Progetto Introduzione all'Intelligenza Artificiale

Studenti:

Brahas Eduard

Chionne Daniel

Docente

Poggioni Valentina

Anno accademico 2023/2024

Indice

- Prefazione
- Progetto Uniform coloring
- Descrizione del dominio e vincoli
- Classi utilizzate per la ricerca nello spazio, la classe problem e le sue componenti
- Sistema di Rilevamento e Classificazione di Caratteri Alfanumerici Utilizzando Reti Neurali Convoluzionali e OpenCV
- Ricerca
- Introduzione all'Applicazione Web
- Statistiche e Valutazione dei Risultati (da sistemare sulla base degli aggiornamenti sul modello)

Prefazione

Il progetto consiste nella realizzazione di una applicazione di Intelligenza Artificiale completa degli aspetti di gestione di: sensori per l'acquisizione dei dati dall'esterno relativi a stati e obiettivi, ragionamento/ricerca della soluzione per i goal acquisiti, esecutori per la realizzazione delle azioni che conducono alla soluzione.

Progetto Uniform coloring

Uniform Coloring è un dominio in cui si hanno a disposizione alcune celle da colorare, e vari colori a disposizione.

Per semplicità immaginiamo una griglia rettangolare in cui è possibile spostare una testina colorante fra le celle attigue secondo le 4 direzioni cardinali (N,S,E,W), senza uscire dalla griglia.

Tutte le celle hanno un colore di partenza (B=blu, Y=yellow, G=green) ad eccezione di quella in cui si trova la testina indicata con T. La testina può colorare la cella in cui si trova con uno qualsiasi dei colori disponibili a differenti costi (cost(B)=1, cost(Y)=2, cost(G)=3), mentre gli spostamenti hanno tutti costo uniforme pari a 1.

> L'obiettivo è colorare tutte le celle dello stesso colore (non importa quale) e riportare la testina nella sua posizione di partenza.

La codifica di tutto il dominio (topologia della griglia, definizione delle azioni etc.) è parte dell'esercizio. Partendo dalla posizione iniziale della testina e combinando azioni di

spostamento e colorazione, si chiede di trovare la sequenza di azioni dell'agente per raggiungere l'obiettivo.

La posizione iniziale della testina, la struttura della griglia e la colorazione iniziale delle celle sono passati al sistema tramite un'immagine.

Inizializzazione delle librerie e moduli

```
        #prog-intro.py
        import uniformcoloring as uc
        import lettermodel as lm
        import os
        import sys
        import time
        import datetime
        import cv2
        import numpy as np
        import signal
        import warnings
        from flask import Flask, Response, render_template

        #lettermodel.py
        import tensorflow as tf
        import numpy as np
        import matplotlib
        matplotlib.use('Agg')
        import matplotlib.pyplot as plt
        import cv2
        import os
        import datetime
        from emnist import list_datasets
        from tensorflow import keras
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import ConfusionMatrixDisplay
        from sklearn.metrics import recall_score, precision_score,
f1_score, accuracy_score
        from emnist import extract_training_samples
        from emnist import extract_test_samplesmanca classfic
        #manca classification report se va implementato

        #uniformcoloring.py
        from utils import *
        from search import *
        from enum import Enum
        import time
```

Il file python del progetto è suddiviso in 3 files:

- `prog-intro-ai.py` : utilizza OpenCV per integrare la manipolazione delle immagini, l'addestramento del modello e gli algoritmi di ricerca, offrendo un'interfaccia Web per l'interazione mediante l'utilizzo di Flask.
- `lettermodel.py` : utilizza TensorFlow per addestrare un modello di rete neurale convoluzionale (CNN) sull'insieme di dati EMNIST bilanciato, quindi utilizza il modello per classificare regioni di interesse (ROI) estratte dalle immagini tramite manipolazioni di immagini e contorni sfruttando la libreria OpenCV. Inoltre implementa Keras per la costruzione, l'addestramento e la valutazione del modello di deep learning.
- `uniformcolring.py` : definisce una classe di problemi di colorazione uniforme e implementa diversi algoritmi di ricerca non informata e informata (come A* e ricerca greedy) per risolvere il problema. Si utilizza per determinare la sequenza di azioni che porta al raggiungimento dell'obiettivo, dove ogni cella di una griglia deve essere colorata in modo che tutte le celle adiacenti abbiano colori uguali.

Descrizione del dominio e vincoli

Elementi del dominio:

- *Celle*: Presenti in una griglia rettangolare/quadrata nella quale è possibile spostare una testina colorante. Ogni cella ha un colore di partenza ed è possibile ricolorarle con i tre colori disponibili (yellow, blue, green).
- *Testina colorante*: È l'agente che, nella griglia fornita in input come immagine, può spostarsi tra le celle e cambiarne il colore. Una delle celle rappresenta la posizione iniziale della testina prima di muoversi e sulla quale dovrà ritornare dopo aver svolto le azioni richieste.

