Relazione progetto di Machine Learning

AttnGAN e DualAttnGAN per Text-to-Image Synthesis

Giulia Giusti (Matr. 943176)
Matteo Trentin (Matr. 953888)

Indice

1 Obiettivo del progetto

1.1 Text-to-Image Synthesis

Quando gli umani ascoltano o leggono una storia, disegnano immediatamente immagini mentali visualizzando il contenuto nella loro testa. La capacità di visualizzare e comprendere l'intricata relazione tra il mondo visivo e il linguaggio è così naturale che raramente ci pensiamo.
La sintesi da testo a immagine si riferisce a metodi computazionali che traducono descrizioni sotto forma di parole chiave o frasi, in immagini con significato semantico simile al testo. Inizialmente la sintesi di immagini si basava principalmente sull'analisi della correlazione tra parola e immagine combinata con metodi supervisionati per trovare il miglior allineamento del contenuto visivo corrispondente al testo. I recenti progressi nel Deep Learning (DL) hanno portato a una nuova serie di metodi di apprendimento non supervisionati, in particolare deep generative models che permettono di generare immagini visive realistiche utilizzando modelli di rete neurale adeguatamente addestrati.
Mentre il testo fornisce un modo efficiente, efficace e conciso per comunicare, il contenuto visivo, come le immagini, rappresenta un metodo più completo e accurato di condivisione e comprensione delle informazioni.
La generazione di immagini da descrizioni di testo è un problema complesso di computer vision e machine learning che ha registrato grandi progressi negli ultimi anni. La generazione automatica di immagini dal linguaggio naturale può consentire agli utenti di descrivere elementi visivi attraverso descrizioni di testo visivamente ricche.

1.2 Fasi del progetto

Lo svolgimento del progetto è stato diviso in tre fasi principali, di seguito elencate:

- Impostazione dell'ambiente di sviluppo e testing, e riproduzione dei risultati con AttnGAN
- Implementazione delle modifiche descritte nell'articolo riguardante DualAttnGAN
- Testing e confronto tra le due implementazioni

In questo modo è stato possibile osservare in maniera riproducibile gli effetti delle modifiche sui risultati del modello, confrontando le immagini ottenute sulla base di due metriche distinte, ovvero Inception Score e Fréchet Inception Distance.

1.3 Strumenti e dataset

Il dataset utilizzato per il training dei modelli è CUB [4](Caltech-UCSD Birds-200-2011), contenente 11.788 immagini divise in 200 categorie, con annotazioni associate ad ogni immagine (descrivendone le caratteristiche in forma testuale). Per quanto riguarda l'implementazione effettiva del modello, è stato utilizzato come base il codice pubblicamente disponibile a `https://github.com/taoxugit/AttnGAN` [8], scritto utilizzando PyTorch.
Il testing è stato effettuato utilizzando una GPU Nvidia RTX 2080Ti, e ha impiegato una media di 6 minuti per ogni epoch; a causa di malfunzionamenti dell'hardware nelle fasi finali del progetto, e dei tempi proibitivi del training su macchine meno potenti (circa una settimana per un training completo), è stata interrotta anticipatamente la fase di raccolta dei risultati, e sono stati utilizzate le immagini ottenute fino a quel momento (250 epoch sulle 600 totali necessarie ad AttnGAN); non è stato quindi purtroppo nemmeno possibile testare l'architettura su altri dataset, ed effettuare tuning degli iperparametri per un'analisi più approfondita del modello.

2 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) hanno avuto un enorme successo da quando sono state introdotte nel 2014 da Ian J. Goodfellow nell'articolo Generative Adversarial Nets.

2.1 Esempio iniziale

Supponiamo di essere interessati a generare immagini quadrate in bianco e nero di cani con una dimensione di n per n pixel. Possiamo rimodellare ogni dato come un vettore dimensionale N = nxn in modo tale che un'immagine del cane possa essere rappresentata da un vettore. Tuttavia, questo non significa che tutti i vettori rappresentino un cane una volta modellato in un quadrato. Quindi, possiamo dire che i vettori N dimensionali che effettivamente rappresentano qualcosa che assomiglia a un cane sono distribuiti secondo una distribuzione di probabilità molto specifica sull'intero spazio vettoriale N dimensionale. Nello stesso modo, esistono, su questo spazio vettoriale N dimensionale, distribuzioni di probabilità per immagini di gatti, uccelli e così via.
Quindi, il problema di generare una nuova immagine del cane equivale al problema di generare un nuovo vettore seguendo una determinata distribuzione di probabilità sullo spazio vettoriale N dimensionale. Inoltre, è importante notare che:

- La distribuzione dei vettori che rappresentano un cane è una distribuzione molto complessa su uno spazio vettoriale molto ampio.
- Anche se possiamo presumere l'esistenza di tale distribuzione sottostante ovviamente non sappiamo come esprimere esplicitamente questa distribuzione.

Entrambi i punti precedenti rendono il processo di generazione piuttosto difficile.
L'obiettivo dei modelli generativi è proprio quello di cercare di apprendere l'effettiva distribuzione p_{data} dei dati reali appartenenti al training set e costruire una distribuzione di probabilità p_{model} più vicina possibile a p_{data}.

2.2 Funzionamento delle GAN

L'idea di base dei GAN è creare un gioco tra due giocatori:

- **Generator**: crea campioni simili a quelli presenti nel training set. Esso viene addestrato per ingannare il discriminator. In altre parole, il generator prende come input una variabile latente e restituisce una variabile che segue una determinata distribuzione obiettivo.
- **Discriminator**: esamina i campioni per determinare se sono reali o falsi. Il discriminator apprende utilizzando le tradizionali tecniche di apprendimento supervisionato, dividendo gli input in due classi (reali o false).

Formalmente, le GAN sono un modello generativo contraddittorio in cui vengono addestrati i due modelli (o giocatori) rappresentati da generator e discriminator. Entrambi giocatori sono rappresentati da una funzione, ciascuna delle quali derivabile sia rispetto ai suoi input che rispetto ai suoi parametri. La discriminator function D prende in input x e usa $\theta^{(D)}$ come parametri. Mentre, la generator function G prende in input z e usa $\theta^{(G)}$ come parametri.
I due giocatori hanno funzioni di costo definite in termini dei parametri di entrambi i giocatori. Il discriminator ha come scopo quello di minimizzare la funzione $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ e può farlo solo controllando i parametri in $\theta^{(D)}$. Analogamente, il generator ha come scopo quello di minimizzare la funzione $J^{(G)}(\theta^{(G)}, \theta^{(D)})$ e può farlo solo controllando i parametri in $\theta^{(G)}$.
Visto che il costo di ogni giocatore dipende dai parametri dell'altro giocatore, ma ogni giocatore non può controllare i parametri dell'altro, allora questo scenario è più semplice da descrivere come un gioco piuttosto che come un problema di ottimizzazione. La soluzione ad un gioco è nota come *Nash equilibrium*, in questo contesto il Nash equilibrium è una tupla $(\theta^{(D)}, \theta^{(G)})$ che è un minimo locale della funzione $J^{(D)}$ rispetto a $\theta^{(D)}$ e un minimo locale della funzione $J^{(G)}$ rispetto a $\theta^{(G)}$.

Generator Il generatore è semplicemente una funzione derivabile G che prende in input un campione z da una semplice distribuzione a priori e restituisce un campione di x in accordo con p_{model}.
In genere, una rete neurale profonda viene utilizzata per rappresentare G, sul design di tale rete ci sono poche restrizioni. Gli unici requisiti necessari se vogliamo che p_{model} abbia pieno supporto sullo spazio x sono i seguenti: abbiamo bisogno che la dimensione di z sia grande almeno quanto la dimensione di x e che G sia derivabile.

2.3 Funzioni di costo

Tutti i diversi giochi progettati finora per le GAN utilizzano lo stesso costo per il discriminatore, $J^{(D)}$. Differiscono solo in termini di costo utilizzato per il generatore, $J^{(G)}$.

2.3.1 Cost function del discriminator

Il costo utilizzato per il discriminatore è:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\,\mathbb{E}_{x\sim p_{data}}[log\ D(x)] - \frac{1}{2}\,\mathbb{E}_{z\sim p_z}\,[log\ (1 - D(G(z)))]$$

dove:

- $D(x)$: distribuzione della probabilità che il discriminator etichetti come reale un'istanza x presa dalla reale distribuzione p_{data}.
- $G(z)$: è l'output del generator quando gli viene fornito il rumore z.

- $D(G(z))$: distribuzione della probabilità che il discriminator etichetti come reale un'istanza x presa dalla distribuzione dei dati generati dal generator p_{model}. Quindi $1 - D(G(z))$ equivale alla probabilità dell'evento complementare, quindi rappresenta la probabilità che un'istanza falsa venga riconosciuta come falsa dal discriminatore.

- $\mathbb{E}_{x \sim p_{data}}[log\ D(x)]$: cross entropy tra: distribuzione dei dati reali p_{data} e distribuzione della probabilità che D riconosca come reale un'istanza di p_{data}.

