Aerial Semantic Segmentation
Visione Artificiale 2021/2022

Achilli Mattia
Rettaroli Andrea

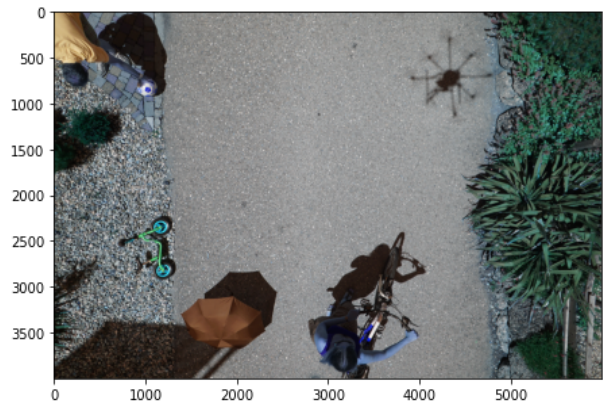
Introduzione al problema

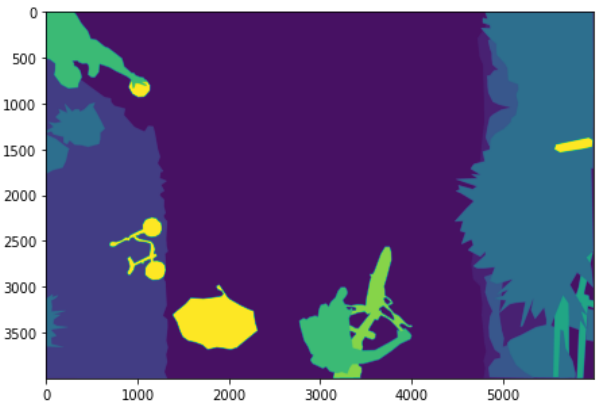
Si tratta di un problema di semantic segmentation su immagini aeree scattate da alcuni droni con lo scopo di migliorare la guida autonoma in volo e le procedure di atterraggio.

Data understanding

Le immagini sono acquisite da droni ad una altitudine dai 5 ai 30 metri dal suolo con una fotocamera ad alta risoluzione **6000x4000px** (**24Mpx**).

Le immagini originali sono RGB mentre le maschere del ground truth presentano come valore di ogni pixel la classe di appartenenza (da 0 a 22).





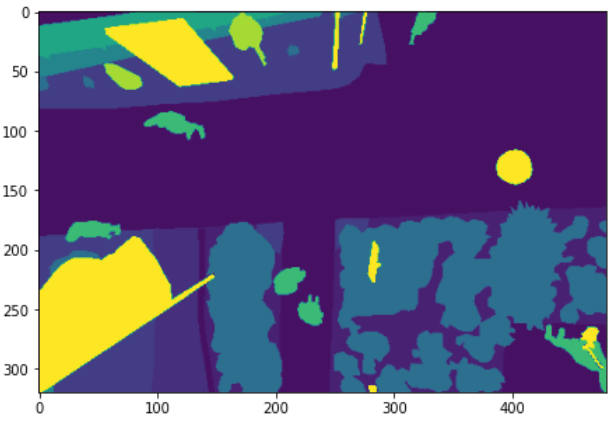
Le immagini pubbliche che vengono utilizzate per l'addestramento contengono 400 immagini e le corrispondenti 400 maschere di ground truth. Sono inoltre presenti 200 immagini private utilizzate come test set.

Le classi del problema sono 23 di vario tipo:

	name	r	g	b
0	unlabeled	0	0	0
1	paved-area	128	64	128
2	dirt	130	76	0
3	grass	0	102	0
4	gravel	112	103	87
5	water	28	42	168
6	rocks	48	41	30
7	pool	0	50	89
8	vegetation	107	142	35
9	roof	70	70	70
10	wall	102	102	156
11	window	254	228	12
12	door	254	148	12
13	fence	190	153	153
14	fence-pole	153	153	153
15	person	255	22	96
16	dog	102	51	0
17	car	9	143	150
18	bicycle	119	11	32
19	tree	51	51	0
20	bald-tree	190	250	190
21	ar-marker	112	150	146
22	obstacle	2	135	115

Caricamento e preprocessing delle immagini

Ogni immagine viene letta e ridimensionata ad una dimensione specificata mantenendo l'aspect ratio delle dimensioni originali, dopo diverse prove abbiamo scelto **480x320**, inoltre i pixel delle immagini vengono normalizzati da un range di 0-255 a 0-1 per migliorare le prestazioni. Per quanto riguarda le maschere, vengono lette in scala di grigi, ridimensionate come le immagini facendo attenzione però a conservare al massimo le informazioni delle classi per ogni pixel, per questo viene fatta una interpolazione.

Dopo il caricamento, la shape delle immagini corrisponde a (400, 320, 480, 3) mentre quella delle maschere a (400, 320, 480).

