

Rilevamento piante infestanti

Nicol Alesi, Mario Corrente

February 2025

1 Abstract

Il riconoscimento delle piante infestanti è una sfida cruciale in agricoltura, con implicazioni dirette sulla produttività e sull'uso sostenibile delle risorse. Questo lavoro presenta lo sviluppo di un modello di intelligenza artificiale per distinguere automaticamente tra colture e piante infestanti. A tal fine, sono stati combinati più dataset per creare un set di dati bilanciato, utilizzando tecniche di data augmentation e generazione di immagini sintetiche tramite DCGAN. Sono stati sperimentati diversi algoritmi di classificazione, tra cui Convolutional Neural Networks (CNN), Support Vector Machine (SVM), K-Nearest Neighbors (KNN) e Random Forest (RF), con un confronto delle loro prestazioni. Nonostante l'accuratezza raggiunta, il modello CNN ha mostrato problemi di overfitting, che sono stati affrontati con strategie di regolarizzazione e cross-validation. Il lavoro evidenzia i vantaggi e le limitazioni delle tecniche adottate, ponendo le basi per futuri miglioramenti attraverso ottimizzazioni hardware e modelli più efficienti.

2 Introduzione

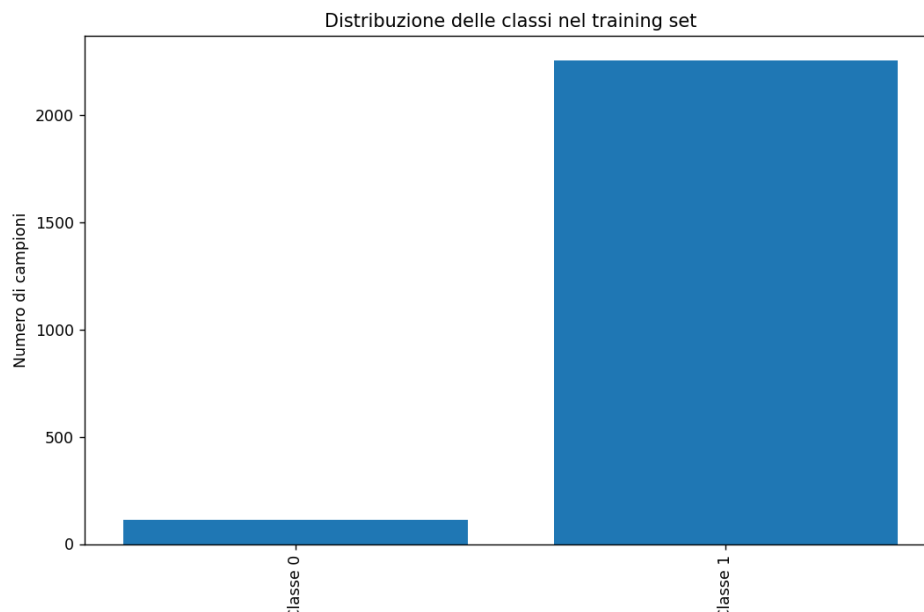
Il fine che ci si propone, è quello di allenare un modello che sia in grado di riconoscere una pianta infestante da una da coltura. L'obiettivo finale, con più lavoro e più tempo, sarebbe quello di permettere agli agricoltori di inquadrare una qualsiasi coltura per ricevere l'aiuto del sistema che suggerirà loro se sono presenti delle piante infestanti, e magari, permettere loro di sapere come trattare l'infestazione, in un possibile upgrade dell'app 2.0. Nel progetto proposto questo non è ancora possibile, ci si concentra sul reperire il dataset,

verificarne il bilanciamento ed utilizzarlo per addestrare modelli in grado di riconoscere piante infestanti o meno in casi semplici.

