APPUNTI DEL CORSO DI INFORMATICA MULTIMEDIALE INTELLIGENZA ARTIFICIALE

Corso di laurea Magistrale in Comunicazione e Culture Digitali (LM-19)

Sommario

1. Machine Learning for Image Analysis

Il machine learning è un sottoinsieme dell'intelligenza artificiale (Artificial Intelligence - AI). Il suo compito è addestrare gli algoritmi ad imparare dai dati e a migliorare con l'esperienza, anziché essere appositamente programmato per riuscirci. Nel machine learning gli algoritmi vengono addestrati a far emergere schemi e correlazioni da grandi set di dati e a formulare le migliori decisioni e previsioni sulla base di tali analisi. Il machine learning come dice la parola stessa insegna ai computer quello che gli umani naturalmente fanno e quindi imparare dall'esperienza. Gli algoritmi di machine learning utilizzano dei modelli computazionali più o meno complessi per imparare delle informazioni direttamente dai dati quindi senza doversi basare su dei modelli matematici predefiniti. Essi migliorano le loro performance quindi ad esempio l'accuratezza di classificazione via via che il numero di campioni disponibili aumenta e questo è proprio quello che succede agli esseri umani che, studiando e focalizzandosi sempre su esempi diversi riescono poi a raggiungere dei livelli di performance accettabili.

Gli algoritmi di machine learning vanno a cercare dei pattern naturali all'interno dei dati, siano dati di tipo immagine o video, quindi in generale dati multimediali, ma anche altri tipi di dati come ad esempio le sequenze temporali. Attraverso l'analisi di questi pattern naturali presenti all'interno dei nostri dati ci aiutano a prendere decisioni migliori oppure effettuare delle predizioni. In realtà vengono già adesso utilizzati in una grande varietà di applicazioni che vanno dalla diagnosi medica, all'analisi del comportamento umano o in ambito finanziario.

Grazie all'avvento dei big data e alla disponibilità di queste grandi quantità di dati, i metodi di machine learning sono diventati particolarmente diffusi e importanti. Quindi abbiamo approcci nel dominio della finanza e della biologia, nella produzione di energia ovviamente nel settore automotive (ad esempio, self driving cars) e ovviamente al natural language processing (google home o alexa che probabilmente abbiamo nelle nostre case), o il riconoscimento facciale come accade con il nostro smartphone oppure per la rilevazione del movimento o per la cosiddetta detection, l'individuazione di oggetti all'interno di immagini o video.

Non sempre dobbiamo introdurre il machine learning per risolvere dei problemi. Utilizziamo il machine learning quando abbiamo una grande quantità di dati e molte variabili e soprattutto quando non esiste un'equazione o una formula in grado di legare tutte queste variabili e tutti questi dati. Se vogliamo definire l'equazione di una retta non abbiamo bisogno di un algoritmo di machine learning perché esiste già una formulazione matematica che ci permette di legare un input ad un output. Invece, il machine learning è una buona opzione quando ad esempio abbiamo delle equazioni troppo complesse o delle regole "scritte a mano" quindi che non siamo in grado di modellare tramite un un'equazione matematica oppure quando le regole del nostro problema cambiano in continuazione e quindi c'è bisogno continuamente di adattarsi ai nuovi dati a disposizione oppure quando la natura dei dati continua a cambiare. Se pensiamo che banalmente la risoluzione delle camere dei nostri smartphone diventa ogni volta migliore o appunto abbiamo immagini sempre a risoluzione maggiore capiamo che abbiamo bisogno di alcuni strumenti matematici che ci permettono di far fronte a questo cambiamento.

1.1 Machine Learning Challenges

Esistono diverse sfide che dobbiamo affrontare nel momento in cui vogliamo utilizzare i nostri algoritmi di machine learning. Primo fra tutti è che dobbiamo trovare il miglior modello e dobbiamo capire come maneggiare i nostri dati. I dati al giorno d'oggi sono disponibili in tutte le forme in tutte le dimensioni (pensiamo di nuovo alla risoluzione delle immagini del nostro smartphone). I dataset presentano spesso diverse problematiche, possono essere disordinati possiamo avere dati incompleti, ad esempio perché manca la misura dell'immagine in un determinato momento della giornata, perché il sensore si è rotto, oppure perché gli operatori stavano facendo della manutenzione e ovviamente vengono in una varietà di formati diversi pensiamo alle immagini possono essere png possono essere jpeg e così via. Possiamo avere semplici dati numerici ma possiamo avere anche combinazioni di diversi tipi di dato quindi possiamo avere un dato numerico assieme ad un'immagine assieme ad un video assieme al dato di un sensore. Ovviamente spesso c'è bisogno di processare i nostri dati e per far questo c'è bisogno di una conoscenza specializzata nell'ambito, non possiamo immaginarci di avere dei dati e di buttare brutalmente questi dati darli in ingresso al nostro algoritmo di machine learning e sperare di ottenere un risultato.

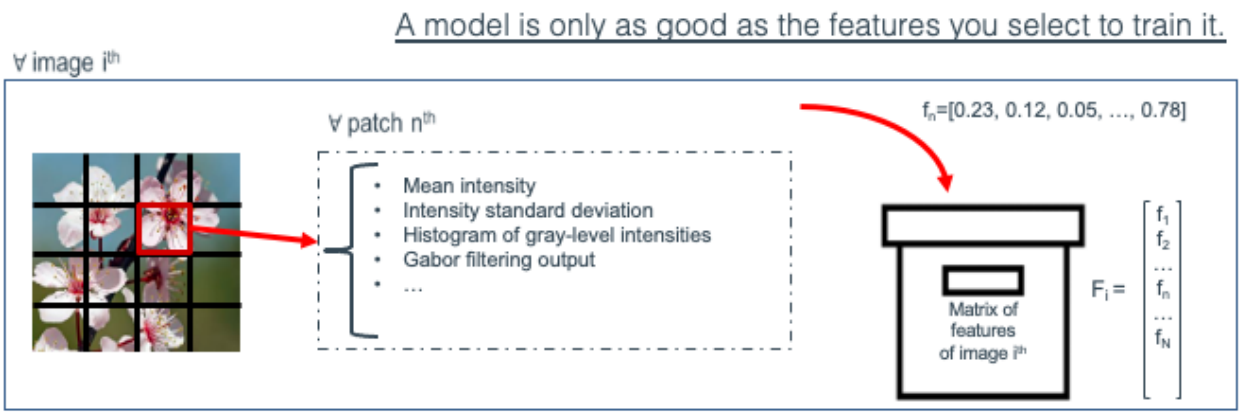
Quello che spesso si dice è “garbage in” “garbage out”. Se io do spazzatura in ingresso con i dati non processati al mio algoritmo di deep learning o di machine learning molto probabilmente quello che si avrà in uscita sarà lo stesso spazzatura quindi qualcosa che non ha alcun significato alcuna valenza pratica.
Dati diversi richiedono ovviamente diversi approcci se vogliamo fare diagnosi ad esempio di un macchinario in una fabbrica che non funziona dobbiamo capire come questo macchinario è fatto. Stessa cosa se vogliamo fare diagnosi analizzando immagini di risonanza magnetica dobbiamo capire come si verifica una determinata patologia e come si evidenzia e visualizza una determinata patologia all'interno di un'immagine di risonanza. Abbiamo poi bisogno di tempo per cercare il modello migliore che vada a “fittare” i nostri dati quindi rappresentare i nostri dati o meglio la relazione che c'è tra i nostri dati e l'output del nostro modello. Scegliere il modello giusto non è semplice richiede una procedura di trial and error: da una parte vogliamo che i nostri modelli siano accurati dall’altra parte però vogliamo che i nostri modelli non overfittino.

Il nostro modello overfitta quando non va descrivere le relazioni, i pattern naturali all'interno dei nostri dati ma va a descrivere il rumore. Questo è un problema perché sappiamo benissimo che ogni misura che noi effettuiamo è accompagnata da un certo tipo di rumore, ad esempio il rumore di misura, quindi, se il nostro modello si attacca ai dati quello che succede è che quando avrò una nuova realizzazione della mia misura avrò un rumore diverso e il mio modello di machine learning non sarà in grado di fornirmi una predizione, un risultato corretto. Altre considerazioni da appunto tenere a mente sono che spesso abbiamo bisogno di avere un modello non solo accurato ma anche veloce. Pensiamo se vogliamo fare un qualche tipo di riconoscimento real-time, ad esempio se un anziano cade vogliamo saperlo subito non vogliamo che il nostro modello ci dia la predizione dopo quattro ore perché non vogliamo lasciare la persona in una situazione di pericolo senza supporto per tanto tempo. Altra importante considerazione da fare è che spesso ci verrà richiesto di spiegare perché il nostro modello di machine learning funziona, questa è una domanda abbastanza complessa che è legata ad un'attività di ricerca che parla della spiegabilità dei modelli di machine learning. Magari il nostro modello più accurato e anche il modello più complesso e non siamo in grado di spiegare al nostro interlocutore come funziona, quindi, preferiamo magari rinunciare a parte dell'accuratezza in favore di una spiegabilità più alta dei meccanismi che governano l'algoritmo di machine learning.

1.2 Handcrafted Feaures

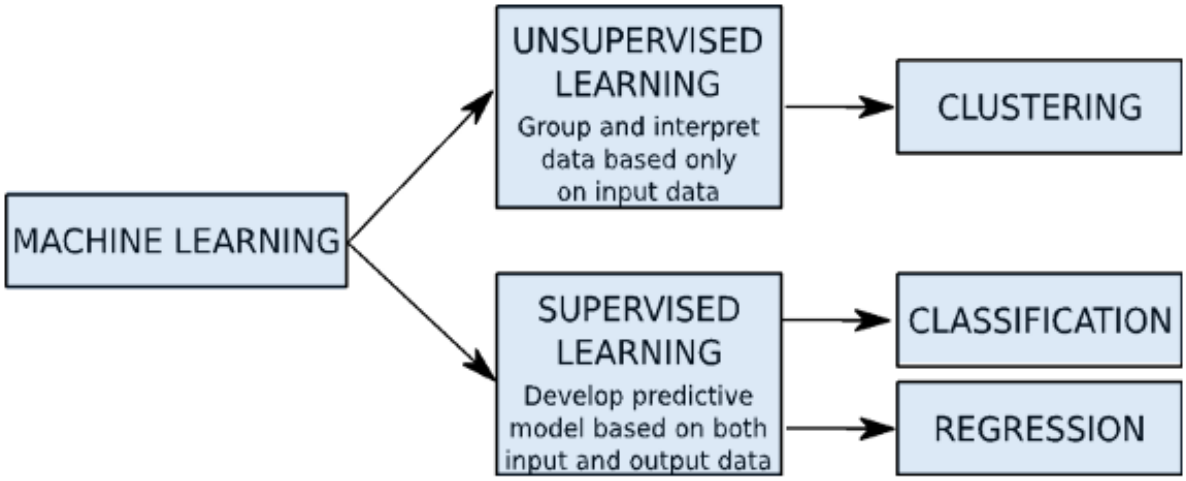
Per scendere un po’ più nel pratico andiamo a definire con che tipi di dati lavoriamo. Gli algoritmi di machine learning standard (a questo punto escludiamo il deep learning che guarderemo approfonditamente nelle prossime sezioni) lavorano su delle cosiddette “feature” quindi delle caratteristiche delle immagini che vengono estratte dai dati. Gli algoritmi di machine learning prevedono una prima fase di estrazione delle feature e una seconda fase di analisi di queste feature. Per estrarre queste feature dobbiamo in qualche modo modellarle quindi dobbiamo scegliere che tipo di informazione, di caratteristica vogliamo estrarre dalle nostre immagini, per questo queste feature si chiamano *handcrafted*. La feature è quindi un pezzo di informazione che è rilevante per risolvere un problema in una certa applicazione. Un modello di machine learning sarà tanto buono quanto è buona la modellazione delle feature che appunto siamo in grado di fornire.

Prendiamo ad esempio un'immagine qualsiasi. In questo caso abbiamo scelto l’immagine di seguito con dei fiori. La prima cosa che si fa è dividere questa immagine in tante piccole sotto immagini che si chiamano patch quindi porzioni di immagine. Questo perché le caratteristiche a livello di immagine sono variabili da zona a zona dell'immagine e quindi vediamo che in basso a destra abbiamo il prato invece al centro abbiamo un fiore. Un fiore visivamente è molto diverso dal prato. Quindi vogliamo delle feature che siano locali e non globali all'immagine. Da ogni patch andiamo a estrarre una serie di feature che di nuovo varieranno da patch a patch. Alcuni esempi che possiamo fare sono l'intensità media oppure la standard deviation dell'intensità oppure l'istogramma dei livelli di grigio qualche tipo di uscita di un filtraggio, ad esempio, per estrarre i contorni. Tutte queste feature per ogni patch vengono poi concatenate per creare la cosiddetta matrice delle feature dell'immagine. Ogni immagine sarà quindi caratterizzata da una serie di feature come, ad esempio, l’intensità media o la standard deviation e così via. A questo punto queste feature verranno calcolate per diverse immagini, ovviamente, non solo per una, perché gli algoritmi di machine learning lavorano su grandi quantità di dati. Verranno successivamente date in ingresso al nostro algoritmo di machine learning.



1.3 Come scegliere il miglior algoritmo di Machine Learning?

Torniamo alla domanda più importante: come scegliamo il miglior algoritmo di machine learning? Innanzitutto, esistono due paradigmi di learning: **apprendimento supervisionato** e **apprendimento non supervisionato**. Nel caso degli algoritmi supervisionati alleniamo un modello dando in ingresso un input e un output conosciuti. Si chiama supervisionato proprio per questo motivo perché l'output almeno nella fase nella fase iniziale che è la fase di training è conosciuto. Diversamente, nel caso dell'apprendimento non supervisionato non abbiamo a disposizione questo output e quindi lo scopo degli algoritmi non supervisionati, è quello invece di andare a cercare dei pattern nascosti o intrinseci dei nostri dati di input senza aver alcun tipo di match con l'output desiderato. Nel caso del non supervisionato probabilmente l'algoritmo più famoso il clustering, nel caso invece dell'apprendimento supervisionato avremo da una parte la classificazione dall'altra la regressione.



1.3.1 Unsupervised learning

Focalizziamoci sui grafici di seguito in cui abbiamo uno spazio con feature 1 e features 2. Queste feature 1 e feature 2 sono esattamente le stesse handcrafted feature quindi ad esempio immaginiamoci sulla sinistra l'intensità media sull'asse y la standard deviation. I colori in questo grafico in realtà non hanno niente a che fare con la classe di appartenenza di questi campioni proprio perché l'apprendimento è di tipo non supervisionato quindi non conosciamo la classe di appartenenza o il valore di appartenenza dei nostri dati in ingresso. Il clustering è la forma più comune di apprendimento non supervisionato. In questo caso quello che si fa e appunto andare a raggruppare i nostri dati nello spazio delle feature andando ad imporre delle metriche basate sulla distanza, che può essere ad esempio la più semplice la distanza euclidea, la quale viene calcolata

fra i dati all'interno dello spazio delle feature. Se ho quindi un'immagine 1 è un'immagine 2 andrò a definire la distanza fra queste due immagini in termini delle feature che rappresentano le immagini stesse, quindi di nuovo intensità media standard deviation dell'intensità, istogramma e così via.

1.3.2 Supervised learning

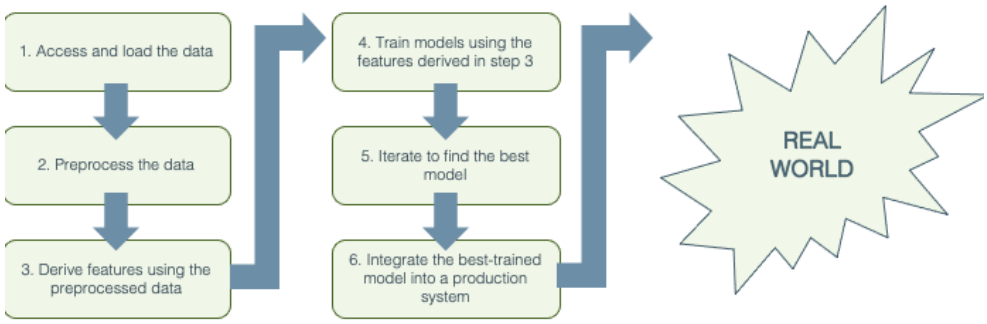
Nel caso invece dell'apprendimento supervisionato lo scopo è diverso in quanto voglio andare a definire una risposta di uscita dati dei dati di ingresso. Voglio sapere ad esempio se l'immagine rappresenta un cane un gatto, oppure se l'immagine rappresenta un bicchiere con 10 o 100 millilitri di acqua. I due tasks principali nel caso dell'apprendimento supervisionato sono da una parte la classificazione dall'altra parte la regressione. Come dicono le parole stesse classificazione vuol dire che io ho a che fare con delle classi quindi i modelli di classificazione "classificano" i dati di input in categorie; anche in questo caso ricordiamoci che i dati di input sono le nostre feature quindi il vettore di feature per tutte le immagini di training. Nel caso invece delle tecniche di regressione, facciamo qualcosa di diverso quindi non andiamo a classificare in categorie, ma andiamo a predire risposte di tipo continuo come, ad esempio, i cambi della temperatura oppure le fluttuazioni nella richiesta di energia. Sulla base di queste informazioni possiamo capire come scegliere il nostro algoritmo. La scelta dipende dalla tipologia di dati con cui stiamo lavorando, se sono categorici o numerici, e dalle informazioni che abbiamo a disposizione su quei dati. Nel caso in cui abbiamo a disposizione un valore vero e quindi possiamo allenare il nostro modello, utilizziamo degli approcci supervisionati altrimenti utilizziamo degli approcci non supervisionati. Inoltre, se abbiamo a che fare con dati categorici ci occuperemo di classificazioni e con dati numerici ci occuperemo invece di regressione.

Choose **supervised learning** if you need to train a model to make a prediction — for example, the future value of a continuous variable, such as temperature or a stock price, or a classification — for example, identify cars from webcam video footage.

Choose **unsupervised learning** if you need to explore your data and using a model to find a good internal representation, such as splitting data up into clusters.

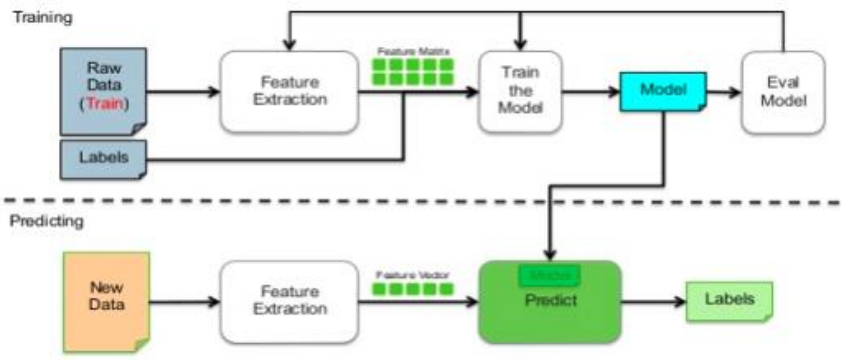
Di solito, nella pratica abbiamo i nostri dati che ad esempio scarichiamo da un cloud oppure abbiamo a disposizione dal gestore di un'azienda o da un clinico, processiamo i nostri dati con coscienza perché conosciamo o meglio dobbiamo preoccuparci di conoscere l'ambito in cui lavoriamo, calcoliamo le feature dai dati preprocessati. Alleniamo il nostro modello nel caso in cui abbiamo un tipo di apprendimento supervisionato con le feature calcolate continuiamo a iterare fin quando non troviamo il modello migliore quindi quello che ci permette di avere una buona accuratezza, un buon tempo di predizione, una buona interpretabilità che alla fine integriamo il nostro modello all'interno di un sistema di produzione ci spostiamo quindi nel mondo reale.

1.4 A focus on Supervised Learning

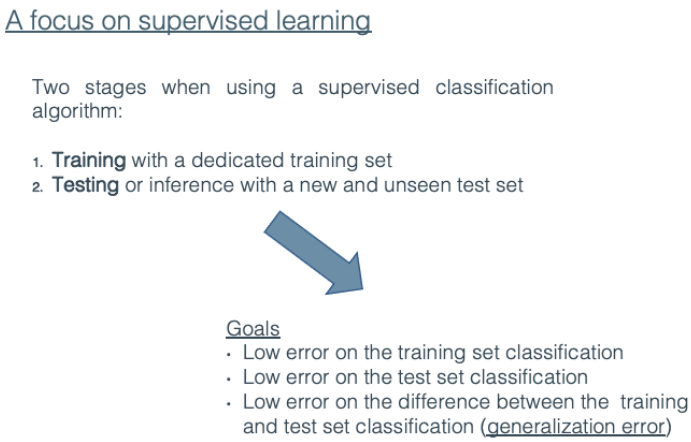


Prestiamo attenzione più nello specifico qual è il workflow che governa gli algoritmi di apprendimento supervisionato sia che questi siano di tipo machine learning o deep learning. Quando parliamo di apprendimento supervisionato parliamo di un apprendimento che coinvolge da una parte i dati grezzi quindi raw data che fanno parte del nostro set di training, dall'altra parte invece un set di label; infatti, ad ogni dato grezzo ad esempio ad ogni immagine abbiamo un label associato. Pensiamo di avere un dataset di immagini che mostrano animali, quando avremo l'immagine che mostra un cane il label corrispondente sarà "cane", l'immagine che mostra il gatto avrà come label corrispondente "gatto". Da questi raw data siccome adesso parliamo di machine learning e non di deep learning abbiamo una procedura chiamata feature extraction con cui andiamo ad estrarre e le nostre handcrafted features (quindi intensità media, standard deviation, istogramma dell'intensità istogramma dei gradienti orientati in generale qualsiasi tipo di feature possa essere di interesse per il campo che stiamo analizzando). Da questa procedura di feature extraction sarà prodotta una matrice di feature in cui ogni riga della matrice di feature è associata ad una determinata immagine di ingresso. Questa matrice di features assieme al vettore di label corrispondente verrà data in ingresso per il training del nostro modello. Alla fine di questa procedura di training avremo il nostro modello allenato che sarà in grado di darci una predizione. Tornando alle immagini di cane e gatto ci dirà se il risultato è cane, il risultato è gatto se il risultato è cavallo. A questo punto si passa alla valutazione del modello quindi ad esempio calcoleremo l'accuratezza di classificazione se questa accuratezza non ci soddisfa torneremo indietro o continuando ad allenare il nostro modello oppure andando a modificare il vettore di feature quindi ad esempio andando ad aggiungere o eliminare delle feature. Una volta che il modello invece ci fornirà una valutazione coerente con i nostri requisiti potremo finalmente utilizzare il modello in fase di predizione quindi nella realtà.

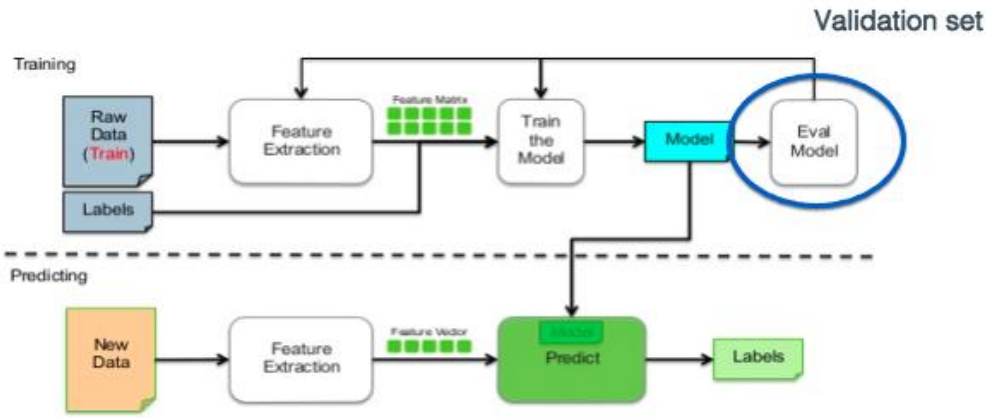
A questo punto ci arriva un nuovo dato quindi una nuova immagine che mostra un cane, passeremo alla fase anche in questo caso di feature extraction e attenzione che la fase di feature extraction deve essere esattamente la stessa che abbiamo eseguito durante il training sia in termini di tipo di feature che estraiamo sia in termini di ordine in cui presentiamo queste feature al nostro modello. Ovviamente in questo caso non avremo una matrice di feature, ma avremo un vettore di feature perché abbiamo solamente un'immagine in ingresso. Daremo in ingresso questo vettore di features al nostro modello predittivo che è stato allenato durante il training e in uscita otterremo il nostro label quindi se l'immagine in ingresso rappresentava un cane, il nostro label sarà "cane".