Come ultima operazione, alle maschere vengono aggiunti dei vettori per ogni pixel di dimensione pari al numero di classi (23), ogni vettore contiene 1 in corrispondenza dell'indice del valore della classe e 0 in tutto il resto. Ad esempio la classe 10 di un pixel sarà rappresentata come:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0.], dtype=float32)
```

Questo va fatto per permettere alla rete in seguito di prevedere un probabilità di appartenenza ad ogni classe di ogni pixel.

Distribuzione delle classi

Il problema in sè è sbilanciato, infatti non tutte le classi hanno lo stesso numero di pixel:

```
Percentage class 0 over total of pixel: 0.00394%
Percentage class 1 over total of pixel: 0.37672%
Percentage class 2 over total of pixel: 0.03194%
Percentage class 3 over total of pixel: 0.19951%
Percentage class 4 over total of pixel: 0.07292%
Percentage class 5 over total of pixel: 0.02210%
Percentage class 6 over total of pixel: 0.00718%
Percentage class 7 over total of pixel: 0.00638%
Percentage class 8 over total of pixel: 0.07091%
Percentage class 9 over total of pixel: 0.07350%
Percentage class 10 over total of pixel: 0.02683%
Percentage class 11 over total of pixel: 0.00559%
Percentage class 12 over total of pixel: 0.00031%
Percentage class 13 over total of pixel: 0.00955%
Percentage class 14 over total of pixel: 0.00053%
Percentage class 15 over total of pixel: 0.01051%
Percentage class 16 over total of pixel: 0.00014%
Percentage class 17 over total of pixel: 0.00785%
Percentage class 18 over total of pixel: 0.00216%
Percentage class 19 over total of pixel: 0.02049%
Percentage class 20 over total of pixel: 0.01329%
Percentage class 21 over total of pixel: 0.00228%
Percentage class 22 over total of pixel: 0.03538%
```

Per questo motivo si è deciso di calcolare i pesi per ciascuna classe del problema assegnando un peso maggiore alle classi con pixel meno presenti.

Suddivisione dei dati

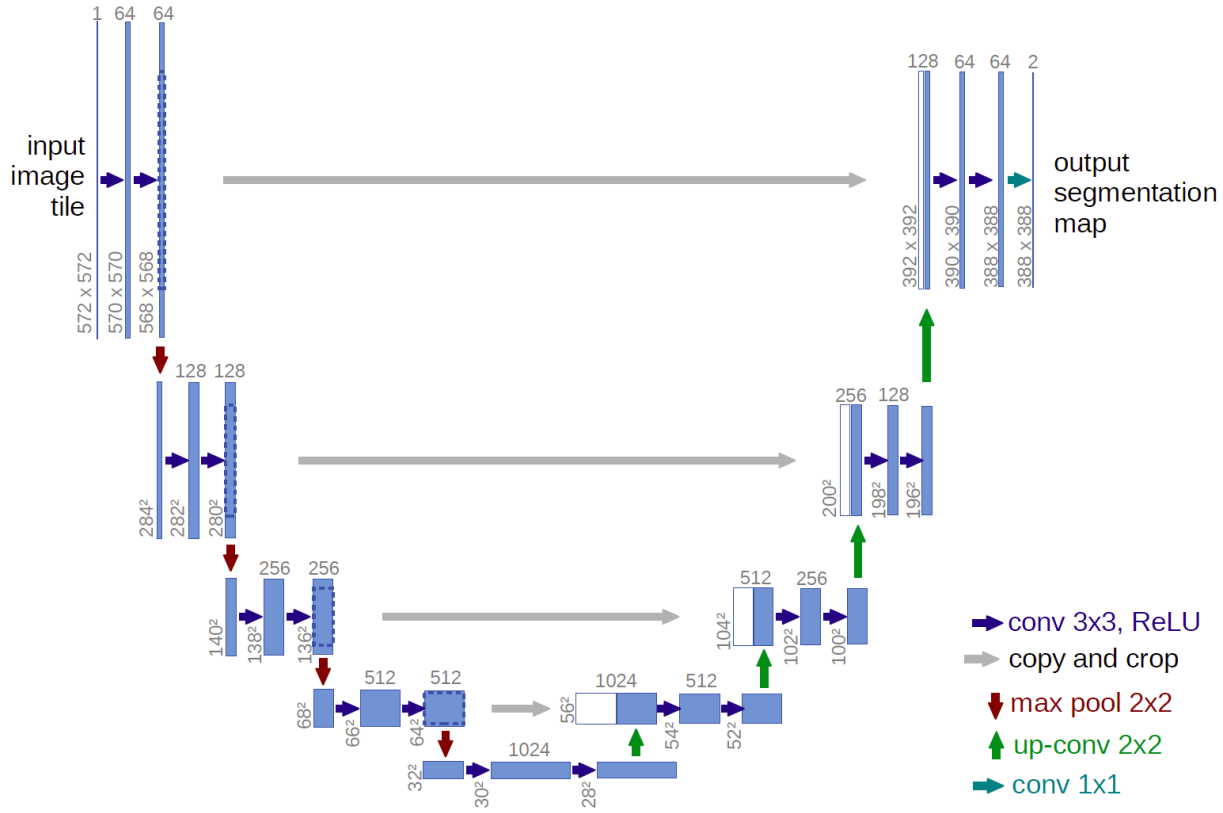
Le immagini e corrispondenti maschere vengono suddivise nei classici training, validation e test set, vengono utilizzate 200 immagini per il training set, 100 immagini per il validation set e 100 immagini per il test set. Ovviamente la suddivisione dei tre set tiene conto della distribuzione di frequenza delle classi suddividendo i pixel per classe in maniera equa tra i tre set.