Relazioni:

- Ad ogni cella dell'immagine è associata un'etichetta rappresentante uno dei colori disponibili (Y, B, G).
- La testina inizialmente dovrà essere posizionata sempre sulla cella con etichetta T.

Regole:

- La testina può spostarsi nelle sole direzioni UP, DOWN, LEFT, RIGHT.
- La testina può cambiare colore nella cella in cui è posizionata. Colorare le celle ha un costo che varia in base al colore (cost(B) = 1, cost(Y) = 2, cost(G) = 3).
- Il passaggio da una cella all'altra ha sempre costo 1.

- **GOAL**: Colorare tutte le celle dello stesso colore. La testina dovrà trovare un modo per farlo nella maniera più efficiente possibile, sia in termini di colori che di numero di passi effettuati per muoversi tra le celle.

Vincoli:

- v0="La griglia può essere rettangolare e quadrata".
- v1="L'agente può compiere un solo passo alla volta".
- v2="L'agente si può muovere solo fra celle adiacenti".
- v3="Nella griglia non esistono celle vuote, tutte devono essere colorate in partenza".
- v4="Le celle devono essere tutte raggiungibili dalla posizione iniziale della testina".
- v5="Il costo delle azioni di movimento è uniforme (1), mentre il costo della colorazione dipende dal colore scelto".
- v6="L'agente non può colorare la posizione di partenza (quindi bisogna trovare un modo per evitarlo)".
- v7="Dopo che l'agente ha colorato tutte le celle, deve ritornare alla posizione di partenza".
- v8="Senza la posizione di partenza la griglia non è utilizzabile"

Esempi di problemi, con possibili soluzioni e costi:

Stato iniziale

G	T	G	B
G	Y	G	B

Possibile stato goal

G	T	G	G
G	G	G	G

Possibile soluzione

Sud, col-G, East, East, col-G, Nord, col-G, West, West

Costo = 15 Lunghezza = 9

Stato iniziale

G	G
Y	Y
Y	T

Possibile stato goal

Y	Y
Y	Y
Y	T

Possibile soluzione

Nord, Nord, col-Y, West, col-Y, East, Sud, Sud

Costo = 10 Lunghezza = 8

Classi utilizzate per la ricerca nello spazio, la classe problem e le sue componenti

Il codice Python presenta una serie di classi e funzioni per risolvere problemi di colorazione uniforme su una griglia. Le principali componenti includono:

1. Definizione di costanti come `MOV_COST`, che rappresenta il costo di un movimento.

2. Definizione di enumerazioni come `Colors` e `Directions`, che rappresentano rispettivamente i colori disponibili e le direzioni in cui ci si può spostare sulla griglia.
3. Definizione di una classe `State`, che rappresenta lo stato del problema con la posizione corrente sulla griglia e l'identificatore unico.
4. Definizione di una classe `UniformColoring` che eredita da una classe astratta `Problem` e implementa i metodi per rappresentare il problema, come le azioni possibili in uno stato, il risultato di un'azione, il test dell'obiettivo e il calcolo del costo del percorso.
5. Implementazione di algoritmi di ricerca come A* e ricerca greedy per risolvere il problema di colorazione uniforme sulla griglia.
6. Altre funzioni ausiliarie come `initialize_state` per inizializzare lo stato iniziale del problema.

Di seguito si spiegano nel dettagli le parti fondamentali

Classe `UniformColoring`:

Metodo `__init__`: Inizializza il problema con lo stato iniziale e il tipo di euristica da utilizzare.

Metodo `actions`: Restituisce le azioni possibili in uno stato, che possono essere il cambio di colore di una casella o il movimento in una direzione.

Metodo `result`: Restituisce lo stato successivo dato uno stato e un'azione.

Metodo `goal_test`: Verifica se uno stato è uno stato di obiettivo, cioè se tutte le caselle hanno lo stesso colore.

Metodo `path_cost`: Calcola il costo del percorso per raggiungere uno stato successivo dato uno stato e un'azione.

Metodo `color_initial_choice`: Calcola il colore iniziale che verrà utilizzato per la colorazione.

Metodo `heuristic`: Calcola il valore euristico di uno stato dato il tipo di euristica selezionato.

Metodo `h`: Calcola il valore euristico di uno stato.

Classi degli algoritmi di ricerca:

`best_first_graph_search`: Implementa l'algoritmo di ricerca best-first-graph-search, che da la precedenza ai nodi con i valori di `f` più bassi.

`iterative_deepening_search` : Implementa l'algoritmo di ricerca iterative deepening search, che esegue una ricerca a profondità limitata aumentando progressivamente la profondità massima. Sfrutta l'algoritmo `depth_limit_search` con un limite incrementato fino a quando non si trova la soluzione. La funzione `is_cycle` verifica se esiste un ciclo nel percorso fino a un nodo. Per avere un funzionamento abbastanza veloce è stato imposto un limite alla profondità massima di 25.

