UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Titolo della Tesi

RELATORE

Prof. Nome relatore

Dott. Nome tutor

Università degli Studi di Salerno

CANDIDATO

Antonio Cimino

Matricola: 0522500922

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

Dedica o citazione

Abstract

Oggi è molto diffuso l'utilizzo di sistemi che permettono di analizzare le immagini per trarne informazioni in tantissimi campi. Il presente lavoro di ricerca ha lo scopo di trovare una soluzione per riconoscere i capi di abbigliamento all'interno delle immagini. In letteratura esistono già diversi software che riescono a raggiungere tale scopo, il nostro intendo sarà sia ottenere dei risultati migliori, si testarne l'affidabilità anche al di fuori di un contesto sperimentale. A tale scopo, andremo a strutturare una rete neurale con cui generiamo dei modelli che possano soddisfare tale obiettivo e li metteremo alla prova in più modi. I risultati dell'analisi hanno dimostrato che i modelli che sono risultati dal nostro studio sono soddisfacenti a metà, perché riescono a riconoscere bene nel memento in cui gli elementi da classificare sono ben visibili, meno se le immagini hanno rumore cosa che nella realtà può accadere.

Indice

Elenco delle Figure **iii**

Elenco delle Tabelle **v**

1 Introduzione **1**

1.1 Contesto applicativo 1

1.2 Motivazioni e Obiettivi 2

1.3 Risultati 2

1.4 Struttura della tesi 2

2 Background e stato dell'arte **4**

2.1 Background 4

2.1.1 Classificazione delle immagini 4

2.1.2 Rete neurale e reti neurale profonda 5

2.1.3 Dataset (MMINST e Fashion MMINST) 8

2.2 Stato dell'arte 10

3 Metodologia proposta **16**

3.1 Struttura delle rete 17

3.2 Dataset 20

3.3 Data collection & analysis 21

3.3.1 Esperimento in-vitro 21
3.3.2 Esperimento in-vivo 25

4 Risultati **29**
4.1 Risultati iniziali 29
4.2 Risultati dei modelli con data agumentation 32
4.3 Risultati dei modelli con immagini filtrate 33
4.3.1 Immagini con filtro Blur 34
4.3.2 Immagini con filtro Contrast 36
4.3.3 Immagini con filtro Cut 37
4.3.4 Immagini con filtro Rain 39
4.3.5 Immagini con filtro Occlusion 41
4.3.6 Immagini con filtro Rotate 42
4.4 Risultati dei modelli con immagini colorate 44
4.5 Riassunto dei risultati 45

5 Conclusioni **48**
5.1 Interpretazione dei risultati 48
5.2 Limitazioni e sviluppi futuri 49

Bibliografia **50**

Elenco delle figure

2.1 Immagine che mostra come i neuroni x generano un risultato y . . . 6
2.2 Struttura di una rete neurale 6
2.3 Struttura di una rete neurale convoluzionale, immagine di [1] 7
2.4 Immagini presenti nel dataset di MNIST 9
2.5 Immagini presenti nel dataset F-MNIST 9
2.6 Struttura utilizzata nello studio [2] 11
2.7 Esempio di funzionamento della tecnica HOG preso da [3] 12
2.8 Grafici di accuratezza e loss function degli esperimenti nell'articolo [4] 14
2.9 Rappresentazione della rete utilizzata in [5] 14
2.10 Grafici di accuratezza degli esperimenti nell'artcolo [5] 15

3.1 Struttura rete costruita 17
3.2 Funzionamento del kernel 18
3.3 Funzionamento del MaxPooling 19
3.4 Funzionamento del Dropout, immagine di [6] 19
3.5 Funzionamento del livello di Flatten [7] 20
3.6 Struttura del dataset di MNIST, immagine presa da [8] 21
3.7 L'immagine di un vestito su cui è stata applicata la data augumentation 23
3.8 Esempio grafici di accuratezza e loss function degli esperimenti . . . 23
3.9 Esempio di matrice di confusione 24

3.10 L'immagine di una scarpa che è stata modificata con i filtri utilizzati per i test 25

4.1 Grafici di accuratezza e loss function del modello con id 11 30
4.2 Matrice di confusione generata dai test con il modello con id 11 . . . 31
4.3 Classi che hanno silhouette simili 32
4.4 Matrice di confusione del con id 5 33
4.5 Rappresentazioni di tutte le classi con il filtro blur 34
4.6 Matrice di confusione dei test con il filtro blur 35
4.7 Rappresentazioni di tutte le classi con il filtro contrast 36
4.8 Rappresentazioni di tutte le classi con il filtro contrast 37
4.9 Matrice di confusione dei test con il filtro cut 38
4.10 Rappresentazioni di tutte le classi con il filtro rain 39
4.11 Matrice di confusione dei test con il filtro rain 40
4.12 Rappresentazioni di tutte le classi con il filtro occlusion 41
4.13 Rappresentazioni di tutte le classi con il filtro rotate 42
4.14 Matrice di confusione con il filtro rotate 43
4.15 Rappresentazioni di tutte le classi con il filtro colored 44

Elenco delle tabelle

2.1 Tabella con con i migliori risultati di alcuni degli algoritmi studiati in [9] 12
2.2 Tabella con i risultati dei test delle reti convoluzionali costruite 13

4.1 Tabella con i risultati dei test delle reti convoluzionali costruite 30
4.2 Tabella con i risultati dei test con data augumentation 32
4.3 Tabella con i risultati dei test con filtro blur 35
4.4 Tabella con i risultati dei test con filtro contrast 36
4.5 Tabella con i risultati dei test con filtro cut 38
4.6 Tabella con i risultati dei test con filtro rain 40
4.7 Tabella con i risultati dei test con filtro occlusion 42
4.8 Tabella con i risultati dei test con filtro blur 43
4.9 Tabella con i risultati dei test con filtro color 45
4.10 Tabella riassuntiva dei risultati della ricerca 47

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

Il progetto svolto è incentrato sul problema del riconoscimento degli elementi nelle immagini, nel nostro caso più specificatamente di capi di abbagliamento. Il riconoscimento avviene attraverso la computer vision che e una branca dell'intelligenza artificiale (AI) che studia algoritmi e tecniche che consentono ai computer di replicare i processi e le funzioni dell'apparato visivo umano per rilevare e interpretare informazioni importanti in un'immagine digitale, un video o altri input visivi. Per poter funzionare e riuscire a interpretare i contenuti presente all'interno di un immagine, i sistemi di computer vision devono essere prima "addestrati" attraverso un processo di machine learning e utilizzando una grande quantità di immagini catalogate, le quali saranno la base del dataset che permetterà all'algoritmo di riconoscere quelle successive in modo intelligente. La specializzazione del nostro sistema a riconoscere capi d'abbigliamento nasce dai multipli utilizzi che può avere. Parliamo di elementi che utilizziamo tutti i giorni e oltre che ai semplici utilizzi commerciali, come ad esempio impedire repliche tra i vari marchi, o suggerire alle persone cose più vicine ai propri gusti, può essere utilizzato anche in altri contesti, come ad esempio se parliamo dell'ambito ecologico, per suddividere al meglio i prodotti, e cosi via.

1.2 Motivazioni e Obiettivi

L'obiettivo di tesi quindi è quello di costruire una rete che ci permetta di generare un modello per riconoscere i capi all'interno delle immagini. Ovviamente procederemo prima con il capire cosa troviamo attualmente in natura, e poi sulla base di quei lavori seguiremo cercando di migliorare o avvicinare quei risultati, con la costruzione di una nostra rete e successivo allenamento della rete. Ovviamente al termine del progetto noi vogliamo portare la nostra ricerca al di fuori del contesto sperimentale, quindi avremo anche lo scopo di capire anche come ciò che produciamo come funziona in situazioni reali.

1.3 Risultati

Dopo la costruzione della rete passeremo alla valutazione. Per valutare al meglio la qualità del sistema introdurremo dei parametri nel capitolo 3 che ci permetteranno di studiare come il software agisce per ogni classe, in modo da capire se funziona bene o se non funziona bene, e nel secondo caso capirne il perché. Suddivideremo la sperimentazione in due parti, testando il sistema prima in un contesto puramente sperimentale, quindi artificioso, poi andando a simulare la realtà attraverso l'ausilio di filtri che hanno l'obiettivo di simulare condizione atmosferiche o altre condizioni che influenzano la foto, per capire se il sistema si comporta bene anche al di fuori, e quindi applicabile in un contesto reale. Nel caso si volessero effettuare dei test il codice si può trovare su `https://github.com/AntonioCimino/Reconigtion_clothes_in_image`.

1.4 Struttura della tesi

L'elaborato si compone di quattro capitoli: Introduzione, Background e Stato dell'arte, Metodologia di ricerca, Risultati e Conclusioni. Nel primo capitolo parliamo dello scopo del nostro progetto, andando ad introdurre ciò che andremo a fare per raggiungere tale obiettivo. Nel secondo capitolo verrà spiegato cosa tratteremo a livello teorico, cercando di capire più nello specifico cosa abbiamo scelto di fare e

perché. Approfondiremo poi i lavori già presenti in natura che avevano uno scopo di base molto simile al nostro, analizzando pro e contro e traendo informazioni utili per il nostro studio. Nel terzo capitolo invece ci sarà scritto quello che effettivamente è stato fatto, quindi come abbiamo strutturato la rete, come abbiamo organizzato gli esperimenti e infine che tipi di metodi di valutazione abbiamo utilizzato poi per capire la riuscita o meno della ricerca. Nel quarto capitolo vengono mostrati tutti i risultati ottenuti. I risultati vengono mostrati andando a suddividere le varie fasi di ricerca che comprende prima la costruzione della rete, poi il miglioramento con i test corrispettivi e infine gli esperimenti su quelle che sono le immagini che dovrebbero simulare la realtà. Nel quinto e ultimo capitolo vi è un commento generale sull'analisi condotta. Verrà fatto un resoconto dei risultati più significativi cercando di capire se alla fine abbiamo ottenuto il sistema che fa fronte al nostro obiettivo iniziale.

CAPITOLO 2

Background e stato dell'arte

2.1 Background

2.1.1 Classificazione delle immagini

Negli ultimi anni si è registrato un costante aumento di nuovi algoritmi e tecniche di classificazione. Quindi, è nata la necessità di identificare la tipologia di classificazione appropriata per uno studio specifico. Partiamo dalla natura del campione che abbiamo a disposizione. Nel caso in cui abbiamo una conoscenza preliminare di come dobbiamo classificare gli elementi, allora stiamo parlando di una classificazione supervisionata. Il vantaggio principale della classificazione supervisionata è la maggiore facilità con cui si possono rilevare gli errori e correggerli. Gli svantaggi di questa tecnica sono i tempi e i costi. Inoltre, i dati di allenamento iniziali potrebbero non evidenziare tutte le caratteristiche per una buona classificazione e quindi si può essere soggetti a errori umani [10]. Mentre se non esiste una conoscenza preliminare allora possiamo lavorare con i classificatori non supervisionati. I vantaggi in questo caso sono la velocità, l'assenza di errori umani e non vi è alcun obbligo di conoscenza preventiva dettagliata. Il principale inconveniente di questa tecnica sono i cluster separabili al massimo [10]. Altro elemento di valutazione per la scelta del classificato-

re è dato del tipo di oggetto su cui andiamo a lavorare. Tecniche di classificazione delle immagini possono essenzialmente essere suddivisi in due categorie: la classificazione basata sui pixel e classificazione basato su oggetti. I classificatori per pixel sono i classificatori tradizionali, aiutano a combinare i valori di tutti i pixel del set di allenamento con una determinata funzione specificata. La combinazione risultante conterrà i contributi di tutti i pixel del training set e ignorerà i problemi dei pixel misti. Esempio: distanza minima, massima probabilità, macchina vettoriale di supporto, rete neurale artificiale, albero decisionale [10]. Mentre con i classificatori orientati agli oggetti, la segmentazione delle immagini fonde i pixel in oggetti. La classificazione viene effettuata sulla base di oggetti e non su singoli pixel. Esempio: eCognition [10]. La scelta varia anche in base al numero di output generati. In questo caso possiamo avere una classificazione hard in cui classifichiamo l'immagine, basandoci sul pixel, nelle varie categorie. Questi algoritmi aiutano nella categorizzazione di tutti i pixel in classi. Può causare un gran numero di errori dai dati di risoluzione spaziale grossolana a causa dei pixel misti. Esempio: massima probabilità, macchina vettoriale di supporto, ISODATA (Classificazione non supervisionata), parallelepipedo, centroide (k significa), rete neurale, albero decisionale [10]. C'è anche la classificazione soft che è stata proposta come alternativa alla classificazione hard per trattare con pixel misti. Esempio: classificazione della massima probabilità, classificatori fuzzy-set, classificatore sub pixel, analisi della miscela spettrale [10]. Da qui è facile capire che per la classificazione delle immagini dovremmo optare per una classificazione tramite reti neurali, alberi decisionali, e così via. In particolare ci concentreremo sulle reti neurali che tendono ad essere quelle più utilizzate in questi contesti, in particolare le reti neurali convoluzionali.

2.1.2 Rete neurale e reti neurale profonda

Ci sono un gran numero di pubblicazioni relative all'elaborazione delle immagini con reti neurali. Il nostro lavoro è legato a questo tipo di ricerca, dove la CNN viene utilizzata per classificare le immagini. Le reti neurali artificiali (ANN), sono sistemi informatici ideati dall'osservazione del cervello umano. Il cervello, infatti, è formato da neuroni che si scambiano segnali per comunicare tra di loro. Allo stesso modo, una

ANN si compone di nodi interconnessi. Ad ogni connessione è associato un valore numerico chiamato peso. Quando un nodo riceve l'input effettua una somma pesata, ovvero moltiplica ogni input per il peso della connessione su cui lo ha ricevuto e somma tutti i risultati. Al risultato di questa somma applica una funzione non lineare e restituisce un output. L'output rappresenta la predizione che viene fatta [Figura 2.1].