- $\mathbb{E}_{z \sim p_z}\ [log\ (1 - D(G(z)))]$: cross entropy tra: distribuzione dei dati noise p_z e distribuzione della probabilità che D riconosca come falsa un'istanza di p_{model}.

Addestrando il discriminator siamo in grado di ottenere una stima del rapporto:

$$\frac{p_{data}(x)}{p_{model}(x)}$$

per ogni punto x. La stima di questo rapporto ci consente di calcolare un'ampia varietà di divergenze e i loro gradienti.
L'approssimazione GAN è soggetta ai problemi dell'apprendimento supervisionato: overfitting e underfitting. In linea di principio, con un'ottimizzazione perfetta e dati di training sufficienti, questi problemi possono essere superati.

2.3.2 Minimax game e cost function per il generator

Come descritto precedentemente, le GAN possono essere naturalmente analizzate come modello contraddittorio e quindi mediante strumenti della teoria dei giochi, questa è la motivazione di "adversarial" all'interno del nome.
A questo punto, è necessario specificare anche la cost function per il generator. Il modo più semplice per definire la cost function per il generator è quella di considerare uno *zero-sum game*, nel quale la somma dei costi dei due i giocatori è sempre zero, in questo modo otteniamo:

$$J^{(G)} = -J^{(D)}$$

Visto che $J^{(G)}$ è ottenuto direttamente da $J^{(D)}$, possiamo riassumere l'intero gioco con una *value function* del seguente tipo:

$$V(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$$

Gli zero-sum game sono anche chiamati *minimax game* perché la loro soluzione prevede la minimizzazione in un ciclo esterno e la massimizzazione in un ciclo interno:

$$arg\ min_{\theta^{(G)}}\ max_{\theta^{(D)}}\ \ V(\theta^{(D)}, \theta^{(G)})$$

Quindi durante il training si addestrano simultaneamente due modelli:

- Modello del generator G che cattura la distribuzione dei dati.

- Modello del discriminator D che stima la probabilità che un determinato campione provenga dal training set o sia un output del modello G.

Di conseguenza, la procedura di addestramento per G consiste nel massimizzare la probabilità che D commetta un errore e per farlo deve modificare i suoi parametri in modo che minimizzino la sua cost function.

La loss function utilizzata per il generatore nel gioco minimax, rappresentata da $J^{(G)} = -J^{(D)}$, è utile per l'analisi teorica, ma non funziona particolarmente bene nella pratica. Nel gioco minimax, il discriminator minimizza la cross entropy, ma il generator massimizza la stessa cross entropy. In questo modo il generator viene penalizzato perché quando il discriminator rifiuta con successo i campioni del generator con elevata sicurezza, allora il gradiente del generator svanisce e di conseguenza il generator non riesce a migliorare i propri output.
Per risolvere questo problema, un approccio consiste nel continuare a utilizzare la minimizzazione della cross entropy per il generator però, invece di capovolgere il segno sul costo del discriminator per ottenere un costo per il generator, capovolgiamo l'obiettivo utilizzato per costruire il costo del generator. Quindi, il costo per il generator diventa:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{z\sim p_z}\ [log\ (D(G(z)))]$$

Nel gioco minimax, il generator riduce al minimo la probabilità logaritmica che il discriminator funzioni correttamente sulle immagini da lui generate. In questo gioco, il generator massimizza la probabilità logaritmica che il discriminator si sbagli.

2.4 Conditional Generative Adversarial Networks (cGAN)

Le Conditional Generative Adversarial Networks (cGAN) sono un miglioramento delle GAN proposte da Mirza e Osindero (2014a) subito dopo l'introduzione delle GAN da Goodfellow et al. (2014). La funzione obiettivo delle cGAN è definita dalla seguente equazione:

$$\begin{aligned} arg\ min_{\theta^{(G)}}\ max_{\theta^{(D)}}\ \ & \mathbb{E}_{x\sim p_{data}(x)}[log(D_{\theta^{(D)}}(x|y))] + \\ & \mathbb{E}_{x\sim p_z(z)}[1 - log(D_{\theta^{(D)}}(G_{\theta^{(G)}}(z|x)))] \end{aligned}$$

che è molto simile a quella che abbiamo visto per le GAN classiche tranne che gli input sia per il discriminator che per il generator sono condizionati da un'etichetta di classe y.
La principale innovazione tecnica introdotta dalle cGAN è quella di aggiungere più input al modello GAN originale, consentendo al modello di essere addestrato su informazioni come etichette di classe o altre variabili condizionali, oltre ai soliti campioni della distribuzione dei dati. Le classiche GAN vengono addestrate solo su campioni dalla distribuzione dei dati ottenendo risultati che riflettono unicamente la distribuzione generale del dataset, invece le cGAN consentono di indirizzare il modello verso la generazione di output più personalizzati in accordo con il vettore di condizione.

Le cGAN hanno la seguente struttura:

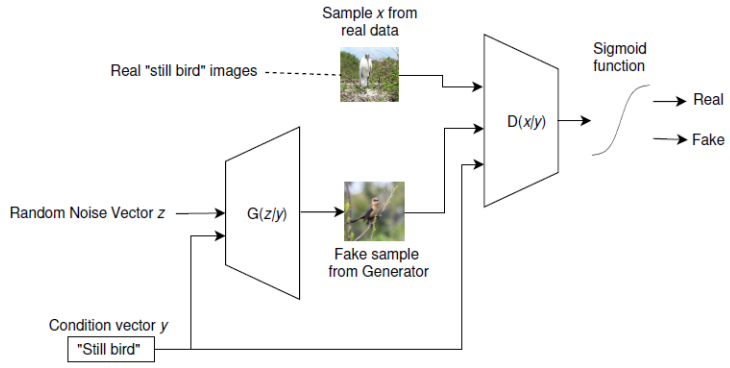

Figure 1: Conditional Generative Adversarial Networks (cGAN)

In Figura 1, il *condition vector* è rappresentato dall'etichetta di classe (stringa di testo) "Red bird", che viene fornita sia al generator che al discriminator. Tuttavia, è importante che il vettore di condizione sia correlato con i dati reali. Questo è importante perché se il modello nella Figura 1 fosse stato addestrato con lo stesso set di dati reali contenente uccelli rossi ma il testo della condition fosse "Yellow fish", il generatore imparerebbe a creare immagini di uccelli rossi quando viene condizionato con il testo "Yellow fish".
Notiamo che il *condition vector* può presentarsi in molte forme diverse, come ad esempio stringhe di testo, e non è limitato alla class label. Questa speciale struttura di GAN fornisce una soluzione diretta alla generazione immagini condizionate da specifiche predefinite. Di conseguenza, le cGAN sono state subito utilizzate nella sintesi da testo a immagine.

3 Recurrent Neural Network e Attention

Reti neurali ricorrenti, LSTM e reti neurali ricorrenti gated in particolare, sono state saldamente stabilite come approcci all'avanguardia nella modellazione di sequenze e problemi di trasduzione come la modellazione del linguaggio e la traduzione automatica. Da allora numerosi sforzi hanno continuato a spingere i confini dei modelli linguistici ricorrenti e delle architetture encoder-decoder.
I modelli ricorrenti tipicamente fattorizzano il calcolo lungo le posizioni dei simboli delle sequenze di input e output. Allineando le posizioni ai passi nel tempo di calcolo, generano una sequenza di stati nascosti h_t, in funzione del precedente stato nascosto h_{t-1} e l'input per la posizione t.
I meccanismi di attenzione sono diventati parte integrante della modellazione di sequenze e dei modelli di trasduzione in vari compiti, consentendo la modellazione delle dipendenze indipendentemente dalla loro distanza nelle sequenze di input o output. In tutti i casi tali meccanismi di attenzione vengono utilizzati insieme a una rete ricorrente.

3.1 Attention

I moduli di attention vengono utilizzati per far apprendere alla CNN come concentrarsi maggiormente sulle informazioni importanti, piuttosto che apprendere informazioni di base non utili.
Una funzione di attenzione può essere descritta come la mappatura di una query e di un insieme di coppie chiave-valore in un output, in tale scenario la query, le chiavi, i valori e l'output sono rappresentati mediante vettori. L'output viene calcolato come somma pesata dei valori, dove il peso assegnato a ciascun valore viene calcolato da una funzione di compatibilità tra query con la chiave corrispondente al valore.

