine delle modifiche cliccare su "Analyze" per verificare che non siano presenti errori o avvisi. Ora la rete è pronta per l'addestramento quindi cliccare su "Export".

3.3 Addestramento rete neurale

Per addestrare la rete, bisogna prima specificare le opzioni di addestramento indicando l'algoritmo di ottimizzazione da utilizzare; per la realizzazione di questo riconoscitore di immagini verrà utilizzata la stima adattiva del momento "adam".

```
options = trainingOptions("adam", ...
    ValidationData=imdsValidation, ...
    ValidationFrequency=5, ...
    Plots="training-progress", ...
    Metrics="accuracy", ...
    Verbose=false);
```

Durante la fase di addestramento i dati di validazione utilizzati vengono ottenuti da "imdsValidation", la sezione di datastore dedicata a questo processo. Tramite il comando "ValidationFrequency=5" si indica che ogni 5 iterazioni di training (cioè aggiornamenti dei pesi della rete) verrà eseguita la validazione. Durante un'iterazione viene analizzata dalla rete un mini-batch, cioè una porzione di datastore di "imdsTrain" con all'interno le immagini campione. Le dimensioni di tale parametro possono essere dichiarate tramite il comando "MiniBatchSize", se non dichiarato all'interno di "trainingOptions" il valore di default è impostato a 128. Un ulteriore parametro possibile da specificare per l'addestramento, è il numero massimo di epoche, che indica quante volte è stato analizzato il set di

dati "imdsTrain"; tramite il comando "MaxEpochs" è possibile modificare tale valore che di default è impostato su 30 epoche massime. Le ulteriori istruzioni generano un grafico dinamico attraverso"Plots=training-progress", così da poter monitorare l'andamento della metrica dell'accuratezza (accuracy) e della perdita (loss) di "imdsTrain" e "imdsValidation". È possibile visualizzare anche un ulteriore output visivo verboso impostando "Verbose=true".

```
net = trainnet(imdsTrain,net_1,"crossentropy",options);
```

Viene avviato il processo di addestramento utilizzando il set dati "imdsTrain" con le opzioni specificate precedentemente utilizzando la cross entropy come funzione di loss (perdita). In figura 3.2 e 3.3 sono presenti gli andamenti del dataset di addestramento (azzurro) e di validazione (arancione) in funzione delle metriche di accuratezza e della perdita durante la fase di addestramento, mostrando come le prestazioni della rete aumentino ad ogni iterazione.