```
Training set X shape: (200, 320, 480, 3)
Validation set X shape: (100, 320, 480, 3)
Test set X shape: (100, 320, 480, 3)

Training set Y shape: (200, 320, 480, 23)
Validation set Y shape: (100, 320, 480, 23)
Test set Y shape: (100, 320, 480, 23)
```

Definizione del modello

La rete utilizzata per il problema è la l'**U-Net** che rappresenta una **fully-convolutiona neural network** originariamente proposta per segmentazione binarie su immagini mediche:



La rete è composta da tre parti:

- **Downsampling:** composta da 4 blocchi con lo scopo di diminuire le dimensioni spaziali mentre si aumenta la profondità. Ogni blocco è composto da due layer convoluzionali e un layer di max-pooling.
- **Bottleneck:** composta da due layer convoluzionali.
- **Upsampling:** composta da 4 blocchi con lo scopo di far ritornare le immagini restituite dalla parte di bottleneck alle dimensioni spaziali originali dell'immagine in input. Ogni blocco consiste in un livello di upsampling per raddoppiare le dimensioni spaziali, un

layer di convoluzione una skip connection con il corrispondente layer del blocco di downsampling e due layer di convoluzione finali.

Tutti i layer di convoluzione utilizzano **ReLU** come funzione di attivazione.

Il layer di convoluzione finale presenta originariamente una funzione di attivazione **sigmoide** ma nel nostro caso viene utilizzata la **softmax** con profondità pari al numero della classi. Per ogni pixel verrà ritornato in un vettore la probabilità di appartenenza a ciascuna delle 23 classi.

Iperparametri

Prima di passare alla fase di addestramento del modello bisogna definire i parametri per la creazione del modello e per la sua "compilazione":

- **Numero di filtri:** numero di filtri che la rete deve apprendere, abbiamo spesso utilizzato i filtri [32, 64..512] ma anche [64, 128..1024] ottenendo risultati peggiori.
- **Ottimizzatore:** abbiamo utilizzato soprattutto **Adam** con un learning rate di 0.0001 (per attenuare le oscillazioni durante l'addestramento) con cui abbiamo raggiunto i risultati migliori, abbiamo provato anche **RMSprop** e **SGD** con cui abbiamo raggiunto risultati leggermente peggiori.
- **Funzione di loss:** la funzione di loss utilizzata è la **categorical_crossentropy** che viene utilizzata per problemi multi-classe e quantifica la differenza tra due distribuzioni di probabilità.
- **Metriche:** le metriche utilizzate sono l'accuratezza e l'indice di **Jaccard**. In questo caso l'accuratezza rappresenta la percentuale di pixel classificati correttamente, avendo ogni pixel una classe il classificatore deve indicare la classe di appartenenza di ciascun pixel. L'indice di Jaccard (o IoU) viene utilizzato per avere una misura più accurata di come si comporta il modello, infatti permette di misurare la sovrapposizione tra le maschere predette e quelle reali.

Mentre per quanto riguarda l'addestramento abbiamo definito:

- **Numero di epoche:** abbiamo sempre utilizzato un numero di epoche elevato data la complessità del problema, per la maggior parte degli addestramenti sono state utilizzate 500/1000 epoche.
- **Batch size:** dato che le immagini del training set non sono tante abbiamo deciso di utilizzare un numero più piccolo di batch, negli addestramenti abbiamo utilizzato un batch size tra i 5 e 16 per poter permettere alla rete di fare più iterazioni durante un'epoca, abbiamo notato come abbassando consapevolmente il batch size le performance tendono ad aumentare ma allo stesso tempo aumenta anche la complessità computazionale e quindi il tempo.

Infine per migliorare l'addestramento abbiamo utilizzato delle **callbacks** di Keras:

- **ModelCheckpoint:** callback che viene utilizzata per salvare il modello ogni qualvolta si ottenga una metrica migliore fino a quel momento sul validation set. Nel nostro caso la metrica da tenere in considerazione è l'indice di Jaccard sul validation set. In questo modo se le prestazioni del modello sul validation set dovessero degradare durante l'addestramento non si perdono i migliori parametri appresi dal modello.