3.1.1 Problema dei seq2seq models e nascita dei moduli attention

Il modello seq2seq è normalmente composto da un'architettura encoder-decode, dove l'encoder elabora la sequenza di input e codifica e comprime le informazioni in un vettore di contesto di lunghezza fissa. Questa rappresentazione dovrebbe essere un buon riassunto dell'intera sequenza di input. Il decoder viene quindi inizializzato con questo vettore di contesto, utilizzando il quale inizia a generare l'output trasformato.
Il problema di questo tipo di modelli è rappresentato dall'utilizzo di un vettore di contesto a lunghezza fissa, il quale non permette di ricordare sequenze più lunghe della dimensione di tale vettore. Per questo motivo, di solito viene dimenticata la parte iniziale della sequenza quando si arriva ad analizzare la parte finale.
Per risolvere questo problema venne introdotto il meccanismo dell'attention. L'idea centrale dei moduli attention è quella di prestare attenzione solo ad alcune parole dell'input rilevanti per l'output che si sta calcolando, quindi che ogni volta che il modello prevede un output, utilizza solo parti dell'input in cui sono

concentrate le informazioni più rilevanti invece dell'intera sequenza.
L'attention è un'interfaccia che collega l'encoder e il decoder e che fornisce al decoder le informazioni da ogni stato nascosto dell'encoder. Con questo modulo aggiuntivo, il modello è in grado di concentrarsi selettivamente su parti rilevanti della sequenza di input e apprendere l'associazione tra esse. Questo aiuta il modello a far fronte in modo efficiente a lunghe sequenze di input.

4 LSTM

`LSTM` è una particolare tipologia di layer ricorrente; la struttura di una generica cella LSTM (Long Short Term Memory) è mostrata nella figura sottostante (Figura: 2); in essa possiamo vedere:

- Lo stato della cella, mostrato nella prima linea orizzontale dall'alto nel diagramma; questo contiene informazioni che vengono modificate relativamente poco, e rappresenta la "memoria" di una LSTM (nella quale l'output della cella viene poi utilizzato come input per il timestep successivo).
- Una serie di gate, formati da un layer con attivazione sigmoide, seguito da un'operazione punto a punto (somma o moltiplicazione).
- L'effettivo dato di input della cella, indicato dalla linea verticale in basso a sinistra nel diagramma.
- Lo stato nascosto, che viene aggiornato tramite l'input e lo stato della cella, e che diventa poi parte dell'input per il successivo timestep.

La funzione dei gate è, nel pratico, quella di determinare quali informazioni mantenere e quali no all'interno dello stato della cella; da sinistra a destra:

- Il primo è detto *forget gate layer* e sulla base dello stato nascosto e dell'input, determina quali valori dello stato della cella azzerare o diminuire; infatti, dopo il layer sigmoide (che dà in output valori in [0, 1]) viene effettuata una moltiplicazione.
- Il secondo (*input gate*) determina quali valori aggiornare; utilizza un layer sigmoide il cui output viene moltiplicato per quello di un layer *tanh*; in questo modo, si ha una serie di valori in [-1, 1], scalati per dei fattori in [0, 1] in base alla necessità di aggiornare i vari punti dello stato della cella; questi valori scalati vengono poi sommati a quest'ultimo.
- Il terzo (*output gate*) stabilisce quali informazioni verranno conservate in output (e nello stato nascosto); di conseguenza, utilizziamo un layer sigmoide per decidere quali informazioni dello stato nascosto mantenere, e una funzione *tanh* per scalare i valori dello stato della cella.

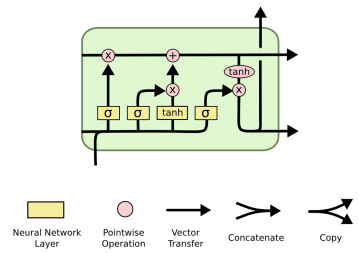

Figure 2: Una cella LSTM, con associata legenda per la rappresentazione.

5 Fase 1: Riproduzione dei risultati di AttnGAN

5.1 Rete AttnGAN

La rete AttnGAN è descritta nell'articolo: "AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks" [15] e il codice dell'implementazione a cui faremo riferimento in questo capitolo è disponibile nel seguente repository github [8].

5.1.1 Introduzione

La generazione automatica di immagini in base alle descrizioni in linguaggio naturale è un problema fondamentale che può essere applicato in diversi ambiti, come la generazione artistica e la progettazione assistita da computer.
I metodi di sintesi da testo a immagine precedenti ad AttnGAN erano basati su GAN (Generative Adversarial Networks) condizionate solo sul vettore di frase globale, essi tendevano a perdere importanti informazioni a grana fine a livello di parola, impedendo la generazione di immagini di alta qualità.
Per risolvere questo problema, nell'articolo relativo all'Attentional Generative Adversarial Network (AttnGAN) viene proposto un raffinamento multistage basato sull'attenzione per la generazione a grana fine di un'immagine da testo.
Il modello è costituito da due componenti:

- **Attentional Generative Network**: in cui viene sviluppato un meccanismo di attention per ogni generatore al fine di disegnare diverse sottoregioni dell'immagine concentrandosi sulle parole che sono più rilevanti per la sottoregione considerata. Nello specifico, oltre a codificare la descrizione in linguaggio naturale in un vettore di frase globale, ogni parola nella frase è anche codificata in un vettore parola. La rete generativa utilizza il vettore frase globale per generare un'immagine a bassa risoluzione nella prima fase. Nelle fasi successive, il modulo attention viene utilizzato per ottenere un vettore di contesto contenente coppie nella forma (sottoregione immagine, parole rilevanti della descrizione) in grado di istruire il generatore della cGAN sulle caratteristiche di una sottoregione dell'immagine di output in base alla descrizione presa in input. Questo permette di produrre in modo efficace un'immagine a risoluzione più elevata con maggiori dettagli in ogni fase.

- **Deep Attentional Multimodal Similarity Model (DAMSM)**: mediante un meccanismo di attenzione, il DAMSM è in grado di calcolare la somiglianza tra l'immagine generata e la frase utilizzando sia le informazioni a livello di frase globale che le informazioni a livello di parola.

La rete è definita come segue, descriveremo tutte le componenti in modo approfondito nelle seguenti sezioni:

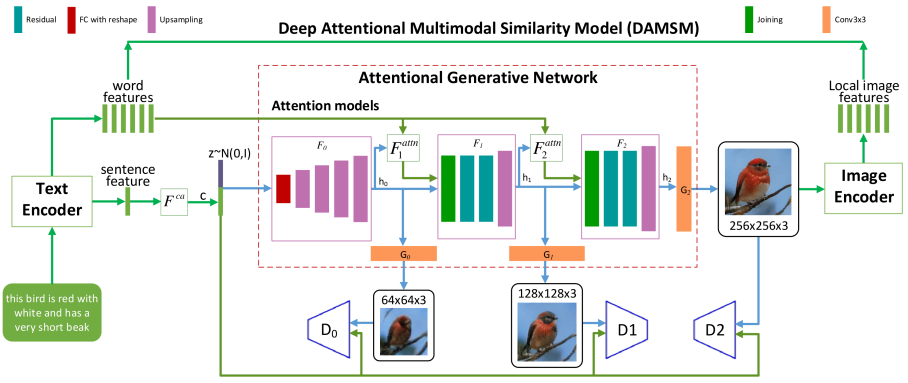

Figure 3: Rete AttnGAN

5.1.2 Attentional Generative Network

La componente chiamata Attentional Generative Network di AttnGAN consente alla rete di disegnare diverse sottoregioni dell'immagine in base alle parole della descrizione che sono più rilevanti per la regione considerata.

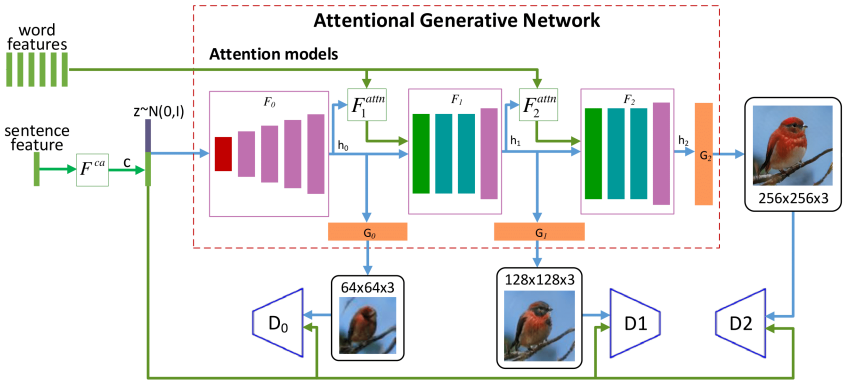

Figure 4: Componente Attentional Generative Network di AttnGAN

L'Attentional Generative Network è formata da m generator identificati da $G_0, ..., G_{m-1}$ che prendono come input l'hidden state della fase precedente, essi sono identificati da $h_0, ..., h_{m-1}$ e restituiscono come output immagini di dimensioni da piccole a grandi $\widehat{x}_0, ..., \widehat{x}_{m-1}$. Nello specifico gli hidden state e i generator sono definiti come segue:

- $h_0 = F_0(z, F^{ca}(\bar{e}))$ perché nella prima fase viene utilizzato il vettore frase globale per generare un'immagine a bassa risoluzione.
- $h_i = F_i(h_{i-1}, F_i^{attn}(e, h_{i-1}))$ perché nelle fasi successive le reti utilizzano un vettore di contesto di parola che identifica le parole rilevanti per ogni sottoregione dell'immagine, tale vettore è viene ottenuto come output del modulo attention F_i^{attn}, che verrà descritto in seguito.
- $\widehat{x}_i = G_i(h_i)$ come descritto precedentemente i generator prendono in input l'hidden state del medesimo stage e restituiscono l'immagine identificata da $\widehat{x}_i$.

