Beyond Faces and Fur: AI-Powered Image Recognition for Pets and People

Introduzione

In questo progetto, abbiamo sviluppato un modello di **classificazione delle immagini** utilizzando la libreria **PyTorch** e l'architettura **ResNet-50**. Ci si è concentrati sull'addestramento di due modelli distinti per affrontare compiti di classificazione in diversi domini: uno per la classificazione di immagini di animali e uno per la classificazione di immagini di persone.

Obiettivo

L'obiettivo principale è stato quello di addestrare un modello capace di riconoscere e classificare immagini di uccelli, gatti e cani, e un secondo modello per distinguere tra immagini di individui maschi e femmine.

Attraverso l'applicazione di tecniche di **apprendimento supervisionato** e l'uso di trasformazioni dei dati, il progetto ha mirato a migliorare la capacità di generalizzazione delle reti neurali convoluzionali per compiti di **computer vision**.

Architettura utilizzata: ResNet50 dopo ResNet18

Nel corso del progetto, è stato inizialmente impiegato il modello ResNet18 per la classificazione di immagini contenenti animali (cani, gatti e uccelli) e persone (maschi e femmine). La scelta di questo modello è stata motivata dalla sua struttura meno complessa e dalla capacità di addestrarsi rapidamente, caratteristiche vantaggiose quando si dispone di un dataset di dimensioni ridotte. Tuttavia, nonostante queste qualità, si è successivamente deciso di utilizzare il modello ResNet50. Questo cambiamento è stato guidato dalla necessità di ottenere elevate prestazioni in termini di accuratezza nella classificazione, riducendo al minimo gli errori e migliorando il riconoscimento delle diverse categorie. Sebbene ResNet50 richieda maggiori risorse computazionali e tempi di addestramento più lunghi, la sua maggiore profondità consente di catturare pattern più complessi nei dati, risultando quindi più adatta per compiti complessi come quello in esame. Nonostante il dataset non sia particolarmente grande, l'adozione di ResNet50 ha garantito un modello più robusto e preciso, in grado di distinguere in modo chiaro tra le categorie di interesse.

Definizione

ResNet18 e ResNet50 sono due varianti della famiglia di reti neurali ResNet (Residual Networks), utilizzate principalmente per compiti di visione artificiale come la classificazione delle immagini. La loro struttura si basa su blocchi residui che

permettono al modello di apprendere in modo più efficiente, mitigando il problema del vanishing gradient, problematica che si riscontra quando i gradienti, utilizzati per l'aggiornamento dei pesi della rete durante l'apprendimento, diventano troppo piccoli e l'apprendimento si blocca, tipico delle reti profonde.

ResNet18 è una versione più semplice e leggera, composta da 18 livelli, ideale per compiti meno complessi e per dataset di dimensioni ridotte, poiché richiede meno risorse computazionali ed è più veloce da addestrare.

ResNet50, invece, è una versione più profonda, con 50 livelli, progettata per catturare pattern più complessi nei dati. Grazie alla sua maggiore capacità di apprendimento, ResNet50 è più adatta per compiti complessi e per dataset più ampi, sebbene richieda un tempo di addestramento più lungo e maggiori risorse computazionali.

Analisi dei Test e Valutazione dei Risultati

Valutazione delle Prestazioni del Modello ResNet50: Test e Risultati

Durante la fase di testing con ResNet50, è stato utilizzato un dataset organizzato in una cartella denominata "test", contenente due sottocartelle: "people" e "animals", ognuna delle quali include immagini specifiche delle rispettive categorie, ovvero persone (maschi e femmine) e animali (cani, gatti e uccelli).

Il modello è stato addestrato utilizzando una cartella "train", fondamentale per il processo di apprendimento. Essa, infatti, contiene le sottocartelle per ciascuna delle categorie menzionate. Queste sottocartelle includono le immagini utilizzate per addestrare il modello, consentendogli di apprendere le caratteristiche distintive di ogni classe.

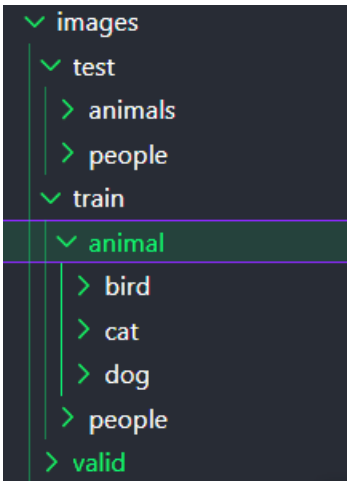
La cartella “test” viene invece usata dai file **Animal.py** e **People.py**, i quali utilizzano le immagini in essa contenute per poterle analizzare e di conseguenza classificarle nelle proprie sottocartelle nella cartella output “valid”.

Il modello è stato implementato nel file **Model.py**, mentre la classificazione delle immagini è stata gestita dai file **Animal.py** e **People.py**, ciascuno responsabile dell'elaborazione delle proprie categorie. In particolare, i file contengono le funzioni responsabili dell’applicazione dei filtri alle immagini classificate correttamente. Ad esempio, quando il modello, eseguendo Animal.py, riconosce un gatto, l'immagine viene salvata nella cartella "valid" nella sottocartella "cat" con un filtro bianco e nero.

Filtri analoghi sono stati applicati per le altre categorie: per i cani, un filtro che ruota l'immagine sottosopra, per gli uccelli un filtro di inversione dei colori, mentre per le persone viene applicato un filtro rosa per le donne e uno azzurro per gli uomini.

I risultati ottenuti con ResNet50 sono stati significativamente migliori fin dalle prime esecuzioni, mostrando una maggiore accuratezza nella classificazione delle immagini rispetto a modelli precedentemente testati.

Le immagini riconosciute e processate correttamente sono state salvate nelle rispettive sottocartelle, pronte per l'analisi finale. Di seguito una chiara struttura delle cartelle.



Esecuzione dei primi test con ResNet18: analisi dei risultati

Nella fase iniziale della nostra sperimentazione, abbiamo utilizzato il modello ResNet18 per eseguire test esclusivamente sulle immagini di animali. Questo approccio ci ha permesso di valutare le capacità del modello nel riconoscere e classificare categorie di animali come cani, gatti e uccelli. I risultati iniziali hanno evidenziato alcuni punti di forza e debolezze del modello, fornendo una base utile per le successive ottimizzazioni.

