UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Analisi di algoritmi di supervised learning applicati nell'ambito della moda

Analysis of supervised learning algorithms applied in the fashion sector

Candidato:
Riccardo Mancini

Relatore:
Prof. Emanuele FRONTONI

Correlatore:
Dott.ssa Marina PAOLANTI

Anno Accademico 2020-2021

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Sommario

Il problema dell'elaborazione delle immagini, mirata all'interpretazione e alla classificazione del contenuto delle stesse, ha attirato l'attenzione dei ricercatori dai primi tempi della nascita e diffusione dei calcolatori. Con il progredire della tecnologia dei sistemi di calcolo, la categorizzazione di immagini ha trovato applicazioni sempre più vaste, riguardando discipline di nuova generazione come l'Object Recognition e la Computer Vision. È proprio in quest'ultima disciplina che si vuole inserire il lavoro di tesi, il cui tema centrale è il Deep Learning e le possibili applicazioni della classificazione di immagini nel settore della moda.

Il lavoro svolto si articola in 3 macro-sezioni. Nella prima parte è stato fatto un approfondimento generale sul Machine Learning, per poi specializzarsi sui vari algoritmi di apprendimento automatico, tra cui proprio quello dell'apprendimento profondo, o Deep Learning. Poi si è passati ad analizzare e a descrivere il concetto di rete neurale artificiale, approfondendo le reti neurali convolutive (CNN) con alcuni cenni matematici e strutturali. Queste ultime sono state il fulcro del progetto, grazie alle quali è stato possibile portare a termine l'intera classificazione.

La seconda sezione del lavoro è stata quella inerente all'estrazione delle feature, con conseguente elaborazione e interpretazione delle immagini da parte delle CNN appena anticipate. Tutto questo è stato attuabile grazie all'implementazione di metodi e tecniche che hanno come fine quello di spiegare la percezione delle reti neurali, e come queste prendano decisioni nel momento in cui gli vengono dati degli input, in questo specifico caso delle immagini.

Infine, la fase finale ha preso il via con l'addestramento di queste reti su due dataset di immagini il cui unico fattore in comune era quello di contenere delle borse. Successivamente sono stati studiati e valutati i comportamenti derivanti dal post classificazione, tramite dei grafici appositamente costruiti durante questa fase e attraverso quelle tecniche di interpretabilità anticipate prima. Tutto questo cercando di dare sempre un collegamento teorico a tutto quello che veniva osservato.

Questo documento, articolato su 5 capitoli, ha come fine quello di fornire un buon punto di partenza sulle potenzialità, che al giorno d'oggi, il deep learning sta fornendo al settore dell'informatica e in generale nelle applicazioni che vengono utilizzate quotidianamente.

Indice

1 Introduzione **1**

1.1 Il contesto 1
1.2 Obiettivi e contributi principali 2
1.2.1 Descrizione dell'approccio 2
1.3 Introduzione ai capitoli successivi 3

2 Stato dell'arte **5**

2.1 Deep learning 5
2.1.1 Il processo di apprendimento 6
2.1.2 Reti neurali artificiali 7
2.1.3 Addestramento di una rete neurale 8
2.1.4 Problemi del training: overfitting e underfitting 11
2.1.5 Reti neurali convoluzionali 13
2.1.6 Architettura e funzionamento generale delle CNN 13
2.1.7 Caratteristica principale delle reti CNN utilizzate 15
2.2 Interpretability 16
2.2.1 L'importanza dell'Interpretability 16
2.2.2 Struttura della metodologia utilizzata per l'Interpretability . 18
2.2.3 Scopo dell'Interpretability 19

3 Materiali e metodi **21**

3.1 Dataset 21
3.1.1 Manipolazione ottimale dei dataset 21
3.2 CNN utilizzate 23
3.2.1 VGG-Net [1] 24
3.2.2 ResNet - Residual Network [2] 25
3.2.3 DenseNet [3] 27
3.2.4 SqueezeNet [4] 30
3.2.5 EfficientNet [5] 32
3.3 Tecniche di interpretability 35
3.3.1 Activation Map (o Feature Map) 35
3.3.2 Saliency Map [6] [7] [8] 39
3.3.3 Integrated Gradient [9] 42
3.4 Implementazione delle tecniche di interpretability applicate alle CNN 44
3.4.1 Implementazione delle Activation Map 44
3.4.2 Implementazione delle Saliency Map 46

3.4.3 Implementazione dell'Integrated Gradient 48

4 Risultati e discussioni **51**

4.1 Metrica di analisi utilizzata 51

4.2 Risultati ottenuti con il primo dataset 52

4.3 Risultati ottenuti con il secondo dataset 55

4.4 Comparazione dei risultati ottenuti attraverso l'interpretabilità dei modelli 57

5 Conclusione **67**

5.1 Sviluppi futuri 68

Elenco delle figure

2.1 Rete neurale composta da un neurone 8
2.2 Rete con hidden layer a 3 neuroni 9
2.3 Neurone ad 1 input - 1 output 10
2.4 Casi di overfitting e underfitting 11
2.5 CNN con input, hidden e output layers 14
2.6 Struttura generale di una CNN 15

3.1 (a) rappresenta una borsa di tipo clutch presente nel primo dataset; (b) rappresenta una borsa di tipo tracolla presente nel primo dataset; (c) è un'immagine presa dal secondo dataset in cui è presente una borsetta; (d) è un'immagine presa dal secondo dataset in cui non è presente una borsetta. 23
3.2 Struttura della VGG 16 24
3.3 Struttura della ResNet 34 26
3.4 Tabella con struttura della ResNet in più versioni 26
3.5 Modello astratto con la concatenazione applicata 27
3.6 Blocchi convoluzionali di una Densenet 28
3.7 Tabella della struttura di una DenseNet 28
3.8 Dense layer 29
3.9 Dense block 30
3.10 Struttura DenseNet 30
3.11 Struttura SqueezeNet 31
3.12 Struttura Fire Module (1) 32
3.13 Struttura Fire Module (2) 32
3.14 Ridimensionamento sulle reti 33
3.15 Tabella con la struttura di una EfficientNet 34
3.16 MBConv di una EfficientNet 35
3.17 Matrice 5x5 36
3.18 Matrice 3x3 36
3.19 Operazione di convoluzione 37
3.20 Effetto della modifica dei filtri su un'immagine 37
3.21 Esempio di Activation map 38
3.22 Schema generale di una rete deconvoluzionale 39
3.23 Esempio di saliency map prodotte da una rete deconvoluzionale . . . 40

3.24 Esempio di saliency map prodotte utilizzando l'algoritmo di backpropagation 40
3.25 Descrizione tecnica dell'algoritmo di backpropagation guidato 41
3.26 Esempio di saliency map prodotte utilizzando l'algoritmo di backpropagation guidato 41
3.27 Interpolazione dell'immagine 43
3.28 Esempio dell'Integrated gradient applicato sull'immagine 43
3.29 Codice per la registrazione dell'hook di ogni layer 45
3.30 8 feature map per ogni layer 45
3.31 Singola feature map per layer 46
3.32 Immagine di input trasformata 47
3.33 Codice dell'istanziazione della classe Backprop 47
3.34 Saliency map dopo l'operazione di backpropagation 48
3.35 Saliency map dopo l'operazione di backpropagation guidata 48
3.36 Immagine manipolata per darla in input alla ResNet 18 49
3.37 Snippet di codice della predizione della classe 50
3.38 Snippet di codice per la visualizzazione dell'immagine con le relative attribuzioni 50

4.1 Grafico della training loss dei vari modelli addestrati sul primo dataset 52
4.2 Grafico della testing loss dei vari modelli addestrati sul primo dataset 53
4.3 Grafico della testing loss della VGG 16 addestrata sul primo dataset 53
4.4 Grafico della testing loss della DenseNet 121 addestrata sul primo dataset 54
4.5 Grafico dell'accuracy di classificazione dei vari modelli addestrati sul primo dataset 54
4.6 Grafico della training loss dei vari modelli addestrati sul secondo dataset 55
4.7 Grafico della testing loss dei vari modelli addestrati sul secondo dataset 55
4.8 Grafico dell'accuracy di classificazione dei vari modelli addestrati sul secondo dataset 56
4.9 Grafico dell'accuracy di classificazione della DenseNet 121 addestrata sul secondo dataset 56
4.10 Grafico dell'accuracy di classificazione della SqueezeNet 1_0 addestrata sul secondo dataset 57
4.11 Saliency map ottenute con la prima immagine presa dal primo dataset 59
4.12 Activation map ottenute con la prima immagine presa dal primo dataset 59
4.13 Integrated gradient applicato alla prima immagine presa dal primo dataset 60
4.14 Saliency map ottenute con la seconda immagine presa dal primo dataset 61
4.15 Activation map ottenute con la seconda immagine presa dal primo dataset 61

4.16 Integrated gradient applicato alla seconda immagine presa dal primo dataset 62
4.17 Saliency map ottenute con la prima immagine presa dal secondo dataset 63
4.18 Activation map ottenute con la prima immagine presa dal secondo dataset 63
4.19 Integrated gradient applicato alla prima immagine presa dal secondo dataset 64
4.20 Saliency map ottenute con la seconda immagine presa dal secondo dataset 65
4.21 Activation map ottenute con la seconda immagine presa dal secondo dataset 65
4.22 Integrated gradient applicato alla seconda immagine presa dal secondo dataset 66

Capitolo 1

Introduzione

In questo capitolo è descritto in breve il contesto applicativo nel quale si inserisce il lavoro di tesi, gli obiettivi e una breve descrizione dell'approccio che hanno portato alle scelte trattate in questo documento. Prima di addentrarsi in maniera dettagliata nella trattazione, è necessaria una doverosa ed importante contestualizzazione delle possibili applicazioni tecniche che saranno esposte nel proseguo di questa tesi.

1.1 Il contesto

Il task su cui si basa l'intera tesi è fondamentalmente la classificazione di immagini attraverso modelli di reti neurali. Il concetto di classificazione, o Image Classification, è quel ramo della Visione Artificiale che prevede l'individuazione delle caratteristiche di un'entità da classificare all'interno di una collezione di immagini o video, con la conseguente associazione di una classe di appartenenza. Tale risultato può essere raggiunto tramite tecniche statistiche o algoritmi di apprendimento automatico. A differenza dell'occhio umano, che riconosce e interpreta un oggetto appartenente a una categoria in maniera immediata e indipendente dalle condizioni di osservazione (es. differente angolazione, parziale occlusione dell'oggetto, formato dell'oggetto), il calcolatore non riesce a determinare in maniera così semplice tali informazioni, ma necessita di processi più complessi per raggiungere un fine che per gli umani è naturale. La via più "semplice" è quella restringere il problema della classificazione ad un ambito specifico e ben definito che permetta di fornire modelli di riconoscimento predefiniti. Quindi data un'immagine che contiene uno o più oggetti e dato un set di labels (etichette), indotte da modelli già conosciuti, il sistema mira ad assegnare tali etichette alle regioni di interesse presenti nell'immagine, identificando così l'oggetto, o gli oggetti, da riconoscere. I compiti necessari ad un calcolatore per raggiungere questo obiettivo vengono descritti tramite algoritmi di apprendimento automatico, in questo specifico caso attraverso algoritmi di Deep Learning (DL), che utilizzano reti neurali artificiali in grado di replicare le caratteristiche del cervello umano, e non solo. Per far si che la classificazione porti risultati notevoli occorre mettere il calcolatore nelle condizioni di poter essere istruito al riconoscimento delle immagini: saranno quindi fornite delle immagini in input alla rete, sia in condizioni originali sia in condizioni di distorsione rispetto alla qualità iniziale, verranno analizzati i dati

elaborati, infine si cercheranno di definire alcuni pattern comportamentali assunti dalla rete in determinate condizioni. Questa fase potrà essere considerata una sorta di fase di testing della robustezza della rete da noi scelta.

1.2 Obiettivi e contributi principali

Il lavoro svolto in questa tesi di laurea ha come obiettivo, come specificato in precedenza, quello di classificare immagini, identificando a quale categoria, o classe, l'oggetto in essa contenuto appartiene. In realtà la tesi si inserisce in un lavoro più ampio, che ha la finalità di creare un sistema, nell'ambito della moda, in grado di classificare delle borsette presenti in delle immagini contenute in dataset differenti. Come accennato, questo è stato possibile grazie all'utilizzo di modelli di reti neurali a cui vengono date in input le immagini con le borsette, e dopo un'opportuna fase di training verranno prodotti specifici risultati riconducibili o meno al corretto apprendimento. In aggiunta ai risultati finali ottenuti dalla fase di classificazione, un altro obiettivo di questo lavoro, per verificare principalmente che questi modelli abbiano appreso, o meglio "interpretato", correttamente le immagini da elaborare, è quello del Feature Visualization; è qua che entra in gioco il campo dell'interpretability (per questo l'utilizzo della parola "interpretato"), che si occupa di rispondere alle seguenti domande: "cosa sta vedendo la rete neurale in questo momento?", "su cosa si sta concentrando la rete?". La descrizione di queste tecniche verrà ovviamente trattata più avanti nel corso di questo documento.

1.2.1 Descrizione dell'approccio

Il lavoro è iniziato con degli studi teorici sul concetto di machine learning, o apprendimento automatico, che è essenzialmente una strada per l'attuazione dell'intelligenza artificiale; questo concetto sfrutta la capacità delle macchine di ricevere una serie di dati e di apprendere da essi, modificando le funzioni matematiche degli algoritmi basandosi sulla grande quantità di informazioni che elaborano. L'apprendimento automatico quindi, si occupa di "educare" un algoritmo in modo che possa apprendere da varie situazioni ambientali. L'educazione, o ancora meglio l'addestramento, implica l'utilizzo di enormi quantità di dati e un efficiente algoritmo al fine di adattarsi (e migliorarsi) in accordo alle situazioni che si verificano. Poi si è passati ad approfondire il concetto di apprendimento approfondito, o deep learning. Il deep learning è uno degli approcci all'apprendimento automatico che ha preso spunto dalla struttura del cervello, ovvero l'interconnessione dei vari neuroni. L'apprendimento approfondito utilizza enormi modelli di reti neurali con varie unità di elaborazione; sfrutta i progressi computazionali e tecniche di allenamento per apprendere modelli complessi attraverso una enorme quantità di dati. Il concetto di apprendimento profondo/approfondito viene a volte indicato semplicemente come "rete neurale profonda", in riferimento ai numerosi livelli coinvolti. Un'applicazione comune del

deep learning è la classificazione delle immagini, uno dei pilastri di questa tesi. Successivamente, sono state analizzate le varie tecniche di interpretability applicate alle reti neurali profonde, queste, sono state utilizzate per poter poi analizzare i risultati ottenuti e renderli più comprensibili all'occhio umano.

Una volta acquisito le conoscenze teoriche necessarie per poter affrontare al meglio questo progetto, si è passati alla costruzione dei dataset utilizzati. Le immagini su cui si sono basati i test sperimentali, appartengono infatti a dataset, più o meno estesi, creati ad hoc per questo lavoro. Per sperimentare i vantaggi dell'insieme di metodologie di Image Classification e di Feature Visualization, si è fatto uso di PyTorch, una libreria di machine learning open source basata sulla libreria Torch, utilizzabile mediante linguaggio python, con ampio supporto alle metodologie allo stato dell'arte di Deep Learning. PyTorch offre delle funzionalità di alto livello come il tensor computing (NumPy), con una forte accelerazione sfruttando l'unità di elaborazione grafica (GPU), e inoltre, fornisce alcune delle architetture di reti neurali profonde (deep neural networks) più utilizzate per lavori di questo tipo. Una libreria di PyTorch, che è risultata di fondamentale importanza, è stata torchvision, attraverso la quale è stato possibile utilizzare i modelli di rete neurale pre-addestrati, contenuti in PyTorch, che sono stati impiegati per compiere la classificazione e le successive analisi associate. L'unica rete neurale non contenuta in questa libreria (la EfficientNet[1]) è stata importata da una repository pubblica di GitHub, ovviamente anche quest'ultima opportunamente pre-addestrata. Nella fase successiva alla classificazione, ma comunque antecedente all'analisi dei risultati, sono state applicate alcune tecniche di Feature Visualization: Activation Map, Saliency Map ed Integrated Gradient. Ognuna di esse possiede un proprio sviluppo, che verrà meglio trattato nei capitoli successivi, ma grazie a queste tecniche è stato possibile applicare sul campo il concetto di Interpretability precedentemente studiato e quindi analizzare come una rete neurale percepisce determinate immagini date in input e su che caratteristiche si focalizza.

Infine, i risultati della classificazione ottenuti sono stati sottoposti ad un'attenta e accurata valutazione grazie a Neptune. Neptune non è altro che una suite di librerie, la quale offre allo stesso tempo anche un'interfaccia web (UI), che si integra perfettamente con i vari strumenti di Machine Learning e che permette di tenere traccia delle informazioni che si vogliono estrarre. E' possibile poi analizzare, con opportuni metodi grafici, le metriche utilizzate per verificare la corretta classificazione delle immagini.

1.3 Introduzione ai capitoli successivi

Si è dunque deciso di considerare le Convolutional Neural Network (CNN) per affrontare la parte di classificazione del progetto. Per utilizzarle è necessario prima

[1] Codice sorgente EfficientNet

di tutto conoscere le nozioni elementari di una rete neurale; dato che molti concetti sono gli stessi o molto simili, sarà anche presente una piccola spiegazione della fase di addestramento di una rete neurale, per poi andare a trattare il funzionamento nello specifico delle CNN, con la definizione delle varie tipologie di layer di cui è composta.

Il capitolo 2 serve proprio per questo scopo. Inoltre, in questo capitolo, verrà approfondito il concetto di interpretabilità di una rete neurale, con la successiva descrizione di alcuni metodi per poterlo applicare concretamente.

Il capitolo 3 andrà a completare il discorso teorico del precedente capitolo, con le metodologie vere e proprie applicate nel progetto. Qui verrà mostrato nel dettaglio come sono composte le varie CNN considerate, con relativa architettura per ognuna, le varie tecniche di interpretability approfondite con rispettivo workflow e come quest'ultime sono state sviluppate nelle CNN prese in considerazione. Prima di questo, verrà prestata attenzione anche ai vari dataset di immagini utilizzati per poter compiere questo progetto, spendendo qualche parola sulla struttura di ognuno di esso e come questi sono stati manipolati per poterli utilizzare a dovere.

A questo punto il capitolo 4 è dedicato esclusivamente all'analisi e al commento dei risultati ottenuti, applicando una speciale libreria sviluppata ad hoc per una migliore esplicazione di quest'ultimi.

Infine, il quinto ed ultimo capitolo sarà riservato alla conclusione del lavoro, affiancato ad un pensiero personale, alle difficoltà riscontrate e a dei possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

2.1 Deep learning

Il deep learning, o apprendimento profondo, è un insieme di metodi riconducibili alla famiglia del machine learning che sono in grado di fornire dei modelli ad alto livello di astrazione per una vasta gamma di fenomeni non lineari. Tali tecniche hanno portato al raggiungimento di importanti progressi in varie discipline quali computer vision, natural language processing, riconoscimento facciale e vocale e analisi di segnali in generale. Il deep learning si basa su diversi modelli per rappresentare degli oggetti. Un'immagine, per esempio, può essere processata come un semplice vettore di campioni numerici oppure con altri tipi di rappresentazioni. Nello specifico la si potrebbe descrivere a partire da:

- l'intensità dei pixel;
- i bordi degli elementi che la compongono;
- le sue diverse regioni, con forme particolari.

L'uso della giusta rappresentazione rende il compito di apprendimento più efficiente. La ricerca in quest'area si sforza quindi di costruire modelli della realtà quanto più efficienti con l'obiettivo di estrapolare le migliori rappresentazioni da vaste collezioni di dati non strutturati. Numerose tecniche di deep learning sono espressamente influenzate dalla neuroscienza e si ispirano ai modelli di elaborazione dell'informazione e di comunicazione del sistema nervoso, con particolare attenzione al modo in cui si stabiliscono le connessioni tra neuroni in base ai messaggi ricevuti, alle risposte neuronali e alle caratteristiche delle connessioni stesse. Un'altra peculiarità delle tecniche di deep learning consiste nella sostituzione di alcuni artefatti particolarmente complessi con modelli algoritmici di apprendimento, supervisionato o non supervisionato, attraverso tecniche di estrazione gerarchica delle caratteristiche. Le tecniche di apprendimento profondo utilizzano infatti molteplici strati (layer) di unità di elaborazione non lineari per l'estrazione e la trasformazione di feature. Ogni layer prende in input l'output del precedente. Questa natura spiccatamente stratificata permette di operare con l'apprendimento su diversi livelli di dettaglio e rappresentazione dei dati. È quindi possibile passare dall'utilizzo di parametri

di basso livello a parametri di alto livello, dove i diversi livelli corrispondono a diversi livelli di astrazione dei dati. In questo modo diviene possibile avvicinarsi maggiormente al significato semantico dei dati e dare loro la forma di immagini, suoni o testi.

2.1.1 Il processo di apprendimento

Gli algoritmi di apprendimento automatico vengono generalmente applicati nel riconoscimento di pattern e nella classificazione statistica. Inoltre, possono suddividersi in più categorie in base al loro scopo e comportamento: supervisionati, non supervisionati e autogestiti (self-supervised).

Negli **algoritmi supervisionati** si cerca di costruire un modello partendo da dati di addestramento etichettati, con i quali si faranno previsioni su dati non disponibili o futuri. Questo significa che nel dataset di partenza i segnali di output desiderati sono già noti poiché precedentemente etichettati. Questo tipo di apprendimento, basato su etichette delle classi discrete, agevola lo sviluppo delle tecniche di classificazione. Un altro tipo di tecnica utilizzata nell'apprendimento supervisionato è la regressione, dove i segnali di output sono dei valori continui.

Gli **algoritmi non supervisionati** si riferiscono all'uso di algoritmi di apprendimento automatico per identificare modelli di dati contenuti in dataset che non sono né classificati né etichettati. Gli algoritmi sono quindi autorizzati a classificare, etichettare e/o raggruppare i dati contenuti all'interno di dataset senza avere alcuna guida esterna per eseguire tale compito. In altre parole, l'apprendimento non supervisionato consente al sistema di identificare da solo le caratteristiche all'interno dei dataset, esclusivamente sulla base di somiglianze e differenze anche se non sono fornite categorie esplicitamente. In linea generale, questa tipologia di algoritmi possono eseguire attività di elaborazione più complesse rispetto ai sistemi di apprendimento supervisionato.

Gli **algoritmi autogestiti** sono una soluzione emergente che nasce proprio dalla necessità di eliminare la dipendenza dalle etichette associate ai dati. Costruendo modelli in modo autonomo, questa tecnologia riduce i costi e i tempi per la realizzazione dei modelli di machine learning. Infatti può funzionare senza alcuna interazione esterna e mostrare come gli esseri umani possono prendere determinate decisioni con il loro intelletto. La caratteristica comune dell'apprendimento supervisionato e autogestito è che entrambi i metodi costruiscono modelli di apprendimento da dataset i cui dati sono già etichettati. Tuttavia, l'apprendimento autogestito non richiede l'aggiunta manuale di etichette poiché le genera da solo. Quest'ultima è una particolarità che caratterizza anche l'apprendimento non supervisionato; infatti si potrebbe anche dire che gli algoritmi autogestiti sono molto più simili a quelli non supervisionati che a quelli supervisionati (in alcune fonti è addirittura considerato un suo sottoinsieme). Tuttavia, l'apprendimento non supervisionato si concentra sul raggruppamento e sulla riduzione della dimensionalità, mentre l'apprendimen-

to autogestito mira comunque a trarre conclusioni per i compiti di regressione e classificazione.

Una volta chiarita questa sorta di classificazione degli algoritmi, si può trattare più nel dettaglio la fase di apprendimento dati, la quale avviene in diversi passaggi:

1. *Pre-processamento dei dati*: i dati designati per l'apprendimento vengono trasformati e mappati in adeguate matrici numeriche così da poter essere forniti in input al modello. Nel caso di un classificatore, il dataset viene inoltre suddiviso negli insiemi di "training" e "test". Per ogni istanza di questi dati è già presente la rispettiva etichetta di classificazione, indispensabile per il processo di training.

2. *Inizializzazione del modello*: per inizializzare il modello si assegnano dei valori ai parametri dello stesso. Questi valori possono essere assegnati casualmente oppure ereditati da altri modelli di deep learning.

3. *Training*: si ripartisce ulteriormente il set di training in batch. Ogni dato di un batch viene fornito in ingresso al modello, quest'ultimo calcola una score function che assegna al dato dei punteggi, ad esempio dei valori di appartenenza alle categorie di un classificatore. In seguito a questa predizione si calcola una funzione di costo che valuta la differenza tra i valori predetti e quelli reali. Lo scopo della rete neurale è quello di minimizzare il valore della funzione di costo mediante l'aggiornamento progressivo dei propri parametri. Per la fase di training è però necessario fare un discorso più approfondito che verrà trattato successivamente in questo capitolo.

