**INTRODUZIONE GENERICA**

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 script 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. 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 descrizioni delle stesse è ciò che andrà a comporre concretamente il dataset sintetico generato con il software in questione.

**STRUTTURA PROGETTO**

Per descrivere i file che compongono il progetto, verrà effettuato un focus sulla cartella *Assets*[[2]](#footnote-2) e, in particolare, sulle sue subdirectories, contenenti tutto ciò che serve per rendere il progetto funzionante.

Ogni elemento contenuto nella cartella *Assets* implica la presenza di un file omonimo con estensione *.meta*, creato da Unity al momento dell’importazione nel workspace dell’elemento in questione e del quale non risulta di alcun interesse, ai fini del lavoro di tesi, il contenuto.

Come emergerà nel paragrafo *Assets/Scenes*, cartella che verrà approfondita per ultima in quanto dipendente da tutte le altre sottocartelle, la presenza di un file all’interno di *Assets*  o di una delle sue subdirectories non implica il suo utilizzo durante il processo di generazione dati: vi è infatti differenza tra importare un file nel workspace Unity, inserendolo in *Assets*, e importarlo nella scena (insieme di oggetti 3D, governati da script, posti nello spazio tridimensionale) che il motore grafico sfrutterà per generare il dataset.  
È altresì importante far notare che non vi può essere importazione nella scena di un file senza che questo sia già stato importato nel workspace.

***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.

La versione di Unity utilizzata durante il lavoro di tesi supporta modelli 3D in formato non proprietario *.fbx, .dae, .3ds, .dxf* e *.obj*. Sono supportati anche formati proprietari, previa conversione in *.fbx* presa in carico da Unity stesso.

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.

***Assets/GUI***

All’interno della cartella *GUI* (abbreviazione di Graphical User Interface) è presente un unico file contenente tutte le direttive necessarie alla creazione di un’oggetto, all’interno dell’interfaccia che ospiterà l’immagine generata, pronto ad accogliere una bounding box, riquadro colorato che cingerà tra i suoi lati l’oggetto della detection.  
La presenza della bounding box è opzionale e sarà l’utente finale a decidere se impiegarla o meno: durante il lavoro di tesi, ad esempio, sono state utilizzate bounding boxes solamente per verificare se i bounds dell’oggetto venissero calcolati correttamente o meno.  
Il file *CustomSkin.guiskin* è strettamente dipendente dal contenuto di *Assets/Images*.

***Assets/Images***

La cartella analizzata in questo paragrafo contiene tutto il materiale necessario per la rappresentazione di una bounding box: quest’ultima sarà infatti formata dal connubio tra i file *Box.png* (riquadro verde)e *GlowOutline.png* (illuminazione bordi del riquadro) e sarà ridimensionata in funzione dei limiti spaziali rispetto alla telecamera[[4]](#footnote-4) dell’oggetto (o degli oggetti) su cui si basa il dataset, andando a fornire un’informazione visiva sulla porzione di immagine all’interno della quale è presente ciò che ci interessa riconoscere.  
Queste bounding boxes – se l’utente lo desidererà - appariranno sull’interfaccia che riporta, durante il ciclo di esecuzione, le immagini; 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 di schermata dell’interfaccia sopracitata.

***Assets/Materials***

Questa subdirectory contiene il file *backgroundMat.mat*, ossia il materiale che ricoprirà il piano frontale alla telecamera, operante da sfondo per ogni immagine del dataset.

L’immagine che il file “spalma” sul piano di background cambierà ad ogni iterazione del processo di generazione e sarà pescata dalla cartella *Assets/Resources/Textures*.

***Assets/Resources***

La 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.

Le immagini presenti in *Textures* sono state scaricate mediante l’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.

***Assets/StreamingAssets***

Questa subdirectory – inizialmente vuota - una volta terminato il ciclo di generazione immagini, andrà a contenere la cartella UnityStuff, cartella che racchiuderà il dataset sintetico e quanto necessario per trainare l’object detection API di TensorFlow con esso.

All’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 del dataset, e *TFUtils* – copia della cartella omonima contenuta in *SyntheticDataGenerator* – il cui contenuto sarà approfondito nella parte di relazione relativa al training, essendo esso strettamente correlato a suddetta fase.