Osserviamo che:

- z preso in input nella prima fase della rete è un noise vector di solito campionato da una normale standard come media 0 e varianza 1, identificata in figura da $N(0, 1)$.
- $\bar{e}$ è il global sentence vector che rappresenta le features della frase considerata.
- e è una matrice di word vector, ognuno di questi vettori rappresenta le features di ogni word.
- F^{ca} è una rete neurale che rappresenta il Conditioning Augmentation, il quale converte il sentence vector $\bar{e}$ nel conditioning vector c. Nel codice questa rete è definita nella classe `CA_NET` nel file `model.py`.

All'interno del codice l'Attentional Generative Network è definita nella classe `G_NET` del file `model.py` essa si occupa della generazione delle reti F_0, F_1, F_2 e dei generator G_1, G_2, G_3 in figura.

Rete iniziale dell'Attentional Generative Network La rete neurale F_0 viene descritta all'interno del codice nel file `model.py` nella classe `INIT_STAGE_G` in cui il modello della rete è definito come segue:

- Modulo fc composto da un layer lineare che prende in input z e c, una batch normalization del vettore c e un modulo definito nella classe `GLU`.
- Primo blocco di upsampling che da un input 3x4x4 restituisce un tensore di dimensione 3x8x8.
- Secondo blocco di upsampling che da un input 3x8x8 restituisce un tensore di dimensione 3x16x16.
- Terzo blocco di upsampling che da un input 3x16x16 restituisce un tensore di dimensione 3x32x32.
- Quarto blocco di upsampling che da un input 3x32x32 restituisce un tensore di dimensione 3x64x64.

Upsampling block: i blocchi di upsampling sono definiti dalla funzione `upBlock` all'interno di `model.py` e sono formati da: un layer `nn.Upsample` che raddoppia la dimensione dell'input in base all'upsampling algorithm dei nearest neighbor, un layer convoluzionale 3x3 definito nella funzione `conv3x3` e un modulo definito nella classe `GLU`.

Reti successive dell'Attentional Generative Network Le reti neurali F_i con $i \in \{1, ..., m-1\}$ sono definite nella classe `NEXT_STAGE_G` del file `model.py` in cui il modello della rete è definito come segue:

- Layer di joining che effettua il join tra l'output dell'attention model cioè l'output di $F_i^{attn}(e, h_{i-1})$, che verrà discusso nel dettaglio nella prossima sezione, e l'hidden state h_{i-1}, tale join viene implementato mediante un `torch.cat`.
- Due residual block
- Un blocco di upsampling che da un input 3x64x64 restituisce un tensore di dimensione 3x128x128.

Residual block: i residual block sono definiti nella classe `ResBlock` del file `model.py` e sono formati da: un layer convoluzionale 3x3 definito nella funzione `conv3x3`, un batch normalization implementato con `BatchNorm2d`, un modulo definito nella classe `GLU`, un layer convoluzionale 3x3 definito nella funzione `conv3x3` e un batch normalization implementato con `BatchNorm2d`.

Attention model: L'attention model F^{attn} è definito nel file `GlobalAttention` nella classe `GlobalAttentionGeneral`. Tale rete prende in input la matrice $e \in \mathbb{R}^{DxT}$ di word vector detto *word features* e l'hidden state $h \in \mathbb{R}^{\widehat{D}xN}$ del precedente layer detto *image features*.
Per prima cosa il vettore delle word features vengono convertite nel medesimo spazio semantico delle image features ottenendo $e' \in \mathbb{R}^{\widehat{D}xT}$, successivamente un *word context vector* c viene calcolato per ciascuna sottoregione dell'immagine in base alle caratteristiche nascoste descritte dalle image features. Nello specifico, ogni colonna di h rappresenta una sottoregione dell'immagine, se consideriamo la j-esima sottoregione dell'immagine allora il valore j-esimo word context vector c contiene una rappresentazione dinamica dei word vector che sono rilevanti per quella determinata regione.
L'attention model restituisce in output il word context vector $c = (c_0, ..., c_{N-1}) \in \mathbb{R}^{\widehat{D}xN}$ in cui ogni c_j viene calcolato in base alla j-esima sottoregione dell'immagine rappresentata da h_j nel seguente modo:

$$c_j = \sum_{i=0}^{T-1} \beta_{j,i} \; e'_i$$

dove:

- $e'_i \in \mathbb{R}^{\widehat{D}}$
- $\beta_{j,i}$ indica il peso dell'i-esima parola quando si genera la j-esima sottoregione dell'immagine.

Reti dei generator Le reti neurali che rappresentano i generator G_i sono definiti dalla classe `GET_IMAGE_G` del file `model.py` e sono composti da: un layer convoluzionale 3x3 definito nella funzione `conv3x3` e un layer tanh implementato con `nn.Tanh()`.

Reti dei discriminator Le reti neurali che rappresentano i generator D_i sono definiti nelle classi `D_NET64`, `D_NET128` e `D_NET256` del file `model.py`, essi cercano di distinguere tra un campione generato da G_i e un campione del training set.

Objective function dell'Attentional Generative Network Per generare immagini realistiche con più livelli (livello di frase e livello di parola) di condizioni, la funzione obiettivo finale dell'Attentional Generative Network è definita come:

$$\mathcal{L} = \mathcal{L}_G + \lambda \mathcal{L}_{DAMSM}$$

dove:

- $\mathcal{L}_G = \sum_{i=0}^{m-1} \mathcal{L}_{G_i}$: è la loss della GAN che approssima congiuntamente sia le conditional distribution che le unconditional distribution. La loss sia dei generator che dei discriminator è quindi formata da due parti:
  - Unconditional loss determina se l'immagine è reale o falsa, nel caso dei generator essa è definita come segue:
    $$-\frac{1}{2}\ \mathbb{E}_{\widehat{x}_i \sim p_{G_i}}[log\ D_i(\widehat{x}_i)]$$
  - Conditional loss determina se l'immagine e la frase corrispondono o meno, nel caso dei generator essa è definita come segue:
    $$-\frac{1}{2}\ \mathbb{E}_{\widehat{x}_i \sim p_{G_i}}[log\ D(\widehat{x}_i, \bar{e})]$$

  $\Rightarrow$ Il generator G_i cerca di massimizzare la seguente funzione, poiché esso cerca di massimizzare la probabilità che il discriminator si sbagli ed etichetti come corretta l'immagine $\widehat{x}_i$ generata da G_i:

  $$\mathcal{L}_{G_i} = -\frac{1}{2}\ \mathbb{E}_{\widehat{x}_i \sim p_{G_i}}[log\ D_i(\widehat{x}_i)] - \frac{1}{2}\ \mathbb{E}_{\widehat{x}_i \sim p_{G_i}}[log\ D(\widehat{x}_i, \bar{e})]$$

- Analogamente, l'i-esimo discriminator D_i è addestrato per classificare l'input nella classe di istanze reali o false minimizzando la perdita della cross entropy definita da:

$$\mathcal{L}_{D_i} = -\frac{1}{2}\,\mathbb{E}_{x_i \sim p_{data_i}}[log\ D_i(x_i)] - \frac{1}{2}\,\mathbb{E}_{\widehat{x}_i \sim p_{G_i}}[log\ 1 - D(\widehat{x}_i)] + \\ -\frac{1}{2}\,\mathbb{E}_{x_i \sim p_{data_i}}[log\ D_i(x_i, \bar{e})] - \frac{1}{2}\,\mathbb{E}_{\widehat{x}_i \sim p_{G_i}}[log\ 1 - D(\widehat{x}_i, \bar{e})]$$

- λ: è un'iperparametro per bilanciare i due termini dell'equazione.
- $\mathcal{L}_{DAMSM}$: è la loss della corrispondenza immagine-testo a grana fine a livello di parola calcolata dal DAMSM, questa parte della rete verrà definita in seguito.

5.1.3 Deep Attentional Multimodal Similarity Model (DAMSM)

Il Deep Attentional Multimodal Similarity Model (DAMSM) è formato da due reti neurali: text encoder e image encoder, esse mappano le sottoregioni dell'immagine e le parole della frase in uno spazio semantico comune. Infine, il DAMSM misura la somiglianza immagine-testo a livello di parola (l'effettivo procedimento di calcolo della loss è descritto nell'articolo di AttnGAN).

