RELAZIONE PROGETTO DI METODI QUANTITATIVI PER L'INFORMATICA

"Implementazione di un algoritmo di few-shot learning per Duckietown"

Michela Proietti - 1739846

11/07/2020

ABSTRACT

In questa relazione, si effettua un esteso confronto tra due diversi algoritmi di few-shot learning, il cui scopo è la classificazione di immagini di cui si possiedono pochi samples, da utilizzare nell'ambito del progetto Duckietown. In particolare, uno dei due algoritmi fa uso della semplice cross-entropy loss, mentre l'altro utilizza la triplet loss e il metric learning.

PRELIMINARIES

Nei due algoritmi di classificazione a confronto, sono state utilizzate due diverse funzioni di costo: la categorical cross-entropy e la triplet loss.

La **cross-entropy loss** misura le performance di un modello di classificazione, il cui output è una probabilità, il cui valore sarà quindi compreso tra 0 e 1. Essa è definita come segue:

$$CE = -\sum_{i=0}^{m} y_i \log(\hat{y}_i) \tag{1}$$

dove y_i è il groudtruth, $\hat{y}_i$ è la predizione del modello e m è il numero di classi. In genere, prima di dare il risultato del modello in input alla loss function, a tale valore viene applicata un'activation function, perciò potremmo scrivere $f(\hat{y}_i)$. Se la funzione di attivazione utilizzata è la Softmax, parleremo di categorical crossentropy. Nel caso di multiclass classification, che è quello da noi considerato, solamente un elemento nel vettore y , corrispondente alla classe a cui l'istanza considerata appartiene, sarà diverso da 0, perciò solo tale classe, che indicheremo con C_p, manterrà il proprio termine nella funzione di costo. Dunque, partendo dalla formula della softmax, definiamo la **categorical cross-entropy** come segue:

$$f(\hat{y}_i) = \frac{e^{\hat{y}i}}{\sum_{j=0}^{m} e^{\hat{y}j}} \quad \Longrightarrow \quad CE = -\log \frac{e^{\hat{y}i}}{\sum_{j=0}^{m} e^{\hat{y}j}} \tag{2}$$

La **triplet loss** è invece utilizzata nell'ambito del metric learning: tali metodi usano dei mini-batches di samples, chiamati episodi, per addestrare una rete neurale che individui la relazione tra diverse features. Si estraggono quindi degli embeddings dalle immagini di training e test mediante una rete convoluzionale e dopodiché ogni query, cioè ogni immagini priva di label, viene classificata in base alla sua distanza dalle immagini che dispongono di label. Durante l'addestramento, dopo ogni episodio, i parametri della CNN sono aggiornati tramite la backpropagation della loss risultante dall'errore di classificazione sul query set. Lo scopo della triplet loss è quello di fare in modo che due samples con la stessa label abbiano degli embeddings vicini nello spazio degli embeddings e che due immagini con label diverse abbiano embeddings lontani. Per non creare clusters troppo piccoli, si richiede solamente che dati due esempi positivi e uno negativo di una certa classe, quello negativo risulti essere più lontano di quello positivo di un certo margine. Per formalizzare questa richiesta, la loss è definita su triplette di embeddings:

un'ancora, uno positivo della stessa classe dell'ancora e uno negativo di una diversa classe. La loss di una tripletta (a,p,n) è definita come segue:

$$\mathcal{L} = \max\left(d(a,p) - d(a,n) + margin, 0\right) \quad (3)$$

La loss va minimizzata, il che spinge d(a,p) verso lo 0 e d(a,n) ad essere più grande di d(a,p)+margin. Non appena n diventa un "easy negative", la loss diventa 0. In base alla definizione della loss, ci sono 3 categorie di triplette: le easy triplets hanno una loss pari a 0, perché $d(a,p) + margin < d(a,n)$; le hard triplets, per le quali l'esempio negativo è più vicino all'ancora di quello positivo, cioè $d(a,n) < d(a,p)$; le semi-hard triplets, per le quali l'esempio negativo non è più vicino all'ancora rispetto a quello positivo, ma si ha ancora una loss positiva $d(a,p) < d(a,n) < d(a,p) + margin$. I diversi tipi di triplette danno contributi diversi alla buona riuscita dell'algoritmo, perciò è necessario effettuare triplet mining.