Dopodiché, siamo passati all'uso del modello ResNet50, che offre una maggiore profondità e complessità rispetto a ResNet18. Questa transizione ci ha permesso di ampliare il nostro ambito di test includendo anche la classificazione delle immagini di persone, suddivise in maschi e femmine. Con ResNet50, abbiamo potuto affrontare un compito di classificazione più complesso e variegato, incorporando sia animali che persone nel processo di valutazione. I risultati ottenuti hanno mostrato miglioramenti nelle prestazioni complessive, riflettendo la capacità avanzata del modello di gestire categorie più numerose e complesse.

In uno dei primi test con il modello ResNet18, le immagini sono state processate utilizzando i filtri previsti per le rispettive categorie: gatti, uccelli e cani. Ad esempio, le immagini di gatti sono state riconosciute e classificate applicando un filtro bianco e nero, mentre per gli uccelli e i cani sono stati applicati rispettivamente il filtro di inversione dei colori e l'immagine sottosopra.

I risultati hanno rivelato però che il modello ha avuto difficoltà nella classificazione corretta di alcune immagini, con una tendenza a confondere le immagini di uccelli con quelle di gatti.

Di seguito, un estratto dall'output ottenuto eseguendo i primi test con il modello ResNet18.

```
Immagine: 078_red-kite-50498_640.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 079_flamingo-600205_640.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 080_victoria-crowned-pigeon-4486154_640.jpg, Categoria Predetta: dog
Applico filtro cane.
Immagine: 090_bird-3183441_640.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 096_european-robin-isolated-on-white-background.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 136_halcyon-1352522_1280.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 137_eagle-2045655_1280.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 138_bird-3743094_1280.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 139_rooster-1867562_1280.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 141_swan-2077219_1280.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 142_barn-owl-2550068_1280.jpg, Categoria Predetta: cat
Applico filtro gatto.
Immagine: 160_eagle-1072696_1280.jpg, Categoria Predetta: bird
Applico filtro uccello.
Immagine: 164_parrots-3427188_1280.jpg, Categoria Predetta: cat
Applico filtro gatto.
```

Le immagini sopra citate fanno tutte parte della categoria degli uccelli, è dunque evidente la tendenza del modello a confondere le foto di uccelli con quelle di gatti, piuttosto che con quelle di cani.

Utilizzando invece il modello ResNet50, i risultati sono stati più promettenti, come mostreremo più avanti.

Aggiornamento dei Dataset: Ottimizzazione delle Immagini

Durante il corso delle nostre sperimentazioni, abbiamo apportato modifiche significative ai dataset di addestramento e di test per migliorare la qualità e l'efficacia della classificazione. In particolare, abbiamo sostituito e aggiornato alcune immagini nei dataset sulla base di criteri specifici che influenzano la precisione del modello.

I criteri considerati includevano:

- **Distanza del soggetto:** Abbiamo preferito immagini in cui il soggetto fosse ben visibile e non troppo distante, per garantire che le caratteristiche distintive fossero chiare e riconoscibili.

- **Messa a fuoco:** Le immagini sfocate (anche solo leggermente) sono state rimosse o sostituite, poiché la chiarezza dell'immagine è cruciale per una corretta classificazione.
- **Caratteristiche distintive:** È stato essenziale assicurarsi che le immagini riflettessero chiaramente le caratteristiche uniche di ciascuna specie, come il naso del cane o il becco dell'uccello, per ottimizzare l'addestramento del modello e la sua precisione nella classificazione.

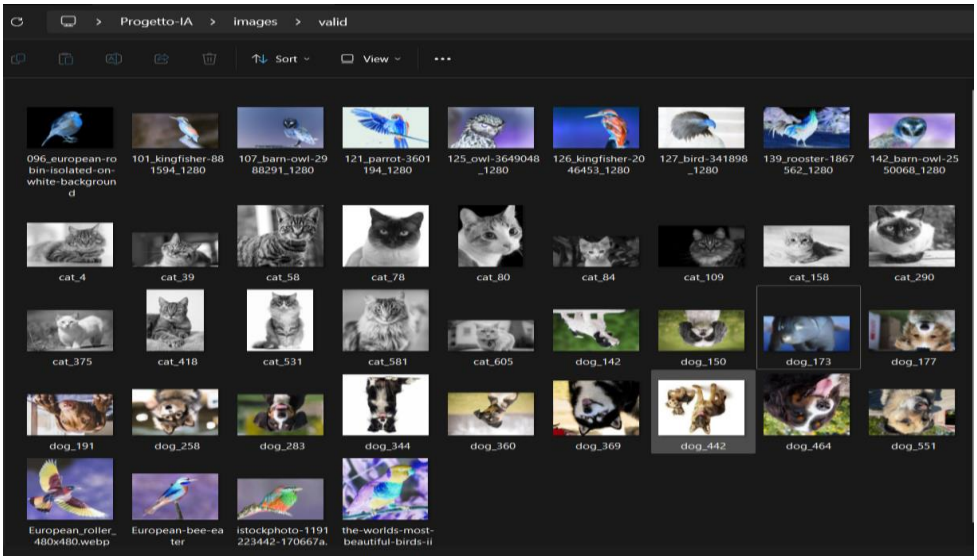
Questi aggiornamenti hanno contribuito a rafforzare la capacità del modello di effettuare previsioni più accurate e affidabili, migliorando complessivamente le sue prestazioni nei test successivi.

Esecuzione dei test successivi con ResNet50: analisi dei risultati

Test per animali (Animal.py)

Dopo aver cambiato modello utilizzando ResNet50, la classificazione delle immagini è stata decisamente migliore, presentando pochi errori.

Di seguito, le immagini salvate in "valid", ciascuna con il proprio filtro applicato (bianco e nero per gatti, foto sottosopra per cani e filtro inverti colori per uccelli).



Possiamo notare che la maggior parte delle immagini sono state riconosciute e classificate in maniera corretta, specialmente per quanto riguarda le categorie dei gatti e degli uccelli. Per i cani, invece, è stato commesso un singolo errore in *dog_173*, che è stato confuso per un uccello.

Nel complesso, possiamo dedurre che, grazie alla scelta del modello ResNet50, i risultati sono nettamente migliorati e la tendenza a commettere errori è diminuita.

Per questo test, nella cartella “valid”, le immagini output venivano salvate in maniera non organizzata. Questo perché, prima di eseguire la suddivisione in sottocartelle di “valid”, l’unico scopo era quello di verificare che la nuova versione con ResNet50 fosse in grado di compiere il suo dovere in modo efficiente. Una volta ottenuto ciò, le sottocartelle vennero create per entrambi gli output di **Animal.py** e **Gender.py**.

