

Università degli Studi Roma Tre
Progetto di Machine Learning

Francesco Salienti 538636, Francesco Spezzano 534050, Francesco Tavaglione 533168

PCB Component Detection using Computer Vision

1 Introduzione

L’obiettivo del progetto¹ è quello di realizzare un software in grado di riconoscere, attraverso tecniche di machine learning e deep learning, i componenti elettronici montati su PCB (Printed Circuit Board). Attraverso l’uso di una rete neurale convoluzionale pre-addestrata si vogliono identificare e localizzare gli oggetti di interesse all’interno dell’immagine. L’obiettivo è individuare le posizioni dei componenti elettronici e assegnare loro le etichette corrispondenti alla tipologia, in modo da dotare le macchine di una “vista” intelligente, permettendo loro di comprendere e interagire con il mondo esterno in modo più efficace. L’applicazione di questo tipo di modelli è sempre più comune nell’industria dell’elettronica, dove è integrato nei macchinari *pick-and-place*, il cui compito è di posizionare correttamente i componenti sulle schede elettroniche.

2 Dataset

Il dataset² utilizzato per il training del modello è composto di 1 039 immagini distinte, che includono rappresentazioni di componenti elettronici sia in isolamento che montati su schede PCB, riflettendo così varie situazioni reali in cui possono essere incontrati. La diversità nelle immagini è necessaria per assicurare che i modelli addestrati siano robusti e versatili, capaci di identificare componenti in diversi contesti e sotto diverse condizioni di illuminazione. Il dataset è organizzato in 6 classi di componenti elettronici:

Classe	Cercamic	Electrolitic	IC	LED	Resistor	Transistor
Immagini	600	500	400	200	1200	400

Table 1: Distribuzione componenti nel dataset

Ogni immagine presente nel dataset è annotata con le coordinate di bounding boxes, che delimitano la posizione esatta dei componenti e le rispettive etichette che ne identificano le classi.

¹https://github.com/Francsco99/pcb_detection

²<https://universe.roboflow.com/ups-cthsu/tesis-abye0/dataset/1>



Figure 1: Annotazioni di componenti isolate

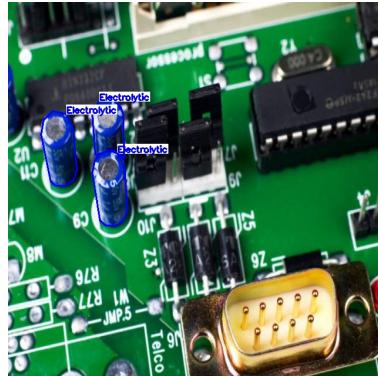


Figure 2: Annotazioni di un componente su PCB

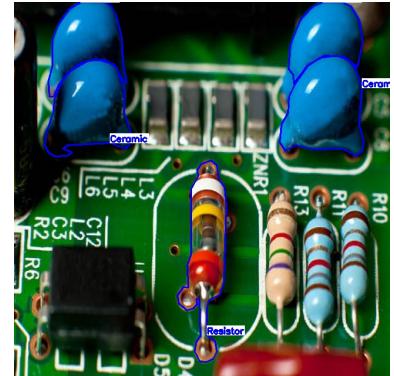


Figure 3: Annotazioni di più componenti su PCB

Si osserva una distribuzione non uniforme delle classi tra le immagini del dataset. In particolare, la classe più rappresentata è Resistor con ~ 1200 istanze. Seguono Ceramic (~ 600 istanze), Electrolytic (~ 500 istanze), IC (~ 400 istanze), Transistor (~ 400 istanze) e LED (~ 200 istanze). Questa disparità porta a:

- bias di addestramento;
- asimmetria delle prestazioni;
- incremento del rischio di classificazioni errate.

Inoltre la distribuzione non uniforme delle istanze delle classi nel dataset condiziona la matrice di correlazione, influenzando la frequenza delle associazioni tra le diverse classi. Questo potrebbe portare a una percezione distorta delle relazioni tra le classi e influenzare la precisione del modello nell'identificare e classificare gli oggetti.

Nei successivi capitoli, esamineremo l'impatto di questa distribuzione non uniforme del dataset in termini di precision e recall nei due approcci adottati.

2.1 Data Pre-processing

Per garantire la consistenza nel training e nelle valutazioni dei modelli è stata applicata una procedura di pre-processing al dataset che prevede:

- auto-orientamento: le immagini vengono orientate automaticamente;
- ridimensionamento: le immagini vengono adattate a una risoluzione di 640x640 pixel;
- filtro sui valori Null: è richiesto che almeno l'85% delle immagini contenga annotazioni.