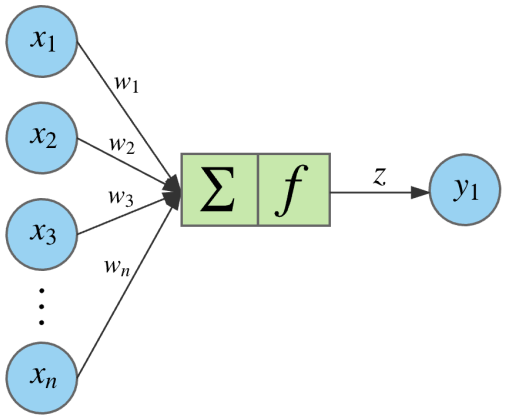

Figura 2.1: Immagine che mostra come i neuroni x generano un risultato y

I neuroni sono disposti su più strati connessi tra loro. Possiamo distinguere tre tipologie di strati [Figura 2.2]: abbiamo uno strato di input e uno di output rispettivamente posti all'inizio e alla fine, poi ci sono quelli che sono definiti strati nascosti, strati in cui viene effettuata la computazione descritta in precedenza.

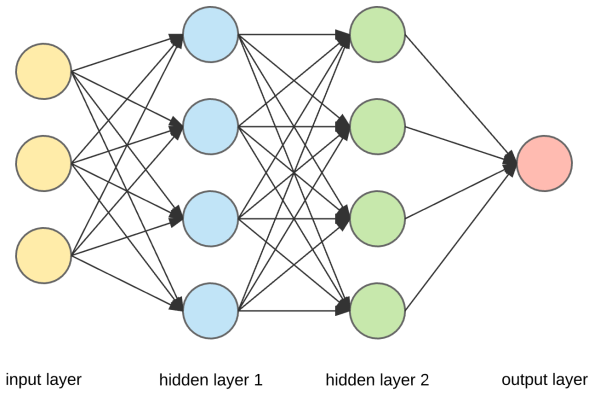

Figura 2.2: Struttura di una rete neurale

Una rete neurale artificiale viene addestrata usando un algoritmo chiamato backpropagation. Questo algoritmo cerca di minimizzare una funzione di perdita in modo

iterativo andando a calcolare quali sono i valori ottimali dei pesi delle connessioni, ovvero quali valori producono una predizione migliore. Quindi la logica dall'algoritmo funziona andando in avanti fino alla fine e ottenere una predizione, a questo punto la rete calcola l'errore dell'output utilizzando una funzione di perdita per paragonare l'output giusto rispetto a quello ottenuto e restituisce la misura dell'errore. Data la misura di errore inizia a tornare indietro calcolando per ogni strato il contributo all'errore. L'algoritmo modifica i pesi per minimizzare la funzione di perdita.

Le reti neurali convoluzionali (CNN) sono un'evoluzione dell'architettura delle ANN. Una delle differenze principali è che i neuroni che compongono gli strati all'interno della CNN sono costituiti da neuroni organizzati in tre dimensioni, la dimensione spaziale dell'input (altezza e larghezza) e la profondità [11]. Le CNN sono composte da tre tipi di livelli [Figura 2.3]. Questi sono livelli convoluzionali, livelli di pooling e livelli completamente connessi. Quando questi livelli sono impilati, si è formata un'architettura CNN [11].

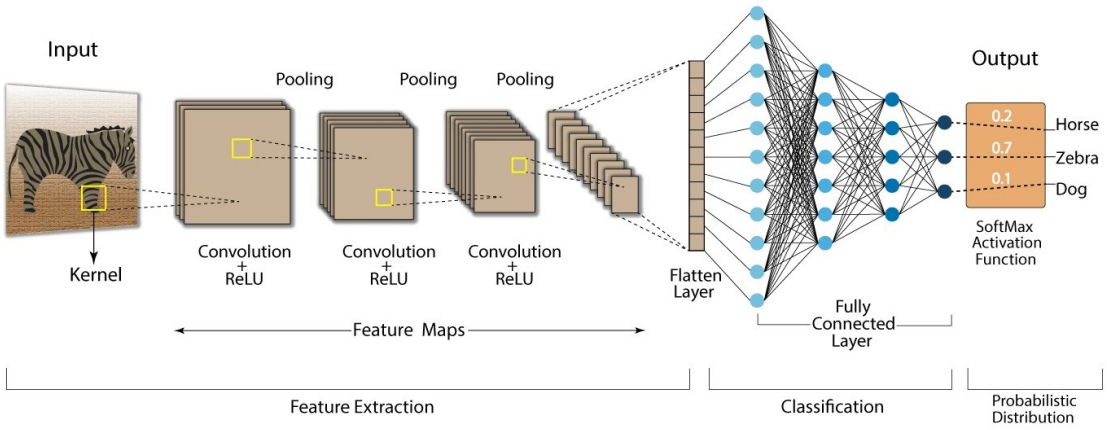

Figura 2.3: Struttura di una rete neurale convoluzionale, immagine di [1]

Gli strati convoluzionali sono la parte fondamentale delle CNN. In questi strati vengono definite delle matrici chiamate kernel o filtri. Questi kernel sono generalmente piccoli, ma si diffondono lungo la totalità dell'input. Quando i dati raggiungono un livello convoluzionali, lo strato fa ruotare ogni filtro sull'input producendo mappa di attivazione. Da questo la rete imparerà i kernel che si "attivano" quando vedono una caratteristica specifica in una data posizione spaziale dell'input. Questi sono co-

munemente note come attivazioni [11]. Attraverso quella caratteristica poi cambierà la predizione finale. Si prende in considerazione solo una parte dell'input alla volta grande quanto la grandezza del kernel visto che si tratta di una moltiplicazione tra matrici. I livelli di pooling mirano a ridurre gradualmente la dimensionalità della rappresentazione, riducendo così ulteriormente il numero di parametri e la complessità computazionale del modello. Il livello di pooling opera su ciascuna mappa di attivazione nell'input e ne ridimensiona la dimensionalità utilizzando la funzione "MAX". Nella maggior parte delle CNN, questi si presentano sotto forma di livelli di max pooling con kernel di una dimensionalità di 2×2 applicati con un passo di 2 lungo le dimensioni spaziali dell'input. Questo ridimensiona la mappa di attivazione fino al 25% della dimensione originale, pur mantenendo il volume di profondità alla sua dimensione standard [11]. Lo strato completamente connesso contiene neuroni di cui sono direttamente collegati ai neuroni nei due strati adiacenti, senza essere collegati ad alcuno strato al loro interno. Questo è analogo al modo in cui i neuroni sono disposti nelle forme tradizionali di ANN [11].

2.1.3 Dataset (MMINST e Fashion MMINST)

Il database MNIST [Figura 2.4] fornisce un compito di classificazione statica relativamente semplice per ricercatori e studenti per esplorare l'apprendimento automatico e le tecniche di riconoscimento dei modelli, risparmiando sforzi inutili sulla formattazione dei dati. I classificatori di rete neurali tendono a funzionare significativamente meglio di altri tipi di classificatori sul MNIST. In particolare, le CNN hanno eccellenti prestazioni di classificazione. Le prestazioni migliori con MNIST raggiungono un tasso di errore dello 0,27% circa raggiunto con un insieme di reti convoluzionali [12]. Ma anche una singola rete neurale convoluzione molto grande e profonda dà anche un tasso di errore basso, corrispondente a 0,35% [13]. Aumentare i dati di allenamento è importante per ottenere tassi di errore molto bassi [14]. Anche la profondità delle reti neurali aiuta ad avere bassi tassi di errore. Senza la struttura della convoluzione e le tecniche preprocessing, come l'aumento dei dati, il tasso di errore più basso in letteratura, 0,83%, è ottenuto utilizzando la rete neurale convessa/profonda [15].

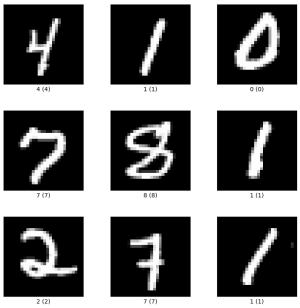

Figura 2.4: Immagini presenti nel dataset di MNIST

Il lavoro che vogliamo portare avanti si concentra sul settore dell'abbigliamento e quindi non lavoreremo su MNIST ma su un altro dataset denominato F-MNIST. L' obiettivo di F-MNIST è quello di avere un buon set di dati di riferimento che ha tutta l'accessibilità di MNIST, vale a dire le sue piccole dimensioni, semplice codifica e licenza permissiva [9]. Hanno adottato l'approccio di le 70, 000 immagini in scala di grigi nella dimensione di 28x28 con 10 classi, come nella MNIST originale [Figura 2.5]. Fashion-MNIST si basa sull'assortimento di vestiti del sito Zalando, dove ogni prodotto di moda ha un set di immagini, che mostrano diversi aspetti del prodotto. Per le etichette di classe, viene utilizzato il codice silhouette del prodotto. L'immagine originale ha uno sfondo grigio chiaro e memorizzato in formato 762x1000 JPEG. Per essere utilizzata in modo efficiente per i componenti del front-end, l'immagine originale viene campionata con risoluzioni multiple [9]. I prodotti provengono da diversi gruppi di genere: uomini, donne, bambini e neutri.

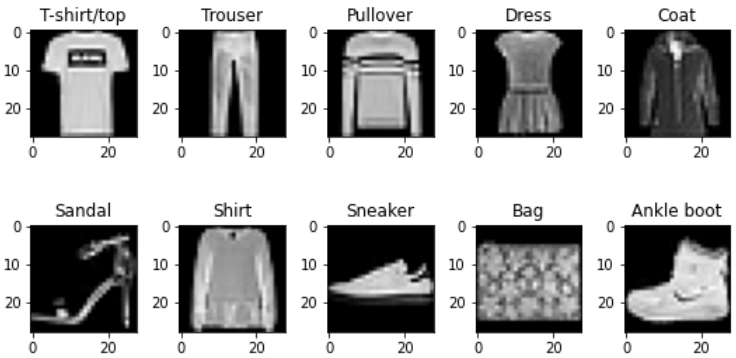

Figura 2.5: Immagini presenti nel dataset F-MNIST

Le miniature vengono convertite, per migliorare il train dei modelli che vogliamo costruire, seguendo il seguendo flusso di operazioni. Tale flusso è stato ripreso dall'articolo [9]:

1. Conversione dell'input in un'immagine PNG.
2. Rifila i bordi vicini al colore dei pixel degli angoli. La "vicinanza" è definita dalla distanza entro il 5% dell'intensità massima possibile nello spazio RGB.
3. Ridimensionando il bordo più lungo dell'immagine a 28 suddividendo i pixel, i.e. alcune righe e colonne vengono saltate.
4. Affilare i pixel usando un operatore gaussiano del raggio e della deviazione standard di 1.0, con effetto crescente vicino ai contorni.
5. Estendere il bordo più corto a 28 e mettere l'immagine al centro della tela.
6. Negando le intensità dell'immagine.
7. Conversione dell'immagine in pixel in scala di grigi a 8 bit.

Sono diversi gli studi effettuati sul dataset presentato con diversi metodi di classificazione. Nel prossimo pragrafo presentiamo dei lavori più nello specifico effettuati sul dataset F-MNIST.

2.2 Stato dell'arte

In questo paragrafo presenteremo alcuni lavori effettuati nell'ambito del riconoscimento dei capi nelle immagini. L. Bossard, M. Dantone, et al. [2] introducono una pipeline per riconoscere e classificare gli abiti in scene naturali. Hanno creato il proprio set di dati con 80000 immagini etichettate in 15 classi. Vengono utilizzati un classificatore Random Forest (per tipi di abbigliamento) e SVM (per attributi di abbigliamento) [Figura 2.6]. Sul loro set di dati, attraverso le Random Forest hanno prodotto un'accuratezza che si aggira intorno al 40% e addirittura meno con SVM. Come si può ben capire risultati poco soddisfacenti.

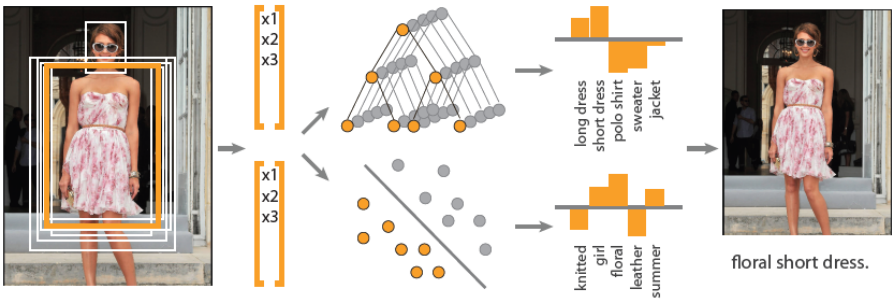

Figura 2.6: Struttura utilizzata nello studio [2]

Il limite di questo studio è dovuto principalmente al dataset che hanno definito loro, attraverso l'utilizzo di immagini prese da ImageNet, un dataset che contiene molto immagini di diverse categorie, anche abbigliamento. Il fatto di avere molte immagini non è per forza una buona cosa sopratutto se tali immagini spesso presentano rumori o contenuti non correlati, e questo non permette alla macchina di comprendere bene le caratteristiche per classificare le immagini. Questo nel nostro lavoro non dovrebbe essere un problema visto l'utilizzo di una dataset consolidato come F-MNIST.

Greeshma KV e Sreekumar K. al [3] propongono di generare un modollo con un addestramento attraverso un SVM, utilizzando la tecnica HOG per estrare le caratteristiche. HOG è un descrittore di funzionalità veloce ed efficiente rispetto, per esempio, a SIFT. Principalmente viene utilizzato per il rilevamento di oggetti nell'elaborazione di immagini e nella visione artificiale. HOG divide l'immagine in piccole celle e calcola le direzioni dei bordi. Nella [Figura 2.7] è mostrata la visualizzazione della dimensione della cella [2 2], [4 4] e [8 8]. Da ciò si comprende chiaramente che la dimensione della cella [2 2] contiene più informazioni sulla forma rispetto alla dimensione della cella di [8 8] nella loro visualizzazione. La cosa migliore, come viene fatto nel loro studio, è scegliere una via di mezzo, ovvere [4 4], perchè troppe infromazione vuol dire molto calcolo computazionale, poche informazioni invece non riesce a far comprendere alla macchina le caratteristiche per la classificazione. Ottenute le caretteristiche poi è stato addestrato un modello attraverso SVM. Attraverso questa combinazione di tecniche lo studio ci mostra che riusciamo ad arrivare ad una percentuale di accuratezza poco inferiore al 90%. Lo studio in questo caso è stato effettuato su F-MNIST rispetto all'articolo precedente e

notiamo un grosso miglioramento. In questo caso, invece, il limite sta nell'utilizzo del SVM come algoritmo di addestramento.

