Progetto di Sistemi Embedded

Cheikh Cisse

Traccia n.11

1 Introduzione

Il progetto consiste nella realizzazione di una rete neurale per il task noto come semantic segmentation, si utilizzerà una rete neurale convoluzionale (CNN). In particolare, si utilizza una CNN che ha come input immagini di dimensione 128x128 pixel e come output una mappa di segmentazione delle immagini, in cui ogni pixel è associato ad una classe tra quelle possibili. La traccia proposta è la seguente:

Implementazione di una CNN per il task di semantic segmentation, su piattaforma embedded STM32F411 NUCLEO-64. La rete dovrà essere realizzata e testata dapprima su PC in linguaggio pythonbased; successivamente dovrà essere esportata e testata su piattaforma embedded (basata su core ARM Cortex-M4) tramite l'estensione STM XCUBE- AI dell'IDE STM32CUBEMX, dedicata all'intelligenza artificiale. (Si suggerisce di realizzare la rete utilizzando layers convoluzionali per compatibilità con il tool ST.)
Dataset:

- **MIT Scene Parsing Benchmark (SceneParse150) Dataset**

2 Sviluppo del codice python

Il codice inizia con l'importazione di diverse librerie utili per il progetto, tra cui la libreria os per operare con il sistema operativo, la libreria cv2 per elaborare le immagini, la libreria glob per eseguire la ricerca di file in una directory, la libreria numpy per la manipolazione di array, la libreria PIL per elaborare le immagini, la libreria matplotlib per visualizzare le immagini, la libreria tensorflow per la creazione della rete neurale convoluzionale, e la libreria segmentation_models per utilizzare una specifica architettura di rete neurale, in particolare sarà usata la UNet.
Successivamente, vengono definite alcune variabili utili per il progetto, come la dimensione delle immagini, il numero di classi, e i percorsi delle directory contenenti le immagini e le relative maschere.
Viene quindi creato un array che conterrà le immagini di training, e per ogni immagine nella directory di training, viene letto il file, ridimensionato alle dimensioni specificate, normalizzato, e infine aggiunto all'array. Lo stesso processo viene ripetuto per le maschere di training.
Le maschere vengono poi codificate utilizzando la funzione LabelEncoder della libreria sklearn. In particolare, le maschere vengono prima riformattate in un array monodimensionale, poi codificate in valori numerici utilizzando la funzione fit_transform di LabelEncoder, e infine riformattate in un array con le dimensioni originali in modo da poterle passsare alla rete nuerale.
Vengono poi creati due array di immagini e maschere di training, uno per il training vero e proprio e uno per la validazione. I due array vengono creati utilizzando la funzione train_test_split della libreria sklearn, che suddivide casualmente i dati in due insiemi in modo da poter valutare le performance della rete neurale su un insieme di dati non visti durante il training.
In seguito si definisce una funzione "to_one_hot" che prende in input un array di etichette e il