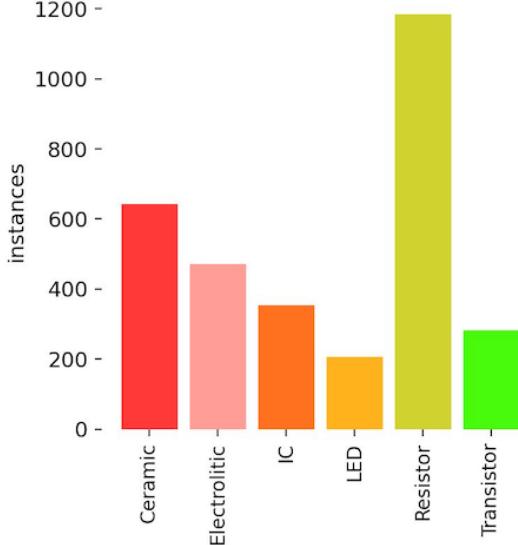


Figure 4: Distribuzione delle classi

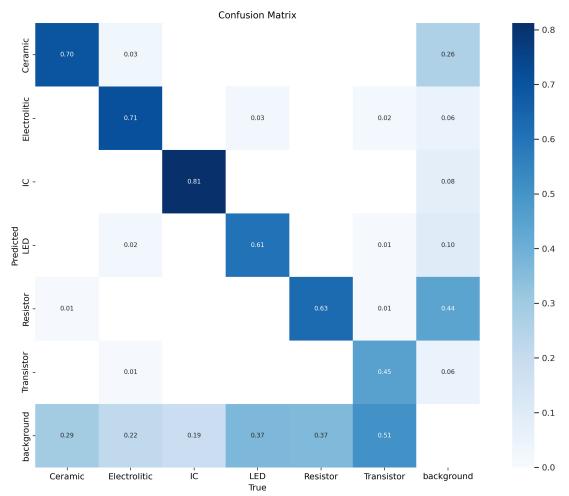


Figure 5: Matrice di correlazione tra classi

2.2 Train-Test Split

La suddivisione del dataset segue una distribuzione che assegna il 79% delle immagini al Training Set, il 14% al Validation Set e il 7% al Test Set. Questa distribuzione è stata scelta per massimizzare la quantità di dati disponibili per l’addestramento dei modelli, pur mantenendo una porzione sufficiente di immagini per la validazione e il test.

3 Training del modello

Inizialmente è stato utilizzato il framework open-source TensorFlow per generare un modello. Successivamente è stato replicato un modello simile utilizzando YOLO, un algoritmo di rilevamento degli oggetti.

Le principali differenze tra TensorFlow e YOLO si riflettono nei loro approcci di implementazione, flessibilità e prestazioni. Il primo è più adatto per progetti che richiedono maggiore controllo e personalizzazione dei modelli, mentre il secondo eccelle in contesti in cui la velocità e l’efficienza del rilevamento sono decisive, come nelle applicazioni di video surveillance e nelle automobili autonome. La scelta tra TensorFlow e YOLO dipende quindi dalle specifiche esigenze del progetto e dalle priorità in termini di prestazioni, flessibilità e facilità d’uso.

3.1 Setup Sperimentale

L’addestramento dei modelli è stato condotto utilizzando il piano gratuito offerto da Google Colab³. Le specifiche dell’hardware utilizzato sono dettagliate nella Tabella 2.

³<https://colab.research.google.com/>

Componente	Specifiche
Modello CPU	Intel(R) Xeon(R)
Frequenza CPU	2.30GHz
Nuclei CPU	2
RAM Disponibile	12GB (espandibile a 26.75GB)
Spazio su Disco	50GB
Modello GPU	Nvidia T4
Memoria GPU	16GB
Frequenza Memoria GPU	1.59 GHz
Performance GPU	8.1 TFLOPS

Table 2: Specifiche dell’hardware utilizzato per gli addestramenti su Google Colab.

3.2 Utilizzo della GPU

Il tempo necessario ai modelli di TensorFlow e YOLOv5 per eseguire il training, varia dai 45 ai 60 minuti (in un ambiente Google Colab free). Questo è dovuto al fatto che il modello, se eseguito in un ambiente Python standard, effettua le elaborazioni utilizzando la CPU. È però possibile abilitare l’uso della GPU per le elaborazioni in modo tale da ottenere dei risultati in tempi molto più brevi. Grazie a questo cambiamento, il tempo necessario al modello per ottenere un’immagine di output accurata è stato ridotto a circa 10 minuti (scheda grafica Nvidia T4).

