



Dipartimento Politecnico di Ingegneria e Architettura (DPIA)
Corso di Visione Artificiale

Progetto: 3D Printer Defects Classification

Studente: Lorenzo Zaccomer

Data: 27 aprile 2022

Estratto

Progetto svolto per il corso di Visione Artificiale tenuto dal Professore Andrea Fusiello nell'anno accademico 2021/2022.

Lo scopo di questo progetto è quello di generare un modello in linguaggio Python affinché possa determinare la difettosità o meno di un pezzo stampato mediante una stampante 3D da un'immagine passata come parametro.

Indice

1	Introduzione	1
1.1	La stampante 3D	1
1.2	Descrizione del problema	2
1.3	Nascita dell'idea per questo progetto	3
2	Descrizione del progetto	4
2.1	Introduzione	4
2.2	Transfer Learning	4
2.3	Struttura del codice	4
2.4	Funzionamento	6
3	Risultati	9
4	Conclusioni	11

1 Introduzione

1.1 La stampante 3D

La stampante 3D è un dispositivo attorno al quale vi ruota un grosso interesse al giorno d'oggi, questo perché è impiegabile in molti campi applicativi, sia in ambito industriale, sia in ambito *user level*, poiché essa permette un grosso risparmio in termini di costi progettuali, in aggiunta alla facilità di manutenzione.

Con una spesa iniziale non troppo elevata per la stampante in sé (200€ per un modello base), permette a chi si interfaccia per la prima volta a questo mondo, di *prodursi* i pezzi desiderati, magari anche progettati da autodidatta, inoltre il suo utilizzo è facilitato dalla presenza di svariati tool di disegno grafico presenti sul mercato, come per esempio *OpenSCAD*, il quale è open-source e scaricabile gratuitamente, e grazie alla comunità di *Thingiverse* gli utenti possono condividere i propri progetti, ed infine con *UltimakerCura* è possibile modificare le impostazioni di stampa.

Si pensi che, ad esclusione ovviamente delle componenti elettroniche, come i servo motori, è possibile crearsi una propria stampante 3D da un'altra stampante 3D.

Il materiale maggiormente impiegato per le stampe è il PLA, la quale è una plastica non tossica.

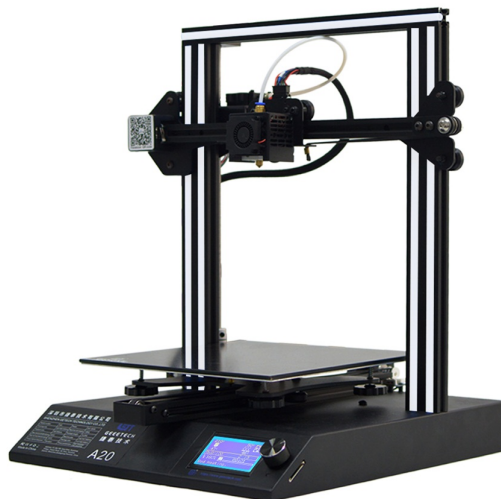


Figura 1.1: Esempio di stampante 3D, modello Geeetech A20

1.2 Descrizione del problema

Essa però presenta una serie di problemi durante l'utilizzo, il più tangibile per esempio è il rumore emesso dall'elettronica e dalle componenti meccaniche, per cui è necessario posizionarla in un'altra stanza ed il più lontano possibile, anche nel caso in cui si impieghi una box dedicata, il problema si attenua ma non scompare, perciò questa problematica legata al rumore rende impossibile un controllo *a vista* dell'esecuzione di stampa, per cui in caso di problemi si rischia di far proseguire la stampa per ore, con uno spreco di materiale di stampa, di tempo e consumo elettrico.

Inoltre un parametro che può rendere difettoso il pezzo è la **temperatura**, in particolare di più elementi:

- *stanza* stessa cui si trova la stampante
- *piano di adesione* della stampa
- *ugello* (nozzle) della stampante

Questo perché se la temperatura di questi tre elementi non è adeguata, si rischia una ritrazione del pezzo, con conseguente difettosità della stampa.

Nel primo caso è maggiormente risolvibile, questo riscaldando la stanza con un classico scaldabagno.

Nel secondo caso l'ugello deve lavorare alla temperatura di fusione del materiale in uso, ovvero intorno ai 200° per il PLA, questo parametro è settabile dalla stampante stessa mediante lo schermo, poiché se la temperatura è troppo bassa, il filamento non si scalda e la stampa non viene eseguita, se la temperatura è troppo alta, il filamento rischia di fare quello che si chiama in gergo *effetto spaghetti*, il quale può causare un danneggiamento estetico del pezzo ed un impedimento al movimento nel nozzle se si deposita sul punto sbagliato.

