UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

**DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE**

**CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE**

# STUDIO SULL'UTILIZZO DI DATI SINTETICI NEL TRAINING DI MODELLI DI SEGMENTAZIONE SEMANTICA


**Relatore: Prof. Pietro Zanuttigh**

**Laureanda: Daria Zanin**

**Correlatore: Dott. Francesco Barbato**



**ANNO ACCADEMICO 2021 – 2022**

**Data di laurea 21 Settembre 2022**

# Sommario

L'elaborato presenta uno studio sull'uso di dati sintetici per addestrare modelli di segmentazione semantica basati su reti neurali. Il vantaggio nell'impiegare dati sintetici consiste nell'evitare l'etichettatura dei dati reali ma comporta la necessità di mettere a punto tecniche utili ad addestrare modelli che siano efficaci nell'inferenza sui dati reali pur essendo stati addestrati sui dati sintetici annotati. La tecnica indagata consiste in un adattamento dei dati sintetici così che appaiano più simili a quelli reali (domain adaptation).

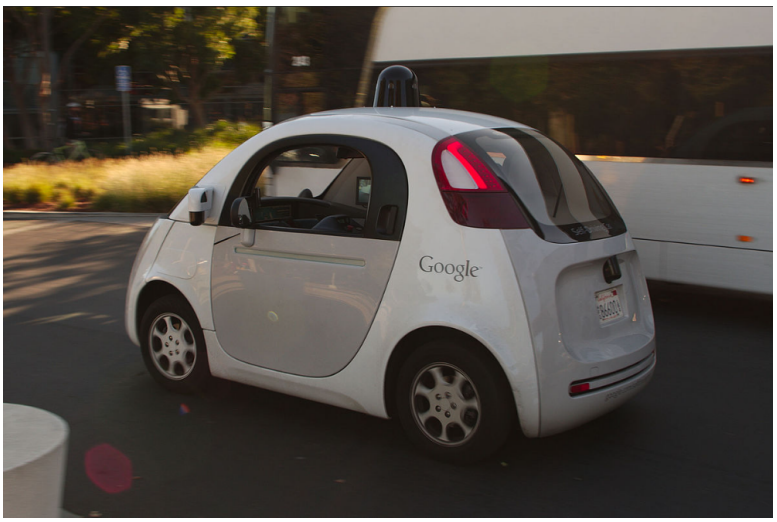
In questo elaborato viene presentata un'introduzione al problema e una rassegna delle tecniche usate, andando in particolare a investigare il metodo proposto in [1], che lavora nel dominio della frequenza. Il problema considerato si riferisce alla guida autonoma per la quale, oltre a dataset reali come ad esempio Cityscapes [2], sono disponibili anche dataset sintetici [3]. Per indagare le tecniche è stato sviluppato del codice in linguaggio Python che permette di visualizzare i dati di segmentazione con delle mappe semantiche e inoltre permette di calcolare le statistiche degli spettri delle immagini e da queste effettuare l'adattamento di dominio come proposto nella letteratura già sopracitata [1].

# Indice

# Elenco delle figure

# 1 Introduzione

La guida autonoma è una delle aree di ricerca e sviluppo attive sia nel contesto universitario che in quello delle aziende. L'obiettivo è quello di realizzare mezzi di trasporto che siano in grado di muoversi in autonomia nei diversi contesti ambientali che si possono presentare. Una componente fondamentale utile a conseguire lo scopo posto viene dalle tecniche di elaborazione delle immagini. La complessità delle diverse scene, che vengono acquisite dalle videocamere di cui vengono dotati i mezzi con guida autonoma, ha trovato nelle tecniche di *Deep Learning* (DL) [4] la tipologia di algortmi utili per assolvere il difficile problema da affrontare. Le metodiche di DL rientrano nella più vasta famiglia delle tecniche di *Machine Learning* (ML) [5], ambito che studia modelli statistico-matematici in grado di essere adattati a formulare delle predizioni che permettono di estrarre in modo automatico informazioni utili da dati di input anche articolati e complessi come può essere un'immagine. Il presente elaborato affronta nel prossimo capitolo una breve introduzione alle metodiche di ML per poi addentrarsi nello specifico nel descrivere i modelli di segmentazione di immagini basati sul DL. Vengono poi presentati in un successivo capitolo il problema specifico dell'adattamento di dominio al caso dei dataset SELMA e Cityscapes utili per la navigazione autonoma. Nella Figura 1 si vede ad esempio Waymo, l'auto a guida autonoma di Google.




(a)

Figura 1: Waymo, l'auto a guida autonoma di Google

Il presente elaborato riporta nel capitolo 2 una breve introduzione al ML e poi al DL applicato per la segmentazione semantica delle immagini, per poi proseguire

con il capitolo 3 dove viene nello specifico descritto il tema argomento della tesi relativo all'adattamento di dominio usato nell'addestramento di modelli utili per la guida autonoma, per poi concludere con il capitolo 4 dove si traggono le conclusioni sul lavoro svolto.

# 2 Breve introduzione al ML e al DL

In questa sezione viene presentata una breve introduzione alle tecniche di machine learning [5] e di deep learning [4] e in particolare alle reti per la segmentazione semantica usate nel contesto della computer vision. Queste reti sono evoluzioni dell'architettura di rete neurale che verrà altresì descritta [6]. In Figura 2 viene riportato

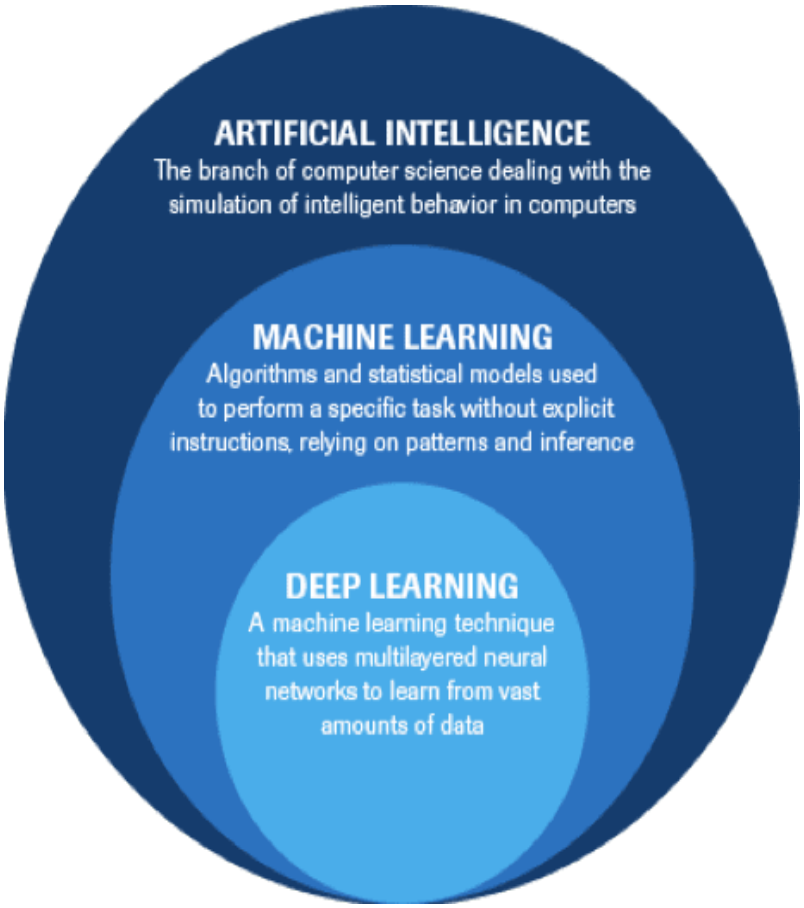



Figura 2: Relazione insiemistica tra metodiche di intelligenza artificiale.

un diagramma insiemistico che rappresenta le relazioni di inclusione delle diverse branche dell'intelligenza artificiale. In particolare, il DL è la disciplina che studia delle specifiche tecniche di ML che sono in grado di sfruttare, per l'apprendimento, una vasta quantità di dati. Lo studio e l'applicazione di queste tecniche, che hanno preso e stanno prendendo sempre più spazio a partire da tempi relativamente recenti, sono riusciti in parte a rispondere alla vera sfida dell'intelligenza artificiale, che è quella di provare a risolvere problemi che appaiono intuitivi ed automatici da compiere agli esseri umani ma che purtroppo risultano alquanto complessi da formalizzare al fine di farli svolgere ad una macchina. Come già detto più sopra, il DL è una tipologia di ML e quindi è

necessario dapprima illustrare i principali concetti e la terminologia del ML. Il ML, detto altresì apprendimento automatico, è una branca dell' *Artificial Intelligence* (AI) che include un insieme di metodi, sviluppati a partire dagli ultimi decenni del XX secolo da varie comunità scientifiche, sotto diversi nomi quali: statistica computazionale, riconoscimento di pattern, reti neurali artificiali, *data mining*, algoritmi adattivi, ecc. Il ML utilizza metodi statistici per migliorare progressivamente la performance di un sistema così che la macchina sia in grado di generalizzare sulla base della propria esperienza, cioè sia in grado di perfezionare i modelli che impiega sia per identificare relazioni per la descrizione dei dati sia per estrapolare previsioni basate sui dati. In questo contesto, per generalizzazione si intende l'abilità di una macchina di portare a termine in maniera accurata esempi o compiti nuovi, mai affrontati in precedenza, dopo aver fatto esperienza su un insieme di dati di apprendimento. Gli esempi di addestramento (in inglese chiamati *training examples*) si assume provengano da una qualche distribuzione di probabilità, generalmente sconosciuta e considerata rappresentativa dello spazio delle occorrenze del fenomeno da apprendere; la macchina ha il compito di costruire un modello probabilistico generale dello spazio delle occorrenze, in maniera tale da essere in grado di produrre previsioni sufficientemente accurate quando sottoposta a nuovi casi. I compiti dell'apprendimento automatico vengono tipicamente classificati in tre ampie categorie, a seconda della natura dei dati utilizzati per l'apprendimento o del feedback che viene reso disponibile al sistema di apprendimento. Queste categorie sono [5]:

- apprendimento supervisionato (*supervised learning*), in cui vi è una fase di addestramento durante la quale vengono forniti degli esempi nella forma di possibili input e dei rispettivi output desiderati. Nel linguaggio del ML le uscite desiderate vengono denominate etichette (*labels*) e l'operazione di predisporre per ciascun ingresso la corrispondente label è detta di etichettatura (*labeling*), si parla in tal caso di dati etichettati. L'apprendimento consiste nell'estrarre una regola generale che associ l'input all'output corretto. Una volta addestrato il sistema, il modello ottenuto viene impiegato per stimare i valori di output a partire dai dati acquisiti. Degli esempi riguardano il campo medico, in cui si può prevedere lo scatenarsi di particolari crisi sulla base dell'esperienza di passati dati biometrici, l'identificazione vocale che migliora sulla base degli ascolti di audio e l'identificazione della scrittura manuale che si perfeziona sulle osservazioni degli esempi sottoposti dall'utente;

- apprendimento non supervisionato (*unsupervised learning*), in cui si cerca di

determinare un modello allo scopo di trovare una struttura negli input forniti, senza che gli input vengano etichettati in alcun modo. Ad esempio, in biologia potrebbe essere utile per derivare tassonomie di animali e piante; un altro esempio è il *clustering*, dove un insieme di dati di input viene diviso in gruppi.

