Final project for machine learning for Vision and Multimedia course: “Shazam Anime”

Author
Alessandro Bresciani
s305903@studenti.polito.it
ale.brex99@gmail.com

Author
Luca Filippetti
s303392@studenti.polito.it
filippetti99.lf@gmail.com

Abstract

Il Deep learning ha compiuto significanti passi in avanti nella realizzazione di compiti quali la classificazione di immagini e il riconoscimento dei loro contenuti, anche se l’applicazione di quest’ultimo in ambito cinematografico e anime è ancora scarsa. Questo paper mira a descrivere l’impiego di tale tecnologia proprio per risolvere alcuni dei problemi legati a quest’ambito.
Nel corso di questo progetto, abbiamo cercato di realizzare un sistema in grado di riconoscere il titolo di un anime da un singolo frame. Attraverso la proposta di una rete Siamese-ResNet50, la quale combina una struttura di rete Siamese con la conosciuta architettura ResNet, valutiamo l’effetto del suo utilizzo partendo da un preallenamento sulla raccolta dati pubblica ImageNet. Abbiamo iniziato affrontando il problema della raccolta dati, creando un dataset adeguato nonostante le sfide legate alla disponibilità limitata di immagini. Successivamente, abbiamo sviluppato una soluzione basata sull'utilizzo di una rete neurale siamese secondo la strategia del few-shot learning, unita all'algoritmo k-Nearest Neighbor (k-NN). Comparata con l’impiego di una VGG-16, una ResNet50 mostra maggior abilità e robustezza nell’estrazione di features da diverse immagini.

1. Introduzione

L'esplosione dei contenuti multimediali ha portato ad una crescente richiesta di soluzioni di riconoscimento e classificazione di scene di film, serie tv e anime. In questo contesto, il nostro progetto "ShazamAnime" si propone di affrontare una sfida affascinante: il riconoscimento di scene Anime da immagini singole.
Nel corso di questo progetto, adotteremo un approccio basato principalmente sul paradigma del few-shot learning [1], esso si distingue dal supervised learning standard per il suo obiettivo, ovvero il learn to learn: si cerca di riconoscere la somiglianza e la differenza tra immagini e non di riconoscere il contesto preciso delle immagini del dataset e generalizzarlo per le immagini del test set. La nostra architettura si basa su una rete neurale siamese [2], che dimostra di essere potente nel riconoscimento di pattern e nello sviluppo di rappresentazioni complesse. Combineremo quest'approccio con l'uso di un algoritmo di k-Nearest Neighbor (k-NN) per ottenere risultati di alta precisione[3].
Successivamente forniremo una panoramica dettagliata del nostro progetto, delineando le sfide, l'importanza e l'approccio adottato. Nonostante le ottime prospettive di sviluppo, vanno tenute in considerazione anche le possibili limitazioni: è risaputo che per realizzare una rete neurale di tale tipologia sono necessari tempi di allenamento estremamente lunghi e un ingente mole di dati e complessità di modello. Il nostro gruppo, limitato dalle risorse computazionali a disposizione e sulla base dei contenuti appresi durante il Corso di “Machine Learning for Vision and Multimedia” ha deciso di contribuire al campo del riconoscimento di scene di generico ambito cinematografico (anime).

2. Dataset

Vista la tipologia di problema che abbiamo cercato di risolvere, è stato necessario effettuare una raccolta di dati completamente personalizzata.
Abbiamo realizzato due dataset di immagini di anime direttamente prelevate da una serie di video scaricati dalla rete (youtube).
Dopo aver completato un primo script python sfruttando la libreria numpy (data_set_extractor.py), è stato possibile acquisire frames da tutti i video scaricati a cadenza variabile (25fps, 15 fps, 10fps) così da salvare quelli più significativi anche dai video più movimentati lasciando spazio di manovra nel codice per eventuali correzioni o sovra scrizioni. Automaticamente lo script salva i frames appartenenti ad ogni specifica classe di anime, anche da video differenti, all'interno di ogni cartella relativa, per un totale di 16 classi (16 cartelle); contemporaneamente preleva randomicamente da ogni video dei frames per aggiungerli alle stesse classi, ma in una cartella separata e adibita alla fase di testing della rete.
Le classi a disposizione sono: A_Silent_Voice (203 immagini), Attack_on_Titans (208), Death_Note (206), Demon_slayer (208), Dragon_Ball (204), Fruits_Basket