Nota aggiuntiva

Nel codice, è stata introdotta una funzione denominata `get_random_images` per migliorare il processo di test del modello di classificazione delle immagini. Quando il modello viene eseguito attraverso la funzione `process_images`, le immagini vengono selezionate in modo casuale dalla cartella di test, che contiene le immagini degli animali da classificare.

L'obiettivo di questa implementazione è garantire che il modello venga testato in modo casuale ogni volta che il programma viene eseguito, anche se le immagini di test rimangono le stesse. Selezionando le immagini in ordine casuale, il modello viene sottoposto a una varietà di sequenze di test, offrendo una verifica più completa delle sue capacità di classificazione. Questo approccio aiuta a confermare che il modello non stia semplicemente memorizzando l'ordine delle immagini, ma stia effettivamente classificando correttamente le immagini basandosi sulle loro caratteristiche.

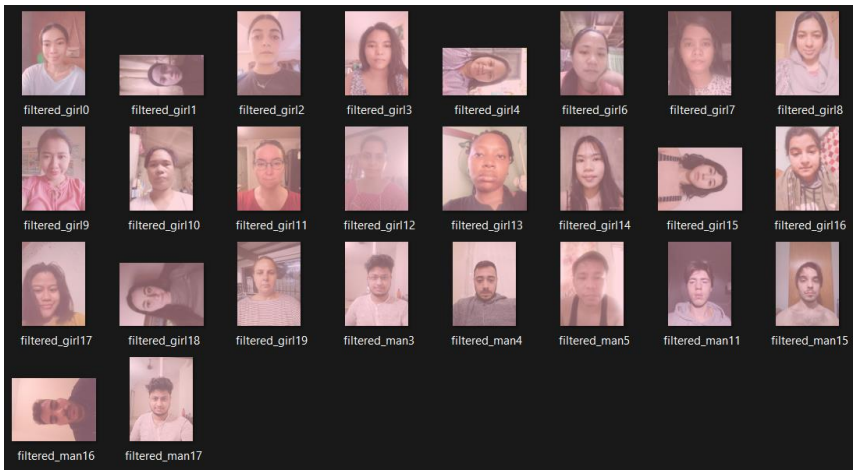
Test per persone (Gender.py)

Utilizzando sempre il modello ResNet50, la categoria delle persone (maschi e femmine) è stata testata diverse volte, rivelando un numero maggiore di errori rispetto alla categoria degli animali. Questo potrebbe essere attribuito a vari fattori, come la selezione delle immagini, dei soggetti e, in particolare, alla presenza di dettagli distintivi degli esseri umani.

I tratti che caratterizzano gli esseri umani, come quelli del volto, possono essere più complessi e difficili da individuare correttamente rispetto ai tratti degli animali per diverse ragioni. Ad esempio, le differenze tra un uomo e una donna possono essere sottili e variare molto da persona a persona, rendendo più difficile per un modello distinguere correttamente tra i due sessi basandosi solo sulle caratteristiche facciali.

In confronto, i tratti distintivi degli animali, come il becco di un uccello o i baffi di un gatto, sono spesso molto più evidenti e specifici a una specie particolare. Questi tratti caratteristici rendono più semplice per un modello identificare e distinguere una specie dall'altra. Per esempio, è più facile per un modello riconoscere un uccello e distinguerlo da un gatto, perché le differenze fisiche tra queste due specie sono marcate e facilmente identificabili. Al contrario, le variazioni sottili nei tratti umani richiedono una maggiore precisione e sensibilità nel riconoscimento, rendendo il compito più impegnativo.

Dopo aver eseguito vari test, nella cartella output “valid” troviamo le immagini di uomini e donne classificate, ciascuna con il proprio filtro applicato. Dato che per ciascun sesso sono presenti 19 foto, notiamo che nella cartella “female” sono state aggiunte diverse immagini appartenenti alla categoria “men”, e che invece solo una immagine di donna è stata mal interpretata e messa nella categoria sbagliata.



Da questo possiamo dedurre che il modello tende a confondere gli uomini con le donne più frequentemente di quanto confonda le donne con gli uomini. Questo potrebbe indicare che il modello ha difficoltà a identificare con precisione i tratti distintivi degli uomini, risultando in un maggiore numero di errori di classificazione per questa categoria.

Una possibile soluzione potrebbe essere quella di possedere una maggiore varietà di immagini per entrambe le categorie, assicurandosi di includere varie pose, etnie età ed espressioni facciali. Altre soluzioni possono essere la scelta di un modello diverso rispetto a ResNet50 oppure considerare il fine-tuning su un dataset specifico di volti umani, per adattare meglio il modello al compito specifico.

Il **fine-tuning** è una tecnica utilizzata per adattare un modello pre-addestrato a un compito specifico, sfruttando la conoscenza già acquisita dal modello stesso. In pratica, si inizia con un modello che è stato precedentemente addestrato su un ampio dataset generico, come ImageNet, il quale ha appreso a riconoscere una vasta gamma di caratteristiche visive di base. Quando si applica il fine-tuning, si modificano gli strati

finali del modello per meglio adattarsi al nuovo compito, come il riconoscimento di volti umani maschili e femminili, mantenendo intatti gli strati iniziali che catturano le caratteristiche generali.

Durante il fine-tuning, il modello pre-addestrato viene addestrato ulteriormente su un dataset specifico relativo al compito in questione. Gli strati nuovi o modificati vengono aggiornati con i dati del nuovo dataset, mentre gli strati precedenti possono essere mantenuti congelati per preservare le competenze già apprese. Questo approccio consente di ottenere un modello che non solo beneficia della conoscenza preesistente, ma si adatta anche in modo ottimale alle specificità del nuovo compito, migliorando così le prestazioni nella classificazione o riconoscimento delle immagini in questione.

Config.json e config_schema.json

Abbiamo creato e implementato i due file per la validazione della configurazione: ‘config_json’ e ‘config_schema.json’:

- **config.json**: contiene tutti i parametri di configurazione necessari per il training e la validazione dei modelli. Ad esempio, include le directory per i dataset, i nomi dei file per salvare i modelli e i parametri di training.
- **config_schema.json**: definisce lo schema JSON per validare il contenuto di config.json. Questo schema è progettato per garantire che config.json abbia la struttura corretta e i tipi di dati giusti prima di essere utilizzato nel codice.