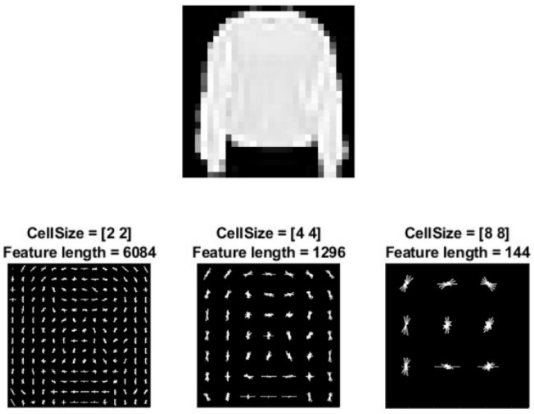

Figura 2.7: Esempio di funzionamento della tecnica HOG preso da [3]

Con SVM e altri algoritmi che non sono le reti neurali non si va oltre il 90%, e questo lo si può notare anche nell'articolo di Han Xiao, et al. [9] dove ci vengono mostrati i risultati prodotti con diversi algoritmi, così da poterli mettere a confronto tra essi e successivamente anche con risultati prodotti da reti neurali. Tutti gli algoritmi vengono ripetuti cinque volte mescolando i dati di allenamento. Vengono riportata la precisione media sul set di test di alcuni degli algoritmi nella [Tabella 2.1].

Nome classificatore	Parametri	Acc. media
SVC	C: 10, kernel: rbf	0.897
KNeighborsClassifie	weights: distance, n_neighbors: 5, p: 1	0.854
GradientBoostingClassifier	n_estimators: 100, loss: deviance, max_depth: 10	0.880
DecisionTreeClassifier	criterion: entropy, max_depth:10, splitter: best	0.798
RandomForestClassifier	criterion:entropy, max_depth: 100, n_estimators: 100	0.873
LogisticRegression	C: 1, multi_class: ovr, penalty: l1	0.842

Tabella 2.1: Tabella con con i migliori risultati di alcuni degli algoritmi studiati in [9]

Cerchiamo, quindi, ora di vedere anche studi che utilizzano le reti convoluzionali, per fare il confronto. Shobhit Bhatnagar, et al. [4] utilizzano un modello, con 2 livelli di pooling convoluzionale e massimo uno dopo l'altro. Ogni strato convoluzionale ha 32 filtri di dimensioni 3x3 ed è stato eseguito il max-pooling. Successivamente l'output è stato appiattito e inserito in una rete la classificazione finale. Usano anche il dropout, prima dell'ultimo livello denso, settato al 50% come misura di regolarizzazione, per evitare che il modello si adatti molto ai dati di addestramento e generalizzi male per i dati di test.

Precisione del test (%)	Modelli
CNN2	91.17
CNN2 + BatchNorm	92.22
CNN2 + BatchNorm + Skip	92.54

Tabella 2.2: Tabella con i risultati dei test delle reti convoluzionali costruite

A questo modello poi è stato aggiunto in un secondo test la normalizzazione batch eseguita prima di ogni strato convoluzionale per migliorare la velocità di addestramento del modello. Infine in un terzo modello vengono utilizzati anche le Skip connection che, come suggerisce il nome, servono a fornire un percorso alternativo per il gradiente con backpropagation. Nella [Tabella 2.2] possiamo osservare i risultati dei test effettuati attraverso l'utilizzo delle reti precedentemente descritte, e sono effettivamente risultati migliori rispetto a quelli degli studi precedenti, cosa che sta a dimostrare che con delle reti neurali convoluzionali si possono ottenere dei risultati migliori. Per il nostro lavoro cercheremo di costruire una rete neurale molto simile a quella dell'articolo [4] cercando però di migliorare i risultati, modificando la rete e cercando di lavorare sui parametri di allenamento. In [Figura 2.8] si possono osservare le tabella che evidenziano l'accuratezza e e la loss function. In natura esistono anche reti già consolidate che hanno dato dei risultati molto preformanti. Mohammed Kayed, et al [5] utilizzano l'architettura LeNet-5 per la classificazione delle immagini Fashion- MNIST.

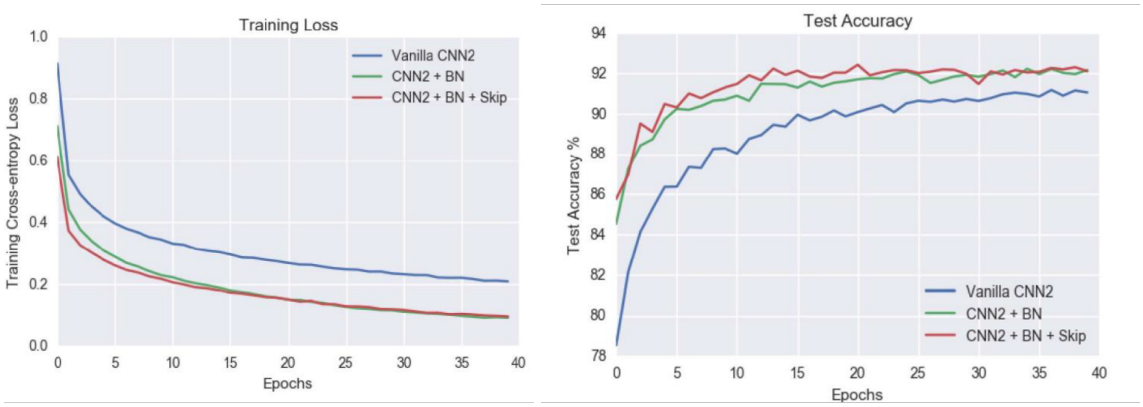

Figura 2.8: Grafici di accuratezza e loss function degli esperimenti nell'articolo [4]

La utilizzano perché è semplice e fornisce risultati ad alte prestazioni. Si basa su campi recettivi locali, pesi condivisi e un sotto-campionamento speciale. Nella [Figura 2.9] viene mostrata la struttura della rete, di cui vengono spiegati bene i vari strati cosa fanno nell'articolo. Per l'addestramento vengono utilizzati i seguenti iper-parametri: a = 0,005, batch size = 32.

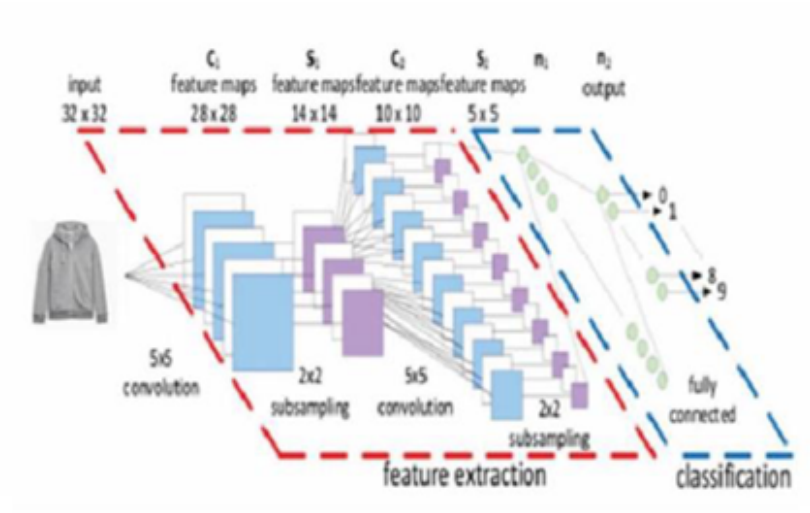

Figura 2.9: Rappresentazione della rete utilizzata in [5]

La [Figura 2.10] illustra l'accuratezza delle diverse suddivisioni di training e testing per le 10 categorie. Confrontando il grafico con quelle precedenti [Figura 2.8] possiamo notare una crescita più costante, il che già ci fa presumere un maggior successo. La media dell'accuratezza su tutte le 10 prove è stata del 98,9%.

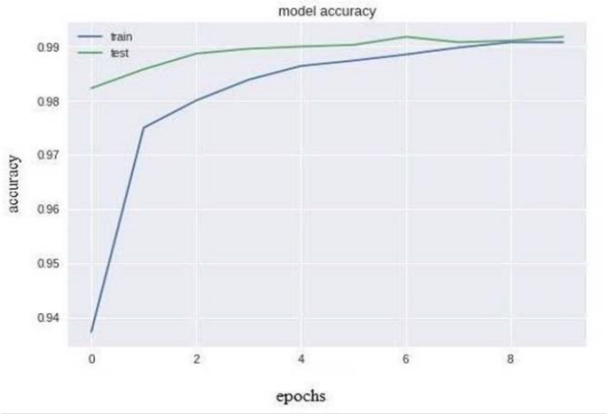

Figura 2.10: Grafici di accuratezza degli esperimenti nell'artcolo [5]

Sull'articolo vengono confrontati il loro modello LeNet-5 con gli altri modelli testati sul dataset Fashion MNIST (SVC, EDEN, e altri) e LeNet-5 raggiunge un'accuratezza superiore rispetto agli classificatori.

CAPITOLO 3

Metodologia proposta

Lo scopo di questo progetto di tesi è quello di definire una rete che ci permetta di generare un modello per riconoscere, al meglio possibile, i capi di abbigliamento nelle immagini, per poi riportarlo in un cotesto reale. Per valutare, alla fine, se il progetto ha portato al raggiungimento dell'obiettivo effettueremo due tipi di esperimenti sui modelli generati, uno per generalizzare il modello e un altro per capire se questo possa funzionare in una situazione più realistica. Quindi definiamo due interrogativi, uno per ogni tipologia di esperimento:

1. Riusciamo ad ottenere un modello che riesce a classificare bene i capi presenti nelle immagini del dataset di partenza?

2. Il modello come si comporta in una situazione più realistica?

Entrambe le domande fanno riferimento alla valutazione del modello, quindi per entrambe verranno utilizzate le stesse metriche di valutazione, ma con la differenza dei dati con cui i modelli produrranno i valori di queste metriche. Per realizzare l'obiettivo dell'esperimento, è stata costruita una rete neurale convoluzionale con l'ausilio della libreria Keras. L'allenamento del modello è stato effettuato su Colab, uno strumento messo a disposizione da Google per utilizzare GPU da remoto. Il modello utilizzato è unico, e nel prossimo capitolo cercheremo di spiegarne la

struttura. Abbiamo effettuato diversi allenamenti cambiando il valore dei parametri di addestramento e successivamente con l'ausilio di una tecnica per aumentare il volume dei dati.

3.1 Struttura delle rete

La struttura della rete la possiamo osservare nella [Figura 3.1] e ora passiamo a spiegare la sua composizione e la spiegazione di alcune scelte effettuate.

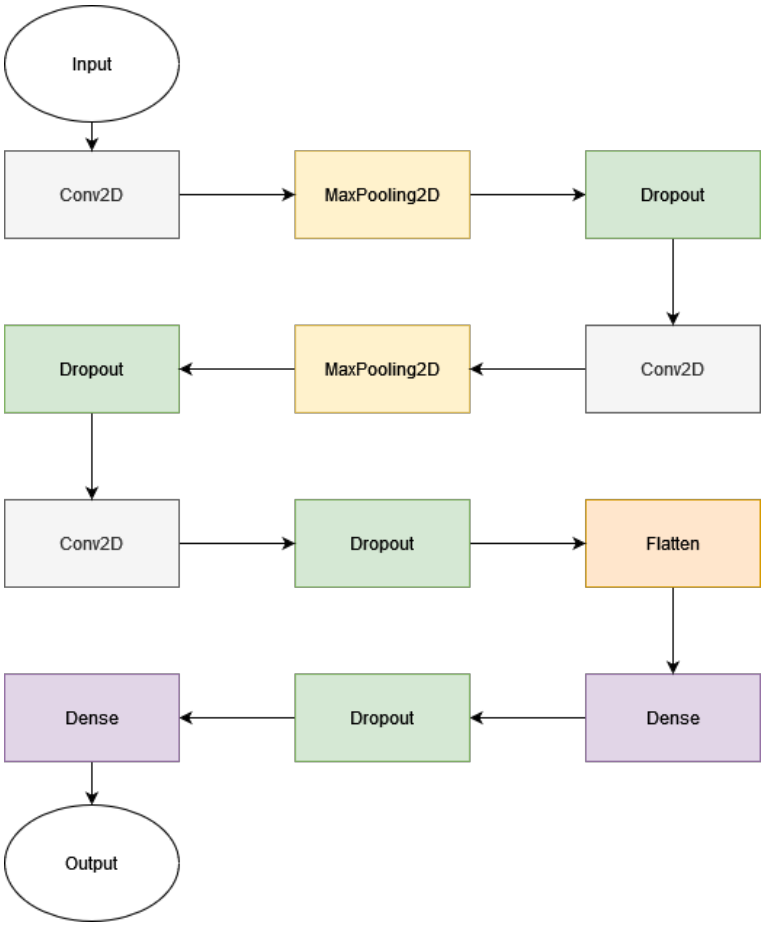

Figura 3.1: Struttura rete costruita

Conv2D è un livello di convoluzione 2D. Questo livello crea un kernel(o filtro) che è una matrice [Figura 3.2] di convoluzione utilizzata per modificare l'immagine in vari modi (sfocare, aumentare la nitidezza, rilievo, rilevamento dei bordi e altro). Per questo livello abbiamo specificato un numero di kernel uguale a 32 nel primo livello di Conv, poi siamo andati a crescere con 64 e 128 con una dimensione ciascuno di

3x3 come [Figura 3.2]. E' stato poi specificato anche la dimensione dell'input, ovvero quanto deve essere grande l'immagine che andiamo a passare per l'addestramento e successivamente per i test, e la dimensione indicata è di 28x28. Altro paramentro importante per il livello convoluzionale è "activation", consente di fornire una stringa che specifica il nome della funzione che si desidera applicare dopo aver eseguito la convoluzione, in questo caso applica la funzione ReLu per quanto riguarda i livelli interni della reta e la funzione SoftMax per quanto riguarda la classificazione finale [16]. SoftMax e ReLu sono le funzioni più comunemente utilizzate e che garantiscono migliori risultati. Infine abbiamo specificato il modo con cui vengono impostati i pesi inizialmente, nel nostro caso utilizziamo una funzione normale. MaxPooling è un processo di discretizzazione, ovvero processo di trasformazione dei modelli matematici basati su equazioni continue in modelli matematici basati su equazioni discrete.