Il processo di apprendimento supervisionato consta di due fasi, il primo è una fase di training con un dataset dedicato chiamato training set che contiene al suo interno sia le immagini sia le etichette e una seconda fase invece di testing o inferenza in cui avremo dati nuovi e mai visti che andranno a diciamo rappresentare il nostro test set. Lo scopo in questo caso è avere un basso errore nella classificazione del training set, un basso errore nella classificazione del test set e anche un basso errore nella differenza fra la classificazione dei dati di training e dati di testing. Questo è chiamato errore di generalizzazione. Vogliamo un errore di generalizzazione basso perché nel momento in cui questo errore alto vuol dire che durante la fase di training abbiamo overfittato, quindi invece che andare a modellare i dati e le loro caratteristiche, il nostro modello si è preoccupato di andare a modellare le caratteristiche legate al rumore. Essendo il rumore diverso nei dati di training e di test questo produrrà un basso errore all'interno del test set.

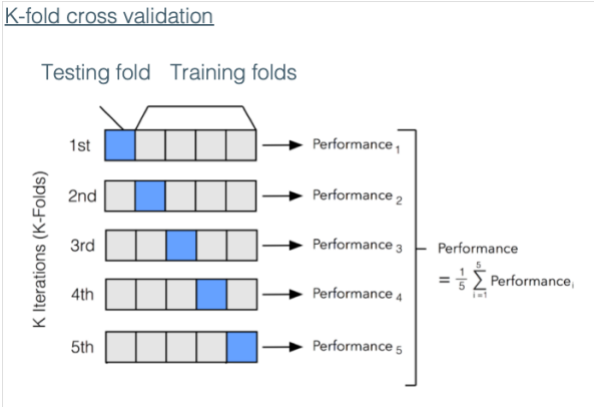


Vi è poi un ulteriore set che è detto set di validazione che serve a monitorare e ottimizzare il processo di allenamento. È chiaro, infatti, che se alleno il modello a partire dai dati del training set e quindi ottimizzo il mio modello sui dati del training set, le performance del modello stesso sul training set saranno alte perché sto ottimizzando sul training set ed è proprio qui che nasce il problema dell'overfitting. Infatti, il modello diventa talmente performante sui dati di training che quando arriva un nuovo dato il mio modello sbaglia a predire o a classificare o a regredire. Quindi quello che si può fare è prendere una parte del dataset di training, metterlo da parte quindi non lo utilizzo per ottimizzare il mio modello ma lo utilizzo per valutare invece il mio modello sempre durante la fase di training. In questo modo se le performance sul set di validazione saranno alte sul, mio validation set, il mio modello non avrà overfittato non sarà quindi andato a modellare il rumore presente nei dati di training. Se questo succede allora ho buone possibilità che anche in fase predittiva sul test set le performance del mio modello saranno alte quindi mio modello sarà un buon predittore. Ovviamente essendo il validation set una parte del training set, non utilizzata però per allenare il mio modello, avrò a disposizione sia i miei raw data sia le mie label quindi effettivamente sarò in grado di poter effettuare una validazione perché avrò da una parte le etichette reali dall'altra parte invece i risultati dati in uscita dal mio modello predittivo.



1.5 Hold-out crossvalidation e K-fold crossvalidation

Per dividere il mio data set in training set e validation set la procedura più utilizzata si chiama Hold-out cross validation in cui nella maggior parte dei casi abbiamo una percentuale di dati, ad esempio, il 70% usati per il training un 20% usato per la validation è un 10% usato invece come testing set. Queste percentuali sono date come esempio per avere un ordine di grandezza delle proporzioni che ci servono fra i vari set. Questa divisione è spesso utilizzata fra chi utilizza gli algoritmi di machine learning a meno che la numerosità del mio dataset è piccola. Infatti, se la numerosità del mio data set è piccola, di conseguenza, anche la numerosità del mio testing set sarà piccola e il problema è che io non posso costruire una statistica caso i miei risultati se ho solamente pochi esempi di testing. Quindi in caso in cui abbia una bassa numerosità di dati all'interno del mio set posso ricorrere alla cosiddetta che il K- fold cross validation dove K è un parametro che di solito vale 5 o 10 e in questo modo cerco di ovviare alla scarsità di dati del mio set. Per capire come funziona la K-fold cross validation riportiamo il seguente esempio: abbiamo 5 fold quindi ogni volta prendo il mio dataset lo divido in cinque parti e prendo un quinto come testing fold, quindi come testing set, e i restanti quattro quinti come training folds in questo modo avrò ovviamente per ogni testing fold delle performance, alla fine avrò ogni cinque performance, una per testing fold, e la performance del mio algoritmo di machine learning sarà dato dalla media di questa misura di performance.



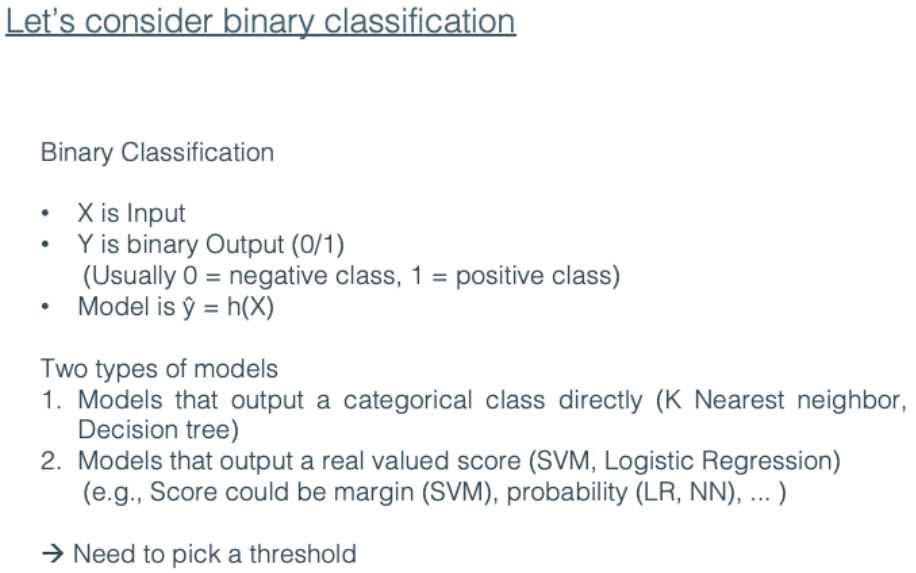
1.6 Binary and multi-class classification

Passiamo ad un concetto molto importante da considerare quando dobbiamo dividere il nostro set di dati nei se di training, testing e validation. All'interno del nostro dataset abbiamo diverse entità e supponiamo che per semplicità di avere un problema di tipo binario quindi classificazione binaria quindi “gatto-cane” e che ogni entità è associata a diversi immagini, ad esempio, ho diverse viste di un cane. È molto importante quando alleniamo e testiamo i nostri algoritmi di machine e deep learning separare le immagini a livello di entità e non a livello di immagine, questo per prevenire alcuni bias che cercheremo di capire in questa slide. Consideriamo che il nostro dataset formato da dieci immagini del cane A dieci immagini del cane B dieci immagini nel gatto C e dieci immagini del gatto D. Lo scopo in questo caso è appunto di effettuare una classificazione binaria quindi sviluppare un modello di machine learning che sia in grado di classificare se l'immagine mostra un gatto o un cane. Quello che potrebbe venire subito in mente è di mixare quindi unire tutte le immagini quindi le quaranta immagini del nostro dataset e poi tenerne il 70 % per il training e il 30% per il testing. Questa procedura è del tutto sbagliata perché potrei avere alla fine sette immagini del cane A in training e 3 immagini del cane A in testing, quindi stessa entità, ma immagini diverse. Questo è del tutto sbagliato perché consideriamo che il nostro classificatore allenato mi dia delle performance alte, 99% di accuratezza è una delle metriche che vedremo nella seconda parte di questo capitolo. Tuttavia, non posso fidarmi di questo risultato o perché non so se le feature che ho imparato sono discriminative per la classe cane in generale oppure sono specifiche per la classe o meglio per l'oggetto cane di tipo A. Ecco questa è la bias che vogliamo evitare, quindi la procedura corretta invece è quella di dividere a priori l'entità e quindi ad esempio utilizzare tutte le immagini del cane A e tutte le immagini del gatto C per il training e usare il resto quindi le dieci immagini del cane B e le dieci immagini del gatto D per testare il mio algoritmo.

1.7 Performance Evaluation metrics

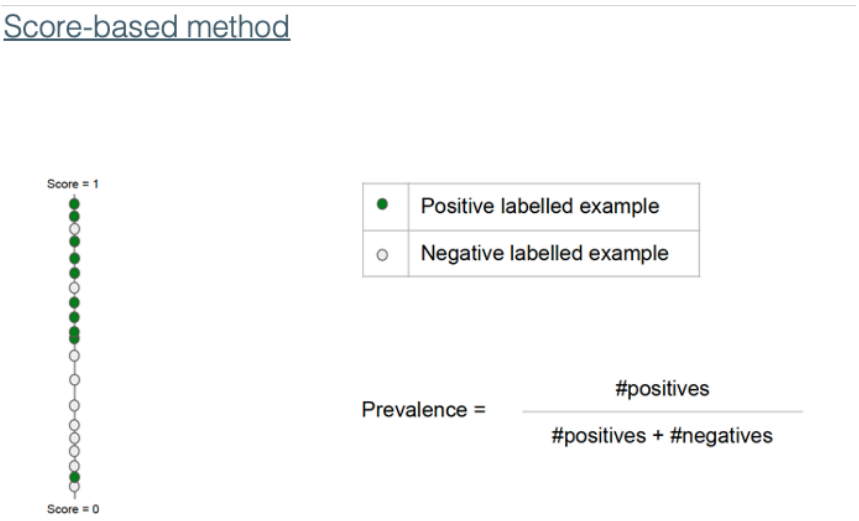
In questa sezione, tratteremo le metriche di valutazione dei nostri modelli di machine learning. Le metriche di performance sono importanti e non devono essere assolutamente sottovalutate innanzitutto perché ci servono per effettuare il nostro allenamento, quindi, rappresentano la cosiddetta funzione di costo. Durante l'allenamento quello che noi vogliamo fare minimizzare questo errore, questa funzione di costo questa loss che è rappresentata dalla differenza fra ciò che il nostro modello di machine learning predice e quello che è il label, l'etichetta che noi stiamo dando in ingresso al nostro modello di machine learning durante l'allenamento, perché parliamo di algoritmi tipo supervisionato. Inoltre, le metriche aiutano a catturare il goal della nostra analisi, se nel linguaggio naturale possiamo dire vogliamo effettuare una diagnosi o vogliamo raggiungere un determinato goal economico invece abbiamo bisogno di trasformare questa definizione qualitativa in un qualcosa di quantitativo in un target quantitativo. Inoltre, le metriche sono utili per quantificare la differenza che c'è fra le performance desiderate ad esempio un 98% di accuratezza nella classificazione gatto-cane e le performance invece correnti del nostro modello. Allo stesso modo le metriche ci aiutano anche a misurare il progresso del nostro modello nel tempo quindi ad esempio inizio con un determinato training ottengo un'accuratezza del 70%, aggiungo nuovi dati la mia accuratezza magari sale e supera il 70%, in questo modo tengo traccia nel tempo. Le metriche sono utili e importantissime per valutare il nostro modello probabilmente questa è la più scontata, cioè come faccio a dire se il mio modello funziona o non funziona? Come faccio a dire se sono soddisfatta del mio allenamento no? Come faccio a dire se il mio modello ha overfittato o no?

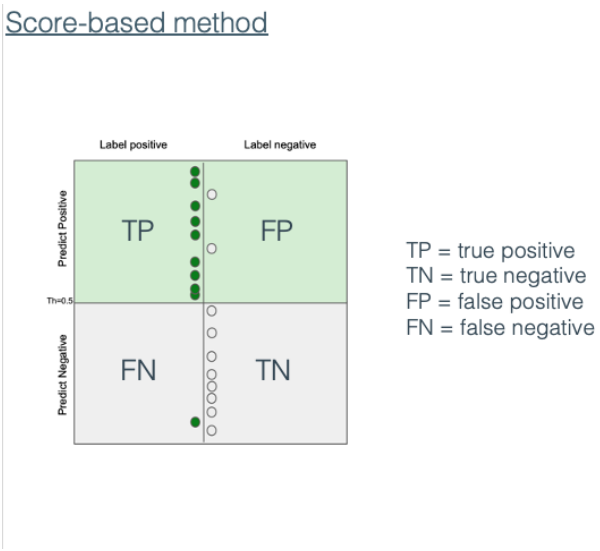
Per semplicità consideriamo un caso di classificazione binaria ovviamente esistono metriche anche per classificazioni multiclasse o per problemi di regressione. Nella classificazione binaria consideriamo x che è il nostro input e y che invece è il nostro output binario 0,1 di solito 0 fa riferimento ai casi negativi e 1 fa riferimento ai casi positivi. Quindi ad esempio se il nostro scopo fosse trovare un malfunzionamento di un macchinario in un'azienda questo malfunzionamento potrebbe essere il caso positivo 1; il nostro modello che sia un modello di machine learning deep learning o qualsiasi modello matematico quindi ci permette di avere dato in ingresso x, il nostro modello è h, e in uscita una y stimata che vogliamo ovviamente che sia tanto più vicina possibile alla y vera. Esistono due grandi tipi di classi di modelli, i modelli che danno direttamente in uscita una classe categorica e modelli che invece danno in uscita un valore reale, come ad esempio le SVM che forniscono dei margini oppure la logistic regression che fornisce invece una probabilità. In questo caso per i modelli che in uscita danno un valore numerico abbiamo bisogno di scegliere una soglia per dire a partire da questo valore numerico se sei sopra la soglia allora la tua classe è 1 altrimenti se sei sotto l tua classe è 0.



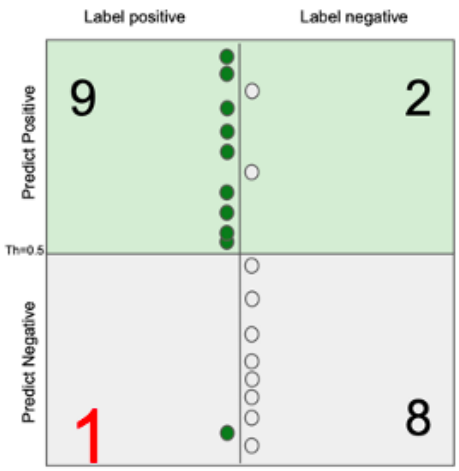
Innanzitutto, consideriamo nel diagramma qui sotto una serie di esempi quindi ogni pallino è un esempio e si passa dallo score 0 allo score 1. Nel mentre abbiamo tutte le nostre probabilità. Il colore dei pallini mi indica invece il label associato ad un determinato esempio quindi diciamo che le palline verdi sono le palline positive (il malfunzionamento della macchina oppure il paziente che ha un tumore) e le palline invece bianche sono quelle negative.

La **prevalenza** è uguale al rapporto fra il numero dei positivi è la somma dei positivi più i negativi. La matrice che vediamo sulla destra in figura, che ha due ingressi, si chiama matrice di confusione. Da una parte abbiamo il label positivi e negativi, quindi la verità il nostro groud truth, il nostro gol standard; dall'altra parte invece abbiamo la predizione che può essere positiva e negativa. Iniziamo dal primo quadrante in cui troviamo i nostri veri positivi "true positive". Un **true positive** si definisce come un caso positivo che è correttamente riconosciuto come positivo dal nostro algoritmo. Stessa cosa per i veri negativi "true negative" quindi sono i casi negativi, quindi ad esempio il paziente è sano, o il macchinario funziona, che sono correttamente classificati come tali, quindi per questo "veri negativi". Veri positivi e veri negativi si trovano sulla diagonale principale della matrice di confusione. Abbiamo però anche altri due ingressi che sono i falsi positivi i falsi negativi. I falsi positivi come dice la parola stessa hanno l'etichetta negativa ma sono riconosciuti in maniera errata dal nostro algoritmo di machine learning quindi sono riconosciuti come positivi. Mentre i falsi negativi sono degli esempi che di partenza che sono positivi, quindi un paziente malato, un macchinario rotto malfunzionante, sono però predetti in maniera errata dal nostro algoritmo di machine learning per quello si chiamano falsi negativi.



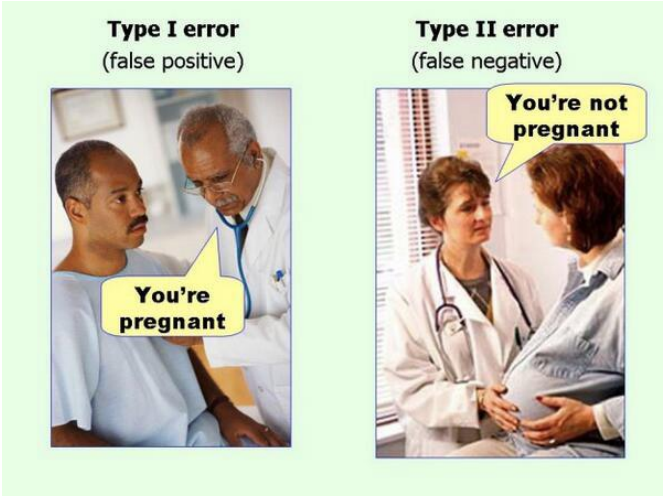


Consideriamo di avere una soglia di 0.5, valore dato assolutamente a caso, ovviamente se cambierà questa soglia cambierà anche il numero di casi considerati veri positivi vele negativi falsi positivi e falsi negativi. In questo caso con una soglia di 0,5 abbiamo 9 pallini verdi quindi 9 veri positivi; abbiamo 8 per i negativi perché abbiamo 8 pallini bianchi nel quadrato in basso a destra della nostra matrice di confusione, sempre sulla diagonale, abbiamo 2 falsi positivi è 1 falso negativo.



Th	TP	TN	FP	FN
0.5	9	8	2	1

L’ immagine seguente è abbastanza esplicativa e parla dei due tipi di errore il falso positivo è chiamato un errore di tipo 1 il falso negativo ha chiamato di un errore di tipo 2. Nell'errore di tipo 1 vediamo che il medico comunica al paziente che è incinta, ovviamente questo è un falso positivo dove positivo è associato al fatto di essere incinta; ovviamente un uomo non può essere incinta, nel secondo caso invece con l'errore di tipo 2 abbiamo un falso negativo quindi la dottoressa dice alla donna incinta di non essere incinta in realtà però questo è un falso perché la donna essendo incinta è una positiva.



L’accuratezza è la metrica un po’ più generale ed è definita come il rapporto fra la somma dei veri positivi più i veri negativi quindi true positive più true negative che troviamo di nuovo sulla diagonale principale della matrice di confusione. L'accuratezza è qiondi il rapporto fra la somma dei veri positivi e veri negativi e il totale di casi che abbiamo a disposizione, in questo caso dati di testing ovvero la somma tra i true positive, true negative, false positive, false negative.

Accuracy

Th	TP	TN	FP	FN	Acc
0.5	9	8	2	1	.85

$$Acc = \frac{TP + TN}{TP + TN + FN + FP}$$

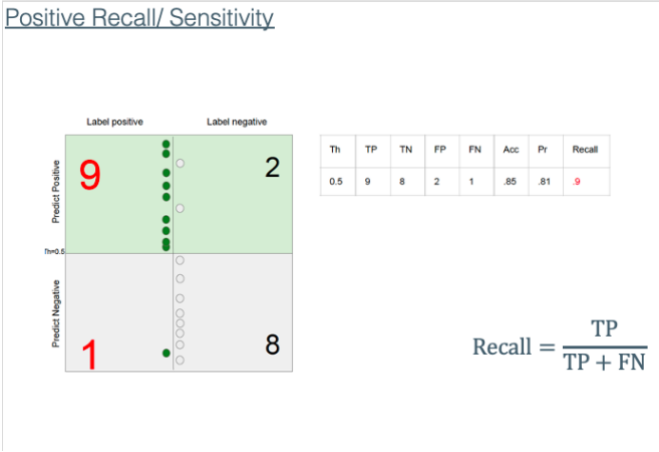
La **precision** è il rapporto fra true positive più false positive ma true positive più false positive che possiamo chiamarli predict positive. Quindi è il rapporto fra i veri positivi è la somma dei casi che sono stati i predetti come positivi.

Precision

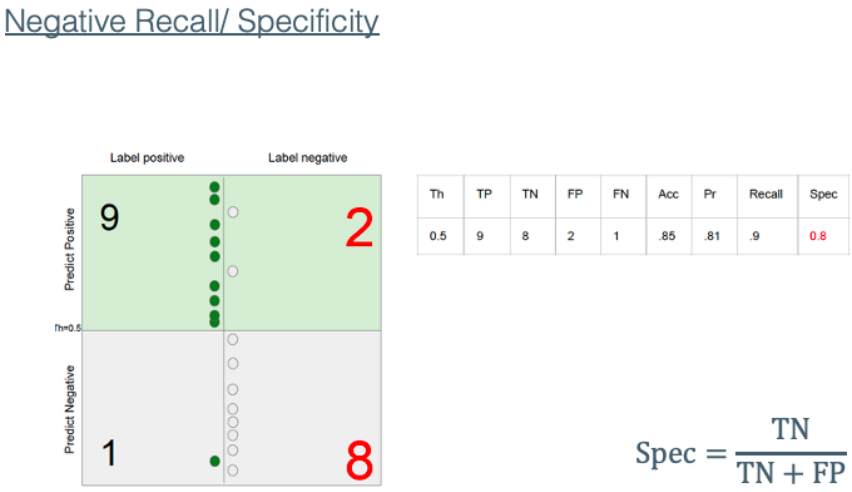
Th	TP	TN	FP	FN	Acc	Pr
0.5	9	8	2	1	.85	.81

$$Pr = \frac{TP}{TP + FP}$$

La **recall** o sensitività è definita come il rapporto fra i veri positivi e la somma dei veri positivi più i falsi negativi, quindi il rapporto fra i veri positivi e il totale di casi positivi. La recall è alta quando tende a 1 e quindi quando il numero di falsi negativi è basso. Quindi in realtà noi vogliamo avere un numero di casi positivi predetti invece come negativi che è basso. Questo è importante, ad esempio, nei test di screening in cui non mi importa di avere falsi positivi quello di cui mi voglio invece assicurare è che ci siano pochissimi casi di malati che vengono persi, poi sarà una seconda fase un secondo test uno screening di secondo livello a dirmi se in realtà quelli che io avevo predetto come positivi sono realmente positivi oppure no.

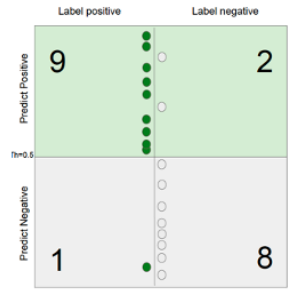


Un'altra metrica è la **specificità** che simile alla sensitività ma al posto dei positivi troviamo i negativi, infatti, la specificità è il rapporto fra i veri negativi è il totale di casi con un'etichetta negativa, quindi true negative più false positive. Esattamente come la sensitività in questo caso per la specificità se vogliamo avere una specificità alta quindi uguale a 1 quello che dobbiamo fare è avere un numero di falsi positivi basso.



Una metrica che combina le precedenti è l'**F1-score** che è definita come due volte il rapporto fra il prodotto della precision e della recall è la somma della precision e della recall. F1-score è particolarmente importante quando abbiamo a che fare con dati sbilanciati. Un dato è sbilanciato quando ha più casi di una classe rispetto ai casi delle altre e questa è una cosa che avviene spesso se pensiamo ai malfunzionamenti dei macchinari, piuttosto che alle malattie, arrivando addirittura le malattie rare. È chiaro che il numero di casi positivi è quasi sempre inferiore rispetto al numero dei casi negativi.

