



UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA

Giuseppe Condorelli

Classificazione di immagini con steganografia UERD tramite
deep learning

PROGETTO DI MULTIMEDIA

Anno Accademico 2022 - 2023

Abstract

La steganografia è una tecnica utilizzata per nascondere informazioni all'interno di altri dati, come immagini o video, avendo come obiettivo quello di essere il meno scopribile possibile. In questo lavoro di ricerca, propongo un approccio basato sul deep learning per la classificazione di immagini con steganografia costruendo un Convolutional Neural Network. Il mio obiettivo è quello di sviluppare un classificatore in grado di distinguere tra immagini con messaggi nascosti e immagini senza steganografia. Ho utilizzato un dataset di immagini bilanciato, composto da immagini "cover" senza steganografia e corrispondenti immagini "stego" con messaggi nascosti. Sono state condotte delle analisi approfondite sfruttando statistiche di primo e secondo ordine sull'intero dataset al fine di studiarne la natura del problema. L'esplorazione del dato ha esposto la alta difficoltà nell'affrontare il problema proposto. I risultati sperimentali ottenuti con vari CNN non hanno dato buoni risultati mantenendosi con un'accuracy del $\sim 50\%$ medio e riuscendo a raggiungere un 50% di F1 macro score.

Acknowledgements

Contents

Abstract	i
1 Introduzione	1
2 Stato dell'Arte	2
3 Dataset	4
3.1 IStego100K	4
3.2 Preprocessing	4
3.3 Esplorazione dei dati	5
4 Metodo Proposto	12
4.1 Introduzione	12
4.2 Preprocessing	13
4.3 ImageDataGenerator	16
4.4 CNN	17
4.5 Metriche e grafici	17
5 Esperimenti	19
5.1 Primo esperimento	19
5.2 Secondo esperimento	22
5.3 Terzo esperimento	25
5.4 Quarto esperimento	28
5.5 Quinto esperimento	31
5.6 Sesto esperimento	34
5.7 Settimo esperimento	37
5.8 Ottavo esperimento	40
6 Conclusioni	43

Chapter 1

Introduzione

La steganografia, un campo della sicurezza informatica, si occupa di occultare informazioni all'interno di dati esistenti come immagini, audio o video, in modo che il messaggio nascosto risulti impercettibile all'osservatore. Questa tecnica ha applicazioni in diversi settori, tra cui sicurezza dei dati, protezione della privacy e sorveglianza.

Negli ultimi anni, l'applicazione di tecniche di machine learning alla steganalisi ha suscitato un vivo interesse nella ricerca. L'obiettivo è sviluppare algoritmi e modelli in grado di individuare la presenza di steganografia nelle immagini e distinguere tra immagini "cover" e immagini "stego". Questo tipo di classificazione è utile in scenari in cui è necessario identificare informazioni nascoste per motivi di sicurezza o investigativi.

Nel presente progetto, si approfondirà l'algoritmo di steganografia UERD [1], concentrandosi su una specifica analisi della steganografia con l'obiettivo di individuare immagini che utilizzano questo algoritmo per nascondere messaggi. A tal fine, verrà utilizzato il dataset IStego100k [2], che include, oltre all'algoritmo UERD, anche altri algoritmi come J-uniward [3], nsF5 [4] e HILL GINA [5], che non saranno oggetto di trattamento specifico. Attraverso osservazioni e pre-elaborazione opportune, si addestrerà una rete neurale convoluzionale specializzata per la classificazione e la discriminazione tra immagini con e senza steganografia UERD.

Chapter 2

Stato dell'Arte

Nella letteratura, sono stati proposti diversi approcci per la classificazione di immagini con steganografia utilizzando tecniche di machine learning.

Marcelinus Henry Menori, nel suo studio "Blind steganalysis for digital images using support vector machine method" Ref. [6], ha condotto un'analisi steganografica utilizzando la tecnica di estrazione delle caratteristiche proposta da Tomas [7]. Questa tecnica si basa sul calcolo delle differenze tra i pixel delle immagini e sull'utilizzo delle catene di Markov per l'estrazione delle features. Successivamente, ha addestrato un Support Vector Machine (SVM) ottenendo un'accuratezza del 73% per le immagini in scala di grigi e del 61% per le immagini a colori.

T. -S. Reine et. in Ref. [8] offrono una dettagliata ricerca riguardo l'applicazione del Deep Learning (DL) nella steganalisi, fornendo una panoramica cronologica sull'evoluzione del campo. Sin dal 2014, sono state proposte diverse reti neurali convoluzionali (CNN) per l'apprendimento non supervisionato, dimostrando risultati superiori a quelli ottenuti dai metodi tradizionali come SRM e SPAM. Queste proposte hanno costituito una solida base per ulteriori ricerche. L'analisi inoltre rivela una vasta varietà di approcci, inclusi set di addestramento specifici per le CNN, trasferimento di parametri tra reti, steganalisi quantitativa, considerazione di immagini di dimensioni arbitrarie, miglioramento dei database di immagini e considerazione degli effetti di "Cover-Source Mismatch" negli esperimenti. Le architetture delle CNN proposte, come QianNet, YuNet, YeNet, YedroudjNet e ZhuNet nel dominio spaziale, insieme all'applicazione di ResNet per la steganalisi nel dominio della frequenza (JPEG), hanno mostrato risultati di rilevamento ottimali nel

dominio spaziale con CNN ZhuNet. È interessante notare l’impiego della metodologia GAN per automatizzare il processo di steganografia, che coinvolge due CNN in competizione, una per la steganografia e l’altra per la steganalisi. La revisione dimostra che l’applicazione del DL alla steganalisi ha superato i risultati dei metodi tradizionali. Sono state sviluppate architetture specifiche per affrontare le sfide della steganalisi, introducendo componenti di rete specializzate come TLU e funzioni di attivazione gaussiane. Tuttavia, i livelli attuali di rilevamento sono ancora lontani dagli obiettivi della comunità di ricerca, come indicato nella conclusione dell’articolo.

Tang et. in Ref. [9] ha condotto una ricerca simile a T. -S. Reine. Nell’articolo viene esplicitata la differenza di steganalisi in dominio spaziale e dominio JPEG. Nel dominio spaziale, sono state proposte reti neurali convoluzionali (CNN) che superano i metodi tradizionali come SRM e SRM+EC. Nel dominio JPEG, le CNN sono utilizzate per migliorare la steganalisi sfruttando la consapevolezza della fase di compressione utilizzata da JPEG. L’applicazione del Deep Learning ha portato a miglioramenti significativi, ma sono necessari ulteriori sviluppi per affrontare le sfide dei moderni algoritmi di steganografia adattiva.

Chapter 3

Dataset

3.1 IStego100K

Il dataset utilizzato per addestrare e valutare il classificatore di steganografia è stato ottenuto da IStego100K [2], un ampio repository di immagini con e senza steganografia. IStego100K contiene un totale di 100.000 coppie di immagini, ognuna composta da un’immagine di copertura (senza steganografia) e la sua controparte con steganografia.

Le immagini sono state raccolte da varie fonti e presentano una varietà di contenuti e formati. È importante notare che le immagini del dataset sono state controllate per garantire la loro legalità e non violare i diritti d’autore.

Si prega di fare riferimento al repository IStego100K [2] per ulteriori informazioni sul processo di raccolta delle immagini e la loro origine.

3.2 Preprocessing

Uno dei primissimi passi cruciali nel processo è stato quello di affrontare la sfida della limitata disponibilità di risorse RAM e computazionali della macchina utilizzata per condurre l’esperimento. Per adattare il problema a tali risorse, si è reso necessario un intervento mirato.

Considerando che si trattava di immagini a colori di dimensioni considerevoli, 1024x1024 pixel, e tenendo conto del grande numero di campioni da analizzare, si è

giunti alla conclusione che ottenere risultati rapidi sarebbe stato altrimenti impraticabile mantenendo l'intero dataset.

Per superare questa sfida, si è presa la decisione strategica di concentrarsi esclusivamente sulle immagini steganografate con l'algoritmo UERD. Questa scelta ha comportato una significativa riduzione del numero di immagini del dataset, passando da 100.000 a 33.416 coppie.

Oltre a semplificare notevolmente l'analisi, questa decisione ha permesso di ottimizzare l'efficienza del processo di steganalisi, consentendo di concentrare le risorse computazionali in modo più mirato e ottenere risultati in tempi ragionevoli.

3.3 Esplorazione dei dati

L'obiettivo dell'esplorazione dei dati è comprenderne la natura al fine di sviluppare una strategia ottimale per correttamente classificare un'immagine che contiene un messaggio nascosto. A tale scopo, è stata eseguita un'analisi sia visiva che statistica su un campione di immagini presenti nel dataset.

L'esplorazione delle immagini è stata condotta convertendole da RGB a scala di grigi per semplificarne l'analisi.

Il primo passo è stato quello di confrontare direttamente una serie di immagini "cover" con le loro controparti "stego", osservando attentamente entrambe le versioni.

Come mostrato nell'immagine Fig.3.1, le differenze visive a livello percettivo tra le due versioni, sebbene presenti come evidenziato dalla loro differenza puntuale, sono praticamente trascurabili, come ci si aspetterebbe da un buon algoritmo di steganografia.

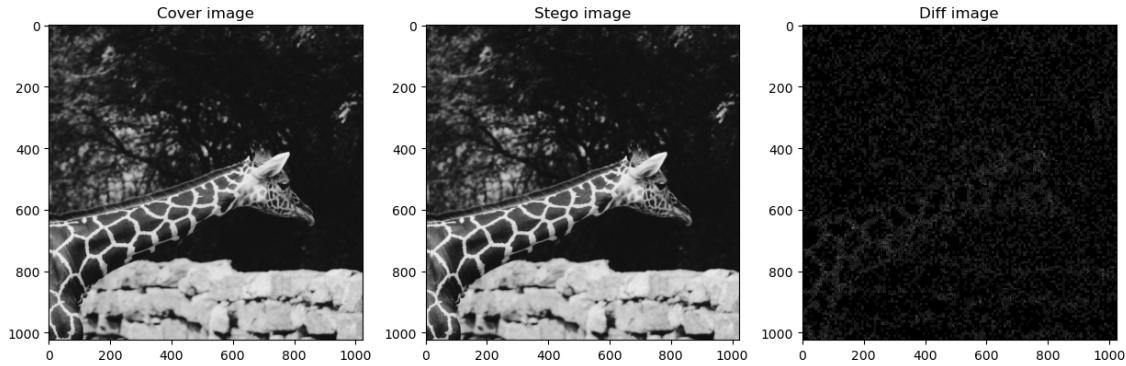


Figure 3.1: Confronto tra cover e stego dell’immagine 001123 in scala di grigi

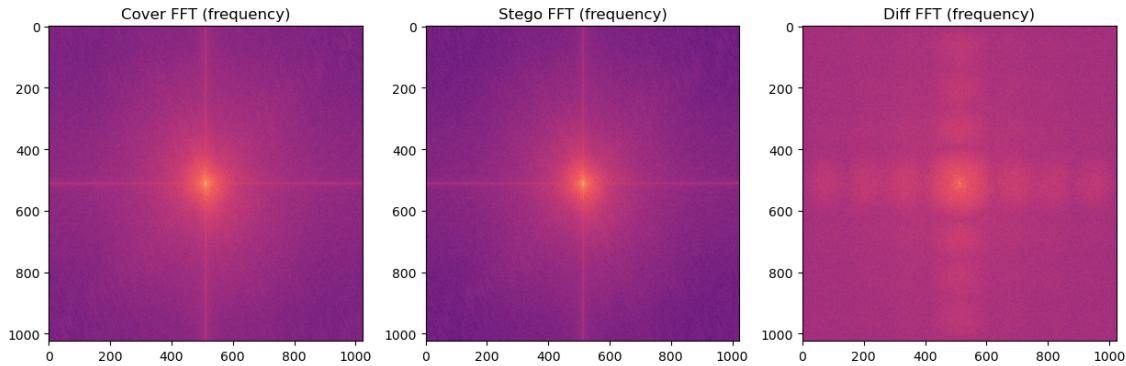


Figure 3.2: Confronto tra l’immagine ”cover” e ”stego” nel dominio delle frequenze dell’immagine 001123

Da questa prima osservazione, possiamo già intuire la sfida nel distinguere le due varianti.

Il passaggio successivo è stato quello di spostarsi nel dominio delle frequenze per analizzare lo spettro di entrambe le immagini al fine di individuare eventuali caratteristiche rilevanti che potessero fornire indizi sulla presenza di steganografia.

