

# Aerial Semantic Segmentation

## Visione Artificiale 2021/2022

Achilli Mattia  
Rettaroli Andrea

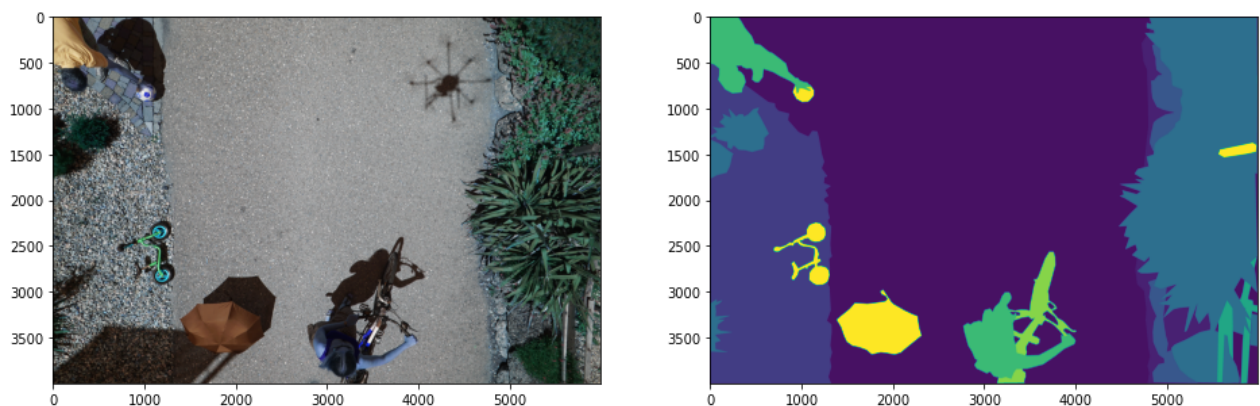
### Introduzione al problema

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 23).



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 24 e sono rappresentate da diverse combinazioni di colore:

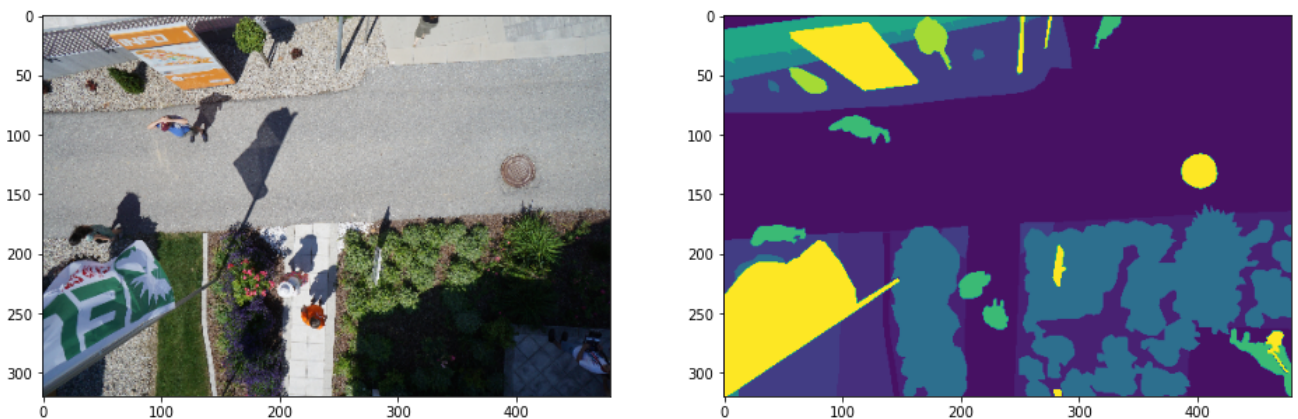
△ name	# r	# g	# b
<div>24</div> <div>unique values</div> 	0	0	0
unlabeled	0	0	0
paved-area	128	64	128
dirt	130	76	0
grass	0	102	0
gravel	112	103	87
water	28	42	168
rocks	48	41	30
pool	0	50	89
vegetation	107	142	35
roof	70	70	70
wall	102	102	156
window	254	228	12
door	254	148	12
fence	190	153	153
fence-pole	153	153	153
person	255	22	96
dog	102	51	0
car	9	143	150
bicycle	119	11	32
tree	51	51	0
bald-tree	190	250	190
ar-marker	112	150	146
obstacle	2	135	115
conflicting	255	0	0

La classe **conflicting** non è presente all'interno delle immagini del dataset perciò in fase di preprocessing si è deciso di rimuoverla.

## 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 (256x256, 240x160, 128x96) abbiamo scelto **480x320**, inoltre i pixel delle immagini vengono normalizzati da un range di 0-255 a 0-1 per migliorare le prestazioni durante l'addestramento.

Ogni maschera viene letta in scala di grigi e ridimensionata come le immagini facendo attenzione però a conservare al massimo le informazioni delle classi per ogni pixel, per questo viene fatta una interpolazione secondo l'algoritmo **Nearest Neighbor**.



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), per ogni pixel si costruisce un one-hot vector ovvero un vettore che 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.], dtype=float32)
```

In seguito questo ci tornerà utile per prevedere un probabilità di appartenenza ad ogni classe di ogni pixel.

## Distribuzione delle classi

Si nota che la distribuzione delle classi all'interno del dataset è sbilanciata; infatti non tutte le classi hanno lo stesso numero di pixel, come mostra la figura seguente.

```
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%
```

Il valore rappresentato va moltiplicato per 100 per ottenere la percentuale. La somma di tali percentuali corrisponde al 100%, ciò implica che tutti i pixel presenti all'interno del dataset vengono correttamente mappati ad una classe di appartenenza.

Per questo motivo si è deciso di calcolare i pesi per ciascuna classe del problema assegnando un peso maggiore alle classi con una minore quantità di pixel da considerare durante l'addestramento. Come mostrato nella figura seguente, alla classe 16 è assegnato un peso di 306 e alla classe 12 un peso 139 entrambi molto superiori rispetto al resto delle classi.