Per l'ottimizzazione della funzione di costo, si è utilizzato il metodo **Adam** (adaptive momentum estimation), che sfrutta i vantaggi sia del momento che del RMSProp, permettendo di arrivare velocemente all'ottimo smorzando contemporaneamente le oscillazioni.

$$v_t = \beta_1 * v_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * {g_t}^2$$

$$\Delta w_t = -\eta * \frac{v_t}{\sqrt{s_t}} * g_t$$

$$w_{t+1} = w_t + \Delta w_t \quad (4)$$

La prima equazione fa riferimento al metodo più legato al momento, mentre la seconda fa più riferimento al RMSProp, perciò mediante gli iperparametri β_1 e β_2, possiamo stabilire quale dei due metodi usare maggiormente. Come vediamo, nell'aggiornare i pesi, il learning rate è moltiplicato per un termine legato a quanto appena detto: il termine del RMSProp tende a smorzare il learning rate, mentre il momento tende ad aumentare il learning rate e questo ci permette di arrivare più velocemente verso l'ottimo, ma di muoverci più lentamente una volta arrivati vicino a esso.

Altre formule utili sono quelle riguardanti i vari metodi di regolarizzazione usati:

$$\text{NORMA L1} = \|x\|_1 = \sum_{i=1}^{n} |x_i| \quad (5)$$

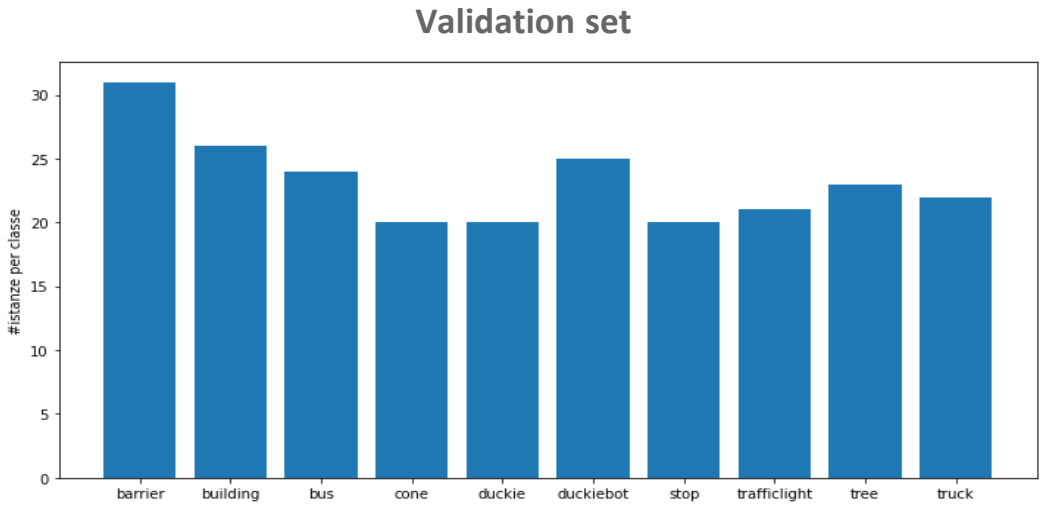
$$\text{NORMA L2} = \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \quad (6)$$

$$\text{COSINE SIMILARITY} = \cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\|\vec{b}\|} \quad (7)$$

SELECTED DATASET

Per questo progetto, è stato realizzato manualmente un piccolo dataset con immagini recuperate tramite il simulatore di Duckietown. Il dataset è composto da 283 immagini di dimensioni diverse, ognuna delle quali contenente uno solo degli oggetti di interesse, catturato da diversi punti di vista. Il numero di classi considerate è pari a 10, perciò 50 immagini sono state destinate al training set e le restanti 233 al test set, utilizzato anche come validation. Dalla tabella e tramite gli istogrammi sottostanti è possibile vedere nel dettaglio come le immagini siano distribuite tra le diverse classi.

Class Name	Training Set	Validation Set
barrier	5	32
building	5	26
bus	5	24
cone	5	20
duckie	5	20
duckiebot	5	25
stop	5	20
trafficlight	5	21
tree	5	23
truck	5	22