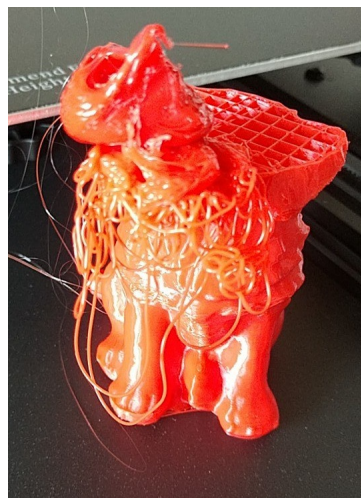


Figura 1.2: Esempio dell'effetto spaghetti

Per quanto riguarda il piano di adesione, le temperature operative si aggirano dai 30° agli 80°, però se esse non sono adeguate si rischia che in un caso la base del pezzo si solidifichi troppo, facendo sì che il pezzo si *stacchi completamente* dal piano, o in effetto minore il sollevamento di una sua parte, infine se le temperature del piano sono troppo alte il pezzo non si raffredda nella maniera giusta, rischiando di produrre delle pieghe sulle superficie esterne del pezzo.

Tale problema di sollevamento del pezzo è mitigabile programmando un layer aggiuntivo di stampa attorno al pezzo, impostabile mediante UltimakerCura.

Perciò l'addetto alla stampa, deve tenere conto di una moltitudine di possibili eventi, soprattutto nel caso in cui un pezzo richieda della ore per essere generato.

1.3 Nascita dell'idea per questo progetto

L'idea nasce quando mi è stata regalata una stampante 3D (Figura 1.1) e le problematiche descritte precedentemente le ho affrontate in prima persona durante l'utilizzo della stessa, ho pensato per mesi a come minimizzare il problema, ma grazie all'esperienza acquisita grazie ad un altro progetto svolto in un altro corso universitario, e dall'opportunità datami per questo corso, unito ad un insieme di fattori tutto questo mi ha spronato a sviluppare questa idea che vi sto presentando.

2 Descrizione del progetto

In questa sezione si andrà ad analizzare le scelte tecniche fatte per l'implementazione, piuttosto che una descrizione riga per riga del codice stesso, inoltre si darà un'infarinatura generale dei concetti teorici che verranno citati.

2.1 Introduzione

Lo scopo di questo progetto è quello di generare un modello, nello specifico una rete convoluzionale, affinché possa determinare da un'immagine di un pezzo prodotto da una stampante 3D se esso sia difettoso oppure no, questo dopo un'attività di training per addestrare la rete convoluzionale al meglio, per fare ciò ho utilizzato la tecnica del *Transfer Learning*.

2.2 Transfer Learning

Tale tecnica permette di addestrare il nostro modello "campione" mediante un modello di training pre-allenato, il quale ha avuto a disposizione un dataset di immagini molto ampio (attorno al milione) per svolgere il suo allenamento, per cui è in grado di classificare correttamente le immagini, mentre il nostro non ha un dataset così ampio per allenarlo correttamente, a me piace utilizzare l'esempio del personal trainer in palestra per descrivere questa tecnica.

Tutto questo permette un vantaggio in termini computazionali.

2.3 Struttura del codice

Esso si compone di 3 layer principali:

- `classification.py` (GitHub)
- `model.py` (GitHub)
- cartella `libraries`

Il primo file (**`classification.py`**), permette l'esecuzione vera e propria della classificazione delle immagini, in particolare è fissata per default una sola immagine (`7.jpg`), e su schermo verrà visualizzata l'immagine normalizzata con un testo che ne indica la predizione, come indicato in Figura 2.1.

