ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO di INFORMATICA - SCIENZA E INGEGNERIA (DISI)
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA
in
Sistemi Digitali M

Identificazione di caratteristiche rilevanti in capi di abbigliamento mediante deep-learning

CANDIDATO

Dott. Fabrizio Torriano

RELATORE

Prof. Stefano Mattoccia

CORRELATORI

Dott. Milo Marchetti
Dott. Filippo Aleotti

Anno Accademico 2021/22
Sessione II

Abstract

Il mondo della moda è in continua e costante evoluzione, non solo dal punto di vista sociale, ma anche da quello tecnologico. Nel corso del presente elaborato si è studiata la possibilità di riconoscere e segmentare abiti presenti in una immagine utilizzando reti neurali profonde e approcci moderni. Sono state, quindi, analizzate reti quali FasterRCNN, MaskRCNN, YOLOv5, FashionPedia e Match-RCNN. In seguito si è approfondito l'addestramento delle reti neurali profonde in scenari di alta parallelizzazione e su macchine dotate di molteplici GPU al fine di ridurre i tempi di addestramento. Inoltre si è sperimentata la possibilità di creare una rete per prevedere se un determinato abito possa avere successo in futuro analizzando semplicemente dati passati e una immagine del vestito in questione. Necessaria per tali compiti è stata, inoltre, una approfondita analisi dei dataset esistenti nel mondo della moda e dei metodi per utilizzarli per l'addestramento.

Il presente elaborato è stato svolto nell'ambito del progetto FA.RE.TRA. per il quale l'Università di Bologna svolge un compito di consulenza per lo studio di fattibilità su reti neurali in grado di svolgere i compiti menzionati.

Indice

Capitolo 1

Introduzione

I vestiti hanno, da sempre, contribuito a definire la cultura e i comportamenti dell'umanità. Il loro utilizzo ha contraddistinto e rafforzato strutture sociali e relazionali nel corso dei secoli. Creati dapprima per un semplice motivo utilitaristico, ovvero l'abito come protezione dagli elementi, i vestiti hanno gradualmente ricoperto anche la funzione di segnalare uno status sociale. Anche in altri campi quali rituali magici, esoterismo e religione gli abiti hanno assunto e svolto nel tempo un forte significato sociale e apotropaico. Si può affermare che per un certo verso i vestiti hanno avuto una parte talmente integrante nella cultura da influenzare i comportamenti umani stessi. Forma e foggia dei vestiti sono mutate notevolmente nel tempo e, a seconda del periodo e della cultura, anche le parti del corpo coperte sono variate enormemente. Si pensi, ad esempio, agli stili più coperti occidentali rispetto a quelli meno coperti di società africane. Queste differenze, oltre che dettate da esigenze legate al clima o a necessità di proteggere il corpo da pericoli potenziali provenienti da animali o da altri elementi naturalistici, non sono però dovute solo a motivazioni pratiche. In molti casi si è potuto verificare come in alcune culture la copertura di determinate parti del corpo non fosse vista come necessaria ma, addirittura, insensata. D'altro canto si è potuto verificare che in certe culture la copertura del viso, specie per le ragazze, avesse una priorità maggiore, in termine di pudore, rispetto ad altre parti del corpo[b]. Al di là di considerazioni etico morali su questi aspetti, non pertinenti alla presente dissertazione ma allo stesso tempo non trascurabili in quanto, verosimilmente, tali considerazioni sono e devono essere ben valutate dal mondo della moda, si deve, infatti, tenere conto di quanto il vestiario possa influenzare, nel suo piccolo, il nostro mondo circostante. Effetti ben visibili della trasformazione della società e dei suoi usi e costumi in termini di mutamenti sociali sono riscontrabili anche attraverso una analisi del vestiario stesso: si pensi, ad esempio, al cambiamento di costumi avvenuto durante gli anni sessanta quando il modo di vestire cambiò radicalmente rispetto agli standard più ingessati ed eleganti del passato per prediligere vestiti più casual, jeans, spingendosi ad adottare anche pettinature più naturali e lunghe, provocando una rottura decisa col continuum dei costumi della generazione precedente. Un

altro esempio calzante di trasformazione delle abitudini legate non solo alla moda e alla comodità ma proprio alla necessità di ricorrere a un capo di abbigliamento si ritrova nel cappello. Il cappello di una determinata foggia è stato per lungo tempo considerato un indispensabile accessorio di moda per uomini di classe agiata attorno al 1920. Il cappello ha poi perso importanza verso gli anni '70, sempre vittima del cambiamento generazionale in seguito agli eventi del sessantotto. La sua importanza è comunque da evidenziare, in quanto un semplice cappello poteva fare comprendere differenze di classe sociale e dare messaggi simbolici sul suo utilizzatore. A riguardo espedienti simili sono stati usati nel cinema come messaggi subliminali riguardo allo status di un personaggio: in un film della celebre saga di Don Camillo, Peppone indossa un cappello Lobbia dopo essere stato nominato senatore della Repubblica mentre in precedenza, quando è in carica come sindaco, indossa un più normale cappello Fedora. Ovviamente questi dettagli possono sembrare secondari e di scarsa importanza al giorno d'oggi ma dimostrano come in passato fossero invece particolari importanti e degni di nota. Un ulteriore aspetto è quello degli abiti "utilitaristici" ovvero quegli abiti che assolvono una funzione non solo di status ma anche pratica. In questo campo ci sono abiti come i pantaloni da lavoro, i completi militari, le armature e altri paramenti atti alla protezione dell'indossatore.

È in questo ambito che si viene a parlare di moda come del cambiamento dei costumi e, più in generale, del modo considerato "normale" o di tendenza di vestirsi. In questo contesto la moda rappresenta semplicemente come la maggior parte delle persone, o degli appartenenti a una classe sociale, si veste e quali particolari sono da ricercare nei loro abiti. Inoltre la moda come ambito significa anche cercare di forzare questi cambiamenti in una direzione, in questi casi spesso per soddisfare i propri interessi da parte di una azienda. Di per sé la moda, dal latino "modus", significante "norma", "modalità", "maniera", altro non è che lo studio di queste tendenze. Nel corso della storia si è cercato di creare gradualmente stili di vestiario differenti, spesso per distinguersi da generazioni precedenti e rafforzare il proprio status sociale. Rottura che poi divenne evidente in seguito alla rivoluzione francese durante la quale molte regole relative al vestirsi vennero abolite in quanto ritenute troppo divisive e minanti i principi di egualità, cardine della rivoluzione stessa[a].

Un altro aspetto interessante è il fatto che spesso vengono indossati vestiti che offrono un messaggio sulla persona stessa. A questo proposito ci si riferisce come “mascheramento” ovvero alla capacità di mostrare aspetti di sé, o di nasconderli, semplicemente in base al proprio vestiario. Un esempio può essere un completo grigio per indicare modernità o l’abbigliamento semplice di Steve Jobs per indicare accessibilità e modernità del prodotto che stava presentando. Al contrario una mancanza di aderenza a tali standard può essere fonte di imbarazzo, come, ad esempio, capitò a John Dalton, noto chimico, che si presentò a una riunione di quaccheri con delle scarpe color rosso acceso ritenute da lui essere di un più sobrio marrone. Tale evento darà poi seguito alla scoperta, o comunque a una prima documentazione, del daltonismo il cui nome deriva proprio da lui.

È dunque evidente quanto interessanti e pervasivi siano la moda e il suo studio, studio che può essere arricchito con strumenti informatici e tecnologici per una sua più attenta catalogazione e previsione nel futuro. Previsione non già finalizzata solo a scopo di lucro, ma anche come inserita in un contesto di studio della psicologia umana e del costume sociale.

In questa trattazione si approcceranno tali elementi in modo improntato all’analisi della stessa a partire da semplici immagini.

Capitolo 2
Strumenti utilizzati

Per affrontare questo studio saranno usati approcci tipici della computer vision, ovvero, dopo avere creato una rappresentazione dell'immagine all'interno del computer, si elaborerà e analizzerà in maniera automatica tale rappresentazione al fine di estrarre dei dati. In questo ambito, viste le recenti scoperte tecnologiche, si è deciso di utilizzare approcci basati sul machine learning invece che puramente algoritmici data la loro maggiore efficacia e capacità di generalizzare.

2.1 Il machine learning

L'apprendimento automatico è una branca dell'intelligenza artificiale che si occupa, tramite diversi metodi, di creare algoritmi per individuare pattern in insiemi di dati. Utilizzando questa metodologia, i vari pattern non vengono definiti dal programmatore mediante una descrizione, ma mostrando a una macchina dei dati che verificano o meno particolari condizioni e, tramite metodi statistici e logici, si lascia che la macchina costruisca una rappresentazione interna del mondo reale. In seguito è possibile sfruttare questa rappresentazione in modo inverso su dati che la rete non ha mai visionato prima al fine di ottenere nuove informazioni su questi nuovi dati.

Questi metodi si basano sull'assunto che la rete sia in grado di "generalizzare in base alla propria esperienza", ovvero si vuole emulare il comportamento cardine comune alle forme di vita secondo il quale ogni avvenimento e ogni situazione incontrata vanno ad accrescere il bagaglio culturale del soggetto in questione. Componente principe di tale metodologia è senz'altro il ragionamento induttivo, cioè il processo attraverso il quale si cerca di stabilire una legge universale, vale a dire una modellizzazione del mondo reale, da casi particolari singoli, ovvero dai dati mostrati. Gli approcci basati sul machine learning sono molto collegati alla statistica che risulta essere una base molto forte sia per la creazione di collezioni di dati sui cui addestrare che per la creazione delle macchine stesse.

Nel contempo, come in qualunque campo, sono nate svariate implementazioni di questi concetti: alberi di decisione con funzione obiettivo, reti neurali, algoritmi genetici, reti bayesiane, macchine a vettori di supporto e molte altre. Ognuno di questi metodi ha ovviamente un campo di applicazione principale; in questa trattazione si farà principalmente riferimento a quelle metodologie che possono essere applicate con successo alla computer vision secondo lo stato attuale dell'arte.

2.1.1 Estrazione di feature e oggetti da una immagine

Un task molto importante nel mondo della computer vision è quello di localizzare oggetti all'interno di una immagine. Esistono molteplici metodologie per raggiungere tale obiettivo che possono variare da metodi "naïve", che in questi contesti indicano una soluzione poco robusta e spesso non adatta al suo utilizzo nella maggior parte dei campi, a soluzioni complesse che richiedono, in quanto tali, ore di lavoro e considerazioni molto approfondite sul dominio in cui si sta lavorando. Le soluzioni più naïve sono solitamente incentrate sul tentativo di localizzare un oggetto all'interno di una immagine mediante comparazioni pixel per pixel di un template che funge da riferimento. Si può considerare come metodo più semplice, infatti, quello di indicare esattamente quale sotto area di una immagine si vuole trovare e in seguito fare un confronto di ogni singolo pixel dell'immagine con la sotto area in questione. Metodi più avanzati usano distanze ad hoc e considerazioni sul fatto che il template potrebbe non essere esattamente identico, venendo in questo modo a creare algoritmi molto più robusti e potenti. Esempi di questi metodi sono SAD e SSD, ma anche NCC e ZNCC.

Per contestualizzare quanto sopra si discuterà brevemente dell'algoritmo SAD, la cui idea di fondo è quella di avere una immagine che funga da template contenente l'oggetto da cercare e lo scopo è il voler trovare la regione nell'immagine dove la distanza da tale template risulta minima.

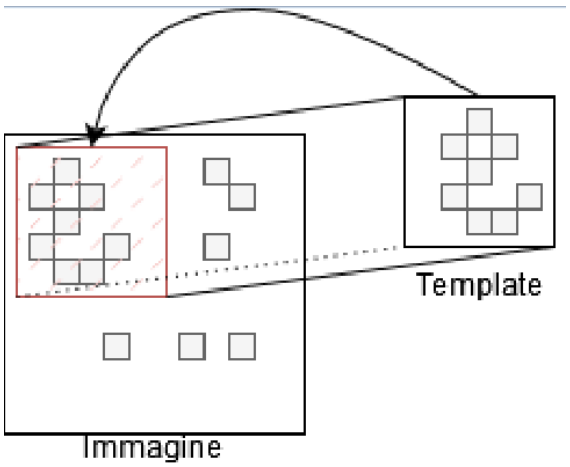

Figura 1: Esempio di ricerca di un template in una immagine.

La metrica relativa alla distanza può variare a seconda dell'algoritmo preso in considerazione, nel caso di SAD si usa la "Sum of Absolute Differences", ovvero la somma del valore assoluto di differenze tra il template e una area dell'immagine avente stessa dimensione. Si noti che in questo caso si sta parlando di un algoritmo non invariante alla scala, ovvero il template deve avere l'esatta dimensione dell'oggetto nell'immagine che si sta analizzando.

$$SAD(i,\ j)\ =\ \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}|I(i+m,\ j+n)\ -\ T(m,n)|$$

$$SSD(i,\ j)\ =\ \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n)\ -\ T(m,n))^2$$

Un metodo diverso, ma sempre basato sull'analisi di un template è SIFT. SIFT (acronimo di Scale-Invariant Feature Transform) è un algoritmo che, dato un template, crea dei descrittori locali per indicare le principali caratteristiche di quel template e poi usa i descrittori per individuare un omomorfismo all'interno dell'immagine target assumendo che in quel punto ci sarà l'oggetto cercato. Il descrittore risulta essere, di fatto, un insieme di gradienti preso nei punti più significativi, spesso gli angoli o i bordi di un oggetto.

Supponendo che ogni oggetto presente in una immagine abbia dei punti salienti che possono essere estratti per fornire una descrizione di caratteristiche di tale oggetto, questa descrizione viene poi usata per il riconoscimento dell'oggetto in una altra immagine. Grande vantaggio di questo metodo è l'invarianza non solo alla

scala ma anche alla rotazione, all'illuminazione e per una certa dose anche al rumore.

I metodi sopra elencati hanno però un grande svantaggio: le loro assunzioni. Per trovare un oggetto in una immagine infatti si suppone di avere un template di tale oggetto, laddove per template si intende una immagine dell'oggetto in condizioni di illuminazione perlopiù analoghe e raffigurato dalla stessa angolazione. Inoltre tali metodi non sono in grado di astrarre un concetto da una immagine, infatti, dato un descrittore troveranno solo qualcosa che corrisponde a tale descrittore. Questi metodi dunque non sono per nulla robusti o scalabili a casi nei quali non si conosce esattamente l'oggetto che si cerca o si vuole trovare tutti gli oggetti di una particolare categoria.

Per risolvere tale problema si possono usare sistemi basati sull'apprendimento automatico. In tale modo si può insegnare a una macchina il concetto cercato e quindi mostrare una qualunque immagine alla macchina e attendere come uscita l'individuazione dell'oggetto cercato. Questo sistema, per quanto efficace, non è immune da problemi: tanto per cominciare occorre un insieme di immagini particolarmente grande e il più possibile vario e completo per insegnare alla macchina cosa si intende per ogni oggetto e poi occorre che tali immagini rispecchino il più possibile gli oggetti cercati. Segue che una attenta ricerca e collezione di queste immagini sia uno dei fattori fondamentali di qualunque applicazione di questo metodo. Si fa riferimento a questo insieme di immagini come dataset.

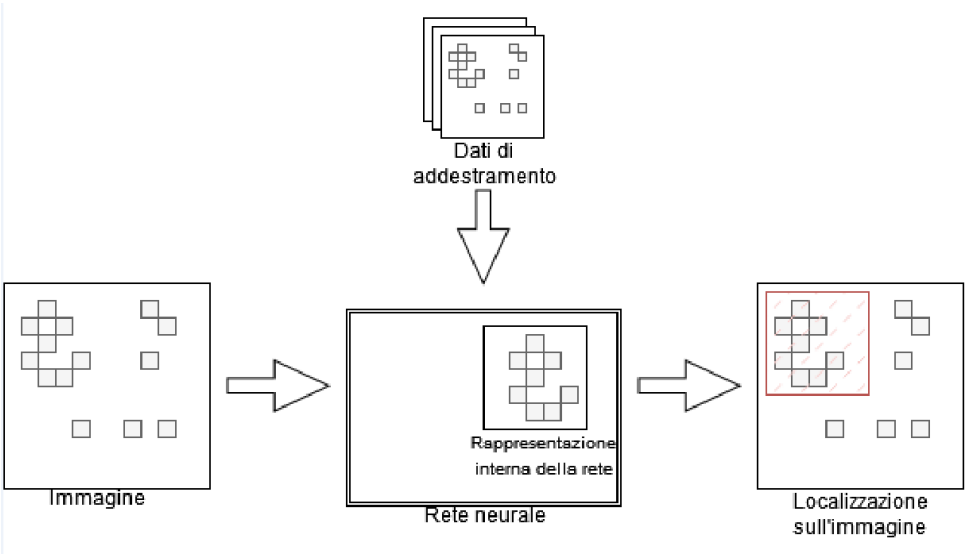

Figura 2: Concetto di funzionamento di una rete neurale.

2.1.2 I dataset

Un dataset è, di fatto, una collezione di dati. Nell'ambito della computer vision i dati in considerazione consistono per la maggior parte di immagini e metadati allegati alle stesse. I metadati sono principalmente informazioni relative agli oggetti rappresentati nell'immagine. È necessario considerare, inoltre, che questi dati sono rappresentati in formati spesso differenti ma che possono afferire a standard per facilitare l'utilizzo del dataset in combinazione con altre reti anche non pensate esplicitamente per tale dataset. Una conseguenza di ciò è che risulta conveniente considerare solo dataset con formati standard in quanto possono essere facilmente utilizzati senza scrivere del codice ad hoc dato che quest'ultimo potrebbe potenzialmente essere erroneo o contenere bug. Inoltre se i dataset rispecchiano lo stesso formato possono anche essere combinati tra loro per creare un unico dataset più grande.

Come già detto un dataset racchiude anche una collezione di metadati oltre alle immagini. In questo ambito le principali categorie di metadati sono, ad esempio, labels o etichette che, di fatto, sono una categorizzazione di ogni immagine in una o più classi di riferimento. Tali etichette permettono di associare le immagini a una rappresentazione comune tra loro e di comunicare il fatto che le immagini sono accomunate da dettagli, oggetti presenti al loro interno o condividono qualche caratteristica semantica. Altre informazioni possono essere le bounding box, ovvero delle regioni di interesse in una immagine. Esse sono genericamente aree

rettangolari dell'immagine all'interno delle quali si trova un oggetto importante al fine del dataset. Genericamente ad ogni bounding box è associata una label o comunque un significato logico. Le maschere sono, invece, essenzialmente un sottoinsieme di una bounding box. Esse indicano quindi dove l'oggetto si trova effettivamente tracciandone un contorno pixel per pixel netto. Di solito la maschera risulta essere connessa ma questo spesso dipende dalla presenza o meno di occlusione. Genericamente la maschera può essere definita pixel per pixel in tutta la sua area con un valore vero o falso, una percentuale di confidenza o solo il suo contorno a seconda del formato utilizzato. I keypoint infine, come dice la parola, sono dei punti chiave che indicano punti rilevanti in una immagine, genericamente essi si trovano sul contorno di una maschera ma potrebbero anche essere punti di snodo di uno scheletro formato dagli assi principali di un oggetto.

Altri dati importanti che potrebbero essere inclusi sono attributi delle singole immagini, di fatto semplicemente label aggiuntive o descrizioni più approfondite dell'immagine stessa o relazioni tra label e immagini o ontologie relazionali tra le stesse.

COCO

COCO (Common Objects in Context)[1] è un dataset contenente degli oggetti in ambienti normali e comuni. Il motivo per cui questo dataset è degno di nota è per il formato utilizzato che viene ad essere di fatto un formato standard per molte reti. Caratteristica di questo dataset è l'avere una serie di file JSON contenenti le informazioni a cui ogni immagine fa riferimento. Inoltre esiste una vasta serie di programmi di utilità per caricare tale formato in memoria a seconda della rete utilizzata per evitare di sviluppare una implementazione in proprio. Utilizzando tali interfacce è possibile utilizzare qualunque dataset con questo formato in modo facile e semplice ed evitare di riscrivere codice già esistente.

Il formato, tra le altre cose, prevede una cartella base con al suo interno una cartella denominata "data" all'interno della quale sono contenute le immagini. Sempre nella cartella base è presente un file in formato JSON che contiene, come già detto, i metadati. Tali metadati rispettano una specifica grammatica che può ovviamente essere estesa a casi particolari per rendere più facile e generico il suo utilizzo.

Struttura delle cartelle:

```
<dataset_dir>/
        data/
              <filename0>.<ext>
              <filename1>.<ext>
              …
        labels.json
```

Grammatica del file labels.json:

```
{
        "info": info,
        "images": [image],
        "annotations": [annotation],
        "licenses": [license],
}
image {
        "id": int,
        "width": int,
        "height": int,
        "file_name": str,
        "license": int,
        "flickr_url": str,
        "coco_url": str,
        "date_captured": datetime,
}
annotation {
        "id": int,
        "image_id": int,
        "category_id": int,
        "segmentation": RLE or [polygon],
        "area": float,
        "bbox": [x,y,width,height],
        "iscrowd": 0 or 1,
}
categories [{
```

```
        "id": int,
        "name": str,
        "supercategory": str,
}]`
```

Nello specifico si può evidenziare come nel caso dell'object detection il format possa rappresentare delle annotazioni, contenenti informazioni quali l'id della categoria, la maschera esatta espressa o come un poligono o con un format RLE, la posizione delle bounding box e se l'immagine contiene molti oggetti o un oggetto solo. Altre estensioni di questo formato sono disponibili a seconda di quali dati si voglia rappresentare e nello specifico quale sia il fine ultimo del dataset.

2.2 Pytorch

Pytorch è un framework open source per l'apprendimento automatico[d]. È utilizzato per la ricerca e l'implementazione di reti innovative sia per la sua facilità di utilizzo e versatilità per la prototipazione che per l'ampia gamma di strumenti e reti già implementate e fornite dalle librerie stesse. Molto importante è il concetto di tensore: per tensore si intende un vettore di dati di vario tipo che può essere utilizzato su schede grafiche con capacità di calcolo su CUDA. Questi tensori sono di fatto degli array che a livello implementativo somigliano a quelli di Numpy per interfaccia. Utilizzando tali strutture dati è possibile creare codice in grado di operare in maniera performante e parallela.

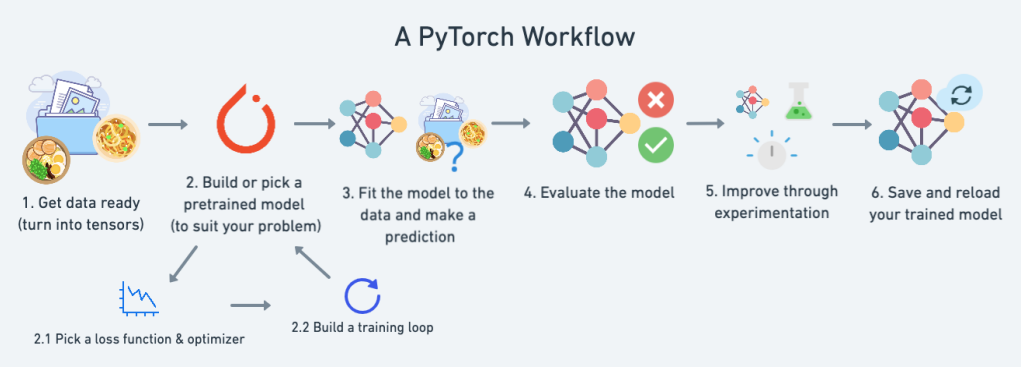

Figura 3: Il WorkFlow di Pytorch tratto dal sito ufficiale.

CUDA[2] è una architettura hardware per l'elaborazione parallela di dati creata da nVidia. Utilizzando tale architettura è possibile sfruttare GPU per effettuare calcoli

non correlati con il rendering grafico, ciò viene detto GPGPU (General Purpose computing on Graphics Processing Units). Usare CUDA ha molteplici vantaggi rispetto ad altre implementazioni. Questi vantaggi possono essere riassunti come memoria condivisa tra thread, letture più veloci da e verso la GPU, supporto per divisione intere e operazioni bit a bit. Ovviamente ci sono anche lati negativi ad esempio una diversa implementazione dello standard float, eventuali colli di bottiglia durante le transizioni tra CPU e GPU. Pytorch permette di interfacciarsi con tale sistema usando delle classi standard che garantiscono compatibilità e portabilità al codice scritto. Inoltre, essendo Python un linguaggio di alto livello, facilita la prototipazione su tali piattaforme.

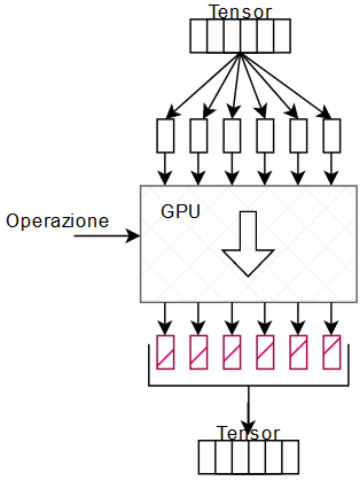

Figura 4: Esempio visuale di come la parallelizzazione funzioni: il tensore viene diviso e, per ogni componente, viene eseguita una operazione secondo il modello SIMD nella tassonomia di Flynn

Vista la sua diffusione nell'ambito della ricerca si è scelto di usare tale libreria per queste sperimentazioni, in ogni caso i seguenti risultati sono riproducibili con ogni principale libreria esistente.

Capitolo 3

I dataset del mondo della moda

I principali dataset analizzati nella presente trattazione sono stati trovati online tramite una ricerca nel panorama della moda. Essi sono principalmente incentrati sul task della classificazione o della segmentazione di abiti. Tramite una analisi di questi dataset è possibile ottenere una buona base di partenza e una comprensione dello stato dell'arte nonché organizzare una valutazione degli strumenti disponibili che fanno da presupposto per svariate esperienze passate. Per trovare i dataset è stata sfruttata principalmente la ricerca su siti di dataset e paper riguardanti la moda e il deep learning.

Molti di questi dataset sono basati su immagini raccolte dal mondo reale tramite scraping da siti di immagini quali Flickr o altri social, nati da collaborazioni con aziende, o direttamente creati da aziende di moda per creare interesse e attenzione a tale ambito e indirizzare più facilmente la ricerca in quelle direzioni. Essendo infatti molto complesso creare un dataset, averne a disposizione uno il più generico possibile realizzato da esperti nel settore in considerazione significa possedere un valore aggiunto: si consideri che la creazione di un dataset è, per le aziende, decisamente una enorme opportunità anche in termini di influenza del settore stesso – nonché di conseguenza un enorme contributo – al mondo della ricerca.

3.1 FashionGen

FashionGen[3] è un dataset principalmente realizzato per lavorare con reti generative che contiene più di trecentomila immagini ad alta definizione fotografate con condizioni analoghe e consistenti. Tutti gli *item* sono stati inoltre fotografati da almeno sei angolazioni differenti a seconda della loro categoria. Il paper di riferimento dichiara che questo dataset è probabilmente il primo per scala e consistenza ad avere queste caratteristiche. Ogni item è inoltre catalogato non solo in una macro categoria che raccomuna i principali tipi di vestiario, ma anche una più granulare sottocategoria avente lo scopo di permettere una più accurata divisione e catalogazione di ogni capo presente. Oltre a ciò il dataset contiene anche una descrizione del capo presente in ogni immagine ottenuta da esperti

designers nel settore della moda. Inoltre per ogni capo è anche presente una serie di metadati che includono caratteristiche quali il materiale, il nome, la stagione di produzione e la marca. Ogni immagine consiste principalmente in una foto di una persona indossante il capo in questione principalmente a figura intera e foto del capo stesso. Come caratteristica comune, ogni immagine contiene solo il soggetto all'interno della stessa, ovvero tutte le immagini hanno sfondo bianco o trasparente.

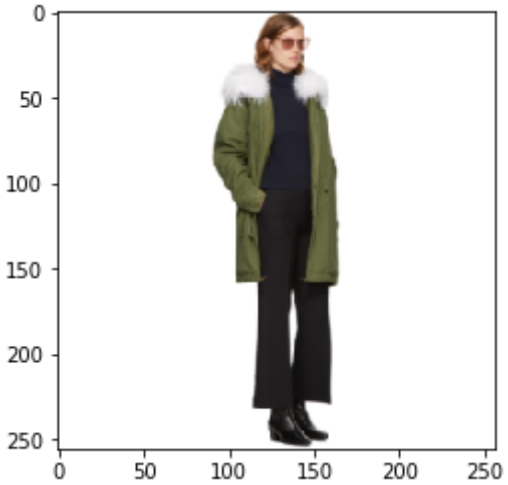

Figura 5: Immagine estratta a caso dal dataset.

Come già accennato questo dataset è principalmente pensato per addestrare reti GAN e P-GAN sia da immagini che dalle descrizioni testuali allegate. In quanto tale esso non contiene alcuna informazione riguardo bounding box, keypoint o maschere. Contiene una serie di labels indicanti, come già menzionato, informazioni quali marca, categoria, sottocategoria, una descrizione breve dell'indumento raffigurato, il genere della persona che indossa tale indumento, il nome del capo e la stagione di uscita del capo rappresentato. Il dataset è in formato HDF5, un formato pensato per contenere grandi quantità di dati numerici spesso provenienti da Numpy. Questo significa che le immagini e tutte le informazioni sono memorizzate come array numerici e già pronte ad essere processate. Il paper allegato illustra anche una P-GAN proposta come baseline per la creazione di nuovi capi a partire dal dataset stesso sia dal nulla che dal testo proposto da un utente.