```
{0: 11.03074442982416,  
1: 0.11541224470493074,  
2: 1.3610632243109284,  
3: 0.21792695276533677,  
4: 0.5962814109357487,  
5: 1.967327631968331,  
6: 6.056129795951599,  
7: 6.8155255542273565,  
8: 0.6131810757349787,  
9: 0.5915654608150996,  
10: 1.6202400471072136,  
11: 7.783112621791651,  
12: 139.28277531811287,  
13: 4.551169433504592,  
14: 81.85647937200733,  
15: 4.138291406976645,  
16: 306.0965220380528,  
17: 5.538149684720278,  
18: 20.163831127914303,  
19: 2.1218358563115136,  
20: 3.2720252861943697,  
21: 19.105445953883855,  
22: 1.2288944282478667}
```

## Suddivisione dei dati

Inizialmente le 400 immagini a nostra disposizione venivano suddivise in un training set di 360 immagini e un validation set di 40 immagini. Il test set doveva essere composto dalle 200 immagini citate su Kaggle, che risultano però essere a scopo privato e a cui non vi è possibilità di accesso.

Successivamente abbiamo deciso di suddividere le immagini e le corrispondenti maschere in tre set: training, validation e test. Questo aspetto ha aggravato ulteriormente la complessità del problema in quanto ci ha costretto a ridurre notevolmente il numero di immagini utilizzate per il training set.

Si è deciso di utilizzare 200 immagini per il training set, 100 immagini per il validation set e 100 immagini per il test set. Inizialmente si pensava fosse necessario l'utilizzo di **stratify** per rendere omogenea la suddivisione delle classi di appartenenza dei pixel all'interno dei vari set; andando a visionare la percentuale di appartenenza delle classi dei pixel nei vari set risulta comunque omogenea per via del seed.

L'immagine seguente presenta la shape dei vari set.

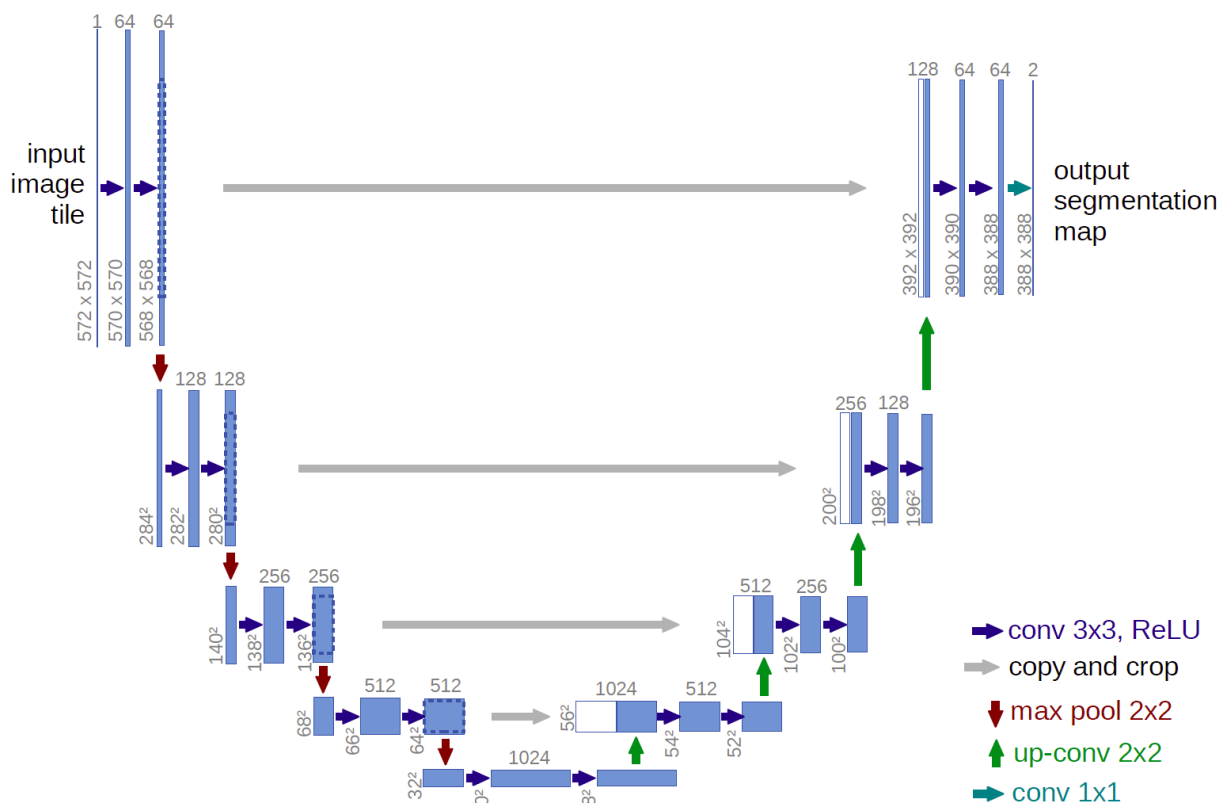
Training set X shape: (200, 320, 480, 3)  
Validation set X shape: (100, 320, 480, 3)  
Test set X shape: (100, 320, 480, 3)

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 l'**U-Net** che rappresenta una **fully-convolutional neural network** originariamente proposta per segmentazione binarie su immagini mediche.

L'immagine seguente mostra la struttura di un modello della U-Net. Ed è proprio per la sua forma che le viene attribuito questo nome.

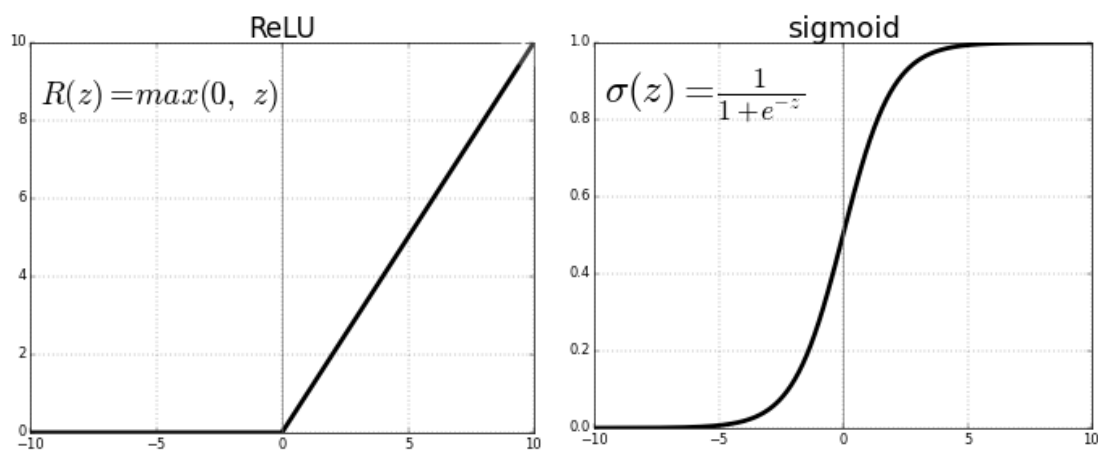


Analizzando la rete si nota che essa è composta da tre parti:

- **Downsampling:** composta da N blocchi, nel nostro caso 4, 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 che collegano la parte downsampling con quella di upsampling.
- **Upsampling:** composta da N blocchi, nel nostro caso 4, 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 che usa 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