4. *Test*: in quest'ultima fase la rete processa i dati di test ed esegue una stima della qualità del modello.

2.1.2 Reti neurali artificiali

Come accennato nel precedente paragrafo, l'apprendimento profondo basa il suo funzionamento sulla classificazione e sulla "selezione" dei dati più rilevanti per giungere ad una conclusione, esattamente come fa un cervello biologico che per formulare una risposta ad un quesito, dedurre un'ipotesi logica, arrivare alla risoluzione di un problema, mette in moto i propri neuroni biologici e le connessioni neurali. Il deep learning si comporta allo stesso modo sfruttando le reti neurali artificiali (Artificial Neural Network, ANN). Una rete neurale artificiale non è altro che un modello matematico di calcolo basato su percettroni, neuroni artificiali in grado di apprendere, ovvero accumulare esperienza. L'elemento costituente più rilevante delle reti neurali è costituito dalle interconnessioni di informazioni che implementano un approccio "connessionistico" dinamico, quindi in grado di modificare la propria struttura in base ai flussi di dati che determinano l'apprendimento. Una rete neurale è organizzata in strati di neuroni, riceve le informazioni d'ingresso attraverso un

input layer e restituisce i risultati dell'elaborazione per mezzo di un output layer; tra questi strati possono esserci uno o più hidden layer. Esse vengono utilizzate per stimare o approssimare funzioni che possono dipendere da un numero elevato di input, molti dei quali spesso non noti. Le reti neurali artificiali vengono generalmente presentate come dei sistemi di "neuroni" fra loro interconnessi, tra i quali avviene uno scambio di messaggi. Ciascuna connessione ha un relativo peso associato, detto weight; il valore dei weight è regolabile in base all'esperienza in questione e ciò rende le reti neurali uno strumento adattabile ai vari tipi di input e con la capacità di apprendere. La più semplice rete neurale che può esistere è quella costituita da un solo neurone, come mostrata nella figura 2.1.

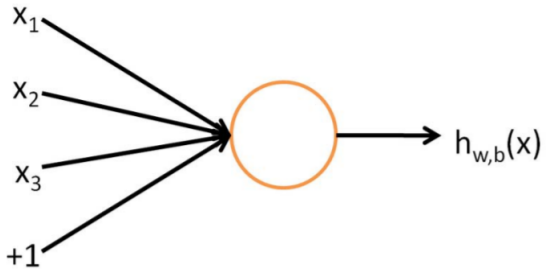

Figura 2.1: Rete neurale composta da un neurone

Un neurone può essere interpretato come un'unità computazionale, la quale prende in input gli ingressi X_1, X_2, X_3 e produce come output $h_{W,b}(X)$, detta attivazione del neurone (poi verrà argomentata meglio nella sezione "2.1.5 Reti neurali convoluzionali"). Si può notare anche un ulteriore input, il quale vale costantemente 1; il suo ruolo verrà spiegato fra poco. In realtà le cose sono leggermente più complesse in quanto un singolo neurone riceve in ingresso un valore numerico (la somma pesata di diversi input), ed esso può attivarsi, elaborando il valore ricevuto e calcolandosi la sua attivazione tramite una specifica funzione, oppure può rimanere inattivo a seconda che venga superata o meno la sua soglia di attivazione. Pertanto, il neurone sarà caratterizzato da una funzione di attivazione e da una soglia di attivazione. Data una determinata funzione di attivazione in certi casi potrebbe essere difficile se non impossibile ricavare con essa alcuni valori di output: potrebbero non esistere delle combinazioni di valori di input e di weight per quella funzione che producano un potenziale output voluto. E' necessario dunque l'utilizzo di un ulteriore coefficiente **b** noto come bias, il cui scopo è quello di permettere la traslazione della funzione di attivazione del neurone in modo da poter ottenere con certi input e weight tutti i potenziali output voluti. Questo è proprio lo scopo dell'input costante +1.

2.1.3 Addestramento di una rete neurale

Come accennato più volte in precedenza per poter ottenere risultati soddisfacenti le reti neurali devono essere addestrate, ed uno dei metodi più noti ed efficaci per il training è il cosiddetto algoritmo di retropropagazione dell'errore (error backpro-

pagation). Esso modifica sistematicamente i weight delle connessioni tra i neuroni in modo tale che la risposta della rete si avvicini sempre di più a quella desiderata. L'addestramento di una rete neurale di questo tipo avviene in due diversi stadi: forward propagation e backward propagation. Sostanzialmente nella prima fase vengono calcolate tutte le attivazioni dei neuroni della rete, partendo dal primo e procedendo fino all'ultimo layer. Durante questa fase i valori dei weight sinaptici sono tutti fissati; alla prima iterazione si avranno dei valori di default per essi. Nella seconda fase la risposta della rete, o meglio l'uscita reale, viene confrontata con l'uscita desiderata ottenendo così l'errore della rete. L'errore calcolato viene propagato nella direzione inversa rispetto a quella delle connessioni sinaptiche, ovvero in senso opposto alla prima fase. Al termine della seconda fase, sulla base degli errori appena calcolati, i weight vengono modificati in modo da minimizzare la differenza tra l'uscita attuale e l'uscita desiderata. L'intero procedimento viene poi iterato, ripartendo con una forward propagation. Si può mostrare un esempio, applicando un minimo di matematica, utilizzando una tra le più semplici deep neural network e cioè una rete neurale a due ingressi ed una sola uscita con un hidden layer a 3 neuroni (verranno considerati dei neuroni sigmoidali per poter semplificare poi la fase di backpropagation). Quindi considerando una rete come illustrata in figura 2.2.

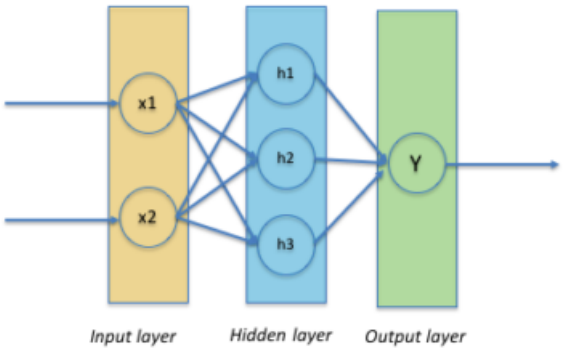

Figura 2.2: Rete con hidden layer a 3 neuroni

Il modello matematico della rete neurale, per la cui implementazione bisogna ricordare che lo strato di input non entra nella somma pesata in quanto l'output dei neuroni dell'input layer è esattamente il vettore dei dati di ingresso, è esattamente il seguente:

$$h1 = F(x1 * w11 + x2 * w12 + b1)$$

$$h2 = F(x1 * w21 + x2 * w22 + b2)$$

$$h3 = F(x1 * w31 + x2 * w32 + b3)$$

$$Y = F(h1 * w1 + h2 * w2 + h3 * w3 + b)$$

ove F è la funzione Sigmoidale, w sono i vari weights, e b è una costante e rappresenta il bias che serve per far adattare meglio la rete e lo si può immaginare come un ulteriore neurone che non accetta valori in ingresso, mentre l'uscita Y sarà quindi:

$$Y = \frac{1}{1 + e^{-(w_1 * \frac{1}{1+e^{-(x_1 * w_{11} + x_2 * w_{12} + b_1)}} + w_2 * \frac{1}{1+e^{-(x_1 * w_{21} + x_2 * w_{22} + b_2)}} + w_3 * \frac{1}{1+e^{-(x_1 * w_{31} + x_2 * w_{32} + b_3)}} + b)}}$$

È intuitivo comprendere quanto complessa può diventare la rete neurale all'aumentare del numero dei neuroni e dei livelli. A questo punto manca solo un ultimo passo, ovvero, come scegliere i pesi e con quale principio? Ed è qua che entra in gioco la backpropagation dell'errore grazie alla quale la rete è in grado di imparare e modificare in automatico i pesi confrontando, in fase di training della rete, il risultato ottenuto con il risultato atteso (quello reale).

Facendo appello ad un modello semplificato di neurone ad un solo ingresso ed una sola uscita come in figura 2.3

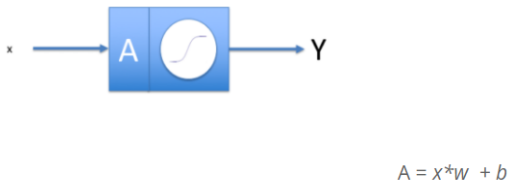

Figura 2.3: Neurone ad 1 input - 1 output

In tal caso, l'algoritmo di backpropagation dell'errore lavorerà secondo il seguente approccio:

$$E_{rr} = \hat{Y} - Y$$

$$Y = \frac{1}{1 + e^{-(A)}}$$

$$Delta_y = E_{rr} * \partial Y = E_{rr} * Y * (1 - Y)$$

$$w = w + (\varepsilon * Delta_y * x)$$

$$b = b + (\varepsilon * Delta_y)$$

In pratica, partendo da valori casuali di w e b si fa un test utilizzando dati di ingresso il cui valore di uscita è noto; si calcola il valore dell'errore come differenza tra il valore di uscita noto e quello ottenuto, ed il delta y come moltiplicazione tra il valore dell'errore e la derivata di Y che, in questo caso, essendo una sigmoide, è proprio Y*(1-Y). A questo punto, si calcolano semplicemente w e b applicando la formula indicata sopra. Il valore di epsilon, chiamato fattore di apprendimento, viene scelto liberamente, ma sarà opportuno non assegnare ad esso valori troppo piccoli perché, altrimenti, si rischierebbe di non raggiungere la convergenza ottimale della

rete; invece, assegnando valori troppo grandi, aumenterebbe la velocità della fase di addestramento, come anche la possibilità di errori e di forti oscillazioni. I valori di epsilon dovranno essere, comunque, sempre compresi tra 0 e 1. Per concludere, sui livelli intermedi della rete non può essere fatto lo stesso ragionamento del livello di uscita, se non altro perché non si ha un valore di riferimento atteso noto, in tal caso, per calcolare i vari pesi sui neuroni interni, si procede propagando indietro l'errore dell'uscita (backpropagation). Tutto ciò si ridurrà alle seguenti semplici formule:

$$Err_h = Delta_y * w$$

$$\delta_h = Err_h * H * (1 - H)$$

$$w_h = w_h + (\varepsilon * \delta_h * x)$$

$$b_h = b_h + (\varepsilon * \delta_h)$$

2.1.4 Problemi del training: overfitting e underfitting

Nella fase di allenamento risulta fondamentale comprendere quali possono essere i possibili errori di predizione che provocano una scarsa capacità di generalizzazione della rete. In tal senso è bene conoscere i concetti di bias (anche se già anticipato in precedenza) e di varianza ,dal momento che tra questi esiste il cosiddetto problema del tradeoff nel cercare di minimizzarli.

- Bias: è la differenza tra la previsione media e il valore atteso. Rappresenta l'errore sistematico non derivante dalla casualità. Un modello con bias elevato presenta delle scarse prestazioni sul training set.
- Varianza: è la variabilità di previsione del modello. Rappresenta la sensibilità del modello alla casualità dei dati. Un modello con elevata varianza non è in grado di generalizzare e pertanto risente di scarse prestazioni sul validation set.

Figura 2.4: Casi di overfitting e underfitting

Nella figura 2.4 si notano delle situazioni che possono verificarsi in fase di training. Il modello perfetto è caratterizzato da bassa varianza e basso bias (in basso a sinistra) mentre il modello peggiore presenta valori elevati sia di varianza che di bias (in alto a destro). Tra queste due casistiche si presentano altre due situazioni intermedie che possono essere corrette adottando varie soluzioni.

1. Underfitting: si verifica quando il modello presenta elevato bias e bassa varianza. In questa condizione, la rete, a causa della sua semplicità, non riesce a comprendere i pattern dei dati presentati. Le possibili soluzioni sono:

- Utilizzo di reti con più layer;
- Modifica dell'architettura della rete;
- Training più lunghi.

2. Overfitting: si verifica quando il modello presenta basso bias ed elevata varianza. In questa situazione, la rete, a causa della sua sensibilità al rumore dei dati, non è in grado di generalizzare adeguatamente. Quindi avviene quando un modello, invece di imparare a generalizzare sui dati di allenamento (e quindi comprendere ciò che sta facendo), si limita ad imparare a memoria tali dati, avendo quindi degli scarsi risultati quando si tratta di vedere i suoi progressi sui dati di test. La sua accuratezza smette di crescere nel tempo ed inizia ad oscillare o a diminuire. Questo è un problema, in quanto generalmente si vuole massimizzare tale accuratezza, facendo sì che i modelli addestrati riescano a generalizzare la loro conoscenza sugli esempi che gli vengono dati da elaborare. Le possibili soluzioni per risolvere questo problema sono:

- Aumentare i dati a disposizione per il training. L' incremento dei dati può essere realizzato anche in modo 'fittizio' attraverso tecniche di data augmentation che eseguono rotazioni, flip, distorsioni, zoom, modifiche di colore ecc sui dati del training.
- Suddivisione del dataset in Training set e Validation set. Questo può essere eseguito mediante cross-validation per visualizzare il comportamento della rete quando vengono presentati esempi diversi durante il training.
- Early Stopping: l'allenamento viene interrotto nel momento in cui la rete inizia a sovra utilizzare i dati del training set.
- Dropout: nel corso dell'allenamento vengono casualmente spenti alcuni neuroni in modo da rendere la rete meno sensibile al peso di alcuni neuroni a discapito di altri.
- Regolarizzazione: consiste nell'aggiunta di un termine alla funzione di costo, chiamato parametro di regolarizzazione.

2.1.5 Reti neurali convoluzionali

Dopo aver speso qualche parola sul training delle reti neurali e sulle unità computazionali di cui sono costituite (neuroni artificiali), tra le categorie più diffuse di reti neurali si hanno: reti neurali convoluzionali, reti neurali ricorsive, reti neurali ricorrenti e reti deep belief. In questo documento verrà trattata solamente la prima tipologia, in quanto per il lavoro sono state utilizzate solo reti di questo tipo. In machine learning, una rete neurale convoluzionale (CNN o ConvNet) è di fatto una rete neurale artificiale il cui modello di riferimento si ispira alla corteccia visiva animale. Proprio come altre reti neurali infatti, anche le reti neurali convoluzionali sono costituite da neuroni collegati fra loro tramite dei rami pesati (weight), nello specifico le ConvNet consistono in una pila multistrato di percettroni, il cui scopo è quello di processare piccole quantità di informazioni; i parametri allenabili delle reti sono sempre quindi i weight ed i bias. Inoltre bisogna evidenziare come le reti neurali convoluzionali si stanno affermando per essere utilizzate in una vasta gamma di applicazioni,trovando largo impiego nei campi del natural language processing e della computer vision, la cui applicazione verrà analizzata nel dettaglio in questo lavoro.

2.1.6 Architettura e funzionamento generale delle CNN

Una rete neurale convoluzionale può avere decine o centinaia di layer, ciascuno dei quali apprende feature diverse di un'immagine. A ciascuna immagine di addestramento vengono applicati dei filtri di diverse risoluzioni e l'output di ciascuna immagine convoluta viene utilizzato come input per il layer successivo. I filtri possono essere inizialmente feature molto semplici, ad esempio la luminosità o i bordi, e diventare sempre più complessi fino a includere feature che definiscono in modo univoco l'oggetto. Come precedentemente anticipato, analogamente a una rete neurale tradizionale, una CNN possiede neuroni con pesi e bias. Il modello apprende questi valori durante l'addestramento e li aggiorna costantemente con ogni nuovo esempio di addestramento. Tuttavia, nel caso delle CNN, i valori dei pesi e dei bias sono gli stessi per tutti i neuroni nascosti in un determinato layer. Ciò significa che tutti i neuroni nascosti rilevano la stessa feature, come bordi o macchie, in diverse aree dell'immagine. Ciò rende la rete tollerante alla traslazione di oggetti in un'immagine. Ad esempio, una rete addestrata a riconoscere automobili sarà in grado di farlo indipendentemente dal tipo di automobile presente nell'immagine. Analogamente ad altre reti neurali, una CNN è costituita da un layer di input, un layer di output e tanti layer intermedi nascosti. L'idea è quella raffigurata in figura 2.5 .

Questi layer intermedi nascosti eseguono operazioni che alterano i dati al fine di apprendere le feature specifiche dei dati stessi. È importante a questo punto dedicare alcune parole a tre layer che si incontrano molto spesso nelle reti neurali convoluzionali.

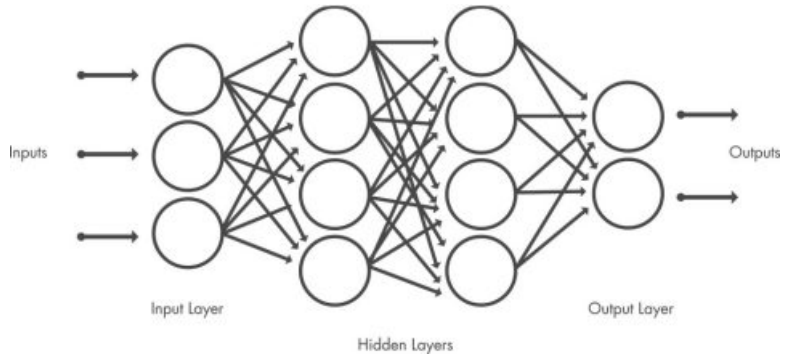

Figura 2.5: CNN con input, hidden e output layers

La **convoluzione**, componente principale delle ConvNet, non è altro che un operatore che ha come compito principale quello di estrarre caratteristiche (feature) dai dati presi in input. Si consideri ad esempio l'elaborazione di un'immagine. Nei moduli di convoluzione i dati vengono processati con il meccanismo della sliding window, dove una piccola matrice chiamata kernel di convoluzione, i cui valori caratterizzano l'operatore, viene opportunamente traslata seguendo la struttura dell'immagine. La convoluzione è un operatore locale in quanto prende in input una piccola matrice di valori e fornisce in output un solo valore in corrispondenza del centro della matrice e viene applicata in modo da mantenere la relazione spaziale tra i pixel. I kernel di convoluzione contengono i pesi utilizzati per calcolare la score function e sono associati a un bias che viene sommato al risultato della convoluzione. Quando, nel corso di questo documento, verranno trattate alcune tecniche di interpretability, verrà fornita un'illustrazione grafica dell'operazione di convoluzione per poter comprendere al meglio questo concetto.

L'**attivazione** o **ReLU** è un altro layer che si incontra spesso nelle CNN in quanto consente di eseguire un addestramento più rapido ed efficace mappando i valori negativi a zero e mantenendo quelli positivi. Questa operazione è talvolta definita attivazione, dal momento che solo le feature attivate vengono trasmesse al layer successivo. La funzione di attivazione o di trasferimento viene solitamente applicata in seguito all'operazione di convoluzione e determina quindi il comportamento in output di un neurone in funzione del suo livello di eccitazione. Queste funzioni sono solitamente non-lineari ma continue e differenziabili comprese nell'intervallo [-1, 1] come ad esempio la Standard Logistic Function (sigmoide) o la tangente iperbolica.

Un ultimo layer che si trova spesso è il **pooling**. Esso si occupa di aggregare l'input e ridurne il volume per mezzo di un sotto-campionamento così da snellire l'elaborazione per i layer successivi. Il pooling, così come avviene per la convoluzione, agisce con l'ausilio di un kernel traslato lungo tutta l'immagine. Una tecnica diffusa di sottocampionamento è il MaxPooling che ad ogni spostamento del kernel restituisce soltanto il valore massimo presente.

Queste operazioni appena descritte vengono reiterate su decine o centinaia di layer

e ciascun layer impara ad identificare feature diverse. Dopo aver appreso le feature in numerosi layer, l'architettura di una CNN passa alla fase di classificazione. A valle di una CNN si trova il penultimo layer che è completamente connesso, o **fully connected layer**. Questo emette un vettore di dimensioni K, dove K è il numero di classi che la rete sarà in grado di prevedere. Questo vettore contiene le probabilità per ciascuna classe di qualsiasi immagine classificata.

L'ultimo layer dell'architettura CNN utilizza un layer di classificazione come una **softmax** per fornire l'output della classificazione. La funzione Softmax va prendere un vettore di K numeri reali e poi lo normalizza in una distribuzione di probabilità costituita proprio da probabilità K. Prima di applicare la softmax, alcuni componenti del vettore potrebbero essere negativi o maggiori di uno e potrebbero non sommarsi a 1, ma dopo aver applicato softmax ogni componente sarà compreso tra 0 e 1 e verrà sommato a 1; può quindi essere interpretato come una probabilità. L'utilizzo di questa funzione è computazionalmente costoso in quanto vengono calcolati molti termini in esponente, ma è un metodo molto utile per evitare problemi di gradiente che scompare o che esplode, ovvero evitare di ottenere risultati molto piccoli o molto grandi in cui si troveranno difficoltà nell'applicazione dei pesi o nella terminazione dell'algoritmo.

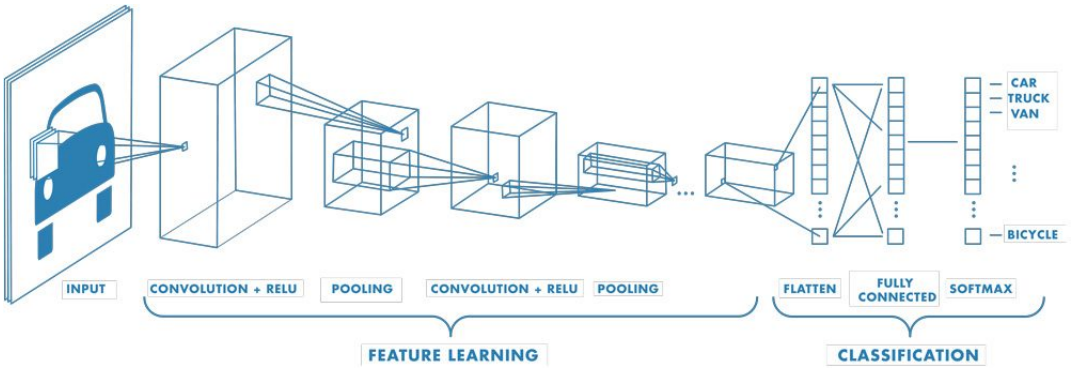

Figura 2.6: Struttura generale di una CNN

2.1.7 Caratteristica principale delle reti CNN utilizzate

Concluso il discorso sull'architettura e sul funzionamento generale di una rete neurale convoluzionale, è giusto spendere qualche parola sulla tipologia che accomuna le reti scelte per il progetto e poi elencare quali sono state scelte. Infatti non sono stati sviluppati dei modelli da zero ma si sono prese delle reti già presenti e pre-addestrate, puntando sull'efficienza e sulla maggiore precisione dei risultati. Utilizzare per poi ottimizzare una rete pre-addestrata è in genere una soluzione molto più rapida e semplice rispetto all'addestramento da zero. La quantità di dati e le risorse computazionali richieste sono minime. Si parla di transfer learning, quando vengono utilizzate le conoscenze derivanti da un tipo di problema per risolvere problemi simili. Si inizia con una rete pre-addestrata che ha già appreso un esteso set di

feature e queste possono essere applicate a una vasta gamma di attività simili. Ad esempio, come nel caso specifico di questo lavoro, è stato possibile prendere più reti addestrate su milioni di immagini e riaddestrarle per una nuova classificazione di oggetti utilizzando solo centinaia di immagini. E' quindi doveroso a questo punto elencare i modelli utilizzati, con associati i loro paper di riferimento. Le reti neurali prese in considerazione sono:

1. *ResNet 18* [2];
2. *VGG 16* [1];
3. *DenseNet 121* [3];
4. *SqueezeNet 1* [4]
5. *EfficientNet (b0)* [5].

2.2 Interpretability

Non esistono definizioni matematiche di interpretabilità applicata a modelli neurali artificiali, ma si potrebbe dare una definizione generale del tipo: l'interpretabilità è il grado in cui un essere umano può comprendere la causa di una decisione; oppure, l'interpretabilità è il grado in cui un essere umano può prevedere in modo coerente il risultato di un modello. Quello che si deduce quindi, è che maggiore è l'interpretabilità di un modello di apprendimento automatico, più è facile per qualcuno comprendere il motivo per cui sono state prese determinate decisioni o previsioni. Dunque, un modello è meglio interpretabile rispetto un altro, se le sue decisioni sono più facili da comprendere per un essere umano rispetto alle decisioni dell'altro modello. In linea generale l'interpretability permette di:

- Comprendere il sistema in modo retrospettivo: comprendere quindi, rispetto a un'azione o decisione dannosa, cosa è andato storto e perché;
- Comprendere il sistema in modo prospettico: prevedere, mitigare e prevenire future azioni o decisioni dannose.