Per assicurare che il file di configurazione rispetti il formato e i requisiti definiti, abbiamo integrato la libreria **jsonschema**. Questa libreria consente di validare il file di configurazione rispetto allo schema JSON. In particolare, durante l'esecuzione del programma, jsonschema carica sia config.json che config_schema.json e verifica che il primo aderisca alle specifiche del secondo. Se il file di configurazione non rispetta le regole definite, viene segnalato un errore e l'esecuzione del programma viene interrotta.

L’implementazione e configurazione di questi file ci ha portato diversi vantaggi:

1. Separazione della configurazione: Rendendo il codice più pulito e leggibile, poiché i parametri di configurazione sono esterni al codice principale
2. Flessibilità e riutilizzabilità: Permette di modificare facilmente i parametri di configurazione o riutilizzare il codice per addestrare e inferire su diversi modelli senza modificare il codice sorgente.
3. Scalabilità: Facilita l’espansione futura del progetto senza dover rifattorizzare il codice esistente, poiché le modifiche possono essere gestite attraverso i file di configurazione

Incorporare la validazione tramite ‘jsonschema’ contribuisce a garantire l'integrità e la coerenza dei dati di configurazione, riducendo il rischio di errori e migliorando la robustezza complessiva del sistema.

Rappresentazione dei dati

Per rappresentare i dati ottenuti dopo l'addestramento dei modelli e quindi dall'esecuzione dei file **Animal.py**, **Gender.py** e **Model.py**, abbiamo utilizzato la libreria *matplotlib.pyplot* per effettuare una distribuzione delle probabilità (o confidenze) sulle classi possibili; quindi, graficare la distribuzione di queste probabilità per vedere quanto il modello è sicuro delle sue predizioni.

La libreria utilizzata ci consente di visualizzare i risultati delle predizioni dei modelli in modo chiaro e comprensibile.

Passaggi eseguiti:

1. **Preparazione dei dati:** Abbiamo creato un DataFrame che contiene i nomi delle immagini, le categorie predette (animali e persone) e le rispettive confidenze.
2. **Creazione del grafico:** utilizzando plt.bar(), abbiamo creato un grafico a barre in cui l'**asse x** rappresenta i **nomi** delle immagini e l'**asse y** la **confidenza** della predizione; le barre sono colorate in base alla classe (verde per la confidenza della predizione degli animali e blu per quella relativa al genere), con bordi neri per migliorare la visibilità e l’alpha (trasparenza) impostata a 0.7 per un effetto visivo più gradevole.
3. **Personalizzazione del grafico:** abbiamo impostato etichette per gli assi e un titolo per il grafico; le etichette dell’asse x sono state ruotate di 45 gradi per evitare sovrapposizioni e migliorare la leggibilità.

Esecuzione di Model.py

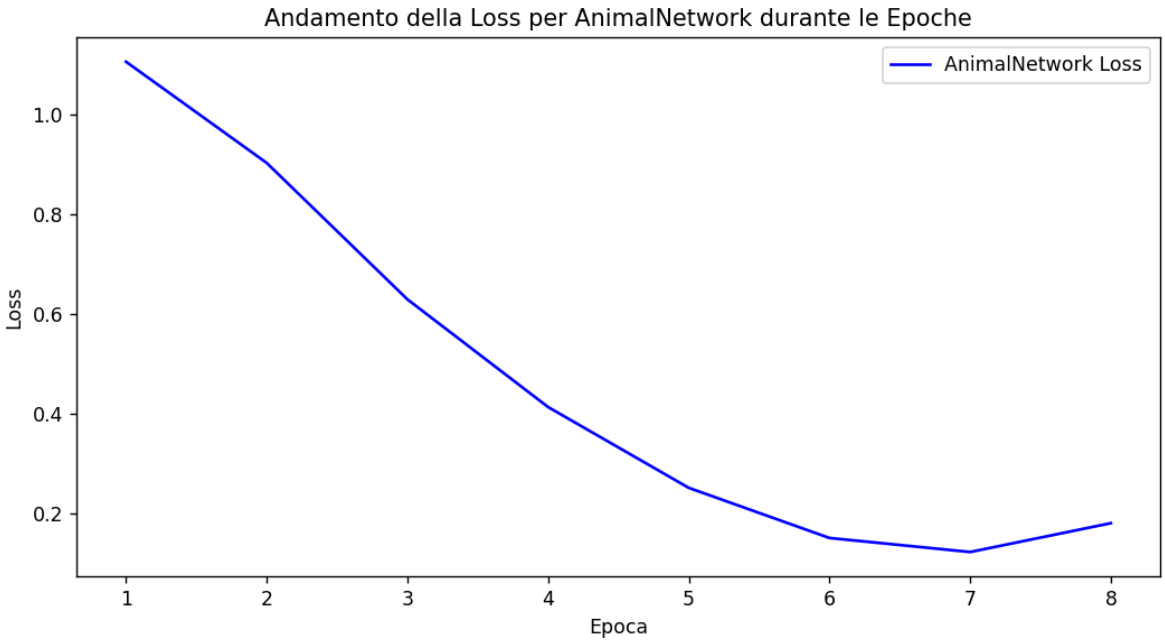
Abbiamo agito similmente anche per graficare la funzione di loss presente nel file **Model.py**.