Un altro task che questo dataset può ricoprire è quello di fornire una base per la generazione di capi di abbigliamento a partire da testo; per fare ciò esiste una rete baseline basata su reti di tipo LSTM in grado di generare capi a partire da

descrizioni. Questo grazie al campo di descrizione che ogni immagine ha allegato ai suoi metadati.

3.2 Fashion MNIST

Questo dataset è stato sviluppato da Zalando[4] come una alternativa al classico dataset MNIST dopo avere realizzato che in molti casi MNIST era diventato molto usato e, in generale, con casi troppo facili per fornire un benchmark ancora valido visto lo stato attuale dell'arte. Il dataset MNIST consiste in un vasto dataset contenente cifre scritte a mano genericamente usato appunto per testare casi di apprendimento automatico; esso contiene sessantamila immagini di training e diecimila di test. Il dataset è stato creato nel 1998 per studiare problemi di riconoscimento automatico dei caratteri. Nel corso del tempo l'utilizzo di tale dataset si è affermato a tal punto da diventare uno standard per il testing delle reti, talmente tanto da diventare addirittura troppo utilizzato e, secondo alcuni, forse di natura controproducente[5, 6].

In questo scenario Zalando ha creato questo dataset, consistente anch'esso in sessantamila immagini di training e diecimila di test, come un rimpiazzo diretto del vecchio dataset MNIST. Siccome rispetta le specifiche del dataset MNIST ogni immagine ha dimensioni 28x28 in scala di grigi ed è salvata nel formato standard MNIST. Il dataset ha inoltre dieci macro classi per indicare la tipologia di vestiti. Le labels disponibili sono: "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot", indicanti la macro tipologia del capo rappresentato. L'idea di questo dataset, rispecchiante l'utilizzo del dataset MNIST è per provare tasks più o meno semplici riguardanti la catalogazione di immagini nelle rispettive classi.

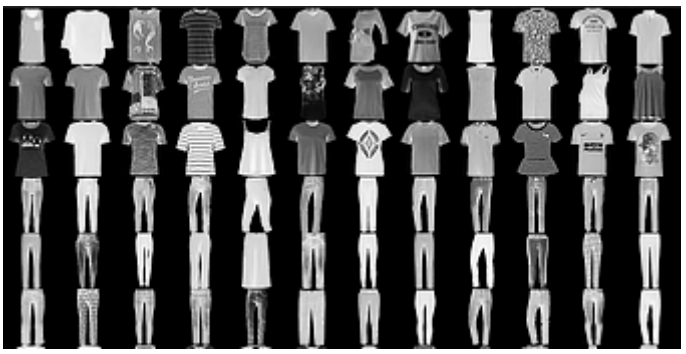

Figura 6: Rappresentazione delle immagini del dataset Fashion MNIST

Essendo usato in tasks di catalogazione esso contiene solo le labels già menzionate e non ha bounding box o maschere anche se sono facilmente ottenibili in quanto le immagini sono tutte a sfondo trasparente. È importante notare come queste immagini siano molto piccole e quindi questo dataset possa servire più come dataset di test che come un dataset di produzione.

3.3 Visuelle

Questo dataset contiene circa 5500 immagini rappresentanti capi di abbigliamento su sfondo trasparente e per ogni capo delle informazioni relative alla tipologia di vestito, la stoffa utilizzata e il colore principale di tale capo[7]. Allegato a tali dati, per ogni label, è presente un insieme di dati che descrivono la tendenza presi dalle ricerche Google Trend dal 2016 al 2019 settimanalmente. Il dataset è stato strutturato per predire il successo di determinati capi in base alle loro caratteristiche principali. Il paper allegato sottolinea come prevedere le vendite sia in generale un classico task comunemente diffuso in applicazioni economiche e finanziarie. L'idea del paper di riferimento consiste nel voler estendere tale concetto a campioni, in questo caso immagini di vestiti, mai visti dalla rete stessa, ovvero fare sì che decisioni genericamente prese in modo soggettivo da esperti in tale campo possano essere automatizzate. Il paper presenta anche un modello basato su un GTM Transformer che cerca di imitare tale comportamento al fine di ottenere una previsione sensata. La rete proposta nel paper descrive un approccio di previsione delle vendite a partire dalle immagini usando i Google Trends e una analisi delle feature di ogni immagine.

Una attenta e accorta analisi deve essere inoltre fatta quando si considerano dati provenienti da Google Trend. È importante considerare che tali dati saranno particolarmente spuri tra di loro e non è detto che, semanticamente, siano sempre relativi ai termini che ci si aspetta come significato. Per fare un esempio a proposito di ciò si può pensare che se si cerca il termine “arancio” ci si potrebbe riferire al colore (e quindi siamo interessati a contarlo nella rete di riferimento) o all’albero che produce le arance e in quanto tale creerebbe rumore se siamo solo alla ricerca di valori che fanno riferimento al colore.

Ogni immagine è rappresentata nel colorspace RGB avente varie dimensioni. Le immagini sono state catturate in un ambiente mediamente controllato anche se alcune di esse presentano segmentazioni non troppo appropriate o categorizzazioni alle volte erronee. Le immagini rappresentano, come già detto, solo il capo di vestiario non indossato al fine di non fornire eventuali bias a una rete addestrata su tali dati.

Figura 7: Un capo di abbigliamento estratto da Visuelle.

Il dataset ha inoltre una serie di dati relativi alle vendite per ogni capo presente su base settimanale in 100 negozi in una azienda di fast fashion italiana. Fattore degno di nota è il fatto che tutti i dati sono riferiti a tale azienda e in quanto tale è necessario ricordare il business model di una azienda di fast fashion: di fatto esso si incentra nel cambiare molto spesso i capi venduti e nel portare capi mostrati in passerella nei negozi il prima possibile. In quanto tale i dati di vendita riportati fanno spesso parte di un solo anno. Dati testuali ulteriori sono la categoria, facente riferimento a 27 etichette rappresentati il tipo di vestiario e alcune delle

caratteristiche fondamentali; il colore, che rappresenta il colore dominante dell'abito; il materiale di cui è composto, preso da un vocabolario di 58 elementi e il periodo di vendita del capo stesso.

Riguardo ai dati Google Trend essi sono semplicemente raccolti per ogni label in un periodo dal 2016 al 2019 e sono principalmente relativi alle categorie, colore e materiale e successivamente normalizzati sul valore massimo.

Visto il fatto che le immagini contengono solo il capo di vestiario senza altri sfondi o oggetti estranei, non sono presenti bounding box, keypoint o maschere ma sono mediamente ottenibili.

3.4 Visuelle 2.0

Il dataset contiene dati analoghi a Visuelle, ma è più incentrato sui dati di vendita; in quanto tale contiene una alta quantità di dati di vendita per ogni immagine[8]. Per la precisione contiene dati per 5355 capi di vestiario prodotti da una azienda di fast fashion nel periodo che va dal 2017 al 2019. Come per Visuelle il dataset è accompagnato da immagini ad alta definizione dei capi di riferimento e informazioni testuali sugli stessi. I dati presenti sono principalmente gli stessi presenti in Visuelle ma hanno in aggiunta le informazioni sulle vendite. Tali informazioni sono complessivamente più di seicentomila valori relativi a ogni capo presi da diversi negozi sul territorio tramite carte fedeltà. I dati contengono informazioni quali identificativo del prodotto comprato, date di riferimento, quantità comprata in tale periodo e identificativo del negozio di riferimento. Questi dati risultano quindi essere, per ogni capo, per ogni negozio e per ogni settimana, una quantificazione delle vendite e dei riassortimenti effettuati per ogni negozio. Inoltre sono riportati anche valori meteorologici per ogni area al fine di predire una correlazione tra fattori meteo e l'acquisto di particolari capi di abbigliamento.

Come in precedenza gli autori propongono una rete di riferimento che funge da baseline per il dataset strutturata principalmente su reti ricorsive e analisi delle serie; in questo paper la rete è incentrata sulla previsione del riassortimento di un negozio sulla base delle vendite complessive di ogni capo. Il paper illustra come tale rete si comporti se comparata con altre soluzioni basate su kNN, metodi naïve

– per fare ciò viene descritto di usare gli ultimi valori osservati come previsione del futuro – o appunto una rete basata su RNN e meccanismi di attention. Viene inoltre spiegato come i fattori visivi, ovvero le feature di ogni immagine, vadano a svolgere un ruolo chiave per la corretta previsione di ogni serie futura.

Una problematica esistente in tale dataset è la presenza di immagini non ritagliate correttamente, immagini non sempre fotografate in posizioni analoghe e errori di labelling, spesso relativi a colori. La bassa cardinalità di alcune categorie può inoltre avere dei fattori negativi se si impiega questo dataset in task di classificazione. Altra problematica, ma insita nei dati stessi, è il fatto che, facendo riferimento a una azienda di fast fashion, i dati per ogni capo di abbigliamento si rapportano a un solo anno o a una sola stagione.

3.5 DeepFashion

Questo dataset contiene più di ottocentomila immagini categorizzate su più di 50 labels per tipologia[9]. In tale dataset sono disponibili alcuni sottoinsiemi dello stesso che possono essere utilizzati per la predizione di attributi sui vestiti, consumer to shop retrieval o altri task minori[10, 11]. Le immagini sono associate a vari attributi quali keypoint e descrizioni sia dettagliate che generiche, ma non sono disponibili delle bounding box. Alcune fonti sostengono che i keypoint non sono definiti in maniera uniforme all'interno del dataset. Questo dataset è stato presentato principalmente per sopperire alla mancanza di grandi dataset per classificare i tipi di vestiti e eventuale estrazione di attributi da ogni immagine.

Un altro particolare di questo dataset è la divisione tra vestiti "shop" e vestiti "street": i primi sono, infatti, stati presi da negozi online mentre i secondi sono stati ottenuti o da social o comunque da persone comuni. Questo può essere utile per quei tasks dove l'obiettivo è riconoscere vestiti indossati da qualcuno tra esempi riportati in un catalogo digitale online.

Figura 8: Un esempio di una immagine "shop" dal dataset DeepFashion.

Il dataset non è in formato COCO e necessita di una implementazione ad hoc per potere essere interfacciato con reti che usano tale formato.

Allegato a questo dataset è presente una rete baseline per l'esecuzione dei task sopraelencati.

3.6 DeepFashion2

L'obiettivo di questo dataset è quello di ovviare ad alcune mancanze presenti nel dataset DeepFashion come la mancanza di maschere, bounding box o landmark[12]. Inoltre cerca di sopperire a errori o all'utilizzo non standard dei keypoint in DeepFashion. Questo dataset infatti contiene 491000 immagini categorizzate in tredici diverse macro categorie ottenute sia da negozi online che da consumatori e utenti di tali siti. Ogni capo di abbigliamento è inoltre categorizzato con informazioni quali la scala, l'occlusione, la categoria, lo stile e le già menzionate maschere, bounding box e landmark. Inoltre i dati sono riportati in formato compatibile con COCO[13], cosa che rende questo dataset facilmente accessibile ed utilizzabile in svariate reti già esistenti che utilizzano tale formato.

Figura 9: Un esempio di una immagine "street" dal dataset DeepFashion2.

Il dataset è composto da cartelle contenenti gli split del dataset in train, test e validation. Questi ultimi contengono metadati, come già detto, in formato compatibile con COCO. È importante ricordare come il dataset usi un sistema standard di landmark, ovvero keypoint standard per ogni tipologia di vestiario sempre presenti nelle stesse posizioni. Questo può anche essere utilizzato come punti per pose estimation di persone in una immagine, task presente nelle reti di benchmark proposte.

Altri task proposti con questo dataset sono, appunto, la classificazioni di vestiti all'interno delle principali label contenute, fashion understanding (ovvero analizzare le immagini per ottenere come sono fatte) e la segmentazione dei capi nell'immagine.

Per la parte di labelling il paper illustra come per le categorie e per le bounding box si siano usati annotatori umani al fine di assegnare una categoria e una bounding box per ogni capo di abbigliamento. Si nota inoltre come DeepFashion abbia più di 50 label ma come esse abbiano cardinalità molto basse o comunque siano ambigue tra loro. Per evitare ciò DeepFashion2 ha solo tredici classi. Anche per i landmarks gli annotatori hanno segnato anche se il landmark era visibile o occluso per ognuno di essi al fine di ottenere una massima precisione anche in assenza della presenza visibile dello stesso. Le maschere invece sono state generate automaticamente dai contorni e poi rifinite in maniere manuale.

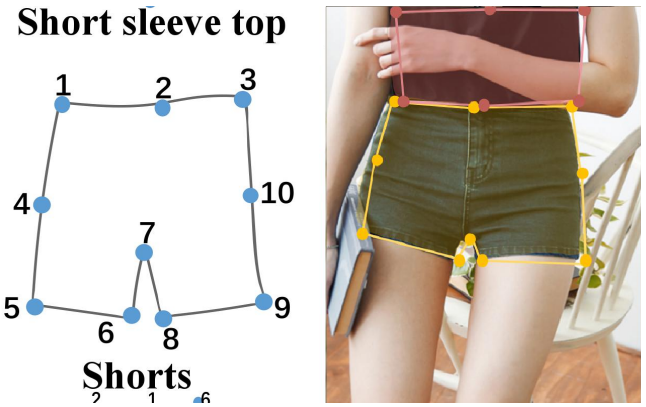

Figura 10: Nell'immagine, i keypoint e il contorno di un paio di pantaloncini nel dataset DeepFashion2

Collegato a questo dataset è presente la rete Match R-CNN che svolge sia la parte di estrazione di maschere, label e bounding box sia la parte street-to-shop extraction. Si farà riferimento a questa rete più avanti in questa trattazione.

3.7 FashionPedia

Questo dataset contiene una ontologia molto complessa per la definizione di categorie di vestiti[14, 15]. Tale ontologia è composta principalmente da 27 macrocategorie e 300 attributi definiti da esperti nel mondo della moda. In questo database sono presenti più di quarantottomila immagini ognuna con maschere di segmentazioni e bounding box.

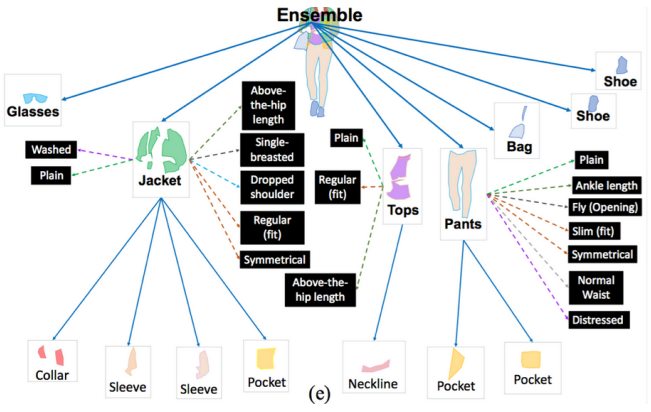

Figura 11: L'ontologia di FashionPedia relativa agli attributi e alle segmentazioni.

L'idea di questo dataset è quella di creare una unione tra ModaNet e DeepFashion2, due importanti dataset nel mondo della moda, finalizzata al contenere sia labels generiche relative alla tipologia (come ad esempio "Jacket")

che attributi riguardanti il capo rappresentato stesso (e.g. “Pockets”) con relativa estrazione dei punti salienti. Le maschere di segmentazione presenti non sono solo relative all’indumento nel suo complesso ma anche alle sue sottoparti: sono disponibili infatti maschere per l'estrazione di colletti, borchie, maniche e altri particolari che compongono un vestito, il che rende questo dataset adatto ad addestrare reti a estrarre parti di un capo specifico. Inoltre è stato effettuato il raccoglimento di un insieme di dati su cui è stata effettuata una serie di operazioni di descrizione, ad esempio il riconoscimento di attributi, la localizzazione, ovvero indicare dove nell’immagine questi vestiti si trovino, e la denominazione, intesa come riconoscimento della categoria a cui appartiene un capo di abbigliamento. Il paper di riferimento evidenzia come il riconoscimento degli attributi, e non solo delle classi stesse, possa aiutare in maniera considerevole qualora si voglia utilizzare la rete anche su insiemi di dati che presentano classi non appartenenti al dataset di addestramento.

Come già menzionato questo dataset presenta una ontologia complessa, consistente non solo in una serie di categorie e attributi, ma anche di una serie di relazioni tra gli stessi. Questo consente di creare un sistema molto articolato che permette di categorizzare svariati casi mantenendo una accuratezza molto alta durante task di classificazione. Per la precisione si definiscono categorie generali quelle presenti per ogni indumento; pertanto, ogni indumento è etichettato per ogni sua parte principale con una di queste categorie. Una serie di attributi più precisa è invece utilizzata in seguito per identificare la tipologia di vestito e come essi siano fatti. Queste categorie rispettano le categorizzazioni sia di tipo (e.g. “Giacca”), che di forma (e.g. “Simmetrico”) che di funzione (e.g. “Pocket”). Esiste inoltre una serie di relazioni stabilite per ogni categoria e attributi al fine di garantire una consistenza nella ontologia.

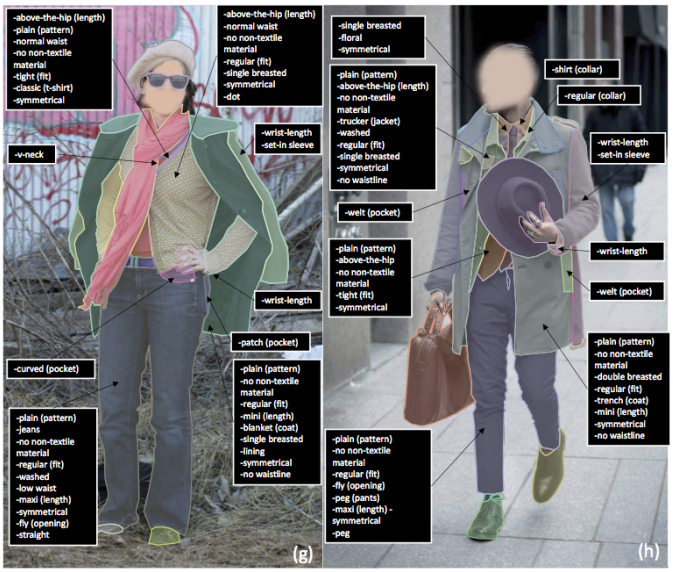

Figura 12: Esempio di attributi e segmentazioni a partire da una immagine definiti in FashionPedia.

Le immagini sono state raccolte da siti online quali Flickr e altri siti privi di licenza o con licenza free; tale raccolta è stata effettuata a mano per evitare presenza di bias e avere solo immagini che garantissero una diversità maggiore possibile. A seguito di ciò le immagini sono state annotate a mano da esperti nel settore, sia per le maschere che per le annotazioni.

Il dataset è strutturato in formato COCO con aggiunta di alcuni attributi per ottenere informazioni più dettagliate e approfondite; tali informazioni variano dal tipo di colletto alla lunghezza di un vestito. Questi attributi permettono di ottenere una descrizione molto accurata del vestito, o comunque di estrarre caratteristiche molto ben definite dalle quali è possibile fare ulteriori analisi[16].

Collegata a questo dataset è presente una rete baseline di riferimento che sarà discussa più avanti in questa trattazione.

Capitolo 4

Deep learning

Le reti neurali sono modelli computazionali nei quali un insieme di neuroni artificiali si scambiano informazioni tra loro al fine di raggiungere un risultato. Questi modelli prendono spunto dalle reti neurali biologiche presenti negli esseri viventi, ma, essendo appunto modelli, risultano essere molto semplici rispetto alle complessità delle reti neurali reali. Recentemente questo approccio è stato molto sviluppato, come già detto, nel campo dell'intelligenza artificiale principalmente in settori nei quali si vuole imparare dai dati raccolti o da esempi noti.

4.1 Obiettivi delle reti neurali nella computer vision

Nell'ambito della computer vision si possono individuare almeno tre obiettivi principali che le reti neurali possono avere[c]. In molti casi questi obiettivi non sono distinti, ma collegati tra loro. Un primo obiettivo è la classificazione di immagini, attività finalizzata ad indicare se una immagine, o più in generale un input, fa parte o meno di un insieme predefinito di possibilità. Questo risulta essere di fatto una versione un po' più approfondita dei problemi di clustering, in quanto un insieme di dati di ingresso è raggruppato a seconda di caratteristiche comuni. Ovviamente per potere distinguere le varie categorie occorre che le classi presenti siano effettivamente distinte da qualche caratteristica che la macchina sia in grado di raccogliere e codificare.

Un obiettivo differente è invece quello della regressione. Nella regressione si cerca una funzione, chiamata appunto regressore, che cerca di stimare al meglio una funzione esistente. È possibile, facendo restituire alla rete un valore numerico, impiegare una rete neurale per scopi di regressione se non si conosce la funzione che si vuole regredire e si hanno solo a disposizione dei campioni spuri. Il vantaggio di questo metodo è che a differenza della regressione normale non è necessario inserire una funzione base da cui fare partire la regressione. Questo metodo risulta essere molto potente laddove si abbiano solo i valori o osservazioni della realtà. Come nel caso precedente, occorre ci sia una correlazione logica tra i dati di input e il valore che si vuole regredire

Il terzo obiettivo trattato è la segmentazione dell'immagine, o più in generale la localizzazione di un oggetto in una immagine e la creazione di bounding box. In questo caso la rete cercherà nell'immagine dove si trova un oggetto, o, in casi molto più specifici, ritaglierà dall'immagine solo quell'oggetto. Questo task risulta complesso per vari motivi: intanto si può verificare occlusione, ovvero l'oggetto non è perfettamente visibile perché nascosto da altri oggetti, oppure l'oggetto rappresentato potrebbe avere ombre, colori o forme leggermente diverse da quelli con cui la rete è stata addestrata. Questo task, inoltre, è quello che ha bisogno di reti più potenti e complesse. Questa segmentazione è collegata al concetto di segmentazione semantica, ovvero il fatto che ogni area segmentata ha un significato semantico differente dalle altre. Connessa con questa attività c'è quella di trovare le bounding box per un oggetto. Questo task, spesso in combinazione con la classificazione, mira a trovare non solo dove l'oggetto si trovi nell'immagine, ma anche che oggetto sia.

4.2 Struttura di una rete neurale

Di base un modello consiste in un insieme di interconnessioni tra neuroni artificiali che ricevono in input un valore numerico, lo processano sulla base di pesi definiti e restituiscono un output numerico[41]. Tali neuroni sono collegati tra loro in maniera variabile a seconda della configurazione della rete. Nelle reti più semplici, quelle lineari, i neuroni sono collegati tra loro in maniera diretta dall'inizio alla fine della rete. Questo però fa sì che anche la rete abbia un output tendenzialmente lineare, cosa che non permette di modellare la realtà in maniera ottimale. Alcune strategie possono essere applicate per rendere tali output non lineari. In tali contesti si parla di strati (o layer) per indicare ogni insieme di neuroni che è presente tra l'input e l'output. Il numero di tali neuroni è variabile anche a seconda del tipo di strato in cui ci si trovi e dalla complessità della rete. È importante menzionare il fatto che più una rete sarà profonda e avrà più strati più le risorse di calcolo necessarie sia per addestrarla che per utilizzarla saranno maggiori.

Genericamente parlando un neurone riceve in ingresso una serie di stimoli, ovvero un insieme di dati numerici, da altri neuroni e li elabora tramite una funzione che, per semplicità, si può pensare come la somma pesata di ogni input.

Successivamente tale somma viene resa non lineare da una funzione di attivazione che fa passare tale valore solo se supera una certa soglia. In caso di reti completamente connesse, laddove tutti i neuroni di uno strato sono connessi con tutti i neuroni dello strato precedente e successivo, questo procedimento avviene per ogni neurone.

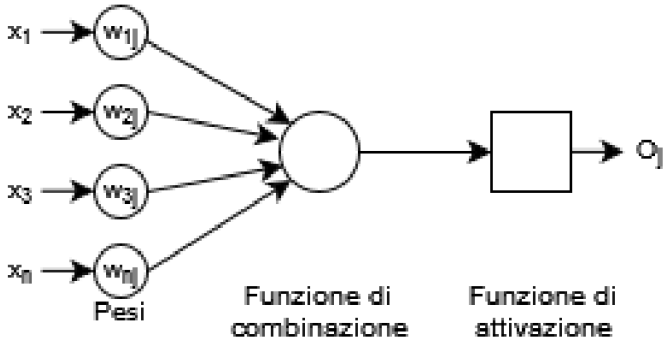

Figura 13: Esempio semplificativo di un neurone.

Il susseguirsi di queste operazioni fa sì che, per ogni immagine mostrata, la rete impari a riconoscere alcuni pattern che rendono vera una condizione di controllo. Da tale condizione è dunque possibile tarare la rete in modo che possa assumere risultati sempre più corretti per ogni ciclo di addestramento che viene effettuato. Questo ciclo di addestramento è detto epoca.

Per migliorare l'addestramento è possibile utilizzare la retropropagazione dell'errore ovvero un procedimento, usato spesso in combinazione con un ottimizzatore, che consiste nel calcolare una funzione di costo, o di perdita, per ogni risultato ottenuto. L'obiettivo diventa quello di ridurre al minimo la funzione di costo per ottenere un risultato più corretto possibile. Questo metodo può essere utilizzato solo in casi in cui si effettua un addestramento nel quale il dataset ha a disposizione le risposte al problema che la rete vuole risolvere.

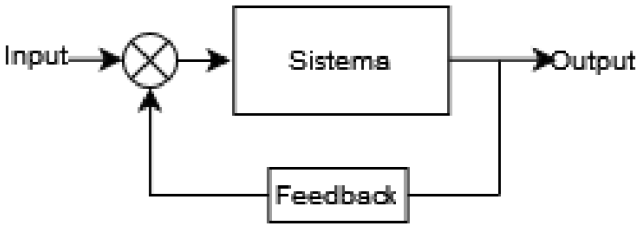

Figura 14: Diagramma della retroazione di un sistema chiuso.

Una volta calcolato tale errore è possibile utilizzarlo per bilanciare i pesi della funzione, ovvero tutte le connessioni tra i neuroni affinché, durante l'esecuzione successiva della fase di training, sia possibile ottenere un valore minore come errore. Questo fa sì che la rete gradualmente migliori le sue predizioni. Logicamente ciò significa che per ogni addestramento si dovrà inizializzare la rete con valori casuali, o in alcuni casi con valori medi per ottenere un addestramento più rapido, e poi procedere con i cicli di addestramento. Per ogni ciclo si avrà una fase in avanti, dove il pattern verrà estratto casualmente tra i valori di input e nel quale si calcolano i valori dei singoli neuroni. Questi valori saranno dunque tutti processati dai neuroni finché non si raggiungerà la fine della rete, si potrà dunque confrontare tale valore col valore atteso dalla rete per potere quindi calcolare l'errore. Tale errore, come detto in precedenza, verrà utilizzato per bilanciare e ricalcolare i pesi della rete per poter procedere col successivo ciclo di addestramento.

Strategie di ottimizzazione utilizzabili sono, ad esempio, la discesa stocastica del gradiente (SGD). Questo metodo funziona iterativamente per ottimizzare funzioni differenziabili e si può applicare in tutti i casi in cui la funzione da ottimizzare ha la forma di una somma. Si tratta di un metodo probabilistico che utilizza, invece del gradiente stesso, una stima del gradiente su un sottoinsieme della funzione costo. È uno dei metodi più usati nel machine learning per addestrare la maggior parte delle reti neurali artificiali.

Un'estensione di questo metodo è Adam (Adaptive moment estimation) che tiene conto anche di altri parametri quali il momento primo e secondo. Questo metodo risulta spesso più stabile di SGD ma per questa trattazione, come risultati, i due metodi sono equiparabili.

4.3 Le funzioni di perdita

La funzione di perdita, o loss, è la funzione che viene minimizzata durante il training. Essa può essere di tipi differenti a seconda dell'operazione che si vuole svolgere. Per task di regressione essa misurerà la distanza tra il valore ottenuto e quello reale, mentre per task di classificazione lo stesso concetto sarà invece applicato alla confidenza di avere trovato la classe giusta.

4.3.1 L1 Loss

Questo tipo di loss è essenzialmente la distanza in valore assoluto tra i punti trovati[17]. Per la sua natura viene usata principalmente per task di regressione, è meglio usare questa in caso siano presenti degli outliers.

$$L(x,\ y)\ =\ \left\{l_1,\ \cdots,\ l_n\right\},\ l_i\ =\ \left|x_i - y_i\right|$$

È implementata dalla classe `torch.nn.L1Loss`.

4.3.2 MSE Loss

Questa loss è di fatto la distanza euclidea tra due punti, in questo caso i valori trovati e quelli di riferimento[18]. Anche questa loss è principalmente da usare per task di regressione.

$$l_2(x,\ y)\ =\ \left\{l_1,\ \cdots,\ l_n\right\},\ l_i = \left(x_i - y_i\right)^2$$

È implementata dalla classe `torch.nn.MSELoss`.

4.3.3 Cross Entropy Loss

Di tutte le loss trattate fino a questo punto risulta essere la più complessa. Questa loss si applica per casi di classificazioni[20].

È implementata dalla classe `torch.nn.CrossEntropyLoss`.