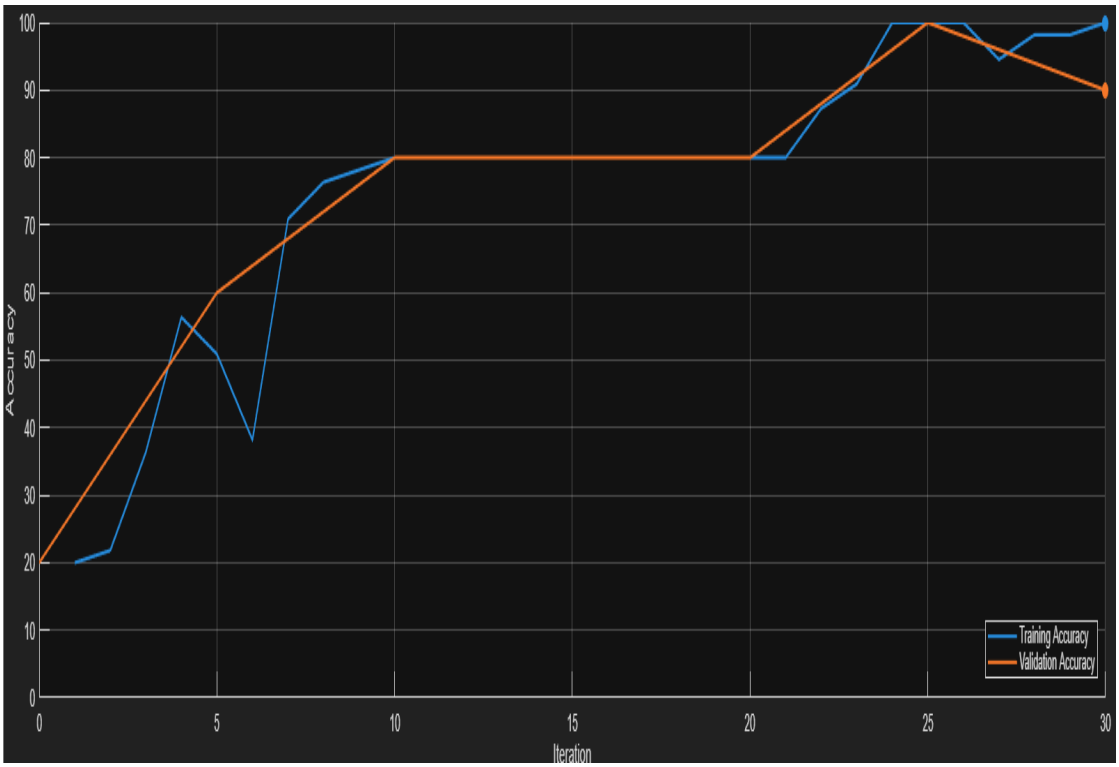

Figura 3.2: Andamento Accuratezza durante la fase di addestramento

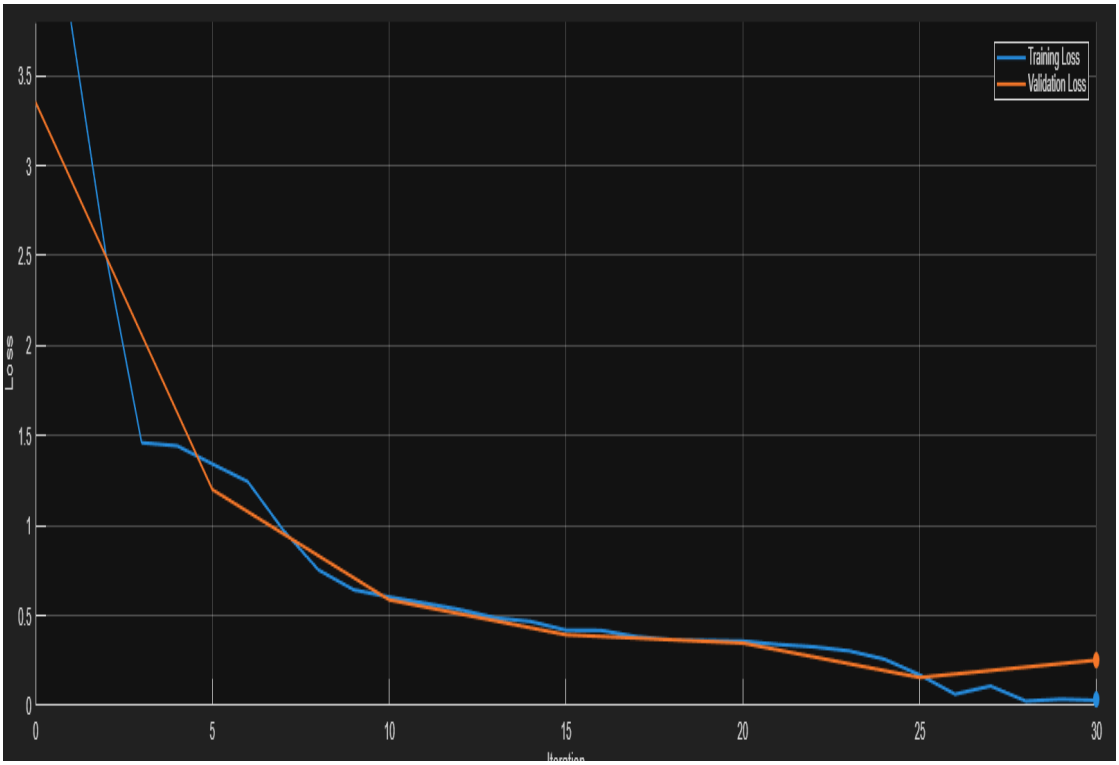

Figura 3.3: Andamento Perdita durante la fase di addestramento

3.3.1 Overfitting

Durante il processo di addestramento, è possibile osservare come in figura 3.2 l'accuratezza di "imdsTrain", con l'avanzare delle epoche, aumenti progressivamente fino a raggiungere un valore vicino al 100%; mentre in figura 3.3 la cross-entropy loss diminuisce fino ad avvicinarsi quasi a zero, mostrando come il numero delle previsioni corrette rispetto al totale della rete stia aumentando. Bisogna prestare attenzione però che le sole metriche di "imdsTrain" non sono sufficienti per una valutazione, in quanto è necessario analizzare le metriche correlate a "imdsValidation". Infatti questo set di dati consente di capire effettivamente se la rete stia imparando a riconoscere le immagini, effettuando predizioni su dati diversi dal set di addestramento, ovvero su oggetti mai visti. Se durante l'addestramento si verifica un peggioramento dei valori di accuratezza e di cross-entropy loss riguardanti a "imdsValidation", mentre i parametri rispettivi di "imdsTrain" stanno migliorando, si sta verificando il fenomeno dell'overfitting. Questo comportamento è visibile in figura 3.2 dove nelle fasi finali dell'addestramento, l'accuratezza sul dataset di validazione accenna a una di-

minuzione. Tale situazione induce la rete a migliorare le performance sui dati di addestramento, ma non con quelli di validazione, come se stesse imparando a "memoria" i dati, non riconoscendo informazioni diverse. Questo fenomeno si può verificare quando:

Dataset ridotto: con pochi dati il modello non identifica schemi generali validi e si adatta troppo ai pochi esempi disponibili.

Training troppo lungo: portando il modello a "memorizzare" i dati di training.

Modello troppo complesso: il modello può imparare anche i dettagli inutili (rumore) dei dati di training, perdendo capacità di generalizzazione.

Trainset non rispecchia i dati reali: il modello si addestra su dati non utili non riuscendo a generalizzare.

Per ridurre l'overfitting si può ricorrere a diverse soluzioni:

Regolarizzazione: tramite diverse tecniche di addestramento si vanno a introdurre modifiche sulla complessità della rete, come ad esempio la regolarizzazione Ridge (L2). Di seguito viene illustrata la funzione di perdita (cross entropy loss) con l'aggiunta della regolarizzazione