METHOD

Il modello che, nel corso dei vari esperimenti effettuati, fornisce i risultati migliori è il seguente:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            multiple                  3584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 multiple                  0
_________________________________________________________________
dropout_2 (Dropout)          multiple                  0
_________________________________________________________________
conv2d_3 (Conv2D)            multiple                  295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 multiple                  0
_________________________________________________________________
dropout_3 (Dropout)          multiple                  0
_________________________________________________________________
global_max_pooling2d_1 (Glob multiple                  0
_________________________________________________________________
flatten_1 (Flatten)          multiple                  0
_________________________________________________________________
dense_2 (Dense)              multiple                  32896
_________________________________________________________________
dense_3 (Dense)              multiple                  1290
=================================================================
Total params: 332,938
Trainable params: 332,938
Non-trainable params: 0
_________________________________________________________________
```

I primi due livelli della rete sono livelli convoluzionali, che applicano alle immagini dei filtri 3*3 e utilizzano come funzione di attivazione la ReLU, che fa sì che un nodo resti disattivato nel caso in cui il valore dell'input risulti essere minore di 0, altrimenti restituisce direttamente tale valore. È stata scelta questa

funzione di attivazione perché permette di fare i calcoli in maniera molto veloce e ha fornito i risultati migliori in fase di test. Vi è poi un Dense 128, avente come funzione di attivazione la ReLU, e un Dense con 10 nodi, pari al numero delle classi considerate, avente funzione di attivazione softmax, dal momento che la loss scelta per l'addestramento è la categorical cross-entropy, indicata per problemi di multiclass classification. Come metodo di ottimizzazione è stato scelto l'Adam, perché permette di arrivare velocemente all'ottimo, ma anche di smorzare notevolmente le oscillazioni durante il processo di ottimizzazione della loss. È stato inoltre implementato l'early stopping, per evitare di andare in overfitting, monitorando la validation loss e controllando che questa continuasse a diminuire nel corso del training.

Un altro approccio utilizzato è stato quello del metric learning, che prevede l'utilizzo di un encoder, rappresentato da una semplice rete convoluzionale, per individuare una metrica di distanza in base alla quale samples appartenenti alla stessa classe risulteranno essere vicini nel nuovo spazio, mentre samples di classi diverse risulteranno essere lontani. In questo caso, si è utilizzato un encoder costituito da 4 livelli convoluzionali:

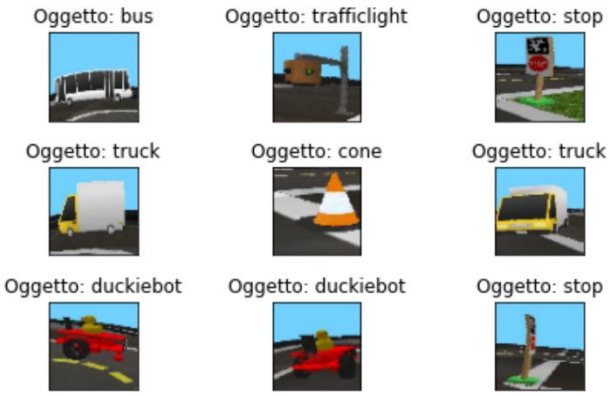
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
global_max_pooling2d_1 (Glob (None, 256)               0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
=================================================================
Total params: 388,416
Trainable params: 388,416
Non-trainable params: 0
_________________________________________________________________
```

A tale encoder è stato poi aggiunto un layer Dense di 10 nodi, con funzione di attivazione softmax, per pre-addestrare l'encoder utilizzando la categorical cross-entropy. Dopodiché, sono stati fatti vari esperimenti costruendo modelli diversi, aggiungendo all'encoder due ulteriori livelli di diverso tipo e facendo uso di diversi tipi di regolarizzazione.

Nelle prossime sezioni del report indicheremo i due diversi algoritmi con SA (simple algorithm) e MLA (metric learning algorithm).

IMPLEMENTATION

In fase di preprocessing, in entrambi gli algoritmi sono stati effettuati il ridimensionamento delle immagini a 64*64 e il rescaling. Sono stati fatti dei test in cui si effettuavano operazioni di data augmentation, in particolare il flipping orizzontale e la rotazione delle immagini entro un range di 30 gradi, ma questo non portava ad un miglioramento significativo dei risultati nel caso del SA. Nell'immagine qui accanto, è possibile visionare alcuni samples randomici del dataset, dopo il preprocessing.



EXPERIMENTS / EXPERIMENTAL SETUP / EXPERIMENT PROCEDURES

Per quanto riguarda l'algoritmo MLA, che fa uso di metric learning, dopo la fase di preprocessing è stato addestrato direttamente l'encoder avente la struttura mostrata nella sezione method di questa relazione, facendo uso della semplice categorical cross-entropy (si veda formula (2)). Per questa ragione, è stato necessario aggiungere un layer Dense di 10 nodi avente funzione di attivazione softmax. Dopodiché, utilizzando l'encoder preaddestrato, sono stati effettuati esperimenti in cui si utilizzano loss e regolarizzazioni diverse. In particolare, triplet loss con regolarizzazione tramite norma l1, norma l2, mixed norms (norma l1 e l2 con funzione di attivazione relu), cosine_similarity, learnt norms (funzione di attivazione ReLU), mixed similarity (norme l1 e l2 con funzione di attivazione sigmoid) e learnt similarity (funzione di attivazione sigmoid) e cross-entropy loss con regolarizzazione tramite cosine similarity, mixed similarity e learnt similarity. Inoltre, in alcuni degli esperimenti viene aggiunto all'encoder un ulteriore

