**Fish Finder**

**STF e Funzionamento**

|  |  |  |
| --- | --- | --- |
| Versione Documento |  | Data Documento |
| 1.0 |  | 10/06/2024 |
|  |  |  |

SOMMARIO

[1 STORIA DEL DOCUMENTO 3](#_Toc169123979)

[2 INTRODUZIONE 4](#_Toc169123980)

[2.1 Definizioni, abbreviazioni, acronimi 4](#_Toc169123981)

[2.1.1 Definizioni 4](#_Toc169123982)

[2.1.2 Acronimi 4](#_Toc169123983)

[2.2 Documenti di riferimento 4](#_Toc169123984)

[3 INFORMAZIONI GENERALI 5](#_Toc169123985)

[4 Impostazione progetto 5](#_Toc169123986)

[4.1 Tecnologie utilizzate 5](#_Toc169123987)

[4.1.1 Linguaggio 5](#_Toc169123988)

[4.1.2 Librerie 5](#_Toc169123989)

[4.1.3 Ambiente di Sviluppo 5](#_Toc169123990)

[5 Componenti 5](#_Toc169123991)

[5.1 Dataset di immagini 5](#_Toc169123992)

[5.2 Trasformazione delle immagini 5](#_Toc169123993)

[5.3 Rete Neurale Convoluzionale (CNN) 6](#_Toc169123994)

[5.4 Addestramento del Modello 6](#_Toc169123995)

[5.5 Validazione e Test 6](#_Toc169123996)

[5.6 Inferenza 6](#_Toc169123997)

[6 ResNet18 6](#_Toc169123998)

[6.1 Architettura profonda e semplice 6](#_Toc169123999)

[6.2 Prestazioni 6](#_Toc169124000)

[6.3 Efficienza Computazionale 7](#_Toc169124001)

[6.4 Flessibilità 7](#_Toc169124002)

[6.5 Funzione di attivazione 7](#_Toc169124003)

[6.5.1 Caratteristiche di ReLu 7](#_Toc169124004)

[7 Pseudo-Labeling 8](#_Toc169124005)

[7.1 Preaddestramento 8](#_Toc169124006)

[7.2 Generazione delle Pseudo-Etichette 8](#_Toc169124007)

[7.3 Combinare i dati 8](#_Toc169124008)

[7.4 Ri-addestramento 8](#_Toc169124009)

[8 Addestramento e Risultati 8](#_Toc169124010)

[8.1 Primo Training (immagini etichettate) 8](#_Toc169124011)

[8.1.1 Performance Training (immagini etichettate) 9](#_Toc169124012)

[8.1.2 Performance Validazione (Immagini etichettate) 10](#_Toc169124013)

[8.1.3 Conclusioni (Immagini etichettate) 10](#_Toc169124014)

[8.2 Secondo Training (tutte le immagini) 10](#_Toc169124015)

[8.2.1 Performance Training (tutte le immagini) 11](#_Toc169124016)

[8.2.2 Performance Validazione (tutte le immagini) 11](#_Toc169124017)

[8.3 Conclusioni finali 12](#_Toc169124018)

[9 Test 12](#_Toc169124019)

[10 PUNTI DI ATTENZIONE 14](#_Toc169124020)

[11 ALLEGATI 14](#_Toc169124021)

# STORIA DEL DOCUMENTO

Le informazioni contenute in questo documento aggiornano e sostituiscono quelle contenute nelle versioni precedenti. Ogni modifica del documento sarà controllata nella apposita procedura.

|  |  |  |  |
| --- | --- | --- | --- |
| **Versione** | **Data** | **Descrizione della Revisione** | **Modificato da** |
| 1.0 | 10/06 /2024 | Prima versione | Carlo Blatti |

# INTRODUZIONE

## Definizioni, abbreviazioni, acronimi

### Definizioni

### Acronimi

* CNN: Convolutional Neural Network
* ReLU: Rectified Linear Unit
* STF: Specifiche Tecnico Funzionali

## Documenti di riferimento

# INFORMAZIONI GENERALI

Il progetto sviluppato è un **classificatore di immagini** basato su una rete neurale convoluzionale (CNN). Questo tipo di modello è utilizzato per assegnare un'etichetta predefinita (classe) a ciascuna immagine del dataset, imparando a riconoscere le caratteristiche visive distintive che definiscono ciascuna classe. In particolare le immagini su cui è stato allenato il modello sono immagini di pesci.

# Impostazione progetto

Il progetto dell’applicativo è stato sviluppato utilizzando Colab come ambiente di sviluppo e runtime.

