# Relazione Progetto

## Classificazione Animali

Lisandro Antonioli

CORSO: Laboratorio di ottimizzazione, intelligenza artificiale e machine learning - CE

## Obiettivo  iniziale

L'obbiettivo iniziale del progetto è addestrare un classificatore di immagini che riconosca, una volta passata un'immagine, se presente un animale e quale animale vi è presente.

Andando avanti ho capito che sarebbe stato estremamente complesso se non impossibile.

Avrei dovuto  caricare tutti gli animali esistenti al mondo ed inoltre avrei dovuto trovare 500 immagini, numero corretto a mio parere per addestrare bene un modello, per ognuno di questi animali.

## Dataset

Il dataset nella mia idea iniziale avrebbe dovuto avere una cartella con tutte le immagini utili a fare il progetto, tutte mischiate insieme.

A queste avrei fatto una label personalizzata per ciascuna immagine con "Studio Label" , che grazie a un file con formato CoCo avrebbe passato i giusti valori al modello.

Su questo file CoCo ci sarebbe stato il tipo di animale, con anche indicato il riquadro dove esso si trovava, ed avrei avuto una cartella singola con tutte le label.

Il mio modello invece ha preso un'altra piega.

Sono partito a cercare un dataset su "Kaggle", sito che mi ha aiutato anche per altri progetti, ed ho trovato questo che comunque mi piaceva.

Anche se con limitazioni di immagini basate su cinque classi era buono per il mio progetto.

Una volta scaricato il datset ho visto che era senza label (non sarebbe stato un problema questo perché le avrei fatte io) ma soprattutto era diviso in directory.

Aveva cinque directory, una per ogni specie di animale.

Sono entrato nelle cartelle e per ognuna di queste specie ho selezionato 500 immagini.

Poi mi sono andato ad informare se avessi dovuto unirle o se potevo lasciarle separate cosi, anche perché erano più ordinate e belle da presentare.

In effetti potevo lasciare il dataset com'era: non dovendo fare ne object detection ne segmentation, non era necessario creare etichette manuali.

Per la classificazione era sufficiente organizzare le immagini in cartelle, una per classe, i nomi delle cartelle avrebbero funto poi da "label" durante il training.

Nella classificazione PyTorch usa automaticamente i nomi delle directory come etichette di classe. Quindi la struttura che mi era già stata fornita dal dataset su Kaggle sarebbe stata sufficiente.

## Augment_Train

Avevo ridotto il dataset a 500 immagini per ogni categoria di animale per aggiungerne altre inplace artificialmente.

Cosi ho preso un venti per cento delle immagini reali e ho applicato sopra le matrici delle trasformazioni.

Per fare questo ho creato uno script che ho chiamato "Augment_Train" .

In questo file è presente un controllo per verificare che gli argomenti dati siano effettivamente delle immagini, e varie modifiche alle immagini:

Rotazione casuale,  che fa ruotare l'immagine verso destra o verso sinistra di massimo dodici gradi;

Flip orizzontale, che capovolge l'immagine in maniera orizzontale cioè destra e sinistra.

Variazione di luminosità, che aumenta o diminuisce la luminosità per un massimo di venticinque per cento;

Blur, che applica una leggera sfocatura all'immagine per simulare foto fuori foco o mosse;

Rumore gaussiano ovvero, che aggiunge piccole variazioni casuali ai valori della matrice rendendo il modello più robusto a disturbi e compressioni;

Il resto del file serve per:

Scegliere in maniera casuale le immagini reali e applicare dalle una alle tre trasformazioni elencate sopra;

Salvare i nuovi file nella stessa cartella con suffisso “aug” per far comprendere al modello che sono già state modificate;

Integrazione di seed per rendere i risultati riproducibili;

Per concludere questa parte ci tengo a specificare che il Train soltanto avrà sia immagini reali che modificate da questo script, quelle di val invece avranno immagini solo reali per aver risultati migliori risultati.

## Train

Nel training si usa ResNet18, una rete neurale per immagini già addestrata su ImageNet.

È impiegata per la classificazione: si sostituisce l’ultimo strato con uno adatto alle N classi e lo si addestra sul dataset.

Le immagini sono organizzate in cartelle per classe e lette con ImageFolder.

Prima di entrare nel modello si applicano tre passaggi semplici:

Resize: rende tutte le immagini della stessa grandezza, così il modello le legge senza problemi;

ToTensor: mette la foto in un formato che PyTorch capisce per fare i calcoli;

Normalize: porta i pixel su una scala coerente e li centra; aiuta il modello ad allenarsi in modo più stabile.

In ogni epoca il modello guarda le immagini, calcola l'errore e aggiorna i propri pesi.

Alla fine si prova sul set di validazione.

Se il risultato è migliore di prima, si salva il best model, l’ultimo modello allenato viene salvato in ogni caso.

È anche impostato un seed fisso per avere risultati ripetibili e la scelta tra CPU e GPU è automatica.

All’inizio l’idea era di fare due directory diverse per le immagini di train, una dove vi erano solo immagini reali e l’altra dove vi erano immagini reali con quelle create con augment_train.

L’obbiettivo era vedere se vi erano differenze nel risultato dell’apprendimento o meno, se c’erano variazioni importanti a livello di loss e accuracy.

Alla fine però questo mio esperimento è risultato presso a poco identico in ambo i casi.

## Json

All’inizio la mia idea era quella di mettere il main direttamente dentro al train cosi da non avere troppi file.

Poi ho optato per una strada con più file, ma dove non dovessi mettere mano al codice, in questo il train è diventto riusabile anche per ipotetici altri progetti.

Il json funziona cosi:

Il file config.json viene letto da main.py che passerà i valori al modulo di training.

Il training quindi carica i dati, costruisce il modello e lo allena secondo i parametri indicati.

I campi più importanti del JSON sono:

Durectory, cioè i percorsi delle due cartelle principali, ovvero train_directory che serve per l'addestramento e val_directory che serve per la validazione, entrambe organizzate per classe

Parametri di addestramento, che controllano come si allena il modello:

Epochs, numero di epoche, cioè cicli completi sull'insieme di addestramento;

Batch_size, quante immagini vengono elaborate per batch;

Learning_rate, quanto velocemente si aggiornano i pesi a ogni passo di ottimizzazione.

Mi sono permesso di prendere giù delle note dai miei studi.

Se la loss fatica a scendere conviene diminuire leggermente il valore di learning_rate.

Se l’accuracy in validazione non migliora conviene aumentare le epoche.

## Test

Ho implementato alcuni test rapidi per verificare le parti fondamentali del mio train:

test_pick_device: verifica se la funzione pick_device ritorna cpu oppure cuda. Serve a confermare la scelta automatica del dispositivo.

test_set_seed: verifica la ripetibilità del risultato impostando lo stesso seed due volte: genera due numeri casuali (A e B) e controlla che coincidano.
Serve a garantire la riproducibilità: stessi risultati a parità di condizioni.

test_prepare_dataset: verifica il caricamento del dataset creando una mini copia del dataset e controllando che le classi lette siano corrette. La funzione salva anche la mappatura tra indici e nomi di classe nella cartella di output.

(es. 0→“cat”, 1→“dog”)

test_build_model: crea il modello impostato per 2 classi e controlla che lo strato finale abbia 2 uscite (una per classe). Serve a verificare che il modello si adatti al numero di classi richiesto