



Politecnico
di Bari



Deep Learning

Dipartimento di Ingegneria Elettrica e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Artificial Intelligence and Data Science

PikaPikaGen: Generative Synthesis of Pokemon Sprites from Textual Descriptions

Professore
Prof. Vito Walter Anelli

Studente
Valerio Salvatore Di Maggio

Indice

1	Introduzione	2
1.1	Contesto e obiettivi	2
1.2	Architettura proposta	2
1.3	Dataset e valutazione	3
1.4	Live Demo	3
2	Dataset e Pre-processing	4
2.1	Suddivisione dei dati	4
3	Metodologia sperimentale e Modello	6
3.1	Configurazione sperimentale	6
3.2	Metriche di valutazione	6
3.3	Architettura del modello	7
3.3.1	Text Encoder	7
3.3.2	Image Decoder	7
4	Sperimentazione	9
4.1	Esperimento 1: Modello di baseline con Loss L1	9
4.1.1	Configurazione	9
4.1.2	Risultati e analisi	9
4.2	Esperimento 2: Edge-aware e perceptual loss	15
4.3	Esperimento 3: Data augmentation	19
4.4	Esperimento 4: DCGAN condizionata	24
4.5	Esperimento 5: Architettura a due stadi (ispirata a AttnGAN)	29
4.5.1	Risultati e analisi	31
4.6	Valutazione finale e scelta del modello	35
5	Conclusioni e sviluppi futuri	37
5.1	Sintesi del percorso e risultati finali	37
5.2	Sviluppi futuri	37

Capitolo 1

Introduzione

1.1 Contesto e obiettivi

La sintesi di immagini da descrizioni testuali (*Text-to-Image*) è un compito fondamentale nell’intersezione tra Computer Vision e Natural Language Processing, il cui scopo è generare immagini fotorealistiche o stilizzate a partire da un input semantico. Questo progetto, denominato **PikaPikaGen**, affronta tale sfida nel dominio specifico della generazione di sprite di Pokémon. L’obiettivo principale è sviluppare un modello generativo in grado di sintetizzare uno sprite visivamente coerente con una descrizione testuale che ne specifichi aspetto, tipo e caratteristiche.

Questa relazione documenta il processo di sviluppo iterativo del modello. Si partirà da un’architettura di baseline per poi descrivere una serie di esperimenti mirati a migliorarne progressivamente i componenti, analizzandone di volta in volta i risultati.

1.2 Architettura proposta

Per raggiungere questo scopo, si propone un’architettura basata su un modello **Encoder-Decoder** con un meccanismo di **Attention**. L’architettura è così composta:

- **Encoder Testuale:** Un modello Transformer processa la descrizione in input. Il suo strato di embedding viene inizializzato con vettori pre-addestrati **bert_mini** (256 dimensioni) e successivamente ottimizzato (fine-tuning) durante l’addestramento per adattarsi al lessico specifico del dominio.
- **Decoder convoluzionale (generatore):** Una rete neurale convoluzionale (CNN) genera l’immagine finale. Utilizzando strati di convoluzione trasposta, il decoder esegue un upsampling da un vettore latente fino a raggiungere la dimensione dell’immagine di output.
- **Meccanismo di attention:** Unisce l’encoder e il decoder, consentendo a quest’ultimo di pesare dinamicamente l’importanza delle parole nella descrizione durante la sintesi delle corrispondenti regioni dell’immagine.

1.3 Dataset e valutazione

Il modello sarà addestrato e valutato utilizzando il dataset pubblico “The Pokemon Dataset”, una risorsa che include sprite ad alta risoluzione con canale alfa e descrizioni testuali per ogni Pokémon. La performance del modello sarà valutata sia qualitativamente, attraverso l’ispezione visiva delle immagini generate, sia quantitativamente, tramite metriche di ricostruzione. Il risultato finale del progetto includerà un’analisi critica del modello e una demo interattiva basata su Gradio per la generazione live di sprite.

1.4 Live Demo

È stata realizzata una demo interattiva del progetto, ospitata sulla piattaforma Hugging Face Spaces. Questo servizio permette di creare e condividere pubblicamente applicazioni di machine learning, rendendole accessibili tramite un’interfaccia web.

La demo consente di interagire direttamente con il modello per generare nuovi sprite in tempo reale ed è accessibile al seguente indirizzo:

huggingface.co/spaces/Val-2/PikaPikaGen

Capitolo 2

Dataset e Pre-processing

Il dataset utilizzato per l’addestramento viene preso da un repository pubblico su GitHub. Prima di scaricare lo ZIP del repository da GitHub e di estrarlo, si verifica la presenza del dataset in locale e si procede solo se non presente.

Per ogni Pokémon, vengono utilizzate due informazioni principali: la descrizione testuale, estratta dal file `pokemon.csv`, e l’immagine corrispondente, situata nella directory `small_images`. La classe `PokemonDataset` orchestra il caricamento e la trasformazione di questi dati.

Vi sono due pre-processing dei dati:

- **Testo:** La descrizione viene tokenizzata tramite il tokenizer di BERT-mini. Questo processo converte il testo in una sequenza di token a lunghezza fissa generando i tensori dei token e le relative attention mask.
- **Immagini:** Le immagini PNG, che possiedono un canale alfa per la trasparenza, vengono prima composte su uno sfondo bianco per ottenere un’immagine RGB standard. Successivamente, vengono ridimensionate a 256x256 pixel e, infine, normalizzate nell’intervallo [-1, 1]. Il ridimensionamento da 215x215 a 256x256 viene eseguito sia per evitare di dover avere parametri di padding e stride inusuali nell’ultimo layer di deconvoluzione, portando la dimensione a una potenza di 2, sia per standardizzare la dimensione delle immagini, in quanto alcune non sono di dimensioni 215x215. La normalizzazione, invece, viene eseguita per allineare la distribuzione dei dati di input con l’output della funzione di attivazione `tanh` del generatore.

Ogni campione del dataset è quindi una coppia composta da un testo tokenizzato e un’immagine normalizzata, pronta per essere fornita in input al modello.

2.1 Suddivisione dei dati

Per questo progetto, il dataset è stato partizionato in tre sottoinsiemi distinti: training set, validation set e test set.

La suddivisione è eseguita a livello di entità Pokémon, ovvero tutte le informazioni relative a un singolo Pokémon (sprite e descrizione) sono assegnate a uno solo dei tre set.

Considerando un totale di 898 Pokémon unici nel dataset, la ripartizione è stata effettuata secondo le seguenti proporzioni:

- **Training Set:** 70% dei Pokémon. Questo set è stato utilizzato per l’addestramento del modello.

- **Validation Set:** 10% dei Pokémon. Questo set è stato impiegato durante la fase di sviluppo per la valutazione qualitativa intermedia, la selezione degli iperparametri e le decisioni sulle modifiche architetturali.
- **Test Set:** 20% dei Pokémon. Questo set è stato mantenuto completamente separato e utilizzato una sola volta al termine di tutto il processo di sviluppo, per ottenere una valutazione quantitativa finale e imparziale del modello.

La riproducibilità della suddivisione dei dati è stata garantita dall'impiego di un seed fisso per l'operazione di shuffling e splitting del dataset. In questo modo, la composizione di training, validation e test set è mantenuta costante durante le varie fasi di sperimentazione.

Capitolo 3

Metodologia sperimentale e Modello

3.1 Configurazione sperimentale

- **Framework:** L'implementazione è basata su PyTorch.
- **Hardware:** L'addestramento viene eseguito su GPU NVIDIA, sfruttando l'interfaccia CUDA di PyTorch.
- **Ottimizzatore:** Viene utilizzato l'ottimizzatore Adam (`torch.optim.Adam`) per il generatore, con come parametri iniziali un learning rate di 2×10^{-4} e parametri beta impostati a (0.5, 0.999).
- **Dimensione del Batch:** Viene utilizzata una dimensione del batch di 16.

3.2 Metriche di valutazione

La performance dei modelli viene valutata tramite una combinazione di metriche quantitative e analisi qualitativa:

- **L1 Loss:** Misura la Mean Absolute Error tra i pixel dell'immagine generata e quella reale, fornendo una valutazione diretta della fedeltà a livello di pixel.
- **SSIM (Structural Similarity Index Measure):** Valuta la somiglianza strutturale tra due immagini, confrontando luminanza, contrasto e struttura. La loss è calcolata come $1 - \text{SSIM}$.
- **Perceptual Loss (VGG19):** Invece di confrontare i pixel direttamente, questa loss calcola la distanza L1 tra le mappe di attivazione estratte da vari livelli di una rete VGG19 pre-addestrata su ImageNet. Questo favorisce immagini percettivamente più simili a quelle reali.
- **Sobel Loss:** Loss edge-aware che incoraggia la somiglianza dei bordi tra l'immagine generata e quella reale, calcolando la loss L1 tra le rispettive mappe dei gradienti ottenute con l'operatore di Sobel.
- **Analisi Qualitativa:** Vengono generate periodicamente delle griglie di confronto che mostrano le immagini reali accanto a quelle generate dallo stesso prompt, permettendo una valutazione visiva della qualità e della coerenza.

3.3 Architettura del modello

Viene usato un modello generativo text-to-image progettato per sintetizzare immagini a partire da descrizioni testuali. L'architettura è composta da due moduli principali che operano in cascata: un **Text Encoder**, che interpreta il prompt, e un **Image Decoder**, che genera l'immagine.

3.3.1 Text Encoder

Il Text Encoder ha il compito di trasformare il testo in input in una serie di rappresentazioni vettoriali che guideranno il processo di generazione. Questo modulo è composto da due stadi:

1. **Embedding Layer**: I token del testo vengono inizialmente convertiti in vettori densi utilizzando lo strato di embedding del modello BERT pre-addestrato ([prajjw1/bert-mini](#)).
2. **Transformer Encoder**: Gli embedding vengono poi processati da una pila di 4 strati di Transformer. Da notare che questo stadio sfrutta la maschera di attenzione fornita dal tokenizer per ignorare esplicitamente i token di padding.

L'output è una sequenza di vettori, uno per ogni token del prompt, che cattura il significato del testo nel suo contesto.

3.3.2 Image Decoder

L'Image Decoder è il componente generativo che, partendo da un vettore di rumore casuale e dalle feature testuali, sintetizza l'immagine finale. L'architettura del decoder è progettata per integrare l'informazione testuale in diversi punti.

1. Vettore di contesto testuale Prima di iniziare la generazione, il decoder calcola un singolo vettore di contesto globale in questo modo:

- Il modello calcola prima uno score di "importanza" per ogni parola dell'output dell'encoder testuale tramite una rete feed-forward.
- Applica poi la maschera di attenzione per annullare i punteggi dei token di padding.
- Infine, normalizza questi punteggi con una funzione Softmax per ottenere dei pesi.

Il vettore di contesto finale è una media pesata delle feature di tutte le parole, garantendo che sia basato solo sul contenuto effettivo del prompt.