Questo significativo incremento delle prestazioni ci ha consentito di effettuare diverse elaborazioni in tempi ridotti, portando quindi alla realizzazione di numerose immagini risultato.

4 TensorFlow

TensorFlow⁴ è un framework open-source per il machine learning sviluppato da Google, che supporta una vasta gamma di compiti di machine learning e deep learning. In particolare, offre la possibilità di costruire, addestrare e distribuire modelli di object detection con relativa facilità.

4.1 Preparazione del dataset

La suddivisione del dataset in Training Set e Validation Set è stata effettuata dagli autori, l’unica operazione di preparazione necessaria è stata generare un file CSV contenente le informazioni sulle immagini e le loro annotazioni, insieme a un file `label_map.pbtxt` che effettua il mapping tra gli ID delle classi e i nomi dei relativi componenti elettronici.

4.2 Fase di training

Il modello di rete neurale convoluzionale (CNN) pre-addestrato utilizzato è denominato `ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8`. Questo modello viene utilizzato per l’object de-

⁴<https://www.tensorflow.org/>

tection ed è basato sull’architettura Single Shot Multibox Detector (SSD) con una rete MobileNetV2 come feature extractor.

- **Architettura SSD:** SSD combina rilevamento e classificazione in un’unica rete neurale, utilizzando feature maps a diverse scale per predire bounding box e classificazioni simultaneamente.
- **MobileNetV2:** MobileNetV2 è una CNN progettata per applicazioni con risorse computazionali limitate, ottimizzata per fornire alte prestazioni con una computazione ridotta.
- **FPN-Lite:** FPN-Lite è una versione leggera dell’architettura Feature Pyramid Network, utilizzata per estrarre feature multiscale da immagini.
- **Dataset COCO:** il modello è stato preaddestrato sul dataset COCO⁵, che contiene oltre 20 000 immagini annotate con oltre 80 categorie di oggetti comuni.

Nel file di configurazione del modello sono stati impostati i seguenti parametri:

- **num_step:** indica il numero totale di passi che verranno eseguiti sull’intero set di dati durante l’addestramento del modello. Un passo rappresenta un’iterazione in cui il modello riceve un batch di dati di addestramento, calcola le perdite e aggiorna i pesi del modello tramite ottimizzazione. Si è scelto di impostare questo parametro a 10 000 dato che il valore di default (50 000) avrebbe reso il processo di training estremamente lungo.
- **batch_size:** indica il numero di campioni di addestramento utilizzati in ogni iterazione del processo di addestramento. Un batch è un insieme di campioni di dati di addestramento che vengono processati insieme in parallelo. Questo parametro è stato impostato su 16 (valori multipli di 8 permettono di ottenere migliori prestazioni quando si utilizzano TPU Google).

```
!python model_main_tf2.py --pipeline_config_path=/mydrive/customTF2/
    data/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config --
    model_dir=/mydrive/customTF2/training --alsologtostderr
```

4.3 Rapporto Precision-Recall

Il grafico evidenzia che tutte le classi hanno una tendenza generale comune: la precision diminuisce man mano che la recall aumenta. Questo è tipico dei sistemi di object detection poiché aumentare la recall di solito comporta l’accettazione di più falsi positivi, che abbassa la precision. In generale, è importante trovare un trade-off tra questi due parametri che soddisfi i requisiti specifici dell’applicazione.

Le variazioni osservate nelle curve di precision e recall delle diverse classi possono essere attribuite alla disparità nella distribuzione delle classi all’interno del set di dati. Tale disparità implica che le classi con un maggior numero di esempi di addestramento mostrano curve di precision e recall più stabili rispetto a quelle con una numerosità inferiore.

