UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Matematica e Fisica

Corso Minor in Data Science

Matlab for Neural Networks

Anno Accademico 2023/2024

Matlab for Neural Networks

In questo progetto, si sviluppa un sistema di classificazione automatica dei numeri civici utilizzando una rete neurale convoluzionale pre-addestrata. In particolare, si utilizzerà la rete ResNet-18, una delle architetture più efficaci per l'elaborazione delle immagini.
L'obiettivo è quello di utilizzare le capacità della rete presa in considerazione per poter classificare correttamente le immagini contenute nel dataset *Street View House Numbers* (SVHN).
Il dataset SVHN è una raccolta di immagini di numeri civici ottenuti da Google Street View.
È progettato per compiti di riconoscimento e classificazione automatica dei numeri nelle immagini e rappresenta una sfida significativa per i modelli di deep learning a causa della sua variabilità e complessità.
Il formato delle immagini è a colori (RGB) e ha una risoluzione di 32x32 pixel. Le immagini contengono uno o più numeri civici.
Il dataset include 10 classi corrispondenti ai numeri da 0 a 9. Ogni immagine è etichettata con il numero civico presente in essa.
Il dataset SVHN presenta una sfida interessante per la classificazione automatica a causa della variabilità delle immagini, che includono diverse condizioni di illuminazione, angolazioni e sovrapposizioni di oggetti.
Si è utilizzato MATLAB come ambiente di sviluppo, adattando la rete ResNet-18 alle specificità del problema preso in considerazione.

1.1 Studio del codice

Di seguito si andrà ad analizzare il codice implementato passo per passo.

1.1.1 Preparazione dei dataset

In questa prima parte di codice, si procede a scaricare e caricare il dataset SVHN preso in esame.

```
1  svhnDatasetPath = fullfile(tempdir, 'SVHN');
2
3  if ~exist(svhnDatasetPath, 'dir')
4      mkdir(svhnDatasetPath);
5  end
6
7  urlTrain = 'http://ufldl.stanford.edu/housenumbers/train_32x32.mat';
8  urlTest = 'http://ufldl.stanford.edu/housenumbers/test_32x32.mat';
9
10 trainFile = fullfile(svhnDatasetPath, 'train_32x32.mat');
11 testFile = fullfile(svhnDatasetPath, 'test_32x32.mat');
12
13 if ~exist(trainFile, 'file')
14     disp('Scaricamento del dataset di training SVHN...');
15     websave(trainFile, urlTrain);
16 end
17
18 if ~exist(testFile, 'file')
19     disp('Scaricamento  del dataset di test SVHN...');
20     websave(testFile, urlTest);
21 end
22
23 [trainingImages, trainingLabels, testImages, testLabels] = loadSVHNData(
       svhnDatasetPath);
```

- r. 1: viene creato un percorso in cui il dataset SVHN sarà memorizzato.
- r. 3-5: si effettua un check sull'esistenza della directory specificata, se non esiste viene creata usando *mkdir*.

- r. 7-8: vengono specificati gli URL da cui scaricare i file del dataset, rispettivamente del train set e del test set.
- r. 10-11: vengono riscritti i percorsi completi dove i file scaricati saranno salvati localmente.
- r. 13-16: viene verificato se il file di training esiste già nella directory specificata. Se non esiste, viene scaricato dal web utilizzando *websave*.
- r. 18-21: si esegue lo stesso procedimento per il test set.
- r. 21: viene chiamata una funzione, analizzata di seguito, che carica i dati di training e di test dai file memorizzati. Le variabili *trainingImages*, *trainingLabels*, *testImages* e *testLabels* contengono rispettivamente le immagini e le etichette del dataset di training e di test.

Dunque, questa parte del progetto verifica se il dataset SVHN è già disponibile localmente; se non lo trova, lo scarica da Internet e lo salva in una cartella. Successivamente, carica le immagini e le etichette dal dataset per poterlo utilizzare successivamente.