4.4 La validation e il suo significato durante l'addestramento

Come già menzionato lo scopo dell'addestramento è quello di mostrare ripetutamente il dataset alla rete al fine di calibrarla per riconoscere e classificare correttamente quei campioni. Siccome spesso mostrare una sola volta il dataset non risulta sufficiente, si mostra il dataset più volte alla stessa rete in modo sequenziale. Come definizione si parla di "epoca" ogni volta che un dataset intero viene mostrato alla rete. Segue che se si mostra due volte il dataset alla rete si avranno due epoche. Un altro metodo di definizione ma del tutto analogo è quello di "step" nel quale vengono contati i campioni totali mostrati, ovvero se il dataset è grande 15 e ci sono 30 step si mostra il dataset due volte assumendo di sottoporre alla rete una immagine alla volta. Un addestramento di questo tipo viene eseguito nel suo intero fino a quando il numero predeterminato di cicli non termina.

Un problema che viene a crearsi, però, è quello dell'overfitting. Si parla di overfitting per indicare il fatto che la rete si è "specializzata troppo" sul dataset di addestramento, ovvero la rete è in grado di riconoscere perfettamente ogni elemento del dataset ma non ha generalizzato e non è in grado di riconoscere nuovi dati e di comportarsi di conseguenza in maniera sperata. Per questo si fa seguire a ogni epoca (o a un numero predeterminato di esse) una procedura detta validazione. Durante la validazione si mostrano campioni che non sono mai stati mostrati alla rete prima e di cui si conosce l'output atteso della rete. Si calcola quindi la loss su quelle immagini e se è minore di ogni loss di validazioni mai ottenuta si considerano quei pesi ottenuti come validi altrimenti si continua ad addestrare finché non si trovano risultati migliori. Questo processo permette di eliminare il problema dell'overfitting e di ottenere risultati che approssimano al meglio casi reali.

4.5 Struttura delle reti neurali

Avendo fornito un'idea di come il processo di addestramento funziona ci si può ora concentrare sui modelli di reti principalmente esistenti. Come già menzionato i modelli sono basati su nodi intercomunicanti tra di loro. Ogni nodo performa un'operazione dato un ingresso e la sua uscita si propaga a uno o più neuroni di uno strato successivo. Uno strato è dunque un insieme di nodi con caratteristiche simili

posti alla stessa “altezza” nell’albero della rete. Di solito si parla di strato di input per indicare quello che di fatto riceve ed elabora i dati in ingresso dall’esterno, di strato nascosto, cioè quello che di fatto elabora i dati in maniera completa, e strato di output, che semplicemente raccoglie i risultati per passarli all’uscita della rete. Ovviamente ognuno di questi strati è a sua volta composto da vari singoli sottostrati con funzioni e forme differenti.

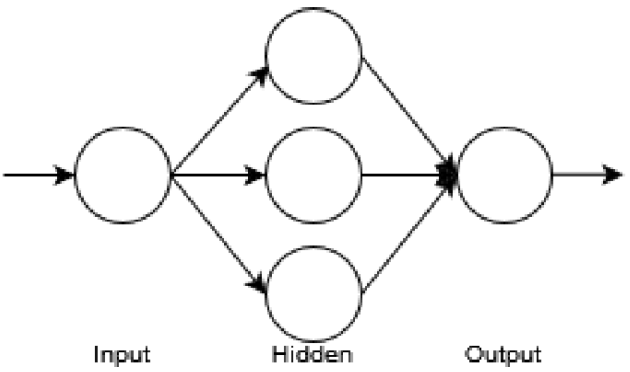

Figura 15: Rappresentazione dei tre strati principali di una rete neurale profonda.

4.5.1 Strati lineari

Uno strato lineare è di fatto l’applicazione di una trasformazione lineare dei dati in ingresso[20]. Il concetto è semplicemente quello di prendere i dati, applicare un peso ad ognuno e di fornire in uscita tale risultato. Matematicamente è scritto come:

$$y = xA^T + b.$$

Questi strati sono molto semplici e, in Pytorch, sono implementati dalla classe `nn.Linear`. Va notato che, essendo lineari, da soli possono solo approssimare dipendenze lineari. Questo è, di fatto, un lato negativo poiché la maggior parte delle relazioni non è lineare. Per ovviare a questo problema si usano delle funzioni di attivazione quali ReLU o Sigmoide che attivano un neurone solo se il valore di uscita supera una certa soglia. Questo accorgimento permette di approssimare relazioni non lineari con strati lineari.

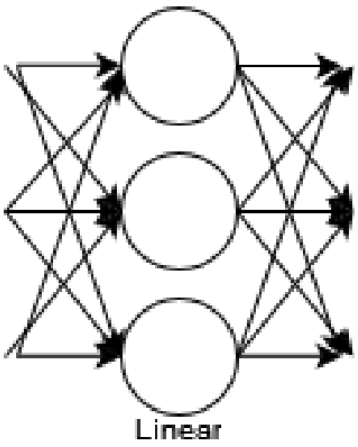

Figura 16: Esempio di uno strato lineare connesso.

L'uso di questi strati è spesso utilizzato per risolvere problemi di classificazione e come strati di uscita di una rete.

4.5.2 Strati convoluzionali

Uno strato convoluzionale, invece, è uno strato che applica un'operazione di convoluzione sul suo input[21]. Questo strato è stato creato al fine di imitare l'organizzazione della corteccia visiva animale. Lo scopo è quello di effettuare una preelaborazione, una riduzione della complessità dell'input in una maniera ispirata dai processi biologici. Come menzionato questi strati sono principalmente utilizzati per l'analisi di immagini, ma possono essere anche utilizzati per trovare correlazioni tra serie di dati numerici. L'utilizzo principale di queste reti è quello di categorizzare le immagini secondo vari criteri o direttamente estrarre feature da una rete per poi passarle in analisi ad altre reti o altri sistemi. Le feature sono un aspetto molto importante delle reti convoluzionali quando si parla di immagini: di fatto si può pensare ad esse come una sorta di "descrizione" delle caratteristiche salienti di una immagine che possono essere usate sia per riconoscere l'immagine stessa che elementi rilevanti al suo interno.

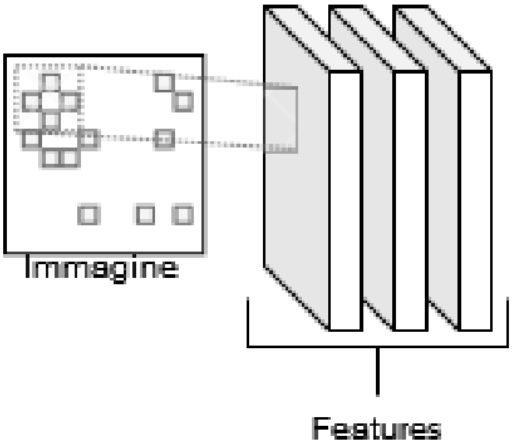

Figura 17: Input e output di una rete convoluzionale, si noti l'aumento di dimensioni dell'uscita.

L'idea di base è quella di applicare una convoluzione 2D su un segnale di ingresso, matematicamente significa ottenere uno strato che per ogni ingresso ha come uscita:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{in} - 1, k) \star input(N_i, k).$$

Ovviamente questa formula vale nel caso discreto in quanto i pixel di una immagine sono discretizzati per loro natura. Si confronti tale formula con la definizione continua:

$$(f \star g) := \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau.$$

In Pytorch è implementato dalla classe `nn.Conv2d` per il caso bidimensionale e `nn.Conv1d` per il caso monodimensionale. Genericamente parlando per creare una rete convoluzionale si usano principalmente questi blocchi: lo strato convoluzionale, uno di pooling per raccogliere le informazioni in uscita, uno strato di attivazione (tipicamente una ReLU) e uno strato lineare per la classificazione in uscita dell'immagine.

4.5.3 ResNet

Una rete neurale residuale, o ResNet[22, 23], è usata per estrarre feature da una immagine. Può essere pensata come una rete convoluzionale molto potente

realizzata per estrarre feature ed essere usata come backbone, ovvero come elemento base, utilizzabile da reti più complesse. Anche qui l'idea è di avere molteplici strati convoluzionali, alcuni dotati di bypass a strati successivi, per effettuare l'estrazione di feature. Utilizzando questo metodo è possibile aggiungere molteplici strati residuali: le implementazioni standard di Pytorch arrivano fino a 101. Si nota che l'utilizzo di questa rete come backbone permette di creare reti complesse e veloci avendo a disposizione direttamente le feature estratte da una immagine e senza bisogno di progettare tali strati. Per di più esistono implementazioni già fatte e addestrate all'estrazione di feature su dataset molto complessi che possono essere sfruttate in progetti con dataset più piccoli. Questo permette di ridurre la complessità del dataset e rendere l'addestramento più semplice.

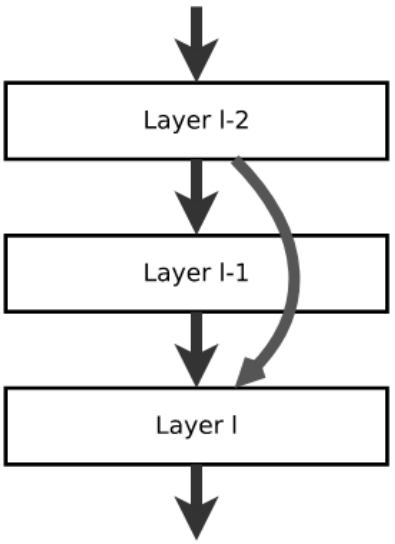

Figura 18: Rappresentazione della forma canonica della ResNet, si noti lo skip del layer l-1.

Capitolo 5

Sperimentazione con reti neurali profonde

Ci si pone un obiettivo principale: la segmentazione di vestiti presenti all'interno di una immagine; si è scelto, pertanto, di analizzare le tecnologie presenti e, se necessario, di implementarne una in proprio. In questa fase si analizzeranno le tecnologie più diffuse basate sulla segmentazione di immagini usando la rete MaskRCNN o comunque dei derivati della stessa.

5.1 Il caricamento dei dati

Pytorch, come già menzionato, può essere usato principalmente per sviluppare reti neurali profonde (Deep Neural Network) e per accelerare le stesse utilizzando GPU. Ciò significa che usando le librerie di Pytorch è possibile velocizzare calcoli, spesso parallelizzabili o necessitanti di risorse maggiori, all'interno di una scheda grafica. Per lo svolgimento di alcuni test di seguito riportati ci si è avvalsi di computer aventi fino a quattro schede grafiche RTX-3090 e più di 256 GB di RAM. Tuttavia è possibile ripetere gli stessi esperimenti su qualunque computer a scapito ovviamente del tempo necessario a svolgere tali operazioni.

Per comprendere al meglio cosa e soprattutto come le reti neurali lavorino è opportuno effettuare una analisi delle tecnologie ad oggi a disposizione, per capire come le soluzioni rappresentanti lo stato dell'arte siano strutturate.

5.1.2 Dataloader e struttura in Pytorch

Un dataloader[24] è una struttura dati che si occupa di gestire e caricare in memoria un dataset; serve per rendere il processo di caricamento dei dati più semplice, ottimizzato e gestibile. Di fatto è strutturato come un oggetto software implementante un iteratore in grado di fornire il prossimo elemento del dataset assieme ai metadati associati. In Pytorch questo sistema è implementato da due classi principali: Dataset e DataLoader. La classe Dataset sarà dunque una rappresentazione del dataset da caricare in memoria, con eventuali miglioramenti e collazioni ai dati. Al suo interno è rappresentato l'oggetto immagine da passare in seguito alla rete che si sta addestrando. La classe DataLoader, invece, è quella che

implementa l'iteratore e si occupa di caricare in memoria il dataset e permette di estrarre i singoli campioni.

Per prima cosa occorre scegliere un dataset fra quelli elencati in precedenza come riferimento, in questa serie di esempi si è scelto DeepFashion2 in quanto è quello con dati più variegati e annotato in maniera consona. Per caricare il dataset occorre semplicemente utilizzare il già menzionato formato COCO. DeepFashion2 infatti contiene annotazioni che possono essere convertite in formato COCO tramite uno script fornito. Le informazioni principali dunque saranno:

- `box`, ovvero le bounding box per ogni capo di abbigliamento presente nell'immagine in questione;
- `labels`, ovvero a quale categoria le singole box appartengono;
- `image_id`, ovvero un identificativo associato all'immagine;
- `masks`, ovvero la maschera esatta di come è fatto l'oggetto, in formato RTE, convertibile in una maschera normale grazie alla funzione `pycocotools.coco.COCO.annToMask(...)`;
- `keypoints`, ovvero i punti salienti e caratteristici di ogni oggetto presente nell'immagine.

Una volta lette queste informazioni dal file occorre semplicemente convertirle in dizionari Python e passarli alla rete da addestrare in questione.

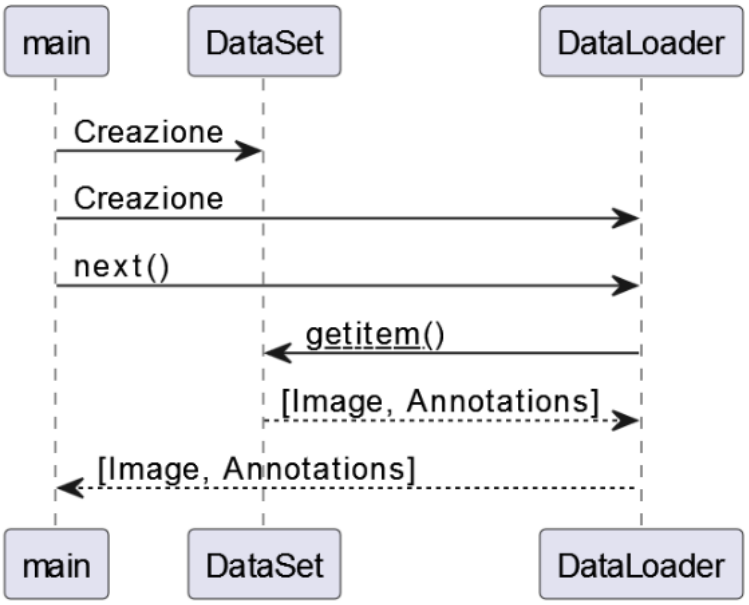

Figura 19: Relazione tra DataSet, DataLoader e chiamante.

Per l'implementazione della classe di lettura si fa riferimento alle interfacce di Pytorch relative ai dataset di cui si è parlato in precedenza.

```
class DataSet(torch.utils.data.Dataset):
      def __init__(self, root, annos, transforms=None):
            self.root = root
            self.transforms = transforms
            self.coco = COCO(annos)
            self.ids = list(sorted(self.coco.imgs.keys()))
      def __getitem__(self, idx):
            coco = self.coco
            img_id = self.ids[idx]
            ann_ids = coco.getAnnIds(imgIds=img_id)
            coco_annotation = coco.loadAnns(ann_ids)
            path = coco.loadImgs(img_id)[0]['file_name']
            img = Image.open(os.path.join(self.root, path))
            num_objs = len(coco_annotation)
            boxes = []
            labels = list()
            keypoints = list()
            for i in range(num_objs):
                  xmin = coco_annotation[i]['bbox'][0]
                  ymin = coco_annotation[i]['bbox'][1]
                  xmax = xmin + coco_annotation[i]['bbox'][2]
                  ymax = ymin + coco_annotation[i]['bbox'][3]
                  boxes.append([xmin, ymin, xmax, ymax])
                  labels.append(
                        coco_annotation[i]["category_id"]
                  )
                  kp = coco_annotation[i]["keypoints"]
                  keypoints.append(kp)
            boxes = torch.as_tensor(boxes, dtype=torch.float32)
            img_id = torch.tensor([img_id])
            areas = []
            for i in range(num_objs):
```

```
            areas.append(coco_annotation[i]['area'])
        areas = torch.as_tensor(areas, dtype=torch.float32)
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)
        masks = list()
        for i in range(len(coco_annotation)):
            masks.append(
                np.array(coco.annToMask(
                    coco_annotation[i])
                )
            )
        my_annotation = {
            "boxes": boxes,
            "labels": torch.as_tensor(
                        labels,
                        dtype=torch.int64,
            ),
            "image_id": img_id,
            "area": areas,
            "iscrowd": iscrowd,
            "masks": torch.as_tensor(
                        masks,
                        dtype=torch.uint8,
            ),
            "keypoints": torch.as_tensor(
                        keypoints,
                        dtype=torch.float,
            ),
        }
        return img, my_annotation
```

5.2 FasterRCNN

Questo tipo di rete è basata sulla classe delle Region-based CNN, questo significa che il suo fine ultimo è quello di indicare una area di una immagine all'interno della quale si trova presumibilmente l'oggetto cercato[25]. L'uscita di questa rete

sarà una o più bounding box all'interno delle quali la rete ritiene con una adeguata confidenza ci sia un campione degli oggetti cercati. Questa rete risulta avere più di una testa e oltre all'area di riferimento essa restituisce anche la classe dell'oggetto che ritiene essere in quell'area, risolvendo così sia un problema di classificazione che di localizzazione. Questa rete è un miglioramento di un'altra rete chiamata FastRCNN[26] e il paper di riferimento sostiene che questa implementazione risulti essere molto più veloce (da cui il nome). Inoltre questa rete usa una RPN (Region Proposal Network) che condivide parametri con gli strati convoluzionali per implementare, tra le altre cose, il meccanismo dell'attenzione sulle feature provenienti dagli strati convoluzionali.

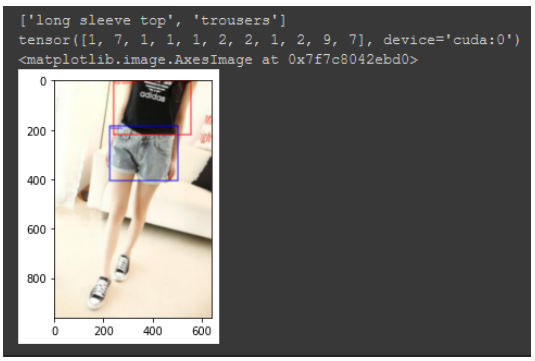

Figura 20: L'uscita della rete dopo avere visto l'immagine, si notino le label di uscita.

Il suo funzionamento è il seguente: in un primo strato la rete RPN propone una area candidato per diventare una bounding box e nel secondo passaggio un'altra rete, di fatto quello che era FastRCNN, estrae feature da uno strato chiamato RoIPool per ogni candidato e performa sia una classificazione che la regressione per le bounding box. Le feature usate da questi due strati risultano essere condivise per una implementazione più rapida.

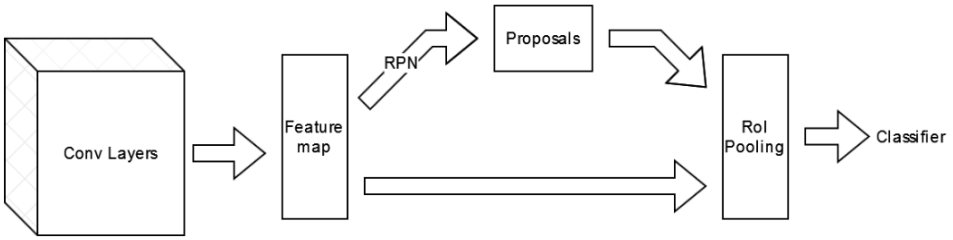

Figura 21: Diagramma di flusso della struttura di FasterRCNN.

Questa rete usa una backbone VGG16 nel paper di riferimento, ma una ResNet50 nell'implementazione di Pytorch, ciò non è però rilevante ai termini dell'analisi delle prestazioni in quanto praticamente equivalenti come funzioni. Gli esempi di test in seguito sono stati calcolati, come detto, su un sottoinsieme del dataset DeepFashion2 e, nonostante il basso numero di immagini utilizzate, ha rivelato un'accuratezza particolarmente alta, dimostrando come questa rete sia scalabile e versatile in molteplici contesti.

5.2.1 Dettagli implementativi

Fatte queste considerazioni, è possibile scaricare una rete già implementata dall'hub online di Pytorch. Facendo ciò si può testare e analizzare il comportamento della rete al fine di capire cosa faccia e quali siano i risultati ottenibili. Operativamente parlando, una volta scaricata la rete occorre addestrarla per un numero fissato di epoche mostrando per ogni epoca ogni immagine del dataset di training.

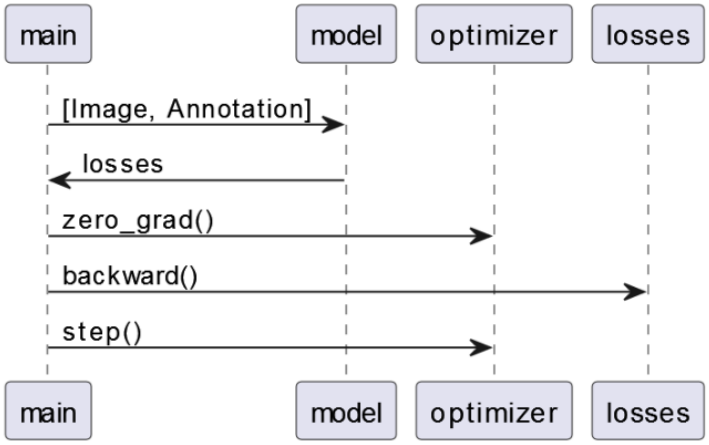

Figura 22: Diagramma delle relazioni tra il modello, l'ottimizzatore e le losses.

Durante questi test si è usato un approccio standard con un numero variabile di epoche in vari test per evitare scenari di overfitting.

```
def get_model_instance_segmentation(num_classes):
      model = torchvision.models.detection.
                  fasterrcnn_resnet50_fpn(
                        pretrained=False,
                        num_classes=num_classes+1,
```

```
            )
        in_features = model.roi_heads.box_predictor.
            cls_score.in_features
        model.roi_heads.box_predictor = FastRCNNPredictor(
            in_features,
            num_classes,
        )
        return model
```

Per l'addestramento è stato utilizzato un ottimizzatore SGD.

```
params = [p for p in model_f.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(
        params,
        lr=0.005,
        momentum=0.9,
        weight_decay=0.0005,
)
len_dataloader = len(data_loader)
for epoch in range(num_epochs):
        model_f.train()
        for imgs, annotations in data_loader:
            imgs = list(img.to(device) for img in imgs)
            annotations = [{k: v.to(device)
                            for k, v in t.items()}
                            for t in annotations]
            loss_dict = model_f(imgs, annotations)
            losses = sum(loss for loss in loss_dict.values())
            optimizer.zero_grad()
            losses.backward()
            optimizer.step()
```

Per il testing, invece, è sufficiente mostrare alla rete ottenuta una immagine per ottenere la classe di appartenenza inferita.

```
img = Image.open(name)
model_f.eval()
```

```
t_img = torchvision.transforms.ToTensor()(img)
        .unsqueeze_(0).to(device)
outputs = model_f(t_img)
```

5.3 MaskRCNN

Questa rete risulta essere di fatto un'estensione di FasterRCNN con l'aggiunta di una testa in grado di calcolare una maschera di segmentazione su una area di interesse in parallelo alle teste esistenti per la categorizzazione e per le bounding box[27]. La parte che calcola le maschere è una piccola rete completamente connessa applicata a ogni area di interesse che predice una maschera pixel per pixel per localizzare dove l'oggetto si trova in quell'area. Questo significa che la maschera sarà un sottoinsieme dell'area rettangolare formata da una bounding box.

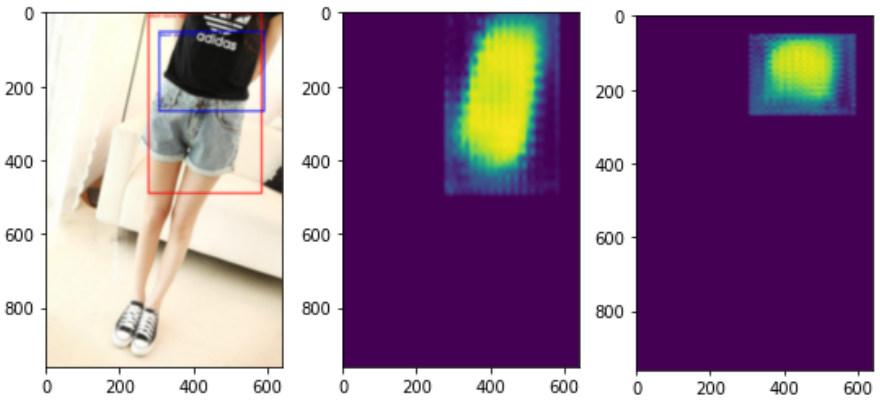

Figura 23: Estrazione delle bounding box e maschere da una immagine.

Come già detto il suo funzionamento riflette ed estende quello di FasterRCNN e quindi risulta essere compatibile su task analoghi. In riferimento alla rete precedente questa rete usa gli stessi due passaggi già menzionati ma nel secondo effettua un calcolo delle maschere binari per ogni area di interesse rilevata. Questo calcolo viene effettuato in parallelo ai precedenti per garantire velocità e prestazioni elevate.

5.3.1 Dettagli implementativi

Anche in questo caso è possibile scaricare dall'hub di Pytorch una implementazione di questa rete e addestrarla come nel caso precedente. Anche il codice risulta analogo e in quanto tale non viene riportato.

```
def get_model_instance_segmentation_m(num_classes):
      model = torchvision.models.detection.maskrcnn_resnet50_fpn(
            pretrained=False,
            num_classes=num_classes+1,
            pretrained_backbone=True,
      )
      return model
```

5.4 KeypointRCNN

Questa rete è sempre presentata nello stesso paper di MaskRCNN e utilizza un approccio simile a un problema leggermente diverso. Un keypoint risulta essere un punto di interesse nell'immagine. Tramite i keypoint è possibile indicare punti salienti in una immagine per calcolare ad esempio i contorni di una immagine o lo scheletro di un oggetto.

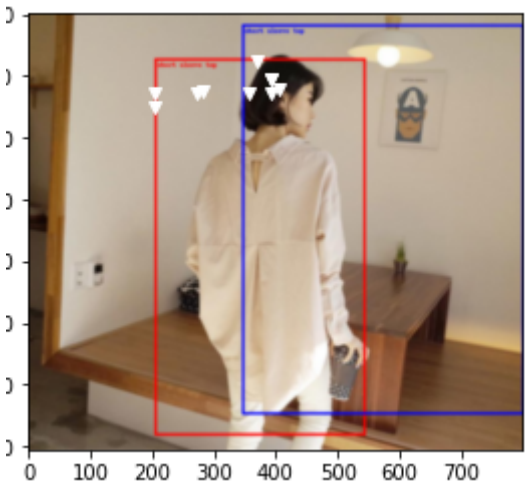

Figura 24: Estrazione dei keypoint da una immagine.

Il dataset DeepFashion2 offre informazioni relative ai keypoint e quindi questa rete può estrarli senza problemi se addestrata a tale scopo.

5.4.1 Dettagli implementativi

Anche in questo caso è possibile scaricare dall'hub di Pytorch una implementazione di questa rete e addestrarla come nel caso precedente. Anche il codice risulta analogo e in quanto tale non viene riportato.

```
model_k = torchvision.models.detection.keypointrcnn_resnet50_fpn(
        pretrained=False,
        num_keypoints=294, num_classes=13+1,
)
```

5.5 YOLOv5

YOLOv5, acronimo per "You Only Look Once"[28, 29] è una rete per l'object detection che permette di individuare bounding box attorno ad un oggetto in una immagine a partire da un dataset su cui addestrare la rete. Al termine dell'addestramento viene individuato un insieme di pesi tramite i quali è possibile estrarre bounding box per ogni immagine presentata.

YOLO è stato ideato attorno al 2016 e, da allora, ne sono state sviluppate diverse versioni; quella presa in considerazione in questa sperimentazione è la quinta versione. Questa metodologia risulta essere molto moderna ed efficace, di fatto questa rete divide l'immagine in una griglia e, per ogni cella, si effettua una inferenza al fine di stabilire se un oggetto cercato è presente o meno nell'area presa in considerazione. Questo metodo, utilizzando anche il valore di confidenza ottenuto dalla ricerca, è in grado di filtrare mediante un threshold le regioni "più interessanti" ovvero quelle dove è più probabile che si trovi un oggetto cercato.

L'utilizzo di YOLO risulta molto simile all'applicazione di una blackbox, infatti basta partire da un dataset, addestrare e attendere una uscita. Ovviamente questo approccio risulta molto riduttivo in termini di complessità di informazioni ottenibili, in quanto non è possibile estrarre maschere. Anche se si considerano aspetti come la configurazione o il miglioramento della rete si è un po' limitati in quanto la rete offerta risulta abbastanza rigida e non modificabile dall'utente direttamente. La rete, usata in questo modo, permette solo di aggiustare i pesi e di scaricare determinati modelli preaddestrati da specializzare poi per il caso corrente tramite addestramento specifico.