- apprendimento con rinforzo (*reinforcement learning*), in cui il modello interagisce con un ambiente dinamico nel quale cerca di raggiungere un obiettivo (per esempio guidare un veicolo), disponendo di una metrica per capire se l'obiettivo viene raggiunto o vi è un miglioramento. Un altro esempio è quello di imparare a giocare un gioco giocando contro un avversario, dove la metrica deve essere in grado di stabilire se ci si trova o meno in posizione di vantaggio. La metrica viene data scegliendo una funzione, detta *reward function*, che viene calcolata allo scopo di determinare se il comporatmento del sistema sia stato o meno vantaggio rispetto alla situazione corrente. Con i continui aggiustamenti del modello dati dalla politica di rinforzo si cerca di ottenere un sistema che attui una *policy* ottimale: cioè che dato uno stato e una serie di azioni possibili, sia in grado di indicare la prossima azione così da massimizzare la funzione di reward.

Si può effettuare una categorizzazione dei compiti dell'apprendimento automatico andando a considerare l'output desiderato del sistema di apprendimento automatico. Si distinguono pertanto le tecniche di:

- Classificazione, dove gli output sono divisi in due o più classi e il sistema di apprendimento deve produrre un modello che assegni gli input non ancora visti a una o più di queste. Il filtraggio anti-spam è un esempio di classificazione, dove gli input sono le email e le classi sono ”spam” e ”non spam”.

- Regressione, che rientra tra le metodiche di apprendimento di tipo supervisionato, ma in cui i valori delle variabili di output e il modello utilizzati assumono valori continui. Un esempio di regressione è la determinazione della quantità di olio presente in un oleodotto, avendo le misurazioni dell'attenuazione dei raggi gamma che passano attraverso il condotto. Un altro esempio è la predizione del valore del tasso di cambio di una valuta nel futuro, dati i suoi valori in tempi recenti.

Per tutte le tecniche di ML, che derivano i modelli da impiegare sulla base dei dati acquisiti, si può intuire quanto sia fondamentale la rappresentazione dei dati stessi al fine di ottenere delle buone prestazioni. A tale proposito, le tecniche più tradizionali di ML basano i modelli su delle caratteristiche (in inglese *features*) estratte dai dati di input.

Quindi, molti risultati di intelligenza artificiale sono stati ottenuti grazie ad una accurata preelaborazione dei dati al fine di estrarre le caratteristiche di interesse (*features extraction*), tali da essere particolarmente descrittive per il corretto conseguimento del compito da svolgere. Purtroppo, uno dei principali ostacoli all'ottenere dei modelli di ML che abbiano prestazioni soddisfacenti è proprio quello di individuare quali siano le caratteristiche migliori da selezionare, come vederemo una possibile soluzione a evitare questa difficoltà è offerta dalle tecniche di DL.

## 2.1 Metodi supervisionati

La metodica che si andrà a considerare è di tipo supervisionato, vengono pertanto discusse alcune caratteristiche comuni a tutti i metodi supervisionati relative alle modalità di addestramento del sistema. Innanzitutto, si assume che il sistema da addestrare debba stimare a partire da un vettore di features $\boldsymbol{x}$ con $M$ componenti un vettore di uscite $\boldsymbol{y}$ con $P$ componenti. Per l'addestramento del sistema si deve disporre di un insieme pre-etichettato di coppie $(\boldsymbol{x}(n), \boldsymbol{y}(n))$ con $n = 1, .., N$, dove per ciascuna coppia il vettore $\boldsymbol{y}(n)$ rappresenta l'uscita desiderata in corrispondenza dell'ingresso $\boldsymbol{x}(n)$. Questo insieme di coppie viene denominato *training set*. Deve essere poi reso disponibile un ulteriore insieme di coppie etichettate, distinto dal training set e denominato *test set*, sul quale valutare le prestazioni del sistema rispetto a dati mai sottoposti durante la fase di addestramento. Questo perchè, lo scopo principale del ML è di realizzare dei sistemi 'intelligenti', che siano in grado di trattare correttamente anche gli esempi mai visti in precedenza, distinti da quelli su cui il modello è stato allenato. Questa abilità è detta generalizzazione (*generalization*). Disponendo del training set e del test set, al fine di valutare lo stato di addestramento e la capacità di generalizzazione di un algoritmo di learning, è necessario disporre di una misura quantitativa della prestazione. Un esempio di accuratezza potrebbe essere dato da una metrica che valuti lo scarto tra i valori attesi e i valori effettivamente stimati dal modello sui dati di training (*training error*), durante la fase di addestramento, poi sui dati di test (*test error*), durante la fase di verifica della capacità di generalizzazione. La scelta della misura di prestazione rientra tra i fattori determinanti nella generazione del modello. Oltre a questa, anche le modalità con cui i campioni di training vengono sottoposti al sistema per l'apprendimento incidono sul risultato finale (suddivione dei campioni di training in *mini batch* e numero di epoche [5, 6]). Durante la fase di addestramento, algoritmi di ottimizzazione sono impiegati per ridurre il training error e quando questo risulta sufficientemente modesto si passa a verificare il modello corrente

sui dati di test. Questo processo è il più delle volte iterato fino al raggiungimento dello scopo prefisso o alla considerazione di un diverso approccio, qualora il tipo di modello addestrato non riesca a fornire i risultati desiderati. Durante l'iterazione con l'alternanza di addestramento e verifica del sistema, si continuano a perseguire i seguenti obiettivi:

- Mantenere l'errore di training basso;
- Mantenere la differenza tra l'errore di training e l'errore di test modesta.

Queste due attività corrispondono direttamente a controllare due comportamenti fondametali in cui potrebbe ricadere il modello addestrato: l'*underfitting* e l'*overfitting*. L'underfitting si verifica nella fase di training quando il modello non è in grado di ottenere un errore di training sufficientemente basso. Mentre, l'overfitting si manifesta quando la differenza tra errore di training ed errore di test è troppo marcata e il sistema non generalizza in modo appropriato. Un altro modo per controllare se un modello

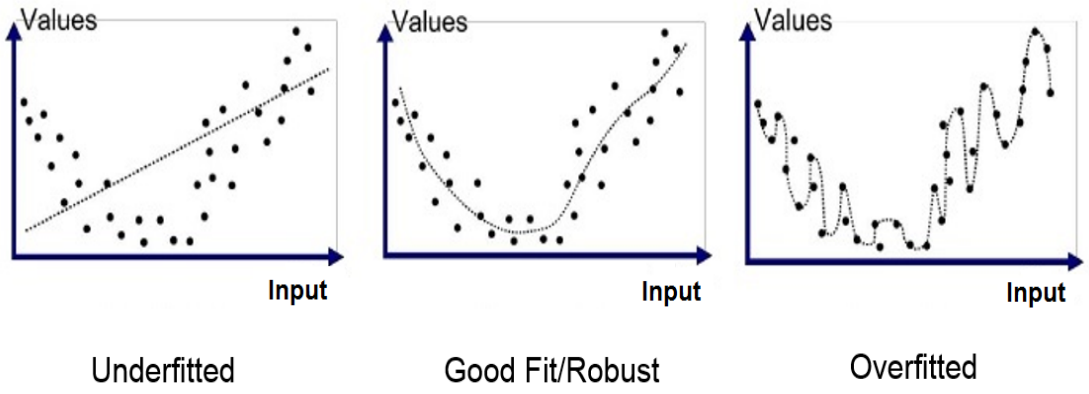



Figura 3: Esempio di underfitting, good fit e overfitting con modello polinomiale di grado sempre più elevato.

tende di più all'overfitting o all'underfitting è alterandone la sua capacità (*capacity*), caratteristica che indica quanto un modello riesca ad adattarsi ad una vasta varietà di funzioni di stima distinte. Quindi, un modello con bassa capacità faticherà ad adattarsi già in principio durante la fase di training. Contrariamente, modelli con alta capacità potrebbero ricadere nel problema dell'overfitting, adattandosi anche fluttuazioni sempre presenti nei dati di training che non sono utili al fine dell'apprendimento. In Figura 3 è riportato un esempio di come un modello troppo semplice (retta) e di bassa capacità dia luogo all'underfitting, un modello complesso dia luogo all'overfitting, mentre un modello di capacità adeguata possa ben adattarsi ai dati. Gli algoritmi di machine

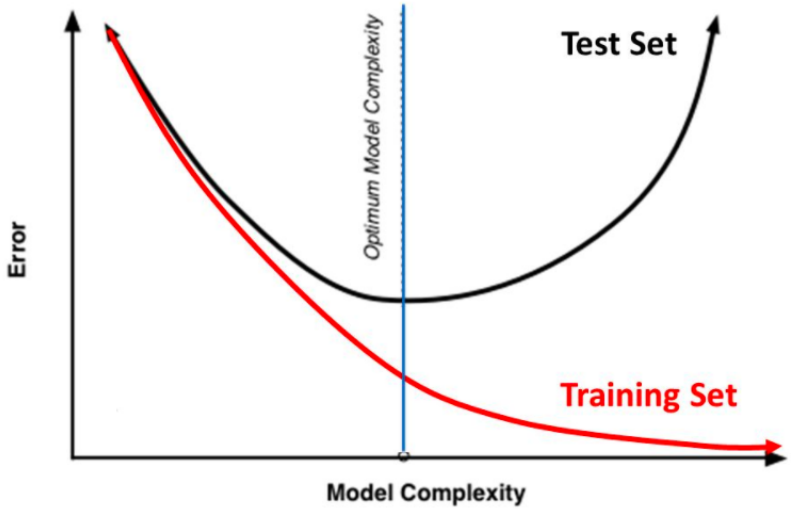



Figura 4: Andamento degli errori di training e test ripetto alla complessità del modello.