F1 score

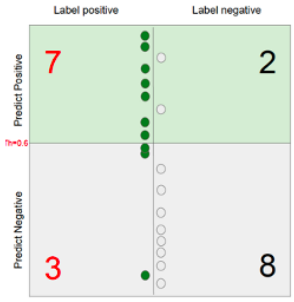


Th	TP	TN	FP	FN	Acc	Pr	Recall	Spec	F1
0.5	9	8	2	1	.85	.81	.9	.8	.857

$$F1 = 2 * \frac{Prec * Recall}{Prec + Recall}$$

Torniamo alla nostra soglia; prima abbiamo detto scegliamo 0.5 a caso adesso cambiamo la soglia la soglia da essere 0.5 diventa 0.6; cambia il numero di veri positivi numeri di veri negativi l'accuratezza la precision, la recall e l'F1-score. Cambiando la soglia ci possiamo aspettare che potrebbero cambiare anche veri positivi i falsi positivi e di conseguenza anche la specificità. In generale il concetto è che se cambia la soglia cambiano anche i valori di performance che calcoliamo.

When we change the threshold…

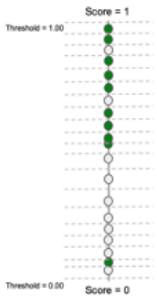


Th	TP	TN	FP	FN	Acc	Pr	Recall	Spec	F1
0.6	7	8	2	3	.75	.77	.7	.8	.733

… the performance values change!!!

Nella seguente immagine vediamo l'esempio di prima in cui a sinistra abbiamo ogni pallino che rappresenta un caso, i pallini verdi sono i casi positivi i pallini bianchi sono i casi negativi e ovviamente abbiamo uno score che alla varia da 0 a 1 potrebbe ad esempio rappresentare una probabilità, quindi, la probabilità con cui il mio classificatore assegna una classe piuttosto che un'altra.

When we change the threshold …

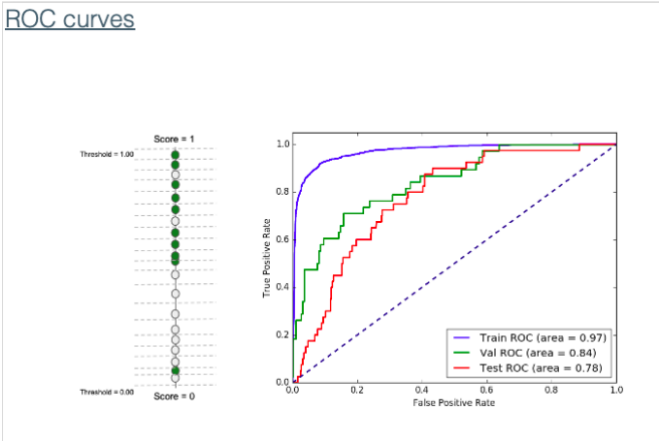


Threshold	TP	TN	FP	FN	Accuracy	Precision	Recall	Specificity	F1
1.00	0	10	0	10	0.50	1	0	1	0
0.95	1	10	0	9	0.55	1	0.1	1	0.182
0.90	2	10	0	8	0.60	1	0.2	1	0.333
0.85	2	9	1	8	0.55	0.667	0.2	0.9	0.308
0.80	3	9	1	7	0.60	0.750	0.3	0.9	0.429
0.75	4	9	1	6	0.65	0.800	0.4	0.9	0.533
0.70	5	9	1	5	0.70	0.833	0.5	0.9	0.625
0.65	5	8	2	5	0.65	0.714	0.5	0.8	0.588
0.60	6	8	2	4	0.70	0.750	0.6	0.8	0.667
0.55	7	8	2	3	0.75	0.778	0.7	0.8	0.737
0.50	8	8	2	2	0.80	0.800	0.8	0.8	0.800
0.45	9	8	2	1	0.85	0.818	0.9	0.8	0.857
0.40	9	7	3	1	0.80	0.750	0.9	0.7	0.818
0.35	9	6	4	1	0.75	0.692	0.9	0.6	0.783
0.30	9	5	5	1	0.70	0.643	0.9	0.5	0.750
0.25	9	4	6	1	0.65	0.600	0.9	0.4	0.720
0.20	9	3	7	1	0.60	0.562	0.9	0.3	0.692
0.15	9	2	8	1	0.55	0.529	0.9	0.2	0.667
0.10	9	1	9	1	0.50	0.500	0.9	0.1	0.643
0.05	10	1	9	0	0.55	0.526	1	0.1	0.690
0.00	10	0	10	0	0.50	0.500	1	0	0.667

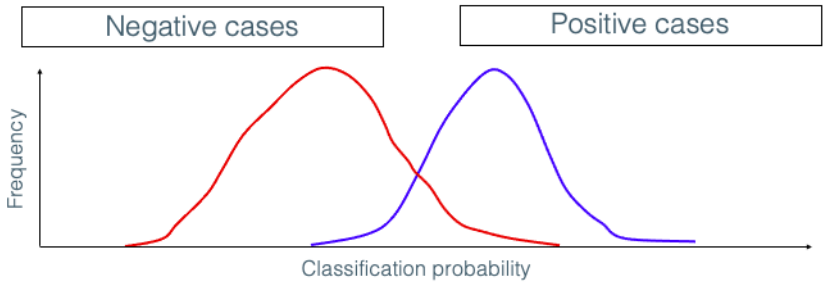
Nella tabella vediamo che al variare delle soglie da 1 a 0 con uno step di 0.05 i valori di true positive e true negative e false positive e false negative cambiano in continuazione quindi la nostra matrice di confusione cambia in continuazione e i valori delle nostre metriche cambiano.

1.8 ROC Curves

È quindi necessario definire come scelgo la soglia, come scelgo di imporre un valore di soglia da applicare all'output numerico probabilistico del mio classificatore. Esistono le curve ROC, illustrate nella seguente figura.

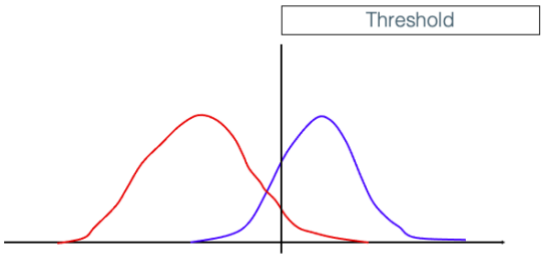


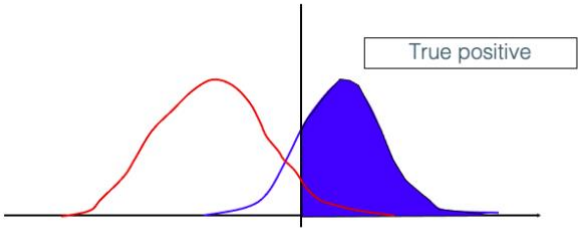
ROC sta per Receiver Operating Characteristic, in realtà sono anche delle curve abbastanza anziane nel senso che sono state sviluppate intorno agli anni 40 e 50 dello scorso secolo. Sono nate con l'analisi dei segnali e si sono spostate poi all'ambito medico ma sono in realtà adesso molto importanti anche nell'analisi delle immagini e anche nelle applicazioni di machine learning per valutare le performance il classificatore. Possono essere utilizzate per confrontare ad esempio diversi algoritmi machine learning ma questo vale in generale per tutte le metriche che abbiamo visto. Se osserviamo la seguente immagine abbiamo da una parte quindi sull'asse x la probabilità di classificazione e sull'asse y la frequenza con il numero di casi che presentano una determinata probabilità di classificazione. I casi blu sono i positivi e casi i rossi sono i negativi; quindi ad esempio positivi di nuovo un paziente malato oppure un malfunzionamento macchinario o nei casi negativi tutto va bene.

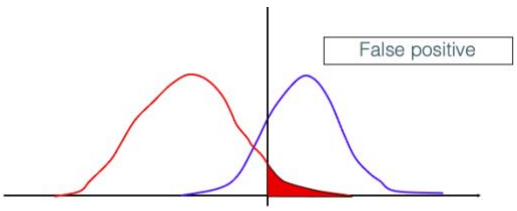


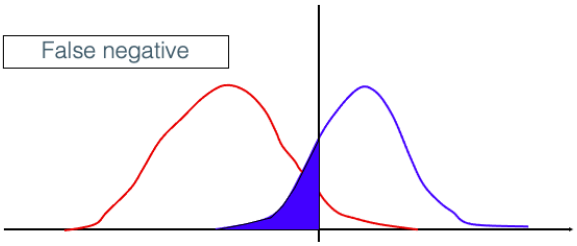
Distribution of test results (e.g., output classification probability)

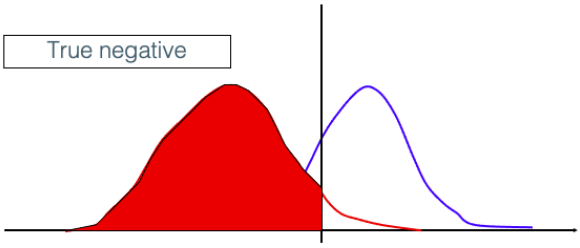
Fissiamo una threshold esattamente come abbiamo fatto prima, una caso. Avremmo quindi la rappresentazione grafica dei veri positivi dei falsi positivi dei veri negativi e dei falsi negativi.



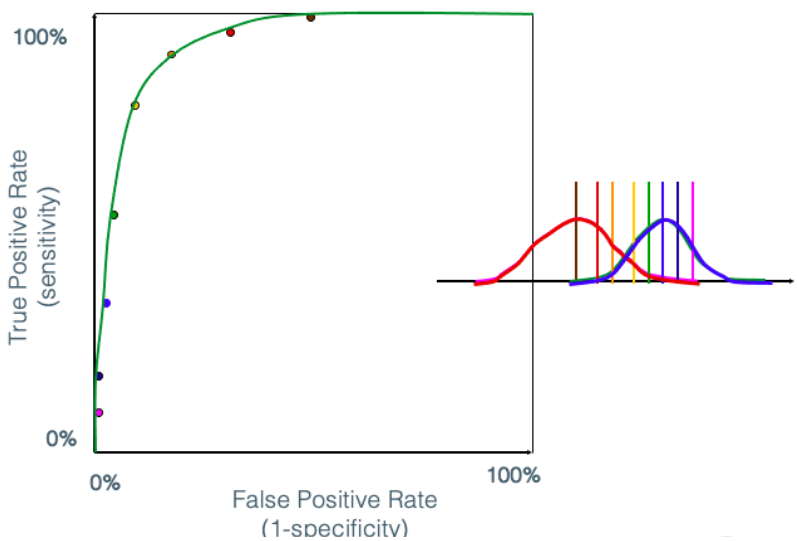




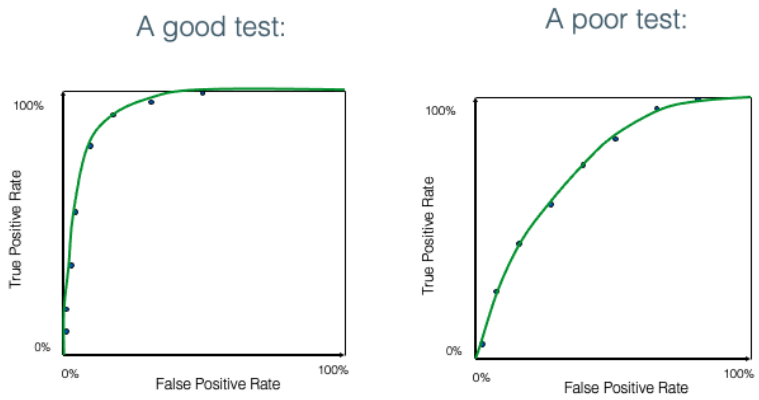




Se sposto la soglia le cose cambiano, cambiano i veri positivi i veri negativi falsi positivi e falsi negativi. Man mano che sposto la soglia quindi queste righe colorate verticali verso destra costruisco la mia curva ROC.



La curva ROC quindi è un grafico che viene ottenuto spostando cambiando i valori di un parametro di soglia applicato sull'uscita numerica del nostro classificatore, e che ha sull'asse x il false positive rate, quindi la frequenza dei falsi positivi anche chiamata **1-specificità** e sull'asse y true positive rate che corrisponde alla sensitività. Ovviamente la curva ROC è costruita con intervalli che variano da 0 a 1 oppure da 0 al 100% se lo vogliamo fare in percentuale. Attraverso le curve ROC possiamo capire quando abbiamo un buon test è quando un test è meno buono, quindi quando abbiamo un buon risultato del nostro algoritmo di machine learning e quando no. Possiamo considerare il nostro algoritmo di machine learning come un algoritmo con delle buone performance quando la curva ROC si avvicina all'angolo in alto a sinistra invece tanto più la curva ROC si adagia sulla bisettrice del primo e terzo quadrate tanto più il nostro test sarà vicino al caso quindi il nostro algoritmo di machine learning non sarà in grado di fornire dei risultati accettabili.



Per descrivere numericamente i risultati ottenuti tramite la ROC possiamo considerare l'area sottesa alla ROC quindi tutta l'area del grafico che si trova al di sotto della curva ROC. Quindi abbiamo una o un'area sotto la curva ROC che è uguale a 0.5 quando la curva ROC è adagiata sulla bisettrice del primo e terzo quadrante, l'area sarà 0.5 e il nostro algoritmo sarà assimilabile al caso. Abbiamo un'area sotto la ROC uguale a 1 quando abbiamo il miglior modello possibile quindi quando la nostra ROC è vicina all'angolo in alto a sinistra del grafico. Infine, abbiamo una curva ROC uguale a zero quando il modello sta confondendo le classi quindi predice benissimo i positivi come negativi e i negativi come positivi.

Area under the ROC curve

The area (AUC) under the ROC tells how much the model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting negative as negative and positive as positive.

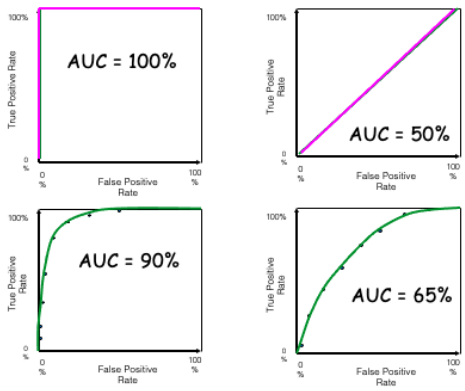
AUC = 0.5 → chance

AUC = 1 → best model possible

AUC = 0 → model is actually reciprocating the classes

Nella figura di seguito vediamo in alto vediamo un'area sotto la ROC del 100% quindi classificato da perfetto poi un'area sotto la rocca del 50% quindi il caso e poi dai esempi intermedi in cui abbiamo un 90% quindi vuol dire che il nostro classificatore abbastanza bravo a fare il suo lavoro a una ROC del con un out del 65 % in cui invece c'è ancora spazio per migliorare il nostro algoritmo di machine learning.

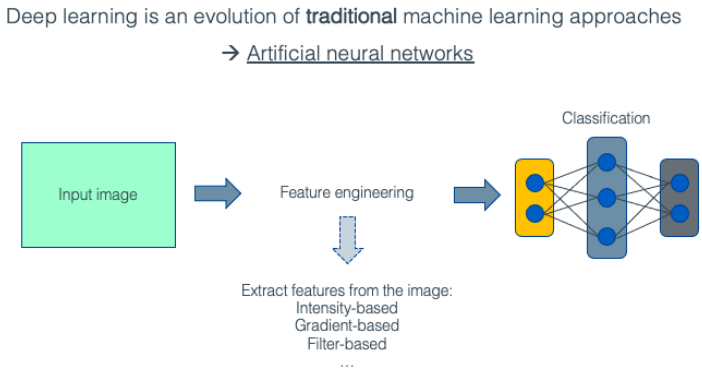
ROC curves



2. Deep Learning for Image Analysis

In questo capitolo, ci occuperemo del deep learning per l'analisi delle immagini. Partiremo con una breve introduzione sul deep learning, spostandoci poi al uno dei primi esempi di reti neurali artificiali e quindi il percettrone multistrato parleremo poi di features apprese automaticamente, in contrasto rispetto alle handcrafted features di cui abbiamo parlato con i metodi di machine learning, e infine vedremo i blocchi principali di una rete neurale convoluzionale. Il deep learning è una branca del machine learning che si basa sulle reti neurali artificiali. Nell'ultimo decennio le applicazioni di deep learning sono esplose sia a livello di ricerca sia a livello di applicazioni nella vita reale e questo perché effettivamente questi algoritmi hanno dimostrato di funzionare in maniera appropriata in diversi task della computer vision. Per questo questi algoritmi vengono ad oggi utilizzati in maniera sempre maggiore per l'analisi di video, immagini e in generale di dati multimediali. Le applicazioni includono la *classificazione* quindi data un'immagine capire se nell'immagine c'è un gatto, un cane o un cavallo; la *segmentazione*, quindi l'individuazione dei pixel dell'immagine che contengono il gatto o cavallo; altre applicazioni riguardano la *localizzazione* quindi capire dove il gatto è presente all'interno dell'immagine e *quantificazione* quindi ad esempio se è un gatto è più grande di un altro. Esistono diversi tipi di algoritmi che fanno parte della grande famiglia del deep learning e delle reti neurali artificiali. Focalizzeremo l'attenzione sul precettrone multistrato e poi le reti neurali convoluzionali, introducendo brevemente le reti generative di tipo avversario.

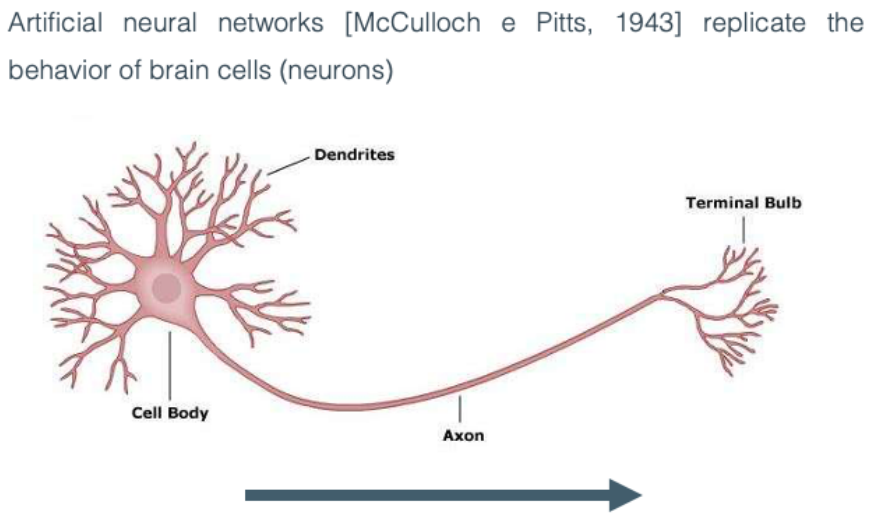
Esattamente come i metodi di machine learning, una rete artificiale può essere utilizzata come classificatore al posto, ad esempio, di una support vector machine o di un k-nearest neighbor o di un albero decisionale. Abbiamo la nostra immagine, abbiamo il nostro processo di estrazione di feature abbiamo la nostra classificazione. Lasciatemi ricordare che in questo caso il processo di estrazione di feature porta all'estrazione delle caratteristiche dell'immagine cosiddette handcrafted perché siamo noi a descrivere a decidere la formulazione matematica di queste features quindi ad esempio l'intensità media la standard deviation dell'intensità dell'immagine, ad esempio una per ogni canale rgb se abbiamo un'immagine a colori oppure l'istogramma dell'intensità, l'istogramma dei gradienti orientati possiamo avere una serie di uscite di filtri che estraggono i contorni delle immagini. Queste feature di nuovo vanno disegnate a seconda dell'ambito di cui ci si occupa e quindi richiedono una profonda conoscenza dell'ambito applicativo.



2.1 Neurone Biologico

La rete neurale artificiale nasce con una profonda ispirazione biologica, infatti, quello che vediamo nella seguente figura è un neurone, quindi una cellula del nostro sistema nervoso. Il neurone è formato da un corpo cellulare, alcuni dendriti che ricevono le informazioni dalle cellule circostanti e queste informazioni vengono in qualche modo elaborate dal corpo cellulare e la risposta io già poi attraverso un assone per arrivare fino ai bulbi terminali che porteranno l'informazione agli altri neuroni attraverso le sinapsi. Il cervello è composto da centinaia di miliardi di cellule nervose, i neuroni. Ogni volta che il cervello riceve uno stimolo sensoriale un gruppo di neuroni riceve un segnale, chiamato potenziale d'azione, se il potenziale di azione è abbastanza forte viene propagato ai neuroni vicini tramite dei canali chiamati sinapsi. Questo processo si ripete a cascata, neuroni attivano altri neuroni, fino a quando il segnale si esaurisce. I neuroni che si attivano insieme

si legano insieme, questo è il concetto alla base dell'apprendimento (apprendimento hebbiano), così i neuroni si uniscono in complessi reticoli che sono proprio le reti neurali, la sede di tutta la nostra conoscenza.



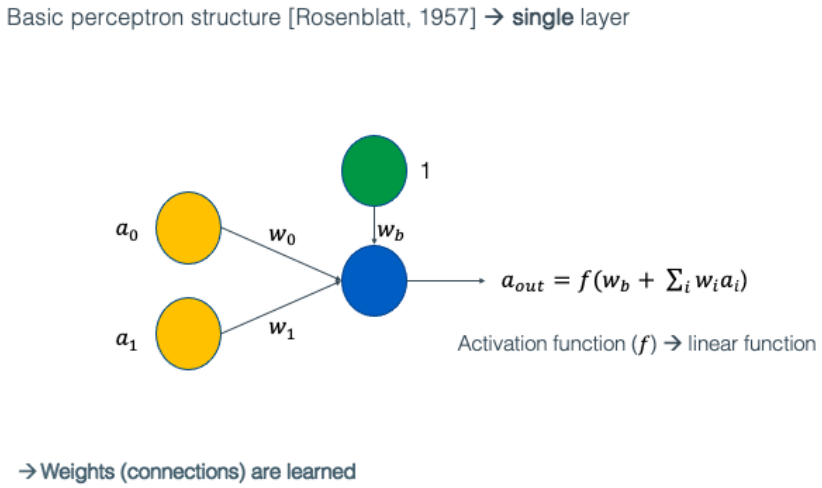
2.2 Neurone Artificiale

Se guardiamo quindi quella che è l'architettura biologica del neurone capiamo benissimo che possiamo riportarla all'architettura della nostra rete neurale artificiale basica. Le reti neurali artificiali utilizzano lo stesso approccio per permettere alle macchine di apprendere dai dati, cercando relazioni tra di essi e raggiungendo livelli di astrazione sempre più profondi. All'interno di una rete neurale artificiale i neuroni sono disposti su più strati:

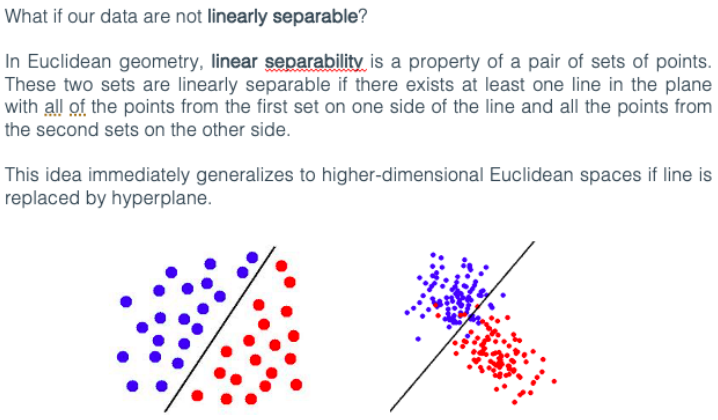
- Uno strato di input: che prende in ingresso i dati, ogni neurone di questo strato rappresenta una proprietà del dataset
- Uno strato di output: che fornisce il risultato della rete neurale
- Uno o più strati nascosti: che si trovano tra lo strato di input e quello di output. Il compito degli strati nascosti è di utilizzare le proprietà (features) del dataset per apprendere nuove proprietà.

