Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in
Ingegneria Informatica

Sviluppo software di un sistema di telecamere intelligenti per la sicurezza dei bambini in piscina

Relatrice:
Prof.ssa Gloria Beraldo

Laureanda:
Sofia Fragomeni

Correlatori:
Alessandro Devo
Matteo Pozzi

Anno Accademico 2022/2023
Data di Laurea 21/11/2023

“A chi ha sempre creduto in me”

Abstract

La sicurezza in ambienti quali le piscine assume un'importanza fondamentale, considerando il carattere delicato di tali contesti, dove gli incidenti possono avere conseguenze gravi, specialmente per i più giovani, basti pensare che l'annegamento è la seconda causa di morte per lesioni non intenzionali tra questi ultimi. Questa tesi si concentra sulla rivelazione del pericolo di annegamento in piscina.
Lo scopo della tesi sarà quindi quello di progettare e implementare un sistema intelligente basato su algoritmi di *Machine Learning* in grado di riconoscere queste situazioni pericolose a partire da immagini RGB raccolte mediante una rete di telecamere.
A tale fine, a partire dallo studio della letteratura inerente, utilizzando le librerie scikit-learn e Keras si sviluppano diversi classificatori come: ResNet50, VGG, IncepionV3 ed SVMs. L'addestramento dei modelli scelti viene condotto nell'ambiente di Google Colaboratory, con l'obiettivo preciso di riconoscere le due classi: *drowning* (annegamento) e *swimming* (nuoto).
Si sono presi in esame tre datasets differenti: *Drowning people* scaricato da Roboflow e composto da 163 immagini che raffigurano più situazioni di nuoto e annegamento, *Swimming and Drowning Detection* scaricato anch'esso da Roboflow è un dataset composto da 12365 immagini suddiviso in tre classi, infine *Water Behavior Dataset* gentilmente concesso da Electrical Engineering del Rochester Institute of Technology Dubai composto da più video in acqua.
I migliori modelli hanno portato alle seguenti performance: 73.62% di *accuracy* sul dataset *Drowning people* e 71.57% sul dataset *Water Behavior*.

Indice

Capitolo 1

Introduzione

I primi riferimenti all'intelligenza artificiale risalgono al 1943 quando Warren McCulloch e Walter Pitt presentarono alla comunità scientifica il primo neurone artificiale [1]. I neuroni artificiali sono il nucleo delle reti neurali, basati su modelli matematici, simulano il comportamento dei neuroni biologici (Figura 1.1).
Nonostante l'entusiasmo iniziale per le possibili applicazioni che queste scoperte scientifiche avrebbero potuto portare in molteplici ambiti da quello medico a quello militare, gli sviluppi non furono immediati, principalmente a causa delle limitate tecnologie dell'epoca. È negli ultimi anni che possiamo dire di aver avuto uno sviluppo sempre più crescente in questo campo, dove i neuroni artificiali sono diventati il nucleo fondamentale delle attuali reti neurali. È grazie a queste reti che ad oggi possiamo avere automobili con guida autonoma, miglioramenti nelle diagnosi in campo medico, chatbot e molto altro.

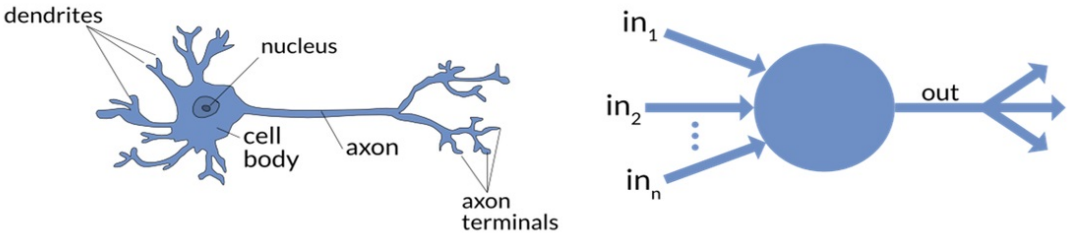


Figura 1.1: I dendriti di un neurone biologico si occupano della ricezione di segnali da altri neuroni, allo stesso modo si comporta il neurone artificiale, il quale riceve le informazioni su in_1, in_2, e in_3 e li elabora per restituirle in output.

In questa tesi verranno utilizzati modelli di reti neurali, per lo più *Convolutional neural network* (CNN), specializzate nell'elaborazione e nel riconoscimento delle immagini, dunque in grado di riconoscere situazioni di pericolo in piscina.

1.1 Motivazioni

Vari studi dimostrano come la mortalità da annegamento in piscine sia un problema diffuso. In America l'annegamento è la seconda causa principale di morte per lesioni non intenzionali tra i minori, che ha portato alla morte di 1000 giovani nel 2003. Gran parte degli annegamenti avviene in luoghi non custoditi, come piscine private, ma anche in luoghi sorvegliati come piscine pubbliche [2]. Se in quest'ultime è dimostrato che la presenza di bagnini adeguatamente preparati riduca il rischio di annegamento [2], nelle piscine private questo problema risulta più delicato e complicato da gestire.
Per sottolineare l'importanza nel trovare un'adeguata soluzione a questo problema basti pensare che la maggior parte delle morti registrate sono quelle di minori di età compresa tra gli 1 e i 4 anni [3]. Inoltre la piscina risulta essere il luogo di annegamento nel 77% dei casi, di cui il 35% in piscine residenziali [3]. Al momento le precauzioni prese per scongiurare questi incidenti consistono nell'uso di:

- barriere di sicurezza (Figura 1.2a) che se perfettamente funzionanti ed in linea con gli standard rendono l'accesso alla piscina impossibile per il minore senza l'aiuto di un adulto [4];

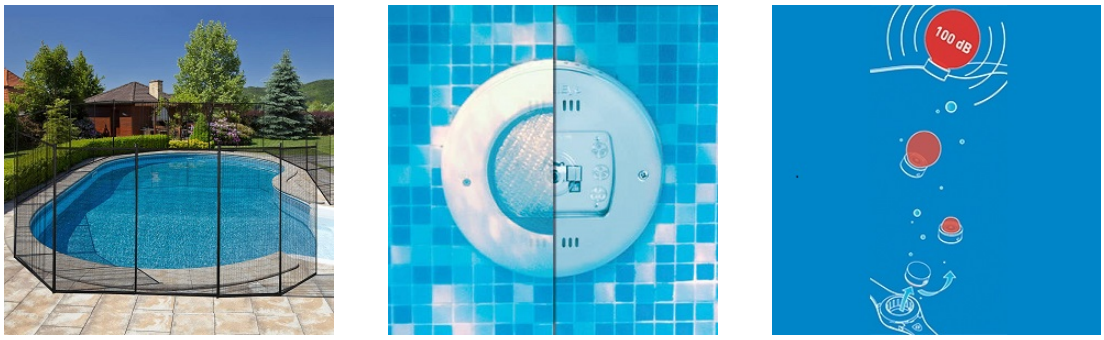
- telecamere intelligenti come Angel Eye[1] (Figura 1.2b), che, attraverso l'analisi di immagini provenienti da videocamere fisse installate sott'acqua e/o in superficie, riescono a rilevare un corpo che rimane immerso troppo a lungo, emettendo un segnale d'allarme e permettendo così alle persone addette alla sorveglianza di intervenire in modo rapido e mirato;

- altri strumenti come orologi da polso o semplici dispositivi di allarme da installare ai bordi della piscina. Il primo[2] è dotato di un sensore che rileva l'aumento di pressione in caso di affondamento e, attraverso il rilascio di un palloncino rosso (Figura 1.2c) che risale a galla, attrae l'attenzione di persone addette al salvataggio. Il dispositivo di allarme, invece, si limita a suonare dopo aver rilevato la caduta in acqua di una o più persone. Di questo, sono presenti sul mercato vari modelli.

I recenti sviluppi tecnologici e il sempre crescente impiego delle intelligenze artificiali nella vita quotidiana con la loro capacità di riconoscere e prevenire gli

[1] `https://www.angeleye.tech/`
[2] `https://www.bluefox-swiss.com`

incidenti consentono di teorizzare e realizzare ulteriori soluzioni tecnologiche attraverso l'utilizzo del *Machine Learning*, che saranno oggetto di questa tesi.



(a) (b) (c)

Figura 1.2: (a) Esempio di sistema di barriere, (b) sistema di sicurezza Angel Eye installabile al posto delle luci di una piscina, (c) funzionamento dell'orologio con rilascio del palloncino

1.2 Scopo della tesi

Questa tesi propone diversi modelli basati su *Convolutional Neural Network* (CNN) e *Support Vector Machine* (SVM) con lo scopo di riconoscere e rilevare situazioni pericolose riguardanti l'incolumità dei minori all'interno di una piscina e del loro approccio con essa con lo scopo di prevenire gli incidenti. Le CNN sono un tipo di apprendimento profondo, in generale, sono caratterizzate da diversi strati ognuno con un ruolo e funzionamento specifico all'interno della rete. Questi ultimi determinano la complessità e l'accuratezza del modello. I *Convolutional layers* sono il cuore di una CNN, responsabili dell'apprendimento analizzano in modo dettagliato le caratteristiche dell'input identificando *pattern* significativi essenziali per la comprensione e distinzione delle classi.
Le SVM sono un tipo di apprendimento supervisionato efficace per la classificazione e la regressione, in generale, il loro scopo è quello di cercare un iperpiano che separi al meglio i dati appartenenti a classi diverse. Questo iperpiano verrà scelto in modo tale da massimizzare la distanza tra i punti appartenenti a classi diverse. Gli addestramenti di questi modelli sono stati fatti con soggetti minori e non, a causa della scarsità di datasets disponibili in letteratura che contengono immagini inerenti.
Uno studio preso in esame si occupa del rilevamento della posizione del nuotatore per identificare i nuotatori sommersi e statici utilizzando un set di dati composto da video catturati sopra e sott'acqua [5]. I video descrivono 3 tipi di attività: nuotare, annegare e rimanere inattivi. La metodologia utilizzata in questo studio

è l'uso di tre architetture di *Deep Neural Network* (DNN) pre-addestrate e successivamente adattate allo scopo di riconoscimento di primi segnali di annegamento. Dagli esperimenti emergono dei dati rassicuranti, la migliore rete tra queste ha una precisione in media del 96.85% nel caso di valutazione per classificazione della scena, mentre il metodo con l'analisi della posa mediante l'utilizzo dei *keypoint detection* ha un'accuratezza anche migliore.
Partendo da questa ricerca, lo scopo della tesi è quindi quello di sviluppare un sistema intelligente basato su algoritmi di *Machine Learning* in grado di riconoscere le situazioni di pericolo in piscina, nello specifico distinguere il nuoto da casi di annegamento. Questo progetto è stato svolto con il coinvolgimento e il supporto delle aziende Ensys e Red Lynx.

1.3 Struttura della tesi

Nel secondo capitolo verranno elencate e descritte le librerie utilizzate per lo sviluppo e i dataset impiegati per l'addestramento e la fase di test.
Nel terzo capitolo verrà descritta la procedura seguita per la fase di addestramento e test dei vari modelli di CNN e SVMs, dopo una breve introduzione sul funzionamento dei modelli approfonditi in questa tesi.
Infine nell'ultimo capitolo verranno riportati i risultati ottenuti con alcune discussioni e sviluppi futuri che potrebbero apportare delle migliorie.

Capitolo 2

Metodologia

Quando si vuole utilizzare una *Convolutional Neural Network* o modelli di apprendimento supervisionato come le *Support Vector Machines* bisogna effettuare delle attività pregresse che andranno a preparare le basi per l'addestramento del modello.
Il primo passo è la scelta delle librerie che si vogliono utilizzare, tra i *framework* di *deep learning* più comuni abbiamo TensorFlow, Keras, PyTorch.
Il passo successivo da affrontare è la ricerca dei datasets, questo è fra i compiti più importanti e delicati perché andrà ad incidere direttamente sulla precisione e sull'efficienza del modello. È imperativo trovare datasets specifici per l'argomento che si vuole riconoscere in modo tale da permettere nel caso di una CNN, ad esempio, di generare dei filtri il più su misura possibile inoltre, anche la varietà dei dati è a dir poco rilevante, poiché renderà possibile il riconoscimento delle specifiche situazioni al nostro modello. Un esempio pertinente all'oggetto di studio preso in considerazione è proprio la necessità nel riuscire a far riconoscere alla rete neurale tutte le varianti possibili di pericolo di annegamento.
Successivamente il dataset di immagini deve essere preparato al meglio per facilitare l'apprendimento.

2.1 Librerie

La scelta delle librerie incide in modo significativo sull'addestramento di una CNN. Esse forniscono *Application Programming Interface* (API), un insieme di definizioni e protocolli che permettono la comunicazione tra più applicazioni. Ogni libreria è caratterizzata: dalla sua complessità di utilizzo, da diverse fun-

zionalità e dalle sue prestazioni. Seguirà una breve analisi delle librerie utilizzate nell'ambito di questa tesi.

2.1.1 Pandas

Pandas[1] è una libreria *open-source* potente e versatile per il linguaggio di programmazione Python. Avviata da Wes McKinney nel 2008 fornisce strutture di dati veloci, flessibili ed espressive, progettate per rendere facile e intuitivo il lavoro con i dati "relazionali" o "etichettati". Il suo scopo è quello di essere il blocco fondamentale di alto livello per l'analisi di dati pratici e reali in Python.
Pandas è molto utile quindi per organizzare datasets in cui le etichette ed altre informazioni vengono espresse in un file di formato CSV [6].

2.1.2 NumPy

NumPy[2] è una libreria *open-source* di Python per il calcolo numerico, che fornisce implementazioni efficienti di strutture dati di base come matrici e vettori. Fondamentale per i campi che richiedono il calcolo numerico come l'apprendimento automatico, che viene trattato in questa tesi. NumPy permetterà di rappresentare il dataset di immagini come un array NumPy contenente i pixel, in modo da eseguire le varie operazioni mantenendo alta l'efficienza [7].

2.1.3 Matplotlib

Matplotlib[3] è una libreria completa per la creazione di grafici 2D statici, animati o interattivi in Python. Introdotta nel 2002 da John Hunter si basa sugli array della libreria NumPy. L'utilizzo di questa libreria ha reso possibile la supervisione delle manipolazioni sui datasets e la visualizzazione di grafici [8].

2.1.4 TensorFlow

TensorFlow[4] è una libreria *open-source* specializzata nel *Machine Learning* e *Deep Learning* sviluppata da Google nel 2015. È indicata per la gestione di tutti gli aspetti in un sistema di apprendimento automatico grazie alla sua API di alto livello che ne facilita la costruzione e la formazione. È disponibile per diversi

[1] `https://pandas.pydata.org`
[2] `https://numpy.org/`
[3] `https://matplotlib.org/`
[4] `https://www.tensorflow.org/`

linguaggi tra cui Python.
Viene utilizzata quindi per la costruzione ed addestramento della rete neurale [9].

2.1.5 Keras

Keras[5] è una libreria *open-source* che fornisce un'interfaccia in Python per le reti neurali, basata su TensorFlow. È stata fondata da François Chollet e rilasciata nel 2015. L'obiettivo di questa libreria è quello di fornire agli sviluppatori un'API di alto livello ma di semplice utilizzo senza però perdere in prestazioni. Supporta vari tipi di modelli tra cui: reti neurali convoluzionali (CNN), reti neurali ricorrenti (RNN) e reti neurali generative avversarie (GAN). Questa libreria viene quindi ampiamente utilizzata in questo elaborato con i modelli CNN: VGG16, VGG19 e ResNet-50 e InceptionV3 [10].

2.1.6 Scikit-learn

Scikit-learn[6] è una libreria *open-source* semplice ed efficiente rilasciata nel 2007 da David Cournapeau. Fornisce una vasta gamma di algoritmi di apprendimento automatico tra cui: regressione, classificazione e clusterizzazione (Figura 2.1). Ciò che caratterizza questa libreria, oltre alla semplicità di utilizzo, è la sua ampia documentazione e predisposizione ad essere modulare, offrendo così agli sviluppatori la possibilità di combinare diversi algoritmi per creare modelli personalizzati. La libreria in questione viene utilizzata in particolare per i suoi modelli di *Support Vector Machines* (SVM) [11].

2.1.7 Scikit-image

Scikit-image [8] è una raccolta di algoritmi per l'elaborazione delle immagini, funge da integrazione a Scikit-learn. È disponibile gratuitamente e senza restrizioni dal 2014. Il suo utilizzo permette la trasformazione di datasets composti da immagini RGB in formati compatibili con la libreria Scikit-learn, nello specifico il passaggio dal formato JPEG ad array monodimensionali [12].

[5] `https://keras.io/`
[6] `https://scikit-learn.org/stable/`
[8] `https://scikit-image.org/`

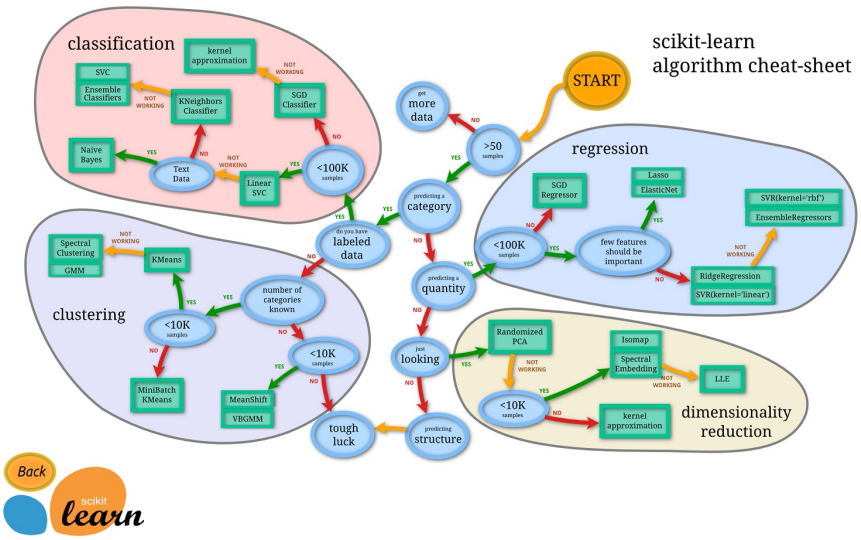


Figura 2.1: Diagramma per la scelta del giusto classificatore (figura presa dal sito di scikit-learn[7]).