learning tendono ad avere prestazioni migliori quando la loro capacità è appropriata, quindi la complessità del modello scelto dovrà dipendere sia dalla difficoltà del compito da trattare sia dalla quantità di dati di addestramento disponibili. Questo ricordando che modelli con capacità insufficiente non sono capaci di risolvere compiti complessi, mentre modelli con capacità elevata pur essendo in grado di risolvere compiti complessi, possono incorrere nel fenomeno dell'overfitting. Come riportato in Figura 4, la selezione di un modello di adeguata capacità può avvenire controntando gli andamenti degli errori di training e di test, selezionando quel modello per cui si raggiunga il minimo errore di test. Per poter affrontare problemi complessi, che richiedono l'impiego di modelli ad elevata capacità, è possbile ricorrere alle tecniche di regolarizzazione per tenere sotto controllo il problema dell'overfitting. In particolare, la regolarizzazione è qualsiasi modifica apportata a un algoritmo di apprendimento con lo scopo di ridurre il suo errore di generalizzazione forzando la scelta di una soluzione più semplice e stabile. I metodi di regolarizzazione rivestono un ruolo centrale nel campo dell'apprendimento automatico e permettono di controllare l'addestramento di modelli di elevata capacità, ottenendo modelli in grado di risolvere problemi complessi. In Figura 5 viene riportato un esempio di una possibile tecnica di regolarizzazione nel caso di un fit di una curva. Nello specifico, il modello adottato dipende da un vettore di parametri $\boldsymbol{w}$, che possono corrispondere ad esempio ai coefficienti di un modello polinomiale con cui si vogliono approssimare i dati. Il termine di errore che si vuole minimizzare è composto dal sommando $\boldsymbol{E}_{in}(\boldsymbol{w})$, che risulta dipendente dai parametri del modello e dai dati di training, e un termine dipendente solo dai parametri del modello e da un fattore

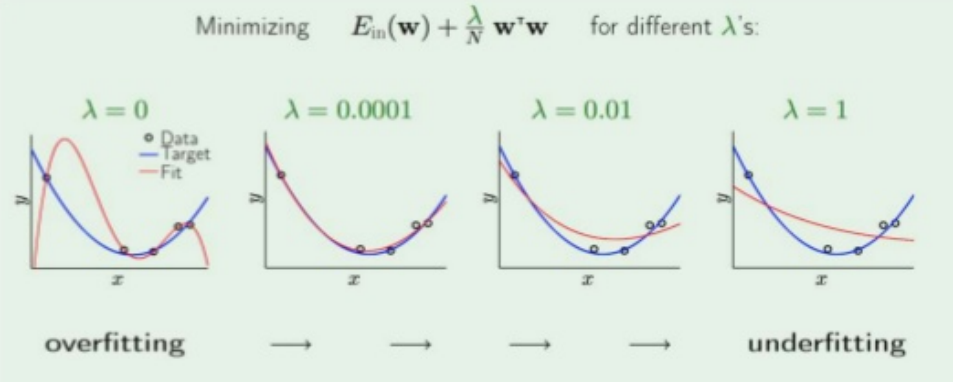



Figura 5: Esempio di tecnica di regolarizzazione.

di regolarizzazione $\lambda$. All'aumentare del valore del parametro di regolarizzazione, i parametri del modello sono costretti a essere sempre più limitati, diminuendo la capacità del modello. In questo modo il parametro $\lambda$ controlla il compromesso tra overfitting e underfitting, rendendo possibile la determinazione di un modello adeguato.

## 2.2 Multilayer Perceptron

Dopo questa breve introduzione alla terminologia relativa al ML, si passa a descrivere succintamente una tipologia di modelli specifica nota con la denominazione inglese di *Multilayer Perceptron* (MLP), da questo tipo di modello sono derivate poi le reti neurali convoluzionali (*Convolutional Neural Network* - CNN) utilizzate nella computer vision [7]. L'architettura MLP è una tipologia di modello impiegata per realizzare sia classificatori sia regressori e viene addestrata in modo supervisionato. Si assume, nella spiegazione che segue, di avere dei dati di input che sono già stati elaborati e che corrispondono pertanto alle caratteristiche di interesse. Il diagramma in Figura 6 rappresenta lo schema generale di una rete neurale MLP completamente connessa che viene ora descritta. La rete MLP è costituita da un'architettura a strati (*layers*) che vanno dallo strato di ingresso (*input layer*) allo strato di uscita (*output layer*), passando per gli strati nascosti intermedi (*hidden layers*). All'ingresso della rete vengono presentati i valori delle caratteristiche usate per effettuare la stima delle uscite. In particolare, le caratteristiche sono costituite da $M$ valori numerici di ingresso indicati in Figura 6 con $x_m$ dove $m = 1, .., M$. La rete MLP determina $P$ distinti valori di uscita indicati con $y_p$ con $p = 1, .., P$, che corrispondono alle grandezze che si desidera stimare. Lo schema di calcolo usato dall'architettura MLP prevede di convogliare i valori calcolati a ciascun livello della rete allo strato successivo. I nodi di calcolo usati, che prendono il

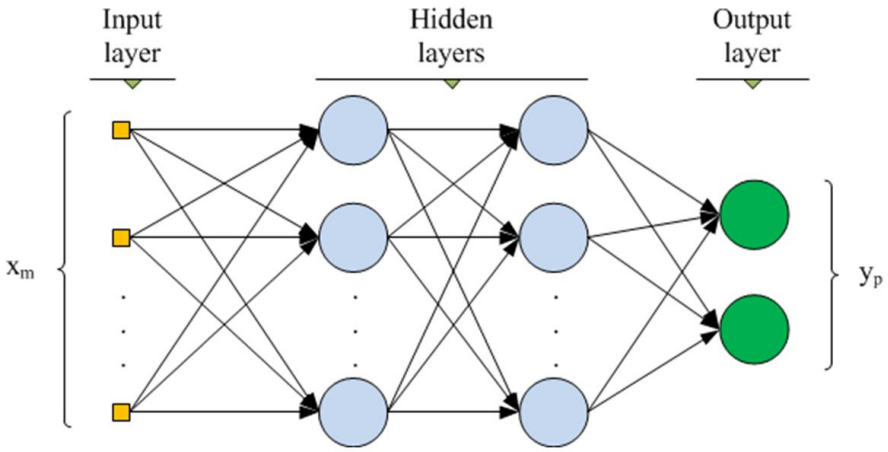



Figura 6: Schema di una rete neurale MLP.

nome di neuroni, sono rappresentati in Figura 6 con i cerchi colorati in azzurro per gli strati nascosti e in verde per i neuroni dello strato di uscita. Ogni strato ha un numero diverso di neuroni e il numero di strati usati determina la profondità della rete MLP. Lo schema della figura riporta ad esempio una rete con 2 strati nascosti e lo strato di uscita. In generale, la rete è composta da $L$ strati nascosti e lo strato $l-esimo$ comprende $M_l$ neuroni con $l = 1,..,L$. Indicato con ${x_k}^l$, $l = 1,..,L$ e $k = 1,..,M_l$, il valore di uscita dal neurone $k-esimo$ dello strato $l-esimo$ della rete, lo schema di calcolo del singolo neurone viene riportato in Figura 7. L'uscita dal neurone viene ottenuta dalla

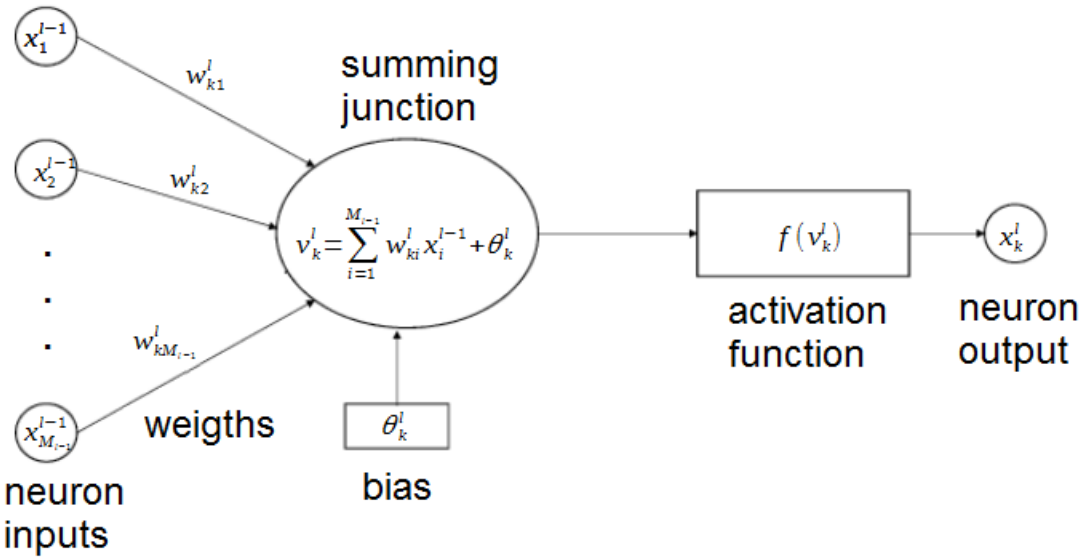



Figura 7: Schema di un singolo neurone.