Nella Figura Fig.3.2, riportata di seguito, è possibile osservare il confronto tra le immagini ”cover” e ”stego” dell’immagine 001123 nel dominio delle frequenze.

Come nel primo passo, anche in questo caso non si notano differenze significative. Tuttavia, una leggera caratteristica che risalta osservando lo spettro di entrambe le immagini è la presenza di alcune frequenze leggermente attenuate nell’immagine con

steganografia.

Nella Figura Fig.3.2, è possibile visualizzare questa sottile differenza nel comportamento delle frequenze tra le due immagini.

In seguito, considerando l'incapacità di rilevare sottili differenze visive sia a livello percettivo che nel dominio delle frequenze, sono state calcolate diverse misure descrittive delle immagini. Queste misure includono statistiche di primo e secondo ordine nel dominio delle frequenze, nonché il valore massimo e l'energia totale. Inoltre, dato il tipo di algoritmo utilizzato (UERD), sono state calcolate anche statistiche simili per i coefficienti DCT, insieme all'entropia dell'immagine stessa.

Come si può osservare nella Tabella Tab.3.1, le differenze sono estremamente minime.

Table 3.1: Caratteristiche delle immagini

Features	FFT mean	FFT std	FFT max	Total energy
Cover image	31.907080	432.102943	365634.505891	1.968502e+11
Stego image	31.846040	432.109147	365643.635547	1.968518e+11
Diff image	1.647518	1.966960	1307.960539	6.903032e+06

Features	DCT mean	DCT std	DCT cov	Entropy
Cover image	0.000243	0.423124	1740.459961	6.528988
Stego image	0.000249	0.423126	1702.329224	6.518282
Diff image	0.000008	0.002506	333.112518	-

Un ulteriore conferma delle differenze quasi impercettibili è stata ottenuta tramite il calcolo della distanza del coseno tra le due immagini. Questa misura fornisce un valore vicino a 1 quando le immagini sono identiche, e in questo caso specifico il valore ottenuto è 0.9999824663599037.

È stata anche condotta un'analisi approfondita dei bit planes dell'immagine sia nella sua forma originale Fig. 3.3 (cover) che nella forma modificata Fig. 3.4 (stego).

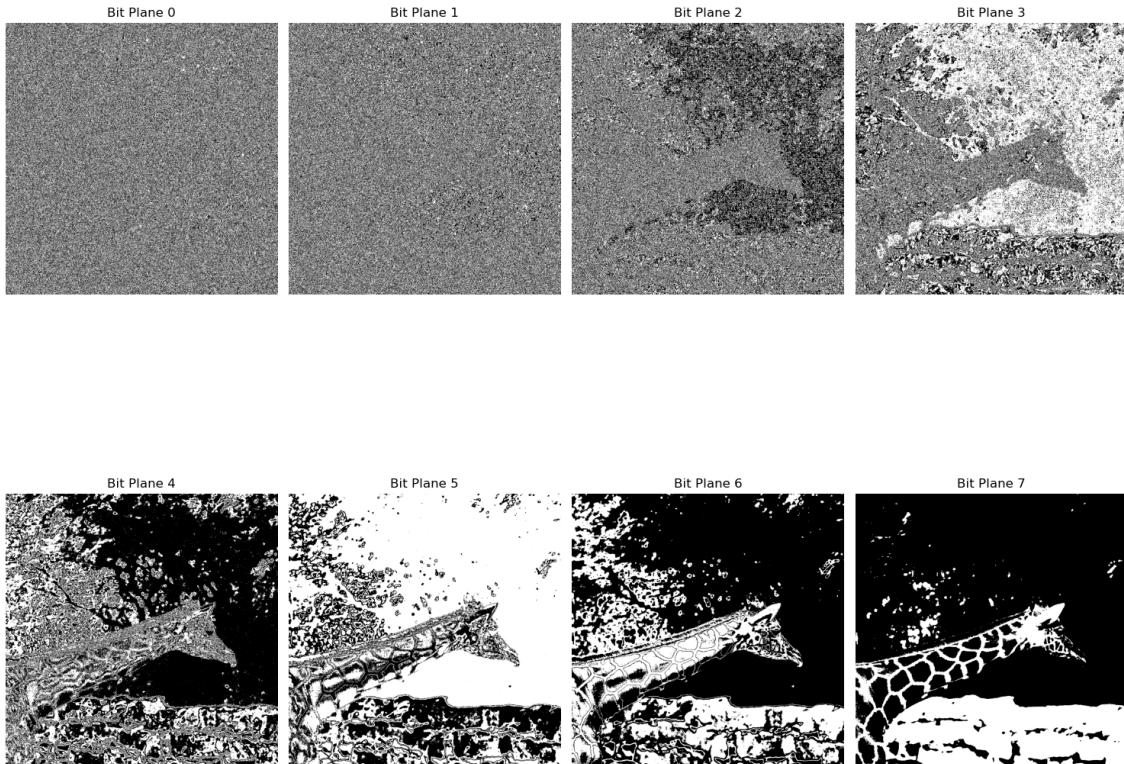


Figure 3.3: Cover Bit Planes dell'immagine 001123

Questa analisi ci ha permesso di evidenziare il tipico comportamento di un buon algoritmo di steganografia, il cui obiettivo principale è quello di passare inosservato dal punto di vista percettivo e, in secondo luogo, dalle tecniche di steganalisi.

Come è possibile notare Fig. 3.5, nonostante le differenze tra le due varianti siano difficilmente distinguibili ad occhio nudo, è nelle porzioni meno significative dei bit (LSB) che si riscontrano le differenze sostanziali. Questo fenomeno è dovuto al fatto che tali bit contengono dettagli estremamente sottili che non sono percepibili dal punto di vista visivo.

Per concludere la nostra analisi, abbiamo calcolato le statistiche (FFT, DCT, Entropy, Cosine Similarity) sull'intero dataset al fine di ottenere una visione generale e trarre conclusioni significative. Questo approccio ci ha permesso di valutare le differenze statistiche nel loro insieme e di comprendere meglio la presenza di steganografia UERD.

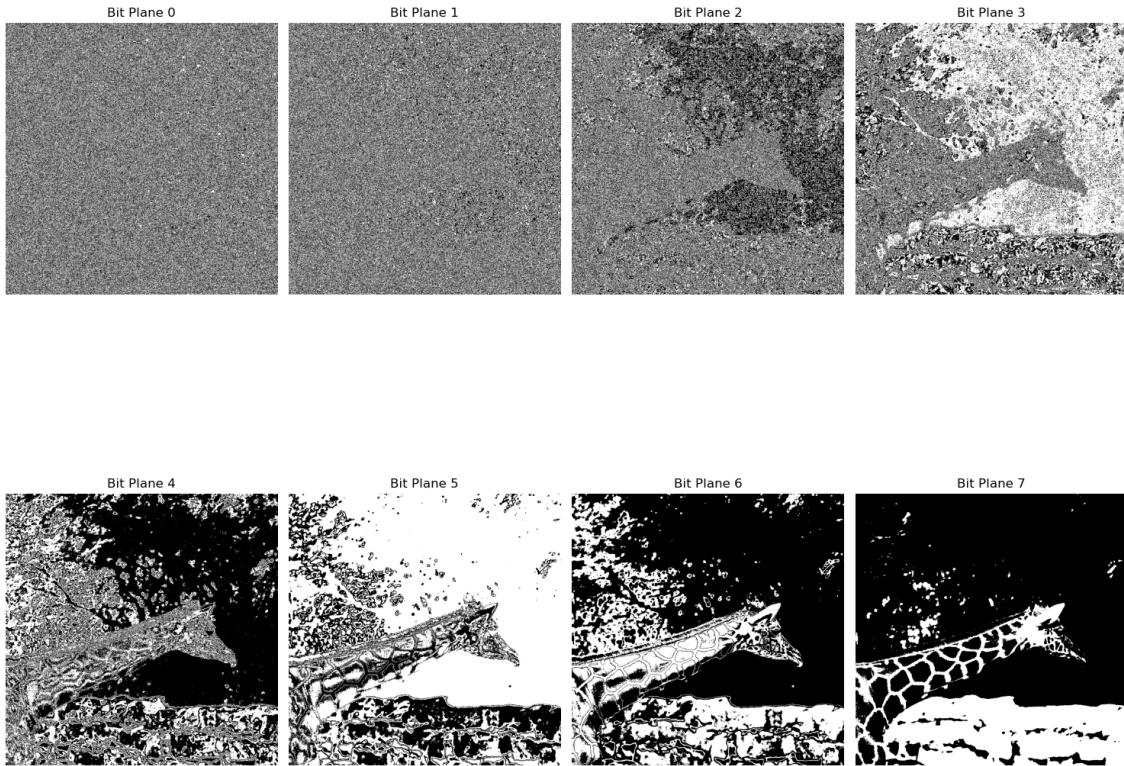


Figure 3.4: Stego Bit Planes dell’immagine 001123

Come mostrato nelle gaussiane rappresentate nella Figura 3.6, è evidente che le differenze statistiche sull’intero set di dati sono praticamente inesistenti. Questo risultato ci indica che le caratteristiche considerate presentano un’omogeneità significativa, rendendo difficile l’individuazione di tratti distintivi tra le immagini con e senza steganografia.

Inoltre, la gaussiana relativa alla similarità del coseno tra le immagini di cover e stego (Figura 3.7) conferma in maniera ancora più decisa l’assenza di differenze significative. La sovrapposizione quasi completa delle distribuzioni di similarità del coseno sottolinea la scarsa rilevabilità della steganografia UERD attraverso questa metrica.

Questi risultati evidenziano la sfida che rappresenta l’individuazione di immagini con steganografia UERD basata sulle caratteristiche statistiche considerate. È

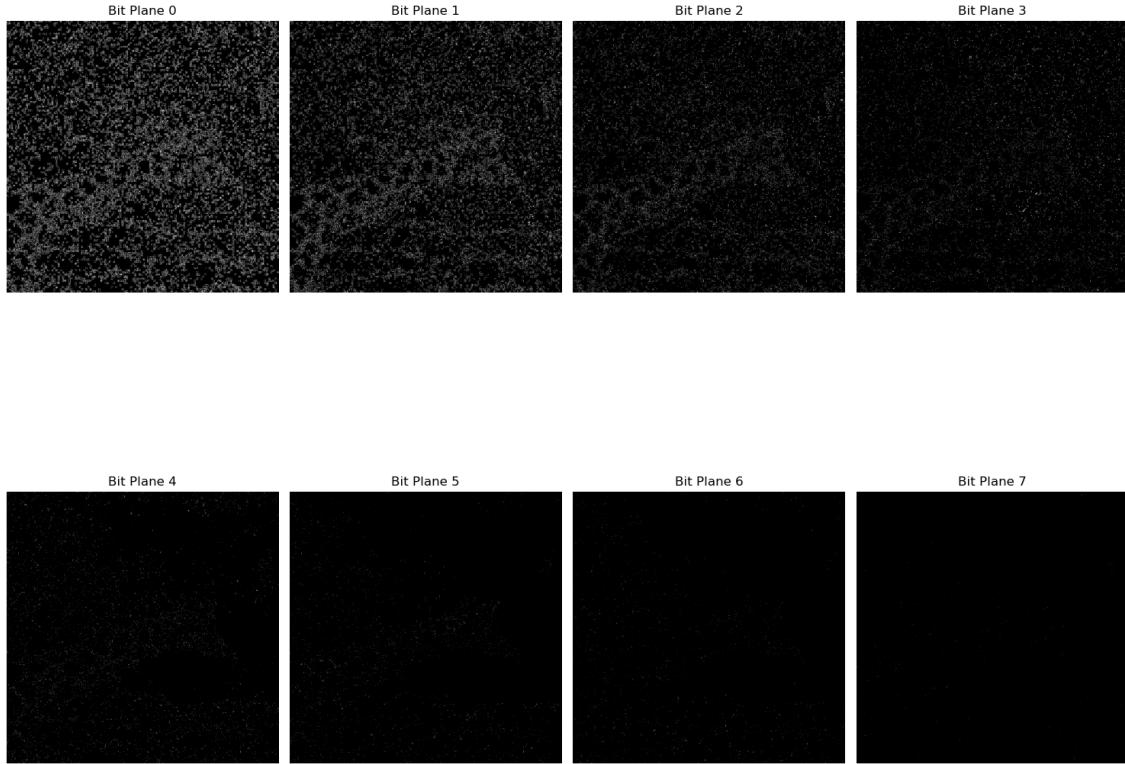


Figure 3.5: Differenza tra cover e stego Bit Planes dell'immagine 001123

necessario adottare approcci più sofisticati e specifici per affrontare questa sfida e sviluppare strumenti di analisi più avanzati.