3 Metodologia

3.1 Dataset utilizzato

Il dataset di base è stato reperito su Kaggle: [Grass-Weed-Detection](#). È un dataset piuttosto limitato (2822 immagini), per questa ragione si è deciso di unirlo ad altri dataset contenenti [sia piante, sia colture](#) che solamente [colture](#). Non è stato aggiunto un dataset contenente solamente piante infestanti perché quello originale risultava essere piuttosto sbilanciato circa le categorie contenute. È possibile osservarlo nella figura sottostante, in cui “Classe 0” rappresenta la classificazione “crop” ossia, raccolto, e la “Classe 1” che rappresenta “weed”, ossia, pianta infestante.



Il dataset base, in formato YOLO, che ha una struttura particolare e delle annotazioni in .txt che descrivono rispettivamente :

- *Object class*
- *X center*
- *Y center*

- *Width*
- *Height*

In questo modo si riesce a etichettare l'immagine.

Per questa ragione, i dataset aggiunti manualmente, che non erano strutturati in questo formato, sono stati convertiti ed è stato segnalato, tramite la notazione .txt, di prendere come riferimento l'intera foto anziché una sola parte.

Oltre all'aggiunta manuale di altri dati, in ogni addestramento è stato effettuato un bilanciamento, quindi un'aggiunta artificiale di dati, che poi si osserverà nel dettaglio sulle spiegazioni dei vari script. Inoltre sono state effettuate delle operazioni di data augmentation, in particolare sulla CNN.

3.2 Algoritmi e tecniche di IA utilizzate

Si è deciso di utilizzare più metodologie di addestramento in modo da produrre più metriche per una scelta oculata della migliore. Sono state utilizzate le seguenti tecniche:

- **Convolutional Neural Network**
- **Support Vector Machine**
- **K Nearest Neighbour**
- **Random Forest**

Le reti convoluzionali sono le uniche ad essere state addestrate con le immagini così come sono. Le altre invece non accettano l'immagine come dato grezzo, (SVM lavora con dati tabulari) per questa ragione sono stati estratti gli HOG (Histogram of Oriented Gradients), ossia le caratteristiche salienti delle immagini, per poi passarle ai modelli come vettore. Al contrario, le reti convoluzionali, attraverso i filtri convoluzionali, apprendono automaticamente dall'immagine i pattern rilevanti. Perciò, **SVM**, **KNN**, **RFM**, hanno in comune, oltre al caricamento del dataset, l'operazione di estrazione degli HOG e anche l'utilizzo della libreria *SMOTE*, per creare dei campioni di dati "sintetici", che permette di equilibrare il dataset.

3.2.1 Support Vector Machine

L'algoritmo Support Vector Machine (SVM) è stato utilizzato per la classificazione delle immagini nel dataset. SVM è un modello supervisionato che trova l'iperpiano ottimale per separare le classi nel dataset.

Nel progetto, è stata utilizzata una SVM con kernel RBF (Radial Basis Function), che consente di gestire dati non linearmente separabili. Il modello è stato addestrato dopo l'estrazione delle feature HOG (Histogram of Oriented Gradients) e l'applicazione della tecnica SMOTE per bilanciare le classi. Le prestazioni del modello sono state valutate sul validation e test set utilizzando accuracy score e classification report. Il modello finale è stato salvato con Joblib per usi futuri. È stato scelto il kernel RBF anche sulla base delle metriche raccolte utilizzandone altri.

```
Accuracy: 0.7264150943396226
```

Classification Report:					
		precision	recall	f1-score	support
	0	0.70	0.82	0.76	219
	1	0.76	0.63	0.69	205
	accuracy			0.73	424
	macro avg	0.73	0.72	0.72	424
	weighted avg	0.73	0.73	0.72	424

```
svm_model = SVC(kernel='linear', C=10, gamma=0.01, probability=True)
```

Figura 1: Metriche modello SVM con kernel lineare

```
Accuracy: 0.7806603773584906
```

