Pokédex - Airceus

September 9, 2025

Luca Montefusco, Giacomo Tronca

Abstract

Il progetto analizza metodi di Machine Learning per la generazione di nuovi Pokémon a partire da input non convenzionali: l'idea è di tradurre segnali come file audio o testo in rappresentazioni che possano guidare la creazione di sprite unici. In questo modo, dati diversi vengono ricondotti a un unico obiettivo creativo: la progettazione di nuove creature ispirate al mondo Pokémon.

1. Creazione Database

Il database è stato implementato sotto forma di dizionari Python, in cui a ciascun Pokémon è associato un insieme di informazioni strutturate. Una lista ordinata di Pokémon con le informazioni a loro relative sono state recuperate grazie a (1); le cartelle degli sprite e versi sono state recuperate nel sito (2).

Struttura Alla lista di oltre 500 Pokémon, contenente le tipologie, una descrizione fisica e il peso, ordinata per numero di Pokédex, viene prima associata una classe utile nella distinzione testuale, dopodiché vengono aggiunti, tra le caratteristiche, i percorsi file per arrivare ai file di sprite e versi relativi ad ogni mostriciattolo tascabile all'interno del Drive. Per concludere, tramite il percorso file del verso viene calcolata la frequenza media di quest'ultimo e aggiunta alle caratteristiche del Pokémon in questione.

Spiegazione della struttura Questi dati sono stati scelti perché ciascuna caratteristica supporta uno dei due metodi di generazione: le frequenze e i versi per l'approccio basato su audio, le descrizioni testuali e le classi per l'approccio linguistico, in particolar modo la presenza della classe consente una distinzione più facile dei Pokémon al modello di training.

Email: Luca Montefusco <montefusco.2129071@studenti.uniroma1.it>, Giacomo Tronca <tronca.2155292@studenti.uniroma1.it>.

Machine Learning 2025, Sapienza University of Rome, 2nd semester a.y. 2024/2025.

2. Generazione tramite musica

Per quanto riguarda la generazione di Pokémon tramite musica, il procedimento è cambiato spesso durante il corso del progetto. Dopo aver introdotto la frequenza media del verso di ogni Pokémon all'interno del database, lo stesso procedimento è statao applicato alle canzoni: trovando la loro frequenza media, essa viene usata nella funzione *SearchPkmn* per generare una lista di Pokémon che rientrano in un range calcolato tramite la frequenza media della canzone. Nella parte relativa al training l'algoritmo si allena usando tutti i Pokémon presenti nel database, ma dopo questa fase la produzione dei nuovi mostriciattoli viene stretta per usare come modelli i Pokémon risultanti dalla ricerca della funzione citata precedentemente.

Modelli usati

- Inizialmente il modello usato per testare la generazione di immagini è stato **RealNVP** (7), ma i risultati dell'allenamento erano immagini rumorose e molto scure, perciò abbiamo implementato una funzione chiamata *crop non black* per eliminare i contorni neri intorno agli sprite. Per quanto abbiamo notato un miglioramento nella saturazione dei colori, le immagini sono rimaste rumorose.
- Il secondo modello usato è stato **VAE** (4), che è quello che ha dato i migliori risultati cambiando le canzoni: forme e colori erano differenti, ma le immagini non erano ben definite. Cambiare i parametri di training e l'aumento delle epoche ha portato a un miglioramento nelle forme e nei colori, dando origine a dei mostriciattoli credibili, ma sempre poco definiti, quasi fuori fuoco. Nel tentativo di aumentare la precisione del modello è stato introdotto, durante le epoche di training, un maggiore confronto tra i vari Pokémon esistenti, ma questo ha portato ad un risultato indipendente da training e input musicale, ovvero la produzione a immagine del primo Pokémon del database, nel nostro caso Bulbasaur.
- **VAE GAN** (3), l'ultimo modello usato in questa sezione del progetto, è stato quello più efficiente anche grazie all'introduzione della funzione *check-*

points, dando delle forme più distinguibili e una miglior distribuzione dei colori. Per monitorare meglio l'andamento della fase di training, la fase di validazione è stata modificata in modo tale da riportare in maniera più accurata, ad ogni epoca, i valori di *lossVae*, *lossRec* e *valRec*. I risultati migliori provengono da un allenamento del modello di circa 300 epoche e con una batch size di 64.