2. Proiezione iniziale Il vettore di contesto globale viene concatenato con un vettore di rumore gaussiano. Il risultato è proiettato da una feed-forward per formare una mappa di feature iniziale di piccole dimensioni spaziali (4×4), ma con un alto numero di canali (256). Questo blocco include anche normalizzazione e attivazione **LeakyReLU** per stabilizzare l'inizio del processo di generazione.

3. Blocchi di upsampling con attenzione Il cuore del decoder è una catena di sei **DecoderBlock**, ognuno responsabile di aumentare la risoluzione dell’immagine e di integrarla con il contesto testuale.

- **Cross-Attention e fusione:** Nei primi tre **DecoderBlock** (quelli a bassa risoluzione), un meccanismo di cross-attention permette alle feature dell’immagine di consultare le feature del testo. Questo arricchisce l’immagine con le informazioni semantiche più pertinenti fornite dal prompt. Le feature originali e quelle arricchite dal testo vengono poi fuse insieme tramite una convoluzione 1×1 .
- **Convoluzione trasposta:** Subito dopo, una convoluzione trasposta raddoppia le dimensioni spaziali della mappa di feature (es. da 8×8 a 16×16). Infine, il blocco applica normalizzazione e attivazione **LeakyReLU**.

4. Strato finale Al termine della catena di upsampling, una convoluzione finale trasforma la mappa di feature ad alta risoluzione (con 16 canali) nell’immagine a 3 canali (RGB). Un’attivazione **Tanh** normalizza i valori dei pixel nell’intervallo $[-1, 1]$.

Capitolo 4

Sperimentazione

4.1 Esperimento 1: Modello di baseline con Loss L1

Il primo esperimento è stato condotto per stabilire una baseline di performance, utilizzando la configurazione architettonale e la funzione di loss consigliate nelle istruzioni per il progetto. L'obiettivo era valutare le capacità del modello prima di introdurre modifiche più complesse.

4.1.1 Configurazione

La scelta della funzione di perdita è ricaduta sulla **Loss L1**, nota anche come Mean Absolute Error (MAE). Tale funzione calcola la media della somma delle differenze assolute tra i pixel dell'immagine generata e quelli dell'immagine reale.

La giustificazione di questa scelta, rispetto alla loss L2, è data dall'essere preferita in compiti di generazione di immagini poiché tende a produrre risultati meno sfocati, penalizzando in modo lineare le differenze di pixel.

Inoltre il training viene fermato quando non si nota un miglioramento sostanziale delle immagini per un certo numero di epoche.

4.1.2 Risultati e analisi

La valutazione del modello è stata eseguita sia a livello quantitativo che qualitativo.

Analisi Quantitativa L'andamento della loss L1, illustrato in Figura 4.1, rivela un possibile caso di overfitting. Mentre la curva di training mostra una diminuzione costante, indicando l'apprendimento dei dati di addestramento, la curva di validazione si stabilizza dopo un calo iniziale e rimane stabile. L'ampio divario tra le due curve dimostra che il modello non riesce a generalizzare la conoscenza a dati mai visti.

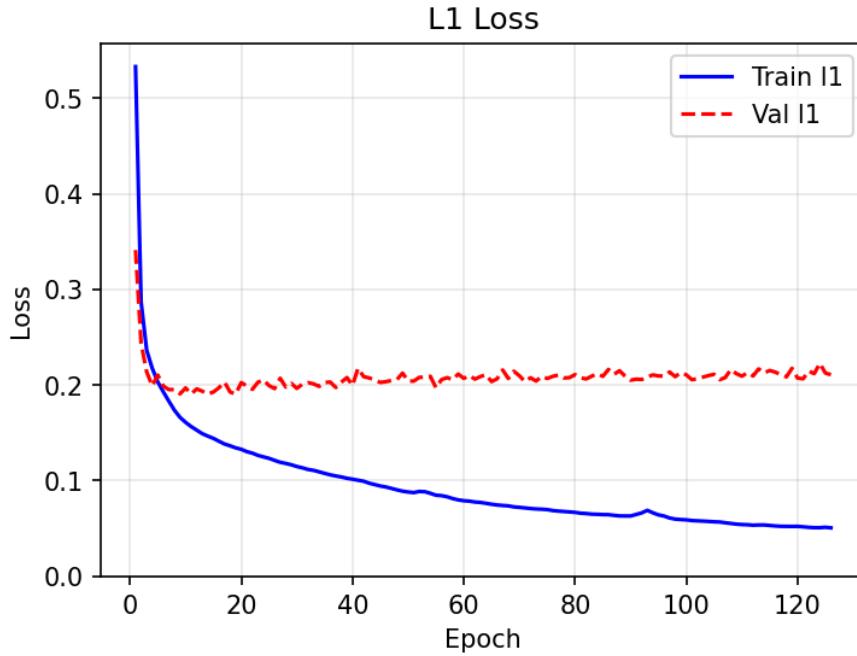


Figura 4.1: Loss L1 - Esperimento 1

Analisi Qualitativa L’ispezione visiva degli output generati rivela i limiti intrinseci di questo approccio di base.

- **Risultati sul Training Set:** Le immagini generate per i campioni del training set, sebbene riconoscibili nelle loro forme e colori generali, presentano una diffusa sfocatura. I dettagli fini sono assenti e i contorni appaiono ”morbidi”, un possibile segno che il modello apprende una media delle possibili rappresentazioni piuttosto che una singola istanza nitida.
- **Risultati sul Validation Set:** I limiti del modello diventano ancora più evidenti sui dati di validazione. Le immagini generate sono molto sfocate e prive di definizione. I contorni, sia esterni che interni, risultano quasi inesistenti, fondendosi in transizioni di colore graduali. L’output non rappresenta un Pokémon riconoscibile, ma appare piuttosto come una macchia di colore quasi astratta.

Epoch 125 - Train Comparison

	#701: A Dedenne's whiskers pick up electr...	#90: Its shell is extremely hard. It can...	#418: It floats using its well-developed ...	#30: Nidoqueen is better at defense than...
Real				
Generated				

Figura 4.2: Esempi di generazioni di Pokemon nel training set - Esperimento 1

Epoch 125 - Val Comparison				
	#6: When it retracts its long neck into...	#11: In battle, it flaps its wings at gr...	#300: Delcatty prefers to live an unfette...	#692: After using the feelers on its over...
Real				
Generated				

Figura 4.3: Esempi di generazioni di Pokemon nel validation set - Esperimento 1

Analisi delle mappe di attenzione Per comprendere più a fondo il comportamento del modello, è stata analizzata la visualizzazione delle mappe di attenzione, che mostrano come il modello correla le parole del prompt alle regioni dell'immagine generata. L'esempio mostra la generazione di Dedenne (#701), un campione del training set e di Squirtle (#7), campione del validation set.

L'analisi rivela due aspetti chiave:

1. **Contesto iniziale globale:** Il grafico in alto mostra come il modello calcola il vettore di contesto generale prima di iniziare la generazione. Si nota un picco di attenzione sul token ‘##ne‘ della parola ”dedenne“, indicando che il modello identifica correttamente il soggetto principale del prompt.

2. Cross-Attention nel decoder: Le griglie inferiori mostrano le mappe di attenzione a diverse risoluzioni. Analizzando in particolare l'ultimo strato (16x16), il modello dimostra di aver associato parole a particolari zone dell'immagine:

- I token che compongono "Dedenne" (`de`, `##den`) attivano l'area corrispondente all'incirca al corpo dello sprite.
- Anche la parola `whiskers` (baffi) porta il modello a concentrarsi sulla regione corretta del muso.
- Sull'altro Pokemon invece si può notare `neck` che si concentra sulla zona dove si potrebbe trovare il collo e `shell` che attiva all'incirca tutta la zona del corpo.
- Inoltre in entrambe le visualizzazioni, in corrispondenza del token `..`, si può osservare l'attivazione dell'area corrispondente allo sfondo bianco dell'immagine.

Questa analisi dimostra che, almeno in alcuni casi come questi appena mostrati, il modello è in grado di creare connessioni semanticamente corrette tra testo e immagine quando opera sui dati di training. Tuttavia, analizzando altri casi, sia in queste visualizzazioni, sia per altre immagini del validation set, si può dedurre che questa capacità non è stata esattamente generalizzata, almeno non per tutti i casi. È possibile che il modello abbia imparato a mappare con successo le descrizioni per alcuni esempi specifici, senza però sviluppare una regola applicabile a dati mai visti, un comportamento che si allinea con l'ipotesi di overfitting.

Epoch 125: Attention Visualization for Pokémon #701 (Train)