In conclusione, l'analisi delle statistiche sull'intero dataset ha confermato la presenza di differenze statistiche trascurabili e la difficoltà di individuare immagini con steganografia UERD utilizzando le metriche considerate. Questo lavoro fornisce una base solida per futuri studi e sviluppi nell'ambito della steganalisi.

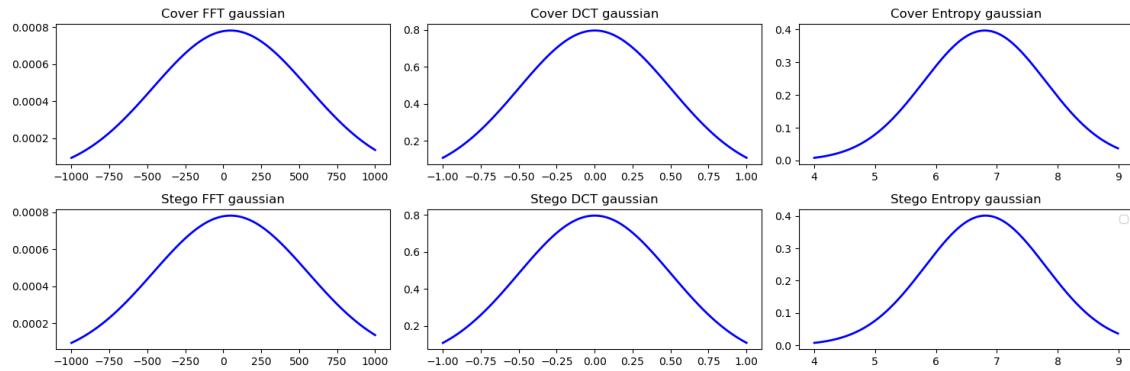


Figure 3.6: Guassiane dell'intero dataset

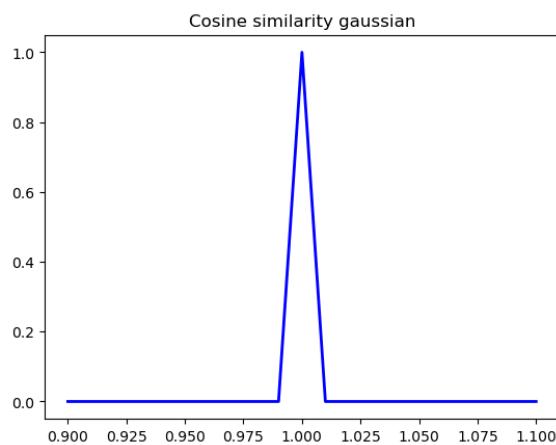


Figure 3.7: Guassiana dell'intero dataset sulla similarità del coseno tra cover e stego

Chapter 4

Metodo Proposto

4.1 Introduzione

In questo capitolo, esamineremo le ragioni alla base delle decisioni prese per affrontare il problema proposto, basandoci sull'analisi esplorativa dei dati presentata nel Capitolo 3.3. Concentreremo la nostra attenzione sul preprocessing delle immagini e discuteremo i risultati ottenuti utilizzando una rete neurale convoluzionale (CNN).

Approfondiremo le scelte fatte per il preprocessing delle immagini, comprendendo le trasformazioni e le operazioni applicate per preparare i dati prima di introdurli al modello CNN. Esploreremo le motivazioni dietro queste decisioni e come esse possano influenzare l'efficacia del modello nell'affrontare il problema specifico.

Inoltre, analizzeremo i risultati ottenuti utilizzando la CNN. Discuteremo le metriche di valutazione, come l'accuracy, e valuteremo l'efficacia della rete nell'affrontare il problema in esame. Esploreremo eventuali sfide incontrate durante l'addestramento e l'interpretazione dei risultati.

Attraverso questa analisi dettagliata, acquisiremo una comprensione più profonda delle decisioni prese e dei risultati ottenuti utilizzando la CNN per risolvere il problema proposto. Questo ci consentirà di valutare l'efficacia complessiva del nostro approccio e considerare eventuali miglioramenti o alternative per futuri sviluppi.



Figure 4.1: Confronto tra le varianti cover e stego originali

4.2 Preprocessing

Le immagini fornite non possono essere analizzate per intero da un modello di deep learning CNN a causa delle scarse risorse della macchina utilizzata, principalmente dovute alle loro dimensioni di 1024x1024 pixel. Per affrontare questo problema, sono state considerate due diverse soluzioni. La prima e più semplice opzione è applicare uno scaling a tutte le immagini per ridurle da 1024x1024 a 128x128 pixel, che sperimentalmente si è dimostrata la dimensione massima accettabile in termini di tempi di elaborazione. La seconda opzione consiste nell'effettuare un crop di tutte le immagini, dividendole in sotto-immagini più piccole di dimensione 128x128.

Al fine di prendere una decisione, è stata considerata l'immagine della giraffa utilizzata durante l'esplorazione dei dati (vedi sezione 3.3) per valutare gli effetti di ridimensionamenti o crop su un'immagine contenente steganografia.

Per condurre questi esperimenti, abbiamo utilizzato [Resemble.js](#), sviluppato da James Cryer, una piattaforma online che permette di confrontare due immagini evidenziandone le differenze nel dominio spaziale, risultando molto utile per il nostro caso d'uso.

Il primo passo compiuto prima di applicare scaling o cropping è stato osservare le differenze tra la variante cover e la stego, per avere un'idea chiara della situazione iniziale.

Il risultato ottenuto nella Figura 4.1 ci offre un ottimo punto di partenza per



Figure 4.2: Confronto tra le varianti cover e stego con scaling



Figure 4.3: Confronto tra le varianti cover e stego con cropping

valutare la possibilità di attuare il cropping o lo scaling. Il nostro obiettivo è quello di mantenere intatte, o eventualmente accentuare, le differenze tra le due immagini, al fine di consentire ai modelli di deep learning di ricavarne la funzione che permetta la classificazione tra le due.

Con questo obiettivo in mente, ho inizialmente provato applicando una trasformata di scaling all'immagine, riducendola dalle dimensioni originali di 1024x1024 pixel a 128x128 pixel. Come si può notare dalla Figura 4.2, la trasformazione non ha disperso le differenze, ma le ha anzi amplificate ulteriormente.

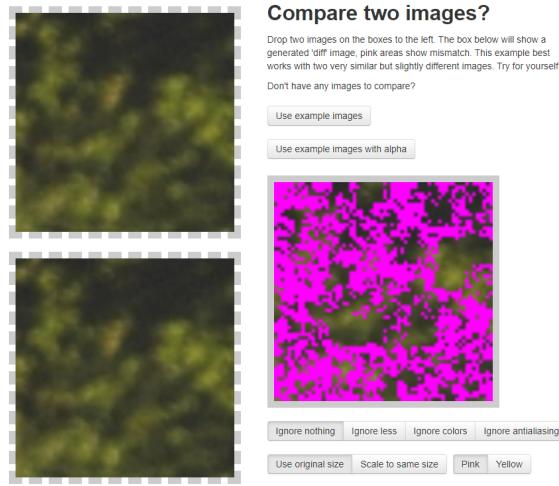


Figure 4.4: Confronto tra le varianti cover e stego con cropping e scaling

D'altra parte, il cropping, come evidenziato nella Figura 4.3, permette di mantenere intatte le differenze, raggiungendo lo stesso obiettivo.

Dato il successo di entrambe le metodologie, ho deciso di combinare entrambe le tecniche, effettuando uno scaling di una porzione ritagliata dall'immagine di partenza, ottenendo così l'immagine finale di dimensioni 64x64 pixel (Figura 4.4). Questa pipeline ha dimostrato di fornire i migliori risultati, accentuando notevolmente le differenze tra le varianti cover e stego e permettendo, tramite la divisione dell'immagine in più sottoimmagini, di attuare una strategia di data augmentation utile per consentire al modello di avere più dati da elaborare.

L'attuazione di questa strategia mi ha permesso di ottenere un dataset molto popolato e facilmente trattabile dalla mia macchina.

Nella tabella Tab. 4.1 e' possibile visionare un riassunto del preprocessing applicato al dataset con le dimensioni in pixel e numero di campioni.

Table 4.1: Tabella riassuntiva dataset

Preprocessing	Dimensioni	Campioni
Nothing	1024x1024	24.428
Scaling	128x128	24.428
Cropping	128x128	3.126.784
Cropping e scaling	64x64	3.126.784

4.3 ImageDataGenerator

Per caricare il set di immagini su python per poter essere utilizzati da un modello CNN abbiamo utilizzato ImageDataGenerator fornito dalla libreria Keras.

ImageDataGenerator, fornito dalla libreria [10], è un ottimo tool che permette, specificato il path del set di immagini ed altri parametri discussi più avanti, di fare il load delle immagini in memoria ed applicarne alcune trasformate al fine di adattare i dati al modello nonché farne Data Augmentation utile per migliorarne ulteriormente le prestazioni.

Il costruttore è così definito:

```
ImageDataGenerator(
    validation_split=0.2,
    rescale=1./255)
```

- **Validation_split:** questo parametro permette di fare uno split del nostro dataset in training e validation. È una caratteristica molto comoda del tool che semplifica ulteriormente la gestione dei dati in modo molto semplice e veloce. Mettendo come valore 0.2, abbiamo così creato un training set dell'80% e un validation set del 20% rispetto al totale del dataset.
- **Rescale:** questo parametro permette di fare una normalizzazione dei valori contenuti nelle immagini (che sappiamo andare da 0 a 255), così da portarli nel range [0, 1]. Questa azione è molto importante poiché favorisce ulteriormente l'apprendimento di un modello di Deep learning.

4.4 CNN

L'esplorazione dei dati svolta nel Capitolo 3.3 ha rivelato la complessità del problema di steganalisi proposto sull'algoritmo UERD.

Dall'analisi risulta molto chiaro come le due distribuzioni di immagini cover e stego siano molto simili tra di loro, il che rende difficile riuscire a trovare delle caratteristiche determinanti per stimare un threshold che permetta una distinzione tra le due.

Vista la difficoltà nel trovare una buona funzione di rappresentazione per il dataset, ho optato per la realizzazione di un modello CNN (Convolutional Neural Network), più o meno complesso, che permetesse, tramite l'applicazione di convoluzioni alle immagini del dataset, di riuscire a cogliere le lievi e quasi impercettibili differenze tra le due classi.

Al fine di ottenere un CNN capace di fare ciò, ho adottato una metodologia bottom-up in cui sono partito con modelli molto semplici (con pochi layer) aumentandone progressivamente la complessità (aggiungendo più layer alla rete).

Tutti gli esperimenti sono stati condotti con un numero ridotto di epoche (10) con 50 passi per epoca e 25 per la fase di validation, al fine di permettere una più rapida sperimentazione che altrimenti non sarebbe stata possibile.

Questo metodo mi ha permesso di condurre esperimenti più o meno esaustivi, evitando esecuzioni troppo lunghe e consentendo di aggiustare di volta in volta il modello in base ai risultati ottenuti.

4.5 Metriche e grafici

Le metriche e grafici utilizzati sono:

- **Training and Validation Accuracy:** Questo grafico permette di avere una visuale completa sull'andamento dell'accuracy con lo scorrere delle epoche al fine di comprendere il comportamento del modello in fase di training.
- **Training and Validation Loss:** Questo grafico permette di capire se il modello sta riuscendo a convergere col passare delle epoche.
- **ROC:** Questo grafico si basa sulle metriche True Positive Rate e False Positive Rate e ci permette di comprendere quanto il nostro modello sta andando bene sul test set.
- **Confusion Matrix:** Una importante matrice che mostra le predizioni del modello rispetto alle classi reali al fine di stimarne altre metriche utili per l'analisi.
- **F1 score:** La metrica più utilizzata che ci permette di avere un valore reale sull'accuratezza del nostro modello sul test set.

Durante gli esperimenti, osserveremo attentamente tutti i grafici e le metriche citate al fine di valutarne correttamente le capacità, al fine di implementare strategie migliori per la costruzione del modello.

Chapter 5

Esperimenti

5.1 Primo esperimento

Table 5.1: Modello primo esperimento

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 62, 62, 128)	3584
max_pooling2d_2 (MaxPooling2D)	(None, 31, 31, 128)	0
flatten_2 (Flatten)	(None, 123008)	0
dense_8 (Dense)	(None, 64)	7872576
dense_9 (Dense)	(None, 2)	130
Total params		7,876,290
Trainable params		7,876,290
Non-trainable params		0

Il primo modello risulta essere molto semplice. Infatti presenta solo 1 layer di Convolution ed 1 di avg pooling per poi intrecciarsi con due layer densi di cui l'ultimo utile alla classificazione binaria mediante la funzione di attivazione Softmax.