- **EarlyStopping:** callback che permette di fermare l'addestramento prima del numero di epoche definito qualora non ci sia un miglioramento delle prestazioni ogni **N** epoche definite. Per miglioramento delle prestazioni si intende monitorare una metrica durante l'addestramento come l'accuratezza o l'indice di Jaccard. Nel nostro caso ogni 20 epoche si monitora l'andamento dell'indice di Jaccard sul validation set.

Fasi di addestramento del modello

Dopo aver definito gli iperparametri si arriva alla fase di addestramento del modello, prima di arrivare al modello definitivo sono state effettuate innumerevoli addestramenti utilizzando diverse tecniche per migliorare le performance del modello:

1. Inizialmente il modello è stato addestrato diverse volte variando gli iperparametri base come il numero di filtri, l'ottimizzatore e il batch size. I risultati ottenuti in questo modo erano buoni ma molto migliorabili, infatti l'indice di Jaccard massimo raggiunto sul validation set in questa prima fase risultava ~0.50/0.55.
2. In seguito dato che il problema come detto precedentemente risulta sbilanciato sono stati utilizzati i pesi relativi alle classi da tenere conto durante l'addestramento. Per fare ciò è stata predisposta una matrice da utilizzare nell'addestramento con numero di righe uguale al numero di esempi del training set, dove ogni riga contiene i pixel dell'immagine (320 x 480 valori) e in ogni cella relativa all'immagine **i** del pixel **j** viene inserito il peso relativo alla classe del pixel **j.** In questo modo i risultati sono migliorati raggiungendo dei risultati più accurati sul validation set, con un indice di Jaccard del ~0.60. Oltre all'utilizzo della matrice dei pesi, è stato aggiunto un layer (**Reshape** di Keras) alla rete per ridimensionare l'output della rete per poter utilizzare i pesi, il layer ridimensiona l'output "distendendo" le immagini in 320 x 480 valori in cui ad ogni pixel è associato un vettore one-hot encoded. Inoltre, si è notato che addestrando la rete senza pesi e moltiplicando i pesi per le probabilità date dalla rete per ogni classe di un pixel il risultato rimane simile, facendo ciò si ottiene anche un addestramento meno pesante.
3. Come ultima prova è stato deciso di utilizzare tecniche di data augmentation (attraverso la libreria **albumentations**) per aumentare il numero e la variabilità delle immagini nel training e validation set, lasciando il test set invariato. Le operazioni di data augmentation effettuate sono il flip delle immagini in orizzontale e in verticale. Le immagini di training sono state portate da 200 a 800 mentre quelle di validation da 100 a 400. Il flip in verticale è stato effettuato sia sulle immagini del flip orizzontale che sulle immagini originali. Però, dato che il numero di immagini aumenta considerevolmente, anche lo spazio in memoria aumenta e dato

che Keras necessità di allocare più memoria di quella di cui ha disposizione molto spesso il kernel Jupyter va in out-of-memory, per questo motivo viene utilizzato un **ImageDataGenerator** (di Keras) vuoto a cui vengono passati i vettori di immagini e maschere con data augmentation. In questo modo utilizzando questa classe si utilizza molta meno memoria dato che le immagini vengono caricate in batch e non tutte subito come viene fatto solitamente. Utilizzando la data augmentation senza la matrice dei pesi ma moltiplicando i pesi in un secondo momento l'indice di Jaccard ottenuto sul validation set risulta del ~0.65.

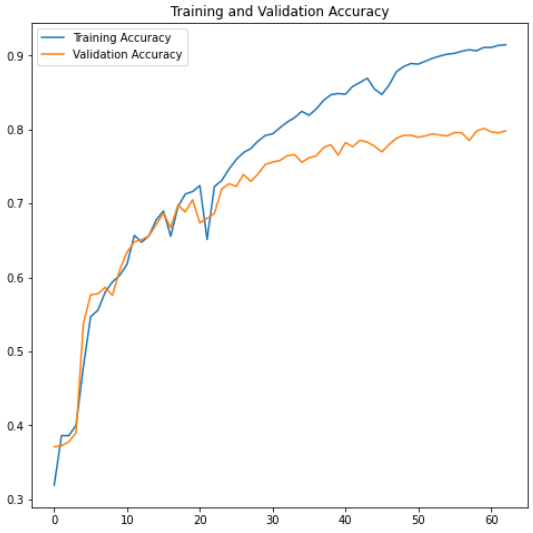
Il modello migliore

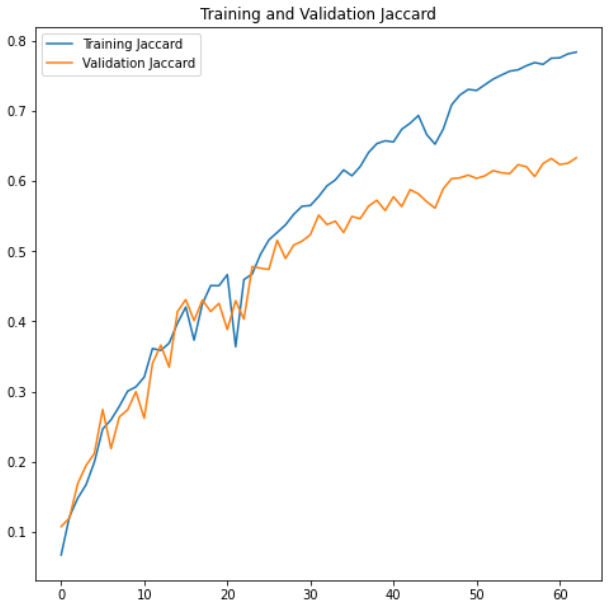
Il modello migliore ottenuto risulta quello con data augmentation, infatti il modello raggiunge un indice di Jaccard di ~0.65 sul validation set con un'accuratezza del 80% circa.