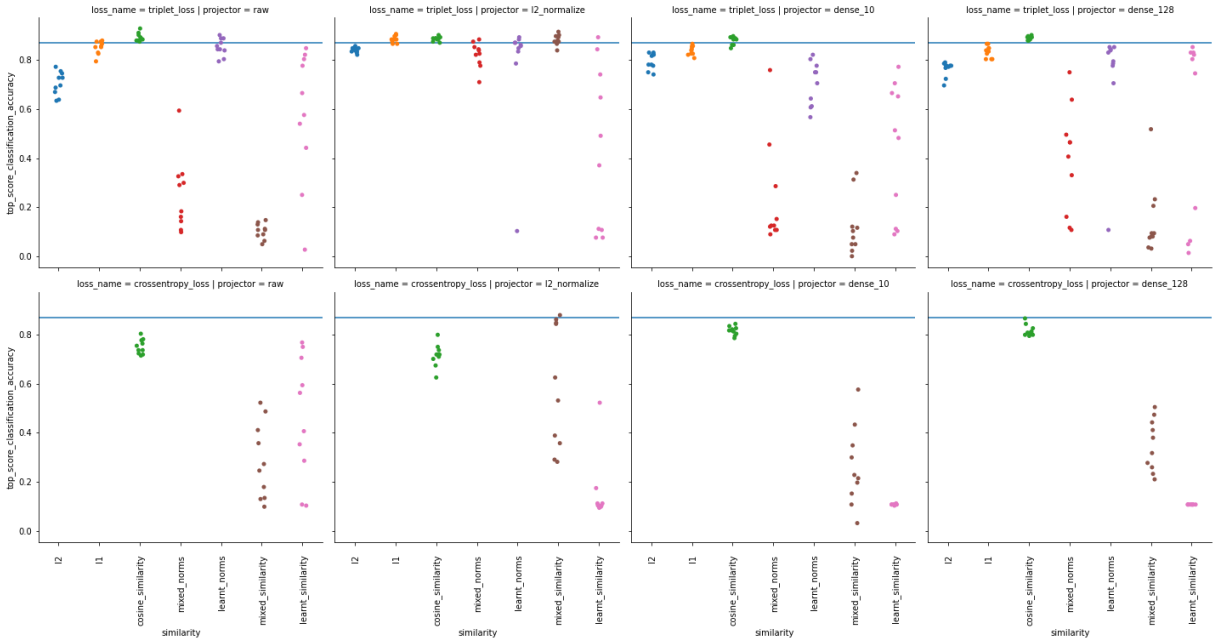
livello, definito projector, che può essere un Dense 10, un Dense 128 o una normalizzazione tramite norma l2. Per effettuare tali esperimenti, sono state utilizzate le funzioni definite nella directory keras_fsl della repository *https://github.com/few-shot-learning/Keras-FewShotLearning*. In particolare, come metrica è stata utilizzata la classification accuracy definita proprio per tali esperimenti, che considera l'accuracy più alta ottenuta nel corso dell'addestramento. Tuttavia, andando ad addestrare e successivamente testare il modello usando l'accuracy classica, si nota un'evidente discrepanza tra i risultati ottenuti con le due metriche e, come verrà mostrato più in dettaglio nella prossima sezione del report, i cattivi risultati ottenuti nella classificazione risulteranno essere più in linea con i bassissimi valori ottenuti mediante l'utilizzo dell'accuracy tradizionale.

Per quanto riguarda, invece, il SA, si è partiti da una rete convoluzionale avente la stessa struttura dell'encoder utilizzato nell'algoritmo precedente, al quale sono stati inizialmente aggiunti un Dense 256 con funzione di attivazione ReLU e un layer Dense 10 con funzione di attivazione softmax dal momento che, trattandosi di un problema di multiclass classification, era stata scelta come loss la categorical cross-entropy. Volendo utilizzare un modello che fosse il più leggero possibile, ho provato a ridurre il numero di livelli convoluzionali utilizzati e il numero di nodi al loro interno, arrivando alla struttura mostrata precedentemente. Inoltre, per non rischiare che il modello andasse in overfitting, è stato impostato un dropout di 0.3, che nel corso delle varie prove effettuate è risultato essere il valore che permetteva di raggiungere i risultati migliori. I parametri dell'Adam sono stati impostati nel seguente modo: learning rate pari a 0.001, β_1 pari a 0.5, per fare in modo che il learning rate non crescesse troppo rapidamente, e β_2 pari a 0.999. Il batch size è stato impostato a 5 e il numero di epoche a 150, ma è stato implementato l'early stopping con una patience di 10 epoche, per evitare che il modello facesse overfitting, essendo però sicuri di non fermarsi troppo presto con l'addestramento. Una patience pari a 5 faceva sì che il training terminasse troppo presto, mentre una patience più alta di 10 faceva continuare l'addestramento senza che si ottenesse alcun miglioramento nei risultati sul validation set. Inoltre, aumentando il batch size e diminuendo il numero di epoche, si riscontrava un peggioramento nei risultati. In questo caso, è stata utilizzata come metrica l'accuracy, che è pari al rapporto tra i risultati corretti e il numero totale dei risultati forniti dal nostro algoritmo, dal momento che il nostro scopo è quello di classificare correttamente il maggior numero di samples.