somma pesata delle uscite dei neuroni dello strato precedente a cui viene sommato un

valore fisso detto *bias*. Il valore così ottenuto, indicato con $v_k{}^l$ in Figura 7, viene detto potenziale di attivazione del neurone $k-esimo$ dello strato $l-esimo$ della rete. I fattori moltiplicativi che intervengono nel calcolo del potenziale di attivazione sono detti pesi, in Figura 7 sono stati denotati $w_{ki}{}^l$ con i pesi del neurone $k-esimo$ dell'strato $l-esimo$, che vanno a moltiplicare i valori di ingresso calcolati dallo strato precedente. L'uscita del singolo neurone si calcola infine mediante una funzione non lineare, detta di attivazione, applicata al potenziale di attivazione. La scelta della funzione di attivazione determina le caratteristiche della rete e in letteratura ne sono proposte di diverse che possono essere impiegate nella stessa rete in strati distinti. Le più frequentemente usate sono riportate in Figura 8. Il modello di una rete MLP viene pertanto determinato dalla scelta del

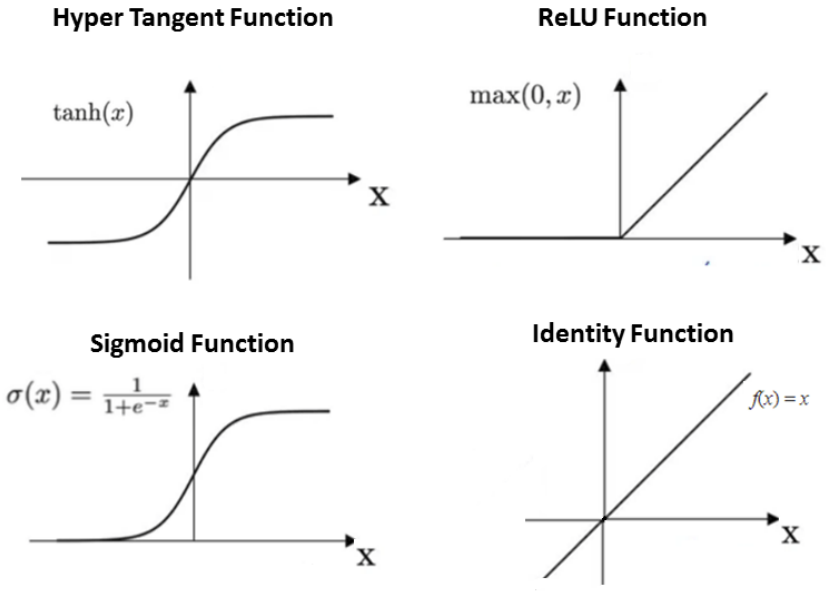



Figura 8: Funzioni di attivazione di uso comune.

numero di dati di input e di output, dal numero di strati nascosti e dal numero di neuroni che li costituiscono e dal tipo di funzioni di attivazione usate dai diversi neuroni. Oltre a questo, interviene anche la modalità con cui la rete viene addestrata e cioè il metodo con cui si determinano i pesi della rete a partire dai dati usati nella fase di training. Gli elementi che determinano la modalità di training sono: la metrica usata per determinare gli errori di training, la quantità e la tipologia di dati usati per il training, le tecniche di regolarizzazione impiegate. L'algoritmo di ottimizzazione fondamentale, usato per l'addestramento di una rete MLP e che ne ha permesso la diffusione, è quello della *back-propagation*. Per una descrizione approfondita dell'algoritmo di *back-propagation* si rimanda alla letteratura specializzata, in particolare [6]. Per la trattazione corrente è

sufficiente ricordare che si tratta di un algoritmo efficiente per effettuare una discesa di gradiente della *loss function* rispetto alle variazioni dei pesi della rete. L'algoritmo è denominato *back-propagation* perchè i gradienti della *loss function* vengono propagati all'indietro nella rete, dall'output all'input.

## 2.3 Reti neurali convoluzionali - CNN

Come già accennato, per affrontare problemi complessi risulta difficile effettuare la selezione delle features a partire dai dati grezzi acquisiti. Inoltre, sono richiesti modelli di elevata complessità. L'avvento di dispositivi di calcolo sempre più performanti, ha permesso il diffondersi di tecniche basate su reti neurali MLP con un elevato numero di strati. Queste reti, che rientrano nella categoria di modelli di tipo DL, riescono a realizzare in tutt'uno sia l'estrazione delle features dai dati sia la stima. In particolare, per il trattamento delle immagini, i dati di input sono costituiti dai valori assunti dai pixel, che tipicamente sono rappresentati da una terna di valori corrispondenti all'intensità del rosso (Red), verde (Green) e blu (Blue), immagine RGB. Le risoluzioni delle immagini possono variare, ma in molte applicazioni ci si trova a dover trattare immagini costituite da centinaia di migliaia di pixel. Per poter gestire la mole di dati di input viene impiegata un'architettura di tipo CNN, che prevede una condivisione dei pesi della rete neurale e prevede inoltre che la stessa sia per la maggior parte realizzata da strati non completamente connessi.

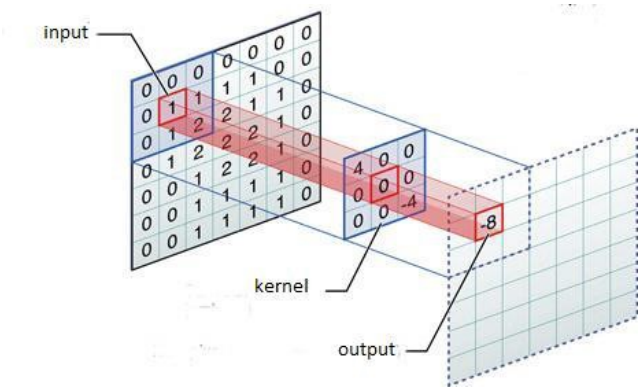



Figura 9: Filtro di convoluzione su un immagine 2D.

In particolare, come verrà di seguito approfondito, la rete CNN risulta formata da un'alternanza di strati convoluzionali e strati di riduzione della risoluzione (denominati in inglese strati di *pool*) e diverse configurazioni di uscita a seconda del tipo di stimatore realizzato. L'operazione base degli strati convoluzionali è rappresentata nelle Figure 9 e 10. Nella prima di queste viene raffigurata l'operazione di convoluzione di filtro di dimensione 3x3 (i pesi del filtro – *kernel* - vengono determinati durante la fase di apprendimento) applicato all'immagine di ingresso. La maschera del filtro viene fatta scorrere sui pixel dell'intera immagine di ingresso per ottenere la corrispondente immagine di uscita.

Quando l'immagine all'ingresso dello strato convoluzionale ha un volume maggiore a

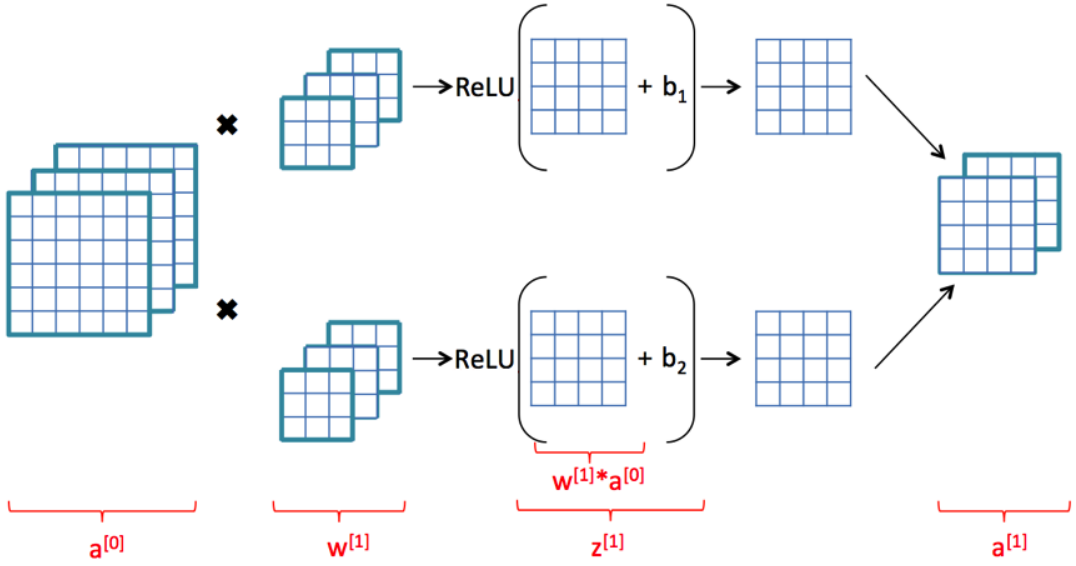



Figura 10: Esempio di strato convoluzionale con ingresso immagine a 3 componenti e 2 filtri.

1, come nel caso delle immagini RGD che hanno volume 3, lo strato convoluzionale è costituito da filtri della stessa dimensione, ma distinti per ciascun canale del volume di ingresso. Inoltre, lo strato convoluzionale è composto da più filtri di questo tipo. Ad esempio, nella Figura 10 l'ingresso è costituito da un'immagine a 3 canali e l'operazione si riferisce a 2 filtri con kernel di dimensione 3x3. Ciascuno dei filtri viene applicato al corrispondente canale immagine e le immagini risultanti vengono sommate pixel a pixel, viene poi sommato il coefficiente di bias e i dati così ottenuti sono passati alla funzione di attivazione (ReLU in figura). Nella figura si vede inoltre che i canali immagine di uscita risultano di dimensioni ridotte perchè solo i pixel che permettono all'intera maschera del filtro di sovrapporsi all'immagine vengono calcolati. Per evitare questa riduzione dovuta agli effetti di bordo, in taluni casi si effettua il filtraggio su delle immagini di ingresso allargate con un bordo posto a zero (*zero padding*). Negli strati convoluzionali, il numero di filtri che compongono la batteria è elevato per aumentare la capacità del modello. Per ridurre la complessità computazionale e per poter analizzare l'immagine su diverse scale, spesso gli strati convoluzionali prevedono un parametro detto di passo (*stride*) che indica di calcolare i filtri avanzando da un pixel al successivo saltando in avanti sia orizzontalmente sia vericalmente del passo indicato. La modalità appena descritta è rappresentata in Figura 11 dove si considera in ingresso un'immagine a 3 canali che è stata allargata con la cornice di zero padding per permettere il filtraggio con dei kernel