2.2 Datasets

I datasets sono collezioni di diverse dimensioni di dati differenti, come: immagini, video, file audio e altri. La scelta della tipologia di dato dipende dallo studio che si vuole andare a fare ed è necessario che ogni dato appartenga ad una classe specifica tramite l'etichettatura.
Nel caso in esame vengono utilizzati datasets che contengono immagini RGB suddivise nelle classi: annegamento e nuoto.
Prima di essere utilizzato è opportuno pre-processare il dataset al fine di organizzare le classi e visionare le immagini per accertarsi che siano coerenti con l'attesa classificazione e scegliere come suddividere le immagini.

2.2.1 Drowning people Dataset

Il primo dataset è stato scaricato da Roboflow[9], una collezione *open-source* di datasets per Computer Vision e APIs. Esso è composto da 163 immagini in formato JPEG con dimensioni 500x500 pixels suddivise nelle cartelle: *train, valid e test*. Ogni cartella presenta il suo file di etichettatura il cui formato può essere scelto durante il download a seconda della libreria o metodologia utilizzata, nel caso in esame è stata scelta un'etichettatura in formato CSV, compatibile con la manipolazione tramite la libreria Pandas, che riporta le colonne: *filename, width, height, class, xmin, ymin, xmax, ymax*. Essendo Roboflow orientato alla

[9] `https://universe.roboflow.com/`

Computer Vision, ogni immagine riporta nelle colonne: *xmin, ymin, xmax, ymax* le coordinate necessarie per evidenziare l'area dell'immagine in cui, in questo caso, si trova la persona in acqua. Tali informazioni non sono state utilizzate all'interno di questa tesi.
Come detto in precedenza utilizzando la libreria Pandas per leggere il file CSV e, tramite un codice creato su misura, le immagini vengono suddivise in cartelle per classe, le classi presenti in questo dataset sono due: *drowning* e *swimming*, con rispettivamente 132 e 31 immagini.
Per comodità nei paragrafi a seguire il nome di riferimento di questo dataset sarà il suo acronimo Dp [13].

2.2.2 Swimming and Drowning Detection Dataset

Anche questo dataset è stato scaricato da Roboflow[10], presenta la stessa suddivisione in cartelle del precedente ed è stato scelto anche qui il file CSV per l'etichettatura. Presenta molte più immagini, nello specifico 12365, rendendolo più completo e vario. Il formato delle immagini è JPEG e le dimensioni 416x416 pixels.
Come il precedente, anche qui, le immagini vengono suddivise in cartelle per classe, le classi presenti sono: *Drowning* (9592), *Swimming* (2588) e *Person out of water* (185). Quest'ultima è stata eliminata per mantenere la coerenza con gli altri datasets.
È stata anche effettuata una selezione manuale di alcune immagini da eliminare poiché, essendo sempre un dataset pensato per la Computer Vision, presentavano caratteristiche incoerenti con le classificazioni senza area di interesse utilizzate in questo elaborato, dopo questa manipolazione il dataset comprende 7678 immagini. Nei paragrafi a seguire utilizzeremo l'acronimo SaD per riferirsi a questo dataset [14].

2.2.3 Water Behavior Dataset

Parte della ricerca contenuta in questo articolo utilizza Water Behavior dataset raccolti dal dipartimento di Electrical Engineering del Rochester Institute of Technology Dubai [5].
Il dataset si presenta in un formato video, diverso da quello desiderato per lo scopo, quindi ha subito delle modifiche ed elaborazioni prima di essere utilizza-

[10] `https://universe.roboflow.com/`

to. Inizialmente il dataset presenta due macrocategorie superficie e sottacqua, ognuna di queste è suddivisa in *train* e *test* che a loro volta sono suddivise nelle classi: *drowning* e *swimming*. Dopo la fase di preprocessing il dataset ha solo le cartelle train, che contiene 5709 immagini, e test, con 4020 immagini, con le classi: *drowning* (3185) e *swimming* (6544). Anche qui utilizzeremo un acronimo per riferimenti futuri all'interno della tesi che sarà WB.

2.3 Ambiente di sviluppo

L'ambiente di sviluppo utilizzato è Colaboratory di Google[11] un servizio di Jupyter Notebook in hosting che non richiede alcuna configurazione per essere utilizzato e fornisce accesso gratuito, seppure con delle limitazioni, alle risorse di calcolo tra cui: GPU, TPU e CPU.
È stato scelto poiché particolarmente adatto all'apprendimento automatico con numerose linee guida per lo sviluppo. Basato sul linguaggio Python è perfettamente integrato con TensorFlow ma anche con le altre librerie scelte, inoltre la sua struttura basata su Jupyter Notebook permette una suddivisione del lavoro con elementi testuali avanzati che possono contenere equazioni, grafici e molto altro.

[11] `https://colab.google/`

Capitolo 3

Sviluppo

3.1 Classificatori

Esistono molti tipi di classificatori, generalmente un classificatore è un algoritmo che mappa un determinato input in una specifica categoria.
Per una migliore comprensione dei modelli scelti seguirà una spiegazione dettagliata delle caratteristiche principali dei suddetti.

3.1.1 Convolutional Neural Network

Le *Convolutional Neural Network* (CNN) sono un tipo di rete neurale artificiale particolarmente adatte al riconoscimento di immagini con un'architettura costituita da strati. Segue una spiegazione dei vari strati possibili di una CNN, quelli essenziali sono:

- ***Convolutional layer***: basato sull'uso di kernel apprendibili e funzioni di attivazione. I kernel, anche detti filtri, sono generalmente piccoli nella dimensionalità spaziale ma si estendono lungo l'intera profondità dell'input. Quando l'input viene fatto passare attraverso uno strato convoluzionale entra in gioco il kernel che, applicato su tutta la dimensionalità, produce una mappa di attivazione 2D. Durante questa procedura la rete apprenderà i kernel che "si attivano" quando vedono una caratteristica specifica, imparando a riconoscerla in altri input. Lo spostamento del kernel lungo l'input è regolato dallo *stride*, passo in italiano, uno *stride* piccolo permette al kernel di catturare più caratteristiche dell'input, viceversa se lo stride è grande. Queste operazioni andranno a ridurre la dimensionalità dell'input, talvolta viene utilizzato il *padding*, in italiano imbottitura, che farà in modo di

mantenere la stessa dimensione dell'input [15].
Il processo di applicazione del kernel all'input, considerando input un vettore monodimensionale x con n elementi e il vettore kernel k con l elementi, l'operazione di convoluzione ha la formula matematica seguente:

$$z_i = \sum_{j=1}^{l} k_j x_{j+i-(l+1)/2} \tag{3.1}$$

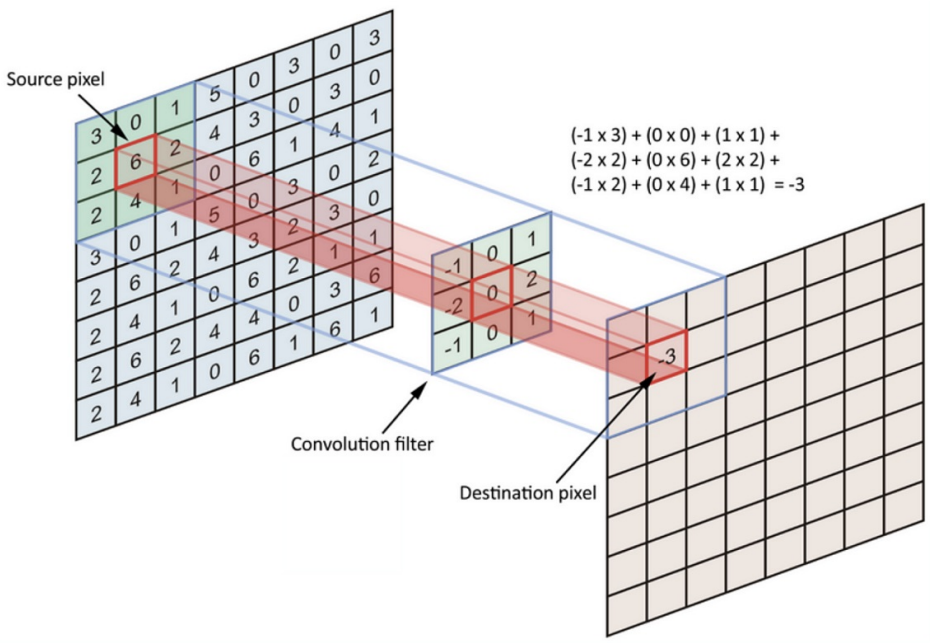


Figura 3.1: Applicazione del filtro convoluzionale al pixel di partenza (*source pixel*) per estrarne le caratteristiche locali con l'utilizzo della formula 3.1

- ***Pooling layers***: hanno lo scopo di ridurre gradualmente la dimensionalità dell'input e di conseguenza la complessità del modello. Ricevono in input una dimensionalità, ad esempio 2x2 che indica su quanti pixel viene applicato il *pooling*.
  Le metodologie più utilizzate sono:

  1. ***Max-pooling***: seleziona il valore massimo del blocco in ingresso.
  2. ***Average-pooling***: calcola la media del blocco in ingresso.
  3. ***General-pooling***: questo strato è composto da neuroni in grado di eseguire diverse operazioni tra cui la normalizzazione L1/L2, generalmente meno utilizzato [15].

- ***Fully-connected layers***: strato fondamentale della rete neurale, costituito da un insieme di neuroni completamente connessi con lo strato precedente e successivo e da una funzione di attivazione. Viene spesso messo tra gli ultimi strati poiché permette alla rete di apprendere le relazioni complesse date dai *convolutional layers*. Prende in input il numero di neuroni utilizzati e la funzione di attivazione.

Altri strati opzionali per la costruzione di una CNN, considerabili come strati di supporto sono:

- ***Dropout layer***: strato opzionale ma molto importante nelle reti complesse. Il suo compito è quello di ridurre i collegamenti che si vanno a creare durante l'addestramento della rete tramite retropropagazione, questa pratica riduce il rischio di *overfitting* obbligando la rete ad essere più robusta (Figura 3.2). Questo strato prende in input un valore decimale che indica la probabilità che un neurone venga disattivato.

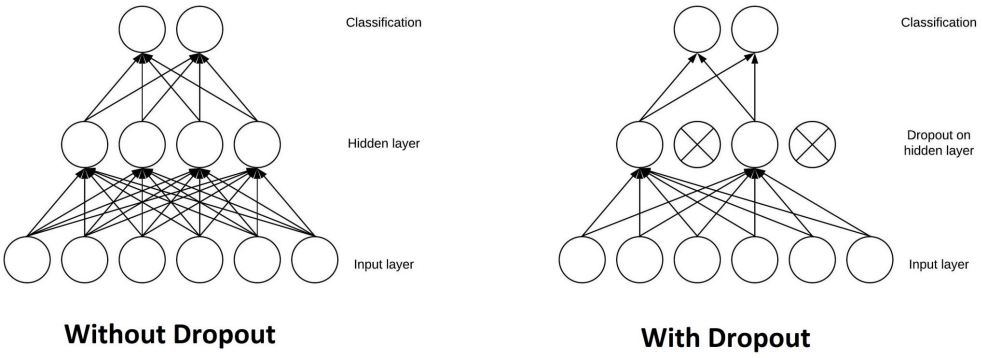


Figura 3.2: Nella figura viene riportato un esempio grafico di funzionamento del *dropout layer*, a sinistra la rete neurale con i vari collegamenti tra i neuroni, a destra la rete dopo aver applicato il *dropout layers* con i collegamenti rimossi e i neuroni disattivati.

- ***Normalization layers***: utilizzano varie tecniche per normalizzare le distribuzioni degli strati intermedi. Permette di avere gradienti più fluidi, un addestramento più veloce e una migliore accuratezza di generalizzazione. Ne esistono di più tipi, segue una breve spiegazione di quelli utilizzati per questa tesi:
  - ***BatchNormalization***: normalizza le attivazioni di un *batch* mantenendo la media degli output vicina a 0 e la deviazione standard vicina ad 1.

  - ***LayerNormalization***: a differenza del precedente viene applicato ad ogni singolo strato della rete neurale. Normalizza le attivazioni dello strato precedente per ogni esempio dato in un batch in modo indipendente, applicando una trasformazione che mantiene la media di attivazione all'interno di ogni esempio vicina a 0 e la deviazione standard di attivazione vicina a 1.

  Per i *normalization layers* non viene usato nessun input in particolare.

- ***Image augmentation layers***: modificano gli input applicando ruotazioni, traslazioni, cambi di luminosità ed altro. Utilizzati per aumentare la varietà tra gli elementi di un dataset e rendere di conseguenza il modello più resistente alle variazioni. Gli input utilizzati per questi strati sono valori numerici, interi o decimali, che indicano la misura della modifica.

- ***Flatten layer***: utilizzato per appiattire i dati, trasforma input multidimensionali in array monodimensionali. Tipicamente viene posizionato prima del *dense layer* finale.

Alcuni strati delle CNN utilizzano delle funzioni di attivazione, quelle utilizzate in questa tesi sono:

- **ReLu**: acronimo per *Rectified Linear Unit* è una funzione non lineare utilizzata con i modelli di apprendimento automatico per aggiungere non linearità. È una delle funzioni più utilizzate con le CNN ed è rappresentata dalla funzione:

$$f(x) = max(0, x) \tag{3.2}$$

  restituisce il valore in input se è maggiore di zero, altrimenti zero.

- **ReLu6**: variante della precedente (Formula 3.2) è descritta dall'equazione seguente:

$$f(x) = min(6, max(0, x)) \tag{3.3}$$

  restituisce il valore in input se maggiore di zero, altrimenti 6. È una funzione più robusta che impedisce all'output di diventare troppo grande rendendo il modello più stabile. Viene utilizzata prevalentemente con il modello InceptionV3.

- **Sigmoid**: funzione non lineare utilizzata per aggiungere non linearità ai modelli:

$$f(x) = 1/(1 + e^{-x}) \tag{3.4}$$

restituisce un valore compreso tra 0 e 1. Tale funzione è ancora molto usata nonostante sia stata superata da funzioni come ReLu e Softmax. In questa tesi viene utilizzata con tutti i modelli nello strato finale poiché indicata per dataset con due sole classi.

3.1.2 Support Vector Machines

Le *Support Vector Machines* (SVMs) sono state sviluppate da Cortes e Vapnik nel 1995 per la classificazione binaria [16]. Approccio molto popolare all'inizio degli anni 2000, le SVM presentano tre proprietà interessanti:

1. costruiscono un separatore con massimo margine, un confine di decisione con la massima distanza possibile dai punti rappresentanti le classi;

2. creano un iperpiano di separazione lineare, ma sono anche capaci di dividere dati non linearmente separabili grazie al *kernel trick*;

3. sono non parametriche, costruiscono l'iperpiano con un set di punti di esempio rendendole più flessibili. Di questi punti vengono mantenuti solo quelli più vicini al piano di separazione che diventeranno i *support vector* (Figura 3.3).

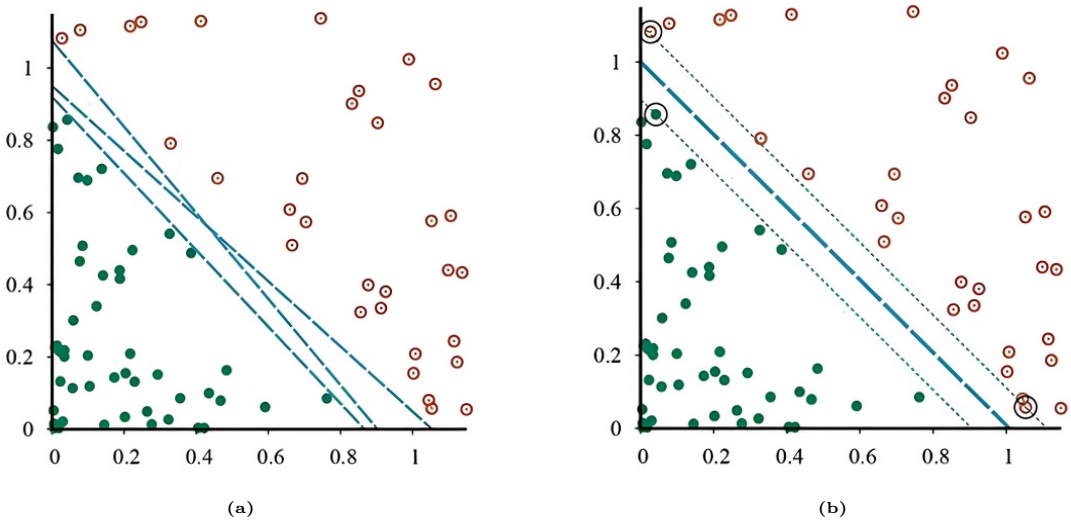


Figura 3.3: Esempio di classificazione con SVM, le classi sono rappresentate dai cerchietti vuoti e pieni. (a) Rappresentazione dei tre possibili separatori lineari. (b) Separatore con massimo margine in blu tratteggiato situato a metà del margine con cerchiati i vettori di supporto.