⁵<https://cocodataset.org>

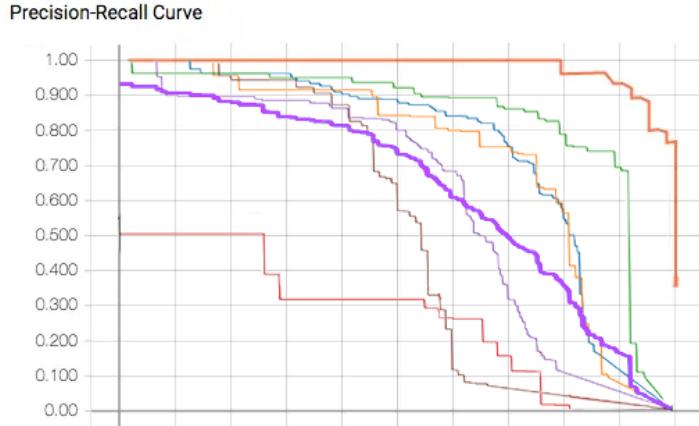


Figure 6: Grafico Precision-Recall per le diverse classi di componenti elettronici.

4.4 Output del modello

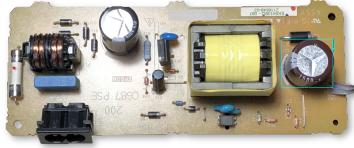


Figure 7: Test con luce incidente

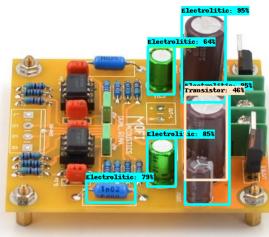


Figure 8: Test su Validation Set

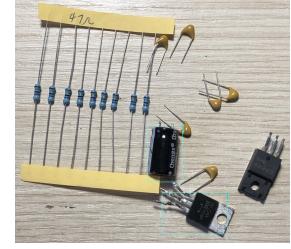


Figure 9: Test su componenti isolati

5 YOLOv5

YOLOv5⁶ è un moderno algoritmo di rilevamento di oggetti che opera in tempo reale e fa parte della famiglia YOLO di modelli di deep learning. A differenza dei metodi di rilevamento in due fasi come R-CNN, YOLOv5 adotta un approccio in un'unica fase (You Only Look Once), eseguendo contemporaneamente classificazione e localizzazione degli oggetti durante una singola passata attraverso la rete neurale. Divide l'immagine in una griglia e predice, per ogni cella, le bounding boxes e le relative probabilità di appartenenza alle classi, utilizzando attributi come coordinate centrali, dimensioni e probabilità di confidenza.

YOLOv5 è disponibile con diversi modelli preaddestrati sul dataset COCO 4.2, che variano per dimensione e complessità (da YOLOv5s a YOLOv5x), permettendo l'utilizzo efficiente del transfer learning per adattare questi modelli a compiti specifici, riducendo così il tempo di addestramento e la quantità di dati necessari.

⁶<https://github.com/ultralytics/yolov5>

5.1 Fase di Training e Validation

Il modello di rete neurale convoluzionale (CNN) pre-addestrato scelto dopo alcuni test, è stato il modello `yolov5m` per un totale di 100 epoche, ottenendo un buon compromesso tra prestazioni e tempo di addestramento.

```
!python train.py --img 416 --batch 16 --epochs 100 --data dataset.yaml --  
weights yolov5m.pt --cache --name run_50_medium_1
```

I parametri specificati nel comando sono:

- `-img 416`: per ridimensionare le immagini di input a 416x416 pixel;
- `-batch 16`: numero di immagini processate in parallelo durante l'addestramento;
- `-epochs 100`: numero totale di epoche di addestramento, con un'epoca che rappresenta una passata completa attraverso il dataset;
- `-weights yolov5m.pt`: il file dei pesi iniziali del modello preaddestrato;
- `--cache`: memorizza in cache le immagini di addestramento per ridurre i tempi di caricamento nelle epoche successive.