(201), Full_Metal_Alchemist_Brotherhood (205), Gintama (204), Hunter_X_Hunter (202), Jujutsu_Kaisen (210), My_Hero_Academia (208), Naruto (206), One_piece (205), One_Punch_Man (205), Re_ZERO (202), Steins;Gate (206).
Successivamente, per ogni cartella formatasi, è stato fondamentale selezionare ed eliminare a mano le immagini troppo difficoltose per l'apprendimento della rete, ossia eccessivamente mosse e poco descrittive della classe in questione, oppure troppo simili tra di loro così da evitare un possibile overfitting. Inoltre, per l'eliminazione delle immagini di Test, si è trovato opportuno eliminare quelle già presenti nella cartella adibita alla realizzazione del train set, in questo modo è stata scongiurata la possibilità di vanificare l'operazione di testing.
L'intero Dataset fin qui ottenuto è stato impiegato per la fase di classificazione affidata al k-NN finale, esso corrisponde a 200/204 immagini per classe, per un totale di 3280 immagini (da cui derivano quelle del dataset per la siamese) usate per il training e 714 di test usate solo alla fine nel k-NN, circa il 20%.
Il formato delle immagini è jpeg rgb, a 24 bit di profondità, 512x512 pixels e 96 dpi di risoluzione.
Per la realizzazione del dataset specifico per la siamese, un primo script (make_siamese_dataset.py) è stato utilizzato per la costruzione di un file CSV contenente tutti i possibili accoppiamenti tra i percorsi google drive delle immagini all'interno delle cartelle. Quindi per ogni percorso di un'immagine, sono stati associati tutti quanti i percorsi, di tutte le altre immagini, legando ad ogni coppia l'etichetta 0, nel caso si trattasse della medesima classe (la stessa cartella di origine) e 1 in caso contrario. Ciò ha permesso di realizzare delle combinazioni di immagini in modo da allenare la rete siamese, la quale necessita di casi positivi e di casi negativi esplicativi della somiglianza tra anime.
L'ultimo script make_siamese_dataset_balanced.py, ci ha permesso di bilanciare il file CSV precedentemente creato. Infatti, partendo da tutti i possibili accoppiamenti, si è ottenuto un file CSV dove le coppie con etichetta negativa, corrispondenti ad anime differenti, superavano di molto quelle con etichetta positiva; perciò, il nuovo file CSV creato allinea il numero di casi negativi al numero dei positivi per favorire un allenamento rapido e corretto.
Il dataset contenuto nel file CSV (siamese_ds_balanced) per la rete siamese Resnet50 usata è stato caricato all'interno di un oggetto dataset della libreria tensorflow. In totale la dimensione del dataset conta 1344976 coppie. Successivamente, dopo averlo opportunamente mischiato, è stato processato a batch size di 128 in modo da convertire tutti quanti i cammini associati in effettive coppie di immagini. Si è selezionata la dimensione finale di 256x256 pixels per immagini (type float 32), in quanto, viste le risorse a nostra disposizione, ha permesso effettivamente di inziare gli esperimenti, altrimenti impossibili da condurre.
Il dataset è stato scomposto in una parte per il training, 1.210.496 coppie ed una per il validation, ovvero 134.498 coppie, il 10%. Invece il Test set impiegato corrisponde a quello derivante dal primo dataset, quindi quello di 714 immagini , con l'obbiettivo di applicarlo al k-NN finale.