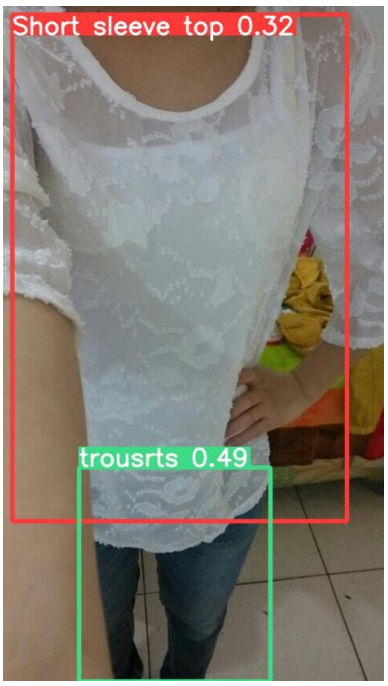

Figura 25: Estrazione delle bounding box con YOLOv5

Per utilizzare YOLOv5 assieme al dataset di riferimento quale DeepFashion2 occorre convertire nel formato utilizzato dalla rete le annotazioni tramite script che si possono facilmente reperire online. In questo caso specifico si può usare uno script che converte da COCO a YOLOv5 che viene fornito dagli stessi creatori della rete. In alternativa è anche possibile aggiungere manualmente le bounding box a delle immagini già esistenti, ovvero creare il proprio dataset, con strumenti online quali RoboFlow.

Addestrare YOLOv5 è alquanto semplice se si utilizzano dei file di utility forniti come prova: dopo avere effettuato il labelling delle immagini, basta invocare il comando:

```
python3 train.py --img 640 \
 --batch 16 \
 --epochs 3 \
 --data coco128.yaml \
 --weights yolov5s.pt \
 --cache
```

e l'addestramento comincia ed esegue per il tempo necessario. Per l'inferenza si potrà, in seguito, utilizzare il comando:

```
python detect.py --weights yolov5s.pt \
```

```
--img 640 \
--conf 0.25 \
--source data/images
```

Le immagini e i dati estratti saranno salvati in una cartella apposita. La potenza di questa rete è, come si può notare, la facilità di utilizzo.

Un lato negativo di questa rete è che estrae solo bounding box e non maschere; dunque essa, per quanto interessante, non risulta essere particolarmente utile per task di segmentazione, mentre invece risulta essere molto efficace e flessibile per task di localizzazione in una immagine.

5.6 FashionPedia

Presentata come rete baseline al dataset FashionPedia, lo scopo di questa rete è quello di segmentare gli abiti in ogni loro sotto parte e nel contempo effettuare la classificazione e un assegnamento di attributi sugli stessi[30]. Questa rete è fornita come baseline per poter confrontarla con altre reti che possono essere realizzate sul dataset ovvero per mostrare come sfruttare al meglio l'ontologia di partenza. Inoltre vengono forniti svariati checkpoint già addestrati per potere analizzare la rete senza la necessità di provvedere in proprio all'addestramento, attività dispendiosa sia in termini di tempo che di risorse.

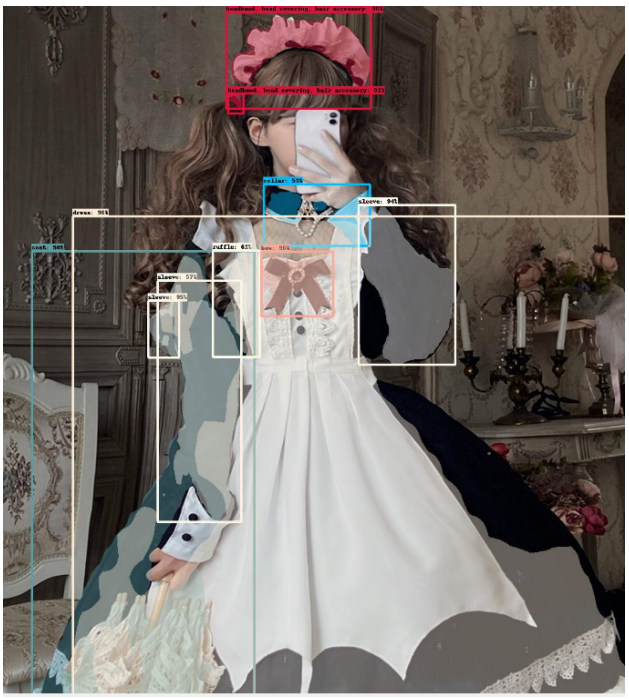

Figura 26: Estrazione di maschere e bounding box da una immagine "difficile" con la rete di FashionPedia.

La rete è basata sulla già menzionata MaskRCNN e, come molte reti di questo tipo, aggiunge ad essa svariate teste in grado di estrarre la classe, gli attributi e anche le bounding box per ogni oggetto presente in ogni immagine. Come backbone usa una rete ResNet50 o una SpineNet dalla quale si estraggono delle feature che vengono poi passate agli strati successivi della rete. Per migliorare la rete durante l'addestramento le immagini in ingresso sono processate con traslazioni orizzontali o trasformazioni di scala per garantire un maggiore grado di apprendimento dalle feature ottenute.

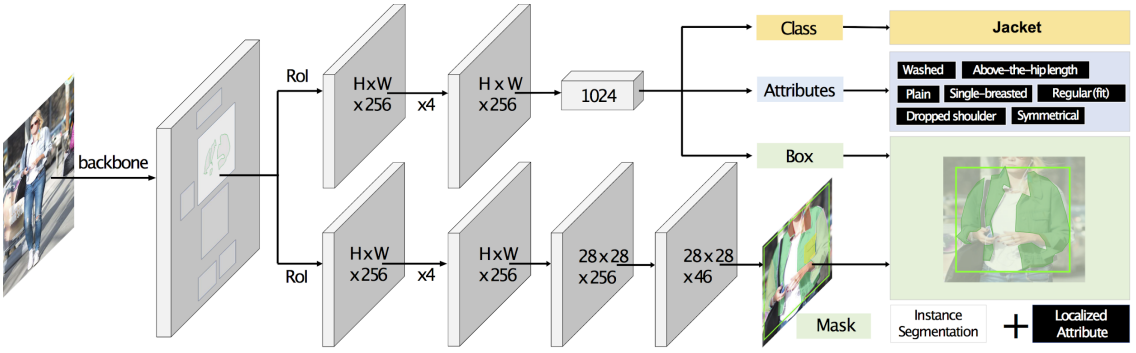

Figura 27: Il diagramma di flusso della rete FashionPedia.

I risultati di questa rete, dunque, rispecchiano i dati presenti nel dataset di riferimento: si è infatti in grado di estrarre maschere con precisione al pixel, la

classe di riferimento del vestito – ovvero a che tipologia appartengono – e gli attributi – di fatto una descrizione più granulare del singolo vestito basata sulle caratteristiche che gli stessi hanno.

5.6.1 Dettagli implementativi

La rete ha principalmente due formati di uscita: o come array di Numpy o come immagine con le bounding box e le aree disegnate sopra. Il secondo metodo è principalmente utilizzabile per effettuare analisi e stabilire le prestazioni della rete. L'uscita numerica, invece, è il formato più appetibile specie se si vuole processare l'uscita in stadi successivi della rete. Il formato di uscita della rete è il seguente:

```
[{
        'image_file': <file_name>,
        'boxes': <array di boxes>,
        'classes': <array di classi>,
        'scores': <array di confidenze>,
        'attributes': <array di attributi>,
        'masks': <array di maschere codificate in formato coco>,
}]`
```

Esempi

Per testare la rete sono state utilizzate immagini non provenienti da dataset noti e contenenti capi anche difficili da segmentare per simulare al meglio casi reali.

Figura 28: Esempi di estrazioni di maschere con FashionPedia da immagini mai viste dalla rete.

Si può notare come sia le segmentazioni che le bounding box risultano essere molto accurate, fallendo solo in casi in cui:

- Nell'immagine siano presenti categorie di indumenti sui quali la rete non è stata addestrata (e.g. la parte superiore delle calze può essere incorrettamente catalogata come "sleeve", il concetto è approssimativamente lo stesso visivamente ma la catalogazione è ontologicamente sbagliata. In tali casi è molto probabile che la rete fornisca una segmentazione (maschera) corretta ma una catalogazione sbagliata).
- Gli indumenti sono estremamente strani (e.g. la maglietta con le facce del bambino) in tal caso la rete fallisce completamente in quanto non è addestrata su indumenti con tali feature (e forse è meglio così).
- In alcuni casi anche sfondi complessi o posizioni del corpo strane possono "ingannare" la rete riducendo le maschere trovate.

Gli esempi sono stati ricavati usando la rete addestrata con backbone ResNet-50 FPN [58] con meno parametri possibili e con l'IoU più basso appunto per scoprire possibili casi limite d'utilizzo.

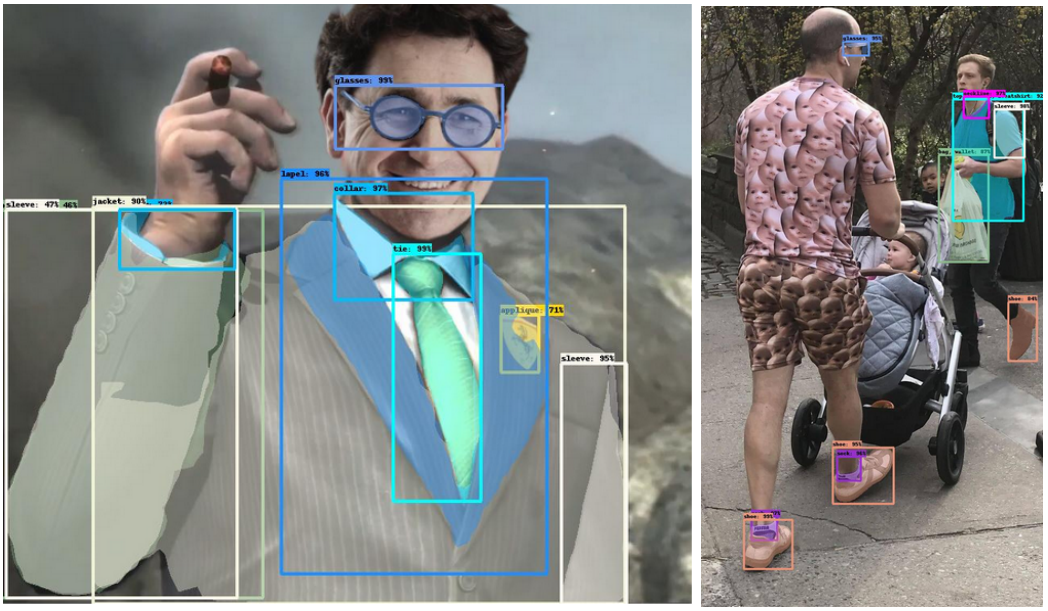

Figura 29: Altri esempi di estrazioni "difficili" come la maglia con le facce.

5.6.2 Problemi della rete

Questa rete presenta, purtroppo, problemi di configurazione e per farla funzionare in maniera appropriata occorre modificare pesantemente il codice in molteplici punti. Questo potrebbe essere dovuto sia al fatto che non si dispone della configurazione originaria o al fatto che è pensata per funzionare sulla piattaforma Cloud TPU di Google. Anche l'addestramento di tale rete risulta alquanto complesso in quanto richiede una quantità di memoria su GPU molto elevata.

5.6.3 Prestazioni

È stato in seguito eseguito un semplice test per calcolare le prestazioni della rete su un dataset di 100 immagini tratte da DeepFashion2. Per valutare i risultati di questo test è però necessario tenere conto di alcuni caveat:

- Il dataset di DeepFashion2 e quello di FashionPedia, seppur simili, sono differenti per label e bounding box. Questo, però, non è un fattore particolarmente rilevante in questo momento in quanto seppure le diverse bounding box sono state inserite a mano da operatori umani e i metodi utilizzati saranno genericamente equivalenti. Si può dunque immaginare che operatori umani abbiano indicato lo stesso tipo di vestiti allo stesso modo.
- Risulta complicato effettuare una Intersection of Unions (IoU) in quanto le maschere sono diverse ed esiste un ovvio problema di corrispondenza tra le maschere di riferimento e quelle in uscita.
- L'IoU deve essere inteso solo come una misura della precisione della rete e non col suo significato classico in quanto la rete non è stata addestrata su questa bounding box e quindi risulta scorretto attendere un risultato alto da tale analisi.
- È d'obbligo ricordare che FashionPedia segmenta cose come "sleeve", "collar", "rivet" mentre DeepFashion2 avrà tutto dentro un'unica box o maschera col nome dell'indumento generico. Quindi solo le categorie principali saranno considerate mentre tutte quelle secondarie scartate ai fini di questo test in quanto non presenti in DeepFashion2.

Il test sarà svolto in tre passaggi: per prima cosa si leggerà l'immagine dal dataset DeepFashion2 e verrà segmentata con la rete di FashionPedia. Per ogni maschera trovata si selezionerà quella che ha l'IoU più alta rispetto a ogni box dell'immagine

originale presa dalle annotazioni di DeepFashion2, memorizzando l'IoU, la confidenza e le due labels. A questo punto, per ogni coppia, si valuta se le due classi corrispondono e, se sì, si definisce tale ricerca un successo. Questo ultimo passaggio viene effettuato manualmente vista la difficoltà di associare le label tra loro in maniera logica visto che DeepFashion2 ha solo una decina di labels rispetto alle oltre 40 di FashionPedia.

Filename	Confidence	IoU	Found	Real
000001.jpg:	0,9806	0,1106	sleeve	short sleeve top
000001.jpg:	0,8696	0,3634	top, t-shirt, sweatshirt	short sleeve top
000001.jpg:	0,2422	0,4121	shorts	short sleeve top
000001.jpg:	0,1471	0,00047	pants	short sleeve top
000001.jpg:	0,1212	0,1479	hood	short sleeve top
000001.jpg:	0,1192	0,412	skirt	short sleeve top
000001.jpg:	0,1151	0,06468	hat	short sleeve top
000001.jpg:	0,1115	0,4108	shirt, blouse	short sleeve top
000001.jpg:	0,1098	0,4191	dress	short sleeve top

Figura 30: Esempio del file prodotto per una immagine, confidenza per ogni bounding box trovata, massimo IoU trovato e confronto tra label trovata e quella della bounding box con cui ha massimo IoU. Se appartengono alla stessa "categoria logica", allora sono un match.

Occorre tuttavia considerare che questo metodo ci espone a casi sfortunati nei quali, nonostante la segmentazione sia stata eseguita correttamente, il massimo IoU non era con la box corretta ma con altre. Questi casi non sono stati ignorati per garantire una imparzialità del test ma si è considerato che in un caso di funzionamento concreto siano da tenere in conto. Non è comunque difficile immaginare un sistema in grado di tenere conto di queste occorrenze.

Definendo dunque l'IoU come il rapporto tra l'intersezione delle bounding box e la loro unione, è possibile calcolare la media dell'IoU di ogni immagine ottenendo un valore di: 0.268.

$$J(A,\ B)\ = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$$

Questo valore può essere confrontato con quello dichiarato dalla rete di 0.434 e nonostante risulti minore può essere comunque considerato un risultato favorevole specie se si considerano le condizioni sfavorevoli alle quali la rete è sottoposta in questo test. Per un IoU sulle maschere e se un dataset è compatibile con FashionPedia si rimanda alla documentazione ufficiale dove tali dati sono riportati nella validazione di ogni modello preaddestrato. Occorre altresì notare che in 11

casi la rete non ha trovato segmentazioni corrette o comunque il sistema non le ha associate in modo corretto.

	Avgerage	Weighted average	Average of >.7 conf	Median
Confidence	0,736285464	0,733378874	0,951935593	0,8641
IoU	0,268084124	N/A	0,26552661	0,2756

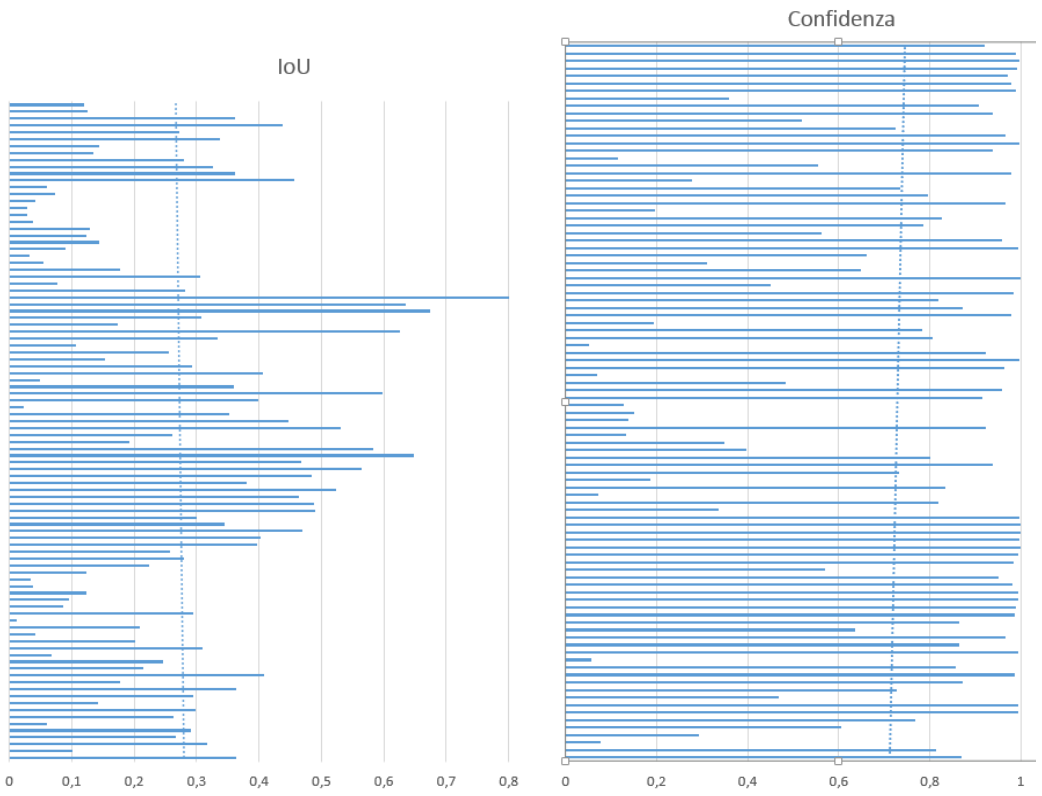

Figura 31: Rapporto tra IOU e Confidenza per alcune immagini di prova con FashionPedia.

In una analisi delle prestazioni risulta d'obbligo notare come questo sistema mostri la granularità della rete nell'effettuare le segmentazioni, restituendo risultati riguardanti la segmentazione che permettono l'estrazione di dettagli quali le singole borchie nei pantaloni o la presenza di cerniere, cappucci, occhiali e altri accessori. Ovviamente questa granularità è un riflesso della granularità stessa del dataset.

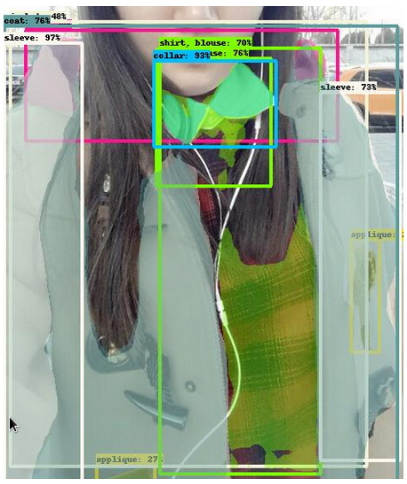

Figura 32: Estrazione con FashionPedia.

5.7 Match RCNN

Anche questa rete[31] è una implementazione baseline per un dataset, ma questa volta per DeepFashion2. L'obiettivo di questa rete è, come negli altri casi trattati, la segmentazione degli abiti da una immagine e, in aggiunta a ciò, il riconoscimento del capo tra altri presenti nel dataset al fine di fare street-to-shop matching. Ci saranno dunque due teste principali, una addetta all'individuazione dell'abito nell'immagine e alla sua segmentazione e un'altra che si occupa invece di calcolare delle feature che possono poi essere comparate con altre feature presenti al fine di trovare quelle che hanno distanza minima e quindi di potere indicare una corrispondenza tra due abiti.

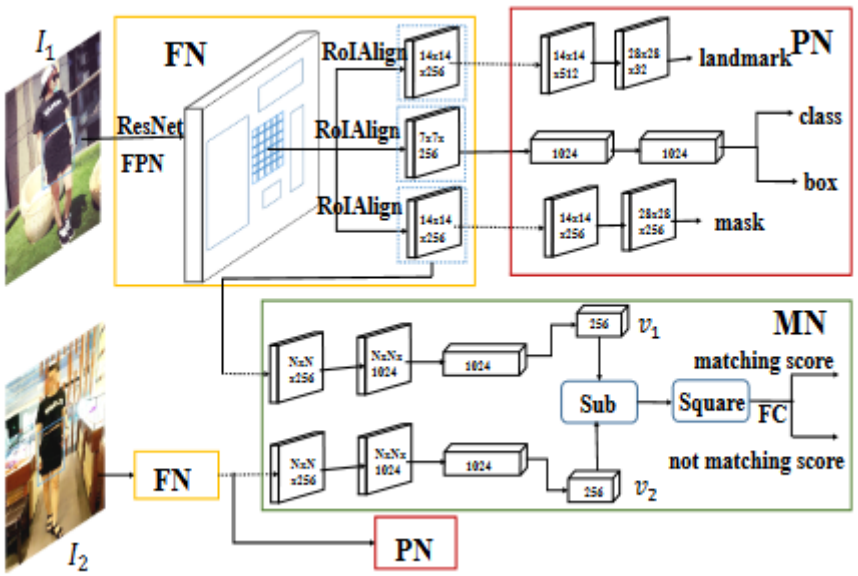

Figura 33: Diagramma della struttura della rete Match RCNN.

Match RCNN è principalmente addestrabile sul dataset DeepFashion2, ma, siccome accetta genericamente annotazioni in formato COCO, è possibile convertire altri dataset per utilizzarli con questa rete senza troppa difficoltà. Operativamente parlando accetta due immagini di ingresso, una proveniente da un sito di e-commerce e l'altra da utenti, passa la prima attraverso uno strato ResNet e svariati strati di estrazione delle feature, in seguito usa una perception network per l'estrazione di landmark, delle labels, delle maschere e delle bounding box per ogni abito rilevato nell'immagine. In seguito confronta anche le feature in uscita dalla prima rete con quelle in uscita a una analoga per determinare il matching score, ovvero un indice di somiglianza, tra le due immagini.

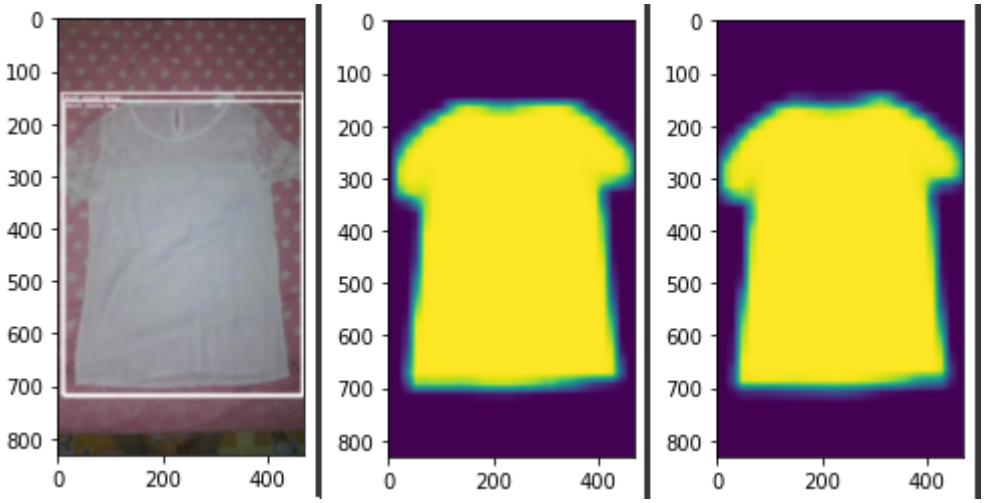

Figura 34: Estrazione della rete Match RCNN relativa alla maschere.

5.7.1 Dettagli implementativi

Per testare questa rete è sufficiente addestrarla sul dataset fornito e poi sperimentare coi risultati ottenuti. Per semplificare questo processo si può fare riferimento ad un'altra rete che implementa gli stessi concetti: SEAM Match RCNN, la quale è essenzialmente una rete identica a Match RCNN e una sua estensione per potere lavorare con filmati presi dal web. È infatti l'obiettivo di questa rete quello di riconoscere abiti in filmati presi dai social e fare il processo di street-to-shop matching. In più, oltre al codice sorgente, è anche fornita una implementazione di una rete Match RCNN già addestrata che può essere usata come primo stage per il resto della rete SEAM. In questo caso la rete sarà usata per analizzare il comportamento di Match RCNN stessa, in quanto implementazione funzionante di tale rete. Siccome il dataset originario era molto vasto si è deciso di utilizzare un sottoinsieme per i seguenti risultati avente dimensioni significativamente ridotte, ma comunque aventi rilevanza statistica.

I risultati saranno suddivisi in due parti, quella di estrazione di maschere e quella di feature matching.

Per sperimentare con la rete è bastato clonare la repository SEAM Match RCNN da GitHub, istanziare il modello:

```
model = SEAM.models.matchrcnn
       .matchrcnn_resnet50_fpn(num_classes=14)
PATH = "/df2matchrcnn"
ckpt = torch.load(PATH, map_location=device)["model_state_dict"]
ckpt = {k if k.find("model.") > 0 else k[7:]: v
             for k, v in ckpt.items()
       }
model.load_state_dict(ckpt)
```

In seguito basta caricare il dataset in formato COCO utilizzando una implementazione custom della classe Dataset di Pytorch e addestrare col solito ciclo di addestramento come già fatto in precedenza.

5.7.1.1 Mask extraction

La rete dimostra la sua efficacia specialmente sulle immagini del dataset di provenienza, per i test di seguito sono state prese in considerazione immagini di solito usate per la validazione e, come del resto atteso, il tasso di confidenza e di accuratezza sia delle maschere che delle bounding box è risultato alto. Il fatto che provengano dal dataset di validazione garantisce che la rete non abbia mai visto tali immagini fornendo risultati più parziali.

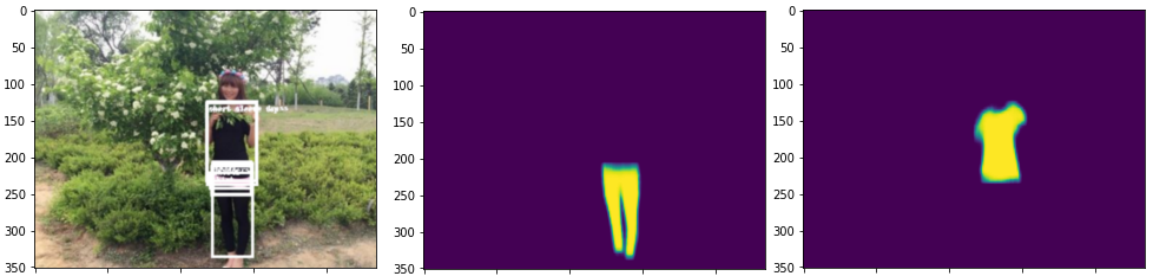

Figura 35: Bounding Box estratte e maschere da una immagine di text.

Operativamente parlando la rete ritorna un dizionario Python di valori ognuno facente riferimento a un campo diverso di applicazione della rete, nello specifico:

- ➔ boxes: array di coordinate rappresentanti la posizione nell'immagine della bounding box trovate
- ➔ labels: array delle label corrispondenti
- ➔ scores: la confidenza di ogni box trovata
- ➔ masks: le maschere trovate dalla rete
- ➔ match_features: 256 feature in uscita dalla matching network
- ➔ w: i pesi della matching network
- ➔ b: il bias della matching network

5.7.1.2 Feature matching

Come accennato sopra la rete è in grado di riconoscere non solo dove gli indumenti si trovino nell'immagine ma anche di associarli a un'altra immagine presa da siti di e-commerce. Il dataset DeepFashion2 infatti ha degli appositi campi che indicano se una specifica immagine è stata recuperata da un social media ("in the wild") o se invece proviene da un sito di e-commerce. Questi due tipi di immagine vengono dette user e shop. L'idea della rete è di dare in uscita, come

menzionato, una serie di feature, dei pesi e un bias, tramite questi è possibile calcolare la distanza euclidea pesata per trovare l'immagine shop correlata.

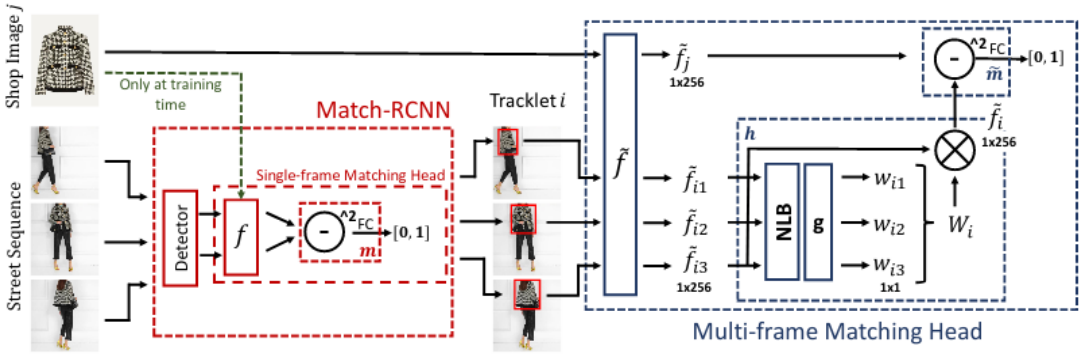

Figura 36: Diagramma della rete SEAM Match RCNN.