`astar_search` : Implementa l'algoritmo di ricerca A*, che combina il costo effettivo del percorso finora con un'euristica per stimare il costo rimanente.

`greedy_search` : Implementa l'algoritmo di ricerca greedy, che seleziona il successore più promettente secondo l'euristica.

Euristica utilizzata:

Le euristiche utilizzate sono le seguenti:
`euristica_color_use_most_present` : Calcola il valore euristico sommando la distanza di Manhattan tra ogni casella non colorata e la posizione attuale, con una penalità aggiuntiva se il colore della casella non è il colore più presente nella griglia.
Questa euristica cerca di minimizzare la distanza tra le caselle non colorate e la posizione attuale, preferendo anche i colori più presenti nella griglia per ridurre il numero di caselle non colorate.

`Nearest-kneighbor-distance` : questa funzione euristica viene utilizzata per stimare il costo di raggiungere un obiettivo specifico partendo da un nodo corrente. Utilizza la distanza di Manhattan tra le celle non ancora colorate e il nodo corrente, insieme ad altre informazioni come il costo del movimento, per determinare il valore euristico. Questo valore viene quindi utilizzato dall'algoritmo di ricerca del percorso per guidare la selezione del prossimo nodo da esplorare, con l'obiettivo di ottimizzare il percorso verso l'obiettivo finale.

Sistema di Rilevamento e Classificazione di Caratteri Alfanumerici Utilizzando Reti Neurali Convoluzionali e OpenCV

Addestramento del Modello:

Modello CNN: Viene definito un modello di rete neurale *convoluzionale* (CNN) per la classificazione delle immagini. Un modello convoluzionale è una rete neurale *progettata per l'elaborazione delle immagini, utilizzando strati convoluzionali per rilevare pattern e caratteristiche*. Il modello è composto da diversi strati, tra cui strati di convoluzione, strati di max pooling, strati di dropout e strati densamente connessi. Gli strati di convoluzione applicano filtri per estrarre caratteristiche dalle immagini. I max pooling riducono la

dimensionalità preservando le caratteristiche principali. Gli strati di dropout combattono l'overfitting disattivando casualmente alcuni neuroni. Gli strati densamente connessi combinano le caratteristiche estratte per la classificazione finale.

```
        seq_lett_model = keras.Sequential([
        keras.Input(shape=(28, 28, 1)), # Input shape: 28x28 pixels, 1
color channel
        keras.layers.Conv2D(28, (3, 3), activation='relu'), #(number of
layers, dimension of kernel, activation funztion)
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Conv2D(128, (3, 3), activation='relu'),
        keras.layers.Dropout(0.5), #used for preventing overfitting
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Conv2D(128, (3, 3), activation='relu'),
        keras.layers.Flatten(), #convert a multidimensional input into a
one dimensional vector
        keras.layers.Dense(512, activation='relu'), #al neurons of this
layer are connected with al neurons of the previus layer
        keras.layers.Dropout(0.5),
        keras.layers.Dense(4, activation='softmax')  # Output layer for 4
letters
        ])
```

Preparazione del dataset: Prima dell'addestramento è necessario filtrare il dataset ed estrarre solo gli esempi utili al nostro progetto. Questo viene fatto definendo tutte le etichette definite dal dataset EMNIST (`LABELS`), applicare la funzione `filterDataset(X_data, y_data)` che filtra il dataset in base a etichette specifiche ('T', 'B', 'G', 'Y') restituendo un nuovo insieme di dati e relative etichette e infine la funzione `remodulate(y)` , che sostituisce le etichette 'T', 'B', 'Y', 'G' con i valori numerici 0, 1, 2, 3 rispettivamente.

Addestramento del Modello: Dopo aver preparato il dataset, è importante normalizzare sia il set di addestramento che quello di test per standardizzare i valori dei pixel delle immagini nell'intervallo (0,1), assicurando che il modello CNN impari senza distorsioni dovute a scale pixel diverse. Successivamente, mediante reshape, le immagini vengono ridimensionate per adattarle al formato richiesto dal modello. Si definisce quindi il batch size, che rappresenta il numero di campioni passati attraverso la rete contemporaneamente durante l'addestramento. Le epoche, invece, indicano quante volte l'intero dataset viene presentato al modello per l'addestramento.
Il modello CNN viene compilato utilizzando la funzione di perdita `sparse_categorical_crossentropy` (calcola la perdita tra le etichette vere e le previsioni del modello) e l'ottimizzatore `adam` , quindi addestrato utilizzando i dati di addestramento, riservandone una frazione come dati di convalida.
Viene anche utilizzato il callback `EarlyStopping` per interrompere l'addestramento se la perdita sui dati di convalida smette di diminuire.