1.1.1.1 Funzione *loadSVHNData*

La funzione MATLAB *loadSVHNData* è progettata per caricare i dati del dataset SVHN da file locali.
La funzione prende un input, che rappresenta il percorso della cartella dove sono salvati i file del dataset SVHN e restituisce in output:

- XTrain: le immagini di training,
- YTrain: le etichette di training,
- XTest: le immagini di test,
- YTest: le etichette di test.

```
1  function [XTrain, YTrain, XTest, YTest] = loadSVHNData(svhnData)
2      trainData = load(fullfile(svhnData, 'train_32x32.mat'));
3      XTrain = permute(trainData.X, [2, 1, 3, 4]);
4      YTrain = trainData.y;
5
6      testData = load(fullfile(svhnData, 'test_32x32.mat'));
7      XTest = permute(testData.X, [2, 1, 3, 4]);
8      YTest = testData.y;
9
10     YTrain = categorical(YTrain);
11     YTest = categorical(YTest);
12 end
```

- r. 2: la funzione carica il file situato nel percorso specificato da *svhnData*. Questo file contiene le immagini e le etichette del set di training.
- r. 3-4: la funzione *permute* viene utilizzata per riorganizzare le dimensioni dell'array *trainData.X* da [height, width, channels, numImages] a [width, height, channels, numImages].
  Questa riorganizzazione è necessaria perché MATLAB carica i dati delle immagini in un formato che non è quello che l'algoritmo si aspetta.
  Mentre *trainData.y* contiene le etichette associate alle immagini di training.
- r. 6-8: lo stesso processo viene eseguito sul dataset di test.
- r. 10-11: le etichette *YTrain* e *YTest* vengono convertite in un formato *categorical*.

La funzione dunque ha il compito di carica i dati di training e di test del dataset SVHN, riorganizzare le dimensioni delle immagini per essere compatibili con la rete neurale utilizzata, per poi convertire le etichette in formato categorico, restituendo i dati pronti per essere utilizzati nelle operazioni di addestramento.

1.1.2 Modifica dei dataset

In questa parte di codice, si preparano i dataset che si andranno a utilizzare. Nel dettaglio, si ridimensionano le immagini del dataset SVHN a una dimensione standardizzata (224x224x3) e si crea *augmentedImageDatastore* per le immagini di training e di test. Questi datastore permettono di gestire efficientemente il caricamento e la preparazione delle immagini durante le fasi di addestramento e valutazione.

```
1 inputSize = [224 224 3];
2
3 trainingImagesResized = resizeImages(trainingImages, inputSize);
4 testImagesResized = resizeImages(testImages, inputSize);
5
6 augmentedTrainImages = augmentedImageDatastore(inputSize,
    trainingImagesResized, trainingLabels);
7 augmentedTestImages = augmentedImageDatastore(inputSize,
    testImagesResized, testLabels);
```

- r. 1: si specifica la dimensione delle immagini che verranno utilizzate come input per il modello. In questo caso, le immagini verranno ridimensionate a 224x224 pixel, con 3 canali (RGB). Questa dimensione è tipica per modelli pre-addestrati come quelli della famiglia ResNet.
- r. 3-4: viene utilizzata la funzione *resizeImages*, spiegata in seguito, per ridimensionare tutte le immagini di training a 224x224 pixel, in modo che abbiano la dimensione richiesta dal modello. Il processo di ridimensionamento è applicato sia per il train set sia per il test set.
- r. 6-7: si utilizza un oggetto di Matlab, *augmentedImageDatastore*, per applicare le trasformazioni necessarie. Questo processo è applicato sia per il train set sia per il test set.

1.1.2.1 Funzione *resizeImages*

La funzione MATLAB *resizeImages* è progettata per ridimensionare un set di immagini a una nuova dimensione specificata.
La funzione prende in input un array contenente un insieme di immagini e un vettore che specifica la nuova dimensione a cui ridimensionare le immagini.

```
1 function resizedImgs = resizeImages(images, newSize)
2     nImages = size(images, 4);
3
4     resizedImgs = zeros([newSize, nImages], 'like', images);
5
6     for i = 1:nImages
7         resizedImgs(:,:,:,i) = imresize(images(:,:,:,i), newSize(1:2));
8     end
9 end
```

- r. 2: si estrae il numero di immagini presenti nell'array, guardando la quarta dimensione dell'array.
- r. 4: si crea un array di 4 dimensioni, inizializzato a tutti 0.
- r. 6-8: il ciclo for scorre attraverso ciascuna immagine nell'array *images*. Per ogni immagine, utilizza la funzione *imresize* per ridimensionarla. Infine, l'immagine ridimensionata viene quindi salvata nella corrispondente posizione nell'array *resizedImgs*.

In conclusione, questa funzione prende un set di immagini, ridimensiona ciascuna immagine alla nuova dimensione specificata e restituisce il set di immagini ridimensionate.