Il paper di riferimento indica che l'accuratezza di questo metodo garantisce con una possibilità dell'ottanta percento (80%) che l'immagine shop cercata sia tra i primi 5 risultati.

Per effettuare questo processo l'implementazione baseline definisce una funzione distanza custom, per comodità lo stesso è stato fatto durante la sperimentazione, creando una funzione compatibile per l'uso.

```
def distance(feature, input, w, b):
      sq = (feature[np.newaxis] - input[np.newaxis]) ** 2
      raw_sc = sq.detach().cpu().numpy()
            @ w.detach().cpu().numpy()
                  .transpose().astype(np.float16)
            + b.detach().cpu().numpy()
                  .astype(np.float16)
      cls_sc = np.exp(raw_sc)
            / np.exp(raw_sc).sum(1)[:, np.newaxis]
      return cls_sc[:, 1]
```

Operativamente parlando il processo di matching consiste nell'estrarre tutte le feature in fase di addestramento salvandole in memoria per poi potere confrontarle con l'immagine su cui vogliamo effettuare inferenza. Si noti che questo metodo consente l'aggiunta di nuovi indumenti semplicemente aggiungendo nuove immagini (e feature da confrontare) senza necessità di riaddestrare la rete. Inoltre le

feature valgono per ogni singola bounding box e non per l'immagine intera, questo significa che l'immagine I_0, avente come bounding box $B_0 = [b_{0,1}, b_{0,2}, \cdots, b_{0,n}]$ avrà come feature $F_0 = [f_{0,1}, f_{0,2}, \cdots, f_{0,n}]$. Ognuna delle $f_{i,j}$ dovrà dunque essere confrontata con ognuna delle f_{d_k, d_l} di ogni immagine d del dataset Shop. (Con d_k ogni immagine di indice k del dataset shop e d_l ogni bounding box di d_k con indice l).

Il processo consiste, dunque, nelle seguenti operazioni

- Estrarre le feature $f_{i,j}$ dalla nostra immagine campione di cui vogliamo conoscere quali indumenti siano contenuti.
- Estrarre le feature f_{d_k, d_l} da tutte le immagini del dataset (o averle in cache).
- Calcolare la distanza euclidea pesata tra le feature dell'immagine campione e quelle di ogni immagine del dataset.
- Estrarre le prime 5 per ogni match e confrontare se l'immagine che si cercava è presente tra quelle.

Avendo così ottenuto una lista di matches coerenti possiamo dunque estrarre il primo (o i primi se si vuole essere precisi) per valutare il funzionamento della rete.

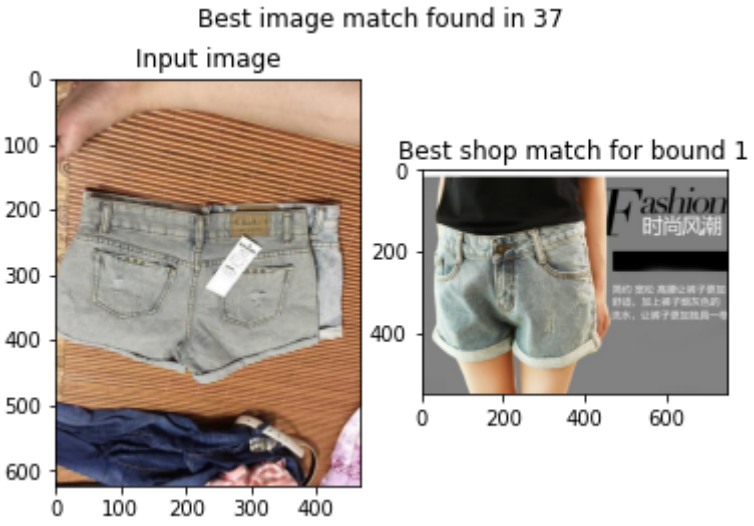

Figura 37: Esempio di matching della rete tra immagine "street" (sx) e "shop" (dx).

5.7.2 Prestazioni

Applicando il precedente algoritmo a diverse immagini user del dataset è possibile creare una tabella mostrante le prestazioni della rete che permette di notare come l'effettiva immagine shop corrispondente sia tra le prime 5 come dichiarato nel paper SEAM, se non addirittura la prima. E' opportuno sottolineare che trattandosi del dataset sul quale la rete è stata addestrata è plausibile che i risultati osservati risultino più alti rispetto ad immagini "in the wild". Va inoltre considerato anche che per potere esserci un match che abbia un significato logico occorre che sia presente del dataset shop un'immagine dello stesso indumento che si vuole riconoscere.

È importante notare che nel dataset vi sono più immagini shop che possono essere associate a una singola immagine user quindi l'uscita della rete potrà dare anche più di una immagine corretta come match in uscita.

Inoltre tali match sono relativi alle immagini, quindi se si vogliono informazioni riguardanti label e affini occorre mantenere in memoria informazioni riguardanti le immagini shop e, sfruttando le box di tali immagini, se aventi labels, tramite l'IoU stabilire l'esatta posizione degli indumenti in questione nell'immagine in caso di più elementi o matches nell'immagine al fine di ottenere una migliore accuratezza del sistema. Se questo non viene effettuato, come del resto fatto per semplicità in questa trattazione, l'accuratezza della rete diminuisce ma rimane comunque accettabile come dimostrato sopra.

Image	Matches	Confidenza	Corretto
5	28	0.99978	No
"	28	0.99978	No
"	8	0.99977	Sì
"	8	0.99976	Sì
"	28	0.99975	No
21	26	0.99993	Sì
"	24	0.99992	Sì
"	25	0.99989	Sì
"	32	0.99986	Sì
"	23	0.99985	Sì
33	37	0.99989	Sì
"	35	0.99983	Sì
"	36	0.99979	Sì
"	38	0.99940	Sì
"	34	0.99927	Sì
52	61	0.99993	Sì
"	61	0.99993	Sì
"	62	0.99992	Sì
"	55	0.99992	Sì
"	55	0.99991	Sì
72	69	0.99994	No
"	69	0.99994	No
"	69	0.99994	No
"	69	0.99992	No
"	77	0.99973	Sì
91	93	0.99993	Sì
"	93	0.99993	Sì
"	97	0.99993	Sì
"	97	0.99992	Sì
"	94	0.99992	Sì

Tabella 1: Tasso di confidenza per il matching di ogni immagine.

Dall'analisi precedente possiamo ottenere un tasso di successo del 100% se consideriamo l'affermazione del paper di trovare un risultato corretto tra i primi 5 output e un tasso del 76%, con soli 7 fallimenti localizzati in due immagini "difficili", se si considera solo il risultato al primo posto. Per contro quei fallimenti sono presenti nelle due uniche immagini nelle quali l'uscita della rete non ha prodotto come primo risultato il match corretto, portando dunque sotto quel punto di vista una precisione del 66% che il primo risultato della rete sia quello corretto. Occorre comunque ricordare che, nonostante questi test siano stati condotti con sole 6 immagini d'ingresso ovvero non un campione rappresentativo completo, risultano ciononostante coerenti con quanto dichiarato nei vari paper.

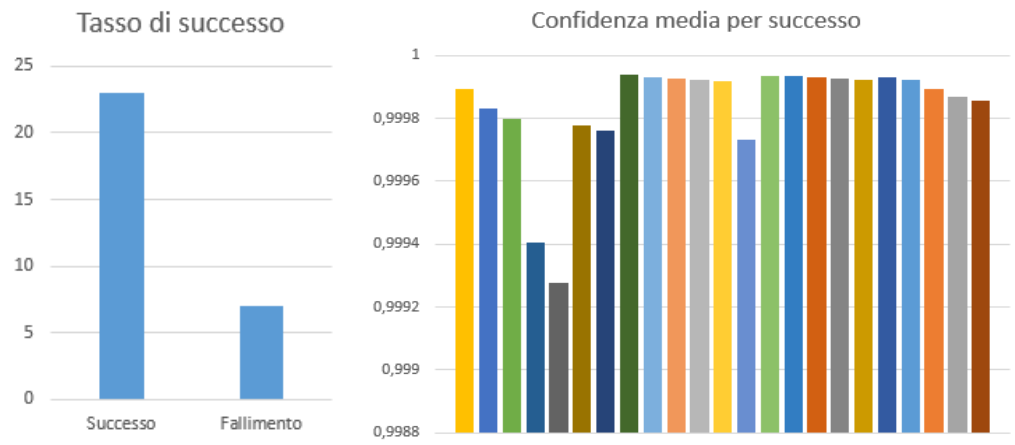

Figura 39: Istogrammi rappresentanti il tasso di confidenza per il matching di ogni immagine.

5.7.3 Conclusione

La rete in analisi si dimostra una alternativa accettabile alla rete FashionPedia per la segmentazioni dei vestiti ma difetta di una segmentazione e di una ontologia tanto accurata o comunque comparabile nonostante siano entrambe basate su una MaskRCNN. Si noti che questo è anche una conseguenza del fatto che il dataset DeepFashion2 su cui questa rete è addestrata non presenta tali ontologie, dunque scambiando i due dataset si può teoricamente fare sì che questa rete riconosca anche tali sotto parti.

La rete inoltre sembra fornire dati interessanti riguardo all'estrazione della tipologia di vestito mediante il match user e shop anche se l'implementazione della stessa risulta essere un po' "rozza" in quanto occorre memorizzare tutte le feature e conseguentemente compararle all'immagine che si sta analizzando; questo però permette di inserire nuove immagini dinamicamente senza bisogno di riaddestrare la rete. Non è in ogni caso così impensabile potere addestrare un'ulteriore rete per effettuare questo feature matching, evitando così di dovere memorizzare tutti i dati relativi a ogni immagine shop, di contro tale approccio significherebbe il riaddestramento della rete per ogni immagine shop aggiunta in seguito.

Si riscontrano anche problemi riguardo la creazione di un dataset omnicomprensivo di ogni indumento shop e l'indicizzazione efficiente per il calcolo delle distanze a runtime, nonché il bisogno di ottimizzare tale calcolo che, comunque, può essere parallelizzato su un numero n di processori con n il numero di immagini shop.

Capitolo 6

Segmentazione di abiti con MaskRCNN

Valutate attentamente le reti a disposizione, si è scelto di approfondire la MaskRCNN come soluzione possibile per la segmentazione dei vestiti. Si è scelta tale rete perché è la più libera da ogni sovrastruttura aggiuntiva e, essendo disponibile una implementazione già testata e facile da usare in Pytorch, quella con i maggiori benefici. Per le seguenti sperimentazioni dunque è stato scelto di basarsi sul dataset di FashionPedia e di porsi come obiettivo la capacità di potere addestrare la rete su tale dataset con una o più GPU. Questa serie di test integra ed estende quanto già detto nel precedente capitolo riguardante questo tipo di rete e ha come obiettivo la creazione di una pipeline completa per il suo addestramento e utilizzo in applicazioni reali.

6.1 Caricamento dei dati

Siccome il dataset scelto per questa rete è FashionPedia la soluzione utilizzata per il caricamento dei dati è simile a quella già descritta in precedenza. Si tratta dunque di estendere la classe Dataset presente in Pytorch. Questa operazione risulta alquanto facile grazie alle librerie messe a disposizione da COCO.

La classe si occuperà di caricare le immagini ogni volta che sarà richiesto. Questo permette di tenere meno immagini aperte in memoria per evitare di consumare tutta la memoria disponibile. La classe restituirà in seguito a una specifica chiamata una coppia di tensori, uno rappresentante l'immagine e l'altro il dizionario con le annotazioni richieste che potranno essere passate direttamente alla rete.

Un'altra operazione che il Dataset dovrà svolgere è quella di ridimensionare le immagini, le maschere e le bounding box di ingresso. Questo deve essere fatto perché la ResNet utilizzata preferisce input di dimensione 224x224, quindi le immagini e le annotazioni devono essere scalate di conseguenza. Per scalarle si usa una semplice proporzione per le box mentre per l'immagine e le maschere delle trasformazioni già predefinite di Pytorch.

6.2 Struttura della rete

La rete è sempre quella discussa in precedenza presa dall'hub di Pytorch, ma, siccome si vuole rendere la rete parallela su molteplici GPU, occorre utilizzare un modulo apposta fornito dalle librerie.

6.2.1 L'addestramento parallelo

Parallelizzare ha lo scopo, in questo caso, di sfruttare più GPU in contemporanea, questo significa caricare molte più immagini simultaneamente, utilizzare batch più grandi e ridurre i tempi di addestramento[33]. Pytorch mette a disposizione due principali astrazioni per la parallelizzazione: DataParallel[32] e DistributedDataParallel[34]. Il primo è pensato per parallelizzare reti su una macchina sola mentre il secondo è per parallelizzare su sistemi distribuiti.

Il funzionamento di DataParallel è alquanto semplice: essendo un'estensione della classe nn.Module può essere usata per incapsulare il modello da parallelizzare ed essere quindi utilizzata come se fosse una rete normale. La parallelizzazione avverrà quindi automaticamente in seguito operata dall'implementazione stessa.

```
device = torch.device("cuda:0")
model.to(device)
input = input.to(device)
model = nn.DataParallel(model)
```

In seguito a svariati test si è potuto verificare che come soluzione sia molto legacy e le guide ufficiali di Pytorch consigliano di usare DistributedDataParallel impostato per il funzionamento su un nodo solo. Un grande vantaggio di DistributedDataParellel è il fatto che utilizza un modello di parallelismo migliore rispetto a DataParallel: ciò è dovuto alla struttura stessa del linguaggio Python. DataParallel, infatti, si appoggia al global interpreter lock (GIL) presente in Python. GIL[36] è utilizzato per sincronizzare i thread in esecuzioni in modo che uno solo thread nativo, ovvero uno del sistema operativo sottostante, possa essere eseguito in ogni momento dato. Questo sistema viene usato molto spesso in linguaggi di programmazione interpretati e, seppur molto potente, ha come effetto negativo l'esecuzione di un solo thread per volta, cosa che può essere un beneficio in

situazioni dove si lavora con semafori e altre strutture dati bloccanti, ma se si cerca un puro parallelismo risulta inferiore. Questo significa che DistributedDataParallel, che non usa questo sistema in quanto si appoggia su thread nativi differenti, o, nel caso di sistemi distribuiti, su computer differenti, sarà più veloce e prestante in ogni situazione. Anche DistributedDataParallel risulta facile da implementare ma necessita di alcuni accorgimenti: intanto occorre impostarlo, almeno in questo caso, per il funzionamento su un nodo unico e poi occorre usare un DistributedSampler nel DataLoader al fine di distribuire correttamente le immagini tra i vari nodi[35].

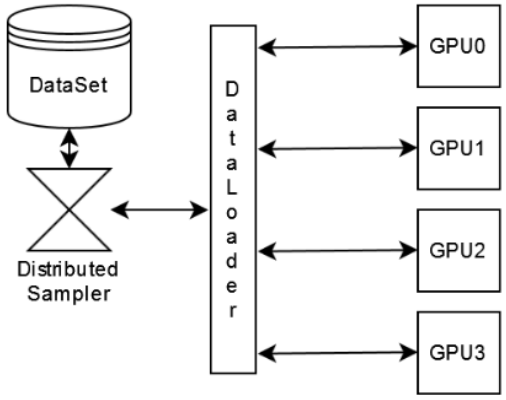

Figura 40: Relazione tra il DataSet e il DataLoader in presenza di più GPU.

Un DistributedSampler è una struttura che si occupa di gestire il caricamento dei dati da un unico dataset e di conseguenza smistare le singole immagini in ogni GPU coinvolta. Come detto questo sistema consente di usare batch più grandi ad ogni epoca e quindi rendere il processo di addestramento più veloce in generale. A livello effettivo usare quattro GPU consente di quadruplicare il batchsize. Occorre, però, notare che non è equivale a ridurre a un quarto il tempo di addestramento in generale visto che nel progetto la validazione avviene in maniera single GPU e ci sono delle perdite dovute alle barriere di sincronizzazione al termine di ogni epoca.

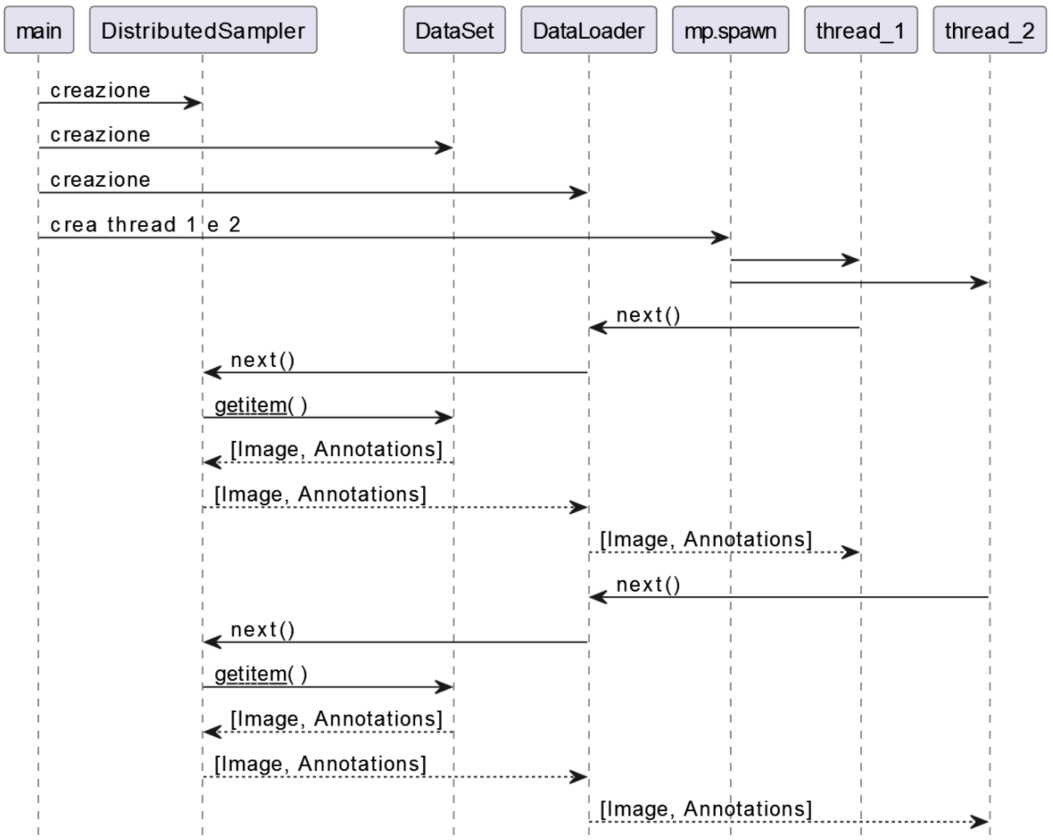

Figura 41: Diagramma che illustra l'addestramento multi GPU-

Come già accennato per una progettazione parallela avanzata ed efficiente occorre superare le problematiche stesse del linguaggio Python e del GIL. Per effettuare questa operazione Pytorch offre a disposizione una libreria di multiprocessing[37] che di fatto risulta essere un impacchettamento in codice di alto livello delle librerie offerte direttamente da Python[38]. Questa libreria offre API di concorrenza sia locale che remota libere di eseguire dal GIL. Questo permette di implementare vero parallelismo sui dati anche su più macchine. Ovviamente l'implementazione di Pytorch di questa libreria è completamente compatibile con l'originale, ma è altamente specializzata per l'elaborazione di tensori.

Essenzialmente tale libreria è visibile solo nella riga

```
mp.spawn(
        utils.main_worker,
        nprocs=settings.world_size,
        join=True,
        args=(model, settings, tr, va)
)
```

dove invoca il worker incaricato di effettuare il training. Per completezza:

```
main_worker(
        gpu: int,
        model: nn.Module,
        settings: Settings,
        tr: dataset.DataSet,
        va: Optional[dataset.DataSet]=None
) -> nn.Module
```

Inoltre verranno anche utilizzati concetti come semafori e barriere per sincronizzare i vari thread durante la fase di validazione, che sarà eseguita in modalità non distribuita per ragioni implementative. Tali concetti sono forniti da Pytorch stesso.

6.2.2 Il calcolo delle loss

Osservando attentamente il modello utilizzato in questione, si può notare che esso generi le funzioni di perdita solo durante la fase di addestramento e non quella di valutazione. Questo è un fattore negativo e rende di fatto impossibile calcolare le perdite durante la fase di validazione. Per ovviare a questo problema si utilizza semplicemente una funzione di loss custom che richiama le funzioni utilizzate dalla rete per calcolare le perdite. In questo modo è possibile calcolare in modo efficace ed efficiente le perdite durante la fase di validazione.

6.2.3 Dati tecnici, iperparametri e addestramento

Sono state effettuate diverse run, sia parallele che non; in tutti i casi durante la sperimentazione si è deciso di usare i seguenti parametri: 10 epoche, ottimizzatore di tipo SGD con learning rate 0.005, momento 0.9 e decadimento dei pesi 0.0005, uno scheduler di tipo StepLR con step size 5 e gamma 0.3, come dimensione dei batch 12 (si ricorda che in caso di multi GPU il batch size rimane per GPU quindi batch size 12 e 4 GPU significa un batch size complessivo di 48 immagini). Per completezza si ricorda che il dataset ha circa 45000 immagini.

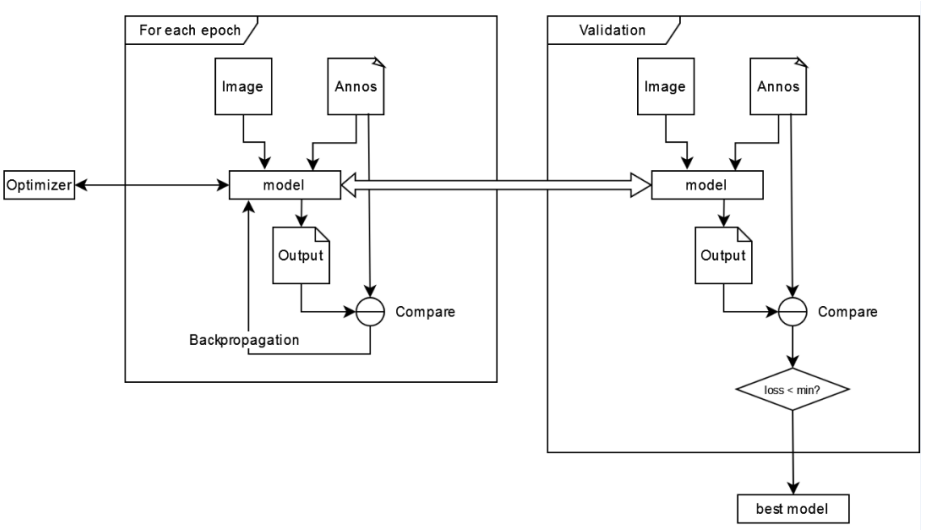

Figura 42: Breve diagramma della pipeline di addestramento di una rete neurale.

Durante il corso di questi esperimenti non si è effettuata una ricerca troppo estensiva degli iperparametri per due principali motivi: il primo è senz'altro il fatto che la ricerca degli iperparametri ottimali è decisamente una questione di sperimentazioni e verifiche spesso molto correlate al dataset stesso; si è quindi deciso che modificare gli iperparametri avrebbe potuto solamente portare influenze al progetto e per l'effetto rendere più complicato il processo di capire se la rete stava migliorando in seguito a cambiamenti tecnici o meno. Una seconda motivazione è stata quella di volere evitare discrepanze tra l'implementazione mono GPU e quella distribuita al fine di comprendere se i miglioramenti fossero dovuti alla distribuzione del modello o a altri fattori.

Una ricerca di iperparametri ottimali è comunque consigliata come fase finale di questa parte del progetto o come un futuro miglioramento dello stesso.

6.3 Risultati ottenuti

L'addestramento della rete è stato eseguito sia in modo parallelo che in modo single GPU, in modalità parallela il tempo di esecuzione è stato di 3 ore e 40 minuti per eseguire 10 epoche, mentre sequenzialmente ci sono volute 7 ore per raggiungere lo stesso risultato. Per l'addestramento sono state usate quattro RTX 3090. Occorre ricordare che la fase di validazione è seguita in modalità single GPU in tutti i casi.

Di seguito è rappresentata la loss della fase di validazione per dimostrare la buona riuscita dell'addestramento:

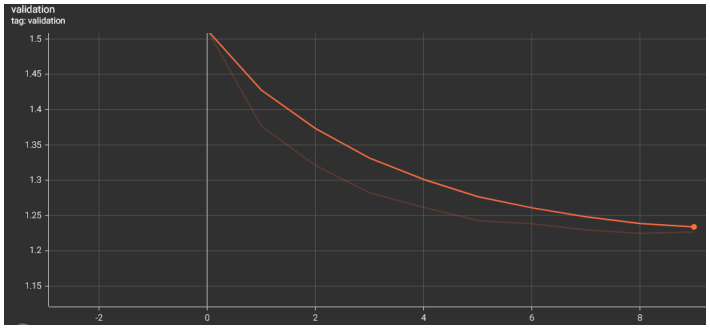

Figura 43: Il diagramma illustra la discesa della loss durante l'addestramento.

Riguardo i risultati sulla segmentazione e sull'individuazione di bounding box si può fare riferimento ai risultati sotto riportati. Come si può notare la segmentazione risulta migliore utilizzando l'approccio distribuito.

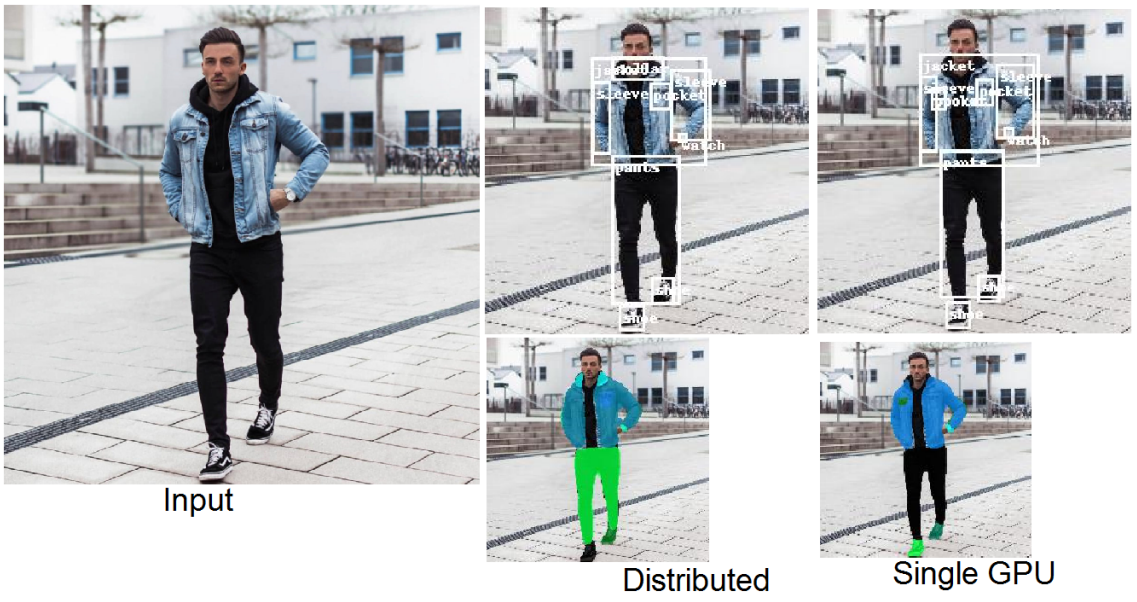

Figura 44: Estrazione di output sia nel caso distribuito che single GPU.

In questa estrazione si può notare come la rete sia stata in grado non solo di segmentare e riconoscere pantaloni e giacche, ma anche sotto parti come il colletto, le tasche e l'orologio indossato. Segue che con un dataset con una segmentazione più granulare tutte le parti principali degli abiti in questione potrebbero essere riconosciute e segmentate.

Un altro esempio è stato ottenuto usando una immagine più “difficile”. Questa immagine, usata anche in altre parti di questo elaborato come benchmark è particolarmente difficile da segmentare visto l’alto numero di accessori, la mancanza di un indossatore e lo stile particolare.

Figura 45: Estrazione di output sia nel caso distribuito che single GPU, questa immagine era molto più difficile delle altre in quanto un capo di abbigliamento mai visto prima.

Come si può notare questa immagine non è stata segmentata in modo perfetto, infatti il caso single GPU risulta avere qualche problema. Un esempio lampante di ciò è la scarpa individuata in basso a sinistra. In generale occorre comunque ricordare che questa è una immagine “difficile”. Una estensione del dataset di addestramento includente immagini più variegate o un tempo di addestramento più lungo potrebbero rendere la rete più precisa anche in casi particolari come questo.

6.4 Modularità

Per comodità è stato scelto di organizzare il progetto nel modo più modulare possibile e parte dei componenti sono stati riutilizzati o adattati da prove precedenti[42].

Il progetto risulta articolato in diversi file per garantire compartimentazione delle funzioni. Degni di nota sono: dataset.py, dove risiede la classe DataSet che si

occupa del caricamento del dataset in formato COCO, network.py, che include l'implementazione del DistributedDataParallel e il modello della MaskRCNN, validation.py, con al suo interno le funzioni per calcolare la fase di validazione e le perdite e utils.py che racchiude l'intera fase di training. Esposte all'utente rimangono inference.py e main.py che consentono di effettuare l'inferenza su una immagine e l'addestramento della rete.