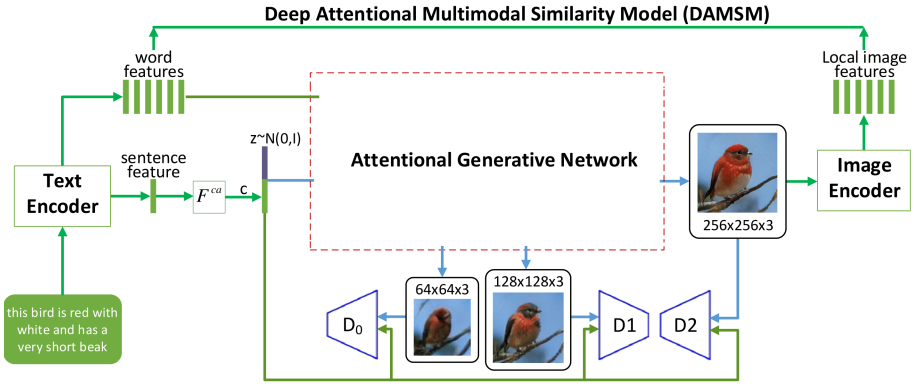

Figure 5: Componente Deep Attentional Multimodal Similarity Model di AttnGAN

Text Encoder Il text encoder è rappresentato da una Long Short-Term Memory (LSTM) bidirezionale che estrae vettori semantici dalla descrizione del testo. Nell'LSTM bidirezionale, ogni parola corrisponde a due stati nascosti, uno per ogni direzione. Quindi, concatena i suoi due stati nascosti per rappresentare il significato semantico di una parola. La matrice delle features di tutte le parole è rappresentata dal vettore $e \in \mathbb{R}^{DxT}$, dove:

- i-esima colonna di questo vettore e_i rappresenta il word vector della i-esima parola, esso ha dimensione D.
- D è la dimensione del word vector.
- T numero di parole.

Nel frattempo, gli ultimi stati nascosti dell'LSTM bidirezionale sono concatenati per costruire il vettore frase globale, indicato con $\bar{e} \in \mathbb{R}^D$.

Image Encoder L'image encoder è una Convolutional Neural Network (CNN) che mappa le immagini in vettori semantici. Gli strati intermedi della CNN apprendono le features locali di diverse sottoregioni dell'immagine, mentre gli strati successivi apprendono le caratteristiche globali dell'immagine. Per fare questo l'image encoder si basa sul modello Inception-v3 pre-addestrato su ImageNet.

5.1.4 Training AttnGAN

Il training viene eseguito in maniera piuttosto semplice: ad ogni epoch vengono generati gli output relativi al batch corrente da parte del generatore; questi sono organizzati come una lista di tre tensori, ognuno corrispondente alle immagini di una certa risoluzione (rispettivamente 64x64, 128x128 e 256x256); i tre discriminatori vengono poi calibrati utilizzando come input le immagini generate, quelle reali e le relative caption (sotto forma di embedding).

Successivamente viene calcolata la loss per i generatori, utilizzando i discriminatori calibrati col batch corrente; all'errore totale viene poi aggiunta la loss del DAMSM per incentivare la vicinanza tra gli encoding delle immagini e del testo.

5.2 Riproduzione dei risultati di AttnGAN

5.2.1 Creazione dell'ambiente per eseguire AttnGAN

Per riprodurre i risultati di AttnGAN è stato prima di tutto creato un ambiente isolato utilizzando `conda`; questo in quanto il codice pubblicato presenta una dipendenza da Python 2.7, e di conseguenza versioni datate di Pytorch, cuda e cuDNN; a causa di ciò si è cercato di evitare possibili conflitti con altre librerie, creando un ambiente a se stante. Nello specifico, è stato utilizzato Pytorch 0.4.0, cuda92 e cuDNN 7.1.2; le altre dipendenze sono risultate compatibili senza fissare la versione.

5.2.2 Impostazione dei parametri ed esecuzione

Per quanto riguarda gli iperparametri, sono stati usati quelli consigliati nei file di configurazione del progetto, in modo da poter riprodurre i risultati nel modo più coerente possibile; anche la generazione delle immagini è stata effettuata mantenendo i parametri uguali a quelli già presenti nelle configurazioni.

Figure 6: Alcune immagini generate da AttnGAN dopo 600 epoch

6 Fase 2: Implementazione di DualAttnGAN

La rete DualAttnGAN è descritta nell'articolo: "DualAttn-GAN: Text to Image Synthesis with Dual Attentional Generative Adversarial Network" [3] e il codice dell'implementazione a cui faremo riferimento in questo capitolo è disponibile nel seguente repository github [9].

6.1 Rete DualAttnGAN

La rete DualAttnGAN è descritta nell'articolo: "DualAttn-GAN: Text to Image Synthesis with Dual Attentional Generative Adversarial Network" [3].

I metodi basati su GAN (Generative Adversarial Networks) per la sintesi da testo a immagine hanno mostrato buone prestazioni. E i meccanismi di attenzione sono stati utilizzati nella generative adversarial network per la sintesi da testo a immagine. L'articolo su AttnGAN è stato il primo ad introdurre meccanismi di attenzione nei framework GAN, che guidano la generazione di dettagli raffinati questo causa la distorsione della forma dell'oggetto.
Per risolvere il problema appena descritto, l'articolo su DualAttnGAN propone la struttura di Dual Attentional Generative Adversarial Networks. Al fine di sintetizzare dettagli locali interessanti con strutture globali significative, DualAttnGAN introduce dei Dual Attention Module (DAM), che include tre componenti principali: Textual Attention Module (TAM), Visual Attention Module (VAM), Attenion Embedding Module (AEM).
L'architettura di rete è sempre composta da più generator e discriminator e segue un'architettura ad albero. Il primo ramo genera un'immagine a bassa risoluzione che contiene i colori corretti e la struttura approssimativa dell'oggetto. I rami successivi generano immagini ad alta risoluzione che si concentrano sui dettagli dell'immagine.
Le principali differenze tra AttnGAN e DualAttnGAN sono localizzate nella porzione di rete denominata Attentional Generative Network, tali differenze verranno illustrate nella sezione seguente.

6.1.1 Attentional Generative Network

La componente di Attentional Generative Network di DualAttnGAN è la seguente:

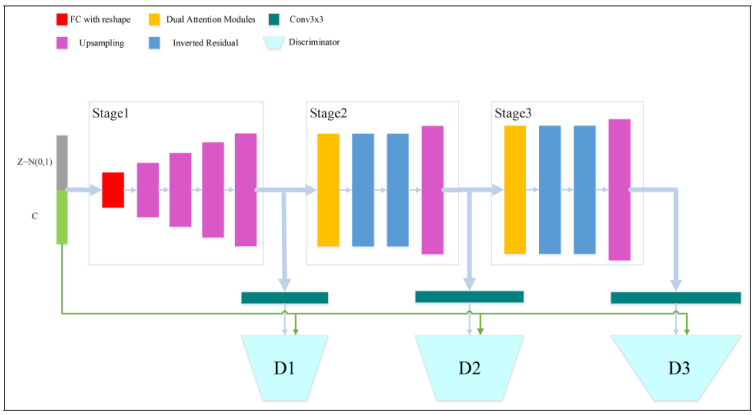

Figure 7: Componente Attentional Generative Network di DualAttnGAN

Lo stage iniziale ha la stessa struttura descritta per la rete iniziale F_0 di AttnGAN, con l'unica differenza che al posto dell'attivazione GLU viene utilizzata una ReLu.
Gli stage successivi di DualAttnGAN introducono le seguenti differenze rispetto alla rete AttnGAN:

- Il modulo di attention e il layer di joining vengono sostituiti dal Dual Attention Module (DAM), utilizzato per migliorare i dettagli delle immagini generate dalla rete attraverso l'estrazione di caratteristiche da parole rilevanti della frase analizzata e diverse regioni visive dell'immagine.
  Il **Dual Attention Module (DAM)** è formato dalle seguenti componenti:
  - Textual Attention Module (TAM): progettato per esplorare l'interazione dettagliata tra visione e linguaggio.
  - Visual Attention Module (VAM): componente che modella le rappresentazioni interne dell'immagine rispetto al canale e agli assi spaziali, in modo tale da estrarre al meglio le strutture globali dell'immagine.
  - Attenion Embedding Module (AEM): componente utilizzato unire e combinare le features estratte dall'analisi testuale del TAM e dall'analisi visuale del VAM.
- Il due moduli residui di AttnGAN vengono sostituiti da una struttura di inverted residual, in cui i colli di bottiglia contengono effettivamente tutte le informazioni importanti.
- In DualAttnGAN, per stabilizzare l'addestramento delle GAN, viene adottata la spectral normalization delle reti dei generator e dei discriminator.