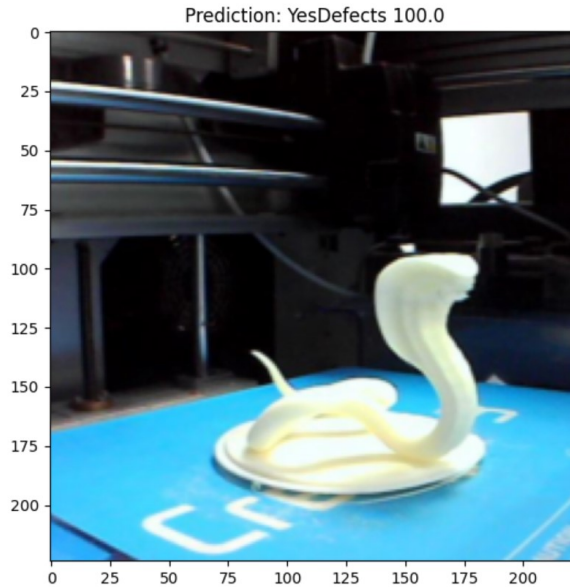


Figura 2.1: Esempio di predizione che viene prodotta

Ho implementato inoltre anche una parte di compilazione personalizzata del codice affinché l'utente possa effettuare un *tuning* molto veloce di alcuni parametri, come per esempio il learning rate, il percorso delle immagini e così via.

Per esempio:

```
yourpath\3D-printer-recognition>python .\Project\classification.py  
↪ --test_image_path './Images\classification-images\9.jpg'
```

Codice 2.1: Primo esempio di compilazione personalizzata

In questo caso vogliamo cambiare l'immagine di test che verrà visualizzata, mentre in questo altro caso si può personalizzare il numero di epoch.

```
yourpath\3D-printer-recognition>python ./Project/classification.py  
↪ --epochs 3
```

Codice 2.2: Secondo esempio di compilazione personalizzata

Il secondo file (**model.py**), racchiude tutte le funzionalità necessarie per la generazione del modello e la sua visualizzazione, includendo quindi due funzioni specifiche per questo compito, vedasi le funzioni citate al listato 2.4.

Mentre nella cartella **libraries**, sono contenuti tutti quei file aventi tutte le altre funzioni, questo per non raggruppare il tutto in un singolo file (complicando la scrittura e visualizzazione del codice).

Infine, è stata implementata tutta una sequenza di logging affinché essi vengano salvati su un file specifico, questo per permettere un debugging facilitato rispetto al piazzare delle print in giro per il codice e una compilazione del progetto in maniera personalizzata, in modo da eseguire un tuning molto veloce di quelli che possono essere alcuni parametri, senza metter mano al codice.

2.4 Funzionamento

Si va ad eseguire dall'IDE il file `classification.py`, il quale inizialmente effettua un controllo sui parametri che vengono passati dalla linea di comando¹ e se ci sono errori nella scelta dei parametri l'esecuzione viene terminata immediatamente, questo per far sì che non si verifichino problemi durante l'esecuzione del codice stesso, questo controllo viene svolto mediante la seguente funzione:

```
CheckParametersErrors(option.epochs, option.learning_rate,  
    ↪ option.iteration, option.visualize_prediction,  
    ↪ option.test_image_path, option.model_path,  
    ↪ option.dataset_images_path)
```

Codice 2.3: Controllo errori (`classification.py`)

Successivamente, in base all'opzione di iterazione, si passa o meno alla funzione che genera il nostro modello:

```
if option.iteration == 1:  
    print('execution of model generation function')  
    ModelGeneration(option.dataset_images_path, option.model_path,  
        ↪ option.epochs, option.learning_rate)  
elif option.iteration == 0 and option.visualize_prediction == 1:  
    print('skip model generation, it loads the model from path and  
        ↪ visualize the results')  
    LoadModelVisualization(option.dataset_images_path,  
        ↪ option.model_path)  
    exit()  
else:  
    print("loading image classification ..")
```

Codice 2.4: Opzione di iterazione (`classification.py`)

Si hanno quindi tre modalità operative in base al valore di due flag che gli vengono passati, *iteration* e *visualize prediction*, dove se:

¹O dal file `launch.json` se si utilizza il debugger di Visual Studio Code

- *iteration = 1*, allora **viene eseguito tutto il codice**, ovvero la generazione con conseguente salvataggio del modello, la visualizzazione del modello con un numero di immagini pari al *batch size*², questo prelevando delle immagini randomiche dal dataset "mixato", ed infine la classificazione dell'immagine di prova.
- *iteration = 0 e visualize prediction = 1*, in questo caso viene eseguita **solamente la visualizzazione della predizione**, saltando la generazione del modello, questo può tornare utile per fare più prove sullo stesso modello senza rigenerarlo continuamente.
- *iteration = 0 e visualize prediction = 0*, è il **caso di default**, ovvero viene eseguita solamente la classificazione dell'immagine desiderata, saltando la parte di generazione del modello e la visualizzazione della predizione.

Proseguiamo ora con la prima scelta.