$$E_R(\theta) = E(\theta) + \lambda\Omega(w) \tag{3.1}$$

dove w è il vettore di peso, λ è il fattore di regolarizzazione (coefficiente) e la funzione di regolarizzazione Ω(w) è

$$\Omega(w) = \frac{1}{2}w^T w \tag{3.2}$$

che aggiunge penalità sui pesi grandi, evitando di memorizzare dettagli inutili, obbligando il modello a trovare soluzioni più semplici. In Matlab è possibile specificare il fattore di regolarizzazione λ tra le opzioni di addestramento

```
1    options = trainingOptions('adam', ...
2    L2Regularization=0.01, ...
3
```

Arresto anticipato dell'addestramento: serve per evitare che la rete inizi a riconoscere il "rumore" dei dati di training, prestando attenzione a terminare il processo dopo che il modello sia effettivamente accurato.

Aumento del dataset: il modello con più dati a disposizione tende a identificare schemi generali.

3.3.2 Underfitting

Durante il processo di addestramento può verificarsi che il valore di accuracy "imdsTrain" risulti molto basso e il loss si stabilizzi in un valore elevato fino alla fine dell'addestramento; di conseguenza anche le performance su "imdsValidation" o "imdsTest" saranno scarse. Questo fenomeno viene identificato come underfitting, ovvero l'incapacità del modello di imparare a riconoscere i dati forniti durante l'addestramento; si verifica quando:

Modello troppo semplice: la struttura del modello non è sufficientemente complessa per cogliere i pattern nei dati in input.

Addestramento insufficiente: il modello non ha avuto il tempo necessario per apprendere dai dati, ad esempio perché è stato interrotto troppo presto o i dati forniti erano troppo pochi o poco informativi.

Per ridurre l'underfitting si può ricorrere a diverse soluzioni:

Aumentare complessità del modello: permette di sfruttare meglio le informazioni contenute nei dati in input e di riconoscere pattern più complessi, migliorando così la capacità del modello di apprendere relazioni significative.

Aumentare durata addestramento: aumentare il tempo di addestramento può migliorare l'apprendimento della rete prestando sempre attenzione ai possibili problemi di overfitting.

aumentare qualità/varietà dati: maggiori dati permettono miglior apprendimento e capacità di generalizzazione, inoltre aumentare la qualità di essi

evita che il modello impari relazioni sbagliate ad esempio con immagini sfocate o scure.