numero di classi e restituisce una rappresentazione one hot dell'array di etichette. La funzione prima limita i valori delle etichette all'intervallo [0, num_classes-1], quindi crea una matrice identità di dimensione num_classes, e infine seleziona i vettori corrispondenti alle etichette dalle righe della matrice identità.
Successivamente, la funzione viene utilizzata per trasformare due array di etichette di immagini, y_train e y_valid, in formato one hot. La forma dell'array di etichette viene modificata per avere una dimensione aggiuntiva per rappresentare le classi one hot. La nuova forma è quindi (n_immagini, altezza, larghezza, n_classi)
Viene infine definita la rete neurale convoluzionale utilizzando l'architettura UNet, che consiste in una serie di layer convoluzionali e di max-pooling per l'estrazione delle features, seguiti da una serie di layer convoluzionali di dimensioni sempre maggiori per la ricostruzione dell'immagine di segmentazione. Viene utilizzato come backbone della rete il modello ResNet34, che è stato già pre-addestrato su immagini di dimensioni simili a quelle del progetto.
Il modello è composto da un'input layer con dimensioni "size_x" e "size_y", seguito da una serie di strati di convoluzione, di dropout e di max pooling per l'encoder, e di strati di convoluzione trasposta, di concatenazione e di dropout per il decoder. Il modello è in grado di gestire immagini grayscale, poiché l'ultimo canale dell'input layer è impostato a 1.
L'ultimo layer di output è un'operazione di convoluzione con kernel di dimensione 1x1 e funzione di attivazione softmax, che restituisce una mappa di probabilità per ogni pixel dell'immagine di input.
Il modello viene istanziato come un'istanza della classe "Model" di Keras, passando l'input layer e l'output layer come argomenti.
Poi si definisce la funzione di perdita e le metriche di valutazione per un modello di deep learning utilizzando la libreria di segmentation models (sm).
La DiceLoss è una funzione di perdita utilizzata spesso per la segmentazione di immagini, che misura la sovrapposizione tra la maschera di segmentazione predetta e quella corretta. È un indicatore della similarità tra le due e può essere utilizzato per supervisione debole. In questo caso, la DiceLoss viene utilizzata con un vettore di pesi di classe uguale [0.25, 0.25, 0.25, 0.25, 0.25].
La CategoricalFocalLoss è un'altra funzione di perdita utilizzata spesso nella classificazione di immagini che pesa le classi in modo da concentrarsi sulle classi più difficili da predire. In questo caso, la CategoricalFocalLoss viene utilizzata con un peso di default.
La total_loss è la somma delle due funzioni di perdita con un peso di 1 assegnato alla CategoricalFocalLoss.
Le metriche di valutazione definite sono IOUScore e FScore. La IOUScore (Intersect over Union) è una metrica di valutazione che misura la sovrapposizione tra la maschera di segmentazione predetta e quella corretta. L'FScore è una metrica di valutazione che misura la precisione e la completezza delle previsioni.
Il modello viene quindi compilato con l'ottimizzatore Adam e la funzione di perdita personalizzata, insieme alle metriche di valutazione definite in precedenza (accuracy, IOUScore e FScore). Infine, viene stampato il sommario del modello.
La funzione di loss personalizzata permette di ottenere risultati migliori per quanto riguarda le metriche di valutazione che di solito vengono tenute in considerazione quando si fa la semantic segmentation, ovvero l'IOUScore e l'FScore. Mentre l'utilizzo della classica funzione di perdite: categorical_crossentropy permette di ottenere risultati leggermente migliori per quanto riguarda l'accuracy della rete in questione.
Poi si passa alla parte relativa all'addestramento del modello di deep learning utilizzato con la libreria keras. In particolare, viene utilizzata la funzione fit del modello per addestrarlo sui dati di addestramento x_train e y_train_cat, utilizzando una dimensione di batch di 32, per 40

epoche.
Inoltre, viene passato un set di dati di validazione (x_valid, y_valid_cat) per monitorare le prestazioni del modello sulla generalizzazione durante l'addestramento. Inoltre, viene utilizzato un oggetto di callback checkpoint per salvare il miglior modello ottenuto durante l'addestramento, basandosi sulla metrica di precisione sul set di validazione.
Il training del modello di rete è stato fatto sia in locale sia con il google colab, in locale riesco a caricare più immagini per la valutazione del modello a discapito dell'assenza di GPU, il colab è stato utilizzato per fare dei test veloci sulla rete per avere un'idea di come sta andando la rete neurale. Riporto ora lo screen delle ultime 10 epoche della fase di training del modello:

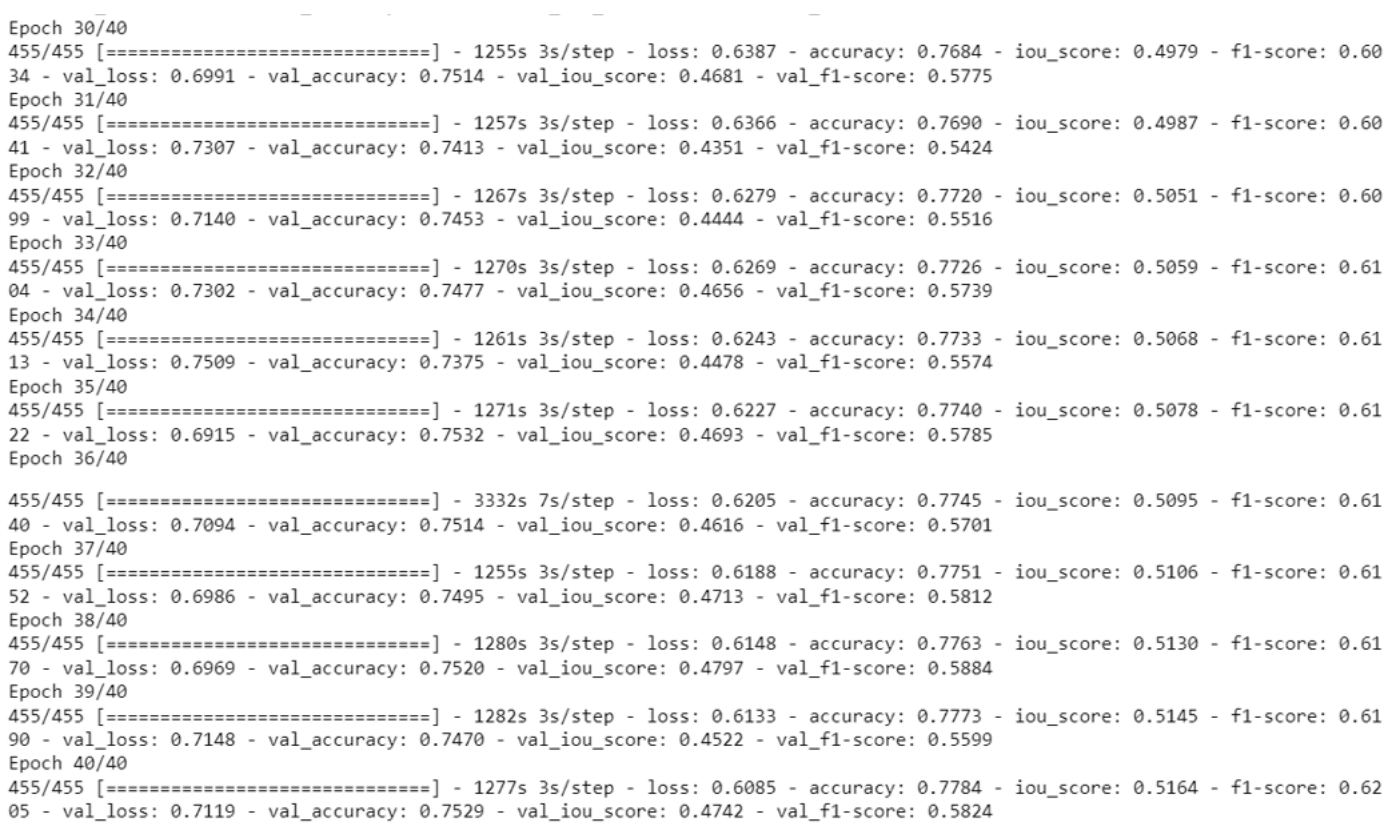
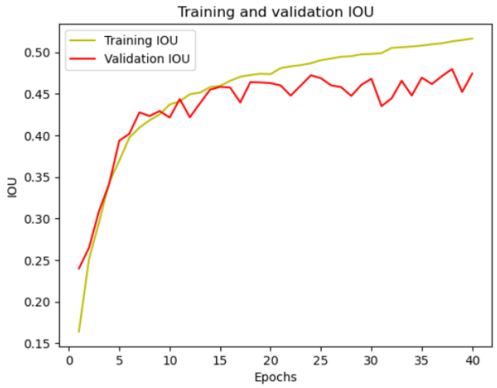


Figure 1: Training modello

Per questo training è stata usata la funzione di loss personalizzata perdendo circa 2 punti percentuale di accuracy, però ottenendo valori migliori di IOU e di F1Score. Si nota dall'immagine una validation accuracy pari al 75,20%, mentre si ha un IOU di cira il 48% e un F1Score di circa il 59%.
Dopo l'addestramento del modello, il codice utilizza la storia dell'addestramento (i.e., l'oggetto history) per tracciare la curva di apprendimento del modello, sia per la perdita (loss) che per la metrica di valutazione (IOU) sul set di addestramento e sul set di validazione. Questi sono stati i risultati:



In seguito, viene utilizzato il modello addestrato per effettuare una previsione su un esempio casuale x_valid[500] del set di validazione, e viene visualizzata l'immagine originale e l'immagine prevista dal modello. Si riporta quindi il risultato della segmentazione:

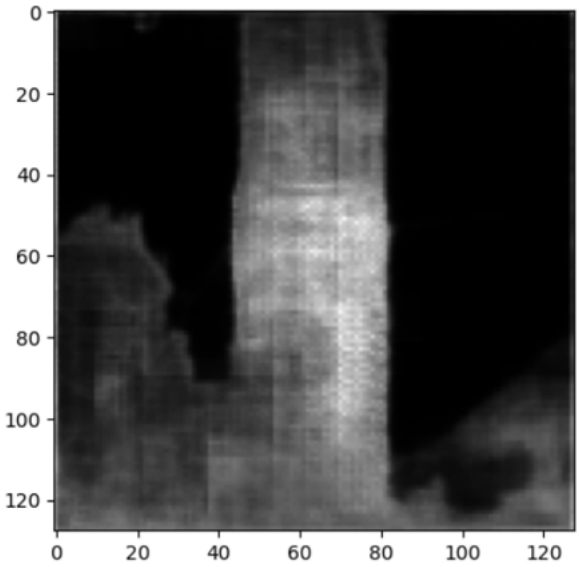


Figure 2: Segmentazione immagine di test

Infine, si definiscono due funzioni, get_file_size(file_path) e convert_bytes(size, unit=None), che vengono utilizzate per calcolare e stampare la dimensione di un file in byte, KB o MB.
Successivamente, si converte un modello Keras (model) in un modello TensorFlow Lite utilizzando il convertitore TFLiteConverter fornito da TensorFlow Lite. Il modello risultante viene salvato in un file TFLite chiamato "progetto.tflite", nella directory corrente.
Quindi si utilizzano le funzioni get_file_size() e convert_bytes() per calcolare e stampare la dimensione del file del modello TensorFlow Lite salvato, in megabyte (MB).
In seguito, si converte il modello Keras model in un modello TensorFlow Lite con quantizzazione a intervallo dinamico utilizzando il convertitore TFLiteConverter. L'ottimizzazione predefinita viene abilitata per quantizzare tutti i parametri fissi (ad esempio i pesi). Il modello quantizzato viene salvato in un file TFLite chiamato "progetto_quantDyn.tflite", nella directory corrente.
Infine, si utilizzano le funzioni get_file_size() e convert_bytes() per calcolare e stampare la dimensione del file del modello TensorFlow Lite quantizzato con intervallo dinamico salvato, in megabyte (MB).

3 Estrazione su board

Parte finale del progetto è l'estrazione del modello di rete neurale, la rete viene quindi portata su piattaforma embedded, la scheda fornita è la STM32F411 Nucleo-64, di seguito l'immagine della scheda:

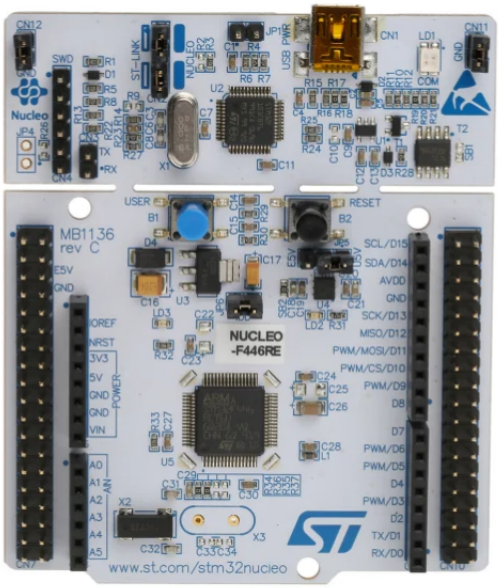


Figure 3: STM32F411 Nucleo-64

A causa dei problemi con il TFLite, che si verificano con le reti che effettuano segmentazione, è stato utilizzato il file in h5. Questo mi ha portato a dover diminuire ulteriormente i parametri della rete, visto che è una quantizzazione più debole rispetto al TFLite. La validation on desktop è andata a buon fine, l'unico problema è dovuto al fatto che il CUBEIDE non riesce a classificare la mia rete neurale come classificatore, altro problema che si verifica con le reti che effettuano la semantic segmentation. A causa di questo problema non riesce a calcolare le metriche necessarie per definire l'accuratezza della rete, ma guardando l'errore che mostra si nota che è basso quindi si può concludere che si abbia più o meno la stessa accuratezza e che sia effettivamente buona.
Per quanto riguarda la validation on target ho riscontrato problemi, in particolare si è verificato un errore sui driver del pc per la connessione con la board. Però visto il corretto andamento dell'analyze e della validation on desktop si può ritenere che anche quella on target sarebbe andata a buon fine.