Classification Report:						
		precision	recall	f1-score	support	
	0	0.74	0.89	0.81	219	
	1	0.85	0.66	0.75	205	
	accuracy			0.78	424	
	macro avg	0.79	0.78	0.78	424	
	weighted avg	0.79	0.78	0.78	424	

```
svm_model = SVC(kernel='poly', C=10, degree=3, probability=True)
```

Figura 2: Metriche modello SVM con kernel polinomiale

Accuracy: 0.8066037735849056						
Classification Report:						
			precision	recall	f1-score	support
		0	0.78	0.88	0.82	219
		1	0.85	0.73	0.78	205
		accuracy			0.81	424
		macro avg	0.81	0.80	0.80	424
		weighted avg	0.81	0.81	0.81	424
svm_model = SVC(kernel='rbf', C=10, gamma=0.01, probability=True)						

Figura 3: Metriche modello SVM con kernel radiale

3.2.2 K Nearest Neighbour

L'algoritmo K-Nearest Neighbors (KNN) è stato impiegato per la classificazione delle immagini. KNN è un modello basato sulla somiglianza, in cui la classe di un'istanza viene determinata dai K vicini più prossimi nel dataset di addestramento.

Nel progetto, KNN è stato addestrato dopo l'estrazione delle feature HOG (Histogram of Oriented Gradients) e la gestione dello sbilanciamento dei dati tramite SMOTE. È stato utilizzato un modello con 100 vicini (si nota però, che da k=50 in poi non ci sono miglioramenti), pesati in base alla distanza euclidea:

Le prestazioni sono state valutate su validation e test set utilizzando accuracy score e classification report. Il modello finale è stato salvato con Joblib per future applicazioni. In questo caso si nota che a parità di k, la scelta tra la distanza euclidea o Manhattan non cambia le metriche.

Accuracy: 0.8813559322033898						
Classification Report:						
			precision	recall	f1-score	support
		0	0.46	1.00	0.63	12
		1	1.00	0.87	0.93	106
		accuracy			0.88	118
		macro avg	0.73	0.93	0.78	118
		weighted avg	0.95	0.88	0.90	118
knn_model = KNeighborsClassifier(n_neighbors=100, weights='distance', metric='manhattan')						

Figura 4: Metriche modello KNN con distanza euclidea

```

Accuracy: 0.8813559322033898

Classification Report:

```

		precision	recall	f1-score	support
	0	0.46	1.00	0.63	12
	1	1.00	0.87	0.93	106
	accuracy			0.88	118
	macro avg	0.73	0.93	0.78	118
	weighted avg	0.95	0.88	0.90	118

```

knn_model = KNeighborsClassifier(n_neighbors=100, weights='distance', metric='manhattan')

```

Figura 5: Metriche modello KNN con distanza Manhattan

3.2.3 Random Forest

Il modello di classificazione utilizzato è la Random Forest, un algoritmo di apprendimento supervisionato che combina più alberi decisionali per migliorare le previsioni e ridurre il rischio di overfitting. Il modello è configurato con 200 alberi, una profondità massima di 20 e pesi bilanciati per le classi. Nonostante l'uso di SMOTE per bilanciare il dataset, il modello tendeva a predire principalmente una sola classe (addestrato sul dataset di base), probabilmente a causa della poca eterogeneità dei dati.

```

Accuracy: 0.8983050847457628

Classification Report:

```

		precision	recall	f1-score	support
	0	0.00	0.00	0.00	12
	1	0.90	1.00	0.95	106
	accuracy			0.90	118
	macro avg	0.45	0.50	0.47	118
	weighted avg	0.81	0.90	0.85	118

```

rf_model = RandomForestClassifier(
    n_estimators=150,
    max_depth=15,
    random_state=42,
    class_weight="balanced"
)
rf_model.fit(x_train_resampled, y_train_resampled)

```

Figura 6: Metriche RF prima dell'unione con dataset eterogenei

```

Accuracy: 0.5377358490566038

Classification Report:

```

		precision	recall	f1-score	support
	0	0.53	0.94	0.68	219
	1	0.63	0.11	0.18	205
	accuracy			0.54	424
	macro avg	0.58	0.52	0.43	424
	weighted avg	0.58	0.54	0.44	424

```

# Inizializza e allena il modello Random Forest
rf_model = RandomForestClassifier(
    n_estimators=150,
    max_depth=15,
    random_state=42,
    class_weight="balanced"
)
rf_model.fit(x_train_resampled, y_train_resampled)

```

Figura 7: Metriche RF dopo l'unione di dataset eterogenei

Questo approccio è efficace nel trattare problemi di classificazione, ma l'eccessiva predizione di una singola classe rappresenta una limitazione da affrontare, specialmente in contesti con classi sbilanciate. Aggiungendo dati si risolve il problema della classificazione di una sola classe, ma si riduce notevolmente l'accuracy, rendendo però il modello più realistico.

3.2.4 Convolutional Neural Network

Il modello utilizzato in questo codice è una Rete Neurale Convolutionale (CNN) basata sull'architettura InceptionV3, pre-addestrata su ImageNet e adattata per il task di classificazione di immagini. Le CNN sono un tipo di rete neurale particolarmente efficace per il riconoscimento e la classificazione delle immagini, grazie alla loro capacità di rilevare automaticamente le caratteristiche visive (come bordi e texture) attraverso i vari strati convoluzionali.

Nel codice, il modello InceptionV3 è utilizzato come base senza la parte finale di classificazione (top layer), e su di esso vengono aggiunti strati personalizzati per la classificazione. Inoltre, viene eseguita un'operazione di data augmentation per arricchire il training set con variazioni delle immagini (rotazioni, traslazioni, zoom), migliorando la robustezza del modello rispetto a variazioni nei dati.

Per affrontare il problema di classi sbilanciate nel dataset, vengono utilizzati class weights per bilanciare l'importanza delle classi durante l'addestramento vengono calcolati i pesi delle classi in base alla distribuzione del dataset. In particolare, viene applicata una formula che tiene conto della frequenza di ciascuna classe nel dataset e calcola un “peso” che verrà usato per correggere lo sbilanciamento tra le classi.

Dopo l'addestramento, il modello viene salvato e valutato sui dati di test, con metriche come accuratezza, matrice di confusione e classification report. Inoltre, vengono tracciati e visualizzati i progressi dell'accuratezza e della perdita durante l'addestramento.

3.3 Strumenti e Framework

L'addestramento basato su SVM, RF e KNN condividono la maggior parte delle librerie :

- **os**: Gestione dei percorsi e navigazione nel filesystem.
- **cv2 (OpenCV)**:
 - Caricamento, elaborazione e ridimensionamento delle immagini.
- **numpy**: Gestione di array e operazioni numeriche.
- **scikit-learn (sklearn)**:
 - *SVC/RandomForestClassifier/KNeighborsClassifier* : in base al modello di addestramento che si vuole utilizzare
 - *LabelEncoder*: Codifica delle etichette in valori numerici.
 - *accuracy_score, classification_report*: Metriche di valutazione del modello.
- **scikit-image (skimage)**:
 - *hog*: Estrazione delle feature HOG (Histogram of Oriented Gradients).
- **joblib**: Salvataggio e caricamento del modello addestrato.
- **imbalanced-learn (imblearn)**:
 - *SMOTE*: Tecnica di oversampling per bilanciare le classi nel dataset.