```
model.evaluate(validation_x, validation_y)
```

```
4/4 [==============================] - 2s 240ms/step - loss: 0.9100 - acc: 0.7992 - jaccard_index: 0.6494
```

Di seguito vengono riportati anche i grafici legati alla history dell'addestramento relativi ad accuratezza, loss e indice di Jaccard:





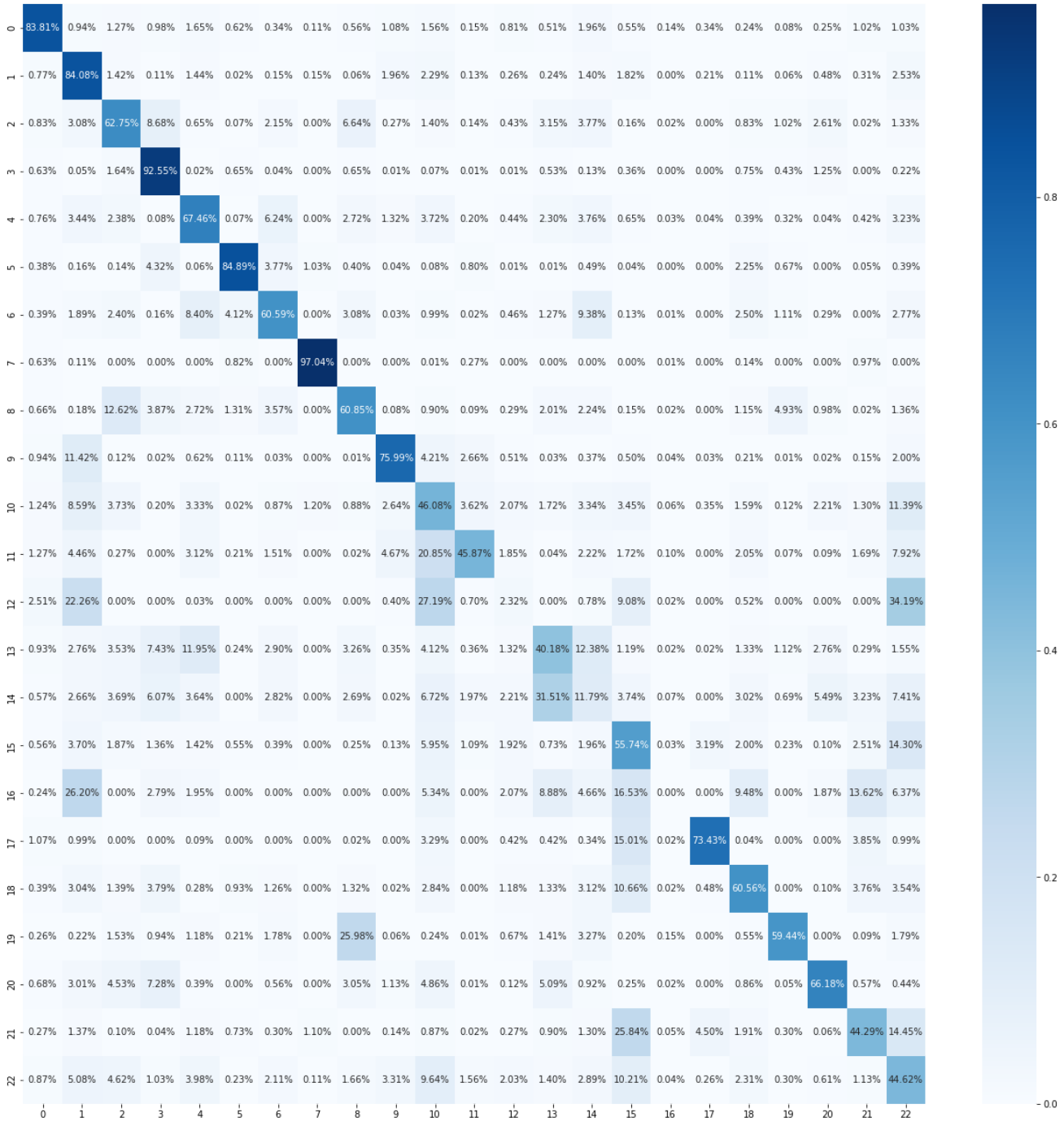
Il modello si comporta bene anche sul Test set, infatti i risultati raggiunti sono leggermente migliori di quelli raggiunti sul validation set.