## Tecnologie utilizzate

### Linguaggio

* **Python**: Il linguaggio di programmazione principale utilizzato per implementare il classificatore di immagini.

### Librerie

* **Librerie**:
  + **PyTorch**: Libreria principale per il deep learning e la costruzione della rete neurale.
  + **Torchvision**: Utilizzata per gestire dataset di immagini e modelli pre-addestrati.
  + **Pandas**: Per la manipolazione e analisi dei dati tabulari (file CSV).
  + **Matplotlib**: Per la visualizzazione dei dati, come curve di accuratezza e perdita.
  + **scikit-learn**: Per la suddivisione dei dati in set di addestramento e validazione.

### Ambiente di Sviluppo

* **Google Colab**: Ambiente di notebook basato su cloud che offre accesso gratuito a GPU, utilizzato per eseguire il codice Python e accelerare l'addestramento del modello.

# Componenti

## Dataset di immagini

* Le immagini sono fornite in un file CSV che contiene i nomi dei file e le rispettive classi.
* Le immagini vengono suddivise in un set di addestramento e un set di validazione.

## Trasformazione delle immagini

Le immagini sono preprocessate tramite trasformazioni che includono ridimensionamento, ritaglio e normalizzazione per migliorare la robustezza e le prestazioni del modello.

## Rete Neurale Convoluzionale (CNN)

* **ResNet18**: Un modello pre-addestrato di rete neurale convoluzionale, noto per la sua capacità di apprendere rappresentazioni profonde delle immagini.
* **Fine-tuning**: Modifica dell'ultimo livello del modello per adattarsi al numero specifico di classi nel dataset.

## Addestramento del Modello

* Utilizzo di tecniche di ottimizzazione come l'algoritmo Adam per aggiornare i pesi del modello.
* Mixed Precision Training per velocizzare l'addestramento e ridurre il consumo di memoria.

## Validazione e Test

* Monitoraggio delle prestazioni del modello su un set di validazione per evitare il sovra-addestramento.
* Visualizzazione delle curve di perdita e accuratezza per valutare il comportamento del modello durante l'addestramento.

## Inferenza

* Test del modello su immagini di validazione per verificare l'accuratezza delle previsioni.
* Visualizzazione delle immagini con le predizioni del modello e confronto con le etichette corrette.

# ResNet18

ResNet18 è stata scelta per questo progetto di classificazione delle immagini per diversi motivi che la rendono particolarmente adatta.

## Architettura profonda e semplice

* **Reti Residuali**: ResNet sta per "Residual Network". L'architettura di ResNet introduce connessioni residue che aiutano a mitigare il problema del gradiente vanishing nelle reti profonde, consentendo un addestramento più efficace di reti profonde.
* **Profondità Moderata**: ResNet18 è una versione relativamente leggera con 18 strati. È meno complessa rispetto alle versioni più profonde, ma offre comunque una capacità di apprendimento sufficiente per molti compiti di classificazione delle immagini.

## Prestazioni

* **Pre-addestramento su ImageNet**: ResNet18 viene solitamente pre-addestrata su ImageNet, un dataset molto grande e variegato. Questo pre-addestramento permette al modello di iniziare con caratteristiche già ben definite, accelerando il processo di fine-tuning e migliorando le prestazioni anche su dataset più piccoli o specifici.
* **Accuratezza**: ResNet18 ha dimostrato di ottenere buoni risultati su una vasta gamma di compiti di visione artificiale, offrendo un buon equilibrio tra accuratezza e complessità del modello.

## Efficienza Computazionale

* **Velocità**: Rispetto a modelli più complessi e profondi, ResNet18 è più veloce sia nell'addestramento che nell'inferenza, il che è particolarmente importante quando si utilizza Google Colab, dove le risorse computazionali possono essere limitate.
* **Efficienza della Memoria**: ResNet18 richiede meno memoria rispetto alle versioni più profonde di ResNet, rendendola una scelta pratica per ambienti con risorse limitate.

## Flessibilità

* **Facilità di Adattamento**: ResNet18 può essere facilmente modificata per adattarsi a diverse classi di output. Nel progetto, l'ultimo livello del modello è stato sostituito per adattarsi al numero specifico di classi del dataset, rendendo il modello versatile per vari compiti di classificazione.
* **Compatibilità con le Tecniche di Ottimizzazione**: Supporta facilmente tecniche di ottimizzazione come il mixed precision training (torch.cuda.amp), che migliora ulteriormente l'efficienza dell'addestramento.

## Funzione di attivazione

La funzione di attivazione utilizzata nel modello ResNet18 è **ReLU**. Questa funzione di attivazione è integrata nei blocchi della rete ed è fondamentale per le sue capacità di addestramento, specialmente in reti profonde, grazie alla sua semplicità, efficienza computazionale, e capacità di mitigare il problema del gradiente vanishing.