6.1.2 Dual Attention Module (DAM)

Il Dual Attention Module, rappresentato con i blocchi arancioni nella figura 7, ha la seguente struttura interna:

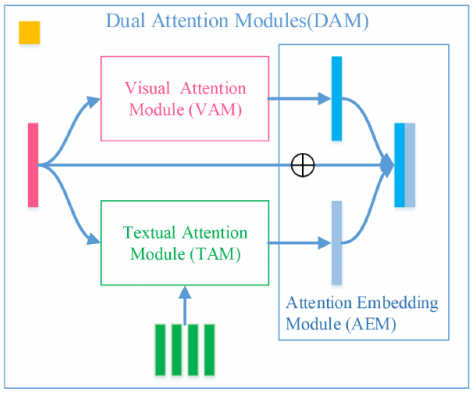

Figure 8: Componente Dual Attention Module di DualAttnGAN

Textual Attention Module (TAM) Il Textual Attention Module $F_t^{attn}(e, h)$ serve per modellare la relazione di mapping tra le word features e le image features h dal precedente hidden layer, esso è in grado di generare dettagli visivi molto specifici collegati a aspetti semantici del testo.
All'interno di tale modulo vengono eseguite le seguenti operazioni:

1. Allineamento delle word features e delle image features all'interno dello stesso spazio semantico.

2. I pesi di attention vengono ottenuti calcolando la rilevanza delle word features rispetto ad ogni sottoregione dell'immagine rappresentate dall'image features mediante un prodotto punto a punto e una normalizzazione tramite softmax.

3. Il *word context vector* c_i per ogni sottoregione i dell'immagine viene calcolato come media pesata sulle word features e_i con i pesi dell'attenzione.

Infine, la *word context matrix* $\{c_0, ..., c_i, ...\}$ viene passata all'Attenion Embedding Module (AEM).

Visual Attention Module (VAM) Il Visual Attention Module $F_v^{attn}(h)$ è progettato per migliorare la qualità della rappresentazione, esso può imparare a utilizzare le informazioni globali per concentrarsi selettivamente su caratteristiche importanti e sopprimere quelle non necessarie.
Il modulo VAM utilizza le informazioni da due prospettive, vale a dire channel e spatial axes, per apprendere "cosa" e "dove" sono le parti informative.

Per prima cosa, date le image features h dell'hidden layer precedente, viene applicato il Channel Attention Module $F_{v_c}^{Attn}(h)$ per ottenere una mappa pesata rispetto ai channel denominata h'. Successivamente, passiamo in input h' allo Spatial Attention Module $F_{v_s}^{Attn}(h')$ ottenendo h''. Quindi, il Visual Attention Module può essere rappresentato come segue:

$$F_v^{Attn} = F_{v_s}^{Attn}(F_{v_c}^{Attn}(h))$$

Le componenti del Visual Attention Module hanno le seguenti strutture:

- Channel Attention Module (CAM) è stato progettato al fine di identificare cosa sia importante nelle image features estraendo relazioni interessanti dalle channel features delle immagini. Al fine di prendere in considerazione l'intera struttura globale allora sia le average features che le maximum features vengono elaborate simultaneamente.
  Possiamo riassumere il funzionamento del Channel Attention Module nel seguente modo:

  $$F_{v_c}^{Attn}(h) = \sigma(MLP(AvgPool(h) + MLP(MaxPool(h)) \otimes h$$

  dove:

  - σ: funzione sigmoide
  - MLP: rete FC-ReLu-FC
  - $\otimes$: la moltiplicazione punto a punto

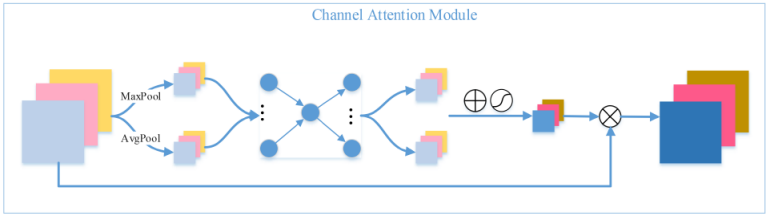

Figure 9: Channel Attention Module (CAM) di DualAttnGAN

- Spatial Attention Module (SAM) è stato progettato per estrarre le relazioni contestuali, esse sono fondamentali per catturare le dipendenze globali, tale modulo è in grado di codificare le caratteristiche locali dell'immagine in informazioni contestuali interessanti.

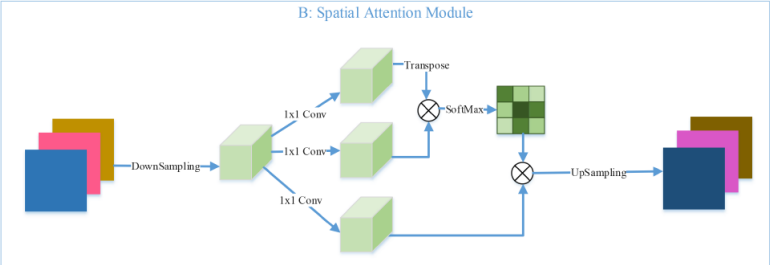

Figure 10: Spatial Attention Module (SAM) di DualAttnGAN

La struttura del Spatial Attention Module viene descritta in modo approfondito nel seguente articolo: "Self-Attention Generative Adversarial Networks" [10].

Attenion Embedding Module (AEM) L'Attenion Embedding Module serve per unire insieme le textual features ottenute dal Textual Attention Module e le image features ottenute dal Visual Attention Module. Per fare questo, le input features h vengono aggiunte agli output dei due moduli precedenti (TAM e VAM) mediante una skip-connection e successivamente viene effettuata la concatenazione per completare l'unione.

6.2 Modifiche al codice di AttnGAN

Al fine di implementare l'architettura di rete descritta nell'articolo relativo a DualAttnGAN [3] è stata effettuata una modifica del codice relativo ad AttnGAN [8], le modifiche al codice descritte in questa sezione sono disponibili al seguente repository github del progetto: `https://github.com/mattrent/AttnGAN` [9].

Le aggiunte principali al codice sono state implementate nel file `dualAttnModel.py`, nel quale sono presenti:

- Dual Attention Module, a sua volta composto dai sottomoduli
  - Visual Attention Module, composto da Channel Attention Module e Spatial Attention Module
  - Text Attention Module, corrispondente al modulo attention già presente in AttnGAN
- I diversi stage dell'effettiva GAN, modificati per includere il Dual Attention Module (portando quindi la maggior parte delle modifiche nello stage intermedio)

L'implementazione del modulo ha seguito quella descritta nell'articolo, salvo una modifica nel Channel Attention Module per ragioni di prestazioni, descritta di seguito; è importante specificare che non sono state invece implementate le

modifiche relative agli inverted residual layer (al posto dei semplici residual layer), in quanto non è stato possibile testare a fondo il modello completo (e quindi annotare l'effetto di questi ultimi sui risultati prodotti).

La principale deviazione rispetto alla specifica dell'articolo è stata, come prima affermato, nell'implementazione del Channel Attention Module; in fase di training è stato infatti rilevato un utilizzo proibitivo di memoria da parte del MLP successivo al pooling, che aveva il compito di riportare l'immagine alla sua dimensione originale (per il passaggio attraverso lo Spatial Attention Module); a causa di ciò si è scelto di ridurre dimensionalmente i dati passanti per i layer densi, applicando layer `conv2d` con `stride=2` e riduzione del numero di canali in output ad un ottavo di quelli in input.
Così facendo è stato quindi possibile effettuare il training e la valutazione del modello; è probabile che la perdita di informazioni derivante da questa modifica abbia causato risultati inferiori a quelli attesi, ma (come visibile nelle successive tabelle 3 e 4) sono state ugualmente prodotte immagini di buona qualità secondo le metriche utilizzate.

6.3 Esecuzione di DualAttnGAN e risultati ottenuti

L'esecuzione di DualAttnGAN a scopo di validazione è stata effettuata seguendo la modalità già esistente per AttnGAN; il file `eval_bird_dualattn.yml` è stato aggiunto, così come `bird_dualattn.yml` (per il training); in entrambi è stato inserito il nuovo parametro `DUAL_ATTN`, ad indicare l'utilizzo del nuovo modello.

L'unica modifica sostanziale ai parametri passati è stata nel batch size, sempre per ragioni di utilizzo di memoria; è stato infatti visto che il batch size di 20, utilizzato da AttnGAN, portava ad un errore OOM nella VRAM; si è scelto quindi di ridurre il parametro ad un valore di 8, permettendo il training senza ulteriori errori; in fase di produzione immagini è stato invece possibile utilizzare una batch size di 20, grazie al minore utilizzo di memoria.

Figure 11: Alcune immagini generate da DualAttnGAN dopo 250 epoch

7 Fase 3: Confronto AttnGAN e DualAttnGAN

Per valutare e confrontare la qualità delle immagini ottenute dal training delle due reti analizzate in questo progetto, AttnGAN [15] e DualAttnGAN [3], sono state utilizzate le seguenti due metriche: Inception Score (IS) e Frechet Inception Distance (FID).