Valutazione delle Prestazioni del Modello:

Confusion Matrix, grafici e Metriche di Valutazione: Viene calcolata una matrice di confusione per valutare le prestazioni del modello sulla classificazione dei caratteri alfanumerici. Vengono inoltre calcolate e visualizzate metriche per valutare le prestazioni del modello su ciascuna classe di caratteri alfanumerici. Di seguito se ne riportano i vari utilizzati:

- *Precisione (Precision)*: Misura la proporzione di istanze positive correttamente predette tra tutte le istanze predette come positive.
  Formula: Precisione = TP / (TP + FP), dove TP sono i veri positivi e FP sono i falsi positivi.
- *Recall (Recall)*: Misura la proporzione di istanze positive correttamente predette tra tutte le istanze positive effettive.
  Formula: Richiamo = TP / (TP + FN), dove TP sono i veri positivi e FN sono i falsi negativi.
- *F1-Score*: È la media armonica di precisione e richiamo e fornisce un'unica misura del modello.
  Formula: F1-Score = 2 x (Precisione x Richiamo) / (Precisione + Richiamo)
- *Accuratezza (Accuracy)*: Misura la frazione di predizioni corrette rispetto al totale delle predizioni.
  Formula: Accuratezza = (TP + TN) / (TP + TN + FP + FN), dove TP sono i veri positivi, TN sono i veri negativi, FP sono i falsi positivi e FN sono i falsi negativi.
- *Perdita (Loss)*: La perdita è una misura dell'errore tra le predizioni del modello e le etichette vere. L'obiettivo durante l'addestramento è minimizzare questa perdita.

Accuracy in tests: 0.984375

Loss in tests: 0.04550512135028839

Reports

Metrics class T: {'precision': 0.99, 'recall': 0.98, 'f1-score': 0.98, 'accuracy': 0.98}

Metrics class B: {'precision': 0.99, 'recall': 0.98, 'f1-score': 0.98, 'accuracy': 0.98}

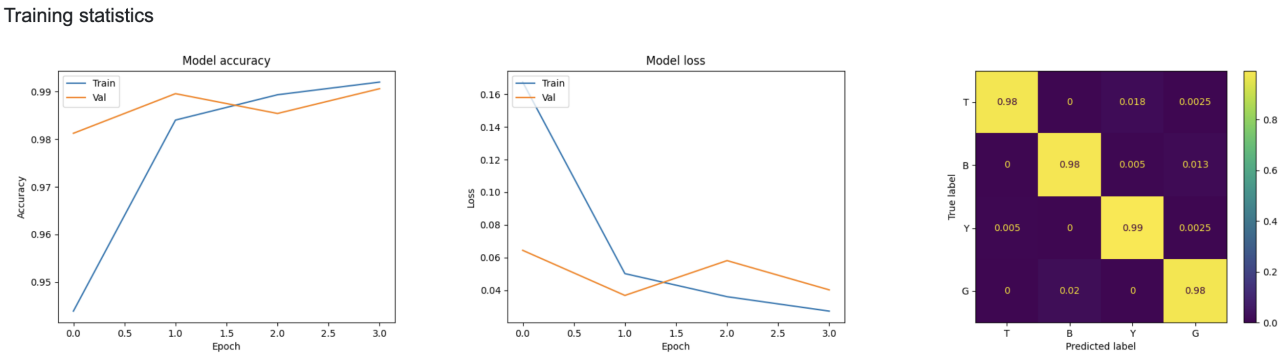
Metrics class Y: {'precision': 0.97, 'recall': 0.99, 'f1-score': 0.98, 'accuracy': 0.98}

Metrics class G: {'precision': 0.99, 'recall': 0.99, 'f1-score': 0.99, 'accuracy': 0.98}

		PREDICTED CLASS	
ACTUAL CLASS		Yes	No
	Yes	TP	FN
	No	FP	TN

Valutazione del Modello: Dopo l'addestramento, il modello viene valutato utilizzando i dati di test. Viene calcolata l'accuratezza del modello e vengono generate visualizzazioni della perdita e dell'accuratezza durante l'addestramento, oltre che la matrice di confusione

Esempio di grafici mostrati per la valutazione



Tutti i vari grafici vengono memorizzati nella cartella plots del progetto

Acquisizione e Processing delle Immagini:

Acquisizione dell'Immagine dalla Webcam: Viene definita una funzione per acquisire un frame dall'input della webcam. Il frame viene quindi restituito in formato PNG. Si può anche fare l'upload di immagini già fatte.