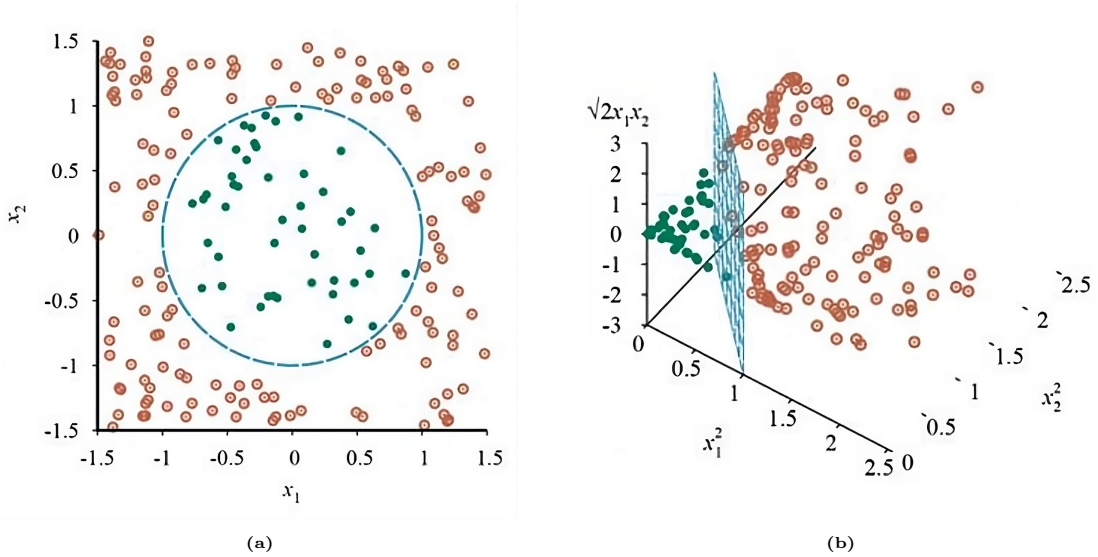


Figura 3.4: Esempio di *kernel trick*:(a) rappresentazione di un problema non risolvibile linearmente, (b) mappatura in dimensione maggiore ed individuazione del separatore.

Questo separatore è definito da un insieme di punti $\{x : w \cdot x + b = 0\}$ e la soluzione ottimale viene trovata risolvendo:

$$argmax_\alpha \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \ \alpha_k \ y_j \ y_k (x_j \cdot x_k) \tag{3.5}$$

con i vincoli $\alpha_j \geq 0$ e $\sum_j \alpha_j y_j = 0$.
Il *kernel trick* funziona così: inserendo questi kernel $K(x_j, x_k)$ nella parte finale dell'equazione 3.5, si possono trovare separatori lineari ottimi in modo efficiente all'interno di spazi di caratteristiche con miliardi di dimensioni. I separatori lineari risultanti, quando rimappati nello spazio di input di origine, possono corrispondere a confini di decisione non lineari e assai sinuosi tra gli esempi positivi e negativi, si ha così una versione dell'algoritmo definita *'kernelized'*.
Le SVM sono dei modelli ancora ampiamente utilizzati per problemi di classificazione e regressione, grazie alle loro prestazioni nonostante siano computazionalmente più semplici delle reti neurali.
A seguire dei sottocapitoli con brevi presentazioni dei vari modelli scelti.

3.1.3 VGG

Il nome completo di VGG è *Visual Geometry Group*, è una rete convoluzionale proposta dal Dipartimento di Scienza e Ingegneria dell'Università di Oxford. A partire da VGG sono stati rilasciati una serie di modelli che possono essere ap-

plicati al riconoscimento dei volti e alla classificazione delle immagini. Lo scopo originale della ricerca di VGG è capire come la profondità delle reti convoluzionali influisca sulla precisione e sull'accuratezza della classificazione e del riconoscimento delle immagini su larga scala [17].
L'ingresso voluto da questi modelli è un'immagine RGB di dimensioni 224x244. I modelli presi in esame sono VGG16 e VGG19 che, differiscono tra loro solo per il numero di strati, dove 16 e 19 indicano rispettivamente gli strati con peso nella loro architettura (Figura 3.5).
Più nello specifico:

- l'architettura VGG16 comprende 13 *convolutional layers*, 5 *max pooling layers*, 3 *fully-connected/dense layers* e ultimo strato con la funzione di attivazione Softmax.

- l'architettura VGG19 comprende 14 *convolutional layers*, 5 *max pooling layers*, 3 *fully-connected/dense layers* e ultimo strato con la funzione di attivazione Softmax.

Entrambi i modelli sono stati pre-addestrati sul dataset ImageNet[1] che contiene 1000 classi differenti ed è proprio per questo che lo strato finale softmax contiene 1000 canali, esattamente uno per classe.

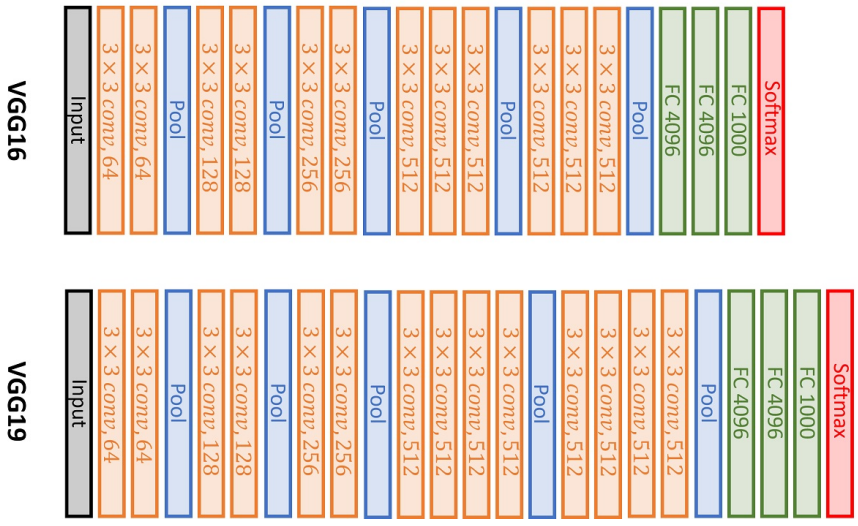


Figura 3.5: Differenza di architettura tra la VGG16 e VGG19

[1] `https://www.image-net.org/`

3.1.4 ResNet50

ResNet50 è una rete convoluzionale profonda sviluppata da un team di Microsoft Research nel 2015[18]. È una variante dell'arichettura ResNet (acronimo di *'Residual Network'*) molto popolare, la peculiarità di queste reti è la presenza di connessioni di salto, usati per permettere al modello di imparare senza soffrire della scomparsa del gradiente. Questo problema si verifica quando, durante l'addestramento di reti profonde, il gradiente diventa molto piccolo, rendendo difficile l'apprendimento.
Composta da innumerevoli strati, in generale, comprende: *convolutional layers*, blocchi di identità che contengono le connessioni di salto, *normalization layers*, *pooling*, *flatten* (Figura 3.6).

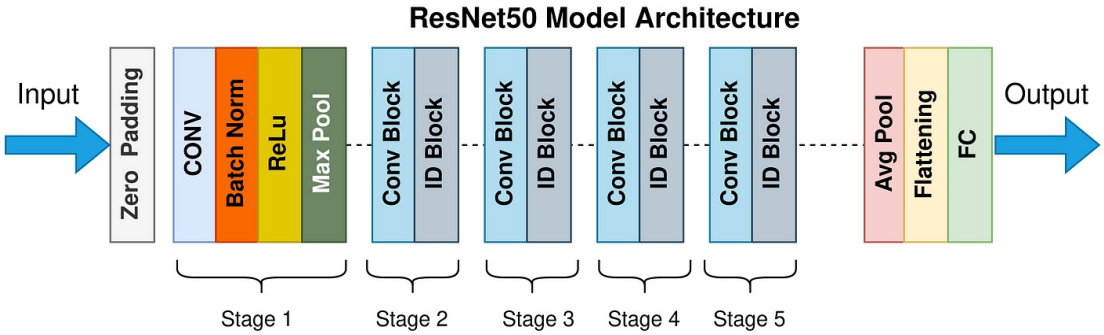


Figura 3.6: Architettura della ResNet50

3.1.5 InceptionV3

Rilasciata nel 2015 da Google Inc [19], anche conosciuta come GoogLeNet, questo modello si basa sui precedenti Inception V1-V2 apportando delle migliorie. Le caratteristiche principali di questa rete sono:

- i blocchi a cascata Inception, in cui l'uscita di un blocco funge da ingresso al blocco successivo;
- l'utilizzo dell'ottimizzatore RMSprop che, simile all'algoritmo di discesa del gradiente, limita le oscillazioni in direzione verticale aumentando la velocità di apprendimento del modello;
- la *spatial factorization* che riduce i costi computazionali delle convoluzioni e insieme all'utilizzo dello *stem module* migliora il flusso di informazioni che attraversano la rete[2];

[2] `https://www.xenonstack.com/blog/inception-architecture-computer-vision`

- *label smooting reguralization* è un metodo che regolarizza il classificatore stimando l'effetto del *label-dropout* durante l'addestramento. Impedisce al classificatore di predire una classe con troppa sicurezza, migliorando dello 0.2% il tasso di errore[3].

La sua architettura (Figura 3.7) molto complessa comprende: *convolutional layers*, *average pooling layers*, *max pooling layers*, *dropout*, *fully-connected* e Softmax. Prende in input immagini della dimensione 299x299.

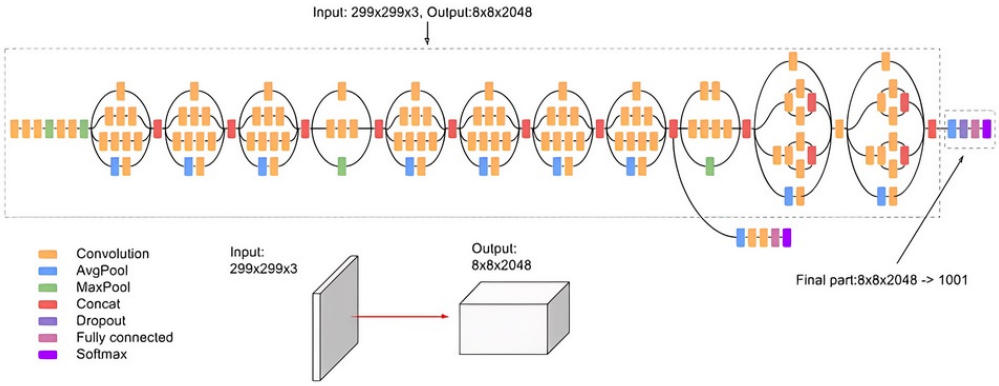


Figura 3.7: Esempio di architettura della rete InceptionV3

3.1.6 SVMs scikit-learn

Le *Support Vector Machines* (SVMs) in scikit-learn supportano vettori di campioni sia densi (numpy.ndarray) che sparsi (scipy.sparse). In questo elaborato si utilizzano campioni densi.
Ogni addestramento viene fatto con diversi i kernel e parametri C e gamma, segue une breve descrizione dei kernel:

- ***Linear Kernel***: è un kernel semplice tra i più utilizzati quando i dati sono linearmente separabili con formula: $k(x, y) = x^\top y$.
  Per questo kernel viene utilizzata la funzione LinearSVC della libreria scikit-learn poiché più efficiente su grandi dataset che richiederebbero troppo tempo altrimenti per l'addestramento.

- ***Polynomial Kernel***: è un kernel non lineare che rappresenta la similarità tra due vettori non solo sotto la stessa dimensione ma anche tra dimensioni diverse. Utilizza la seguente formula: $k(x, y) = (\gamma x^\top y + c_0)^d$

[3] `https://www.geeksforgeeks.org/inception-v2-and-v3-inception-network-versions/`

- ***Sigmoid Kernel***: similmente al precedente è un kernel non lineare che rappresenta la similarità tra due vettori, la differenza sta nella formula matematica utilizzata per fare ciò: $k(x, y) = tanh(\gamma x^\top y + c_0)$
- ***RBF Kernel***: è un kernel stazionario. Traspone i dati in uno spazio in cui la loro similarità è descritta dalla distanza euclidea, quindi due punti saranno simili se vicini tra loro. Utilizza la seguente formula: $k(x, y) = exp(-\gamma||x - y||^2)$

In sintesi la differenza di questi kernel (Figura 3.8) sta nella formula matematica utilizzata per verificare la similarità dei dati che li rende più o meno compatibili con i vari problemi di classificazione.

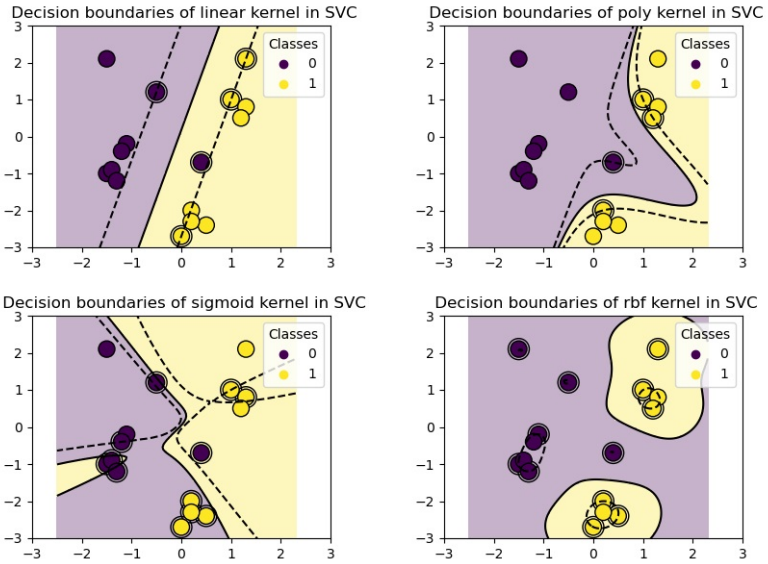


Figura 3.8: Esempio illustrativo del funzionamento dei vari kernel di una SVM (figura presa dal sito scikit-learn[4])

3.2 Training

Questa sezione è composta da un sottocapitolo per ogni dataset, quest'ultimo è a sua volta suddiviso per i modelli utilizzati con una relazione sul lavoro svolto per addestrare i suddetti, esposizione sia grafica che discorsiva dei risultati ottenuti durante l'addestramento e confronti con le reti. Si riportano gli esempi relativi agli addestramenti con migliore performance nella successiva fase di testing e/o qualche altro esempio per effettuare considerazioni tra le metodologie usate.
Di seguito, onde evitare inutili ripetizioni, un elenco delle procedure comuni negli addestramenti delle CNN:

1. **Preparazione del dataset**: suddivisione delle classi in *train/validation set* con utilizzo dello stesso *seed* (quando specificato permette di effettuare una divisione casuale e riproducibile), *batch* composti da 32 elementi, ridimensionamento dell'immagine coerentemente col modello utilizzato.

2. **Creazione istanza modello**: il modello scelto viene inizializzato con l'esclusione dello strato finale e congelamento dei precedenti per evitare un ri-addestramento.

3. **Personalizzazione**: rimuovendo lo strato finale la rete scelta diventa incapace di effettuare classificazioni, vengono aggiunti al modello diversi *layers*:

   - *convolutional*: permette alla rete di apprendere nuove *features*, caratteristiche in italiano, che vengono aggiunte al modello pre-addestrato.
   - *pooling*: utilizzato per ridurre la mappa delle *features* generata dalle convoluzioni ed alleggerire il carico di lavoro della rete;
   - *normalization*: questi strati vengono utilizzati per migliorare la generalizzazione del modello;
   - *dropout*: ulteriore strato per migliorare la generalizzazione, combatte il sovra-adattamento della rete eliminando neuroni;
   - *flatten*: solitamente posizionato prima di un *fully-connected* per appiattire le *features* e facilitarne la classificazione.
   - *fully-connected*: prende in input tutte le *features* apprese e successivamente appiattite, utilizzato per ripristinare la capacità di classificazione della rete.

   Di seguito vengono usate diverse configurazioni di strati aggiuntivi indicati con la nomenclatura V1 e V2 (Listing 3.1), questo per quanto riguarda Dp e SaD, con WB sono stati utilizzati diversi approcci che vengono approfonditi successivamente. Per ogni addestramento è stato utilizzato l'ottimizzatore Adam.

```
#Prima configurazione (V1)
model.add(layers.Flatten())
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation="sigmoid"))

```

```
#Seconda configurazione (V2)
model.add(layers.Conv2D(64, (3,3), activation="relu"))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.25))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation="sigmoid"))
```

Listing 3.1: Le due versioni di strati personalizzati utilizzati durante gli addestramenti.

Di seguito, vengono riportate le procedure comuni negli addestramenti delle SVMs:

- **Preparazione del dataset**: creazione di un dizionario che conterrà l'immagine e la sua etichetta. Ogni immagine viene ridimensionata ed appiattita per diventare un array monodimensionale Numpy. Infine il dizionario viene trasposto in due array Numpy che contengono rispettivamente le immagini e le etichette.
- **Creazione istanza modello**: inizializzazione di vari modelli SVM con diversi kernel e diversi parametri.
- **Addestramento**: ogni modello inizializzato viene addestrato sul dataset scelto.