Output epochs:

```

Epoch 1/10
50/50 [=====] - 54s 1s/step - loss: 1.1067 - accuracy: 0.5031 - val_loss: 0.6938 -
    val_accuracy: 0.4978
Epoch 2/10
50/50 [=====] - 52s 1s/step - loss: 0.6949 - accuracy: 0.4930 - val_loss: 0.6941 -
    val_accuracy: 0.5028
Epoch 3/10
50/50 [=====] - 52s 1s/step - loss: 0.6941 - accuracy: 0.4948 - val_loss: 0.6941 -
    val_accuracy: 0.4988
Epoch 4/10
50/50 [=====] - 51s 1s/step - loss: 0.6935 - accuracy: 0.5025 - val_loss: 0.6938 -
    val_accuracy: 0.4825
Epoch 5/10

```

```

50/50 [=====] - 54s 1s/step - loss: 0.6939 - accuracy: 0.4967 - val_loss: 0.6933 -
    val_accuracy: 0.4994
Epoch 6/10
50/50 [=====] - 60s 1s/step - loss: 0.6936 - accuracy: 0.5005 - val_loss: 0.6951 -
    val_accuracy: 0.4916
Epoch 7/10
50/50 [=====] - 51s 1s/step - loss: 0.6935 - accuracy: 0.4978 - val_loss: 0.6936 -
    val_accuracy: 0.4991
Epoch 8/10
50/50 [=====] - 51s 1s/step - loss: 0.6934 - accuracy: 0.4942 - val_loss: 0.6935 -
    val_accuracy: 0.4944
Epoch 9/10
50/50 [=====] - 50s 999ms/step - loss: 0.6938 - accuracy: 0.5083 - val_loss: 0.6938 -
    val_accuracy: 0.5016
Epoch 10/10
50/50 [=====] - 50s 1s/step - loss: 0.6933 - accuracy: 0.5022 - val_loss: 0.6938 -
    val_accuracy: 0.4934

```

Come è possibile analizzare dai grafici proposti (Fig. 5.1(a) e Fig. 5.1(b)) e dall'output riguardo il training nel corso dell'epoch posso confermare che il modello non riesce a convergere, restando bloccato ad un loss di circa 0.69 ed un'accuracy del 50%.

Inoltre il ROC (Fig. 5.1(c)) e la Confusion Matrix (Fig. 5.1(d)) mostrano chiaramente come il modello predica sempre cover il che si traduce in un pessimo risultato.

Table 5.2: Classification Report primo esperimento

Class	Precision	Recall	F1-Score	Support
cover	0.50	0.96	0.66	512
stego	0.44	0.03	0.06	512
Accuracy				0.50
Macro avg	0.47	0.50	0.36	1024
Weighted avg	0.47	0.50	0.36	1024

La tabella 5.2 ci dà un'ulteriore conferma del pessimo risultato, con un F1 macro score del solo 36%.

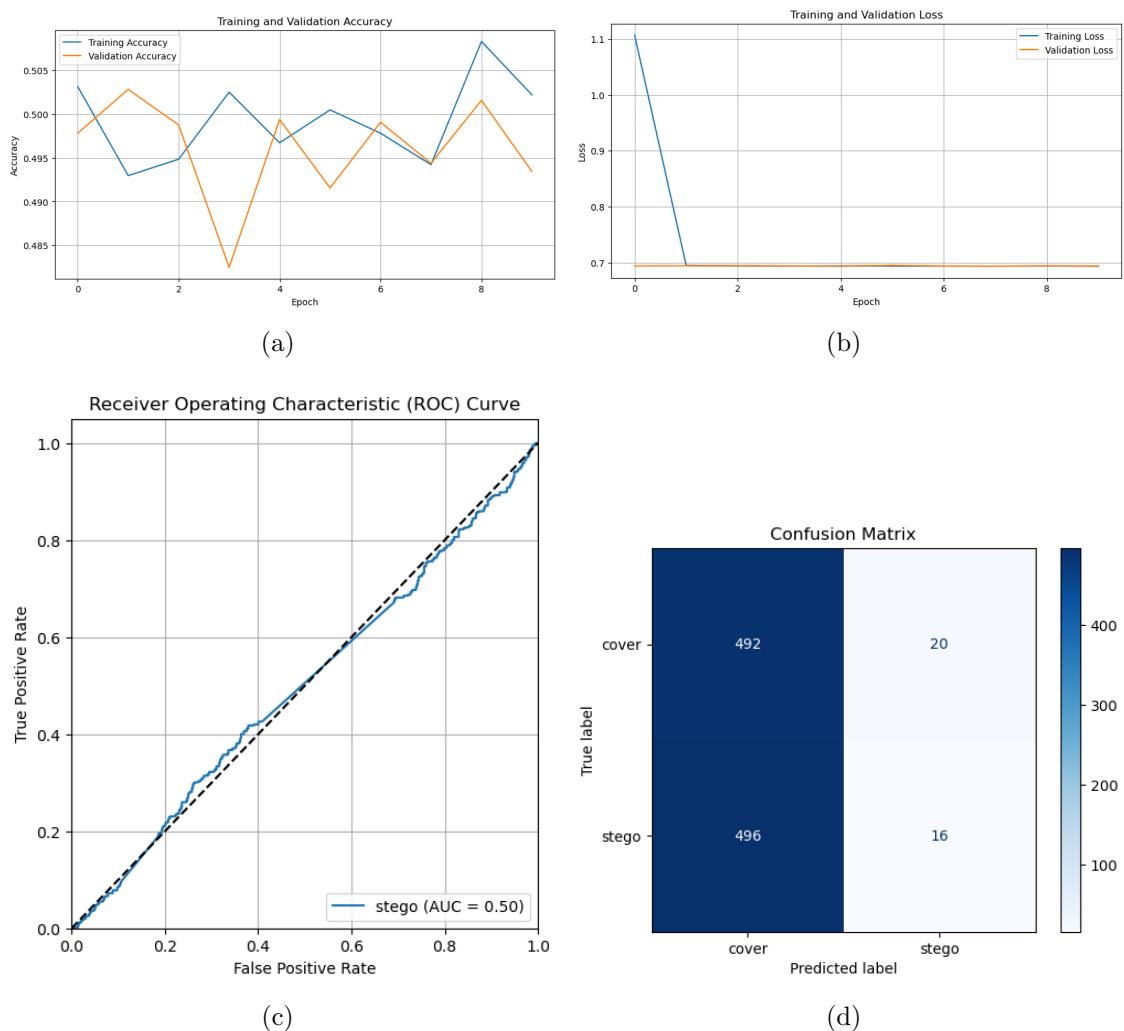


Figure 5.1: Grafici primo esperimento

5.2 Secondo esperimento

Table 5.3: Modello secondo esperimento

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 62, 62, 128)	3584
average_pooling2d (AveragePooling2D)	(None, 31, 31, 128)	0
conv2d_6 (Conv2D)	(None, 29, 29, 64)	73792
average_pooling2d_1 (AveragePooling2D)	(None, 14, 14, 64)	0
flatten_4 (Flatten)	(None, 12544)	0
dense_12 (Dense)	(None, 64)	802880
dense_13 (Dense)	(None, 2)	130
Total params		880,386
Trainable params		880,386
Non-trainable params		0

Visto l'incapacità di convergere del primo modello creato ed allenato nel primo esperimento, in questo secondo ho deciso di complicare leggermente di più il modello aumentandone i livelli di layer. Inoltre ho deciso di utilizzare, invece del MaxPooling2D, l'AveragePooling2D la cui caratteristica permette al modello di essere più sensibile alle piccole variazioni (proprietà importante per affrontare il problema)

Output epochs:

```

Epoch 1/10
50/50 [=====] - 54s 1s/step - loss: 0.7027 - accuracy: 0.5139 - val_loss: 0.6931 -
    val_accuracy: 0.5059
Epoch 2/10
50/50 [=====] - 53s 1s/step - loss: 0.6931 - accuracy: 0.5086 - val_loss: 0.6930 -
    val_accuracy: 0.5081
Epoch 3/10
50/50 [=====] - 53s 1s/step - loss: 0.6931 - accuracy: 0.5077 - val_loss: 0.6932 -
    val_accuracy: 0.4988
Epoch 4/10
50/50 [=====] - 56s 1s/step - loss: 0.6933 - accuracy: 0.4900 - val_loss: 0.6931 -
    val_accuracy: 0.5025
Epoch 5/10
50/50 [=====] - 61s 1s/step - loss: 0.6931 - accuracy: 0.5050 - val_loss: 0.6930 -
    val_accuracy: 0.5163
Epoch 6/10
50/50 [=====] - 74s 1s/step - loss: 0.6933 - accuracy: 0.4945 - val_loss: 0.6932 -
    val_accuracy: 0.4922
Epoch 7/10
50/50 [=====] - 66s 1s/step - loss: 0.6932 - accuracy: 0.4980 - val_loss: 0.6932 -
    val_accuracy: 0.4972
Epoch 8/10
50/50 [=====] - 59s 1s/step - loss: 0.6932 - accuracy: 0.5016 - val_loss: 0.6932 -
    val_accuracy: 0.4972
Epoch 9/10

```

```
50/50 [=====] - 58s 1s/step - loss: 0.6932 - accuracy: 0.4981 - val_loss: 0.6931 -
  val_accuracy: 0.5031
Epoch 10/10
50/50 [=====] - 65s 1s/step - loss: 0.6931 - accuracy: 0.5058 - val_loss: 0.6932 -
  val_accuracy: 0.4988
```

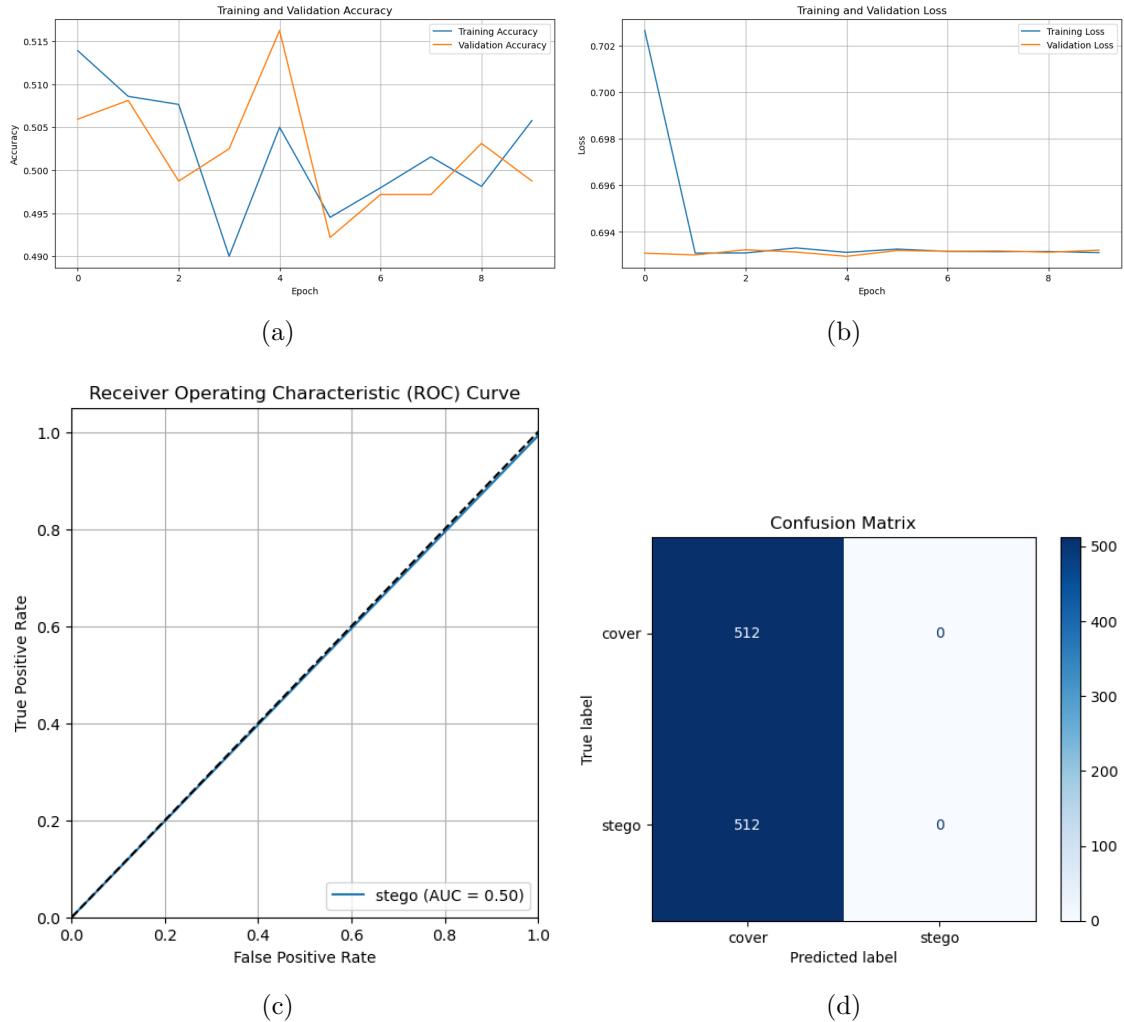


Figure 5.2: Grafici modello secondo esperimento

Così come nel primo esperimento il modello ha avuto problemi nel convergere bloccandosi nello stesso punto di stallo con loss 0.69 ed accuracy del 50%.