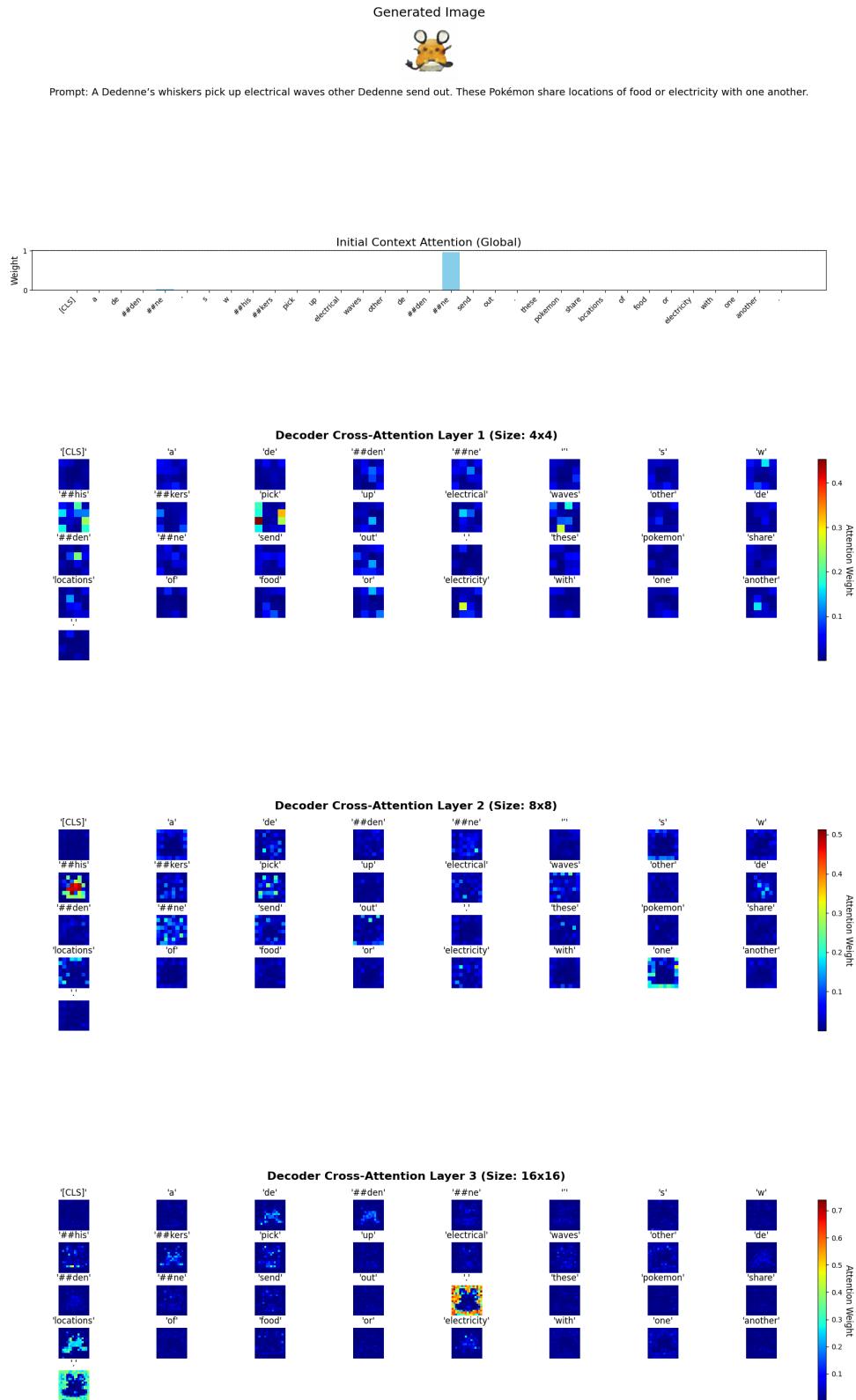


Figura 4.4: Mappe di attenzione per Pokémon #701 (training set) - Esperimento 1

Epoch 125: Attention Visualization for Pokémon #6 (Val)

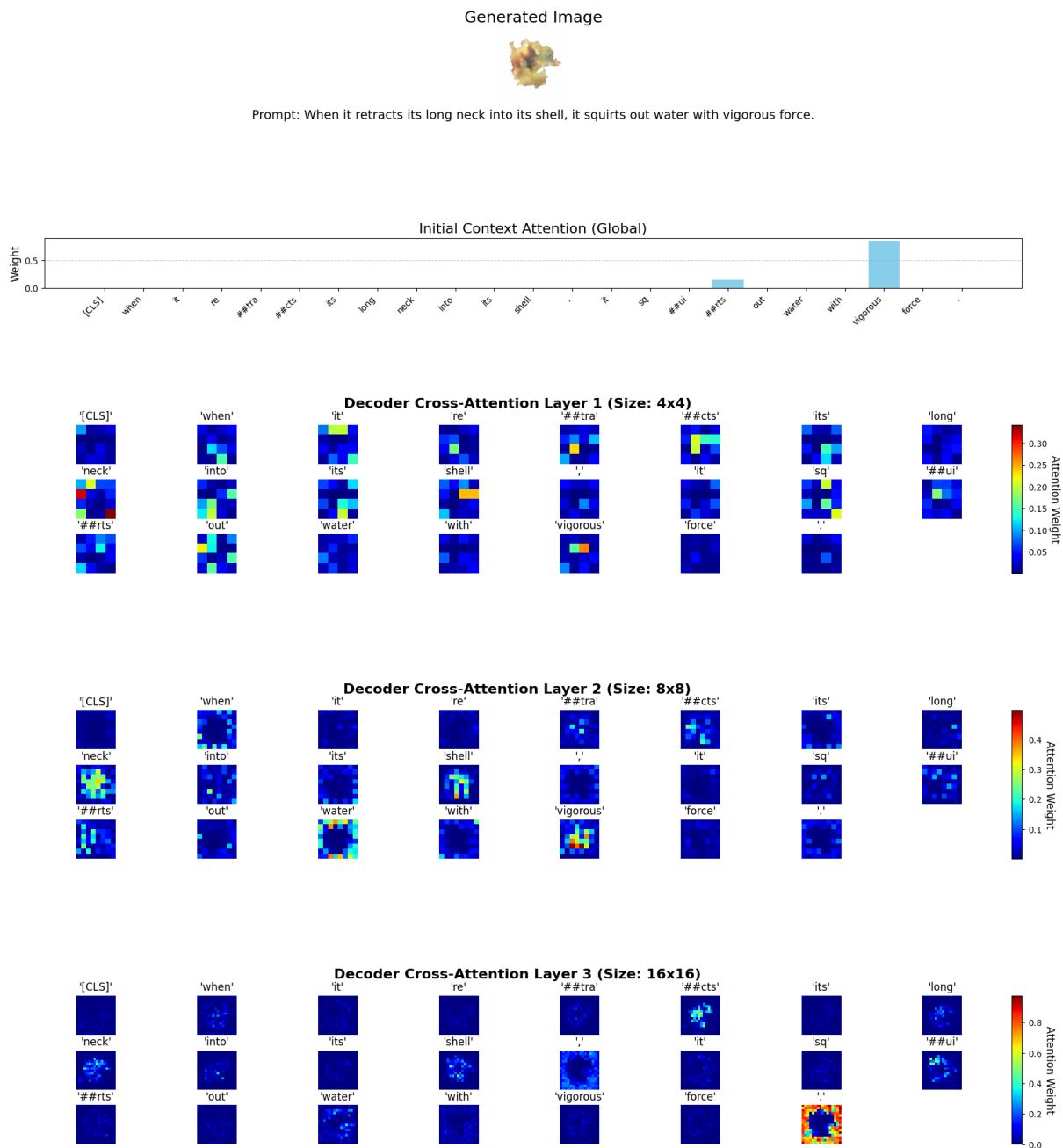


Figura 4.5: Mappe di attenzione per Pokémon #7 (validation set) - Esperimento 1

4.2 Esperimento 2: Edge-aware e perceptual loss

Per affrontare i limiti trovati nel primo esperimento, si introduce una funzione di loss composita, integrando anche una loss edge-aware (SobelLoss) e una perceptual loss (VGG-PerceptualLoss).

Configurazione

La nuova funzione di perdita totale è una combinazione pesata di tre componenti:

1. **Loss L1**: Mantenuta per garantire la fedeltà cromatica di base.
2. **VGG Perceptual Loss**: Questa loss non confronta direttamente i pixel, ma le attivazioni di feature map estratte da una rete pre-addestrata (in questo caso, VGG19). Confrontando le rappresentazioni ad alto livello, si incoraggia il modello a generare immagini che siano semanticamente e percettivamente simili a quelle reali, andando oltre il puro valore dei pixel.
3. **Sobel Loss**: Una *edge-aware loss* che opera applicando il filtro di Sobel sia all'immagine generata che a quella reale. Il filtro calcola il gradiente dell'immagine, evidenziandone i bordi. La loss confronta quindi le mappe dei gradienti, forzando il modello a generare contorni nitidi nelle stesse posizioni dell'immagine target.

La loss totale è definita come la somma di queste tre loss moltiplicate per i rispettivi coefficienti.

I coefficienti λ sono stati determinati empiricamente tramite un processo di *trial and error*. La configurazione finale usata è la seguente: $\lambda_{l1} = 1.0$, $\lambda_p = 0.01$ e $\lambda_s = 2.0$. Si è scelta questa configurazione in quanto, ad esempio, un valore di λ_s troppo elevato produceva bordi marcati ma a discapito dell'informazione di colore, mentre un λ_1 più alto riportava ai risultati sfocati dell'esperimento precedente.

Risultati e analisi

Analisi quantitativa I grafici in Figura 4.6 mostrano l'andamento delle singole componenti della loss e della loss totale. Similmente al primo esperimento, si osserva un possibile fenomeno di **overfitting** in tutte le loss. La perceptual e la sobel loss, dopo un breve calo iniziale, tendono a risalire leggermente.

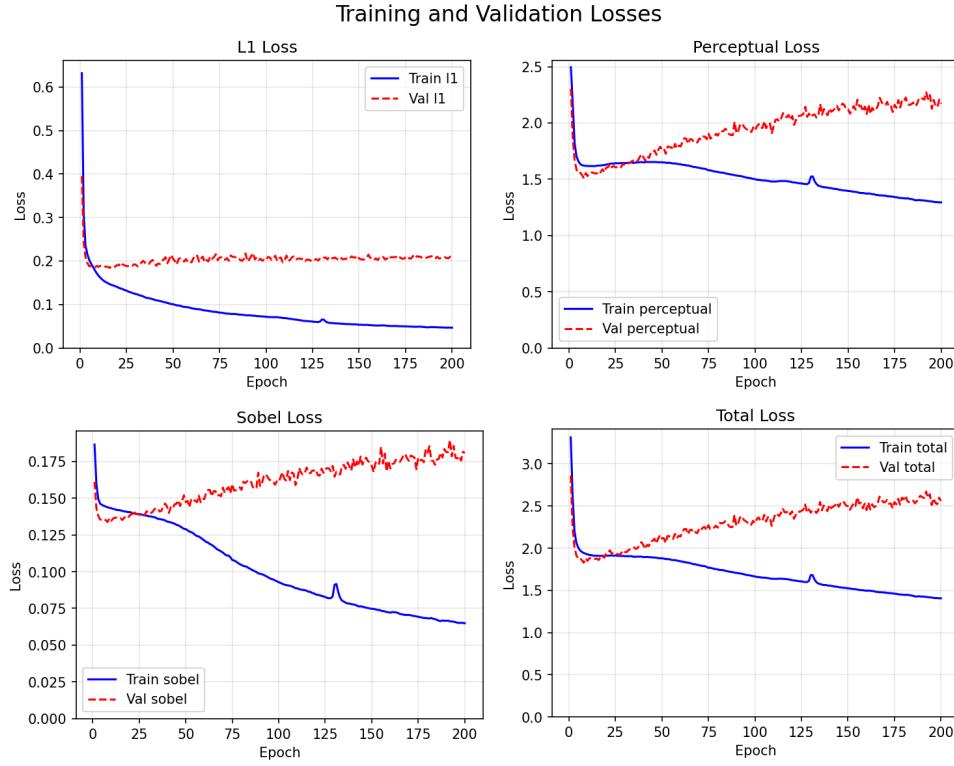


Figura 4.6: Andamento delle loss - Esperimento 2

Analisi qualitativa L’analisi delle immagini generate rivela un risultato migliorato da un punto di vista, ma non ancora del tutto soddisfacente. Da un lato, il problema della sfocatura è stato parzialmente mitigato, in particolare nei Pokémon presenti nel training set: le immagini non sono più solo macchie di colore, ma presentano texture ad alta frequenza, simili a bordi. Questo conferma che la Sobel Loss ha avuto effetto, spingendo il modello a produrre output con gradienti più marcati.

Tuttavia, il risultato non è molto soddisfacente. Il modello ha imparato cosa sia un bordo, ma non come strutturare i bordi per formare una figura coerente.