Per ogni simulazione con le CNN è stata utilizzata la T4 GPU con 8GB di RAM dedicata e ulteriori 12GB di RAM mentre per le SVMs la CPU con 51GB di RAM, entrambe le risorse presenti su Google Colaboratory.

3.2.1 Training sul Drowning people Dataset

Sono stati addestrati tre modelli di CNN col dataset Dp: VGG16, VGG19 e ResNet50. Su questo dataset vengono svolti addestramenti con molte epoche 100-200, proprio perché ogni addestramento veniva svolto in pochi minuti, nello specifico di 2.6 minuti per un addestramento da 200 epoche. Per quanto riguarda la SVMs sono stati invece testate le SVC di scikit-learn con vari kernel e parametri C e gamma.

3.2.1.1 Risultati del training basato su VGG16

La rete VGG16 è stata inizialmente addestrata con la configurazione V1 e con epoche 10 e 20. Durante entrambi gli addestramenti la rete si comporta bene con una *loss* in calo sia nel *train* che *validation set* ed un'*accuracy* in aumento, questo però potrebbe essere sintomo di *overfitting*. Si nota qualche instabilità nel grafico dell'*accuracy* da 20 epoche (Figura 3.9).
Successivamente è stata utilizzata la configurazione V2 ed epoche: 10, 50, 100, 200 (Figura 3.10). Il modello è più stabile durante l'addestramento anche con maggiori epoche in cui migliora gradualmente raggiungendo *accuracy* del 100% e *loss* molto basse sul *train set*, questo comportamento influenza i risultati sul *validation set* dove possiamo notare un peggioramento dopo il picco sulle 40 epoche, sintomo che la rete sta imparando troppo bene i dati in input (Figura 3.10).

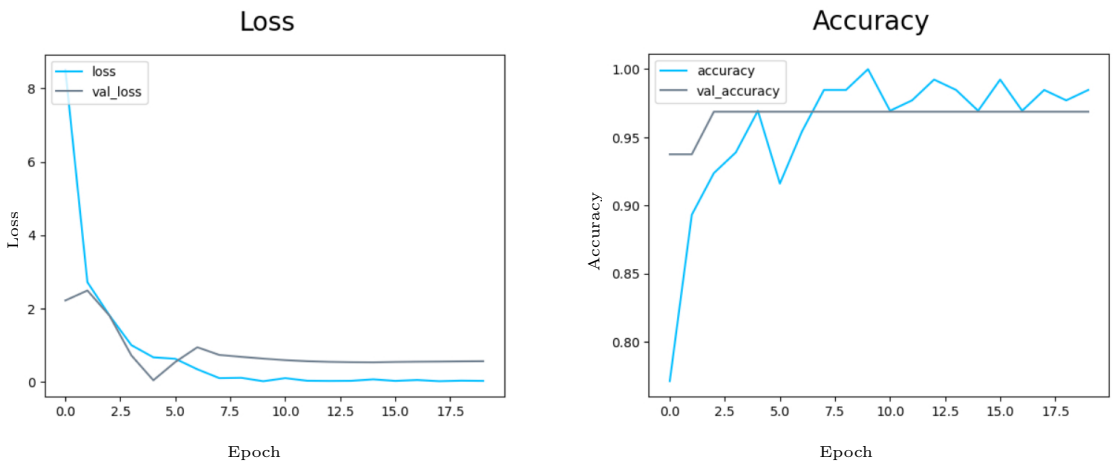


Figura 3.9: Addestramento con VGG16 20, epoche e configurazione degli strati V1 su Dp

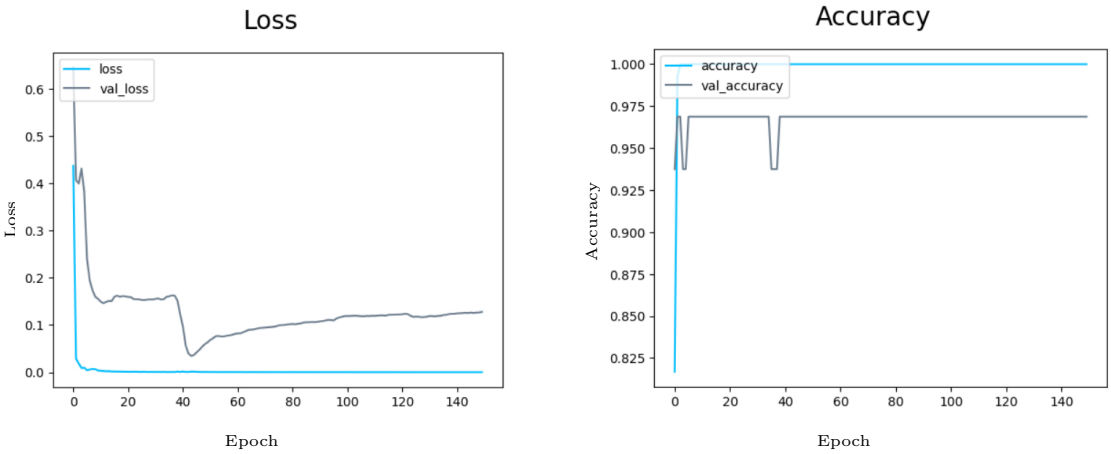


Figura 3.10: Addestramento con VGG16 200, epoche e configurazione degli strati V2 su Dp

3.2.1.2 Risultati del training basato su VGG19

La rete è stata addestrata solo con la configurazione V2, visto i miglioramenti nelle prestazioni che vedremo successivamente nella fase di testing, sarà quindi quella utilizzata da qui in avanti.
Il numero di epoche utilizzate sono: 10, 20, 50, 80, 100. Si comporta in modo simile alla precedente con qualche instabilità in meno e senza peggiorare, dopo un inizio leggermente altalenante l'addestramento risulta migliorare al trascorrere delle epoche (Figura 3.11).

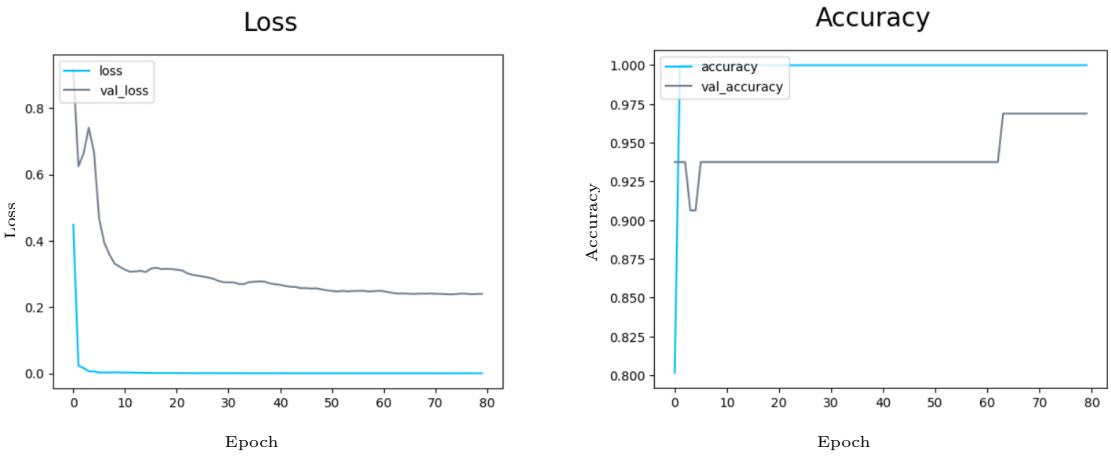


Figura 3.11: Addestramento con VGG19, 80 epoche e configurazione degli strati V2 su Dp

3.2.1.3 Risultati del training basato su ResNet50

Infine sono stati effettuati anche degli addestramenti con la rete ResNet50 e con epoche 10, 20, 50. Questa rete mostra performance ancora più stabili delle

precedenti su entrambi i set, con qualche picco. Nel complesso la *loss function* migliora all'aumentare delle epoche, mentre l'*accuracy* dopo un miglioramento iniziale si stabilizza al 100% (Figure 3.12).

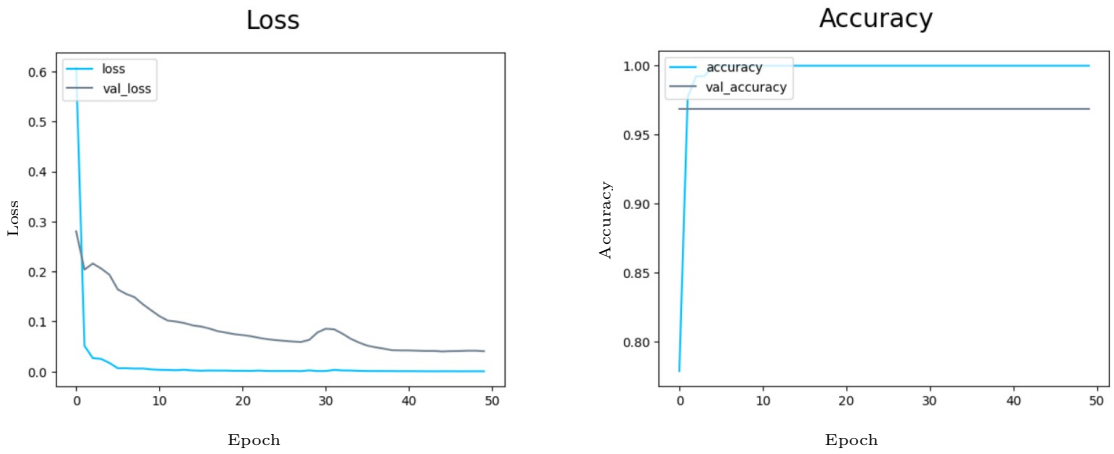


Figura 3.12: Addestramento con ResNet50 50 epoche e configurazione degli strati V2 su Dp

3.2.1.4 Risultati del training basato su SVMs

Questo dataset essendo di piccole dimensioni si è prestato ad un addestramento più dettagliato e vario, sono stati applicati vari kernel e parametri (Listing 3.2) senza dover impiegare giorni di addestramento.

```
param_grid={"C":[0.1,1,10,100],"gamma":[0.0001,0.001,0.1,1],"
    kernel":["rbf","poly", "sigmoid"]}
clf = svm.SVC(verbose = 1)
model_1 = GridSearchCV(clf,param_grid)
model_2 = svm.LinearSVC(verbose = 1)
```

Listing 3.2: Modelli di SVM inizializzati per l'addestramento con Dp.

Come per le CNN, viene riportato l'addestramento che ha dato migliori risultati in fase di testing (Tabella 3.1).

3.2.2 Training sul Swimming and Drowning Detection Dataset

Sul dataset SaD sono stati provati i modelli di CNN: VGG16, VGG19, ResNet50 e InceptionV3. Sono stati effettuati addestramenti con minor numero di epoche, visto che questo dataset è composto da molti più elementi (7678).
Anche le SVMs sono state addestrate con configurazioni più semplici per lo stesso motivo sopracitato constatando che, dopo un tentativo iniziale con stesso

	Precision	Recall	F1-Score	Support
drowning	1.00	1.00	1.00	132
swimming	1.00	1.00	1.00	31
accuracy			1.00	163
macro avg	1.00	1.00	1.00	163
weighted avg	1.00	1.00	1.00	163

Tabella 3.1: Report metriche addestramento del modello di LinearSVC con parametri C e gamma di default su Dp

addestramento eseguito su Dp, in 24 ore non era stato eseguito neanche metà del'allenamento.

3.2.2.1 Risultati del training basato su VGG16

Con SaD la rete è stata addestrata solo una volta su 10 epoche, dimostrando un andamento con leggere instabilità perlopiù sul *validation set* (Figura 3.13).

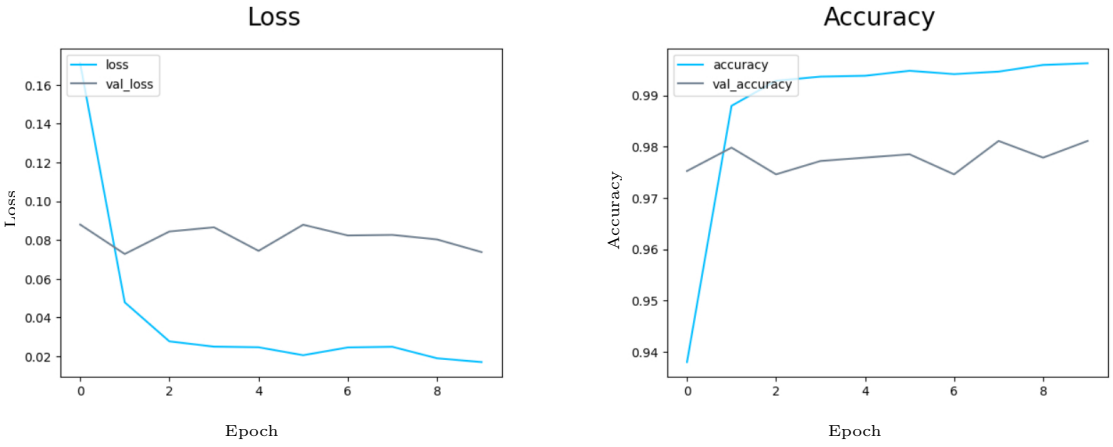


Figura 3.13: Addestramento con VGG16, 10 epoche e configurazione degli strati V2 su SaD

3.2.2.2 Risultati del training basato su VGG19

Sono stati effettuati 3 addestramenti con rispettivamente 10, 20, 50 epoche. In tutti e tre gli addestramenti l'andamento è altalentante, con diversi picchi sia con numero basso che alto di epoche soprattutto sul *validation set*. Viene riportato solo l'addestramento a 50 epoche (Figura 3.14) poiché gli altri si sono comportati in modo analogo.

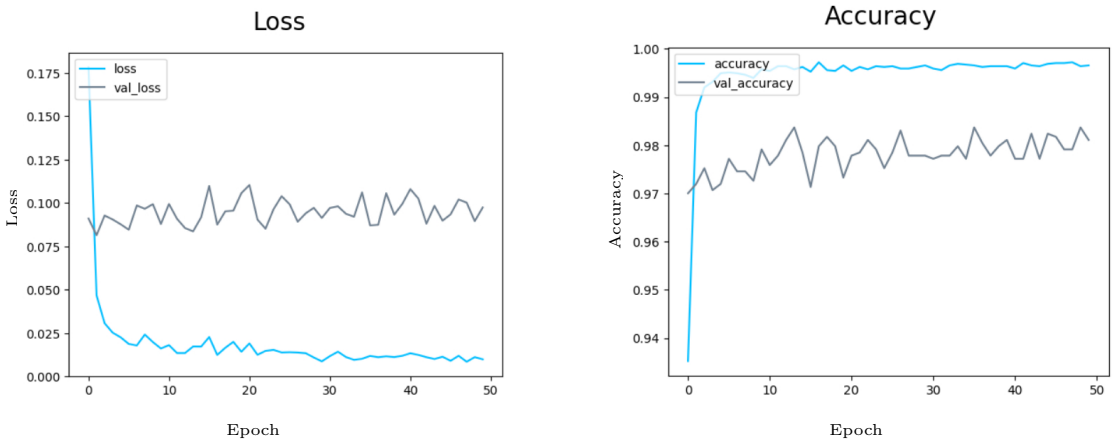


Figura 3.14: Addestramento con VGG19 50 epoche e configurazione degli strati V2 su SaD

3.2.2.3 Risultati del training basato su ResNet50

È stata addestrata con 8-10-20 epoche. Si è comportata praticamente allo stesso modo nei vari addestramenti, solo nell'ultimo dopo 17 epoche si nota un calo considerevole di *accuracy* con conseguente innalzamento della *loss* sul *validation set* (Figura 3.15).

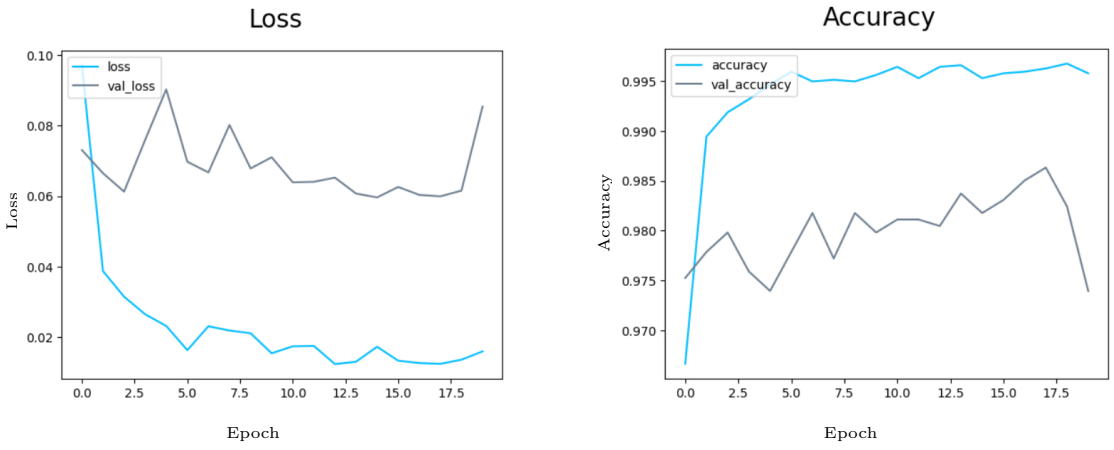


Figura 3.15: Addestramento con ResNet50 20 epoche e configurazione degli strati V2 su SaD

3.2.2.4 Risultati del training basato su InceptionV3

Questa rete è stata addestrata più a fondo, con rispettivamente 10-20-40 epoche e l'utilizzo di diverse funzioni di attivazione sul primo strato della configurazione V2: ReLu e ReLu6.

