UNIVERSITÀ
DEGLI STUDI
DI UDINE
hic sunt futura

Dipartimento di Scienze
Matematiche, Informatiche e Fisiche

TESI DI LAUREA IN
INFORMATICA

Testing automatizzato di classificatori di immagini basati su Deep Learning

CANDIDATO
Stefano MONTE

RELATORE
Prof. Vincenzo RICCIO

Anno accademico 2022-2023

Contatti dell'Istituto
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

Indice

I

Contesto

1

Introduzione

L'introduzione delle reti neurali rappresenta un punto cruciale nell'ambito dell'Informatica e dell'Intelligenza Artificiale, svelando un universo complesso di algoritmi ispirati al funzionamento del cervello umano. Le reti neurali hanno rapidamente assunto un ruolo centrale, emergendo come uno strumento cruciale nella risoluzione di compiti complessi e nell'apprendimento automatico. La crescente adozione di reti neurali in una vasta gamma di applicazioni, dalla visione artificiale al linguaggio naturale, sottolinea l'importanza di comprendere profondamente il loro comportamento e la loro affidabilità. Le reti neurali costituiscono un paradigma computazionale ispirato al funzionamento del cervello umano, che si basa sull'elaborazione di informazioni attraverso nodi interconnessi chiamati neuroni artificiali. L'evoluzione di questo campo ha portato alla creazione di architetture sempre più sofisticate, tra le quali le reti neurali convoluzionali, progettate specificamente per affrontare problemi legati alla visione artificiale.

Questa tesi si propone di esplorare a fondo l'utilizzo delle reti neurali, concentrandosi in particolare reti neurali convoluzionali profonde (DNN) per la classificazione di immagini.

In questo contesto, la pratica del testing delle reti neurali emerge come un approccio essenziale per garantire la robustezza e l'affidabilità di tali modelli, specialmente considerando la complessità delle relazioni apprese durante il processo di addestramento. In particolare, é sempre piú diffusa la pratica di generare artificialmente input che non erano presenti nel dataset originale da cui tali DNN hanno appreso. Tale pratica permette di valutare la qualità delle DNN in casi estremi o imprevisti. Questa tesi descriverà brevemente la struttura delle reti neurali, esplorando le loro componenti fondamentali e analizzando come queste interagiscano per consentire l'apprendimento e l'elaborazione delle informazioni. In particolare, si approfondiranno le peculiarità delle reti neurali convoluzionali, che si sono dimostrate particolarmente efficaci nell'analisi di dati multimediali, come le immagini. In seguito, la tesi si focalizzerà sul testing, uno degli aspetti critici nel processo di sviluppo di software basato su reti neurali. In particolare, saranno esaminate le metodologie e le metriche utilizzate per valutare l'affidabilità e l'efficacia di tali modelli. Infine, la tesi presenterà un contributo originale attraverso l'introduzione del framework OlymTIGs. Questo framework è stato sviluppato per affrontare una sfida cruciale nell'ambito delle reti neurali, ovvero il confronto tra diversi generatori di input avversari. Attraverso OlymTIGs, si esploreranno diversi approcci per la creazione di input artificiali, mettendo in discussione la robustezza dei modelli di deep learning e aprendo nuove prospettive nel campo dell'affidabilità delle reti neurali.

2

Le Deep Neural Networks

Le DNN (Deep Neural Networks) rappresentano una categoria di algoritmi di machine learning in grado di simulare il funzionamento del cervello umano per effettuare previsioni su input specifici.

L'utilizzo di DNN è cresciuto esponenzialmente negli ultimi anni grazie alla loro abilità di svolgere compiti estremamente complessi basati su dati di addestramento. Ad esempio, la classificazione di immagini con le DNN rappresenta un'applicazione chiave nel campo del computer vision, in cui le reti neurali vengono addestrate per assegnare una classe o categoria specifica a un'immagine di input. Questa tecnologia ha rivoluzionato numerosi settori, tra cui la sorveglianza video, la diagnosi medica, l'automazione industriale e molte altre aree in cui l'interpretazione delle immagini è fondamentale.

Questa capacità crea una netta separazione rispetto ai classici sistemi software, ma porta con sé l'impossibilità di prevedere il loro comportamento analizzando semplicemente il codice sorgente con cui sono implementate. Tale aspetto pone una nuova sfida ai processi di verifica e validazione del software, che risultano inefficaci nel testing di DNN.

2.1 Il neurone naturale

Lo sviluppo delle reti neurali artificiali trae ispirazione dalla biologia, poiché il funzionamento delle reti artificiali riflette quello del cervello. Le reti neurali artificiali sono infatti composte da neuroni artificiali interconnessi che lavorano in parallelo per elaborare un input e generare un output, che viene poi inviato ai neuroni successivi.

Un neurone biologico [10] è composto da un corpo centrale chiamato soma, che contiene il nucleo. Dai dendriti, collegati al soma, provengono segnali in ingresso da altri neuroni vicini che vengono trasportati verso il corpo cellulare. L'assone, invece, conduce il segnale di output verso altre cellule circostanti. Oltre ad avere compiti specifici differenti, i dendriti e gli assoni differiscono anche per forma. I dendriti, infatti, si assottigliano dalla parte iniziale a quella terminale e sono pessimi conduttori, ciò comporta la diminuizione dell'intensità dei segnali. Al contrario, invece, gli assoni hanno una sezione costante e sono avvolti da mielina, che li rende eccellenti conduttori.

Un neurone può esistere in due stati: uno stato attivo, in cui i segnali passano attraverso di esso, e uno stato di riposo, in cui i neuroni sono polarizzati, creando una differenza di carica tra l'interno ed l'esterno e impedendo quindi il passaggio di informazioni attraverso i neuriti.

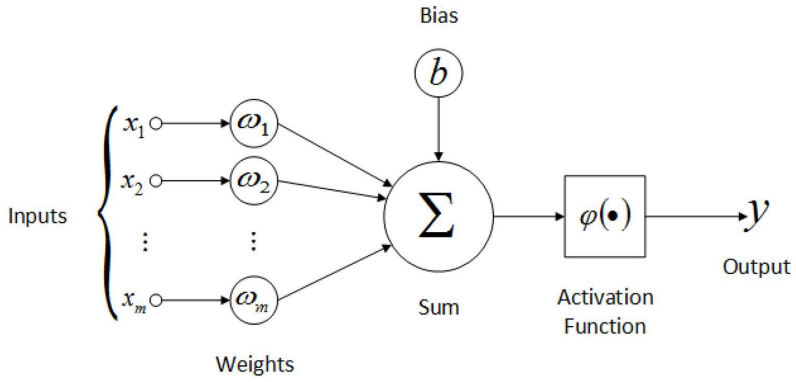

Figura 2.1: Il neurone artificiale.

In un neurone biologico, l'input è composto da una combinazione di segnali che arrivano attraverso i dendriti. Questi segnali in input possono essere sia eccitatori che inibitori, ciascuno contribuendo a un valore specifico. La combinazione dei segnali in ingresso, se supera un certo valore di soglia, attiva il neurone, consentendo ai segnali elaborati di fluire attraverso l'assone. In caso contrario, il neurone rimane inattivo e il segnale non viene trasmesso.

Un aspetto affascinante del funzionamento del cervello è la sua capacità di modificare costantemente i pesi delle connessioni tra i neuroni. Ciò consente la classificazione e la generalizzazione degli stimoli esterni, favorisce l'adattamento apprendendo basandosi sui segnali passati e, di conseguenza, permette di elaborare in modo più rapido ed accurato i nuovi dati.

2.2 La rete neurale artificiale

Le reti neurali artificiali [8], sebbene siano modellate matematicamente, sono ispirate al funzionamento del cervello umano. Un neurone artificiale è definito come un nodo ed è caratterizzato da una funzione di attivazione, una soglia e, talvolta, un bias. Ciascun nodo riceve segnali in ingresso dalle unità precedenti, pesati, e la loro combinazione, a cui si somma il bias, se presente. Questa combinazione diventa quindi l'input per la funzione di attivazione, determinando l'attivazione o la disattivazione del neurone.

Una rete neurale è costituita da nodi organizzati in strati, noti come "layer", e collegati tra loro da pesi. Il primo strato è il "layer di input", l'ultimo è il "layer di output", mentre quelli intermedi sono chiamati "hidden layer". Questi strati intermedi non sono accessibili dall'esterno, poiché tutte le informazioni sulla rete sono conservate nelle matrici che definiscono i pesi. La configurazione più comune è la "feedforward", in cui ciascun nodo è collegato a quelli del layer precedente, dai quali riceve gli input, e a quelli del layer successivo, ai quali fornisce l'output.

L'output del neurone quindi si ottiene come:

$$o = g\left(\sum_{i=1}^{n} w_i \cdot I_i\right)$$

dove:

- I_i rappresenta i segnali in ingresso al neurone, che possono includere sia gli input del problema sia le uscite provenienti da altri neuroni

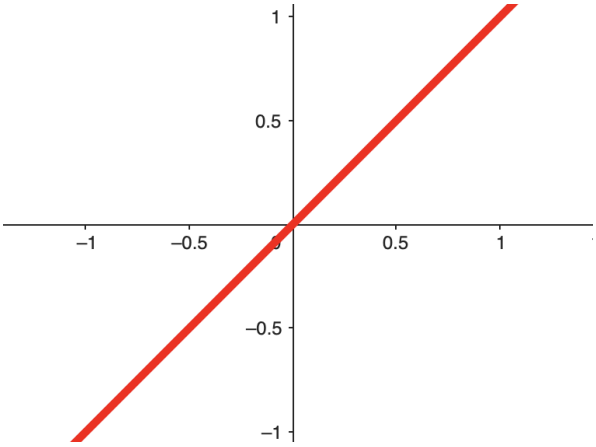

Figura 2.2: Funzione lineare.