RealNVP **VAE** **VAE GAN**

Figure 1. Confronto dei risultati dei vari modelli

3. Generazione tramite testo

La generazione di nuovi Pokémon a partire da prompt testuale ha rappresentato la sfida più impegnativa del progetto: esistono numerosi modelli pre-addestrati che avrebbero potuto generare immagini soddisfacenti, ma l'obiettivo era quello di addestrare da zero un modello che potesse generare immagini coerenti al dataset a disposizione a partire da un prompt testuale coerente con le descrizioni testuali presenti per ogni immagine. L'algoritmo, nella sua prima fase, si occupa di creare un embedding testuale tramite una funzione *Encoder*, di modo da trasformare ogni descrizione e futuro input-prompt in un vettore. Successivamente, tramite una funzione *Decoder*, l'algoritmo crea una immagine a partire da un embedding testuale. Il modello viene dunque allenato minimizzando una *loss* che rappresenta la somiglianza tra l'immagine vera derivante da un determinato prompt e quella generata dal modello. Terminato l'allenamento, l'algoritmo può dunque generare muove immagini a partire da input di testo.

Modelli usati

- Il primo modello analizzato è stato RealNVP: è stato il primo modello studiato che avrebbe potuto generare nuove immagini a partire da testo. Tuttavia, come già spiegato in precedenza, i risultati sono stati ben lontani dal desiderato: la natura del modello infatti non era adeguata per l'allenamento e la conseguente generazione di immagini a bassa risoluzione, come quelle del database.

- Un altro modello analizzato è stato VAN (5): il modello, come evidenziato in precedenza, ha le potenzialità per essere molto utile per la generazione di immagini a bassa risoluzione e coerenti con il dataset, in quanto facilmente addestrabile da zero. Il problema principale è stato tuttavia capire come implementare efficacemente un text embedder che potesse essere compatibile con il modello: nonostante i numerosi tentativi fatti, non si è mai potuto sperimentare la bontà del modello per questo scopo, soprattutto per le difficoltà riscontrate nell'adattamento del codice alle esigenze correnti.

- L'approccio con probabilmente il maggior potenziale è stato quello del fine tuning attraverso LoRa (6). Data la difficoltà nell'addestramento di un modello da zero e la qualità degli output, la tecnica del fine tuning, ovvero di prendere un modello già ampiamente addestrato e "adeguare" i pesi in base al dataset fornito, poteva essere un giusto compromesso per alleggerire il training di un modello e al contempo generare immagini di qualità molto superiore. L'idea era quella di prendere un modello addestrato, come Stable-Diffusion, e addetsrare un modello LoRa con l'obbiettivo di bilanciare i pesi del modello e ottenere immagini coerenti al dataset. Anche in questo caso tuttavia, nonostante il potenziale, è stato piuttosto complicato fare progressi, sia per difficoltà nell'adattamento dei vari modelli, sia per una mancanza di potenza di calcolo: LoRA si è dimostrato infatti molto dispendioso a livello di utilizzo di VRAM, tanto da eccedere i limiti di Colab e del calcolatore personale utilizzato, dunque, di nuovo, non è stato possibile verificare con mano la bontà dell'idea.

