REPORT PROGETTO CGAN VANGI

Sommario

[OBIETTIVO PROGETTO 1](#_Toc146970859)

[APPROCCIO SCELTO PER RISOLVERE IL PROBLEMA 2](#_Toc146970860)

[FUNZIONAMENTO CGAN 2](#_Toc146970861)

[TENSORFLOW 3](#_Toc146970862)

[AMBIENTE DI SVILUPPO 4](#_Toc146970863)

[DATA PREPROCESSING E ARCHITETTURA DEL MODELLO 4](#_Toc146970864)

[PREPROCESSING 4](#_Toc146970865)

[TRAIN, VALIDATION E TEST SET 5](#_Toc146970866)

[PREPROCESSING ACCESSORI 5](#_Toc146970867)

[DATASET 6](#_Toc146970868)

[ARCHITETTURA MODELLO 6](#_Toc146970869)

[IMMAGINE IN INPUT AL GENERATORE 6](#_Toc146970870)

[GENERATORE 7](#_Toc146970871)

[DISCRIMINATORE 9](#_Toc146970872)

[LOSS FUNCTION UTILIZZATA 10](#_Toc146970873)

[SETTING HYPERPARAMETERS 11](#_Toc146970874)

[FUNZIONE TRAIN\_STEP 12](#_Toc146970875)

[VALUTAZIONE MODELLO 13](#_Toc146970876)

[FUNZIONE TRAIN 14](#_Toc146970877)

[RISULTATI DEL TRAINING, VALIDATION E TEST SET 15](#_Toc146970878)

[RISULTATI TRAINING 15](#_Toc146970879)

[TRAINING BOTTOMS 15](#_Toc146970880)

[TRAINING SHOES 16](#_Toc146970881)

[TRAINING ACCESSORIES 16](#_Toc146970882)

[RISULTATI VALIDATION 17](#_Toc146970883)

[RISULTATI TEST SET 20](#_Toc146970884)

[SFIDE AFFRONTATE E SVILUPPI FUTURI DEL MODELLO 20](#_Toc146970885)

# OBIETTIVO PROGETTO

L’obiettivo del seguente progetto è la generazione di un outfit tramite il modello di Deep Learning GAN (Generative Adversarial Network).

L’outfit è comprensivo della parte superiore (per esempio una t-shirt), una parte inferiore (un pantalone o una gonna), delle scarpe e un accessorio.

In particolare, il progetto richiede la creazione di un outfit completo, partendo dall’immagine di una maglietta.

# APPROCCIO SCELTO PER RISOLVERE IL PROBLEMA

L’approccio scelto per risolvere il problema e sviluppare il progetto è stato quello di utilizzare una Conditional GAN che si differisce dalla GAN standard per il “condizionamento” del generatore e del discriminatore tramite un’etichetta o label passata all’interno dell’architettura della GAN.

Immagine che contiene schermata, testo, linea, diagramma

Descrizione generata automaticamente

## FUNZIONAMENTO CGAN

Il generatore in un CGAN produce dati sintetici basati sia sul rumore casuale che sulla condizione fornita, in questo caso l’etichetta. Il discriminatore, d'altra parte, cerca di distinguere tra dati reali e dati sintetici, prendendo in considerazione sia il dato reale che il dato sintetico insieme alla condizione. Questo processo di competizione tra il generatore e il discriminatore continua fino a quando il generatore è in grado di produrre dati sintetici che sono difficili da distinguere dai dati reali, tenendo conto della condizione specificata.

I CGAN consentono di controllare e guidare la generazione dei dati sintetici in modo più specifico rispetto ai GAN tradizionali, rendendoli molto potenti per molti compiti di generazione di contenuti.

Il processo seguito è stato quello di creare un generatore al quale viene passato un vettore di rumore casuale z e l’immagine di una maglietta, così da generare delle immagini false che il discriminatore, tramite il dataset reale fornito e l’etichetta dell’item dell’outfit desiderato (bottom, shoes o accessories), potesse classificare come reali o false le immagini create dal generatore.

La scelta progettuale per soddisfare le richieste del progetto è stata quella di creare 3 CGAN separate, ognuna delle quali crea, partendo dalla stessa maglietta, un’immagine di un pantalone, di una scarpa e di un accessorio.

Esiste inoltre, anche una CGAN aggiuntiva per poter dimostrare la bontà del modello sulla categoria degli accessori, ma su un’altra tipologia di accessorio.

All’interno della cartella zip del progetto avremo 5 notebook, 4 file con estensione ipynb che rappresenta ognuno una CGAN per ogni categoria (bottoms, shoes), due per gli accessori (che lavorano per creare dei guanti e una cintura), e un file per il preprocessing dei dati ma soprattutto la creazione e il salvataggio dell’embedding dell’immagine della maglietta).

Quest’ultimo file (Preprocess\_Data\_&\_Embedding) è stato creato per snellire il codice delle 4 CGAN, soprattutto per incorporare tutte le fasi di preprocessing dei set di training, validation e test dell’intero progetto, e creare l’embedding della maglietta. Questa scelta snellisce di molto il codice e rende più modulare e compatto il progetto.