Preprocessing dell'Immagine per il Rilevamento delle Griglie: L'immagine catturata dalla webcam viene preelaborata per il rilevamento delle griglie. Questo include la conversione in scala di grigi, l'applicazione di una sfocatura gaussiana per ridurre il rumore e l'applicazione di una soglia adattiva per creare un'immagine binaria.
La griglia viene memorizzata nella cartella grids del progetto

Estrazione dei ROIs (Region Of Interests): Vengono estratte le regioni di interesse (ROIs) dall'immagine preelaborata utilizzando i contorni delle aree significative. Le ROIs corrispondono alle singole celle della griglia contenenti caratteri alfanumerici.
L'immagine rielaborata e i singoli ROIs (celle della griglia) vengono memorizzate nella cartella manipulated_grids del progetto fino a riesecuzione della predizione

Preprocessing delle ROIs per la Predizione: Le ROIs vengono preelaborate ulteriormente, ridimensionate e normalizzate, per poi essere utilizzate per la predizione dei caratteri alfanumerici.

Predizione dei Caratteri alfanumerici: Le ROIs preelaborate vengono utilizzate come input per il modello CNN addestrato precedentemente. Il modello restituisce le predizioni dei caratteri alfanumerici per ciascuna ROI e genera una matrice delle stesse dimensioni della griglia. La funzione è la seguente

```
def prediction(ROIs, n, seq_lett_model):
    # Preprocess the ROIs and make predictions
    l = []
    for i in range(1, len(ROIs)):
        im = preprocess_image(f"./manipulated_grids/ROI_{i}.png")
        if im is not None:
            prediction = seq_lett_model.predict(im)
```

```
            max = np.where(prediction == np.amax(prediction))
            l.append(int(max[1][0]))

    # Check if T is present in the predictions
    if 0 not in l:
        print("T not found in the grid")
        return None  # Return none as an error

    # Create the grid from the predictions
    nrow = len(l) // n if n < len(l) else n // len(l)
    nrow = int(nrow)

    mat = np.array(list(reversed(l)))
    if nrow == 1:
        try:
            grid = mat.reshape(nrow, n-1)
        except:
            print("Reshape error")
            return None
    else:
        try:
            grid = mat.reshape(nrow, n)
        except:
            print("Reshape error")
            return None

    return grid
```

Ricerca

Dopo la predizione delle lettere della griglia, si passa alla fase effettiva di ricerca. Il programma prevede l'esecuzione di tutti gli algoritmi citati precedentemente, mostrando per ogni algoritmo lo stato finale, definito da una griglia che mostra come la testina ha colorato la griglia, il costo per raggiungere il goal, il tempo di esecuzione e la sequenza di azioni effettuate. Questo permette di effettuare una valutazione sull'efficacia degli algoritmi. All'inizio è riportata la griglia predetta dal modello mediante la funzione prediction

Qui sotto si riporta nello specifico il processo di ricerca informata e non informata

Traduzione dei Dati: Le immagini analizzate vengono tradotte in stati iniziali (`stato_iniziale`) e stati obiettivo (`stato_goal`) secondo la rappresentazione definita per il problema di colorazione uniforme. Questa rappresentazione coinvolge la creazione di un oggetto `State` che tiene traccia della griglia di colori e della posizione corrente del cursore nella griglia.

Invocazione del Solutore: Viene invocato il solutore del problema, rappresentato dalla classe `UniformColoring` , utilizzando una tecnica di ricerca informata e non informata. Le

tecniche di ricerca informata includono A* e Greedy Search, mentre la ricerca non informata è rappresentata dall'Iterative Deepening Search e UCS. Questi algoritmi cercano di trovare una sequenza di azioni ottimali per raggiungere lo stato obiettivo a partire dallo stato iniziale.

