**2. Generazione dataset sintetico**

La generazione del dataset sintetico avviene tramite un software, reperito sul web[[1]](#footnote-1) e adattato alle specifiche del progetto, chiamato *SyntheticDataGenerator*, operante sul motore grafico Unity 3D (versione 2019.2.0f1) sotto il controllo di scripts C#.

Il compito di questo software è quello di generare immagini contenenti l’oggetto (o gli oggetti) della detection, effettuandone il labelling in un formato leggibile dallo strumento che si prenderà carico del training del detector, con la possibilità di scegliere se evidenziare visivamente i limiti spaziali di tali oggetti tramite bounding boxes o meno.   
Nel caso rappresentato da questo lavoro di tesi, l’oggetto della detection è una mano e il software che si occupa del training è TensorFlow (versione 1.15), che allenerà una sua API per l’object detection.

Il connubio tra immagini e labels delle stesse è ciò che andrà a comporre concretamente il dataset sintetico generato con il software in questione.

Immagine che contiene uomo, tavolo, donna, cibo

Descrizione generata automaticamente



Figura 2.1 – esempio di immagine generabile con SyntheticDataGenerator e relative informazioni di labelling. L’oggetto della detection, in questo esempio, è una mano.

**2.1 La cartella *Assets***

Immagine che contiene mappa

Descrizione generata automaticamentePer descrivere i file che compongono il progetto, verrà effettuato un focus sulla cartella *Assets*[[2]](#footnote-2), contenuta nella folder di progetto *SyntheticDataGenerator*, in particolare sulle sue subdirectories, ospitanti tutto ciò che serve per rendere il progetto funzionante.

Come anticipato nei paragrafi introduttivi, ogni elemento contenuto nella cartella *Assets* implica la presenza di un file omonimo con estensione *.meta*, del quale non verrà analizzato il contenuto.

Figura 2.2

**2.1.1 *Assets/3DModels***

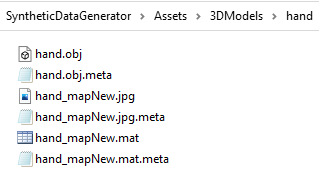
La cartella *3DModels*, come intuibile dal nome, è la cartella che ospita i modelli 3D che potranno essere utilizzati per la generazione del dataset tramite la loro importazione nella scena. Per la generazione di un dataset adatto al training dell’hand detector, è stato utilizzato un modello 3D di mano, reperito sul web[[3]](#footnote-3), in formato *.obj* e contenuto nella cartella *hand*.  
Il file *hand.obj* all’interno dell’omonima directory, contenente la riproduzione mediante una maglia poligonale (polygon mesh) di una mano, va integrato con il file *hand\_mapNew.mat*, adattamento allo spazio 3D di hand*\_mapNew.jpg*, che fornisce copertura alla mesh, aggiungendole la pelle e dettagli, altrimenti mancanti, come unghie e rughe.

Figura 2.3

Immagine che contiene mano, pezzo, tavolo, asse

Descrizione generata automaticamente

Figura 2.4 – hand.obj + hand\_mapNew.mat

Immagine che contiene disegnando, computer, cibo

Descrizione generata automaticamenteImmagine che contiene interni, sedendo, computer, tastiera

Descrizione generata automaticamenteQualsiasi modello 3D inserito all’interno della scena Unity presenta inizialmente bounds statici, indipendenti dalla prospettiva dalla quale l’oggetto viene osservato: questi limiti spaziali verranno ricalcolati in base alla posizione della telecamera per ogni oggetto presente nella scena ad ogni spostamento dello stesso - spostamento che coinciderà con la generazione di una nuova immagine, di modo da garantire un’informazione precisa sulla porzione di immagine nella quale l’oggetto si trova, predisponendo un dataset accuratamente labellato.  
Da ora in poi, quando si parlerà di limiti spaziali o di bounds di un oggetto, si farà riferimento a quelli calcolati in funzione della posizione della telecamera.