Solitamente prima della fase di creazione di un modello di machine learning, il dataset viene analizzato e manipolato per estrarre proprietà significative da quelle di cui già si dispone. Questo processo prende il nome di feature extraction e ha un ruolo fondamentale per la creazione di un modello di machine learning robusto. Una rete neurale artificiale automatizza il processo di feature extraction all'interno di ogni strato nascosto, per questo motivo le reti neurali artificiali sono definite modelli end-to-end, appunto perché prendono come input i dati grezzi del dataset per svolgere attività come classificazione, regressione o clustering. Una rete è definita "profonda" (dall'inglese deep) quando contiene due o più strati nascosti, in questi casi la rete utilizzerà le proprietà apprese in uno strato nascosto per apprendere ulteriori nuove proprietà ancora più significative nello strato successivo, fino a utilizzarle per eseguire classificazione, regressione o clustering nello strato di output. Quindi qui vediamo la rete neurale artificiale più semplice che possiamo avere: abbiamo due ingressi a_0 e a_1 (vediamoli ad esempio come la media e la deviazione standard dell'intensità della nostra immagine) questi ingressi vengono pesati tramite due pesi w_0 e w_1 e dati in ingresso eventualmente con una bias. Nel caso più semplice questa funzione di attivazione è una funzione lineare. Quindi i dati in ingresso due feature a_0 e a_1 e dati due pesi w_0 e w_1 e poi ovviamente $w_b$quindi il peso della bias, a seconda del valore di questa sommatoria la nostra funzione di attivazione ci darà un'uscita a a_{out} diversa. Quello che è importante notare è che potrebbe essere del tutto non del tutto scontato è che i pesi quindi le connessioni degli ingressi quindi delle feature rispetto al corpo cellulare sono apprese durante l'allenamento esattamente come succede alle nostre connessioni cerebrali quindi ogni volta che apprendiamo qualcosa di nuovo si creano delle nuove sinapsi. Il significato dei pesi w_0 e w_1 essendo pesi abbiamo detto che pesano l'ingresso quindi ad esempio se nel mio problema di classificazione. La feature a_0 è più importante rispetto alla feature e a_1. Durante il mio processo di training il peso w_0 assumerà un valore maggiore rispetto al peso w_1. Per assurdo se la feature

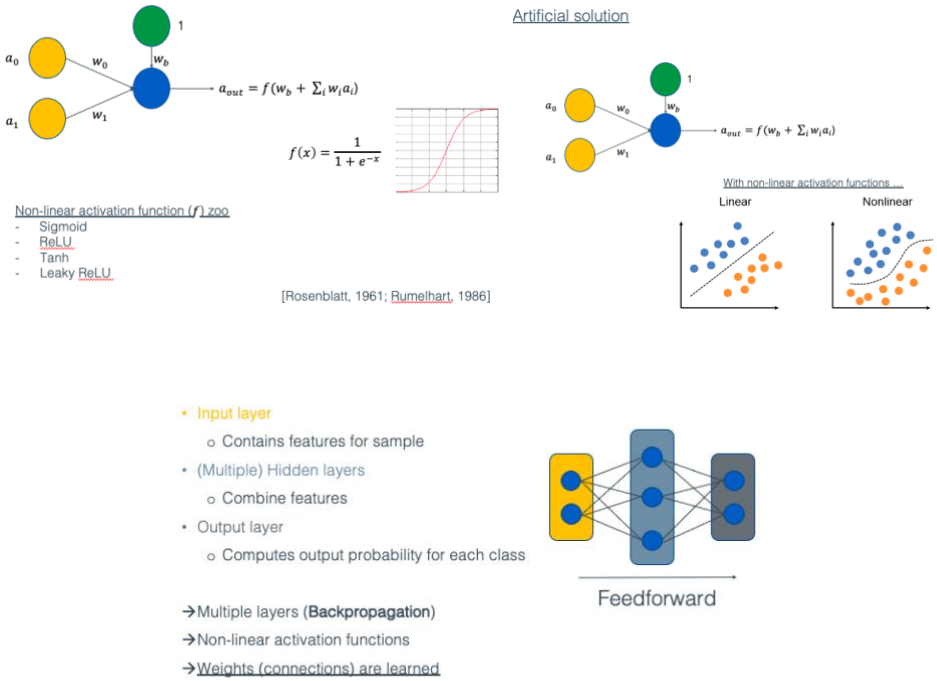
$a_1$fosse del tutto irrilevante per il mio task di classificazione il peso corrispondente w_1 tenderebbe ad avere valore nullo. La bias invece all'interno delle nostre reti neurali artificiali ha sempre valore unitario e viene pesata tramite il peso w_b. La bias all'interno di questi modelli assume un po' il valore dell'intercetta quando pensiamo ad una retta. Senza intercetta la retta a prescindere dalla sua inclinazione, dal suo coefficiente angolare passa sempre per l'origine utilizzando invece il parametro intercetta riusciamo a traslare in alto in basso nel diagramma cartesiano la nostra retta.



Il problema di avere una struttura come la precedente, abbastanza semplice, è che non è in grado di descrivere dei problemi complessi come quelli che poi avvengono nella vita reale. Infatti, dobbiamo parlare di una problematica che si chiama separabilità. Quindi nel caso in cui i dati siano separabili come nell'immagine di sinistra della seguente figura, ci basta una retta quindi una funzione lineare per dividere i puntini rossi da quelli blu. Nel momento in cui invece ci spostiamo a problematiche reali, quindi dove ad esempio abbiamo rumore o caratteristiche diverse ma simili delle classi che stiamo analizzando potrebbe succedere che una retta non sia più in grado di dividere pallini blu dai pallini rossi, come nell'immagine di destra in figura.



Dobbiamo immaginare che il piano su cui lavoriamo è sempre un piano bidimensionale dove troviamo ovviamente sull' asse x e sull'asse y feature 0 e feature 1, quindi di nuovo ad esempio media standard deviation delle intensità. Ovviamente quando parliamo di linearità questa cosa nel piano vuol dire avere una retta, se ci spostiamo in dimensioni maggiori vuol dire avere nel 3D un piano e poi nel 4D e superiori un iperpiano. Quindi per riuscire ad elaborare e a classificare dati che non sono linearmente separabili, la prima cosa che si fa è cambiare la funzione di attivazione, quindi, funzione di attivazione che non è più una retta ma diventa una funzione più complessa quindi ad esempio possiamo avere una sigmoide, oppure, altre funzioni come la tangente iperbolica o la ReLu che sta per Rectified Linear Unit e così via.

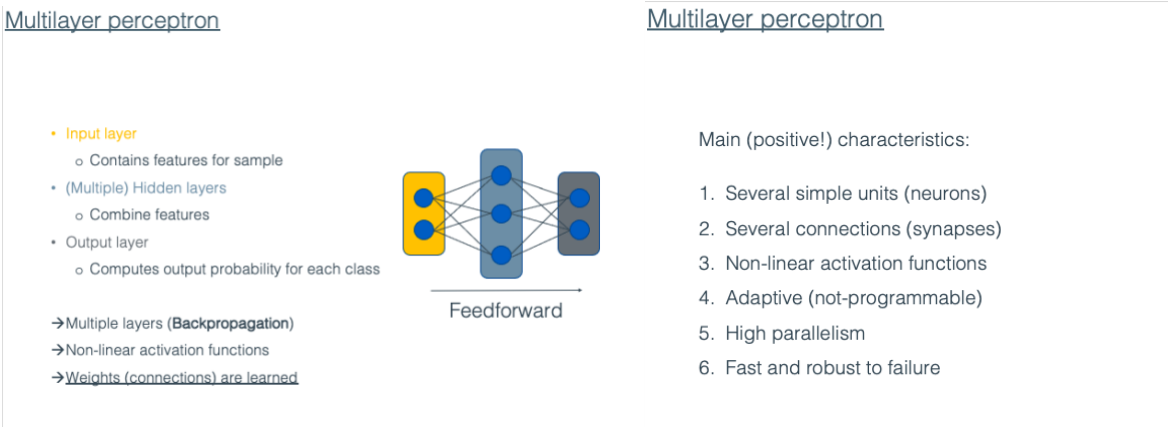


La seconda fonte di complessità della rete neurale artificiale può essere data dal numero di neuroni che vanno a rappresentare la nostra rete. Quindi nel caso precedente, avevamo un unico neurone blu che è il neurone di output che riceve le informazioni da due neuroni di input quindi a_0 e a_1. Se invece che avere solamente uno strato di input quindi uno strato di feature e uno strato di output quindi lo strato che ci dà la nostra classificazione andiamo inserire anche altri neuroni all'interno del cosiddetto strato nascosto che questo strato blu che vediamo in mezzo capiamo che ovviamente tanti più strati nascosti abbiamo tanto più siamo in grado di modellare problemi complessi. Qual è la problematica in questo caso? Non possiamo aggiungere un numero infinito di strati interni o nascosti perché dobbiamo sempre ricordarci che le connessioni fra i neuroni vengono apprese durante l'allenamento. Quindi tanti più strati interni abbiamo tante più connessioni abbiamo tanti più parametri e quindi pesi w_0 e w_1 dobbiamo imparare. Se non abbiamo a disposizione una sufficiente quantità di dati, la problematica che si presenta è la problematica di overfitting perché ho tantissimi parametri da imparare non abbastanza dati e quindi il mio allenamento non va a buon fine e questo si riduce o si riflette in un errore di generalizzazione basso.

2.3 Multilayer Perceptron

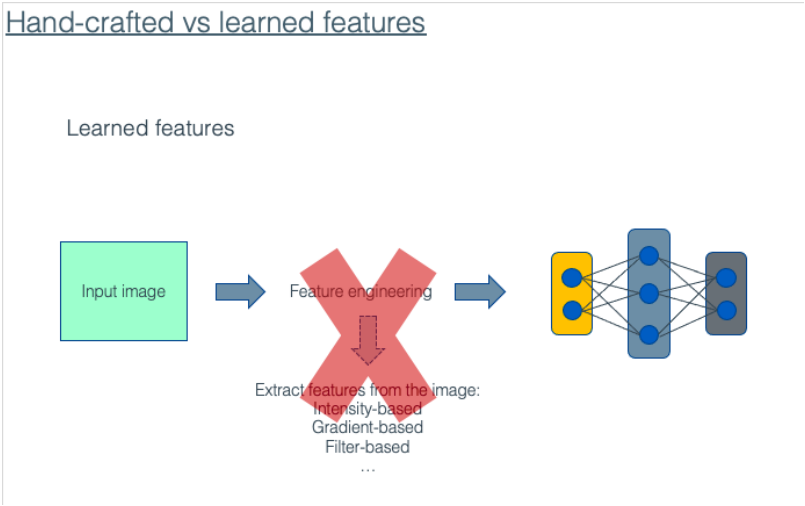
Il percettrone multistrato (in inglese "multilayer perceptron", MLP) è stata un po' un'innovazione dello scorso secolo ha diverse caratteristiche positive essendo una rete neurale artificiale composta da più strati di neuroni interconnessi. I neuroni in un MLP sono organizzati in modo che ciascun neurone di uno strato sia connesso a tutti i neuroni dello strato successivo. Ciò significa che l'output di ciascun neurone nello strato precedente viene propagato come input a tutti i neuroni dello strato successivo. Inoltre, ciascun neurone ha un peso associato che determina l'importanza dell'input che riceve. La rete MLP è in grado di apprendere relazioni complesse tra input e output, grazie alla sua capacità di apprendere rappresentazioni intermedie dei dati attraverso i diversi strati. La funzione di attivazione dei neuroni e i pesi associati alle connessioni sono determinati durante il processo di addestramento della rete, che prevede l'utilizzo di un algoritmo di apprendimento supervisionato. Abbiamo diverse unità semplici che sono i neuroni con diverse connessioni che sono le sinapsi tornando al confronto con l'ispirazione biologica delle reti neurali artificiali, abbiamo funzioni di attivazione non lineari e poi ovviamente sono caratterizzate dal fatto di essere algoritmi adattivi quindi che non sono problemi programmabili hanno un elevato parallelismo quindi nel caso in cui un neurone dovesse dare una risposta sbagliata ci sono tantissimi altri neuroni che possono sopperire a questa risposta

sbagliata esattamente come funziona il nostro cervello. Da questo altro parallelismo deriva il fatto che il percettrone è veloce è robusto al all'errore al fallimento.

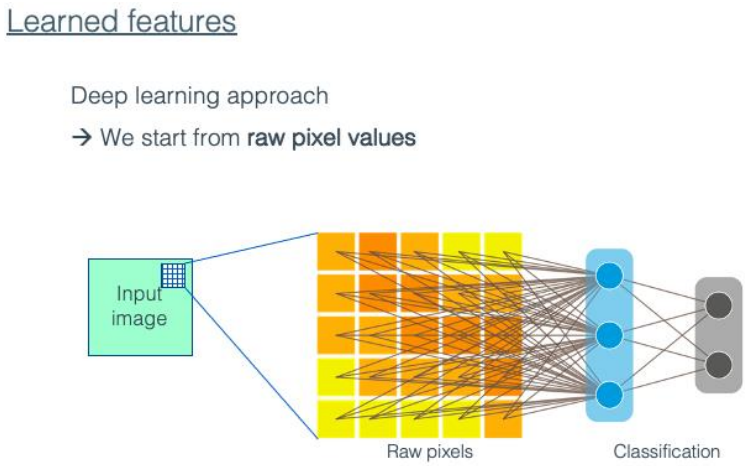


2.4 Hand-crafted vs learned features

Il passaggio di feature engineering quindi di design delle handcrafted feature scompare, quindi il nostro algoritmo di Deep Learning apprende le feature di interesse per un determinato problema direttamente a partire dall' immagine. Ora questo vuol dire vuol dire che se non siamo noi esseri umani dotati della nostra intelligenza a scegliere quali sono le feature di interesse e quindi se chiediamo all'algoritmo di scegliere da solo le feature di interesse all'interno dell'immagine capiamo che è la quantità di dati che dobbiamo fornire durante l'allenamento aumenta in maniera esponenziale; questo perché il problema è molto complesso ed essendo molto complesso essendo la variabilità all'interno dei dati molto alta, capiamo come per far fronte a questa variabilità da una parte c'è bisogno di un modello complesso quindi un algoritmo di deep learning complesso, dall'altra parte però abbiamo anche bisogno di una grande quantità di dati per far fronte a questa complessità.



Con gli approcci deep learning partiamo direttamente dai valori di pixel grezzi quindi dalle intensità grezze. Quindi avremo una prima fase in cui l'algoritmo di deep learning estrarrà in maniera automatica le feature dalla nostra immagine e la seconda fase in cui invece verrà effettuata la classificazione da queste feature esattamente come abbiamo visto con gli altri classificatori. Quello che cambia è che con le reti neurali convoluzionali non separiamo la fase di estrazione delle feature dalla fase di classificazione ma entrambe anche se in maniera sequenziale vengono effettuate all'interno dello stesso algoritmo.



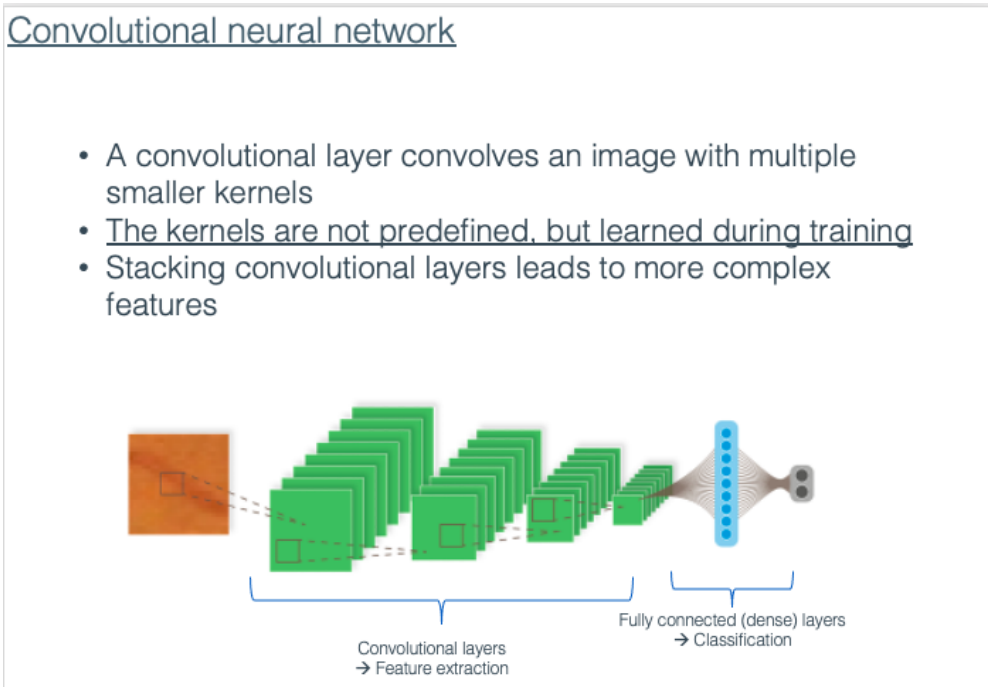
2.4 Convolutional Neural Network

Una rete neurale convoluzionale (in inglese "Convolutional Neural Network", CNN) è un tipo di rete neurale artificiale che è stata progettata per l'elaborazione di dati che hanno una struttura di griglia, come le immagini. Le CNN utilizzano una tecnica chiamata convoluzione, che prevede l'applicazione di filtri (o kernel) su una finestra mobile dell'input. I filtri sono matrici di pesi che eseguono una somma ponderata dei pixel sottostanti e producono un output che rappresenta una caratteristica specifica dell'immagine, come un bordo, una texture o un colore. La convoluzione viene applicata ripetutamente su tutto l'input, con differenti filtri e differenti dimensioni della finestra, per estrarre molteplici caratteristiche. Dopo l'estrazione delle caratteristiche, le CNN utilizzano una serie di strati di pooling per ridurre la dimensione dell'output e rendere la rete più efficiente dal punto di vista computazionale. Infine, i dati vengono passati attraverso uno o più strati di neuroni completamente connessi per la classificazione o la regressione. Le CNN sono state utilizzate con successo in molte applicazioni di elaborazione delle immagini, come il riconoscimento di oggetti, la classificazione di immagini, la segmentazione delle immagini e il riconoscimento di volti.
Le CNN sono costituite da diversi strati, ognuno dei quali esegue una particolare operazione sull'input. In generale, una CNN è composta dai seguenti strati:

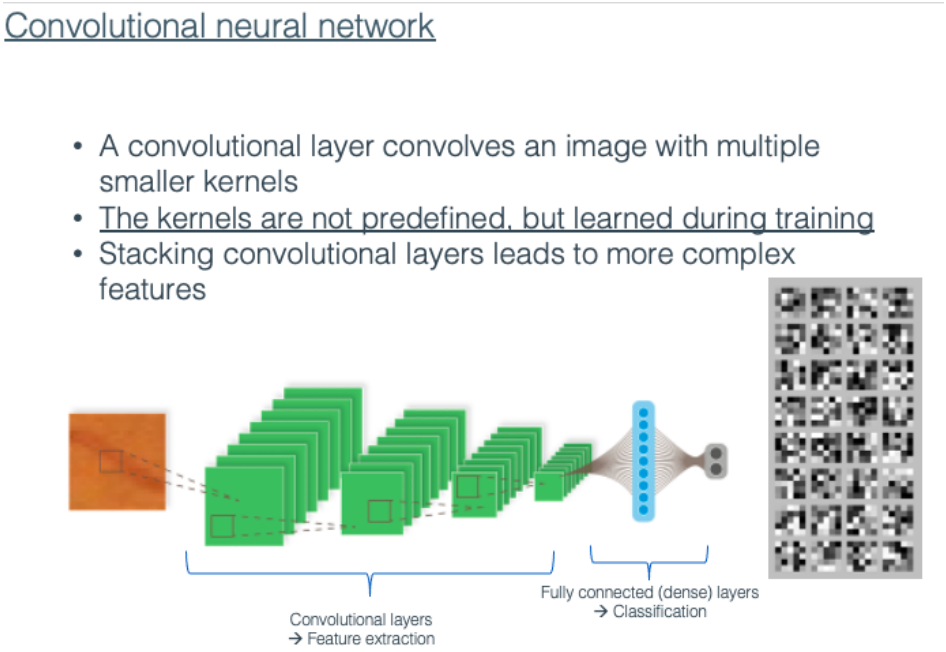
- Strato di input: accetta l'immagine di input e la trasforma in un tensore multidimensionale, solitamente con dimensioni (larghezza, altezza, canali).
- Strati di convoluzione: applicano una serie di filtri (o kernel) all'immagine di input per estrarre le caratteristiche salienti dell'immagine stessa. In ogni strato di convoluzione, ogni filtro viene applicato all'immagine di input e viene prodotto un'immagine di output chiamata "mappa di attivazione". Questi strati possono essere ripetuti più volte per estrarre caratteristiche sempre più complesse dell'immagine.
- Strati di attivazione: dopo ogni strato di convoluzione, viene applicata una funzione di attivazione non lineare (come la funzione ReLU) per introdurre la non linearità e migliorare la capacità di generalizzazione del modello.
- Strati di pooling: riducono la dimensione dell'output dei precedenti strati di convoluzione e semplificano la rappresentazione delle caratteristiche dell'immagine. I due tipi di pooling più comuni sono il max pooling e l'average pooling.
- Strato di flatten: converte l'output dei precedenti strati di pooling in un vettore unidimensionale.
- Strati fully connected: eseguono l'elaborazione finale del vettore di output e producono l'output finale del modello.
- Strato di output: produce l'output finale della rete neurale, che può essere una singola classe (ad esempio, nel caso di un problema di classificazione) o un vettore di valori (ad esempio, nel caso di un problema di regressione).

In sintesi, una CNN è costituita da diversi strati che lavorano insieme per estrarre le caratteristiche salienti dell'immagine di input e produrre l'output finale del modello.

Osserviamo la seguente immagine:



Questa è la versione diciamo più semplice e schematizzata di una rete neurale convoluzionale. Abbiamo a sinistra un'immagine qualsiasi che viene data in ingresso ad una rete neurale convoluzionale; vediamo una serie di strati chiamati convoluzionali quindi quei quadrati verdi che vediamo ovviamente consecutivi che servono ad effettuare l'estrazione delle feature. Quindi immaginiamoci invece che un ingegnere che decide dall'immagine di estrarre la standard deviation o l'intensità media, chieda agli strati convoluzionali di estrarre in maniera automatica informazioni rilevanti dall'immagine. Queste feature una volta estratte dovranno essere classificate. Ad esempio diciamo che in questa immagine c'è un vaso sanguigno oppure non c'è quindi è un problema di tipo binario allora a partire dalle feature la rete estrarrà in maniera automatica queste feature, queste feature verranno date in ingresso al classificatore, il classificatore che all'interno di una rete si chiama strato fully connected quindi se vogliamo completamente connesso. All'uscita degli strati fully connected o anche densi per il fatto che appunto sono tutti connessi fra di loro avremo in questo caso due uscite perché il nostro problema di classificazione un problema di classificazione è binario quindi dobbiamo immaginarci che i due neuroni grigi che vediamo alla fine saranno uno dedicato alla classe 0 "non c'è il vaso" e l'altra dedicata alla classe 1 "c'è il vaso". Ognuno ci darà in uscita la probabilità che l'immagine contenga o meno un vaso; quindi ad esempio se questa immagine ha un vaso ci auguriamo che l'uscita sia 0,95 per il neurone che si occupa della classe "è vaso" e 0.05 per il neurone che si occupa invece della classe "non vaso". Verrà poi utilizzata una threshold da applicare a questa probabilità che poi mi dirà se l'uscita è 0 o 1; di solito quello che si fa è prendere la probabilità massima che in questo caso era già 0.95 per la classe 1 e quindi assegnare all'immagine la classe "c'è il vaso". Quando parliamo di percettroni quindi di reti neurali artificiali semplici (non reti convoluzionali!) quello che si apprende durante l'allenamento sono i valori delle connessioni fra i neuroni e quindi cosa viene appreso quando parlo di una rete neurale convoluzionale? Verranno sicuramente apprese le connessioni dagli strati fully connected perché sono semplicemente dei neuroni così come li abbiamo visti per il percettrone ma in maniera ancora più importante vengono appresi i valori dei kernel degli stati convoluzionali. Infatti, quelli che vedete a destra in bianco e nero nella seguente figura sono delle rappresentazioni dei kernel che fanno parte degli stati convoluzionali. Il kernel rappresenta una matrice di pesi utilizzata per applicare la convoluzione su un'immagine o su altri dati strutturati a griglia, allo scopo di estrarre le caratteristiche salienti del dato. Il kernel può avere diverse dimensioni e forme, e i valori dei suoi pesi sono determinati durante il processo di addestramento della rete, allo scopo di massimizzare la performance del modello. L'utilizzo del kernel consente di eseguire l'elaborazione in maniera efficiente, riducendo il numero di parametri della rete e permettendo una maggiore generalizzazione delle caratteristiche estratte.