La scelta di dividere il progetto in 3 modelli separati facilita la parallelizzazione delle 3 CGAN, qualora si volesse eseguire più CGAN contemporaneamente, e la scalabilità del codice.

## TENSORFLOW

Ricordiamo che per il seguente progetto è stato utilizzato il package Tensorflow.

TensorFlow è un framework di deep learning open-source sviluppato da Google. È uno dei framework più popolari e ampiamente utilizzati per la creazione, l'addestramento e l'implementazione di reti neurali artificiali, comprese le Generative Adversarial Networks (GAN). TensorFlow offre una serie di strumenti e librerie per il machine learning e il deep learning.

Nell'ambito delle GAN, TensorFlow svolge diversi ruoli cruciali:

* Creazione di GAN: TensorFlow fornisce un'ampia gamma di funzionalità e livelli che consentono agli sviluppatori di definire e creare facilmente modelli GAN. È possibile utilizzare TensorFlow per definire il generatore e il discriminatore, impostare le funzioni di perdita e gestire il processo di addestramento delle GAN.
* Addestramento delle GAN: Fornisce ottimizzatori e metriche per gestire l'addestramento e monitorare le prestazioni della rete.
* Librerie aggiuntive: Insieme al framework principale, TensorFlow ha molte librerie aggiuntive che semplificano il lavoro con GAN e reti neurali in generale. Ad esempio**, TensorFlow-Keras** offre un'API ad alto livello per la definizione dei modelli, semplificando la creazione di GAN.
* TensorBoard: TensorFlow include TensorBoard, uno strumento di visualizzazione e monitoraggio che consente di esplorare visivamente il processo di addestramento delle GAN, inclusi grafici di perdita, immagini generate e altro ancora.

## AMBIENTE DI SVILUPPO

Come “ambiente di sviluppo” si è utilizzato inizialmente Google Colab dato che si è lavorato in precedenza su questo ambiente nelle lezioni di pratica del corso, ma successivamente si è scelto di utilizzare Pycharm perché elaborava in maniera leggermente più veloce i training-step e non causava bruschi interruzioni durante il training.

# DATA PREPROCESSING E ARCHITETTURA DEL MODELLO

## PREPROCESSING

Definito il modello di GAN utilizzato, si è passati al preprocessing dei dati.

Come dataset ci è stato fornito polyvore\_outifts, un dataset che viene comunemente utilizzato per progetti di Intelligenza Artificiale e Deep Learning.

Il dataset è formato da un file JSON per ogni set (train.json, valid.json e test.json) nel quale era presente una lista di oggetti JSON.

La struttura dati dei file è la seguente:

* La chiave "items" contiene una lista di oggetti JSON che rappresentano elementi con le seguenti chiavi:
  + "item\_id": un identificatore unico per l'oggetto, in questo caso un elemento dell’outfit.
  + "index": un valore numerico che rappresenta l’ordine dell'oggetto all'interno della lista "items".
  + La chiave "set\_id" contiene un identificatore unico per il set di oggetti rappresentato dall'oggetto JSON principale, che rappresentava l’ID dell’outfit completo dei vari elementi.

Due ulteriori file molto importanti per il preprocessing dei dati sono il file item\_metadata.json che comprendeva la coppia ID e semantic category di ogni capo d’abbigliamento, e una cartella con tutte le immagini di capi d’abbigliamento.

Il nome delle immagini JPG della cartella image combaciava con l’ID all’interno dei file JSON così da permetterci di avere una corrispondenza tra dataset JSON e immagine corrispondente.

Si è quindi lavorato sul migliorare la struttura del file JSON per renderla adatta ad essere utilizzata nella nostra CGAN.

Sono stati effettuati i seguenti passaggi:

1. Recuperare dal file polyvore\_item\_metadata.json la semantic category di ogni item e aggiungere questa informazione al file train.json.
2. Utilizzare solo gli outfit che hanno al loro interno almeno 4 items;
3. Eliminare gli outfit con all’interno due capi della stessa categoria.

## TRAIN, VALIDATION E TEST SET

Esistono due gruppi formati da train, validation e test set. Il primo gruppo riguarda i set che sono utilizzati per creare i vari dataset per le categorie semantiche Bottoms e Shoes; l’altro gruppo invece viene utilizzato per creare i dataset della categoria Accessories.

La differenza tra i due gruppi è che il gruppo dei set degli accessori hanno al loro interno un attributo fondamentale per lo svolgimento del progetto, cioè category\_id, la sottocategoria degli accessori. Vengono effettuati ovviamente ulteriori operazioni su questo gruppo, differenti rispetto a quelli sui set di Bottoms e Shoes e quindi si è deciso di creare due gruppi di Train, Validation e Test set.

### PREPROCESSING ACCESSORI

Per quanto riguarda le CGAN degli accessori sono stati effettuati ulteriori lavori sul dataset dato che con la seguente impostazione non si ottenevano dei risultati soddisfacenti al momento della generazione del capo.

Per risolvere questa problematica si è recuperato dal file item\_metadata.json precedentemente citato un ulteriore informazione molto importante: category\_id.