### Caratteristiche di ReLu

* **Semplicità**: La funzione ReLU è molto semplice da calcolare, essendo lineare per tutti i valori positivi e zero per i valori negativi.
* **Sparsità**: ReLU introduce sparsità nelle attivazioni, poiché molti dei valori negativi vengono mappati a zero. Questo può migliorare l'efficienza computazionale.
* **Mitigazione del Problema del Gradiente Vanishing**: ReLU aiuta a mitigare il problema del gradiente vanishing, che può essere prevalente in reti profonde, mantenendo i gradienti significativi per attivazioni positive.

# Pseudo-Labeling

Oltre alle immagini etichettate, il cui nome e relativa classe sono contenute nel csv di training, si è voluto usare un set di immagini non etichettate.

L'inclusione delle immagini non etichettate attraverso lo pseudo-labeling permette di utilizzare una quantità maggiore di dati, migliorando potenzialmente le prestazioni del modello. Questo approccio sfrutta la capacità del modello di generalizzare e apprendere dalle nuove informazioni, portando a una migliore accuratezza e robustezza nel riconoscimento delle immagini.

## Preaddestramento

Il modello è stato inizialmente addestrato utilizzando solo i dati etichettati. Questo fornisce una base solida di caratteristiche apprese.

## Generazione delle Pseudo-Etichette

Le immagini non etichettate sono state passate attraverso il modello addestrato per generare etichette predette.

## Combinare i dati

I dati etichettati originali e quelli non etichettati con pseudo-etichette sono stati combinati per creare un dataset più grande.

## Ri-addestramento

Il modello è stato ri-addestrato utilizzando questo dataset combinato, potenzialmente migliorando le sue prestazioni grazie alla maggior quantità di dati disponibili.

# Addestramento e Risultati

Il training è durato 1m e 32 per il dataset contenente le immagini etichettate e 35m e 17s per il dataset completo di tutte le immagini.

## Primo Training (immagini etichettate)

Alla fine delle 10 epoche del primo training si sono ottenuti i seguenti risultati:

Epoch 0/9

----------

train Loss: 2.5358 Acc: 0.2250

val Loss: 2.3280 Acc: 0.2000

Epoch 1/9

----------

train Loss: 0.9321 Acc: 0.7500

val Loss: 1.9768 Acc: 0.3333

Epoch 2/9

----------

train Loss: 0.5385 Acc: 0.8750

val Loss: 2.2904 Acc: 0.3000

Epoch 3/9

----------

train Loss: 0.3172 Acc: 0.9167

val Loss: 2.8322 Acc: 0.3667

Epoch 4/9

----------

train Loss: 0.3579 Acc: 0.9167

val Loss: 2.4085 Acc: 0.4667

Epoch 5/9

----------

train Loss: 0.2903 Acc: 0.9167

val Loss: 1.3135 Acc: 0.7667

Epoch 6/9

----------

train Loss: 0.2692 Acc: 0.9083

val Loss: 3.0307 Acc: 0.5667

Epoch 7/9

----------

train Loss: 0.3656 Acc: 0.9083

val Loss: 7.3186 Acc: 0.2667

Epoch 8/9

----------

train Loss: 0.2201 Acc: 0.9500

val Loss: 8.5100 Acc: 0.2000

Epoch 9/9

----------

train Loss: 0.2954 Acc: 0.9083

val Loss: 8.5215 Acc: 0.3000

Training complete in 1m 32s

Best val Acc: 0.7667

Immagine che contiene testo, diagramma, Diagramma, linea

Descrizione generata automaticamente

### Performance Training (immagini etichettate)

* La perdita (Loss) sui dati di addestramento diminuisce rapidamente da 2.5358 a valori intorno a 0.2201-0.3656.
* L'accuratezza (Acc) sui dati di addestramento aumenta rapidamente da 0.2250 a valori tra 0.9083 e 0.9500.

### Performance Validazione (Immagini etichettate)

* La perdita sui dati di validazione mostra un comportamento variabile, inizialmente diminuendo ma poi aumentando significativamente, raggiungendo valori molto alti come 7.3186 e 8.5100.
* L'accuratezza sui dati di validazione inizia a 0.2000 e aumenta fino a 0.7667 per poi calare drasticamente, tornando a 0.3000.