3. Metodi

3.1. Approccio al Task di Riconoscimento di scene anime

Per affrontare il compito di riconoscimento di scene anime da immagini, abbiamo adottato un approccio basato sulla combinazione di una rete neurale siamese e un algoritmo k-Nearest Neighbor (fig. 1). Questa scelta è stata dettata dalla natura del problema, che richiedeva un modello in grado di apprendere le sottili differenze tra scene anime senza ricorrere a un addestramento intensivo di un classificatore tradizionale.
L'approccio si basa sull'idea di insegnare alla rete neurale siamese a misurare la somiglianza tra due immagini. Questo processo di apprendimento avviene attraverso la comparazione delle rappresentazioni delle immagini estratte da una backbone di rete neurale pre-allenata su ImageNet. La rete neurale siamese calcola la distanza tra le rappresentazioni delle due immagini e impara a distinguere tra immagini simili e diverse.
Successivamente, utilizziamo l'algoritmo k-NN per classificare effettivamente le immagini di test. Il vantaggio di questo approccio risiede nella sua scalabilità e nella sua capacità di gestire grandi quantità di dati senza richiedere un ulteriore addestramento.

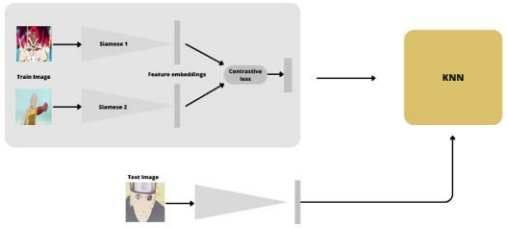

Figura 1. Schema generale dell'approccio (rete siamese e k-NN).

3.2. Architettura del Modello Siamese

L'architettura della nostra rete neurale siamese inizia con la backbone ResNet-50. Il modello ResNet-50 appartiene ad una famiglia di reti artificiali neurali conosciute come reti residuali (Residual Networks). Questa rete è stata pre-

allenata su ImageNet, il che significa che è in grado di estrarre features significative da immagini complesse. Le features estratte passano attraverso un Global Average Pooling e uno strato Dense con 512 unità, rendendo le rappresentazioni ancora più informative. Utilizziamo quindi una Lambda custom layer per calcolare la distanza euclidea tra le feature ottenute dai due rami della rete siamese, fondamentale per misurare la somiglianza tra le immagini (fig. 2).

input_1	input:	[(None, 256, 256, 3)]
InputLayer	output:	[(None, 256, 256, 3)]

input_2	input:	[(None, 256, 256, 3)]
InputLayer	output:	[(None, 256, 256, 3)]

Base_model_RESNET	input:	(None, 256, 256, 3)
Functional	output:	(None, 512)

Normalizzatore_Features	input:	(None, 512)
Lambda	output:	(None, 512)

Distanza_euclidea	input:	[(None, 512), (None, 512)]
Lambda	output:	(None, 1)

Figura 2. "Summary" dell'architettura della rete siamese usata.

3.3. Funzione di Perdita, matematica, allenamento della rete Siamese

Learning rates : 1.0/0.1 – 0.00006/0.00001 e lr adattiva
Ottimizzatore impiegato : Adam, ema_momentum 0.99
Loss function: Contrastive loss function
Batch size: 128 samples
Dimensione delle immagini in allenamento : 256x256
Modello base : Resnet50, 175 layers

Durante l'addestramento di una rete siamese vengono codificati due o più input (due nel nostro caso, essendo un one shot learning). I due rami della rete siamese condividono i pesi e vengono allenati simultaneamente. L'obbiettivo è insegnare alla rete a collocare all'interno di uno spazio di rappresentazione di feature vectors le immagini, in modo che i campioni simili risultino vicini tanto più la loro somiglianza è alta e lontani viceversa. Durante l'apprendimento della rete si è deciso di impiegare una funzione che calcoli la distanza euclidea

$$d(i,j)=\sqrt{\sum_{k=1}^{n}(x_{ik}-y_{jk})^2}$$

tra le coppie di immagini di training producendone il risultato dopo averle trasformate in embeddings. Per poter allenare la rete si è deciso di sfruttare la così detta Contrastive Loss, la cui formula completa è:

$$L_{\text{contrastive}}(Y,D) = \tfrac{1}{2}(1-Y)D^2 + \tfrac{1}{2}Y\max(0,m-D)^2$$

essa prende come input una coppia di campioni simili o dissimili, avvicina i campioni simili e allontana i campioni dissimili. Data una coppia di immagini *(I1, I2)* e un'etichetta ***Y*** pari a 0 se i campioni sono simili e pari ad 1 altrimenti. Per estrarre una rappresentazione a bassa dimensionalità di ciascun campione, utilizziamo la nostra rete ***F*** che codifica le immagini di input in uno spazio dove ***x1 = F(I1), x2 = F(I2)*** (fig.3).
Durante la backpropagation la rete cercarà di minimizzare la Loss contrastiva, quindi, se i campioni sono simili (Y=0), la D, ossia la distanza euclidea tra x1 e x2, viene minimizzata, invece se i campioni sono dissimili, (Y=1), viene minimizzato il secondo termine, ovvero sarà massimizzata la loro distanza euclidea fino ad un certo limite selezionabile ***m.***
Come ottimizzatore della rete si è scelto di impiegare Adam. Esso appartiene alla famiglia degli adaptive gradient methods che differenziano la learning rate per ogni parametro. Ad ogni istante di tempo, Adam computa la media mobile della media e della varianza dei gradienti sfruttando il momentum (strumento per aumentare la velocità). È stata fatta questa scelta in quanto, basandosi sui laboratori didattici e sui riferimenti a papers di reti siamesi, è stato definito il più adattivo e prestazionale.

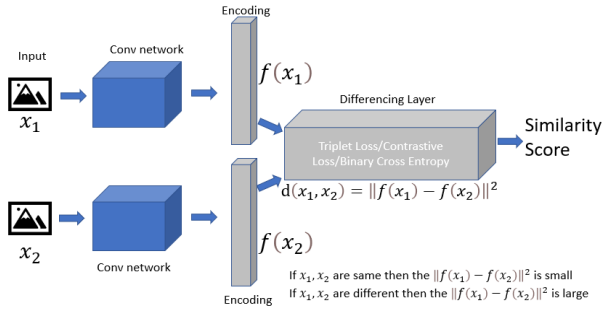

Figura 3. Modello rete siamese completa.

Per tutti gli addestramenti tentati sono state analizzate le complicazioni derivate da un cambiamento del numero di unità del layer Dense finale, aggiunto per poter estrapolare i features vectors una volta terminato l'intero processo di allenamento. Si è notato, dopo svariati tentativi, che il miglior compromesso è stato impiegare il layer Dense da 512 unità. Purtroppo, le risorse a disposizione non sono state sufficienti per impiegare layer Dense da 2048, ovvero pari all'uscita in profondità dalla Resnet (16, 16, 2048), il che avrebbe garantito una migliore accuratezza nella fase di testing finale.
Inoltre, si è deciso di mantenere il dataset di partenza particolarmente ricco di campioni a disposizione, sia per poter evitare l'underfitting della rete, sia per ottenere risultati migliori nelle performance: è stato condotto un esperimento sfruttando anche il dataset originale e non bilanciato, il che ha creato un'impossibilità nel proseguire l'addestramento visto il quantitativo di accoppiamenti esistenti. Un altro tentativo è stato condotto impiegando una riduzione del dataset per la rete siamese della metà.