Questa informazione, che è presente per ogni categoria semantica, e permette di discriminare in maniera più precisa e dettagliata l’enorme campo degli accessori.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteInfatti, gli accessori, come anche la categoria bottoms o shoes, hanno delle sottocategorie (nel caso degli accessori alcune categorie sono: umbrella, gloves, belt, ecc..) che ci permettono di lavorare in maniera più precisa sulla generazione dell’immagine.

Si è recuperata quindi questa informazione, rappresentata da un numero intero, e successivamente trasformata in stringa tramite la funzione “replace\_category” che sostituiva il valore intero con il secondo valore della riga corrispondente del file categories.csv.

*Sulla sinistra un estratto della struttura del file categories.csv.*

Il secondo valore di ogni riga del file categories rappresenta sotto forma di stringa la sottocategoria di ogni categoria semantica.

Tutte questo lavoro di preprocessing dei data è racchiuso all’interno del file “Preprocess\_Data\_&\_Embedding”.

### DATASET

Risolta la problematica riguardante gli accessori si sono creati 3 dataset separati per ogni semantic category, quindi 1 dataset di soli pantaloni, 1 di sole scarpe e 1 di soli accessori. Questa scelta è stata ovviamente sviluppata anche sul Validation e Test set.

È stata una scelta progettuale molto importante sia per diminuire il tempo di training evitando di scorrere tutto l’intero set di informazioni formato da migliaia di oggetti ad ogni training di ogni CGAN, e anche per snellire il codice sui 4 notebooks.

I dataset sono formati dalla coppia (immagine, etichetta). Ogni immagine è stata recuperata tramite la funzione “load\_and\_process\_image” che sfrutta la correlazione nome\_immagine della cartella images e ID\_item all’interno del file JSON che abbiamo ottenuto tramite il preprocessing.

Le etichette in questo caso sono state poi convertite in interi per poter facilmente combinare, per tipologia di attributo, le immagini e le etichette.

Il dataset utilizzato è un dataset Tensorflow che ha una forma per quanto riguarda le immagini di (300, 300, 3) [300 sono i pixels, e 3 i canali] e per le etichette la forma di un Tensore unidimensionale.

Questo lavoro di creazione del dataset viene allargata anche ai file JSON di validazione e test set.

Dopo aver creato correttamente il dataset di training le immagini e le etichette vengono normalizzate con dimensione 128x128x3 per le immagini e impostato il batch\_size=128 per le immagini e le etichette.

È stata impostata abbastanza alta la batch size per poter velocizzare l’esecuzione in parallelo di ogni immagine processata con le immagini del dataset, ma questo dato verrà poi preso in considerazione per il fine-tuning degli hyperparameters.

## ARCHITETTURA MODELLO

### IMMAGINE IN INPUT AL GENERATORE

La terza scelta progettuale importante che è stata presa durante il progetto è stata l’implementazione all’interno dell’architettura della GAN, in particolare all’interno del generatore, dell’immagine della maglietta.

Il generatore condizionale accetta come input un vettore di rumore casuale e un’etichetta, quindi non accetta immagini o strutture simili.

Per risolvere questa problematica si è scelti di utilizzare una rete convoluzionale, in questo caso una RESNet, per trasformare l’immagine della maglietta in un embedding con forma (1,128) che potesse rappresentare le caratteristiche della maglietta e poter utilizzare l’embedding come etichetta da poter passare al generatore e al discriminatore.

Una volta creato l’embedding della maglietta si salva l’embedding in un file numpy per poter riutilizzare sempre e comunque lo stesso embedding nelle diverse CGAN.

### GENERATORE

Successivamente si è creato il generatore condizionale che prende come input un vettore di rumore casuale di dimensione latente 100 e l’embedding della maglietta di dimensione (1,128) e genera un’immagine disturbata dal rumore.

Nell’architettura del generatore condizionale l’embedding e il vettore devono essere concatenate; quindi, i due tensori devono avere la stessa forma, soprattutto la prima dimensione, cioè la dimensione del batch size.

Per fare ciò abbiamo creato le funzioni label\_conditional\_generator() e latent\_input()

definendo i nodi 128\*4\*4 per l’embedding e 512\*4\*4 per il vettore di rumore, successivamente per l’embedding è stato utilizzato un layer Dense e Reshape per ottenere il tensore della forma (4\*4\*128) mentre per quanto riguarda il vettore di rumore oltre ai layer Dense e Reshape è stato utilizzato il layer ReLU che contribuirà a introdurre non linearità nei dati di input, aiutando così il generatore a catturare pattern complessi nel vettore latente o nella condizione aggiuntiva fornita come input.

Normalizzate le due componenti verranno concatenate per creare un unico tensore della forma (4,4, 128+512) che passerà attraverso i vari layer che formano il generatore per poter ottenere come output un tensore della forma (128,128,3) che quindi sarà la forma della nostra immagine fake che il generatore ha creato tramite l’embedding e il vettore di rumore.

Ricordiamo che il generatore non è altro che una Neural Network.