### Conclusioni (Immagini etichettate)

* La perdita sui dati di validazione mostra un comportamento variabile, inizialmente diminuendo ma poi aumentando significativamente, raggiungendo valori molto alti come 7.3186 e 8.5100.
* L'accuratezza sui dati di validazione inizia a 0.2000 e aumenta fino a 0.7667 per poi calare drasticamente, tornando a 0.3000.
* C'è un miglioramento iniziale nella performance sui dati di validazione fino a un'accuratezza di 0.7667 alla quinta epoca. Tuttavia, questo miglioramento non è sostenuto nelle epoche successive.

Il modello mostra un chiaro segno di overfitting, con una performance molto buona sui dati di addestramento ma scarsa capacità di generalizzare ai dati di validazione.

## Secondo Training (tutte le immagini)

Alla fine delle 10 epoche del secondo training si sono ottenuti i seguenti risultati:

Epoch 0/9

----------

train Loss: 1.2140 Acc: 0.5841

val Loss: 1.8214 Acc: 0.4333

Epoch 1/9

----------

train Loss: 1.0478 Acc: 0.6344

val Loss: 2.1608 Acc: 0.3667

Epoch 2/9

----------

train Loss: 0.9879 Acc: 0.6522

val Loss: 2.3595 Acc: 0.3000

Epoch 3/9

----------

train Loss: 0.9590 Acc: 0.6637

val Loss: 2.3528 Acc: 0.4667

Epoch 4/9

----------

train Loss: 0.9293 Acc: 0.6710

val Loss: 1.8443 Acc: 0.3667

Epoch 5/9

----------

train Loss: 0.9000 Acc: 0.6782

val Loss: 1.4038 Acc: 0.4667

Epoch 6/9

----------

train Loss: 0.8803 Acc: 0.6874

val Loss: 1.6182 Acc: 0.5333

Epoch 7/9

----------

train Loss: 0.8700 Acc: 0.6899

val Loss: 1.5088 Acc: 0.5000

Epoch 8/9

----------

train Loss: 0.8458 Acc: 0.7010

val Loss: 1.5987 Acc: 0.5333

Epoch 9/9

----------

train Loss: 0.8254 Acc: 0.7074

val Loss: 1.2500 Acc: 0.6333

Training complete in 35m 17s

Best val Acc: 0.6333

Immagine che contiene diagramma, Diagramma, linea, testo

Descrizione generata automaticamente

Validation Loss: 1.2501 Acc: 0.6333

### Performance Training (tutte le immagini)

* La perdita sui dati di addestramento diminuisce costantemente da 1.2140 a 0.8254.
* L'accuratezza sui dati di addestramento aumenta costantemente da 0.5841 a 0.7074.

### Performance Validazione (tutte le immagini)

* La perdita sui dati di validazione mostra un comportamento fluttuante inizialmente, ma con una tendenza generale alla diminuzione da 1.8214 a 1.2500.
* L'accuratezza sui dati di validazione varia, con un miglioramento significativo nell'ultima epoca, raggiungendo un valore finale di 0.6333.

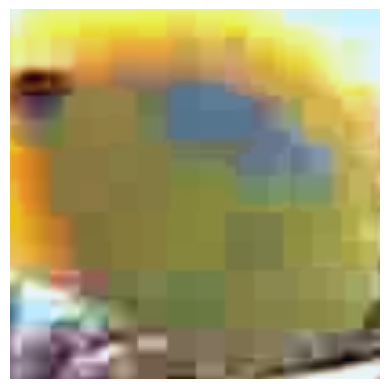
## Conclusioni finali

* Inizialmente, sembra esserci una tendenza all'overfitting. Nelle prime epoche, la perdita sui dati di validazione aumenta mentre quella sui dati di addestramento diminuisce, indicando che il modello sta memorizzando i dati di addestramento piuttosto che generalizzare.
* Tuttavia, nelle ultime epoche, c'è un miglioramento sia nella perdita che nell'accuratezza sui dati di validazione, suggerendo che il modello potrebbe aver iniziato a generalizzare meglio.

Il modello mostra un miglioramento, ma ci sono segnali iniziali di overfitting che vengono parzialmente mitigati nelle epoche successive.

# Test

A seguito del training il modello è stato testato utilizzando il dataset di validazione, dimostrando di ritornare la classe corretta il 65% delle volte.

Di seguito alcuni dei risultati del test:  
  


Predicted: 1, Correct: 1

Immagine che contiene acquario, Organismo, Biologia marina, subacqueo

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

# PUNTI DI ATTENZIONE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | Descrizione | Owner | Decisioni | Stato |
|  |  |  |  |  |
|  |  |  |  |  |

# ALLEGATI

|  |  |  |
| --- | --- | --- |
| **ID** | Allegato | Descrizione |
|  |  |  |
|  |  |  |
|  |  |  |