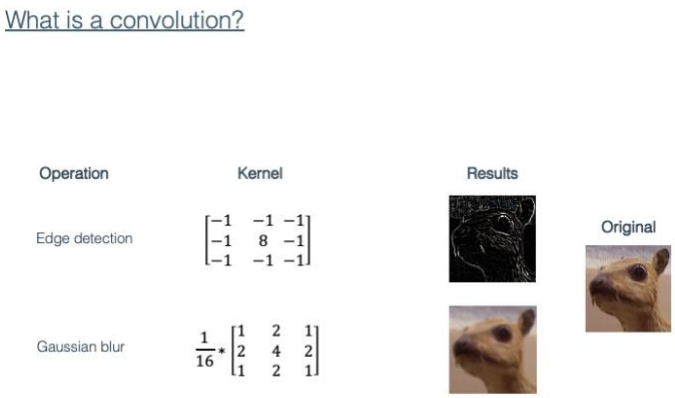


Come avevamo detto in precedenza, una delle problematiche che ci si deve porre quando si sviluppa un algoritmo di machine learning è capire se il nostro interlocutore è interessato meno interpretare in maniera semplice il risultato del nostro algoritmo. Ecco con le reti neurali artificiali convoluzionali questa cosa non si può sempre fare quindi non riusciamo a spiegare perché è un determinato kernel degli strati convoluzionali ha quel determinato valore. Nonostante questo, però la rete riesce ad ottenere dei risultati accettabili e migliori rispetto agli algoritmi standard di machine learning. In realtà e si fa di nuovo un salto indietro rispetto all'ispirazione biologica di questi modelli, quello che si è visto è che gli strati più superficiali della rete (quelli più vicini alle immagini in ingresso) sono in grado di estrarre informazioni più semplici come ad esempio i contorni o i contrasti all'interno delle immagini. Via via che si scende in profondità degli strati convoluzionali questa informazione diventa sempre più astratta ed è esattamente quello che succede all'interno del nostro sistema nervoso. Partendo dalla retina che con i suoi coni e bastoncelli è in grado di distinguere i contrasti ed è in grado di distinguere i colori quando invece si sale su fino ad arrivare alla corteccia cerebrale questa informazione diventa sempre più astratta. Il massimo dell'astrazione si ottiene all'interno degli strati densi fully connected in cui perdo del tutto la correlazione rispetto alla posizione dei pixel nell'immagine perché appunto nei fully connected in realtà ho delle connessioni fra i neuroni quindi qualcosa di completamente slegato rispetto alla convoluzione, la posizione di un determinato oggetto nell'immagine.

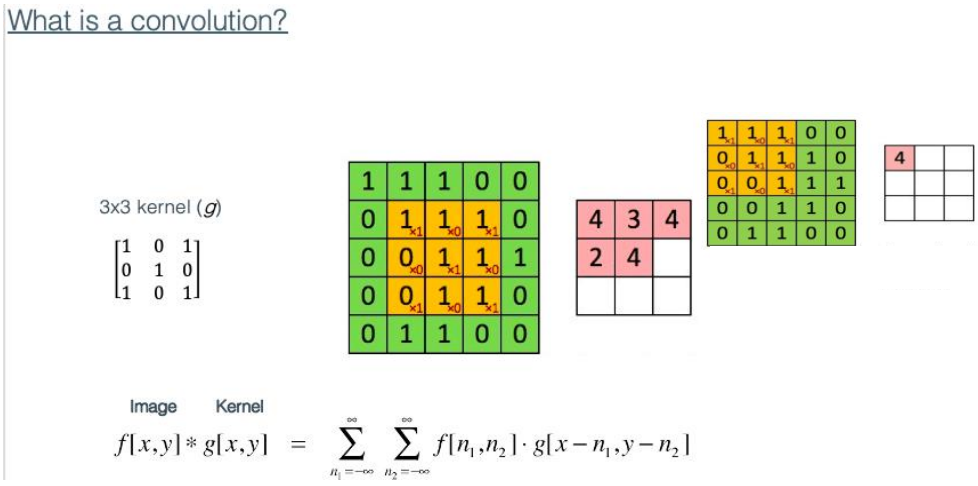
2.4.1 La convoluzione

Nel contesto del deep learning, la convoluzione è un'operazione fondamentale utilizzata nelle reti neurali convoluzionali (CNN) per l'elaborazione delle immagini. In una CNN, la convoluzione viene utilizzata per applicare un insieme di filtri (o kernel) all'immagine di input, con lo scopo di estrarre le caratteristiche salienti dell'immagine stessa. Ogni filtro rappresenta una piccola finestra di pixel che viene spostata su tutta l'immagine in modo da eseguire l'operazione di convoluzione su ogni parte dell'immagine. Durante l'operazione di convoluzione, il valore di ogni pixel nell'output viene calcolato come la somma pesata dei valori dei pixel dell'input corrispondenti, secondo i pesi definiti dal kernel. Questo processo consente di evidenziare le caratteristiche dell'immagine, come ad esempio i bordi, le texture, le forme, etc. Dopo l'operazione di convoluzione, l'output viene passato attraverso un'attivazione non lineare, come la funzione ReLU, che consente di introdurre la non linearità e migliorare la capacità di generalizzazione del modello. Le reti neurali convoluzionali utilizzano diverse operazioni di convoluzione in sequenza, con diversi filtri e dimensioni della finestra, per estrarre caratteristiche sempre più complesse dell'immagine. Inoltre, le CNN utilizzano anche tecniche di pooling per ridurre la dimensione dell'output e rendere la rete più efficiente dal punto di vista computazionale. In sintesi, la convoluzione rappresenta un'operazione fondamentale nelle reti neurali convoluzionali per l'elaborazione delle immagini, consentendo di estrarre le caratteristiche salienti dell'immagine e migliorare la capacità di generalizzazione del modello.

Prendiamo ad esempio un tipo di operatore come il detettore di contorni e per andare a ricercare i contorni all'interno di un'immagine posso ad esempio utilizzare un Kernel come quello riportato nella seguente figura quindi con un 8 al centro e tutti -1 ai lati.



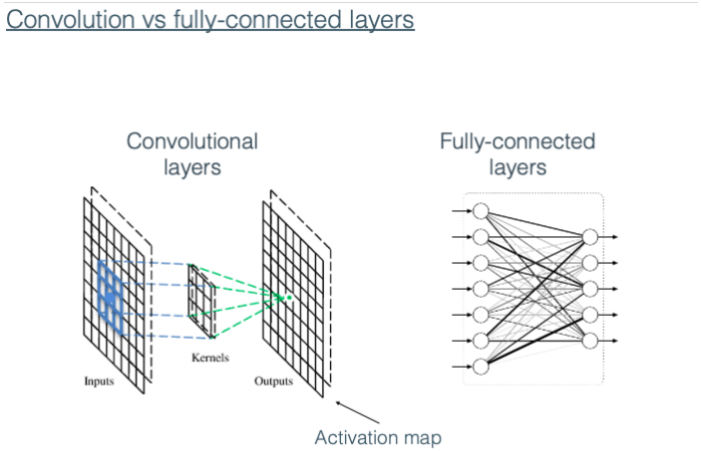
Data l'immagine originale quello che otteniamo è un risultato con i contorni in bianco proprio perché il filtro è stato attivato in presenza di un contorno, invece, laddove non ci sono contorni ma ci sono zone un'intensità omogenea il risultato della convoluzione dell'immagine col filtro è zero perché il filtro non si è attivato non essendoci contorni. Un altro filtro importante è il filtro gaussiano che fondamentalmente sfoca l'immagine per questo si chiama Gaussian blur, perché il blur è la sfocatura dell'immagine. Quindi fondamentalmente stiamo pesando l'intensità di ogni pixel considerando anche i pixel nel contorno, quindi, stiamo facendo in questo caso una media dell'intensità. Ma che cos'è la convoluzione e come funziona? Innanzitutto, vediamo che la convoluzione richiede la definizione di un filtro quindi questa matrice che vediamo nella seguente immagine, chiamiamola g è una matrice in questo caso di dimensioni 3 x 3; la dimensione del Kernel si chiama campo recettivo e ovviamente questa dimensione può variare in questo caso diamo come esempio una dimensione appunto 3x3. Il kernel corrisponde a questa matrice arancione che scorre invece sull'immagine che è questa matrice verde 5x5. Come funziona la convoluzione? Ogni volta che il kernel viene sovrapposto ad una porzione dell'immagine si effettua una moltiplicazione pixel a pixel dei pixel dell'immagine con i pixel del kernel. Alla fine, il risultato di questa moltiplicazione pixel a pixel viene sommato e viene assegnato al valore centrale diciamo dell'area occupata dal kernel sovrapposto rispetto all'immagine.



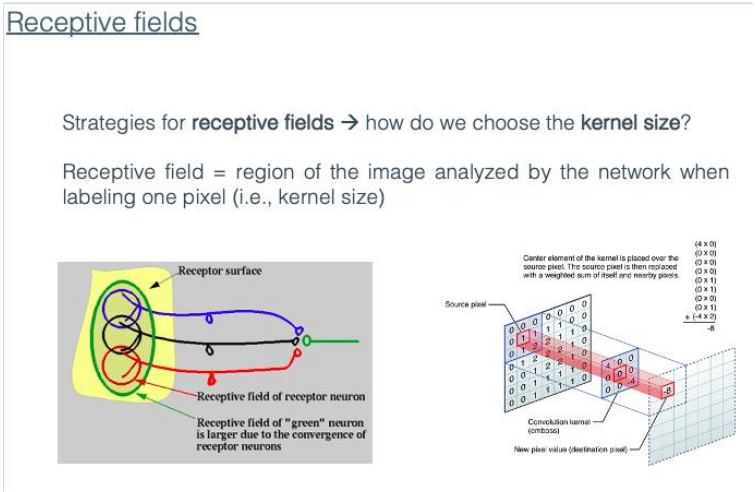
2.4.2 Strati Fully Connected

Gli strati fully connected (o densi) sono uno dei tipi di strati utilizzati nelle reti neurali, compresi le reti neurali convoluzionali (CNN). Gli strati fully connected sono composti da un insieme di neuroni collegati a tutti i neuroni del livello precedente. In pratica, gli strati fully connected prendono in input un vettore di feature estratte dai livelli precedenti e producono un vettore di output, che viene poi passato allo strato successivo.

Ogni neurone del livello fully connected è collegato a tutti i neuroni del livello precedente mediante un peso, che rappresenta l'importanza del contributo di ogni neurone di input al neurone di output. Il numero di neuroni negli strati fully connected dipende dalle dimensioni dell'input e dell'output della rete neurale. Solitamente, gli strati fully connected hanno un grande numero di parametri, poiché ogni neurone del livello fully connected è collegato a tutti i neuroni del livello precedente. Gli strati fully connected sono utilizzati soprattutto nei problemi di classificazione, in cui l'obiettivo è classificare l'input in una delle diverse classi possibili. In questo caso, l'output dell'ultimo strato fully connected è solitamente passato attraverso una funzione di attivazione softmax, che produce un vettore di probabilità per ogni classe possibile. Immaginiamo di avere l'uscita di una convoluzione di alcuni degli strati convoluzionali della mia rete neurale. Quello che succede con uno strato denso è che ognuno dei pixel di output dalla convoluzione, questa si chiama mappa di attivazione proprio perché è stata attivata la un kernel, viene dato un ingresso ad uno dei neuroni dello strato fully connected quindi capiamo che la numerosità è elevatissima e gli strati che ricevono questo ingresso lo pesano, quindi che ricevono in ingresso il valore di ogni pixel della mappa di attivazione. Ricevono questo ingresso e lo danno a loro volta attraverso la una funzione di attivazione ai neuroni dello strato successivo e tutti i neuroni all'interno degli strati fully connected sono a loro volta tutti connessi uno fra l'altro e i valori di queste connessioni quindi i pesi vanno appresi durante l'allenamento. Quindi per riassumere quando alleniamo una rete neurale convoluzionale quello che andiamo a fare è per prima cosa, assegnare dei valori ai kernel degli strati convoluzionali e come secondo step assegnare dei valori alle connessioni degli strati fully connected.

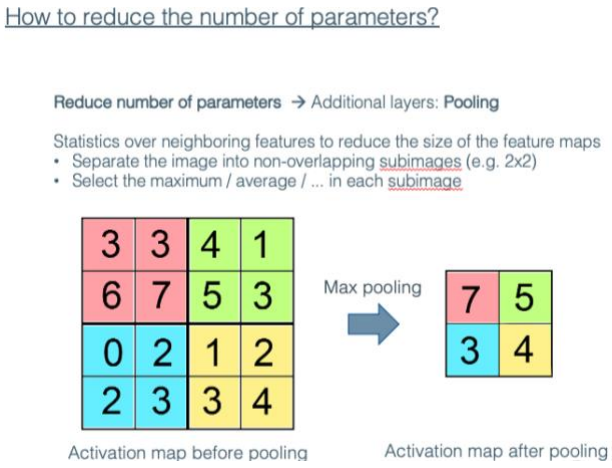


È però importante capire come scegliere la dimensione del kernel perché ricordiamoci quello che impariamo noi è il valore diciamo dei pixel del kernel, quello che invece fissiamo a priori e il numero di strati della rete, il numero di kernel per stato è la dimensione del kernel. E questa è detta architettura della mia rete neurale. La dimensione del kernel è legata al chiama campo recettivo tanto più è grande il kernel tanto maggiore sarà l'area che sarà interessata dalla convoluzione. Tanto più piccolo il kernel tanto minore sarà l'area interessata alla connessione. Quindi quello che succede è che esattamente ciò che succede all'interno del nostro cervello con i nostri neuroni. All'inizio abbiamo dei campi recettivi piccoli che quindi si occupano di zone piccole quindi porzioni dell'immagine piccole porzioni del nostro campo visivo piccole, questi campi visivi sono in qualche modo over lappati fra di loro e le uscite di questi neuroni con i campi recettivi piccoli vengono poi in qualche modo combinate per avere una superficie più alta. È ciò che accade quando noi guardiamo una scena prima abbiamo una visione generale del campo che ci si presenta davanti, quindi un campo recettivo ampio, dopodiché ci andiamo a focalizzare sui singoli dettagli quindi con un campo recettivo più basso.

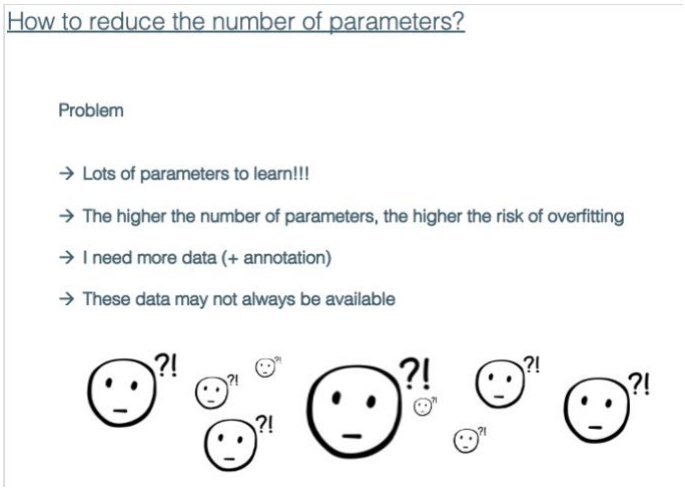


2.4.3 Max Pooling

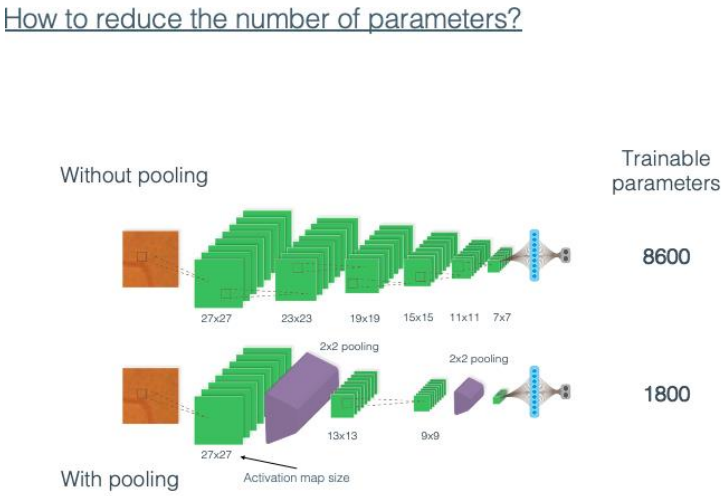
L'ultimo concetto da definire per avere in mano quelli che sono gli strumenti di base di una rete neurale convoluzionale è il max pooling. Abbiamo detto che abbiamo bisogno di imparare molti parametri, quello che dobbiamo imparare sono da una parte i valori dei kernel degli strari convoluzionali dall'altra parte i valori delle connessioni degli stati fully connected. Capiamo che il numero di parametri cresce esponenzialmente con la complessità del modello quindi tanto più voglio un modello complesso tanti più strati convoluzionali e più filtri vorrò avere, oppure tanti più neuroni fully connected dovrò avere. Il problema di avere tanti parametri da imparare è che rischio di overfittare soprattutto se la quantità di dati che ho a disposizione per il training è bassa. Quindi la prima cosa che si può fare è quindi ridurre il numero di parametri utilizzando un operatore che si chiama pooling. Esistono diversi tipi di pooling: max pooling, average pooling, ecc. Ad esempio, nella seguente immagine viene mostrato il pooling; diciamo che l'immagine a sinistra è l'attivazione quindi l'uscita di una convoluzione di un determinato strato. Se applico un max pooling per ogni sotto regione e in questo caso ho un max pooling 2x2 prendo il massimo; quindi nella sotto regione 2x2 rossa il massimo è 7 allora in uscita avrò 7 così 5 per la zona verde 4 per la zona gialla e 3 per la zona blu.



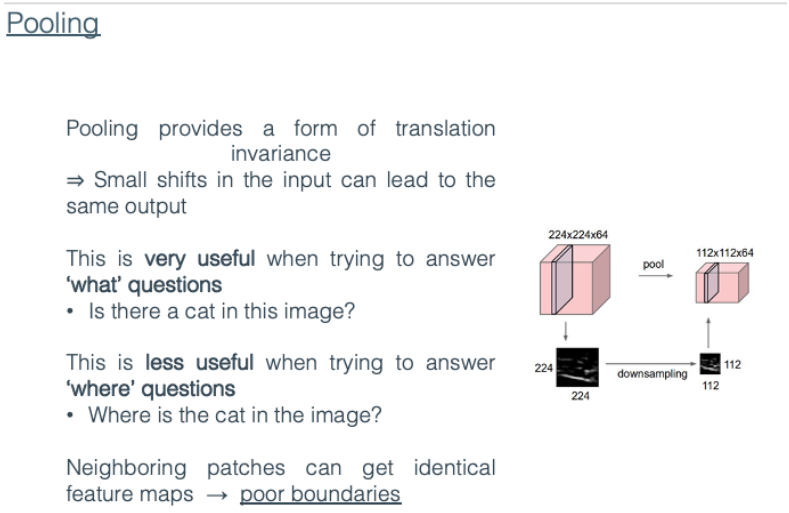
Quindi mi serve il max pooling quando ho tanti parametri da imparare perché è una rete neurale convoluzionale, quando ho alto rischio di aver fitting perché magari non ho molti dati, dovrei avere più dati e soprattutto più annotazioni associate a questi dati ma questi dati possono non sempre essere disponibili.



Utilizzando il max pooling il numero di parametri il cui valore dobbiamo assegnare durante il training passa in questo caso da 8.600 a 1.800. Questi numeri che vediamo sotto ai blocchi verdi sono le dimensioni delle mappe di attivazione, quindi passo da 27 per 27 e vado giù nella parte superiore dell'immagine senza pooling, nella parte inferiore dell'immagine con il pooling. Vediamo che il numero di strati convoluzionali di cui ho bisogno scende, e scende per il fatto che la mia dimensione delle mappe di attivazione sta scendendo. Quindi sia o meno strati convoluzionali di cui ho bisogno sarà anche più basso il numero di parametri dei Kernel di convoluzione che dovrò stimare.

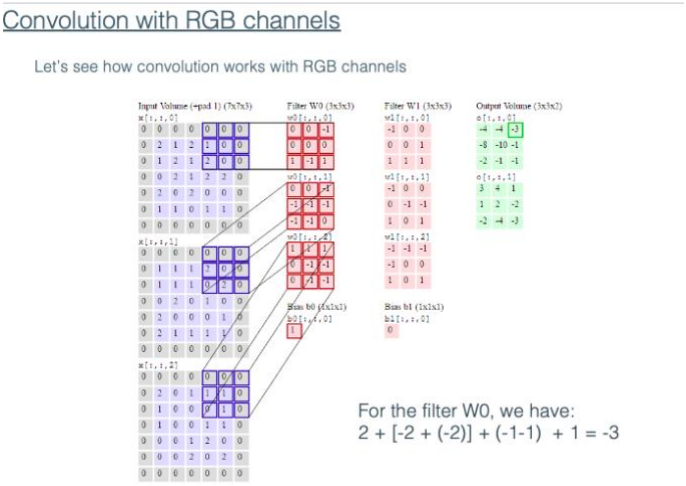


Il problema di utilizzare il pooling risiede nel fatto che il pooling altro non è che una forma di sotto campionamento. Il pooling mi serve quando la mia domanda, quindi quello che voglio che la mia rete convoluzionale predica è una domanda di tipo “What?” “Cosa?”, ad esempio “c'è un gatto nell'immagine?” “c'è un cane nell'immagine?” il cane resta cane il gatto resta gatto anche se avviene un sotto campionamento. Il max pooling o il pooling in generale diventa invece meno utile quando voglio rispondere ad una domanda di tipo “Where?” quindi quando mi chiedo “dov'è il gatto nell'immagine?” questo perché il sotto campionamento così come sia attraverso il pooling così come in generale qualsiasi tipo di sotto campionamento quello che va a provocare è uno sfocamento dei contorni quindi perdo la risoluzione iniziale dell'immagine e di conseguenza perdo anche informazioni su dove il gatto è presente nell'immagine.