5.2 Rapporto Precision-Recall

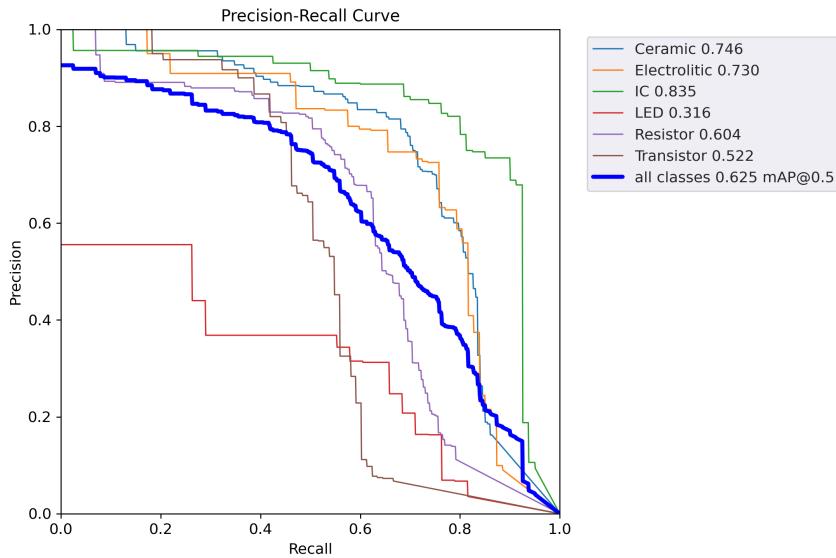


Figure 10: Grafico Precision-Recall per le diverse classi di componenti elettronici.

Anche in questo caso l'andamento delle classi è analogo a quello visto in TensorFlow, la precision diminuisce al crescere della recall.

- **Aree Sotto le Curve (AUC):** ogni classe ha un valore AUC che rappresenta l'area sotto la curva Precision-Recall (PR). Un AUC più alto indica prestazioni migliori, evidenziando una combinazione di alta precisione e recall.
- **Classi Individuali:** ogni classe è valutata separatamente in base al suo AUC. Classi come IC mostrano una performance elevata con un AUC di 0.835, mentre LED ha prestazioni notevolmente inferiori con un AUC di 0.316.
- **Performance Complessiva:** il mean Average Precision (mAP) aggregato per tutte le classi, a una soglia di Intersection over Union (IoU) del 50%, è del 62.5%. Questo indica una buona capacità media del modello di rilevare gli oggetti.

5.3 Grafico F1-Confidence

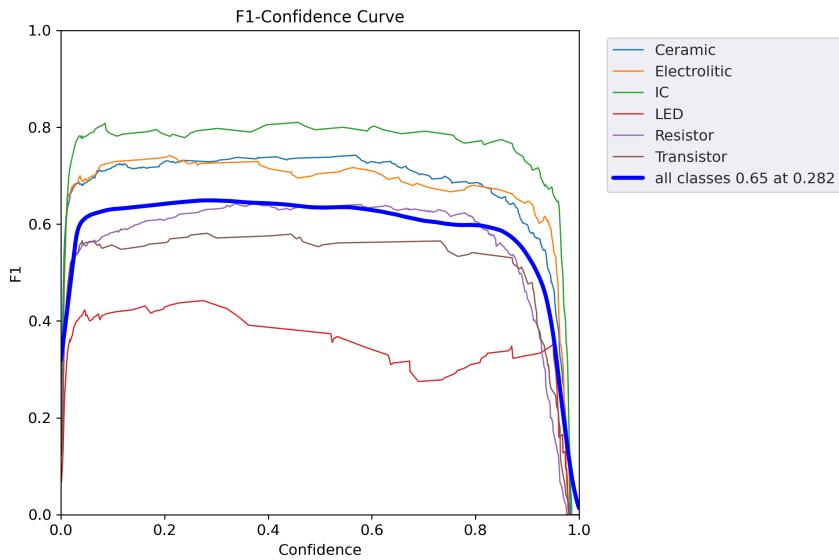


Figure 11: Curva F1-Confidence per diverse classi di componenti elettronici

- **Asse Y (F1 Score):** il punteggio F1 è una misura che combina precisione e recall in un unico valore, utile per valutare il bilanciamento tra l'identificazione accurata degli oggetti (precision) e la capacità di identificare tutti gli oggetti rilevanti (recall).
- **Asse X (Confidence):** l'asse orizzontale rappresenta il livello di confidenza del modello, che è la soglia utilizzata per decidere se una predizione per una certa classe è sufficientemente affidabile da essere considerata corretta.
- **Curve delle classi:**
 - Curve più vicine all'angolo in alto a destra (valori alti di precision e recall) indicano prestazioni migliori.

- La curva ‘IC‘ ha il punteggio F1 più elevato per tutti i livelli di confidenza, il modello è particolarmente efficace nell’identificare correttamente gli oggetti di questa classe.
- Al contrario, la curva ‘LED‘ ha il punteggio F1 più basso, indicando che il modello fatica ad identificare questa classe sia in termini di precision che di recall.
- Le altre classi, come ‘Ceramic‘, ‘Electrolytic‘, ‘Resistor‘, e ‘Transistor‘, hanno prestazioni intermedie.