Ricapitolando, un algoritmo si dice interpretabile se si comprende come lavora e perché ha compiuto una determinata decisione. Inoltre, è importante precisare che è necessario determinare, in alcuni contesti, che tipi di spiegazione sono appropriate e quando devono essere fornite.

2.2.1 L'importanza dell'Interpretability

Per poter analizzare le ragioni per cui l'interpretabilità è fondamentale bisogna prima di tutto capire come questo concetto si verifica negli esseri umani, per poter

capire al meglio da dove deriva la necessità di interpretare, o spiegare, un determinato evento prodotto da un algoritmo di apprendimento automatico.

Gli esseri umani hanno una visione generale del loro ambiente che viene aggiornato quando accade qualcosa di imprevedibile. Questo aggiornamento viene davvero attuato trovando una spiegazione per l'evento imprevisto. Quando inizialmente i primi modelli di machine learning venivano utilizzati nella ricerca, i risultati scientifici rimanevano completamente nascosti se i modelli fornivano solamente predizioni senza alcuna spiegazione. Per facilitare l'apprendimento e soddisfare la curiosità sul motivo per cui determinate previsioni o comportamenti vengono compiuti dalle macchine, l'interpretabilità e le spiegazioni (explainability) sono diventati di fondamentale importanza. Naturalmente, gli esseri umani non hanno bisogno di spiegazioni per tutto ciò che accade. Per la maggior parte delle persone, ad esempio, va bene che non capiscano come funziona un computer, ma è pur sempre vero che gli eventi inaspettati producono curiosità, che nasce dal desiderio di sapere qualcosa e di poterlo quindi spiegare.

In molte discipline scientifiche si passa da metodi qualitativi a metodi quantitativi (es. sociologia, psicologia), e anche verso l'apprendimento automatico (es. biologia, genomica). L'obiettivo della scienza rimane quello di acquisire conoscenza, ma molti problemi vengono comunque risolti con grandi set di dati e modelli neurali percepiti come delle vere e proprie black-box. Perciò il modello stesso diventa la fonte principale della conoscenza dei dati, rendendo quindi l'interpretabilità fondamentale e consentendole quindi di estrarre questa conoscenza aggiuntiva catturata dal modello.

Inoltre, in molti contesti in cui vengono utilizzati questi algoritmi di apprendimento automatico per risolvere problemi reali, i modelli vengono sottoposti ad addestramenti su dei dati (immagini, video, ecc...), sviluppando dei bias che portano a risultati finali non affidabili. Questo può capitare quando un modello, dopo essere addestrato, comincia a prendere decisioni non totalmente eque, discriminando quindi alcuni dati rispetto ad altri. Questo è solo un esempio per cui l'interpretabilità viene utilizzata anche come strumento di debug per rilevare i bias negli algoritmi di apprendimento automatico.

Inoltre, l'interpretabilità di un modello aiuta anche a comprendere la causa di un errore in presenza di una previsione errata. Fornisce una direzione su come riparare il sistema. Considerando ad esempio un classificatore di "husky" e di "lupo" che classifica erroneamente alcuni "husky" come "lupi", utilizzando metodi di apprendimento automatico interpretabili, si scoprirebbe che l'errata classificazione era dovuta alla presenza della neve sull'immagine. Il classificatore ha quindi imparato a utilizzare la neve come caratteristica per classificare le immagini come "lupo", il che potrebbe avere senso in termini di separazione dei "lupi" dagli "husky" nel set di dati di addestramento, ma non nell'uso nel mondo reale.

Assicurarsi che un modello di apprendimento automatico riesca a spiegare le decisioni prese, porta con se altri vantaggi:

- *Equità*: garantire che le previsioni siano imparziali e non discriminino implicitamente o esplicitamente i gruppi sottorappresentati;

- *Privacy*: garantire che le informazioni sensibili nei dati siano protette;

- *Affidabilità* o *robustezza*: garantire che piccoli cambiamenti nell'input non portino a grandi cambiamenti nella previsione;

- *Causalità*: verificare che vengano rilevate solo le relazioni causali;

- *Fiducia*: è più facile per gli umani fidarsi di un sistema che spiega le sue decisioni rispetto a una black-box.

2.2.2 Struttura della metodologia utilizzata per l'Interpretability

I metodi per l'interpretazione dell'apprendimento automatico possono essere classificati secondo vari criteri. Viene fatta una distinzione se l'interpretabilità si ottiene limitando la complessità del modello di apprendimento automatico (intrinseco) o applicando metodi che analizzano il modello dopo l'addestramento (post hoc). L'interpretabilità intrinseca si riferisce a modelli di apprendimento automatico considerati interpretabili a causa della loro struttura semplice, come alberi decisionali brevi o modelli lineari sparsi. L'interpretabilità post hoc si riferisce all'applicazione di metodi di interpretazione dopo la formazione del modello. La permutazione è, ad esempio, un metodo di interpretazione post hoc. In certi casi però, i metodi post hoc possono essere applicati anche a modelli intrinsecamente interpretabili. Ad esempio, la permutazione può essere calcolata per gli alberi decisionali.

Un altro modo per poter differenziare i metodi di interpretability viene fatta in base ai risultati che si ottengono. È però opportuno precisare, prima di vedere questi metodi, che nel campo dell'apprendimento automatico, una caratteristica è una proprietà individuale e misurabile di un fenomeno osservato. Perciò, detto questo, i vari metodi di interpretability sono:

- *Feature summary statistic*: molti metodi di interpretazione forniscono statistiche di riepilogo per ciascuna osservazione. Alcuni metodi restituiscono un singolo numero per feature, oppure risultati più complessi che associano un numero ad ogni coppia di feature.

- *Feature summary visualization*: sono metodi che permettono di visualizzare la maggior parte delle feature summary statistic, che in alcuni casi sono significativi solo ed esclusivamente se visualizzati. La dipendenza parziale di una caratteristica è un caso del genere. I grafici delle dipendenze parziali sono curve che mostrano una caratteristica e il risultato medio previsto. Il modo migliore per presentare dipendenze parziali è disegnare effettivamente la curva anziché stampare le coordinate.

- *Model internals*: l'interpretazione di modelli intrinsecamente interpretabili rientra in questa categoria. Esempi sono i pesi (o learned weights), dei modelli lineari, o la struttura ad albero appresa (le caratteristiche e le soglie utilizzate per le suddivisioni) degli alberi decisionali. Le linee che dividono gli elementi interni del modello e le statistiche di riepilogo delle funzionalità sono davvero sottili, ad esempio nei modelli lineari, i pesi sono contemporaneamente sia gli elementi interni del modello che le statistiche di riepilogo delle funzionalità.

- *Data point*: questa categoria include tutti i metodi che restituiscono data point, o unità di osservazione (già esistenti o appena creati) per rendere interpretabile un modello. Un metodo è chiamato "counterfactual explanations". Esso è utilizzato per spiegare la previsione di un'istanza di dati, il quale trova un data point simile modificando alcune delle caratteristiche per le quali il risultato previsto cambia in modo rilevante (ad esempio un capovolgimento nella classe prevista).

- *Intrinsically interpretable model*: un'altra soluzione per interpretare i modelli è approssimarli (a livello globale o locale) con un modello interpretabile. Il modello interpretabile stesso viene interpretato osservando i parametri del modello interno o le statistiche di riepilogo delle caratteristiche.

Inoltre, i metodi di interpretazione possono variare a seconda della loro dipendenza dal modello o meno. Gli strumenti di interpretazione specifici del modello (model-specific) sono limitati a classi di modelli specifiche. L'interpretazione dei pesi di regressione in un modello lineare è un'interpretazione model-specific, poiché, per definizione, l'interpretazione di modelli intrinsecamente interpretabili è sempre model-specific. Strumenti che funzionano solo per l'interpretazione di, ad esempio, reti neurali, sono specifiche del modello. Gli strumenti indipendenti dal modello possono essere utilizzati su qualsiasi modello di machine learning e vengono applicati dopo che il modello è stato addestrato (post hoc). Questi metodi, detti agnostici, di solito funzionano analizzando le coppie di input e output delle funzionalità. Per definizione, questi metodi non possono avere accesso a elementi interni del modello come pesi o informazioni strutturali.

2.2.3 Scopo dell'Interpretability

Un algoritmo addestra un modello che produce ad esempio: previsioni, classificazioni, ecc... Ogni passaggio può essere valutato in termini di trasparenza o interpretabilità.

La *trasparenza* dell'algoritmo riguarda il modo in cui l'algoritmo apprende un modello dai dati e il tipo di relazioni che può apprendere. Se vengono utilizzate reti neurali convoluzionali per classificare immagini, si può spiegare che l'algoritmo apprende i rilevatori di bordi e i filtri sui livelli più bassi. Questa è una comprensione di come funziona l'algoritmo, ma non per il modello specifico che viene appreso

alla fine, e non per come vengono fatte le previsioni individuali. La trasparenza dell'algoritmo richiede solo la conoscenza dell'algoritmo e non dei dati o del modello appreso. Algoritmi come il metodo dei minimi quadrati per i modelli lineari sono ben studiati e compresi. Sono caratterizzati da un'elevata trasparenza. Invece approcci di deep learning (calcolare un gradiente attraverso una rete con milioni di pesi) sono meno compresi, perciò sono da considerarsi meno trasparenti.

Per spiegare l'output del modello *globale*, sono necessari: il modello addestrato, la conoscenza dell'algoritmo e la relativa conoscenza dei dati. Questo livello di interpretabilità riguarda la comprensione del modo in cui il modello prende le decisioni, sulla base di una visione olistica delle sue caratteristiche e di ciascuno dei componenti appresi come pesi, parametri e strutture. Quali caratteristiche sono importanti e che tipo di interazioni hanno luogo tra loro? L'interpretabilità del modello globale aiuta a comprendere la distribuzione del risultato target in base alle caratteristiche. L'interpretabilità del modello globale è molto difficile da ottenere nella pratica. È improbabile che qualsiasi modello che superi una manciata di parametri o pesi si adatti alla memoria a breve termine dell'essere umano medio. Di solito, quando le persone cercano di comprendere un modello, ne considerano solo parti, come i pesi nei modelli lineari. Ad esempio, un modello di classificazione "Naive Bayes" con molte centinaia di funzioni sarebbe troppo grande da tenere nella memoria. E anche se riuscisse a memorizzare tutti i pesi, non saremmo in grado di fare rapidamente previsioni per nuovi punti dati. Inoltre, bisogna anche tenere a mente la distribuzione congiunta di tutte le caratteristiche, per stimarne l'importanza di ciascuna, e come queste influenzano in media le previsioni. Un compito impossibile. Ma si puoi facilmente capire un singolo peso. Sebbene l'interpretabilità del modello globale sia solitamente fuori portata, ci sono buone possibilità di comprendere almeno alcuni modelli a livello *modulare*. Non tutti i modelli sono interpretabili a livello di parametro. Se si considerano i modelli lineari, le parti interpretabili sono i pesi, per gli alberi sarebbero le divisioni e le previsioni dei nodi foglia. I modelli lineari, ad esempio, sembrano interpretabili perfettamente a livello modulare, ma l'interpretazione di un unico peso è intrecciata con tutti gli altri pesi. I pesi in un modello lineare possono però ancora essere interpretati meglio dei pesi di una rete neurale profonda.

Si può ingrandire una singola istanza ed esaminare cosa prevede o cosa percepisce il modello per questo input e spiegare perché. Quindi *localmente*, la previsione o la classificazione potrebbe dipendere solo linearmente o monotonicamente da alcune caratteristiche, piuttosto che avere una dipendenza complessa da esse.

Quindi generalizzando a più istanze, una spiegazione può essere derivata da metodi di interpretazione del modello globale, a livello modulare, o con spiegazioni di singole istanze. I metodi globali possono essere applicati prendendo il gruppo di istanze, trattandole come se il gruppo fosse l'insieme di dati completo e utilizzando i metodi globali con questo sottoinsieme. I singoli metodi di spiegazione possono essere utilizzati su ciascuna istanza e quindi elencati o aggregati per l'intero gruppo.

Capitolo 3

Materiali e metodi

3.1 Dataset

I dataset utlizzati in questo progetto sono in particolare due. Il primo dataset è stato ottenuto nell'ambito del progetto T-WINNING estraendo le immagini dal social-network Instagram utilizzando i seguenti hashtag: bauletto, clutch, hobo, marsupio, sacca, secchiello, shopping, tracolla e zaino. Questo dataset contiene immagini di sole borse classificate secondo gli hashtag utilizzati per estrarli. Si contano ben 2275 immagini suddivise quindi nelle varie categorie: bauletto (454), clutch (425), hobo (241), marsupio (216), sacca (50), secchiello (176), shopping (170), tracolla (497) e zaino (46).

Il secondo invece, è un estratto dal dataset LABIC [1]. Questo dataset è stato costruito combinando immagini scansionate da "Chictopia.com" e immagini da tre dataset pubblici esistenti: CCP (Clothing Co-Parsing dataset), CFPD (Colorful Fashion Parsing Dataset) e Fashionista dataset.

Tutte le immagini del dataset sono standardizzate a 400x600 pixel in RGB e sono state annotate manualmente utilizzando JS Segment Annotator, uno strumento gratuito web-based di annotazione delle immagini. Gli abiti contenuti nelle immagini sono stati raggruppati in classi come rappresentato nella tabella 3.1

Visto che il progetto riguardava la classificazione delle borse, è stata fatta una scansione completa di questo dataset, dal quale sono state estratte esattamente 3000 immagini. 2650 immagini in cui è presente almeno una borsa e le restanti 350 immagini senza. Da questo deriva il nuovo dataset, il quale contiene immagini con o senza borsette, classificate proprio a seconda della presenza o meno di queste ultime.

Le immagini contenute nella figura 3.1 rappresentano soltanto qualche esempio delle immagini contenute nei dataset appena descritti.

3.1.1 Manipolazione ottimale dei dataset

Per ciascuna categoria di ogni dataset, le immagini raccolte dovranno essere suddivise in un training set ed un validation set; quest'ultimo solitamente contiene una minima parte del totale delle immagini di una classe. È importante, tuttavia,

[1] Documentazione dataset LABIC

Classes	# of instances
Bag	2650
Belt	1364
Coat	1551
Dress	1065
Eyewear	1349
Footwear	4448
Hair	4446
Headwear	782
Neckwear	383
Pants	1464
Rompes/Jumpsuits	107
Shirt	3014
Shorts	747
Skin	4500
Skirt	1231
Socks	435
Stocking	450
Sweater	566

Tabella 3.1: Struttura originale del secondo dataset

fare attenzione che le immagini presenti nel validation set non siano presenti anche sul training set, in quanto verrebbero prodotti dei dati di accuracy non sensati, perché non si testerebbe l'effettiva capacità della rete di classificare immagini mai viste prima. Sarebbe infatti opportuno, sempre al fine di migliorare il modello finale della rete, avere nel validation set immagini con sfondi mai utilizzati nel training set; in questo modo si suppone che i dati finali di accuracy rispecchino di più la realtà. Si è deciso quindi di suddividere i dataset nel seguente modo: 80% di immagini per il training set e il rimanente 20% di immagini destinate al validation set; questo per entrambi i dataset utilizzati.

In genere, per ogni classe di immagine, la situazione ottimale sarebbe quella di avere nel training set sia immagini rappresentanti l'oggetto in questione con uno sfondo, sia immagini rappresentanti solamente l'intero oggetto, senza alcuno sfondo; in questo modo la rete, durante la fase di allenamento, diventerebbe più robusta, riuscendo a distinguere in maniera migliore l'oggetto di interesse anche in situazioni più complicate. Può essere utile inoltre preparare un ulteriore modesto insieme di immagini, che rappresentino tutte le categorie del dataset, per un test finale delle reti. Tali immagini dovrebbero rappresentare la realtà con cui normalmente esse verrebbero acquisite con l'applicazione finale.

In aggiunta, è importante puntualizzare che tutte le immagini contenute in entrambi i dataset, prima di venire date in input e quindi elaborate dalle CNN che verranno descritte, sono state sottoposte a delle trasformazioni e ad una successiva normalizzazione, portandole ad una dimensione di 224x224 pixel. Questo permette

(a)

(b)

(c)

(d)

Figura 3.1: (a) rappresenta una borsa di tipo clutch presente nel primo dataset; (b) rappresenta una borsa di tipo tracolla presente nel primo dataset; (c) è un'immagine presa dal secondo dataset in cui è presente una borsetta; (d) è un'immagine presa dal secondo dataset in cui non è presente una borsetta.

di fornire immagini omogenee alle reti neurali.

3.2 CNN utilizzate

A questo punto è opportuno trattare più approfonditamente le CNN prese in considerazione per portare a termine la classificazione, e sulle quali poi sono state applicate le varie tecniche di interpretability.

3.2.1 VGG-Net [1]

Le VGG-NET, il cui nome deriva da Visual Geometry Group, si affacciarono nel 2014 come progetto di spicco per l'ILSVRC per il task di classificazione pur vincendo il contest per quanto concerne il task di localizzazione. Posero agli occhi di tutti quella che era a loro avviso la svolta concettuale per ottenere migliorie: la profondità della rete. Questo permise ad un modello di rete convolutiva di raggiungere un rate di errore inferiore al 10% per la prima volta. Inoltre, furono tra le prime reti che sfruttarono l'idea dei filtri 3x3, quindi filtri più piccoli, ma ripetuti in sequenza, permettendo di ottenere gli stessi risultati ottenuti usando filtri recettivi molto più grandi; idea fortemente contrastante con le reti sviluppate fino a quel momento (ad esempio la AlexNet). Chiaramente questo aumentò il numero di filtri da utilizzare. Ottennero un grosso risparmio computazionale visto che diminuì il numero di parametri, pur producendo una mappatura migliore tra le immagini e le etichette di categoria di classificazione. Per la classificazione venne progettata una rete che di base proponeva due livelli convolutivi basati sulla funzione di attivazione ReLu e con un layer SoftMax finale ottimizzato proprio per l'operazione di classificazione. Per di più la regressione logistica fu rimpiazzata con la funzione di perdita euclidea. I modelli proposti furono: VGG 11, 16 e 19; differiscono tra loro per il numero di layer, con rispettivamente 8, 13, 16 layer convolutivi.

Analizzando nel dettaglio la struttura della VGG 16, un'illustrazione valida viene fornita nella figura 3.2.

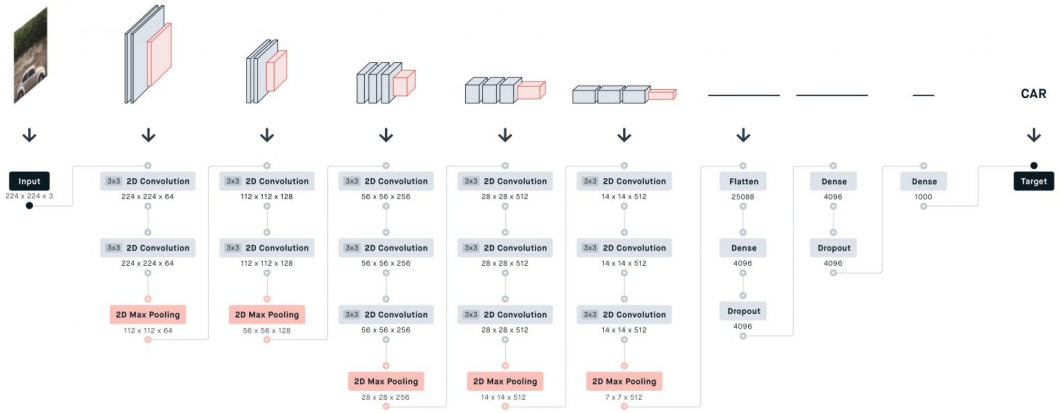

Figura 3.2: Struttura della VGG 16

Si vede come la rete accetta immagini di input di dimensione 224x224x3, permettendo di processare immagini a colori (a tre canali, RGB).

Partendo dal primo stack, vengono applicati 64 kernels, filtri convoluzionali bidimensionali di 3×3 con stride 1 e padding 1. All'output di 224x224x64 viene applicato poi un altro filtro convoluzionale bidimensionale identico al precedente. L'output, dimensionalmente inalterato, subisce una trasformazione con la sovrapposizione di un max pooling layer bidimensionale che produce un volume finale per il primo

layer di 112x112x64. Passando al secondo stack il numero di kernel raddoppia: 128 filtri convoluzionali bidimensionali, sempre da 3×3 producono un volume iniziale di 112x112x128, a cui viene applicato un altro filtro convoluzionale e infine un max pooling layer che produce un output finale di 56x56x128. Nel terzo stack viene prelevato l'input del layer precedente applicando 256 kernel di dimensioni 3×3, bidimensionali. Questo viene iterato per 3 volte. Al risultato, si sovrappone un max pooling layer per ottenere un output di 14x14x256 che finisce dritto al quarto layer. A questo punto, per 3 volte in successione, 512 kernels 3×3 manipolano i dati e in seguito all'applicazione di un max pooling si ottiene un volume finale di 14x14x512. Il quinto stack è molto simile al precedente: 3 layer convoluzionali e 1 max pooling, che produce un output di 7x7x512. Al sesto stack, inizia la vera e propria rete neurale profonda, costituita da un primo layer che appiattisce il volume di 7x7x512 attraverso 25088 neuroni fully connected a un dense layer di 4096 neuroni a loro volta connessi a un dropout layer di altrettanti neuroni. A questo punto si passa al settimo stack costituito da un primo dense layer di 4096 neuroni e un secondo dropout layer di altrettanti neuroni. Completa così l'architettura l'ottavo stack composto da l'output layer di tipo dense con 1000 neuroni, uno per ogni classe.

3.2.2 ResNet - Residual Network [2]

Nel 2015 Kaming progettò una rete rivoluzionaria che cambiò qualcosa che era stata alla base delle precedenti proposte. ResNet nacque dall'osservazione che con l'aumento di livelli ci potesse essere il rischio di imbattersi in peggioramenti della rete. Intuitivamente, reti neurali più profonde non dovrebbero performare peggio di quelle poco profonde, o almeno non durante l'allenamento quando non vi è alcun rischio di overfitting. Tuttavia, al crescere della profondità della rete questo non è sempre vero. Gli sviluppatori di ResNet ricondussero questo problema all'ipotesi che le mappature dirette sono difficili da allenare e proposero un rimedio, ovvero l'uso del blocco residuale: l'idea fu che la rete ora potesse imparare principalmente le differenze tra i layers in ingresso e in uscita dal blocco. I layer delle reti proposte variarono allora tra "18", "34", "50", "101", "152", "1202". Ad esempio, la più popolare cioè la ResNet50 disponeva 49 layer convolutivi e di uno fully connected.

Le ResNet sono delle reti "feed forward" con una "residual connection"; quest'ultima, costruita con diversi tipi di blocchi in base al tipo di architettura scelta. Prima di allora era molto diffuso il problema dell'annullamento del gradiente, la cui discesa, data dalla minimizzazione della funzione di errore, si riduce esponenzialmente attraverso la retropropagazione degli strati precedenti. In sostanza, il percorso lungo gli strati precedenti rendeva gli errori talmente piccoli da non permettere alla rete di apprendere. Con le ResNet iniziarono a vedersi reti con innumerevoli strati caratterizzati da un elevato grado di accuratezza. A questo punto è doveroso un piccolo approfondimento dell'architettura di questa rete. Si prenda ad esempio in considerazione la ResNet 34, la cui architettura è illustrata in figura 3.3.

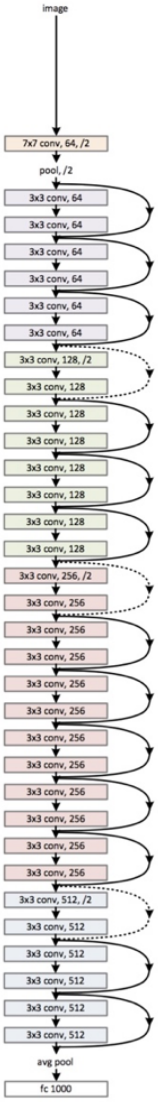

Figura 3.3: Struttura della ResNet 34

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

Figura 3.4: Tabella con struttura della ResNet in più versioni

Il primo step su una ResNet, prima di entrare nei layer che hanno un comportamento comune, è un blocco chiamato Conv1, che consiste nella seguente serie di operazioni: Convoluzione + Batch Normalization + MaxPooling. Quindi, prima c'è un'operazione di convoluzione. Nelle illustrazioni dell'architettura si può notare che vengono usati filtri di dimensione 7 e una feature map di dimensione 64. Tenendo conto di ciò, la dimensione di output di tale operazione avrà un volume di 112x112. Poiché ogni filtro di convoluzione (dei 64) fornisce un canale nell'uscita, ci si ritrova con un volume di output pari a 112x112x64. Poi viene eseguita la Batch Normalization, che non va ad influire in alcun modo nella dimensione dell'output. Infine, si ha un'operazione di Max Pooling (3x3) con uno stride pari a 2.

