Implementazione di una Convolutional Neural Network per la classificazione di razze di cane

Luca Arrotta

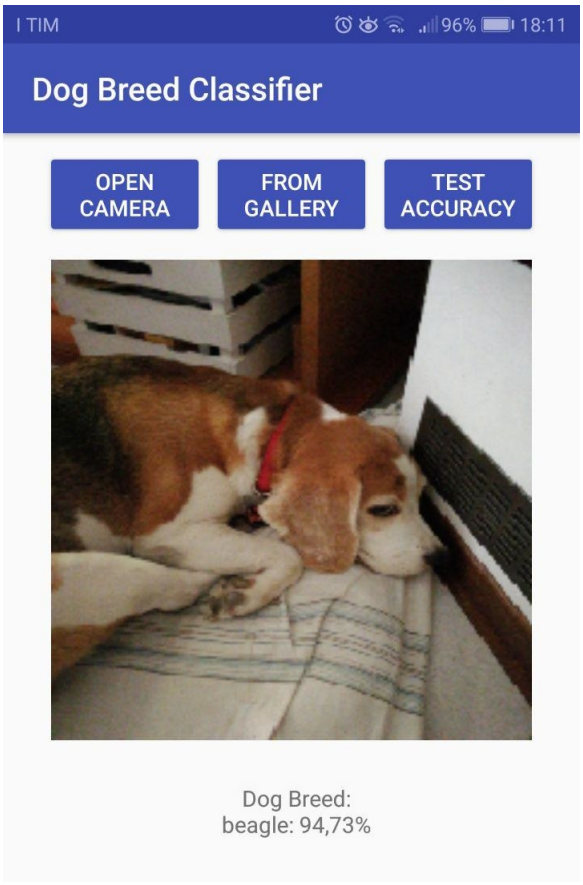

Fig. 1: Schermata principale dell'applicazione Android sviluppata al termine del progetto.

1 Introduzione

In questo progetto è stata implementata in Python una Convolutional Neural Network (CNN) per la classificazione di razze di cane a partire da immagini. Tale lavoro si basa su un altro progetto presente su Github [1], che, utilizzando la libreria Keras in Python, implementa una CNN con una test accuracy pari al 51.07%. L'obiettivo di questo lavoro è stato quello di sviluppare una CNN che riuscisse a migliorare le performance di quella del progetto preso come riferimento. Infine, anche se non necessario ai fini del progetto, la CNN implementata è stata utilizzata all'interno di un'applicazione Android (Figura 1).

L. Arrotta, Corso di Metodi Statistici per l'Apprendimento, A/A 2017-2018, Universitá di Milano, via Comelico 39/41, Milano, Italy
E-mail: luca.arrotta@studenti.unimi.it

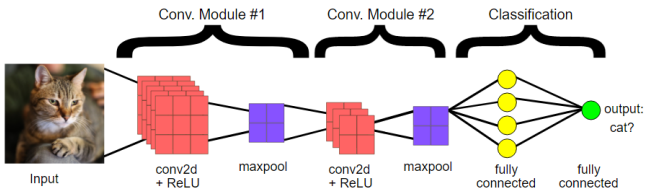

Fig. 2: Un esempio di CNN.

2 Analisi dei dataset disponibili

Online sono disponibili due dataset che permettono di implementare un classificatore di razze di cani.

Il primo dataset considerato è lo Stanford Dog Dataset [2] [3], che include più di 20 mila immagini etichettate (prelevate dal database ImageNet) appartenenti a 120 razze di cane differenti. La piattaforma online Kaggle nel 2017 ha lanciato la competizione "Dog Breed Identification" [4] che utilizza questo primo dataset. Il progetto preso come riferimento [1] presenta una soluzione a tale competizione, sviluppando una CNN a partire da zero. Su Github si trovano anche altri progetti relativi alla competizione lanciata da Kaggle, ma molti di essi utilizzano CNN pre-addestrate (come ad esempio il progetto [5]) e quindi non sono stati considerati per lo sviluppo di questo lavoro.

Il secondo dataset considerato è reso disponibile da Udacity, un'organizzazione che eroga diversi corsi online. Questo dataset è a disposizione per uno dei progetti che fanno parte del corso online di Intelligenza Artificiale [6]. Tale dataset include più di 8 mila immagini appartenenti a 133 razze di cane differenti.