7.1 Inception Score (IS)

Per la fase di evaluation dei risultati del modello viene usato l'inception score come misura di valutazione quantitativa, tale misura è descritta nel dettaglio nell'articolo: "Improved Techniques for Training GANs" [6].
L'inception score (IS) è una metrica popolare per giudicare gli output delle immagini di una Generative Adversarial Networks (GAN). L'IS prende in input un insieme di immagini generate dalla GAN che si vuole analizzare e restituisce in output un punteggio rappresentato da un valore reale, tale valore misura quanto sia realistico l'output della GAN.
L'inception score misura:

- La varietà delle immagini
- Se ogni immagine analizzata assomiglia distintamente a qualcosa che appartiene al dataset di training

Se entrambe queste richieste sono verificate allora il punteggio sarà alto. Altrimenti, il punteggio sarà basso.

Formalmente, per calcolare l'Inception Score si applica una rete Inception-v3, allenata precedentemente su database ImageNet, alle immagini generate, confrontando poi la distribuzione condizionata e quella marginale per le categorie inferite dalla rete. Si ha che l'inception score viene calcolata nel seguente modo:

$$IS = exp(E_{x \sim p_g} D_{KL}(p(y|x)||p(y)))$$

dove:

- x è l'immagine prodotta dalla nostra rete.
- Distribuzione condizionata $p(y|x)$: questa distribuzione di probabilità ci dice quanto semplicemente riusciamo ad etichettare le immagini prodotte dalla nostra rete, cioè come sono distribuite le probabilità su tutte le possibili etichette y dato un campione x generato dalla nostra rete.
- Distribuzione marginale $p(y)$: questa distribuzione ci dice come sono distribuite le probabilità relative alla classificazione nella categoria y da Inception-v3 allenata su ImageNet sulle immagini generate dalla nostra rete.

Un IS più alto rappresenta una maggiore D_{KL} tra le due distribuzioni, e quindi immagini generate di migliore qualità perché vogliamo che le due distribuzioni analizzate siano il più distanti possibile, in quanto vogliamo che:

- Le immagini generate dalla nostra rete siano realistiche e quindi semplicemente classificabili da parte di Inception-v3 allenata su ImageNet, di conseguenza si vuole che la distribuzione $p(y|x)$ sia prevedibile, cioè concentrata su un determinato valore di y.
- Ci sia varietà tra le immagini generate dalla nostra rete, di conseguenza si vuole che la distribuzione $p(y)$ sia poco prevedibile, cioè il più possibile uniforme sui valori di y.

All'interno dell'articolo di AttnGAN [15], viene usata la misura dell'inception score al fine di confrontare i risultati di AttnGAN con i precedenti modelli per la risoluzione del text-to-image problem. Per calcolare l'inception score abbiamo utilizzato il codice disponibile al seguente repository git: StackGAN-inception-model [11].

7.2 Frechet Inception Distance (FID)

La Frechet Inception Distance misura la distanza tra due curve:

- La distribuzione delle immagini reali (nel nostro caso le immagini del dataset CUB)
- La distribuzione delle immagini generate (nel nostro caso, quelle generate da AttnGAN o DualAttnGAN)

La Frechet Inception Distance utilizza Inception-v3 allenata su ImageNet in modo diverso rispetto a Inception Score, in questo caso si estraggono le features da un suo layer intermedio, e successivamente si mettono a confronto le features estratte dalle immagini reali e quelle estratte dalle immagini generate. Utilizzando la Frechet Inception Distance non si ha una valutazione isolata dei dati prodotti, come avveniva nel calcolo dell'Inception Score, bensì una comparazione tra questi e dati reali.

All'interno dell'articolo di DualAttnGAN, viene usata la misura della Frechet Inception Distance (FID) al fine di valutare le immagini prodotte dalla rete. Per calcolare la Frechet Inception Distance abbiamo utilizzato il codice disponibile al seguente repository git: Two time-scale update rule for training GANs [13], aggiungendo un resize delle immagini del dataset CUB al fine di confrontarle con le immagini di dimensione 256x256 ottenute dalle due reti considerate in questo progetto.

7.3 Evaluation dei risultati di AttnGAN

7.3.1 Inception Score sui risultati AttnGAN

Il calcolo dell'inception score per la fase di evaluation dei risultati della rete di AttnGAN è stata effettuata sui risultati ottenuti dalla rete ogni 50 epoch, al fine di analizzare il valore di tale score al variare del numero di epoch, ottenendo i seguenti valori:

Numero di epoch	Inception score
0	1.10 ± 0.00
50	4.13 ± 0.22
100	4.27 ± 0.19
150	4.18 ± 0.15
200	4.22 ± 0.15
250	4.13 ± 0.14
300	4.19 ± 0.16
350	4.26 ± 0.18
400	4.17 ± 0.17
450	4.25 ± 0.15
500	4.23 ± 0.15
550	4.13 ± 0.16
600	4.59 ± 0.10

Table 1: Inception Score per risultati di AttnGAN

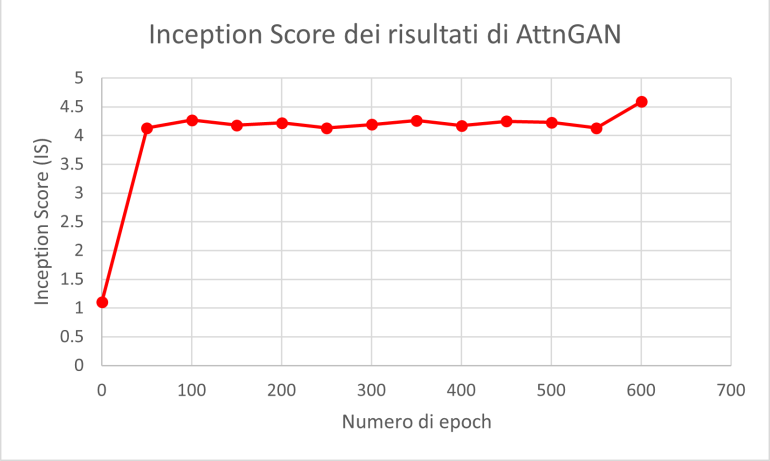

Figure 12: Valore Inception Score (IS) dei risultati di AttnGAN al variare del numero di epoch

7.3.2 FID sui risultati AttnGAN

Il calcolo della FID per la fase di evaluation dei risultati della rete di AttnGAN è stata effettuata sui risultati ottenuti dalla rete ottenendo i seguenti valori:

Numero di epoch	FID
0	316.94
100	27.94
150	24.35
200	20.91
250	21.25
600	19.10

Table 2: Frechet Inception Distance per risultati di AttnGAN

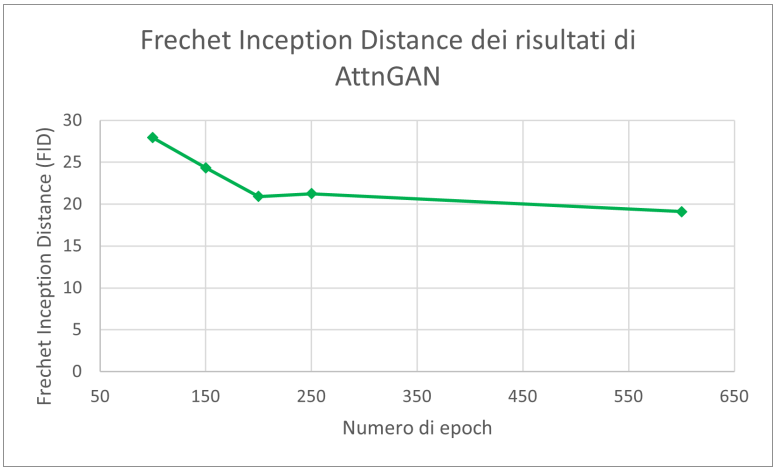

Figure 13: Valore Frechet Inception Distance (FID) dei risultati di AttnGAN al variare del numero di epoch

7.4 Evaluation dei risultati di DualAttnGAN

7.4.1 Inception Score sui risultati DualAttnGAN

Il calcolo dell'inception score per la fase di evaluation dei risultati della rete di DualAttnGAN è stata effettuata sui risultati ottenuti dalla rete ottenendo i seguenti valori:

Numero di epoch	Incetion Score
100	4.07 ± 0.14
150	4.26 ± 0.18
200	4.26 ± 0.15
250	4.19 ± 0.13

Table 3: Inception Score per risultati di DualAttnGAN

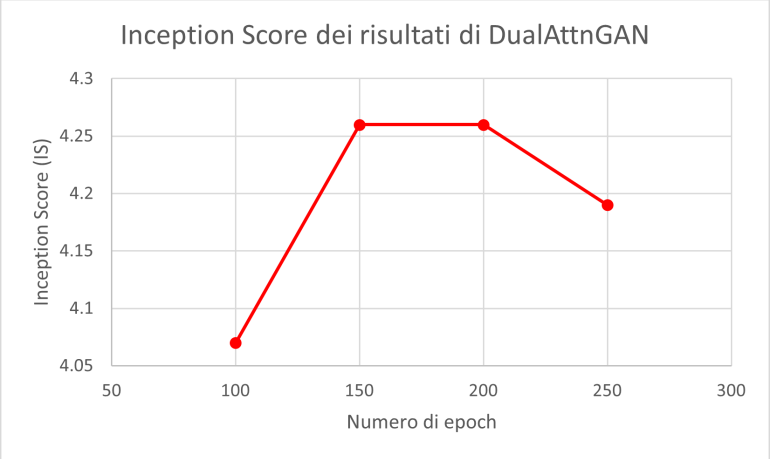

Figure 14: Valore Inception Score (IS) dei risultati di DualAttnGAN al variare del numero di epoch