Figura 2.6 – bounds oggetto ricalcolati in funzione della posizione della telecamera

Figura 2.5 – bounds iniziali oggetto

**2.1.2 *Assets/GUI***

Immagine che contiene screenshot

Descrizione generata automaticamenteAll’interno della cartella *GUI* (abbreviazione di Graphical User Interface) è presente un unico file contenente tutte le direttive necessarie alla creazione di uno spazio, all’interno dell’interfaccia che mostrerà in real-time le immagini generate, pronto ad accogliere bounding boxes, riquadri colorati che cingeranno tra i loro lati gli oggetti della detection.  
La presenza delle bounding boxes è opzionale e sarà l’utente finale a consentirla o meno: durante il lavoro di tesi, ad esempio, sono state utilizzate bounding boxes per verificare se i bounds della mano venissero calcolati correttamente o meno, ma si è deciso di escluderne la presenza nel dataset finale.

Figura 2.7

Il file *CustomSkin.guiskin* è strettamente dipendente dal contenuto di *Assets/Images*.

**2.1.3 *Assets/Images***

Immagine che contiene screenshot

Descrizione generata automaticamenteLa cartella analizzata in questo paragrafo contiene tutto il materiale necessario per la rappresentazione di bounding boxes: queste saranno infatti formate dal connubio tra i files *Box.png* (riquadro verde)e *GlowOutline.png* (illuminazione bordi del riquadro) e saranno ridimensionate in funzione dei limiti spaziali dell’oggetto a cui fanno riferimento, andando a fornire un’informazione visiva sulla porzione di immagine all’interno della quale è presente ciò che ci interessa riconoscere.  
Le bounding boxes – nel caso l’utente lo desiderasse - appariranno sull’interfaccia che riporta, durante il ciclo di esecuzione, le immagini generate (predisposta ad accoglierle dal contenuto di *Assets/GUI*); la loro presenza, tuttavia, si estenderà anche alle immagini contenute nel dataset in quanto le fotografie contenute in quest’ultimo non sono altro che una cattura dell’interfaccia sopracitata.

Figura 2.8

Immagine che contiene edificio, verde, sedendo, facciata

Descrizione generata automaticamente

Figura 2.9 – esempio di immagine generata con aggiunta di bounding box

**2.1.4 *Assets/Materials***

Immagine che contiene screenshot

Descrizione generata automaticamenteQuesta subdirectory contiene il file *backgroundMat.mat*, materiale che ricoprirà il piano frontale alla telecamera operante da sfondo per il dataset.

Figura 2.10

L’immagine che il file “spalma” sul piano di background, adattandola allo spazio 3D mediante file sopracitato, cambierà ad ogni iterazione del processo di generazione e sarà pescata dalla cartella *Assets/Resources/Textures*.

**2.1.5 *Assets/Resources***

Immagine che contiene screenshot

Descrizione generata automaticamenteLa cartella *Assets/Resources* contiene la directory *Textures*, all’interno della quale sono poste tutte le fotografie che fungeranno da background per le immagini del dataset che verrà generato.

Figura 2.11

Immagine che contiene interni, vivendo, stanza, soffitto

Descrizione generata automaticamenteLe 1020 immagini presenti in *Textures* sono state scaricate mediante la già citata estensione Chrome chiamata Fatkun Batch Downloader e raffigurano, perlopiù, l’interno di abitazioni.  
La scelta di utilizzare immagini con questo specifico soggetto è dettata dal fatto che l’efficacia del detector allenato con questo dataset verrà verificata mediante le immagini provenienti da una webcam posta all’interno di un’abitazione: si è quindi cercato di rendere il dataset il più rappresentativo possibile della situazione reale all’interno della quale si sarebbe effettuata, almeno durante la fase di testing, la detection.