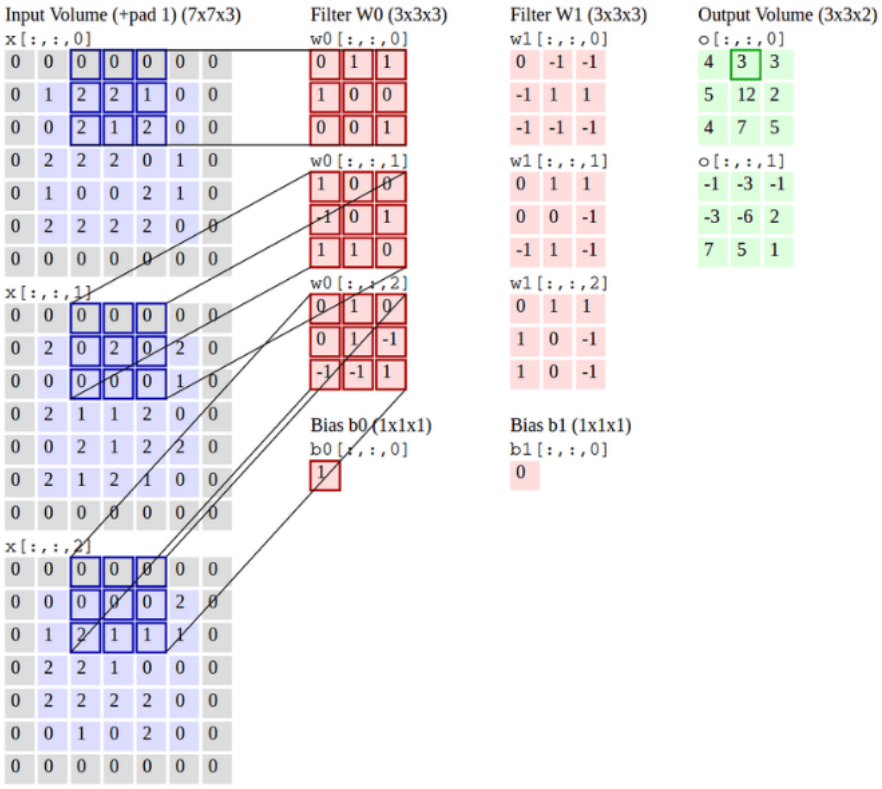



Figura 11: Esempio di strato convoluzionale composto da 2 filtri di dimensione 3x3 a passo 2 con zero padding.

di dimensione 3x3. La batteria dello strato è composta da 2 filtri e lo stride impostato è 2. Oltre agli strati convoluzionali appena descritti, l'architettura CNN prevede la

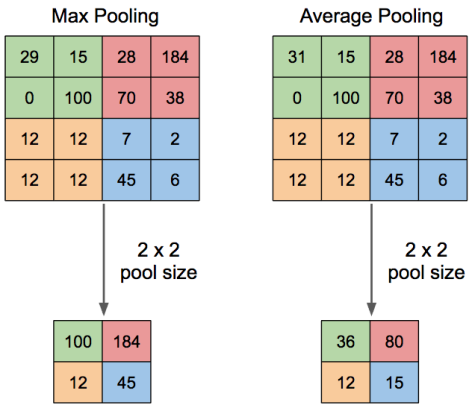



Figura 12: Esempio di pooling layer.

possibilità di inserire degli strati detti di pooling (*pooling layer*). Questi prevedono la riduzione delle features andando ad applicare delle operazioni di massimo o di media

su una maschera di dimensioni tipicamente 2x2 o 3x3. Come esempio, in Figura 12 sono rappresentate le due tipologie di strato di pool con una maschera 2x2: si fa notare come per entrambi abbia luogo una riduzione della dimensione dell'immagine risultato.

## 2.4 Reti neurali per segmentazione semantica delle immagini

Si descrivono di seguito alcuni modelli per la segmentazione semantica delle immagini. Per un recente riepilogo delle diverse soluzioni adottate si veda [8]. In Figura 13 sono rappresentati i risultati ottenibili con la segmentazione semantica su alcune immagini di esempio. Le architetture e le tipologie di modello usate nella segmentazione semantica

Figura 13: Esempi di segmentazione semantica.

sono un'area attiva di ricerca. In particolare, la ricerca considera vari aspetti quali:

- Aumentare l'accuratezza dei modelli;
- Ridurre la complessità di calcolo per le applicazioni su dispositivi mobili;
- Semplifcare il training dei modelli;

- Utilizzare dati sintetici per addestrare nuovi modelli che siano adattabili anche a dati reali.

Sono così state presentate diverse tipologie di innovazione che si riassumono nel diagramma di Figura 14 che riporta la denonominazione delle principali architetture proposte e l'anno di introduzione dell'innovazione indicata. Molti dei modelli proposti in

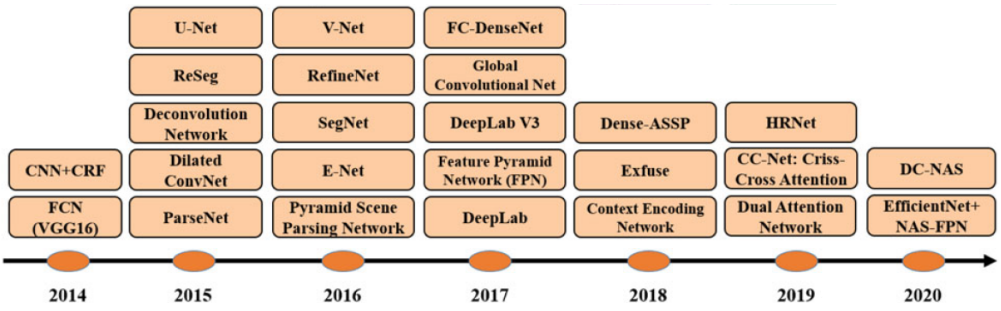
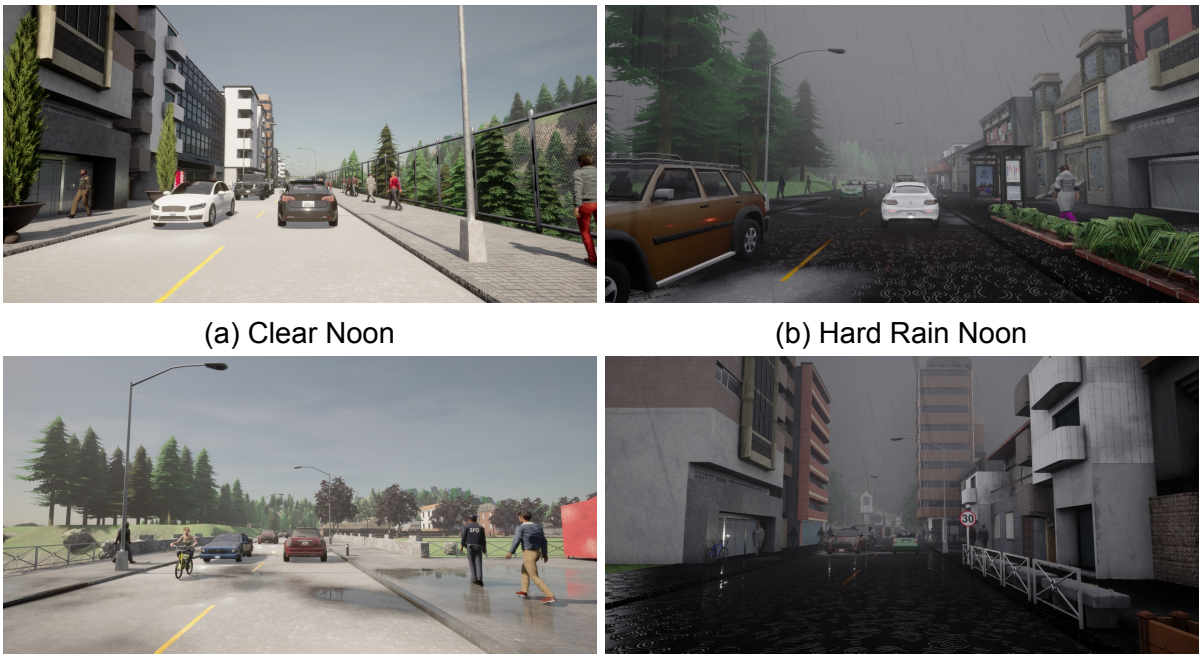



Figura 14: Timeline dei modelli DL per segmentazione semantica più significativi.

tempi più recenti usano un'architettura di tipo encoder-decoder, dove la parte di codifica viene demandata a reti pre-addestrate (*backbones*) su dataset di grandi dimensioni. Usando questa strategia, risulta possibile addestrare reti di segmentazione di buona accuratezza, pur disponendo di un limitato numero di immagini etichettate per il training.

# 3 Adattamento di dominio da dataset sintetici a reali

Per poter addestrare nuovi modelli DL senza dover etichettare le immagini usate per il training, risulterebbe vantaggioso poter usare delle immagini sintetiche, per le quali l'etichettatura viene generata automaticamenta con l'immagine stessa. Allo scopo, una delle tecniche proposte, denominata *Domain Adaptation*, consiste nell'applicare alle immagini sintetiche una trasformazione che le renda "simili" alle immagini reali, che dovranno poi essere date in input al modello DL nell'applicazione finale. Recentemente, per effettuare la *Domain Adaptation* è stato proposto un metodo che si basa sull'analisi in frequenza delle immagini sintetiche e reali [1]. La metodica consiste nel sostituire il contenuto di bassa frequenza del modulo dello spettro di un'immagine sintetica con lo spettro riconducibile alle immagini reali. Una possibilità è quella di calcolare lo spettro medio di un certo numero di immagini reali, per poi usarlo per l'adattamento delle immagini sintetiche. Nell'ambito della guida autonoma, sono stati presentati sia dataset sintetici (ad esempio SELMA) [3] sia dataset di immagini acquisite da scene reali (ad esempio Cityscapes) [2]. Nella Figura 15 sono riportate alcune immagini esemplificative del dataset sintetico SELMA.