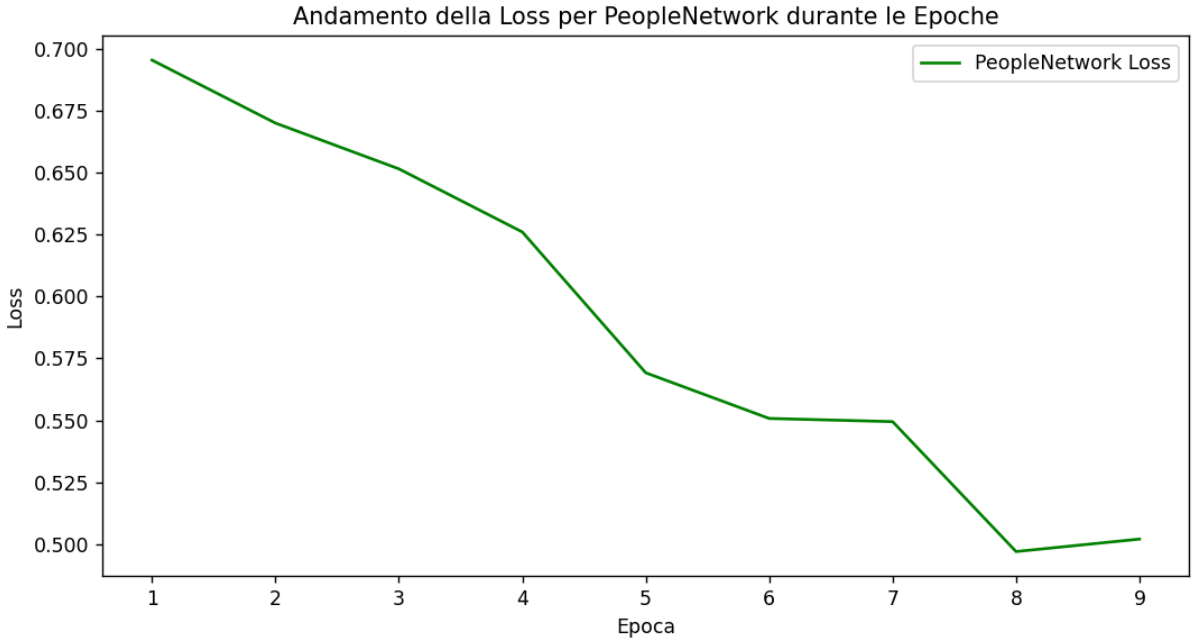
Invece di usare un grafico a barre, abbiamo ritenuto essere più corretto utilizzare un grafico a linee per rappresentare l’andamento della **loss** durante le epoche di addestramento dei modelli. Questo tipo di grafico è utile per visualizzare le variazioni e le tendenze di una variabile continua nel corso di un intervallo, come le epoche di addestramento in questo caso. Nello specifico, all’**asse delle x** viene assegnato il numero di **epoche** e, di conseguenza, ha un intervallo da 1 a 10, mentre all’**asse delle y** viene assegnato il valore della **loss**.

Per distinguere le due rette sono stati usati due colori diversi: arancione relativa alla loss di PeopleNetwork, azzurra per AnimalNetwork.

Nota aggiuntiva

Eseguendo questo script, dopo aver visualizzato i plot delle due loss, verranno eseguiti in automatico quelli relativi a Gender ed Animal. In un primo momento ciò non accadeva ma, volendo aggiungere qualche feature al nostro progetto e per non doverlo fare manualmente, abbiamo deciso di usare la libreria `subprocess.` Subrocess è una libreria python usata per eseguire comandi del sistema operativo come se fossero eseguiti dalla riga di comando; quindi abbiamo usato `subprocess.run()` per eseguire il file desiderato specificandone nome ed estensione.





Dal grafico sopra mostrato, possiamo notare i seguenti aspetti:

1. **Tasso di Convergenza:**
   - **Loss di AnimalNetwork:** La loss diminuisce rapidamente e costantemente da oltre 1.0 a circa 0.2 nelle prime otto epoche e aumenta leggermente successivamente. Questo indica un miglioramento significativo nelle prestazioni della rete, suggerendo che ha appreso rapidamente i modelli nei dati.
   - **Loss di PeopleNetwork:** La loss ha circa lo stesso andamento di quello relativo ad AnimaNetwork, con la differenza che è meno graduale. Infatti, in alcuni punti (per esempio dal valore di loss pari a 0.625 circa a 0.560 circa) la loss diminuisce in maniera significativa e meno graduale rispetto all'andamento generale di AnimallNetwork. Questo indica che la rete sta apprendendo correttamente, anche se impiega più tempo rispetto all'altra, portando ugualmente buoni risultati.
   - **Valori finali di loss:** Il valore finale di AnimalNetwork è inferiore a quello di PeopleNetwork, suggerendo che è più accurata o meglio ottimizzata per il suo compito. Ciò potrebbe essere dovuto dal fatto che è più complicato riconoscere un volto maschile da uno femminile poichè le differenze sono meno grossolane e/o evidenti rispetto al riconoscimento di un uccello da un gatto, le cui differenze sono molto più evidenti.
2. **Osservazioni:**
   - AnimalNetwork supera costantemente PeopleNetwork in termini di loss durante l'intero processo di addestramento. Questo potrebbe essere dovuto a

differenze nei dataset, alla complessità dei compiti o alle architetture delle reti.

Salvataggio del modello migliore

Nel corso dell'addestramento di reti neurali, è fondamentale salvare il modello che raggiunge le migliori prestazioni, piuttosto che salvare semplicemente l'ultimo stato del modello alla fine dell'addestramento. Per questo motivo, è stata implementata una nuova funzionalità in **Model.py** che permette di monitorare la performance del modello su un set di validazione e salvare il modello solo quando viene raggiunta la miglior prestazione (misurata, ad esempio, dalla loss più bassa o dalla più alta accuratezza).

Questa modifica garantisce che il modello salvato al termine del processo di addestramento sia effettivamente quello più performante, riducendo il rischio di sovradattamento (overfitting) che può verificarsi nelle ultime epoche. Il modello migliore viene salvato con il suo stato ottimale, pronto per essere utilizzato nelle fasi successive del progetto, come ulteriori analisi.

Per implementare questa funzionalità, è stato aggiunto un controllo durante ogni epoca di addestramento che verifica se la loss corrente è inferiore a quella del miglior modello salvato fino a quel momento. In caso ciò accada, il modello viene salvato con il suo stato attuale, garantendo così la miglior efficacia possibile.

Integrazione di Early Stopping

Durante l'addestramento delle reti neurali, è fondamentale evitare l'overfitting, che può verificarsi quando il modello continua ad addestrarsi anche dopo che le sue prestazioni su un set di validazione iniziano a peggiorare. Per affrontare questo problema, è stata implementata una nuova funzionalità di early stopping all’interno di **Model.py**. Questa tecnica monitora la perdita di validazione durante l'addestramento e interrompe il processo quando non si osservano miglioramenti significativi per un numero specifico di epoche consecutive.

L'integrazione di questa tecnica garantisce che l'addestramento si interrompa al momento ottimale, preservando il miglior modello possibile e riducendo i tempi di addestramento inutili. Ciò è particolarmente utile in scenari in cui il sovradattamento può degradare le prestazioni del modello nelle fasi finali dell'addestramento.

Per implementare questa funzionalità, è stato aggiunto un meccanismo che controlla la perdita di validazione dopo ogni epoca. Se la perdita non migliora rispetto alla miglior perdita osservata entro un certo numero di epoche (definito dal parametro **patience**), l'addestramento viene interrotto anticipatamente. Questo assicura che il modello finale sia il più generalizzabile possibile, riducendo il rischio di overfitting e ottimizzando l'efficacia complessiva del modello per l'inferenza o ulteriori applicazioni.

Conlusioni e possibili soluzioni

Per PeopleNetwork, potrebbe essere utile esplorare altre architetture, la regolazione degli iperparametri o diverse tecniche di addestramento per migliorare l'apprendimento.