Negli step successivi si incontrano i 4 layer, di cui verranno fornite le caratteristiche generali, che costituiscono una ResNet. Ogni layer di questa rete è costituita da blocchi; questo perché quando la ResNet scende in profondità, normalmente lo fa aumentando il numero di operazioni all'interno di un blocco, lasciando invariato il numero di livelli totali. Un'operazione si riferisce a una convoluzione, una Batch Normalization e un'attivazione ReLU a un ingresso, tranne l'ultima operazione di un blocco, che non ha la ReLU. Pertanto, i blocchi si distinguono in: Basic Block (blocchi che includono 2 operazioni) e Bottleneck Block (blocchi che includono 3 operazioni).

3.2.3 DenseNet [3]

GaoHuang e il suo gruppo nel 2017, proposero le DenseNet, cioè delle CNN "Densely Connected", con dei layer densamente collegati tra loro in modo che l'output del precedente fosse direttamente collegato ai successivi. Concatenando le feature map apprese da differenti layer si ottiene un aumento di efficienza e aumenta la variazione sugli input: questa è la principale differenza rispetto alle ResNet. Come conseguenza diretta della concatenazione, le feature map sono quindi utilizzabili in tutti i layer successivi alla loro produzione e viene anche incoraggiato il riutilizzo all'interno della rete creando un modello sempre più compatto. La figura 3.5 riassume molto semplicemente questo nuovo concetto.

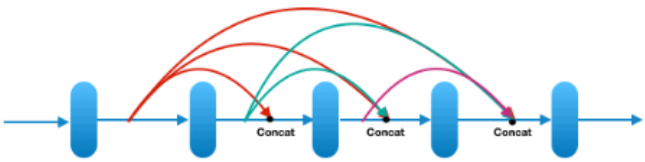

Figura 3.5: Modello astratto con la concatenazione applicata

Tra gli altri vantaggi riconosciuti, sono da sottolineare la riduzione della scomparsa del gradiente, la riduzione dei parametri della rete ma anche la qualità di propagazione del flusso di informazioni e del gradiente tra i vari layer; ogni layer ha infatti accesso diretto ai gradienti, a partire dalla funzione di perdita fino all'input. Può essere definito uno dei modelli architetturali meglio performanti per il task di classificazione e perciò è necessario analizzare più da vicino l'architettura di questa rete per capirne le potenzialità, prendendo in considerazione, ad esempio, la DenseNet-121.

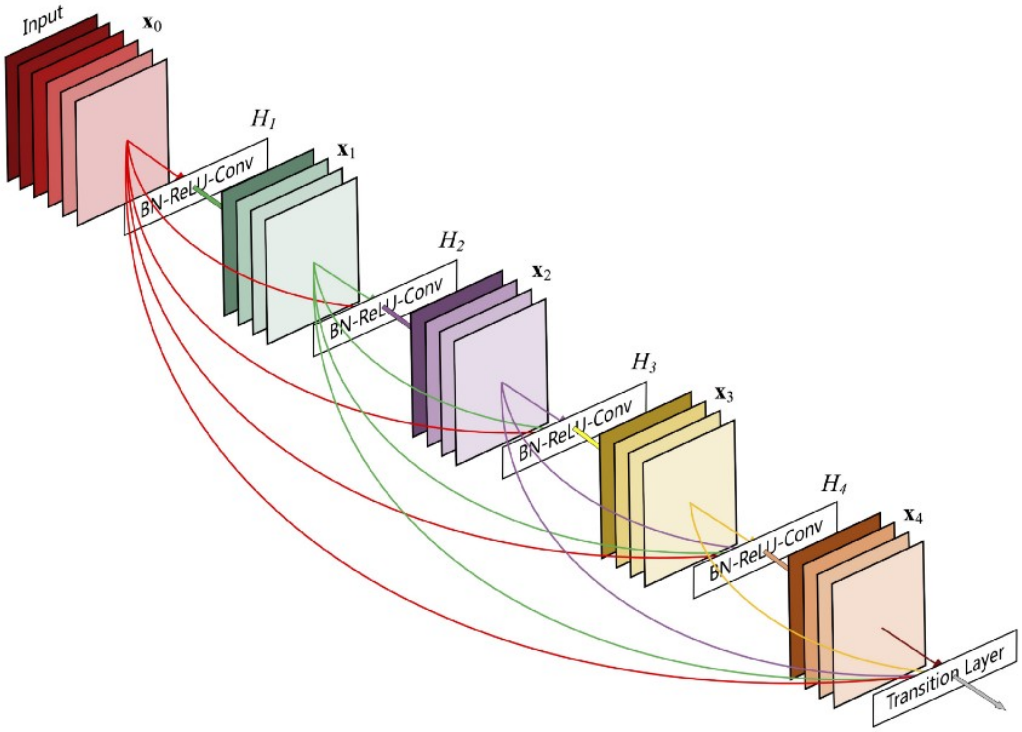

Figura 3.6: Blocchi convoluzionali di una Densenet

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figura 3.7: Tabella della struttura di una DenseNet

Come si può osservare nella figura 3.6, è essenziale chiarire che ogni blocco convoluzionale della rete, che verrà ovviamente trattato, è costituito da una serie di 3 operazioni: BatchNormalization -> ReLu -> Conv.

Chiarito questo si può iniziare a trattare la struttura di questa rete partendo dal concetto con la quale è stata presentata. Il concetto di connessioni dense viene rappresentato grazie ai dense block. Un dense block comprende un dense layer. Questi dense layer sono collegati tramite un circuito "denso", grazie al quale ogni dense layer riceve feature maps da tutti i livelli precedenti e le passa a tutti i livelli successivi. Le dimensioni delle feature (larghezza, altezza) rimangono invariate all'interno dello stesso blocco. Ogni dense layer, come illustrato in figura 3.8 è costituito da 2 operazioni convoluzionali: 1x1 conv (operatore di convoluzione convenzionale per l'estrazione di feature) e 3x3 conv (riduttore della profondità delle feature e conteggio dei canali).

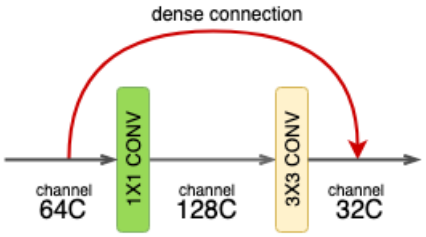

Figura 3.8: Dense layer

La DenseNet-121 comprende 6 dense layer per ogni dense block. La profondità dell'output di ogni strato è uguale al tasso di crescita del dense block stesso. Il tasso di crescita (o Growth rate (k)) è sostanzialmente il numero di canali emessi da un dense layer (1x1 conv → 3x3 conv). Gli autori della rete hanno utilizzato un valore di k = 32 per gli esperimenti pubblicati nel paper ufficiale della rete. Ciò significa che il numero di caratteristiche ricevute da uno dense layer (l), dal suo danse layer precedente (l-1), è 32. Questo è indicato come tasso di crescita perché dopo ogni strato, 32 caratteristiche del canale sono concatenate e alimentate come input allo strato successivo.

Alla fine di ogni dense block, il numero di feature maps si accumula e raggiunge un valore pari a:

$$input\_features + (numero\_di\_dense\_layers * growth\_rate)$$

Quindi per le feature di 64 canali che entrano in un dense block con 6 dense layer con tasso di crescita 32, il numero di canali accumulati alla fine del blocco sarà :

$$64 + (6 * 32) = 256.$$

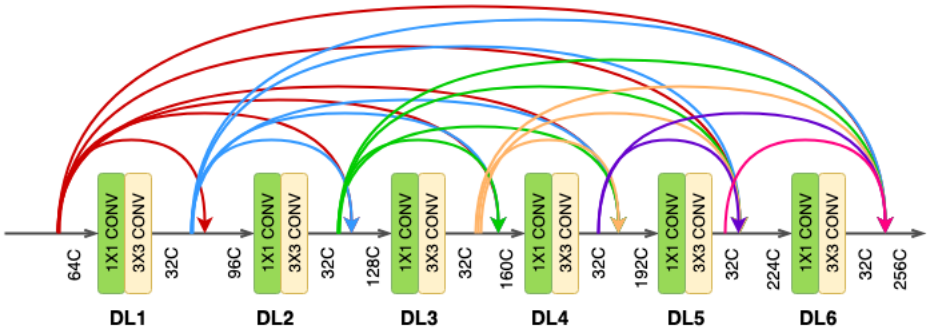

Figura 3.9: Dense block

Per ridurre questo numero di canali, viene aggiunto uno strato di transizione (o blocco) tra due dense block. Lo strato di transizione è costituito da: 1x1 conv e 2x2 Avg Pool. L'operazione 1x1 riduce il numero di canali alla metà; mentre il layer 2x2 Avg Pool è responsabile del downsampling delle caratteristiche in termini di larghezza e altezza.

La struttura appena descritta della DenseNet-121 può essere rappresentata come nella figura 3.10.

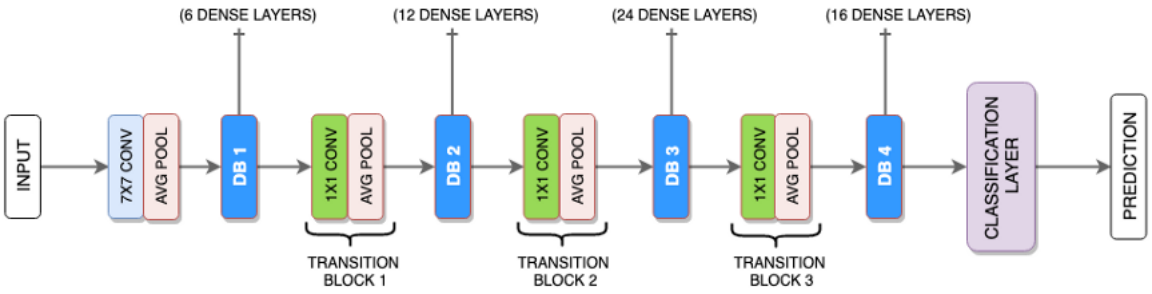

Figura 3.10: Struttura DenseNet

3.2.4 SqueezeNet [4]

La SqueezeNet è una delle architetture più recenti e riprende diversi concetti delle ResNet, proponendo una migliore architettura a livello di design. Dimensionalmente più piccola e con un numero ridotto di parametri, consente di effettuare un training distribuito in maniera più efficiente, il quale, permette di produrre meno overhead. Questi risultano essere i vantaggi di una architettura di dimensioni minori, che presenta una grande accuratezza, paragonabile a quella delle reti AlexNet [10]. In particolare, per ridurre il numero di parametri e la dimensione, si possono usare varie strategie, tra cui: sostituire i filtri con filtri 1x1 o con filtri 3x3, oppure cercare di diminuire il numero di canali di input (sottocampionare, o downsample) sulla rete in un momento successivo alla convoluzione, in modo che il layer convolutivo

abbia una mappa di attivazione più grande. Uno degli elementi che balza all'occhio osservando l'architettura è l'assenza di fully connected layer o di dense layer alla fine della cascata di moduli che compongono la rete, infatti, entrambi sono usati per permettere la classificazione e agiscono in modo totalmente differente; in questo caso, l'operazione di classificazione viene completata all'interno dei "moduli Fire" della rete. La figura 3.11 mostra la struttura di una generica Squeezenet.

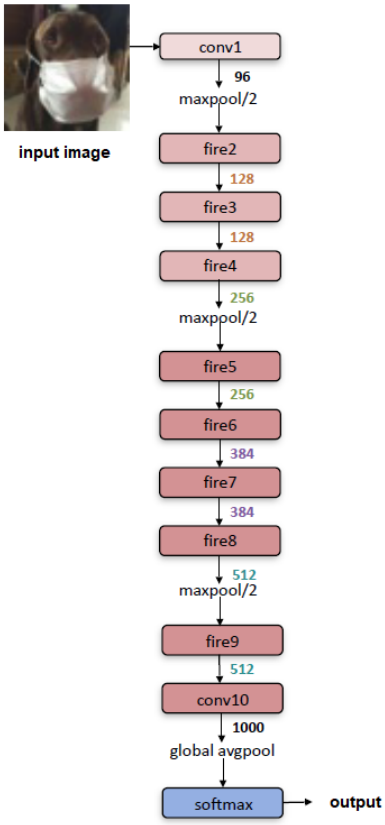

Figura 3.11: Struttura SqueezeNet

Come già affermato la Squeezenet è caratterizzata da una struttura molto semplice; infatti, inizia con un livello di convoluzione autonomo (conv1), seguito da 8 moduli Fire (fire2-9), e termina con un livello conv finale (conv10). Il numero di filtri per i Fire module viene gradualmente aumentato dall'inizio alla fine della rete. Il Max Pooling con uno stride di 2 viene eseguito dopo i livelli conv1, fire4, fire8 e conv10. Per quanto riguarda i moduli Fire, sono composti da: uno squeeze convolution layer (con solo filtri 1×1), che alimenta un expand layer che ha un mix di filtri di convoluzione 1×1 e 3×3.

Il modulo Fire è inoltre ideato su 3 dimensioni, dette iperparametri: s1×1, e1×1 ed e3×3.

- 1×1: 1×1 conv filters nello squeeze layer;
- e1×1 e e3×3: 1×1 e 3×3 conv filters nell'expand layer.

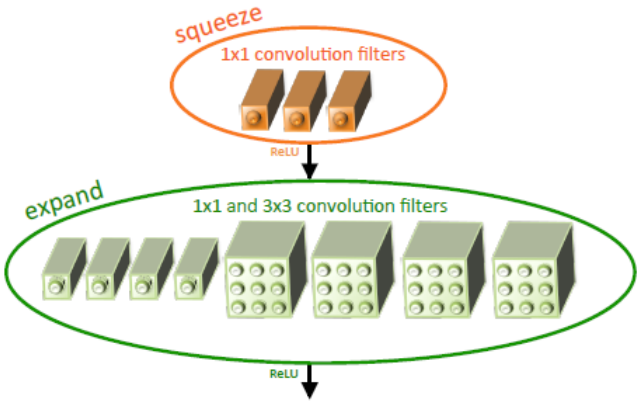

Figura 3.12: Struttura Fire Module (1)

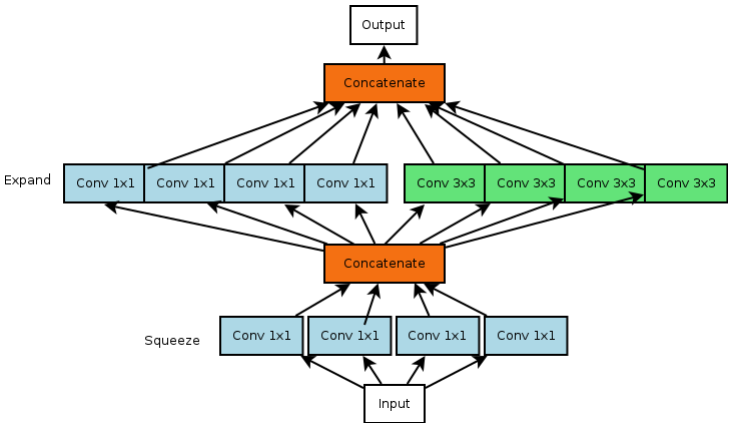

Figura 3.13: Struttura Fire Module (2)

3.2.5 EfficientNet [5]

EfficientNet è una rete neurale convoluzionale all'avanguardia che è stata addestrata e rilasciata al pubblico nel 2019. Con il tempo, i ricercatori hanno sperimentato e cercato di elaborare architetture sempre migliori per migliorare l'accuratezza dei modelli su diversi compiti, e con la EfficientNet non ci si è concentrati solo sul miglioramento della precisione, ma anche sull'efficienza dei modelli. Quest'ultimo obiettivo è stato centrato trattando efficacemente il ridimensionamento del modello. Ci sono tre dimensioni di ridimensionamento di una CNN: profondità, larghezza e risoluzione.

- *Profondità*: significa semplicemente quanto è profonda la rete e ciò è equivalente al numero di strati in essa contenuti. Questo parametro, come visto in altri modelli, è il modo più comune di ridimensionare, in quanto la profondità può essere aumentata o ridotta aggiungendo o rimuovendo i livelli. L'intuizione è che una rete più profonda può catturare funzionalità più ricche e complesse e generalizzare bene su nuove attività. In realtà non sempre le prestazioni della

rete migliorano. (es. I gradienti di fuga sono uno dei problemi più comuni che sorgono man mano che si va in profondità nella rete)

- *Larghezza*: significa quanto è ampia la rete. Una misura della larghezza, ad esempio, è il numero di canali in un livello Conv. Questa scala viene comunemente utilizzato quando si vuole mantenere il modello piccolo, perché delle reti più grandi solitamente tendono a catturare caratteristiche più dettagliate. Il problema è che con modelli meno profondi ma più ampi la precisione si satura rapidamente.
- *Risoluzione*: è la risoluzione dell'immagine che viene trasmessa a una CNN. Intuitivamente si può pensare che con immagini ad alta risoluzione le reti dovrebbero funzionare meglio. Nella realtà questo non scala linearmente, infatti il guadagno di precisione diminuisce molto rapidamente (es. aumentare la risoluzione da 500x500 a 560x560 non produce miglioramenti significativi).

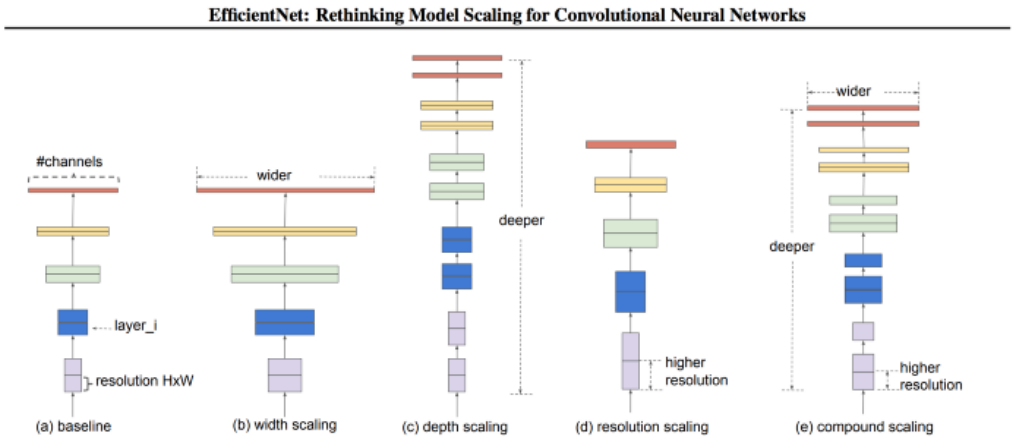

Figura 3.14: Ridimensionamento sulle reti

La figura 3.14 rappresenta il ridimensionamento di un modello nei 3 parametri appena descritti.

Si può evincere quindi che l'aumento di qualsiasi dimensione della rete (larghezza, profondità o risoluzione) migliora la precisione, ma il guadagno di precisione diminuisce per i modelli più grandi. Per questo motivo si è puntato sul ridimensionamento combinato di queste 3 dimensioni, ovviamente bilanciandole durante il ridimensionamento delle CNN per ottenere una maggiore precisione ed efficienza, in quanto la maggior parte delle volte che si attua un ridimensionamento manuale si traduce in una precisione ed efficienza non ottimali.

Gli autori della EfficientNet hanno proposto una tecnica di ridimensionamento semplice ma molto efficace che utilizza un **coefficiente composto δ** per ridimensionare in modo uniforme larghezza, profondità e risoluzione della rete in un modo di principio:

$$\boldsymbol{depth : d} = \alpha^{\phi}$$

$$\boldsymbol{width : w} = \beta^{\phi}$$

$$\boldsymbol{resolution : r} = \gamma^{\phi}$$

$$s.t.\ \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2\ (\alpha \geq 1, \beta \geq 1, \gamma \geq 1)$$

δ è un coefficiente specificato dall'utente che controlla quante risorse sono disponibili; mentre α, β e γ specificano come assegnare queste risorse rispettivamente a profondità, larghezza e risoluzione della rete. Il ridimensionamento non modifica le operazioni sui livelli, quindi è meglio avere prima una buona rete di base e poi ridimensionarlo lungo diverse dimensioni utilizzando il ridimensionamento composto proposto. Gli autori hanno ottenuto la loro rete di base eseguendo una ricerca sull'architettura neurale che ottimizza sia l'accuratezza che i FLOPS (numero di operazioni in virgola mobile eseguite in un secondo dalla CPU).

L'architettura di una EfficientNet viene mostrata in figura 3.15

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	28 × 28	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

Figura 3.15: Tabella con la struttura di una EfficientNet

L'elemento costitutivo principale di questa rete è MBConv (raffigurato nella figura 3.16), a cui viene aggiunta un'ottimizzazione di compressione ed eccitazione. Questi formano una connessione rapida tra l'inizio e la fine di un blocco convoluzionale. Le feature map dell'input vengono prima espanse utilizzando operatori di convoluzione 1x1 per aumentare la profondità delle feature map. Seguono convoluzioni 3x3 depth-wise e convoluzioni point-wise che riducono il numero di canali nella feature map di output. Esistono poi delle connessioni scorciatoia che collegano i livelli più stretti, mentre nei livelli più ampi, vengono favoriti i skip-layer (connessioni da saltare). Questa struttura aiuta a ridurre il numero complessivo di operazioni richieste e le dimensioni del modello.

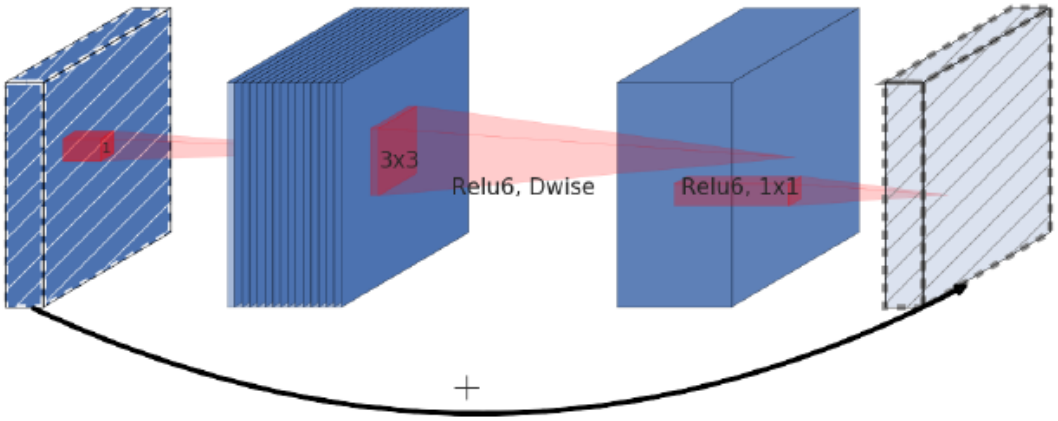

Figura 3.16: MBConv di una EfficientNet

Una volta realizzato la rete di base, bisognerà solamente ricercare i valori ottimali per i parametri di scala precedentemente discussi:

$$\alpha, \beta, \gamma, \delta$$

Per una EfficientNet-b0 sono stati ottenuti i seguenti valori ottimali: con

$$\delta = 1 -> \ \alpha = 1.2, \beta = 1.1, \gamma = 1.15 \ \ t.c. \ \alpha * \beta^2 * \gamma^2 \approx 2.$$

Mentre per tutti gli altri modelli di EfficienNet (b1-b7) i valori ottimali sono stati ottenuti ponendo α, β e γ (con i valori precedentemente trovati) come costanti, e variando il valore δ.

3.3 Tecniche di interpretability

Tra i vari metodi di interpretabilità precedentemente discussi, per il progetto sono state adottate alcune tecniche che verranno illustrate successivamente, in grado di applicare il concetto di Feature Visualization.

3.3.1 Activation Map (o Feature Map)

Nel tempo, riprendendo ciò che è stato detto nel capitolo 2.2, sono stati sviluppati diversi approcci per comprendere al meglio le CNN, in parte come risposta alla critica comune che le caratteristiche interne apprese in una CNN non siano interpretabili.