Figura 2.12 – una delle immagini di background utilizzate durante la generazione

**2.1.6 *Assets/StreamingAssets***

Immagine che contiene screenshot

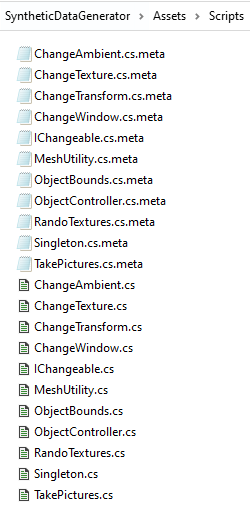
Descrizione generata automaticamenteQuesta subdirectory, inizialmente vuota, una volta terminato il ciclo di generazione immagini andrà a contenere la cartella *UnityStuff*, che ospiterà il dataset sintetico e quanto necessario per trainare l’object detection API di TensorFlow con esso.

Figura 2.13

Immagine che contiene screenshot

Descrizione generata automaticamenteAll’interno di questa cartella compariranno i file *train.txt* e *labelmap.pbtxt*, contenenti rispettivamente le informazioni di labelling del dataset e le direttive per aiutare TensorFlow a distinguere tra loro gli oggetti della detection.  
Saranno inoltre presenti le cartelle *train*, contenente le immagini generate, e *TFUtils* – copia della cartella omonima contenuta in *SyntheticDataGenerator* – il cui contenuto sarà approfondito nella parte di relazione relativa al training, essendo strettamente correlato a suddetta fase.

Figura 2.14

**2.1.7 *Assets/Scripts***

La cartella analizzata in questo paragrafo contiene tutto ciò che regola e automatizza il processo di generazione del dataset. Per capire perfettamente come questo avviene, verrà fatta una breve descrizione di ogni script C# presente in questa cartella.  
Ogni script rappresenta un’interfaccia oppure una classe, astratta o meno.

Figura 2.15

Interfacce:

* *IChangeable.cs*:interfaccia implementata da tutte le classi il cui nome contiene la parola “*Change*”, impone a queste ultime di realizzare una funzione senza ritorni chiamata *ChangeRandom()*;

Classi astratte:

* *Singleton.cs*:  
  classe che implementa il pattern Singleton[[4]](#footnote-4), garantendo a tutte le classi che la ereditano un’unica istanza e un singolo punto d’accesso ad essa;

Classi (eccetto *RandoTextures.cs* e *ObjectController.cs*, sono tutte classe figlie di *MonoBehaviour[[5]](#footnote-5)*):