In generale, il grafico indica che AnimalNetwork sta performando bene e apprendendo rapidamente, mentre PeopleNetwork mostra un progresso di apprendimento più lento e potrebbe necessitare di aggiustamenti per migliorarlo ulteriormente.

Valutazione dei dati ottenuti durante l’esecuzione del progetto

Di seguito osserviamo i risultati ottenuti con una breve spiegazione, ma prima una rapida contestualizzazione: la **confidence** (confidenza) in un modello di classificazione rappresenta la probabilità stimata che il modello attribuisce a una determinata classe per un dato campione. Le confidenze vicine a 1.0 mostrano che il modello è molto sicuro della classificazione per certe immagini.

Per cominciare, viene eseguito il file **Model.py**, ottenendo il seguente output (a cui seguono i grafici riguardanti la loss per ciascuna categoria analizzati in precedenza):

```
Il file config.json è valido.
Inizio dell'addestramento del modello animal_model.pth...
Epoch 1, Loss: 1.105249285697937
Modello migliorato e salvato in 'best_animal_model.pth' con Loss: 1.1052
Epoch 2, Loss: 0.9026020963986715
Modello migliorato e salvato in 'best_animal_model.pth' con Loss: 0.9026
Epoch 3, Loss: 0.6293690204620361
Modello migliorato e salvato in 'best_animal_model.pth' con Loss: 0.6294
Epoch 4, Loss: 0.4136364956696828
Modello migliorato e salvato in 'best_animal_model.pth' con Loss: 0.4136
Epoch 5, Loss: 0.25180740654468536
Modello migliorato e salvato in 'best_animal_model.pth' con Loss: 0.2518
Epoch 6, Loss: 0.15139003843069077
Modello migliorato e salvato in 'best_animal_model.pth' con Loss: 0.1514
Epoch 7, Loss: 0.12329011410474777
Modello migliorato e salvato in 'best_animal_model.pth' con Loss: 0.1233
Epoch 8, Loss: 0.18106773495674133
Early stopping attivato. Addestramento interrotto
Modello salvato come 'animal_model.pth
Inizio dell'addestramento del modello people_model.pth...
Epoch 1, Loss: 0.6953122913837433
Modello migliorato e salvato in 'best_people_model.pth' con Loss: 0.6953
Epoch 2, Loss: 0.6699106395244598
Modello migliorato e salvato in 'best_people_model.pth' con Loss: 0.6699
Epoch 3, Loss: 0.6514388918876648
Modello migliorato e salvato in 'best_people_model.pth' con Loss: 0.6514
Epoch 4, Loss: 0.6259838044643402
Modello migliorato e salvato in 'best_people_model.pth' con Loss: 0.6260
Epoch 5, Loss: 0.5691805779933929
Modello migliorato e salvato in 'best_people_model.pth' con Loss: 0.5692
Epoch 6, Loss: 0.5507648885250092
Modello migliorato e salvato in 'best_people_model.pth' con Loss: 0.5508
Epoch 7, Loss: 0.5494830310344696
Modello migliorato e salvato in 'best_people_model.pth' con Loss: 0.5495
Epoch 8, Loss: 0.497076153755188
Modello migliorato e salvato in 'best_people_model.pth' con Loss: 0.4971
Epoch 9, Loss: 0.5021054446697235
Early stopping attivato. Addestramento interrotto
Modello salvato come 'people_model.pth
```

Da ciò si evidenzia il processo di addestramento dei due modelli, rispettivamente animal_model.pth e people_model.pth, distribuito su dieci epoche per ognuno.

Osserviamo il calcolo della metrica loss per ciascuna epoca, che rappresenta l'errore tra le previsioni del modello e i valori reali. Un valore di loss più alto indica una maggiore discrepanza, mentre un valore più basso suggerisce che il modello sta migliorando nell'apprendimento delle caratteristiche delle immagini.

Come possiamo osservare, entrambi i modelli hanno mostrato miglioramenti significativi nelle prime epoche. Tuttavia, intorno all'epoca 8, si è verificato un aumento della loss per entrambi i modelli, che ha portato all'attivazione dell'early stopping.

Per il modello AnimalNetwork, la loss è migliorata costantemente fino all'epoca 7, ma è aumentata nell'epoca 8, attivando l'early stopping. Questo aumento potrebbe suggerire una possibile instabilità dell'addestramento o un inizio di sovradattamento.

Analogamente, il modello PeopleNetwork ha mostrato un miglioramento fino all'epoca 8, con un incremento della loss all'epoca 9 che ha causato l'attivazione dell'early stopping.

In entrambi i casi, l'early stopping ha funzionato come previsto, interrompendo l'addestramento quando la metrica di valutazione non ha mostrato ulteriori miglioramenti.