3 Convolutional Neural Network

Le CNN sono una classe di Reti Neurali ampiamente utilizzate nell'ambito del riconoscimento e della classificazione di immagini [7]. Una CNN riceve in input un'immagine e, tramite una serie di strati (layer), impara ad estrarre le feature di tale immagine e a riconoscere quale oggetto è presente in essa [8].

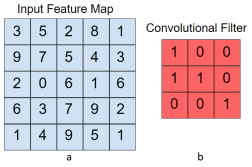

Fig. 3: a) input della convoluzione b) filtro da applicare

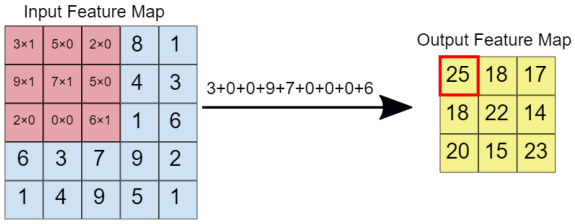

Fig. 4: applicazione del filtro all'input

Per l'estrazione delle feature, la CNN utilizza Moduli di Convoluzione (Convolution Module) che tipicamente effettuano tre operazioni sull'input:

- convoluzione
- funzione di attivazione Rectified Linear Unit (ReLU)
- pooling

Per la classificazione dell'immagine, invece, la CNN utilizza layer chiamati Fully Connected. La Figura 2 mostra un esempio di CNN.

3.1 Layer di una CNN

3.1.1 Convolution Module

Convoluzione

L'operazione di convoluzione applica dei filtri all'input che riceve, per estrarre nuove feature da esso. L'applicazione di ogni singolo filtro genera in output una feature map diversa. L'output della convoluzione è la composizione delle diverse feature map ottenute in questo modo. Quindi, il primo Convolution Module riceverà in input un'immagine del dataset, mentre i successivi moduli riceveranno come input l'output generato dal modulo precedente della CNN.

La convoluzione è definita principalmente da tre parametri:

- la dimensione dei filtri da applicare all'input (tipicamente 3x3 o 5x5 pixel);
- la profondità dell'output, che corrisponde al numero di filtri che si vogliono applicare;
- lo stride, ovvero di quanti pixel viene fatto scorrere il filtro lungo l'input ad ogni passo della convoluzione.

Consideriamo di voler applicare all'input di Figura 3a il filtro 3x3 di Figura 3b con un valore di stride pari a 1. La Figura 4 mostra come avviene la convoluzione: viene fatto scorrere il filtro lungo l'input e, per ogni posizione, viene calcolato il prodotto tra ogni valore del filtro ed il corrispettivo pixel dell'input. I risultati di queste moltiplicazioni vengono sommati per ottenere un singolo valore nella feature map di output.

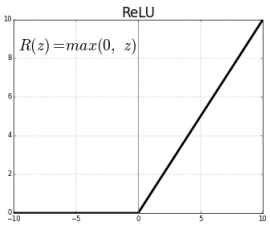

Fig. 5: La funzione di attivazione ReLU

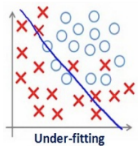

Fig. 6: Esempio di underfitting in un classificatore binario. Il modello è troppo semplice per classificare al meglio i dati.

Valori diversi all'interno del filtro produrranno feature map diverse in output. Maggiore è il numero di filtri che applichiamo, maggiore sarà quindi il numero di feature che verranno estratte dalla CNN. Durante la fase di training, la CNN impara i valori dei filtri, in modo da riuscire ad estrarre dall'input delle feature significative (texture, forme, bordi, ...).

Rectified Linear Unit (ReLU)

La ReLU è attualmente la funzione di attivazione più utilizzata nell'ambito delle CNN. Le funzioni di attivazione in una CNN trasformano l'output della convoluzione prima di passarlo al nodo successivo del modello.

Le funzioni di attivazione servono ad introdurre non-linearità all'interno del modello. Un modello è lineare se il suo output è una combinazione lineare degli input che riceve [9]. Un modello lineare non sarebbe in grado di estrarre dalle immagini feature non lineari, producendo il problema noto come underfitting (vedi Figura 6). Per questo motivo vengono utilizzate le funzioni di attivazione [10].

La ReLU mappa i valori negativi dell'input a zero e ne mantiene i valori positivi (Figura 5). Nell'ambito delle CNN, questa funzione di attivazione è molto diffusa perchè consente un addestramento più rapido ed efficace del modello [11].