- w_i rappresenta i pesi o sinapsi che vengono utilizzati per ponderare ciascun input, determinando in che misura quell'input contribuisce all'attività del neurone
- la sommatoria $\sum$ fornisce la somma pesata degli ingressi.
- g è la funzione di attivazione responsabile di definire l'output del neurone in base al risultato prodotto dal sommatore.

2.3 Le funzioni di attivazione

La scelta della funzione di attivazione [9] è un aspetto chiave e differisce notevolmente dai neuroni biologici. Mentre il neurone biologico si comporta in modo simile a un modello di regressione lineare, la scelta di una funzione non lineare consente di rappresentare in modo più accurato i segnali e affrontare distribuzioni dati complesse.

Le funzioni di attivazioni possono essere raggruppate in tre tipi principali:

2.3.1 La funzioni di attivazione lineare

La funzione di attivazione lineare, chiamata anche funzione dell' identità è direttamente proporzionale all'input, l'intervallo è compreso tra $-\infty$ e $+ +\infty$. Questa funzione somma semplicemente il totale ponderato dagli input e porta al risultato.

$$f(x) = x$$

Questa non è un'attivazione binaria, poiché la funzione di attivazione lineare produce una serie di attivazioni. È possibile collegare neuroni in modo da calcolare il massimo (o il massimo morbido) delle attivazioni se ce ne sono di multiple. La derivata di questa funzione è costante: il che significa che il gradiente non dipende dall'input x.

2.3.2 Funzione di attivazione passo binario

La funzione di attivazione passo binario si comporta come un interruttore, un neurone viene attivato o disattivato in base al confronto tra il valore di ingresso e un valore di soglia. L'attivazione avviene quando

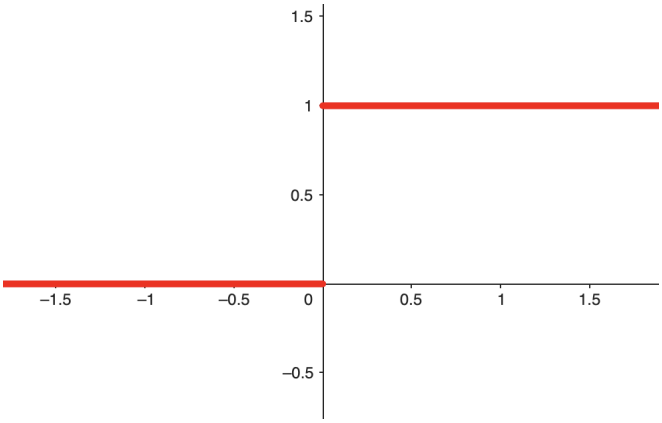

Figura 2.3: Funzione passo binario.

il valore di ingresso supera la soglia, se ciò non avviene il neurone rimane disattivato. In quest'ultimo caso, l'output del neurone non viene trasmesso al livello successivo o nascosto.

$$f(x) = \begin{cases} 0 \text{ se } x < 0 \\ 1 \text{ se } x \geq 0 \end{cases}$$

Questo tipo di unzione di attivazione non è in grado di generare output multivalore, quindi non è adatta per problemi di classificazione multiclasse. Inoltre, è importante notare che il suo gradiente è costantemente zero, il che rende problematica l'applicazione del processo di retropropagazione degli errori.

2.3.3 Funzione di attivazione non lineare

Le funzioni di attivazione non lineare sono ampiamente utilizzate nella costruzione di reti neurali artificiali. Queste funzioni semplificano la capacità di adattare il modello della rete neurale a una vasta gamma di dati e consentono quindi di ottenere output differenziati.

Le funzioni di attivazione non lineare rendono possibile impilare più strati di neuroni in una rete neurale. Ciò è possibile poiché l'output di ogni strato sarebbe ora una combinazione non lineare degli input provenienti dai livelli precedenti.

Queste funzioni di attivazione sono comunemente categorizzate in base alla loro portata e alla forma delle curve che generano.

La funzione sigmoide

La funzione sigmoidea accetta un valore numerico in ingresso e restituisce un valore compreso tra 0 e 1. La sua semplicità d'uso la rende una scelta eccellente, poiché possiede tutte le caratteristiche desiderabili delle funzioni di attivazione: non linearità, derivabilità continua, monotonia e un intervallo di output specifico.

Questo tipo di funzione è prevalentemente impiegato in situazioni di classificazione binaria, in quanto fornisce la probabilità dell'appartenenza a una classe specifica.

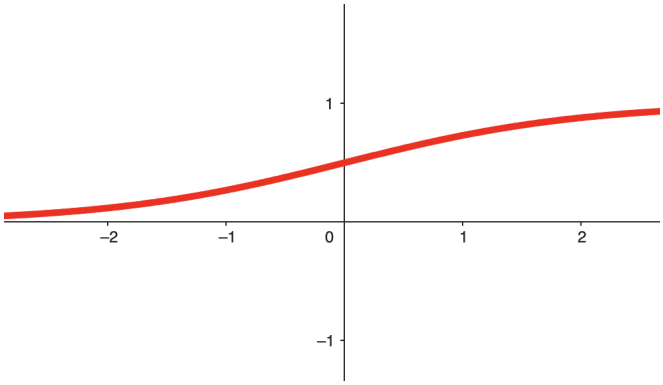

Figura 2.4: Funzione sigmoide.

La funzione Sigmoide è di natura non lineare, ciò sta a significare che le sue combinazioni non saranno lineari e produrranno un'attivazione analoga, a differenza di una funzione di attivazione a passi binari. Questa caratteristica la rende adatta per affrontare problemi di classificazione.

$$f(x) = \frac{1}{1 + e^{-x}}$$

L'uscita di questa funzione di attivazione sarà sempre compresa nell'intervallo (0,1), a differenza della funzione di attivazione lineare, che ha un intervallo da $-\infty$ a $+\infty$. Questa limitazione in termini di output definisce chiaramente l'intervallo in cui operano le attivazioni.

Tuttavia, la funzione Sigmoide presenta alcuni svantaggi. Essa è soggetta al problema dei "gradienti evanescenti", in cui i gradienti diventano molto piccoli durante la retropropagazione, rendendo l'addestramento più difficile. Inoltre, i valori di output non sono centrati su zero, il che può causare aggiornamenti del gradiente che vanno in direzioni molto diverse, complicando ulteriormente l'ottimizzazione.

Queste sfide possono portare la rete a rifiutarsi di apprendere o a renderne estremamente lenta l'addestramento.

La funzione ReLU

ReLU (Rectified Linear Unit) rappresenta una delle funzioni di attivazione maggiormente impiegate in diverse applicazioni. La sua popolarità è dovuta al fatto che risolve il problema della scomparsa del gradiente, in quanto il valore massimo del gradiente per la funzione ReLU è sempre uno. Inoltre, supera il problema della saturazione del neurone poiché la sua pendenza non si annulla mai. Un'altra caratteristica distintiva di ReLU è il suo intervallo, che si estende da zero all'infinito.

$$f(x) = max(0, x)$$

La funzione ReLU presenta diverse caratteristiche importanti da considerare. Prima di tutto, è notevolmente più efficiente dal punto di vista computazionale rispetto alle funzioni sigmoide e TanH, poiché attiva solo un numero limitato di neuroni.

Inoltre, ReLU contribuisce ad accelerare la convergenza durante la discesa del gradiente verso il minimo globale della funzione di perdita. Questo vantaggio deriva dalla sua natura lineare e dalla

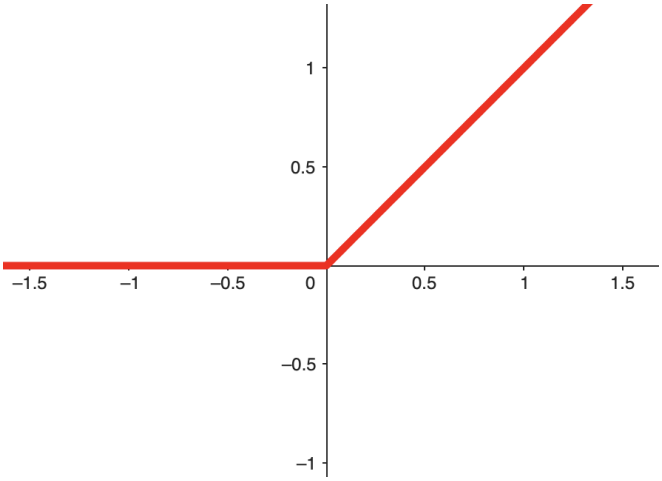

Figura 2.5: Funzione Relu.

mancanza di saturazione.

Tuttavia, è essenziale notare che ReLU dovrebbe essere utilizzata esclusivamente all'interno dei livelli nascosti di un modello di rete neurale artificiale. Ciò è dovuto al fatto che durante l'addestramento, alcuni gradienti possono diventare fragili, in particolare per le attivazioni nella regione (x¡0) di ReLU. In questa zona, la pendenza è zero, il che significa che i pesi non vengono regolati durante la discesa del gradiente. Di conseguenza, i neuroni in questo stato smettono di rispondere alle variazioni dell'input, creando ciò che è noto come il "problema ReLU morente".

La funzione Softmax

La funzione Softmax è una combinazione di molte sigmoidi ed è utilizzata per determinare le probabilità relative tra diverse classi o etichette. In modo simile alla funzione di attivazione sigmoidea, la funzione Softmax calcola le probabilità associate a ciascuna classe o etichetta. Questo la rende particolarmente adatta alla classificazione multiclasse, ed è spesso impiegata nell'ultimo livello di una rete neurale per questo scopo.