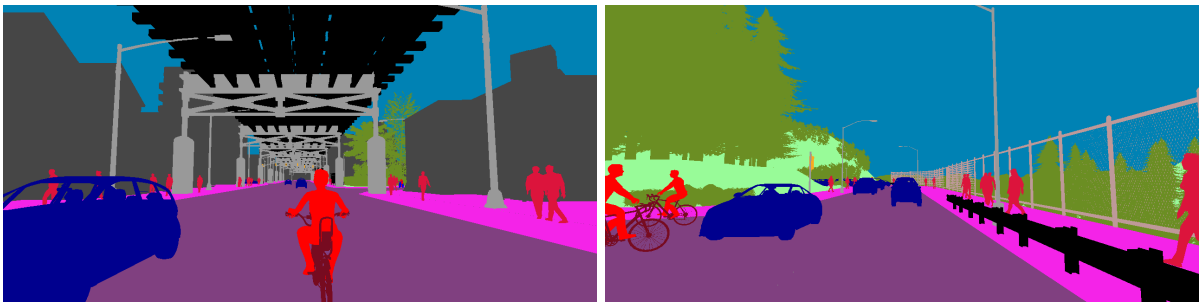
(a) Clear Noon (b) Hard Rain Noon

(c) Wet Noon (d) Hard Rain Sunset

Figura 15: Esempi di immagini sintetiche del dataset SELMA

Il dataset SELMA include immagini sintetiche di percorsi stradali nei quali sono presenti diverse categorie di mezzi (camion, automobili, moto, etc.), persone e altri oggetti (alberi,

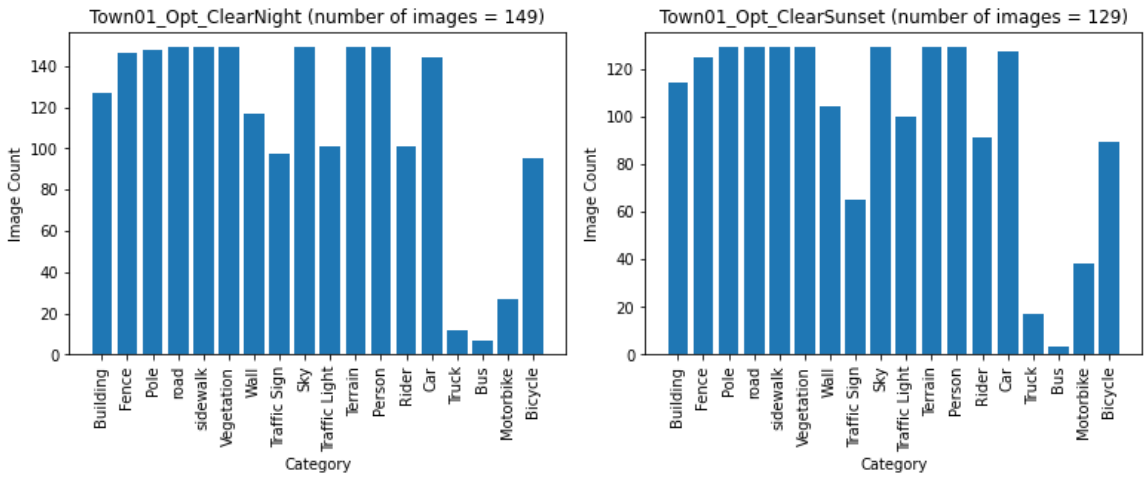
lampioni, segnaletica stradale, edifici, etc.). Le immagini del dataset cercano inoltre di riprodurre l'effetto di diverse condizioni atmosferiche nelle diverse ore del giorno. In tal modo, potendo generare sinteticamente i dati, risulta possibile creare una molteplicità di situazioni in combinazione diversa, molto utile per verificare la capacità di adattamento degli algoritmi di elaborazione alle diverse condizioni. Esempi di segmentazione per alcune immagini del dataset SELMA in Figura 16.



(a) Town03_Opt_CloudySunset (b) Town01_Opt_ClearNight

Figura 16: Esempi di segmentazione di alcune immagini del dataset SELMA

Nel dataset SELMA sono etichettate diverse tipologie di classi, che sono state individuate e contate. Si sono così potuti generare degli istogrammi per il conteggio delle classi presenti come quelli riportati ad esempio in Figura 17. Per poter efficacemente




(a) Clear Night (b) Clear Sunset

Figura 17: Istogrammi delle etichette presenti nelle immagini SELMA per diverse condizioni luce e atmosferiche

usare i dati sintetici, che per loro natura sono annotati già alla generazione, si devono esplorare tecniche che siano in grado di utilizzare questi dati nell'addestramento di

modelli di segmentazione semantica utili per trattare i dati acquisiti da videocamere. Nell'affrontare questa sfida risulta utile avere a disposizione dei dataset, come Cityscapes, che contengono una numerevole quantità di immagini reali già etichettate. Immagini esemplificative del dataset Cityscapes sono visibili in Figura 18.

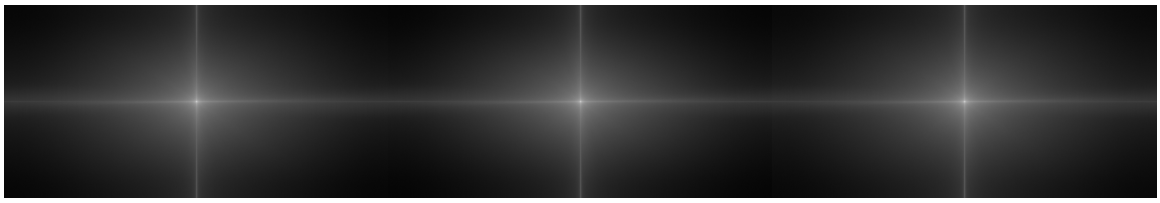
(a) frankfurt_000001_073243_left (b) lindau_000038_000019_left

(c) lindau_000042_000019_left (d) munster_000118_000019_left

Figura 18: Esempi di immagini reali del dataset Cityscapes

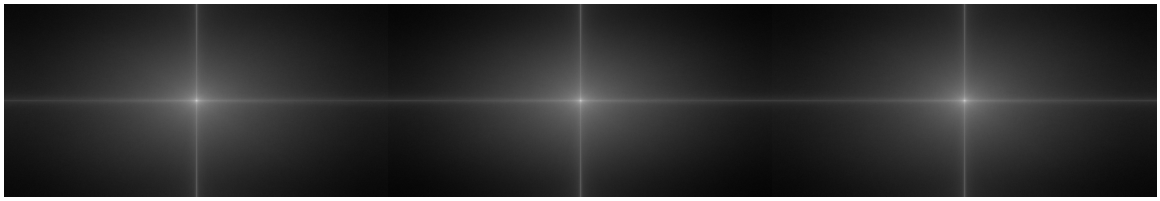
Per adattare al dominio delle immagini Cityscapes le immagini del dataset SELMA, si è scelto di provare a utilizzare la metodica descritta in [1], sostituendo le componenti del modulo dello spettro a più bassa frequenza con quelle ottenute dalle statistiche degli spettri medi del dataset Cityscapes. Allo scopo sono state calcolate le statistiche del modulo degli spettri per ciascun canale RGB delle immagini nel dataset. I moduli degli spettri medi ottenuti per i divesi canali RGB sono poi stati rappresentati in scala logaritmica per poter meglior visualizzazione. In particolare gli Spettri medi delle componenti RGB delle immagini Cityscapes Frankfurt visibili in Figura 19, come quelle Lindau in Figura 20 e infine quelle Munster visibili in Figura 21. Per poter confrontare la differenza rispetto agli spettrii medi delle componenti RGB delle immagini SELMA sono state calcolate le medesime statistiche per diverse condizione di luce e atmostferiche. In particolare si vedano i casi Town01_Opt_ClearNight visibili in Figura 22 e Town01_Opt_ClearSunset visibili in Figura 23.

Per apprezzare le differenze esistenti tra i diversi spettri medi delle componenti RGB di un dataset sono stati messi a confronto i moduli degli spettri medi dei canali Red e



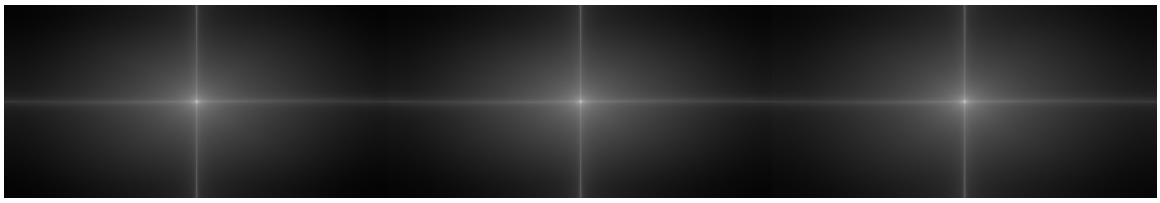
(a) red (b) green (c) blue

Figura 19: Modulo dello spettro medio delle componenti RGB per immagini Cityscapes Frankfurt



(a) red (b) green (c) blue

Figura 20: Modulo dello spettro medio delle componenti RGB per immagini Cityscapes Lindau



(a) red (b) green (c) blue

Figura 21: Modulo dello spettro medio delle componenti RGB per immagini Cityscapes Munster

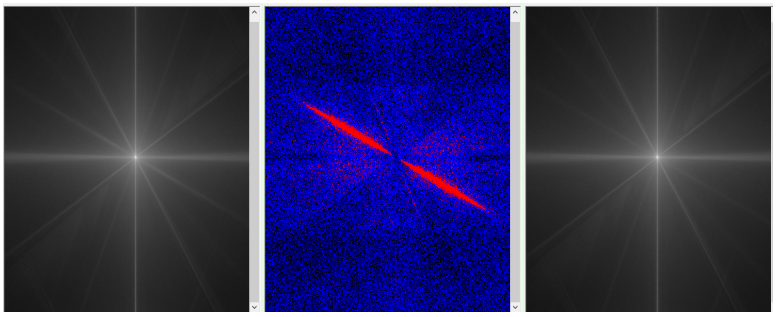
Green. Il confronto è rappresentato in Figura 24 dove si riportano ai lati gli spettri e al centro una visualizzazione a colori dei pixel che differiscono tra i due spettri presi in considerazione. In particolare, sono colorati in blu i pixel per cui gli spettri delle due componenti differiscono meno di una soglia di tolleranza (soglia fissata a 16 livelli di grigio su 256), sono colorati in rosso quei pixel per cui la differenza supera la soglia di tolleranza, infine, sono marcati in nero i pixel per cui si ha un'esatta corrispondenza dei valori nelle due componenti considerate. Per studiare come il valore assoluto delle immagini reali target influenzi le immagini sintetiche del dataset SELMA, quest'ultime sono state ricostruite utilizzando come valore assoluto lo spettro medio delle immagini target e conservandone la fase originale. Degli esempi di immagini adattate sugli spettri medi delle immagini target sono visibili in Figura 25.