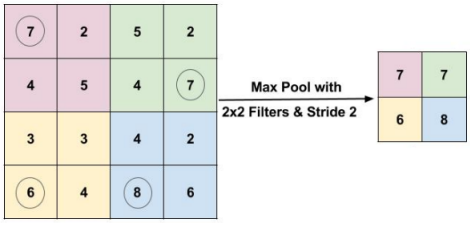

Fig. 7: Applicazione dell'operazione di Max Pooling

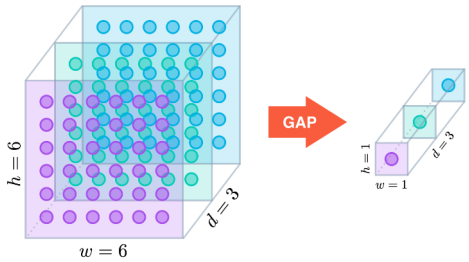

Fig. 8: Funzionamento di un layer GAP.

Pooling

L'operazione di Pooling riduce la dimensionalità di ogni feature map, mantenendo le informazioni più importanti sulle feature. Con questa operazione, si riduce il numero di parametri del modello da addestrare, alleggerendo la computazione. Inoltre, un minor numero di parametri permette di ridurre l'overfitting (vedi Sezione 3.2).

Uno degli algoritmi di Pooling più comuni è il Max Pooling, in cui si definisce un filtro che viene fatto scorrere lungo l'input. Ad ogni posizione del filtro, viene selezionato l'elemento dell'input maggiore tra quelli che si trovano all'interno di tale filtro. Tutti i valori selezionati in questo modo faranno parte dell'output dell'operazione.

Il Max Pooling considera principalmente due parametri:

- la dimensione del filtro da applicare all'input (tipicamente 2x2 pixel);
- lo Stride.

Un esempio di Max Pooling è mostrato nella Figura 7.

3.1.2 Fully Connected Layer

Dopo i Convolution Module, di solito sono presenti uno o più Fully Connected Layer. In un Fully Connected Layer ogni nodo è connesso ad ogni altro nodo degli strati del modello adiacenti. Lo scopo di questi layer è quello di classificare l'immagine sulla base delle feature estratte dai Convolution Module.

Tipicamente, l'ultimo Fully Connected Layer contiene una funzione di attivazione Softmax, il cui output è un valore di probabilità compreso tra 0 e 1 per ogni classe che il modello deve imparare a riconoscere. La somma di queste probabilità è pari a 1.

3.1.3 Global Average Pooling (GAP)

Gran parte dei parametri di una CNN sono legati alla presenza dei layer Fully Connected. Negli ultimi anni si è iniziato a sostituire i Fully Connected Layer con layer chiamati Global Average Pooling (GAP), con lo scopo di minimizzare l'overfitting riducendo il numero di parametri del modello [12].

Come i pooling layer, i GAP riducono la dimensionalità delle feature map, ma in maniera più estrema: dato un input di dimensioni h x w x d, esso viene ridotto a dimensioni 1 x 1 x d (vedi Figura 8). I layer GAP riducono così ogni feature map h x w ad un singolo numero, ottenuto prelevando il valore medio tra tutti gli elementi della feature map stessa.

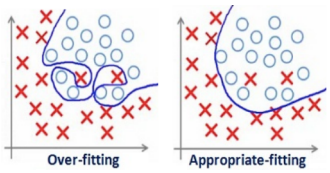

Fig. 9: Esempio di overfitting e appropriate-fitting per un classificatore binario.

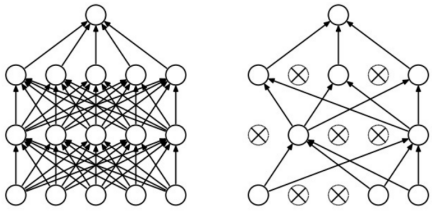

Fig. 10: Sinistra: CNN senza l'applicazione del Dropout. Destra: stessa CNN dopo l'applicazione del Dropout.

3.2 Riduzione dell'Overfitting

Si verifica overfitting quando il modello si adatta troppo ai dati del training set e non riesce a classificare correttamente nuovi dati. Il modello diventa più incline all'overfitting quando il dataset è troppo piccolo rispetto al numero di parametri che devono essere addestrati [13] [14]. In Figura 9 è mostrato un esempio di overfitting ed uno di appropriate-fitting per un classificatore binario.