Inoltre anche qui i grafici mostrano in pieno l'inefficacia di tale modello, mostrandoci, nella matrice di confusione, come il modello predica sempre la stessa classe,

avendo così un f1 macro score pessimo del 33%.

Table 5.4: Classification Report secondo esperimento

Class	Precision	Recall	F1-Score	Support
cover	0.50	1.00	0.67	512
stego	0.00	0.00	0.00	512
Accuracy		0.50		
Macro avg	0.25	0.50	0.33	1024
Weighted avg	0.25	0.50	0.33	1024

5.3 Terzo esperimento

Table 5.5: Modello terzo esperimento

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 62, 62, 128)	3584
average_pooling2d_2 (AveragePooling2D)	(None, 31, 31, 128)	0
conv2d_8 (Conv2D)	(None, 29, 29, 64)	73792
average_pooling2d_3 (AveragePooling2D)	(None, 14, 14, 64)	0
batch_normalization (BatchNormalization)	(None, 14, 14, 64)	256
flatten_5 (Flatten)	(None, 12544)	0
dense_14 (Dense)	(None, 64)	802880
dense_15 (Dense)	(None, 2)	130
Total params		880,642
Trainable params		880,514
Non-trainable params		128

In questo esperimento ho deciso di aggiungere un layer di Batch normalization per permettere al modello di riuscire a convergere meglio.

Output epochs:

```
{Epoch 1/10
50/50 [=====] - 62s 1s/step - loss: 0.9681 - accuracy: 0.5133 - val_loss: 0.6950 -
    val_accuracy: 0.4969
Epoch 2/10
50/50 [=====] - 60s 1s/step - loss: 0.7087 - accuracy: 0.4995 - val_loss: 0.6941 -
    val_accuracy: 0.4850
Epoch 3/10
50/50 [=====] - 59s 1s/step - loss: 0.7006 - accuracy: 0.4998 - val_loss: 0.6933 -
    val_accuracy: 0.4919
Epoch 4/10
50/50 [=====] - 60s 1s/step - loss: 0.6956 - accuracy: 0.5056 - val_loss: 0.6939 -
    val_accuracy: 0.4953
Epoch 5/10
50/50 [=====] - 57s 1s/step - loss: 0.6933 - accuracy: 0.4934 - val_loss: 0.6931 -
    val_accuracy: 0.4981
Epoch 6/10
50/50 [=====] - 59s 1s/step - loss: 0.6933 - accuracy: 0.5042 - val_loss: 0.6934 -
    val_accuracy: 0.4884
Epoch 7/10
50/50 [=====] - 58s 1s/step - loss: 0.6931 - accuracy: 0.5070 - val_loss: 0.6934 -
    val_accuracy: 0.4881
Epoch 8/10
50/50 [=====] - 58s 1s/step - loss: 0.6931 - accuracy: 0.5047 - val_loss: 0.6932 -
    val_accuracy: 0.5000
Epoch 9/10
50/50 [=====] - 57s 1s/step - loss: 0.6931 - accuracy: 0.5067 - val_loss: 0.6932 -
    val_accuracy: 0.4988
```

```
Epoch 10/10
50/50 [=====] - 53s 1s/step - loss: 0.6932 - accuracy: 0.4991 - val_loss: 0.6930 -
val_accuracy: 0.5122}
```

Così come nei precedenti due esperimenti non sono riuscito ad ottenere un miglioramento determinante, ottenendo sempre gli stessi valori per le metriche analizzate.

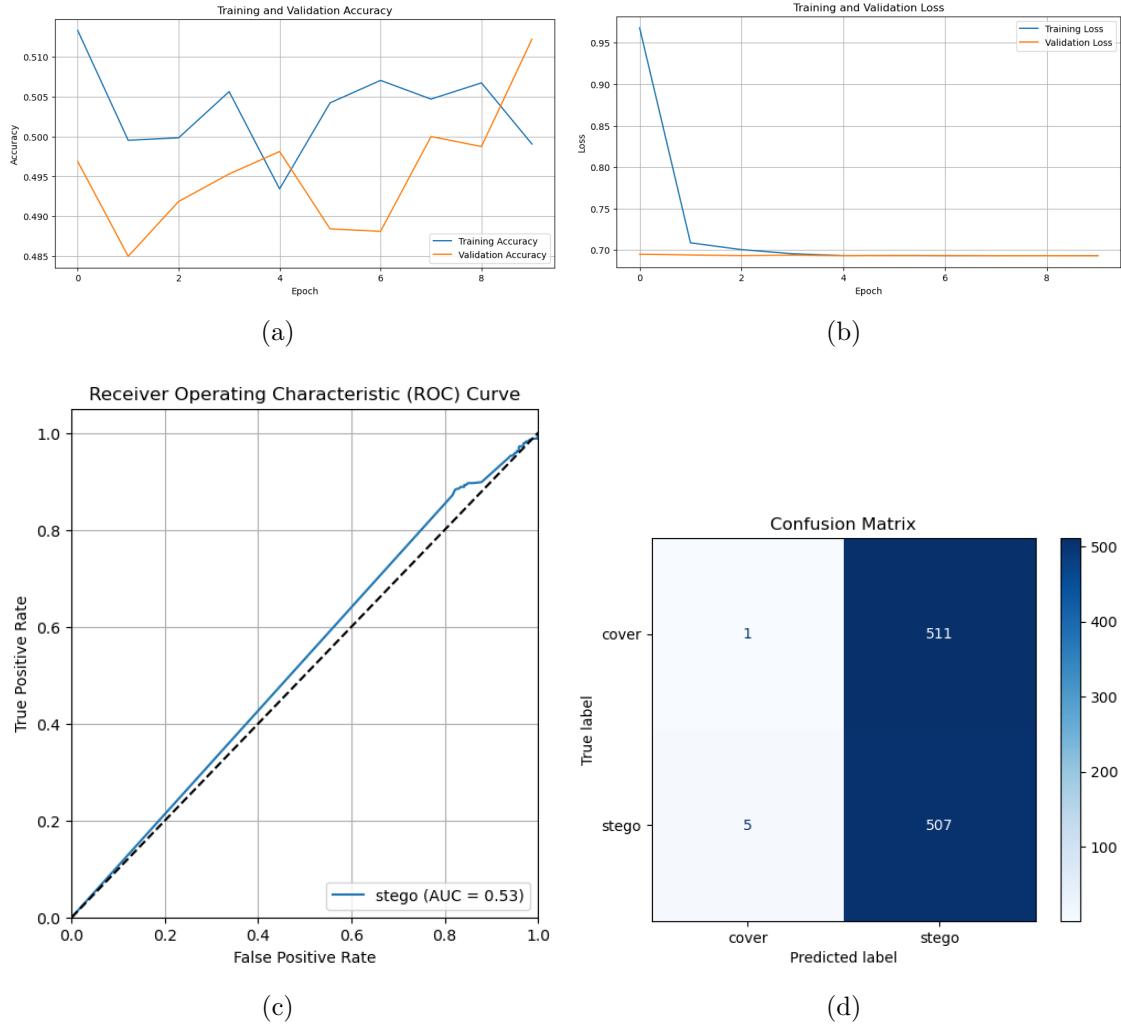


Figure 5.3: Grafici modello terzo esperimento

Table 5.6: Classification Report terzo esperimento

Class	Precision	Recall	F1-Score	Support
cover	0.17	0.00	0.00	512
stego	0.50	0.99	0.66	512
Accuracy		0.50		
Macro avg	0.33	0.50	0.33	1024
Weighted avg	0.33	0.50	0.33	1024

5.4 Quarto esperimento

Table 5.7: Modello quarto esperimento

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 62, 62, 128)	3584
average_pooling2d_4 (AveragePooling2D)	(None, 31, 31, 128)	0
conv2d_10 (Conv2D)	(None, 29, 29, 64)	73792
average_pooling2d_5 (AveragePooling2D)	(None, 14, 14, 64)	0
conv2d_11 (Conv2D)	(None, 12, 12, 64)	36928
average_pooling2d_6 (AveragePooling2D)	(None, 6, 6, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 6, 6, 64)	256
flatten_6 (Flatten)	(None, 2304)	0
dense_16 (Dense)	(None, 64)	147520
dense_17 (Dense)	(None, 2)	130
Total params		262,210
Trainable params		262,082
Non-trainable params		128

Qua ho aggiunto ulteriori layer di convoluzione e pooling sempre allo scopo di rendere il modello più elaborato.

Output epochs:

```
{
Epoch 1/10
50/50 [=====] - 61s 1s/step - loss: 0.7430 - accuracy: 0.5044 - val_loss: 0.6931 -
    val_accuracy: 0.4991
Epoch 2/10
50/50 [=====] - 61s 1s/step - loss: 0.7081 - accuracy: 0.5017 - val_loss: 0.6947 -
    val_accuracy: 0.5044
Epoch 3/10
50/50 [=====] - 62s 1s/step - loss: 0.7068 - accuracy: 0.4980 - val_loss: 0.6937 -
    val_accuracy: 0.4966
Epoch 4/10
50/50 [=====] - 69s 1s/step - loss: 0.7046 - accuracy: 0.4942 - val_loss: 0.6946 -
    val_accuracy: 0.5041
Epoch 5/10
50/50 [=====] - 63s 1s/step - loss: 0.7019 - accuracy: 0.5084 - val_loss: 0.6964 -
    val_accuracy: 0.5006
Epoch 6/10
50/50 [=====] - 64s 1s/step - loss: 0.7031 - accuracy: 0.5023 - val_loss: 0.6980 -
    val_accuracy: 0.4947
Epoch 7/10
50/50 [=====] - 65s 1s/step - loss: 0.7004 - accuracy: 0.5033 - val_loss: 0.6948 -
    val_accuracy: 0.5081
Epoch 8/10
```

```

50/50 [=====] - 67s 1s/step - loss: 0.7022 - accuracy: 0.5013 - val_loss: 0.7083 -
    val_accuracy: 0.4881
Epoch 9/10
50/50 [=====] - 65s 1s/step - loss: 0.7008 - accuracy: 0.5136 - val_loss: 0.6939 -
    val_accuracy: 0.5081
Epoch 10/10
50/50 [=====] - 64s 1s/step - loss: 0.7006 - accuracy: 0.5078 - val_loss: 0.6943 -
    val_accuracy: 0.5013
}

```

Guardando i risultati ottenuti nel corso delle epochs sembra che il risultato non sia cambiato tanto, tuttavia, attenzionando le altre metriche e matrice di confusione esce fuori un risultato decisamente migliore, determinato dal fatto che adesso il modello se la cava meglio a predire entrambe le classi, avendo dunque un f1 score migliore.

