Progetto per il corso di Machine Learning: Rilevamento di incendi su un piccolo set di dati con ResNet-50

Valeria Passerini, 489351

Sommario

Il lavoro svolto ha come obiettivo quello di fornire un modello autonomo per la rilevazione della presenza di incendi su un set di immagini, tramite l'applicazione di tecniche di apprendimento automatico su un problema di classificazione binaria, o regressione logistica. In particolare sono state adottate techiche di apprendimento profondo e modelli di reti neurali per il processamento delle immagini e la classificazione di esse. Il problema della classificazione è stato affrontato attraverso la minimizzazione di una opportuna funzione di costo e l'aggiornamento dei parametri del modello attraverso la tecnica di discesa del gradiente e della retropropagazione, implementate attraverso le librerie di TensorFlow. Il codice è disponibile su Github al seguente link.

1 Introduzione

Gli incendi rappresentano una delle minacce più gravi per l'ambiente naturale e per la vita sulla Terra. Ogni anno, milioni di ettari di foreste, savane e vegetazione vengono devastati dalle fiamme, infliggendo danni irreparabili agli ecosistemi, alle risorse naturali e alla biodiversità. Questi incendi possono essere innescati da una serie di fattori, tra cui attività umane irresponsabili, cambiamenti climatici, condizioni meteorologiche estreme e, in alcuni casi, fenomeni naturali come i fulmini. La loro diffusione incontrollata rappresenta non solo una minaccia immediata per la flora e la fauna, ma anche per la sicurezza delle comunità locali e per la qualità dell'aria e dell'acqua nelle aree colpite. Con il cambiamento climatico e la frequenza sempre più crescente con cui si verificano tali minaccie la ricerca si sta interessando sempre più a metodi automatici per prevenire, rilevare e mitigare tali minaccie. Il progetto proposto ha come scopo quello di mostrare come sia possibile realizzare soluzioni che possano aiutare in questa direzione.

2 Metodo

La classificazione binaria, è una tecnica di machine learning che si occupa di assegnare un'osservazione a una delle due classi possibili. Nella classificazione binaria, il modello viene addestrato su un insieme di dati di addestramento contenente esempi etichettati, cioè osservazioni conosciute che sono già state categorizzate nelle due classi. Utilizzando queste informazioni, il modello impara a distinguere tra le due classi e a fare previsioni su nuovi dati non visti in precedenza e quindi a generalizzare correttamente su questi.

Dato un insieme di punti in uno spazio n-dimensionale, classificati in due categorie (0 classe negativa, 1 classe positiva), l'obiettivo è quello di trovare il decision boundary, ossia la curva che divide le due classi. Il problema della regressione logistica è un problema di apprendimento supervisionato utilizzato per la classificazione binaria, cioè quando si devono classificare le osservazioni in una di due categorie possibili. La regressione logistica utilizza la funzione logistica (detta anche sigmoide) per trasformare la somma pesata dei valori delle caratteristiche in un valore compreso tra 0 e 1, che rappresenta la probabilità che un'osservazione appartenga alla classe positiva. La funzione logistica è definita come:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

dove z è la somma pesata delle caratteristiche, definita come:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

dove $w_0, w_1, \ldots, w_n$ sono i pesi associati a ciascuna caratteristica, $x_1, x_2, \ldots, x_n$ sono i valori delle caratteristiche dell'osservazione. La funzione logistica restituisce valori compresi tra 0 e 1 e può essere interpretata come la probabilità che un'osservazione appartenga alla classe positiva. Un valore maggiore di 0.5 indica che l'osservazione è più probabile di appartenere alla classe positiva, mentre un valore minore di 0.5 indica che è più probabile appartenere alla classe negativa.

Durante l'addestramento del modello di regressione logistica, i pesi $w_0, w_1, \ldots, w_n$ vengono ottimizzati per massimizzare la verosimiglianza dei dati di addestramento. Questo viene fatto minimizzando una funzione di costo, spesso chiamata cross-entropy loss function, definita come:

$$J(w) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

dove m è il numero di osservazioni nei dati di addestramento, y_i è l'etichetta della classe per l'osservazione i (0 o 1) e $\hat{y}_i$ è la previsione del modello per l'osservazione i.