Le tecniche principali per ridurre l'overfitting quando si costruisce una CNN sono:

- Regolarizzazione L1 e L2 (Weight Regularization)
- Regolarizzazione Dropout
- Data Augmentation

3.2.1 Regolarizzazione L1 e L2

Modelli semplici sono meno inclini all'overfitting rispetto a quelli più complessi. Un modo per mitigare l'overfitting consiste nell'inserire dei vincoli alla complessità del modello, forzando i suoi pesi ad essere valori piccoli (per questo si parla di Weight Regularization). Per ottenere ciò, la loss function della CNN viene penalizzata quando vengono usati pesi con valori troppo grandi [15].

Nel caso della Regolarizzazione L1, la penalizzazione è proporzionale al valore assoluto dei pesi. Nel caso della Regolarizzazione L2, la penalizzazione è proporzionale al quadrato del valore dei pesi.

3.2.2 Dropout

Il Dropout, applicato ad un layer della CNN, consiste nel selezionare casualmente alcuni nodi di tale layer e disattivarli durante la fase di training [16] (vedi Figura 10). L'idea è quella di ridurre la dipendenza tra i nodi del layer durante l'addestramento e, di conseguenza, incrementare la robustezza del modello.

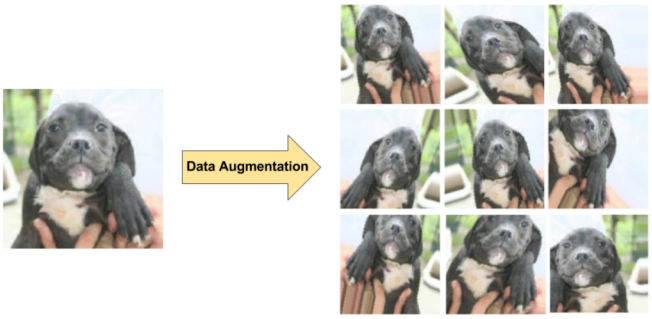

Fig. 11: Sinistra: immagine presente nel training set. Destra: nove nuove immagini generate a partire dall'originale.

3.2.3 *Data Augmentation*

Il Data Augmentation consiste nell'aumentare la diversità ed il numero di dati all'interno del training set. Ciò viene fatto applicando trasformazioni (rotazione, flipping, zoom, ...) alle immagini già presenti nel training set (Figura 11). Il Data Augmentation è utile soprattutto quando il training set è relativamente piccolo [17].

3.3 Epoche e Batch Size

Durante la fase di training, i dati del training set vengono divisi in gruppi (batch). Tali batch vengono passati uno ad uno alla CNN. Questo è necessario poichè non è possibile passare l'intero training set al modello in un unico passo vista la dimensione dei dati. Ad ogni step, i pesi della CNN vengono aggiornati sulla base dei dati presenti nel batch passato al modello [18].

In una singola Epoca della fase di training, l'intero training set viene passato alla CNN tramite un certo numero di batch. Il Batch Size è il numero di esempi presi dal training set presenti in un singolo batch.

3.4 Batch Normalization [19]

Viene definita Internal Covariate Shift la variazione nella distribuzione degli input che riceve ogni singolo layer della rete. Ciò è causato dal fatto che l'input di ogni layer è influenzato dal cambiamento dei parametri nei layer precedenti. Questo implica che i layer del modello devono continuamente adattarsi sulla base della distribuzione dell'input che ricevono. Maggiore è il numero di layer del modello, più sarà amplificato il problema dell'Internal Covariate Shift.

L'idea della Batch Normalization è quella di rendere fissa la distribuzione dell'input che riceve un layer del modello. Questa soluzione riduce l'Internal Covariate Shift ed accelera la fase di training. In particolare, la Batch Normalization consiste nel rendere la media e la varianza degli input che riceve un layer rispettivamente pari a 0 e ad 1.

Inoltre, la Batch Normalization regolarizza il modello, riducendo il bisogno di utilizzare altre tecniche di regolarizzazione.

```
model = Sequential()
model.add(BatchNormalization(input_shape=(224, 224, 3)))
model.add(Conv2D(filters=16, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(Conv2D(filters=32, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(Conv2D(filters=64, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(Conv2D(filters=256, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(GlobalAveragePooling2D())

model.add(Dense(133, activation='softmax'))

model.summary()
```

Fig. 12: Architettura del modello implementato nel progetto di riferimento.