Figura 4.7: Esempi di generazioni di Pokemon nel training set - Esperimento 2



Figura 4.8: Esempi di generazioni di Pokemon nel validation set - Esperimento 2

Analisi delle Mappe di Attenzione L’analisi delle mappe di attenzione per lo stesso Pokémon mostra un comportamento diverso rispetto all’esperimento 1. Le mappe appaiono più ”diffuse”. Parole chiave come `**whiskers**` , che nel primo esperimento si concentravano su aree specifiche, ora attivano una serie di punti sparsi su tutta la sagoma dell’immagine.

Questo comportamento potrebbe spiegare gli artefatti visivi osservati. Per soddisfare la loss Sobel, il modello potrebbe aver imparato a generare dettagli locali (bordi e texture) in modo diffuso, anche tramite i layer di attenzione.

Epoch 200: Attention Visualization for Pokémons #701 (Train)

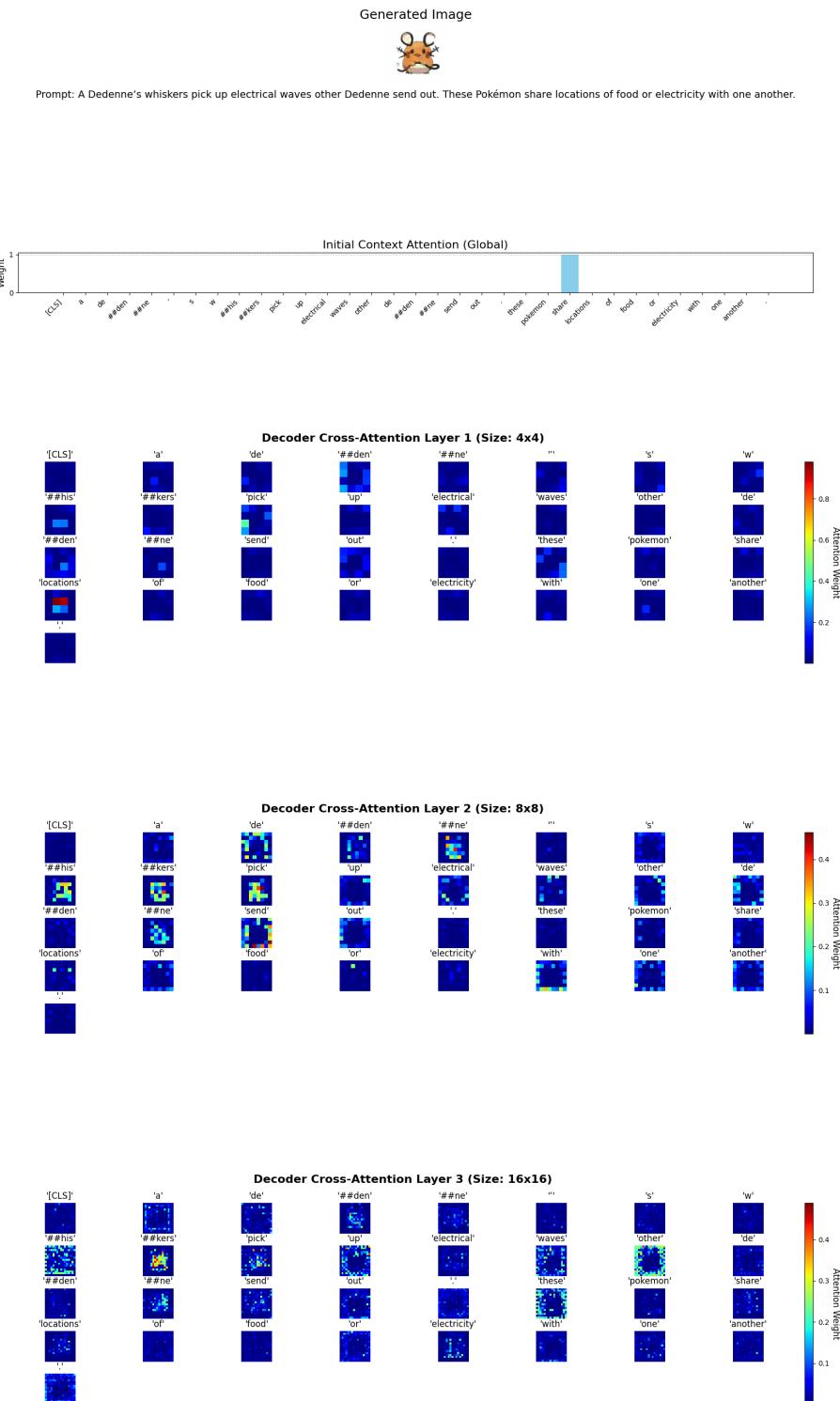


Figura 4.9: Mappe di attenzione per Pokémons #701 (training set) - Esperimento 2

4.3 Esperimento 3: Data augmentation

Configurazione e giustificazione

Il secondo esperimento, introducendo loss strutturali, ha permesso al modello di generare contorni definiti, un passo avanti rispetto alle immagini sfocate del primo esperimento. Tuttavia, esiste ancora il problema del possibile overfitting derivante dall'analisi quantitativa dei precedenti esperimenti. Per affrontare direttamente questo problema si introduce una pipeline di **Data Augmentation**, progettata per forzare il modello a generalizzare meglio.

La pipeline di augmentation è stata inserita all'interno del flusso di pre-processing, seguendo l'ordine *Resize* → *Augmentation* → *Normalization in $[-1, 1]$* .

Nella pipeline vengono applicate le seguenti trasformazioni:

- **Trasformazioni geometriche:** Per garantire l'invarianza alla posizione e all'orientamento.

`RandomHorizontalFlip` con $p = 0.5$

`RandomAffine` con `degrees=15, translate=(0.1, 0.1), scale=(0.9, 1.1)`

- **Trasformazioni relative ai colori:** Aggiunta per tentare di disaccoppiare la forma e la struttura dal colore. Varia la tonalità e la luminosità dell'immagine, portando modello a non memorizzare le esatte palette cromatiche, ma a concentrarsi sulla struttura.

`ColorJitter` con parametri `brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1`

- **Trasformazione RandomErasing:** Nasconde parti casuali dell'immagine, sostituendole con rumore casuale, con probabilità $p = 0.25$.

In Figura 4.10 vengono mostrati degli esempi di queste trasformazioni applicate casualmente allo stesso Pokemon.

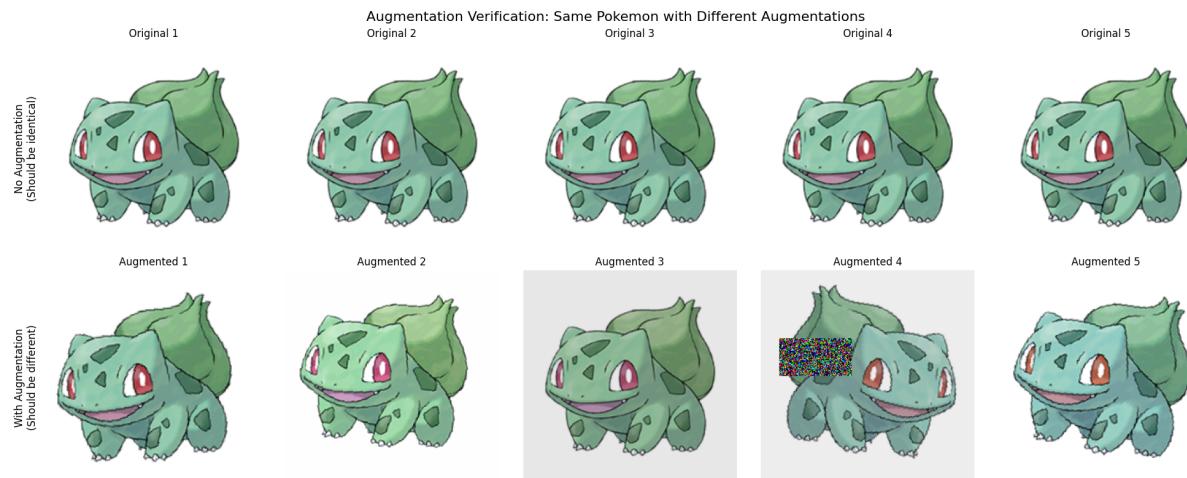


Figura 4.10: Esempi di data augmentation - Esperimento 3

Inoltre vengono usate le loss L1 con $\lambda = 1$ e SobelLoss con $\lambda = 2$.

Risultati e Analisi

Il processo di addestramento per questo esperimento ha richiesto un intervento correttivo in corso d'opera. Si è osservato che, nonostante più di 100 epoche di training, il modello faticava a convergere in modo significativo, con le immagini generate sia per il training che per la validazione che apparivano come macchie di colore non molto distinte.



Figura 4.11: Figure non ancora definite - Epoca 110

Si è presupposto che questo comportamento sia stato derivante da una pipeline di augmentation eccessivamente aggressiva nella modifica delle immagini. Di conseguenza, a partire dall'epoca 110, i parametri delle trasformazioni sono stati resi meno stringenti nel seguente modo:

- **RandomAffine** con parametri `degrees=10, translate=(0.05, 0.05), scale=(0.95, 1.05)`
- **RandomErasing** con probabilità $p = 0.15$

Analisi quantitativa I grafici della funzione di loss in figura 4.12 riflettono chiaramente questo intervento. Nella prima fase (epoche 0-110), le curve mostrano un andamento stagnante. Il cambio di parametri all'epoca 110 causa un'immediata e visibile variazione nelle loss, che hanno subito una breve discesa.

Questo miglioramento non è durato per molte epoche e dopo poco le curve hanno ripreso l'andamento stagnante antecedente. Dunque si può notare ancora fatica ad apprendere da parte del modello.

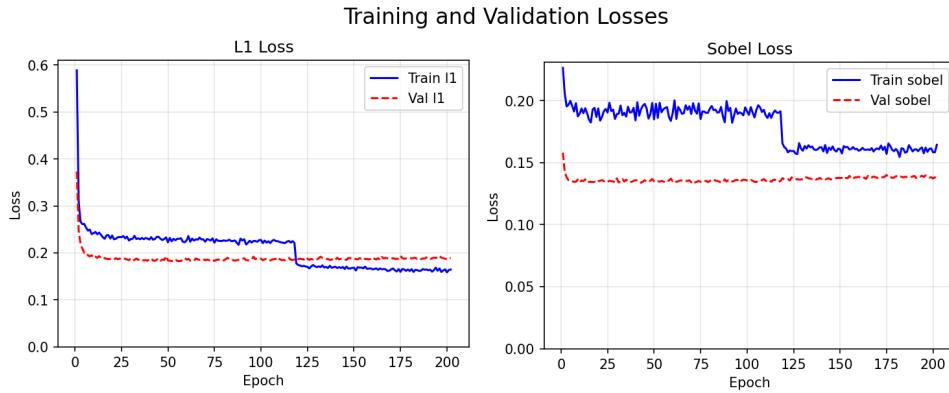


Figura 4.12: Andamento delle loss - Esperimento 3

Analisi qualitativa L’analisi qualitativa delle immagini generate a fine addestramento mostra risultati ancora modesti. Sebbene la forma generale dei Pokémon sia vagamente accennata, le immagini soffrono di una notevole sfocatura sui bordi esterni e di una mancanza di dettagli interni. È inoltre visibile un artefatto ricorrente: un pattern ad alta frequenza spaziale, simile a delle striature verticali. Questo è un probabile effetto collaterale della Sobel Loss, per cui il modello introduce gradienti per minimizzare la loss senza però riuscire a organizzarli in contorni strutturalmente coerenti.