Table 5.8: Classification Report quarto esperimento

Class	Precision	Recall	F1-Score	Support
cover	0.45	0.20	0.28	512
stego	0.48	0.75	0.59	512
Accuracy		0.48		
Macro avg	0.47	0.48	0.43	1024
Weighted avg	0.47	0.48	0.43	1024

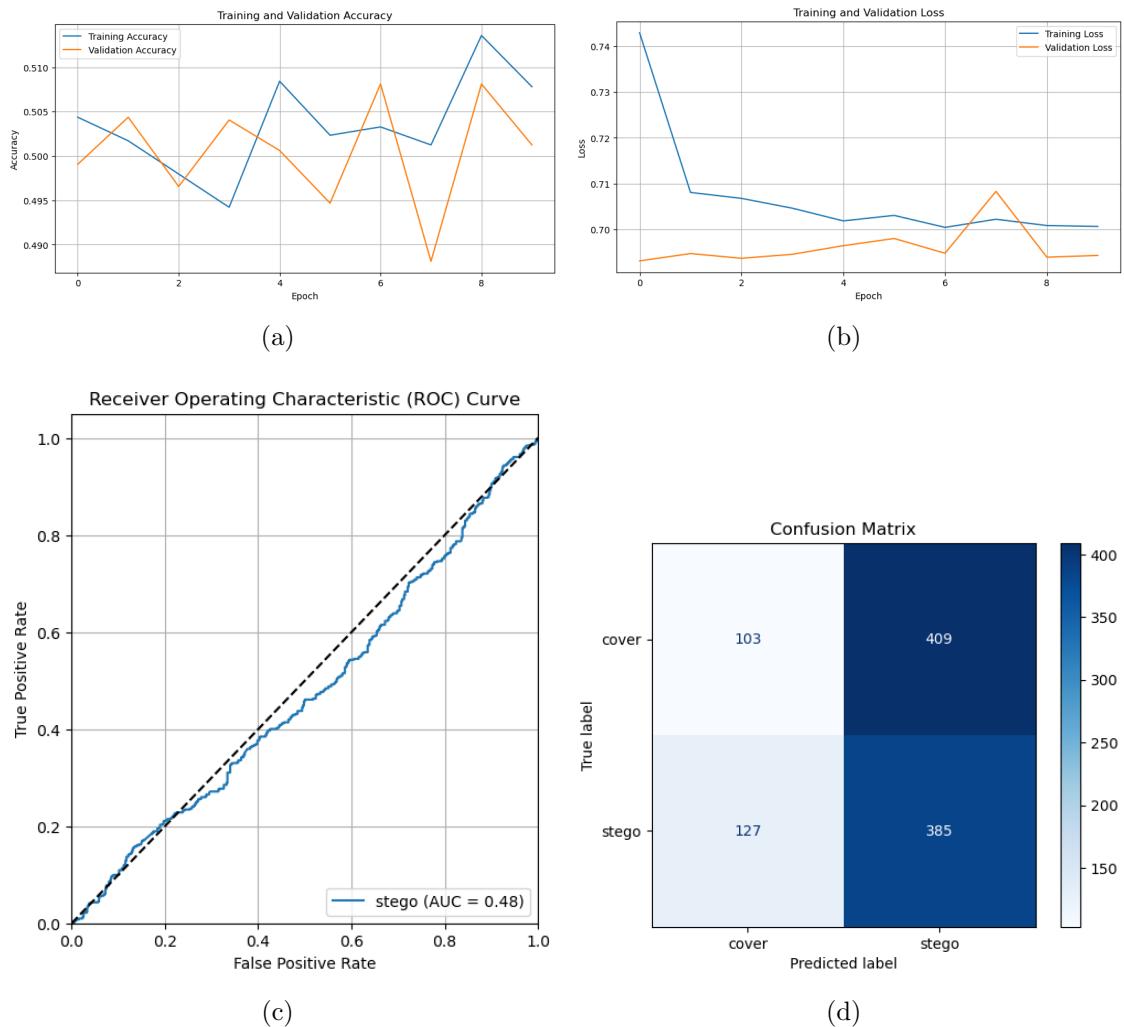


Figure 5.4: Grafici modello quarto esperimento

5.5 Quinto esperimento

Table 5.9: Modello quinto esperimento

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 62, 62, 128)	3584
average_pooling2d_8 (AveragePooling2D)	(None, 31, 31, 128)	0
batch_normalization (BatchNormalization)	(None, 31, 31, 128)	512
conv2d_9 (Conv2D)	(None, 29, 29, 64)	73792
average_pooling2d_9 (AveragePooling2D)	(None, 14, 14, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 14, 14, 64)	256
conv2d_10 (Conv2D)	(None, 12, 12, 64)	36928
average_pooling2d_10 (AveragePooling2D)	(None, 6, 6, 64)	0
batch_normalization_2 (BatchNormalization)	(None, 6, 6, 64)	256
flatten_2 (Flatten)	(None, 2304)	0
dense_7 (Dense)	(None, 64)	147520
dense_8 (Dense)	(None, 32)	2080
dense_9 (Dense)	(None, 2)	66
Total params		264,994
Trainable params		264,482
Non-trainable params		512

Visto il miglioramento ottenuto nello scorso risultato ho deciso di continuare sulla stessa strada aumentando il numero di batch normalization ed introducendo un nuovo layer con funzione di attivazione Relu.

Output epochs:

```
{
Epoch 1/10
50/50 [=====] - 93s 2s/step - loss: 0.7377 - accuracy: 0.5052 - val_loss: 0.6981 -
    val_accuracy: 0.5038
Epoch 2/10
50/50 [=====] - 84s 2s/step - loss: 0.7266 - accuracy: 0.4933 - val_loss: 0.6963 -
    val_accuracy: 0.4988
Epoch 3/10
50/50 [=====] - 86s 2s/step - loss: 0.7175 - accuracy: 0.5038 - val_loss: 0.6950 -
    val_accuracy: 0.5047
Epoch 4/10
50/50 [=====] - 85s 2s/step - loss: 0.7097 - accuracy: 0.5000 - val_loss: 0.7022 -
    val_accuracy: 0.4972
Epoch 5/10
50/50 [=====] - 86s 2s/step - loss: 0.7058 - accuracy: 0.5005 - val_loss: 0.6964 -
    val_accuracy: 0.4928
Epoch 6/10
```

```

50/50 [=====] - 87s 2s/step - loss: 0.7096 - accuracy: 0.5002 - val_loss: 0.6973 -
    val_accuracy: 0.4947
Epoch 7/10
50/50 [=====] - 86s 2s/step - loss: 0.7035 - accuracy: 0.4986 - val_loss: 0.6988 -
    val_accuracy: 0.4966
Epoch 8/10
50/50 [=====] - 86s 2s/step - loss: 0.6996 - accuracy: 0.5045 - val_loss: 0.6960 -
    val_accuracy: 0.4984
Epoch 9/10
50/50 [=====] - 86s 2s/step - loss: 0.7000 - accuracy: 0.4973 - val_loss: 0.6961 -
    val_accuracy: 0.4941
Epoch 10/10
50/50 [=====] - 85s 2s/step - loss: 0.6971 - accuracy: 0.4992 - val_loss: 0.6955 -
    val_accuracy: 0.5031
}

```

Anche qui, come nel precedente, sono riuscito a migliorare leggermente l'f1 score.

Table 5.10: Classification Report quinto esperimento

Class	Precision	Recall	F1-Score	Support
cover	0.50	0.31	0.38	512
stego	0.50	0.70	0.58	512
Accuracy		0.50		
Macro avg	0.50	0.50	0.48	1024
Weighted avg	0.50	0.50	0.48	1024

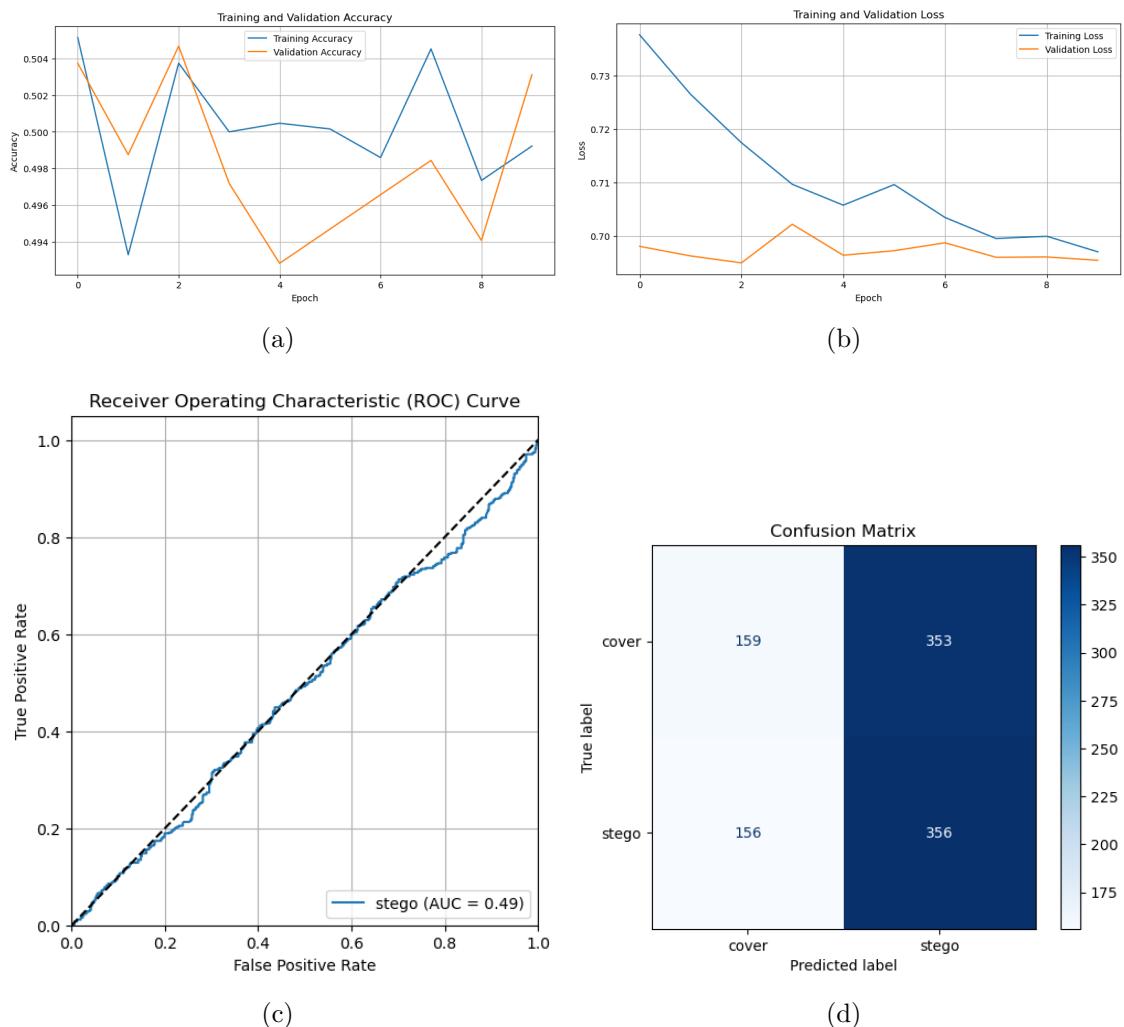


Figure 5.5: Grafici modello quinto esperimento

5.6 Sesto esperimento

Table 5.11: Modello sesto esperimento

Layer (type)	Output Shape	Param #
conv2d_27 (Conv2D)	(None, 62, 62, 128)	3584
average_pooling2d_27 (AveragePooling2D)	(None, 31, 31, 128)	0
batch_normalization_8 (BatchNormalization)	(None, 31, 31, 128)	512
conv2d_28 (Conv2D)	(None, 29, 29, 64)	73792
average_pooling2d_28 (AveragePooling2D)	(None, 14, 14, 64)	0
batch_normalization_9 (BatchNormalization)	(None, 14, 14, 64)	256
conv2d_29 (Conv2D)	(None, 12, 12, 64)	36928
average_pooling2d_29 (AveragePooling2D)	(None, 6, 6, 64)	0
batch_normalization_10 (BatchNormalization)	(None, 6, 6, 64)	256
conv2d_30 (Conv2D)	(None, 4, 4, 64)	36928
average_pooling2d_30 (AveragePooling2D)	(None, 2, 2, 64)	0
batch_normalization_11 (BatchNormalization)	(None, 2, 2, 64)	256
flatten_7 (Flatten)	(None, 256)	0
dense_24 (Dense)	(None, 64)	16448
dense_25 (Dense)	(None, 64)	4160
dense_26 (Dense)	(None, 32)	2080
dense_27 (Dense)	(None, 2)	66
Total params		175,266
Trainable params		174,626
Non-trainable params		640

Visto il continuo successo ho ulteriormente aggiunto altri layer densi e di convoluzione seguiti da pooling.