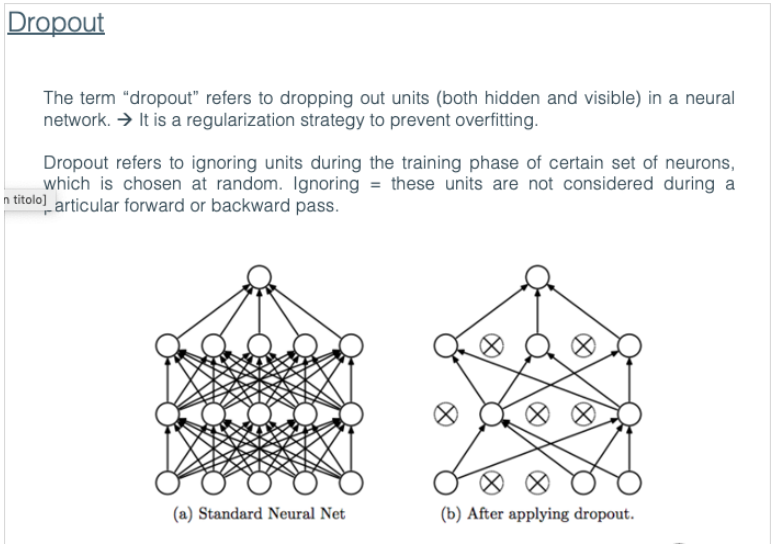
2.3.4 Convoluzione con immagini RGB

Per quanto riguarda la convoluzione con canali RGB, abbiamo un'immagine a colori (immagini png quello che vogliamo), quando leggiamo questa immagine questa ha tre canali quello del rosso quello del verde quello del blu (facciamo attenzione che quando leggiamo le immagini png anche se sono immagini in bianco e nero queste immagini comunque sono salvate in con tre canali quindi una volta caricata sui nostri programmi verrà percepita una punto un'immagine a tre canali). In questo caso abbiamo quindi il nostro input volume che è un 7x7x3 quindi abbiamo un'immagine che è 7x7 e abbiamo uno due e tre proprio perché abbiamo tre canali infatti vedete la terza dimensione 0,1,2, sono i 3 canali. Abbiamo due filtri quindi ci immaginiamo che il primo strato nella nostra rete neurale convoluzionale sia fatta da due filtri w_0 e w_1, filtri che hanno dimensione 3x3x3 lo vediamo qui per tre questo perché appunto abbiamo tre canali in ingresso che sono gli RGB e ovviamente abbiamo la nostra solita bias quindi il nostro kernel 1, kernel 2, kernel 3 la bias. Questi valori vengono appresi durante l'allenamento. Stessa cosa per i valori dei kernel del secondo filtro. Dato in ingresso un'immagine RGB, dati due filtri w_0 e w_1 in uscita le nostre attivazioni sono ovviamente 2, due attivazioni perché abbiamo due filtri. In fondo a destra vi sono tutti i calcoli per vedere appunto qual è il risultato della convoluzione per il primo filtro, qui nel primo caso abbiamo tutti zero, l'unico numero che abbiamo è un 2 che viene per 1 quindi 2x1=2. Passiamo poi alla seconda componente del filtro w_0 , anche in questo caso abbiamo zero quindi la moltiplicazione è ovviamente 0, gli unici valori sono -2 che è dato da 2 x-1= -2, poi di nuovo 0,0 poi di nuovo 2 x -1 = -2; infatti abbiamo - 2 + - 2. Passiamo invece al terzo canale di nuovo moltiplicazione -1-1 e poi somma quindi fondamentalmente convoluzione e in questo caso abbiamo quindi 0,0 gli unici valori che sono diversi da zero sono 1 che viene moltiplicato per -1 e poi quest'altro 1 che viene moltiplicato per quest'altro -1 e quindi abbiamo meno 1 -1. E poi ovviamente sommiamo la bias + 1. Quindi il risultato è 2 - 4 +2 +1= -3. Infatti, andiamo a mettere il meno 3 nell'output volume. Quindi ricordiamoci che anche se abbiamo due filtri ovviamente il numero di canali per ogni filtro dipende dall'ingresso quindi se in ingresso abbiamo tre canali perché abbiamo un'immagine RGB ovviamente anche il filtro w_0 dovrà avere tre canali.



2.3.4 Dropuot

Il dropout è una tecnica di regolarizzazione ampiamente utilizzata nelle reti neurali, che aiuta a prevenire l'overfitting durante l'addestramento. Il dropout funziona eliminando casualmente alcuni neuroni della rete neurale durante l'addestramento. In pratica, durante l'addestramento, i neuroni vengono eliminati con una probabilità prefissata, solitamente compresa tra il 20% e il 50%. In questo modo, la rete neurale non può dipendere troppo da un singolo neurone, costringendola ad imparare a distribuire l'informazione su più neuroni. Durante la fase di test, tutti i neuroni vengono mantenuti e i pesi vengono riscalati in modo che la somma complessiva dell'input rimanga invariata rispetto alla fase di addestramento. Il dropout è stato dimostrato efficace nella prevenzione dell'overfitting, poiché costringe la rete neurale a trovare soluzioni più robuste e generali, anziché imparare a memorizzare i dati di addestramento. Inoltre, il dropout aiuta a ridurre la dipendenza tra i neuroni e a ridurre l'effetto delle interazioni tra i neuroni, aumentando così la capacità di generalizzazione della rete neurale. Il termine dropout fa quindi riferimento all'esclusione di alcune unità, siano queste nascoste o visibili all'interno della nostra rete che serve come strategia di regolarizzazione e quindi per evitare per diminuire le probabilità di overfitting. Fondamentalmente il drop out si riferisce appunto all'ignorare alcune unità durante la fase di training. Le unità da ignorare sono scelte in maniera del tutto random quindi fondamentalmente ignorare queste unità vuol dire che queste unità non sono considerate durante un passo di forward o di backward durante il training della nostra rete neurale convoluzionale e quindi vedete ad esempio in questo caso il neurone viene ignorato perché non è connesso né agli strati precedenti né agli strati successivi.



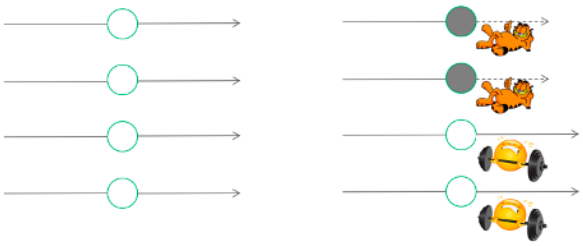
Il dropout può essere utilizzato sia negli strati convoluzioneali sia negli strati densi, in realtà però è prettamente utilizzato negli strati densi e questo perché gli strati densi e quindi gli strati fully connected sono quelli a cui è associato il numero maggiore di parametri da imparare. Ricordiamo che abbiamo tutte le connessioni dobbiamo imparare il valore di tutte queste connessioni, i pesi di tutte queste connessioni. Quindi qui sorge un problema di codipendenza fra i diversi neuroni nella fase di training; quello che succede è che si verifica la cosiddetta coadaptation quindi un coadattamento lo vediamo in questa immagine abbastanza esplicativa in cui alcuni neuroni fanno tutto il lavoro mentre altri neuroni all'interno dello stesso strato si adattano e fondamentalmente perdono una capacità predittiva proprio perché questa capacità predittiva viene tutta messa sulle spalle degli altri. Fondamentalmente è come se il numero di neuroni diminuisse perché invece che averne 4, 2 si adattano quindi solo due diventano predittivi, e quindi in questo modo diciamo la probabilità di fornire una previsione errata la probabilità di overfittare aumentano proprio perché è come se stessimo per assurdo riducendo la complessità del nostro modello.

Dropout

Dropout is mainly used in dense layers.

A fully connected layer occupies most of the parameters, and hence, neurons develop **co-dependency amongst each other** during training which curbs the individual power of each neuron leading to over-fitting of training data.

One major issue in learning large networks is **co-adaptation**. In such a network, if all the weights are learned together, it is common that some of the connections will have more predictive capability than the others.



Il dropout cambia ovviamente in training e testing. Durante il training per ogni strato per ogni esempio per ogni trazione ignoro una frazione random dei nodi e le corrispettive ovviamente attivazioni quindi l'uscita di questi nodi; questa frazione p è una probabilità. Durante la fase di testing utilizzo tutte le attivazioni quindi non vado più escludere alcune attivazioni proprio perché non vado più ad ignorare alcuni nodi della mia rete, però ovviamente le attivazioni verranno scalate da un fattore p che è della stessa probabilità con cui era stato fatto il dropout proprio per far fronte alle attivazioni mancanti durante il training.

Dropout

Dropout is mainly used in dense layers.

A fully connected layer occupies most of the parameters, and hence, neurons develop **co-dependency amongst each other** during training which curbs the individual power of each neuron leading to over-fitting of training data.

One major issue in learning large networks is **co-adaptation**. In such a network, if all the weights are learned together, it is common that some of the connections will have more predictive capability than the others.

Training phase
For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, p, of nodes (and corresponding activations).

Testing phase
Use all activations, but scale them by a factor p (to account for the missing activations during training).

3. Training CNNs

In questo capitolo occupiamo dell'allenamento delle reti neurali convoluzionali che è quel processo che ci permette di apprendere i valori dei kernel degli strati convoluzionali delle reti e i valori delle connessioni degli strati fully connected. Innanzitutto, per capire come avviene l'allenamento dobbiamo tornare indietro sulle metriche di valutazione quindi dobbiamo capire quanto bene la nostra rete predice. Il quanto bene la nostra rete predice è determinato dalla differenza dall'errore dalla cosiddetta loss, fra i label, quello che è la nostra rete predice e i label di riferimento. Quando parliamo di label di riferimento ricordiamo che stiamo parlando di un apprendimento di tipo supervisionato. Quindi dato un esempio *i*-esimo *i* considerata la predizione p_i e il target , quindi il valore vero t_i allora la loss può essere definita come segue: se abbiamo un output continuo come nei casi di regressione la loss può essere definita come l'errore quadratico, quindi il quadrato della differenza fra il target la probabilità di output; mentre per gli output categorici, quindi ad esempio quando parliamo di classificazione, "c'è o no un gatto nell'immagine", una metrica di loss o di errore può essere l'accuratezza che vediamo qui esplicitata in formula nella seguente immagine ricordiamo essere anche il rapporto fra gli elementi sulla diagonale della matrice di confusione è tutti gli elementi facenti parte del test set. L'accuratezza viene utilizzata per i task di classificazione in fase di predizione e non invece in fase di allenamento. Se osserviamo il grafico in basso quindi abbiamo i nostri quadratini gialli e arancioni che vanno un po' a simboleggiare le mappe di attivazione degli strati convoluzionali e poi invece i blocchi grigi e celesti che rappresentano gli strati fully connected dove i pallini sono i neuroni degli strati fully connected. Vediamo che infatti ogni quadratino arancione o giallo viene collegato ad ognuno dei neuroni dal primo strato fully connected che è quello celeste. Nell'esempio abbiamo un due neuroni nello strato di output che in uscita hanno delle probabilità quindi in questo caso il problema è di classificazione binaria, abbiamo due classi, il primo neurone ci dà una probabilità 0,35 di appartenere alla classe 0 e il secondo neurone ci dà una probabilità 0,65 di appartenere alla classe 1.

How well does the network predict?
→Determined by the loss/error between the predicted labels and reference labels.

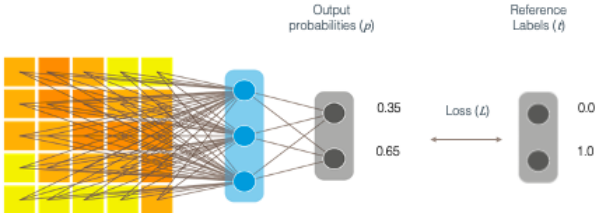
For sample i, given prediction p_i and target t_i, we define the loss L_i:

For continuous output:
• **Squared error**
(For regression tasks)

$$L_i = (t_i - p_i)^2$$

For categorical output:
• **Accuracy**
(For classification tasks)

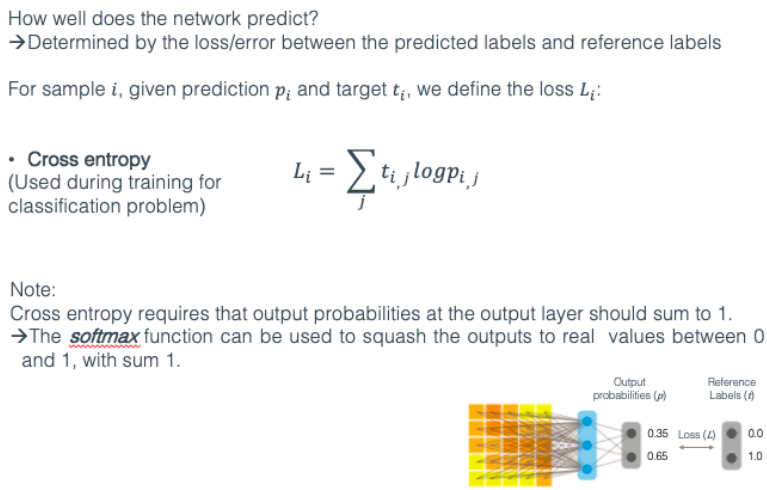
$$L_i = I\left(t_i = argmax_j p_{i,j}\right)^2$$



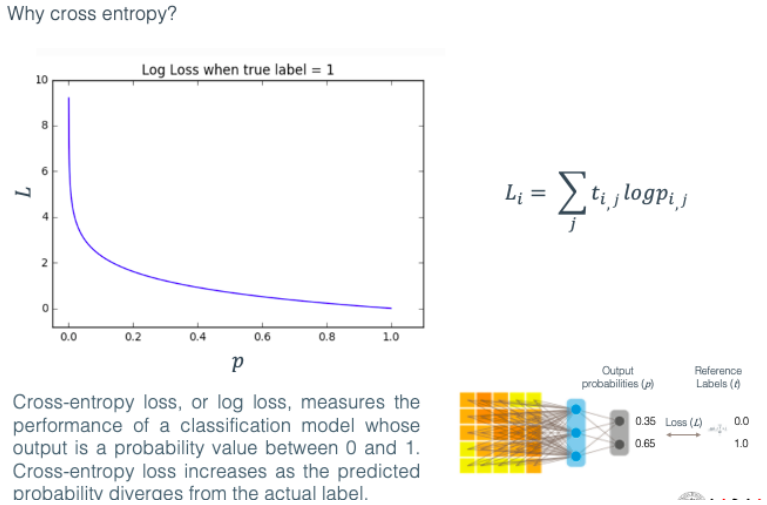
3.1 Metrics for training

Per l'allenamento utilizziamo una metrica diversa metrica che si chiama cross entropia, quindi, la cross entropia è utilizzata durante il training per problemi di classificazione. Cosa distingue la cross entropia dall'accuratezza? Visivamente, troviamo un logaritmo di una probabilità che viene moltiplicato per il target il ground truth e vengono sommate per tutte le classi j, i prodotti fra punto il target quindi il label è il logaritmo della probabilità. Abbiamo già parlato di probabilità però niente ci dice che l'uscita di una rete neurale convoluzionale o meglio dell'ultimo strato fully connected di una rete neurale collezionale sia una probabilità. Cosa dobbiamo fare per avere una probabilità di uscita? Quello che ci serve è una funzione che è chiamata

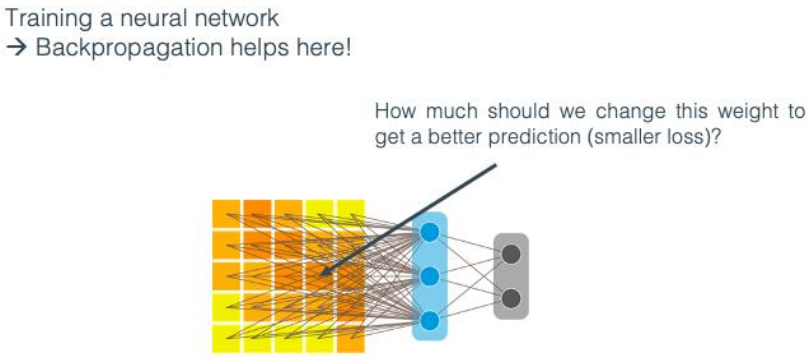
softmax. La funzione softmax viene utilizzata per trasformare l'uscita dell'ultimo strato convoluzionale in modo che abbia dei valori reali fra 0 e 1 con la somma che è 1 che è appunto la definizione di probabilità.



Ma perché la cross entropia e non un'altra metrica? La cross entropia che vediamo nella figura di seguito è proprietà molto importante quindi man mano che la probabilità aumenta quindi si avvicina a 1 la loss, quindi la nostra cross entropia scende. Ma questo non avviene in maniera lineare perché la cross entropia aumenta man mano che la probabilità predetta diverge dal label reale e infatti vediamo che con probabilità vicine allo zero la cross entropia con la nostra loss, il nostro errore è molto molto alto.

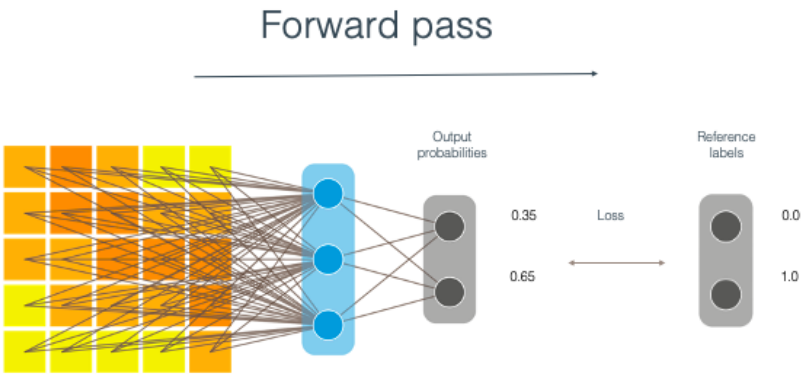


Ma quindi come minimizziamo la nostra loss, cosa possiamo fare affinché la loss raggiunga il valore zero? l'unica cosa su cui possiamo agire come ci siamo già detti diverse volte è cambiare i valori dei kernel dagli strati convoluzionali oppure i pesi delle connessioni degli strati fully connected. Ci dobbiamo chiedere quanto ogni pixel che forma un kernel o quanto ogni peso di uno strato fully connected contribuisce alla nostra loss, al valore che la nostra loss assume.



3.2 Backpropagation e Forward pass

La backpropagation e il forward pass sono due fasi fondamentali dell'addestramento delle reti neurali. Il forward pass è la fase iniziale dell'addestramento, durante la quale l'input viene propagato attraverso la rete neurale fino alla produzione di un'uscita. Durante questa fase, i pesi dei neuroni vengono moltiplicati per l'input corrispondente e sommati insieme, producendo un valore di attivazione. Questo valore di attivazione viene poi passato attraverso una funzione di attivazione non lineare, come la funzione ReLU o la funzione sigmoide, producendo l'output del neurone. Il forward pass viene eseguito iterativamente per tutti gli strati della rete neurale, fino alla produzione dell'output finale. Durante questa fase, la rete neurale utilizza i pesi attuali per produrre un'uscita, senza apportare alcuna modifica ai pesi.



La backpropagation, invece, è la fase successiva dell'addestramento, durante la quale l'errore viene propagato all'indietro attraverso la rete neurale. In pratica, durante questa fase, l'errore tra l'output della rete neurale e l'output atteso viene calcolato utilizzando una funzione di costo, come la perdita quadratico-media o la cross-entropy. L'errore viene quindi propagato all'indietro attraverso la rete neurale, calcolando i gradienti rispetto ai pesi dei neuroni. I gradienti vengono utilizzati per aggiornare i pesi della rete neurale, utilizzando un algoritmo di ottimizzazione come la discesa del gradiente. In sintesi, il forward pass è la fase iniziale dell'addestramento delle reti neurali, durante la quale l'input viene propagato attraverso la rete neurale fino alla produzione di un'uscita. La backpropagation, invece, è la fase successiva dell'addestramento, durante la quale l'errore viene propagato all'indietro attraverso la rete neurale per aggiornare i pesi dei neuroni utilizzando un algoritmo di ottimizzazione come la discesa del gradiente.
L'algoritmo di back propagation ci dice questo, ci aiuta a capire questo quindi nel cosiddetto farward pass, quello che facciamo è dati dei pesi degli strati fully connected dati dei valori degli strati convoluzionali la nostra loss assume un determinato valore ma sappiamo benissimo che questo valore di loss dipende da alcuni parametri che sono i pesi delle connessioni dei fully connected è i kernel degli strati di convoluzione.

Come ci spostiamo all'interno dello spazio dei pesi? La loss dipende dai pesi, quindi la loss dipende dai valori dei kernel degli stati convoluzionali e dal valore delle connessioni degli strati fully connected. Quindi immaginiamoci di avere questo spazio n dimensionale dove n è il numero di parametri della nostra rete neurale convoluzionale, esiste una combinazione ottima di questi parametri che ci permette di minimizzare la loss. Quando minimizziamo la loss stiamo risolvendo un problema di ottimizzazione stiamo andando a cercare quello che si chiama minimo dalla loss. Il minimo di questa loss lo raggiungiamo tramite un algoritmo che si chiama gradiente discendente che appunto è un modo abbastanza intelligente di muoversi nello spazio n dimensionale. Qui vediamo come ci muoviamo all'interno dello spazio dei pesi.

3.3 Learning Rate

Il parametro che abbiamo visto prima nella regola di update si chiamava learning rate; in questa applicazione vediamo visivamente come il learning rate influisce nella minimizzazione della nostra loss. Se iniziamo da un learning rate basso allora abbiamo dei passi piccoli nello spazio dei pesi, verso il raggiungimento del minimo. Se andiamo ad aumentare il learning rate ci muoviamo in maniera più veloce all'interno dello spazio dei pesi. Se invece abbiamo un alto learning rate quello che succede è che iniziamo a saltare da una parte all'altra. Il learning rate (tasso di apprendimento) è un parametro importante che viene utilizzato durante l'addestramento delle reti neurali, inclusa quella delle reti neurali convoluzionali (CNN). Il learning rate determina la velocità con cui i pesi della rete neurale vengono aggiornati durante la fase di addestramento. Un learning rate alto significa che i pesi della rete neurale verranno aggiornati rapidamente durante la discesa del gradiente, mentre un learning rate basso significa che i pesi verranno aggiornati lentamente. Se il learning rate è troppo alto, i pesi della rete neurale possono oscillare e divergere, portando ad un addestramento instabile o addirittura al fallimento dell'addestramento. D'altra parte, se il learning rate è troppo basso, l'addestramento della rete neurale può essere troppo lento e richiedere molto tempo per raggiungere una buona accuratezza. Il learning rate deve essere scelto con attenzione per garantire un addestramento stabile ed efficiente della CNN. In genere, si utilizzano tecniche di ottimizzazione che adattano automaticamente il learning rate, come la discesa del gradiente con momento o l'Adaptive Moment Estimation (Adam), che adattano il learning rate in base alla natura del problema e alla situazione specifica durante l'addestramento.

3.4 Gradient Descent

La discesa del gradiente (gradient descent in inglese) è un algoritmo di ottimizzazione utilizzato per minimizzare una funzione di perdita in una rete neurale. In pratica, la discesa del gradiente viene utilizzata per aggiornare i pesi della rete neurale in modo da ridurre gradualmente la perdita durante l'addestramento. Il principio alla base della discesa del gradiente è quello di calcolare il gradiente della funzione di loss rispetto ai pesi della rete neurale. Il gradiente è una misura della pendenza della funzione di perdita rispetto ai pesi e indica la direzione in cui i pesi devono essere aggiornati per ridurre la perdita. In altre parole, il gradiente indica come la funzione di perdita cambierà al variare dei pesi. Una volta calcolato il gradiente, la discesa del gradiente aggiorna i pesi della rete neurale nella direzione opposta al gradiente, in modo da ridurre gradualmente la perdita. Il parametro che determina la lunghezza del passo dell'aggiornamento dei pesi è il learning rate (tasso di apprendimento), che indica quanto si deve spostare i pesi nella direzione del gradiente. La discesa del gradiente viene solitamente ripetuta molte volte durante l'addestramento della rete neurale, e questo processo viene chiamato epoca (epoch). Durante ogni epoca, il modello esegue una predizione sui dati di addestramento, calcola la funzione di perdita e aggiorna i pesi della rete neurale utilizzando la discesa del gradiente. In sintesi, la discesa del gradiente è un algoritmo di ottimizzazione che viene utilizzato per aggiornare i pesi di una rete neurale durante l'addestramento in modo da minimizzare la funzione di perdita. L'algoritmo calcola il gradiente della funzione di perdita rispetto ai pesi e aggiorna i pesi nella direzione opposta al gradiente, utilizzando il learning rate come parametro per determinare la lunghezza del passo dell'aggiornamento dei pesi. La discesa del gradiente viene ripetuta molte volte durante l'addestramento della rete neurale, in modo da ottenere pesi ottimizzati per la funzione di perdita.