Figura 4.13: Esempi di generazioni di Pokemon nel training set - Esperimento 3

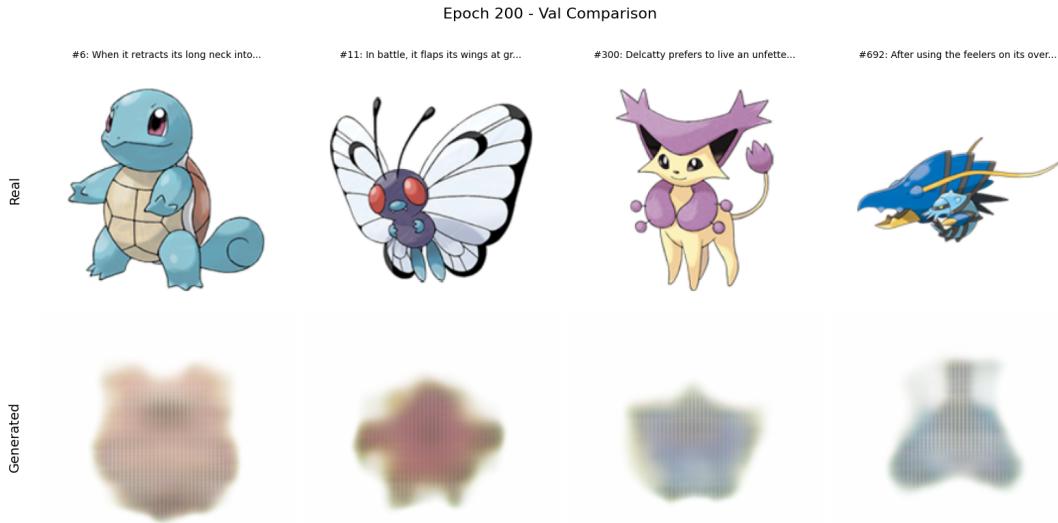


Figura 4.14: Esempi di generazioni di Pokemon nel validation set - Esperimento 3

Analisi delle Mappe di Attenzione L’analisi delle mappe di attenzione (Figura 4.15) mostra un’ulteriore evoluzione rispetto all’esperimento precedente. Dopo l’introduzione della data augmentation, le mappe di cross-attention, in particolare nello strato finale (16x16), appaiono più localizzate e meno rumorose rispetto a quelle dell’esperimento 2.

Parole come ‘##ne’ (di Dedenne) o ‘electrical’ ora attivano aree più definite. Questo suggerisce che il modello sta tentando nuovamente di associare i concetti semantici a regioni spaziali specifiche. D’altra parte, queste aree di attenzione sono ancora molto grossolane e ”a blocchi”, il che potrebbe contribuire alla mancanza di dettagli fini e alla sfocatura generale dell’immagine generata.

Epoch 200: Attention Visualization for Pokémon #701 (Train)

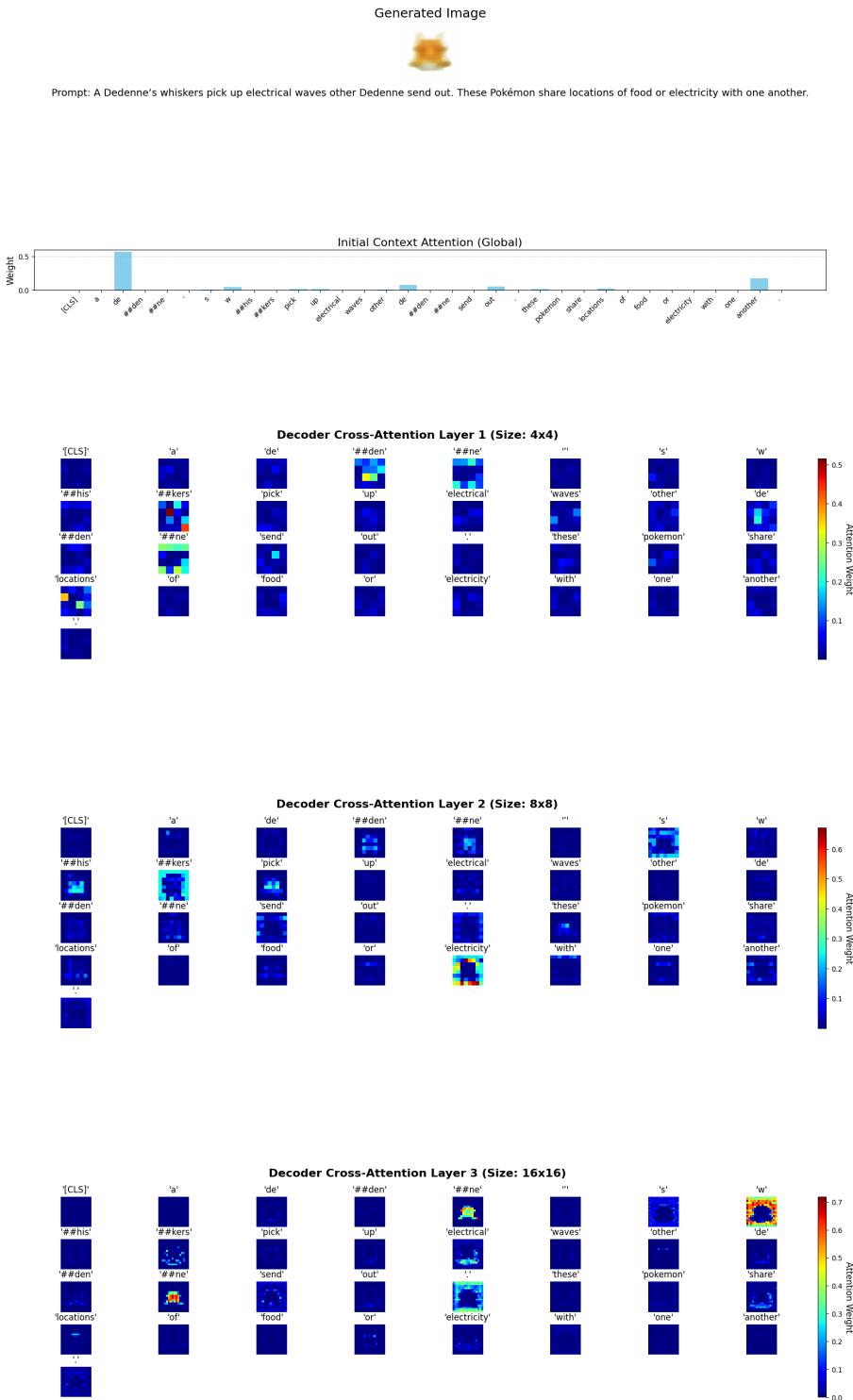


Figura 4.15: Mappe di attenzione per Pokemon #701 (training set) - Esperimento 3

4.4 Esperimento 4: DCGAN condizionata

Dati i limiti emersi negli esperimenti precedenti, dove l'ottimizzazione di loss come L1 e Sobel non è riuscita a produrre immagini strutturalmente coerenti, si è deciso di cambiare approccio. In questo esperimento, si adotta un paradigma di Generative Adversarial Network (GAN), in cui il modello impara implicitamente una funzione di loss attraverso un gioco competitivo tra due reti. Nello specifico, si implementa una **Deep Convolutional GAN Condizionata (cDC-GAN)**, che unisce i principi delle architetture convoluzionali profonde delle DC-GAN [3] con il meccanismo di condizionamento testuale proposto in [2].

Configurazione e giustificazione

L'architettura della cDC-GAN è composta da un generatore e un discriminatore. Il discriminatore è stato progettato con due percorsi di elaborazione paralleli:

- **Percorso per le immagini:** Una CNN standard che processa l'immagine in input. È composta da una serie di strati **Conv2d**, **BatchNorm2d** e attivazioni **LeakyReLU**, che progressivamente riducono la dimensione spaziale dell'immagine per estrarre un vettore di feature visive.
- **Percorso per il testo:** Un'architettura che elabora la descrizione testuale. Le feature del testo, estratte da un'istanza del text encoder separata da quella del generatore, vengono processate da un Multi-Layer Perceptron (MLP) per proiettarle in uno spazio compatibile con quello delle feature visive.

Il condizionamento avviene tramite la **concatenazione** dei vettori di feature provenienti dai due percorsi. Il vettore risultante viene quindi dato in input a un classificatore finale, un altro MLP, che produce un singolo valore di probabilità indicante se la coppia immagine-testo è reale o generata.

Inoltre, seguendo il paper originale, i layer di convoluzione e convoluzione trasposta vengono inizializzati da una distribuzione normale con media 0.0 e deviazione standard 0.02, mentre i layer di BatchNorm vengono inizializzati da una distribuzione normale con media 1.0 e deviazione standard 0.02, mentre i bias vengono inizializzati a 0. Per i layer rimanenti si lascia l'inizializzazione di default di PyTorch.

I primi tentativi di addestramento con questa architettura hanno rivelato una criticità: si è osservato che la loss del discriminatore crollava a zero entro le prime 5-10 epoche, arrestando di fatto ogni progresso. Questo fenomeno si verifica quando il compito per il discriminatore è troppo facile. Nelle fasi iniziali, il generatore produce output simili a rumore casuale, che il discriminatore impara a distinguere dalle immagini reali molto velocemente. Una volta raggiunto questo stato, il gradiente che dovrebbe guidare il generatore svanisce, rendendo l'addestramento inutile.

Per mitigare questo problema, sono state apportate le seguenti modifiche alla configurazione di training:

- **Riduzione del batch size:** Per l'addestramento è stato utilizzato un batch size di 8.
- **Loss L1 ausiliaria:** per evitare che il generatore si blocchi nelle prime epoche a causa di segnali poco utili dal discriminatore, si aggiunge un termine L1 alla loss del generatore con peso $\lambda = 1$.

- **Disattivazione dell'augmentazione del dataset:** È stato fatto ciò per evitare che il discriminatore apprenda gli artefatti generati dall'augmentazione e non impari davvero a distinguere immagini reali da quelle generate.

Risultati e analisi

Nonostante le modifiche apportate per stabilizzare l'addestramento, l'esperimento ha portato a risultati con alcuni progressi, anche se ancora non del tutto soddisfacenti.

Analisi quantitativa Si analizzano i due grafici delle funzioni loss, uno in Figura 4.16 che confronta la loss di generatore e discriminatore, e l'altro in Figura 4.17 che mostra l'andamento della loss L1.