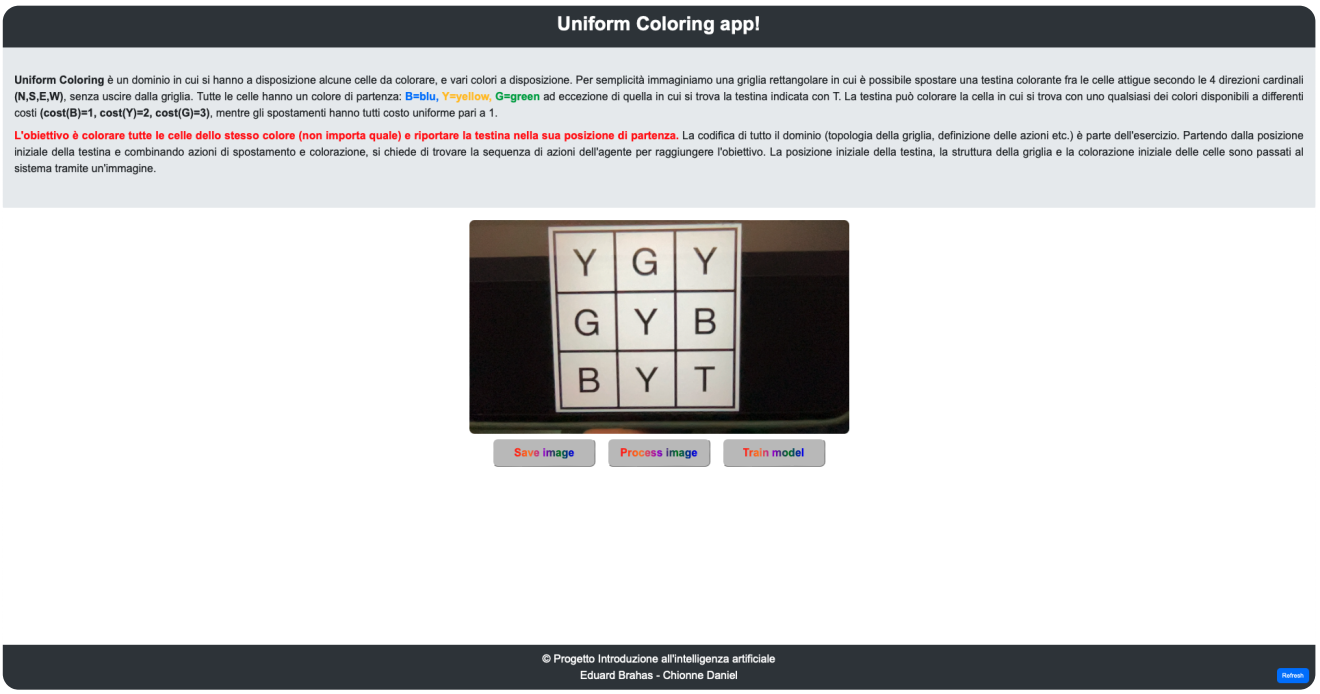
Produzione della Soluzione: Se esiste una soluzione, cioè una sequenza di azioni che porta dallo stato iniziale allo stato obiettivo, viene restituita dal solutore. La sequenza di azioni rappresenta le mosse da effettuare sulla griglia per raggiungere lo stato obiettivo.

Esempio di risultato mostrato

Introduzione all'Applicazione Web

L'applicazione web sviluppata utilizza Flask come framework per implementare le funzionalità di backend e fornisce un'interfaccia utente intuitiva per mostrare il problema della colorazione uniforme.

Spiegazione del Funzionamento
Il paragrafo iniziale dell'applicazione fornisce una descrizione dettagliata del funzionamento del problema della colorazione uniforme. Viene chiarito l'obiettivo del programma e come esso sia in grado di risolvere il problema attraverso l'utilizzo di algoritmi di ricerca.



Acquisizione e Salvataggio dell'Immagine

Nella sezione dedicata all'acquisizione dell'immagine, l'utente può utilizzare la webcam del proprio computer per scattare una foto alla griglia che desidera processare. Una volta catturata l'immagine, è possibile salvarla premendo il pulsante "Save image". L'immagine salvata viene memorizzata su `/grids`, una directory utilizzata per conservare lo stato iniziale del problema. Inoltre è possibile effettuare un upload di immagini da parte dell'utente.

Elaborazione dell'Immagine

Dopo aver salvato l'immagine, l'utente può procedere con la sua elaborazione premendo il pulsante "Process image". Durante questo processo, l'applicazione utilizza gli algoritmi implementati per risolvere il problema di colorazione uniforme. I risultati ottenuti vengono visualizzati successivamente.

Addestramento del Modello

Il pulsante "Train model" avvia la fase di addestramento del modello utilizzato per risolvere il problema. Durante il training, vengono mostrati grafici e metriche per valutare le prestazioni del modello.

Feedback durante l'Esecuzione

Durante l'elaborazione dell'immagine e l'addestramento del modello, l'applicazione fornisce un feedback visivo sotto forma di una barra di avanzamento. Questo permette all'utente di comprendere che il programma è in esecuzione.

Alcuni degli output mostrati sugli algoritmi di ricerca

Prediction

T	Y	B
B	Y	G
Y	G	Y

UCS Results

0	1	1
1	1	1
1	1	1

Solution cost: 14

Time of execution: 3.0400571823120117

Solution: [<Directions.RIGHT: (0, 1)>, <Colors.BLUE: 1>, <Directions.DOWN: (1, 0)>, <Colors.BLUE: 1>, <Directions.RIGHT: (0, 1)>, <Colors.BLUE: 1>, <Directions.DOWN: (1, 0)>, <Colors.BLUE: 1>, <Directions.LEFT: (0, -1)>, <Colors.BLUE: 1>, <Directions.LEFT: (0, -1)>, <Colors.BLUE: 1>, <Directions.UP: (-1, 0)>, <Directions.UP: (-1, 0)>]

IDS Results

0	2	2
2	2	2
2	2	2

Solution cost: 18

Time of execution: 0.2784888744354248

Solution: [<Directions.RIGHT: (0, 1)>, <Directions.RIGHT: (0, 1)>, <Directions.DOWN: (1, 0)>, <Colors.YELLOW: 2>, <Directions.UP: (-1, 0)>, <Colors.YELLOW: 2>, <Directions.DOWN: (1, 0)>, <Directions.DOWN: (1, 0)>, <Directions.LEFT: (0, -1)>, <Colors.YELLOW: 2>, <Directions.LEFT: (0, -1)>, <Directions.UP: (-1, 0)>, <Colors.YELLOW: 2>, <Directions.UP: (-1, 0)>]