• Prestazioni generali del modello:

- La linea blu spessa rappresenta la media delle prestazioni su tutte le classi, indicata dal valore mAP (mean Average Precision) a una soglia di confidenza specifica. In questo grafico, il valore è 0.65 a una soglia di confidenza di 0.282. Nel complesso il modello ha una buona capacità di identificazione degli oggetti, ma ci sono variazioni significative tra le diverse classi.

5.4 Applicazione Web

Per rendere accessibile a un utente l’ottenimento di risultati attraverso il modello, abbiamo scelto come interfaccia un’applicazione web sulla quale è possibile caricare l’immagine di una PCB e viene utilizzato il modello addestrato nella sezione 5 per effettuare la detection dei componenti. Questa applicazione web è stata costruita con l’ausilio del micro-framework “Flask” per Python.

5.5 Output del modello

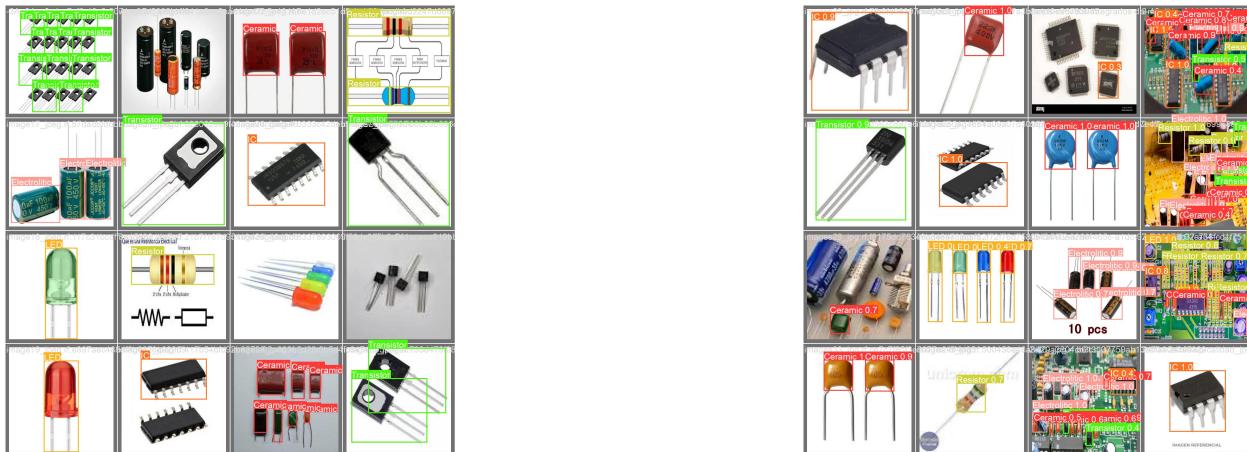


Figure 12: Test su Validation set



Figure 13: Immagine di test con luce incidente

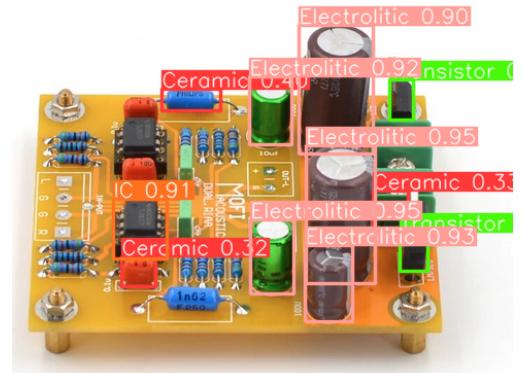


Figure 14: Immagine di test con luce uniforme

6 Conclusioni

La comparazione tra l'utilizzo del framework TensorFlow e l'algoritmo YOLO nel riconoscimento di componenti elettronici mostra che entrambi offrono prestazioni elevate ma con differenze in termini di velocità di inferenza e gestione delle risorse.

YOLOv5 si distingue per la sua velocità e efficienza, rendendolo adatto per applicazioni in tempo reale che richiedono rapidità di elaborazione. TensorFlow, d'altra parte, offre un'elevata flessibilità e una vasta gamma di opzioni di tuning, che possono tradursi in un'accuratezza leggermente superiore a scapito della velocità di elaborazione.

La distribuzione non uniforme delle classi tra le immagini del dataset ha influito negativamente sulle performance di riconoscimento di TensorFlow e YOLOv5, causando una scarsa precisione nel rilevare alcune classi e una sovrastima nel riconoscere altre. Complessivamente entrambi i modelli hanno portato a risultati soddisfacenti nella maggior parte delle immagini presenti nel Test Set.