3 Dataset

Il dataset preso in esame per l'addestramento e la valutaione del modello è denominato Fire Detection[1], il quale contiene una vasta gamma di immagini contenenti scenari con e senza incendi, fornendo così una base per lo sviluppo e la valutazione del modello di rilevamento degli incendi. Il dataset utilizzato viene fornito dalla piattaforma Kaggle, progettato per supportare la ricerca e lo sviluppo nel campo del rilevamento degli incendi. Questo dataset contiene una varietà di immagini, 651 in totale, alcune mostrate in Figura 1 che rappresentano sia scenari con la presenza di fuoco che scenari senza fuoco. Le immagini possono comprendere ambienti interni ed esterni, varie condizioni di illuminazione e diversi tipi di fonti di fuoco. Ogni immagine è stata etichettata in base alla presenza o all'assenza di fuoco rispettivamente con 1 e 0, fornendo un set di dati bilanciato per l'addestramento e la valutazione dei modelli di rilevamento degli incendi.

4 Implementazione

Il problema è stato affrontato tramite l'API di TensorFlow, una libreria open-source per il machine learning e l'intelligenza artificiale e si basa su una implementazione originale reperita online[2].

Vengono poi pre-processate le immagini tramite la classe `tf.keras.preprocessing.image.ImageDataGenerator` di tensorflow che può applicare trasformazioni casuali alle immagini durante l'addestramento, come rotazioni, zoom, traslazioni, riflessioni e regolazioni del contrasto. Questo aiuta ad aumentare la varietà dei dati di addestramento e ad evitare il sovra-addestramento del modello.

Viene utilizzata una rete neurale convoluzionale, per l'identificazione automatizzata della presenza di fuoco nelle immagini. E' un modello pre-addestrato usato come base di partenza per addestrare il nuovo modello, una tecnica chiamata comunemente in gergo, "apprendimento per trasferimento". A questo punto vengono costruiti i livelli superiori utilizzando tre librerie di Tensorflow: Sequential, DropOut e Dense.

- **Sequential** (`tf.keras.Sequential`) è un tipo di modello di rete neurale sequenziale di Tensorflow dove i layer vengono aggiunti uno dopo l'altro in sequenza. E' comune nelle reti neurali dove l'output di ciascun layer diventa l'input per il successivo.

- **Dropout** (`tf.keras.layers.Dropout`) è una tecnica di regolarizzazione utilizzata nelle reti neurali per prevenire l'overfitting. Durante l'addestramento, l'unità di dropout "disattiva" casualmente un determinato numero di nodi in un layer con una certa probabilità, che in questo caso è 30%. Questo "disattivamento" temporaneo forza la rete neurale a imparare features più robuste e generalizzabili, riducendo così il rischio di overfitting. Durante il test o la valutazione, il dropout non viene applicato e tutti i nodi rimangono attivi, ma i pesi dei nodi sono scalati di conseguenza per mantenere la coerenza.

[1] Il dataset è scaricabile all'indirizzo https://www.kaggle.com/datasets/christofel04/fire-detection-dataset.
[2] https://www.kaggle.com/code/jvkchaitanya410/fire-detection-using-resnet-50-accuracy-97

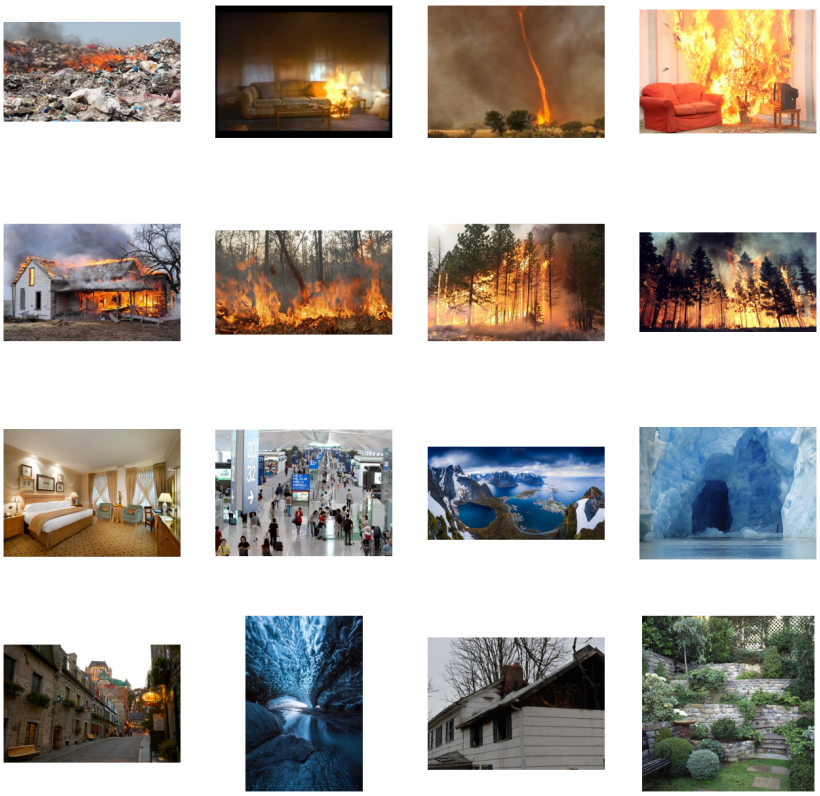

Figura 1: 8 immagini campione, metà con la presenza di fuoco e l'altra metà senza.