Abbiamo utilizzato, come descritto precedentemente, la libreria Keras di Tensorflow che ci offre la possibilità di generare in maniera semplificata il modello del generatore e del discriminatore.

Il sommario dei filtri utilizzati nel generatore è il seguente:

Input\_3 è il nostro vettore di rumore e input\_embedding è ovviamente l’embedding della maglietta.

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, menu

Descrizione generata automaticamente

Immagine che contiene schermata, testo, Rettangolo, modello

Descrizione generata automaticamenteSe quindi volessimo generare una prima immagine con rumore generata dal generatore condizionale con input il vettore di rumore e l’embedding della maglietta il risultato sarebbe il seguente:

### DISCRIMINATORE

Per quanto riguarda il discriminatore condizionale avremo come input un tensore della forma (128,128,3) che rappresenta a seconda delle circostanze l’immagine generata dal generatore o l’immagine reale del dataset di training; e l’etichetta che a seconda dei casi (che verranno esplicitati nella funzione di train\_step) rappresenterà l’etichetta del dataset che contiene la categoria desiderata come output (bottoms, shoes o accessories) quindi l’etichetta reale o l’embedding.

L’obiettivo del discriminatore sarà quello di restituire come output un valore intero (1 o 0) che ci permette di classificare l’immagine generata dal generatore come reale o falsa.

Quindi l’etichetta diventerà un embedding (None, 1, 128) tramite la funzione label\_conditional\_disc, e, tramite funzioni molto simili a quelle utilizzate per il generatore, verrà combinata con l’immagine (None, 128, 128,3) dal generatore in precedenza.

Dopo aver concatenato i due input verranno utilizzati per alimentare la rete di layers convoluzionali, di densità e reshape per ottenere alla fine un tensore di una dimensione che rappresenti la “risposta” del discriminatore, cioè un tensore con un unico valore.

In seguito, sono presenti alcuni screenshot che rappresentano l’architettura di layers del discriminatore condizionale:

Immagine che contiene testo, schermata, menu

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### LOSS FUNCTION UTILIZZATA

Successivamente abbiamo definito la binary cross function, la Loss Function più comunemente utilizzate nelle reti neurali quando si tratta di problemi di classificazione binaria. È particolarmente adatta quando si affronta problemi in cui ogni esempio di training può appartenere a una delle due classi distinte, come ad esempio "classe positiva" e "classe negativa".

La Binary Cross-Entropy viene utilizzata per misurare la discrepanza tra le previsioni del modello e le etichette reali (ground truth) dei dati di training.

La generator loss è calcolata con l’etichetta uguale a 1 perché si vuole che il generatore inganni il discriminatore e generi immagini molto vicine a quelle reali. La Loss del generatore quantifica quanto è stato in grado di ingannare il discriminatore. Intuitivamente, se il generatore funziona bene, il discriminatore classificherà le immagini false come reali (o 1).

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

La discriminator loss invece verifica tramite la cross-entropy tra le etichette e gli output finali. Questo metodo quantifica quanto bene il discriminatore è in grado di distinguere le immagini reali da quelle false.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

### SETTING HYPERPARAMETERS

Dopodiché abbiamo settato degli iperparametri di un modello di CGAN simile a quello del progetto in questione e che sono stati presi come iperparametri (di partenza) di training per poi successivamente modificarli (per la fase di fine-tuning) in base alla qualità delle prestazioni del modello.

Gli iperparametri settati sono:

* learning\_rate = 0.0002;
* generator\_optimizer e discriminator\_optimizer: Vengono creati due oggetti ottimizzatori Adam learning rate di 0.0002 e specificando i parametri beta\_1 e beta\_2. Adam è un algoritmo di ottimizzazione comunemente utilizzato per addestrare reti neurali, ed è spesso una buona scelta per GAN. I parametri beta\_1 e beta\_2 sono coefficienti di momento e seconda media mobile che influenzano il comportamento dell'ottimizzatore.
* generator\_variables e discriminator\_variables: Queste due variabili sono utilizzate per raccogliere le variabili trainabili dei modelli del generatore e del discriminatore. Le variabili trainabili sono i pesi e i bias che vengono aggiornati durante l'addestramento del modello. Queste variabili vengono spesso raccolte per passarle agli ottimizzatori in modo che durante l'addestramento sia possibile aggiornare solo queste variabili e non tutti i parametri del modello. In questo modo, il generatore e il discriminatore possono essere addestrati separatamente durante il processo di addestramento GAN.

### FUNZIONE TRAIN\_STEP

Successivamente ho definito una funzione dal nome train\_step che sarebbe il passo di training della CGAN.

Inizialmente prende come input le immagini reali del dataset, e la loro etichetta, ma si è decisi di aggiungere anche delle variabili, inizialmente col valore 0, delle Loss del generatore e del discriminatore che potessero essere aggiornate ad ogni timestep, questa scelta è stata fatta per poter visualizzare meglio e avere sott’occhio tutti i valori delle Loss della CGAN nel momento del training.