4 Analisi del progetto preso come riferimento [1]

Il progetto preso come riferimento implementa una CNN a partire da zero, utilizzando lo Stanford Dog Dataset e raggiungendo una test accuracy pari al 51.07%. In Figura 12 è mostrato il codice Python utilizzato per l'implementazione dell'architettura di tale CNN tramite la libreria Keras. Ecco alcune precisazioni su questo codice, sulla base di quanto spiegato nella Sezione 3:

- ”input_shape” specifica che in input al modello verranno passate immagini di 224x224 pixel con 3 livelli colore;
- ”filters” indica il numero di filtri che si vogliono applicare durante la convoluzione;
- ”kernel_size” indica la dimensione dei filtri da applicare all'input durante l'operazione di convoluzione (”kernel_size=3” implica l'applicazione di filtri 3x3);
- ”kernel_initializer” specifica quale tecnica si vuole utilizzare per l'inizializzazione dei pesi del layer. L'Inizializzatore di He [20] è la tecnica di inizializzazione più utilizzata quando la funzione di attivazione del layer è la ReLU;
- ”pool_size” specifica la dimensione del filtro da applicare all'input durante l'operazione di pooling (”pool_size=2” implica l'applicazione di filtri 2x2);
- ”Dense” indica un Fully Connected Layer. Il parametro ”133” specifica il numero di unità di tale layer.

Per quanto riguarda il Data Augmentation, il progetto di riferimento effettua le seguenti operazioni sulle immagini durante la fase di training:

- shift orizzontale verso destra o sinistra fino al 10% della larghezza dell'immagine stessa;
- shift verticale verso l'alto o il basso fino al 10% dell'altezza dell'immagine stessa;
- flip orizzontale dell'immagine.

La fase di training del progetto preso come riferimento è composta da:

- 20 epoche con batch size pari a 20
- 5 epoche con batch size pari a 64.

5 Implementazione

Nella seguente lista sono elencati i passi seguiti durante lo sviluppo di questo lavoro:

1) selezione di 30 delle 120 razze di cane presenti nel primo dataset considerato; alle immagini relative a queste 30 razze sono state aggiunte ulteriori immagini prelevate dal secondo dataset considerato, ottenendo un nuovo dataset che da qui in poi verrà chiamato "dataset esteso";
2) training di un modello utilizzando il dataset esteso e gli stessi parametri descritti nella Sezione 4;
3) analisi del migliore batch size da utilizzare durante la fase di training;
4) analisi per capire come migliorare le performance modificando il Data Augmentation rispetto al progetto di riferimento;
5) analisi per capire come migliorare le performance modificando l'architettura del modello rispetto al progetto di riferimento;

5.1 Dataset Esteso

Come detto precedentemente, dallo Stanford Dog Dataset sono state prelevate le immagini relative a 30 delle 120 razze di cane presenti. A queste, sono state aggiunte ulteriori immagini prelevate dal secondo dataset considerato. Non tutte le 30 razze selezionate dal primo dataset erano presenti nel secondo dataset, di conseguenza il dataset esteso è sbilanciato rispetto al numero di immagini a disposizione per ogni singola razza. Ad esempio, sono disponibili 269 immagini di Beagle, ma solo 152 immagini di Rottweiler.

Il dataset esteso presenta un totale di 6.542 immagini per 30 razze di cane differenti. Per l'implementazione del modello, tale dataset è stato suddiviso nella seguente maniera:

- l'80% del dataset è stato dedicato al training set;
- un 10% del dataset è stato dedicato al validation set;
- un'ulteriore 10% del dataset è stato dedicato al test set.

Tale suddivisione è stata effettuata utilizzando la funzione train_test_split della libreria Scikit-learn.

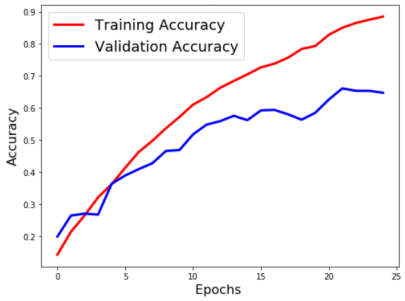

Fig. 13: Confronto tra training e validation accuracy dopo aver provato gli stessi parametri del progetto di riferimento con il dataset esteso.