3.3.1 CNN

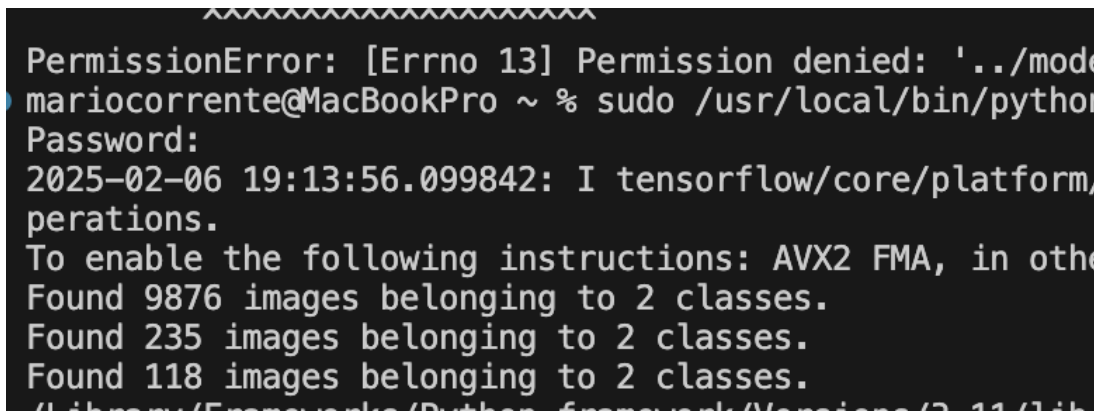
- **os**: Gestione dei percorsi dei file e navigazione nel filesystem.
- **numpy**: Gestione di array e calcoli numerici.
- **matplotlib.pyplot**: Visualizzazione di grafici.
- **tensorflow.keras.applications.InceptionV3**: Modello pre-addestrato per il transfer learning.
- **tensorflow.keras.layers**:
 - *GlobalAveragePooling2D*: Riduzione spaziale delle feature map.
 - *Dense*: Layer completamente connessi per la classificazione.
 - *Dropout*: Prevenzione dell'overfitting.
 - *BatchNormalization*: Normalizzazione nei layer densi.
- **tensorflow.keras.models.Model**: Creazione di modelli personalizzati.
- **tensorflow.keras.optimizers.Adam**: Ottimizzatore per il training.
- **tensorflow.keras.preprocessing.image**:
 - *load_img, img_to_array*: Caricamento e conversione delle immagini.
 - *ImageDataGenerator*: Data augmentation.
- **Callbacks per il Training**:
 - *LearningRateScheduler*: Regolazione del learning rate.
 - *EarlyStopping*: Interruzione anticipata se non ci sono miglioramenti.
 - *ModelCheckpoint*: Salvataggio del miglior modello.
 - *ReduceLROnPlateau*: Riduzione del learning rate in caso di stagnazione.
- **Scikit-learn (Metriche di Valutazione)**:
 - *accuracy_score*: Calcolo dell'accuratezza.
 - *classification_report*: Precision, recall e F1-score.
 - *confusion_matrix*: Matrice di confusione.

4 Problematiche

4.1 Data Augmentation

4.1.1 DCGAN

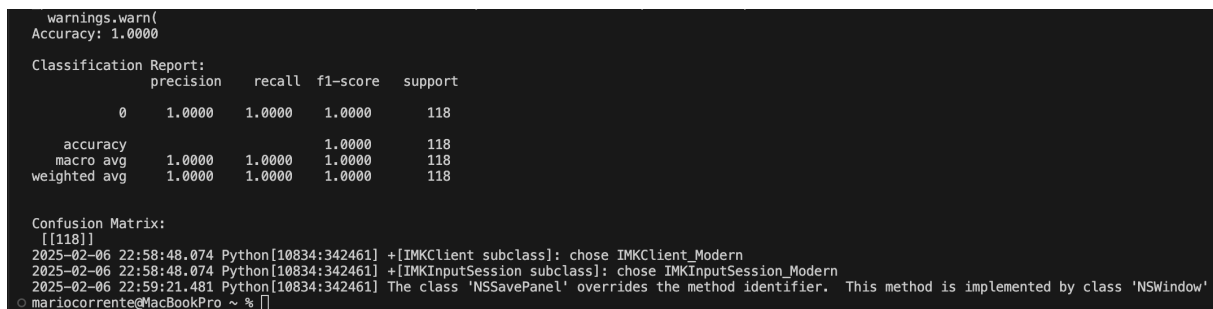
Con il fine di tentare una riduzione dell'overfitting in cui il modello incorreva, si è deciso di procedere con un aumento fisico delle immagini attraverso l'utilizzo della **DCGAN** (*Deep Convolutional Generative Adversarial Network*), portando prima il dataset a circa 10 000 immagini, come si può osservare dallo screen che segue.



```
XXXXXXXXXXXXXXXXXXXX
PermissionError: [Errno 13] Permission denied: '../model
mariocorrente@MacBookPro ~ % sudo /usr/local/bin/python
Password:
2025-02-06 19:13:56.099842: I tensorflow/core/platform
operations.
To enable the following instructions: AVX2 FMA, in othe
Found 9876 images belonging to 2 classes.
Found 235 images belonging to 2 classes.
Found 118 images belonging to 2 classes.
/Library/Frameworks/Python.framework/Versions/3.11/Lib
```