Come fatto in precedenza vengono riportati i grafici con più epoche per averne una maggiore visione, tenendo conto che il comportamento della rete è molto simile nei vari addestramenti.

- **ReLu**: l'addestramento mostra un andamento più stabile sul *train set*, ma numerosi picchi sul *validation set* (Figura 3.16).

- **ReLu6**: mostra un comportamento completamente differente, dopo qualche picco iniziale a 5 epoche si stabilizza diventando quasi costante sia sul *train* che *validation set*, senza però raggiungere risultati buoni in *accuracy* e *loss* rispetto gli altri addestramenti (Figura 3.17).

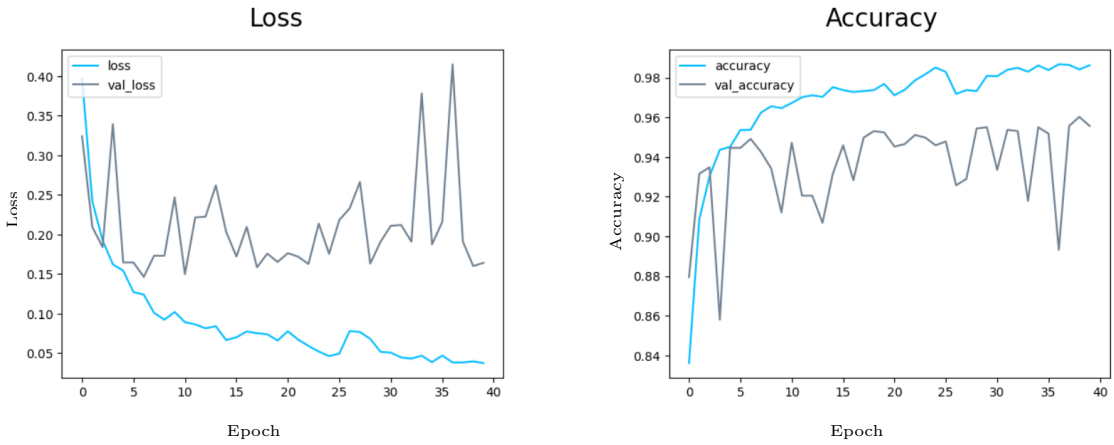


Figura 3.16: Addestramento con InceptionV3, attivazione ReLu, 40 epoche e configurazione degli strati V2 su SaD

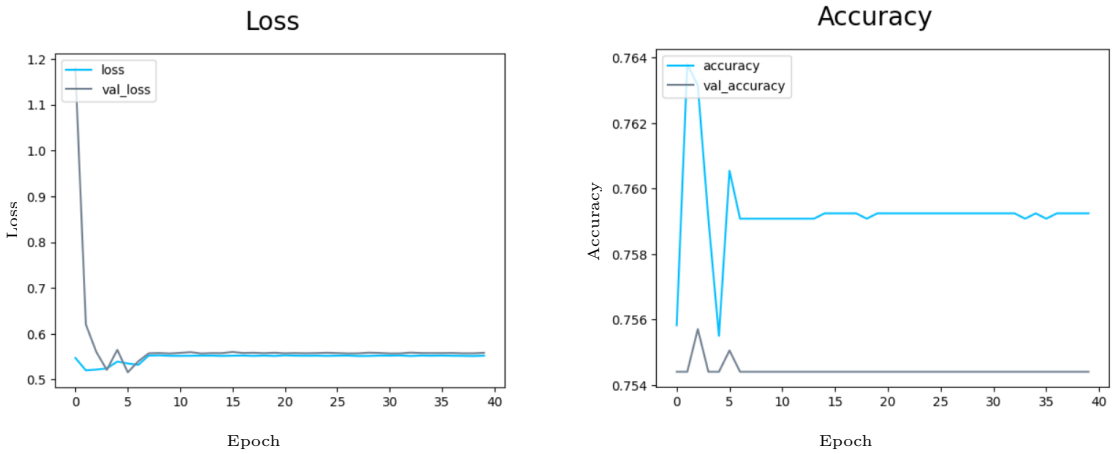


Figura 3.17: Addestramento con InceptionV3, attivazione ReLu6, 40 epoche e configurazione degli strati V2 su SaD

3.2.2.5 Risultati del training basato su SVMs

Come anticipato è stato fatto un addestramento con meno combinazioni di parametri C e gamma, vengono invece utilizzati i valori di default. Sono sempre stati

provati i vari kernel (Listing 3.3). Vengono riportate le metriche sull'addestramento con LinearSVC (Tabella 3.3).

```
model_1 = svm.LinearSVC(dual = True , random\_state= 0, tol= 1e
    -5, verbose = 1)
model_2 = svm.SVC( kernel = "rbf")
model_3 = svm.SVC(kernel = "sigmoid")
model_4 = svm.SVC(kernel = "poly")
```

Listing 3.3: Modelli di SVM inizializzati per l'addestramento con SaD.

	Precision	Recall	F1-Score	Support
drowning	1.00	1.00	1.00	5821
swimming	1.00	0.99	1.00	1857
accuracy			1.00	7678
macro avg	1.00	1.00	1.00	7678
weighted avg	1.00	1.00	1.00	7678

Tabella 3.2: Report metriche addestramento del modello di LinearSVC con parametri C e gamma di default su SaD

3.2.3 Training sul Water Behavior Dataset

Per il dataset sono stati usati i modelli delle CNN: VGG16, VGG19, ResNet50 e InceptionV3. Inizialmente per ognuno sono stati eseguiti due addestramenti con 10 e 20 epoche con la configurazione V2.
Non ci sono comportamenti particolari, gli addestramenti sulle varie epoche si comportano allo stesso modo, già dalla seconda epoca l'*accuracy* si stabilizza ad 1, il che non è un buon segno, ci suggerisce che si sta verificando *overfitting*.
Nel tentativo di ovviare a questa problematica sono state formulate diverse configurazioni in cui viene effettuato un *preprocessing* dei dati sia interno al modello che precedentemente sul dataset stesso prendendo spunto dalla documentazione Keras[5], a cui ci riferiremo rispettavamente con: *in-model, pre-model*.
Vengono quindi aggiunti numerosi *image augmentation layers* per rendere il dataset il più vario possibile (Figura 3.18), inoltre in alcuni casi è stata nuovamente utilizzata la funzione di attivazione ReLu6 al posto di ReLu con ulteriori piccole modifiche ai parametri dei vari strati rispetto V2 (Listing 3.4).

```
#Preprocessing applicato direttamente sul modello (in-model)
```

[5] https://keras.io/guides/preprocessing_layers/

```
model.add(layers.RandomFlip())
model.add(layers.RandomRotation(factor = 0.3))
model.add(layers.RandomTranslation(height_factor = (-0.3,0.3),
    width_factor=(-0.3, 0.3)))
model.add(layers.RandomZoom(height_factor=(0.2, 0.3),width_factor
    =(0.2, 0.3)))
model.add(layers.RandomBrightness(factor = -1))


# Preprocessing applicato al dataset (pre-model)
data_preprocessing = keras.Sequential(
    [
        layers.RandomFlip(seed = 666),
        layers.RandomRotation(0.3, seed = 666),
        layers.RandomTranslation(height_factor = (-0.3,0.3),
    width_factor=(-0.3, 0.3), seed = 666),
        layers.RandomBrightness(factor = -1, seed = 666),
        layers.RandomZoom(height_factor=(0.2, 0.3),width_factor
    =(0.2, 0.3))
    ]
)


# Terza configurazione (V3)
model.add(layers.Conv2D(64, (3,3), activation="relu"))# o relu6
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.40))
model.add(layers.LayerNormalization())
model.add(layers.Flatten())
model.add(layers.Dense(1, activation="sigmoid"))
```

Listing 3.4: Ogni porzione di codice deve essere considerata indipendente, ad ogni *preprocessing* viene applicato V3.

Queste due tipologie di di addestramenti con *preprocessing* non sono completamente equivalenti:

- **in-model**: nel *preprocessing* non è utilizzato alcun *seed* e il *validation set* viene ricavato dal *train set*, solo nel caso della VGG19 si utilizza un *callback* con *EarlyStopping* che monitora la *loss*;
- **pre-model**: in questo caso viene scelto un *seed*, il *validation set* viene ricavato dal test set e durante l'addestramento si utilizza un *callback* con *EarlyStopping* che monitora la *loss*.

Visto i numerosi addestramenti effettuati a seguire vengono analizzati quelli con i risultati più rilevanti per quanto riguarda le performance nella fase di *testing*.

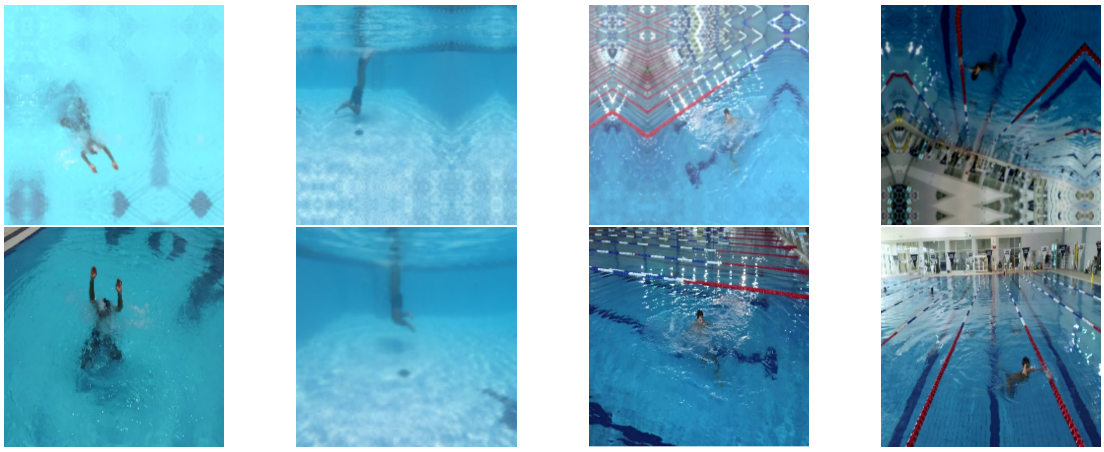


Figura 3.18: In alto nella prima riga esempio di immagini a cui è stato applicato il *preprocessing*, in basso esempio di immagini originali.

3.2.3.1 Risultati del training basato su VGG16

Gli addestramenti più significativi sono quelli con 30 epoche e funzione di attivazione ReLu, grazie ai *preprocessing* il modello non sembra andare in *overfitting*.

In-model Mantiene *accuracy* minore del 71% con *loss* decrescente per il *train set*, mentre sul *validation set* si hanno *accuracy* e *loss* migliori anche se con numerosi picchi (Figura 3.19).

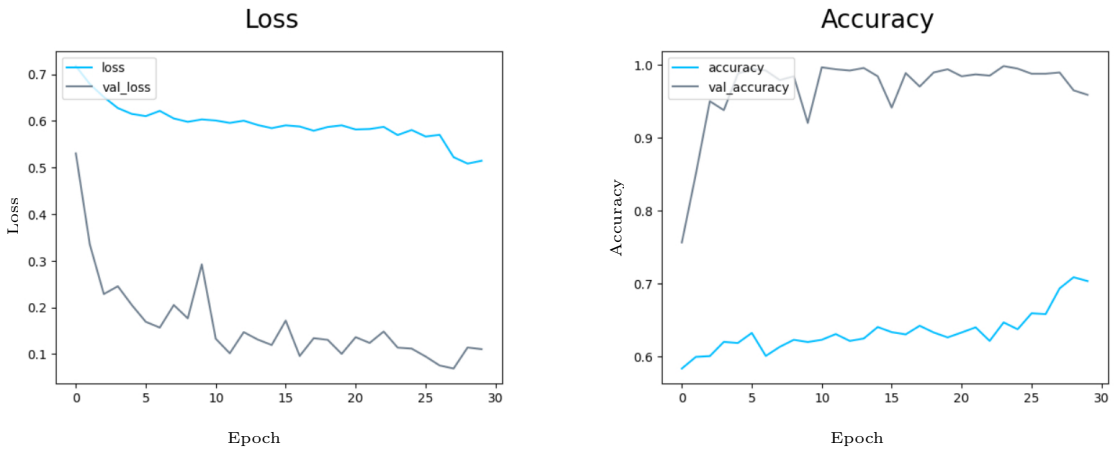


Figura 3.19: Addestramento con VGG16, attivazione ReLu, *preprocessing* in-model, 30 epoche e strati V3 su WB

Pre-model Dopo 18 epoche si attiva il *callback* arrestando così l'addestramento. Qui l'*accuracy* sul *train set* è più alta ma senza raggiungere il 100%.

Possiamo notare (Figura 3.20) che i risultati sul *validation set* sono più instabili rispetto all'esempio riportato sopra (Figura 3.19), questo può essere considerato normale visto che il modello effettua questa procedura su immagini mai viste ricavate dal training set (Figura 3.20).

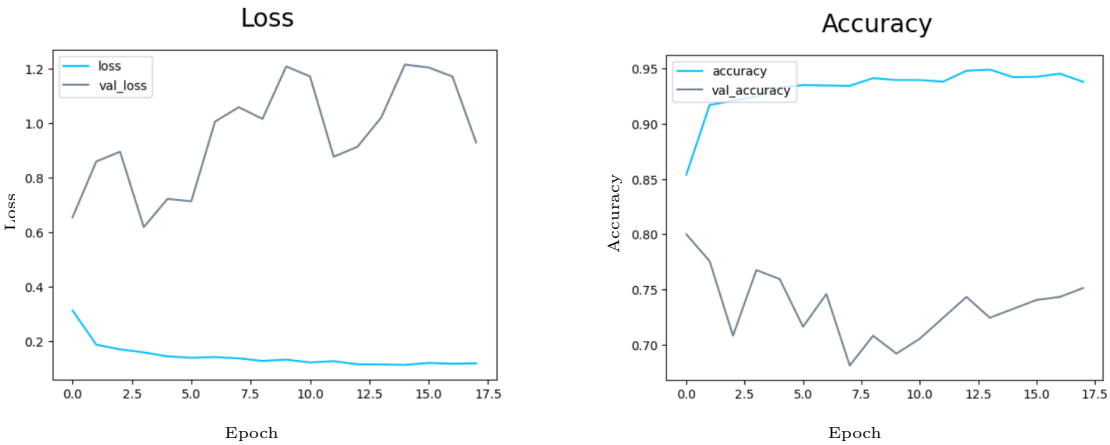


Figura 3.20: Addestramento con VGG16, attivazione ReLu, *preprocessing* pre-model, 30 epoche e strati V3 su WB

3.2.3.2 Risultati del training basato su VGG19

Gli addestramenti riportati per questo modello stati effettuati su 25 epoche: il primo con funzione di attivazione ReLu6, il secondo con ReLu.

In-model Per tutto il processo sul *train set* l'*accuracy* non riesce a superare il 62% ma la *loss* continua a diminuire, vuol dire che il modello sta imparando.

Sul *validation set* si hanno risultati buoni fin da subito, ciò è influenzato sicuramente dalla scelta di utilizzare una porzione del *train set* per l'operazione. A 12 epoche si attiva il *callback* (Figura 3.21).

Pre-model Il *callback* si è attivata a 18 epoche, anche qui l'*accuracy* non raggiunge il 100%, per quanto riguarda i grafici il comportamento è analogo alla VGG16.

3.2.3.3 Risultati del training basato su ResNet50

Con la rete ResNet50 viene riportato l'addestramento senza *preprocessing* con strati V2 e sole 10 epoche visto che tende a peggiorare all'aumentare di quest'ultime probabilmente a causa della sua architettura.