Generiamo un tensore di rumore casuale utilizzando TensorFlow (tf.random.normal). Il rumore è campionato da una distribuzione normale (gaussiana) standard. Il risultato è un tensore con dimensioni [numero\_di\_istanze del dataset, dimensione\_latente], dove numero\_di\_istanze rappresenta il numero di campioni o esempi che si desidera generare, e dimensione\_latente è la dimensione del vettore latente che verrà utilizzato, per generare nuovi dati in una GAN.

Questo rumore verrà utilizzato come input al generatore per generare immagini fake.

Normalizzo la dimensione del batch size dell’embedding in base alla dimensione del batch, in questo caso il batch size sarà pari a 128, ma potrebbe assumere ulteriori valori nel momento in cui si lavora sul dataset.

Effettuate queste piccole operazioni ci apprestiamo a generare l’immagine tramite il conditional generator tramite il vettore di rumore e l’embedding.

Viene calcolata la perdita per il discriminatore con le etichette reali:

* real\_output = conditional\_discriminator([images, target], training=True):  
  Il discriminatore condizionale valuta l'immagine reale (images) insieme all'etichetta target fornita. Questo calcolo è eseguito durante l'addestramento del discriminatore con l'obiettivo di distinguere le immagini reali da quelle generate;
* real\_targets = tf.ones\_like(real\_output): Vengono generati i target reali, che sono impostati su 1 perché si sta addestrando il discriminatore con immagini reali.
* disc\_loss1 = discriminator\_loss(real\_targets, real\_output): Viene calcolata la perdita del discriminatore in base alla sua capacità di distinguere le immagini reali

Ora che abbiamo utilizzato il discriminatore e calcolato la sua Loss function si può utilizzare il Gradient Descent per ottimizzare il discriminatore per le etichette reali:

* gradients\_of\_disc1=disc\_tape1.gradient(disc\_loss1, conditional\_discriminator.trainable\_variables): Vengono calcolati i gradienti della perdita del discriminatore rispetto ai parametri del discriminatore.
* discriminator\_optimizer.apply\_gradients(zip(gradients\_of\_disc1, conditional\_discriminator.trainable\_variables)): Vengono applicati gli aggiornamenti dei parametri del discriminatore utilizzando l'ottimizzatore.

Il discriminatore viene poi nuovamente addestrato con immagini generate (generated\_image) e le stesse etichette target.

Viene calcolata una loss (disc\_loss2) che misura quanto bene il discriminatore riesce a distinguere le immagini generate come false.

I gradienti della loss rispetto ai parametri del discriminatore vengono calcolati (gradients\_of\_disc2).

Viene nuovamente ottimizzato il discriminatore utilizzando questi gradienti.

Ora avviene finalmente il training del generatore condizionale dove si calcola la Loss del generatore sulla base della capacità di ingannare il discriminatore. I gradienti vengono calcolati e gli aggiornamenti vengono applicati ai parametri del generatore. Il ragionamento è il medesimo delle etichette reali.

In conclusione, accumuliamo ad ogni step le losses del generatore e del discriminatore per avere un’ulteriore analisi di dati prestazionali del nostro modello.

### VALUTAZIONE MODELLO

Le prossime funzioni verranno utilizzate per definire al meglio le metriche di valutazione utilizzate per valutare le performance del modello allenato sul validation set e “modellarle” ai dati delle CGAN.

Le metriche di valutazione utilizzate sono l’Inception Score e la FID.

1. Fréchet Inception Distance (FID):

**Descrizione**: Il FID è una metrica che misura la somiglianza tra le immagini generate dalla CGAN e le immagini reali utilizzando le feature estratte da un modello di classificazione preaddestrato, spesso InceptionNet.

Il calcolo del FID coinvolge due passaggi principali:

* Estrazione delle feature: Si utilizza il modello InceptionNet per estrarre feature (spesso vettori) dalle immagini generate e dalle immagini reali.
* Calcolo delle statistiche di Fréchet: Si calcolano la media e la matrice di covarianza delle feature estratte per le due distribuzioni e quindi si calcola la distanza di Fréchet tra queste due distribuzioni.

Un valore di FID più basso indica che le immagini generate sono più simili a quelle reali. Quindi un FID più basso è un indicatore di maggiore qualità nella generazione delle immagini.

2. Inception Score:

L'Inception Score valuta la qualità e la diversità delle immagini generate da una CGAN. Si basa sull'assunzione che immagini di alta qualità dovrebbero avere una distribuzione uniforme delle classi quando classificate da un modello di classificazione preaddestrato.

Il calcolo dell'Inception Score comporta due passaggi principali:

* Classificazione delle immagini: Si utilizza il modello InceptionNet per classificare le immagini generate e si registra la distribuzione delle classi predette per ogni immagine.
* Calcolo dell'entropia e della media: Si calcola l'entropia della distribuzione delle classi e la media delle entropie per tutte le immagini.

Un Inception Score più alto indica una maggiore qualità e diversità nelle immagini generate.

Per apprendere queste metriche abbiamo importato alcuni package importanti da Tensorflow come keras e in particolare Inceptionv3, e implementato tramite le funzioni *inception\_features*, *calculate\_fid* e *calculate\_inception\_score* FID e Inception.