Una volta giunti a questo punto, prima di passare alla fase di addestramento del modello, è necessario identificare e definire gli iperparametri per la creazione del modello.

Gli iperparametri necessari al modello sono:

- **Numero di filtri:** numero di filtri che la rete deve apprendere e utilizzare per l'estrazione delle features.
- **Ottimizzatore:** ottimizzatore da utilizzare durante la fase di training per l'apprendimento dei parametri e minimizzazione della funzione di loss, tra gli ottimizzatori più utilizzati: Adam, SGD e RMSprop.
- **Funzione di loss:** funzione da minimizzare durante l'addestramento.
- **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:** un'epoca rappresenta il passaggio di tutti gli esempi del training set che la rete deve apprendere.
- **Batch size:** il batch size indica il numero di esempi da considerare in ogni iterazione di un'epoca.



Una volta identificati gli iperparametri, si testano i seguenti valori:

- **Numero di filtri:** abbiamo spesso utilizzato i filtri [32, 64..512] ma anche [64, 128..1024] ottenendo risultati peggiori si pensa che un numero di filtri così alto richieda una maggiore quantità di dati.
- **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 confrontando i due vettori in output dalla softmax.
- **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 si è 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 di addestramento.

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

- **ModelCheckpoint:** callback che viene utilizzata per salvare il modello ogni qualvolta si ottenga una metrica migliore 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 dovessero degradare durante l'addestramento non si perdono i migliori parametri appresi dal modello.
- **EarlyStopping:** callback che permette di interrompere l'addestramento prima del numero di epoche definito qualora non ci sia un miglioramento delle prestazioni dopo 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. La fase di addestramento solitamente prevede di testare varie combinazioni di iperparametri al fine di trovare la combinazione che determina il modello migliore. Ripercorriamo il percorso che ci ha portati a trovare il modello finale:

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 il massimo indice di Jaccard raggiunto, sul validation set, è stato  $\sim 0.50/0.55$ .