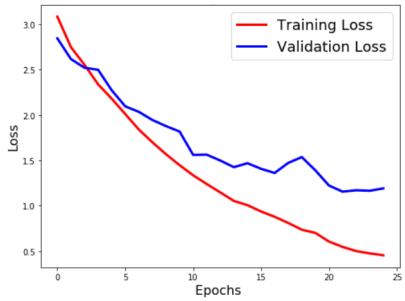

Fig. 14: Confronto tra training e validation loss dopo aver provato gli stessi parametri del progetto di riferimento con il dataset esteso.

5.2 Progetto di riferimento e Dataset Esteso

Utilizzando il dataset esteso, è stato addestato un modello con gli stessi parametri del progetto di riferimento, descritti nella Sezione 4. L'unica differenza rispetto a quanto visto nella Sezione 4 è che, in questo modello, il Fully Connected Layer dell'architettura presenta 30 unità invece di 133.

Il modello sviluppato in questo modo ha raggiunto una test accuracy pari al 62.07%. Lo scopo di questo lavoro è quello di migliorare tale valore. In Figura 13 è mostrato il confronto tra l'andamento della training accuracy e quello della validation accuracy dopo la fase di training di tale modello. In Figura 14 è invece mostrato il confronto tra la training loss e la validation loss.

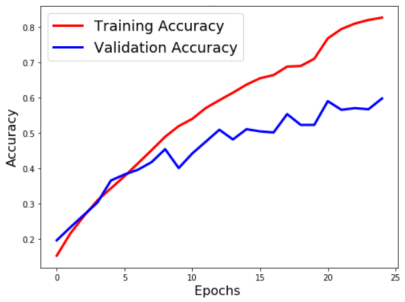

Fig. 15: Confronto tra training e validation accuracy con i parametri del progetto di riferimento durante l'analisi del batch size.

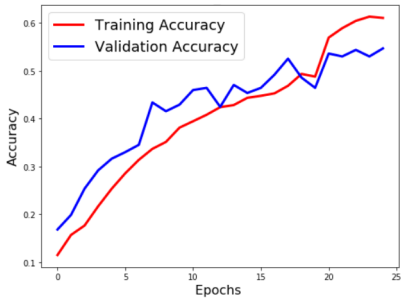

Fig. 16: Confronto tra training e validation accuracy nell'esperimento con le migliori performance durante l'analisi del batch size.

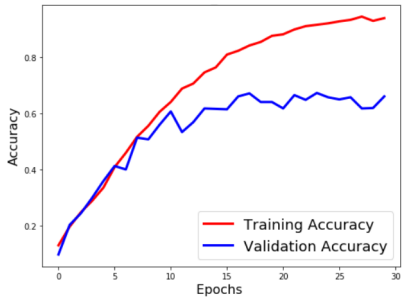

Fig. 17: Confronto tra training e validation accuracy con i parametri del progetto di riferimento durante l'analisi del Data Augmentation.

5.3 Analisi del Batch Size

Utilizzando sempre la stessa suddivisione del dataset in training, validation e test set, e mantenendo fissi i parametri del Data Augmentation e dell'architettura, sono stati addestrati diversi modelli. Le loro performance sono state analizzate sulla base del numero di epoche e dei batch size usati durante le rispettive fasi di training.

Durante questi esperimenti, le performance migliori sono state ottenute effettuando una prima fase di training con un batch size molto basso, aumentandolo nelle successive fasi. In Figura 15 è mostrato il confronto tra training e validation accuracy utilizzando i parametri del progetto di riferimento: 20 epoche con batch size pari a 20, seguite da 5 epoche con batch size pari a 64. In Figura 16 è mostrato lo stesso confronto dopo aver effettuato la fase di training con i seguenti parametri: 20 epoche con batch size pari a 4, seguite da 5 epoche con batch size pari ad 8. Si può notare che in questo caso la validation accuracy non subisce un grosso cambiamento, ma viene ridotto l'overfitting: la training accuracy presenta infatti valori molto più vicini a quelli della validation accuracy rispetto al grafico di Figura 15.

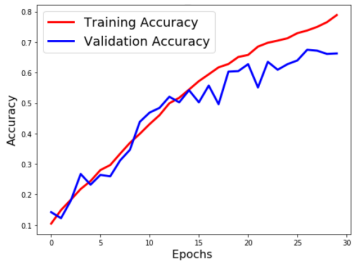

Fig. 18: Confronto tra training e validation accuracy nell'esperimento con le migliori performance durante l'analisi del Data Augmentation.