La tecnica di visualizzazione più diretta consiste nel mostrare le attivazioni della rete durante il "forward pass", ossia quel processo di calcolo dei valori di output partendo dai dati di input, attraversando tutti i neuroni dal primo all'ultimo strato di una rete. Quindi, detto in parole semplici, le funzioni di attivazione aiutano a decidere se un neurone debba essere attivato, questo aiuta a determinare se le informazioni che il neurone sta ricevendo sono rilevanti per l'input. La funzione

di attivazione è una trasformazione non lineare che avviene su un segnale di input, e l'output trasformato viene inviato al neurone successivo. Le *Activation maps* [2] sono solo una rappresentazione visiva di questi numeri di attivazione dei vari livelli della rete, mentre una data immagine avanza come risultato di operazioni algebriche lineari.

Per poter però illustrare un'activation map è necessario fare un passo indietro e definire un altro piccolo concetto. Le CNN derivano il loro nome dall'operatore "convoluzione". Lo scopo principale di questa operazione nelle reti CNN è quello di estrarre le funzionalità dall'immagine di input. La convoluzione preserva la relazione spaziale tra i pixel imparando le caratteristiche dell'immagine utilizzando piccoli quadrati di dati di input. Senza entrare nei dettagli matematici, verrà direttamente illustrato sulle immagini. Ogni immagine può essere considerata come una matrice di valori di pixel. Per esempio si prenda un'immagine 5x5, i cui valori dei pixel sono solo 0 e 1 (per un'immagine in scala di grigi, i valori dei pixel vanno da 0 a 255, perciò la matrice verde in figura 3.17 è un caso speciale in cui i valori dei pixel sono solo 0 e 1).

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Figura 3.17: Matrice 5x5

Inoltre, si consideri una matrice 3x3 come in figura 3.18.

1	0	1
0	1	0
1	0	1

Figura 3.18: Matrice 3x3

Quindi, la convoluzione dell'immagine 5x5 consiste nello scorrere, di un pixel alla volta (detto anche "stride"), la matrice 3x3 sopra l'immagine originale. Calcolando la moltiplicazione delle due matrici, e aggiungendo i risultati dell'operazione, si ottiene il numero intero finale che forma un singolo elemento della nuova matrice di output. Si noti che la matrice 3×3 "vede" solo una parte dell'immagine di input in ogni stride. In figura 3.19 vengono raffigurate le precedenti matrici impegnate nell'operazione appena descritta, più la loro risultante.

Parlando di CNN, la terminologia corretta per la matrice 3x3 appena osservata è: "filter", "kernel" o "feature detector". Invece la matrice risultante ottenuta dall'operazione di convoluzione viene detta: "Convolved Feature", "Activation Map" o "Feature Map". È importante notare che i filtri agiscono come rilevatori di

[2] Esempio di utilizzo e implementazione

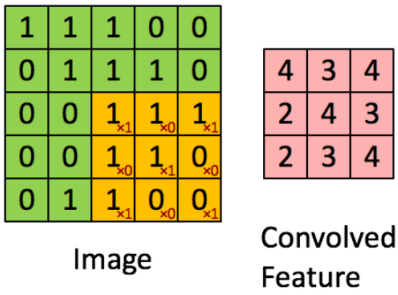

Figura 3.19: Operazione di convoluzione

caratteristiche dell'immagine originale data in input. È quindi evidente che per valori diversi della matrice "filter" verranno prodotte Activation map differenti per la stessa immagine di input. Questo lo si può notare nella figura 3.20.

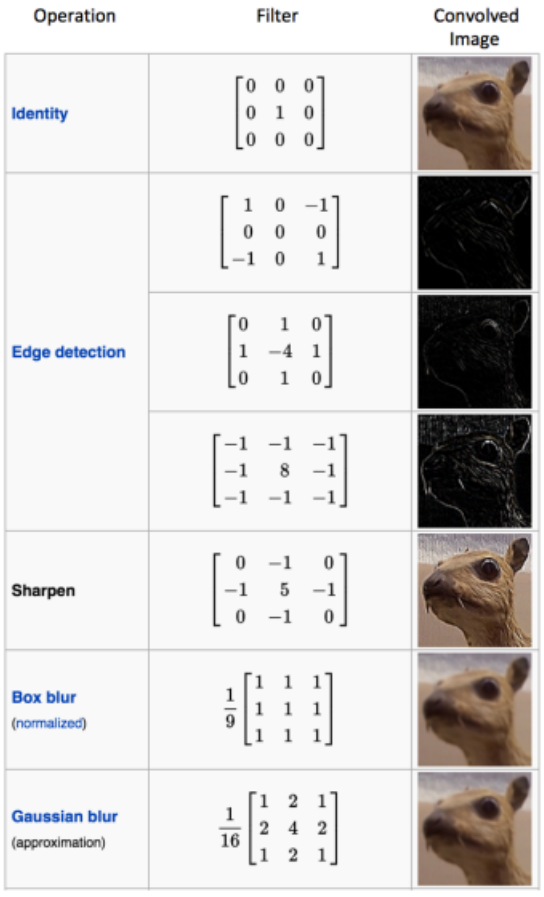

Figura 3.20: Effetto della modifica dei filtri su un'immagine

Quindi si possono eseguire operazioni come: rilevamento bordi, contrasto e sfocatura semplicemente modificando i valori numerici della matrice di filtri prima dell'operazione di convoluzione. Questo significa che filtri differenti possono rilevare

altrettante caratteristiche da un'immagine, ad esempio bordi, curve ecc...

In pratica invece, una CNN apprende da sola i valori di questi filtri durante il processo di addestramento, anche se bisogna comunque specificare parametri come numero di filtri, dimensione del filtro, architettura della rete, ecc... prima del processo di addestramento. Maggiore è il numero di filtri, più caratteristiche dell'immagine vengono estratte e migliore diventa la capacità della rete nel riconoscere gli oggetti nelle immagini. La dimensione della feature map viene controllata da tre parametri che devono essere specificati prima della fase di convoluzione:

- *Depth*: La profondità corrisponde al numero di filtri che vengono utilizzati per l'operazione di convoluzione. Ad esempio, se vengono utilizzati tre filtri distinti, vengono a sua volta prodotte tre diverse mappe di caratteristiche.

- *Stride*: In italiano il passo, è il numero di pixel di cui si fa scorrere la matrice di filtri sulla matrice di input. Quando il passo è 1, i filtri si spostano un pixel alla volta. Quando il passo è 2, i filtri saltano di 2 pixel alla volta mentre scorrono. Avere un passo più ampio produrrà feature map più piccole.

- *Zero-padding*: A volte, è conveniente riempire la matrice di input con zeri in corrispondenza del bordo, in modo da poter applicare il filtro agli elementi confinanti della matrice iniziare. Quindi lo zero-padding permette di controllare la dimensione delle feature map. Aggiungendo lo zero-padding si avrà una convoluzione ampia, mentre evitandone l'utilizzo produrrà una convoluzione stretta.

Di seguito (figura 3.21) un esempio di Activation map, senza però entrare troppo nello specifico, in quanto verranno successivamente analizzate più nel dettaglio nel corso di questo capitolo. Fornendo in input un'immagine ed estraendo le feature map di ogni layer di una CNN, il risultato ottenuto sarà il seguente.

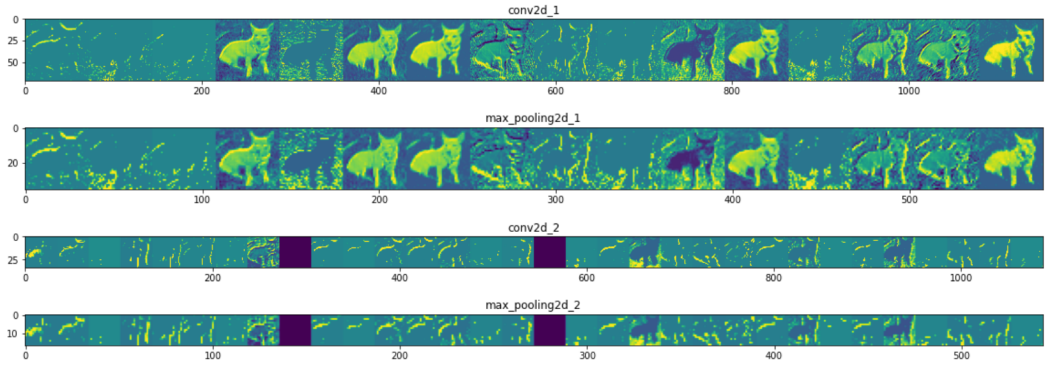

Figura 3.21: Esempio di Activation map

3.3.2 Saliency Map [6] [7] [8]

La Saliency Map è un altro metodo utilizzato per interpretare le previsioni delle CNN. Questa è probabilmente la tecnica più antica e utilizzata nel deep learning. Fondamentalmente, la Saliency map di un'immagine di input, indica le parti di essa che contribuiscono maggiormente all'attività di uno specifico livello nella rete, o alla decisione della rete nel suo insieme. Data un'immagine di input ad una CNN, esistono tre approcci principali per ottenere una Saliency map.

Il primo approccio proposto utilizza le *reti deconvoluzionali*. Queste reti, per poter riconoscere quali caratteristiche nell'immagine di input sta cercando uno strato intermedio della rete, ricostruiscono l'input stesso dall'attivazione del livello in cui si trova. Per fare ciò, le operazioni eseguite tra l'input ed uno specifico livello vengono invertite. In particolare, la deconvoluzione viene utilizzata come inversa della convoluzione (con la versione trasposta degli stessi filtri), l'unpooling viene utilizzata come inversa del pool, e la ReLU viene utilizzata come inversa di se stessa, ma bloccando i valori negativi del segnale che va indietro dallo spazio di attivazione fino all'immagine. Vale la pena notare che, poiché l'operazione di pooling non è invertibile, gli autori di questa rete hanno utilizzato un modulo chiamato "switch" nella rete deconvoluzionale, per recuperare le posizioni dei massimi nel passaggio in avanti (forward pass). L'intero processo può essere schematizzato come in figura 3.22.

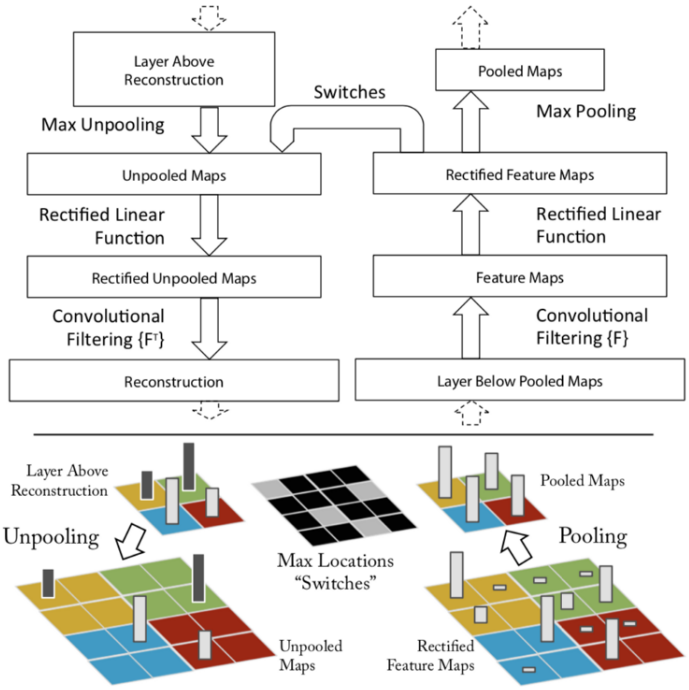

Figura 3.22: Schema generale di una rete deconvoluzionale

Nella figura 3.23 vengono illustrate le Saliency map prodotte, ricostruendo l'imma-

gine data in input da uno strato qualsiasi di una generica CNN. Come si può vedere, il livello della rete, dalla quale sono state estratte le Saliency map, può riconoscere le informazioni semantiche specifiche della classe di appartenenza dell'immagine.

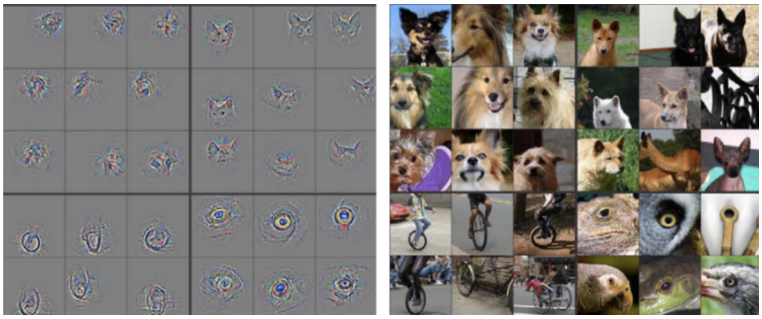

Figura 3.23: Esempio di saliency map prodotte da una rete deconvoluzionale

Il secondo approccio, e probabilmente il più diretto per ottenere una Saliency map, consiste nell'utilizzare l'*algoritmo di backpropagation* per calcolare i gradienti delle previsioni che il modello computa in base all'ingresso della rete (si noti che nell'algoritmo di backpropagation utilizzato per l'addestramento della rete, i gradienti sono calcolati rispetto ai parametri della rete). Usando la retropropagazione, si possono quindi evidenziare i pixel dell'immagine di input (come in figura 3.24) in base alla quantità del gradiente che ricevono, mostrando il loro contributo al punteggio finale.

Figura 3.24: Esempio di saliency map prodotte utilizzando l'algoritmo di backpropagation

Successivamente è stato dimostrato che il metodo di backpropagation è abbastanza simile al metodo in qui vengono utilizzate le reti deconvoluzionali, ma con una grande differenza; nella rete deconvoluzionale, la ReLU blocca un segnale all'indietro (backward signal) negativo proveniente dall'attivazione di uno strato intermedio, mentre il metodo di retropropagazione blocca il gradiente, se l'ingresso della ReLU attraverso il forward pass, era negativo. Combinando questi approcci è stato proposto l'algoritmo di *backpropagation guidato* come terzo modo per ottenere Saliency map.

Infatti, invece di mascherare il segnale in base alle posizioni dei valori negativi del segnale di ingresso in forward-pass (backpropagation) o dei valori negativi del segnale che scorre in una rete deconvoluzionale, viene mascherato solo se si verifica uno di questi casi. La figura 3.25 descrive nel dettaglio il funzionamento di questo algoritmo.

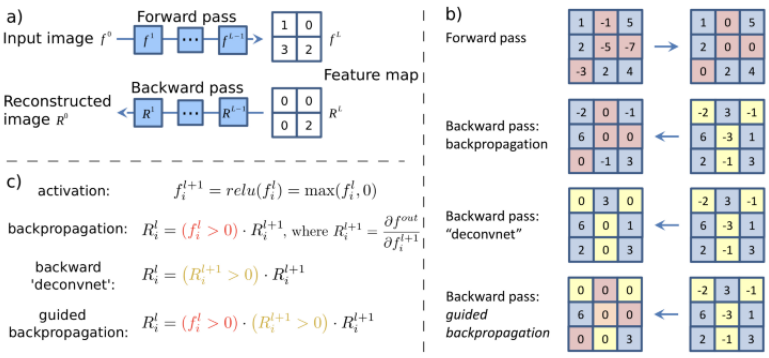

Figura 3.25: Descrizione tecnica dell'algoritmo di backpropagation guidato

In figura 3.26 invece, si possono notare alcune Saliency map prodotte dal metodo di retropropagazione guidata appena discusso. Come si può vedere, questo algoritmo è abbastanza predisposto a catturare Saliency map ad alta risoluzione e molto nitide.

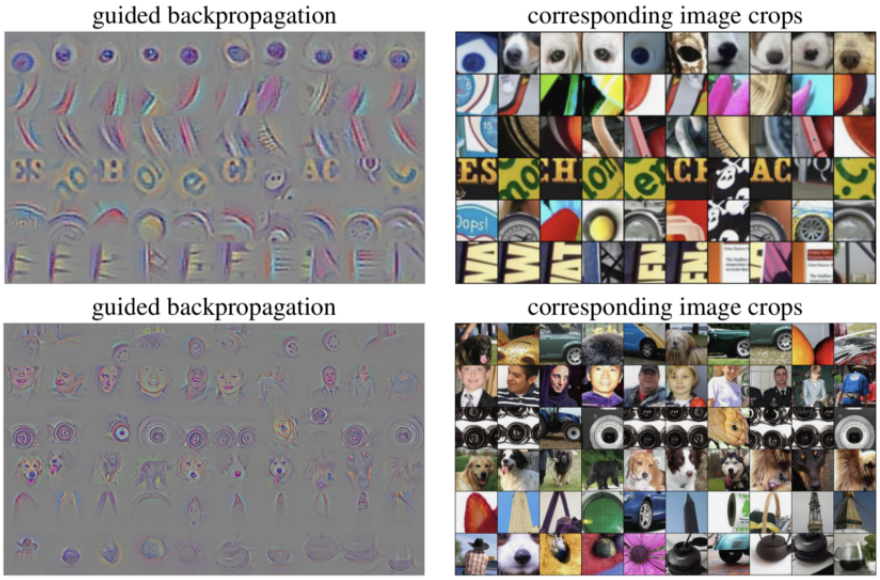

Figura 3.26: Esempio di saliency map prodotte utilizzando l'algoritmo di backpropagation guidato

Sebbene queste mappe siano principalmente utilizzate per interpretare le CNN, poiché il concetto di gradiente esiste in tutte le reti neurali, è possibile tuttavia

utilizzarlo per qualsiasi rete neurale artificiale arbitraria. Pertanto, potrebbe essere considerato un metodo di interpretazione indipendente dal modello.

3.3.3 Integrated Gradient [9]

L'Integrated Gradient è una tecnica di interpretabilità per reti neurali profonde che, come anche le altre tecniche di interpretability appena descritte, permette di visualizzare l'importanza delle caratteristiche presenti in un'immagine data in input, che contribuiscono alla previsione del modello. Questa tecnica va fondamentalmente a calcolare il gradiente dell'output previsto di un modello, rispetto alle sue caratteristiche di input, senza richiedere alcuna modifica alla rete neurale profonda originale. Inoltre, può essere applicato a qualsiasi dato in ingresso come: immagine, testo o anche dati strutturati. In genere, l'Integrated Gradient può essere utilizzato per:

- Comprendere l'importanza delle caratteristiche estraendo le regole della rete utilizzata;
- Debug delle prestazioni dei modelli di deep learning;
- Identificare i dati distorti comprendendo le caratteristiche importanti che contribuiscono alla previsione.

Questa tecnica basa l'intero funzionamento su due assiomi che devono essere sempre soddisfatti: la *sensibilità* e l'*invarianza d'implementazione*. Per calcolare la sensibilità, viene stabilita un'immagine di riferimento (baseline image) come punto di partenza; poi viene costruita una sequenza di immagini, che deriva dall'alterazione dell'immagine di base, sulla quale poi viene calcolato il gradiente. L'invarianza di implementazione invece è soddisfatta quando due reti sono funzionalmente equivalenti, ovvero quando i loro output sono uguali per tutti gli input nonostante abbiano implementazioni molto diverse. Inoltre, l'invarianza si ha quando le due reti hanno assegnazioni identiche per la stessa immagine di input e l'immagine di base.

A questo punto è utile riassumere tutto quello che si è detto descrivendo il workflow che si ha quando viene applicato l'Integrated Gradient ad una CNN.

1. Si parte da un'immagine di riferimento che può essere un'immagine nera i cui pixel sono tutti settati a zero, un'immagine completamente bianca, oppure anche un'immagine casuale.
2. Viene poi generata un'interpolazione lineare tra l'immagine di riferimento e quella originale. Come si nota nella figura 3.27, le immagini alterate sono il risultato di piccoli step (α) nello spazio delle caratteristiche tra l'immagine di riferimento e l'immagine di input. Aumentando α costantemente, aumenta anche l'intensità di ogni immagine interpolata.
3. Vengono calcolati i gradienti per misurare la relazione tra le modifiche di una feature e le modifiche nelle previsioni del modello. Il gradiente informa quale

Figura 3.27: Interpolazione dell'immagine

pixel ha più influenza sulle probabilità della classe predetta dalla rete. La variazione della variabile porterà ad una modifica dell'output; questa però verrà assegnata in modo tale che aiuti a calcolare caratteristiche più importanti dell'immagine di input. Una variabile che non influisce sull'output non riceve alcuna attribuzione.

4. Calcolare l'approssimazione numerica attraverso la media dei gradienti

5. Scalare l'Integrated gradient all'immagine di input per garantire che i valori di attribuzione accumulati su più immagini interpolate siano tutti nelle stesse unità. Viene applicata a questo punto l'Integrated gradient direttamente sull'immagine di input con le importazioni di pixel, come viene illustrato nella figura 3.28.

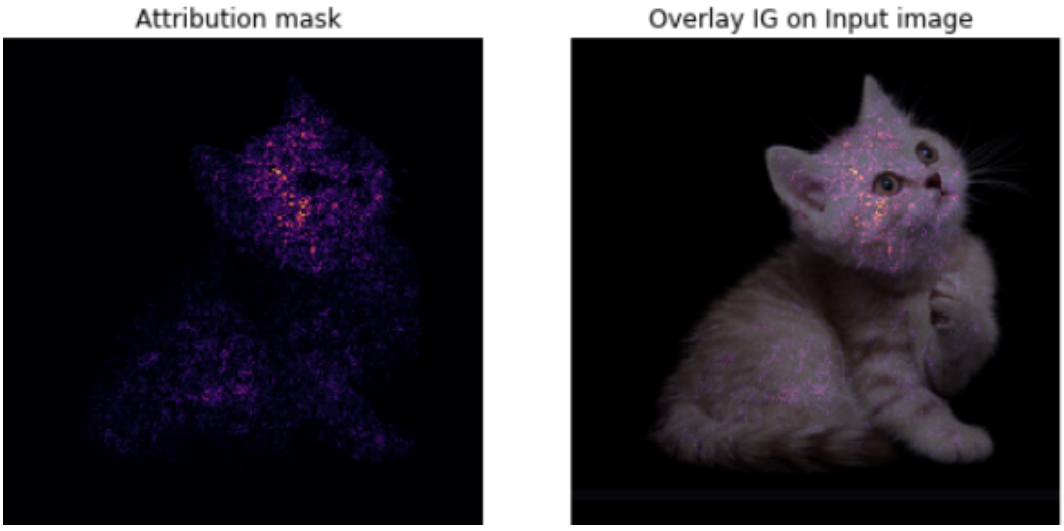

Figura 3.28: Esempio dell'Integrated gradient applicato sull'immagine

Attraverso la descrizione del workflow di questa tecnica si può notare che l'Integrated gradients attribuisce le previsioni di una rete profonda direttamente ai suoi input. Inoltre, è implementabile utilizzando poche chiamate all'operatore gradiente e allo stesso tempo può essere applicato a una varietà di reti neurali profonde. Questo lo rende di sicuro uno dei metodi migliori per poter interpretare una rete neurale.

3.4 Implementazione delle tecniche di interpretability applicate alle CNN

Questa sezione sarà dedicata all'implementazione delle tecniche di interpretazione appena descritte, sulle CNN scelte per la classificazione delle immagini.

3.4.1 Implementazione delle Activation Map

Le Activation Map sono state utilizzate esclusivamente per la EfficientNet (nelle prossime sezioni si capirà meglio il perché), ma sono ricavabili da qualsiasi CNN esistente. Infatti, la discussione di come sono state implementate può essere portata avanti utilizzando, ad esempio, una VGG, la cui struttura è molto più comprensibile rispetto alla struttura della EfficientNet.

La possibilità di poter implementare questa tecnica su ogni rete, deriva dal fatto che esiste un procedimento comune per poter ottenere le varie mappe. Ottenere le feature map è in realtà piuttosto semplice. L'idea generale che sta dietro a questa tecnica, infatti, è quella di passare un'immagine di input attraverso la CNN e di registrare le attivazioni intermedie. Infine, vengono selezionate casualmente alcune feature map che saranno poi visualizzate.

Per poter registrare le attivazioni intermedie è stata adottata una particolare soluzione basata sull'utilizzo degli **hooks**. Gli hooks sono delle specifiche funzioni direttamente collegate a ogni livello di una CNN, che vengono richiamate ogni volta che quest'ultimi vengono utilizzati. In pratica consentono di bloccare l'esecuzione del forward pass o del backward pass su un modulo specifico ed elaborare i suoi ingressi e uscite.

In PyTorch, si può registrare un hook nei seguenti modi:

- forward prehook (eseguito prima del forward pass);
- forward hook (eseguito dopo del forward pass);
- backward hook (eseguito dopo il backward pass).

In questo caso si è scelto, indistintamente, il secondo approccio. Il forward hook viene registrato con il metodo "register_forward_hook(hook)"; manipolando correttamente il valore di ritorno di questo metodo è possibile rimuovere l'hook da ogni modulo elaborato. Tutta questa procedura viene illustrata nello snippet di codice in figura 3.29.

Fatto ciò, l'hook verrà chiamato dopo ogni forward pass di ogni strato convoluzionale.