1.1.3 Adattamento della rete pre-addestrata

In questa parte di codice si va a modificare la rete neurale convoluzionale pre-addestrata, *resnet18*, per adattarla al problema di classificazione in esame.

```
1  net = resnet18;
2
3  nClasses = 10;
4
5  layergraph = layerGraph(net);
6
7  newLearnableLayer = fullyConnectedLayer(nClasses, 'Name', 'new_fc', '
       WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10);
8  layergraph = replaceLayer(layergraph, 'fc1000', newLearnableLayer);
9
10 newClassLayer = classificationLayer('Name', 'new_classoutput');
11 layergraph = replaceLayer(layergraph, 'ClassificationLayer_predictions',
       newClassLayer);
```

- r. 1: si carica la rete neurale da utilizzare.
- r. 3: si specifica il numero di classi che si hanno all'interno del dataset.
- r. 5: si converte la rete net in un grafo di strati, permettendo di manipolare singoli strati della rete.
- r. 7-8: viene creato un nuovo livello completamente connesso con nClasses unità di output. Questo livello è configurato per avere un fattore di apprendimento dei pesi e dei bias aumentato di 10 volte rispetto al default.
- 10-11: viene creato un nuovo strato di classificazione. Questo nuovo strato sostituisce lo strato di classificazione originale della rete pre-addestrata.

A fine di questa sezione, si avrà la rete ResNet-18 modificata per un nuovo compito di classificazione con 10 classi, cambiato di livello completamente connesso finale e di livello di classificazione per adattarsi al nuovo numero di classi.
Inoltre, i fattori di apprendimento per i pesi e i bias del nuovo livello completamente connesso sono aumentati per accelerare l'addestramento di questi nuovi parametri.

1.1.4 Configurazione degli iperparametri

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize', 64, ...
    'MaxEpochs', 3, ...
    'InitialLearnRate', 1e-3, ...
    'Verbose', false, ...
    'Plots', 'training-progress');
```

Il codice configura le opzioni per l'addestramento di una rete neurale con l'algoritmo Stochastic Gradient Descent with Momentum.
Si usa un mini-batch di 64 campioni: ciò che significa che l'algoritmo di addestramento aggiornerà i pesi della rete dopo aver elaborato ogni gruppo di 64 esempi di addestramento.
Il numero di epoche massime è impostato a 3. Un'epoca è un ciclo completo attraverso tutto il dataset di addestramento.
Il tasso di apprendimento iniziale è impostato a 0.001. Questo determina di quanto i pesi della rete vengono aggiornati in ogni passaggio dell'ottimizzazione. Un tasso di apprendimento più alto accelera l'apprendimento ma può rendere l'addestramento instabile, mentre un tasso più basso rende l'addestramento più stabile ma più lento.
L'output dettagliato è disabilitato, disabilitando così la stampa di messaggi dettagliati durante l'addestramento.
Viene poi visualizzato un grafico del progresso dell'addestramento. Questo grafico mostra metriche come l'accuratezza o la perdita in funzione delle epoche, permettendo di monitorare visivamente come sta procedendo l'addestramento.

1.1.5 Allenamento e valutazione

Nell'ultima parte, dopo la preparazione della rete e del dataset, si inizia l'addestramento.

```
trainedNet = trainNetwork(augmentedTrainImages, layergraph, options);

YPred = classify(trainedNet, augmentedTestImages);