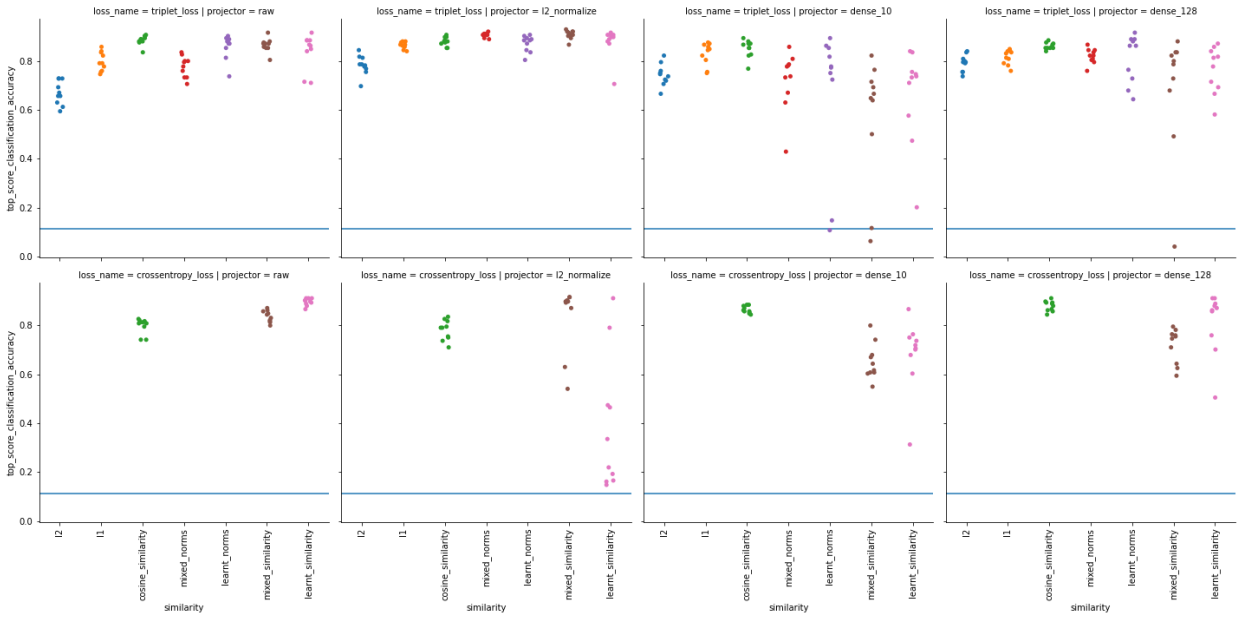
Per effettuare gli esperimenti è stato utilizzato Google Colab, perciò è stato possibile misurare i tempi necessari per la classificazione in fase di testing sia utilizzando semplicemente la CPU che la GPU.

RESULTS

Il MLA ha fornito, nei vari esperimenti effettuati, i seguenti risultati:



Nei grafici sulla prima e seconda riga rispettivamente, troviamo la top score classification accuracy dei modelli addestrati mediante triplet loss e cross-entropy loss, utilizzando i diversi tipi di projector: partendo da sinistra abbiamo il caso senza projector, quello con la normalizzazione l2 dell'output, il Dense 10 e il Dense 128. I diversi colori, invece, rappresentano i diversi tipi di regolarizzazione usati, indicati in basso lungo l'asse x. Nei grafici in basso, è possibile vedere i risultati ottenuti negli stessi esperimenti, avendo però effettuato data augmentation.



Risulta evidente che, nel caso del MLA, il data augmentation porti a un miglioramento dei risultati sia che si usi la triplet loss sia che si usi la cross-entropy loss. Mi sono dunque concentrata sull'esperimento che sembrava dare buoni risultati sia con che senza data augmentation e che implicava l'utilizzo di una rete più leggera, ovvero quello che fa uso della triplet loss, della cosine similarity come metodo di regolarizzazione e non prevede l'utilizzo del projector (nei grafici appena visti corrisponde a quello verde nella subplot in alto a sinistra). Nelle tabelle sottostanti, è possibile vedere i risultati ottenuti: risulta evidente che i valori assunti dall'accuracy sono ben diversi da quelli assunti dalla classification accuracy.

	Classification_accuracy	Accuracy
Test loss	0.0200045108795166	0.020448246970772743
Test accuracy	0.8616071343421936	0.0357142873108387

CPU time (seconds)	GPU time (seconds)	Correctly classified samples	Incorrectly classified samples
1.2590	0.0449	6	227

Come si vede dalla tabella qui accanto, i pessimi risultati della classificazione sul test set sembrano dare ragione ai valori dell'accuracy classica.