Output dell’esecuzione del file Animal.py

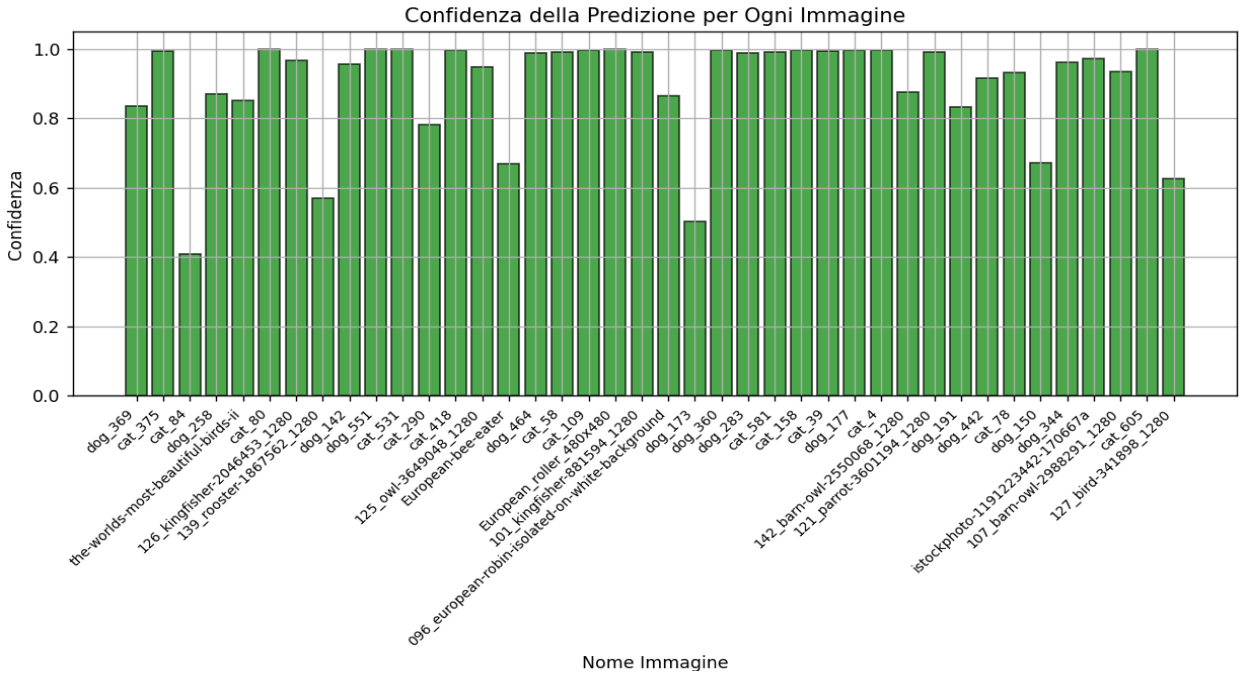
```
Il file config.json è valido.
                                          Name Category  Confidence
0                                      dog_369      dog    0.836120
1                                      cat_375      cat    0.994197
2                                       cat_84     bird    0.409951
3                                      dog_258      dog    0.871150
4                 the-worlds-most-beautiful-birds-ii     bird    0.850781
5                                       cat_80      cat    0.998902
6                        126_kingfisher-2046453_1280     bird    0.967437
7                           139_rooster-1867562_1280     bird    0.568808
8                                      dog_142      dog    0.956943
9                                      dog_551      dog    0.999803
10                                     cat_531      cat    0.998499
11                                     cat_290      cat    0.781133
12                                     cat_418      cat    0.996839
13                              125_owl-3649048_1280     bird    0.947063
14                               European-bee-eater     bird    0.669337
15                                     dog_464      dog    0.987912
16                                      cat_58      cat    0.989994
17                                     cat_109      cat    0.997202
18                         European_roller_480x480     bird    0.999140
19                        101_kingfisher-881594_1280     bird    0.990808
20  096_european-robin-isolated-on-white-background     bird    0.865300
21                                     dog_173      dog    0.502914
22                                     dog_360      dog    0.997754
23                                     dog_283      dog    0.987847
24                                     cat_581      cat    0.991708
25                                     cat_158      cat    0.996326
26                                      cat_39      cat    0.994607
27                                     dog_177      dog    0.997314
28                                       cat_4      cat    0.995347
29                         142_barn-owl-2550068_1280     bird    0.875198
```

Da questi dati riusciamo a capire che il modello è generalmente molto sicuro delle sue previsioni per le immagini di tutte le categorie. Notiamo un errore nel riconoscimento di “cat_84”, che è stato mal interpretato, ma nel complesso i risultati sono incoraggianti.

Per risolvere questo tipo di errori, ottenendo così una confidence migliore, si potrebbero esaminare le immagini fraintese per capire se hanno caratteristiche particolari che rendono difficile la classificazione.

Una volta ottenuto l'output precedentemente osservato, un grafico a barre verrà visualizzato. Esso mostra il valore di confidenza della previsione per ciascuna immagine analizzata dal modello per la categoria degli animali.

Come spiegato in precedenza, le confidenze prossime a 1.0 indicano che il modello è estremamente sicuro nella classificazione di alcune immagini; perciò, grazie a questo grafico possiamo analizzare il comportamento del modello per ciascuna foto.

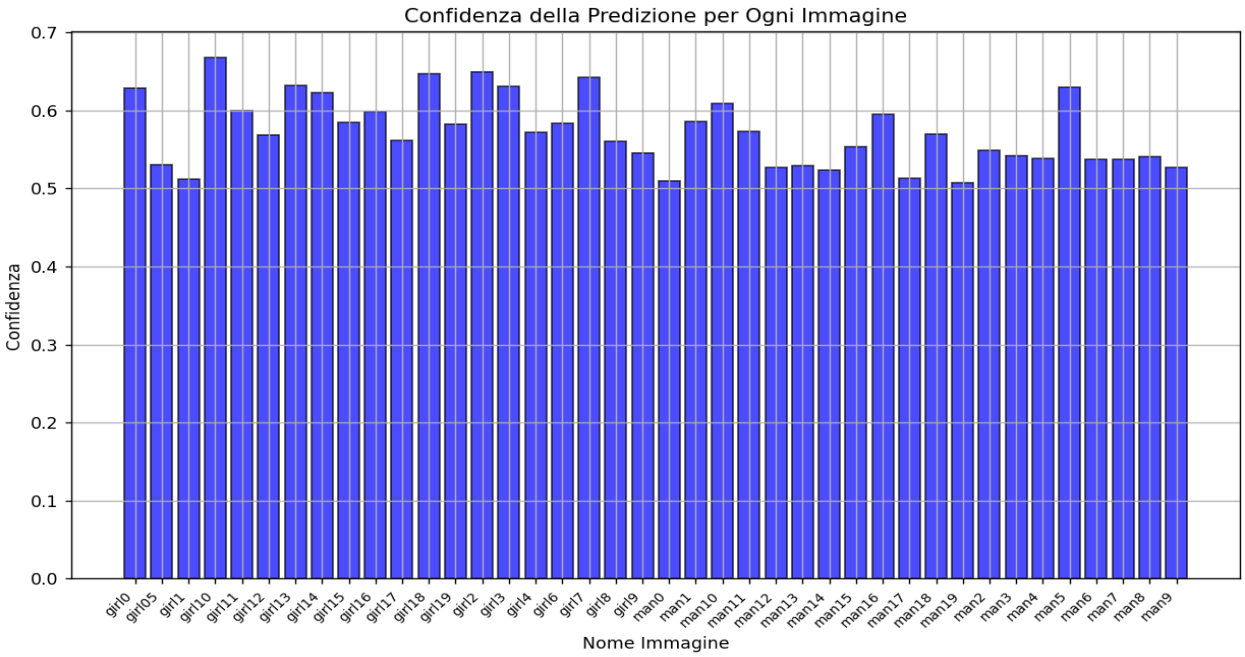


Output dell’esecuzione del file Gender.py

```
Il file config.json è valido.
      Name  Gender  Confidence
0    girl0  female    0.628036
1   girl05  female    0.530278
2    girl1  female    0.512056
3   girl10  female    0.667490
4   girl11  female    0.599375
5   girl12  female    0.568841
6   girl13  female    0.632173
7   girl14  female    0.622016
8   girl15  female    0.584757
9   girl16  female    0.598849
10  girl17  female    0.561899
11  girl18  female    0.646329
12  girl19  female    0.582207
13   girl2  female    0.649577
14   girl3  female    0.631175
15   girl4  female    0.571227
16   girl6  female    0.583255
17   girl7  female    0.642393
18   girl8  female    0.559878
19   girl9  female    0.544722
20    man0  female    0.509318
21    man1  female    0.585908
22   man10  female    0.608916
23   man11  female    0.572901
24   man12    male    0.527188
25   man13  female    0.528703
26   man14  female    0.523438
```