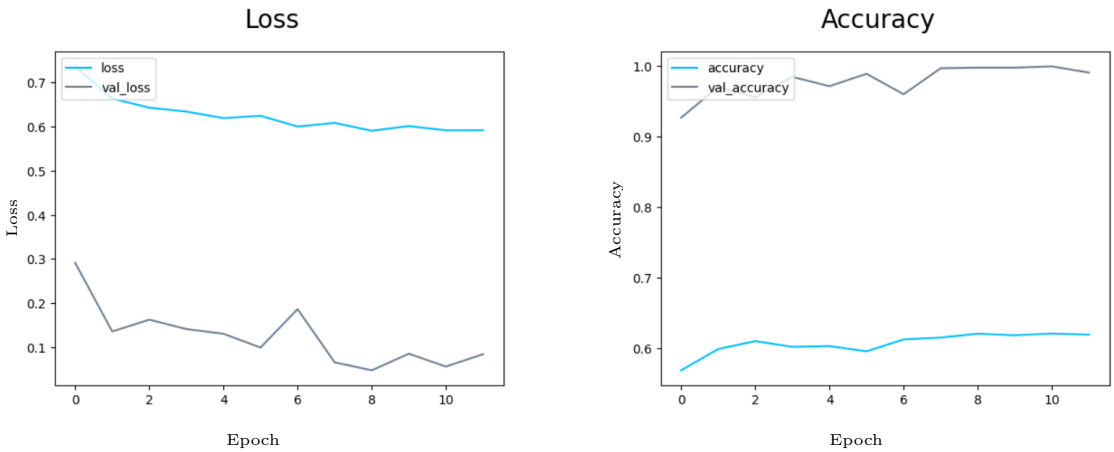


Figura 3.21: Addestramento con VGG19, attivazione ReLu6 e *preprocessing* in-model, 25 epoche e strati V3 su WB

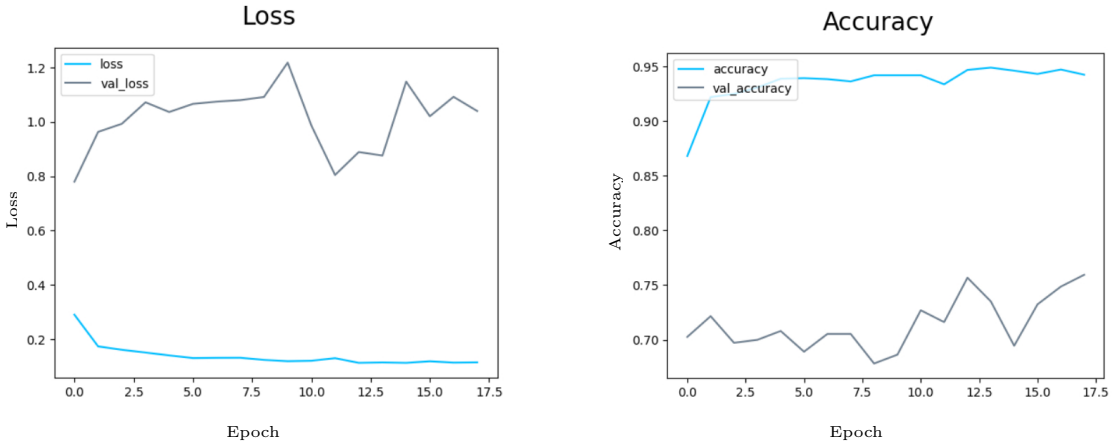


Figura 3.22: Addestramento con VGG19, attivazione ReLu e *preprocessing* pre-model, 25 epoche e strati V3 su WB

Come detto in precedenza possiamo notare che con questa configurazione la rete raggiunge subito *accuracy* massima e *loss* molto piccole dell'ordine di 10^{-5} (Figura 3.23).

3.2.3.4 Risultati del training basato su InceptionV3

Anche su questo dataset InceptionV3 si dimostra una rete con meno probabilità di *overfitting* ma difficile da addestrare. Vengono riportati gli addestramenti su 20 epoche.

In-model Viene utilizzata la funzione di attivazione ReLu6, possiamo notare un andamento molto altalenante sul *validation set* ma che nonostante i picchi migliora gradualmente, mentre abbiamo crescente miglioramento sul *train set* (Figura 3.25).

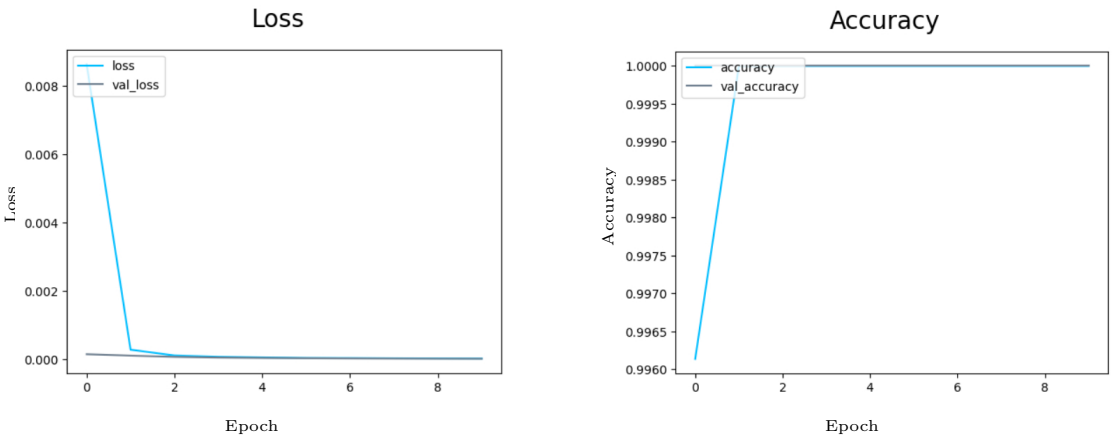


Figura 3.23: Addestramento con ResNet50, attivazione ReLu, 10 epoche e configurazione degli strati V2 su WB

Epoch

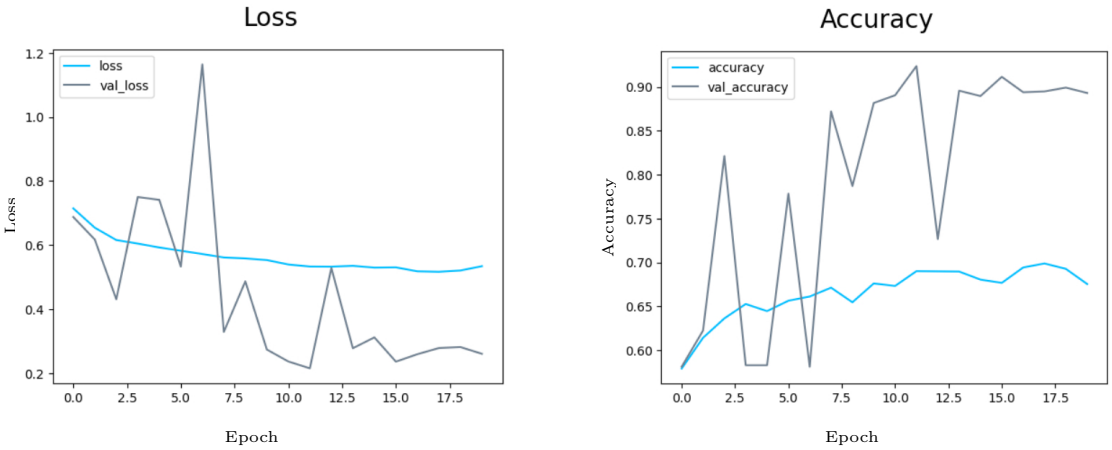


Figura 3.25: Addestramento con InceptionV3, attivazione ReLu6 e *preprocessing* in-model, 20 epoche e configurazione degli strati V3 su WB

Pre-model Viene riportato l'addestramento con funzione di attivazione ReLu, andamento abbastanza instabile sul *validation set* ma buono sul *train set* (Figura 3.26).

3.2.3.5 Risultati del training basato su SVMs

Per il dataset WB sono stati addestrate delle SVMs con diversi kernel ma parametri C e gamma di default come col dataset precedente. (Listing 3.3). Vengono riportate le metriche dell'addestramento con kernel RBF.

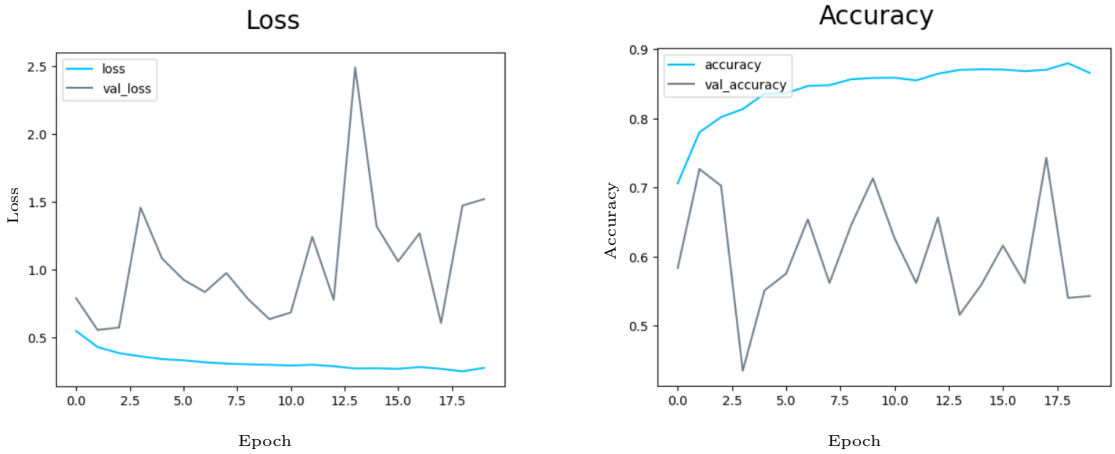


Figura 3.26: Addestramento con InceptionV3, attivazione ReLu e *preprocessing* pre-model, 20 epoche e configurazione degli strati V3 su WB

	Precision	Recall	F1-Score	Support
drowning		1.00	1.00	2440
swimming	1.00	0.99	1.00	3269
accuracy			1.00	5709
macro avg	1.00	1.00	1.00	5709
weighted avg	1.00	1.00	1.00	5709

Tabella 3.3: Report metriche addestramento del modello con kernel RBF e parametri C e gamma di default su WB

3.3 Testing

Ogni addestramento è stato seguito da una fase di testing con lo scopo di valutare le performance dei modelli su dati sconosciuti. Questa fase è essenziale per capire se il modello è in grado di generalizzare correttamente.
Ogni classificatore viene valutato su almeno un *test set*. Per quanto riguarda le CNN la metrica utilizzata è la matrice di confusione, mentre per le SVMs vengono utilizzate diverse metriche e matrice che saranno riportate in una tabella e matrice di confusione. A seguire una relazione per ogni datasets con, dove necessario, delle sottosezioni per classificatore.

3.3.1 Testing sul Drowning people Dataset

Viene utilizzato un *test set* formato da alcuni elementi presi da SaD, è stata fatta una selezione delle immagini che differivano maggiormente dal *train set* per un totale di 1226 immagini suddivise nelle classi: *drowning* (457) e *swimming* (769).

A seguire vengono rappresentati i risultati migliori in *accuracy* di ogni rete addestrata.

3.3.1.1 Risultati del testing su VGG16

Con questa rete sono stati fatti in precedenza addestramenti con configurazione di strati differenti (V1 e V2). Prendendo in esame i risultati migliori (Figura 3.27) si ottengono rispettivamente *accuracies*: 37.35% e 42.65%. Nonostante un miglioramento dell'*accuracy* del 5.3% per la configurazione V2 il modello non riesce a generalizzare bene con precisione su quest'ultimo per le classi *drowning* e *swimming* rispettivamente del: 78.11% e 21.59%.

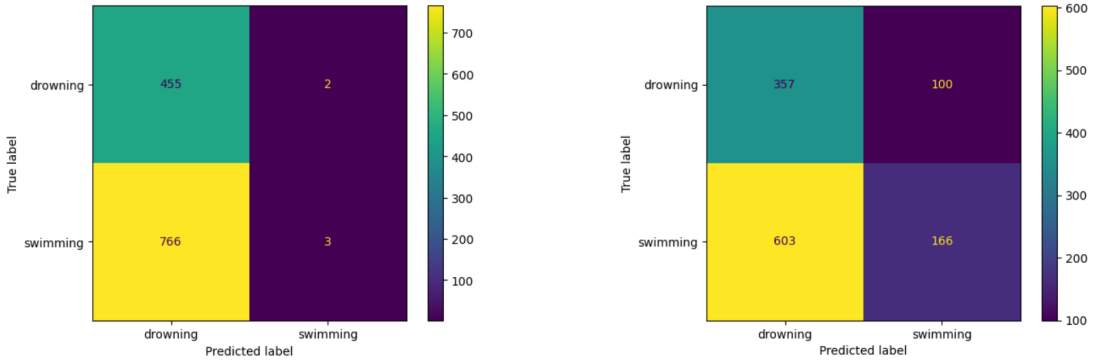


Figura 3.27: Risultati testing con matrici di confusione della rete VGG16 addestrata su Dp: (a) con 20 epoche e strati V1, (b) con 10 epoche e strati V2.

3.3.1.2 Risultati del testing su VGG19

Questa rete è stata addestrata solo con la configurazione di strati V2, nonostante i numerosi addestramenti il risultato migliore dei test (Figura 3.28) raggiunge *accuracy* del 42.33% con precisione del 75.05% su *drowning* e del 22.89% su *swimming*, il modello è incapace di generalizzare correttamente.

3.3.1.3 Risultati del testing su ResNet50

Anche la rete ResNet50 non riesce a raggiunge buone *accuracy* con solo il 39.65% nel caso migliore (Figura 3.29), con precisione di *drowning* dell'88.18% e di *swimming* del 10.79%. Questi risultati sono dovuti all'*overfitting* che si verifica nella fase di addestramento a sua volta collegato alla scarsità di varietà del dataset stesso.

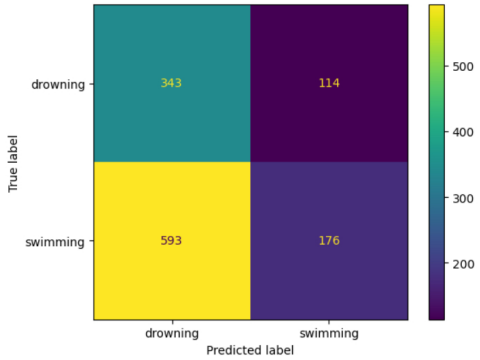


Figura 3.28: Risultati testing con matrice di confusione della rete VGG19 addestrata su Dp con 50 epoche e strati V2.

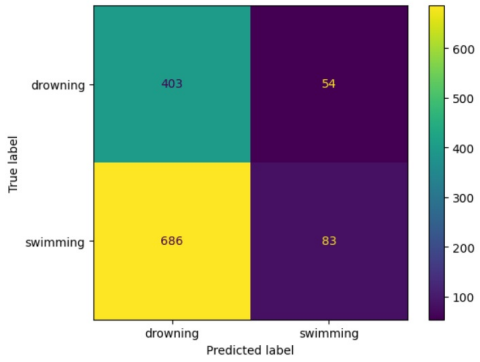


Figura 3.29: Risultati testing con matrice di confusione della rete ResNet50 addestrata su Dp con 10 epoche e strati V2.

3.3.1.4 Risultati del testing su SVMs

Anche le SVMs non raggiungono risultati soddisfacenti, nonostante i molteplici addestramenti con i diversi parametri gamma e C, non supera il 38% di *accuracy* (Figura 3.30). Questo risultato è ottenuto con il modello LinearSVC e parametri gamma e C di default (Tabella 3.4).

	Precision	Recall	F1-Score	Support
drowning	0.37	0.96	0.54	457
swimming	0.59	0.03	0.06	769
accuracy			0.38	163
macro avg	0.48	0.50	0.30	1226
weighted avg	0.51	0.38	0.24	1226

Tabella 3.4: Report metriche testing del modello di LinearSVC con parametri C e gamma di default su Dp

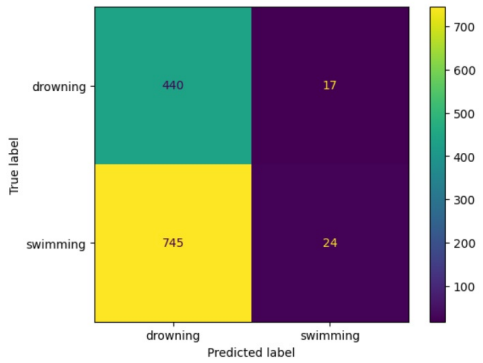


Figura 3.30: Risultati testing con matrice di confusione del modello LinearSVC con parametri gamma e C di default addestrato su Dp.

3.3.2 Testing sul Swimming and Drowning Detection Dataset

In questa fase di testing viene utilizzato Dp come *test set* con le sue 163 immagini. In generale questo dataset è stato capace di addestrare modelli che classificano in modo migliore rispetto ai test col precedente dataset, con una media *accuracy* del 71%.

3.3.2.1 Risultati del testing su VGG16

La VGG16, con un solo addestramento, ha dimostrato che su un dataset più vario e vasto può dare migliori risultati con *accuracy* del 65.64% (Figura 3.31), ma si ha precisione per le classi *drowning* e *swimming* rispettivamente: 79.54% e 0.06%.

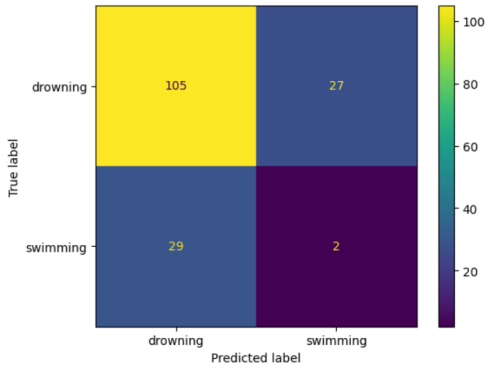
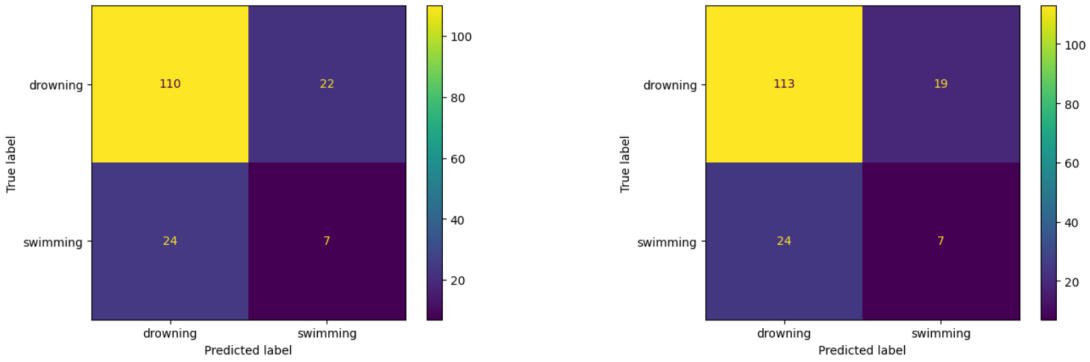


Figura 3.31: Risultati testing con matrice di confusione della rete VGG16 addestrata su SaD con 10 epoche e strati V2.