2. Successivamente, tenendo in considerazione che il problema è sbilanciato sono stati introdotti i pesi relativi alle classi; durante l'addestramento vengono passati come parametri i pesi al modello tramite il parametro **sample\_weight**. 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 di ogni immagine contiene i pixel (320 x 480 valori) e in ogni cella dell'immagine **i**, del pixel **j**, viene inserito il peso relativo alla classe del pixel **j**. Abbiamo constatato che i risultati sono migliorati raggiungendo dei risultati più accurati sul validation set, con un indice di Jaccard di  $\sim 0.60$ . Oltre all'utilizzo della matrice dei pesi, è stato aggiunto un layer Keras per il reshape alla rete al fine di ridimensionare l'output della stessa 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 in output il risultato rimane costante. Applicando la seguente tecnica l'addestramento risulta meno costoso in termini di tempo e risorse.

3. Infine si è scelto di utilizzare tecniche di data augmentation, che inizialmente consideravamo difficile da applicare per via dello spazio necessario in memoria. Mediante la data augmentation si aumenta il numero e la variabilità delle immagini nel training e validation set, lasciando il test set invariato. Attraverso la libreria **albumentations** vengono effettuate le operazioni di flip delle immagini e delle maschere in orizzontale e in verticale. Il numero delle immagini di training è stato aumentato da 200 a 800 mentre quelle di validation da 100 a 400. Si nota che applicando il flip in verticale sia sulle immagini del flip orizzontale che sulle immagini originali si ottengono più immagini. Una volta riusciti a generare immagini e maschere correttamente e corrispondenti, il numero di immagini aumentava considerevolmente e così anche il loro spazio in memoria bloccando il kernel. Si sceglie quindi di utilizzare un **ImageDataGenerator** di Keras che non effettua elaborazioni a cui vengono passati i vettori di immagini e maschere precedentemente generati. Inizialmente si era provato ad effettuare la data augmentation tramite ImageDataGenerator, che prende in input (X, y, ...filtri da applicare) dove y è una label e X è un'immagine; nel nostro caso y è un'immagine rendendo inutilizzabile questo metodo in maniera banale. Sfruttando questa classe si utilizza molta meno memoria, ed è così possibile addestrare il modello, dato che le immagini vengono caricate già suddivise in batch e non tutte assieme. In questo caso la matrice dei pesi non viene passata in input al modello ma viene moltiplicata sulle predictions del modello come scoperto nel punto 2. L'indice di Jaccard ottenuto sul validation set risulta del ~0.65.

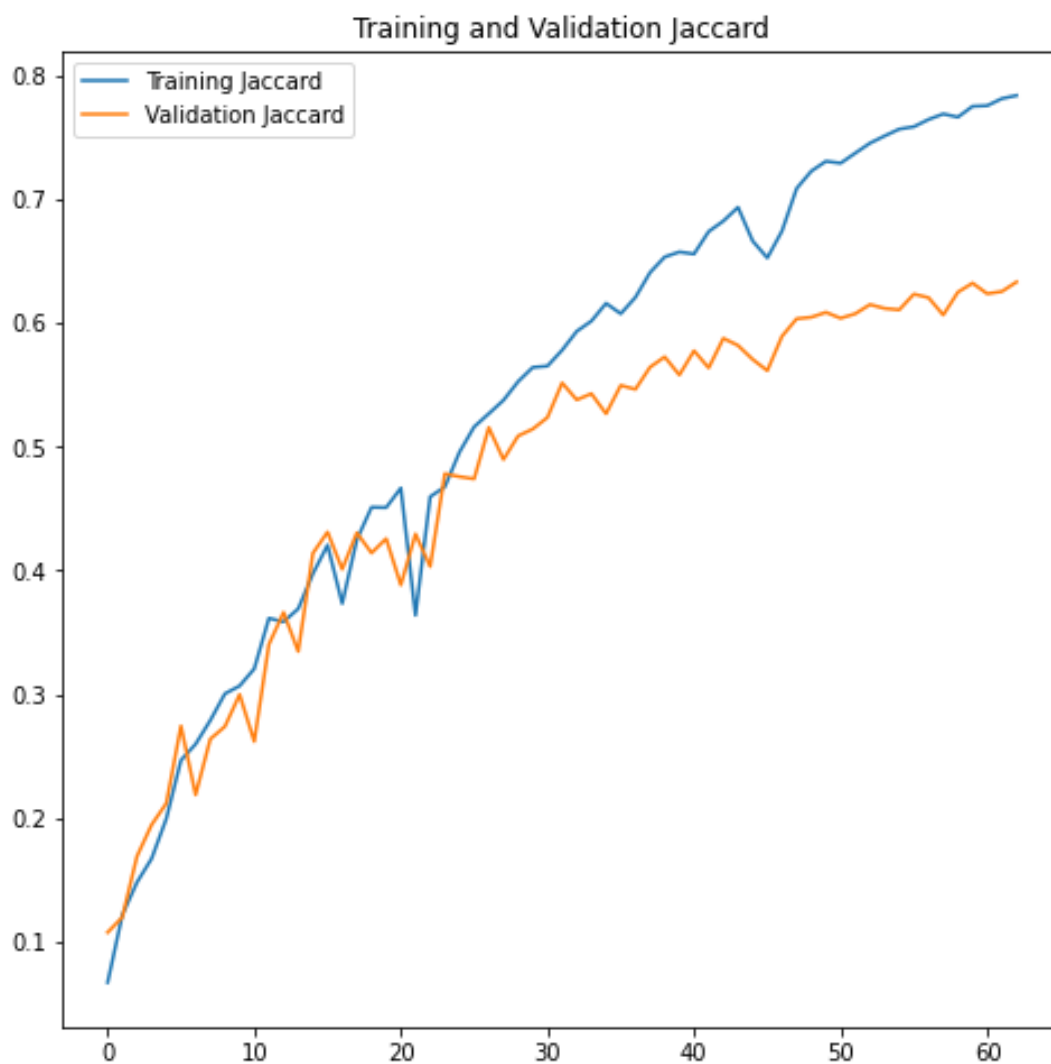
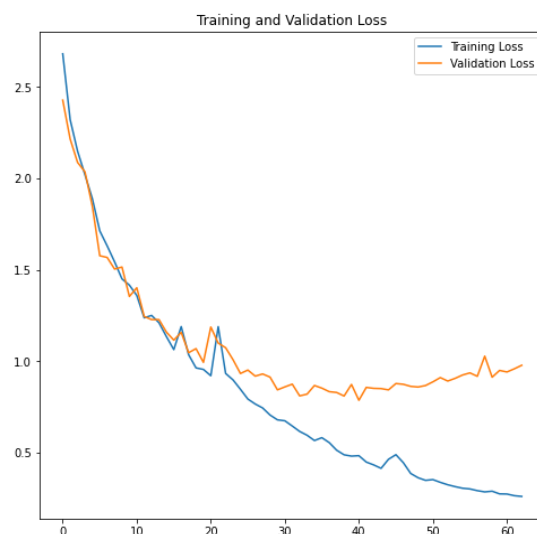
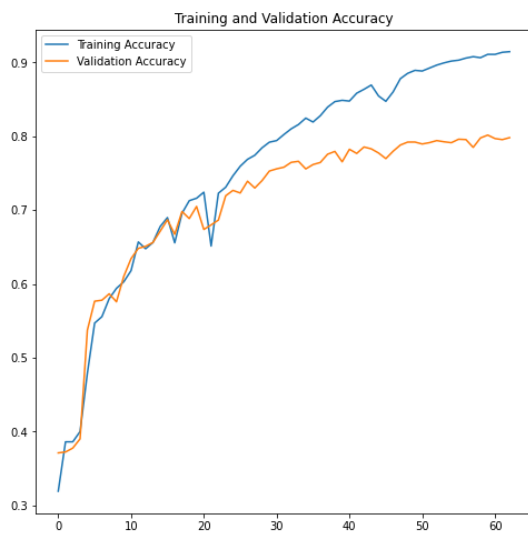
## Il modello migliore

Il modello migliore è quello ottenuto a seguito della applicazione della data augmentation, il modello raggiunge un indice di Jaccard del ~0.65 sul validation set con un'accuratezza del 80% circa come mostrato nella figura seguente.