***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.

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[[5]](#footnote-5), garantendo a tutte le classi che la ereditano un’unica istanza e un singolo punto d’accesso ad essa tramite la proprietà *Instance* e il metodovirtuale *Awake()*;

Classi (eccetto *RandoTextures.cs* e *ObjectController.cs*, sono tutte classe figlie di *MonoBehaviour[[6]](#footnote-6)*):

* *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, ridefinisce *Awake()* della classe *Singleton* da cui deriva, aggiungendo un mescolamento delle immagini presenti in *Assets/Resources/Textures* al momento della prima chiamata, tramite la funzione *Shuffle()*.  
  Tali immagini vengono poi passate, una per ogni invocazione, tramite il metodo *GetRandomTexture(),* nell’ordine in cui si trovano dopo essere state mescolate;
* *ChangeTexture.cs*:  
  cambia la fotografia che farà da sfondo all’immagine generata, ricevendola sfruttando l’accesso singleton di *RandoTextures.cs* tramite l’implementazione di *ChangeRandom()*;
* *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 +/- 180 gradi, il dataset risultava troppo dispersivo, dato il numero molto più ampio di posizioni assumibili dall’oggetto, spesso molto diverse tra loro.  
  Traslando questo discorso all’hand detection, quanto scritto implica che il detector riconoscerà solo mani con il palmo rivolto verso la telecamera;
* *ChangeWindow*.*cs*:  
  questa classe mette a disposizione un’implementazione di *ChangeRandom()* che, ricevendo l’oggetto relativo alla finestra di esecuzione di Unity dalla funzione *GetMainGameView()*, la ridimensiona, ridimensionando di pari passo l’immagine che andrà a finire nel dataset, essendo essa una cattura della schermata in questione;
* *MeshUtility.cs*:  
  classe utility la cui funzione *GetMesh*, esigente come parametro un oggetto presente nella scena Unity, ritorna i parametri delle mesh relative a tutti gli oggetti figli di quello che funge da parametro;
* *ObjectBounds.cs*:  
  classe che, prelevando la mesh dell’oggetto a cui è assegnata (insieme a quelle di eventuali oggetti figli) sfruttando *MeshUtility*, ridefinisce i limiti dell’oggetto, inizialmente statici e non coincidenti coi vertici della maglia, traslandoli nel sistema di riferimento che ha come centro l’obiettivo della telecamera e facendoli coincidere con i vertici visibili con x e y minime e massime: si configureranno, dunque, 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.
* *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;
* *TakePictures.cs*:  
  script principe del progetto, sfrutta tutti le classi descritte finora per predisporre la cartella *UnityStuff* all’interno di *Assets/StreamingAssets*, 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: ad ognuno di loro viene assegnato un numero identificativo e un nome.  
  Una volta fatto tutto ciò, si procede con la generazione di un numero di immagini deciso dall’utente, con l’opzione di far apparire una bounding box nel prodotto finito.  
  Le immagini in questione nascono da una randomizzazione di tutto ciò che è stato predisposto per essere cambiato mediante gli script descritti finora, realizzata chiamando in causa tutti gli script implementanti *IChangeable*; viene poi effettuata una fotografia di quanto visto dalla Main Camera di Unity e il risultato di questo procedimento viene proiettato sull’interfaccia utente, pronto a 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 che farà parte del dataset formato *.jpg*.  
  Per ogni immagine generata, viene compilata una riga del file *train.txt* seguendo il seguente formato(dove *n* = numero oggetti della detection):  
  *numero immagine, larghezza immagine, altezza immagine, label, xmin box 1, xmax box 1, ymin box 1, ymax box 1, …, xmin box n, xmax box n, ymin box n, ymax box n*.  
  Le coordinate delle boxes, a differenza di quanto visto su Unity – dove erano tipi float – sono interi, come invece richiesto da TensorFlow.

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. Si parlerà di telecamera (o “Main Camera”) di Unity, ma è bene precisare che, in questa specifica applicazione, essa svolgerà il ruolo di fotocamera: le immagini saranno infatti una mera riproduzione dell’interfaccia utente, contenente quanto visto dall’obiettivo della Main Camera in un dato momento più eventuali bounding boxes [↑](#footnote-ref-4)
5. Pattern che fa parte della “Gang of Four”, descritta nel libro “Design patterns” che fornisce patterns utilizzabili nell’OOP [↑](#footnote-ref-5)
6. Classe dalla quale devono derivare gli script assegnati a un oggetto della scena Unity [↑](#footnote-ref-6)