Tutti gli esperimenti che consideravano una dimensione delle immagini di 512x512 pixels sono falliti, l'allenamento non era in grado di iniziare con le risorse a nostra disposizione.
Durante l'addestramento, abbiamo operato la strategia di transfert learning (TL) per migliorare l'apprendimento della rete siamese (con Resnet50). Essa consiste in una prima fase di "warm-up", in cui dopo aver costruito un base model (backbone), abbiamo eliminato i layers terminali (dense layers) della ResNet50, per poi addestrare solo i layers aggiunti della rete siamese sul nostro dataset di scene anime, ovvero un global avarage pooling, un layer Dense, un normalizzatore di features e un layer Lambda per il calcolo della distanza euclidea con una batch size di (128, 256, 256, 3). Durante questa fase è stato possibile allenare per un massimo di 2 epoche totali, impiegando la strategia, ad ogni epoca totale, di una steps per epochs pari a 256.
Sono state sperimentate una serie di configurazioni in modo da acquisire le migliori performance.
Le configurazioni utilizzate partono da una batch size di (256, 256, 256, 3), con la quale si è trovata forte difficoltà nel far progredire gli allenamenti a causa delle grandi dimensioni di quest'ultima. Pertanto, nei successivi esperimenti abbiamo ridotto la batch size a (128, 256, 256, 3) e allenato per un totale di 37 epoche (step per epochs pari a 256) attuando salvataggi ogni 128 steps tramite una callback di checkpoint in modo da aver la possibilità, in caso di interruzioni, di ricaricare nuovamente il modello e il suo stato.
Durante la fase di fine tuning della rete, i primi esperimenti sono stati eseguiti sfruttando l'intera rete completamente "unfreezed", attraverso una batch size di 32 , 128 steps per epochs e 296 epoche, con una learning rate pari a 0.1: l'esperimento è stato un fallimento e a causa delle limitate risorse disponibili (saturazione della RAM di sistema e della RAM della GPU), è stato possibile allenare solo fino all'ottava epoca, raggiungendo una Loss e valuation Loss pari a 0.22.
I tentativi successivi si sono basati su una modifica degli steps per epochs e della learning rate (ridotta fino a 0.0001), ma sono falliti allo stesso modo.
Dunque, si è deciso di sfruttare un fine tuning parziale: sono stati resi non allenabili i primi 140 layers su 175 della ResNet50, operando solo sul 65% dei parametri totali.
Anche in questo caso sono state sperimentate una serie di configurazioni cambiando batch size e learning rate. In molti casi si è optato per impiegare un decadimento esponenziale pari a 0.1 ogni 8 epoche (sulle 37 totali) e un decadimento fisso (tramite l'impiego di una callback) in modo da diminuire, sempre dello stesso tasso, la learning rate nel caso l'ottimizzatore avesse incontrato dei plateaux. Ciò è stato realizzato basandosi su varie ricerche e si è scoperto che aiuterebbe di molto l'ottimizzatore.

Gli esperimenti sono stati condotti impostando i layers di batch normalization in modalità di inferenza. Essa è una buona pratica per quando si eseguono operazioni di fine tuning. Impostando il congelamento dei layers durante la fase di warmup, i layers di Batch Normalization saranno eseguiti in modalità di inferenza e non verranno aggiornate le loro statistiche quindi, quando successivamente essi vengono scongelati, è necessario mantenere la modalità di inferenza poiché altrimenti gli aggiornamenti dei pesi non allenati potrebbero distruggere ciò che il modello ha appreso a causa della presenza di due diverse distribuzioni nei layers.

In questa versione la fase di warmup è stata condotta con batch size da 128, step per epochs da 256, per un totale di 37 epoche e una learning rate a decadimento esponenziale di 0.1 partendo da 0.1 come valore iniziale e terminando a 0.00001. Per il fine tuning si è usata invece una learning rate a decadimento esponenziale di 0.1 partendo da 0.0001. Abbiamo ottenuto una loss di 0.236 per il warmup e di 0.221 per il fine tuning (fig. 4).

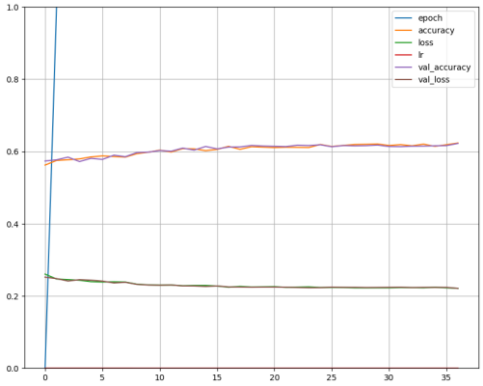

Figura 4. Grafico che mostra l'andamento dei parametri.