5.4 Analisi del Data Augmentation

Per effettuare Data Augmentation è stata utilizzata la funzione ImageDataGenerator della libreria Keras. Durante questa analisi sono stati sperimentati i seguenti parametri:

- rotation_range: specifica fino a quanti gradi ruotare casualmente l'immagine;
- width_shift_range: specifica fino a quanto spostare casualmente l'immagine verso destra o sinistra rispetto alla sua larghezza;
- height_shift_range: specifica fino a quanto spostare casualmente l'immagine verso l'alto o il basso rispetto alla sua altezza;
- shear_range: specifica l'intensità di shear da applicare all'immagine.
- zoom_range: specifica fino a quanto ingrandire o rimpicciolire l'immagine casualmente.
- horizontal_flip: effettua casualmente il flip orizzontale dell'immagine originale.

In questo caso, sono stati addestrati diversi modelli modificando solo i parametri della funzione ImageDataGenerator. In Figura 17 è mostrato il confronto tra training e validation accuracy dopo il training di un modello che utilizza gli stessi parametri di Data Augmentation del progetto di riferimento:

- width_shift_range = 0.1
- height_shift_range = 0.1
- horizontal_flip = True

In Figura 18 è mostrato lo stesso confronto per l'esperimento che ha condotto alle migliori performance. Tale esperimento ha utilizzato i seguenti parametri:

- width_shift_range = 0.2
- height_shift_range = 0.2
- horizontal_flip = True
- rotation_range = 30
- shear_range = 0.05

Anche in questo caso, tra i due grafici è possibile notare una buona riduzione dell'overfitting.

5.5 Analisi dell'Architettura del modello

Mentre nelle analisi precedenti lo scopo principale è stato quello di ridurre l'overfitting, in questi esperimenti ci si è concentrati sull'aumento della test accuracy. Per

```
input_shape=(200, 200, 3)
nClasses = 30

model = Sequential()
model.add(BatchNormalization(input_shape=input_shape))
model.add(Conv2D(filters=16, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(Conv2D(filters=64, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=512, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(Conv2D(filters=512, kernel_size=3,
     kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(GlobalAveragePooling2D())

model.add(Dense(nClasses, activation='softmax'))
```

Fig. 19: Architettura del modello che ha condotto ai risultati migliori.

questo, sono state sperimentate diverse architetture più complesse rispetto a quella del progetto di riferimento (Figura 12).

In Figura 19 è mostrato il codice Python utilizzato per implementare, tramite la libreria Keras, l'architettura del modello che ha portato alle performance migliori. Rispetto all'architettura del progetto di riferimento, si può notare che la differenza principale consiste nell'aumento del numero dei Convolutional Layer (Conv2D in keras).

L'architettura del progetto di riferimento utilizzata con il dataset esteso ha portato il numero di parametri da addestrare a poco più di 400 mila. L'architettura mostrata in Figura 19, invece, ha portato il numero di parametri da addestrare a più di 3 milioni.

All'architettura di Figura 19, sono stati anche aggiunti layer di Dropout e regolarizzatori L1 e L2. Queste aggiunte non hanno però condotto a risultati migliori.

Come si può notare dal codice di Figura 19, in input a questo modello vengono date immagini di dimensioni 200x200 e non più immagini di dimensioni 224x224 come nel caso del progetto di riferimento. Questo cambiamento è stato necessario per evitare il verificarsi di Memory Error durante la fase di training.

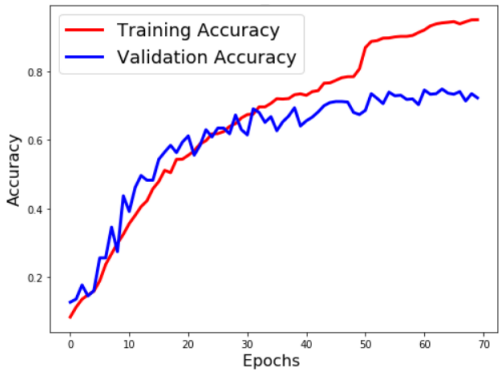

Fig. 20: Confronto tra training e validation accuracy dopo il training del modello migliore.

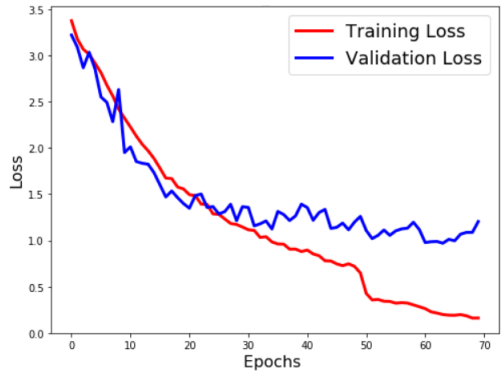

Fig. 21: Confronto tra training e validation loss dopo il training del modello migliore.