(a) red (b) green (c) blue

Figura 22: Modulo dello spettro medio delle componenti RGB per immagini SELMA ClearNight

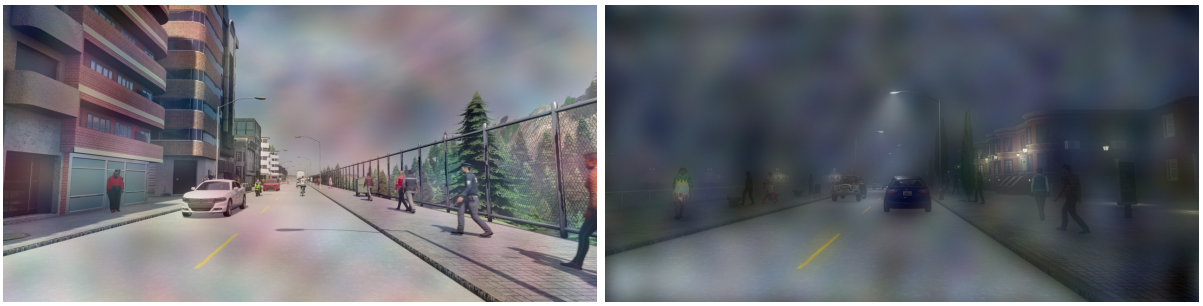
(a) red (b) green (c) blue

Figura 23: Modulo dello spettro medio delle componenti RGB per immagini SELMA ClearSunset



(a)

Figura 24: Spettri di diverse componenti RGB a confronto



(a) Adapted Clear Noon (b) Adapted Cloudy Night

Figura 25: Esempi di adattamento sugli spettri medi delle immagini reali (target)

Di seguito sono riportati i Listati del codice Python sviluppato per poter ottenere le

immagini presentate più sopra. In particolare, la funzione *get_filepaths* per generare una lista dei file contenuti nella directory su cui la funzione viene invocata

```
import os

def get_filepaths(directory):
    """
    This function will generate the file names in a directory
    tree by walking the tree either top-down or bottom-up.
    """
    file_paths = [] # List which will store all of the full filepaths.

    # Walk the tree.
    for root, directories, files in os.walk(directory):
        for filename in files:
            # Join the two strings in order to form the full filepath.
            filepath = os.path.join(root, filename)
            extension = os.path.splitext(filename)[1]
            file_paths.append(filepath) # Add it to the list.

    return file_paths
```

Il Codice Python nel seguente listato è stato scritto per eseguire l'overlay delle immagini del dataset SELMA con le rispettive label

```
import cv2
import os
import colorMap
import fileUtility

directory = "D:\\datasetSELMA"
subdirs = [os.path.join(directory, o) for o in os.listdir(directory) if os↩
    .path.isdir(os.path.join(directory,o))]

#Creazione di tutte le sottodirectory OVCAM_FRONT_LEFT (solo la prima ↩
    volta)
for mydir in subdirs:
```

```
   path = os.path.join(mydir, "SEGOVCAM_FRONT_LEFT")
   if os.path.exists(path) == False:
      os.mkdir(path)

cmap = colorMap.computeLabelColorMap()

for mydir in subdirs:
    imgpath = os.path.join(mydir, "CAM_FRONT_LEFT")
    imglabelpath = os.path.join(mydir, "SEGCAM_FRONT_LEFT")

    img_files = fileUtility.get_filepaths(imgpath)
    imglabel_files = fileUtility.get_filepaths(imglabelpath)
    saveoverlaydir = os.path.join(mydir, "SEGOVCAM_FRONT_LEFT")

    for imgfile, imglabelfile in zip(img_files, imglabel_files):
        img = cv2.imread(imgfile)
        imglabel = cv2.imread(imglabelfile, cv2.IMREAD_UNCHANGED)
        overlayfile = os.path.join(saveoverlaydir, os.path.basename(↩
            imglabelfile))
        imglabelheatmap = cmap[imglabel]
        added_image = cv2.addWeighted(img,0.7,imglabelheatmap,0.3,0)
        cv2.imwrite(overlayfile, added_image)
```

Il precedente codice utilizza la funzione *Colormap* per codificare le classi in modo da renderle facilmente riconoscibili

```
import numpy as np

def computeLabelColorMap():
    # initialize the colormap
    cmap = np.zeros((256, 3), dtype=np.uint8)
    cmap[1] = [70, 70, 70] # Building
    cmap[2] = [190, 153, 153] # Fence
    cmap[5] = [153, 153, 153] # Pole
    cmap[6] = [128, 64, 128] # road
    cmap[7] = [128, 64, 128] # road
    cmap[8] = [244, 35, 232] # sidewalk
    cmap[9] = [107, 142, 35] # Vegetation
```

```
    cmap[11] = [102, 102, 156] # Wall
    cmap[12] = [220, 220, 0] # Traffic Sign
    cmap[13] = [0, 130, 180] # Sky
    cmap[18] = [250, 170, 30] # Traffic Light
    cmap[22] = [152, 251, 152] # Terrain
    cmap[40] = [220, 20, 60] # Person
    cmap[41] = [255, 0, 0] # Rider
    cmap[100] = [0, 0, 142] # Car
    cmap[101] = [0, 0, 70] # Truck
    cmap[102] = [0, 60, 100] # Bus
    cmap[103] = [0, 80, 100] # Train
    cmap[104] = [0, 0, 230] # Motorbike
    cmap[105] = [119, 11, 32] # Bicycle

    # convert it from RGB to BGR, flipping the last dimension
    # since opencv expects images in that format
    cmap = cmap[:,::-1]

    return cmap

def getValidLabelDescriptions():
    d = dict([
    (1, 'Building'),
    (2, 'Fence'),
    (5, 'Pole'),
    (6, 'road'),
    (7, 'road'),
    (8, 'sidewalk'),
    (9, 'Vegetation'),
    (11, 'Wall'),
    (12, 'Traffic Sign'),
    (13, 'Sky'),
    (18, 'Traffic Light'),
    (22, 'Terrain'),
    (40, 'Person'),
    (41, 'Rider'),
    (100, 'Car'),
    (101, 'Truck'),
```

```
    (102, 'Bus'),
    (103, 'Motorbike'),
    (104, 'Motorbike'),
    (105, 'Bicycle')])

    return d
```

Il seguente listato riporta il codice Python per calcolare la distribuzione delle classi nei campioni di segmentazione, usato per poter poi ottenere gli istogrammi di Figura 17.

```
import cv2
import os
import numpy as np
import fileUtility

directory = "D:\\datasetSELMA"
subdirs = [os.path.join(directory, o) for o in os.listdir(directory) if os↩
    .path.isdir(os.path.join(directory,o))]

# Navigate every light condition directory
for mydir in subdirs:
    # Navigate every directory with images labels
    imglabelpath = os.path.join(mydir, "SEGCAM_FRONT_LEFT")
    imglabel_files = fileUtility.get_filepaths(imglabelpath)

    # Prepare an histogram of labels initialized to all zeros
    histoFileLabels = np.zeros((256), dtype=np.uint32)
    for imglabelfile in imglabel_files:
        # Read label image
        imglabel = cv2.imread(imglabelfile, cv2.IMREAD_UNCHANGED)
        # Compute histogram
        hist = cv2.calcHist([imglabel],[0],None,[256],[0,256])
        # Find non zero bin (actual labels)
        histNonZeroBins = np.ndarray.nonzero(hist)[0]
        # Increment statistics forcurrent image labels
        histoFileLabels[histNonZeroBins] += 1

    # Find non zero bins of the histogram
```

```
    histNonZeroBins = np.ndarray.nonzero(histoFileLabels)[0]

    # 2D array with labels and corresponding occurences within the images ↩
        in the current
    # "SEGCAM_FRONT_LEFT" directory
    histoLabelFinal = np.vstack((histNonZeroBins, histoFileLabels[↩
        histNonZeroBins])).T

    # Save computed statistics in "SEGCAM_FRONT_LEFT" directory
    np.savetxt(imglabelpath + '\\array.csv', histoLabelFinal, delimiter=',↩
        ', fmt='%5d', header='label, count')
```