A questo punto è necessario vedere un esempio per poter capire a quali risultati porta questo approccio. Come anticipato prima, verrà fornita un'immagine casuale in input ad una VGG per poterne poi estrarre le feature map di alcuni livelli. I vari strati convoluzionali della VGG verranno chiamati in questa maniera: blockX_convY.

```
hook_handles = []

for layer in model.modules():
    if isinstance(layer, torch.nn.modules.conv.Conv2d):
        handle = layer.register_forward_hook(save_output)
        hook_handles.append(handle)
```

Figura 3.29: Codice per la registrazione dell'hook di ogni layer

Ad esempio, il secondo filtro nel terzo blocco di convoluzione si chiama block3_conv2. Fornendo quindi un'immagine di input, eseguendo la procedura precedentemente descritta ed estraendo 8 feature map per layer, il risultato non sarà altro che quello riportato in figura 3.30.

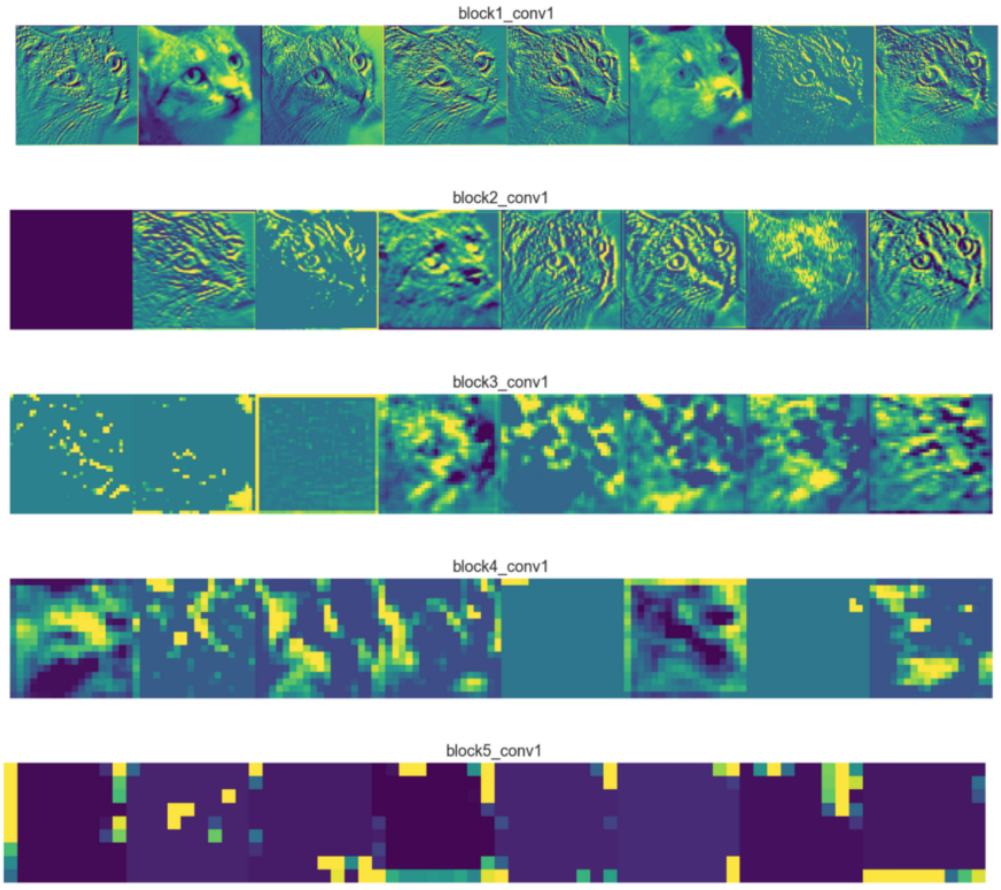

Figura 3.30: 8 feature map per ogni layer

Da questi risultati è possibile analizzare diverse cose; prima di tutto, le aree luminose non sono altro che le regioni "attivate", il che significa che il filtro ha rilevato il pattern che stava cercando.

Inoltre, ci sono alcune osservazioni interessanti da fare sulle feature map che si ottengono mentre si avanza nei livelli. È possibile evidenziarle meglio se ci si concentra su una singola feature map per livello, come raffigurato in figura 3.31.

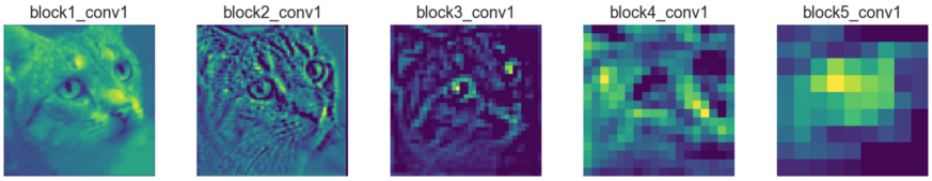

Figura 3.31: Singola feature map per layer

Si può notare che le feature map del primo livello (block1_conv1) conservano la maggior parte delle informazioni presenti nell'immagine. Nelle architetture CNN i primi strati di solito fungono proprio da rilevatori di bordi. Man mano che si va più in profondità nella rete, le feature map assomigliano meno all'immagine originale e più ad una rappresentazione astratta di essa. Come si puoi vedere in block3_conv1 il gatto è in qualche modo visibile, ma dopo diventa irriconoscibile. Il motivo è che le feature map più profonde codificano concetti di alto livello come "naso di gatto" o "orecchio di cane" mentre le feature map di livello inferiore rilevano bordi e forme semplici. Ecco perché le mappe più profonde contengono meno informazioni sull'immagine e più sulla classe dell'immagine. È vero che codificano ancora funzioni utili, ma sono meno interpretabili visivamente dal cervello umano. Le feature map quindi, diventano più rade man mano che andiamo più in profondità, il che significa che i filtri rilevano meno caratteristiche. Ha senso perché i filtri nei primi livelli rilevano forme semplici e ogni immagine le contiene, andando però più in profondità si iniziano a cercare elementi più complessi che non appaiono in tutte le immagini. Ecco perché nella figura 3.30 con 8 feature map per livello, si vedono più mappe delle caratteristiche vuote man mano che si va più in profondità (block4_conv1 e block5_conv1).

3.4.2 Implementazione delle Saliency Map

Le Saliency Map sono state utilizzate come tecniche di intepretabilità dei modelli che appartengono alla libreria torchivision, ossia: ResNet 18, Vgg 16, DenseNet 121 e SqueezeNet1_0. Questo perché, per poter implementare queste mappe al meglio, è stato utilizzato un toolkit specifico chiamato FlashTorch, il quale è stato ideato e sviluppato solo su queste reti.

Di seguito verrà spiegato, utilizzando anche degli snippet di codice, come questo sia stato possibile. La libreria FlashTorch [3] è stata inoltre progettata con delle funzioni, contenute in "utils", per semplificare la manipolazione dei dati. Infatti, prima di tutto verranno applicate alcune trasformazioni all'immagine presa d'esempio per rendere la sua forma, la sua grandezza ed altre caratteristiche ad essa associate, adatti come input a una CNN. Verrà utilizzata come esempio l'immagine "great grey owl", rappresentata in figura 3.32 (prima e dopo le trasformazioni applicate).

[3] Codice sorgente FlashTorch

Figura 3.32: Immagine di input trasformata

Una volta manipolata l'immagine di input, si passa a quello che viene definito il fulcro nella creazione di Saliency map: l'operazione di backpropagation. FlashTorch fornisce la classe Backprop, che si occupa di eseguire proprio questa operazione. Nell'istanziazione della classe Backprop (figura 3.33), viene accettato un modello pre-addestrato appartenente a torchvision e di conseguenza viene registrato l'hook per tutti i layer della rete interessati, in modo che vengano estratti i gradienti intermedi per la visualizzazione delle mappe. Questi gradienti intermedi non sono immediatamente disponibili, a causa di come è progettato PyTorch, ed è proprio questo uno dei motivi per cui è stato sviluppato FlashTorch.

```
from flashtorch.saliency import Backprop

model = models.resnet18(pretrained=True)

backprop = Backprop(model)
```

Figura 3.33: Codice dell'istanziazione della classe Backprop

Prima però di calcolare il gradiente, è necessario ricavare i gradienti della target class (classe di destinazione) rispetto all'immagine di input. Tuttavia, il modello è pre-addestrato sul dataset "ImageNet" costituito da 1000 classi, pertanto la previsione finale dell'input non è altro che un modello, il quale associa ad ogni classe di queste mille la probabilità di essere presente nell'immagine. Perciò viene prima individuato il valore della classe di destinazione (nel caso in esempio, il "grande gufo grigio") tra questi 1000 valori per evitare calcoli inutili e concentrarsi solo sulla relazione tra l'immagine di input e la target class.

Una volta identificata anche la target class, viene calcolato il gradiente e si ottiene il risultato in figura 3.34.

Si possono apprezzare i pixel nell'area in cui è presente l'animale, i quali hanno un'influenza maggiore sulla previsione. Ma allo stesso tempo è ancora un po' "rumoroso", infatti il segnale è abbastanza diffuso e non dice molto sulla percezione del gufo da parte della rete neurale.

```
from flashtorch.utils import visualize

backprop = visualize (input_image, class_target)
```

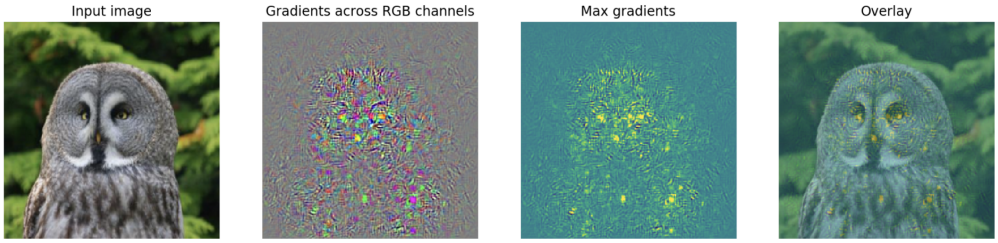

Figura 3.34: Saliency map dopo l'operazione di backpropagation

Per poter migliorare questa situazione, facendo anche riferimento alla parte teorica delle Saliency map precedentemente discussa, è stato introdotto un modo per poter ridurre il rumore durante il calcolo del gradiente. Questo è possibile grazie alla backpropagation guidata, tramite la quale, i neuroni che non hanno effetti, o hanno effetti negativi, sul valore di previsione di una certa classe target vengono mascherati e ignorati. In tal modo, viene impedito il flusso del gradiente attraverso tali neuroni, con conseguente minor rumore. Questo metodo viene implementato anche da FlashTorch, ed è possibile utilizzarlo semplicemente passando "guided=true" al metodo "visualize" precedentemente illustrato. Ottenendo così il risultato rappresentato nella figura 3.35.

```
from flashtorch.utils import visualize

backprop = visualize (input_image, class_target, guided = "true")
```

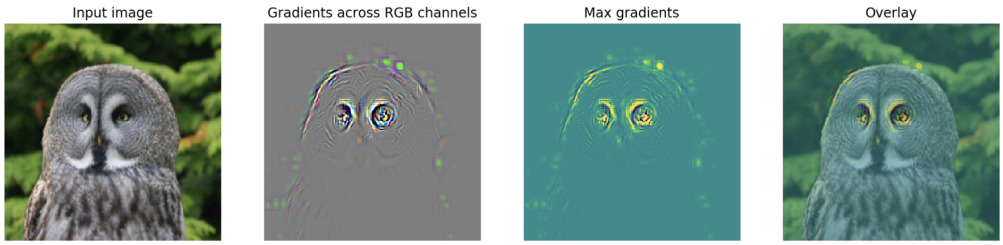

Figura 3.35: Saliency map dopo l'operazione di backpropagation guidata

Ora si può vedere chiaramente che la rete sta prestando attenzione agli occhi infossati e alla testa rotonda di un gufo. Queste sono le caratteristiche che hanno "convinto" la rete a classificare l'oggetto come "great grey owl".

3.4.3 Implementazione dell'Integrated Gradient

L'Integrated Gradient è un metodo di attribuzione assiomatica semplice ma allo stesso tempo molto potente, che non richiede alcuna modifica della rete originale. Proprio per questo motivo, l'Integrated gradient è stato implementato su tutte le CNN

a disposizione, utilizzando un particolare strumento chiamato Captum [4]. Captum è una libreria open source ed estensibile, utilizzata prettamente per l'interpretabilità del modello. Questa fornisce un'implementazione generica dei gradienti integrati che possono essere utilizzati con qualsiasi modello di PyTorch. Essa fornisce a ricercatori e sviluppatori un modo semplice per capire quali caratteristiche di un'immagine stanno contribuendo all'output di un modello. Quindi verrà fornito un quadro generale di come è stata utilizzata questa libreria per implementare l'Integrated Gradient.

Si prenda in considerazione un modello pre-addestrato tra quelli utilizzati, ad esempio la ResNet18. Prima di tutto, deve essere manipolata l'immagine che si vuole dare in input alla rete attraverso delle funzioni di trasformazione e normalizzazione, ottenendo quindi la seguente immagine iniziale.

Figura 3.36: Immagine manipolata per darla in input alla ResNet 18

A questo punto l'immagine può essere passata alla ResNet, la quale predice la classe alla quale è associata all'immagine di input. Si tratta della stessa procedura descritta per l'implementazione delle Saliency map con FlashTorch. Si può proseguire con il calcolo dell'Integrated gradient, per poi visualizzarlo direttamente sull'immagine. A livello matematico, questa tecnica consiste nel calcolare l'integrale dei gradienti dell'output del modello per la classe prevista "pred_label_idx" rispetto ai pixel dell'immagine di input ottenuto partendo dall'immagine di riferimento (procedura approfondita nel capitolo 3.3.3).

Questa prima fase può essere riscritta sottoforma di codice come raffigurato nello snippet di codice in figura 3.37.

Infine, si può visualizzare l'immagine e le relative attribuzioni sovrapponendo queste ultime all'immagine di partenza, come illustrato in figura 3.38.

[4]Codice sorgente Captum

```
output = model(input)
output = F.softmax(output, dim=1)
prediction_score, pred_label_idx = torch.topk(output, 1)

pred_label_idx.squeeze_()
predicted_label = idx_to_labels[str(pred_label_idx.item())][1]
print('Predicted:', predicted_label, '(', prediction_score.squeeze().item(), ')')
```

```
Predicted: goose ( 0.4569324851036072 )
```

```
integrated_gradients = IntegratedGradients(model)
attributions_ig = integrated_gradients.attribute(input, target=pred_label_idx, n_steps=200)
```

Figura 3.37: Snippet di codice della predizione della classe

```
default_cmap = LinearSegmentedColormap.from_list('custom blue',
                                                 [(0, '#ffffff'),
                                                  (0.25, '#000000'),
                                                  (1, '#000000')], N=256)

_ = viz.visualize_image_attr(np.transpose(attributions_ig.squeeze().cpu().detach().numpy(), (1,2,0)),
                             np.transpose(transformed_img.squeeze().cpu().detach().numpy(), (1,2,0)),
                             method='heat_map',
                             cmap=default_cmap,
                             show_colorbar=True,
                             sign='positive',
                             outlier_perc=1)
```

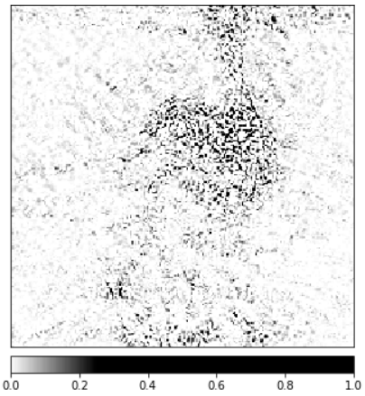

Figura 3.38: Snippet di codice per la visualizzazione dell'immagine con le relative attribuzioni

Capitolo 4

Risultati e discussioni

4.1 Metrica di analisi utilizzata

Valutare un algoritmo di apprendimento automatico è una parte essenziale di un progetto. Esistono moltissime metriche per la valutazione di un algoritmo, ma la maggior parte delle volte viene utilizzata la Classification accuracy per misurare le performance di un modello concentrato sulla classificazione di immagini.

Inoltre, questa particolare metrica che poi verrà trattata, viene molto spesso affiancata alle curve di apprendimento (learning curve). Una curva di apprendimento è un grafico che tiene traccia delle prestazioni di apprendimento del modello. Le curve di apprendimento sono uno strumento diagnostico ampiamente utilizzato nell'apprendimento automatico per algoritmi che apprendono in modo incrementale da dataset di addestramento. Il modello può essere valutato sul set di dati di addestramento e su un set di dati di convalida iterativamente. A questo punto, i grafici delle prestazioni misurate vengono creati per mostrare l'andamento delle curve di apprendimento. La revisione di queste curve può essere utilizzata per diagnosticare problemi con l'apprendimento, come problemi di underfit o overfit, nonché se i dataset di addestramento e convalida sono adeguatamente rappresentativi. Un esempio di curva di apprendimento che viene generalmente utilizzata è la loss function. La loss function, o funzione di costo, è una funzione che valuta le prestazioni di un modello durante l'addestramento. Rappresenta una perdita e quindi l'obiettivo di un buon apprendimento è la minimizzazione della loss function: minore è la loss, migliore è il modello. Perciò un addestramento ottimale si ottiene quando queste due funzioni tendono a 0.

Invece, la Classification accuracy che era stata anticipata, è una metrica che riassume le prestazioni di un modello classificatore, calcolando il rapporto tra il numero di previsioni corrette e il numero totale di previsioni. Questa metrica implica prima di tutto l'utilizzo di un modello di classificazione in grado di eseguire previsioni per ogni dato contenuto in un sottoinsieme di dati di test, derivanti dal dataset principale. Le previsioni vengono quindi confrontate con le etichette note per quegli esempi nel dataset. L'accuratezza a questo punto viene calcolata come il rapporto tra il numero di previsioni corrette sulla porzione di esempi contenuti nell'insieme di test e il numero totale di previsioni effettuate sempre sullo stesso insieme di dati. Al

contrario, il tasso di errore può essere calcolato come il numero totale di previsioni errate fatte sul set di test diviso per tutte le previsioni fatte sul set di test.

Prima di applicare queste metriche di analisi, le reti CNN approfondite in precedenza, sono state addestrate per un totale di 50 epoche sui due dataset in possesso. A questo punto quindi nelle successive due sezioni, verranno illustrati i risultati, approfondendo ed analizzando qualche risultato in particolare. Nella sezione finale di questo capitolo verranno interpretati invece tutti i modelli, fornendo ad ognuno due immagini per dataset, valutandone il loro comportamento ed evidenziando le principali differenze che vengono osservate.

4.2 Risultati ottenuti con il primo dataset

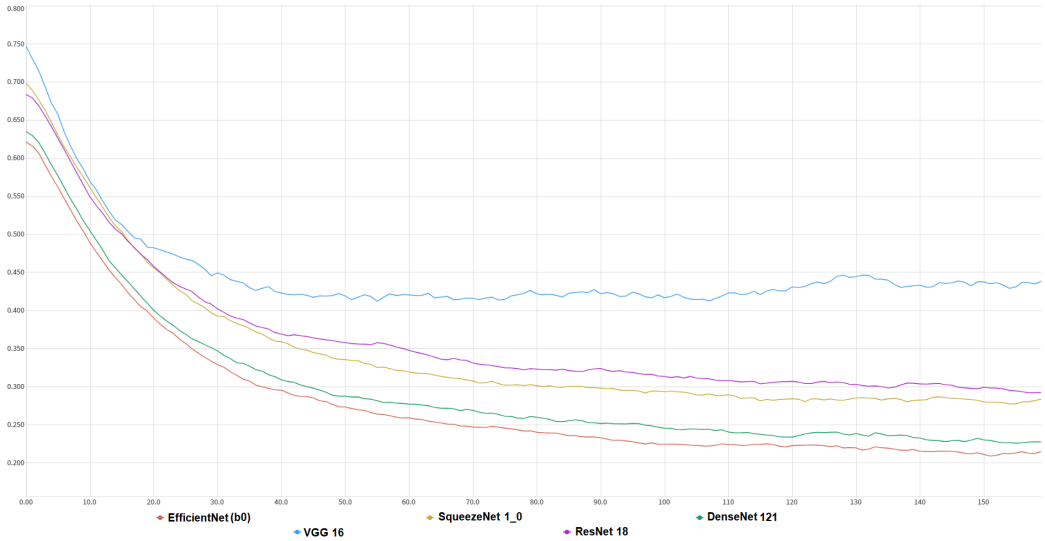

Figura 4.1: Grafico della training loss dei vari modelli addestrati sul primo dataset

Dopo aver completato la fase di training delle reti sul primo dataset, si ottengono i grafici raffiguranti le funzioni di costo (figure 4.1 e 4.2), relativamente al training loss e al testing loss. Si può notare come la training loss di ogni rete, procedendo con l'addestramento, tende ad avvicinarsi a 0, ossia il valore considerato ottimale per queste funzioni; mentre la testing loss decresce fino ad un determinato punto, dopo il quale comincia a crescere. Questo fenomeno viene definito overfitting; era stato approfondito a livello teorico nei capitoli precedenti e qui se ne vede concretamente il suo effetto. L'overfitting si riferisce a un modello che ha appreso troppo bene il dataset di addestramento, incluso il rumore statistico o le fluttuazioni casuali, se presenti. Il problema dell'overfitting è che più il modello diventa specializzato nell'addestramento dei dati, meno è in grado di generalizzare su nuovi dati, con conseguente aumento dell'errore di generalizzazione. Questo aumento dell'errore di generalizzazione può essere poi misurato dalle prestazioni del modello sul set di dati destinati al test. Ciò si verifica spesso se il modello ha una capacità maggiore di

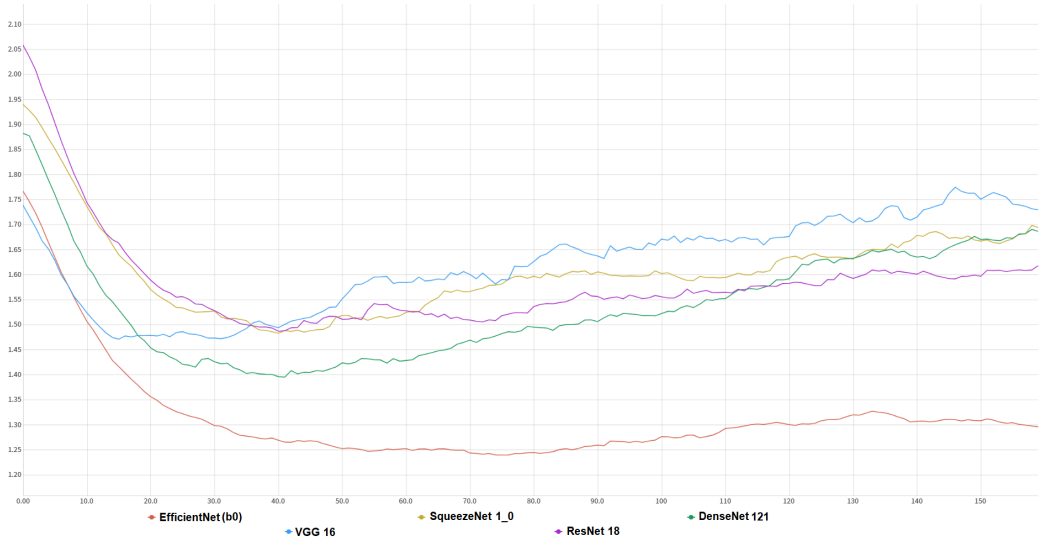

Figura 4.2: Grafico della testing loss dei vari modelli addestrati sul primo dataset

quella richiesta per il problema e, di conseguenza, una flessibilità eccessiva. Può verificarsi anche se il modello viene addestrato troppo a lungo.

Si può notare come questo fenomeno colpisce maggiormente la VGG 16 e la DenseNet 121 (figure 4.3 e 4.4), reti che comunque possono gestire grandi capacità di dati e che con il dataset in questione, non essendo molto grande, soffrono di questo problema.