3.3.2.2 Risultati del testing su VGG19

La VGG19 ha superato le performance di VGG16 a pari epoche, raggiungendo il 71.78% di *accuracy* (Figura 3.32a) e si è leggermente superata ad epoche maggiori con *accuracy* del 73.62% (Figura 3.32b). In quest'ultima ha precisione del 85.61% su *drowning* e 22.58% su *swimming*.

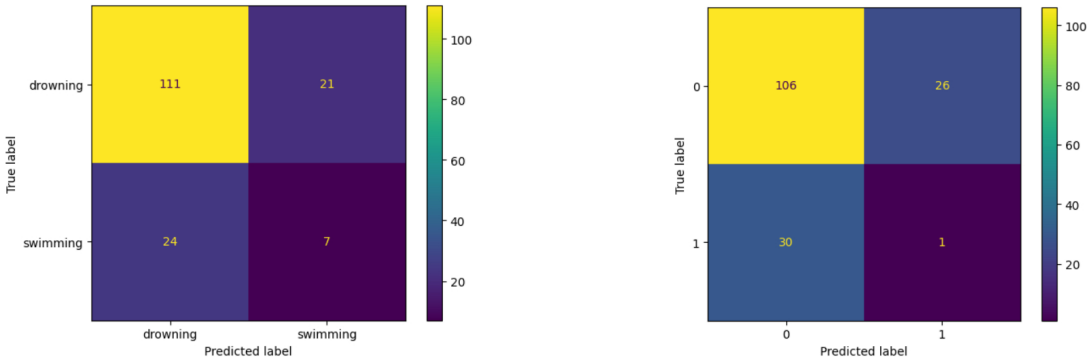


(a) (b)

Figura 3.32: Risultati testing con matrici di confusione della rete VGG19 addestrata su SaD: (a) con 10 epoche e strati V2, (b) con 20 epoche e strati V2.

3.3.2.3 Risultati del testing su ResNet50

Questa rete ha mostrato buoni risultati a sole 8 epoche con *accuracy* del 72.39% (Figura 3.33a), sfiorando la stessa precisione della VGG19 su 20 epoche, ma all'aumentare delle epoche la classificazione peggiora con un calo dell'*accuracy* a 65.65% (Figura 3.33b). Per quanto riguarda il caso migliore, di 72.39% di *accuracy*, si ha precisione su *drowning* del 84.09% e *swimming* del 22.58%.



(a) (b)

Figura 3.33: Risultati testing con matrici di confusione della rete ResNet50 addestrata su SaD: (a) con 8 epoche e strati V2, (b) con 20 epoche e strati V2.

3.3.2.4 Risultati del testing su InceptionV3

Con questa rete sono state utilizzate due funzioni di attivazione come precedentemente specificato (vedi Sezione 3.2.2.4). La rete InceptionV3 con la funzione di attivazione ReLu ha portato a migliori risultati con 10 epoche con *accuracy* del 73% (Figura 3.34a), di questa la precisione corrisponde a 86.36% per *drowning* e 16.13% per *swimming*, mentre nella rete addestrata con ReLu6 le predizioni sulle varie epoche vengono sempre classificate con la classe *drowning*, il modello non è in grado di distinguere le due classi (Figura 3.34b).

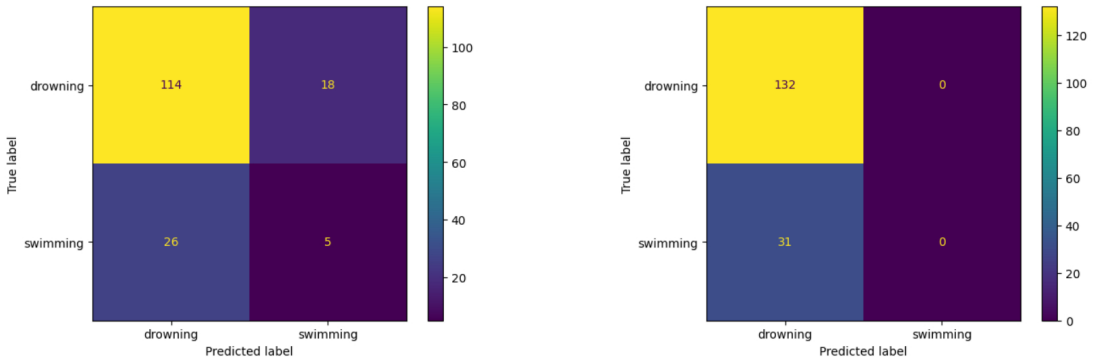


Figura 3.34: Risultati testing con matrici di confusione della rete InceptionV3 addestrata su SaD: (a) con 10 epoche e strati V2 e funzione di attivazione ReLu, (b) con 10 epoche e strati V2 e funzione di attivazione ReLu6.

3.3.2.5 Risultati del testing su SVMs

Quasi tutte le SVC si comportano bene con *accuracies* che variano dal 69% al 87%, il modello LinearSVC ha ottenuto il risultato migliore con 87% di *accuracy*. Di seguito le metriche (Tabella 3.5) e rispettiva matrice di confusione (Figura 3.35).

	Precision	Recall	F1-Score	Support
drowning	0.91	0.93	0.92	132
swimming	0.68	0.61	0.64	31
accuracy			0.87	163
macro avg	0.79	0.77	0.78	163
weighted avg	0.87	0.87	0.87	163

Tabella 3.5: Report metriche testing del modello di LinearSVC con parametri C e gamma di default su SaD

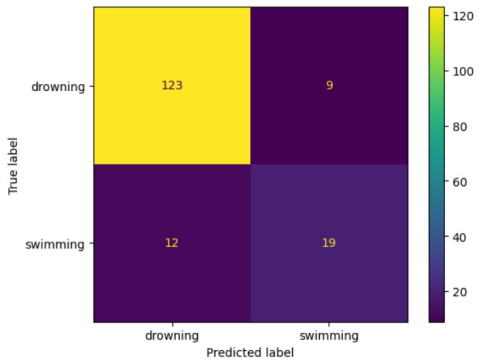


Figura 3.35: Risultati testing con matrice di confusione del modello LinearSVC con parametri gamma e C di default addestrato su Dp.

3.3.3 Testing sul Water Behavior Dataset

I primi addestramenti con gli strati V2 e senza gli *image augmentation layers* nella fase di test hanno mostrato risultati, talvolta, anche molto differenti a seconda del modello con *accuracy* media tra tutte le reti del 45.25%, i risultati migliori si hanno su ResNet50 e i peggiori su InceptionV3 con rispettivamente *accuracy* del 54% e 24%.
Per quanto riguarda invece gli addestramenti con il *preprocessing*, i *test set* utilizzati sono 2, composti da:

1. **t1**: una parte di WB stesso che non è stata usata nell'addestramento dove:

   - per i test con *in-model* contiene tutte le 4020 immagini, circa il 41.5% delle immagini totali.
   - per i test con *pre-model* il set ha subito delle modifiche nella classe *swimming* per equilibrarla con la classe *drowning*. Le rimozioni sono state fatte in modo omogeneo per ogni set di immagini ricavati dai vari video, fino a contenere 1106 immagini per *swimming* e 745 per *drowning*.

2. **t2**: alcune immagini selezionate dal dataset SaD che corrispondono a circa il 14% del dataset totale.

Nei successivi paragrafi sarà seguito quest'ordine per elencare i risultati e si farà riferimento agli stessi mediante gli acronimi, inoltre verrà adottata la struttura utilizzata nel *training* per distinguere i due *preprocessing*.

3.3.3.1 Risultati del testing su VGG16

Come nei precedenti test la VGG16 si dimostra una rete capace di imparare meglio il dataset su cui si sta addestrando a scapito della generalizzazione.

In-model In generale le matrici di confusione sono mediocri su t1 (64.32%) e peggiori su t2 (46.74%). Per quanto riguarda la precisione delle classi *drowning* e *swimming*, su t1 si hanno rispettivamente 29.93% e 72.15, mentre su t2 66.52% e 34.98% (Figura 3.36).

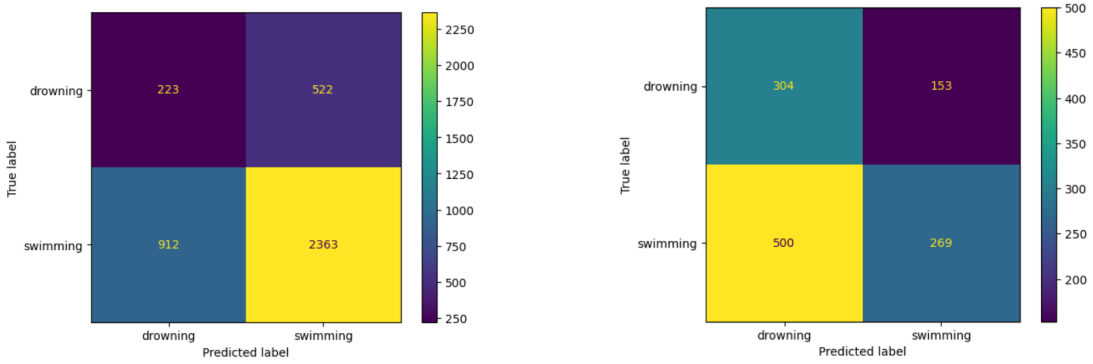


Figura 3.36: Risultati testing con matrici di confusione della rete VGG16 addestrata su 30 epoche con *preprocessing* in-model, funzione di attivazione ReLu e strati V3 su WB: (a) predizioni su t1 (b) predizioni su t2.

Pre-model Per questo modello t1 è stato modificato come spiegato in precedenza. Le matrici di confusione si mostrano più coerenti su t1 (50.13%) e t2 (47.8%). Le precisioni per le rispettive classi *drowning* e *swimming* sono: 54.63% e 47.11% su t1, 52.74% e 44.86% su t2 (Figura 3.37).

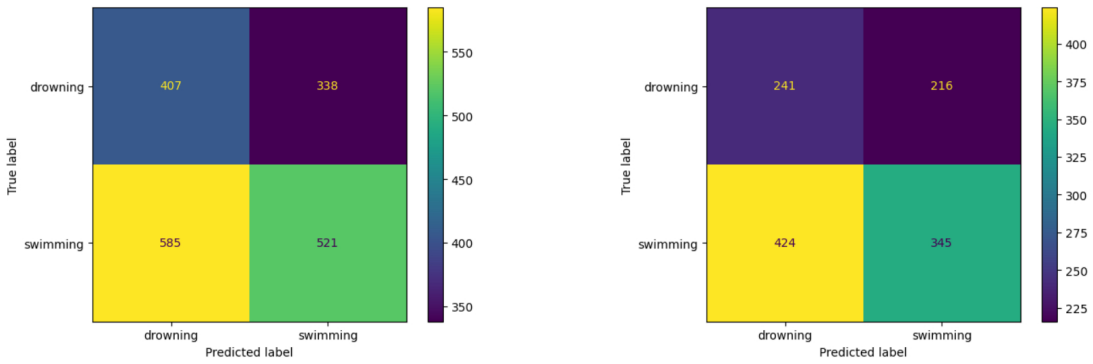


Figura 3.37: Risultati testing con matrici di confusione della rete VGG16 addestrata su 30 epoche con *preprocessing* pre-model, funzione di attivazione ReLu e strati V3 su WB: (a) predizioni su t1 (b) predizioni su t2.

3.3.3.2 Risultati del testing su VGG19

Questo modello cerca di essere migliore della VGG16 in quanto a generalizzazione, infatti a scapito della precisione su t1 guadagna su t2. Comunque i risultati sono mediocri con *accuracy* che non supera il 60.17% su t1 e 58.08% su t2.

In-model Si è ottenuta la matrice di confusione migliore sull'addestramento di 25 epoche e funzione di attivazione ReLu6 con prestazioni molto simili su t1 e t2 (Figura 3.38), con rispettivamente *accuracy* del 60.17% e 58.08%. Mentre le precisioni mostrano qualche differenza in più di risultati tra i due *test set*, prendendo le classi nell'ordine *drowning* e *swimming* si hanno precisioni del: 34.36% e 66.05% su t1, 14.66% e 83.88% su t2.

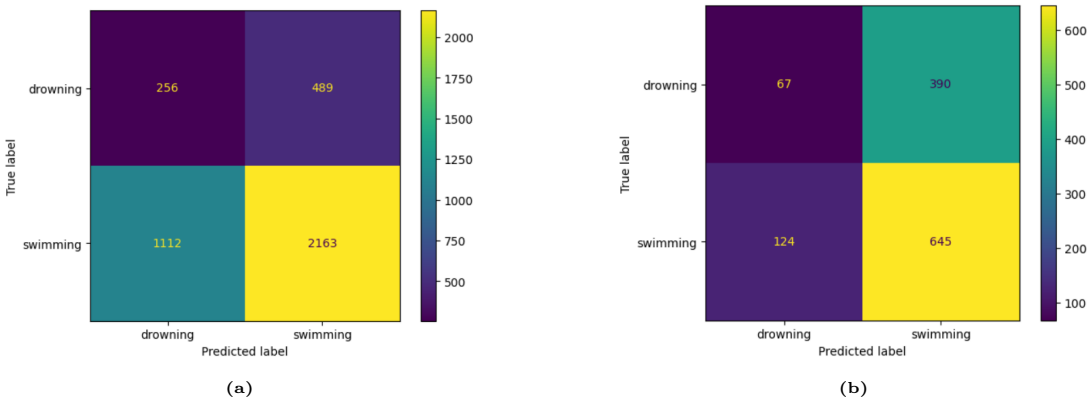
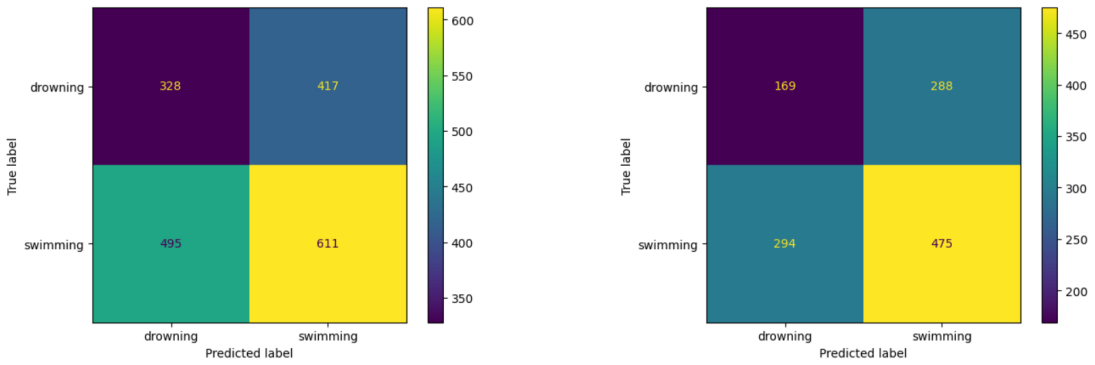


Figura 3.38: Risultati testing con matrici di confusione della rete VGG19 addestrata su 25 epoche con *preprocessing* in-model, funzione di attivazione ReLu6 e strati V3 su WB: (a) predizioni su t1 (b) predizioni su t2.

Pre-model Anche per questo caso viene preso in esempio il caso con 25 epoche ma funzione di attivazione ReLu, ancora una volta si sono ottenute prestazioni equivalenti su t1 e t2 (Figura 3.39), rispettivamente 50.73% e 52.53% di *accuracy*, anche la precisione sulle classi è simile per i due **test set**, prendendo le classi nel solito ordine si hanno: 44.03% e 55.24% su t1, 36.98% e 61.77% su t2.

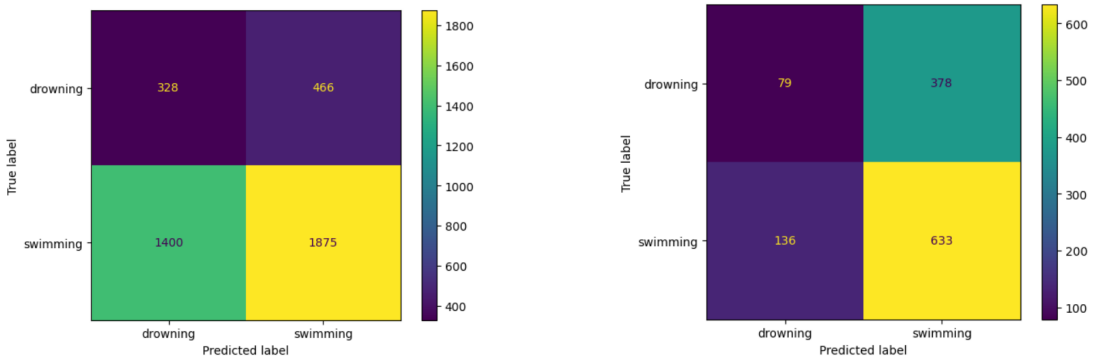
3.3.3.3 Risultati del testing su ResNet50

ResNet50 ha ottenuto risultati simili alle precedenti reti anche senza il *preprocessing*, di seguito la matrice di confusione su t1 e t2, con le rispettive *accuracy* del 54.14% e 58.08%. Questa rete sulle classi ha precisione del: 19.46% e 83.42% su t1, 39.17% e 59.95% su t2 (Figura 3.40).



(a) (b)

Figura 3.39: Risultati testing con matrici di confusione della rete VGG19 addestrata su 25 epoche con *preprocessing* pre-model, funzione di attivazione ReLu e strati V3 su WB: (a) predizioni su t1 (b) predizioni su t2.