Figura 8: Screen numero immagini per addestramento

Nonostante l'aumento del dataset a 10000 immagini, il modello aveva comunque problemi di overfitting.



```
warnings.warn(
Accuracy: 1.0000

Classification Report:
      precision    recall  f1-score   support

     0       1.0000      1.0000      1.0000        118

 accuracy          1.0000          1.0000          1.0000        118
 macro avg          1.0000          1.0000          1.0000        118
weighted avg          1.0000          1.0000          1.0000        118

Confusion Matrix:
[[118]]
2025-02-06 22:58:48.074 Python[10834:342461] +[IMKClient subclass]: chose IMKClient_Modern
2025-02-06 22:58:48.074 Python[10834:342461] +[IMKInputSession subclass]: chose IMKInputSession_Modern
2025-02-06 22:59:21.481 Python[10834:342461] The class 'NSSavePanel' overrides the method identifier. This method is implemented by class 'NSWindow'
mariocorrente@MacBookPro ~ %
```

Figura 9: Caption

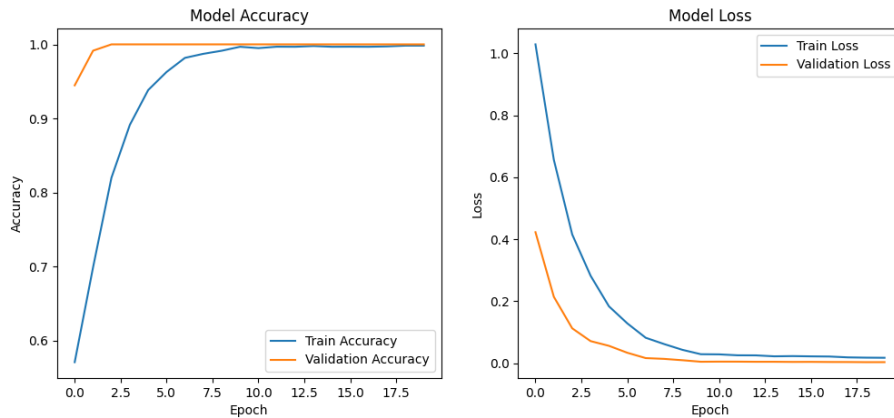


Figura 10: Caption

Riscontrando problemi di overfitting, con una precision del 100%, si è deciso di aumentare nuovamente il dataset fino a 15000 immagini.

```
mariocorrente@MacBookPro ~ % /usr/local/bin/python3
2025-02-10 18:05:31.309831: I tensorflow/core/p
To enable the following instructions: AVX2 FMA,
Found 14814 files belonging to 2 classes.
Found 235 files belonging to 2 classes.
Found 118 files belonging to 2 classes.
```