Ottenuti questi dati, possiamo dedurre che il modello riesce a riconoscere i due sessi in maniera discreta. Inoltre, il modello sembra essere più sicuro nella predizione delle donne rispetto agli uomini, come indicato dalle confidenze più alte e più consistenti per le immagini femminili. Potrebbe essere utile analizzare le immagini con basse confidenze per i maschi e per le femmine, e vedere se ci sono caratteristiche comuni che causano confusione al modello.

Come avviene per **Animal.py**, anche in questo caso viene generato un grafico a barre che aiuta a comprendere il comportamento del modello nell’analisi e classificazione delle immagini di questa categoria.



Criticità riscontrate

- **Errori di classificazione iniziali**: Durante i primi test, il modello commetteva errori nel distinguere tra alcune categorie, come gatti e uccelli. Per risolvere questo problema, sono stati eseguiti ulteriori cicli di addestramento e modifiche al dataset, come la sostituzione di immagini poco chiare o ambigue, per migliorare la precisione del modello. Inoltre, la scelta di cambiare modello si è rivelata fondamentale per ottenere risultati più accurati.
- **Dipendenza dalle immagini di addestramento**: Inizialmente, il modello sembrava dipendere troppo dalle immagini specifiche usate nell'addestramento, con una ridotta capacità di generalizzare su nuovi dati. Questo problema è stato affrontato ampliando e diversificando il dataset, includendo immagini con variazioni significative in termini di angolazioni e qualità.
- **Bilanciamento delle categorie**: All'inizio, alcune classi erano rappresentate in modo sproporzionato, il che portava a un bias del modello verso le categorie più frequenti. Per risolvere questo problema, è stato necessario bilanciare meglio il dataset, assicurando una rappresentazione equa di tutte le categorie per migliorare la capacità del modello di generalizzare correttamente.
- **Applicazione dei filtri alle immagini**: in un primo momento avevamo riscontrato la seguente problematica: lo stesso filtro veniva applicato a classi distinte di immagini nonostante venissero implementate diverse funzioni per differenti categorie di immagini (persone e animali). Ciò accadeva poiché usavamo una singola funzione, apply_color_filter per tutte le categorie per evitare codice ridondante (principio DRY). In questo modo, però, la rete memorizzava il primo filtro applicato, per poi propagarlo e applicarlo a tutte le foto successive senza

verificare se l’immagine apparteneva ad una classe diversa dalla prima predetta. Per risolvere abbiamo deciso di adottare un approccio diverso, basato su una serie di funzioni specifiche per ciascuna categoria che applicavano un filtro diverso in base alla classe. Nello specifico abbiamo creato la funzione process_images la quale, in un primo momento, effettua la classificazione dell’immagine e poi, in base al dato ottenuto, applica il filtro specifico. Tutto ciò grazie a costruttori di controllo di flusso ben organizzati che permettono l’applicazione del principio Don’t Repeat Yourself.

Questo approccio ha portato a una notevole riduzione degli errori di classificazione e a un miglioramento della robustezza del modello nel gestire variazioni nei dati di input.

Considerazioni finali

Le considerazioni finali ci permettono di esaminare vari aspetti del progetto:

1. **Performance del modello:** Il progetto ha dimostrato che sia ResNet18 che ResNet50 sono efficaci nella classificazione delle immagini, con ResNet50 che ha mostrato miglioramenti significativi nella precisione, specialmente con un set di dati più complesso che include immagini di persone.

2. **Impatto della qualità delle immagini:** L'importanza della qualità e della chiarezza delle immagini è stata evidenziata attraverso l'analisi dei risultati. La necessità di sostituire alcune immagini in base a fattori come la nitidezza, la visibilità dei dettagli e la posizione del soggetto, ha sottolineato l'impatto significativo che questi aspetti hanno sulle performance del modello.

3**. Limiti e futuri sviluppi:** Anche se i risultati sono stati promettenti, sono emerse alcune limitazioni, come gli errori occasionali di classificazione dovuti a somiglianze tra le classi o a fattori esterni come la qualità dell'immagine. Questi aspetti possono essere esplorati ulteriormente, magari con l'introduzione di tecniche di data augmentation o con l'uso di modelli ancora più sofisticati.

In conclusione, il progetto ha raggiunto gli obiettivi prefissati, dimostrando l'efficacia del modello ResNet50 (rispetto a ResNet18) nella classificazione di immagini complesse, pur evidenziando aree di miglioramento e potenziali direzioni per ricerche future.

Implementazioni future

Di seguito delle implementazioni che ci sarebbe piaciuto applicare al nostro progetto ma che, per mancanza di tempo e/o conoscenze, non siamo riuscite a fare:

- **Object Detection live:** attualmente il progetto si limita a riconoscere e distinguere tratti e dettagli da immagini; uno sviluppo futuro potrebbe essere

effettuare object detection attraverso la fotocamera del dispositivo per riconoscere i tratti tipici delle persone e degli animali.

- **Rilevamento oggetti:** integrare un modello di rilevamento oggetti (come YOLO e Faster R-CNN visti a lezione) per localizzare e identificare persone o animali all'interno delle immagini e non solo immagini intere
- **Object Detection on Cloud:** implementare soluzioni di scaling come l'uso di deployment su server cloud per gestire un grande numero di immagini

Fonti consultate

- Kaggle**:** Utilizzato per ottenere immagini per la categoria animali (cani, gatti e uccelli). Sono stati usati più datasets contenenti varie foto, per poi fare una selezione manuale delle immagini da inserire nelle cartelle "test" e "train".
- Spiegazione sulle ResNet: https://www.productteacher.com/quick-product-tips/resnet18-and-resnet50
- Exploring ResNet50: An In-Depth Look at the Model Architecture and Code Implementation: https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f
- Documentazione di MATLAB: https://it.mathworks.com/help/deeplearning/ref/resnet50.html
- Materiale didattico fornito dai docenti
- Python