Output epochs:

```
{
Epoch 1/10
50/50 [=====] - 107s 2s/step - loss: 0.7122 - accuracy: 0.5045 - val_loss: 0.6933 -
  val_accuracy: 0.5025
Epoch 2/10
50/50 [=====] - 100s 2s/step - loss: 0.7018 - accuracy: 0.4963 - val_loss: 0.6944 -
  val_accuracy: 0.4850
Epoch 3/10
50/50 [=====] - 101s 2s/step - loss: 0.6991 - accuracy: 0.5000 - val_loss: 0.6953 -
  val_accuracy: 0.4897
Epoch 4/10
50/50 [=====] - 85s 2s/step - loss: 0.6973 - accuracy: 0.5005 - val_loss: 0.6986 -
  val_accuracy: 0.4884
```

```

Epoch 5/10
50/50 [=====] - 85s 2s/step - loss: 0.6981 - accuracy: 0.4975 - val_loss: 0.6951 -
    val_accuracy: 0.5038
Epoch 6/10
50/50 [=====] - 85s 2s/step - loss: 0.6962 - accuracy: 0.4995 - val_loss: 0.6938 -
    val_accuracy: 0.5066
Epoch 7/10
50/50 [=====] - 86s 2s/step - loss: 0.6955 - accuracy: 0.4991 - val_loss: 0.6949 -
    val_accuracy: 0.5066
Epoch 8/10
50/50 [=====] - 85s 2s/step - loss: 0.6943 - accuracy: 0.5147 - val_loss: 0.6948 -
    val_accuracy: 0.4984
Epoch 9/10
50/50 [=====] - 85s 2s/step - loss: 0.6945 - accuracy: 0.5147 - val_loss: 0.6943 -
    val_accuracy: 0.5091
Epoch 10/10
50/50 [=====] - 86s 2s/step - loss: 0.6963 - accuracy: 0.4945 - val_loss: 0.6950 -
    val_accuracy: 0.4963
}

```

Anche qui il modello è continuato leggermente a migliorare dal punto del f1 score riuscendo a raggiungere il 48%.

Table 5.12: Classification Report sesto esperimento

Class	Precision	Recall	F1-Score	Support
cover	0.50	0.81	0.62	512
stego	0.49	0.18	0.27	512
Accuracy			0.50	1024
Macro Avg	0.50	0.50	0.44	1024
Weighted Avg	0.50	0.50	0.44	1024

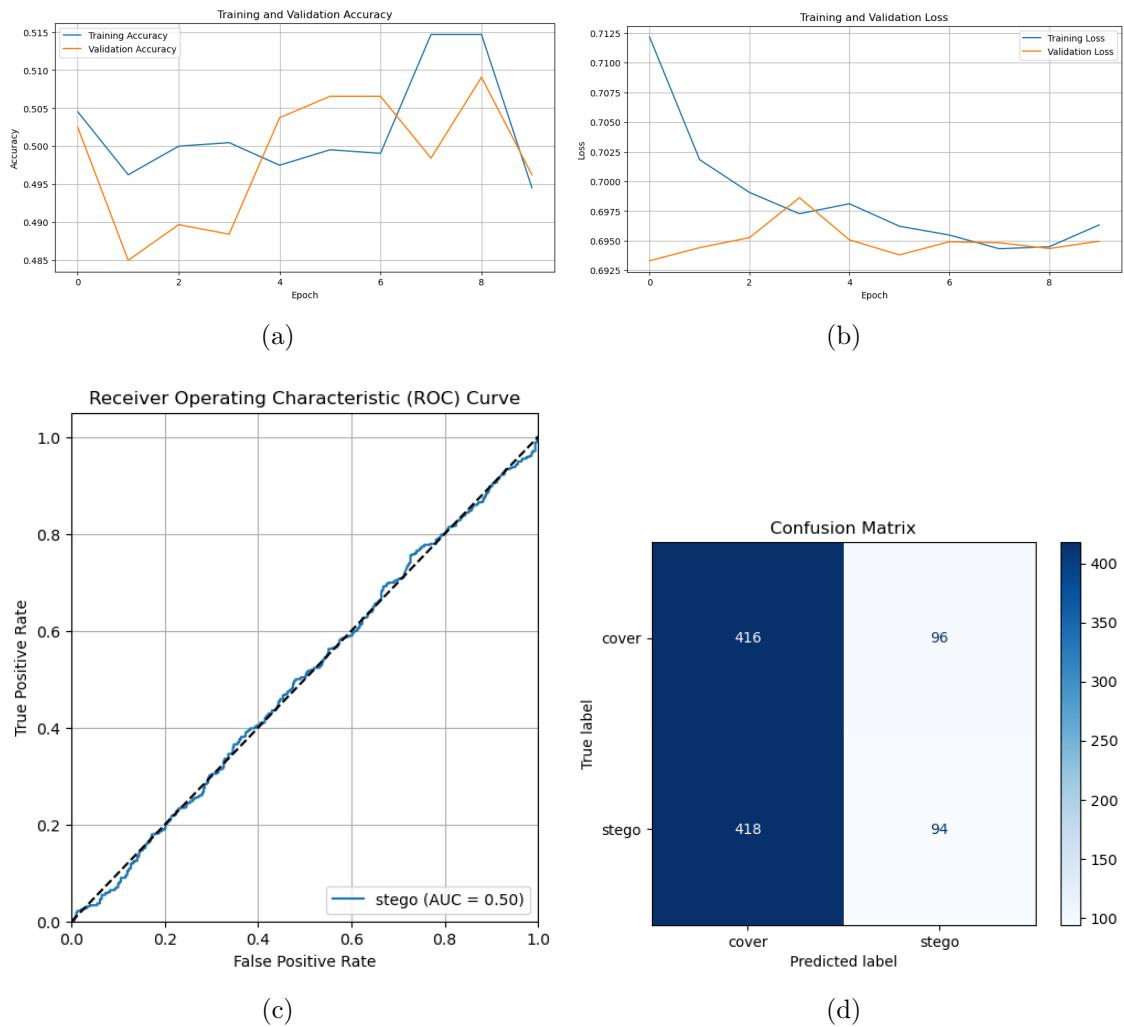


Figure 5.6: Grafici modello sesto esperimento

5.7 Settimo esperimento

Table 5.13: Modello settimo esperimento

Layer (type)	Output Shape	Param #
conv2d_31 (Conv2D)	(None, 62, 62, 256)	7168
average_pooling2d_31 (AveragePooling2D)	(None, 31, 31, 256)	0
batch_normalization_12 (BatchNormalization)	(None, 31, 31, 256)	1024
conv2d_32 (Conv2D)	(None, 29, 29, 128)	295040
average_pooling2d_32 (AveragePooling2D)	(None, 14, 14, 128)	0
batch_normalization_13 (BatchNormalization)	(None, 14, 14, 128)	512
conv2d_33 (Conv2D)	(None, 12, 12, 64)	73792
average_pooling2d_33 (AveragePooling2D)	(None, 6, 6, 64)	0
batch_normalization_14 (BatchNormalization)	(None, 6, 6, 64)	256
conv2d_34 (Conv2D)	(None, 4, 4, 64)	36928
average_pooling2d_34 (AveragePooling2D)	(None, 2, 2, 64)	0
batch_normalization_15 (BatchNormalization)	(None, 2, 2, 64)	256
flatten_8 (Flatten)	(None, 256)	0
dense_28 (Dense)	(None, 128)	32896
dense_29 (Dense)	(None, 64)	8256
dense_30 (Dense)	(None, 32)	2080
dense_31 (Dense)	(None, 2)	66
Total params		458,274
Trainable params		457,250
Non-trainable params		1,024

In questo esperimento, piuttosto che continuare ad aggiungere layer, ho aumentato il numero di kernel e neuroni presenti all'interno della rete al fine di garantire una maggior precisione al modello

Output epochs:

```
{
Epoch 1/10
50/50 [=====] - 167s 3s/step - loss: 0.7128 - accuracy: 0.5045 - val_loss: 0.6953 -
    val_accuracy: 0.4963
Epoch 2/10
50/50 [=====] - 155s 3s/step - loss: 0.6993 - accuracy: 0.5033 - val_loss: 0.6952 -
    val_accuracy: 0.5088
Epoch 3/10
50/50 [=====] - 139s 3s/step - loss: 0.6956 - accuracy: 0.5011 - val_loss: 0.6923 -
    val_accuracy: 0.5222
```

```

Epoch 4/10
50/50 [=====] - 157s 3s/step - loss: 0.6962 - accuracy: 0.5028 - val_loss: 0.6947 -
    val_accuracy: 0.4919
Epoch 5/10
50/50 [=====] - 163s 3s/step - loss: 0.6958 - accuracy: 0.4958 - val_loss: 0.6938 -
    val_accuracy: 0.4928
Epoch 6/10
50/50 [=====] - 143s 3s/step - loss: 0.6952 - accuracy: 0.5025 - val_loss: 0.6931 -
    val_accuracy: 0.5100
Epoch 7/10
50/50 [=====] - 140s 3s/step - loss: 0.6947 - accuracy: 0.5044 - val_loss: 0.6948 -
    val_accuracy: 0.4978
Epoch 8/10
50/50 [=====] - 139s 3s/step - loss: 0.6938 - accuracy: 0.4973 - val_loss: 0.6947 -
    val_accuracy: 0.4994
Epoch 9/10
50/50 [=====] - 158s 3s/step - loss: 0.6962 - accuracy: 0.4917 - val_loss: 0.6927 -
    val_accuracy: 0.5072
Epoch 10/10
50/50 [=====] - 156s 3s/step - loss: 0.6963 - accuracy: 0.5141 - val_loss: 0.6952 -
    val_accuracy: 0.4969
}

```

Questa scelta ha portato ad un ulteriore miglioramento del F1 macro score riuscendo a toccare ben il 50%, un risultato già promettente rispetto ai primi.

Table 5.14: Classification Report settimo esperimento

Class	Precision	Recall	F1-Score	Support
cover	0.51	0.62	0.56	512
stego	0.51	0.39	0.45	512
Accuracy			0.51	1024
Macro Avg	0.51	0.51	0.50	1024
Weighted Avg	0.51	0.51	0.50	1024

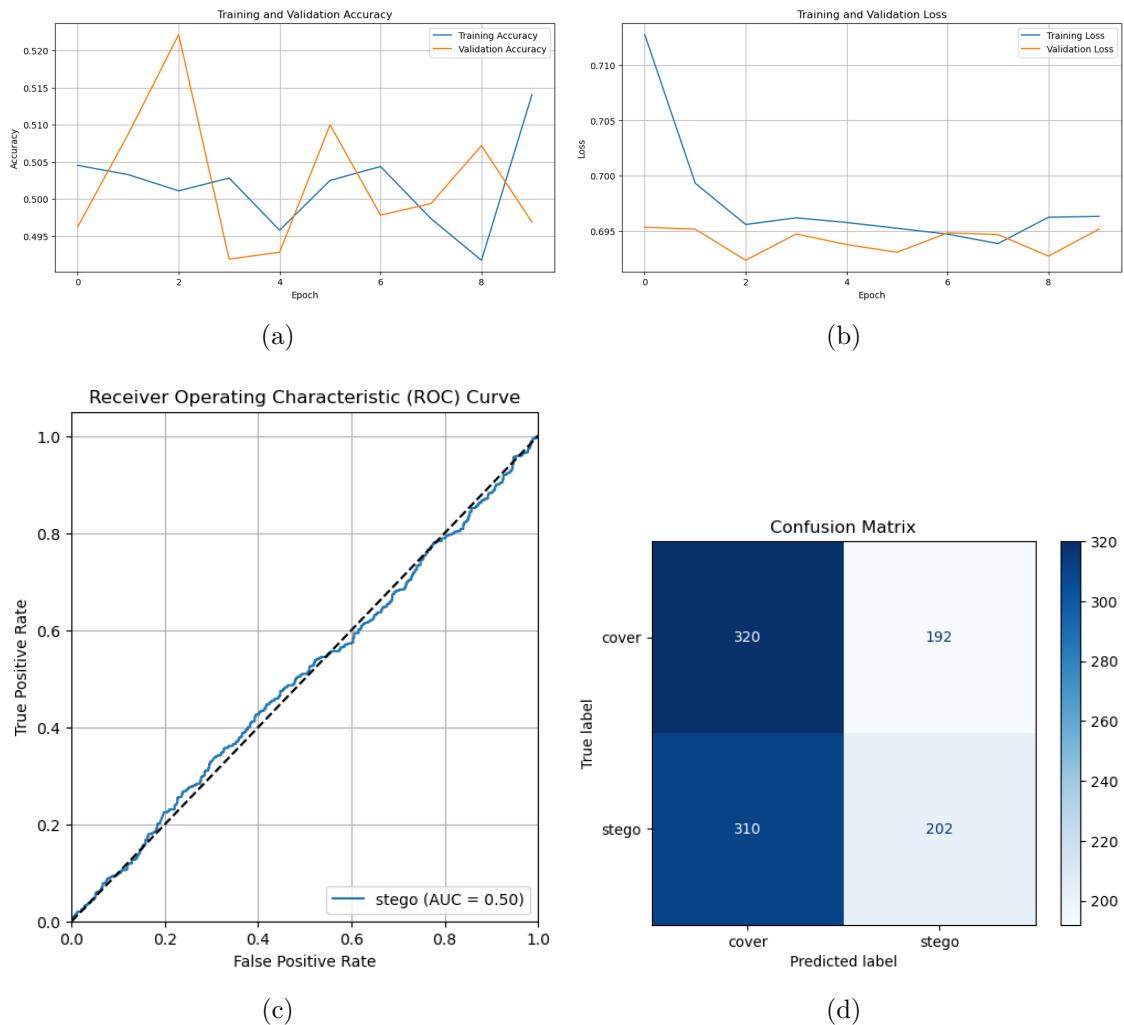


Figure 5.7: Grafici modello settimo esperimento

5.8 Ottavo esperimento

Table 5.15: Model ottavo esperimento

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 1280)	4049571
batch_normalization_7 (BatchNormalization)	(None, 1280)	5120
dense_23 (Dense)	(None, 2)	2562
Total params		4,057,253
Trainable params		4,012,670
Non-trainable params		44,583

Per concludere lo studio ho deciso di applicare un modello pre-addestrato, ovvero EfficientNetB0.