Per facilitare il caricamento delle diverse impostazioni e configurazioni è stato scelto di memorizzare in un file JSON tutti i parametri principali della rete. Grazie a questo file è possibile memorizzare varie configurazioni per eseguire test differenti senza cambiare parametri via codice. Questa parte è implementata dalla classe Settings, che, dato il path del file JSON, legge e crea i vari oggetti necessari all'addestramento secondo quanto impostato.

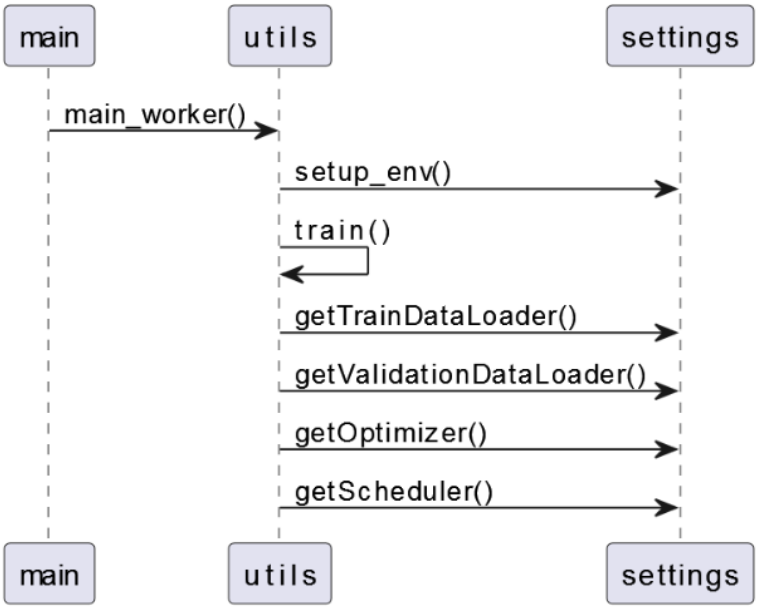

Figura 46: Funzionamento della classe Settings.

6.4.1 Caricamento impostazioni da file

Come menzionato la classe Settings è in grado di garantire una alta scalabilità e modularità al progetto e rendere le sperimentazioni il più possibile facili e tracciabili. In questo file, infatti, sono contenute le informazioni necessarie per addestrare la rete:

```
{
	"general":{
		"epochs": 10,
		"distributed": false,
		"perform_validation": true,
		"tensorboard": true,
		"dataset_root": "../fashionpedia/dataset",
		"labels_description": "labels_description.json",
		"output": {...},
		...
	},
	"optimizer": {
		"type": "SGD",
		"parameters": {
			"learning_rate": 0.005,
			"momentum": 0.9,
			"weight_decay": 0.0005
		}
	},
	"scheduler":{
		"type": "StepLR",
		"parameters": {
			"step_size": 5,
			"gamma": 0.3
		}
	},
	"train":{
		"path": "/train",
		"annos": "/instances_attributes_train2020.json",
		"batch_size": 12,
		"num_workers": 12,
		"shuffle": true
	},
	"validation":{
		"path": "/val_test",
```

```
            "annos": "/instances_attributes_val2020.json",
            "batch_size": 12,
            "num_workers": 12,
            "shuffle": true
        }
}
```

6.5 Conclusioni

Questa rete, seguendo il metodo illustrato, si dimostra flessibile e semplice da addestrare. Inoltre, grazie alla pipeline parallela creata, è semplice da addestrare su dataset grandi, variegati e complessi in tempi relativamente brevi e con un numero di epoche sufficienti. I risultati raggiunti durante questa sperimentazione sono molto promettenti e il loro utilizzo con dataset idonei è decisamente una strada percorribile per la segmentazione e la classificazione di abiti.

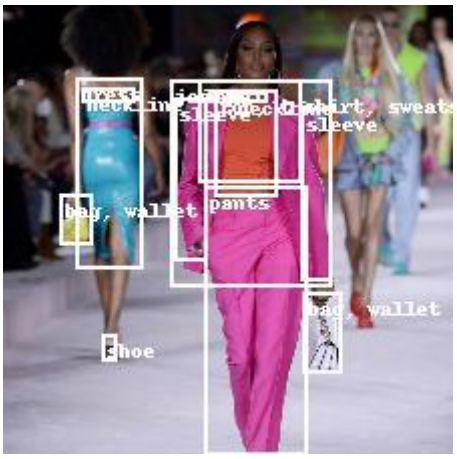

Figura 47: Estrazione delle bounding box da una immagine di test reperita su internet.

Il modello finale è dunque in grado di estrarre la maggior parte degli abiti in una immagine semplicemente dopo 10 epoche con una definizione e dovizia di particolari degne di nota. Inoltre, grazie al supporto del dataset al formato COCO è facile aggiungere ed estendere il dataset con altri dataset aventi classi compatibili o aggiungere nuove classi semplicemente estendendo le annotazioni presenti con bounding box e segmentazioni.

Idealmente, in un caso operativo, non sarebbe neanche necessario eseguire un nuovo addestramento se si vuole estendere la conoscenza della rete: è, infatti, possibile caricare un modello già addestrato in precedenza e mostrare i nuovi dati.

Per aumentare le classi in uscita, invece, è necessario un nuovo addestramento da zero o comunque modificare il numero di classi cablate internamente alla rete. Questa operazione, comunque, non è complicata.

In generale questa rete unita alla pipeline creata permette di riconoscere e segmentare gli abiti in maniera flessibile e precisa. Inoltre la pipeline sviluppata permette di testare facilmente, in modo tracciabile e scalabile varie configurazioni grazie al file di configurazione.

Dall'analisi condotta risulta evidente che la tecnologia attuale permette di creare soluzioni in grado di segmentare vestiti da una immagine con successo. La difficoltà, seppur grande, che rimane è quella di procurarsi dataset in grado di fornire segmentazioni di partenza con sufficiente precisione e definizione e in grado di potere indicare qualunque parte di un capo di abbigliamento.

6.5.1 Problemi della segmentazione

Se si considerano le prestazioni occorre parlare anche dei risultati logico/ontologici, e in questa prospettiva diviene immediatamente evidente che una rete di questo genere fa molta fatica, anche se addestrata su dataset estensivi, a distinguere particolari tipologie di vestiti. Si consideri l'esempio seguente come indicativo del problema: si distingue il concetto di "panciotto" da "gilet" laddove un panciotto, generalmente indossato solo sotto la giacca, è fatto dello stesso materiale di cui è fatta la giacca stessa o comunque di un tessuto di egual fattura, un gilet, invece, è generalmente accettato come meno formale e spesso di lana o un tessuto a trama semi spessa tipo cotone o poliestere. Nel vestiario comune risulta raro trovare occorrenze di persone indossanti entrambi, ma non è impensabile che qualcuno possa indossare un gilet sotto la giacca. A questo punto risulta il problema per la nostra rete di distinguere i due indumenti. Se ingenuamente si potrebbe arguire che un gilet non abbia i bottoni frontali, o le tasche, questo è sbagliato perché è il materiale che ne determina la differenza e, in alcuni casi, il materiale stesso non è sufficiente per indicare tale differenza e occorre guardarne il retro, in quanto il gilet è composto dello stesso materiale da ambo i lati mentre il panciotto è fatto di fodera sul retro. Questo è ovviamente non ottenibile da una fotografia.

Figura 48: A sinistra un gilet e a destra un panciotto

Occorrerà un dataset molto completo per rilevare queste differenze, e si tenga conto che ciò vale per un capo di vestiario solo mentre nella realtà le differenze tra vestiti sono molto più fini e variegate, portando il dataset a potenzialmente dimensioni colossali. Altri esempi possono essere le differenze tra body senza maniche, costumi da bagno interi o calze da donna e calzetti thigh high e tutti quegli altri casi dove i vestiti in considerazione sono morfologicamente simili ma ontologicamente diversi.

Si dovrà dunque per forza operare una compressione di alcuni vestiti verso macro categorie o elementi comuni per garantire la fattibilità della rete. Sussiste anche il rischio di incorrere nel fatto che la rete possa non essere in grado di distinguere differenze tanto sottili data la profondità attuale e una modifica alla stessa possa essere necessaria per garantire l'estrazione di un numero sufficiente di feature per districarsi nella complessa ontologia vestiaria. Queste considerazioni dovranno essere tenute in considerazione sia per la creazione di dataset futuri che per la creazione di reti in quanto si impatta con la definizione stessa di categorie.

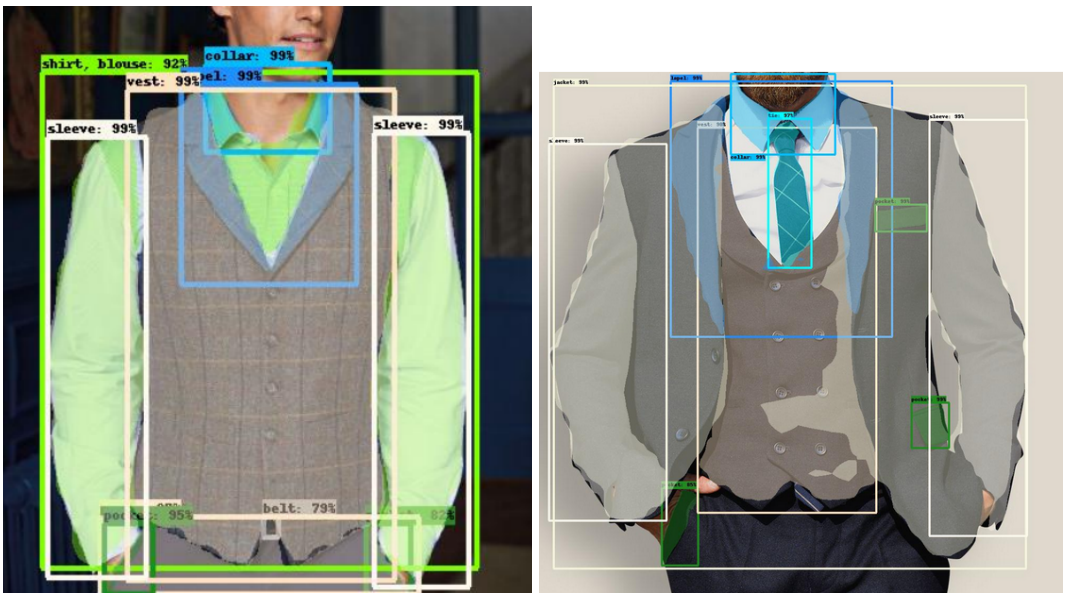

Figura 49: In queste due immagine un "vest" è stato riconosciuto propriamente ma, nel caso di destra, il revers del panciotto non sono stati propriamente identificati. Forse un panciotto con revers indossato sotto la giacca non era nel dataset?

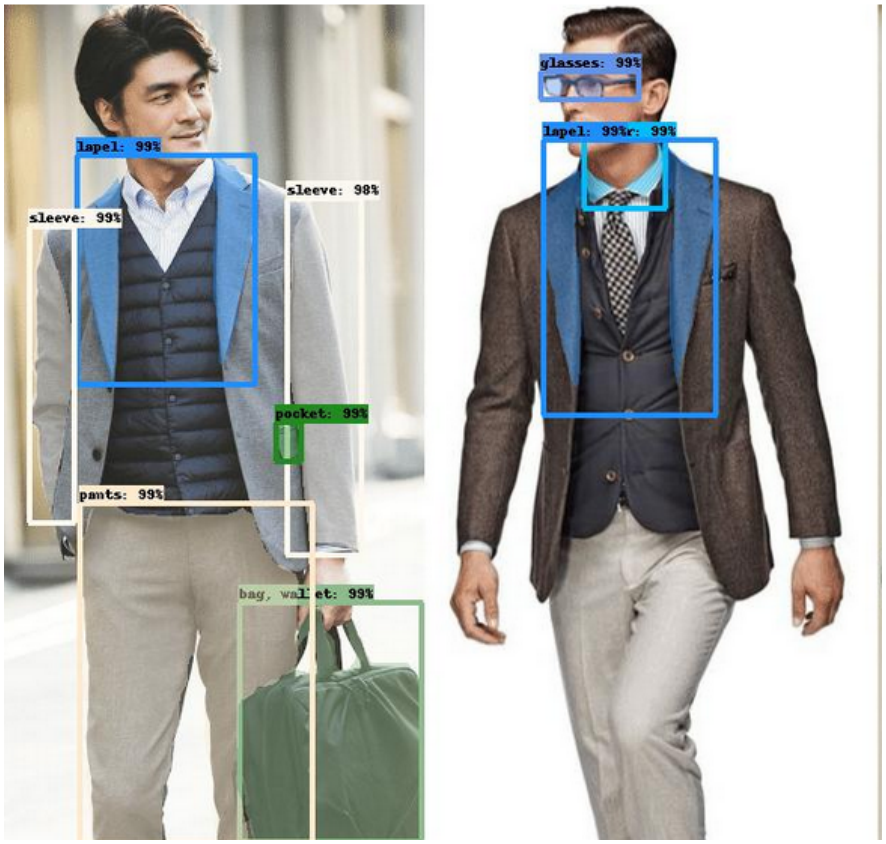

Figura 50: Un caso di fallimento catastrofico nell'individuare un gilet imbottito sotto una giacca.

Capitolo 7

Estrarre il successo di un abito da una sua immagine

Una importante caratteristica dei capi di abbigliamento, oltre a tutte quelle oggettive, è se il capo avrà o meno successo tra il pubblico, per prevedere se incontrerà i favori della maggior parte dei consumatori o addirittura riuscirà ad entrare a fare parte di quel piccolo insieme di capi che si possono definire iconici. Per analizzare questa caratteristica si è scelto di sperimentare con alcune reti, di vario genere, al fine di comprendere se fosse possibile o meno conoscere il gradimento di un capo semplicemente osservandone la forma e le caratteristiche principali. Per fare ciò due strade principali possono essere scelte: effettuare una regressione sui valori di vendita, ovvero stimare per ogni capo rappresentato quanto venderanno, o classificare i capi tra "di successo" e "non di successo". Ma per prima cosa occorre fare la domanda: "Cos'è il successo?". Questa domanda risulta essere alquanto complicata poiché si insinua in diversi campi del soggettivo e ha implicazioni molto grandi nei confronti della rete che si vorrà realizzare. In questa seconda parte l'obiettivo era quello di verificare se fosse possibile creare una rete in grado di predire se un abito sarebbe diventato un successo o meno partendo solamente dalle sue feature.

Come dataset, visto che era quello con il maggior numero di informazioni sulle vendite, si è scelto di usare Visuelle 2.0. Questo ha permesso una analisi su come le vendite si articolano nel corso del tempo.

7.1 Analisi sui valori di vendita

Un'attenta analisi dei dati di vendita annuali ha permesso di delineare come gli andamenti dei vari vestiti si articolino lungo gli anni[43]. Occorre inizialmente capire quali dati si abbiano a disposizione; come già accennato i dati di vendita sono settimanali e sono raggruppati per le etichette in cui il dataset è suddiviso. Questo significa che è possibile estrarre serie numeriche temporali al fine di visualizzare gli andamenti. Un'altra serie a disposizione è quella delle ricerche su piattaforme come Google. Queste sono state raccolte e mostrate con cadenza settimanale nel corso di svariati anni. Con questi dati è possibile dunque procedere con una analisi

puramente sulle serie per capire se un vestito che corrisponde alle label in questione possa essere di successo o meno nel corso del tempo. Questo significa associare il concetto di "capo di successo" a "capo molto ricercato su internet": ciò è in via di massima vero, ma non è detto in generale che si verifichi tale occorrenza, come delineato in seguito.

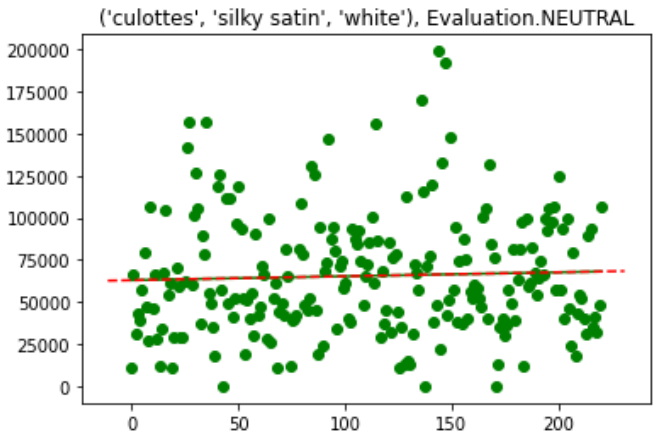

Figura 51: Diagramma a punti rappresentante l'andamento di un capo relativamente alle ricerche su Google Trend.

Si considerano le serie presenti nel dataset come suddivisibili in tre principali categorie: tipologia di vestito, colore e materiale. Ogni capo esistente apparterrà a una e una sola di ognuna di queste categorie. Essendo serie non collegate tra loro occorre (in alcuni casi) unificarle in un insieme di dati logico da potere analizzare con strumenti matematici. Una possibilità è quella di utilizzare quello che risulta essere un AND logico matematico, nella prima parte della trattazione si è infatti scelto di moltiplicare tra loro tali valori $C(a, b, c) = \frac{a \cdot b \cdot c}{3}$. La divisione per tre è solo per ridurre la dimensione dati e normalizzarli. Si è quindi proceduto con alcuni metodi naïve per analizzare tali serie: un primo esempio è stata l'applicazione di una funzione sinusoidale da usare come regressore al fine di estrarre la tendenza dei dati. Tale funzione è stata scelta come $y = (m + A\sin(x + w))(x + w) + h$. Il procedimento è stato dunque il seguente:

- ➔ processare i dati con una media mobile di periodo 13 (ovvero le settimane in una stagione);
- ➔ effettuare un campionamento con periodo 13 (equivalente a fare una media di passo 13 su tutto il dataset);

➔ applicare il regressore di funzione:

$y = (m + A\sin(x + w))(x + w) + h.$

➔ processare m come indicatore di successo.

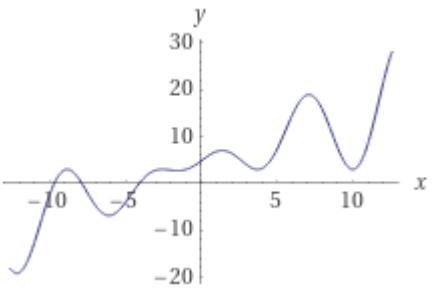

Figura 52: Rappresentazione della funzione scelta per la regressione.

Esistono inoltre altri indicatori utilizzabili a partire da tale funzione:

- A l'ampiezza della funzione
- w la fase della funzione
- h l'offset della funzione

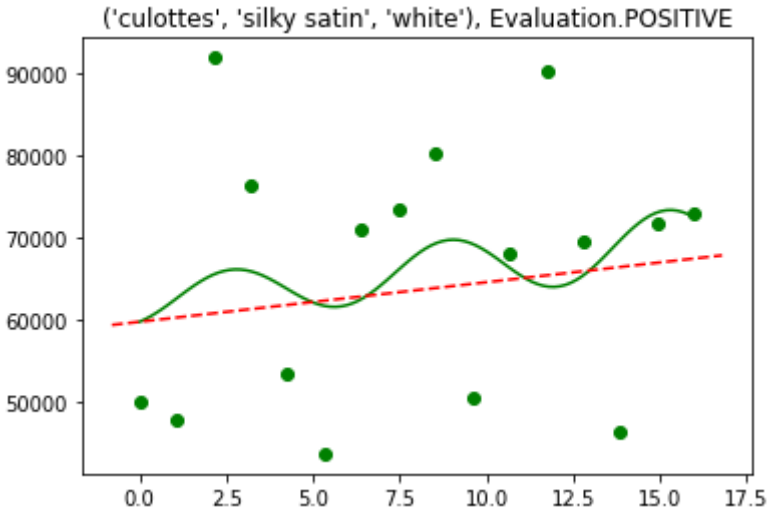

Figura 53: Applicazione del regressore dei dati reali.

Un altro metodo applicabile è l'analisi delle derivate della funzione serie. Questa analisi, concettualmente analoga alla prima, consente di controllare come la funzione varia nel tempo e quanto essa sia positiva o negativa. Permette inoltre di effettuare una compressione dei dati in un intervallo minore.

Questi metodi risultano, però, poco performanti alla luce di diverse osservazioni e ragionamenti che possono essere facilmente effettuati: intanto si sta introducendo una funzione arbitraria scelta solo per la sua forma e priva di significato reale e in secondo luogo i dati analizzati hanno comunque problemi di correlazione, ovvero, quando si analizzano dati come "fur" e "shorts" essi hanno periodo analogo (annuale) ma fase differente (uno viene cercato d'inverno e l'altro d'estate). Questo fa sì che qualunque metodo che non tiene conto di questo periodo possa perdere tali informazioni facendo sembrare due serie non correlate magari anche in tendenza positiva come "unite da qualche relazione positiva se prese insieme". Ciò non è per niente assicurato in caso di mancanza di correlazione tra le stesse.

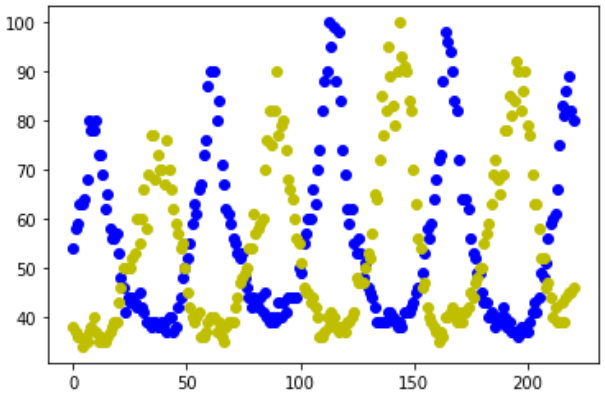

Figura 54: Rappresentazione della popolarità dei termini "fur" e "shorts", si noti l'andamento stagionale delle ricerche.

In secondo luogo analizzare da solo il coefficiente angolare causa perdita di informazioni, ovvero se consideriamo due serie con coefficiente angolare $m_1 = 0.3$ e $m_2 = 0.4$ potremmo dire che m_2 ha più successo, ma se poi scopriamo che $q_1 = 300$ e $q_2 = 2$ ovvero la prima retta era nettamente spostata verso l'alto e dunque avente più successo analizzare solo il coefficiente angolare causa perdita di informazioni e potenzialmente conclusioni errate.

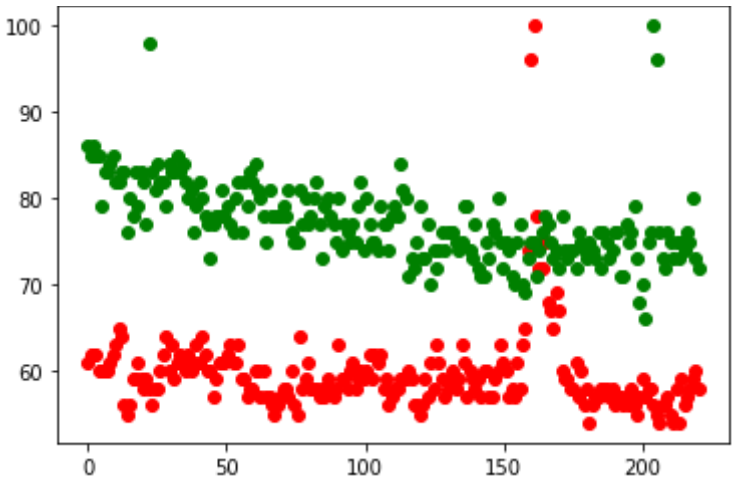

Figura 55: Esempio di dati aventi una q differente.

Alla luce di questi ragionamenti si è deciso di concentrarsi sulle vendite e di lasciare la regressione e la classificazione alla rete stessa, spostando questo problema all'interno della rete per lasciare che eventuali parametri e osservazioni di questo tipo possano essere ricavate automaticamente

7.2 Il problema del labelling

Per effettuare una divisione tra capi di successo e capi non di successo si è scelto di usare il percentile sui dati di vendita. Questo significa che si è deciso di rappresentare come di successo quei capi che erano oltre a una soglia del percentile di vendita e non di successo altrimenti. Una accurata analisi è stata effettuata per verificare dove si trovasse la migliore divisione nel dataset disponibile.

Occorre inoltre arginare il problema dei dataset sbilanciati: siccome il percentile di successo non è il 50% nella maggior parte dei casi, ciò significa che alcune classi avranno più campioni rispetto agli altri e questo può provocare dei bias nella rete. Per evitare ciò occorre bilanciare il numero di campioni presenti in ogni classe per cercare di non mostrare alla rete che esiste un caso più probabile dell'altro.

Durante i seguenti test quindi, una volta scelta la topologia della rete, si è cercato di individuare la migliore divisione del dataset in classi usando l'assunzione che più comprato significasse più di successo.

I dati in visuelle 2.0 seguono un formato standard basato su CSV, si è implementata una struttura dati DataSet custom per caricare correttamente i vari

dati in combinazione con le immagini. La sua struttura è simile a quella degli altri dataset riportati in questa trattazione.

7.2.1 Distribuzioni nel dataset

Riguardo alla distribuzione dei dati occorre notare che i dati risultano molto variegati nell'insieme d'ingresso: si osservi, ad esempio, la distribuzione delle vendite per i pantaloni neri.

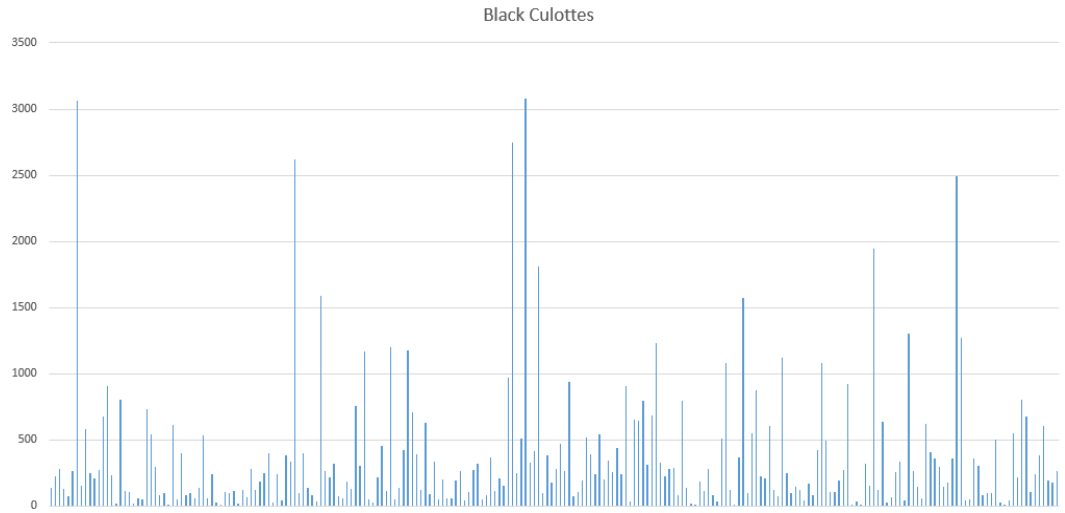

Figura 56: Distribuzione dei valori di vendita nel dataset per ogni capo.

A questo punto si ordinino i dati in maniera crescente in modo da poterne osservare meglio la distribuzione, si ottiene così un nuovo grafico sul quale verranno aggiunte anche le indicazioni su dove i percentili si trovino nella distribuzione al fine di semplificare la comprensione dell'esempio:

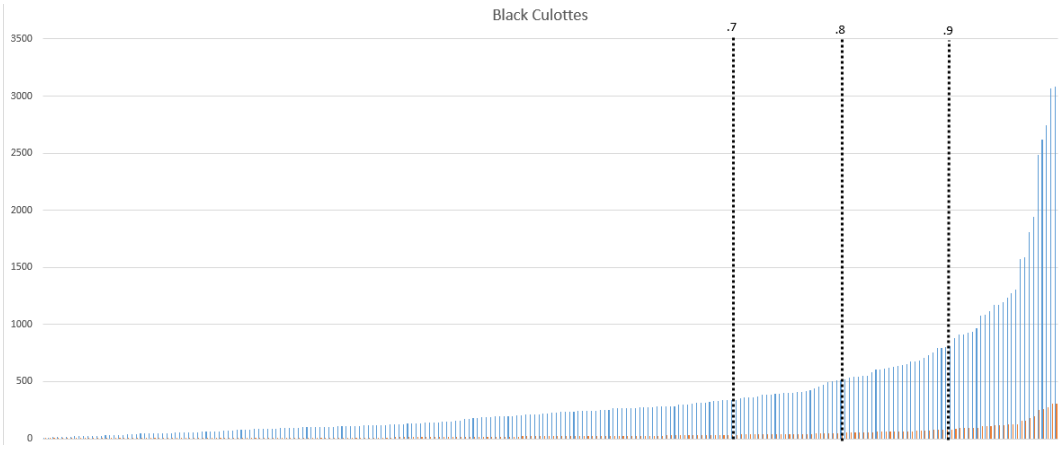

Figura 57: Risultato dell'operazione di ordinamento sui valori di vendita.

Si può notare come il 70% percentile non denoti una netta demarcazione delle vendite rendendo potenzialmente più difficile alla rete di riconoscere le feature in maniera appropriata da porre tale classificazione. Per semplificare si faccia riferimento al prossimo grafico dove le immagini sono raggruppate per quantità di vendita simili, sulle ordinate il numero di campioni e sulle ascisse il numero di vendite.