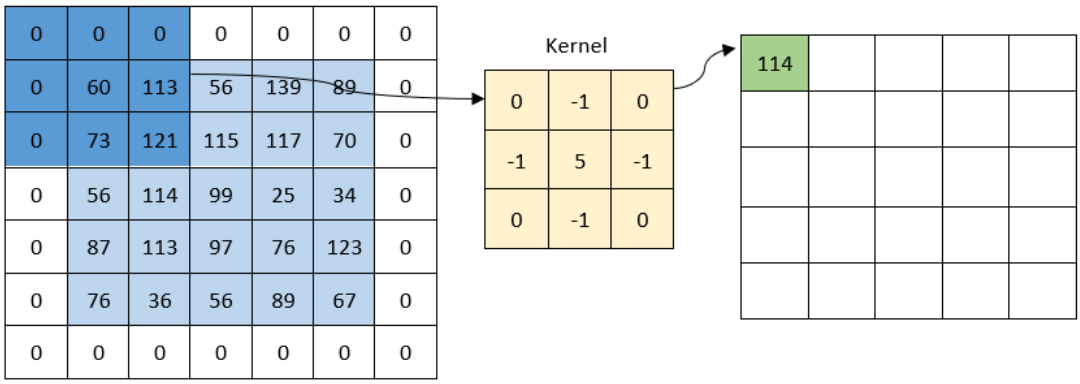

Figura 3.2: Funzionamento del kernel

L'obiettivo è di sotto campionare l'input, riducendone la dimensionalità e consentendo di formulare ipotesi sulle caratteristiche contenute nelle sotto-regioni. Ciò viene fatto in parte per aiutare l'overfitting fornendo una forma astratta della rappresentazione. Inoltre, riduce il costo computazionale riducendo il numero di parametri da apprendere. Il raggruppamento massimo viene eseguito applicando un filtro massimo a sotto-regioni. Nel nostro caso abbiamo un filtro 2x2 che eseguiremo sul nostro input. Per ogni area 2x2 su cui il filtro passerà verrà preso il massimo di quella regione e creeremo una nuova matrice di output in cui ogni elemento è il massimo

di una regione nell'input originale [Figura 3.3]. Il dropout è una tecnica di regolarizzazione in cui i neuroni selezionati casualmente vengono "abbandonati" [Figura 3.4]. I pesi dei neuroni durante l'allenamento vengono sintonizzati per caratteristiche specifiche, fornendo una certa specializzazione.

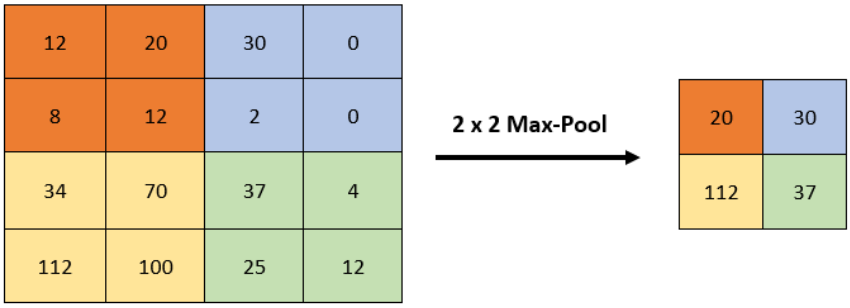

Figura 3.3: Funzionamento del MaxPooling

Se i neuroni vengono eliminati casualmente dalla rete durante l'allenamento, altri neuroni dovranno intervenire e gestire la rappresentazione richiesta per far fronte alla mancanza. Si ritiene che ciò comporti l'apprendimento da parte della rete di più rappresentazioni interne indipendenti. Il dropout aiuta a evitare l'overfitting, come spiegato in [17].

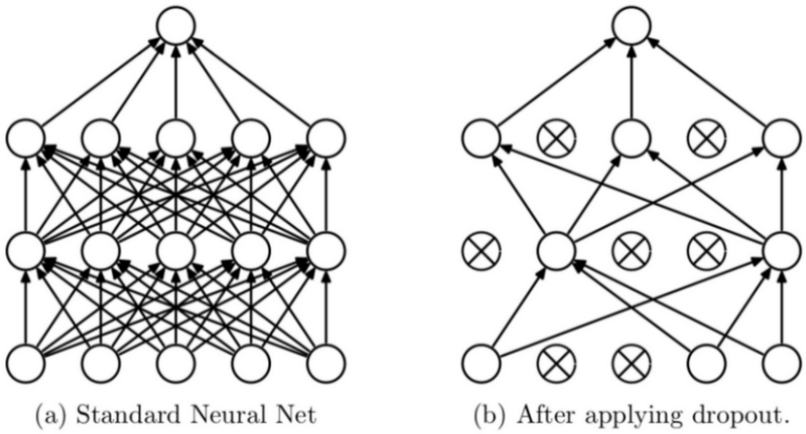

Figura 3.4: Funzionamento del Dropout, immagine di [6]

Il livello di Flatten serve ad appiattire gli input, ovvero serve a trasformare una matrice ad un array da passare al prossimo livello [Figura 3.5]. Stiamo creando un modello di classificazione, il che significa che questi dati elaborati dovrebbero essere un buon input per il modello. Deve avere la forma di un vettore lineare

unidimensionale e non una forma rettangolare o cubica che non rappresentano degli input diretti. Appiattiamo l'output dei livelli convoluzionali per creare un unico vettore di feature.

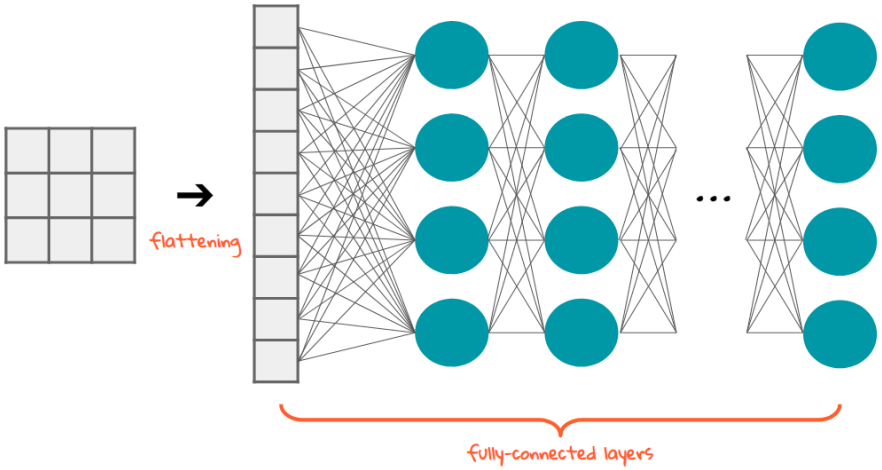

Figura 3.5: Funzionamento del livello di Flatten [7]

Questo livello poi è collegato al modello di classificazione finale, chiamato livello completamente connesso [Figura 3.5]. Infine per addestrare un modello, è necessario specificare una funzione di perdita, un ottimizzatore, ed eventualmente, alcune metriche da monitorare. Per quanto riguarda l'ottimizzatore è stato scelto Adam che è ritenuto uno dei migliori per le CNN. Adam è un ottimizzatore di discesa del gradiente stocastico che lavora su stime adattive. La discesa del gradiente è utile per regolare i pesi nei livelli nascosti.

3.2 Dataset

Il dataset utilizzato per effettuare questo lavoro di ricerca è Fashion-MNIST [18]. Questo dataset è stato scelto per via della sua validità e semplicità, questo dovuto al fatto che sia figlio di uno dei dataset più utilizzati, ovvero MNIST. Quest'ultimo è una raccolta di cifre scritte a mano e contiene 70000 immagini in scala di grigi 28x28, associate a 10 etichette. Fashion-MNIST ha la stessa identica struttura, ma le immagini sono prodotti di moda. Un esempio di questo set può essere visto nella [Figura 3.6]

Figura 3.6: Struttura del dataset di MNIST, immagine presa da [8]

Il set di dati può essere descritto come un CSV da 785 colonne. Ogni riga del CSV è un'immagine che ha una colonna con l'etichetta e le colonne rimanenti per descrivere l'immagine 28x28 pixel, con valori da 0 a 255 che rappresentano la luminosità dei pixel.

3.3 Data collection & analysis

Dopo aver specificato la rete e il dataset con cui andremo a lavorare dobbiamo distinguere le operazione che effettuiamo per dare risposta a quelle che sono le domande che ci siamo posti ad inizio capitolo, quindi andremo a spiegare prima ciò che facciamo per l'esperimento in-vitro, ovvero quello che generalizza i risultati e poi successivamente per l'esperimento in-vivo, ovvero quello che dovrebbe simulare i problemi che possono avere le immagini nella vita reale.

3.3.1 Esperimento in-vitro

Per rispondere alla prima domanda abbiamo effettuato dei train con il set di dati con cui abbiamo parlato prima. Sul dataset non è stato fatto molto lavoro perché di per sé già è sufficientemente consistente a livello qualitativo. Inizialmente la suddivisone dei dati tra set di train e test è 85/15 su 70000 immagini che compongono F-MNIST. Più nello specifico:

- T-shirt/top: Train 6000 or 10.0% and Test 1000 or 10.0%

- Trouser: Train 6000 or 10.0% and Test 1000 or 10.0%
- Pullover: Train 6000 or 10.0% and Test 1000 or 10.0%
- Dress: Train 6000 or 10.0% and Test 1000 or 10.0%
- Coat: Train 6000 or 10.0% and Test 1000 or 10.0%
- Sandal: Train 6000 or 10.0% and Test 1000 or 10.0%
- Shirt: Train 6000 or 10.0% and Test 1000 or 10.0%
- Sneaker: Train 6000 or 10.0% and Test 1000 or 10.0%
- Bag: Train 6000 or 10.0% and Test 1000 or 10.0%
- Ankle boot: Train 6000 or 10.0% and Test 1000 or 10.0%

Successivamente abbiamo applicato la data augumentation, è una tecnica che ci permette di aumentare il numero dei dati a nostra disposizione, andando a modificare sulle immagini che già abbiamo con dei filtri per modificare il colore, ridimensionandole, spostare, ecc.. Nel nostro caso abbiamo mosso gli elementi nelle immagini come possiamo vedere in [Figura 3.7]. Questa tecnica serve ad evitare l'overfitting, ovvero se il modello tende ad avere a disposizione poche informazione per la classificazione e quindi non riesce a distinguere bene gli oggetti. Nel nostro caso avevamo già a disposizione una buona quantità di dati ma sembra che grazie all'utilizzo di questa tecnica c'è stato un miglioramento sul risultato finale. L'idea è nata dal fatto che molti errori erano generati dalla somiglianza che esisteva tra alcuni prodotti (T-shirt e Shirt) che probabilmente venivano confusi dalla macchina e quindi si è provato ad aumentare la quantità di dati, ed effettivamente è migliorata l'accuratezza sul singolo prodotto. Con questa tecnica applicata sul set di train siamo arrivati a 300000 immagini per l'allenamento di cui poi un 20% abbiamo utilizzato per la validazione. Sono stati effettuati diversi allenamenti a partire dalla rete che abbiamo descritto precedentemente generando diversi modelli. Gli allenamenti si distinguevano in base alla modifica del batch size, ovvero il numero di campioni che verranno propagati attraverso la rete, e il numero delle epoche, che definisce il numero di volte in cui l'algoritmo di apprendimento funzionerà sull'intero set di dati di addestramento.