```
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 i grafici legati alla history dell'addestramento relativi ad accuratezza, loss e indice di Jaccard.



Si nota che durante l'addestramento l'andamento delle metriche è lo stesso sia sul training set che sul validation set.

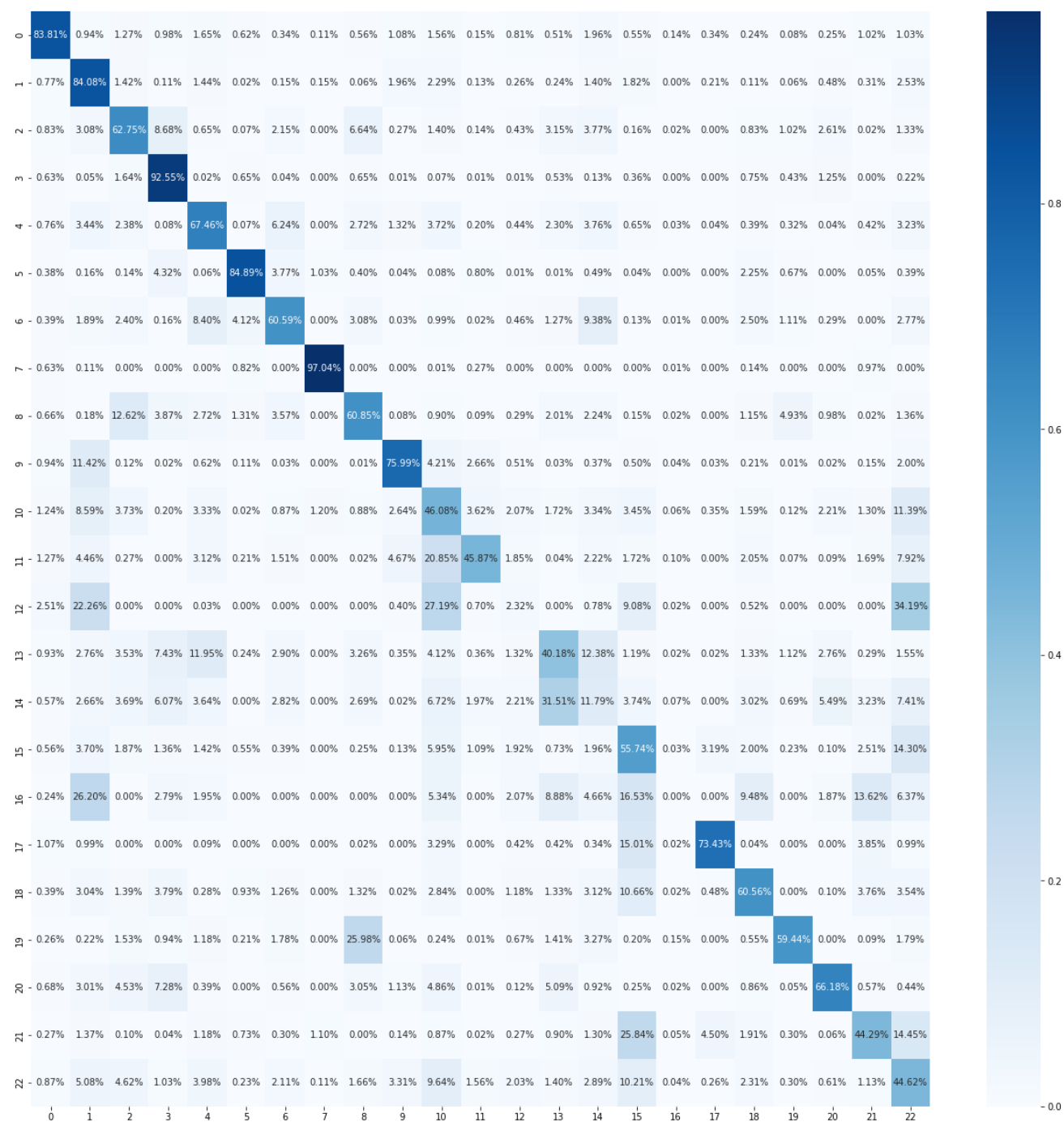
Il modello si comporta bene anche sul test set, come si può notare dall'immagine seguente i risultati raggiunti sono leggermente migliori di quelli sul validation set.

```
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**

Si ritiene che trattandosi di un problema di classificazione, non è sufficiente utilizzare come unica metrica l'accuratezza ma è necessario utilizzare altre metriche come la matrice di confusione; che mostra sulla diagonale la percentuale di istanze predette correttamente di ogni classe. Ogni colonna della matrice rappresenta i valori predetti mentre le righe i valori reali. Di seguito viene riportata la matrice di confusione sul test set.