La caratteristica distintiva della funzione Softmax è che fornisce la probabilità della classe corrente rispetto alle altre classi possibili. In altre parole, non si limita a stabilire la probabilità di un'unica classe, ma tiene conto delle possibilità relative di tutte le classi, consentendo di effettuare una scelta ponderata in termini di probabilità per la classe corrente rispetto alle alternative. Questo la rende una scelta fondamentale per problemi di classificazione multiclasse in cui è necessario considerare più opzioni contemporaneamente.

La funzione Softmax eccelle nell'imitare l'etichetta codificata in modo più efficace rispetto all'utilizzo di valori assoluti. Utilizzando il modulo (valore assoluto), rischiamo di perdere informazioni importanti, ma l'esponenziale nella funzione Softmax gestisce questa problematica in modo automatico.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Va notato che la funzione Softmax non è solo appropriata per la classificazione multiclasse, ma può anche essere utilizzata con successo in problemi di classificazione multi-etichetta e persino in attività di

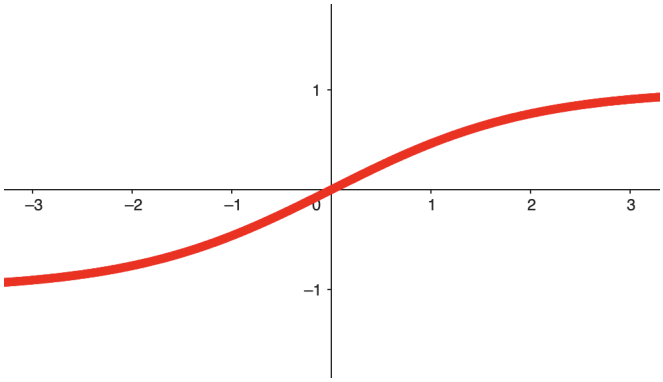

Figura 2.6: Funzione Softmax.

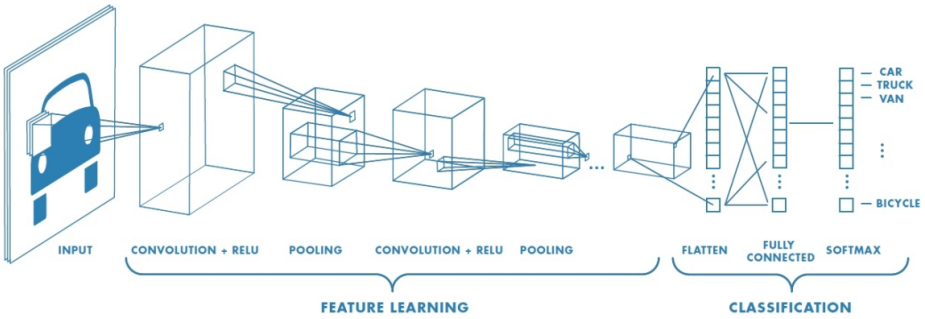

Figura 2.7: Rete convoluzionale.

regressione. La sua capacità di fornire probabilità relative tra le diverse classi o etichette la rende una scelta versatile per una vasta gamma di applicazioni di apprendimento automatico.

2.4 Le Reti convoluzionali

Le reti neurali convoluzionali, note anche come Convolutional Neural Networks (CNN), costituiscono un tipo particolare di rete neurale artificiale ed hanno dimostrato prestazioni eccezionali nel rilevamento di pattern e nel riconoscimento di immagini. Sono ampiamente utilizzate nell'ambito della classificazione di immagini, in cui sono tra le architetture più popolari.

Queste reti neurali si distinguono da altri tipi di reti per la loro abilità nell'elaborazione di immagini, input vocali e segnali audio. La loro architettura è composta da tre tipi principali di strati, ognuno con una funzione specifica: il livello convoluzionale, il livello di pooling e il livello completamente connesso.

Il livello convoluzionale è il punto di partenza di una CNN ed è fondamentale per la sua operazione. In questo livello, vengono applicati filtri convoluzionali all'input, noti come kernel (tipicamente di dimensione 3x3), al fine di attivare specifiche feature dell'immagine (*Fig. 2.8*). Questa operazione di

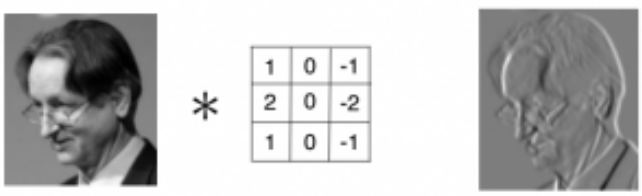

Figura 2.8: Applicazione di un filtro.

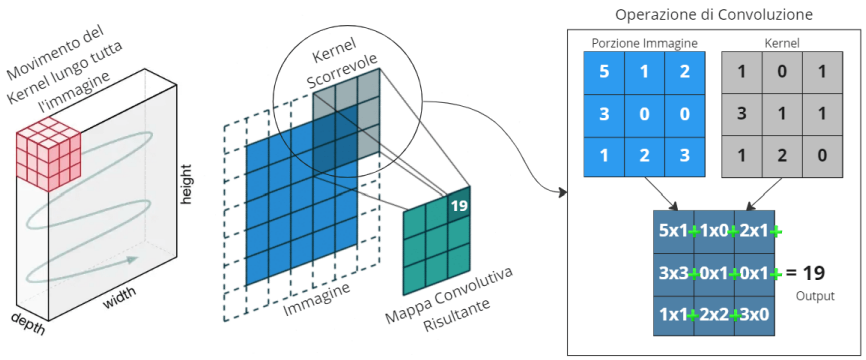

Figura 2.9: Livello convoluzionale

convoluzione è responsabile della maggior parte dei calcoli e consiste nel muovere il kernel attraverso l'immagine, calcolando prodotti di punti tra il kernel e l'input in diverse posizioni (*Fig. 2.9*). Il risultato di questa operazione è una mappa delle funzioni o mappa di attivazione, che rappresentano le caratteristiche estratte dall'immagine.

Dopo ciascuna operazione di convoluzione, una CNN applica una funzione di attivazione ReLU (Rectified Linear Unit) per introdurre non linearità nel modello. Questo passaggio è cruciale per consentire alle reti neurali di apprendere pattern complessi.

Un aspetto importante di una CNN è la possibilità di avere più strati convoluzionali, ciò rende l'architettura gerarchica. Ogni strato successivo può vedere i pixel all'interno dei campi ricettivi dei livelli precedenti, il che consente di riconoscere pattern sempre più complessi nell'immagine. Ad esempio, in una CNN che riconosce una bicicletta, i primi strati possono concentrarsi su feature come i colori e i contorni, mentre strati successivi possono riconoscere parti specifiche della bicicletta.

I livelli di pooling, noti anche come livelli di sottocampionamento, svolgono un ruolo importante nella riduzione della dimensionalità dell'input. Questo processo aiuta a ridurre il numero di parametri da apprendere e a migliorare l'efficienza del modello. Esistono due tipi principali di pooling: il max pooling, che seleziona il valore massimo all'interno di un'area dell'input, e l'average pooling, che calcola la media dei valori nell'area. Sebbene il pooling comporti una perdita di informazioni, contribuisce significativamente a evitare l'overfitting.

Infine, il livello completamente connesso (FC) rappresenta l'ultimo stadio di una CNN. Qui, tutti i neuroni del livello sono collegati a tutti quelli del livello successivo, consentendo di combinare le informazioni estratte dai livelli precedenti per la classificazione dell'immagine. Solitamente, questo livello fa uso di una funzione di attivazione softmax per assegnare una probabilità alle diverse classi di output, indicando con quale confidenza l'immagine appartiene a ciascuna categoria.

2.5 Addestramento delle reti convoluzionali

L'allenamento e la successiva valutazione della bontà della rete costruita sono le attività principali che si svolgono quando si costruisce una rete neurale

L'addestramento di una rete neurale comporta l'ottimizzazione dei pesi delle connessioni tra i nodi (o neuroni) al fine di apprendere dalle osservazioni fornite durante il processo di addestramento. Il suo obiettivo principale è mappare in modo accurato le relazioni tra gli input e gli output.

2.5.1 Discesa del gradiente

Nel tipico approccio di apprendimento, noto come discesa del gradiente, l'obiettivo è minimizzare la funzione di errore $E(W)$. Per ciascun pattern x_i, questa funzione di errore calcola la discrepanza tra la previsione della rete $f_W(x_i)$ (che dipende dai pesi W) e il valore reale (l'etichetta). L'approccio è iterativo, e ad ogni iterazione, ci si sposta nella direzione opposta al gradiente. La regola di aggiornamento dei pesi è data da:

$$W(t+1) = W(t) - \eta \frac{\partial E(W)}{\partial W}\bigg|_{W=W}$$

Dove η rappresenta il parametro di apprendimento, noto come tasso di apprendimento (learning rate).

Nel processo iterativo di addestramento, l'aggiornamento dei pesi avviene sulla base dell'errore. Questo aggiornamento può avvenire in due modi distinti:

- Approccio batch: le modifiche ai pesi vengono apportate solo dopo aver presentato tutti i pattern dell'insieme di addestramento alla rete. In altre parole, si valuta l'errore della rete sull'intero insieme di esempi prima di procedere con l'aggiornamento dei pesi. L'idea principale è apportare poche ma significative modifiche ai pesi.

- Approccio on-line: le modifiche ai pesi avvengono dopo la presentazione di ciascun singolo pattern. Si procede quindi con molte piccole modifiche, aggiornando i pesi ad ogni passo del processo di apprendimento.

L'approccio più comunemente adottato è il secondo, poiché fissando il parametro di apprendimento η a un valore sufficientemente basso e selezionando casualmente gli esempi da presentare alla rete, si permette una vasta esplorazione dello spazio della funzione di costo.

Questa tecnica presenta però anche alcuni svantaggi:

- il parametro di apprendimento η è cruciale, poiché controlla la correzione dei pesi ad ogni iterazione, influenzando la velocità di convergenza. η troppo piccolo rallenta la convergenza, mentre η troppo grande può causare oscillazioni indesiderate e ostacolare la determinazione precisa del minimo.