* *ChangeAmbient.cs*:cambia la luce ambientale (non proveniente da una fonte specifica, ma diffusa su tutto lo spazio 3D) della scena Unity, assegnandole una tonalità randomica di bianco, tramite la funzione *ChangeRandom()* e riportandola alla tonalità predefinita a fine ciclo di generazione;
* *RandoTextures.cs*:  
  classe utility, quando chiamata in causa mescola le immagini presenti in *Assets/Resources/Textures* e ne passa una all’oggetto invocante ogni volta che viene invocata *GetRandomTexture()*, seguendo l’ordine in cui le immagini si trovano dopo lo shuffling;
* *ChangeTexture.cs*:  
  cambia la fotografia di background per l’immagine che verrà generata tramite l’implementazione di *ChangeRandom()*, ottenendola sfruttando l’accesso singleton di *RandoTextures.cs*;
* *ChangeTransform*.cs:  
  cambia posizione e rotazione dell’oggetto al quale è assegnata tramite l’implementazione di *ChangeRandom()*.  
  Le coordinate spaziali sono fatte variare randomicamente all’interno di un range di valori che assicurino la visibilità dell’oggetto alla telecamera e quindi la sua presenza all’interno dell’immagine che si andrà a generare.  
  Per quanto riguarda la rotazione, sebbene idealmente ci si vorrebbe assicurare una detection a 360 gradi dell’oggetto, si è scelto di imporre una rotazione massima di +/-20 gradi in quanto, con valori più ampi, il dataset risultava troppo dispersivo, visto l’elevato numero di posizioni assumibili dall’oggetto, spesso molto diverse tra loro;
* *ChangeWindow*.*cs*:  
  questa classe mette a disposizione un’implementazione di *ChangeRandom()* che ridimensiona l’interfaccia utente di Unity, mantenendo le proporzioni tra la sua altezza e la sua larghezza e ridimensionando di pari passo l’immagine che andrà a finire nel dataset, essendo essa una cattura della schermata in questione, contenente quanto visto dalla Main Camera con l’eventuale aggiunta di bounding boxes;
* *MeshUtility.cs*:  
  classe utility la cui unica funzione, esigente come parametro un oggetto presente nella scena Unity, ritorna i riferimenti alle mesh degli oggetti figli di quello passato come parametro. Non dovrà essere assegnata a nessun oggetto della scena proprio a causa di questa capacità di lavorare sul parametro passatole;
* *ObjectBounds.cs*:  
  classe che, prelevando le mesh collegate all’oggetto a cui è assegnata sfruttando *MeshUtility.cs*, ridefinisce i limiti inizialmente statici dell’oggetto facendoli coincidere con i vertici della mesh visibili aventi x e y minime e massime: si configureranno, dunque, quattro coordinate rappresentanti bounds non più relativi all’oggetto, bensì alla sola parte visibile di esso.  
  Nel caso l’utente lo richiedesse tramite il passaggio di un parametro booleano, i bounds saranno evidenziati da una bounding box, che sarà presente anche nell’immagine del dataset oltre che nell’interfaccia utente.
* *ObjectController.cs*:  
  classe derivante da Singleton, funge da centro di controllo delle sovrapposizioni tra oggetti, andando a disattivare gli oggetti le cui bounding boxes si sovrappongono a quelle di altri oggetti per una percentuale maggiore di quella definita dall’utente.  
  La presenza di questo script all’interno del progetto è pensata per rendere il generatore più scalabile, aprendo alla generazione di immagini con disruptors o semplicemente con più oggetti da identificare;
* *TakePictures.cs*:  
  script principe del progetto, si occupa innanzitutto di predisporre *Assets/StreamingAssets/UnityStuff* a contenere il risultato della generazione creando i file *train.txt* e *labelmap.txt* e la cartella *train*, copiando nel contempo la cartella *TFUtils* contenuta in *SyntheticDataGenerator*.  
  Successivamente, viene compilato – in un formato adatto ad essere interpretato da TensorFlow - *labelmap.pbtxt* con tutte le informazioni relative agli oggetti della detection presenti nella scena – anche quelli non visibili: ad ognuno di loro viene assegnato un numero identificativo e un nome.   
  Immagine che contiene screenshot, tenendo, uomo, mano

  Descrizione generata automaticamente  
    
    
    
    
    
    
    
    
    
    
    
    
    
  Una volta fatto tutto ciò, si procede con la generazione di un numero di immagini deciso dall’utente, con l’opzione di far apparire le bounding boxes nel prodotto finito.  
  Prima di “scattare” la fotografia, viene chiamata la funzione *ChangeRandom()* di tutti gli scripts della scena che implementano *IChangeable*, di modo da randomizzare completamente quello che sarà il contenuto dell’immagine.  
  Dopo questo procedimento, viene proiettato sull’interfaccia utente quanto visto dalla Main Camera di Unity, pronto ad essere integrato con eventuali bounding boxes.  
  L’ultimo passo consiste nell’effettuare una cattura di schermata limitata allo spazio dello schermo contenente l’interfaccia utente (bordi esclusi), per andare a ottenere l’immagine *.jpg* che farà parte del dataset.  
  Per ogni immagine generata, infine, viene compilata una riga del file *train.txt* contenente informazioni relative all’immagine e alla posizione degli oggetti visibili in essa[[6]](#footnote-6), fondamentale per TensorFlow per andare ad allenare il suo modello di detection.  
  Immagine che contiene testo

  Descrizione generata automaticamente

Figura 2.16 – compilazione labelmap.pbtxt del caso applicativo

Figura 2.17 – esempio di compilazione train.txt

**2.1.8 *Assets/Scenes***

*Immagine che contiene screenshot

Descrizione generata automaticamente*Immagine che contiene screenshot

Descrizione generata automaticamenteLa cartella analizzata in questo paragrafo contiene quanto necessario per concretizzare il processo di generazione immagini: il file *Create.unity*, infatti, contiene una scena Unity appositamente predisposta per il fine sopracitato.  
La scena in questione si compone di due oggetti principali:

Figura 2.18

Figura 2.19 – oggetti della scena Unity utilizzati durante il lavoro di tesi

* *Main Camera*: telecamera o, dato il suo impiego, fotocamera di scena; quanto catturato da essa sarà la base delle immagini del dataset.  
  Presenta un oggetto figlio, *Canvas*, piano frontale alla telecamera, che presenta a sua volta un erede, *Image*, che andrà a contenere l’immagine di sfondo.  
  A *Image* sono assegnati gli scripts *RandoTextures.cs*, che fungerà da utility, e *ChangeTexture.cs*, che si occuperà di cambiare il background, contenuto in *backgroundMat*;   
    
  Immagine che contiene screenshot

  Descrizione generata automaticamente

Figura 2.20 – scripts e materiali assegnati a Image

* *ObjectsToTrain*:  
  oggetto astratto, ad esso sono assegnati gli scripts *ObjectController.cs*, *ChangeWindow.cs*, *ChangeAmbient.cs* e *TakePictures.cs*.  
  Qualsiasi oggetto che si vuole far apparire nel dataset dovrà essere aggiunto alla scena come oggetto figlio di *ObjectsToTrain*, di modo da ereditare gli scripts sopracitati.   
  Tuttavia, per il corretto funzionamento del software, è necessario assegnare ad ogni oggetto figlio gli scripts *ObjectBounds.cs* e *ChangeTransform.cs*, di modo da poter spostare ogni oggetto e calcolarne i bounds indipendentemente dagli altri oggetti e ottenere il risultato cercato.

Immagine che contiene screenshot

Descrizione generata automaticamenteImmagine che contiene screenshot

Descrizione generata automaticamente

Figura 2.20 – scripts assegnati a ObjectsToTrain e, nell’ambito del lavoro di tesi, a hand

Una volta aggiunti gli oggetti di cui si vuole allenare la detection a *ObjectsToTrain*, sarà sufficiente premere il tasto play dell’editor Unity per generare il numero di immagini specificato in *TakePictures.cs*, script che viene fatto partire alla pressione del tasto play.  
Nel lavoro di tesi l’unico oggetto che appare sotto *ObjectsToTrain* è l’oggetto *hand*, la cui rappresentazione grafica discende dal file *hand.obj* contenuto in *Assest/3DModels*.

Immagine che contiene monitor, interni, televisione, video

Descrizione generata automaticamente

Figura 2.21 – interfaccia utente, contenente un’immagine in via di salvataggio con sovrapposta la relativa bounding box. In alto si può osservare il tasto play, premuto per far partire la generazione

1. https://github.com/MatthewHallberg/SyntheticDataGenerator [↑](#footnote-ref-1)
2. Verranno ignorate le altre cartelle visibili all’interno della directory di progetto *SyntheticDataGenerator*, in quanto contenenti file comuni a qualsiasi soluzione Unity, fatta eccezione per *TFUtils,* che verrà però descritta nei paragrafi inerenti alla fase di training [↑](#footnote-ref-2)
3. https://free3d.com/3d-model/freerealsichand-85561.html [↑](#footnote-ref-3)
4. Pattern che fa parte della “Gang of Four”, descritta nel libro “Design patterns” che fornisce patterns utilizzabili nell’OOP [↑](#footnote-ref-4)
5. Classe dalla quale devono derivare gli scripts assegnati a un oggetto della scena Unity e operanti direttamente su di esso [↑](#footnote-ref-5)
6. *Numero immagine*, *larghezza immagine*, *altezza immagine*, *oggetto1*, *xmin\_oggetto1*, *xmax\_oggetto1*, *ymin­\_oggetto1*, *ymax\_oggetto1*, oggetto2, …, *ymax\_oggettoN* [↑](#footnote-ref-6)