3.4 Test rete neurale

Terminata la procedura di addestramento si può andare a effettuare una fase di test per verificare le performance finali del modello utilizzando "imdsTest".

```
1 inputSize = net.Layers(1).InputSize(1:2);// serve per
     memorizzare dimensioni immagini
2 YTest = minibatchpredict(net,imdsTest);
3 YTest = scores2label(YTest,classNames);
```

Viene memorizzato in "inputSize" la dimensione di altezza e larghezza delle immagini accettate dalla rete, il terzo parametro che non viene considerato è il numero di canali(1 canale se l'immagine è in scala di grigi, 3 per quelle a colori). Tramite il comando "minibatchpredict", vengono inviate alla rete tutte le immagini contenute in "imdsTest" per il riconoscimento dei vari pattern. Per ogni immagine viene generato un vettore punteggi(score), costruendo così una matrice dove le colonne rappresentano le immagini usate per il test, mentre le righe forniscono le varie classi. Ad esempio, dalla figura 3.4, si vede che la prima immagine, "Immagine 1", ha una probabilità dell'80% di essere una torcia, una probabilità del 10% di essere un cubo di Rubik ed infine una pari probabilità del 5% di essere una carta da gioco oppure un cacciavite. Si nota che la somma dei punteggi per ogni colonna deve dare valore unitario, corrispondendo ad una probabilità del 100% di appartenere ad una delle classi riportate. Infine tramite "scores2label" viene creata un'etichetta con il nome della classe corrispondente al punteggio più alto e il valore ottenuto. Nell'esempio di figura 3.4, verrà quindi associata all'immagine 1 una torcia, all'immagine 2 un cappello e, infine, all'immagine 3 un cacciavite.

	Immagine 1	Immagine 2	Immagine 3
Torcia	0.8	0.01	0.0
Cappello	0.0	0.85	0.02
Carte da gioco	0.05	0.1	0.06
Cacciavite	0.05	0.0	0.9
Cubo di Rubik	0.1	0.04	0.02

Figura 3.4: Esempio matrice restituita da YTest

```
1 TTest = imdsTest.Labels;
2 figure
3 confusionchart(TTest,YTest);
```

In "TTest" vengono salvate le etichette originali associate alle immagini prese dal datastore; successivamente in una nuova scheda viene visualizzata una matrice di confusione, come mostrato in figura 3.5, formata dalle etichette originali (TTest) e da quelle predette dalla rete (YTest). Nella diagonale principale sono presenti le immagini che sono state correttamente identificate dalla rete, mentre i valori fuori da essa mostrano quante volte la rete ha predetto in maniera errata l'immagine.