Un secondo esperimento è stato eseguito con i medesimi parametri, ma tentando di sfruttare una lr pari a 1.0, con decadimento ogni 15 epoche fino a 0.1/0.001 ed ogni 2 epoche consecutive di staticità della valuation loss.
Gli stessi iperparametri sono stati applicati nella fase di Fine tuning.

Da questi allenamenti di fine tuning si è notato come la loss divergesse se il lr di partenza fosse alto, probabilmente questa problematica ci mostra come il modello sia instabile e non completamente adatto a risolvere il task.

Si è inoltre deciso di confrontare gli esperimenti precedenti con una nuova versione dell'architettura, ovvero provando a porre, contro indicazioni, i layers di bath normalization in condizioni di training durante warmup e fine tuning.
Abbiamo concluso la fase di warmup allenando la rete con una lr pari a 0.00006 fino alla ventesima epoca, per poi

concludere le epoche a lr 0.00001: il risultato è stato deludente, la Loss non è calata al di sotto di 0.23, probabilmente a causa del limite di complessità del modello in questa fase.
Ulteriori esperimenti finali sarebbe possibile condurli modificando l'ottimizzatore. Attraverso attente ricerche si è scoperto un altro possibile ottimizzatore oltre ad Adam, efficiente nelle reti siamesi, ovvero Nadam. Diversamente un ottimizzatore SGD non darebbe risultati ottimali.

3.4. Model surgery e classificazione

Dopo l'addestramento della rete siamese, abbiamo effettuato un "model surgery" per estrarre uno dei due rami della siamese, utilizzandolo come estrattore di features.(fig. 5) Il k-Nearest Neighbor (k-NN) è un algoritmo di classificazione di apprendimento supervisionato non parametrico che utilizza la prossimità per effettuare previsioni. Dato un vettore da etichettare si cercano i 'k' vettori nel train set che sono "più vicini" al vettore test. Si assegna poi l'etichetta più frequente tra le 'k' etichette. Per calcolare la vicinanza degli elementi abbiamo usato una distanza euclidea:
d:={(x1−y1)^2+(x2−y2)^2}^1/2.
La scelta di utilizzare l'algoritmo k-NN è stata dettata da varie considerazioni: innanzi tutto il numero di classi o titoli nel nostro contesto è relativamente elevato, soprattutto in funzione di quello che sarebbe un sistema realmente praticabile, il che avrebbe reso l'addestramento di un classificatore dense una sfida complessa in termini di risorse necessarie. In un ambito in cui i nuovi titoli anime possono emergere frequentemente, la capacità del k-NN di adattarsi rapidamente a nuove classi senza richiedere una procedura di addestramento completa lo rende una scelta più pragmatica. Un'altra considerazione importante è stata la criticità dei tempi di inferenza, infatti se il k-NN può risultare più pesate rispetto ad un classificatore dense, questa differenza è trascurabile per applicazioni in cui la priorità principale è la precisione nella classificazione.

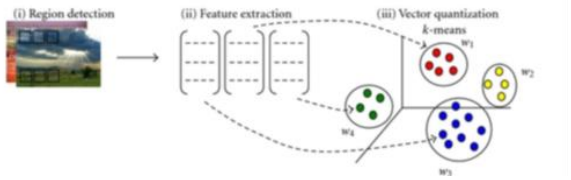

Figura 5. Schema ad alto livello di un feature extraction e k-NN.