accuracy = mean(YPred == testLabels);
disp(['Accuratezza sul set di test: ', num2str(accuracy * 100), '%']);
```

- r. 1: viene utilizzata per addestrare la rete neurale: prende il set di immagini di addestramento, la struttura della rete e le opzioni di addestramento.
- r. 3: viene utilizzata per predire le etichette delle immagini di test utilizzando la rete neurale addestrata.
- r. 5: l'accuratezza del modello viene calcolata confrontando le predizioni *YPred* con le etichette reali *testLabels*.
- r. 6: si visualizza un messaggio che mostra l'accuratezza della rete sul set di test.

1.2 Risultati

Dopo aver usato il modello ResNet-18 pre-addestrato, adattato per la classificazione delle cifre nel dataset SVHN, è stata valutata l'accuratezza del modello sul set di test. L'accuratezza ottenuta rappresenta la percentuale di immagini del set di test per le quali il modello ha predetto correttamente la cifra.
Dopo l'esecuzione di 3 epoche di addestramento, il modello è stato valutato su un set di test separato, ottenendo un'accuratezza complessiva del 94,93%.
Questo risultato indica che il modello è riuscito a classificare correttamente 94,93% delle immagini nel set di test. L'accuratezza ottenuta è un indicatore della capacità del modello di generalizzare e riconoscere correttamente le cifre presenti in nuove immagini. Una percentuale di accuratezza elevata suggerisce che il modello è efficace nel risolvere il problema di classificazione delle cifre su dati mai visti prima.
Di seguito, si mostra il grafico dell'andamento dell'addestramento.
Il primo grafico mostra l'andamento della precisione durante le tre epoche di addestramento. La precisione aumenta rapidamente durante la prima epoca e si stabilizza nelle epoche successive, raggiungendo valori elevati, intorno al 90-100%.
Sono presenti due curve, una per la precisione smussata (linea più scura) e una per la precisione effettiva (linea più chiara), che mostrano la fluttuazione dei valori durante l'addestramento.
Il secondo grafico mostra che la perdita diminuisce rapidamente durante le prime iterazioni della prima epoca e continua a scendere in modo più graduale nelle successive.

Anche qui ci sono due curve: una per la perdita smussata (linea più scura) e una per la perdita effettiva (linea più chiara).
L'addestramento è durato in tutto 319 minuti e 21 secondi, dunque 5 ore e 20 minuti circa. In queste ore, si sono analizzate 3432 iterazioni, che hanno portato ai risultati finali visti prima.

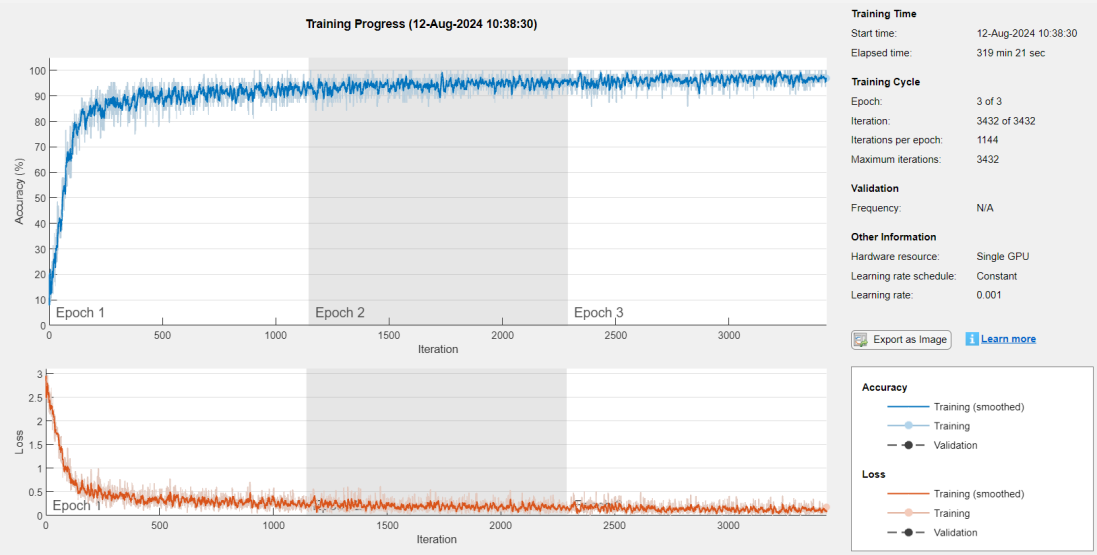

Figura 1.1: Andamento dell'addestramento.

Va notato che il valore dell'accuratezza può essere influenzato da diversi fattori, tra cui:

- **Qualità dei dati**: le immagini possono presentare variazioni di illuminazione, orientamento e qualità delle immagini.
- **Configurazione degli iperparametri**: gli iperparametri scelti influenzano direttamente la velocità e la stabilità della convergenza del modello.
- **Transfer Learning**: l'utilizzo di una rete pre-addestrata come ResNet-18 fornisce al modello una solida base di riconoscimento delle caratteristiche generali delle immagini.

I risultati ottenuti indicano che l'utilizzo di una rete neurale convoluzionale pre-addestrata rappresenta un approccio efficace per risolvere la classificazione delle immagini nel con-

testo del riconoscimento di cifre in ambienti reali.
Questo approccio può essere ulteriormente migliorato sperimentando con altre architetture di rete, tecniche di aumento dei dati, o ottimizzando ulteriormente gli iperparametri per massimizzare l'accuratezza e la generalizzazione del modello.