Come si può notare dalla matrice di confusione, in generale, il modello si comporta bene. Si nota che per la classe 16 come per la 12 il modello non riconosce alcun o pochi pixel data la loro bassissima frequenza.

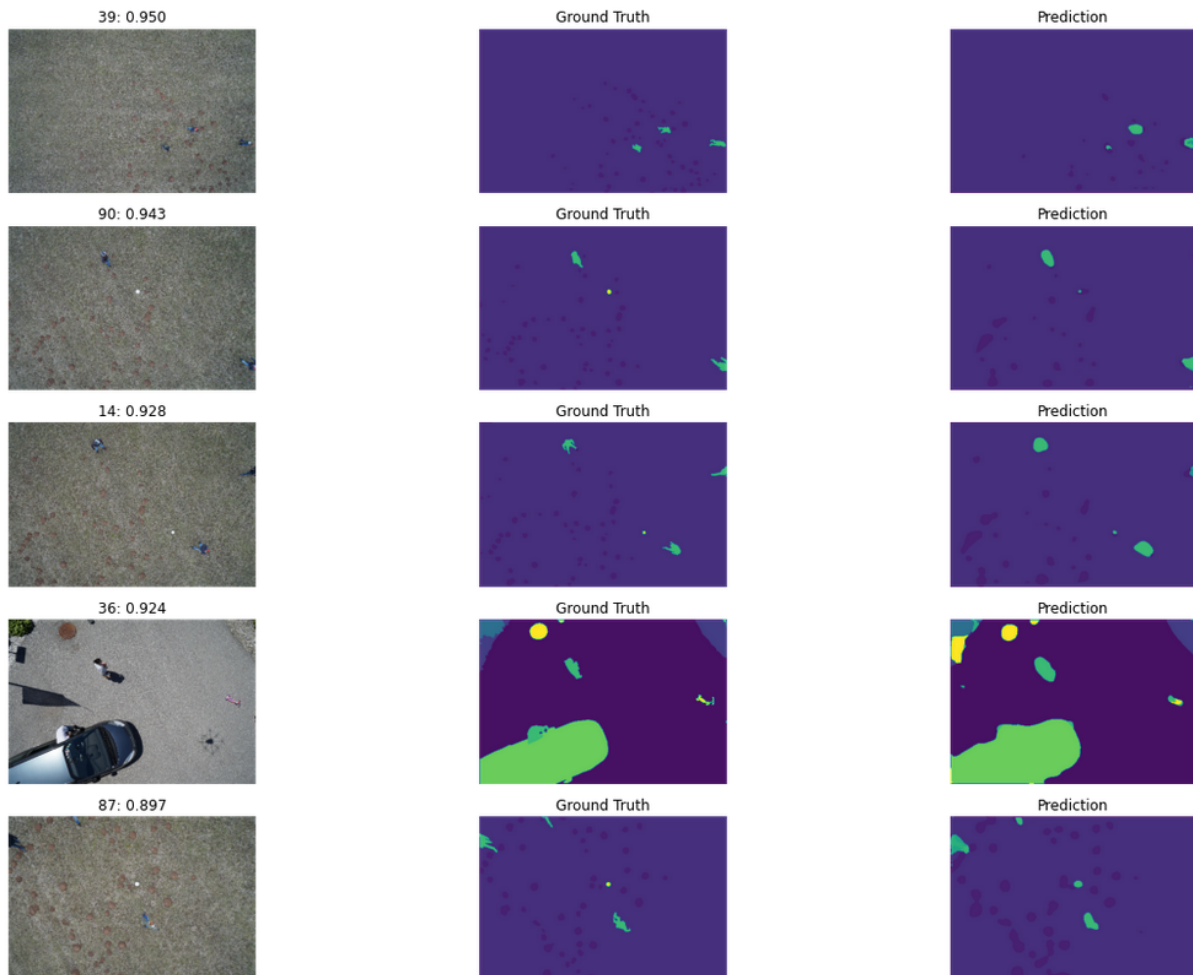
Inoltre si può visualizzare anche precision, recall e f1-score notando come il modello si comporti meglio nelle classi dove la frequenza di pixel è più alta:

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

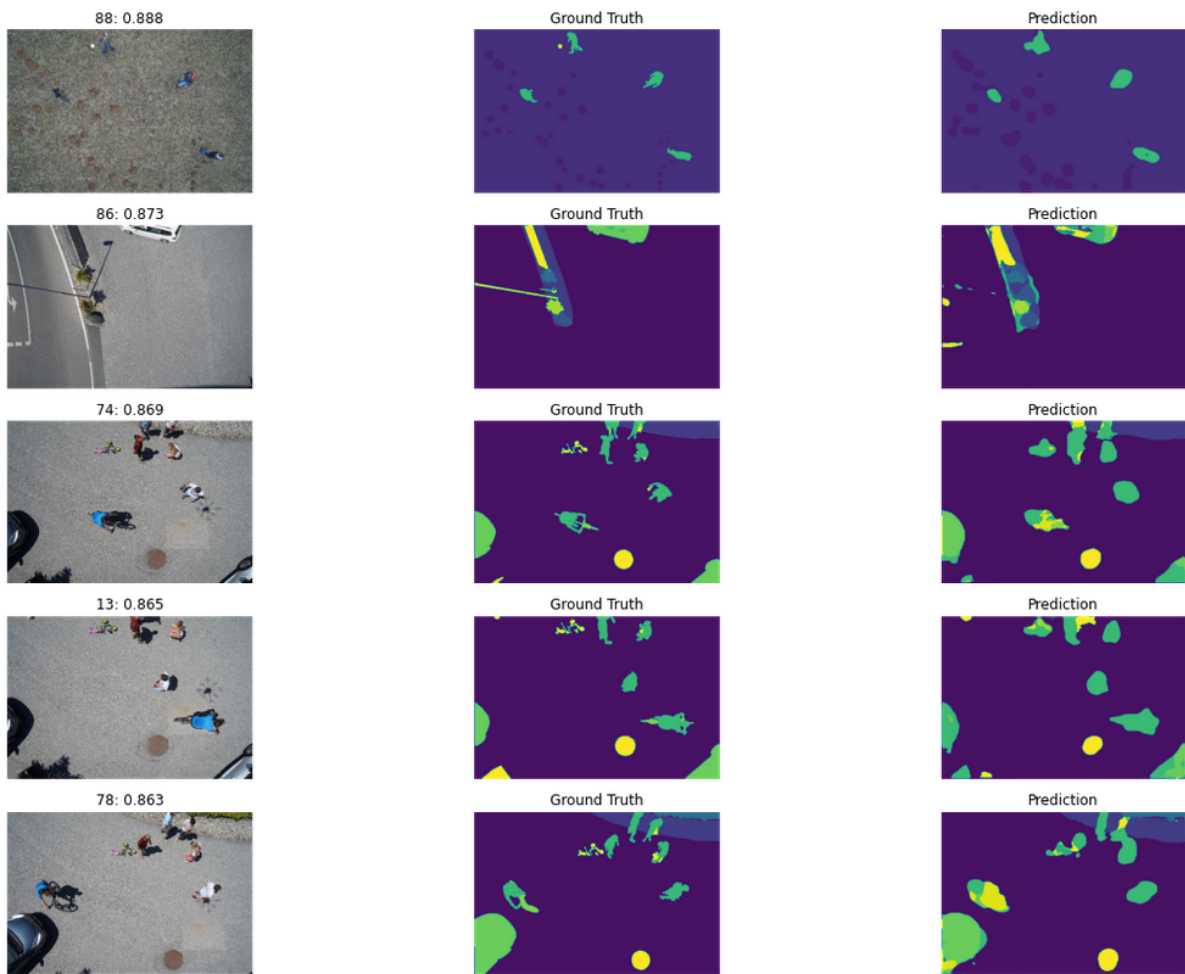
## Analisi dei risultati sul test set

Al fine di valutare la correttezza del modello abbiamo deciso di visualizzare le migliori e le peggiori 15 maschere predette ordinate per indice di Jaccard.

Le immagini seguenti mostrano le migliori 10 maschere predette:

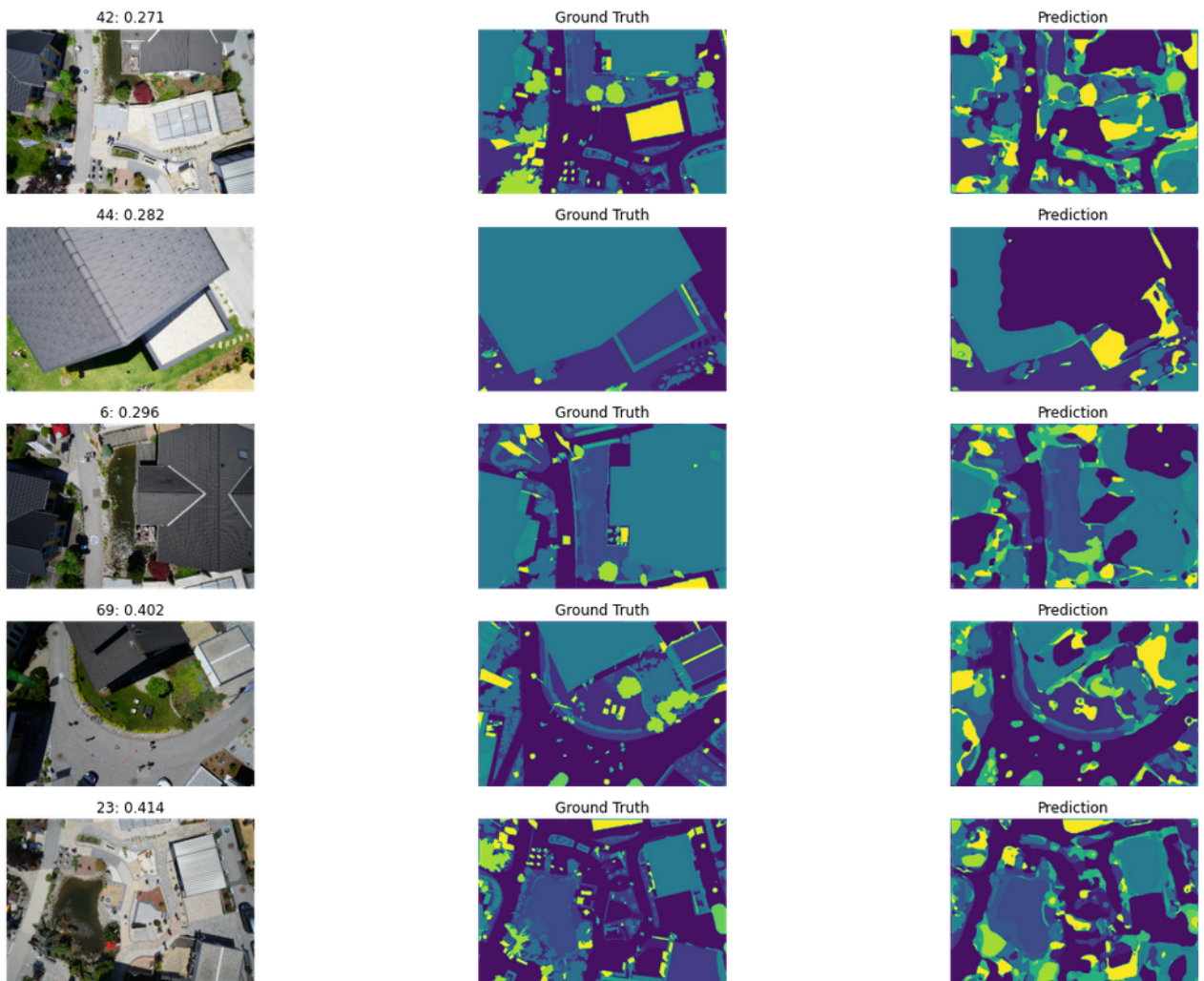


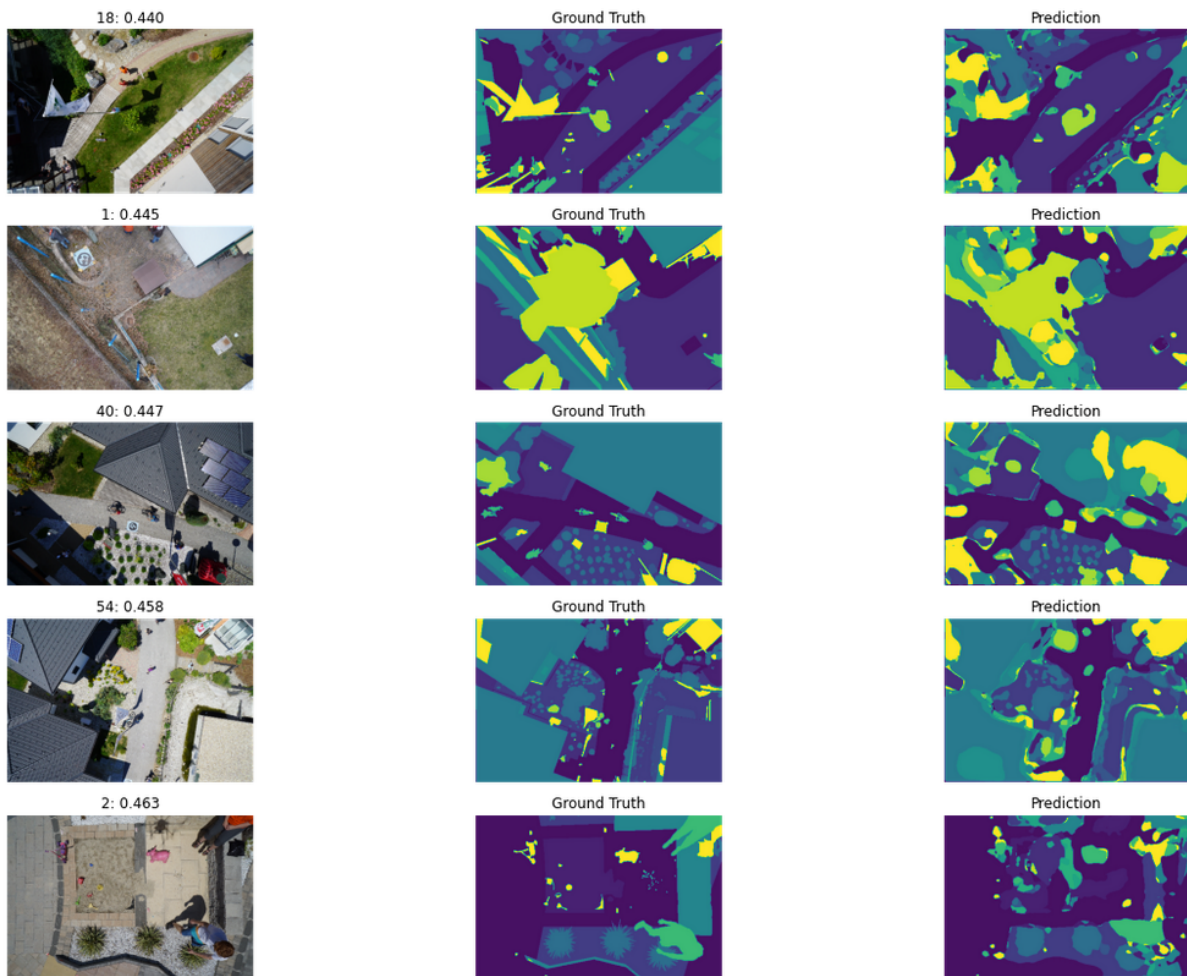




Ad occhio nudo si vede che il modello lavora correttamente.

Le seguenti immagini mostrano le 10 peggiori:





Dalle immagini sopra riportate si può notare come il modello riesca a segmentare meglio immagini meno complesse dove compaiono meno oggetti o oggetti distanti tra loro. Il modello fatica nelle immagini in cui sono presenti più oggetti o più oggetti ravvicinati tra loro.