- **Loss di generatore e discriminatore:** La curva della loss del discriminatore diminuisce rapidamente e si assesta su valori molto bassi, mentre quella del generatore, dopo una fase iniziale, tende a crescere costantemente. Questo andamento suggerisce uno sbilanciamento nella dinamica tra i due modelli: il discriminatore sta diventando progressivamente più forte del generatore, riuscendo a distinguere con troppa facilità le immagini reali da quelle generate; la crescita della loss del generatore indica una difficoltà sempre maggiore nell'ingannare il discriminatore.
- **Loss L1 ausiliaria:** Il grafico della loss L1 mostra che sia sul training set che sul validation set l'errore si stabilizza su un valore relativamente alto rispetto agli altri esperimenti senza augmentazione (circa 0.25), non mostrando una chiara tendenza alla diminuzione. Questo conferma che il modello non si sta più concentrando solo sul riprodurre una figura uguale pixel per pixel, ma anche tentando di creare un'immagine più realistica ma diversa dal target, portando a una loss L1 più alta.



Figura 4.16: Andamento delle loss di generatore e discriminatore - Esperimento 4

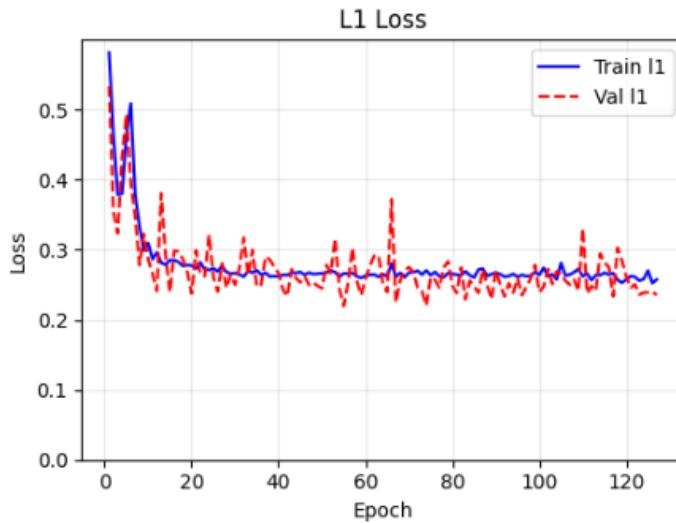


Figura 4.17: Andamento della loss L1 del generatore - Esperimento 4

Analisi qualitativa L’ispezione visiva delle immagini generate segna un progresso rispetto agli esperimenti precedenti. Il modello produce figure con una silhouette più definita e con colori che richiamano di più quelli dei Pokémon.

D’altra parte, le immagini generate non sono esattamente Pokémon riconoscibili. Inoltre, si nota una certa mancanza di varietà negli output: diverse immagini generate presentano strutture molto simili, sebbene con colori diversi. Pur non trattandosi di un mode collapse completo, questo indica una difficoltà del generatore nel rappresentare la diversità delle forme delle immagini target.

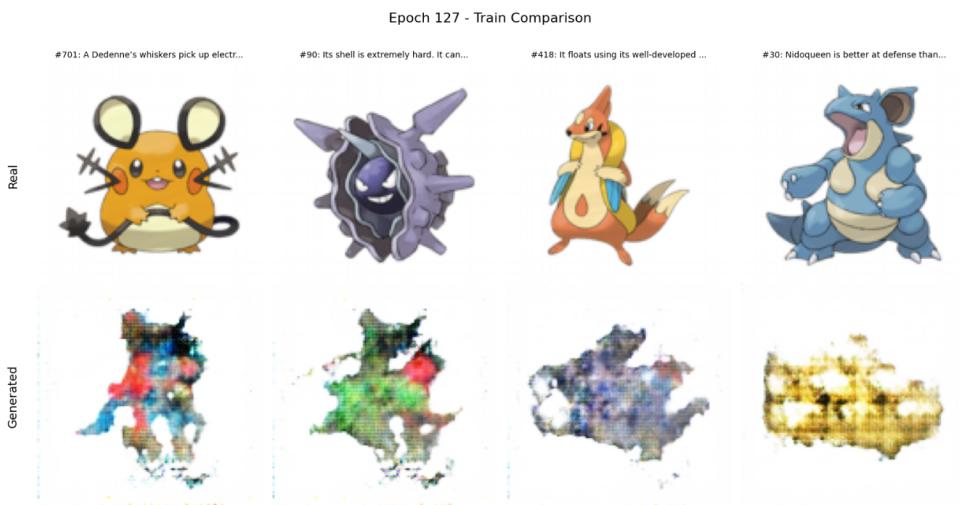


Figura 4.18: Esempi di generazioni di Pokemon nel training set - Esperimento 4



Figura 4.19: Altri esempi di generazioni di Pokemon nel training set - Esperimento 4

Analisi delle mappe di attenzione Analizzando le mappe di attenzione del modello in Figura 4.20, si può notare una potenziale criticità nel vettore di contesto iniziale: i pesi di attenzione assegnati a ciascun token del prompt sono uguali. Una seconda verifica, analizzando il vettore generato, ha poi confermato questa osservazione, notando che i pesi siano quasi uguali, non esattamente identici ma abbastanza da non far apparire alcuna differenza nella visualizzazione. Questo comportamento può essere interpretato principalmente in due modi:

1. **Strategia appresa ottimale:** Il modello ha appreso che il modo migliore di iniziare la generazione delle immagini è semplicemente avere una media non ponderata come vettore di contesto.
2. **Sintomo di vanishing gradient:** Una seconda interpretazione, più probabile, di questo fenomeno è l'essere un sintomo di *vanishing gradient*. Il meccanismo che calcola questi pesi di attenzione si trova all'inizio del decoder, molto distante dalla funzione di loss finale. Durante la backpropagation, il gradiente deve dunque attraversare numerosi strati per raggiungere e aggiornare il relativo modulo, percorso durante cui il gradiente potrebbe attenuarsi eccessivamente e di fatto non modificare in modo significativo i pesi.

Questo potrebbe essere anche una possibile causa della poca varietà degli output riscontrata durante l'analisi qualitativa.

Inoltre, rispetto ai precedenti esperimenti, le mappe di attenzione appaiono più rumorose e non c'è una corrispondenza chiara tra token e parte dell'immagine.

Epoch 120: Attention Visualization for Pokémon #6 (Val)

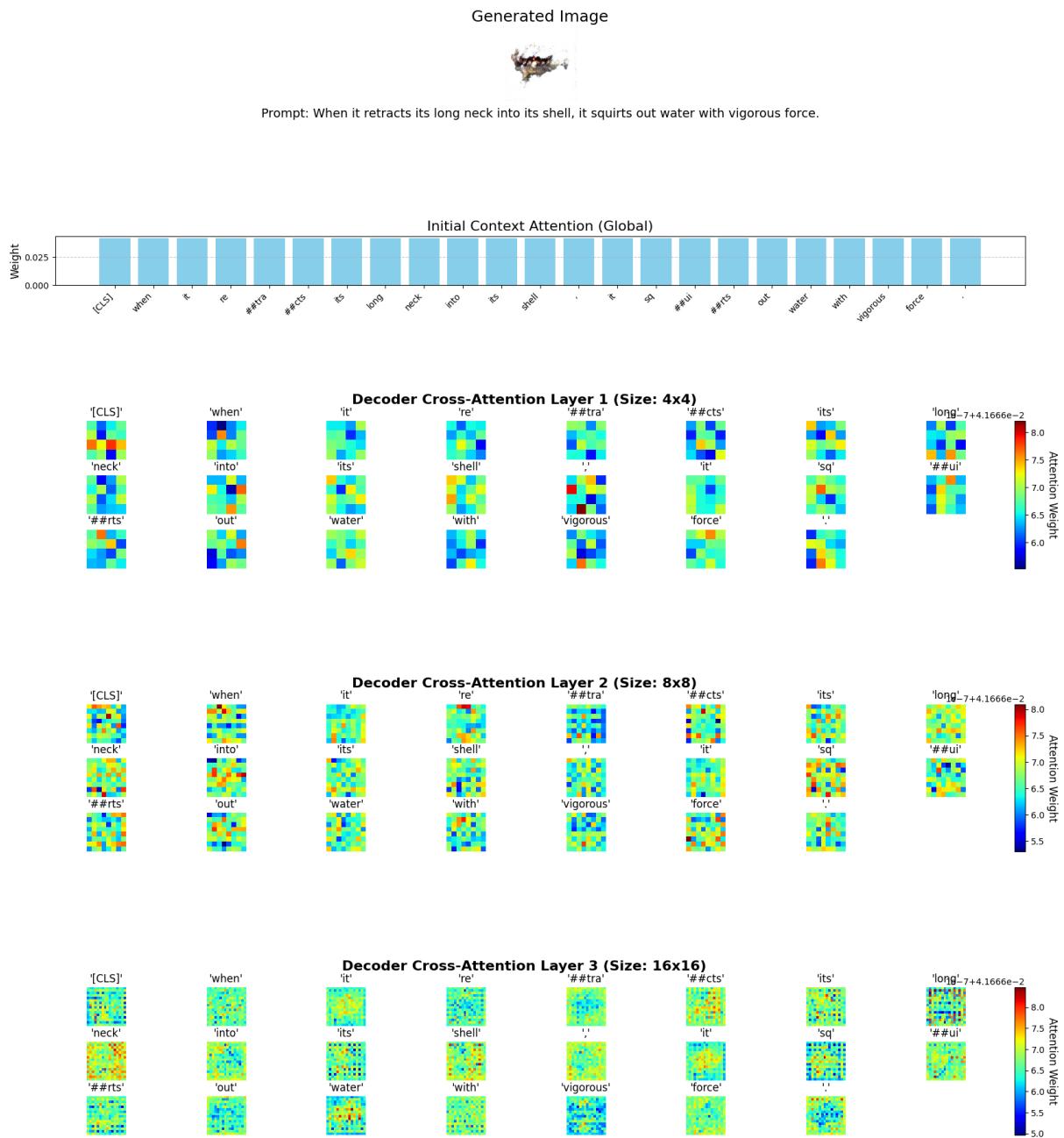


Figura 4.20: Mappe di attenzione per Pokémon #7 - Esperimento 4

4.5 Esperimento 5: Architettura a due stadi (ispirata a AttnGAN)

Dati i risultati dell'esperimento precedente, dove l'approccio con DC-GAN condizionata ha mostrato progressi ma anche varie criticità come lo sbilanciamento tra generatore e discriminatore, la poca varietà negli output e il sospetto di vanishing gradient, si è deciso di implementare una serie di modifiche architettonali per affrontare tali problemi e migliorare la coerenza semantica con il testo.