- L'algoritmo tende a convergere verso minimi locali. La scelta iniziale dei pesi è importante; se i pesi iniziali sono maggiori di W, l'algoritmo raggiunge il minimo globale dell'errore, altrimenti, può convergere in un minimo locale.

- Se durante la discesa del gradiente si raggiunge una zona in cui le funzioni di attivazione diventano costanti durante la discesa del gradiente e la derivata di g è vicina a zero, l'aggiornamento dei pesi diventa molto piccolo, riducendo la velocità di apprendimento. Questo problema può essere mitigato con pesi iniziali bassi, specialmente all'inizio, per evitare la saturazione delle funzioni di attivazione.

- Calcolare la derivata di E rispetto a W (il numero di pesi nella rete) con il rapporto incrementale è computazionalmente costoso ($O(W^2)$).

2.5.2 Backpropagation

L'algoritmo di Back Propagation è una delle principali, se non la principale soluzione, per l'addestramento delle reti neurali. Esso sfrutta la teoria espressa nella discesa del gradiente, che garantisce di trovare il minimo della loss function, avanzando gradualmente sulla superficie della funzione, ricalcolando costantemente i pesi della rete a partire dai valori precedentemente noti e utilizzando l'errore più recente calcolato al fine di ridurlo. Si compone di due fasi, la prima, nota come "feedforward" o fase in avanti, in cui i dati di input vengono propagati attraverso la rete in un processo chiamato "feedforward". I dati vengono trasformati attraverso i pesi e i bias dei collegamenti sinaptici tra i neuroni, passando attraverso le funzioni di attivazione nei neuroni nascosti e nell'output per poi procedere con la valutazione dell'errore calcolando la differenza tra l'output previsto e l'output desiderato.

La loss function è definita come:

$$E(w) = \sum_{i=2}^{M} \frac{1}{2} (d_i - y_i)^2$$

dove d_i è il valore da ottenere, y_i è il valore ottenuto ed M è il numero totale degli elementi.

La seconda fase, chiamata "retropropagazione" o fase all'indietro, è la fase in cui il segnale di errore viene propagato all'indietro attraverso la rete e i pesi dei collegamenti sinaptici vengono aggiornati utilizzando la discesa dei gradienti, in modo da ridurre l'errore di uscita.

Facendo riferimento all'espressione *2.5.1*, un valore basso del tasso di apprendimento η comporta un processo di convergenza molto lento. Questo significa che la regolazione dei pesi della rete avviene gradualmente, il che potrebbe richiedere molto tempo per raggiungere un risultato accettabile. Tuttavia, una volta raggiunti, i pesi tendono ad avvicinarsi ai valori ottimali, consentendo alla rete di apprendere in modo preciso e accurato. D'altra parte, l'uso di valori elevati per il tasso di apprendimento implica un apprendimento più veloce, ma può potenzialmente causare problemi. Un tasso di apprendimento troppo alto potrebbe far oscillare i pesi, rendendo il processo di convergenza instabile e impedendo alla rete di raggiungere una soluzione ottimale. La rete, infatti, potrebbe convergere rapidamente ma con valori subottimali dei pesi, sfociando in previsioni inaccurate. L'algoritmo ottimizza il calcolo della derivata di E rispetto a W ottenendo una complessità totale di $O(W)$.

2.5.3 Come avviene l'allenamento

Dopo aver assimilato il concetto fondante dell'apprendimento, ora ci soffermiamo su come una rete neurale effettivamente viene allenata.

La pratica più comune per valutare l'efficacia è quella di suddividere il dataset (una collezione organizzata di numerose immagini associate alle rispettive etichette) in tre set più piccoli: training, validazione e test.

Il set di training in generale è quello con il maggior numero di input, ed è utilizzato per l'addestramento della rete neurale, il set viene passato alla rete un'immagine alla volta oppure a gruppo di poche immagini (batch) che esegue tutti i processi per il ricalcolo dei pesi e di addestramento spiegati nella sezione precedente. Quando il classificatore scorre tutte le immagini contenute nel dataset si dice che

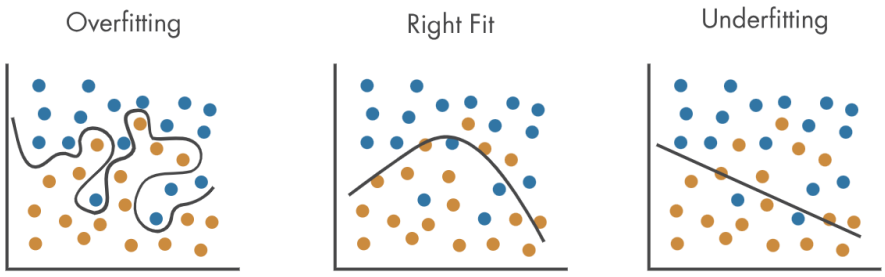

Figura 2.10: overfitting e underfitting.

ha compiuto un'epoca. La pratica comune è compiere più epoche sullo stesso dataset in modo che il classificatore riesca meglio ad apprendere le caratteristiche degli input.

2.5.4 Prevenire l'overfitting

È fondamentale determinare il momento giusto per fermare l'addestramento di una rete neurale al fine di evitare l'overfitting, cioè il problema in cui la rete apprende troppo bene i dati di addestramento a discapito della sua capacità di generalizzazione. La minimizzazione dell'errore sul set di addestramento non è una garanzia di buona generalizzazione, quindi è importante adottare strategie di controllo per evitare che la rete memorizzi i dati anziché apprenderne i modelli.

In questa fase vengono utilizzati le altre due parti dell'intero set lasciate prima in disparte, validazione e test. Il processo di addestramento della rete è monitorato costantemente durante le epoche, e l'errore sulla porzione di dati di validazione o test viene valutato ad ogni epoca. Ci sono diverse strategie per decidere quando fermare l'addestramento:

- Early Stopping: Si ferma l'addestramento quando l'errore sulla porzione di validazione smette di migliorare o inizia a peggiorare. Questa tecnica aiuta a prevenire l'overfitting, poiché indica quando la rete ha smesso di generalizzare bene.
- Limitazione delle epoche: Si può fissare un numero massimo di epoche e fermare l'addestramento quando questo numero viene raggiunto. Questo metodo può essere utile, ma presenta degli svantaggi in quanto non tiene conto delle condizioni reali di convergenza della rete.
- Regolarizzazione: L'uso di tecniche di regolarizzazione come la regolarizzazione L1 o L2 può aiutare a prevenire l'overfitting e permette di prolungare l'addestramento per un maggior numero di epoche senza il rischio di memorizzare i dati.

In generale, l'obiettivo è trovare un equilibrio tra il miglioramento delle prestazioni della rete e la prevenzione dell'overfitting. La valutazione dell'errore sulla porzione di validazione o test è fondamentale per prendere decisioni informate sulla fine dell'addestramento.

3

Il testing

3.1 L'importanza del testing

L'importanza del testing nelle DNN è fondamentale per garantire l'affidabilità, la robustezza e la sicurezza delle applicazioni basate su queste reti neurali. Le DNN sono spesso utilizzate in scenari critici in cui errori o comportamenti indesiderati possono avere conseguenze significative o addirittura critiche o fatali per le vite umane. Il testing delle DNN consente di valutare e validare il comportamento del modello in una varietà di situazioni, garantendo che esso sia in grado di prendere decisioni accurate e coerenti.

Il testing delle DNN è cruciale per garantire che queste reti neurali abbiano un impatto positivo e affidabile in un'ampia gamma di applicazioni, dall'automazione industriale alla diagnosi medica e molto altro. La sua importanza crescerà ulteriormente con l'evoluzione della tecnologia e l'adozione sempre più diffusa delle DNN in diversi campi, contribuendo a garantire l'affidabilità e la sicurezza delle applicazioni basate su intelligenza artificiale.

Il testing delle DNN è una fase fondamentale nel ciclo di sviluppo di queste reti neurali artificiali. Infatti, esso contribuisce a garantire che le DNN siano affidabili, sicure e rispettose degli standard etici. Senza un testing adeguato, le DNN potrebbero produrre risultati imprevedibili, mettendo a rischio l'efficacia e la sicurezza delle applicazioni in cui vengono impiegate. Pertanto, il testing costituisce una componente insostituibile nel processo di sviluppo e implementazione delle DNN.

Il testing delle reti neurali si riflette quindi in diversi aspetti chiave:

- Garantire l'affidabilità. Il testing delle DNN è fondamentale per garantire l'affidabilità dei risultati prodotti da queste reti. A causa della loro profonda complessità e della vastità dei parametri, le DNN possono facilmente comportarsi in modo imprevedibile o indesiderato in situazioni reali. Ad esempio, un modello di riconoscimento delle immagini potrebbe commettere errori gravi, con conseguenze significative, se non sottoposto a un rigoroso testing. Il testing è il mezzo principale per identificare e correggere tali problemi.

- Valutare la capacità di generalizzazione. Una delle sfide principali nell'addestramento delle DNN è la capacità di generalizzazione. Le DNN devono essere in grado di apprendere dai dati di addestramento e applicare tali conoscenze a dati nuovi e mai visti. Il testing è essenziale per

valutare la capacità di una DNN di generalizzare con successo, evitando l'overfitting, in cui il modello si adatta troppo ai dati di addestramento e non è in grado di generalizzare. Le DNN, grazie alla loro capacità di apprendimento automatico, possono adattarsi ai dati di addestramento, ma questo non garantisce che esse generalizzino in modo corretto su dati nuovi e sconosciuti. Il testing aiuta a identificare eventuali deviazioni dal comportamento desiderato, inclusi casi di overfitting, in cui il modello si adatta eccessivamente ai dati di addestramento, diventando incapace di generalizzare in modo affidabile. Attraverso il testing, si possono individuare problematiche relative alla generalizzazione errata, che possono avere ripercussioni gravi in applicazioni come la diagnosi medica o la guida autonoma.