- **Dense** (`tf.keras.layers.Dense`) è un tipo di layer completamente connesso, il che significa che ogni nodo del layer precedente è connesso a ogni nodo del layer successivo. Ogni nodo in un layer Dense esegue una combinazione lineare dei suoi input, seguita da una funzione di attivazione non lineare, una ReLU (Rectified Linear Unit, `tf.keras.layers.ReLU`) nel caso in esame. L'ultimo livello della rete è un layer Dense con funzione di attivazione sigmoide (`tf.keras.layers.Sigmoid`).

Durante l'addestramento del modello è fondamentale utilizzare una funzione di errore appropriata per valutare quanto le previsioni del modello si discostino dalle etichette di classe reali. Nel caso in esame è stato scelto di utilizzare la funzione di errore *Binary Cross Entropy* (`tf.keras.losses.BinaryCrossentropy`). Questa funzione valuta la differenza tra le previsioni generate dal modello e le etichette di classe effettive, fornendo un feedback al modello su quanto si avvicini alla verità durante l'addestramento. Si vuole minimizzare questa funzione di errore durante l'addestramento per ottenere previsioni accurate.

Il metodo *fit* (`model.fit()`) di TensorFlow è uno strumento utilizzato per avviare l'addestramento del modello utilizzando i dati di addestramento forniti. Con questo metodo, possiamo specificare i dati di addestramento, il numero di epoche, ovvero il numero di volte che il modello attraversa l'intero set di dati durante l'addestramento e il batch size per controllare il processo di addestramento, quindi il numero di esempi utilizzati per aggiornare i pesi del modello in ogni iterazione dell'addestramento. Inoltre, possiamo passare *callback* per monitorare l'andamento dell'addestramento e applicare tecniche di regolarizzazione o eseguire azioni personalizzate durante l'addestramento, come ad esempio, nel caso in esame, salvare i pesi del modello con errore minore sul set di validazione alla fine dell'epoca di training attraverso la libreria di checkpoint (`tf.keras.callbacks.ModelCheckpoint`).

Il metodo *compile* (`model.compile()`) di Tensorflow ci permette di impostare l'ottimizzatore utilizzato nel training, il tipo di funzione di errore e la metrica desiderata. Per aggiornare i pesi del modello durante l'addestramento è stato utilizzato l'ottimizzatore Stochastic Gradient Descent (`tf.keras.optimizers.SGD`) per minimizzare la funzione di errore durante l'addestramento. E' stato configurato l'ottimizzatore impostando il tasso di apprendimento e altri iperparametri per regolare il

comportamento dell'ottimizzatore durante l'addestramento.

Dopo l'addestramento del modello, è stata utilizzata la funzione *evaluate* per valutare le prestazioni del modello su dati non visti in precedenza (validation set). E' stata utilizzata la metrica di *Accuracy* (`tf.keras.metrics.Accuracy`) per valutare quanto il modello sia accurato nel fare previsioni sul set di validazione. Questa metrica ci fornisce una misura del grado di correttezza delle previsioni del modello rispetto alle etichette di classe reali.

Per monitorare l'andamento dell'addestramento e visualizzare i grafici dell'andamento della funzione di errore, è stato scelto TensorBoard, una libreria fornita da TensorFlow per la visualizzazione e l'analisi dei dati di addestramento. Durante l'addestramento del modello, sono stati registrati i dati di addestramento, inclusi la funzione di errore e altre metriche di interesse, utilizzando TensorBoard. Questi dati sono stati quindi visualizzati in grafici interattivi e dashboard, consentendo di analizzare l'andamento del modello durante l'addestramento, identificare eventuali problemi e ottimizzare le prestazioni del modello.

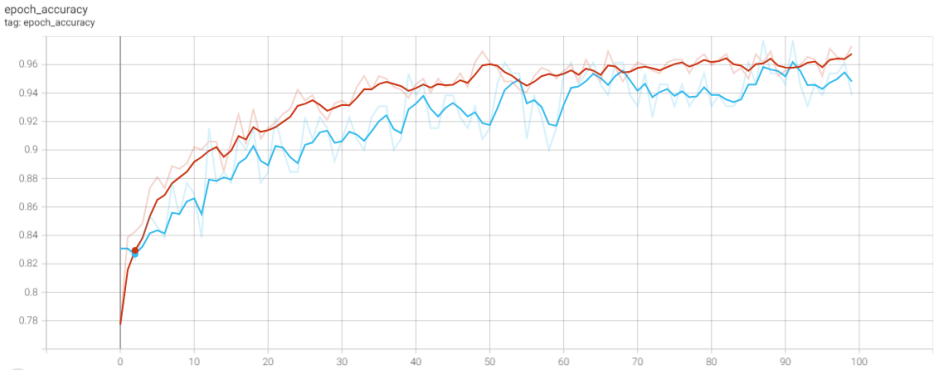

Figura 2: l grafico mostra l'andamento della metrica di accuratezza sulle ordinate all'aumentare delle epoche di allenamento. Il colore arancione corrisponde alla misurazione relativa al set di addestramento mentre il blu corrisponde al set di validazione.