Codice Python per calcolare le statistiche degli spettri medi delle immagini del dataset Cityscapes e la domain adaptation ottenuta utilizzando questi spettri medi come target. Esempi di risultati della *domain adaptation* ottenuta sono riportati in Figura 25.

```
import cv2
import numpy as np
import os
import fileUtility
import sys

# Compute Shifted (Low frequencies at the center of the image) Magnitude
# and Phase of DFT of input image ($imgsrc)
def computeMagnitudePhaseShiftedDFT(imgsrc):
    # convert image to floats and do dft saving as complex output
    dft = cv2.dft(imgsrc, flags = cv2.DFT_COMPLEX_OUTPUT)
    # apply shift of origin from upper left corner to center of image
    dft_shift = np.fft.fftshift(dft)

    # extract magnitude and phase images
    mag, phase = cv2.cartToPolar(dft_shift[:,:,0], dft_shift[:,:,1])

    return mag, phase


# Compute IDFT of Shifted DFT (Low frequencies at the center of the image)↩
     Magnitude
```

```
# and Phase
def computeIDFTfromShiftedMagnitudePhase(mag, phase):
    # convert magnitude and phase into cartesian real and imaginary components
    real, imag = cv2.polarToCart(mag, phase)

    # combine cartesian components into one complex image
    back = cv2.merge([real, imag])

    # shift origin from center to upper left corner
    back_ishift = np.fft.ifftshift(back)

    # do idft saving as complex output
    img_back = cv2.idft(back_ishift)

    imgres = img_back[:,:,0]
    height = imgres.shape[0]
    width = imgres.shape[1]
    imgres /= (height * width)

    return imgres


# Perform channel adaptation based on Soatto proposal of a grayScale image
# imgsrc source image to modify
# imgref reference image used as target for adaptation
# beta float fraction of low frequency rectangle to crop from target spectra and copy to source spectra
def adaptChannelGrayScale(imgsrc, imgref, beta):
    imgsrc_f = imgsrc / 255
    imgsrc_f = imgsrc_f.astype(np.float32)

    imgref_f = imgref / 255
    imgref_f = imgref_f.astype(np.float32)

    meansrc = cv2.mean(imgsrc_f)[0]
    meanref = cv2.mean(imgref_f)[0]
```

```
    imgsrc_0 = imgsrc_f - meansrc
    imgref_0 = imgref_f - meanref

    magsrc, phasesrc = computeMagnitudePhaseShiftedDFT(imgsrc_0)
    magref, phaseref = computeMagnitudePhaseShiftedDFT(imgref_0)

    height = magsrc.shape[0]
    width = magsrc.shape[1]

    heightCrop = height * beta / 2
    widthCrop = width * beta / 2
    ycenter = height / 2
    xcenter = width / 2
    yTop = int(ycenter - heightCrop - 0.5)
    yBottom = int(ycenter + heightCrop + 0.5)
    xLeft = int(xcenter - widthCrop - 0.5)
    xRight = int(xcenter + widthCrop + 0.5)

    magsrc[yTop:yBottom, xLeft:xRight] = magref[yTop:yBottom, xLeft:xRight↩
        ]

    img_back = computeIDFTfromShiftedMagnitudePhase(magsrc, phasesrc)

    imgres = img_back + meanref

    print(imgres.min(), imgres.max())

    #Riscala i valori dei pixel tra 0 e 1
    imgres = imgres - imgres.min()
    imgres = imgres / imgres.max()
    imgres *= 255

    imgres = imgres.astype(np.uint8)

    return imgres

# Perform channel adaptation based on Soatto proposal of a grayScale image
# imgsrc source image to modify
```

```
# magref modulus of spectra of target images (statistical mean of spetra)
# beta float fraction of low frequency rectangle to crop from target ↩
    spectra and copy to source spectra
def adaptChannelGrayScaleToMagnitudeStatistic(imgsrc, magref, beta):
    imgsrc_f = imgsrc / 255
    imgsrc_f = imgsrc_f.astype(np.float32)


    meansrc = cv2.mean(imgsrc_f)[0]

    imgsrc_0 = imgsrc_f - meansrc

    magsrc, phasesrc = computeMagnitudePhaseShiftedDFT(imgsrc_0)

    height = magsrc.shape[0]
    width = magsrc.shape[1]

    heightCrop = height * beta / 2
    widthCrop = width * beta / 2
    ycenter = height / 2
    xcenter = width / 2
    yTop = int(ycenter - heightCrop - 0.5)
    yBottom = int(ycenter + heightCrop + 0.5)
    xLeft = int(xcenter - widthCrop - 0.5)
    xRight = int(xcenter + widthCrop + 0.5)

    magsrc[yTop:yBottom, xLeft:xRight] = magref[yTop:yBottom, xLeft:xRight↩
        ]

    img_back = computeIDFTfromShiftedMagnitudePhase(magsrc, phasesrc)

    imgres = img_back

    print(imgres.min(), imgres.max())

    #Riscala i valori dei pixel tra 0 e 1
    imgres = cv2.normalize(imgres, None, 0, 255, cv2.NORM_MINMAX, cv2.↩
        CV_8U)
```

```
    return imgres


# Perform channel adaptation based on Soatto proposal of a color image
def adaptChannelColor(imgsrc, imgref, beta):
    imgsrc_b, imgsrc_g, imgsrc_r = cv2.split(imgsrc)

    imgref_b, imgref_g, imgref_r = cv2.split(imgref)

    imgres_r = adaptChannelGrayScale(imgsrc_r, imgref_r, beta)
    imgres_g = adaptChannelGrayScale(imgsrc_g, imgref_g, beta)
    imgres_b = adaptChannelGrayScale(imgsrc_b, imgref_b, beta)

    imgres = cv2.merge([imgres_b, imgres_g, imgres_r])

    return imgres

# Perform channel adaptation based on Soatto proposal of a color image
def adaptChannelColorToMagnitudeStatistics(imgsrc, magb, magg, magr, beta)↩
    :
    imgsrc_b, imgsrc_g, imgsrc_r = cv2.split(imgsrc)


    imgres_r = adaptChannelGrayScaleToMagnitudeStatistic(imgsrc_r, magr, ↩
        beta)
    imgres_g = adaptChannelGrayScaleToMagnitudeStatistic(imgsrc_g, magg, ↩
        beta)
    imgres_b = adaptChannelGrayScaleToMagnitudeStatistic(imgsrc_b, magb, ↩
        beta)

    imgres = cv2.merge([imgres_b, imgres_g, imgres_r])

    return imgres

# Compute the mean spectra of the images in the list
# img_files list of image filenames
# width used to resize Cityscapes images to SELMA resolution
```

```
# height used to resize Cityscapes images to SELMA resolution
# enableLogScale if True returns the Log of mean Spectrum for visualization
def computeMagnitudeStatistics(img_files, width, height, enableLogScale):
    numimags = len(img_files)
    magstatistic_b = np.zeros([height, width],dtype=np.float32)
    magstatistic_g = np.zeros([height, width],dtype=np.float32)
    magstatistic_r = np.zeros([height, width],dtype=np.float32)

    for imgfile in img_files:
        img = cv2.imread(imgfile)

        img = cv2.resize(img, (width, height), interpolation = cv2.INTER_AREA)

        img_b, img_g, img_r = cv2.split(img)

        img_fb = np.float32(img_b / 255)
        img_fg = np.float32(img_g / 255)
        img_fr = np.float32(img_r / 255)


        magsrc, phasesrc = computeMagnitudePhaseShiftedDFT(img_fb)
        if enableLogScale:
            magsrc = 20 * np.log(magsrc)
        magstatistic_b = magstatistic_b + magsrc

        magsrc, phasesrc = computeMagnitudePhaseShiftedDFT(img_fg)
        if enableLogScale:
            magsrc = 20 * np.log(magsrc)
        magstatistic_g = magstatistic_g + magsrc

        magsrc, phasesrc = computeMagnitudePhaseShiftedDFT(img_fr)
        if enableLogScale:
            magsrc = 20 * np.log(magsrc)
        magstatistic_r = magstatistic_r + magsrc

    magstatistic_b = magstatistic_b / numimags
```

```
    if enableLogScale:
        magstatistic_b = cv2.normalize(magstatistic_b, None, 0, 255, cv2.↩
            NORM_MINMAX, cv2.CV_8U)

    magstatistic_g = magstatistic_g / numimags
    if enableLogScale:
        magstatistic_g = cv2.normalize(magstatistic_g, None, 0, 255, cv2.↩
            NORM_MINMAX, cv2.CV_8U)

    magstatistic_r = magstatistic_r / numimags
    if enableLogScale:
        magstatistic_r = cv2.normalize(magstatistic_r, None, 0, 255, cv2.↩
            NORM_MINMAX, cv2.CV_8U)

    return magstatistic_b, magstatistic_g, magstatistic_r

# Cityscapes Frankfurt images as reference
directory = "D:\\datasetCityscapes\\frankfurt"


img_files = fileUtility.get_filepaths(directory)
numimags = len(img_files)
print(numimags)

# Compute mean Spectrum of RGB components of Cityscapes Frankfurt images
heightsrc = 640
widthsrc = 1280
magstatistic_b, magstatistic_g, magstatistic_r = ↩
    computeMagnitudeStatistics(img_files, widthsrc, heightsrc, ↩
    enableLogScale = False)

#carica immagine sintetica da adattare
imgfile = "D:\\datasetSELMA\\Town01_Opt_CloudyNight\\CAM_FRONT_LEFT\\↩
    Town01_Opt_CloudyNight_15695704861255172244.jpg"

imgsrc = cv2.imread(imgfile, cv2.IMREAD_UNCHANGED)
heightsrc = imgsrc.shape[0]
widthsrc = imgsrc.shape[1]
```

```
cv2.imshow("ORIGINAL", imgsrc)


#effettua la domain adaptation
beta = 0.02
imgAdapted = adaptChannelColorToMagnitudeStatistics(imgsrc, magstatistic_b
    , magstatistic_g, magstatistic_r, beta)

cv2.imshow("ADAPTED", imgAdapted)
cv2.waitKey(0)
cv2.destroyAllWindows()

cv2.imwrite("D:\\datasetSELMA\\Town01_Opt_CloudyNight\\
    AdaptedTown01_Opt_CloudyNight_15695704861255172244.jpg", imgAdapted)
```

# 4 Conclusioni

Lo svolgimento del presente lavoro ha permesso di approfondire la conoscenza in diversi ambiti. Oltre all'affrontare lo studio degli articoli [1, 3] e dei riferimenti bibliografici ivi indicati, si è dovuto procedere con lo studio per avere una prima infarinatura del machine learning [5] e del Deep Learning in particolare [4], per poi vedere come le tecniche di DL vengano applicate nell'ambito della computer vision sotto forma di modelli CNN [7]. Ci si è poi dedicati allo studio della tecnica di *domain adaptation* presentata in [3], passando poi all'implementazione in linguaggio Python di alcune delle funzioni previste dalla metodica di elaborazione ivi presentata. L'attività svolta potrà essere estesa andando a verificare l'efficacia nell'uso delle immagini adattate per l'addestramento di reti neurali di segmentazione semantica. In questo studio si dovrà considerare come iper-parametro addizionale a disposizione anche la dimensione della regione di spettro a bassa frequenza viene usata per l'adattamento. In particolare, si è già osservato che la sostituzione di una regione troppo estesa dello spettro a bassa frequenza comporta la presenza di artefatti nelle immagini adattate che si ottengono.

# Bibliografia

[1] Stefano Soatto Yanchao Yang. Fda: Fourier domain adaptation for semantic segmentation. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[2] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] Umberto Michieli Marco Giordani Pietro Zanuttigh Michele Zorzi Paolo Testolina, Francesco Barbato. Selma: Semantic large-scale multimodal acquisitions in variable weather, daytime and viewpoints. *JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2022*, 2022.

[4] Courville Aaron Bengio Yoshua and Goodfellow Ian. *Deep Learning*. Morgan Claypool Publisher, 2018.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2009.

[7] Salman Khan, Hossein Rahmani Syed Afaq Ali Shah, and Mohammed Bennamoun. *A Guide to Convolutional Neural Networks for Computer Vision*. MIT Press, 2016.

[8] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.