- Identificare vulnerabilità. Le DNN sono soggette a vulnerabilità, sia nella loro struttura che nei dati su cui sono addestrate. Il testing avversario, che coinvolge l'utilizzo di dati progettati per ingannare o confondere il modello, è essenziale per identificare e mitigare tali vulnerabilità. Ad esempio, nel campo della sicurezza informatica, il testing avversario può rivelare debolezze nella capacità di una DNN di rilevare intrusioni o attacchi informatici.

- Sicurezza ed etica. Il testing svolge un ruolo cruciale nella sicurezza e nell'etica delle DNNs. Poiché queste reti possono apprendere comportamenti discriminatori o non etici dai dati di addestramento, il testing è fondamentale per individuare e affrontare tali problemi. Inoltre, il testing aiuta a garantire che le DNN rispettino le normative e le linee guida etiche, evitando risultati inappropriati o dannosi. Inoltre, il testing delle DNN è cruciale per la sicurezza informatica. Le reti neurali possono essere soggette ad attacchi avversari, in cui input dannosi o manipolati sono progettati per ingannare il modello e indurlo a commettere errori. Questi attacchi possono avere un impatto significativo su applicazioni come la sicurezza dei veicoli autonomi o la rilevazione di minacce in ambienti di sorveglianza. Attraverso il testing e la ricerca di vulnerabilità, è possibile identificare e mitigare potenziali rischi per la sicurezza.

- Miglioramento continuo. Il testing delle DNN non è un processo statico. Poiché le applicazioni e la distribuzione dei dati cambiano nel tempo, è essenziale monitorare e testare le DNN in modo continuo. Questo processo di miglioramento continuo assicura che le DNN mantengano le loro prestazioni e l'adeguatezza nel corso del tempo.

  Il testing delle DNN non è un processo statico, ma un elemento dinamico del ciclo di sviluppo del machine learning. Attraverso il testing continuo e la valutazione delle prestazioni, è possibile ottimizzare i parametri della rete, migliorare l'architettura e raffinare le capacità di previsione. Questo ciclo iterativo è fondamentale per mantenere le DNN al passo con le esigenze in evoluzione dell'applicazione e dell'ambiente in cui operano.

3.2 L'automazione del testing

Data l'importanza che il testing riveste all'interno del ciclo di vita del software ed il suo costo, è necessario automatizzare questo processo. Il testing automatico consiste nell'utilizzo di strumenti e script per eseguire test in modo sistematico e ripetibile. Questa metodologia è fondamentale per valutare e ve-

rificare il corretto funzionamento di un'applicazione software o sistema, identificando potenziali difetti, errori e comportamenti indesiderati. Il testing automatico offre una serie di vantaggi significativi.

Innanzitutto, migliora l'efficienza del processo di testing. L'automatizzazione consente di eseguire una vasta gamma di test in modo rapido e accurato, risparmiando tempo e risorse rispetto al testing manuale. Ciò è particolarmente importante in ambienti di sviluppo agili e DevOps, in cui i rilasci software sono frequenti e richiedono test continui.

In secondo luogo, il testing automatico garantisce la ripetibilità dei test. Una volta creati i casi di test automatizzati, è possibile eseguirli in qualsiasi momento, garantendo che il software continui a funzionare in modo corretto anche dopo le modifiche o gli aggiornamenti. Questo contribuisce a prevenire regressioni, cioè la comparsa di nuovi bug a seguito delle modifiche apportate al codice.

Il testing automatico è essenziale per il controllo della qualità del software. Gli strumenti di testing possono eseguire test complessi su numerosi scenari, consentendo di identificare problemi che potrebbero essere sfuggiti a un tester umano. Ciò riduce il rischio di errori critici che potrebbero compromettere l'esperienza dell'utente o la sicurezza del sistema.

Un altro aspetto rilevante è la scalabilità. Con la crescita di progetti software complessi, il numero di test necessari può aumentare in modo significativo. Il testing automatico consente di gestire facilmente questa complessità, consentendo l'esecuzione simultanea di numerosi test su diverse configurazioni e piattaforme. Questo è particolarmente importante in scenari in cui è richiesto il supporto multipiattaforma o multi-dispositivo.

Un aspetto altrettanto importante è la possibilità di testare in modo continuo. Il testing automatico può essere integrato in processi di integrazione continua (CI) e distribuzione continua (CD), consentendo di eseguire test ogni volta che vengono apportate modifiche al codice sorgente. Questa pratica aiuta a identificare e risolvere problemi in modo tempestivo, riducendo il tempo tra la scoperta di un difetto e la sua correzione.

Infine, il testing automatico può essere impiegato in diversi contesti, inclusi il testing di unità, il testing di integrazione, il testing funzionale, il testing di accettazione, il testing delle prestazioni e il testing di sicurezza. Questa versatilità lo rende una risorsa preziosa per soddisfare diverse esigenze di testing nel ciclo di sviluppo del software.

L'introduzione del testing automatico ha contribuito a migliorare l'efficienza, la qualità e la tempestività del processo di sviluppo del software. Con la sua capacità di scalabilità, integrazione continua e diversificazione delle applicazioni, il testing automatico rappresenta un pilastro imprescindibile nell'industria del software moderno. La sua adozione consente di ridurre il rischio di difetti, garantire la conformità agli standard di settore e fornire prodotti software affidabili e di alta qualità.

3.3 TIGs: Raw Input Manipulation e Generative Deep Learning Based

Il testing delle reti neurali richiede un'enorme quantità di dati, per superare questo difficile ostacolo. A tal fine sono stati proposti numerosi Test Input Generators [7] per la realizzazione di input artificiali per la valutazione dei sistemi basati su Deep Learning. Input che vadano di là dei dataset con cui sono stati allenati in modo da valutarne il comportamento nel modo più completo possibile, anche perchè non

sempre il dataset iniziale è sufficiente a rappresentare tutte le possibili situazioni che possono palesarsi nel mondo reale.

La maggior parte dei Test Input Generators sono sviluppati per la classificazione di immagini, adottando diverse strategie per la generazione degli input. Tra queste possiamo ritrovare la perturbazione di pixel di immagini presenti nel dataset, la modifica della rappresentazione di un modello specifico del dominio oppure la manipolazione della rappresentazione degli input attraverso l'uso di modelli Deep Learning generativi.

In particolare, ci soffermiamo sugli input nei quali la previsione del DL system non corrisponde all'etichetta dell'immagine valutata. Per definire l'etichetta in un input generato queste tecniche consistono nell'applicare piccole perturbazioni agli input di cui l'etichetta è nota assumendo che le modifiche effettuate non alterino l'etichetta stessa.

Per far emergere comportamenti anomali alcuni Test Input Generators (TIGs) si adattano al modello da valutare andando ad ottimizzarne la copertura su tutti i nodi, anche quelli difficilmente raggiungibili. L'idea che sta dietro a questa tecnica è quella di guadagnare maggiore sicurezza sul comportamento del sistema di DL quando tutte le partizioni dell'input di partenza, indotte da uno specifico criterio di copertura impostato a priori sono valutati senza comportamenti anomali oppure se si verificano possono comunque essere utilizzati per migliorare il sistema su quel tipo di input.

Questo tipo di TIGs possono essere implementati sia avendo accesso all'implementazione interna del modello (i.e., livello white-box) e quindi potendo ricavare in modo diretto le informazioni necessarie come i valori di attivazione sei singoli neuroni, che non avendo accesso (i.e., livello black-box) tramite i dati sull'input passatogli e sull'output restituito.

3.3.1 Raw Input Manipulation

Il primo approccio si chiama Raw Input Manipulation (RIM), il quale modifica l'immagine iniziale al livello dei singoli pixel variandone i parametri e quindi creando di fatto una nuova immagine perturbata sulla base di quella precedente.

Nella maggioranza dei casi si utilizzano immagini di partenza di cui si conosce già l'etichetta e che il modello è in grado di elaborare correttamente; successivamente si passa alla modifica della stessa immagine cercando di rendere i ritocchi il più impercettibili possibile, esponendo nello stesso tempo il sistema ad errori di valutazione, ossia ottenendo una predizione diversa dall'etichetta dell'input di partenza.

Le tecniche basate su RIM (Random Input Manipulation) principalmente generano input che sono potenzialmente contraddittori. Ad esempio, simulano perturbazioni che potrebbero essere create da individui malintenzionati con l'obiettivo di indurre il malfunzionamento del sistema attraverso modifiche che sono impercettibili all'occhio umano nelle immagini di input fornite.

Nel mondo reale è difficile osservare questo tipo di distorsioni se non per malfunzionamento degli apparecchi per la cattura delle immagini, questo tipo di distorsioni sono pertanto non sono utili per testare il sistema su normali comportamenti e situazioni che si possono verificare adoperando in modo lecito il modello quanto più che altro per test di sicurezza dei modelli.

Siccome queste tecniche possono modificare solo input già esistenti, le loro performance dipendono dalla qualità delle immagini a monte; non è oltretutto possibile esplorare tutto lo spazio originale degli

input. Applicando solo piccole perturbazioni a immagine valide, queste tecniche rischiano di essere limitate dalle immagini di partenza per quanto riguarda le regioni dello spezio che possono esplorare, lasciando molte altre regioni di input inesplorate e non testabili.