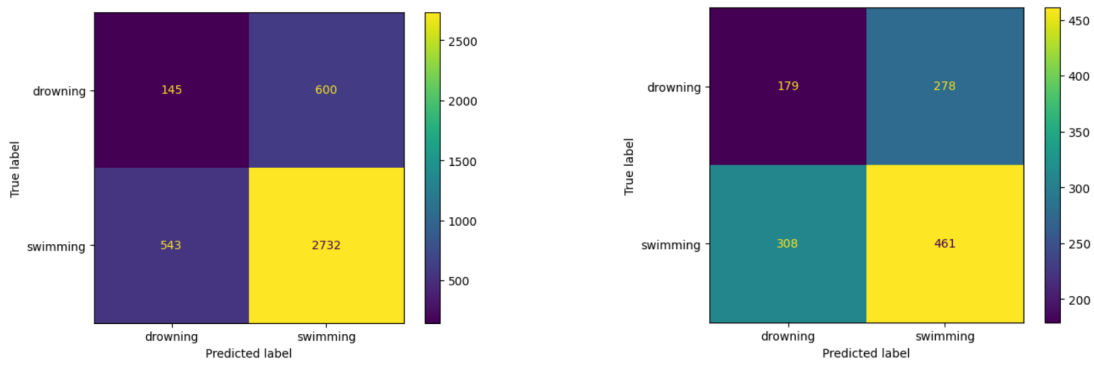
(a) (b)

Figura 3.40: Risultati testing con matrici di confusione della rete ResNet50 addestrata su 10 epoche, funzione di attivazione ReLu e strati V2 su WB: (a) predizioni su t1 (b) predizioni su t2.

3.3.3.4 Risultati del testing su InceptionV3

Tra tutti i test effettuati InceptionV3 ha generato matrici di confusione molto differenti tra loro, talvolta con classificazioni buone e talvolta con classificazioni peggiori anche dei modelli precedenti.

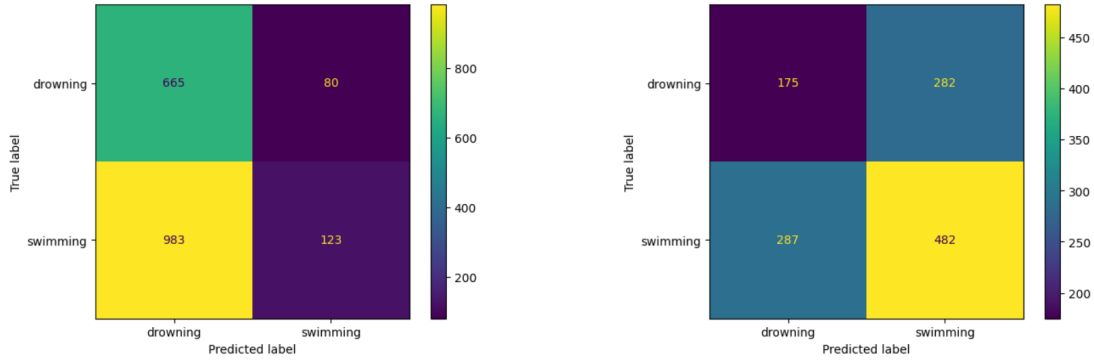
In-model Si ha la matrice di confusione migliore tra tutte le reti con il 71.57% di *accuracy* per quanto riguarda t1, mentre risultati nella media su t2 con 52.20% (Figura 3.41). Nonostante questi risultati però la precisione ci mostra che su t1 il modello predice per lo più la classe *swimming* con 83.42% di precisione e 19.46% su *drowning*, questo alza l'*accuracy* senza però rendere il modello buono su entrambe le classi. Su t2 invece si ha precisione del: 39.17% e 58.95% per *drowning* e *swimming*.



(a) (b)

Figura 3.41: Risultati testing con matrici di confusione della rete InceptionV3 addestrata su 20 epoche con *preprocessing* in-model, funzione di attivazione ReLu6 e strati V3 su WB: (a) predizioni su t1 (b) predizioni su t2.

Pre-model In questo test, purtroppo, la rete non ha restituito buone performance con *accuracy* del 42.57% su t1 e 53.59% su t2 (Figura 3.42). Mentre si ha precisione delle classi *drowning* e *swimming* del: 89.26% e 11.12% su t1, 38.29% e 62.68% su t2.



(a) (b)

Figura 3.42: Risultati testing con matrici di confusione della rete InceptionV3 addestrata su 20 epoche con *preprocessing* pre-model, funzione di attivazione ReLu e strati V3 su WB: (a) predizioni su t1 (b) predizioni su t2.

3.3.3.5 Risultati del testing su SVMs

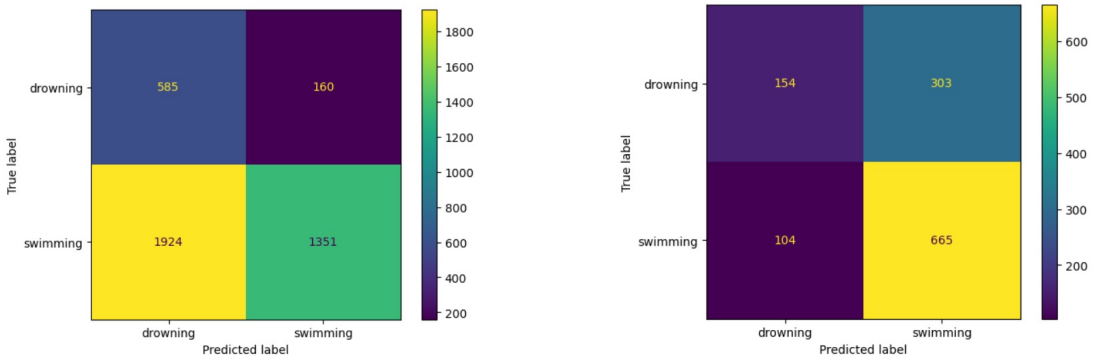
Su questo dataset purtroppo le SVMs di scikit-learn non riescono ad ottenere buoni risultati per quanto riguarda t1, nonostante le varie configurazioni utilizzate l'*accuracy* non supera mai il 48%, questo risultato lo otteniamo con il kernel RBF. Mentre su t2 si ha il risultato migliore con *accuracy* del 67% e LinearSVC. Di seguito vengono esposti i risultati delle metriche migliori per entrambi i *test set* (Tabelle 3.6-3.7) e rispettive matrici di confusione (Figure 3.43a-3.43b).

	Precision	Recall	F1-Score	Support
drowning	0.23	0.79	0.36	745
swimming	0.89	0.41	0.56	3275
accuracy			0.48	4020
macro avg	0.56	0.60	0.46	4020
weighted avg	0.77	0.48	0.53	4020

Tabella 3.6: Report metriche del testing su t1, SVM con kernel RBF e parametri C, gamma di default su WB

	Precision	Recall	F1-Score	Support
drowning	0.60	0.34	0.43	457
swimming	0.69	0.86	0.77	769
accuracy			0.67	1226
macro avg	0.64	0.60	0.60	1226
weighted avg	0.65	0.67	0.64	1226

Tabella 3.7: Report metriche del testing su t2, LinearSVC e parametri C, gamma di default su WB



(a) (b)

Figura 3.43: Risultati testing con matrici di confusione dei modelli SVMs su WB: (a) predizioni su t1 del modello SVM con kernel RBF, (b) predizioni su t2 del modello LinearSVC.

Capitolo 4

Conclusioni

4.1 Discussione dei risultati

Durante tutta la fase di addestramento e testing sui vari datasets si può notare che le CNN hanno ottenuto in generale risultati migliori rispetto le SVMs.
Solo sul dataset SaD i vari modelli di SVMs sono riusciti a eguagliare con risultati incoraggianti, 87% di *accuracy*, ma su tutti gli altri dataset sono stati al di sotto del 48%. È un peccato dover scartare questi modelli poiché possono essere implementati su qualsiasi hardware, come un qualsiasi computer o notebook. Il problema affrontato in questa tesi però, richiede che vengano garantiti alti standard visto che si parla di sicurezza, quindi performance tendenti al 100%.
Per quanto riguarda le CNN, invece, i risultati sono migliori rispetto alle SVMs, con le *accuracies* migliori su SaD del 73.62% e su WB del 71.57%. Tralasciando il dataset Dp, che per questioni intrinseche al dataset stesso non ha superato il 42.65%, analizzando più nel dettaglio gli altri datasets:

- con SaD abbiamo la media di *accuracy* del 71.16% (Tabella 4.1), però facendo i calcoli della precisione per ogni classe i risultati sono meno confortanti. Il modello è poco capace di distinguere la classe *swimming* con una precisione media del 16.93 (Tabella 4.1), questo può dipendere in parte dai pochi elementi della suddetta nel test set, i cui vi sono rispettivamente: 132 immagini per *drowning* e 31 per *swimming*;

- con WB sono stati effettuati addestramenti e test di due tipologie diverse:

  - **In-model**: ottiene una media *accuracy* di 57.33%.
    Andando più nel dettaglio sulle varie precisioni delle classi per t1 e t2

possiamo notare classificazioni contrastanti con la VGG16, dove ottiene sulla classe *drowning* il 29.93% di precisione con t1 ma 66.52% con t2. VGG19 e InceptionV3 si rivelano più coerenti rispetto a VGG16 con precisioni sulle classi simili sia su t1 che t2, ad esempio con VGG19 con la classe *drowning* su t1 si ha una precisione del 34.36% e su t2 del 14.66% (Tabella 4.2).

– **Pre-model**: nonostante la media *accuracy* più bassa, 50.3%, i vari modelli sono molto più equilibrati nella classificazione. Con il WB dataset, VGG16 che VGG19 mantengono la coerenza su t1 e t2, dove per VGG19 si hanno precisione su *swimming* del 47.11% su t1 e 44.86% su t2, sia l'altra classe che VGG16 hanno precisione analoga. non si può dire lo stesso di InceptionV3 (Tabella 4.3).

SaD Testing			
Classifier	Accuracy(%)	Drowning(%)	Swimming(%)
VGG16	65.64	79.54	6.45
VGG19	73.62	85.61	22.58
ResNet50	72.39	84.09	22.58
InceptionV3	73	86.36	16.13

Tabella 4.1: Risultati *accuracy* totale e precisione specifica per classe sul dataset SaD

WB Testing (In-model)				
Classifier	Accuracy(%)	Drowning(%)	Swimming(%)	Test set
VGG16	64.33	29.93	72.15	t1
	46.74	66.52	34.98	t2
VGG19	60.17	34.36	66.05	t1
	58.08	14.66	83.88	t2
InceptionV3	71.57	19.46	83.42	t1
	52.20	39.17	59.95	t2

Tabella 4.2: Risultati *accuracy* totale e precisione specifica per classe sul dataset WB con la metodologia in-model

Basandoci solo sull'*accuracy* SaD, ottiene i risultati migliori di WB.
Anche WB ha del potenziale, il problema di questo dataset è stata la varietà più che la quantità. Durante le varie fasi di addestramento sono stati fatti più tentativi con diversi *image augmentation*. Tuttavia ulteriori test sono necessari in futuro per garantire performance ottimali maggiori del 90%.
Anche se inizialmente il progetto era quello di effettuare una classificazione specifica di minori in situazioni pericolose, i risultati presentati in questa tesi sug-

WB Testing (Pre-model)				
Classifier	Accuracy(%)	Drowning(%)	Swimming(%)	Test set
VGG16	50.14	54.63	47.11	t1
	47.80	52.74	44.86	t2
VGG19	50.73	44.03	55.24	t1
	52.53	36.98	61.77	t2
InceptionV3	42.57	89.26	11.12	t1
	53.59	38.29	62.68	t2

Tabella 4.3: Risultati *accuracy* totale e precisione specifica per classe sul dataset WB con la metodologia pre-model

geriscono che questa specifica non è strettamente necessaria per questo tipo di classificazione. Tuttavia ulteriori test sono necessari in futuro con più dati per confermare questa ipotesi.

4.2 Sviluppi futuri

Un approccio alternativo che potrebbe essere approfondito in futuro prevede l'utilizzo della Computer Vision combinata con i classificatori, argomento ampiamente trattato su Roboflow con datasets pensati ad hoc per questo approccio. Ciò permetterebbe di riconoscere una zona di interesse nelle immagini su cui far allenare il classificatore, soluzione molto utile nel caso di immagini e/o riprese con più persone per permettere a quest'ultimo di giudicare le varie situazioni singolarmente (Figura 4.1).

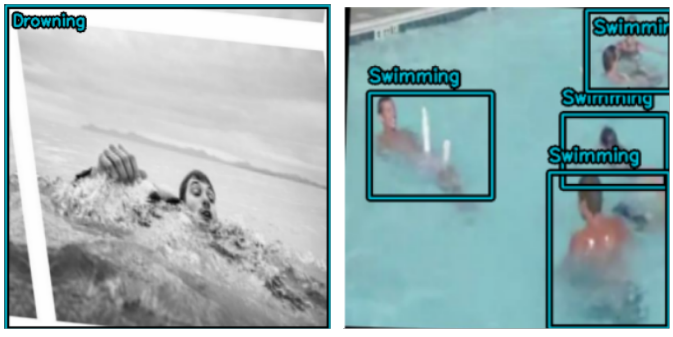


Figura 4.1: Esempi presi dal dataset SaD[14].

Anche l'utilizzo dell'individuazione dei punti del corpo (*body pose estimation*), coerentemente col lavoro svolto nell'articolo scientifico [5], potrebbe migliorare i risultati ottenuti.
Infine avendo constatato, grazie a quest'ultimo, la differenza nelle dinamiche di

nuoto e annegamento sopra e sottacqua, un'altra soluzione potrebbe essere quella di far applicare i classificatori su due immagini riferenti lo stesso istante, una sopra e sottacqua, ricavando le singole predizioni per poi confrontarle tra loro e classificarle in una classe piuttosto che un'altra.
Fino a qualche decennio fa le risposte al problema dell'annegamento e della sicurezza in generale erano limitate. Oggi, grazie ai progressi della tecnologia, è possibile pensare ad un approccio di sistemi di rilevamento basati su intelligenza artificiale, che con il dovuto perfezionamento, potrebbero identificare i bambini che stanno annegando anche quando sono in condizioni di scarsa visibilità o in acqua da soli.
Grazie alla ricerca e all'innovazione sarà forse possibile garantire una maggiore sicurezza in piscina e ridurre queste morti a casi sporadici.

Bibliografia

[1] Warren S. McCulloch e Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[2] David C. Schwebel, Sydneia Lindsay e Jennifer Simpson. «Brief Report: A Brief Intervention to Improve Lifeguard Surveillance at a Public Swimming Pool». In: *Journal of Pediatric Psychology* 32.7 (apr. 2007), pp. 862–868. ISSN: 0146-8693. DOI: 10.1093/jpepsy/jsm019. eprint: https://academic.oup.com/jpepsy/article-pdf/32/7/862/2655111/jsm019.pdf. URL: https://doi.org/10.1093/jpepsy/jsm019.

[3] Gitanjali Saluja et al. «Swimming Pool Drownings Among US Residents Aged 5–24 Years: Understanding Racial/Ethnic Disparities». In: *American Journal of Public Health* 96.4 (2006). PMID: 16507730, pp. 728–733. DOI: 10.2105/AJPH.2004.057067. eprint: https://doi.org/10.2105/AJPH.2004.057067. URL: https://doi.org/10.2105/AJPH.2004.057067.

[4] C Blum e J Shield. «Toddler drowning in domestic swimming pools». In: *Injury Prevention* 6.4 (2000), pp. 288–290. ISSN: 1353-8047. DOI: 10.1136/ip.6.4.288. eprint: https://injuryprevention.bmj.com/content/6/4/288.full.pdf. URL: https://injuryprevention.bmj.com/content/6/4/288.

[5] Saifeldin Hasan et al. «A Water Behavior Dataset for an Image-Based Drowning Solution». In: *2021 IEEE Green Energy and Smart Systems Conference (IGESSC)*. 2021, pp. 1–5. DOI: 10.1109/IGESSC53124.2021.9618700.

[6] The pandas development team. *pandas-dev/pandas: Pandas*. Ver. 1.5.3. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[7] Charles R. Harris et al. «Array programming with NumPy». In: *Nature* 585.7825 (set. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[8] J. D. Hunter. «Matplotlib: A 2D graphics environment». In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[9] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[10] François Chollet et al. *Keras*. https://keras.io. 2015.

[11] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[12] Stéfan van der Walt et al. «scikit-image: image processing in Python». In: *PeerJ* 2 (giu. 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

[13] pwnface4@gmail.com. *drowning people Dataset*. https://universe.roboflow.com/pwnface4-gmail-com/drowning-people. Open Source Dataset. Dic. 2021. URL: https://universe.roboflow.com/pwnface4-gmail-com/drowning-people.

[14] University. *Swimming and Drowning Detection Dataset*. https://universe.roboflow.com/university-g3h71/swimming-and-drowning-detection. Open Source Dataset. Giu. 2023. URL: https://universe.roboflow.com/university-g3h71/swimming-and-drowning-detection.

[15] Keiron O'Shea e Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[16] Corinna Cortes e Vladimir Vapnik. «Support-vector networks». In: *Machine Learning* 20 (3 1995), pp. 273–297. DOI: 10.1007/BF00994018. URL: https://doi.org/10.1007/BF00994018.

[17] Paras Varshney. *VGGNet-16 Architecture: A Complete Guide*. 2020. URL: https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide/notebook.

[18] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[19] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].