```
model.evaluate(test_x, test_y)
```

```
4/4 [==============================] - 1s 244ms/step - loss: 0.8058 - acc: 0.8144 - jaccard_index: 0.6636
```

Matrice di confusione, metriche di valutazione sul Test set

Dato che si tratta di un problema di classificazione non basta utilizzare come metrica l'accuratezza ma è necessario utilizzare altre metriche come la matrice di confusione che mostra sulla diagonale la percentuale di istanze predette di una classe. Ogni colonna della matrice rappresenta i valori predetti mentre le righe i valori reali.



Come si può notare dalla matrice di confusione il modello si comporta bene in generale, purtroppo però per la classe 16 il modello non riconosce alcun pixel di quella classe dato il bassissimo numero di pixel di quella classe.

Inoltre si può visualizzare anche precision, recall e f1-score notando come il modello si comporti meglio nelle classi con presenza di più pixel:

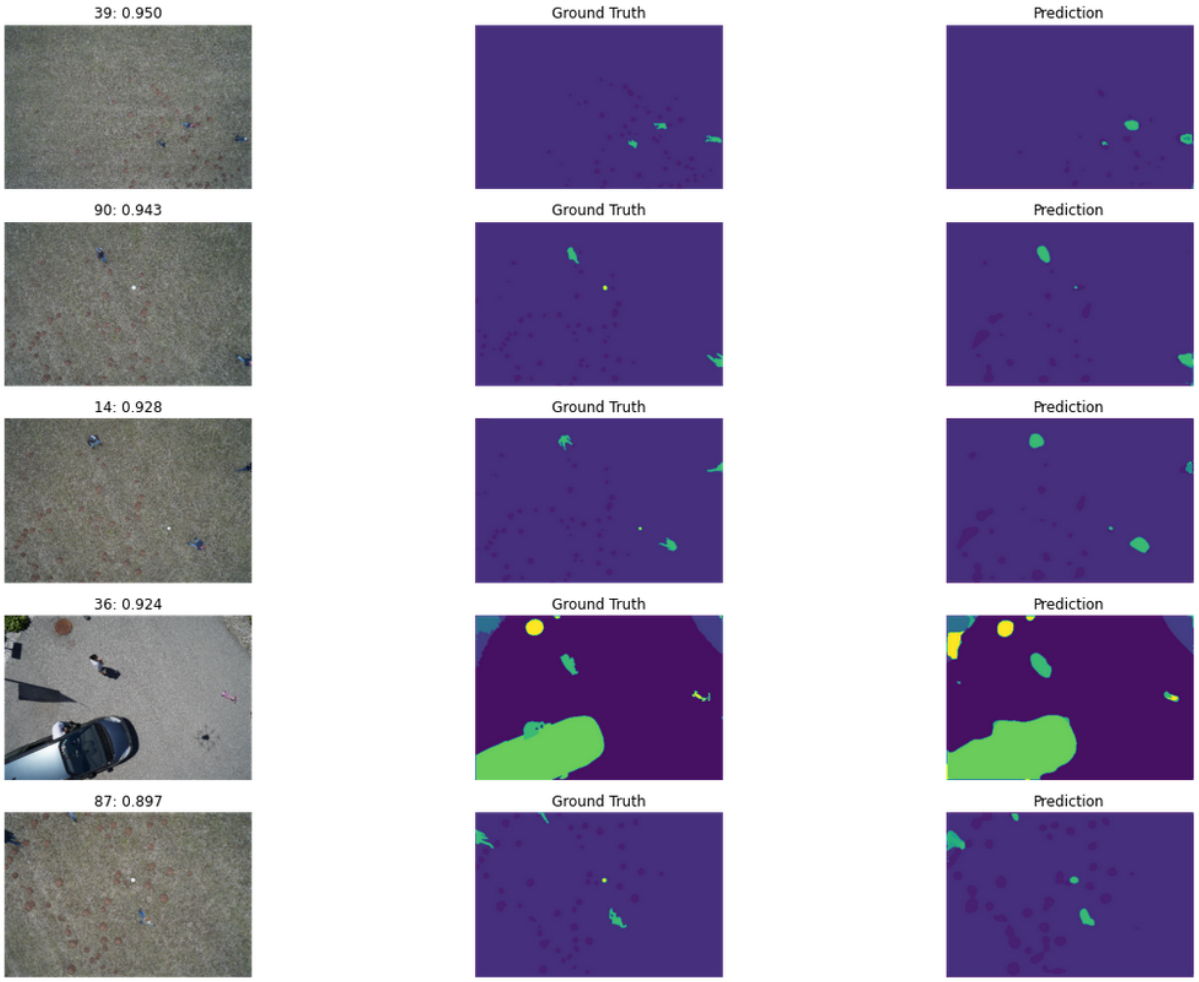
	precision	recall	f1-score	support
0	0.291784	0.778374	0.424455	6.262800e+04
1	0.944297	0.846046	0.892475	5.720312e+06
2	0.427534	0.632316	0.510141	4.800720e+05
3	0.947281	0.914054	0.930371	3.257105e+06
4	0.800690	0.654638	0.720336	1.132868e+06
5	0.833536	0.574096	0.679908	3.041650e+05
6	0.197377	0.415099	0.267540	9.287900e+04
7	0.794249	0.931923	0.857596	9.611200e+04
8	0.764666	0.622833	0.686500	1.138386e+06
9	0.806401	0.692123	0.744904	1.035139e+06
10	0.329553	0.503728	0.398437	4.247830e+05
11	0.220482	0.449726	0.295898	7.469000e+04
12	0.005142	0.047020	0.009270	6.125000e+03
13	0.194719	0.223929	0.208305	1.271920e+05
14	0.002097	0.055178	0.004040	8.971000e+03
15	0.301355	0.598004	0.400755	1.535190e+05
16	0.000000	0.000000	0.000000	2.391000e+03
17	0.798963	0.818723	0.808722	1.004760e+05
18	0.181653	0.641178	0.283100	4.114300e+04
19	0.665451	0.598442	0.630170	2.738580e+05