3.5. Valutazione delle prestazioni e metriche

Per valutare l'efficacia abbiamo utilizzato diverse metriche di valutazione, tra queste una delle più informative è stata la matrice di confusione, oltre che ad un semplice valore di accuratezza. La matrice di confusione confronta, per ciascuna classe, i valori previsti dal modello con quelli reali esistenti nei dati. In particolare, le righe della matrice rappresentano le previsioni del modello e le colonne rappresentano i dati reali. Grazie alla matrice è infatti possibile capire quali sono le classi che il modello confonde maggiormente.
Per ottenere una visualizzazione dettagliata delle feature estratte abbiamo adottato l'algoritmo t-Distributed Stochastic Neighbor Embedding (t-SNE). Il t-SNE è una tecnica di riduzione della dimensione ampiamente utilizzata per visualizzare dati in uno spazio a dimensioni inferiori preservando al massimo la struttura dei dati originali, nel nostro caso tre dimensioni. L'utilizzo del t-SNE ci ha fornito una chiara rappresentazione delle feature estratte dalla rete siamese e ha facilitato l'analisi visiva delle relazioni tra le immagini del dataset, in particolare ci ha aiutato a verificare se le feature fossero state apprese in modo efficace (fig. 6) e se l'approccio fosse in grado di creare rappresentazioni distinte per le diverse classi anime.

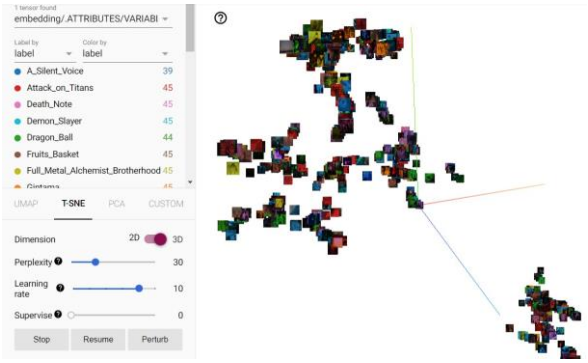

Figura 6. Screen dei cluster del T-sne della "ShazamAnime", (le immagini hanno una colorazione in base alla propria etichetta).

4. Esperimenti e risultati

Per valutare i risultati finali della nostra rete abbiamo deciso di confrontare il modello allenato con una rete baseline ResNet50 non allenata sul nostro dataset, ma solo pre-allenata su Imagenet. I risultati purtroppo non sono soddisfacenti. Infatti, come si può vedere dalla confusion matrix (fig. 7) e l'accuracy di 0.08 (fig. 8) la rete non è in grado di distinguere le varie classi.

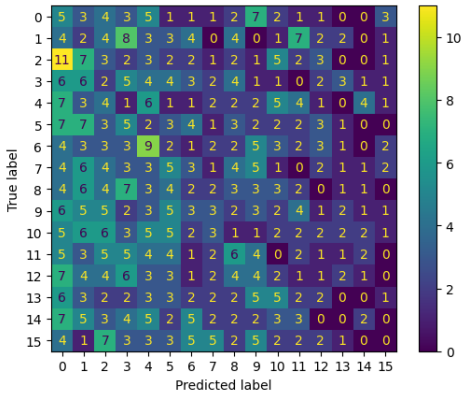

Figura 7. Risultati confusion matrix della "ShazamAnime".

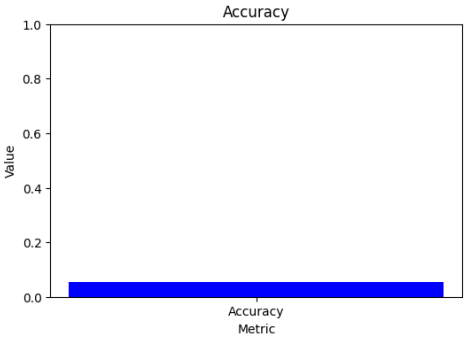

Figura 8. Accuracy della "ShazamAnime".

È stato inoltre realizzato, come pura curiosità, un ulteriore esperimento per effettuare la classificazione finale delle immagini sostituendo al k-NN un layer Dense provvisto di funzione di attivazione softmax. Una volta tagliato uno dei rami della siamese è stato necessario inserire il nuovo layer Dense. È stato condotto un allenamento a batch size 128 e learning rate fissa pari a 0.01 (da una loss: 2.73 a 2.5; val_loss: 2.7 a 2.53; accuracy: 0.10 a 0.21) per le prime 10 epoche. Successivamente sono stati effettuati altri allenamenti: 24 epoche a learning rate fissa (0.001) e 38 epoche a lr variabile (decadimento esponenziale) partendo da 0.001 con step ogni 8 epoche fino a 0.00001.