EfficientNetB0 (ref. [11]) è un'architettura leggera e altamente efficace di reti neurali convoluzionali, che ha guadagnato popolarità grazie al suo uso efficiente delle risorse e alle prestazioni competitive. Questo mi ha spinto a farci una prova adattandolo al problema proposto. Per fare ciò ho semplicemente aggiunto un Batch normalization ed un layer finale con 2 neuroni e la funzione di attivazione SoftMax.

Output epochs:

```
{
Epoch 1/10
50/50 [=====] - 155s 3s/step - loss: 0.9949 - accuracy: 0.5056 - val_loss: 0.7336 -
    val_accuracy: 0.5059
Epoch 2/10
50/50 [=====] - 124s 2s/step - loss: 0.8219 - accuracy: 0.5075 - val_loss: 0.6969 -
    val_accuracy: 0.4778
Epoch 3/10
50/50 [=====] - 113s 2s/step - loss: 0.8254 - accuracy: 0.5138 - val_loss: 0.7148 -
    val_accuracy: 0.4916
Epoch 4/10
50/50 [=====] - 119s 2s/step - loss: 0.7620 - accuracy: 0.5028 - val_loss: 0.6931 -
    val_accuracy: 0.5025
Epoch 5/10
50/50 [=====] - 114s 2s/step - loss: 0.7352 - accuracy: 0.5089 - val_loss: 0.6943 -
    val_accuracy: 0.4963
Epoch 6/10
50/50 [=====] - 123s 2s/step - loss: 0.7331 - accuracy: 0.5009 - val_loss: 0.8899 -
    val_accuracy: 0.5034
Epoch 7/10
50/50 [=====] - 124s 2s/step - loss: 0.7210 - accuracy: 0.4942 - val_loss: 0.7074 -
    val_accuracy: 0.4981
Epoch 8/10
```

```

50/50 [=====] - 124s 2s/step - loss: 0.7223 - accuracy: 0.5048 - val_loss: 0.6935 -
    val_accuracy: 0.4997
Epoch 9/10
50/50 [=====] - 113s 2s/step - loss: 0.7118 - accuracy: 0.4872 - val_loss: 0.6931 -
    val_accuracy: 0.5013
Epoch 10/10
50/50 [=====] - 126s 3s/step - loss: 0.7255 - accuracy: 0.4923 - val_loss: 48.6781 -
    val_accuracy: 0.5084
}

```

Purtroppo però, come osservabile dai grafici e metriche, non sono riuscito ad ottenere un risultato migliore dei modelli precedentemente proposti.

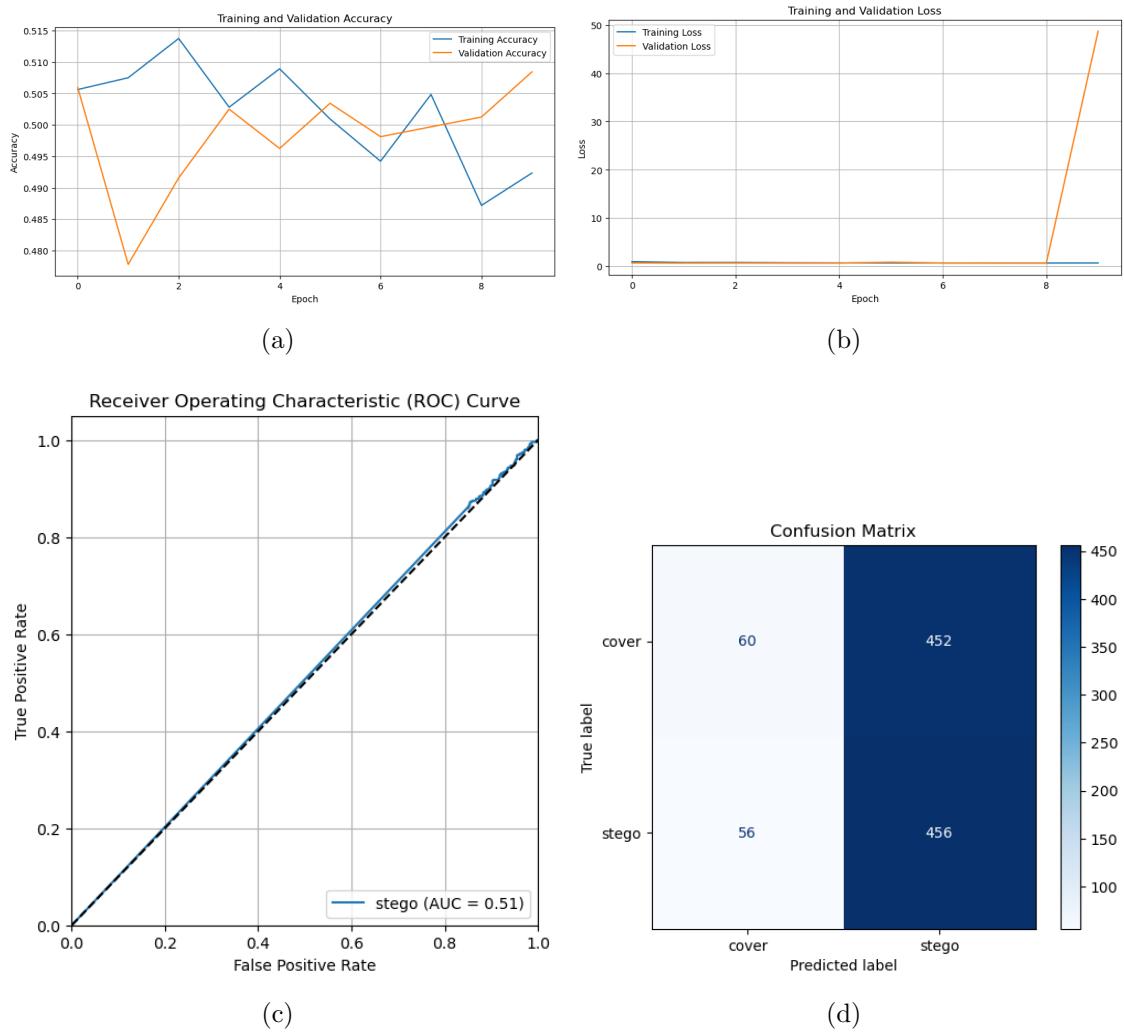


Figure 5.8: Grafici modello ottavo esperimento

Table 5.16: Classification Report ottavo esperimento

Class	Precision	Recall	F1-Score	Support
cover	0.52	0.12	0.19	512
stego	0.50	0.89	0.64	512
Accuracy		0.50		
Macro avg	0.51	0.50	0.42	1024
Weighted avg	0.51	0.50	0.42	1024

Chapter 6

Conclusioni

In questo mio lavoro di ricerca, ho proposto un approccio basato su deep learning per la classificazione di immagini con steganografia.

Il primo passo è stato esplorare le immagini stego e cover al fine di ottenere una descrizione statistica quanto più dettagliata possibile del problema proposto. Questa fase iniziale ha messo in luce le sottili differenze tra le due varianti, confermando la sfida nell'effettuare steganalisi.

Successivamente, ho optato per la creazione ed applicazione di modelli di deep learning basati su Convolutional Neural Network (CNN) di complessità variabile, dato l'insuccesso nel trovare una funzione rappresentativa ottimale per il set di immagini in esame.

Come previsto, i risultati ottenuti durante gli esperimenti hanno mostrato un'accuracy media di circa il 50%, con un punteggio f1 macro che è gradualmente migliorato nel corso delle sperimentazioni, ma senza raggiungere valori altamente promettenti, partendo da un valore medio del 36% e arrivando al 50

Per il futuro, intendo esplorare ulteriori tecniche di machine learning e deep learning, con nuovi possibili approcci per migliorare le prestazioni del classificatore. Sono comunque fiducioso che il mio lavoro possa contribuire allo sviluppo di soluzioni efficaci per la rilevazione e l'analisi della steganografia UERD nelle immagini JPEG.

Bibliography

- [1] L. Guo, J. Ni, W. Su, C. Tang, and Y.-Q. Shi. “Using Statistical Image Model for JPEG Steganography: Uniform Embedding Revisited”. In: *IEEE Transactions on Information Forensics and Security* 10.12 (2015), pp. 2669–2680. DOI: [10.1109/TIFS.2015.2473815](https://doi.org/10.1109/TIFS.2015.2473815).
- [2] Z. Yang, K. Wang, S. Ma, Y. Huang, X. Kang, and X. Zhao. “IStego100K: Large-scale Image Steganalysis Dataset”. In: *International Workshop on Digital Watermarking*. Springer. 2019.
- [3] V. Holub, J. Fridrich, and T. Denemark. “Universal Distortion Function for Steganography in an Arbitrary Domain”. In: *EURASIP Journal on Information Security* 1 (Dec. 2014). DOI: [10.1186/1687-417X-2014-1](https://doi.org/10.1186/1687-417X-2014-1).
- [4] J. Fridrich, T. Pevný, and J. Kodovský. “Statistically Undetectable Jpeg Steganography: Dead Ends Challenges, and Opportunities”. In: *Proceedings of the 9th Workshop on Multimedia & Security*. MM&Sec '07. Dallas, Texas, USA: Association for Computing Machinery, 2007, 3–14. ISBN: 9781595938572. DOI: [10.1145/1288869.1288872](https://doi.org/10.1145/1288869.1288872). URL: <https://doi.org/10.1145/1288869.1288872>.
- [5] Y. Wang, W. Zhang, W. Li, X. Yu, and N. Yu. “Non-Additive Cost Functions for Color Image Steganography Based on Inter-Channel Correlations and Differences”. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 2081–2095. DOI: [10.1109/TIFS.2019.2956590](https://doi.org/10.1109/TIFS.2019.2956590).
- [6] M. H. Menori and R. Munir. “Blind steganalysis for digital images using support vector machine method”. In: *2016 International Symposium on Electronics and Smart Devices (ISESD)*. 2016, pp. 132–136. DOI: [10.1109/ISESD.2016.7886706](https://doi.org/10.1109/ISESD.2016.7886706).

- [7] T. Pevny, P. Bas, and J. Fridrich. “Steganalysis by Subtractive Pixel Adjacency Matrix”. In: *IEEE Transactions on Information Forensics and Security* 5.2 (2010), pp. 215–224. DOI: [10.1109/TIFS.2010.2045842](https://doi.org/10.1109/TIFS.2010.2045842).
- [8] T.-S. Reinel, R.-P. Raúl, and I. Gustavo. “Deep Learning Applied to Steganalysis of Digital Images: A Systematic Review”. In: *IEEE Access* 7 (2019), pp. 68970–68990. DOI: [10.1109/ACCESS.2019.2918086](https://doi.org/10.1109/ACCESS.2019.2918086).
- [9] Y.-H. Tang, L.-H. Jiang, H.-Q. He, and W.-Y. Dong. “A review on deep learning based image steganalysis”. In: *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2018, pp. 1764–1770. DOI: [10.1109/IAEAC.2018.8577655](https://doi.org/10.1109/IAEAC.2018.8577655).
- [10] F. Chollet et al. *Keras*. <https://keras.io>. Versione 2.3.1. 2015.
- [11] M. Tan and Q. V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019.