Figura 11: Screen numero immagini per addestramento

Il tentativo di risolvere l'overfitting del modello CNN con l'ulteriore aumento delle immagini non ha portato ai risultati sperati. Il modello andava in overfitting durante la fase di addestramento già nelle prime epoche.

```
mariocorrente@MacBookPro ~ % /usr/local/bin/python3 /Users/mariocorrente/Desktop
2025-02-10 18:06:45.090360: I tensorflow/core/platform/cpu_feature_guard.cc:210
To enable the following instructions: AVX2 FMA, in other operations, rebuild Te
Found 14814 files belonging to 2 classes.
Found 235 files belonging to 2 classes.
Found 118 files belonging to 2 classes.
Epoch 1/15
926/926 ██████████ 498s 528ms/step - accuracy: 0.7630 - loss: 0.9846
Epoch 2/15
926/926 ██████████ 535s 577ms/step - accuracy: 0.9957 - loss: 0.2918
Epoch 3/15
926/926 ██████████ 570s 615ms/step - accuracy: 0.9991 - loss: 0.1665
Epoch 4/15
250/926 ██████████ 6:40 593ms/step - accuracy: 1.0000 - loss: 0.1023
```

Figura 12: Caption

Inoltre, il problema dell'overfitting non è stato l'unico problema riscontrato. Ciò che ha davvero impedito l'addestramento del modello è stata la potenza computazionale delle macchine a disposizione, come si evince dallo screen seguente.

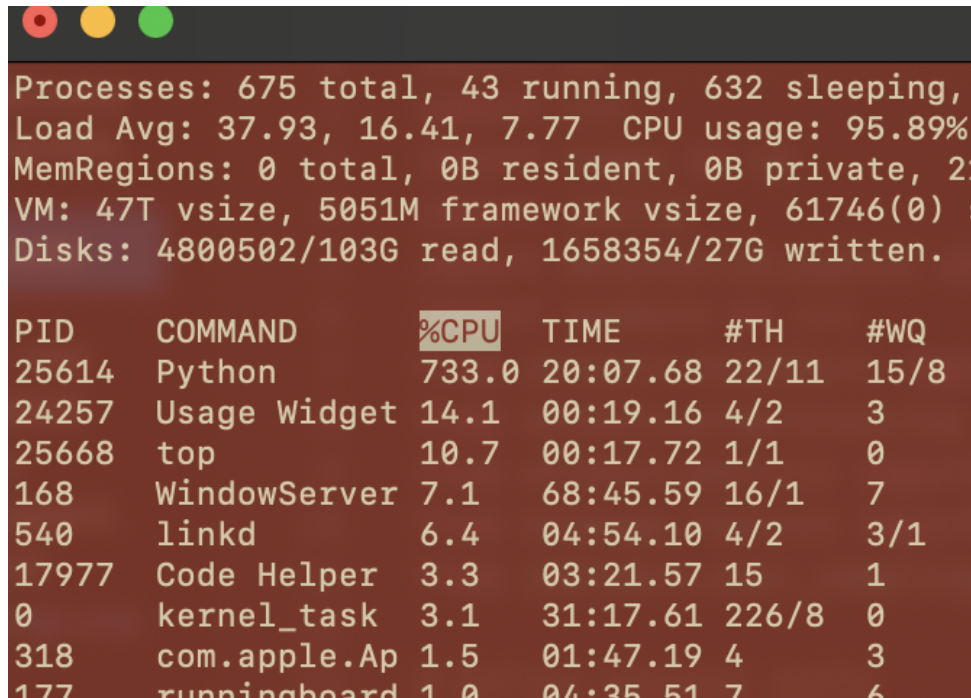


Figura 13: Utilizzo CPU durante addestramento

Purtroppo, non avendo a disposizione delle GPU dedicate all'addestramento non abbiamo potuto affrontare definitivamente il problema e risolverlo.

4.1.2 Cross Validation

In un ultimo tentativo di risolvere l'overfitting, abbiamo applicato la tecnica della Cross-Validation sul dataset impostando il parametro di validazione sui 5 fold. La CV ha impiegato molte risorse della CPU e molte ore per completare il ciclo. Ha impiegato circa 6 ore per completare le operazioni di validazione.