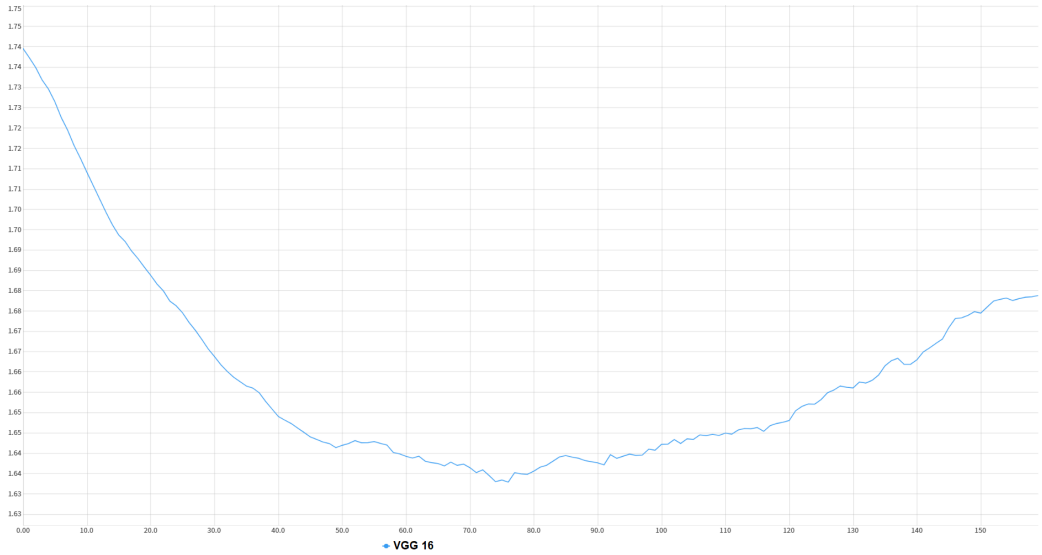

Figura 4.3: Grafico della testing loss della VGG 16 addestrata sul primo dataset

Di conseguenza, come ci si poteva aspettare, l'accuracy della classificazione (riportata in figura 4.5) delle varie reti non è delle migliori.

Nonostante le funzioni lascino pensare che il loro valore tenda lentamente ad 1, e quindi al valore ottimale di accuratezza della classificazione, questo non è un buon

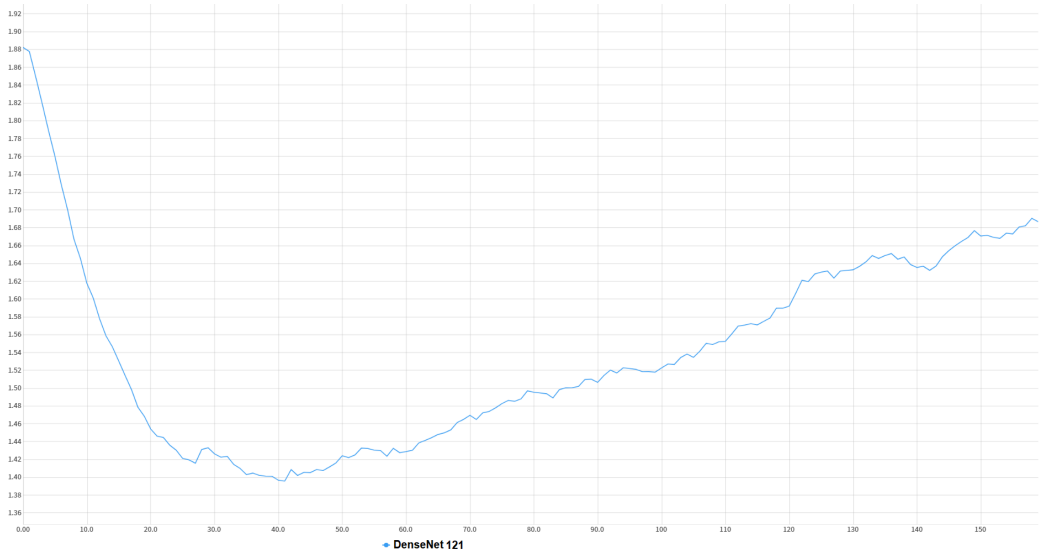

Figura 4.4: Grafico della testing loss della DenseNet 121 addestrata sul primo dataset

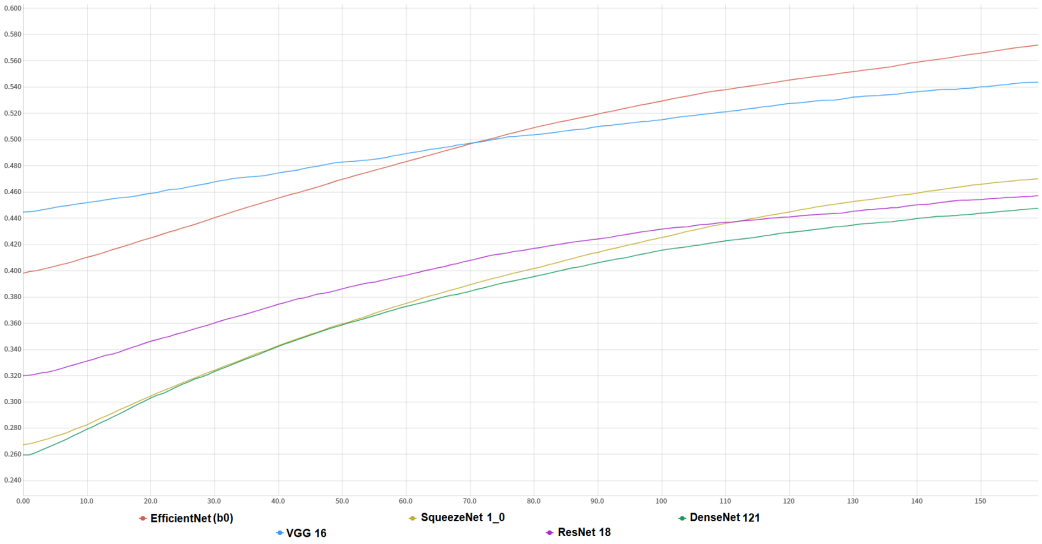

Figura 4.5: Grafico dell'accuracy di classificazione dei vari modelli addestrati sul primo dataset

risultato con un addestramento di 50 epoche. Infatti, la curva di ogni modello cresce molto lentamente e questo produce una classificazione delle immagini non buona, visto che l'accuracy delle reti dopo 50 epoche di addestramento è contenuta tra 0,44 e 0,57. Magari, aumentando notevolmente il numero di epoche, il risultato potrebbe essere differente, ma questo non può considerarsi un approccio efficiente visto il costo a livello di tempo e di complessità speso, derivanti da fasi di training maggiori.

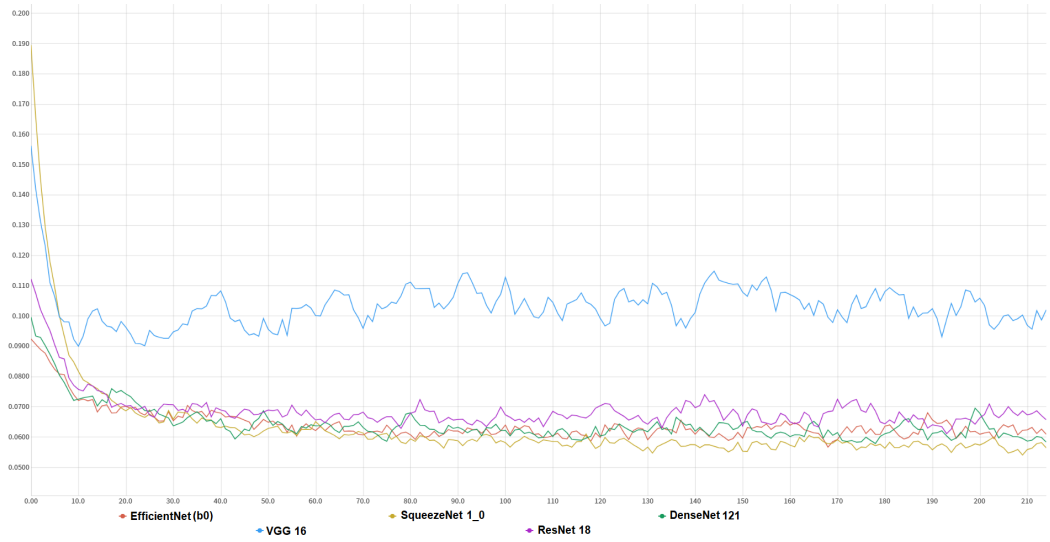

Figura 4.6: Grafico della training loss dei vari modelli addestrati sul secondo dataset

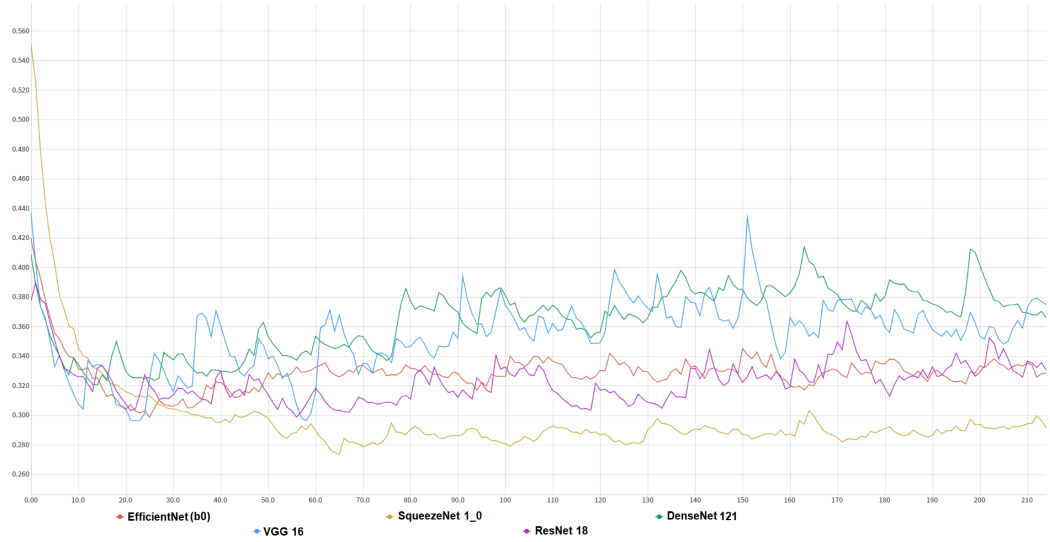

Figura 4.7: Grafico della testing loss dei vari modelli addestrati sul secondo dataset

4.3 Risultati ottenuti con il secondo dataset

In questo caso, per quanto riguarda la funzione di perdita riportata nei grafici in figura 4.6 e 4.7, si nota come le reti, con l'avanzare dell'addestramento e quindi acquisendo esperienza, tendano entrambe a 0. Perciò sotto l'aspetto della perdita, i modelli si comportano in linea generale molto bene; senza lasciar pensare che le reti soffrano di overfitting, in quanto le curve decrescono costantemente senza che vengano localizzati dei punti in cui il comportamento delle reti cambia. Prestando invece attenzione all'accuracy della classificazione dei vari modelli (figura 4.8), non tutti i modelli hanno un comportamento tendente all'ottimale (ossia tendente ad 1).

Infatti, anche se queste hanno un comportamento tendente all'ideale, e quindi

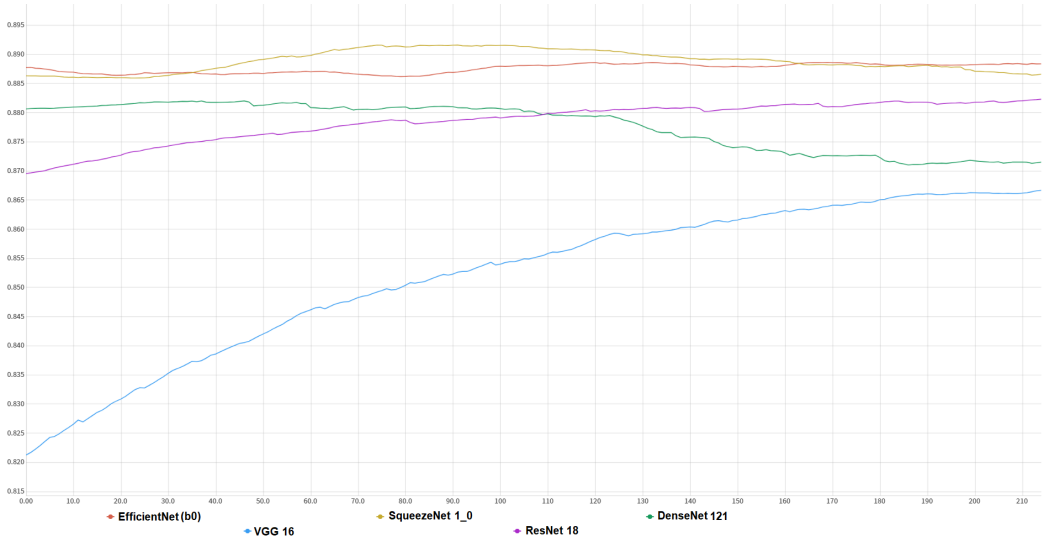

Figura 4.8: Grafico dell'accuracy di classificazione dei vari modelli addestrati sul secondo dataset

un'accuracy della classificazione che potrebbe ritenersi soddisfacente, in realtà si può osservare come nel corso delle 50 epoche alcune reti modificano leggermente il loro comportamento. Le reti che hanno questo comportamento anomalo sono: la DenseNet 121 e la Squeezenet 1_0. Perciò è necessario vedere più da vicino la curva dell'accuracy da loro prodotta, per analizzare meglio il problema.

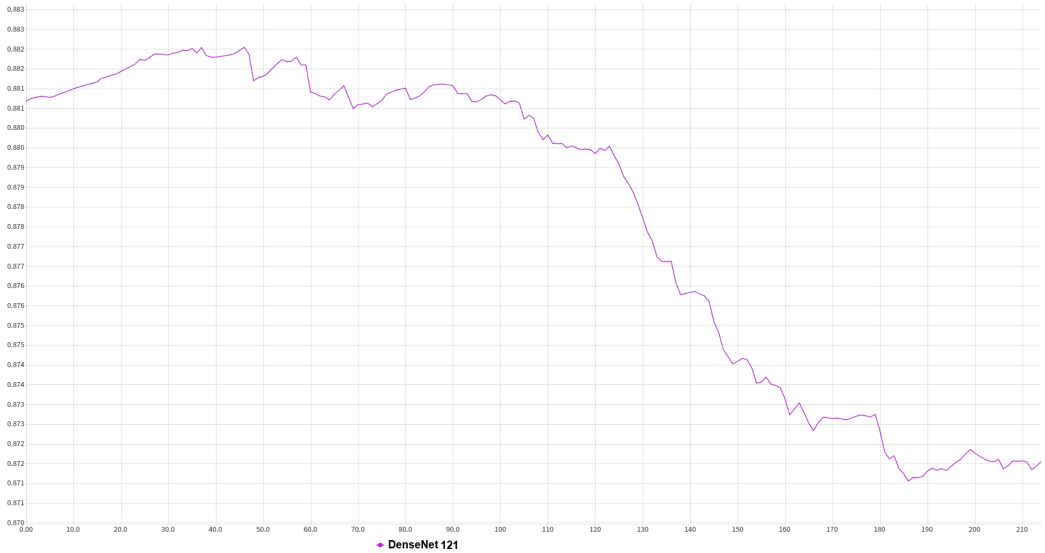

Figura 4.9: Grafico dell'accuracy di classificazione della DenseNet 121 addestrata sul secondo dataset

Come si può evincere dalle figure 4.9 e 4.10, la curva dell'accuracy di classificazione delle due reti tende ad un certo punto ad allontanarsi da 1. L'epoca in cui si interrompe l'addestramento è anche chiamata "early stopping epoch" e denota il punto

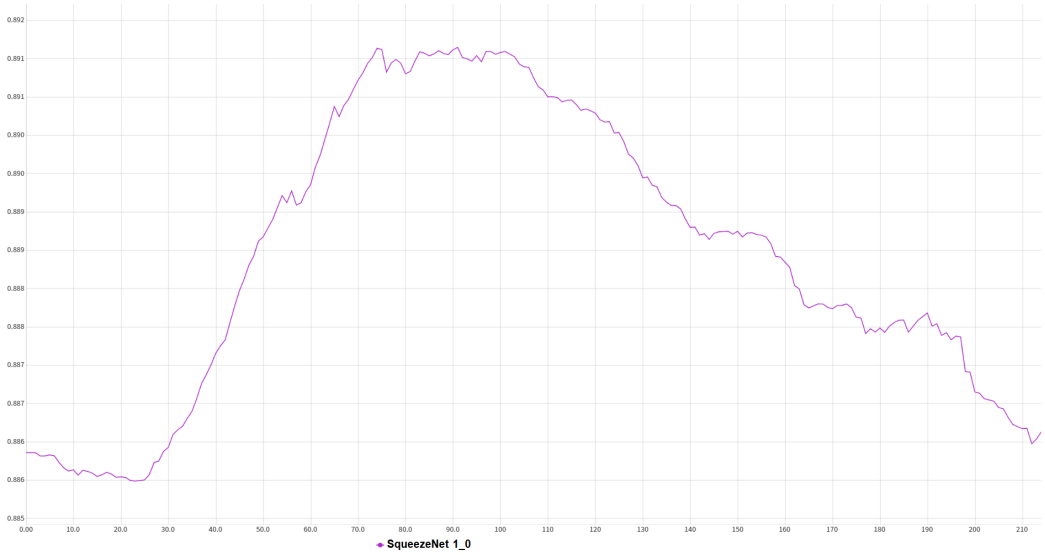

Figura 4.10: Grafico dell'accuracy di classificazione della SqueezeNet 1_0 addestrata sul secondo dataset

di addestramento migliore ottenibile con quel training set e quella configurazione del modello. Questo fenomeno, seppur manifestatosi diversamente, è sempre l'overfitting. In pratica i modelli, piuttosto che imparare le regole generali di riconoscimento dei pattern, è come se avessero imparato a memoria i pattern di esempio, di fatto riconoscendo molto bene quelli già visti ma andando in errore facilmente con i pattern non ancora visionati. Come visto in precedenza, questo sta a significare che i gradi di libertà dei modelli sono eccessivi rispetto alla complessità del training set. Questo solitamente indica che:

- il training set è troppo piccolo o poco vario, quindi il modello non riesce a trovare una regola sufficientemente generalizzata;
- il modello è troppo complesso, quindi trova più facile "aderire" ai singoli esempi che determinare una regola di riconoscimento generalizzata.

È difficile essere completamente certi di quale sia la reale causa di questo fenomeno, perché anche con il primo dataset la DenseNet ne soffriva più delle altre. Però è molto più probabile che, anche in questo caso, la "colpa" sia principalmente del dataset non troppo ampio, visto che la SqueezeNet, che in questo caso soffre anch'essa di overfitting, non è una rete troppo complessa.

4.4 Comparazione dei risultati ottenuti attraverso l'interpretabilità dei modelli

Dai risultati descritti nelle sezioni precedenti si può pensare che le reti con il primo dataset abbiano avuto qualche complicazione durante la fase di training, e

questo poi ha influito durante classificazione. Mentre con il secondo dataset la fase di training è andata piuttosto bene in ogni rete, e complessivamente anche le accuracy di classificazione sono risultate più che accettabili, tranne in un paio di casi.

A questo punto, la mente umana produce naturalmente una sorta di curiosità nel sapere come le varie reti abbiano elaborato e interpretato le varie immagini, e su quali caratteristiche di queste ultime si siano concentrati per produrre certi risultati. Di seguito verranno mostrati alcuni esempi delle varie tecniche di interpretability discusse in precedenza, applicate alle reti addestrate su entrambi i dataset. Verranno quindi fornite in input, ai vari modelli neurali, delle immagini prese da entrambi i dataset, per poi analizzare e discutere i comportamenti che verranno osservati.

Come si può ben notare dagli esempi in cui vengono fornite alle reti delle immagini prese dal primo dataset, è evidente come queste facciano più difficoltà a concentrarsi sulla borsetta presente nell'immagine nel caso in cui sia presente una persona, o comunque siano presenti altre "distrazioni" (figure 4.11, 4.12 e 4.13). Però, nonostante i risultati discussi in precedenza che mostravano come le reti, in presenza di immagini contenute in questo dataset, avessero riscontrato alcune difficoltà nel fornire una corretta classificazione delle borse, si può osservare come nel secondo esempio riferito sempre al primo dataset (figure 4.14, 4.15 e 4.16) abbiano concentrato nella maggior parte dei casi la loro attenzione sulla borsetta. Questo risultato è più scontato, in quanto è stata scelta un'immagine con uno sfondo bianco e senza molti altri oggetti da prendere in considerazione. Nonostante questo è evidente però come anche in questo caso la rete neurale DenseNet, il cui comportamento non molto ideale discusso prima, non è riuscita comunque a riportare la sua attenzione sulla borsetta.

Negli ultimi due esempi invece, sono state passate alle reti due immagini prese dal secondo dataset. Con il terzo esempio (figure 4.17, 4.18 e 4.19) è abbastanza evidente come praticamente tutte le reti si siano concentrate sulla borsetta presente, nonostante l'immagine non sia così semplice. Invece con il quarto ed ultimo esempio (figure 4.20, 4.21 e 4.22), si è voluto evidenziare che, nonostante i modelli avessero un comportamento migliore con il secondo dataset, esistono comunque dei casi negativi in cui la maggior parte delle reti non ripone per niente l'attenzione sulla borsa presente, nonostante quest'ultima sia esposta in maniera evidente nell'immagine. Le uniche reti che fanno meno confusione in questo caso sono: la VGG e la EfficientNet, le quali almeno ne percepiscono i bordi.

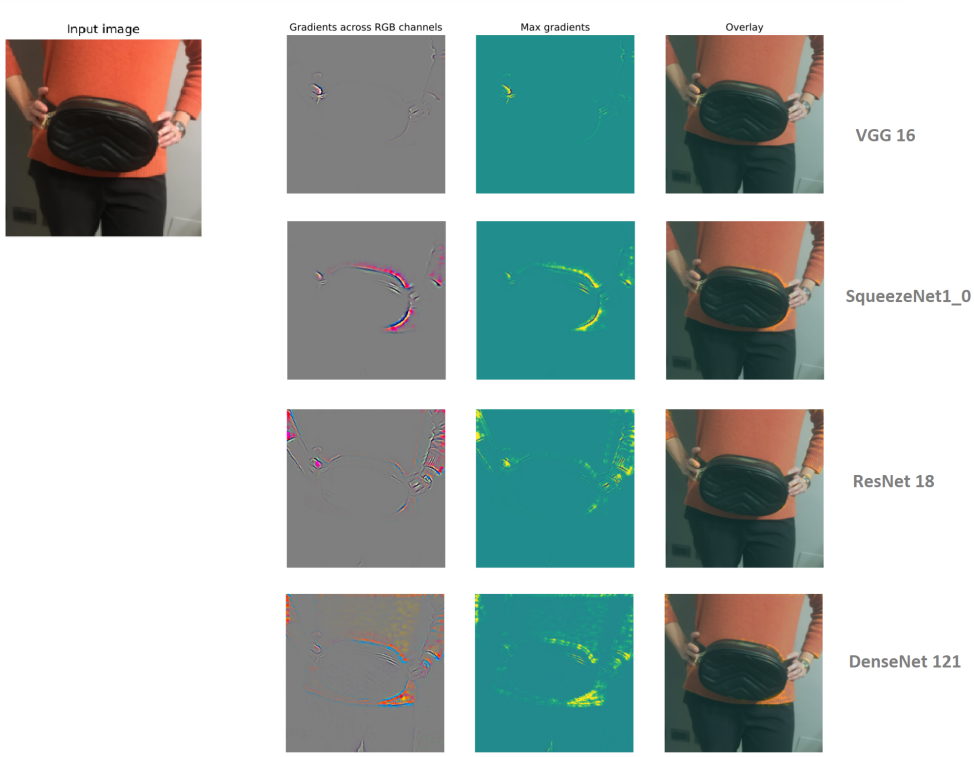

Figura 4.11: Saliency map ottenute con la prima immagine presa dal primo dataset

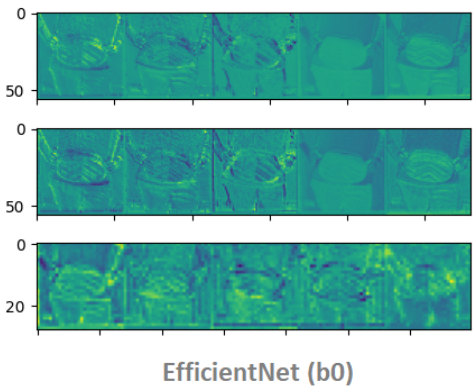

Figura 4.12: Activation map ottenute con la prima immagine presa dal primo dataset