3.3.2 Generative DL Based

Il concetto chiave nei modelli generativi è che se si è in grado di modellare una distribuzione per calcolare le probabilità di determinati input, è possibile anche generare input plausibili campionando da quella distribuzione. Nel corso degli anni è diventato sempre più predominante lo sviluppo di potenti modelli generativi che approssimano distribuzioni, sia in modo esplicito che implicito. Le Reti Generative Avversarie (GANs) utilizzano due reti neurali, una generatrice e una discriminante, dove il compito della generatrice è quello di far corrispondere la distribuzione degli input alla distribuzione desiderata, mentre il discriminante cerca di individuare le discrepanze tra la distribuzione approssimata dal generatore e la distribuzione reale. Il processo di addestramento guida implicitamente la rete generatrice a somigliare sempre di più alla distribuzione reale.

Le PixelCNNs, d'altra parte, si concentrano sulla modellazione delle dipendenze tra i pixel vicini in un'immagine, generando ciascun pixel in modo condizionato ai pixel circostanti. Le PixelCNNs vengono addestrate in modo esplicito per assegnare alta probabilità alle immagini fornite, considerando le interazioni tra i pixel.

Gli Autoencoder Variazionali (VAEs) non effettuano una valutazione diretta delle probabilità associate a un'immagine. Al contrario, concentrano la loro attenzione sull'ottimizzazione dell'Evidence Lower Bound (ELBO), il quale agisce come stimatore inferiore delle probabilità associate all'immagine. L'obiettivo principale di questo approccio è modellare la struttura latente delle immagini, facilitando la generazione di nuove immagini mediante campionamenti da questa specifica struttura latente.

A differenza delle tecniche RIM i generative DL models operano in uno spazio degli input significamente ridotto e producono input realistici che possono essere abbastanza distanti dalle immagini presenti nel dataset originale, questo permette di poter testare il sistema in un più ampio spettro di situazioni differenti rendendole le tecniche migliori per il testing dei modelli in situazioni di normale utilizzo nel mondo reale.

3.4 Competizioni e benchmark tra generatori di test

Lo sviluppo di nuove tecnologie ed approcci nell'ambito del testing automatizzato è fondamentale per il progredire dei sistemi informatici sempre più complessi a cui oggi siamo abituati.

Per questo motivo è sorta la necessità di creare degli ambienti in cui racchiudere e dare visibilità alle principali scoperte nell'ambito del testing ed allo stesso tempo creare uno spazio che possa essere fertile per la generazione e maturazione di nuove idee che potrebbero rivoluzionare ancora una volta questo vastissimo mondo.

Tra le competizioni più significativi nel campo della generazione di test software una delle più rilevanti è la Search-Based and Fuzz Testing (SBFT) [3], workshop internazionale che si tiene ogni anno e che è ormai arrivato alla sedicesima edizione, co-locata con la più importante conferenza sull'Ingegneria del Software: ICSE.

Figura 3.1: SBFT workshop.

Nel campo dello sviluppo del software, le tecniche di ottimizzazione sono applicabili a vari aspetti del processo, costituendo un ramo di ricerca noto come Search-Based Software Engineering (SBSE). Nelle edizioni precedenti del workshop, l'attenzione era rivolta all'applicazione di SBSE per compiti di testing, noti come Search-Based Software Testing (SBST). Le attuali ricerche in SBST e Fuzz Testing stanno proponendo tecniche simili per affrontare questioni di testing con obiettivi paralleli. Questa convergenza ha portato alla decisione di rinominare il workshop come Search-Based and Fuzz Testing (SBFT).

Le strategie di SBFT sono state adottate per una vasta gamma di obiettivi di testing, tra cui il conseguimento di elevata copertura, la rilevazione di difetti e vulnerabilità, e la verifica di varie proprietà legate allo stato del software e non solo, come ad esempio la scalabilità e l'accettabilità. L'obiettivo principale di questo workshop è quello di riunire ricercatori e professionisti del settore, inclusi esperti di SBST e Fuzzing, nonché membri più ampi della comunità di Ingegneria del Software, al fine di condividere esperienze e delineare prospettive per la futura ricerca nell'ambito dell'automazione del testing software. Un secondo obiettivo del workshop è promuovere l'applicazione delle tecniche di ricerca e fuzzing per integrare il testing con altre aree dell'Ingegneria del Software.

Il workshop SBFT, articolato in una giornata, comprende una sezione dedicata alla ricerca, keynote e competizioni tra gli strumenti di testing più diffusi. Inoltre, il workshop fornisce un'opportunità unica per riunire esperti del settore in un dibattito in formato panel. L'insieme di queste attività contribuirà a stabilire nuovi fondamenti nella ricerca di SBFT.

Nel contesto di questo workshop, l'area che ha ispirato profondamente il mio progetto è la Competizione degli Strumenti SBFT, che mira a testare vari strumenti di testing su una gamma diversificata di sistemi e domini, compresi quelli tradizionali ed emergenti. Questa competizione pone l'accento su un obiettivo primario: la creazione di test attraverso l'impiego di tecniche search-based o del fuzzing, una tecnica automatizzata per la verifica del software. Tali tecniche prevedono la generazione di input non validi, imprevisti o casuali. Successivamente, il software esercitato con questi input viene attentamente monitorato per individuare eventuali errori o vulnerabilità, come arresti anomali, affermazioni non valide all'interno del codice o possibili perdite di memoria. Le diverse proposte di testing, sviluppate dai candidati, vengono attentamente valutate e i risultati ottenuti vengono confrontati tra loro per determinare quale abbia mostrato le migliori performance durante tutto l'arco della competizione. Per poter garantire un confronto imparziale e facilitare la partecipazione, devono essere sviluppati opportuni framework e utilizzate metriche significative.

Nonostante la presenza di numerose competizioni all'interno di questo workshop, non ve ne sono sul testing di classificatori di immagini, malgrado l'abbondanza di letteratura pertinente.

II

Sviluppo

4

Il framework OlymTIGs

Il framework sviluppato è stato concepito con l'obiettivo principale di fornire un'interfaccia per la generazione automatizzata di input avversari per i classificatori. Le caratteristiche salienti di questa interfaccia comprendono la capacità di integrare sia classificatori preaddestrati che modelli da addestrare, offrendo così una flessibilità notevole nella gestione dei test. I classificatori inseriti nell'ambiente vengono successivamente sottoposti a una serie di test utilizzando due Generatori di Input Avversari, i quali hanno il compito di generare input che mettano alla prova le capacità del classificatore e rivelino le sue debolezze.

Il passaggio finale del processo è la fase di validazione, un momento critico in cui le immagini generate durante i test sono sottoposte a una rigorosa valutazione automatica da parte di SelfOracle. Questo sistema automatizzato è progettato per determinare se gli input generati soddisfino i criteri di validità definiti, assicurando che i risultati dei test siano attendibili e rappresentativi della capacità del classificatore sotto esame. In questo modo, il framework offre un approccio completo e affidabile per la creazione di immagini avverse dei classificatori sotto esame, consentendo una diagnosi accurata delle loro prestazioni. Per mostrare la flessibilità di OlymTIGs, in esso sono stati integrati due generatori appartenenti a tipologie diverse (i.e., RIM e generative DL based, rispettivamente) e sviluppati con framework differenti (i.e., Tensorflow e Torch).

Il codice sorgente del framework è disponibile nella repository GitHub OlymTIGs al seguente link: https://github.com/mallocchio/OlymTIGs.

4.1 Casi d'uso

Per analizzare in dettaglio il framework, è essenziale iniziare con l'identificazione dell'idea alla base del suo sviluppo. Inizialmente, il processo ha previsto un'analisi approfondita mediante la creazione di un diagramma dei casi d'uso, il quale ha fornito una rappresentazione chiara delle funzionalità che il software doveva essere in grado di offrire e di come gli utenti finali avrebbero interagito con esso.

La *Figura 4.1*, che rappresenta il diagramma dei casi d'uso, è stata la pietra angolare di questo progetto, delineando le funzioni principali del framework.

In primo luogo, si è ritenuto fondamentale consentire all'utente di configurare l'interfaccia iniziale. Questa fase di configurazione permette all'utente di personalizzare l'ambiente secondo le proprie esi-

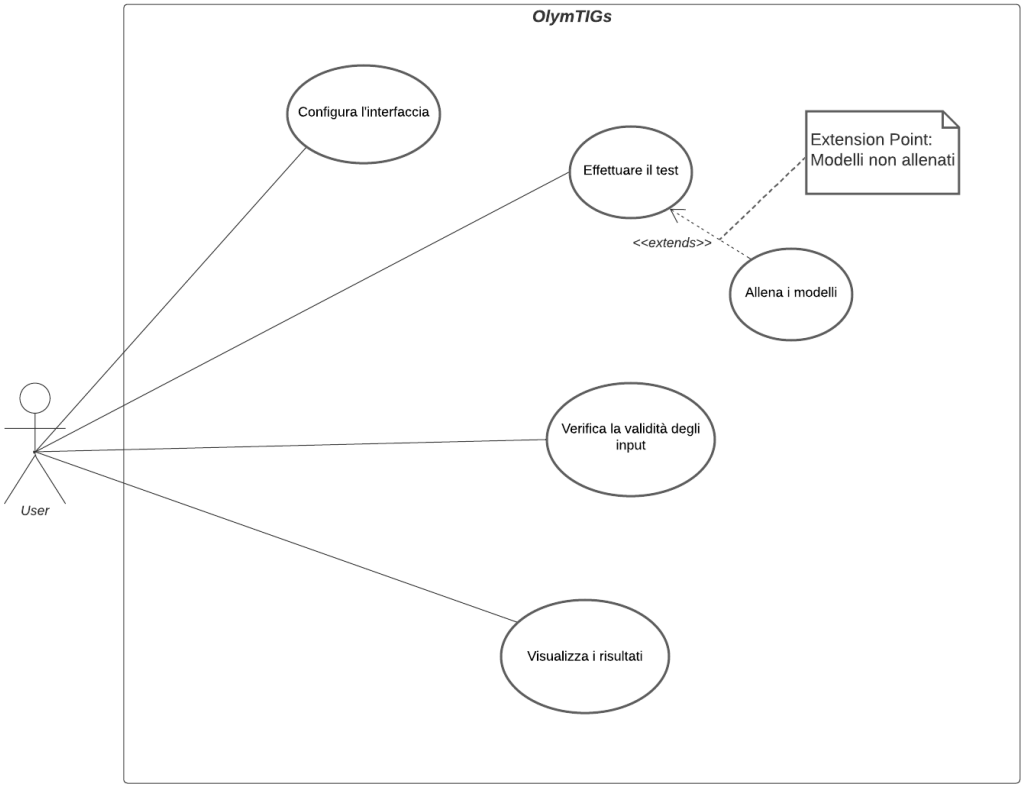

Figura 4.1: Diagramma dei casi d'uso.

genze, fornendo parametri come il modello di classificazione da utilizzare, il dataset di riferimento e la dimensione delle immagini di input.