- La scelta finale del modello è ricaduta su un prompt-to-text più semplice. La prima parte del codice si occupa, come fase preliminare, dell'analisi e la gestione dei prompt: tutte le descrizioni di ogni Pokémon del dataset vengono tokenizzate parola per parola, convertiti in indici numerici tramite un vocabolario. Per ogni parola, gli indici numerici vengono passati attraverso un layer di embedding che trasforma ogni token in un vettore denso di dimensione fissa, così che parole simili semanticamente avranno rappresentazioni vettoriali simili. Successivamente i vari vettori vengono utilizzati per condizionare la generazione di una nuovo immagine, di modo da identificare pattern comuni tra vettori simili e immagini e poter dedurre caratteristiche che dovrebbero avere i nuovi Pokémon generati. Sulla base di una loss MSE semplice si calcola il grado di similarità tra immagine reale e immagine generata dal modello, è si procede durante il training ad aggiustare i pesi di modo da minimizzare tale loss. Durante il training vengono inoltre salvati i pesi ottenuti

in dei checkpoint ad ogni epoca, per non perdere i progressi fatti e per poter fin da subito avere un generatore funzionante basato su un training di 500 epoche

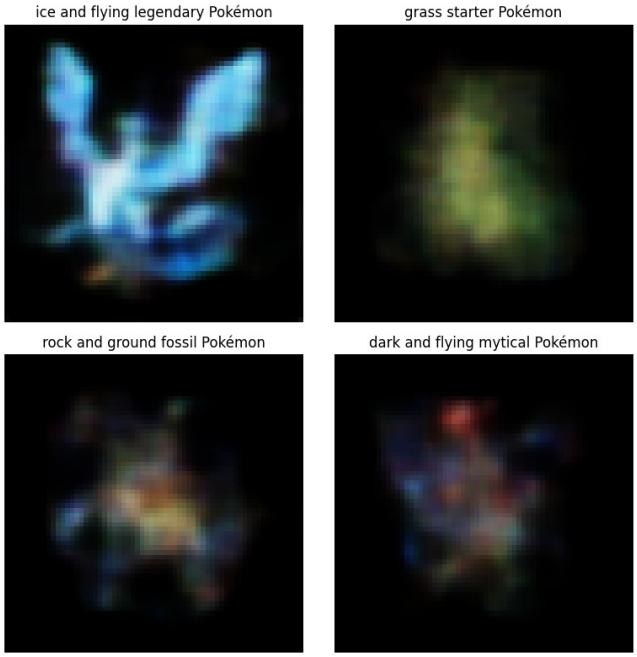

Figure 2. Uno degli ultimi risultati della produzione testuale

Conclusioni

Il progetto ha richiesto un notevole impegno da parte nostra e ci ha portato a fissare obiettivi ambiziosi, non semplici da raggiungere. Nonostante le difficoltà incontrate, riteniamo che i risultati conseguiti siano complessivamente soddisfacenti.

Siamo consapevoli che vi siano margini di miglioramento, in particolare modo attraverso l'utilizzo di un database più ampio, nonché mediante l'utilizzo di un sistema hardware più performante, che permetterebbe di affinare la qualità e la precisione delle immagini generate, nonché di diminuire il tempo necessario alla fase di training.

Tuttavia, il percorso intrapreso ci ha permesso di sperimentare metodologie innovative, e riteniamo che i traguardi raggiunti costituiscano una base su cui poter migliorare ulteriormente il progetto in futuro.

Bibliography

References

[1] Pokémon database. *https://pokemondb.net/tools/text-list*, 2018.

[2] Veekun. *https://veekun.com/dex/downloads*, 2020.

[3] Hugo L. Ole W. Anders B. L. L., Søren K. S. Autoencoding beyond pixels using a learned similarity metric. *Arxiv*, (1512.09300), 2016.

[4] Max Welling Diederik P. Kingma. Auto-encoding variational bayes. *Arxiv*, (1312.6114), 2022.

[5] Meng-Hao Guo, Cheng-Ze Lu, Zheng-Ning Liu, Ming-Ming Cheng, and Shi-Min Hu. Lora control - dynamic weights controller. *ArXiv*, (2202.09741), 2022.

[6] Chirs Heald. Lora control - dynamic weights controller. *Github*, 2023.

[7] Ba J. Kiros J. Swersky K. Liu J., Kumar A. Graph normalizing flows. *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, page 11pp, 2019.