A* Results

0	2	2
2	2	2
2	2	2

Solution cost: 16

Time of execution: 0.003561973571773438

Solution: [<Directions.DOWN: (1, 0)>, <Colors.YELLOW: 2>, <Directions.RIGHT: (0, 1)>, <Directions.DOWN: (1, 0)>, <Colors.YELLOW: 2>, <Directions.UP: (-1, 0)>, <Directions.RIGHT: (0, 1)>, <Colors.YELLOW: 2>, <Directions.UP: (-1, 0)>, <Colors.YELLOW: 2>, <Directions.LEFT: (0, -1)>, <Directions.LEFT: (0, -1)>]

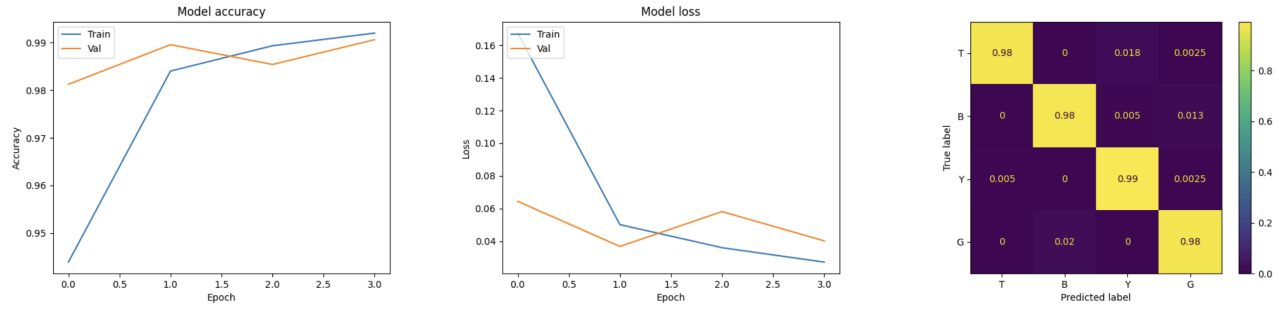
Greedy Best First Search Results

0	2	2
2	2	2
2	2	2

Solution cost: 18

Output mostrato dalla fase di training

Training statistics



Statistiche e Valutazione dei Risultati

Accuratezza media e perdita in test

Nella valutazione dei risultati relativi alla classificazione delle lettere e cifre e all'estrazione degli stati dalle immagini, è stato osservato un'elevata accuratezza nel rilevamento e nell'identificazione delle entità presenti nelle griglie. La classificazione delle lettere e cifre è stata eseguita con un'accuratezza media del 98.7%, consentendo un'efficace estrazione degli stati necessari per risolvere il problema della colorazione uniforme.
La funzione di perdita si riduce in un 3%.

Report sulle metriche di precision, recall, f-measure e accuracy

Per la classe T, la qualità della classificazione può essere considerata buona, poiché la precisione, il recall e l'F1-score sono tutti intorno al 98-99%. Questi valori indicano che il 98-99% delle lettere o cifre classificate come classe T è corretto, mentre il 98-99% delle lettere o cifre appartenenti effettivamente alla classe T è stato identificato correttamente.

Anche per la classe B, la qualità della classificazione può essere considerata buona. Precisione, recall e F1-score sono tutti intorno al 98%, il che significa che circa il 98% delle lettere o cifre classificate come classe B è corretto e il 98% delle lettere o cifre appartenenti effettivamente alla classe B è stato identificato correttamente.

Per la classe Y, la qualità della classificazione può ancora essere considerata buona, sebbene leggermente inferiore rispetto alle classi T e B. La precisione è del 99%, il che indica che il 99% delle lettere o cifre classificate come classe Y è corretto. Tuttavia, il recall è del 97%, il che suggerisce che il 97% delle lettere o cifre appartenenti effettivamente alla classe Y è stato identificato correttamente.

Anche per la classe G, la qualità della classificazione può essere considerata buona. Precisione, recall e F1-score sono tutti intorno al 98%, indicando che circa il 98% delle lettere o cifre classificate come classe G è corretto e il 98% delle lettere o cifre appartenenti effettivamente alla classe G è stato identificato correttamente.

Fase di ricerca
Per quanto riguarda le prestazioni della ricerca nello spazio degli stati, gli algoritmi implementati hanno dimostrato una buona capacità di risolvere i problemi proposti. In particolare, sono stati risolti con successo la maggior parte dei problemi di colorazione uniforme di dimensioni medie e piccole. Tuttavia, per problemi di dimensioni molto grandi, si è riscontrata una maggiore complessità computazionale e alcuni casi in cui la ricerca non è stata in grado di trovare una soluzione ottimale entro i limiti di tempo prestabiliti (per risolvere il problema è stato imposto un limite di tempo sull'esecuzione delle funzioni), ad esempio nel caso di UCS e IDS.