La funzione successiva, evaluate model, prendendo in considerazione il generatore condizionale e il validation dataset, calcola le due metriche di valutazione per capire come lavora il modello di CGAN.

La funzione itera attraverso il validation set, genera l’immagine tramite generatore e confronta FID e Inception tra l’immagine generata e l’immagine reale.

Finite le iterazioni effettuerà la media del risultato di FID e Inception.

### FUNZIONE TRAIN

Il prossimo passo è definire la funzione di train nella quale si passa il dataset di training che verrà utilizzato per allenare il modello, il numero di epoche che verrà utilizzato per iterare il dataset e il dataset del validation set.

Si itera sulla base dell’immagini e delle etichette (che sono state normalizzate) e si effettuano i vari step di training, precedentemente citati sotto forma della funzione *train\_step*.

Ogni 10 epoche per le CGAN di Bottoms e Shoes, e dopo 20 epoche per le CGAN di Accessories verrà valutato tramite FID e Inception le prestazioni del modello sul dataset di validation richiamando la funzione *evaluate\_model*.

Dopo aver completato l’epoca di training e la valutazione del modello, tramite la funzione *generate\_and\_save\_images* verrà mostrata a video l’immagine generata dal modello.

Una parte importante di questa funzione è il salvataggio dei pesi, ogni n epoche, di generatore e discriminatore. Infatti, tramite il salvataggio dei pesi della CGAN, è possibile riprendere il training del modello in un secondo momento salvando però i progressi finora ottenuti. Questo è molto utile dato che i vari training sono molto lunghi.

# ANALISI MODELLO SUL TEST SET

Come ultimo step viene richiamata la funzione evaluate\_model ma ora valutando il modello sul test set e calcolando la media di FID e Inception.

# RISULTATI DEL TRAINING, VALIDATION E TEST SET

## RISULTATI TRAINING

Come prima cosa parto col scegliere l’immagine in input del Top da passare alle varie CGAN. In questo caso l’immagine scelta è la seguente:

Immagine che contiene persona, vestiti, Viso umano, Abbigliamento informale

Descrizione generata automaticamente

### TRAINING BOTTOMS

I risultati dopo 32 epoche per il dataset “bottom” sono i seguenti:

Immagine che contiene schermata, modello, quadrato, Rettangolo

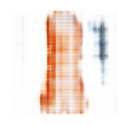
Descrizione generata automaticamenteImmagine che contiene schermata, quadrato, Rettangolo, Policromia

Descrizione generata automaticamenteImmagine che contiene schermata, Policromia, modello, sfocatura

Descrizione generata automaticamenteImmagine che contiene arte

Descrizione generata automaticamente con attendibilità bassaImmagine che contiene modello, tessuto

Descrizione generata automaticamenteImmagine che contiene modello, Tessuto scozzese, scozzese, textile

Descrizione generata automaticamenteImmagine che contiene vestiti, modello, Tessuto scozzese, Modello (moda)

Descrizione generata automaticamenteImmagine che contiene arte, textile

Descrizione generata automaticamente con attendibilità mediaImmagine che contiene vestiti, modello, Tessuto scozzese, scozzese

Descrizione generata automaticamenteImmagine che contiene vestiti, pantalone, persona

Descrizione generata automaticamente

Le immagini rappresentano la generazione del pantalone associato alla maglietta in input dopo 32 epoche (si è decisi di non andare oltre dato che l’elaborazione di questo training ha impiegato ben 8 ore per arrivare a questi risultati).

### TRAINING SHOES

In seguito, abbiamo i risultati del training del modello sul dataset “shoes”:

Immagine che contiene invertebrato

Descrizione generata automaticamenteImmagine che contiene cibo

Descrizione generata automaticamente con attendibilità mediaImmagine che contiene acqua, arte

Descrizione generata automaticamenteImmagine che contiene cibo

Descrizione generata automaticamenteImmagine che contiene schizzo, dipinto, arte

Descrizione generata automaticamenteImmagine che contiene pesce, arte

Descrizione generata automaticamente

### TRAINING ACCESSORIES

#### GLOVES

E in conclusione i risultati del train della CGAN sul dataset “accessories” per la sottocategoria “gloves”.

Immagine che contiene modello

Descrizione generata automaticamenteImmagine che contiene arte, sfocatura

Descrizione generata automaticamente con attendibilità mediaImmagine che contiene rosso, Policromia, Bordeaux, Coquelicot

Descrizione generata automaticamenteImmagine che contiene modello, Policromia, schermata, design

Descrizione generata automaticamenteImmagine che contiene Policromia, arte

Descrizione generata automaticamente

#### BELT

Alcuni risultati del training anche della CGAN riguardante la sottocategoria “belt” di accessories:

Immagine che contiene schermata, Rettangolo, modello, linea

Descrizione generata automaticamenteImmagine che contiene schermata, modello, linea, Policromia

Descrizione generata automaticamenteImmagine che contiene schermata, linea

Descrizione generata automaticamenteImmagine che contiene modello, Policromia, schermata, linea

Descrizione generata automaticamente

In sostanza abbiamo creato 3 CGAN (con i rispettivi hyperparameters, bias e pesi) che lavorano su 3 dataset diversi con un’embedding in input in comune.