Successivamente, un altro caso d'uso di estrema importanza è emerso, ovvero la possibilità per l'utente di eseguire test sui propri classificatori. Per consentire questo processo, è necessario che l'utente possa scaricare il dataset di riferimento dal quale verranno estratte le immagini di input, caricare i modelli di classificazione definiti durante la configurazione, eventualmente allenare nuovi modelli e infine condurre i test sui classificatori. È inoltre necessario implementare un meccanismo di salvataggio dei dati di output, in modo da consentire il recupero e il confronto dei risultati in futuro.

Un altro aspetto cruciale dell'interfaccia è la capacità di validare gli input generati durante i test. Questo passo è cruciale per determinare il successo o l'insuccesso del test e per consentire comparazioni tra test diversi.

Una funzione altamente significativa è la generazione di resoconti dettagliati dei test e delle validazioni. Questi resoconti devono includere informazioni chiave come il tempo di esecuzione dei test, le immagini generate, le immagini valide e quelle non valide. Tale documentazione dettagliata è fondamentale per una valutazione completa delle prestazioni del framework e per consentire un'analisi accurata dei risultati ottenuti.

4.1.1 User stories

Per la ricerca dei casi d'uso è stato necessario formulare le user storys. Di seguito, vengono presentate una serie di storie utente realizzate per descrivere le principali capacità del framework in fase di sviluppo.

- Come utente, desidero configurare l'interfaccia iniziale del sistema. Voglio personalizzare l'ambiente secondo le mie esigenze, inserendo parametri come il modello di classificazione da utilizzare, il dataset di riferimento e la dimensione delle immagini di input.

- Come utente, voglio eseguire test sui miei classificatori. Devo essere in grado di scaricare il dataset di riferimento per le immagini di input, caricare i modelli di classificazione configurati in precedenza, allenare nuovi modelli se necessario e condurre test sui classificatori. Inoltre, voglio poter salvare i dati di output per recuperare e confrontare i risultati in futuro.

- Come utente, devo validare gli input generati durante i test. Questo è un passaggio critico per determinare il successo o l'insuccesso del test e per consentire comparazioni tra test diversi.

- Come utente, desidero generare resoconti dettagliati dei test e delle validazioni. Questi resoconti devono includere informazioni cruciali come il tempo di esecuzione dei test, le immagini generate, le immagini valide e quelle non valide. La documentazione dettagliata è essenziale per valutare completamente le prestazioni del framework e analizzare accuratamente i risultati ottenuti.

4.2 Statico: Package Diagram

Passando alla realizzazione vera e propria, l'interfaccia OlymTIGs si compone di quattro componenti principali che insieme compongono la struttura per il testing automatizzato.

4.2.1 Il file di configurazione

Il primo passo nell'utilizzo di questa interfaccia è la configurazione dei parametri, che avviene attraverso il file denominato "Config.txt". Questo file contiene le informazioni necessarie per guidare il processo di test e valutazione dei classificatori. Le informazioni contenute in questo file consentono all'interfaccia di operare in modo automatizzato, configurando il suo comportamento in base alle esigenze dell'utente.

Innanzitutto, all'interno del file di configurazione, è specificata la modalità di utilizzo, che può essere impostata su "Generation" se si intende condurre il test sui classificatori oppure "Validation" se si vuole validare le immagini avversarie prodotte.

In caso di modalità "Generation", si richiedono ulteriori dettagli, come il percorso del classificatore e il tipo del Test Input Generator da utilizzare. In caso di modalità "Validation", si richiede il validatore da utilizzare e la cartella in cui sono contenute le immagini.

Le specifiche delle immagini utilizzate sono un altro aspetto fondamentale del file di configurazione. Queste specifiche includono l'etichetta specifica, il numero di input avversari da generare e le dimensioni delle immagini in uso.

Infine, è presente un parametro che definisce il percorso dei risultati in cui i dati generati durante il test vengono salvati. Questa informazione è cruciale per archiviare in modo organizzato i risultati delle operazioni effettuate.

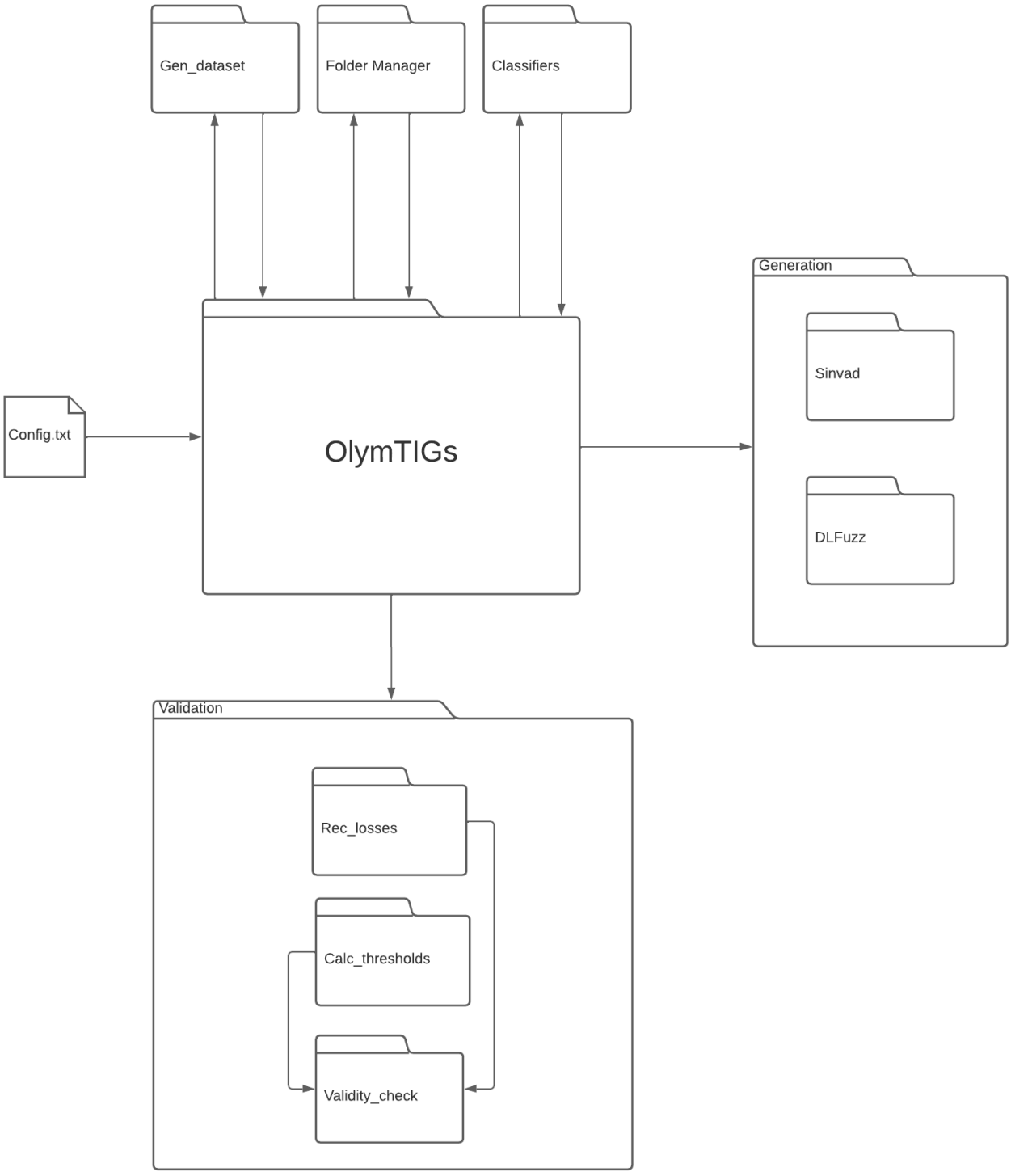

Figura 4.2: Diagramma dei pacchetti.

```
----MODE TO USE: GENERATION, VALIDATION----
mode: validation

----GENERATION----
TIG: sinvad
model_path: ./trained/lenet1.pt

----IMAGE SPEC----
label: 5
imgs_to_sample: 100
img_rows: 28
img_cols: 28

----VALIDATION----
validator: selforacle
images_folder: ./results/sinvad_2023-11-18_20-56-47

results_path: ./results
```

Figura 4.3: File di configurazione.

4.2.2 La generazione del dataset

Il processo di preparazione del dataset è affidata alla classe GenDataset. Essa si occupa di scaricare e preparare il dataset che sarà utilizzato nelle fasi successive del testing. Il file di configurazione fornisce le informazioni necessarie riguardo al dataset da scaricare e, eventualmente, le etichette specifiche che dovranno essere considerate.

La classe è in grado di gestire dataset con diverse caratteristiche, in base al Test Input Generator (TIG) specificato. Ad esempio, il TIG SINVAD richiede che il dataset sia conforme alla libreria Torch [6], mentre il TIG DLFuzz richiede un formato diverso compatibile con TensorFlow [1].