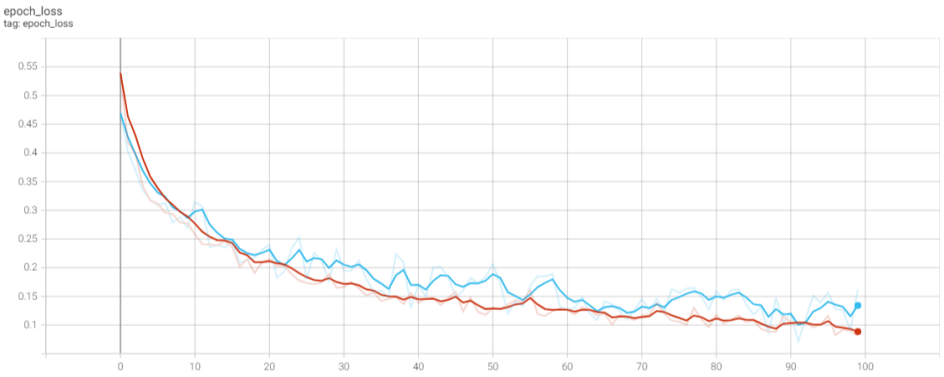

Figura 3: Il grafico mostra l'andamento della funzione di errore sulle ordinate all'aumentare delle epoche di allenamento. Il colore arancione corrisponde alla misurazione relativa al set di addestramento mentre il blu corrisponde al set di validazione.

4.1 ResNet-50

ResNet50 [HZRS15] è una rete neurale convoluzionale (CNN) che è stata progettata per l'addestramento su grandi insiemi di dati di immagini per compiti di classificazione. È parte della famiglia di reti ResNet (Residual Networks) e si distingue per la sua architettura innovativa che utilizza blocchi residui per consentire la costruzione di reti neurali più profonde senza incorrere nel problema della scomparsa del gradiente. ResNet50 è composta da 50 strati, inclusi strati convoluzionali, strati di

Dataset	Cross Entropy Loss	Accuracy
Fire-Detection val	0.148	93.8%

Tabella 1: La tabella mostra i risultati in errore e accuratezza ottenuti dall'addestramento di una rete neurale ResNet-50 sul problema del rilevamento di incendi nelle immagini.

normalizzazione del batch e strati di pooling. Utilizza convoluzioni 3x3 e 1x1 per estrarre features da immagini di input in diverse scale e livelli di astrazione. Inoltre, include strati di normalizzazione del batch per accelerare e migliorare la convergenza dell'addestramento. È stata addestrata su un ampio set di dati di immagini, come ImageNet, ed i pesi pre addestrati sono disponibili per l'utilizzo in TensorFlow (`tf.keras.applications.ResNet50`).

5 Esperimeti e Risultati

Per prima cosa è stata scelta la rete convoluzionale ResNet-50 pre-addestrata su Imagenet. Sono stati congelati i primi 45 layer e lasciati gli ultimi layer addestrabili per essere adattati al dataset specifico, secondo le tecniche usate normalmente per l'addestramento per trasferimento. Per il pre-processamento della immagini sono stati utilizzati i metodi pre-definiti di `tf.keras.preprocessing.image.ImageDataGenerator`. In questo caso viene fornita una probabilità del 50% in cui l'immagine può essere traslata orizzontalmente e un'altra del 20% in cui può essere traslata verticalmente. Inoltre, viene scelto il 20% delle immagini come quelle che verranno utilizzate per il set di validazione (lasciando il restante 80% per l'addestramento effettivo del modello). La dimensione delle immagini RGB in input in seguito al ridimensionamento è 300×300 e viene scelto un numero di epoche di addestramento uguale a 100 con learning rate di 10^{-5} e con batch size uguale a 8. L'addestramento è stato eseguito su una CPU AMD Ryzen 7 5700U. I risultati dell'ottimizzazione sono riportati in Tabella 1, l'andamento della funzione di errore in Figura 3 e della metrica di accuratezza in Figura 2.

Riferimenti bibliografici

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.