## RISULTATI VALIDATION

Dopo aver visualizzato i risultati grafici dei 3 modelli ho lavorato sul validation set, un dataset che non ha nessun’immagine in comune col dataset di training e che, tramite le metriche di valutazione FID e Inception Score ci permetta di capire e valutare al meglio la bontà e la qualità delle CGAN.

Per riassumere tutti i risultati ho creato 3 tabelle, una per ogni dataset, per dimostrare le prestazioni del modello con gli attuali hyperparameters e con una serie di modifiche a quest’ultimi per il **fine-tuning**.

Gli hyperparameters iniziali sono stati già fissati in precedenza, in ogni colonna verrà modificato un parametro alla volta per capire al meglio l’influenza che ogni parametro ha sulle prestazioni del modello. Le metriche di valutazione sono già preprocessate come la media delle loro istanze.

Più il FID è basso e più l’immagine generata è più nitida, più l’Inception Score è alto e più l’immagine generata si differenzia dall’immagine reale per “creatività”.

I migliori risultati (e le metriche utilizzate poi per il test set) verranno evidenziati di giallo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATASET BOTTOMS** | **PARAM BASE** | **LEARNING\_RATE = 0.001** | **DIMENSIONE LATENTE = 128** | **BATCH SIZE = 64** |
| **EPOCA 10** | FID: 55.6,  INCEPTION: 262.7 | FID: 65.1, INCEPTION: 70.9 | FID: 55,4  INCEPTION: 278.7 | FID: 29.2, INCEPTION: 112.9 |
| **EPOCA 20** | FID: 54.8,  INCEPTION: 231.1 | FID: 48.1, INCEPTION: 174.3 | FID: 43.0, INCEPTION: 238.4 | FID: 31.7, INCEPTION SCORE: 44.9 |
| **EPOCA 30** | FID: 51.2,  INCEPTION: 187.8 | FID: 48.3, INCEPTION: 146.0 | FID: 35.7 INCEPTION: 141.6 | FID: 20.3, INCEPTION: 21.4 |
| **AVERAGE** | FID: 53.9  INCEPTION: 227.2 | FID: [53.8]  INCEPTION: [130.4] | FID: [44.3]  INCEPTION: [222.6] | FID: [27.3]  INCEPTION: [59.7] |

**Lr=0.001**

Average Generator Loss: tf.Tensor(1.375, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0.149, shape=(), dtype=float32)

**Lat\_dim = 128**

Average Generator Loss: tf.Tensor(1.580, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0.154, shape=(), dtype=float32)

**Barch\_size = 64**

Average Generator Loss: tf.Tensor(1.467, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0.196, shape=(), dtype=float32)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATASET SHOES** | **PARAM BASE** | **LEARNING\_RATE = 0.001** | **DIMENSIONE LATENTE = 128** | **BATCH SIZE = 64** |
| **EPOCA 10** | FID: 44.7,  INCEPTION: 200.3 | FID: 47.8,  INCEPTION: 863.1 | FID: 53.5,  INCEPTION: 74.5 | FID: 43.9,  INCEPTION: 213.5 |
| **EPOCA 20** | FID: 30.96,  INCEPTION: 169.1 | FID: 48.2, INCEPTION: 338.5 | FID: 59.3,  INCEPTION: 199.1 | FID: 33.5,  INCEPTION: 201.7 |
| **EPOCA 30** | FID: 25.3,  INCEPTION: 78.8 | FID: 55.6, INCEPTION: 198.2 | FID: 43.1,  INCEPTION: 233.9 | FID: 38.4, INCEPTION: 175.3 |
| **AVERAGE** | FID: [33,6]  INCEPTION:129,3 | FID: 50.5  INCEPTION: 466.6 | FID: 51.96  INCEPTION: 169.2 | FID: 38.6 INCEPTION: 196.8 |

A livello grafico, avendo una dimensione latente maggiore dovrebbe dare più dettagli grafici ma in realtà non migliora la qualità dell’immagine e rallenta solamente il training.

**Dimensione latente = 128**:

Average Generator Loss: tf.Tensor(2.01, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0,27, shape=(), dtype=float32)

**Batch\_size = 64:**

Average Generator Loss: tf.Tensor(5.6906, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0.6625, shape=(), dtype=float32)

**Learning\_rate = 0.0001**

Average Generator Loss: tf.Tensor(4.6906, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0.7321, shape=(), dtype=float32)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATASET ACCESSORIES** | **PARAM BASE** | **LEARNING\_RATE = 0.001** | **DIMENSIONE LATENTE = 128** | **BATCH SIZE = 64** |
| **EPOCA 20** | FID: 39.8, INCEPTION: 123.1 | FID: 65.1, INCEPTION: 70.9 | FID: 55.0,  INCEPTION: 278.7 | FID: 47.9,  INCEPTION: 392.9 |
| **EPOCA 40** | FID: 40.9,  INCEPTION: 180.7 | FID: 48.1,  INCEPTION: 174.3 | FID: 43.0,  INCEPTION: 238.4 | FID: 49.3,  INCEPTION: 531 |
| **EPOCA 60** | FID: 39.8,  INCEPTION: 101.4 | FID: 48.4,  INCEPTION: 146.0 | FID: 35.8,  INCEPTION: 141.6 | FID: 44.2, INCEPTION: 358.6 |
| **AVERAGE** | FID: 40.2  INCEPTION: 135.1 | FID: [48.4]  INCEPTION: [146.1] | FID: [35.8]  INCEPTION: [141.6] | FID: 47.1 INCEPTION: 427.5 |