5 Conclusioni

In generale, i modelli di classificazione implementati, inclusi il Support Vector Machine (SVM), il Random Forest (RF), il K-Nearest Neighbors (KNN) e la Convolutional Neural Network (CNN), sembrano promettenti sulla carta e sono in grado di apprendere a partire dai dati. Tuttavia, dopo aver effettuato alcune prove pratiche, emergono delle problematiche significative.

Il modello CNN, pur dimostrando inizialmente buone prestazioni sui dati di addestramento, supponiamo che soffra di overfitting. Questo comportamento è un chiaro indicatore di

una mancata generalizzazione e suggerisce che il modello ha appreso troppo dai dettagli specifici dei dati di addestramento, perdendo la capacità di fare previsioni accurate su dati mai visti prima. Facendo delle prove semplici che potrebbe effettuare un qualsiasi utente, l'applicativo non risulta essere utilizzabile.

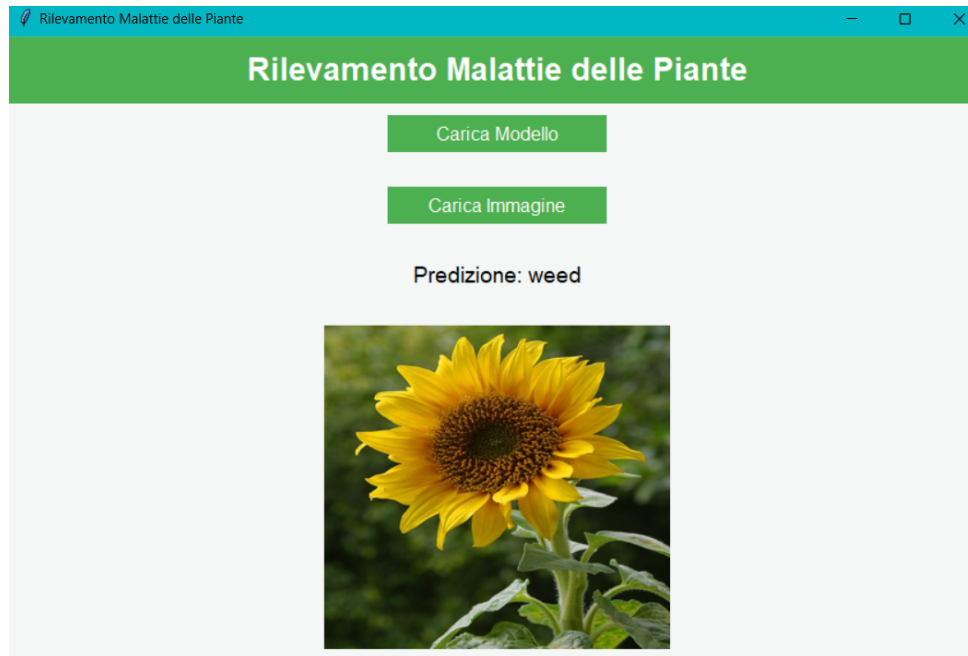


Figura 14: Esempio di interfaccia utente

Gli altri modelli, inclusi SVM, RF e KNN, con il dataset base sembrano performare bene, aggiungendo eterogeneità al dataset risultano molto meno performanti. In alcuni casi, i modelli sembrano classificare in modo quasi casuale, senza alcun pattern evidente che separi correttamente le classi.

Questo potrebbe essere il risultato di una serie di fattori, tra cui la qualità e la quantità limitata dei dati di addestramento, o l'incapacità dei modelli di catturare correttamente le caratteristiche distintive delle classi in modo sufficientemente robusto.

In conclusione, pur mostrando potenziale teorico, i modelli, in particolare CNN che dovrebbe essere il più performante, necessitano di un dataset più ampio ed eterogeneo, che, oltre ad essere difficile da reperire, sarebbe difficile da processare con hardware comuni.