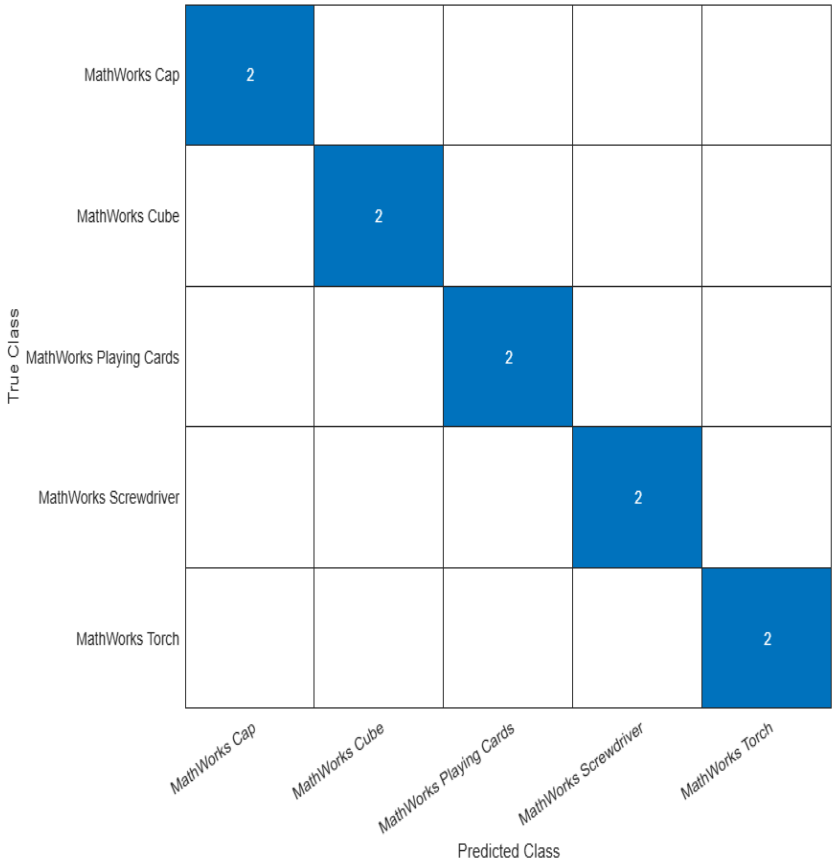

Figura 3.5: Matrice di confusione previsione/reale

3.5 Classificazione nuova immagine

Successivamente si può effettuare un'ulteriore verifica utilizzando un'immagine non presente nel dataset usato per il training tramite il seguente codice

```
1 im = imread("MerchDataTest.jpg");
2 im = imresize(im,inputSize(1:2));
3 X = single(im);
```

L'immagine ""MerchDataTest.jpg" viene caricata e salvata nella variabile "im" e ridimensionata secondo il formato accettato dalla rete tramite "inputSize". Il comando "single" converte le immagini da numeri interi a numeri a virgola

mobile, in quanto la rete lavora con questo formato fornendo una maggiore precisione.

```
scores = predict(net,X);
[label,score] = scores2label(scores,classNames);
```

Il comando "predict" fornisce la previsione dell'immagine inviata alla rete generando un vettore con all'interno i punteggi di ogni classe, viene quindi selezionato il punteggio più alto e creata un'etichetta con la classe e relativo valore.

```
figure
imshow(im)
title(string(label) + " (Score: " + gather(score) + ")");
```

Il comando "imshow" visualizza in una nuova scheda l'immagine caricata per il test con il relativo punteggio e classe predetta. Un esempio è fornito in figura 3.6. In questo caso era stata fornita l'immagine di un cappello e si può vedere che la rete correttamente associa questa immagine alla label "cap", con una probabilità superiore al 99%.

Figura 3.6: Esito previsione nuova immagine

4

Conclusioni

Il lavoro svolto ha permesso di analizzare ed applicare la tecnica di transfer learning, una metodologia sempre più importante e diffusa nel mondo del machine learning, con la quale è possibile ottimizzare reti già esistenti. Infatti, grazie all'utilizzo di modelli pre-addestrati, è possibile adattarli a compiti specifici, portando così dei notevoli vantaggi in termini di tempo di sviluppo, di addestramento e di adattabilità della rete. Grazie all'utilizzo dell'ambiente Matlab, in particolare agli strumenti presenti come Deep Network Designer, è stato possibile modificare la rete neurale SqueezeNet che si è dimostrata efficace per la classificazione di 5 immagini. In merito alla procedura di addestramento, sono stati utilizzati solo parametri standard forniti da Matlab che riguardano l'algoritmo di addestramento adam, la valutazione dell'accuratezza e della perdita. Tuttavia, è possibile ottimizzare e analizzare più approfonditamente la rete, per un possibile miglioramento dei risultati, sperimentando ulteriori algoritmi di addestramento come SGDM o L-BFGS e variando parametri come il numero di epoche e dimensione del mini-batch. Inoltre, è possibile introdurre ulteriori tecniche di regolarizzazione come L2Regularization, DropoutLayer o l'early stopping. Però bisogna considerare che in questo contesto specifico, probabilmente tali modifiche potrebbero avere un impatto trascurabile dovuto alla presenza di un numero limitato di dati. In conclusione, il lavoro realizzato ha dimostrato concretamente come il processo di transfer learning sia una metodologia estremamente efficace per la realizzazione di reti neurali con compiti

specifici, e come l'ambiente di lavoro Matlab consente una sua applicazione rapida anche ad utenti non esperti del settore.

Bibliografia

[1] S. Bozinovski e A. Fulgosi. *The influence of pattern similarity and transfer of learning upon training of a base perceptron B2*. Proceedings of the Symposium Informatica, 1976.

[2] S. Bozinovski. *Teaching space: A representation concept for adaptive pattern classification*. COINS Technical Report, University of Massachusetts at Amherst, 1981.