Per quanto riguarda il SA, la tabella sottostante riporta i risultati ottenuti con i vari modelli testati. In particolare, sono stati utilizzate le due seguenti reti, alle quali sono state effettuate via via delle modifiche.

Initial_model

```
model = Sequential(
  [
    Conv2D(32, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=2),
    Dropout(0.3),
    Conv2D(64, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=2),
    Dropout(0.3),
    Conv2D(128, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=2),
    Dropout(0.3),
    Conv2D(256, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=2),
    Dropout(0.3),
    GlobalMaxPooling2D(),
    Flatten(),

    Dense(256, activation='relu'),
    Dense(10, activation='softmax')
  ]
)
```

Final_model

```
model = Sequential(
  [
    Conv2D(128, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=2),
    Dropout(0.3),
    Conv2D(256, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=2),
    Dropout(0.3),
    GlobalMaxPooling2D(),
    Flatten(),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
  ]
)
```

Model	Test loss	Test accuracy	Correctly classified samples	Incorrectly classified samples
Test 1	0.8802	0.7511	175	58
Test 2	0.8482	0.8156	190	43
Test 3	0.4362	0.8970	209	24
Test 4	0.3086	0.9185	214	19
Test 5	0.3807	0.9313	217	16
Test 6	0.1680	0.9485	221	12

Test 1: initial_model con batch_size=50 e epochs=100

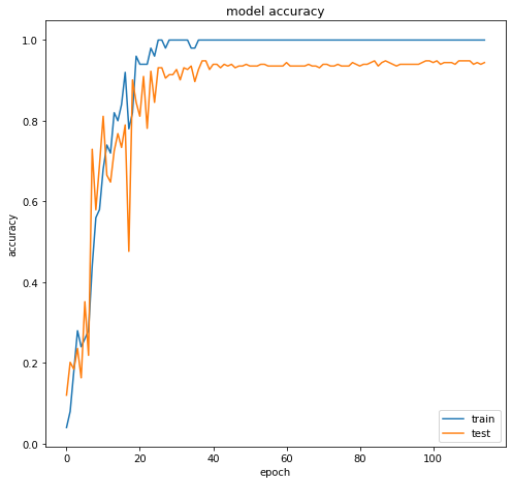
Test 2: initial_model con dropout pari a 0.2

Test 3: come in test 2, ma con il parametro β_1 dell'Adam impostato a 0.5

Test 4: final_model con dropout pari a 0.2 e penultimo livello Dense avente 256 nodi

Test 5: final_model

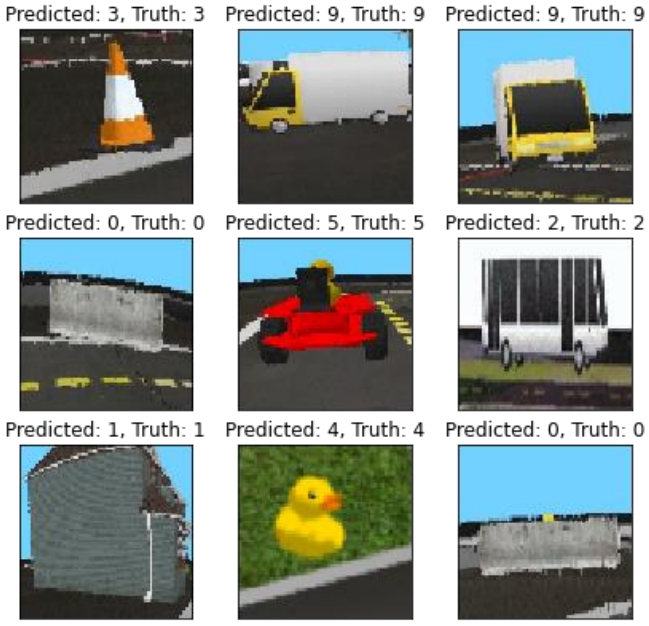
Test 6: final_model con batch_size=5, epochs=150 ed early stopping



Relativamente all'ultimo test, viene riportato qui accanto un grafico rappresentante l'andamento delle accuracy e della loss per il training e il validation in relazione al numero di epoche e in basso una tabella con i tempi impiegati in fase di testing utilizzando CPU e GPU.