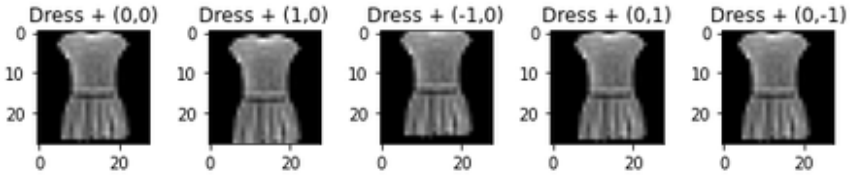

Figura 3.7: L'immagine di un vestito su cui è stata applicata la data augumentation

Quello che ci interessa a noi è rispondere alle domande che abbiamo definito all'inizio e la prima delle due ci chiedeva se riuscivamo a produrre un modello efficiente, e per valutare questa cosa dobbiamo specificare delle metriche. Innanzi tutto per ogni modello sono stati generati alla fine del train due grafici, di accuratezza e di perdita [Figura 3.8]

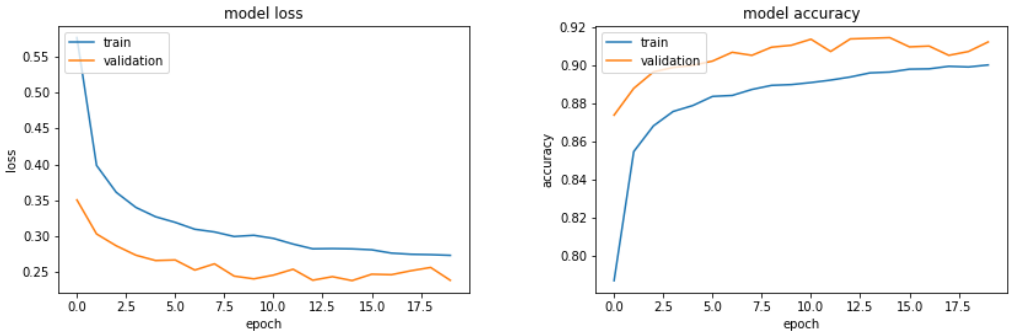

Figura 3.8: Esempio grafici di accuratezza e loss function degli esperimenti

Entrambi questi grafici riportano due dati al loro interno, rispettivamente accuratezza durante l'addestramento e "loss", con cui si intende la correzione effettuata sui valori che permettono la classificazione affinché l'output durante il train risultasse corretta, in pratica rappresenta il feedback della rete per correggere i pesi. La nostra analisi deve concentrarsi sul fatto che nel grafico di loss vogliamo che la curva tende verso il basso e per l'accuratezza invece verso l'alto. Il modello generato poi deve essere valutato anche in base a come si comporta sui dati di test che abbiamo messo da parte, al fine di accertarci della qualità del train.
Per ognuno dei modelli quindi viene effettuato un test con gli stessi dati in modo da capire ognuno di essi come si comporta con le stesso tipo di informazioni. Dai risultati dei test vengono calcolati dei parametri con cui poi potremmo fare delle osservazioni. I parametri sono raccolti rispettivamente per ogni singola classe e anche

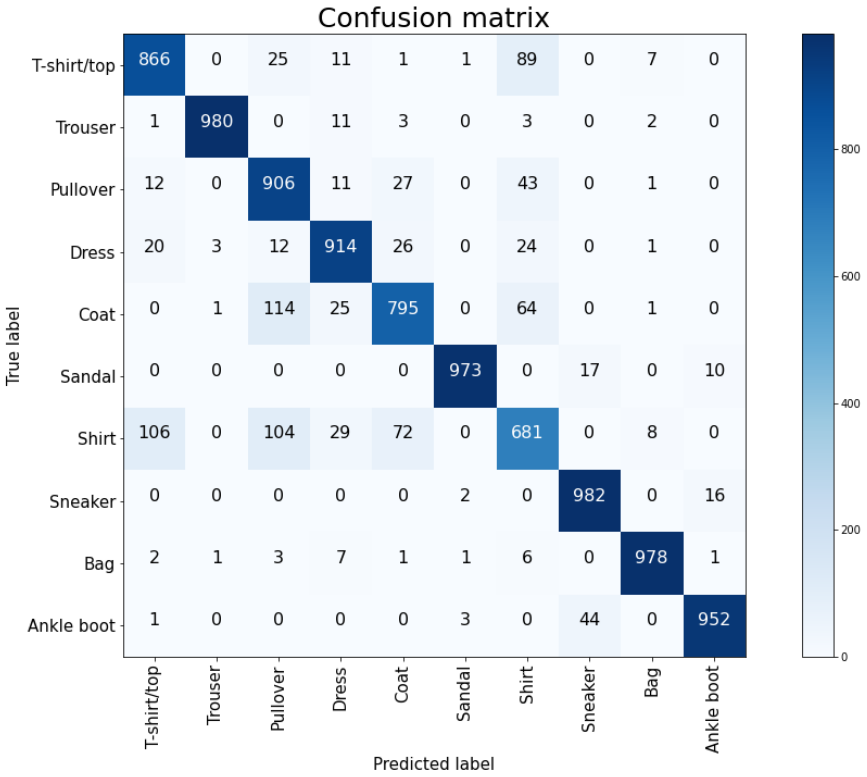

Figura 3.9: Esempio di matrice di confusione

una media e sono:

- Recall che è la capacità di un classificatore di trovare tutte le istanze positive. Per ogni classe è definito come il rapporto tra i veri positivi e la somma dei veri positivi e dei falsi negativi.
- Precision che è l'abilità di un classificatore di non etichettare un'istanza positiva che è in realtà negativa. Per ogni classe è definito come il rapporto tra veri positivi e la somma di veri e falsi positivi.
- F-score è una media armonica ponderata delle metriche Precision e Recall in modo tale che il punteggio migliore sia 1 e il peggiore sia 0.
- Accuracy indica l'accuratezza del modello come dice il nome. Pertanto, la migliore accuratezza è 1, mentre la peggiore è 0.

In più sui risultati andiamo anche a produrre la matrice di confusione come in [Figura 3.9] che ci aiuta a vedere il numero delle previsioni giuste e quelle sbagliate, ed è

utile sopratutto per capire in che modo sbaglia, cioè che tipo di classe individua al posto di quella giusta.

3.3.2 Esperimento in-vivo

Per rispondere invece al secondo quesito, ovvero come si comportano i modelli generati nella vita reali, abbiamo deciso di simulare la realtà. L'idea è stata quella di lavorare sulle immagini di test in modo che acquisissero un aspetto molto più realistico. Con aspetto più realistico si intende un'immagine non pulita, ovvero dove non si vede perfettamente il capo di abbigliamento. Questo tipo di sperimentazione vine fatto anche in [19], dove cercano di valutare il comportamento del software in presenza di situazioni speciali. L'intento è quello di provare a replicare condizioni atmosferiche reali, come per esempio quando c'è nebbia, con il "blur", oppure quando piove con un filtro "rain", o anche quando l'immagine non è totalmente visibile con un filtro che "occlude" l'immagine, ecc..

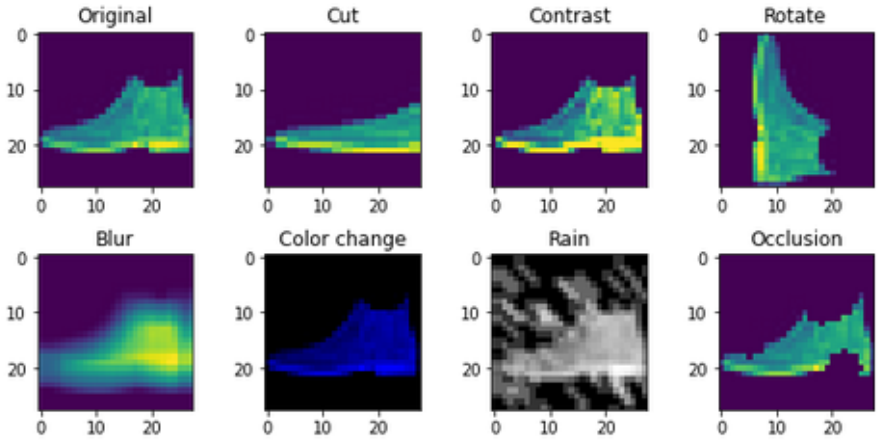

Figura 3.10: L'immagine di una scarpa che è stata modificata con i filtri utilizzati per i test

Da [19] infatti viene ripresa l'idea di quali tipi di filtri da poter implementare che vediamo in [Figura 3.10], aggiungendone qualcuno in più come il "cut". Quello che abbiamo fatto è prendere le immagini di F-MNIST, le 10000 di test, e andare ad applicare diversi tipi di filtri come si può vedere in [Figura 3.10]. Per ogni funzione filtro, di cui inserisco gli script alla fine del paragrafo, nel caso si volessero ripetere i test, abbiamo definito un nuovo set di dati formato dalle 10000 immagini del set di test modificate dal filtro stesso. Quindi, per esempio, è stato creato un set di dati,

che chiameremo "set di immagini tagliate", in cui abbiamo inserito le 10000 foto che sono state lavorare con il filtro "cut_filter", che come indica il nome serve a tagliare l'immagine. Ottenuti i 7 nuovi set di dati, uno per ogni funzione filtro, andiamo a valutare nuovamente i modelli, però stavolta con le immagini modificate, con lo scopo di capire come si comportano nella vita reale, e quindi dare una risposta al secondo quesito. Come per il primo tipo di esperimento anche qui andremo a valutare attraverso le stesse metriche specificate in precedenza (Recall, Accuracy, ecc.). Vedremo che però anche se i modelli e i dati di test sono gli stessi, con qualche modifica sui dati di test a seconda del filtro, i risultati cambiano, anche di molto in alcuni casi, e cercheremo di capire sopratutto grazie all'ausilio della matrice di confusione cosa succede sulle singole classi.

```
#Metodo per ruotare l'immagine
def apply_rotate(input_img):
  buf = cv2.rotate(input_img, cv2.ROTATE_90_CLOCKWISE)
  return buf

#Metodo per aumentare o diminuire il contrasto sull'immagine
def apply_contrast(input_img, contrast):
  f = 131 * (contrast + 127)/(127*(131-contrast))
  alpha_c = f
  gamma_c = 127*(1-f)
  buf = cv2.addWeighted(input_img, alpha_c, input_img, 0, gamma_c)
  return buf

#Metodo per modificare colore sull'immagine
def color_grayscale_arr(arr):
  dtype = arr.dtype
  h = arr.shape[0]
  w = arr.shape[1]
  arr = np.reshape(arr, [h, w, 1])

  x = np.random.randint(0,3)
  if x == 0:
    #red
    arr = np.concatenate([arr, np.zeros((h, w, 2), dtype=dtype)], axis=2)
  if x == 1:
    #green
    arr = np.concatenate([np.zeros((h, w, 1), dtype=dtype), arr,
    np.zeros((h, w, 1), dtype=dtype)], axis=2)
  if x == 2:
    #blue
```

```
31      arr = np.concatenate([np.zeros((h, w, 2), dtype=dtype),arr], axis=2)
32    return arr
33
34  #Metodo per inserire la pioggia sull'immagine
35  def add_rain(image):
36      imshape = image.shape
37      drop_color = (180,180,180)
38      bending = 3
39      drop_length = 3
40      rain_drops = []
41
42      #Definiamo le posizioni delle linee
43      for i in range(50):
44        x = np.random.randint(0,imshape[1])
45        y = np.random.randint(0,imshape[0])
46        rain_drops.append((x,y))
47
48      #Disegniamo le line (pioggia)
49      for drop in rain_drops:
50          cv2.line(image,(drop[0],drop[1]), (drop[0] + bending, drop[1]
51          drop_length),drop_color,1)
52
53      #Aggiungiamo un pò di blur per effetto foschia durante la pioggia
54      image = cv2.blur(image,(2,2))
55      return image
56
57  #Metodo per creare delle occlusioni sull'immagine
58  def add_occlusion(image):
59      imshape = image.shape
60      drop_color = (0,0,0)
61      occlusion_drops = []
62
63      #Definisco la posizione che avranno le occlusioni
64      for i in range(10):
65        x = np.random.randint(0,imshape[1])
66        y = np.random.randint(0, imshape[0])
67        occlusion_drops.append((x,y))
68
69      #Disegno le occlusioni
70      for drop in occlusion_drops:
71          cv2.circle(image,(drop[0],drop[1]),3,drop_color,-1)
72
73      return image
74
75  #Metodo per tagliare l'immagine
76  def cut_filter(img):
77      out = img[0:int(img.shape[1]), 0:int(img.shape[0]/2)]
```

```
78      out = cv2.resize(out, (int(img.shape[1]),
79      int(img.shape[0])), interpolation = cv2.INTER_AREA)
80      return out
81
82  #Metodo per sfocare l'immagine
83  def blur_filter(img):
84      ksize = (6, 6)
85      out = cv2.blur(img, ksize)
86      return out
```

CAPITOLO 4

Risultati

In questo capitolo mostriamo i risultati per ogni test effettuato sui vari modelli, andando a visualizzare e commentare i grafici e i parametri che sono stati utilizzati per la valutazione, di cui abbiamo parlato nel capitolo precedentemente.

4.1 Risultati iniziali

Inizialmente i primi modelli generati sono stati allenati con il dataset originale, senza l'ausilio di data augumentation, quindi sulla base delle 60000 immagini per l'allenamento, che poi abbiamo diviso con 85/15 tra dataset del train e validation. Ogni modello poi generato è stato testato con il dataset di test e nella [Tabella 4.1] vengono mostrati i risultati generali. Come possiamo notare in [Tabella 4.1] i risultati tendono ad avere tutti un valore superiore al 90% il che indica un ottimo risultato della rete. Con un valore medio sul 90% questo ci indica che quando prevede un elemento in una classe mediamente lo fa bene. La recall con percentuali medie sul 90% ci indica che tutte le classi sono utilizzate per classificare in modo proporzionale. Ovviamente studiare risultati medi non ci da la possibilità di comprendere al meglio il comportamento della rete sulle singole classi, e infatti sono stati raccolti anche i risultati sulle singole classi per ogni test effettuato. Siccome i valori dei risultati nella

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	20	16	0.90	0.90	0.90	0.90
1	20	32	0.91	0.91	0.91	0.91
2	20	64	0.91	0.91	0.91	0.91
3	20	128	0.91	0.91	0.91	0.91
4	25	16	0.91	0.90	0.90	0.90
5	25	32	0.91	0.91	0.91	0.91
6	25	64	0.91	0.91	0.91	0.91
7	25	128	0.91	0.91	0.91	0.91
8	30	16	0.90	0.90	0.90	0.90
9	30	32	0.91	0.91	0.91	0.91
10	30	64	0.91	0.91	0.91	0.91
11	30	128	0.92	0.92	0.92	0.92

Tabella 4.1: Tabella con i risultati dei test delle reti convoluzionali costruite

[Tabella 4.1] si somigliano andremo ad approfondire i singoli risultati sul test che ha restituito i valori maggiori, ovvero il modello indicato con id 11. Partiamo col mostrare i grafici di accuratezza e di loss del modello preso in considerazione [Figura 4.1].