**Lr=0.001**

Average Generator Loss: tf.Tensor(1.812, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0.136, shape=(), dtype=float32)

**Lat\_dim = 128**

Average Generator Loss: tf.Tensor(1.199, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0.174, shape=(), dtype=float32)

**Barch\_size = 64**

Average Generator Loss: tf.Tensor(1.727, shape=(), dtype=float32)

Average Discriminator Loss2: tf.Tensor(0.217, shape=(), dtype=float32)

## RISULTATI TEST SET

Sintonizzati gli hyperparameters corretti valutiamo ora il modello allenato sul test set.

La prossima tabella ci permette di visualizzare i risultati delle metriche di performance sul test set del nostro modello.

|  |  |  |
| --- | --- | --- |
| **TEST SET** | **AVG FID** | **AVD INCEPTION SCORE** |
| **BOTTOMS** | 39.57 | 146.07 |
| **SHOES** | 62.03 | 197.35 |
| **GLOVES** | 42.81 | 56.61 |
| **BELT** | 29.79 | 184.29 |

# SFIDE AFFRONTATE E SVILUPPI FUTURI DEL MODELLO

Le sfide affrontate nel corso della costruzione e realizzazione del progetto sono state molteplici. La prima in assoluto è stata la gestione del dataset polyvore\_outfits che comprendeva vari files che potevano essere tutti utili per realizzare il nostro scopo.

Un’altra sfida affrontata è stata l’inserimento all’interno del generatore dell’immagine della categoria semantica top. Come sappiamo, il generatore non accetta immagini e quindi bisognava trovare una soluzione alternativa.

Come soluzione, con l’aiuto del professor Anelli, si è utilizzata una rete convoluzionale nota, una ResNEt per esempio.

Le RESNet sono chiamate "residuali" perché introducono l'idea di "shortcut connections" o "skip connections" che consentono al modello di saltare uno o più livelli durante il processo di propagazione in avanti. Questa innovazione ha risolto il problema noto come "degradazione del training" nelle reti neurali profonde. In generale, le reti neurali profonde tendono a diventare più difficili da addestrare man mano che aumenta la loro profondità a causa del rischio di vanishing gradient (gradiente che tende a diventare molto piccolo durante la retropropagazione), ma le RESNet superano questo problema.

Nel nostro caso la RESNet è stata utilizzata per trasformare l’immagine in un embedding di forma (1,128) che rappresenta le caratteristiche dell’immagine, da passare in input al generatore.

La problematica però più complessa è stata capire e creare la funzione di training del modello, compresa la fase di test dell’architettura del modello nel generare immagini.

E’ stata una fase molto delicata perché è la funzione più importante del progetto dato che è responsabile dell’apprendimento della GAN nel generare le immagini.

Una problematica molto importante si è creata nel momento in cui la CGAN degli accessori non generava immagini soddisfacenti a tal punto da identificare realmente un’immagine come un vero e proprio accessorio.

In questo caso si è risolti andando a modificare tutta quella che è la parte di preprocessing dei dati andando a discretizzare al meglio le sottocategorie degli accessori così da ottenere delle immagini più incentrate su una particolare sezione invece che sull’intero dataset.

Un’altra problematica sopraggiunta è l’elevato numero di epoche necessarie per avere una CGAN performante e produrre un’immagine “accettabile” che potesse essere ricondotta alle categorie semantiche previste.

La soluzione a questo problema è stata suddividere il dataset per categorie per velocizzare il training e fornire dei risultati soddisfacenti.

Un possibile miglioramento o sviluppo del progetto sarebbe poter elaborare su un computer più performante il training e migliorare i risultati ottenuti aumentando il numero di epoche di training.

Uno sviluppo futuro di questo progetto sarebbe, tramite un computer più performante, utilizzare l’intero dataset con l’etichette “bottoms”, ”shoes”,”accessories” unite per poter sviluppare una CGAN generale che crei alla fine 4 immagini separate con una sola iterazione del modello. Questa era l’idea originaria del progetto ma che non è stata perseguita per la difficoltà di allenare un modello così pesante.

Per migliorare questo progetto si potrebbe pensare di rendere le immagini sempre più nitide e di qualità, ma questo richiederebbe sicuramente un ingrandimento delle dimensione delle immagini e quindi appesantirebbe di molto il modello rendendolo più lento.

Un ulteriore sviluppo futuro potrebbe essere ampliare la CGAN sugli accessori affinché possa creare un’immagine per ogni sottocategoria così da far scegliere all’utente finale quale sia l’accessorio migliore per lui.