Si chiama gradient descent in quanto gradient significa gradiente, quindi vuol dire che io mi muovo nello spazio dei pesi seguendo la direzione che minimizza il gradiente; per evitare che si oscilli continuamente all'interno dello spazio dei paesi senza riuscire a raggiungere il minimo globale quindi il punto più basso della loss. Quello che si fa è aggiungere il momentum, che ha a che fare con la derivata seconda, quindi mentre il gradiente è legato alla derivata prima il momentum possiamo vederlo legato alla derivata seconda a grandi linee. Introducendo il momentum quello che succede è che smussiamo quindi regolarizziamo la direzione con cui la discesa del gradiente avviene. Infatti, vediamo che senza momento le oscillazioni nel grafico in mezzo sono molto ampie, mentre se introduciamo il momentum queste oscillazioni diminuiscono. È importante introdurre il momentum e scendere nella maniera più armonica possibile per raggiungere il minimo globale perché il problema quando si effettua un allenamento di una rete neurale convoluzionale è l'overfitting. L'overfitting in questo caso si traduce nell'essere bloccati durante la discesa in un minimo locale. Un minimo locale essendo un minimo, è caratterizzato da un valore della derivata uguale a zero, però se la derivata è uguale a zero non ho più la regola di update perché avrò sempre che il peso vecchio è uguale al peso nuovo, però il minimo è locale quindi vuol dire che non è lo zero ma è un minimo diverso, quindi io rimango bloccato in quel minimo e non riesco a ottimizzare la mia loss, non riesco a minimizzare la mia loss, mi blocco e over fitto perché sono lontana dall' ottimo.
Non ci dimentichiamo che il nostro allenamento ha sempre a che fare con dati, quindi tanti più dati o tanto meglio il mio allenamento procederà, tanto più lontana sarà la probabilità di overfittare. Il problema però è che abbiamo detto che il gradiente discendente lo implementiamo su uno spazio n dimensionale di pesi della rete questo n è davvero molto molto grande e quindi dobbiamo immaginare di caricare sulla ram tutti i pesi tutto il modello che già di per sé sono pesanti e poi anche tutti i dati ed è praticamente impossibile. Per far fronte a questa problematica di computazione si è passati dal gradient descent al minibatch stochastic gradient decent. Quindi invece che calcolare il gradiente su tutti i dati che fanno parte del mio training set vado a calcolare il gradiente solo su un numero piccolo di dati di esempi, questo insieme di dati piccolo che utilizzo per calcolare il gradiente viene chiamato mini batch; quindi quando vado a effettuare l'update dei pesi con la regola di update, l'update quindi il nuovo valore dei pesi viene calcolato in base a questi gradienti calcolati sul mini batch. Ovviamente non è la cosa migliore però bisogna ricorrere al mini batch gradient descent proprio per via del fatto che ci sono delle problematiche di memoria oltre che di potenza di calcolo. L'ipotesi è che tanti piccoli update con dei gradienti buoni ma non perfetti comunque ci aiutano a muoverci verso il minimo della loss, quindi nella giusta direzione.

Supponiamo di avere 1100 esempi di training e vogliamo utilizzare una batch size uguale a 100 in realtà questo è un esempio perché solitamente scegliamo come batch size delle potenze di 2 2 4 8 16 32 però rimaniamo su questo esempio. Quindi alleneremo la rete con 100 campioni alla volta poi con altri 100 campioni poi con altri 100 campioni e così via fin quando non avremo raggiunto in 1.100 campioni di training. I vantaggi sono che in questo caso abbiamo bisogno di meno memoria perché ovviamente il training è più veloce con le mini batch

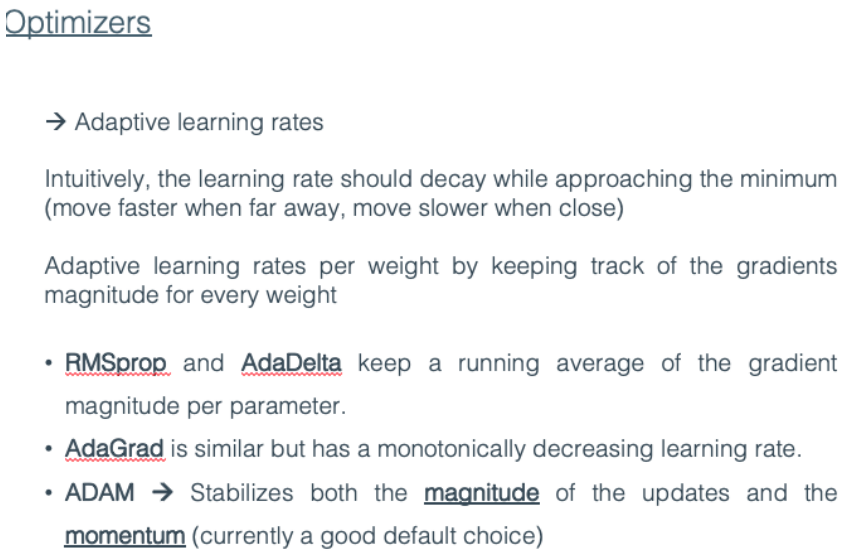
perché abbiamo meno campioni rispetto alla numerosità totale dei campioni di training. Quindi quante volte propagheremo il nostro gradiente, quante volte faremo forward e backward propagation? lo faremo 11 volte quindi abbiamo 11 batch ognuna con 100 campioni quindi 100 è la batch size 11 è il numero di batch ogni volta dopo ogni propagazione di ogni batch avremo l'update dei parametri della rete. A questo punto bisogna dare due definizioni importanti: il numero di epoche e il numero di iterazioni. Il numero di epoche indica quante volte l'intero dataset di training è fatto passare attraverso il forward e backward pass dalla rete, mentre il numero di interazioni quanto è di quante batch abbiamo bisogno per completare un'epoca. In questo caso avremmo bisogno di 11 iterazioni per completare un'epoca perché dobbiamo propagare 11 batch affinché tutto il dataset di training sia visto dalla rete. Dopo aver fatto passare 11 batch allora possiamo dire che è un'epoca è finita. Ovviamente non alleniamo la nostra rete per un'epoca la alleniamo per 50, 100, 1000 e 2000 epoche. Ci sono anche svantaggi perché tanto più piccola è la dimensione del batch tanto meno accurata sarà la stima del gradiente. Il caso opposto rispetto al gradient descents è lo stochastic gradient descent che ha un andamento del tutto peculiare, vediamo diversi cambi di direzione diverse oscillazioni. Questo stochastic gradient descent si riferisce ad un allenamento in cui io faccio vedere alla mia rete un campione una istanza per volta.

3.5 Ottimizzatori

Gli ottimizzatori sono algoritmi utilizzati nelle reti neurali convoluzionali (CNN) per aggiornare i pesi della rete durante l'addestramento. L'obiettivo degli ottimizzatori è quello di minimizzare la funzione di perdita della CNN attraverso la regolazione dei pesi della rete in modo che la CNN possa effettuare predizioni più accurate sui dati di test. L'ottimizzazione dei pesi delle CNN è un processo iterativo e viene eseguito attraverso il calcolo del gradiente della funzione di perdita rispetto ai pesi della rete. L'ottimizzatore utilizza il gradiente per aggiornare i pesi della rete in modo che la funzione di perdita venga minimizzata. Ci sono diversi tipi di ottimizzatori disponibili per l'addestramento delle CNN, ognuno con un'implementazione specifica. Alcuni degli ottimizzatori più comuni includono:

- Stochastic Gradient Descent (SGD): è l'ottimizzatore più semplice ed è spesso utilizzato come punto di partenza per altri ottimizzatori più avanzati. SGD aggiorna i pesi in base alla media dei gradienti calcolati su un singolo esempio di addestramento.
- Adam: è un algoritmo di ottimizzazione adattivo che aggiorna i pesi utilizzando una combinazione del gradiente e delle stime dei momenti di primo e secondo ordine. Adam è particolarmente utile per addestrare reti neurali con molti parametri.
- RMSProp: è un ottimizzatore che divide il tasso di apprendimento per un valore che è una media mobile dei gradienti quadri passati. RMSProp tende a funzionare bene su problemi di addestramento non stazionari o non gaussiani.
- Adagrad: è un ottimizzatore che adatta il tasso di apprendimento a ciascun parametro in modo da dare più peso ai parametri con gradienti bassi.

La scelta dell'ottimizzatore dipende dal problema di addestramento specifico e dalle caratteristiche del set di dati. Gli ottimizzatori vanno fondamentalmente a definire quella che è la regola di update del valore dei pesi. Prima abbiamo detto che è il learning rate è un parametro che definisce lo step con cui io mi muovo nello spazio dei pesi per raggiungere il minimo globale della loss, però in realtà intuitivamente il learning rete non dovrebbe avere un valore costante anzi il suo valore dovrebbe scendere e diminuire via via che mi avvicino al minimo, quindi immaginiamo quando sono lontano dal minimo globale mi muovo velocemente, quando invece mi sto avvicinando al minimo globale non voglio muovermi velocemente perché altrimenti potrei allontanarmi dal minimo stesso voglio invece raffinare la mia discesa, e per questo è stato introdotto il learning rate adattivo, learning rate adattivo il cui valore ha che fare con l'ampiezza del gradiente per ogni peso, perché quando sono lontano dal minimo il mio gradiente è alto quando sono vicina al minimo il mio gradiente è basso. Quindi c'è tutto un insieme di ottimizzatori ad esempio ad esempio RMSprop proprio o AdaDelta oppure diciamo AdaGrad ma quello in realtà probabilmente più famoso al giorno d'oggi è ADAM che stabilizza sia la magnitude sia il momentum del gradiente.

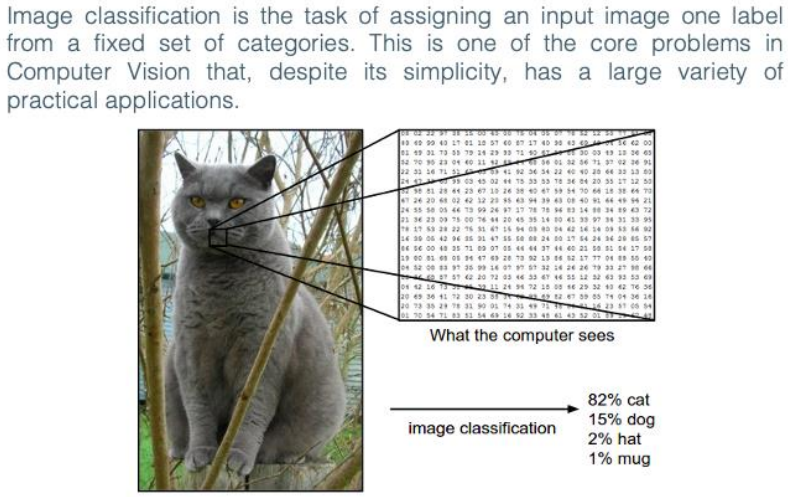


3.6 Data Augmentation

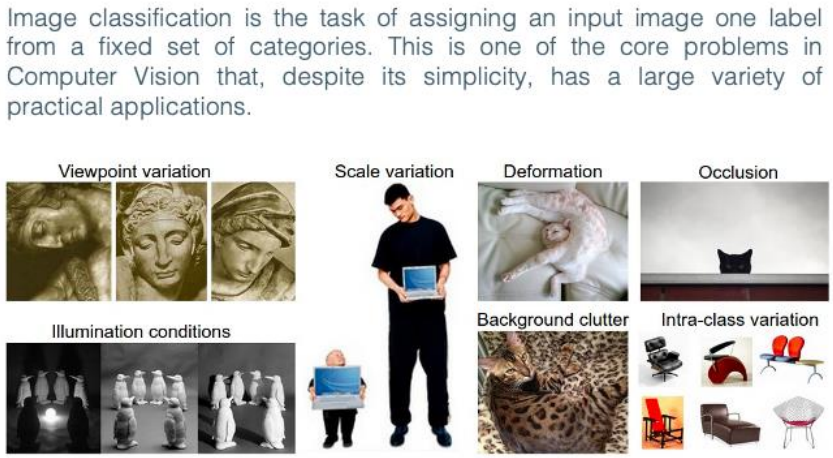
La data augmentation è una tecnica di pre-elaborazione dei dati utilizzata in machine learning e in particolare nelle reti neurali convoluzionali (CNN) per aumentare la quantità di dati disponibili per l'addestramento della rete e migliorare la sua capacità di generalizzazione. Se non ho sufficienti non è sempre semplice raccoglierne altri perché dobbiamo ricordarci che oltre ai dati abbiamo bisogno anche delle annotazioni. Se non riesco a recuperare nuovi dati quello che posso fare è ad esempio avere tecniche di data augmentation da applicare al training set non ci interessa applicare il data augmentation al test set perché il testing lo voglio fare sui dati reali perché voglio aumentare il mio dataset di training perché voglio far fronte alla complessità del mio modello è quindi lì ha senso aumentare i dati in training. In pratica, la data augmentation consiste nell'applicare una serie di trasformazioni casuali ai dati di addestramento, come ad esempio rotazione, mirroring, cropping, deformazioni non lineari come vediamo nella seguente immagine: abbiamo la nostra immagine a un certo punto la flippiamo, ruotiamo, sfochiamo, e applichiamo deformazioni di vario tipo. Attenzione che non tutte le strategie di data augmentation hanno senso; un esempio banale che però secondo è esplicativo è il seguente: se ho una risonanza magnetica cardiaca non mi interessa fare il flipping come tecnica di augmentation perché il cuore è sempre a sinistra se faccio flipping mi ritrovo il quale a destra che non ha assolutamente alcun al senso; quindi l'idea è aumentiamo i dati di training però attenzione ad applicare un data augmentation che abbia senso e sempre guardiamo le immagini che abbiamo ottenuto dopo l' augmentation per assicurarci che abbiano senso.

4. Classification Architectures

In questo capitolo parliamo di alcune delle architetture di reti neurali convoluzionali utilizzate per la classificazione. Quando parliamo di classificare un'immagine parliamo di un task che consiste nell'assegnare un determinato label ad un'immagine; il label che viene classificato fa parte di un insieme di categoria prefissate. Classificare l'immagine nella seguente figura equivale a dire "nell'immagine c'è un gatto" quando partiamo da un dataset più ampio che contiene diverse categorie tra cui cane gatto è così via. Quello che il computer vede è un insieme di pixel, quindi di livelli di intensità che fanno parte della nostra immagine quello che la nostra rete convoluzionale da in uscita è appunto una probabilità che l'immagine appartenga a una delle determinate classi o categorie del nostro dataset. Ad esempio in questo caso la probabilità che nell'immagine ci sia un gatto è 82% ripetiamo perché riusciamo ad avere una probabilità in uscita dalla rete perché abbiamo utilizzato la nostra funzione softmax.



Il problema è che è il task di classificazione dell'immagine non è un task semplice e in realtà è uno dei problemi principali della computer vision che ha ovviamente una grande varietà di applicazioni pratiche però presenta anche diverse sfide quindi diversi punti di vista, variazioni di scala diverse condizioni di illuminazione, deformazione, occlusioni e così via. Lo stesso oggetto come vediamo nell'ultima nell'immagine in basso a destra quindi ad esempio una sedia può avere ovviamente una grande variabilità al proprio interno, però noi comunque così come anche la nostra rete neurale convoluzionale impariamo a riconoscere le caratteristiche e le feature che vengono condivise dalla classe sedia, nonostante le sue diverse rappresentazioni.



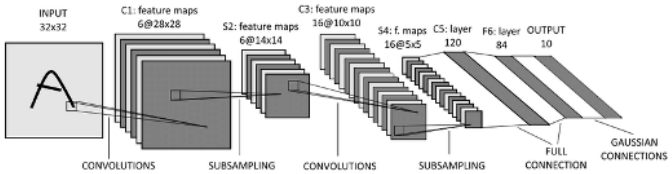
4.1 LeNet

La LeNet (acronimo di LeNet-5) è una delle prime architetture di reti neurali convoluzionali (CNN) sviluppate per la classificazione di immagini. È stata proposta da Yann LeCun, Leon Bottou, Yosuha Bengio e Patrick Haffner nel 1998 per riconoscere cifre scritte a mano. La LeNet ha una struttura a cascata di strati convoluzionali e di pooling seguiti da uno o più strati fully connected. In particolare, la LeNet-5 è composta da sette strati: tre strati convoluzionali, due strati di pooling e due strati fully connected. I primi due strati convoluzionali sono seguiti da strati di pooling per ridurre la dimensione delle mappe delle caratteristiche. Il terzo strato convoluzionale è seguito da uno strato fully connected che combina le mappe delle caratteristiche in un vettore di features unidimensionale. Questo vettore di features viene poi passato attraverso un altro strato fully connected per produrre le probabilità di classificazione per le diverse classi. La LeNet è stata una delle prime CNN a dimostrare la capacità di apprendere e classificare immagini complesse. Ha avuto un grande impatto sulla comunità scientifica e ha ispirato lo sviluppo di molte altre architetture di CNN utilizzate oggi.

Quindi partiamo da un'immagine che ha in questo caso un canale infatti vediamo che la feature map è 1; la dimensione dell'immagine con cui LaNet5 è stata allenata è 32x32 pixel, dopodiché passiamo alla prima convoluzione che in uscita ci dà 6 mappe di feature quindi dandoci 6 feature map vuol dire che abbiamo 6 kernel convoluzionali. La dimensione della mappa di attivazione è 28x28 che si ottiene coinvolgendo l'immagine di input l'immagine originale con un kernel di dimensione 5x5. L'attivazione in questo caso è la tangente iperbolica che è una delle varie funzioni di attivazione come abbiamo visto nelle lezioni precedenti. Dopo la convoluzione abbiamo l'average pooling che in questo caso ovviamente non agisce sul numero di feature map perché non abbiamo dei kernel convoluzionali ma abbiamo un pooling quindi un'operazione di sotto campionamento. Infatti, vediamo che si passa da una dimensione 28 x 28 a una dimensione 14x14 perché la dimensione del pooling è 2x2; anche in questo caso l'attivazione coincide con la tangente iperbolica. Abbiamo quindi di nuovo una convoluzione un'avarage pooling, strato 5 convoluzione è per questo che si chiama LeNEt5 perché ho cinque strati di tipo convoluzionale quindi che fanno parte della porzione convoluzionale della mia rete. Infine abbiamo i due strati fully connected, quindi uno che ha dimensioni 84 vuol dire che abbiamo 84 neuroni è l'output che invece ha 10 neuroni di output perché abbiamo dieci classi, ed è attivato con la softmax che ci permette di avere in uscita 10 classi e quindi 10 probabilità.

LeNet-5 (1998)

Pioneering 7-layer CNN

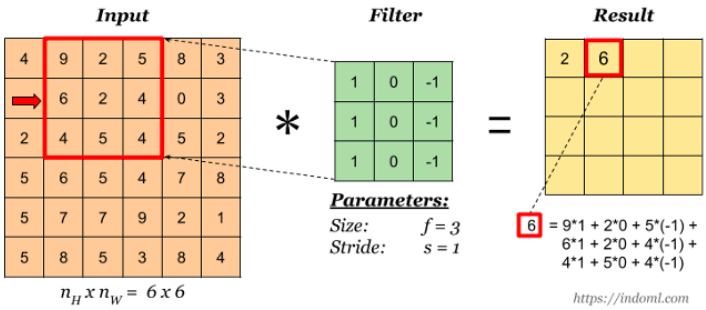


Summary of LeNet-5 Architecture

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Torniamo un attimo per una breve puntualizzazione sul concetto di Stride. Ormai abbiamo capito come si effettua una convoluzione quindi abbiamo il nostro input che può essere l'immagine originale di input oppure una mappa di attivazione in uscita da uno degli stati convoluzionali dalla nostra rete e abbiamo il nostro filtro di convoluzione che in questo caso a dimensione 3x3. Lo stride che vedete indicato dalla freccia rossa indica di quanti pixel mi sposto nell'immagine, nell'input per effettuare la convoluzione quindi in questo caso lo stride è unitario e infatti il nostro filtro di convoluzione viene centrato sul 2 se invece lo stride fosse stato 2 la freccia sarebbe stata più lunga avrebbe preso quindi due pixel e quindi mio filtro sarebbe stato centrato sul 4.

Stride

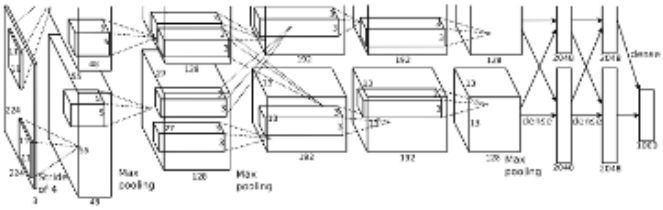


4.2 AlexNet

Dopo LaNEt5 sviluppata nel 1998, la seconda rete pioneristica quindi una delle pietre miliari delle architetture di classificazione AlexNet che è stata invece rilasciate nel 2012. La differenza fra LeNet5 e AlexNet è che innanzitutto vediamo un una profondità maggiore della rete con più filtri per ogni strato e con delle convoluzioni particolari che si chiamano stacked convolutional layers che permettono di condividere parti delle mappe di attivazione per rendere i calcoli più semplici e per alleggerire un po' la potenza di calcolo e la memoria richieste per l'allenamento di questa rete. Anche in questo caso vediamo convoluzione, max pooling convoluzione, max pooling poi fully connected. Quello che cambia è anche che come attivazione abbiamo la RELU quindi la rectified linear unit anzichè la tangente iperbolica e anche Relu quindi fa parte dell'insieme di funzioni di attivazioni che possono essere utilizzate per attivare le uscite delle convoluzioni.

AlexNet (2012)

Deeper, with more filters per layer, and with stacked convolutional layers

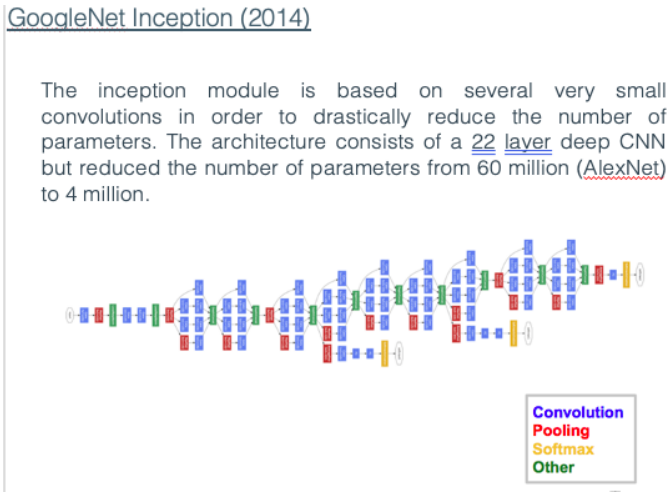


	Layer	Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Facciamo un'altra considerazione su Alex Net e quindi sul perché Alex Net sia stata introdotta solo nel 2012 quindi un bel po' di tempo dopo rispetto alla LeNet5. La spiegazione è abbastanza semplice: da una parte le risorse di calcolo sono aumentate quindi adesso abbiamo a disposizione processori in grado di analizzare e di svolgere compiti complessi, in particolar modo le GPU, quindi le unità grafiche di processamento dell'informazione, dall'altra parte poi abbiamo ovviamente spazi in memoria sempre più grandi, e cosa non banale, in questi anni stiamo capendo come il dato rappresenti la nuova fonte di petrolio, quindi le persone stanno lavorando per raccogliere e annotare sempre una quantità maggiore di dati con una qualità sempre più alta. Quindi cose indispensabili nel momento in cui si vuole sviluppare algoritmi di deep learning.

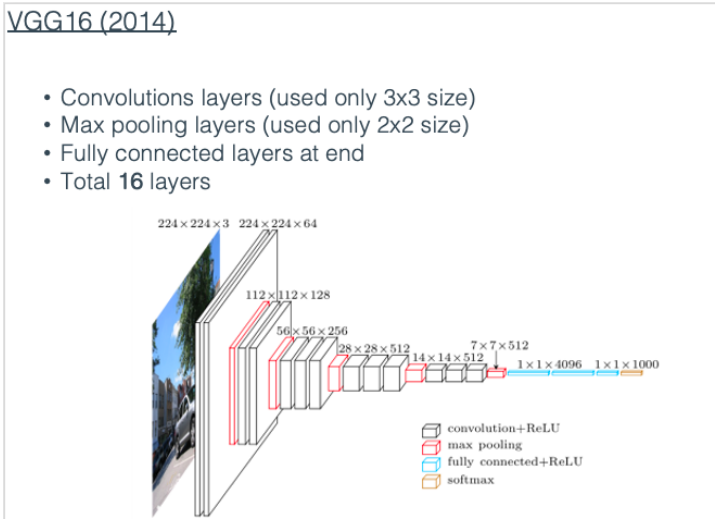
4.3 GoogleNet Inception

Dopo Alex Net e LeNet5 vediamo GoogleNet Inception che è stata proposta nel 2014. L'inception module che è l'innovazione maggiore che è stata portata con il rilascio di GoogleNet è basata su tanti piccoli gruppi di convoluzioni. Ad un certo punto i ricercatori di Google invece che proseguire con l'aggiunta di strati convoluzionali hanno pensato di creare una nuova architettura, con un'architettura che non è più sequenziale ma presenta questi blocchi che si chiamano blocchi di Inception. L'architettura consiste in 22 stati quindi molti di più rispetto a quali di AlexNet ma riduce il numero di parametri da imparare dai 60 milioni di Alex Net ai 4 milioni. Alla fine però i blocchi che costruiscono la nostra rete sono sempre gli stessi quindi convoluzioni, pooling e softmax.