Configurazione e Giustificazione

Per superare questi limiti, si è deciso di rivedere l'architettura ispirandosi al modello **AttnGAN** [4]. Le modifiche chiave sono le seguenti:

1. Architettura a 2 stadi Il generatore non sintetizza più l'immagine in un unico passaggio, ma opera in modo sequenziale su due stadi distinti:

- **Stadio 1 (64x64 pixel):** Il primo stadio genera un'immagine di 64x64 pixel che viene passata a un discriminatore apposito per questa risoluzione.
- **Stadio 2 (256x256 pixel):** Il secondo stadio prende in input l'ultima feature map del generatore precedente e, guidato nuovamente dalla cross-attention con il testo, la raffina per produrre l'immagine finale ad alta risoluzione, aggiungendo dettagli e contorni più nitidi. Anch'esso ha un discriminatore apposito, totalmente separato da quello precedente, che prende in input le immagini di 256x256 pixel generate.

2. Meccanismi di attenzione e residual connections Per migliorare il flusso di informazioni e gradienti, sono state apportate due modifiche ai blocchi del decoder:

- **Cross-Attention:** Per consentire l'uso della cross-attention a diversi livelli di risoluzione (dove il numero di canali della feature map dell'immagine può non corrispondere alla dimensione dell'embedding testuale), è stata introdotta una convoluzione 1×1 prima del meccanismo di attenzione. Questo strato proietta la feature map dell'immagine in uno spazio dimensionale compatibile con gli embedding testuali, rendendo l'attenzione applicabile a qualsiasi stadio del decoder. Ciò viene usato in particolare nella cross-attention del generatore del secondo stadio, in cui il numero di canali della feature map è minore della dimensione degli embedding di testo.
- **Residual connections:** Invece di imparare direttamente la mappa di feature di output, il meccanismo di cross-attention ora apprende la differenza tra la feature map di output e quella di input. Questo facilita il flusso del gradiente attraverso reti molto profonde, combattendo il problema del vanishing gradient.

3. Discriminatore con percorso condizionato e non condizionato Entrambi i discriminatori implementano due percorsi di valutazione distinti:

- **Percorso non condizionato:** Valuta unicamente la realisticità dell'immagine, ignorando il testo. L'obiettivo è distinguere tra immagini reali e generate.
- **Percorso condizionato:** Valuta la coerenza tra l'immagine e la descrizione testuale associata.

Funzione di loss del discriminatore La loss totale del discriminatore è calcolata come la media di cinque componenti, progettate per addestrare entrambi i percorsi su diversi scenari:

1. Loss su (*immagine reale*) per il percorso non condizionato (deve classificare come "reale").
2. Loss su (*immagine reale, testo corretto*) per il percorso condizionato (deve classificare come "reale").
3. Loss su (*immagine reale, testo sbagliato*) per il percorso condizionato (deve classificare come "falso").
4. Loss su (*immagine generata*) per il percorso non condizionato (deve classificare come "falso").

BCEWithLogitsLoss Inoltre, per mitigare ulteriormente il problema del vanishing gradient, si è sostituita la combinazione di attivazione sigmoidea e la loss **BCELoss** con la sola loss **BCEWithLogitsLoss**, che integra la sigmoide direttamente nel calcolo della loss in modo più stabile.

4.5.1 Risultati e analisi

Sono state condotte due run per valutare l'impatto del peso della loss L1 sul generatore ($\lambda_{L1} = 1$ e $\lambda_{L1} = 5$). Per specificare, la loss L1 viene applicata solo sull'immagine finale (256x256 pixel) come negli esperimenti precedenti.

Analisi quantitativa

Dall'analisi dei due grafici dell'andamento delle loss di generatore e discriminatore per entrambe le run (nelle Figure 4.21 e 4.22), si osserva un comportamento simile:

- La loss del discriminatore diminuisce e si stabilizza rapidamente su un valore basso ma non nullo (circa 0.5), indicando che sta apprendendo efficacemente a distinguere le immagini reali da quelle generate.
- La loss del generatore, dopo una fase iniziale di discesa che dura fino all'epoca 70 circa, inizia a risalire costantemente. Questo suggerisce che, superata una certa soglia, il discriminatore diventa troppo abile a distinguere le immagini del generatore.

Questo è un andamento simile all'esperimento precedente, tuttavia, è importante notare che, rispetto all'esperimento precedente, l'aumento della loss del generatore è meno marcato e inizia più tardi, sintomo di una stabilità di training relativamente maggiore.

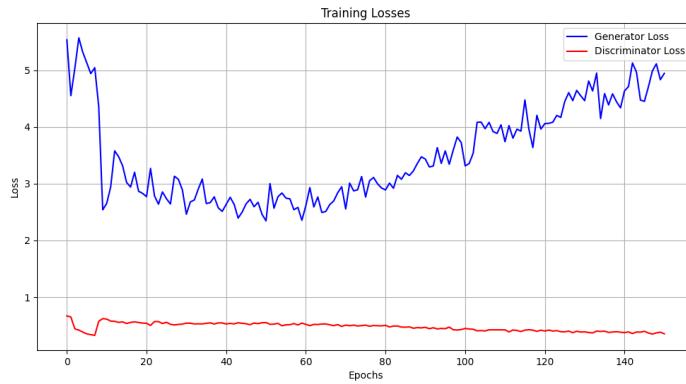


Figura 4.21: Andamento delle loss di generatore e discriminatore in run con $\lambda_{L1} = 1$



Figura 4.22: Andamento delle loss di generatore e discriminatore in run con $\lambda_{L1} = 5$

I grafici delle loss L1 mostrano che sia la loss di training che quella di validazione si stabilizzano su un valore di circa 0.25-0.3. Il fatto che le due curve siano allineate indica che il modello non sta soffrendo di overfitting dal punto di vista della ricostruzione. È inoltre interessante notare che, nonostante λ_{L1} sia più alta nel secondo grafico, il relativo valore di loss non si stabilizzi su un valore significativamente più basso, come ci si potrebbe aspettare dato il peso 5 volte maggiore dato alla loss (anche se effettivamente discende più rapidamente all'inizio). Ciò sta probabilmente a indicare che il generatore abbia comunque come obiettivo principale la generazione di un'immagine plausibile, per confondere il discriminatore, anche se diversa dall'originale.

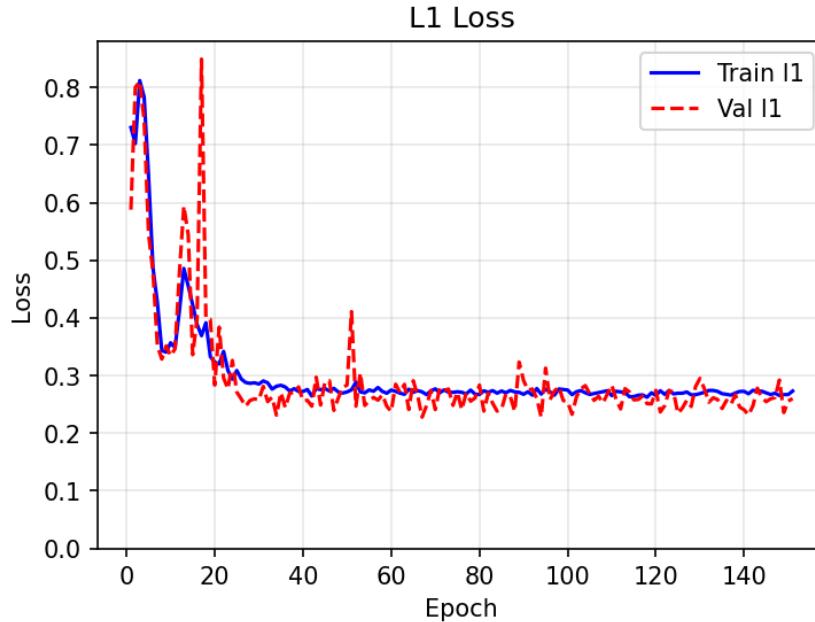


Figura 4.23: Andamento della loss L1 in run con $\lambda_{L1} = 1$

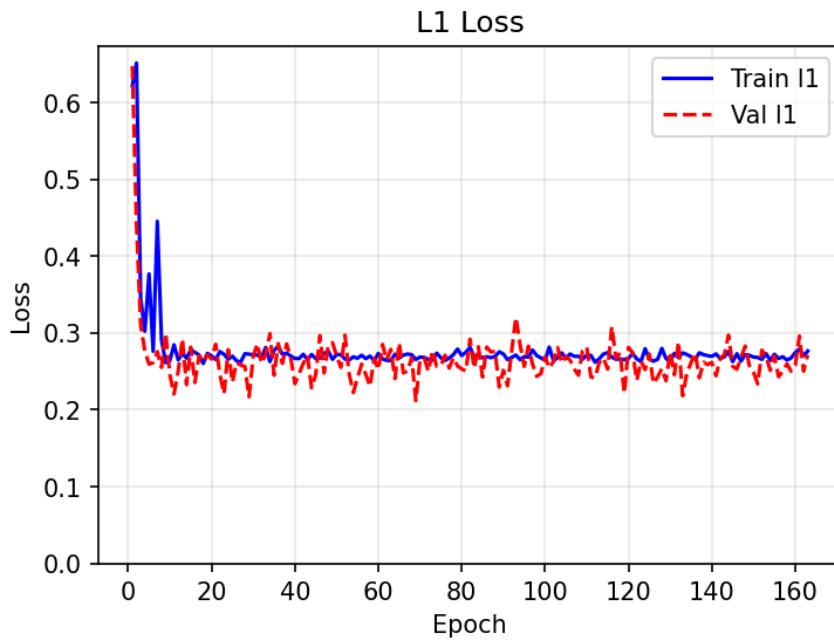


Figura 4.24: Andamento della loss L1 in run con $\lambda_{L1} = 5$

Analisi qualitativa

Andando a visionare le immagini generate, si può notare la **maggior varietà negli output** rispetto all'esperimento precedente, in particolare nel modello addestrato con $\lambda_{L1} = 1$. Questo indica che l'architettura multi-stadio e la nuova loss del discriminatore stanno funzionando nel promuovere la diversità.

Tuttavia, il limite principale risiede ancora nella **mancanza di una struttura riconoscibile**. Le immagini generate, pur essendo pertinenti dal punto di vista dei colori, non hanno una silhouette coerente, arti, occhi o altri dettagli.