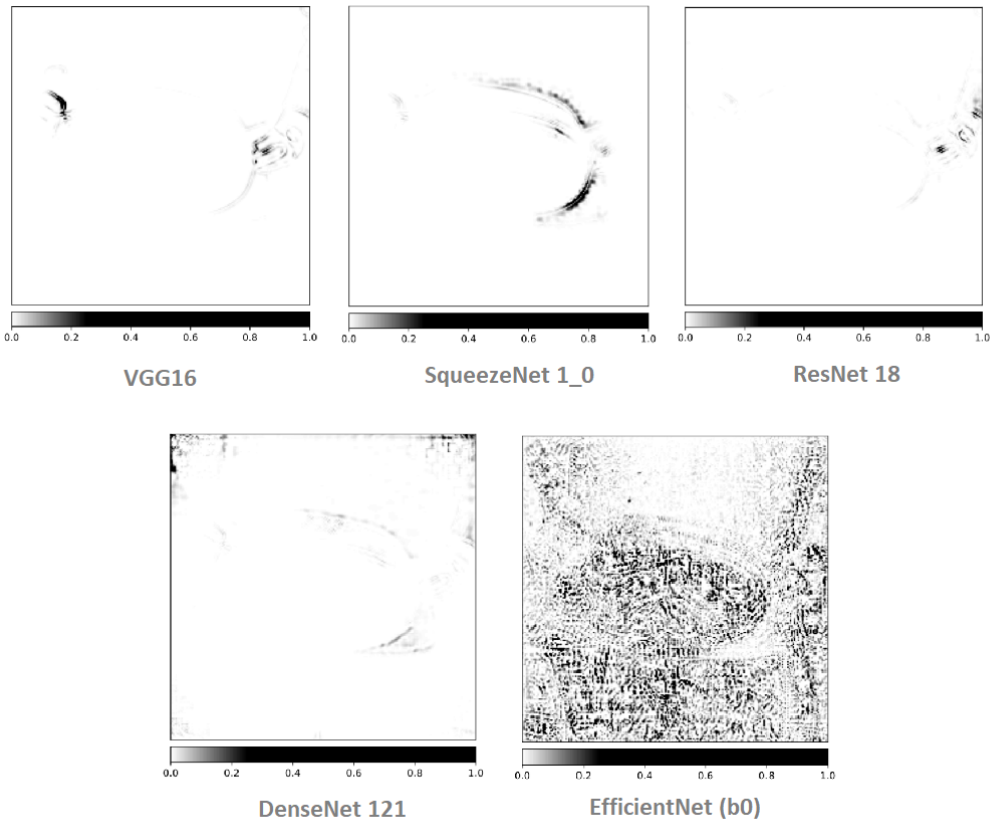

Figura 4.13: Integrated gradient applicato alla prima immagine presa dal primo dataset

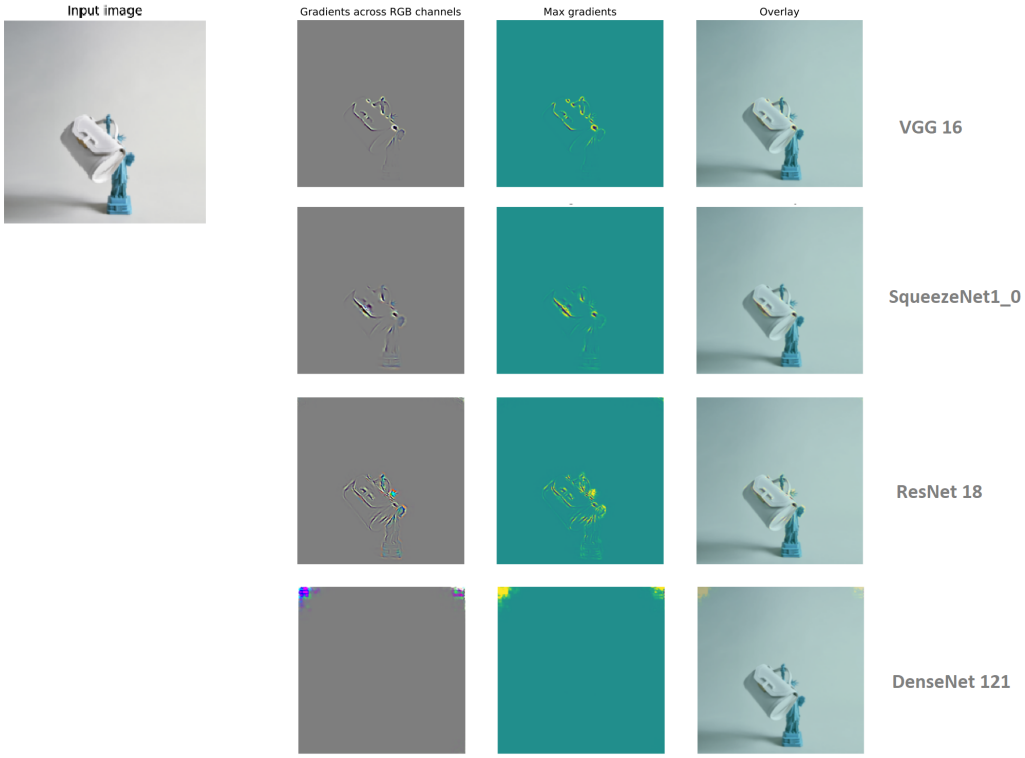

Figura 4.14: Saliency map ottenute con la seconda immagine presa dal primo dataset

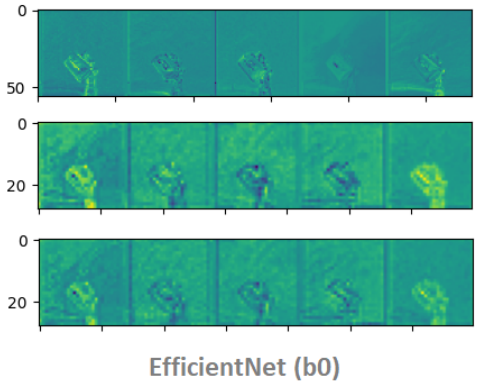

Figura 4.15: Activation map ottenute con la seconda immagine presa dal primo dataset

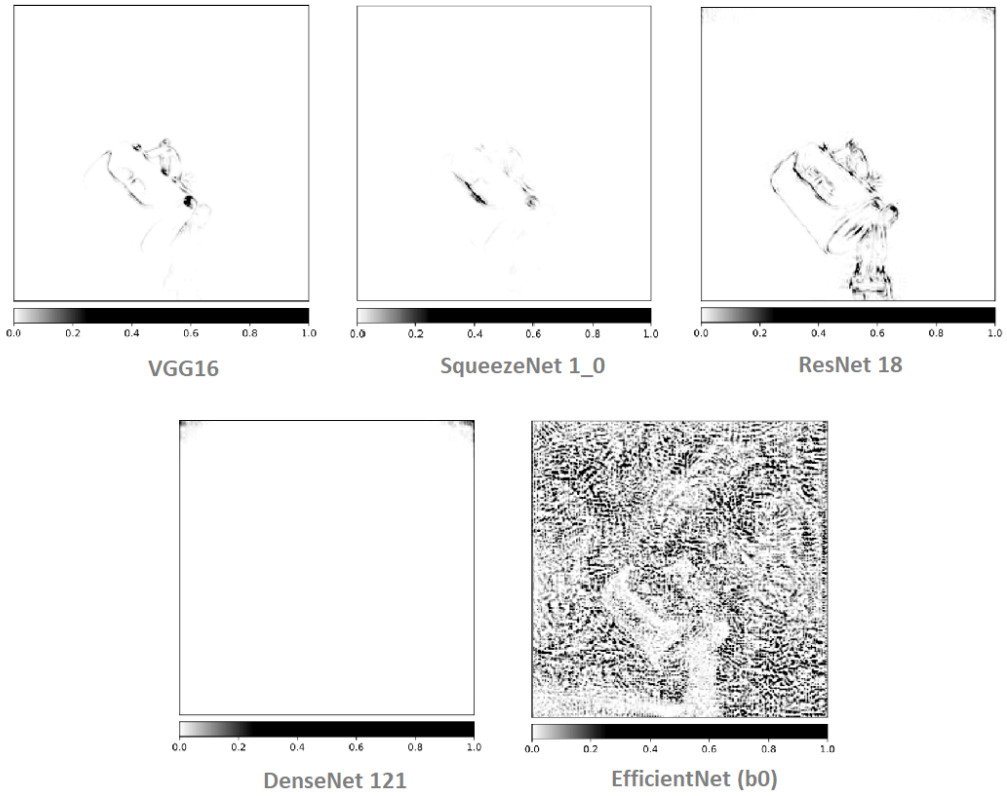

Figura 4.16: Integrated gradient applicato alla seconda immagine presa dal primo dataset

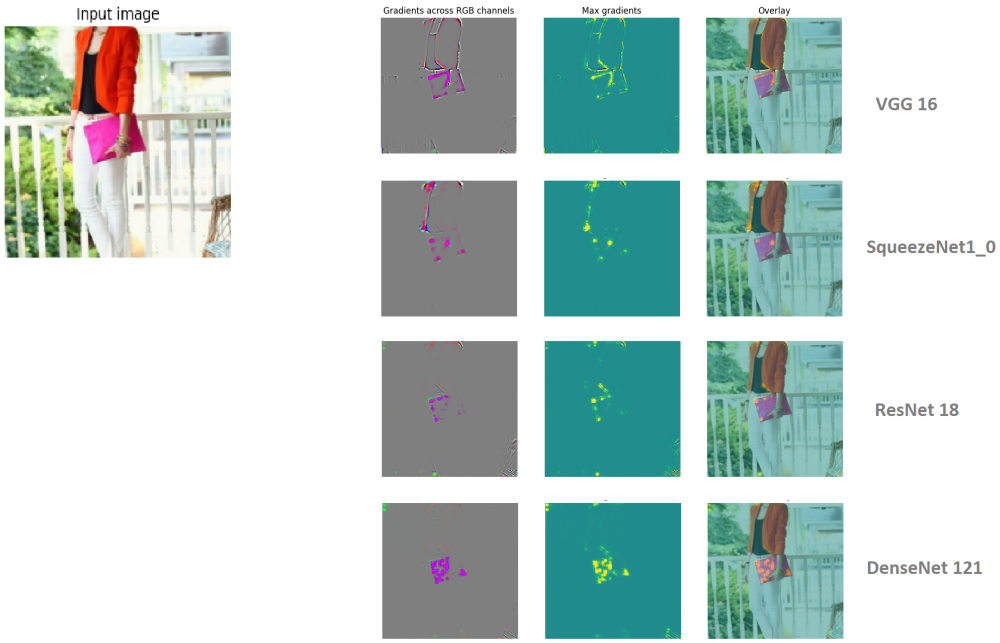

Figura 4.17: Saliency map ottenute con la prima immagine presa dal secondo dataset

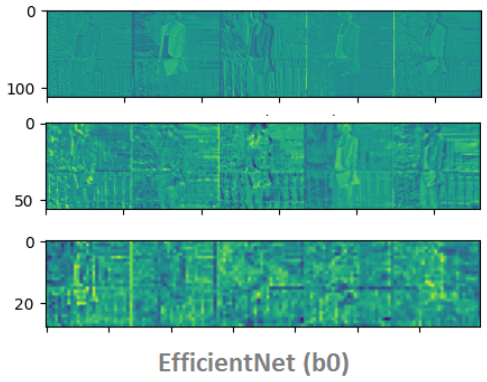

Figura 4.18: Activation map ottenute con la prima immagine presa dal secondo dataset

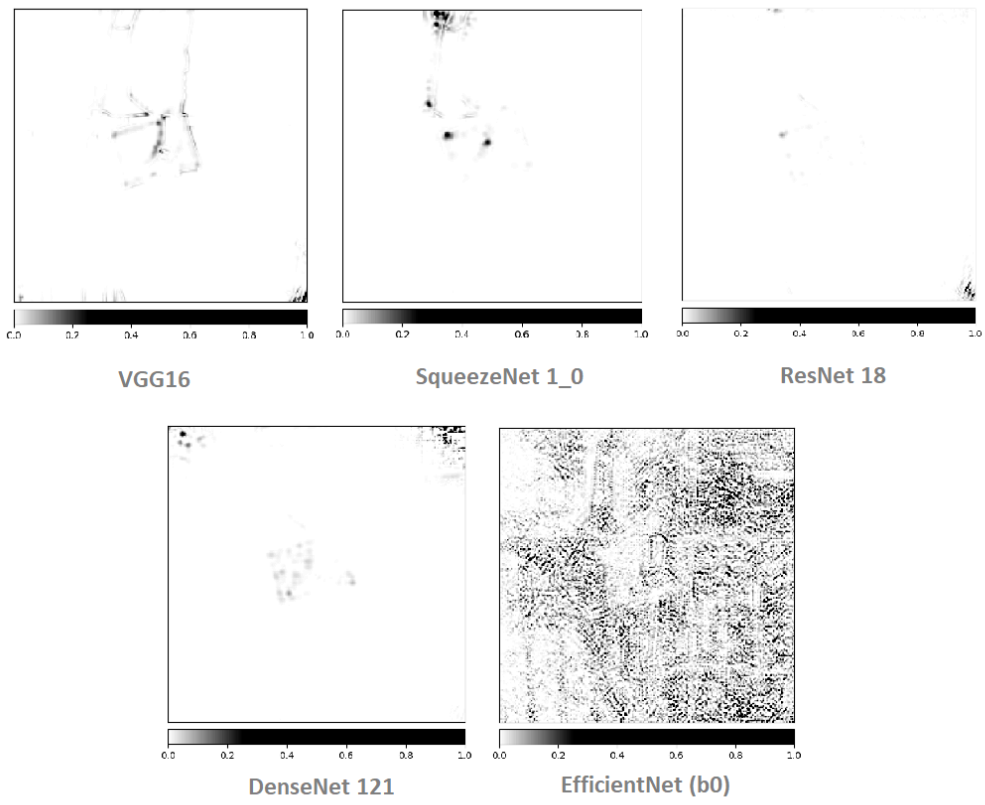

Figura 4.19: Integrated gradient applicato alla prima immagine presa dal secondo dataset

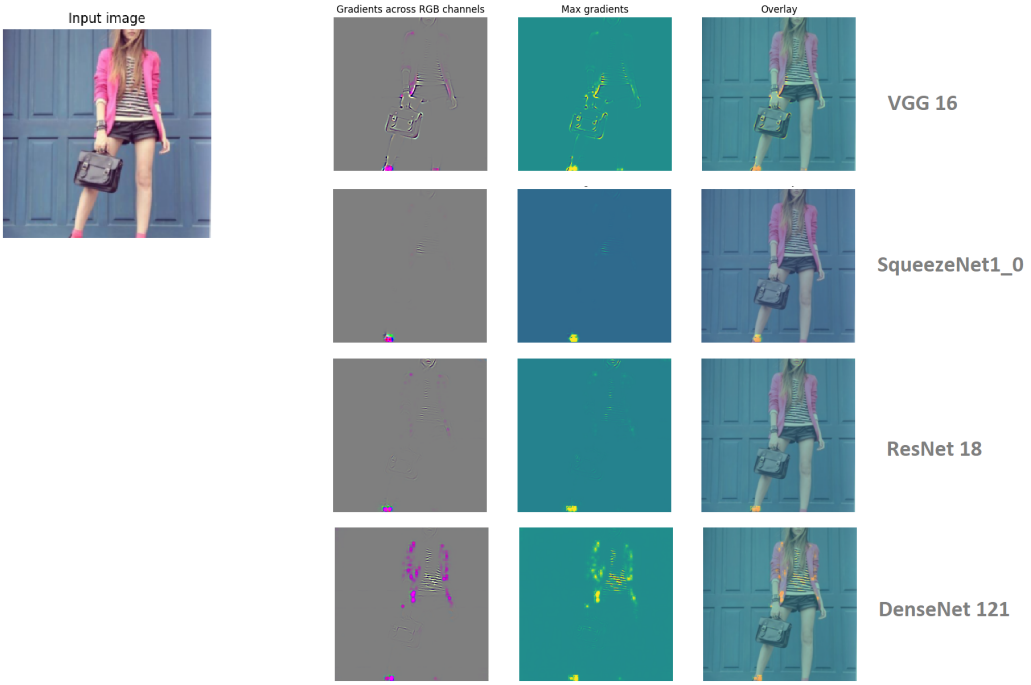

Figura 4.20: Saliency map ottenute con la seconda immagine presa dal secondo dataset

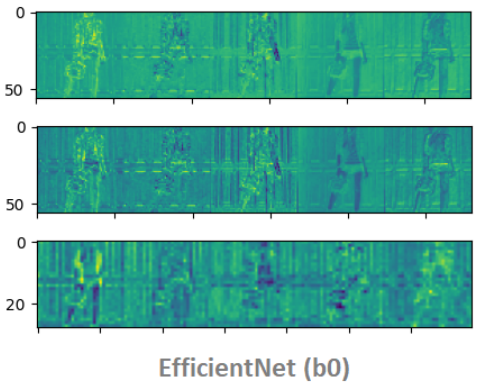

Figura 4.21: Activation map ottenute con la seconda immagine presa dal secondo dataset

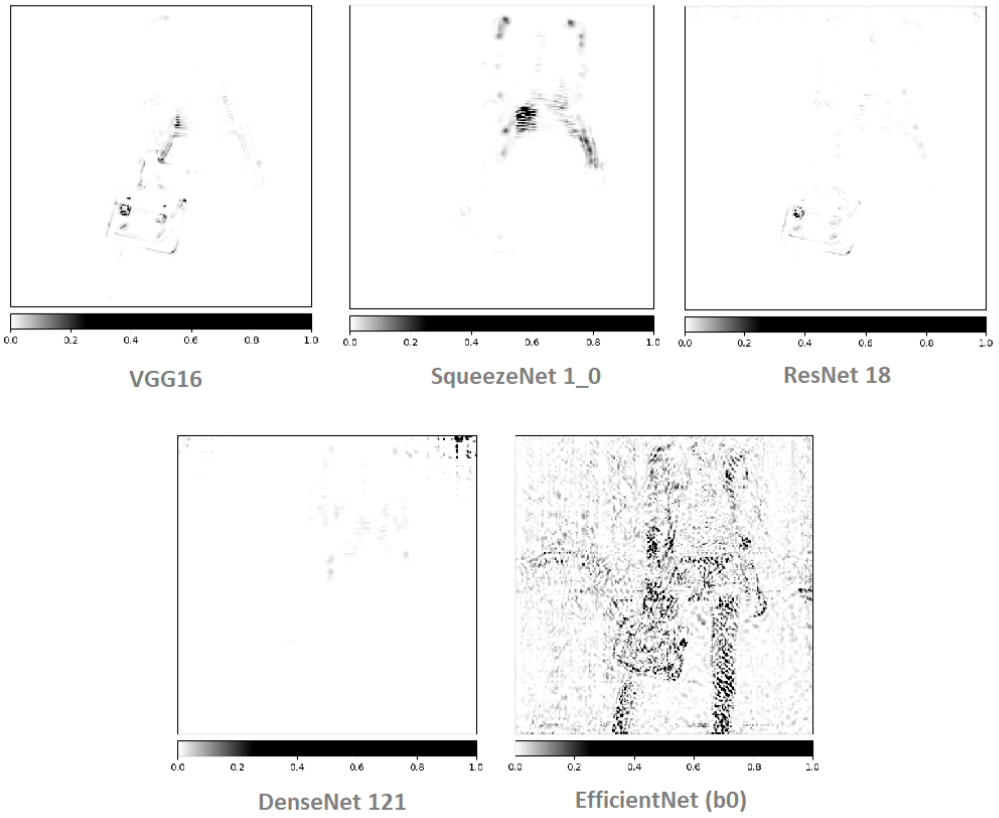

Figura 4.22: Integrated gradient applicato alla seconda immagine presa dal secondo dataset

Capitolo 5

Conclusione

Il percorso di questa tesi sperimentale, articolato in 4 capitoli più il seguente, ha messo in risalto una delle modalità di utilizzo delle reti neurali quando hanno a che fare con le immagini, mostrando alcune possibili applicazioni che si possono compiere attraverso l'intelligenza artificiale, nel campo della Computer Vision. Applicazioni che al giorno d'oggi stanno diventando sempre più di utilizzo comune in ogni ambito. Così, a partire dai primi capitoli, si è cercato dapprima di fornire delle conoscenze basilari che dessero un buon background su quello che si sarebbe visto in termini applicativi, e poi si è tentato di fornire delle nozioni in merito allo stato dell'arte: deep learning e metodi di interpretabilità destinati alle reti neurali. Uno dei task principali su cui si è concentrato lo studio applicativo è stato la classificazione degli oggetti nelle immagini, e nello specifico sulla classificazione di borsette nell'ambito moda. Questa tecnica oggi, generalizzata su altri tipi di immagine, è alla base di sistemi di ricerca su siti di compra-vendita online, per i solcial network e per tante altre applicazioni che molto spesso vengono utilizzate quotidianamente, ma di cui magari non si conosce lo sviluppo e il funzionamento che c'è dietro. Perciò è stato interessante approfondire e comprendere i vari meccanismi sottesi a tutto questo. Un altro task fondamentale di questo lavoro è stato implementare tecniche di Feature Visualization, per poter interpretare il più possibile i risultati ottenuti dalle reti neurali; dimostrando che i modelli neurali non sono riducibili a delle black-box completamente imprevedibili.

Lo scopo parallelo a questi task è stato testare queste tecniche anche per rendersi conto di quanto ci si stia avvicinando sempre più al comportamento dell'essere umano e nello specifico del cervello umano. Questo si è potuto notare nella fase di analisi dei risultati, quando reti più o meno moderne e più o meno diverse tra loro, proponessero comportamenti più o meno accurati in termini di risultati. Ovviamente da questi risultati ottenuti si è anche visto che c'è una forte dipendenza dalla qualità delle immagini contenute nei dataset, di cui si disponeva già all'inizio nel progetto, e da quanto quest'ultimi fossero estesi. Infatti, l'addestramento delle reti su questi dataset poteva incidere fino ad un certo punto nei risultati finali, oltre il quale non si sarebbero più ottenuti risultati migliori; anzi, in certi casi particolari l'aumento dell'addestramento portava a fenomeni indesiderati.

Di per sé comunque, lo studio effettuato nel corso di questa tesi si è rilevato

estremamente utile come primo approccio nel mondo del Deep learning e dell'Intelligenza artificiale in generale, aprendo cosi nuovi orizzonti a tecnologie e ad approcci ingegneristici mai visti prima, ma che oggi giorno sono in continua evoluzione e si trovano alla base della maggior parte delle applicazioni.

5.1 Sviluppi futuri

La procedura realizzata, con una lieve implementazione, può essere considerata una buona base da integrare in altri scenari applicativi che necessitano operazioni di riconoscimento. Uno dei punti di partenza per ottenere risultati più accurati potrebbe essere legato al miglioramento del dataset, o dei dataset, su cui le varie reti neurali devono venire addestrate. Utilizzando quindi delle collezioni di dati di maggiori dimensioni sia come classi che come numero di immagini. Dall'altro lato invece, si potrebbe agire prendendo in considerazione l'idea di impiegare CNN non pre-addestrate e quindi costruirle sulla base delle risorse disponibili.

Oppure, una valida alternativa sarebbe quella di sostituire completamente le CNN con delle reti neurali più all'avanguardia. Oggigiorno, infatti, stanno acquisendo molta popolarità le reti neurali Transformers [11], le quali sono già diventate fondamentali in numerosi campi, tra cui quello della Computer Vision. I transformers sono reti che operano su sequenze di dati, ad esempio un insieme di parole o immagini. Questi insiemi di dati vengono prima tokenizzati e poi dati in input alle reti. A questo punto, i transformers aggiungono "attenzione", ossia il meccanismo principale sulla quale si basano queste reti. L'attenzione non è altro che un'operazione quadratica, attraverso la quale viene calcolato il prodotto interno tra ciascuna coppia di dati tokenizzati. All'aumentare del numero di dati, aumenta anche il numero di operazioni. Le immagini però, rispetto alle parole, sono più difficili da addestrare sui transformers in quanto composte da pixel, e ogni immagine può contenere da migliaia a milioni di pixel. Quindi in una rete di questo tipo, ogni pixel eseguirà un'operazione con ogni altro pixel nell'immagine. Nonostante questo, i transformers poi si comportano molto bene all'atto pratico. Si è visto come le CNN, lavorando con delle immagini, si concentrassero solo sulle feature più importanti, evitando i dettagli di ciascun pixel di un'immagine. Perciò, se venissero inseriti in un modello interi dati rappresentanti ogni immagine, anziché solo le parti che i filtri possono estrarre, o che considerano importanti, le possibilità che il modello funzioni meglio sono maggiori. Questo è esattamente ciò che accade all'interno di queste nuove reti.

Questo non significa però che i transformers sostituiranno definitivamente le CNN, o almeno non subito. Infatti, vengono ancora prodotti modelli come la EfficientNet V2 [12], che a livello di performance è ancora meglio rispetto alle reti la cui architettura si basa sull'attenzione. Ma non è comunque da escludere che, se applicate al progetto trattato in questa tesi, possano portare a risultati brillanti.

Bibliografia

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[4] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[5] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[6] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[7] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[8] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[9] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328. PMLR, 2017.

[10] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[12] Mingxing Tan and Quoc V Le. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*, 2021.