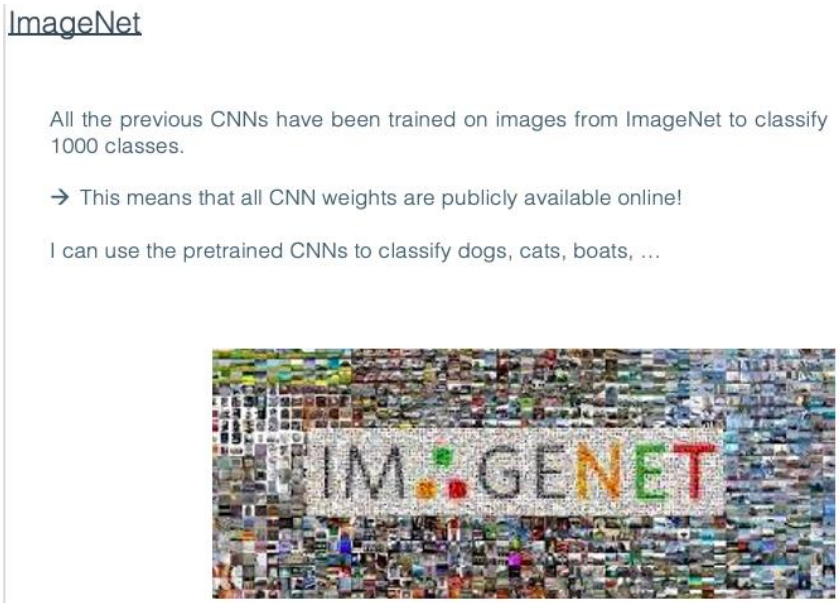
4.4 VGG-16

La VGG16 è una delle architetture di reti neurali convoluzionali (CNN) più utilizzate per la classificazione di immagini. È stata proposta da un gruppo di ricercatori del Visual Geometry Group (VGG) dell'Università di Oxford nel 2014. La VGG16 ha una struttura molto profonda, con 16 strati di tipo convoluzionale e fully connected. La maggior parte dei suoi strati sono convoluzionali con kernel 3x3, con uno stride di 1 pixel e zero padding per mantenere le dimensioni delle mappe delle caratteristiche. Tra gli strati convoluzionali ci sono anche alcuni strati di pooling di tipo max pooling. L'ultimo strato convoluzionale è seguito da tre strati fully connected, che vengono utilizzati per la classificazione. Il primo di questi strati ha 4.096 unità, il secondo 4.096 unità e l'ultimo ha un numero di unità pari al numero di classi di cui si vuole effettuare la classificazione. La VGG16 ha dimostrato di ottenere eccellenti prestazioni in diversi compiti di classificazione di immagini, inclusi quelli su larga scala, come il dataset ImageNet, che contiene oltre 1,2 milioni di immagini appartenenti a 1.000 classi. La VGG16 è stata anche utilizzata come base di partenza per la creazione di molte altre architetture di CNN più complesse e sofisticate.



4.5 ImageNet

Ora che abbiamo finito di vedere quelli che sono i modelli più diffusi o comunque i modelli basilari per la classificazione con reti neurali convoluzionali, parliamo un attimo di un dataset chiama ImageNet. ImageNet è probabilmente uno dei dataset più importanti che esistono nel campo della computer vision; presenta diversi milioni di immagini che sono immagini classificate in 1.000 classi, quindi, se vogliamo è il dataset per eccellenza per i problemi di classificazione delle reti neurali convoluzionali. In particolar modo ognuna di queste immagini ha subito un controllo di qualità ed è stata annotata manualmente. Per ogni immagine un umano ha detto qui c'è un gatto qui c'è un cane e così via. Se torniamo indietro e andiamo vedere i modelli più recenti di cui abbiamo parlato quindi ad esempio VGG16 ci rendiamo conto che l'ultimo strato di queste reti aveva 1000 neuroni e ora capiamo il perché, perché tutti quei modelli sono stati sviluppati e allenati a partire dal dataset ImageNet e quindi è stato proprio ImageNet che ha permesso questo boom, questo sviluppo di modelli di reti neurali convoluzionali e la cosa importante è che tutti questi modelli pre allenati su ImageNet sono in realtà disponibili online. Il fatto che un modello pre-allenato è disponibile online vuol dire che abbiamo sia l'architettura della rete quindi ad esempio la VGG 16 con i 16 strati fatti da convoluzione, pooling e così via, sia i valori dei pesi e delle connessioni di ognuno appunto degli strati di VGG16.

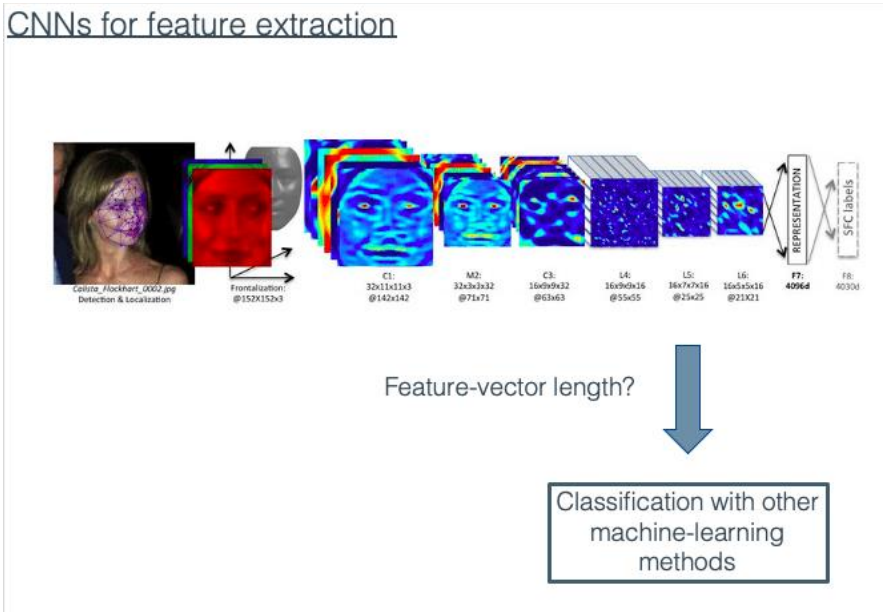


4.6 Fine-Tuning

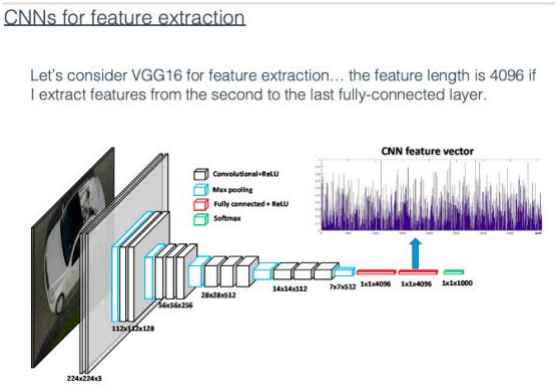
Il fine tuning è una tecnica utilizzata nell'apprendimento profondo (deep learning) per migliorare le prestazioni di una rete neurale pre-addestrata per una specifica attività, adattandola a un compito simile o correlato. In pratica, si tratta di utilizzare una rete neurale già addestrata su un grande dataset per una specifica attività (ad esempio, classificazione di immagini) e di riaddestrarla su un nuovo dataset con un compito simile (ad esempio, una classificazione di immagini simile ma con classi diverse). Il fine tuning si compone di due fasi principali: la prima fase consiste nel prendere una rete neurale pre-addestrata su un grande dataset (ad esempio, una VGG16 addestrata su ImageNet) e di rimuovere l'ultimo strato fully connected, che è quello utilizzato per la classificazione. In questo modo si ottiene una rete neurale che ha imparato a riconoscere delle features comuni a molte immagini. Nella seconda fase, si aggiunge un nuovo strato fully connected, che viene addestrato con un nuovo dataset contenente nuove classi di immagini. In questa fase, i pesi degli strati pre-esistenti sono fissati e non vengono modificati, mentre solo quelli dell'ultimo strato vengono addestrati per il nuovo compito. Questa tecnica è particolarmente utile quando il nuovo dataset disponibile per il nuovo compito è limitato, perché consente di ottenere una rete neurale efficiente ed efficace senza dover addestrare l'intera rete neurale da zero. Inoltre, il fine tuning aiuta a ridurre il rischio di overfitting, poiché la rete neurale pre-addestrata è già stata addestrata su un dataset molto più grande e diversificato di quello utilizzato per il nuovo compito.

4.7 CNNs for feature extraction

Utilizziamo le reti neurali convoluzionali per far estrazione di feature, qui vediamo l'immagine di ingresso, che è un' immagine RGB infatti ha tre canali 152 x152 x3 dove 3 si riferisce appunto al numero di canali; ci spostiamo otteniamo la dimensione dei filtri che è 32 x 11 x 11 x 3, la mappa di attivazione e 142 x142, si scende l'attivazione ha dimensione 71 x71, 63 x 63 e così via. Più scendiamo più la dimensione della mappa di attivazione scende probabilmente perché stiamo effettuando dei pooling quindi la risoluzione scende ma aumenta il numero di canali. Per concludere abbiamo lo strato fully convolution il penultimo che ha 4096 neuroni e l'ultimo che ha 4.030 canali neuroni.



Questo numero ci viene dato dalla scelta dello strato da cui vogliamo estrarre le feature. Ora ritroviamo la nostra VGG16 quindi con i 16 canali convoluzione e Relu, max pooling, fully connected e Relu e in ultimo la softmax che ci dà la predizione delle mille classi di ImageNet. In questo caso se decido di estrarre le feature dal penultimo strato di VGG 16. Le mie feature saranno 4096 che saranno le uscite di questi 4096 neuroni del penultimo strato di VGG16. In questo caso quindi avrò il mio spazio di feature 4096 per ognuna delle immagini di training e procederò ad allenare il mio algoritmo di machine learning standard.



4.8 Transfer Learning

Vediamo ora un secondo concetto che ha a che fare col fine tuning. In realtà oltre al fine tuning c'è anche un altro processo che si chiama transfer learning. Il transfer learning riguarda il trasferimento della rappresentazione appresa durante il training di una CNN su un altro problema. Quindi diciamo che è il nostro problema è classificare se in un'immagine di risonanza c'è o no un tumore, quello che stiamo facendo è prendere una rete pre allenata su ImageNet che quindi riguarda immagini naturali e stiamo trasferendo la conoscenza la rappresentazione appresa su un dataset di immagini naturali per risolvere un problema invece di diagnosi su immagini di risonanza magnetica cardiaca. Ad esempio quello che si può fare è presa VGG16 pre-allenata su ImageNet prendo la stessa rete con i pesi appresi con il training su imagenet e modifico a partire da quei pesi, quindi da quell'inizializzazione dei pesi, il valore dei paesi durante il training. Per esempio, durante il transfer learning invece che modificare tutti i pesi della nuova rete neurale convoluzionale si può decidere di fissare alcuni strati pre-allenati e lasciare che invece gli altri si adattino durante il training al nuovo task.

Transfer learning is about "transferring" the representation learnt during the training of a CNN to another problem. For example, one can use pretrained CNN features to initialize the weights of a new CNN, developed for a different task.

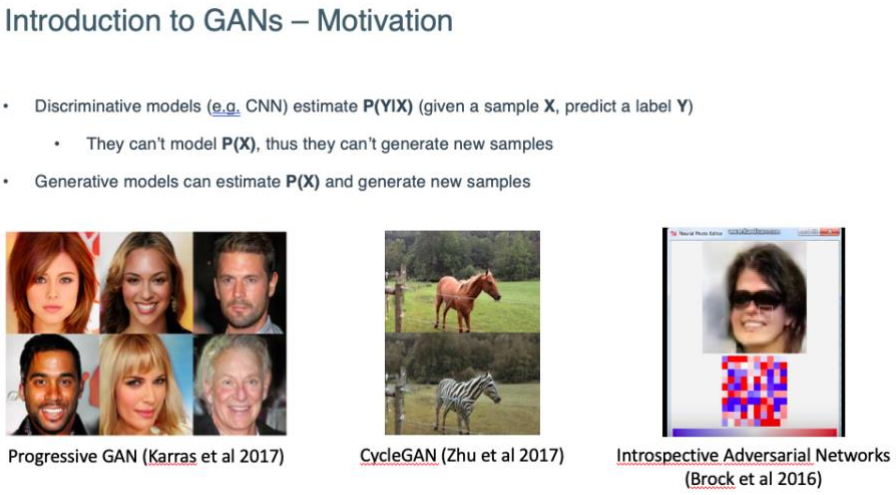
Fine tuning is about making **fine adjustments** to further improve performance. For example, during transfer learning, you can unfreeze some of (or all) the pre-trained CNN layers and let it adapt more to the task at hand.

5. Generative Adversarial Networks (GANs)

In questa lezione approfondiremo i modelli generativi in particolare ci occuperemo delle le Generative Adversarial Networks (GAN). Andremo ad introdurre che cosa sono le GAN la loro definizione, cos'è questo strumento e la loro architettura così come il loro training e vedremo alcuni dei problemi che derivano dalle Gan e alcune delle varianti che sono state proposte. Nei capitoli precedenti avete visto come sistemi di intelligenza artificiale odierni sono in grado di riconoscere persone animali e oggetti con una precisione che è addirittura maggiore di quella dell'uomo. Questo significa che grazie all'apprendimento automatico i dispositivi possono modificare i propri algoritmi per adattarsi a imparare nuove abilità dipendenti dai dati che devono analizzare. Oggi però gli algoritmi di intelligenza artificiale hanno acquisito una qualità nuova, che

sinora era un'esclusiva dell'intelligenza umana questa qualità è la creatività. Questo è merito proprio delle reti generative le GAN, una classe di algoritmi usati nell'apprendimento automatico non supervisionato che implementa non una, ma due reti neurali che si scontrano tra loro per poter creare oltre che a prendere. Le GAN o generative adversarial network, sono quindi modelli generativi attraverso un processo avversario o antagonista che prevede l'allenamento simultaneo di un generatore, generator o G, e di un discriminatore, discriminator o D. Le due reti antagoniste sono addestrate in modo interlacciato quindi il generatore deve produrre immagini realistiche rispetto ad un training set a partire da random noise a livello tale da poter ingannare il discriminatore. Il discriminatore invece deve riuscire a distinguere le immagini generate fake da quelle vere, real. Le GAN sono state definite Yann LeCun padre del deep learning, e vincitore del premio Turing nel marzo 2019, come l'idea più interessante è che abbiamo avuto nel machine learning negli ultimi vent'anni. Al seguente link, potete vedere un video che mostra le potenzialità di questo strumento: https://www.youtube.com/watch?v=8OJnkJqkyio

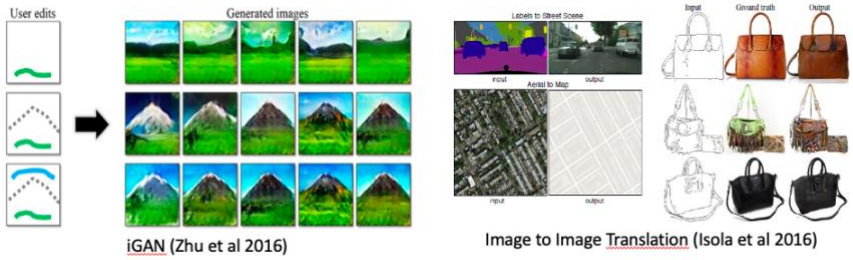
Quello che avete visto è un esempio di deep fake in cui una scena del film "ritorno al futuro" è stata rifatta artificialmente attraverso una GAN e sono stati scambiati i protagonisti Robert Downey Jr and Tom Holland. La motivazione che ha portato all'utilizzo di questo strumento è che i modelli classici o discriminativi possono stimare la probabilità che un elemento appartenga a un certo insieme però non possono creare nuovi esempi, nuovi elementi di questo insieme. Per questo sono necessari dei modelli generativi che simulano la distribuzione di un elemento che possa appartenere a questo insieme e che quindi siano in grado di generare nuovi esempi, elementi che appartengono allo stesso insieme. Nella figura di seguito, vediamo alcuni esempi. Vediamo i deep fake nella Progressive GAN vediamo la Cycle GAN in cui la foto di un cavallo viene trasformata nella foto di una zebra e poi vediamo un esempio di Introspective Adversarial Network in cui può essere effettuata una super risoluzione sommando all'infinito una foto.



Quindi l'obiettivo è di allenare un modello che riconosca una distribuzione di esempi per ottenere nuovi esempi che abbiano una qualità accettabile. Nella seguente immagine, sono illustrati ulteriori esempi di GAN:

Introduction to GANs – Motivation

- In **generative modeling**, we'd like to train a network that models a distribution, such as a distribution over images.
- One way to judge the quality of the model is to sample from it.



iGAN (Zhu et al 2016)

Image to Image Translation (Isola et al 2016)

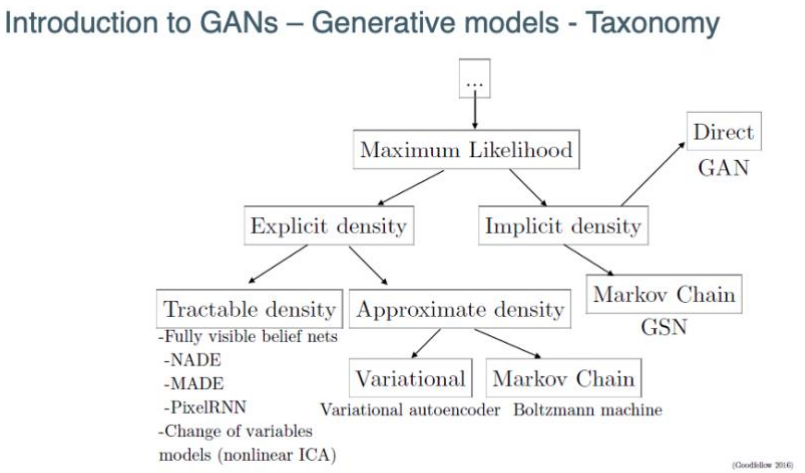
Potete vedere infatti come è possibile disegnare degli schemi che poi la GAN userà per realizzare delle immagini, oppure la cosiddetta image translation in cui ad esempio l'immagine satellitare viene trasformata in una mappa e viceversa. Ci sono diversi tipi di modelli generativi e la GAN è solo uno di questi; al momento è uno dei migliori perché ha una serie di vantaggi come il parallelismo, ha poche restrizioni e in generale gli esempi sono i migliori che possano essere generati ma anche alcuni svantaggi che vedremo poi nel corso di questa lezione; uno tra questi è la difficoltà di effettuare un training adeguato e la difficoltà soprattutto nell' effettuare una valutazione. Infatti, attualmente non esistono delle metriche per valutare il funzionamento di una GAN quindi per valutare la sua qualità, inoltre non può essere calcolata formalmente la cosiddetta likelihood quindi la distribuzione degli esempi generati.

Introduction to GANs – Generative models

Several modern approaches to generative modeling:

- Reversible architectures
- Autoregressive models
- Variational autoencoders
- **Generative Adversarial Networks** (our focus)
    - Advantages: sample in parallel, few restrictions on generator function, no Markov Chain, no variational bound, subjectively better samples
    - Disadvantages: very difficult to train properly, difficult to evaluate, likelihood can't be computed, no encoder (in vanilla GAN)

Nella seguente immagine vediamo alcune classi dei vari modelli generativi e possiamo evincere facilmente che la GAN è un tipo di modello implicito e modelli impliciti insieme modelli espliciti formano i modelli che massimizzano la likelihood.



Vediamo adesso in particolare il modello della GAN cioè della Generative Adversarial Network, che è un tipo particolare di modello implicito. La GAN è stata presentata da Goodfellow nel 2014 e si articola non in una ma in due reti neurali concorrenti, come abbiamo detto all'inizio. La prima indicata come generatore è una rete deconvoluzionale che sulla base di un'immagine iniziale a bassa risoluzione ne elabora una ad alta risoluzione che propone alla seconda, indicata come discriminatore e organizzata invece come una rete convoluzionale. Il discriminatore è alimentato dalla stessa immagine ma ad alta risoluzione da cui va ad estrarre le caratteristiche salienti tipicamente rese in forma di distribuzioni probabilistiche, che confronta con le caratteristiche dell'elaborazione proposta invece dal generatore. Il processo prosegue fin quando le caratteristiche dell'immagine elaborata dal generatore e quelle della stessa immagine ma ad alta risoluzione e nota solo al discriminatore convergono. La GAN consiste in un minimax game tra il modello generativo e quello discriminativo in cui uno tende a massimizzare una funzione e l'altro tende a minimizzarla, quindi vengono allenati in opposizione l'uno all'altro. Nella teoria delle decisioni il minimax è un metodo per minimizzare la massima perdita possibile quindi minimax, oppure per massimizzare il minimo guadagno maximin. L'obiettivo della GAN come per i modelli impliciti è quello di imparare a modellare questa distribuzione.

6. Bibliografia

[1] Zhou, Z. H. (2021). Machine learning. Springer Nature.
[2] Mitchell, T. M. (2007). Machine learning (Vol. 1). New York: McGraw-hill.
[3] Alpaydin, E. (2021). Machine learning. Mit Press.
[4] Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. Electronic Markets, 31(3), 685-695.
[5] Wang, H., Lei, Z., Zhang, X., Zhou, B., & Peng, J. (2016). Machine learning basics. Deep learning, 98-164.
[6] Hahne, F., Huber, W., Gentleman, R., Falcon, S., Gentleman, R., & Carey, V. J. (2008). Unsupervised machine learning. Bioconductor case studies, 137-157.
[7] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160(1), 3-24.
[8] Singh, A., Thakur, N., & Sharma, A. (2016, March). A review of supervised machine learning algorithms. In 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1310-1315). Ieee.
[9] Mahesh, B. (2020). Machine learning algorithms-a review. International Journal of Science and Research (IJSR).[Internet], 9, 381-386.
[10] Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O. P., Tiwari, A., ... & Lin, C. T. (2017). A review of clustering techniques and developments. Neurocomputing, 267, 664-681.
[11] Ray, S. (2019, February). A quick review of machine learning algorithms. In 2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon) (pp. 35-39). IEEE.
[12] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.
[13] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.
[14] Kelleher, J. D. (2019). Deep learning. MIT press.
[15] Rusk, N. (2016). Deep learning. Nature Methods, 13(1), 35-35.
[16] Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. Electronic Markets, 31(3), 685-695.
[17] Deng, L., & Yu, D. (2014). Deep learning: methods and applications. Foundations and trends® in signal processing, 7(3–4), 197-387.
[18] Hua, Y., Guo, J., & Zhao, H. (2015, January). Deep belief networks and deep learning. In Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things (pp. 1-4). IEEE.
[19] Howard, J., & Gugger, S. (2020). Deep Learning for Coders with fastai and PyTorch. O'Reilly Media.
[20] Sejnowski, T. J. (2018). The deep learning revolution. MIT press.
[21] Buduma, N., Buduma, N., & Papa, J. (2022). Fundamentals of deep learning. " O'Reilly Media, Inc.".
[22] Gulli, A., & Pal, S. (2017). Deep learning with Keras. Packt Publishing Ltd.
[23] Du, X., Cai, Y., Wang, S., & Zhang, L. (2016, November). Overview of deep learning. In 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC) (pp. 159-164). IEEE.
[24] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative adversarial networks: An overview. IEEE signal processing magazine, 35(1), 53-65.
[25] Wang, K., Gou, C., Duan, Y., Lin, Y., Zheng, X., & Wang, F. Y. (2017). Generative adversarial networks: introduction and outlook. IEEE/CAA Journal of Automatica Sinica, 4(4), 588-598.
[26] Aggarwal, A., Mittal, M., & Battineni, G. (2021). Generative adversarial network: An overview of theory and applications. International Journal of Information Management Data Insights, 1(1), 100004.
[27] Arjovsky, M., & Bottou, L. (2017). Towards principled methods for training generative adversarial networks. arXiv preprint arXiv:1701.04862.
[28] Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., ... & Kasneci, G. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, 103, 102274.
[29] Zhu, J. J., Jiang, J., Yang, M., & Ren, Z. J. (2023). ChatGPT and environmental research. Environmental Science & Technology.
[30] Patel, S. B., & Lam, K. (2023). ChatGPT: the future of discharge summaries?. The Lancet Digital Health, 5(3), e107-e108.