7.4.2 FID sui risultati DualAttnGAN

Il calcolo della FID per la fase di evaluation dei risultati della rete di DualAttnGAN è stata effettuata sui risultati ottenuti dalla rete ottenendo i seguenti valori:

Numero di epoch	FID
100	28.73
150	26.56
200	24.37
250	23.31

Table 4: Frechet Inception Distance per risultati di DualAttnGAN

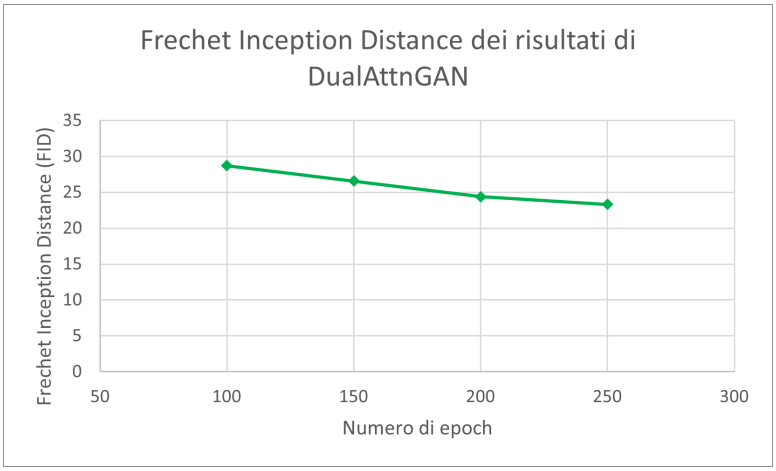

Figure 15: Valore Frechet Inception Distance (FID) dei risultati di DualAttnGAN al variare del numero di epoch

7.5 Confronto tra AttnGAN e DualAttnGAN

Numero di epoch	IS AttnGAN	IS DualAttnGAN
100	4.27 ± 0.19	4.07 ± 0.14
150	4.18 ± 0.15	4.26 ± 0.18
200	4.22 ± 0.15	4.26 ± 0.15
250	4.13 ± 0.14	4.19 ± 0.13

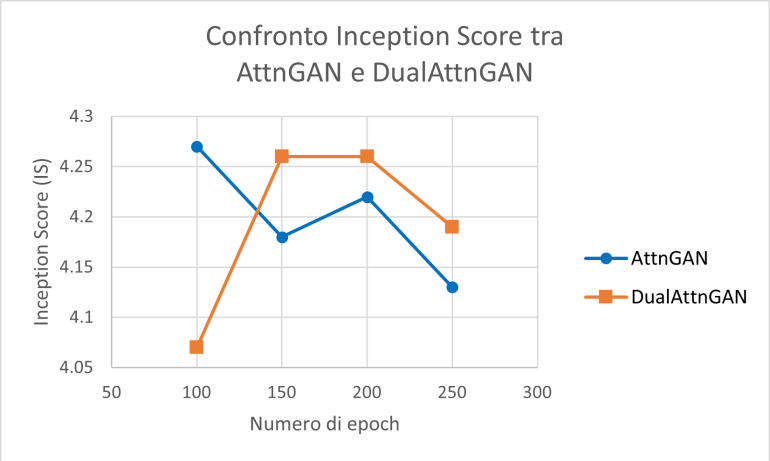

Figure 16: Confronto su Inception Score (IS) tra AttnGAN e DualAttnGAN

Numero di epoch	FID AttnGAN	FID DualAttnGAN
100	27.94	28.73
150	24.35	26.56
200	20.91	24.37
250	21.25	23.31

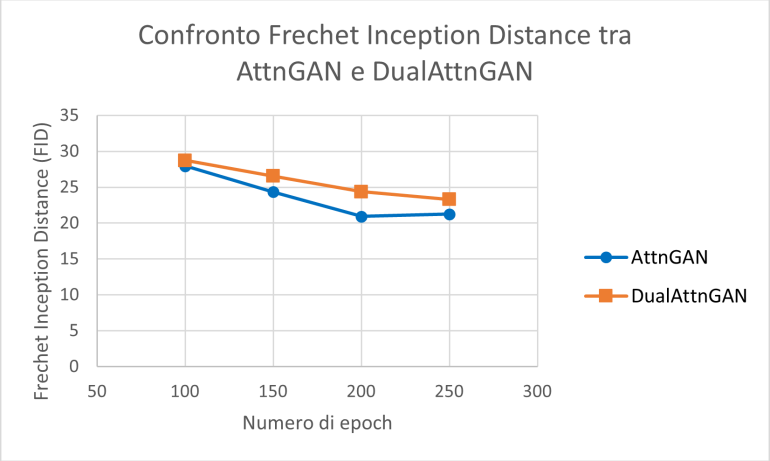

Figure 17: Confronto su Frechet Inception Distance (FID) tra AttnGAN e DualAttnGAN

8 Nota conclusiva e sviluppi futuri

8.1 Nota

Al cominciare del ciclo di training tra le 200 e le 250 epoch di DualAttnGAN si è presentato un malfunzionamento nell'hardware a disposizione; inizialmente si pensava fosse risolto, e che derivasse da una configurazione errata dei driver Nvidia, ma nelle successive epoch (tra le 250 e 300) questo si è ripresentato.

È stato deciso quindi, onde evitare di danneggiare ulteriormente la GPU, e di ottenere risultati inaffidabili, di interrompere il training (e di conseguenza il testing) del modello; questo purtroppo ha reso impossibile un confronto obiettivo tra le due architetture, non essendo stato raggiunto un effettivo plateau né in termini di loss, né di metriche.

Dai risultati ottenuti è comunque possibile vedere come la nuova implementazione sia comparabile, in termini di prestazioni, con quella precedente, e mostri potenzialità di miglioramento su un training completo di 600 epoch; non è stato invece notato un effettivo guadagno visivo in termini di distorsione dell'immagine (come invece era previsto sull'articolo), ma non è noto il margine di miglioramento ottenibile completando la calibrazione.

8.2 Sviluppi futuri

Possibili sviluppi futuri per questo progetto comprendono:

- Implementazione completa delle modifiche descritte nell'articolo, includendo quindi anche gli inverted residual layer.
- Completamento del training con batch size aumentato, per migliorare la stabilità, e su 600 epoch, per effettuare un vero confronto col precedente metodo.
- Ulteriori test del modello su diversi dataset, nello specifico COCO e CelebA; per quest'ultimo, implementare un sistema di traduzione da tag (presenti nel dataset) a caption (attese invece dall'encoder per il testo in DualAttnGAN).
- Analisi e confronto dei risultati delle due reti mediante una o più metriche in grado di misurare la corrispondenza tra il testo preso in input e l'immagine generata, nello specifico la metrica di R precision utilizzata all'interno dell'articolo relativo a AttnGAN [15].

Bibliografia

[1] *A simple explanation of the Inception Score*. URL: https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a.

[2] Jorge Agnese et al. "A Survey and Taxonomy of Adversarial Neural Networks for Text-to-Image Synthesis". In: (2019). URL: https://arxiv.org/pdf/1910.09399.pdf.

[3] Yali Cai et al. "DualAttn-GAN: Text to Image Synthesis with Dual Attentional Generative Adversarial Network". In: (2019). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8930532.

[4] *Caltech-UCSD Birds 200*. URL: http://www.vision.caltech.edu/visipedia/CUB-200.html.

[5] Ian Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: (2016). URL: https://arxiv.org/pdf/1701.00160.pdf.

[6] "Improved Techniques for Training GANs". In: (2016). URL: https://arxiv.org/pdf/1606.03498.pdf.

[7] *PyTorch*. URL: https://pytorch.org/.

[8] *Pytorch implementation for reproducing AttnGAN results*. URL: https://github.com/taoxugit/AttnGAN.

[9] *Repository github contenente la nostra implementazione di DualAttnGAN*. URL: https://github.com/mattrent/AttnGAN.

[10] "Self-Attention Generative Adversarial Networks". In: (2018). URL: https://arxiv.org/pdf/1805.08318.pdf.

[11] *StackGAN-inception-model*. URL: https://github.com/hanzhanggit/StackGAN-inception-model.

[12] "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks". In: (2017). URL: https://arxiv.org/pdf/1612.03242.pdf.

[13] *Two time-scale update rule for training GANs*. URL: https://github.com/bioinf-jku/TTUR.

[14] *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[15] Tao Xu et al. "AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks". In: (2018). URL: https://arxiv.org/pdf/1711.10485.pdf.

Lista delle figure