Per la generazione del modello quello che si svolge in partenza è estrarre due dataset di immagini, aventi con riferimento due cartelle differenti, *train* e *valid* nel nostro caso, dove la prima è molto più grande dell'altra, e si esegue un mix (shuffle) tra questi due dataset in modo da renderli indistinguibili prelevando un numero randomico di immagini da entrambi i dataset, nello specifico questo viene svolto dalla seguente funzione:

```
def ShuffleDatasets(dataset, subs):
    "This function takes the datasets and shuffle them"

    """Not set num_workers because this settings can create
    ↪ compiling error, so leave the default value"""
    return {x: DataLoader(dataset[x], batch_size=4, shuffle=True)
            ↪ for x in subs}
```

Codice 2.5: Funzione ShuffleDatasets (datasets.py)

In questo caso ho lasciato il numero di num workers³ al valore di default (0) perché altrimenti genera un *multi processing error* su sistemi Windows.

Si passa quindi all'elaborazione vera e propria del modello che vogliamo allenare, per cui inizialmente viene caricato il modello di pre-allenato:

```
pretrained_model = models.resnet50(pretrained=True)
```

Codice 2.6: Caricamento del modello pre-allenato (model.py)

²Numero di immagini che si vogliono utilizzare per generare il modello per iterazione

³Numero di sotto processi che si possono eseguire

In questo caso ho scelto il modello resnet50, poiché l'ho trovato molto valido rispetto ad altre opzioni, come per esempio il resnet18.

Infine, dopo aver creato i parametri di ottimizzazione, si passa al training del modello desiderato:

```
starting_time = time.time()
generated_model = train_model(mixed_datasets, dataset_sizes,
    ↪ pretrained_model, criterion, optimizer_ft, exp_lr_scheduler,
    ↪ EPOCH_NUMBER)
logging.info('Training time: {:.10f}
    ↪ minutes'.format((time.time()-starting_time)/60))
```

Codice 2.7: Training del modello (model.py)

Successivamente il modello viene salvato localmente nella directory specificata e se ne visualizzerà la predizione:

```
logging.info("saving the model ..")
torch.save(generated_model, MODEL_PATH)
logging.info("model saved!")

logging.info("visualize generated model ..")
visualize_generated_model(mixed_datasets, labels, generated_model)
print("closing ..")
```

Codice 2.8: Predizione del modello (model.py)

Infine, se tutto è andato nel verso giusto, si esce da questa funzione e si torna al file classification.py, il quale avrà il mero compito di caricare il modello precedentemente creato dalla directory specificata e l'immagine che vogliamo classificare e visualizzarla, come in Figura 2.1.

Inoltre l'utente avrà a disposizione un file di log generato nella directory principale per visualizzare tutti i parametri desiderati e l'andamento di tutti i passaggi svolti dal programma.

3 Risultati

I risultati all'inizio erano molto variabili, ovvero dipendevano fortemente in base a come si concludeva la generazione del modello, ottenendo risultati molto buoni oppure predizioni completamente errate, per cui dopo alcune simulazioni, fissando i seguenti parametri:

$$\begin{aligned} \text{learning rate}^4 &= 0.05 \\ \text{epoch}^5 &= 1 \\ \text{modello pre-allenato} &= \text{resnet50} \end{aligned}$$

Il modello ha cambiato completamente comportamento, aumentando di molto la precisione, puntando però ad essere leggermente aggressivo, ovvero a vedere l'errore anche dove non si trova in alcuni frame, ma con una generale corretta predizione per l'errore, in questo caso il tempo di training per questo modello è sull'ordine di qualche minuto, questo a causa del basso numero di epoch, impostato per ridurre il tempo delle simulazioni stesse in fase di scrittura del codice.

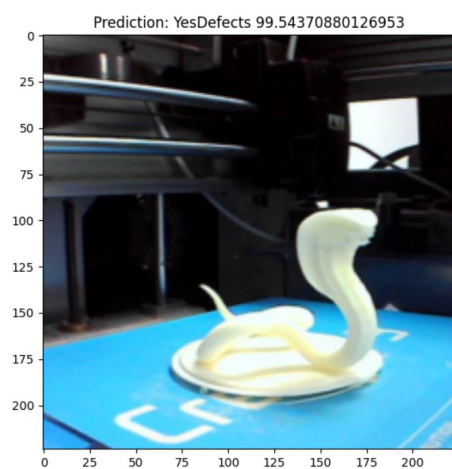
Inoltre è da considerare il fatto che i risultati sono fortemente influenzati da:

$$\begin{aligned} &\text{modello di pre-allenamento}^6 \\ &\text{optimizer}^7 \\ &\text{learning-rate-scheduler} \\ &\text{parametri di tuning (epoch, learning rate, ...)} \end{aligned}$$