## **Considerazioni finali**

Alla luce dei risultati ottenuti, si ritiene che il problema sia stato risolto raggiungendo dei buoni risultati malgrado le difficoltà intrinseche nel dataset derivanti dal numero delle classi e dal limitato numero di immagini.

Si crede che ci sia del margine di miglioramento, in primis si potrebbero provare altre tecniche di data augmentation come rotazione, luminosità e zoom stando attenti alle maschere. Si potrebbe fare un preprocessing più approfondito e addirittura provare altre reti neurali allo stato dell'arte.

Abbiamo constatato che in questo genere di problemi si spende molto tempo negli addestramenti, nel tuning degli iperparametri e a causa di crash improvvisi della macchina dovuti all'out-of-memory; disporre dell'hardware performante in questo ambito fa la differenza in termini di tempo e performance nell'addestramento dei modelli.

## **Eventuali sviluppi futuri**

Una delle prime cose che faremo sarà quella di sottomettere la nostra soluzione a Kaggle per confrontarci con altri pareri e consigli e anche verificare sul loro test le performance.

Dopo questa esperienza abbiamo deciso di formare un team su Kaggle insieme ad altri utenti per metterci alla prova in problemi di visione artificiale e riconoscimento anche su competizioni aperte essendo stimolanti e importanti per il nostro futuro professionale.

## **Conclusioni**

In conclusione riteniamo che lo sviluppo del progetto è stato per entrambi molto interessante e formativo, sia dal punto di vista didattico che professionale.

Siamo entusiasti di aver fronteggiato un problema ritenuto complesso per la sua natura e per quanto descritto sopra.

Ci riteniamo molto soddisfatti dei risultati ottenuti paragonabili anche ai migliori risultati ottenuti da altri utenti su Kaggle.

Siamo certi che gli argomenti trattati in questo corso siano molto attuali, soprattutto se applicati in campi come la medicina, l'ambiente e il settore tecnologico, siano capaci di portare grandi benefici alla società.