Epoca 10	Lr = 0.01	Loss.i = 2,73	Loss.f = 2.5	Acc=0.21
Epoca 74	Lr: 0.001-0.000001	Loss.i = 2.47	Loss.f = 2.46	Acc = 0.24
From scratch: Epoca 38(1)	Lr: 0.1 – 0.00001	Loss.i = 2.7	Loss.f = 2.26	Acc = 0.32
From scratch: Epoca 128 (2)	Lr: 0.01	Loss.i = 2.04	Loss.f = 1.99	Acc = 0.39
From scratch: Epoca 256 (3)	Lr: 0.1 – 0.01 Batch size 256	Loss.i = 2.7	Loss.f = (1.84) 1.73	Acc = (0.47) 0.48

Dopo aver allenato tale classificatore sono stati analizzati i 2 migliori risultati ottenuti rispetto a quelli del k-NN:

X	knn	Dense classif_1	Dense classif_3
accuracy	0.08	0.074	0.061
loss	X	2.26	1.73
Valuation_loss	X	2.32	1.87

La matrice di confusione ottiene circa gli stessi risultati del k-NN. In ogni caso, seppur senza successo, il k-NN dà il risultato migliore.

5. Conclusioni

L'allenamento di un modello in grado di riconoscere il titolo di un anime da immagini statiche è stato un compito impegnativo. La natura stessa del task, che richiede la comprensione del contenuto dell'immagine in assenza di contesto, ha reso necessario un dataset ampio e diversificato. Inoltre, le risorse computazionali limitate di Google Colab hanno rappresentato una sfida nell'allenamento di una rete neurale complessa.
È importante sottolineare come la precisione del modello e la performance sia migliorabile.
Sarebbe logico impiegare dei feature vectors di grandezza pari almeno alla dimensione di uscita della backbone impiegata (2048), in quanto ciò aumenterebbe la precisione delle features. Inoltre, la dimensione delle immagini è stata notevolmente ridotta, dei buoni risultati si potrebbero ottenere impiegando immagini a risoluzione elevata, come 1024x1024, in modo da permettere alla rete un ottimale apprendimento delle caratteristiche più specifiche (riconoscimento di angoli, bordi, ecc.) e più generiche (occhi, visi, arti, ecc.) di ogni rappresentazione di anime.
Miglioramenti evidenti si avrebbero scongelando completamente l'intera rete ResNet50 ed allenando tutti quanti i parametri durante la fase di fine tuning. Tale operazione darebbe origine ad una rete completamente deep, la quale apprenderebbe elementi a maggior complessità, mentre per risolvere il problema del vanishing e dell'exploding dei gradienti e della variabilità dei pesi, sarebbe necessario impiegare una regolarizzazione di questi ultimi.
Essendo partiti da un dataset estremamente denso di campioni, non è stato possibili a livello computazionale, attuare una tecnica di data augmentation; essa sarebbe raccomandata per esperimenti futuri in modo da ridurre un possibile overfitting.

6. References

[1] Yinbo Chen, Zhuang Liu, Huijuan Xu, Trevor Darrell, Xiaolong Wang <<*Meta-Baseline: Exploring Simple Meta-Learning for Few-Shot Learning*>> arXiv:2003.04390v4, 2021.
[2] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov <<*Siamese Neural Networks for One-shot Image Recognition*>> Department of Computer Science, University of Toronto. Toronto, Ontario, Canada, 2015.
[3] Benjamin Villard, Yuichi Mori, Masashi Misawa, Shin-ei Kudo, Hayato Itoh, Masahiro Oda, Kensaku Mori <<*Colorectal Polyp Size Classification Using a Siamese Network*>>, 2019.
[4] github: One-shot Siamese Neural Network: https://github.com/nevoit/Siamese-Neural-Networks-for-One-shot-Image-Recognition.