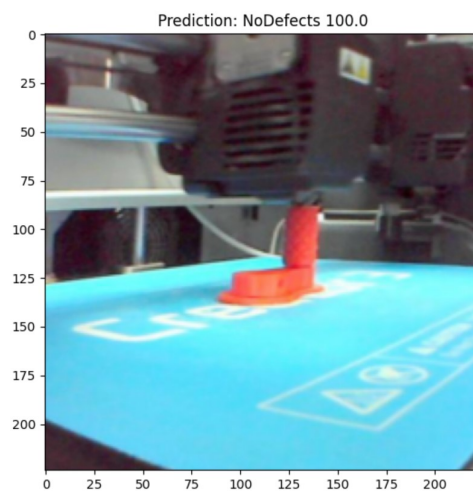
Questo perché per ognuno di questi parametri esistono molte varianti utilizzabili, per cui la scelta risulta molto complessa.

Successivamente, per ottenere una buona precisione anche per i pezzi non difettosi, ho modificato il numero di iterazioni, fissando il numero di epoch a 25, ciò ha richiesto circa 35 minuti per processare il modello, così facendo ho trovato un buon grado di accuratezza nella predizione delle immagini, in entrambi i casi, laddove prima si arriva ad una accuratezza del 76%, ora in vece è pari all'88%.

Alla pagina successiva è possibile visualizzare i risultati ottenuti per alcune immagini di test.

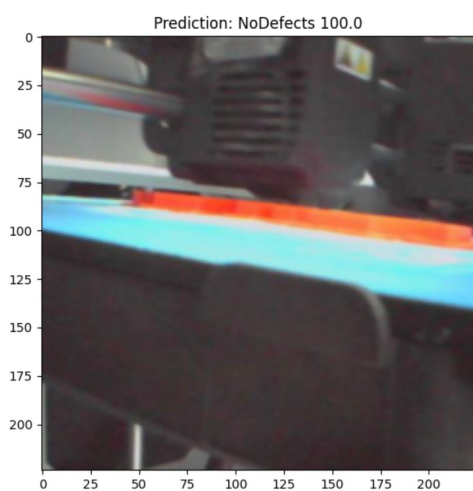


(a) Pezzo difettoso: Predizione corretta

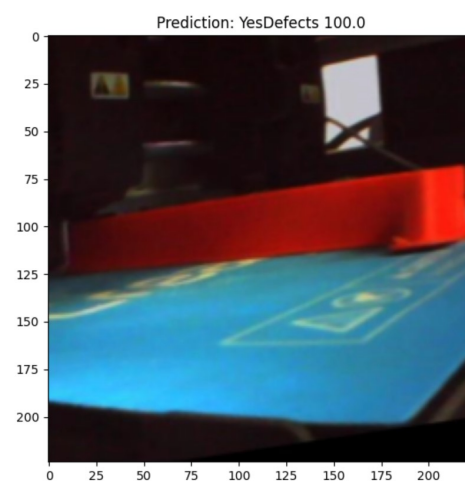


(b) Pezzo non difettoso: Predizione corretta

Figura 3.1: Primo esempio di predizioni



(a) Pezzo non difettoso: Predizione corretta



(b) Pezzo non difettoso: Predizione corretta

Figura 3.2: Secondo esempio di predizioni

4 Conclusioni

Penso che questo progetto possa essere una valida alternativa, se sviluppata nel modo corretto, ad altre soluzioni già presenti, questo perché mostra una buona accuratezza nel svolgere la predizione, inoltre il salvataggio del modello in locale permette di essere utilizzato su dispositivi molto meno potenti di un computer, come per esempio un Raspberry Pi, e utilizzare quest'ultimo solo come mero osservatore dello sviluppo della stampa.

Inoltre come idea si può pensare di dare una maggiore specificità all'errore trovato, per esempio:

- *NoDefects - Spaghetti*
- *NoDefects - Lift*

Questo sarebbe fattibile modificando le cartelle immagini e aggiungendo le immagini giuste nelle corrispondenti cartelle.

Inoltre volendo, mediante le deduzioni fatte precedentemente, il frame da classificare può essere prelevato direttamente dalla fotocamera.

In conclusione ritengo che questo sia solo un punto di partenza per molte altre idee alle quali questo progetto può legarsi.