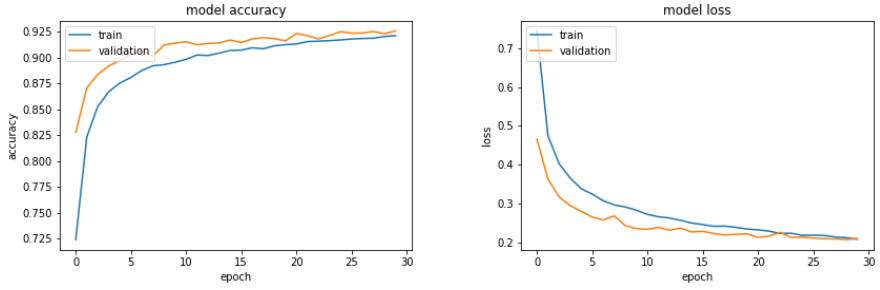

Figura 4.1: Grafici di accuratezza e loss function del modello con id 11

I due grafici ci mostrano, come visto anche dai risultati visti in [Tabella 4.1], che il modello funziona bene, anche perché in una analisi dei grafici noi vogliamo sempre che quello sull'accuratezza continui a crescere, perché comunque vuol dire che man

mano riesce sempre più ad essere preciso nell'individuare la classe di appartenenza di un vestito e mentre per il grafico di loss vogliamo una discesa sempre costante, perché questo vuol dire che sta riuscendo a trovare dei valori che identificano le caratteristiche sempre più efficienti. Quello che però possiamo notare è che i due grafici non si assestano nel crescere o nella discesa, quindi probabilmente possono fare ancora meglio. Per capire su cosa lavorare andiamo ad analizzare anche i risultati del modello 11 in modo più approfondito. Questi risultati sono visibili attraverso la matrice di confusione in [Figura 4.2].

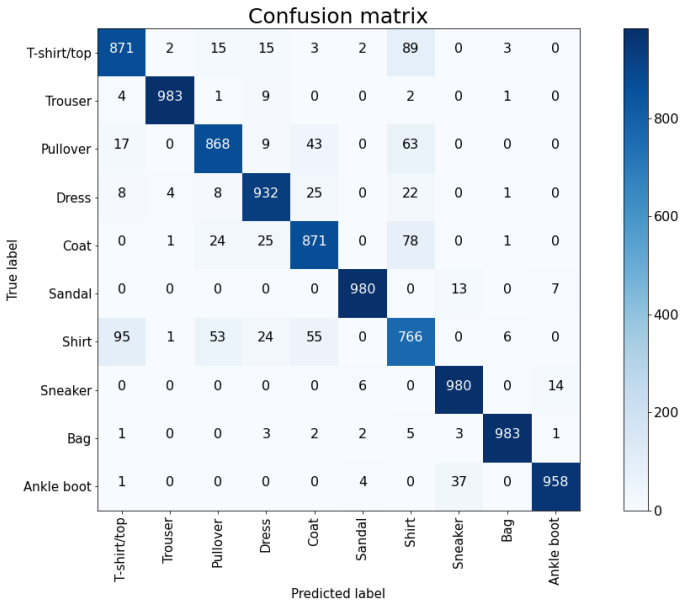

Figura 4.2: Matrice di confusione generata dai test con il modello con id 11

Nella matrice di confusione possiamo osservare che il problema maggiore è dato dalle previsioni della classe shirt, che ha una recall di 0.77, questo ci dice che un elemento su quattro che dovrebbe appartenere alla classe in questione viene classificato come altro, in particolare come t-shirt o come coat, forse perché hanno una silhouette molto simile nelle immagini e quindi non riesce a distinguerle bene [Figura 4.3]. A questo punto l'idea è stata quella di provare ad allenare la rete con più elementi per migliorare anche le percentuali nelle classi su cui sbaglia più previsioni, magari avendo a disposizione più immagini di un elemento la rete riesce a estrapolare più informazioni e quindi riesce a generare un modello più performante, anche su quelle classi.

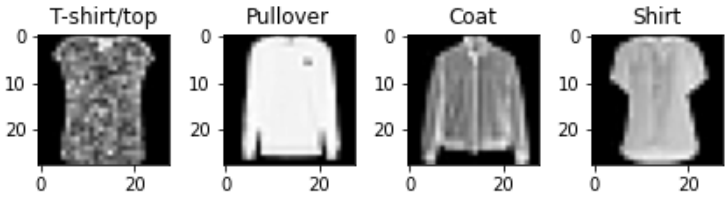

Figura 4.3: Classi che hanno silhouette simili

4.2 Risultati dei modelli con data agumentation

La data agumentation come già abbiamo spiegato serve a generare più elementi di quelle che abbiamo inizialmente, sfruttando gli elementi originali e modificandoli in diversi modi, e come già abbiamo mostrato nel capitolo 3 la scelta per questo lavoro di tesi è ricaduta sul riprodurre nuove immagini andando a spostare l'elemento al centro della foto. A questo punto per il train non avevamo più 60000 immagini ma 300000 che sono state sempre suddivise tra set di train e validation con 85/15.

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	30	128	0.92	0.92	0.92	0.92
1	50	128	0.93	0.93	0.93	0.93
2	60	128	0.93	0.93	0.93	0.93
3	70	128	0.93	0.93	0.93	0.93
4	30	256	0.93	0.93	0.93	0.93
5	50	256	0.93	0.93	0.93	0.93
6	60	256	0.93	0.93	0.93	0.93
7	70	256	0.93	0.93	0.93	0.93

Tabella 4.2: Tabella con i risultati dei test con data augumentation

A questo punto abbiamo riaddestrato i modelli partendo da quello che aveva data risultati migliori con il precedente train, ovvero con i parametri impostati su 30 epoche e con 128 di batch size, e da qui abbiamo settato i parametri a crescere, visto che prima facendo così i risultati ne hanno beneficiato. Siccome abbiamo modelli

nuovi cerchiamo di rianalizzare tutti gli elementi di cui abbiamo discusso prima per vedere se ci sono stati dei miglioramenti. Tutti i risultati sono visibili nella [Tabella 4.4]. Nella tabella notiamo che i risultati sono migliorati, ovviamente i modelli hanno beneficiato sia del cambio ulteriore dei parametri sia grazie all'aumento di dati a disposizione. Infatti sopratutto nella matrice di confusione [Figura 4.4] notiamo che il nostro intento, ovvero quello di migliorare i risultati su la classe che maggiormente veniva sbagliata (shirt), è stato raggiunto, infatti i risultati sono migliorati fino ad avere una precion del 82%.

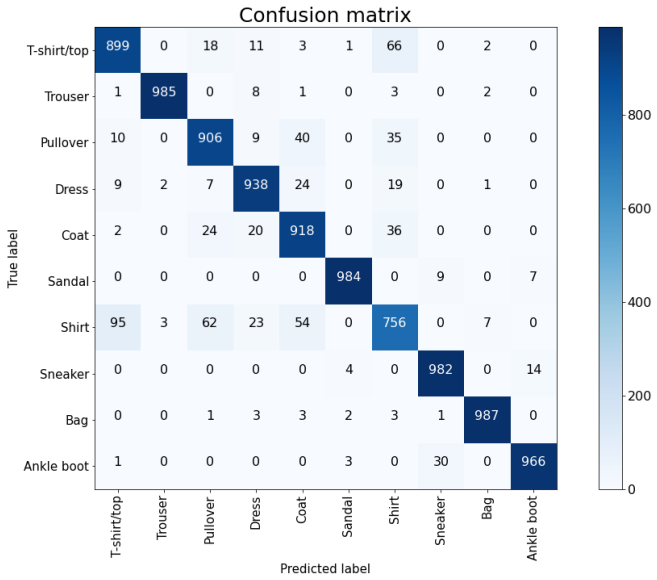

Figura 4.4: Matrice di confusione del con id 5

4.3 Risultati dei modelli con immagini filtrate

In questo paragrafo l'obiettivo sarà quello di capire la potenza reale dei modelli generati, perché comunque è semplice riconoscere immagini simili a quelle con cui sono stati allenati ma il tutto nella realtà diventa nettamente più complesso per via della scarsa qualità che le immagini potrebbero avere. Quindi andiamo ad analizzare i risultati dei modelli testati su immagini che abbiamo costruito attraverso l'utilizzo dei filtri che abbiamo definito nel capitolo 3. Il paragrafo si compone di sotto paragrafi, ognuno per ogni tipo di filtro utilizzato, dove andiamo ad analizzare i risultati e a mostrare le immagini su cui questi test sono stati effettuati, mostrando non esempi

come nel capitolo 3, dove abbiamo voluto solo presentare i vari filtri, ma mostrando come per ogni classe agisse il filtro.

4.3.1 Immagini con filtro Blur

Nella [Figura 4.5] possiamo visualizzare le immagini originali con il filtro blur applicato, questo filtro come possiamo vedere va a sfocare l'immagine e simula sostanzialmente la possibilità che lo strumento utilizzato per catturare l'immagine possa non riprendere perfettamente l'immagine, o perché in movimento o perché in condizioni atmosferiche particolari come nebbia. Nella [Tabella 4.3] possiamo osservare come modelli che prima avevano percentuali altissime, il che era sinonimo di modelli efficienti, ora hanno dei valori drasticamente più bassi. Ora vedendo questi valori i modelli possono anche essere definiti poco efficaci, ma comunque bisogna pur sempre considerare la situazione speciale che abbiamo creato, che ovviamente è sinonimo di ostacolo alla classificazione.

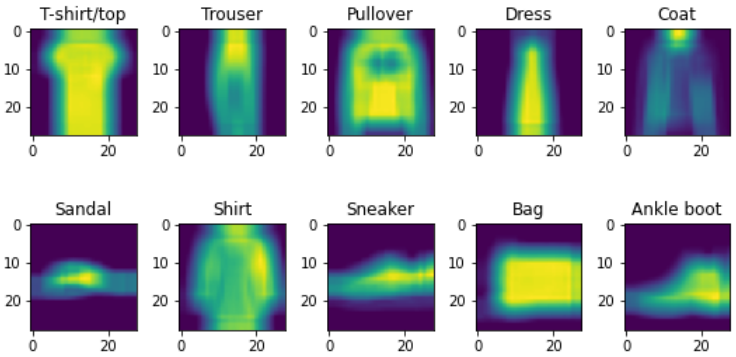

Figura 4.5: Rappresentazioni di tutte le classi con il filtro blur

Il modello migliore in questo caso può esser considerato quello con id 3, che prima comunque risultava essere tra i migliori, quindi cerchiamo di analizzare la matrice che ha generato. Quello che notiamo nella matrice in [Figura 4.6] è che l'effetto applicato ha peggiorato tutte le previsione ma in particolare ha quasi azzerato le possibilità di individuare per bene la classe sandal e coat, probabilmente perché sono quelle che come vediamo da [Figura 4.5] hanno risentito di più di questo filtro, fino a diventare quasi irriconoscibili. Queste due classi vengono confuse con altre perché non hanno più quei dettagli che permetteva il modello di classificarli correttamente,

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	30	128	0.58	0.44	0.38	0.44
1	50	128	0.42	0.40	0.33	0.40
2	60	128	0.56	0.43	0.37	0.43
3	70	128	0.59	0.52	0.46	0.52
4	30	256	0.44	0.47	0.40	0.47
5	50	256	0.59	0.50	0.44	0.50
6	60	256	0.59	0.50	0.45	0.50
7	70	256	0.50	0.49	0.44	0.49

Tabella 4.3: Tabella con i risultati dei test con filtro blur

e si ripresenta il problema di prima in cui il sistema confondeva le classi solo perché avevano la stessa silhouette. Il sandal non viene più distinto dalle sneakers e dalle boot, infatti nella matrice vediamo che il sistema va proprio a sbagliare individuando i sandali in queste due classi e il coat, invece, viene confuso con il pullover.

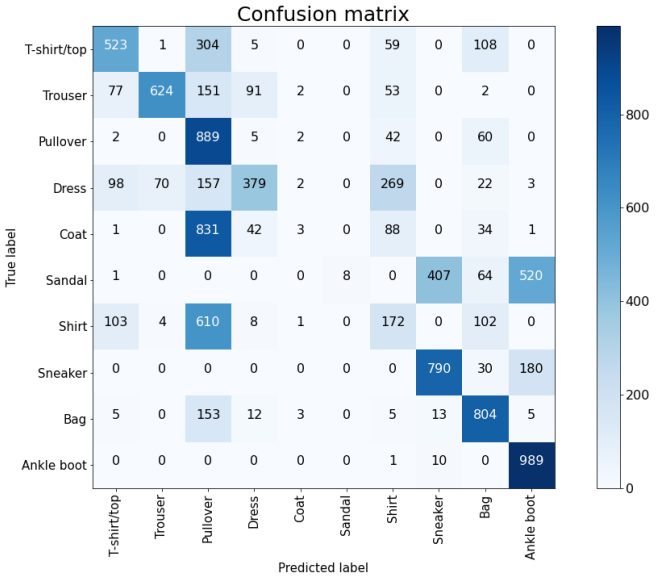

Figura 4.6: Matrice di confusione dei test con il filtro blur

4.3.2 Immagini con filtro Contrast

Nella [Figura 4.7] possiamo visualizzare le immagini originali con il filtro contrast applicato, questo filtro come possiamo vedere serve a rendere i bordi delle figure più nitidi. Questo filtro va a simulare l'effetto che potrebbe esserci nel momento in cui può essere molta luce nell'ambiente circostante.

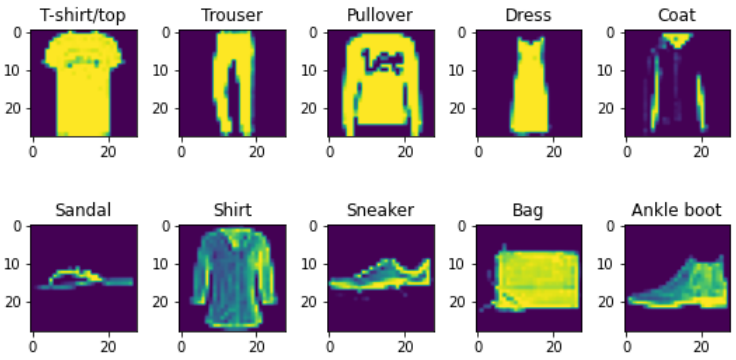

Figura 4.7: Rappresentazioni di tutte le classi con il filtro contrast

Nella [Tabella 4.4] possiamo osservare come modelli rispetto alla situazione che si verificava con il filtro blur riescono ad avere una risposta migliore con questo altro filtro.