6 Risultati Ottenuti

6.1 Modello Migliore

Di seguito, sono elencate le caratteristiche del modello che ha portato ai risultati migliori.

- Architettura: quella mostrata in Figura 19
- Data Augmentation:
  - width_shift_range = 0.2
  - height_shift_range = 0.2
  - rotation_range = 15
  - shear_range = 0.05
  - horizontal_flip = True
- Fase di Training:
  - 50 epoche con batch size pari a 4
  - 10 epoche con batch size pari a 8
  - 10 epoche con batch size pari a 16

6.2 Risultati del Modello Migliore

I risultati migliori del modello descritto nella Sezione 6.1 sono stati ottenuti con i pesi aggiornati dopo l'epoca numero 64. Con questi pesi, il modello ha raggiunto una test accuracy pari al 77.67%.

In Figura 20 è mostrato il confronto tra gli andamenti di training e validation accuracy durante il training di questo modello. In Figura 21 è invece mostrato il confronto tra training e validation loss.

6.3 Progetto di riferimento e suddivisione "migliore" del dataset esteso

La suddivisione del dataset in training, validation e test set può influenzare le performance del modello. Per questo motivo, la suddivisione del dataset è stata salvata dopo la fase di training del modello migliore. Un nuovo modello è stato addestrato utilizzato questa suddivisione del dataset e gli stessi parametri del progetto di riferimento, descritti nella Sezione 4. Tale modello ha raggiunto una test accuracy pari al 64.22%.

6.4 Applicazione Android

Come detto nella Sezione 1, il modello migliore è stato inserito all'interno di un'applicazione Android (Figura 1). Per farlo è stato necessario

- salvare il modello nel formato h5;
- convertirlo nel formato .pb tramite un apposito programma sviluppato nel repository [21];
- importarlo all'interno dell'applicazione per poi usarlo tramite la classe TensorFlowInferenceInterface della libreria Tensorflow.

REFERENCES

[1] S. Vikram, "Dog breed classification stanford dog dataset," https://github.com/saksham789/DOG-BREED-CLASSIFICATION-STANFORD-DOG-DATASET/blob/master/dog_classifier_Scratch.ipynb, 2018.

[2] Stanford dog dataset. [Online]. Available: http://vision.stanford.edu/aditya86/ImageNetDogs/

[3] Novel dataset for fine-grained image categorization: Stanford dogs. [Online]. Available: https://pdfs.semanticscholar.org/b5e3/beb791cc17cdaf131d5cca6ceb796226d832.pdf

[4] Kaggle, dog breed identification. [Online]. Available: https://www.kaggle.com/c/dog-breed-identification

[5] K. Panarin, "Dog breeds classification," https://github.com/stormy-ua/dog-breeds-classification, 2017.

[6] Udacity, convolutional neural network project. [Online]. Available: https://github.com/udacity/dog-project

[7] Use of convolutional neural network for image classification. [Online]. Available: https://www.apsl.net/blog/2017/11/20/use-convolutional-neural-network-image-classification/

[8] Ml practicum: Image classification. [Online]. Available: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

[9] Introduction to neural networks: Anatomy. [Online]. Available: https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/anatomy

[10] Deep convolutional neural networks for image classification: A comprehensive review. [Online]. Available: https://www.mitpressjournals.org/doi/pdf/10.1162/neco_a_00990

[11] Rete neurale convoluzionale. [Online]. Available: https://it.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[12] Global average pooling layers for object localization. [Online]. Available: https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/

[13] An overview of regularization techniques in deep learning. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

[14] Regularization in deep learning. [Online]. Available: https://chatbotslife.com/regularization-in-deep-learning-f649a45d6e0

[15] Explore overfitting and underfitting. [Online]. Available: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit

[16] Dropout layer. [Online]. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/dropout_layer.html

[17] Ml practicum: Image classification. [Online]. Available: https://developers.google.com/machine-learning/practica/image-classification/preventing-overfitting

[18] Epoch vs batch size vs iterations. [Online]. Available: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[21] A. Abdi, "keras to tensorflow," https://github.com/amir-abdi/keras_to_tensorflow, 2018.