CPU time (seconds)	GPU time (seconds)
4.9954139999999825	0.06369300000000067

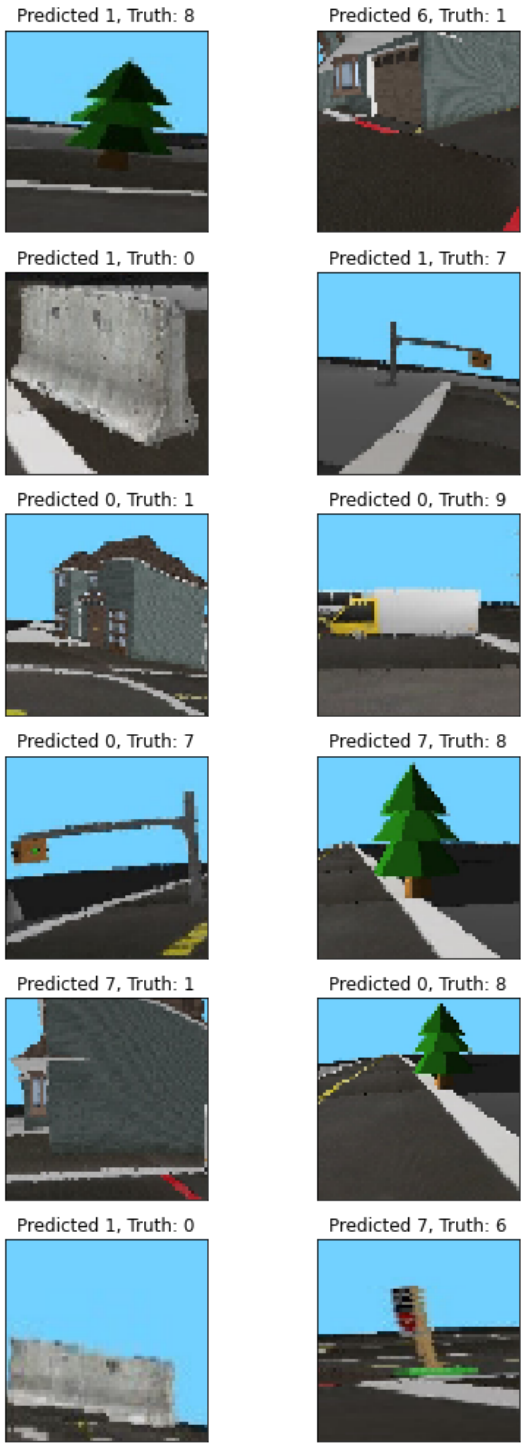
Nella pagina seguente, vengono riportati in alto a sinistra esempi di classificazioni corrette e a destra le 12 immagini classificate in maniera errata. Tutti gli errori vengono compiuti relativamente alle classi barrier, building, trafficlight, truck e tree. In particolare, l'errata classificazione degli alberi sembra essere legata al fatto che essi siano più in ombra e quindi risultino più scuri e simili allo sfondo. Per quanto riguarda i semafori e il camion, l'errore potrebbe essere dovuto al fatto che i pixel dello sfondo vengano considerati come feature rilevanti per la classificazione, e



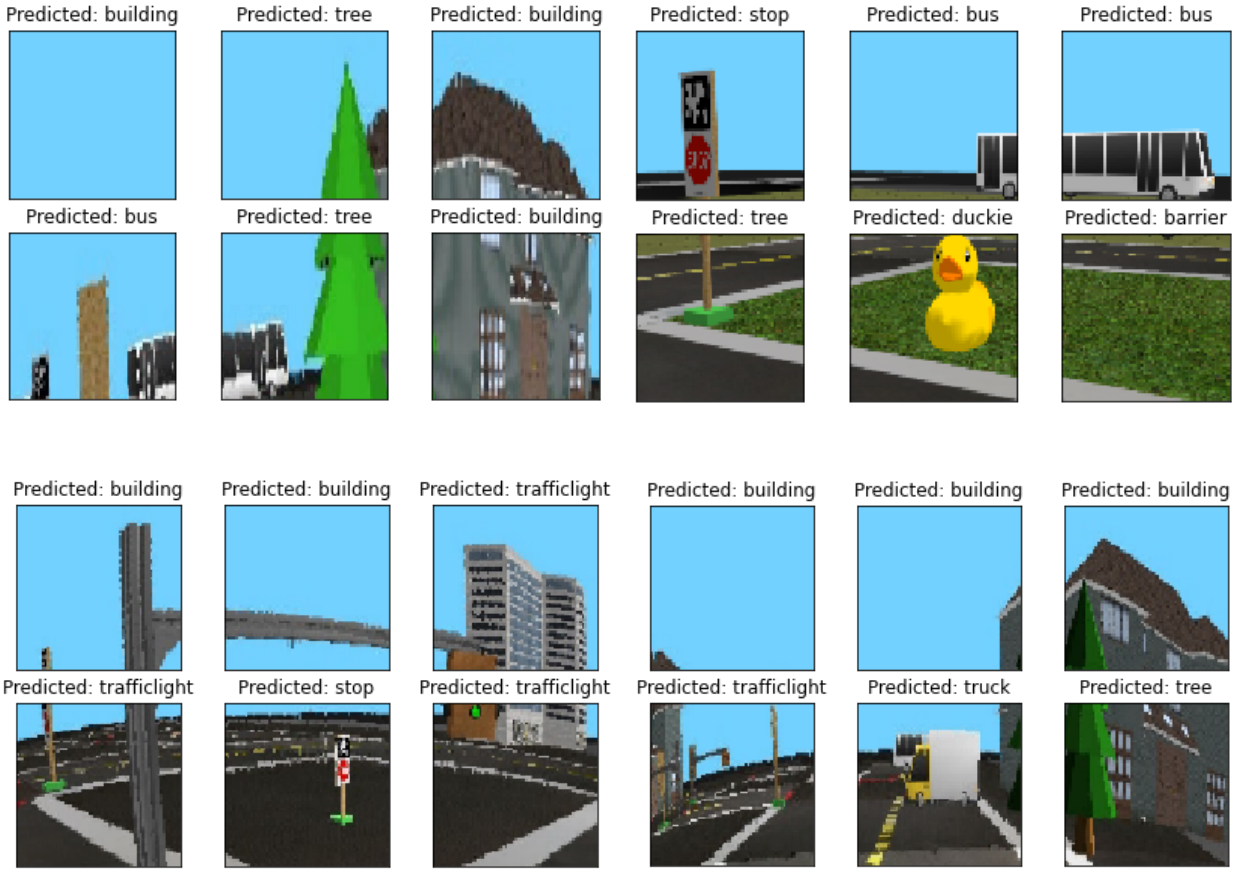
quindi l'immagine venga scambiata per una barriera o un edificio, entrambi grigi come lo sfondo.