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	30	128	0.85	0.84	0.84	0.84
1	50	128	0.84	0.84	0.84	0.84
2	60	128	0.84	0.84	0.84	0.84
3	70	128	0.83	0.82	0.82	0.82
4	30	256	0.85	0.84	0.84	0.84
5	50	256	0.84	0.84	0.84	0.84
6	60	256	0.85	0.84	0.84	0.84
7	70	256	0.85	0.84	0.84	0.84

Tabella 4.4: Tabella con i risultati dei test con filtro contrast

Le percentuali anche se diminuiscono, anche perché questo filtro comunque genera una perdita di informazioni, riescono comunque a mantenersi su valori molto alti. In questo caso non presentiamo la matrice di confusione perché mantiene una forma molto simile a quella in cui abbiamo testato le immagini originali, ovviamente con degli errori in più, ma distribuiti sempre alla stessa maniera.

4.3.3 Immagini con filtro Cut

Nella [Figura 4.8] possiamo visualizzare le immagini originali tagliate a metà. Il taglio per ogni classe è stato fatto verticalmente praticamente al centro dell'immagine. Quello che vogliamo simulare con questo filtro è la situazione in cui in determinato abito all'interno dell'immagine non è completamente ripreso. Nella [Tabella 4.5] possiamo osservare come i modelli ora hanno dei valori drasticamente più bassi, come nel caso delle immagini su cui avevamo applicato il filtro del blur. Questo ci fa presumere che nel caso in cui il modella non riesca ad inquadrare completamente il capo di abbigliamento può avere dei grossi problemi, dovuti a delle grosse perdite di informazioni su cui il modello si basava per classificare alcune immagini in specifiche classi.

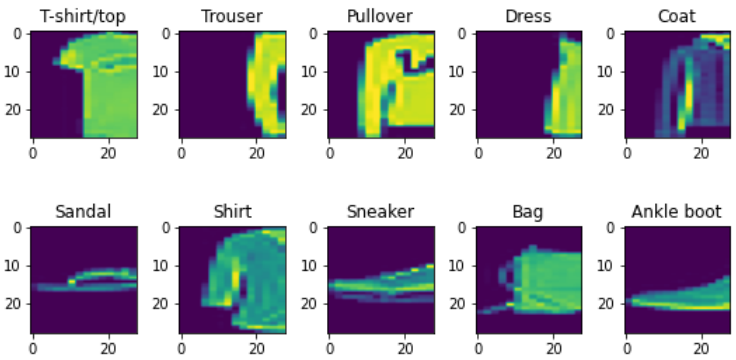

Figura 4.8: Rappresentazioni di tutte le classi con il filtro contrast

Le percentuali comunque sono così basse per via di alcune classi che ora non vengono praticamente più considerate per la classificazione come vediamo nella [Figura 4.9], dove analizziamo come sempre la matrice di confusione generata dal modello che ci ha dato dei risultati migliori.

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	30	128	0.30	0.25	0.19	0.25
1	50	128	0.33	0.31	0.23	0.31
2	60	128	0.31	0.28	0.21	0.28
3	70	128	0.28	0.28	0.21	0.28
4	30	256	0.32	0.28	0.21	0.28
5	50	256	0.33	0.32	0.27	0.32
6	60	256	0.29	0.26	0.19	0.26
7	70	256	0.30	0.26	0.20	0.26

Tabella 4.5: Tabella con i risultati dei test con filtro cut

La classe del trouser e del dress hanno delle percentuali di precision sotto lo il 10%, il che vuol dire che non riesce praticamente più a riconoscere, ed entrambi vengono confusi con la classe del sandal, probabilmente perché così tagliate se viste in orizzontale danno la sensazione che potrebbero essere dei sandal.

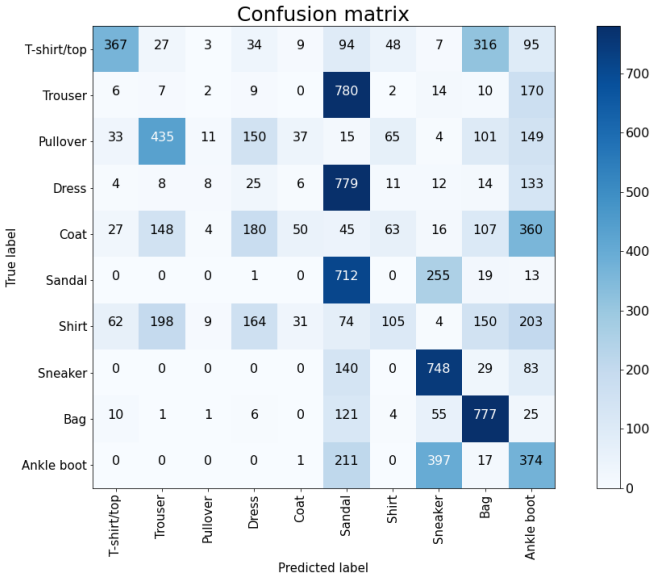

Figura 4.9: Matrice di confusione dei test con il filtro cut

Ovviamente queste due classi sono nettamente le peggiori, ma comunque su tutte c'è un grosso peggioramento tranne che su tre che sono il sandal, la snekers e la

bag, però anche in questo caso si può fare una precisazione nel caso del sandal, ovvero praticamente la maggior parte degli elementi vengono classificati in quel modo quindi è molto semplice che vengano riconosciuti molti degli elementi che appartengono effettivamente a quella classe, infatti questa classe ha una precision del 24% ma una recall del 70%. Poi comunque in generale queste tre classi anche se divise a metà hanno mantenuto molto dei propri elementi che li rendono distinguibili.

4.3.4 Immagini con filtro Rain

Nella [Figura 4.10] possiamo visualizzare le immagini con il filtro rain. Questo filtro come si può intuire dal nome va a simulare la condizione atmosferica della pioggia ed è stato creato andando a generare delle linee che vengono applicate casualmente sull'immagine in maniera obliqua e successivamente sfogando leggermente l'immagine.

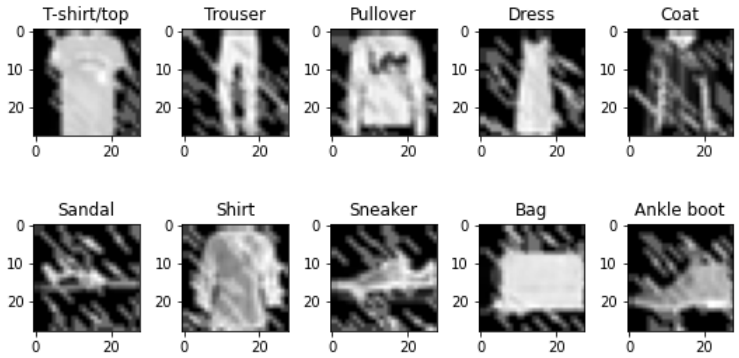

Figura 4.10: Rappresentazioni di tutte le classi con il filtro rain

Nella [Tabella 4.6] vediamo che i modelli peggiorano, ma nemmeno di troppo visto l'applicazione del filtro. Quello che possiamo notare è che comunque anche se l'accuratezza praticamente viene dimezzata rispetto ai test con le immagini originali, la precison rimane comunque elevata. Come sempre però per capire perché dobbiamo studiare i singoli casi, e prediamo come riferimento il modello con id 4. Nella [Figura 4.11] notiamo una cosa molto interessante per quanto riguarda la classe shirt. Infatti, mentre quando venivano effettuati i test con le immagini originali la recall di questa classe tendeva ad essere la più bassa ora invece è diventata la più alta e anche di molto.

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	30	128	0.69	0.54	0.55	0.54
1	50	128	0.67	0.43	0.44	0.43
2	60	128	0.66	0.48	0.49	0.48
3	70	128	0.71	0.58	0.58	0.58
4	30	256	0.72	0.58	0.59	0.58
5	50	256	0.70	0.56	0.57	0.56
6	60	256	0.67	0.50	0.49	0.50
7	70	256	0.70	0.53	0.54	0.53

Tabella 4.6: Tabella con i risultati dei test con filtro rain

Quelle che erano le classi [Figura 4.3] con la stessa silhouette con cui venivano classificate molte delle shirt sbagliando ora vengono classificate esse stesse come shirt, praticamente si sono invertiti i ruoli, questo successo perché per via della perdita di informazione che è avvenuta sugli elementi. Le t-shirt, il pullover e il coat senza elementi distintivi tendono ad essere visti come semplici shirt.

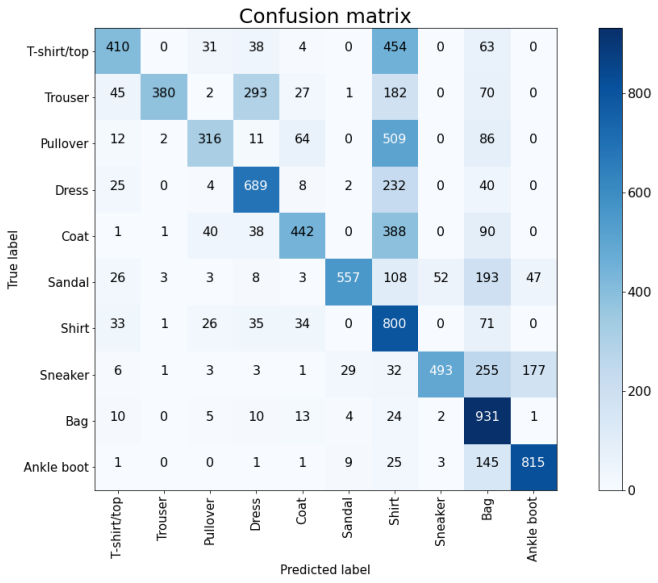

Figura 4.11: Matrice di confusione dei test con il filtro rain

4.3.5 Immagini con filtro Occlusion

Nella [Figura 4.12] vediamo come agisce il filtro occlusion. Questo filtro va a generare dei buchi sugli abiti attraverso la generazione casuale di pallini neri, l'effetto che si vuol ricreare è molto simile a quello che abbiamo visto con il cut, ovvero quando non tutto il capo di abbigliamento è visibile, però in questo caso l'idea è di simulare la situazione in cui davanti all'elemento da classificare ci fosse qualche altro oggetto. Nella [Tabella 4.7] come sempre vediamo i valori abbassarsi rispetto ai test effettuati con le immagini originali, ma ripetendo il fatto che è una cosa che ci può stare visto la perdita di informazioni, vorrei soffermarmi su una cosa che riguarda la classe del sandal.

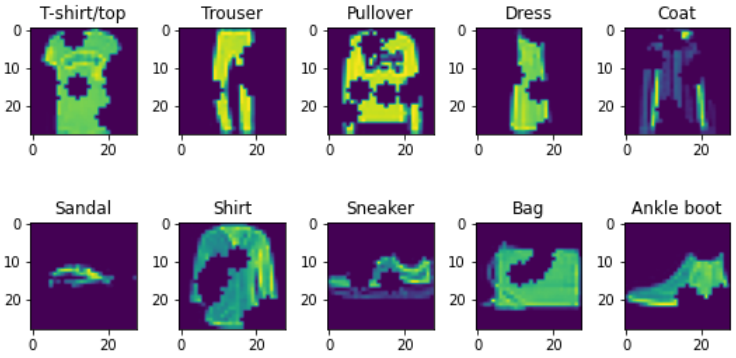

Figura 4.12: Rappresentazioni di tutte le classi con il filtro occlusion

Prendendo in considerazione sempre il modello migliore, in questo caso il modello 0, il valore della recall su questa classe risulta altissimo, tocca il 98% ma con una precision molto bassa intorno al 30%. Questa stessa caratteristica del test l'abbiamo vista quando abbiamo valutato le immagini tagliate, che guarda cosa sono un test molto simile a quello delle occlusion. Questo ci sta a significare che è proprio una proprietà del sistema quindi quella di concentrarsi su certe classi e lasciare indietro altre nel momento in cui riesce a vedere meno, o meglio, non ha la silhouette completa del capo di abbigliamento.

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	30	128	0.59	0.50	0.49	0.50
1	50	128	0.57	0.42	0.42	0.42
2	60	128	0.57	0.39	0.39	0.39
3	70	128	0.59	0.37	0.37	0.37
4	30	256	0.60	0.39	0.39	0.39
5	50	256	0.60	0.43	0.43	0.43
6	60	256	0.57	0.37	0.37	0.37
7	70	256	0.61	0.45	0.45	0.45

Tabella 4.7: Tabella con i risultati dei test con filtro occlusion

4.3.6 Immagini con filtro Rotate

Nella [Figura 4.13] visualizziamo le immagini originali ruotate di 90 gradi. Questo filtro è utilizzato per testare la capacità del sistema di individuare il tipo dei capi anche se viste in un'altra prospettava.

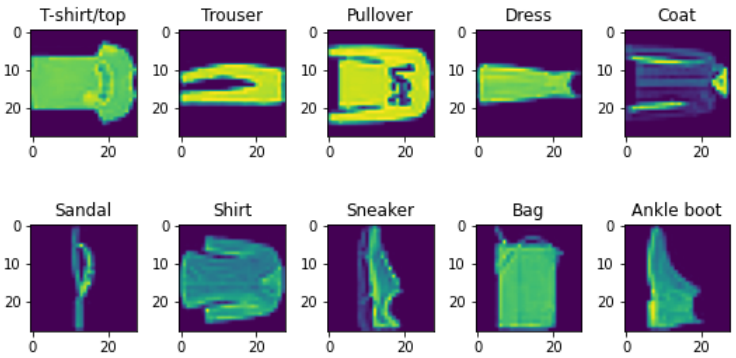

Figura 4.13: Rappresentazioni di tutte le classi con il filtro rotate

Abbiamo lasciato queste ultime immagini da analizzare con i modelli con data augumentation per via dei risultati. Nella [Tabella 4.8] vediamo dei risultati disastrosi, tutti i valori sono crollati intorno al 10%. Anche per questi risultati disastrosi però esiste una spiegazione che è intuibile attraverso la matrice di confusione [Figura 4.14].