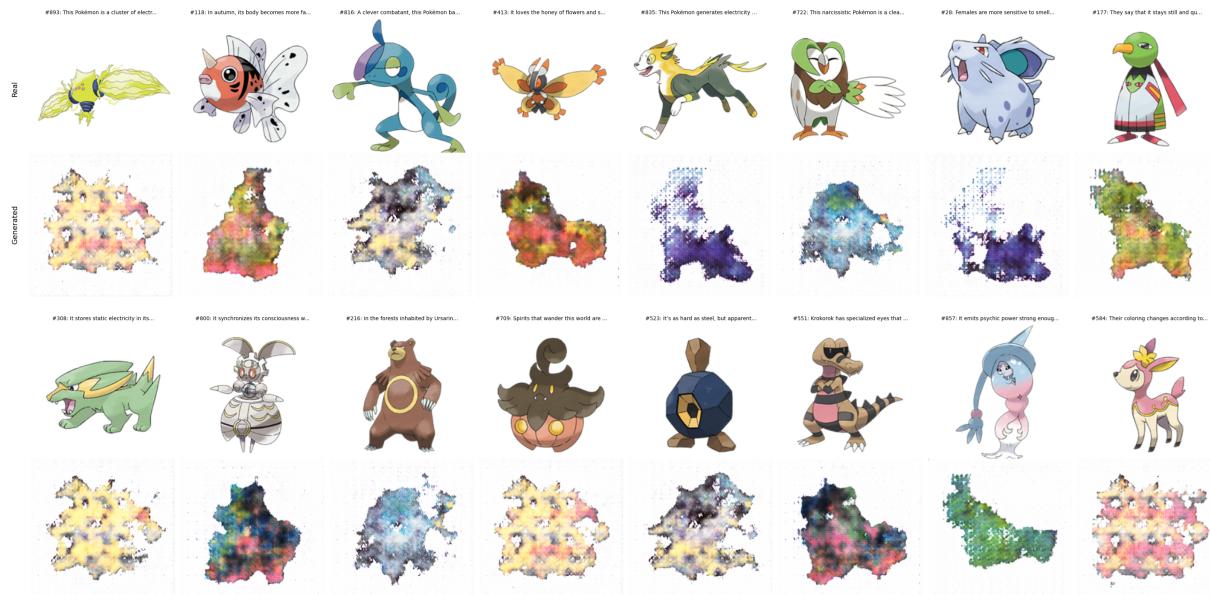


Figura 4.25: Esempi di generazioni di Pokemon nel training set ($\lambda_{L1} = 1$) - Esperimento 5



Figura 4.26: Esempi di generazioni di Pokemon nel training set ($\lambda_{L1} = 5$) - Esperimento 5

Analisi delle mappe di attenzione

Un'analisi diagnostica delle mappe di attenzione ha rivelato che il problema critico individuato nell'esperimento precedente persiste: il meccanismo che calcola il vettore di contesto globale iniziale assegna **pesi quasi identici a ogni parola del prompt**.

Di fatto, il modello non sta imparando a pesare l'importanza di ciascuna parola, possibilmente contribuendo alla mancanza di coerenza strutturale osservata nell'analisi qualitativa.



Figura 4.27: Mappe di attenzione per Pokémon #701 ($\lambda_{L1} = 5$) - Esperimento 5

4.6 Valutazione finale e scelta del modello

Al termine del processo iterativo di sviluppo, è stato selezionato un modello finale per una valutazione quantitativa sul test set, tenuto separato durante tutta la sperimentazione.

Scelta del modello finale Il modello scelto è quello sviluppato nell’Esperimento 5 con un peso per la loss L1 impostato a $\lambda_{L1} = 1$. Sebbene anche la variante con $\lambda_{L1} = 5$ abbia prodotto risultati interessanti, la configurazione con peso inferiore è stata preferita poiché ha mostrato nell’analisi qualitativa una **maggior varietà** negli output.

Metrica di valutazione: Kernel Inception Distance (KID) Per la valutazione quantitativa finale, si è optato per la metrica **Kernel Inception Distance (KID)** [1], un’alternativa alla più comune Fréchet Inception Distance (FID), particolarmente indicata per valutazioni su dataset di piccole dimensioni come quello usato nel progetto. Il principio di base della KID, condiviso con la FID, è quello di misurare la similarità percettiva tra un insieme di immagini reali e un insieme di immagini generate. Entrambe le metriche non eseguono un confronto diretto a livello di pixel, che risulta spesso poco significativo, ma operano invece estraendo le mappe di feature in un layer intermedio di un modello pre-addestrato, InceptionV3, da cui devono il nome. L’idea è che se le immagini generate sono realistiche e variegate, la distribuzione delle relative feature dovrebbe essere indistinguibile da quella delle feature delle immagini reali.

La differenza fondamentale tra KID e FID risiede nel modo in cui viene calcolata la distanza tra queste due distribuzioni di feature.

La FID modella le feature come campioni di una distribuzione gaussiana multivariata, mentre la KID adotta un approccio basato sulla **Maximum Mean Discrepancy (MMD)**.

La MMD è una metrica di distanza tra distribuzioni che non fa alcuna assunzione sulla loro forma. Essa opera proiettando le distribuzioni in uno spazio a dimensionalità maggiore, senza doverla calcolare esplicitamente ma solo determinando la distanza tra le medie delle due distribuzioni tramite il ”kernel trick” e relativa funzione kernel k .

Nello specifico, la KID calcola la MMD al quadrato, utilizzando un kernel polinomiale di terzo grado. Dato l’insieme di feature reali $X = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ e l’insieme di feature generate $Y = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$, la metrica è calcolata in questo modo:

$$\text{KID}(X, Y) = \frac{1}{m(m-1)} \sum_{i \neq j} k(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{n(n-1)} \sum_{i \neq j} k(\mathbf{y}_i, \mathbf{y}_j) - \frac{2}{mn} \sum_{i,j} k(\mathbf{x}_i, \mathbf{y}_j)$$

Intuitivamente, la formula combina tre elementi:

- **Il primo termine** calcola la similarità media interna all’insieme di **immagini reali**. Misura la coerenza e la varietà della distribuzione reale.
- **Il secondo termine** calcola la similarità media interna all’insieme di **immagini generate**.
- **Il terzo termine** (sottratto dai primi due) calcola la similarità media tra **l’insieme di immagini reali e l’insieme di immagini generate**.

Se le due distribuzioni sono identiche, il terzo termine bilancerà le similarità interne e il valore della KID tenderà a zero. Pertanto, come per FID, un valore di KID più basso

indica una maggiore somiglianza tra le immagini generate e quelle reali, con un punteggio ideale di zero che significa perfetta corrispondenza tra le distribuzioni.

Questo approccio conferisce alla KID due vantaggi cruciali rispetto alla FID:

- **Stime affidabili su piccoli campioni:** Nel calcolo della FID è richiesta la stima della matrice di covarianza, che può essere rumorosa se calcolata su piccoli insiemi di dati (meno di 10.000 campioni), portando a valori di FID inaffidabili. La KID fornisce risultati molto più stabili e significativi anche con poche centinaia di campioni.
- **Semplicità computazionale:** Il calcolo della MMD con un kernel polinomiale è semplice e veloce, rendendola pratica per valutazioni su piccoli dataset.

La KID è stata calcolata confrontando le immagini del test set con immagini generate dai rispettivi prompt, utilizzando 20 subset da 50 campioni, ovvero campionando con ripetizioni 50 esempi su cui valutare la metrica per 20 volte e poi calcolando media e deviazione standard.

Per riferimento, un valore buono di KID è tra 0 e 0.10, mentre un valore eccellente è minore di 0.05.

Risultati e analisi I risultati della valutazione sono riportati in Tabella 4.1.

Tabella 4.1: Risultati della metrica KID calcolata sul test set. Un valore più basso indica una migliore qualità e somiglianza percettiva.

Risoluzione	KID (media)	Deviazione Standard
64x64	0.2118	± 0.0095
256x256	0.2966	± 0.0108

I punteggi ottenuti, pur non essendo vicini allo zero ideale, sono in linea con le osservazioni emerse durante la sperimentazione. Il valore di KID conferma che il modello, pur riuscendo a catturare gli attributi cromatici generali, fatica a riprodurre una struttura coerente.

È da notare che il punteggio KID per la risoluzione finale (256x256) sia superiore a quello della risoluzione intermedia (64x64). Questo suggerisce che il primo stadio del generatore riesce a creare una bozza a bassa risoluzione più fedele, mentre il secondo stadio, avendo l'obiettivo di raffinare l'immagine, non riesce ad aggiungere dettagli molto vicini alla figura originale.

Capitolo 5

Conclusioni e sviluppi futuri

5.1 Sintesi del percorso e risultati finali

Questo progetto, denominato **PikaPikaGen**, si è posto l’obiettivo di sviluppare un modello generativo per la sintesi di sprite di Pokémon a partire da descrizioni testuali. Il percorso di sviluppo è stato caratterizzato da un approccio iterativo, partendo da un modello di baseline e introducendo progressivamente modifiche architetturali e di training per affrontare le criticità emerse.

Il punto di partenza è stato un modello Encoder-Decoder con una loss L1, che ha prodotto immagini sfocate e ha mostrato possibili segni di overfitting. L’introduzione di loss percettuali e edge-aware (VGG e Sobel) ha contribuito a mitigare la sfocatura, ma ha al contempo introdotto artefatti ad alta frequenza, e il modello ha mostrato difficoltà nel generalizzare a descrizioni non viste in fase di training. Un tentativo di combattere l’overfitting con una data augmentation troppo aggressiva ha inizialmente ostacolato la convergenza del modello, prima di essere ridotta.

È stato provato un cambio architetturale con l’adozione di una **Deep Convolutional Generative Adversarial Network (DC-GAN)** condizionata. Questo approccio ha prodotto silhouette e colori più definiti, sebbene con una limitata varietà negli output e instabilità nel training.

L’esperimento finale ha introdotto un’architettura più complessa, ispirata ad **Attn-GAN**, con una generazione a due stadi e discriminatori multipli con percorsi condizionati e non. Questo modello ha dimostrato una maggiore stabilità di training e una maggiore varietà negli output, ma la sfida principale è rimasta la **coerenza strutturale** delle immagini generate.

In conclusione, il progetto ha esplorato con successo diverse strategie per la generazione di immagini da testo, evidenziando le sfide legate alla ricostruzione pixel per pixel, alla coerenza strutturale e alla stabilità del training. Le architetture implementate, pur avendo fornito preziosi spunti, hanno mostrato i propri limiti nell’affrontare un compito di tale complessità con il dataset a disposizione.

5.2 Sviluppi futuri

- **Adottare architetture state-of-the-art:**
 - **Cambio architetturale:** L’adozione di un Variational Autoencoder o di un modello diffusivo potrebbe portare a miglioramenti consistenti nella creazione

delle immagini. Questi modelli hanno dimostrato di raggiungere una qualità e una diversità di generazione superiori alle GAN con un processo di training più stabile.

- **GAN più avanzate:** In alternativa, si potrebbero esplorare architetture GAN più moderne come **StyleGAN**, note per riuscire ad avere risultati ottimi anche su piccoli dataset.
- **Espandere e arricchire il dataset:** Un dataset più vasto e variegato potrebbe fornire al modello segnali più forti per l'apprendimento.

Bibliografia

- [1] Mikołaj Binkowski, Dougal J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. In *International Conference on Learning Representations*, 2018.
- [2] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.
- [3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.
- [4] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. AttnGAN: Fine-grained text to image generation with attentional generative adversarial networks, 2017.