Per quanto riguarda le immagini delle case, l'errore potrebbe dipendere dalla presenza delle linee rosse sull'asfalto, che vengono considerate come features rilevanti e portano l'algoritmo a classificare l'oggetto come segnale di stop. Le due barriere vengono classificate invece come edifici, probabilmente per la presenza del rumore che porta ad avere delle macchie più scure sulle barriere, che vengono scambiate per le finestre di un edificio.

Per tentare di ridurre ulteriormente il numero di errori commessi dall'algoritmo in fase di testing, per le suddette classi sono state usate un numero di immagini di training maggiore, pari a 8. Tuttavia, i risultati riportati non sono migliori di quelli avuti in precedenza.



Dal momento che in Duckietown non si dispone di immagini contenenti un singolo oggetto, ma si ha una visione complessiva della strada davanti a sé, ho provato a prendere delle immagini così com'erano dal simulatore, a dividerle in sei parti e a classificare ciascuna di esse con l'algoritmo SA e questi sono i risultati ottenuti:



Nei cinque esempi riportati, l'algoritmo riesce a classificare correttamente quasi tutte le parti delle varie immagini, facendo pochi errori. In particolare, l'aspetto positivo è che se in una parte d'immagine sono presenti più oggetti di interesse, come ad esempio nel caso delle sezioni centrali o quella in basso a destra dell'immagine qui accanto, l'algoritmo predice l'oggetto che si trova più vicino tra quelli presenti, ovvero stop e trafficlight. Il tempo impiegato per la classificazione di tutte e sei le parti delle singole immagini, utilizzando solamente la CPU, è di 12.55 secondi. Supponendo di voler classificare solo gli oggetti che si trovano davanti a noi, cioè nel riquadro in basso al centro, per evitare di andarvi a sbattere, o nel riquadro in basso a destra, nel caso di un cartello o un semaforo, potremmo ridurre ulteriormente i tempi in modo tale da rendere questo algoritmo utilizzabile nell'ambito di Duckietown.

CONCLUSIONS

Sono stati messi a confronto due diversi algoritmi di few-shot learning: uno che fa uso della semplice categorical cross-entropy loss e uno che sfrutta la triplet loss e il metric learning. Nel secondo caso, utilizzando le metriche implementate nella repository indicata nella sezione experimets, si è giunti a risultati apparentemente buoni, ma discordanti rispetto a quelli ottenuti valutando l'accuracy tradizionale. Il primo algoritmo, invece, anche se utilizzato su immagini contenenti più oggetti di interesse, riesce a classificarne correttamente le varie parti, in tempi non troppo elevati, perciò è risultato essere più adatto per Duckietown.

REFERENCES

[1] Mohit Sewak, Md. Rezaul Karim, Pradeep Pujari. Practical Convolutional Neural Networks: Implement advanced deep learning models using Python. 2018.

[2] Florian Schroff, Dmitry Kalenichenko, James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering.

[3] Xiaomeng Li, Lequan Yu, Chi-Wing Fu, Meng Fang, and Pheng-Ann Heng. Revisiting Metric Learning for Few-Shot Image Classification.

[4] Alexander Hermans, Lucas Beyer, Bastian Leibe. In Defense of the Triplet Loss for Person Re-Identification

[5] Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'REILLY. 2019.