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	30	128	0.10	0.06	0.03	0.06
1	50	128	0.10	0.08	0.06	0.08
2	60	128	0.08	0.08	0.06	0.08
3	70	128	0.14	0.09	0.07	0.09
4	30	256	0.10	0.08	0.06	0.08
5	50	256	0.10	0.09	0.06	0.09
6	60	256	0.11	0.09	0.06	0.09
7	70	256	0.11	0.09	0.07	0.09

Tabella 4.8: Tabella con i risultati dei test con filtro blur

Prendiamo in considerazione la matrice del modello con id 7, con i risultati migliori, praticamente tutte le predizioni si sono concentrate su una sola classe, ovvero la classe bag. Probabilmente i modelli hanno appreso che quando l'immagine è in un certo verso e ha una certa forma è una bag e andando a ruotare tutte le immagini vengono classificate tutte in quel modo.

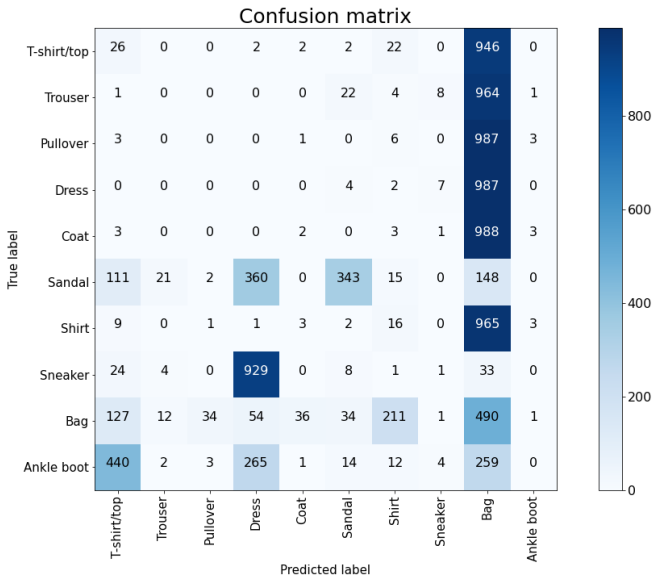

Figura 4.14: Matrice di confusione con il filtro rotate

4.4 Risultati dei modelli con immagini colorate

Dedichiamo un capito a parte per le immagini colorate perché come abbiamo precedentemente scritto per testare con questo tipo di immagini abbiamo dovuto riaddestrare i modelli per specificare che gli input che riceveva ora non avevano più un unico canale di colori ma 3. Quindi ora andremo a valutare modelli riaddestrati completamente da zero, con una data augumentation minore rispetto ai precedenti perché il sistema andava in crash con quella quantità di immagini con tre canali, e da 300000 ora il train è stato effettuato con 120000 immagini. Come vediamo in [Figura 4.15] le immagini sono colorate o di rosso, o di verde o di blu, per il resto l'immagine rimane invariata.

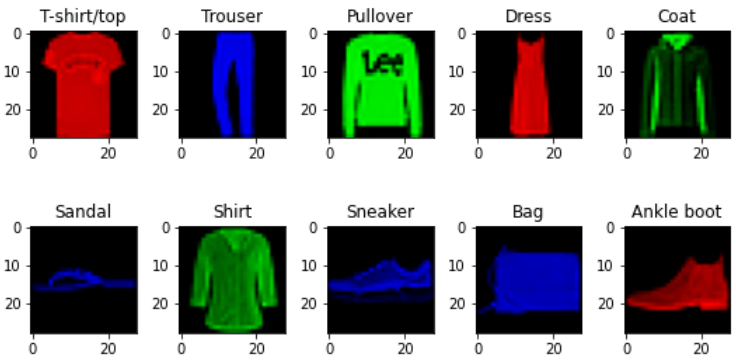

Figura 4.15: Rappresentazioni di tutte le classi con il filtro colored

Nella [Tabella 4.9] notiamo subito una differenza rispetto agli altri esperimenti, che ci sono due modelli in meno. Il train per modelli con 70 epoche mandavano in crash il sistema, e quindi non si ha a disposizione quei modelli per riuscirli a testare. In questo test comunque abbiamo risultati molti buoni, quasi quanto l'esperimento con le immagini originali, però ragionandoci con attenzione visto che in questo caso test e train come nel primo esperimento hanno lo stesso formato di immagine, quindi come mai non hanno risultati alti tanto quanto? Alla domanda si può rispondere ipotizzando che il problema principale sta nel fatto che nel train il sistema non ha a disposizione 3 immagini per ogni capo, visto che abbiamo 120000 immagini, ovvero

Id	Epoche	Batch size	Precision	Recall	F1-score	Accuracy
0	30	128	0.82	0.79	0.80	0.79
1	50	128	0.83	0.80	0.81	0.80
2	60	128	0.75	0.70	0.71	0.70
3	30	256	0.84	0.78	0.78	0.78
4	50	256	0.86	0.83	0.83	0.83
5	60	256	0.82	0.77	0.77	0.77

Tabella 4.9: Tabella con i risultati dei test con filtro color

il doppio del dataset iniziale, di conseguenza non abbiamo una immagine che per ogni capo del dataset che viene riprodotta in tutti e 3 i tipi di colori presi in considerazione, questo porta il dataset per il train ad essere squilibrato. Quindi probabilmente per alcuni elementi il sistema riesce ad individuare meglio la classe di appartenenza quando è di un determinato colore.

4.5 Riassunto dei risultati

In questa sezione andiamo a sintetizzare tutti i risultati di cui abbiamo discusso prima in un'unica tabella [Tabella 4.10], per farne una analisi più veloce e semplice. La tabella si compone di quattro colonne di cui nella prima andiamo a specificare che modelli abbiamo utilizzato, nella seconda le immagini del test con cui otteniamo i risultati, nella terza mettiamo il riferimento alla tabella con i riusltati e infine nell'ultima inseriamo una sintesi delle osservazioni fatte in precedenza. Ovviamente se si vuole approfondire consultare i capitoli specifici. Cercheremo di dare una conclusione alla ricerca in base a tali risultati nel prossimo capitolo, dobbiamo capire se siamo riusciti a rispondere alle domande iniziali.

Modello	Immagini usate	Risultati	Osservazioni
Modello senza data augumentation	Immagini originali	Tab. 4.1	I risultati sono mediamente ottimali ma con possibilità di migliorare perciò si opta per la data augumentation.
Modello con data augumentatioon	Immagini originali	Tab. 4.4	I risultati sono mediamente ottimi, come ci si aspettava migliori rispetto a modelli senza data augumentation.
Modello con data augumentatioon	Immagini originali con blur	Tab. 4.3	I risultati sono peggiorati, sopratutto per la classe sandal e coat, perché vengono confuse avendo perso dettagli che le distinguevano da altre classi con forma simile, es. sandal con boot.
Modello con data augumentatioon	Immagini originali con contrast	Tab. 4.4	I risultati sono inferiori di poco a quelli con le immagini originali, per via di una piccola perdita di informzioni.
Modello con data augumentatioon	Immagini originali tagliate	Tab. 4.5	I risultati sono bassi per via di alcune classi che ora non vengono praticamente più considerate per la classificazione, come per esempio le classi del dress e del trouser che con una parte mancante vengono scambiate per il sandal dal sistema.

Modello con data augumentatioon	Immagini originali con rain	Tab. 4.6	I risultati sono bassi, il sistema si concentra motlo sulla silhouette e poco su dettagli che distingue i vari capi e quindi confonde molto t-shirt, shirt, ecc..
Modello con data augumentatioon	Immagini originali con occlusion	Tab. 4.7	I risultati sono bassi, per via di una bassa visibilità dell'immagine e presenta la stessa situazione delle immagini tagliate.
Modello con data augumentatioon	Immagini originali ruotate	Tab. 4.14	I risultati sono disastrosi, le percentuali sono tutte intorno al 10%, la motivazione sta nel fatto che ruoatate di 90° le immagini vengono scambiate tutte per la bag che prima era l'unica ad essere in quella posizione.
Modello con data augumentatioon e immagini a 3 canali	Immagini originali colorare di blu, rosso o verde	Tab. 4.9	I risultati non sono altissimi, la possibilità in questo caso è che nell'allenameto non sono state mostrate le immagini con tutti i colori analizzati ognuna e il sistema si trova impreparato.

Tabella 4.10: Tabella riassuntiva dei risultati della ricerca

CAPITOLO 5

Conclusioni

Questo capitolo illustra quanto è emerso dalla ricerca, mostrando i risultati più significativi. Sono, inoltre, riportate le limitazioni incontrate e alcuni possibili sviluppi futuri. Nel caso si volessero effettuare dei test il codice si può trovare su `https://github.com/AntonioCimino/Reconigtion_clothes_in_image`.

5.1 Interpretazione dei risultati

Nel presente lavoro di ricerca avevamo lo scopo di costruire un sistema per il riconoscimento dei capi di abbigliamento all'interno delle immagini, e per raggiungere tale scopo è stata costruita una rete neurale. La rete ha generato dei modelli che sono stati valutati per capire se siamo arrivati ad una soluzione sufficientemente soddisfacente. La valutazione è stata suddivisa in due fasi, la prima per capire come i modelli funzionavano in maniera generale e poi se i modelli potessero funzionare anche nella vita reale, attraverso immagini che potessero simulare determinate situazioni. Queste due fasi sperimentali quindi rispondono a due quesiti che abbiamo stabilito all'inizio del capitolo 3. Da tale analisi è stato possibile notare nei primi esperimenti il sistema sembrava risultare quasi perfetto nelle sue previsioni, infatti raggiungiamo delle percentuali che vanno oltre il 92%, cosa che non tutti i sistemi presenti oggi riescono

ad ottenere, allo stesso tempo esistono sistemi migliori come in [5]. Il problema maggiore è stato il problema di alcune classi che venivano confuse per via della loro somiglianza (T-shirt, coat, ecc..) altrimenti probabilmente saremo riusciti ad arrivare anche oltre. Nei secondi esperimenti invece sono nati sempre più problemi, sempre dovuti alla confusione che veniva effettuate sulla forma degli abiti, che con i filtri tendevano ad assomigliare ad altre classi anche se inizialmente totalmente diverse. Nei secondi esperimenti probabilmente possiamo essere contenti tutto sommato di aver raggiunto delle percentuali per alcuni casi vicino al 60%, che anche se bassi, visto come sono state modificate le foto sono risultati buoni, sopratutto perché il modello è stato allenato sul riconoscere immagini in perfette condizioni e nel momento in cui queste immagini perdono le loro condizioni ottimali è intuibile che ci può essere un calo di qualità. Sulla base di tali risultati, è possibile rispondere ai quesiti alla base del lavoro di ricerca e affermare che i modelli generati riescono a riconoscere bene i capi nelle immagini in una situazione generica, mentre in situazioni particolari meno, in particolare in alcune di esse, come nell'esperimento con le immagini ruotate, per nulla.

5.2 Limitazioni e sviluppi futuri

Il sistema ha bisogno, probabilmente, di avere dati più sporchi durante l'allenamento perché poi sia in grado di esser pronto a classificarli. Sembra strano sentir dire che per il train servono dati con imperfezioni, ma il ragionamento è che abituare il sistema a vedere i capi anche in altre situazioni lo potrebbe preparare a non sbagliare la classificazione in quelle situazioni come successo. Il problema è che le risorse a disposizione con cui è stato sviluppato il sistema hanno fatto fatica quando ci si è provato a spingere più in la, come nel caso del train con le immagini colorate, quindi servirebbe molta più potenza computazionale. Un'altra cosa che si potrebbe fare è uno step successivo dal punto di vista sperimentale, riportando il sistema in una situazione reale e non in una che simula la realtà come abbiamo fatto noi, perché comunque ci servirebbe capire come esso si comporta con altri elementi nelle immagini, e da lì andare a studiare un modo per riprendere la forma dell'elemento nel modo migliore.

Bibliografia

[1] "Nozioni di base della cnn nel deep learning," 2022. [Online]. Available: https://zephyrnet.com/it/basics-of-cnn-in-deep-learning/ (Citato alle pagine iii e 7)

[2] L. Bossard, M. Dantone, T. Q. Christian Leistner, Christian Wengert, and L. V. Gool, "Apparel classi

cation with style," 2012. (Citato alle pagine iii, 10 e 11)

[3] G. K. V and S. K., "Fashion-mnist classification based on hog feature descriptor using svm," 2019. (Citato alle pagine iii, 11 e 12)

[4] S. Bhatnagar, D. Ghosal, and M. H. Kolekar, "Classification of fashion article images using convolutional neural networks," 2017. (Citato alle pagine iii, 13 e 14)

[5] M. Kayed, A. Anter, and H. Mohamed, "Classification of garments from fashion mnist dataset using cnn lenet-5 architecture," 2020. (Citato alle pagine iii, 13, 14, 15 e 49)

[6] N. Bertagnolli, "Dropout is drop-dead easy to implement," 2022. (Citato alle pagine iii e 19)

[7] J. Jeong, "The most intuitive and easiest guide for convolutional neural network," 2019. (Citato alle pagine iii e 20)

[8] "Mlp for f-mnist classification," 2022. [Online]. Available: https://www.kaggle.com/code/nikolaosjp/mlp-for-f-mnist-classification (Citato alle pagine iii e 21)

[9] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarkingmachine learning algorithms," 2017. (Citato alle pagine v, 9, 10 e 12)

[10] N. Dey, G. S. Mishra, S. Chakraborty, and J. Kar, "A survey of image classification methods and techniques," 2014. (Citato alle pagine 4 e 5)

[11] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015. (Citato alle pagine 7 e 8)

[12] U. M. D. Ciresan, L. M. Gambardella, and J. Schmidhuber, ""convolutional neural network committees for handwritten character classification," 2011. (Citato a pagina 8)

[13] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," 2011. (Citato a pagina 8)

[14] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis," 2003. (Citato a pagina 8)

[15] L. Deng and D. Yu, "Deep convex network: A scalable architecture for speech pattern classification," 2011. (Citato a pagina 8)

[16] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2019. (Citato a pagina 18)

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," 2014. (Citato a pagina 19)

[18] "Fashion mnist." [Online]. Available: https://www.kaggle.com/datasets/zalando-research/fashionmnist (Citato a pagina 20)

[19] A. Hasnat, M. Rubaiyat, Y. Qin, and H. Alemzadeh, "Experimental resilience assessment of an open-source driving agent," 2018. (Citato a pagina 25)

Ringraziamenti

Ringraziamenti qui...