In generale, l'analisi delle prestazioni degli algoritmi di ricerca ha evidenziato una correlazione tra la dimensione del problema e la complessità computazionale richiesta per la sua risoluzione. Problematiche di dimensioni ridotte sono state risolte in tempi brevi con una buona precisione, mentre problemi di dimensioni maggiori hanno richiesto più tempo e risorse computazionali. Tuttavia, nonostante le sfide incontrate nei problemi di dimensioni più grandi, gli algoritmi hanno dimostrato una buona capacità di approssimazione delle soluzioni anche in questi contesti, consentendo una gestione efficace dei problemi di colorazione uniforme.

Per quanto riguarda il costo degli algoritmi utilizzati, sono gli stessi dei costi definiti a lezioni per gli algoritmi utilizzati. Ovviamente il costo in spazio e in tempo variano in base alla dimensione del problema che deve essere risolto, cioè in base alle dimensioni della griglia data in input.

Costo ricerca non informata:
Ricordiamo che la UCS sia in tempo e in spazio ha un costo di $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$.
Per IDS in tempo abbiamo $O(b^d)$ ed in spazio$O(b^d)$ $O(bd)$ dove d rappresenta la soluzione più profonda. C'è da fare una piccola sottolineatura l'ottimalità non è garantita dato che i costi delle azioni non sono tutti uguali.

Si è stato notato che la UCS è l'algoritmo che in tempo ha prestazioni migliori rispetto all'IDS. Il costo delle azioni può essere migliore di uno o l'altro in base al tipo di griglia fornita

Costo ricerca informata:
Il tempo di esecuzione si è dimostrato minore nella ricerca greedy rispetto ad A* anche se il costo della soluzione è quasi sempre maggiore o uguale ad A*.

Problema: La principale sfida riscontrata nel nostro sistema riguarda la corretta lettura delle griglie, soprattutto quelle create manualmente, da parte delle funzioni di processing delle immagini. Questo problema è principalmente causato dalla forma delle linee e dalla spaziatura tra le lettere e i contorni di ogni cella della griglia. C'è da fare una considerazione riguardo la webcam del computer come qualsiasi altro strumento di cattura immagine di livello non professionale aggiunge un certo livello di distorsione delle immagini e quindi delle linee della tabella non dritte. Per non parlare dell'ambiente in cui viene scattata l'immagine che se non ci sono delle situazioni ideali si posso creare dei artefatti amplificati dalle forti manipolazioni per aumentare la visibilità per il riconoscimento dell'immagine.

Impatto: La corretta lettura delle griglie è fondamentale per il corretto funzionamento del nostro sistema, poiché influisce direttamente sulla predizione accurata dei modelli. Quando le griglie non vengono lette correttamente, si verificano errori nella comprensione e nell'analisi dei dati, compromettendo l'affidabilità e l'efficacia delle nostre previsioni e dei nostri risultati.

Soluzione: Per mitigare questo problema, è consigliabile utilizzare immagini digitali con bordi precisi e ben definiti, e assicurarsi che le lettere siano centrate all'interno di ciascuna cella della griglia. Inoltre, c'è stato il tentativo di implementare algoritmi di pre-processing delle immagini per migliorare la qualità delle griglie manuali, come ad esempio la rimozione del rumore, la correzione della distorsione e l'ottimizzazione dei contrasti.
Per garantire buone performance di risoluzione abbiamo creato delle griglie preimpostate in un documento word la quale possiamo aggiungere o rimuovere lettere che devono avere una dimensione tale da renderle grandi e ben definite nelle celle.

Considerazioni finali
Per realizzare questo progetto sono state implementate tre tecnologie: il modello di riconoscimento delle lettere, la definizione del problema e gli algoritmi di risoluzione di esso e per finire la creazione di un interfaccia grafica per unire il tutto.
Il modello ha delle ottime performance ma non è perfetto, tende a scambiare la lettera G con la B e viceversa. Come detto anche sopra gli errori la maggior parte delle volte sono indotti da come viene letta l'immagine da openCV dato che comunque in casi di immagini non chiare e di griglie molto grandi tende a creare aree di interesse in modo non coerente con la griglia data.
La definizione del problema e gli algoritmi sono anche loro funzionanti nel modo corretto anche se abbiamo dovuto introdurre delle misure di "sicurezza". La limitazione della frontiera ad un numero discreto per evitare attese possibilmente molto lunghe, anche per l'UCS si è fatta praticamente la stessa cosa. Le euristiche implementate sono molto buone e

garantiscono delle performance per la ricerca informata eccelse.
L'interfaccia grafica permette di avere una visualizzazione più intuitiva e accessibile da qualsiasi browser.