20	0.554586	0.733898	0.631765	2.119040e+05
21	0.205436	0.567393	0.301652	2.456500e+04
22	0.430639	0.412116	0.421174	5.907170e+05
accuracy	0.761264	0.761264	0.761264	7.612639e-01
macro avg	0.465126	0.552645	0.482935	1.536000e+07
weighted avg	0.816769	0.761264	0.782862	1.536000e+07

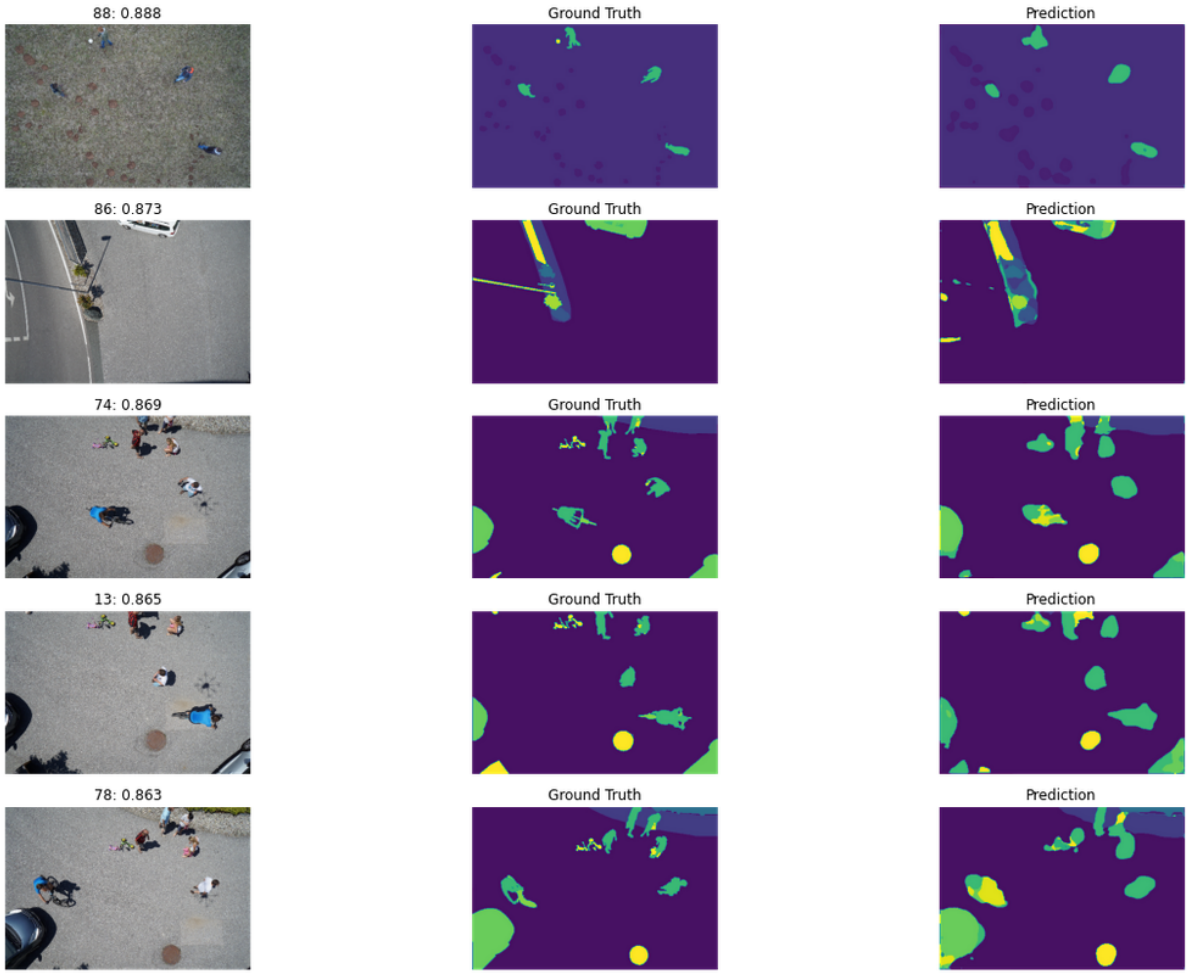
Risultati migliori e peggiori sul Test set

Come ultima cosa abbiamo visualizzato le 15 maschere predette migliori e le 15 peggiori. Per fare ciò viene calcolato l'indice di Jaccard su tutte le immagini del test set, si ordinano e si ottengono le 15 migliori e peggiori.

(Per problemi di spazio ne mostriamo solo alcune)

Le immagini migliori sono le seguenti:

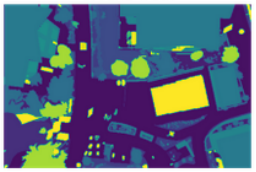




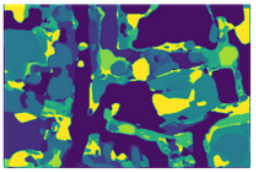
Mentre le peggiori:

42: 0.271

Ground Truth



Prediction



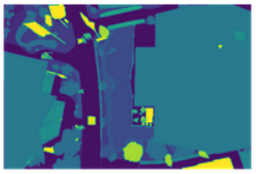
44: 0.282

Ground Truth

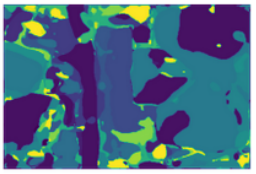
Prediction

6: 0.296

Ground Truth

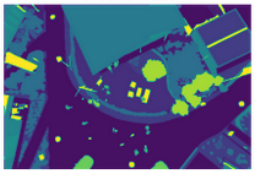


Prediction

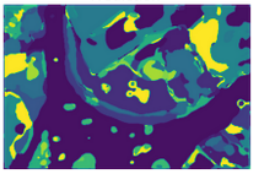


69: 0.402

Ground Truth

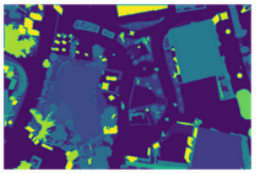


Prediction

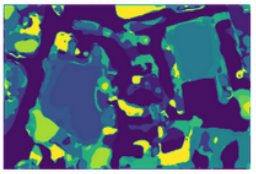


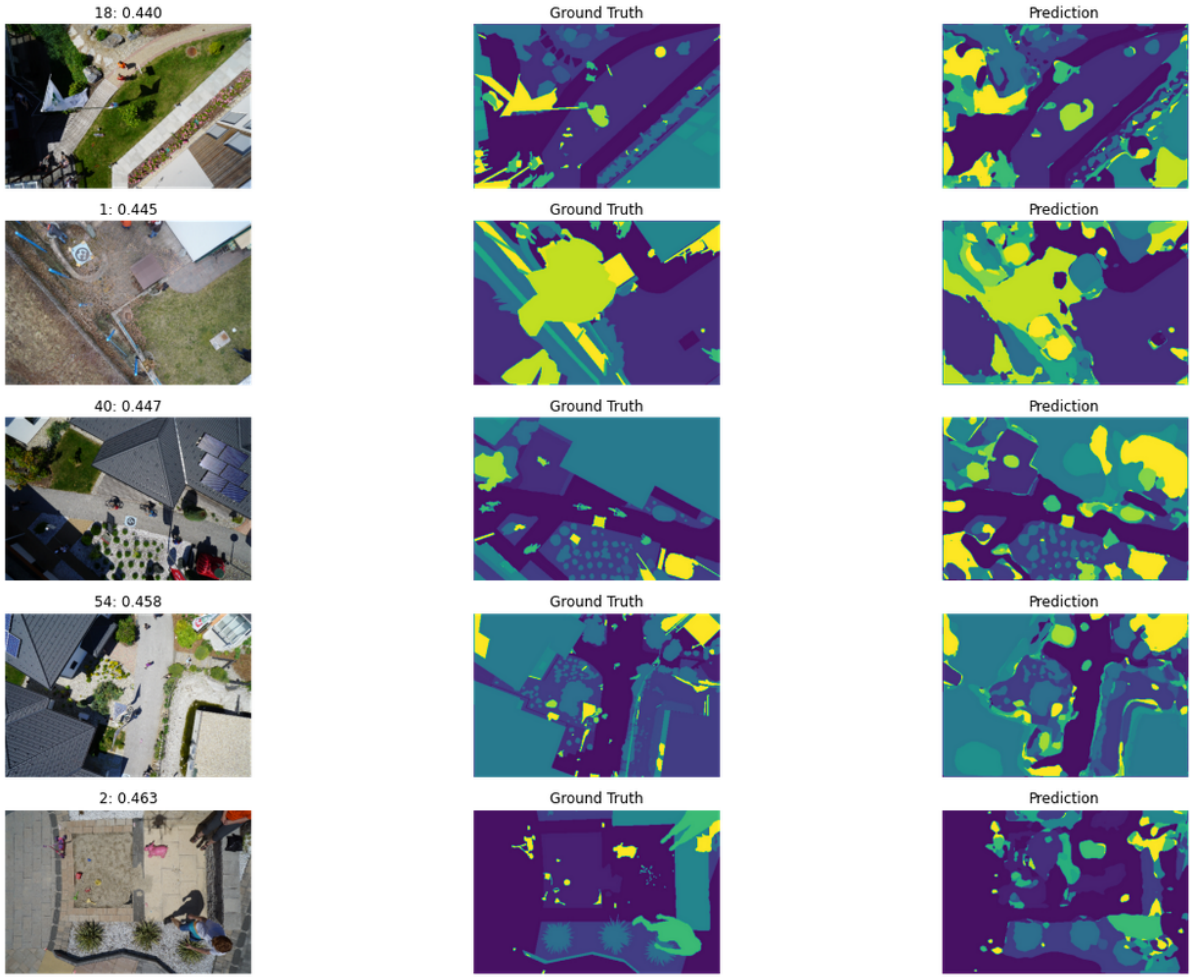
23: 0.414

Ground Truth



Prediction





Dalle immagini sopra riportate si può notare come la rete riesca a segmentare meglio immagini meno complesse dove compaiono meno oggetti o oggetti più distanti da altri a differenza di immagini in cui sono presenti tanti oggetti o ravvicinati tra loro.

Commenti finali sui risultati

In conclusione, il problema è stato risolto raggiungendo dei buoni risultati seppur il dataset in sé presenta delle difficoltà derivanti dal numero delle classi (23) e dal limitato numero di immagini.