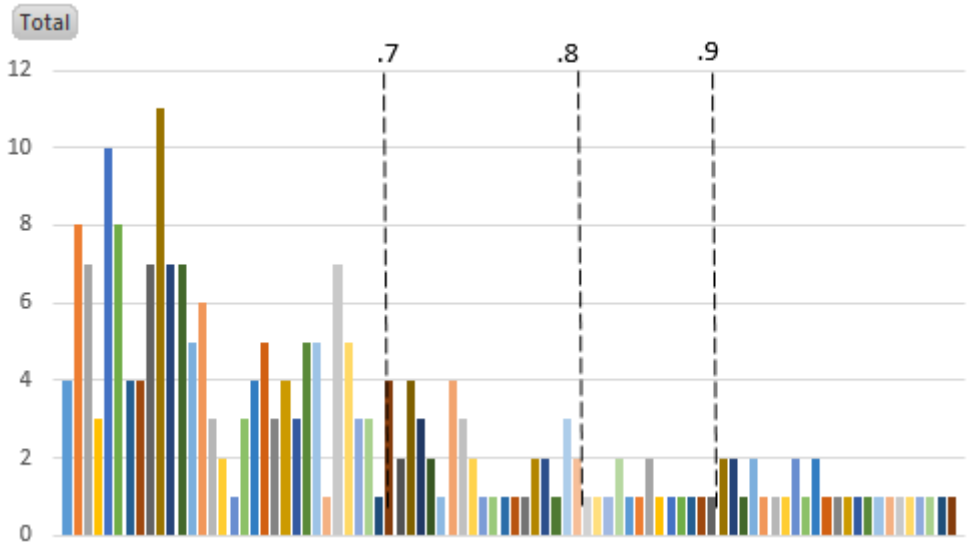

Figura 58: Distribuzione dei dati di vendita per campioni.

Si confronti anche la seguente tabella che illustra come sarebbe per ogni percentile la dimensione dello split del dataset e si consideri che, siccome si vuole evitare di avere dataset sbilanciati, si tenderebbe, in media, a considerare sempre il numero minore di valori per entrambi i dataset, ovvero scartare valori, magari con metodo, in eccesso della classe con più elementi:

Percentile	Samples	Compl
0,6	120	112
0,7	77	155
0,8	48	184
0,9	24	208

Il che significherebbe, qualora si usasse il 90% percentile, di limitare i campioni per entrambe le classi di successo e insuccesso a 24, valore decisamente troppo esiguo per un training corretto.

Per completezza si noti lo stesso pattern in un'altra categoria di vestiti:

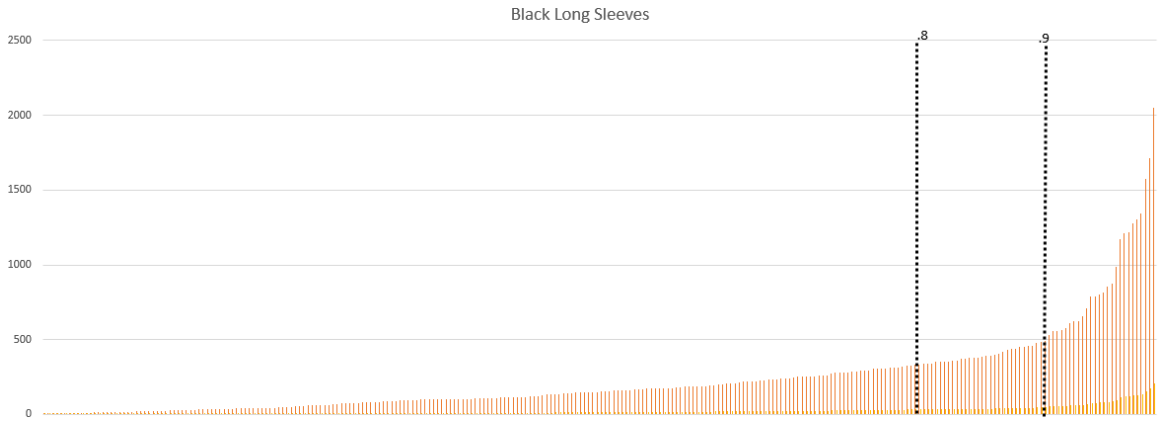

Figura 59: Risultato dell'operazione di ordinamento sui valori di vendita.

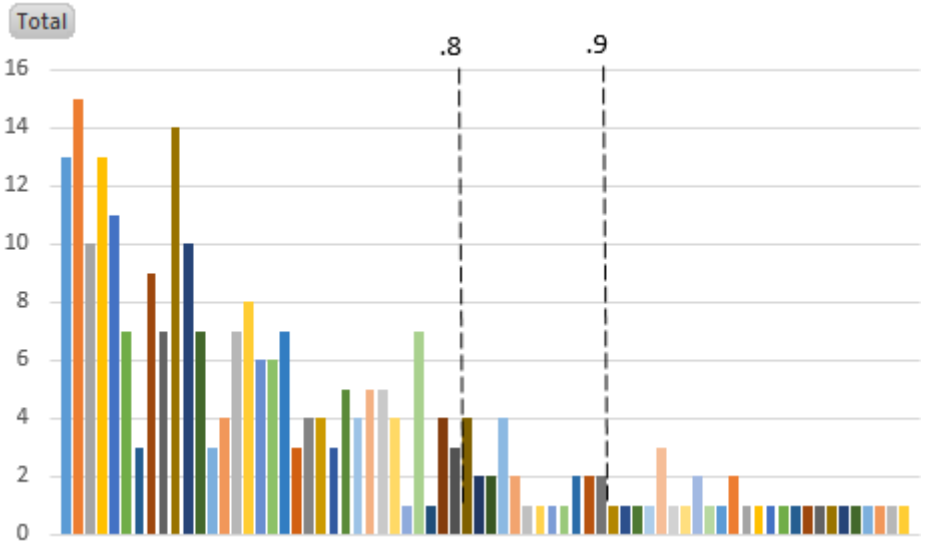

Figura 60: Distribuzione dei dati di vendita per campioni.

Percentile	Samples	Compl
0,6	113	153
0,7	87	179
0,8	56	210
0,9	28	238

Da questi dati si può notare come il vero discriminante sulle vendite considerabili come di successo avviene circa al 95° percentile, ovvero dimezzare ancora il numero dei campioni utilizzabile al 5% in media per classe.

Di seguito le distribuzioni di tutte le tipologie di vestiti, indicati il 30°, 70° e 90° percentile.

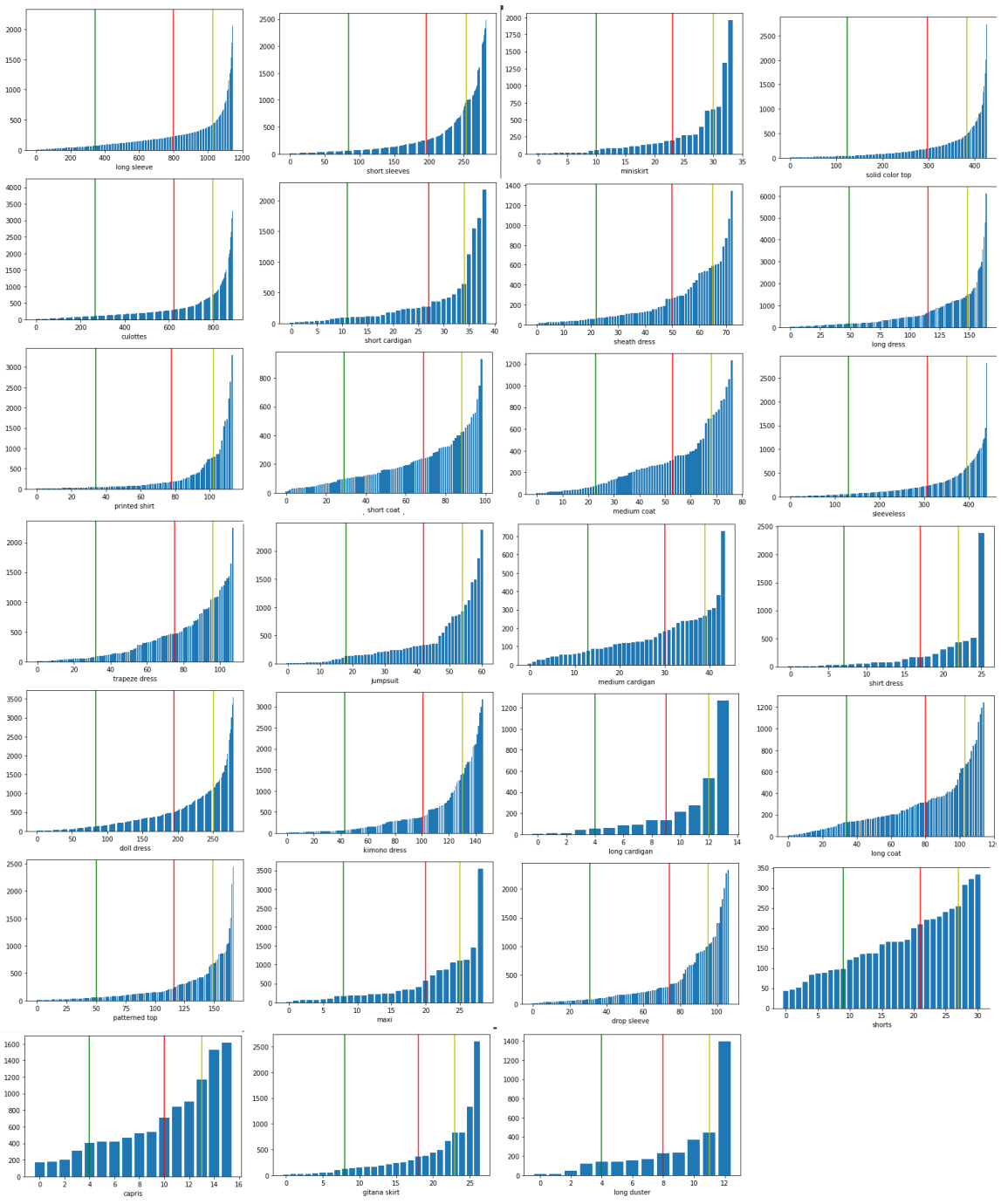

Figura 61: Risultato dell'operazione di ordinamento sui valori di vendita e relativa rappresentazione dei principali percentili.

7.2.2 Bilanciamento del dataset

Per garantire una buona suddivisione del dataset che non includa bias di alcun genere, occorre bilanciare le classi. Questa operazione di bilanciamento avviene riducendo il numero di campioni delle classi più numerose per evitare che la rete assegni "statisticamente" i dati di input alla classe più numerosa.

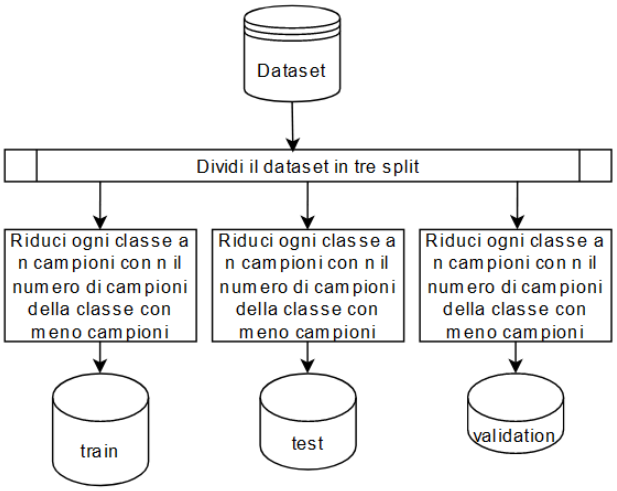

Figura 62: Descrizione del processo di bilanciamento del dataset e divisione in split.

Posto che i dati provenienti da Visuelle 2.0 non sono divisi in train, test e validazione, occorre dividere il dataset in questi tre split manualmente. Il dataset utilizzato consente la divisione utilizzando automaticamente uno split dato in input per facilitare la sperimentazione. Inoltre, per aumentare la precisione, si è effettuato un calcolo del percentile non su tutti i campioni presi assieme ma dividendoli per colore e tipologia. Il percentile, come mostrato negli esempi precedenti, fa riferimento quindi ai valori appartenenti al raggruppamento dei capi aventi stesso colore e tipo, questo per rendere il percentile scelto il più valido possibile per definire il successo

7.3 L'architettura della rete

Per l'estrazione delle feature si è deciso di utilizzare una rete ResNet50 come backbone in quanto candidata migliore tra le altre come rapporto sperimentale tra velocità e accuratezza delle feature estratte: questa decisione è stata presa in seguito a una sperimentazione con una rete convoluzionale creata ad hoc. Tale rete aveva però una capacità di estrazione di feature molto ridotte e si è deciso di concentrarsi verso una soluzione già esistente e sperimentata. La rete in questione, ResNet50, restituisce un vettore di 2048 feature che possono essere poi analizzate da altre reti in passi successivi. Una volta estratte le feature mediante la ResNet è possibile utilizzare degli strati completamente connessi per estrarre le label relative all'immagine inserita. Esempi di tali dati sono il colore, la tipologia e la stagione di introduzione sul mercato del vestito. Al termine degli strati connessi occorre

mettere uno strato Softmax per la normalizzazione dei dati d'uscita. In questo modo è possibile associare un insieme di feature a una classe corrispondente. Questo processo è normalmente detto di classificazione ed è lo stesso operato dalle reti precedenti per il riconoscimento di oggetti.

7.3.1 Differenza tra regressione e classificazione

Un altro tentativo effettuato è stato di usare una rete per la regressione invece che per la classificazione: invece di voler associare una classe a una immagine si propone ora di associare un numero, che dovrà rappresentare al meglio il numero delle vendite attese. Anche in questo caso la rete è simile a quella descritta in precedenza ma privata dello strato Softmax finale. Il problema risulta però molto più complesso per via dei dati in entrata che possono essere anche non correlati con il valore cercato: si confronti ad esempio con il problema di trovare numeri in una immagine o con dati che hanno effettivamente un regressore possibile tra input e output. Questo si traduce come il bisogno di una rete più profonda per potere raggiungere tale risultato.

Per estrarre le vendite future i dati del dataset devono essere prima elaborati, per fare ciò occorre sommare tutte le vendite mensili dei singoli negozi e poi sommare il risultato ottenuto per avere un valore unico per ogni immagine.

$$TotalSales = \sum_{s \in Shops} \sum_{v \in Sales(s)} v.$$

Questo valore sarà utilizzato come riferimento durante l'addestramento con l'obiettivo di ottenere un valore più vicino possibile allo stesso in uscita.

7.4 Sperimentazione

Per verificare la fattibilità di questa rete sono stati effettuati diversi esperimenti al fine di scoprire quale fosse la metodologia migliore e le dimensioni ottimali dei vari dataset di training, test e validazione.

7.4.1 Regressione

I primi esperimenti sono stati eseguiti con la rete in modalità regressione. Questa rete è addestrata per 100 epoche sul dataset con split 70/20/10, un ottimizzatore di tipo Adam e usando L1Loss o MSELoss come funzione di loss. Viene usato il seguente metodo per definire il successo nel valore predetto: $Success(found, expected) = expected \cdot 0.7 < found < expected \cdot 1.3$. Questo significa accettare come valido un valore che si trova nell'intorno del valore atteso. Si noti che per definizione più il numero è grande più l'intorno si ingrandisce per garantire un po' di lasco ai numeri trovati. Applicando tale metodo la rete ottiene un tasso di successo del 21% rispetto al dataset di test. Per completezza il valore del tasso di successo sul dataset di training era superiore al 91%. Un tasso di successo del 21% significa che, considerata la distribuzione dei valori di vendita e la funzione di successo, la probabilità di estrarre a caso un valore in quell'intervallo è del 20% circa, ciò significa che la rete in questione non sta imparando a predire il successo come ci si potrebbe aspettare.

La precedente affermazione è ottenuta calcolando la varianza e la media sulle vendite totali relative a un campione di immagini. Da questi valori è possibile considerare la distribuzione delle vendite come una variabile casuale gaussiana di valore medio e varianza nota e calcolando la probabilità di individuare un valore all'interno dell'intervallo $[Media \cdot 0.7, Media \cdot 1.3]$ ovvero, considerata la media come valore che chi si aspetta si contano i punti attorno ad essa. Un ragionamento analogo e con risultati altresì analoghi lo si può fare utilizzando una variabile casuale uniforme, in tal caso il valore sale al 25% circa ma sempre nell'intervallo espresso dalla rete. Questo significa che questo metodo non dispone di una accuratezza adeguata per risolvere il problema.

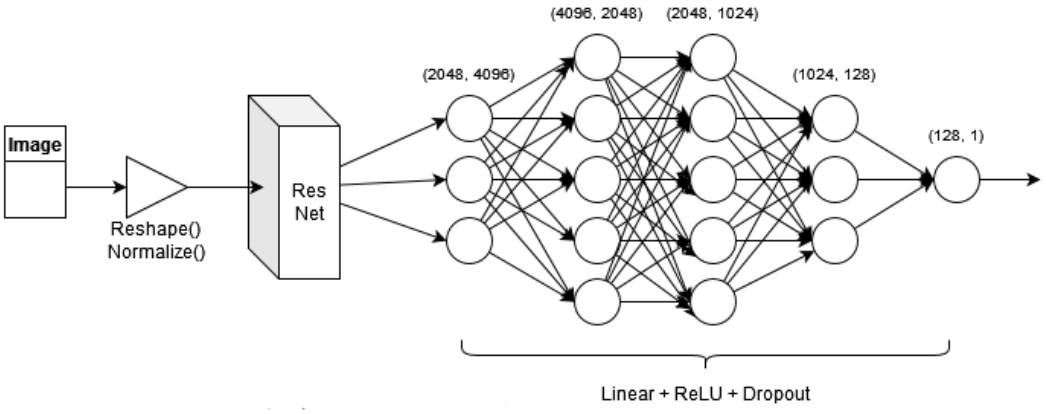

Figura 63: Rappresentazione degli strati della rete in modalità regressione

Come menzionato in precedenza, per la regressione la rete sarà composta da una backbone di tipo ResNet, e da svariati strati completamente connessi, per ogni strato si è utilizzata una funzione di attivazione ReLU. Al termine si ha un solo valore di uscita: il valore previsto dalla rete.

7.4.2 Classificazione

Nei test successivi si è dunque passati a un sistema avente l'obiettivo di identificare e predire e se un vestito potesse essere di successo o meno mediante un classificatore.

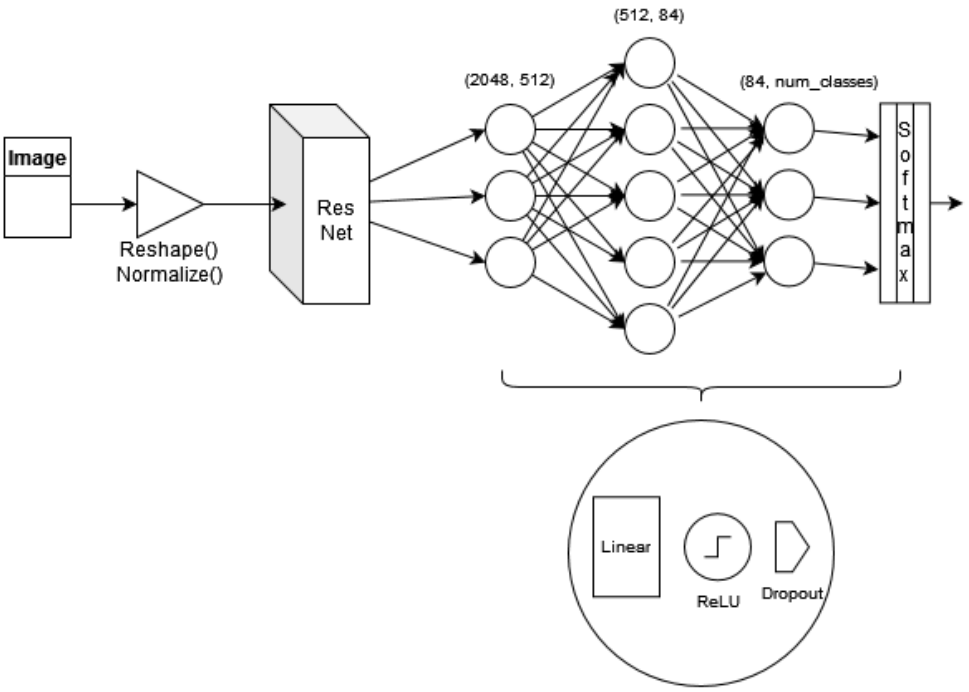

Figura 64: Rappresentazione degli strati della rete in modalità classificazione.

La rete, nel suo formato di regressione, sarà composta da una backbone di tipo ResNet, seguita dagli stessi strati lineari utilizzati anche in precedenza e, al termine, una funzione Softmax per estrarre solo l'indice della classe di appartenenza dell'oggetto cercato.

Un primo esperimento effettuato è stato quello di associare una label "SUCCESSO" ai vestiti con numero vendite maggiori di 300 (numero arbitrario) e "INSUCCESSO" agli altri. Si è proseguito quindi a passare l'intero dataset così etichettato alla rete di classificazione già menzionata. In questo caso il tasso di successo è di circa il 65%, ma occorre notare che tale valore è all'incirca il numero medio di valori di classe "INSUCCESSO". Ciò significa che la rete non ha davvero imparato cosa significhi il concetto di successo. Anche in questo caso occorre ricordare che il tasso di correttezza sul dataset di training era superiore al 90% e che quindi non ci si trova in situazioni nelle quali la rete va in overfitting o non si addestra in maniera propria.

Si consideri a questo punto la strategia di usare il percentile come tasso di successo o insuccesso, ovvero si può indicare un vestito come di successo se appartenente ad uno specifico percentile. Per fare ciò il dataset, già suddiviso in classi analoghe viene integrato di un dato aggiuntivo, ovvero il valore del percentile designato dal quale è possibile calcolare se ogni singolo dato è o meno di successo o meno.

Un altro fattore da considerare è come raggruppare i dati per determinare le divisioni in percentile: per avere più granularità, infatti, è possibile raggruppare i capi per colore, materiale e tipologia e poi calcolare il percentile su ogni singolo raggruppamento per avere risultati più precisi. Nel corso delle sperimentazioni si è visto che una divisione per materiale non risulta conveniente, ma quella ottimale è una divisione per colore e tipologia. Gli esperimenti in seguito riportati fanno riferimento a tale raggruppamento.

Di seguito i grafici dell'accuratezza raggiunta per classi utilizzate con il tasso di successo posto al 60% e al 70% percentile, sulle ordinate l'accuratezza e sulle ascisse il numero di classi incluse nell'esperimento. Come si nota non vi è una vera relazione tra le due.

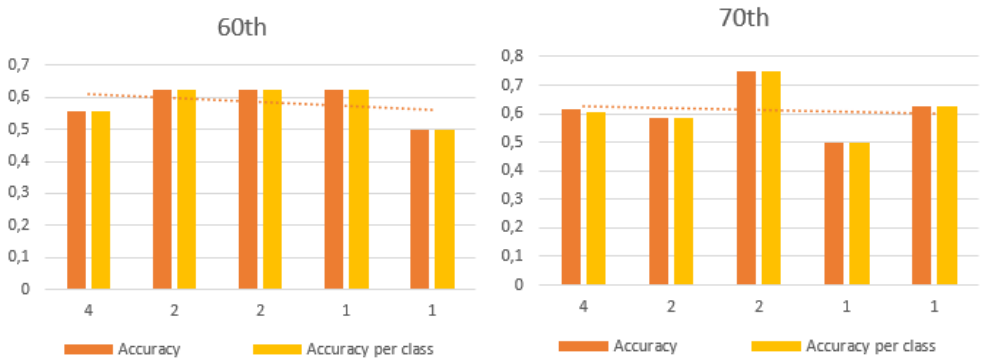

Figura 65: Accuratezza per il caso al 60% e al 70% percentile.

7.4.3 Matrici di confusione

Per comprendere quali valori del percentile fossero i più efficienti sono stati svolti diversi test utilizzando divisioni in classi differenti e valori diversi di percentili. Per comprendere al meglio i risultati ottenuti si è fatto ricorso a uno strumento chiamato “Confusion matrix". Di fatto si tratta di una tabella che rappresenta, per ogni cella, il numero di campioni predetti da una rete neurale in una determinata categoria rispetto alla categoria a cui appartengono effettivamente. Nella sperimentazione in seguito si è voluto cambiare non solo il valore dei percentili, ma anche il numero di classi in cui si voleva dividere i campioni.

7.4.3.1 Implementazione con 3 classi

Utilizzando implementazione con dataset bilanciato in modo che ogni classe abbia esattamente lo stesso numero di elementi, questa rete è stata sottoposta a un addestramento su una sola tipologia di vestiti (pantaloni) finché la validation non diventa stabile per 100 epoche (circa 400 epoche).

I risultati

Nel corso degli esperimenti sono stati variati i percentili per comprendere quale divisione fosse la migliore; la tabella riportata in seguito mostra la confidenza raggiunta nell’individuare i membri di ogni classe e la confidenza media per ogni tentativo effettuato:

	0	1	2	Avg	Intra class difference
50-80	0,32	0,41	0,53	0,42	0,20
50-85	0,38	0,42	0,58	0,46	0,19
55-80	0,47	0,52	0,50	0,50	0,05
55-85	0,33	0,38	0,64	0,45	0,31
60-85	0,35	0,40	0,50	0,42	0,15
60-80	0,38	0,42	0,37	0,39	0,05
60-90	0,31	0,24	0,35	0,30	0,11

Tabella 2: Confidenza per ogni classe per ogni intervallo di percentili provati.

Mediante un grafico si può comprendere meglio come i valori cambino a seconda dell'intervallo: di seguito si annotano l'accuratezza per classe:

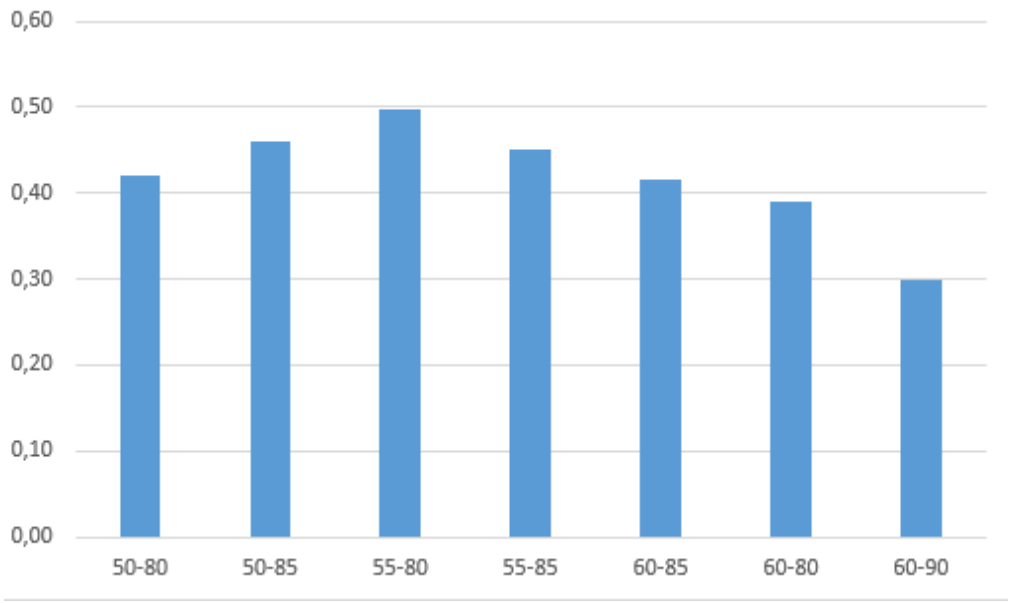

Figura 66: Accuratezza per ogni intervallo di percentili.

e la differenza tra minima e massima accuratezza per ogni classe:

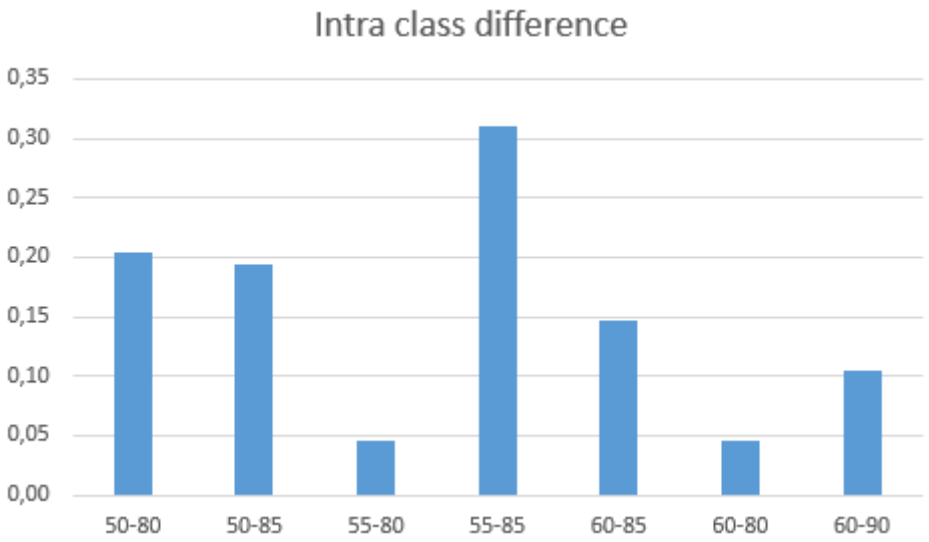

Figura 67: Differenza intraclasse tra minima e massima accuratezza.