Sicuramente c'è del margine di miglioramento, in primis si potrebbero provare altre tecniche di data augmentation come rotazione, luminosità, zoom e quant altro stando attenti alle maschere, si potrebbe fare un preprocessing più approfondito e addirittura provare altre reti neurali allo stato dell'arte. Ci siamo accorti inoltre che in questo genere di problemi

si spende molto tempo tra addestramenti, tuning degli iperparametri e crash improvvisi della macchina a causa di out-of-memory derivato da un utilizzo pesante delle risorse, infatti utilizzare dell'hardware performante in questo ambito fa la differenza soprattutto nei tempi di addestramento dei modelli.

Eventuali sviluppi futuri

Una delle prime cose che faremo sicuramente sarà quella di sottomettere la nostra soluzione su Kaggle per confrontarci con altri aspettando anche pareri e consigli da altri utenti.

Inoltre abbiamo deciso di formare un team su Kaggle insieme ad altri utenti per cimentarci in problemi del genere anche su competizioni aperte essendoci problemi molto interessanti, stimolanti e di grande importanza per le nostre aspettative/carriere future.

Conclusioni

In conclusione lo sviluppo del progetto è stato per entrambi molto interessante e utile, sia dal punto di vista didattico che per il nostro futuro professionale.

Siamo entrambi molto interessati alla branca dell'intelligenza artificiale, seguiamo entrambi corsi relativi al percorso dati e data science e speriamo in futuro di ottenere grandi soddisfazioni e risultati in questo campo, crediamo che l'intelligenza artificiale sia una tecnologia di spicco sia dal punto di vista scientifico che sociale in un futuro prossimo.

Il nostro obiettivo con il tempo e l'esperienza sarebbe quello di diventare competenti abbastanza nell'ambito della data science per permetterci di aprire una propria impresa in Italia o all'estero con l'obiettivo di contribuire soprattutto a problemi sociali come la medicina o l'ambiente ad esempio.