L'accuratezza per ogni classe in ogni tentativo è, invece:

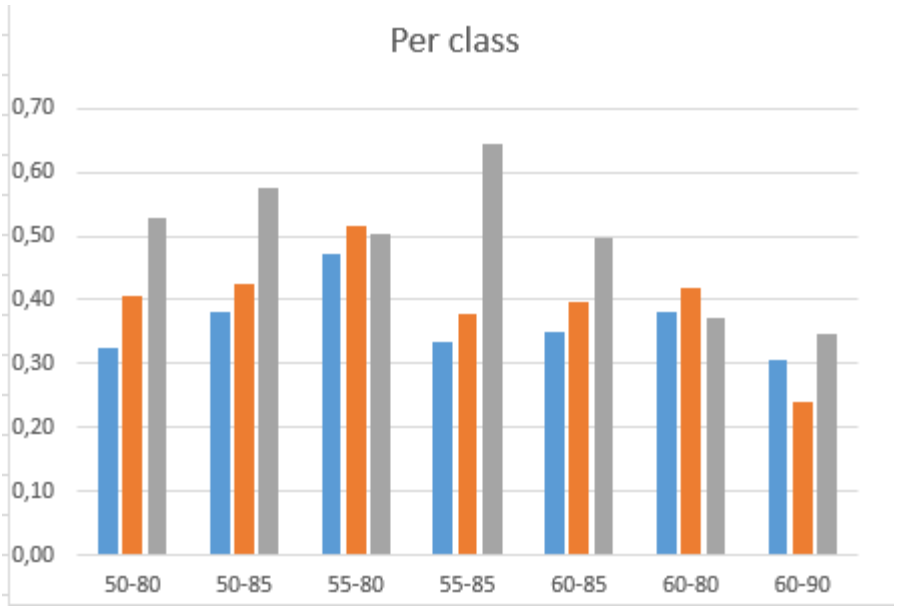

Figura 68: Accuratezza di ogni classe per intervallo di percentile.

Da questa analisi si può notare come i percentili migliori utilizzabili sono il 55° e l'80°, che garantiscono una confidenza del 50%, scelti non solo per l'alta confidenza media ma anche per la minor varianza intraclasse.

Classi divise al 60° e 80° percentile

Di seguito è riportata la matrice di confusione per la divisione delle classi usando come percentile il 60° e l'80°. Tali esperimenti sono stati ripetuti per tre volte, la matrice racchiude una media algebrica di tutte le run:

		Found		
	199,8	0	1	2
Expected	0	25,4	27,2	14,0
	1	17,0	27,8	21,8
	2	21,0	20,8	24,8

Tabella 3: Matrice di confusione rappresentante la media di tutte le run per il caso 60° e 80°.

Si può notare come in questo caso esista confusione tra le classi in fase di inferenza. Un primo tentativo di risolvere il problema è stato quello di variare il percentile più alto.

Classi divise al 60° e 85° percentile

La matrice di confusione per la divisione delle classi usando come percentile il 60° e l'85°, ovvero variando il percentile dall'80° al 85°, viene dunque a variare nel seguente modo (si ricorda che questa sperimentazione è stata ripetuta per tre volte e che la matrice racchiude una media algebrica di tutte le run):

		Found		
	210,0	0	1	2
Expected	0	25,0	28,5	16,5
	1	16,5	27,8	25,7
	2	9,8	25,3	34,8

Tabella 4: Matrice di confusione rappresentante la media di tutte le run per il caso 60° e 85°.

Continuando sulla stessa strada si può verificare confusione tra la classe 1 e la classe 2.

Classi divise al 60° e 90° percentile

Segue la matrice di confusione per la divisione delle classi usando come percentile il 60° e l'90°. Anche tali esperimenti sono stati ripetuti per tre volte e la matrice racchiude una media algebrica di tutte le run:

		Found		
	128,3	0	1	2
Expected	0	17,5	17,8	7,5
	1	13,0	13,8	16,0
	2	10,3	12,8	19,8

Tabella 5: Matrice di confusione rappresentante la media di tutte le run per il caso 60° e 90°.

A questo punto non è più possibile alzare il percentile maggiore, quindi si posta l'attenzione sul minore.

Classi divise al 55° e 85° percentile

Segue la matrice di confusione per la divisione delle classi usando come percentile il 55° e l'85°, in questo caso si è preferito ridurre entrambi i percentili per garantire un numero di campioni appropriato. Tali esperimenti sono stati ripetuti per tre volte, la matrice racchiude una media algebrica di tutte le run:

		Found		
	151,2	0	1	2
Expected	0	16,8	20,0	13,6
	1	10,0	19,0	21,4
	2	5,4	12,6	32,4

Tabella 6: Matrice di confusione rappresentante la media di tutte le run per il caso 55° e 85°.

In questo caso sembra che la classe 2 sia stata riconosciuta abbastanza bene ma permane un po' di confusione tra le altre.

Classi divise al 50° e 80° percentile

Segue la matrice di confusione per la divisione delle classi usando come percentile il 50° e l'80°. Di nuovo questi esperimenti sono stati ripetuti per tre volte, la matrice racchiude una media algebrica di tutte le run:

		Found		
	199,8	0	1	2
Expected	0	21,6	24,6	20,4
	1	14,6	27,0	25,0
	2	10,4	21,0	35,2

Tabella 7: Matrice di confusione rappresentante la media di tutte le run per il caso 50° e 80°.

Classi divise al 50° e 85° percentile

Di seguito è riportata la matrice di confusione per la divisione delle classi usando come percentile il 50° e l'85°. Tali esperimenti sono stati ripetuti per tre volte, la matrice racchiude una media algebrica di tutte le run:

		Found		
	151,2	0	1	2
Expected	0	19,2	17,8	13,4
	1	12,4	21,4	16,6
	2	11,8	9,6	29,0

Tabella 8: Matrice di confusione rappresentante la media di tutte le run per il caso 50° e 85°.

Classi divise al 55° e 80° percentile

Per ultima, la matrice di confusione per la divisione delle classi usando come percentile il 55° e l'80°. Tali esperimenti sono stati ripetuti per tre volte, la matrice racchiude una media algebrica di tutte le run:

		Found		
	199,8	0	1	2
Expected	0	31,4	17,2	18,0
	1	14,0	34,4	18,2
	2	12,8	20,2	33,6

Tabella 9: Matrice di confusione rappresentante la media di tutte le run per il caso 55° e 80°.

In questo ultimo caso si attesta la più alta confidenza e il miglior grado di accuratezza finora rilevato. La ricerca potrebbe continuare, ovviamente, all'infinito, ma si è scelto di terminare qui e di indicare questo intervallo come il migliore, su questo dataset, per identificare le varie classi.

7.4.3.2 Implementazione con 2 classi

L'implementazione utilizzata per questa parte è stata sviluppata in seguito e non ha bilanciamento sulle classi. Inoltre è stata effettuata una suddivisione molto più granulare rispetto alle precedenti: si considerano, infatti, anche i materiali durante la classificazione e definizione dei percentili. L'addestramento avviene su una sola tipologia di vestiti (pantaloni) per 50 epoche.

Classi divise al 70° percentile

La matrice di confusione per la divisione delle classi usando come percentile il 70° è dunque la seguente:

		Found	
	114,17	0	1
Expected	0	46,83	25,67
	1	24,50	17,17

Tabella 10: Matrice di confusione rappresentante la media di tutte le run per il caso 70°.

Come si può notare non presenta una divisione ottimale. Variando il percentile verso l'alto non si ottengono, tuttavia, risultati soddisfacenti.

Classi divise al 85° percentile

Segue la matrice di confusione per la divisione delle classi usando come percentile il 85°, come si può notare sussiste una profonda confusione tra le due classi.

		Found	
	114,50	0	1
Expected	0	74,50	13,33
	1	18,83	7,83

Tabella 11: Matrice di confusione rappresentante la media di tutte le run per il caso 85°.

I risultati

I risultati mostrano che la rete in questione risulta avere dei problemi, che possono essere ricondotti alla divisione in percentili stessa. Infatti, questa rete utilizza non solo i dati dei colori che delle tipologie, ma anche quelli legati al materiale. Questo fa sì che la divisione sia molto più variegata portando, in generale, una diminuzione della precisione della rete. Anche il fatto per cui il dataset utilizzato in questa parte di sperimentazione non sia bilanciato ha una influenza negativa: come già detto, infatti, senza un dataset ben ponderato e calibrato si rischia di introdurre bias nella rete stessa.

In seguito sono stati condotti esperimenti con una rete analoga, ma con dataset bilanciato e senza la divisione per materiale. I risultati ottenuti avevano una accuratezza attorno al 70%, risultato degno di nota, visto il problema.

7.4.3.3 Implementazione con 4 classi

Per questa parte del test si è tornato ad utilizzare l'implementazione con dataset bilanciato in modo che ogni classe abbia esattamente lo stesso numero di elementi. L'addestramento avviene, anche questa volta, su una sola tipologia di vestiti (pantaloni) finché la validation non diventa stabile per 100 epoche (ovvero per circa 400 epoche).

I risultati

Anche nel corso di questi esperimenti sono stati variati i percentili per comprendere quale divisione fosse la migliore:

	0	1	2	3	Total
40-55-80	0,21	0,25	0,37	0,42	0,31
30-55-80	0,32	0,38	0,36	0,41	0,37
25-55-80	0,18	0,27	0,34	0,37	0,29
20-55-80	0,22	0,32	0,48	0,32	0,34

Tabella 12: Confidenza per ogni classe per ogni intervallo di percentili provati.

Di seguito è documentata attraverso un istogramma l'accuratezza per classe:

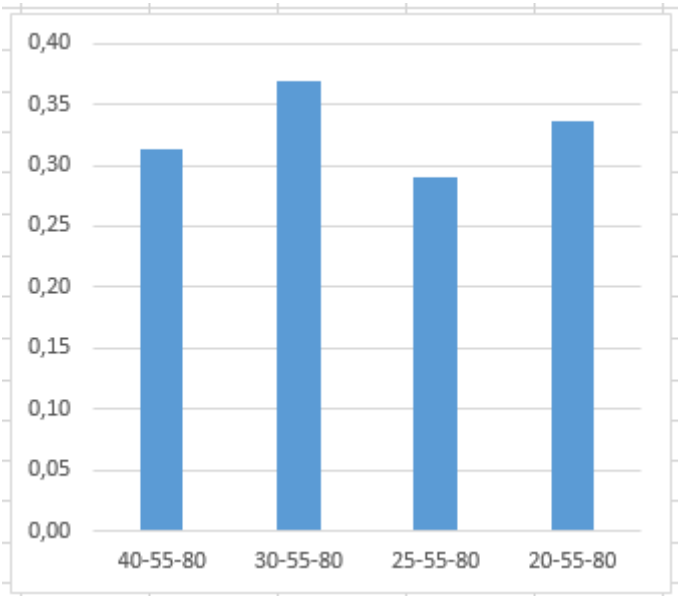

Figura 69: Accuratezza per ogni intervallo di percentili.

La differenza tra minima e massima accuratezza per ogni classe:

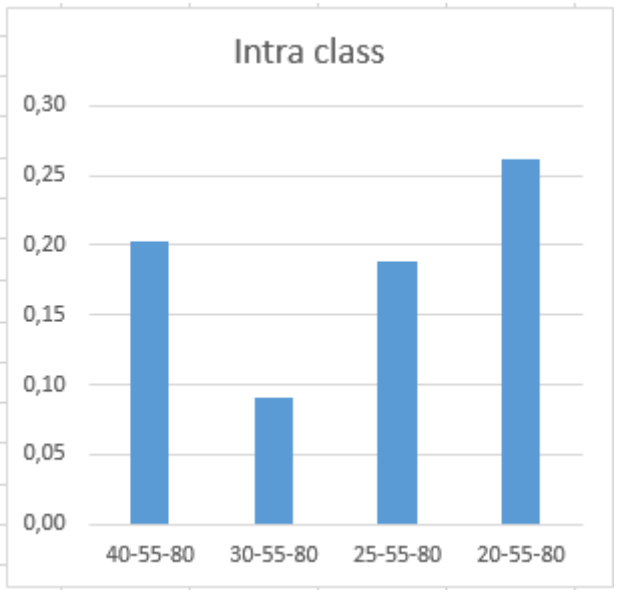

Figura 70: Differenza intraclasse tra minima e massima accuratezza.

Accuratezza per classe è la seguente:

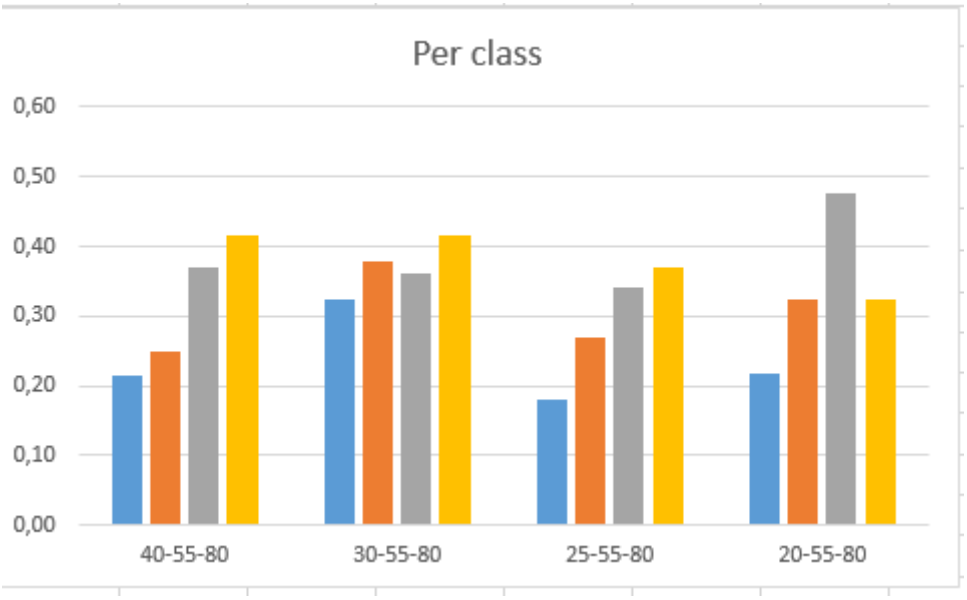

Figura 71: Accuratezza di ogni classe per intervallo di percentile.

Al termine di questa parte si può notare come la divisione migliore fosse quella utilizzante il 30°, il 55° e l'80° percentile.

Classi divise al 40°, 55° e 80° percentile

Di seguito la matrice di confusione per la divisione delle classi usando come percentile il 40°, il 55° e l'80°.

		Found			
	336	0	1	2	3
Expected	0	18	27	15	24
	1	14	21	35	14
	2	6	21	31	26
	3	15	14	20	35

Tabella 13: Matrice di confusione rappresentante la media di tutte le run per il caso 40°, 55° e 85°.

Si può notare come in questo caso sia molto più difficile capire quali siano le classi che causano confusione nella rete visto il loro alto numero; l'obiettivo è, comunque, di ottenere valori il più alto possibile sulla diagonale principale.

Classi divise al 30°, 55° e 80° percentile

Segue la matrice di confusione per la divisione delle classi usando come percentile il 30°, il 55° e l'80°.

		Found			
	444	0	1	2	3
Expected	0	36	45	25	5
	1	18	42	43	8
	2	13	38	40	20
	3	12	26	27	46

Tabella 14: Matrice di confusione rappresentante la media di tutte le run per il caso 30°, 55° e 80°.

Questa è la matrice che al meglio approssima il comportamento cercato, anche se, come si può notare, sussiste ancora qualche confusione tra le classi specie quelle adiacenti tra loro. Questo potrebbe anche significare che si stanno cercando di introdurre più classi rispetto a quelle effettivamente presenti.

Classi divise al 25°, 55° e 80° percentile

Segue la matrice di confusione per la divisione delle classi usando come percentile il 25°, il 55° e l'80°.

		Found			
	444	0	1	2	3
Expected	0	36	45	25	5
	1	18	42	43	8
	2	13	38	40	20
	3	12	26	27	46

Tabella 15: Matrice di confusione rappresentante la media di tutte le run per il caso 25°, 55° e 80°.

Classi divise al 40°, 55° e 80° percentile

Infine, la matrice di confusione per la divisione delle classi usando come percentile il 40°, il 55° e l'80°.

		Found			
	444	0	1	2	3
Expected	0	24	19	42	26
	1	21	36	42	12
	2	6	22	53	30
	3	18	18	39	36

Tabella 16: Matrice di confusione rappresentante la media di tutte le run per il caso 40°, 55° e 80°.

7.4.4 Note sull'addestramento

Per l'addestramento è stata utilizzata una pipeline del tutto simile a quella descritta nella parte precedente: per ogni epoca si effettuava una parte di training seguita da una di validazione al fine di salvare il modello solamente se la perdita durante la validazione attuale era inferiore a quella minima mai riscontrata. In seguito all'esecuzione di un numero di epoche deciso a priori si salvava il modello con loss minima.

7.5 Conclusione

I risultati ottenuti nel corso di questa parte di sperimentazione non sembrano essere quelli ottimali, ma occorre considerare che la rete utilizzata è molto semplice e, se comparata con altre reti strutturate in modo analogo e con task simili, rispetta i risultati allo stato dell'arte. Si possono confrontare, difatti, i risultati ottenuti per due classi (successo e insuccesso) con un paper proveniente dall'università di Tubinga[40] che mirava, tramite un classificatore, a selezionare se una immagine sarebbe stata di gradimento o meno. Tale paper riportava come accuracy finale un valore attorno al 70%, del tutto conforme con quelli ottenuti nel corso di questa sperimentazione. Occorre comunque menzionare l'esistenza di altri paper che riportano valori di accuratezza più alti con task più "semplici" quali la valutazione dell'estetica di una immagine[40]. Per una analisi più precisa, infatti, occorrerà, ad esempio, coinvolgere molti più dati di input, quali dati di vendita, di tendenza e affini, come del resto già delineato nel paper accluso a Visuelle e Visuelle 2.0.

Si è dovuto, inoltre, sperimentare su come creare, dividere e bilanciare un dataset in modo da ottenere risultati il più precisi e liberi da bias possibile, cercando al contempo di avere la dimensione più grande possibile del dataset stesso in modo che fossero variegati. Particolare attenzione è stata fatta anche nella divisione in split di training, test e validazione in modo che il numero delle classi in ognuno fosse uguale così da potere ottenere risultati con rilevanza statistica.

Una domanda interessante, perlopiù posta durante tutta la progettazione di questa rete, rimane come definire il successo. Infatti basti pensare come la moda possa variare non solo in modo lento e collettivo ma anche rapidamente con l'introduzione di accessori mai visti prima, o a seguito di qualche celebrità o

movimento sociale. Questi fattori sono difficili - se non addirittura impossibili - da modellare in una rete neurale visto che sono completamente esogeni al mondo che essa vede.

Capitolo 8
Conclusioni

Nel corso di questa sperimentazione si è potuto studiare nel dettaglio non solo i dataset afferenti al mondo della moda, ma anche come essi siano in continua moderna - e necessaria - evoluzione. Il campo della moda e delle reti neurali applicate al mondo della moda è in una condizione di sviluppo continuo con nuovi e interessanti paper pubblicati ogni giorno. Ciò consente l'applicazione di nuovi approcci a scenari potenzialmente mai visti prima e in continua evoluzione. In una prima fase si è potuto approfondire e ricercare come un dataset sia strutturato, definito e, più importante, capire come sceglierne uno che sia "il migliore" per i compiti per cui si intende utilizzarlo, allo stesso tempo idoneo, adeguato e aggiornato.

Scelto il dataset è stato quindi possibile svolgere un vero e proprio studio su implementazioni e modelli di reti neurali esistenti, creazione di proprie reti e, visti i risultati, catalogazione delle stesse in modo da verificare la fattibilità di task di segmentazione e riconoscimento di abiti all'interno di una immagine. Questo task permette, come sviluppo futuro, la creazione di una applicazione - o più in generale di un sistema - in grado di categorizzare un abito, uno stile o una parte dell'abito a partire semplicemente da una foto. Questo ha permesso, inoltre, di approfondire l'addestramento di reti neurali in modo parallelo, distribuito e multi GPU fino alla creazione di una pipeline per l'addestramento parallelo potenzialmente riusabile in altri progetti e altamente scalabile.

Come task finale è stata studiata la possibilità di usare una rete neurale profonda per riconoscere il successo o meno di un abito presso il pubblico a partire solo da una immagine dell'abito in questione. Questo ha permesso uno studio approfondito del concetto di dataset, considerazioni sul bilanciamento dello stesso e, più in generale, uno studio su come impiegare al meglio e selezionare la rete neurale più adatta allo scopo. Per questo motivo si è sperimentato con reti neurali per la classificazione e per la regressione al fine di trovare quale fosse la migliore soluzione al problema. Al termine della analisi è stato possibile studiare una

soluzione in grado di raggiungere i livelli di diverse reti esistenti e delineare i principali problemi e come migliorare in implementazioni future.

Per finire durante lo svolgimento di questo elaborato si è sperimentato e si è potuto acquisire molteplici competenze relativamente a deep learning, addestramento parallelo multi GPU, segmentazione di immagini e sulla libreria Pytorch.

Bibliografia

1: COCO, Common Objects in Context. Disponibile su: https://cocodataset.org/ [21/07/2022]

2: nVidia, "CUDA Zone", https://developer.nvidia.com/cuda-zone [21/07/2022]

3: Negar Rostamzadeh, Seyedarian Hosseini, Thomas Boquet, Wojciech Stokowiec, Ying Zhang, Christian Jauvin, Chris Pal, "Fashion-Gen: The Generative Fashion Dataset and Challenge", 21 Giugno 2018. Disponibile su: https://arxiv.org/pdf/1806.08317v2.pdf [21/07/2022]

4: Zalando, AA.VV. "Fashion-MNIST". Disponibile su: https://github.com/zalandoresearch/fashion-mnist [21/07/2022]

5: Ian Goodfellow, https://twitter.com/goodfellow_ian/status/852591106655043584 [21/07/2022]

6: François Chollet, https://twitter.com/fchollet/status/852594987527045120 [21/07/2022]

7: Geri Skenderi, Christian Joppi, Matteo Denitto, Marco Cristani, "Well Googled is Half Done: Multimodal Forecasting of New Fashion Product Sales with Image-based Google Trends", 20 Settembre 2021. Disponibile su: https://arxiv.org/pdf/2109.09824v4.pdf [21/07/2022]

8: Geri Skenderi, Christian Joppi, Matteo Denitto, Berniero Scarpa, Marco Cristani, "The multi-modal universe of fast-fashion: the Visuelle 2.0 benchmark", 14 Aprile 2022. Disponibile su: https://arxiv.org/pdf/2204.06972v1.pdf [21/07/2022]

9: Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, Xiaoou Tang, "Large-scale Fashion (DeepFashion) Database". Disponibile su: https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html [21/07/2022]

10: Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, Xiaoou Tang, "DeepFashion: Attribute Prediction". Disponibile su:

https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion/AttributePrediction.html [21/07/2022]

11: Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, Xiaoou Tang, "DeepFashion: Consumer-to-shop Clothes Retrieval". Disponibile su: https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion/Consumer2ShopRetrieval.html [21/07/2022]

12: Yuying Ge, Ruimao Zhang, Lingyun Wu, Xiaogang Wang, Xiaoou Tang, and Ping Luo, "DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images", 23 Gennaio 2019. Disponibile su: https://arxiv.org/pdf/1901.07973.pdf [21/07/2022]

13: switchablenorms, AA.VV. Disponibile su: https://github.com/switchablenorms/DeepFashion2/blob/master/evaluation/deepfashion2_to_coco.py [21/07/2022]

14: Menglin Jia, Mengyun Shi, Mikhail Sirotenko, Yin Cui, Claire Cardie, Bharath Hariharan, Hartwig Adam, Serge Belongie, "Fashionpedia: Ontology, Segmentation, and an Attribute Localization Dataset", 18 Luglio 2020. Disponibile su: https://arxiv.org/pdf/2004.12276.pdf [21/07/2022]

15: cvdfoundation, AA.VV. "Fashionpedia Dataset". Disponibile su: https://github.com/cvdfoundation/fashionpedia#global-attribute-prediction-task-attributes [21/07/2022]

16: KMnP, category attributes descriptions. Disponibile su: https://github.com/KMnP/fashionpedia-api/blob/master/data/demo/category_attributes_descriptions.json [21/07/2022]

17: Pytorch, Documentazione L1Loss. Disponibile su: https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html#torch.nn.L1Loss [21/07/2022]

18: Pytorch, Documentazione MSELoss. Disponibile su: https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html#torch.nn.MSELoss [21/07/2022]

19: Pytorch, Documentazione CrossEntropyLoss. Disponibile su: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss [21/07/2022]

20: Pytorch, Documentazione Linear. Disponibile su: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html [21/07/2022]

21: Pytorch, Documentazione Conv2d. Disponibile su: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html [21/07/2022]

22: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition", 27-30 Giugno 2016. Disponibile su: https://ieeexplore.ieee.org/document/7780459 [21/07/2022] e https://arxiv.org/pdf/1512.03385.pdf [21/07/2022]

23: Vincent Feng, "An Overview of ResNet and its Variants", 15 Luglio 2017. Disponibile su: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035?gi=d4944ef7ef54 [21/07/2022]

24: Pytorch, "Datasets & DataLoaders". Disponibile su: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html [21/07/2022]

25: Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", 6 Gennaio 2016. Disponibile su: https://arxiv.org/pdf/1506.01497.pdf [21/07/2022]

26: Ross Girshick, "Fast R-CNN", 27 Settembre 2015. Disponibile su: https://arxiv.org/pdf/1504.08083.pdf [21/07/2022]

27: Kaiming He, Georgia Gkioxari, Piotr Dolĺar, Ross Girshick, "Mask R-CNN", 24 Gennaio 2018. Disponibile su: https://arxiv.org/pdf/1703.06870.pdf [21/07/2022]

28: Glenn Jocher, "YOLOv5 Documentation". Disponibile su: https://docs.ultralytics.com/ [21/07/2022]

29: ultralytics, AA.VV, "YOLOv5". Disponibile su: https://github.com/ultralytics/yolov5 [21/07/2022]

30: Menglin Jia, Mengyun Shi, Mikhail Sirotenko, Yin Cui, Claire Cardie, Bharath Hariharan, Hartwig Adam, Serge Belongie, "Fashionpedia: Ontology, Segmentation, and an Attribute Localization Datase", 18 Luglio 2020. Disponibile su: https://arxiv.org/pdf/2004.12276.pdf [21/07/2022]

31: Marco Godi, Christian Joppi, Geri Skenderi, and Marco Cristan, "MovingFashion: a Benchmark for the Video-to-Shop Challenge". Disponibile su: https://arxiv.org/pdf/2110.02627.pdf [21/07/2022]

32: Pytorch, Documentazione DataParallel. Disponibile su: https://pytorch.org/docs/stable/generated/torch.nn.DataParallel.html [21/07/2022]

33: Pytorch, "Data Parallelism". Disponibile su: https://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html [21/07/2022]

34: Pytorch, Documentazione DistributedDataParallel. Disponibile su: https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html [21/07/2022]

35: Pytorch, Documentazione torch.utils.data. Disponibile su: https://pytorch.org/docs/stable/data.html [21/07/2022]

36: Python, Global Interpreter Lock. Disponibile su: https://wiki.python.org/moin/GlobalInterpreterLock [21/07/2022]

37: Pytorch, Documentazione torch.multiprocessing. Disponibile su: https://pytorch.org/docs/stable/multiprocessing.html [21/07/2022]

38: Python, "multiprocessing - Process-based parallelism". Disponibile su: https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing [21/07/2022]

39: Xin Fu, Jia Yan, Cien Fan, "IMAGE AESTHETICS ASSESSMENT USING COMPOSITE FEATURES FROM OFF-THE-SHELF DEEP MODELS", 22 Febbraio 2019. Disponibile su: https://arxiv.org/pdf/1902.08546.pdf [21/07/2022]

40: Katharina Schwarz, Patrick Wieschollek, Hendrik P. A. Lensch, "Will People Like Your Image? Learning the Aesthetic Space", 4 Dicembre 2017. Disponibile su: https://arxiv.org/pdf/1611.05203.pdf [21/07/2022]

41: Sebastian Raschka, “Machine Learning con Python”, Apogeo, 2016

42: Mark Summerfield, “Python, Programmazione avanzata”, Apogeo, 2014

43. Sheldon M. Ross, “Probabilità e statistica”, Maggioli Editore, 2015

Note

a: Convenzione francese, Decreto 8 Brumaio anno II. “Nessuna persona, dell’uno o dell’altro sesso, potrà costringere alcun cittadino o cittadina a vestirsi in modo particolare, sotto pena di essere considerata o trattata come sospetta o perturbatrice della pubblica quiete; ognuno è libero di portare gli abiti o gli accessori del suo sesso che preferisce”

b: Edgar Gregersen, The Story of Human Sexuality, F. Watts, 1983

c: Matteo Poggi, Fabio Tosi, and Pierluigi Zama Ramirez, “Deep Scene Understanding From Images”. Disponibile su: https://cvlab-unibo.github.io/deep_scene_understanding_from_images/ [21/07/2022]

d: Pytorch. Disponibile su: https://pytorch.org

&