La specificazione di un'etichetta specifica consente di personalizzare il dataset in modo che contenga solo immagini appartenenti a una classe specifica, il che è fondamentale per condurre test accurati nelle fasi successive.

La classe opera seguendo i seguenti passi:

- Scarico del Dataset: A seconda del TIG specificato, la classe scarica il dataset appropriato. Ad esempio, per il TIG SINVAD, viene utilizzato il dataset MNIST da Torchvision, mentre per il TIG DLFuzz, viene utilizzato il dataset MNIST di TensorFlow.

- Selezione delle Etichette (Label): Se è stata specificata un'etichetta specifica nel file di configurazione, la classe crea un sottoinsieme del dataset contenente solo le immagini associate a quell'etichetta.

- Mescolamento del Dataset: Per garantire una maggiore attendibilità nei test futuri, il dataset viene mescolato in modo che le immagini non siano disposte sempre nello stesso ordine. Ciò evita che il classificatore possa "imparare" dall'ordine delle immagini e garantisce risultati più rappresentativi delle reali prestazioni del classificatore.

4.2.3 La creazione della cartella

La classe "FolderManager" è stata progettata con l'obiettivo di gestire la struttura delle cartelle in cui vengono salvati i risultati ottenuti durante l'esecuzione dei Test Input Generators (TIGs) o durante il processo di validazione nel contesto dell'interfaccia. In particolare, questa classe si occupa di creare una cartella dedicata per ogni esecuzione dell'interfaccia.

La creazione della cartella segue una convenzione specifica. Essa viene collocata all'interno del percorso dei risultati, il quale è definito nel file di configurazione. Il nome della cartella è composto da due parti distintive:

- Nome del TIG: Il nome del TIG utilizzato per il test è utilizzato come parte del nome della cartella. Questo elemento permette di identificare rapidamente quale TIG è stato utilizzato per l'esecuzione, rendendo la struttura delle cartelle più chiara e agevolando la consultazione dei risultati.
- Time-Stamp: Il nome della cartella include un time-stamp generato automaticamente. Questo time-stamp è composto da informazioni temporali, come il giorno dell'anno e l'ora in cui è stata effettuata l'esecuzione. Questo approccio garantisce che ogni cartella abbia un nome unico, evitando ambiguità nel sistema e consentendo la distinzione tra diverse esecuzioni nel tempo.

"FolderManager" contribuisce a organizzare in modo efficace i risultati dell'interfaccia, creando una struttura gerarchica di cartelle in cui ogni cartella è chiaramente identificata dal nome del TIG e dal time-stamp dell'esecuzione. Questa pratica facilita notevolmente la gestione e la consultazione dei risultati ottenuti durante il processo di testing e validazione.

4.2.4 Addestramento o caricamento dei classificatori

La classe "Classifiers" costituisce la componente responsabile dell'addestramento o del caricamento dei modelli di classificazione utilizzati dall'interfaccia. Questa classe permette di definire i modelli da utilizzare durante le operazioni di testing, consentendo l'inclusione di qualsiasi modello sviluppato.

All'interno di questa classe sono presenti le architetture dei modelli, sia in versione Torch che Tensorflow, che poi verranno utilizzati per la fase di generazione. Nel caso specifico sono presenti tre modelli derivati dalla letteratura: LeNet1, LeNet4 e LeNet5, ma è importante notare che è possibile inserire modelli personalizzati.

Oltre alle architetture la classe "Classifiers" è composta principalmente da due funzioni principali: "load" e "train".

- Funzione "load": Questa funzione istanzia il modello specificato nel file di configurazione e carica i pesi associati a tale modello. I pesi sono forniti attraverso un percorso specificato nel file di configurazione, permettendo così di evitare di addestrare il modello ad ogni utilizzo successivo. Il modello istanziato, con i pesi caricati, è quindi restituito all'interfaccia principale, pronto per essere utilizzato nelle operazioni di valutazione.
- Funzione "train": Questa funzione è utilizzata quando non sono disponibili pesi preaddestrati per il modello specificato. In questo caso, il modello indicato viene addestrato utilizzando il dataset designato per le operazioni di testing. Il modello è creato utilizzando la libreria TensorFlow e i pesi

ottenuti durante l'addestramento vengono successivamente salvati in una cartella chiamata "trained". Inoltre, i pesi vengono convertiti anche in formato Torch, in modo che lo stesso classificatore abbia sia una versione sviluppata con Torch che Tensorflow. Questa conversione è necessaria per supportare l'utilizzo dello stesso classificatore di entrambi i TIGs, SINVAD e DLFuzz, in quanto SINVAD è stato creato utilizzando l'infrastruttura Torch e DLFuzz utilizzando TensorFlow. In questo modo i dati restituiti sono i più fedeli ed attendibili possibile.

4.2.5 Generazione di immagini avversarie

All'interno dell'interfaccia OlymTIGs sono stati incorporati due generatori di Test Input della letteratura: SINVAD e DLFuzz.

Una volta effettuato il download del dataset, creato la directory di esecuzione e caricato il modello da testare, l'interfaccia OlymTIGs comunica tali dati agli algoritmi di generazione di Test Input. Questi algoritmi producono immagini avversarie in base al classificatore selezionato.

Per quanto concerne SINVAD, come precedentemente menzionato, fa uso di modelli di deep learning denominati VAE (Variational Autoencoder). Pertanto, prima di avviare l'algoritmo, l'interfaccia carica il modello VAE richiesto e lo passa insieme alle altre informazioni all'algoritmo. DLFuzz, al contrario, non richiede questo passaggio poiché non dipende da ulteriori modelli.

La scelta del generatore di Test Input da utilizzare e il numero di immagini da generare sono specificati all'interno del file di configurazione.

Al termine dell'esecuzione dell'algoritmo, gli input generati vengono archiviati nella directory di esecuzione in formato NumPy per agevolarne la consultazione. Inoltre, viene generato un file denominato "Summary.txt" che raccoglie le informazioni principali sull'esecuzione del processo.

4.2.6 Validazione

Nel caso in cui siano disponibili immagini avversarie da valutare, l'interfaccia offre la possibilità di determinare se tali immagini siano valide o meno. Per valide si intendono immagini riconoscibili da esperti umani nel dominio dell'input, ovvero immagini alle quali un essere umano può assegnare con sicurezza un'etichetta corrispondente al dominio dell'input.

Poiché l'interfaccia è automatizzata, invece di utilizzare esseri umani, fa uso di un validatore denominato "SelfOracle". Questo validatore, in particolare, sfrutta un Autoencoder Variational (VAE) per valutare la capacità di ricostruzione delle immagini, misurando l'errore di ricostruzione. Se l'errore supera una soglia predefinita, l'immagine viene classificata come non valida.

L'interfaccia OlymTIGs procede nel seguente modo: carica il modello VAE per la valutazione, quindi utilizzando il dataset calcola valori di soglia in base all'errore medio nelle immagini standard del dataset. Successivamente, esegue l'algoritmo di valutazione effettivo.

I risultati vengono salvati nella stessa cartella delle immagini generate in forma di file con estensione CSV corrispondenti ai diversi valori di soglia utilizzati. L'algoritmo esegue molte valutazioni sulle immagini di test, ciascuna utilizzando una soglia diversa.

Infine, il file “Summary.txt” viene esteso se già presente nella parte generativa o creato se non esiste. Questo file contiene informazioni sulla soglia utilizzata e il conteggio delle immagini considerate valide e non valide dall’algoritmo di valutazione.

4.3 Comportamento dinamico: sequence diagram

Il diagramma di sequenza rappresentato nella figura *Figura 4.4* descrive il flusso di interazioni tra i partecipanti durante l’esecuzione dell’interfaccia OlymTIGs, che gestisce le operazioni di generazione, addestramento e validazione nell’ambito di un sistema. I partecipanti coinvolti sono i seguenti:

- “User”: un attore esterno che utilizza l’interfaccia OlymTIGs.
- “OlymTIGs”: l’interfaccia stessa, responsabile delle interazioni tra le diverse classi e modalità operative.
- “GenDataset”: una classe che prepara il dataset.
- “FolderManager”: una classe responsabile della creazione di cartelle per i risultati delle esecuzioni.
- “Classifiers”: una classe contenente architetture di modelli e funzioni per l’allenamento e il caricamento dei modelli.
- “Generation”: una classe che raccoglie i dati necessari ed esegue la procedura di test designata
- “SINVAD”: una classe contenente l’algoritmo di generazione degli input SINVAD.
- “DLFuzz”: una classe contenente l’algoritmo di generazione degli input DLFuzz.
- “Validation”: una classe che raccoglie i dati necessari ed esegue la procedura di validazione designata
- “SelfOracle”: una classe contenente l’algoritmo di validazione SelfOracle.

Il flusso di esecuzione inizia con l’utente che compila il file di configurazione “Config.txt” con le informazioni necessarie per il sistema. Successivamente, OlymTIGs raccoglie il file di configurazione ed estrae le informazioni pertinenti per guidare le operazioni.

OlymTIGs offre due modalità di esecuzione: “Generation” e “Validation”.

Nella modalità ”Generation”, l’interfaccia avvia il processo seguendo questa sequenza:

1. OlymTIGs chiama “GenDataset” per generare il dataset, specificando il tipo di Test Input Generator (TIG) da utilizzare (Tensorflow o Torch).
2. “GenDataset” scarica il dataset appropriato e lo restituisce a OlymTIGs.
3. OlymTIGs chiama “FolderManager” per creare una cartella di esecuzione, identificata dal nome del TIG selezionato e un timestamp, dove verranno raccolti i risultati.
4. “FolderManager” crea la cartella e la restituisce a OlymTIGs.

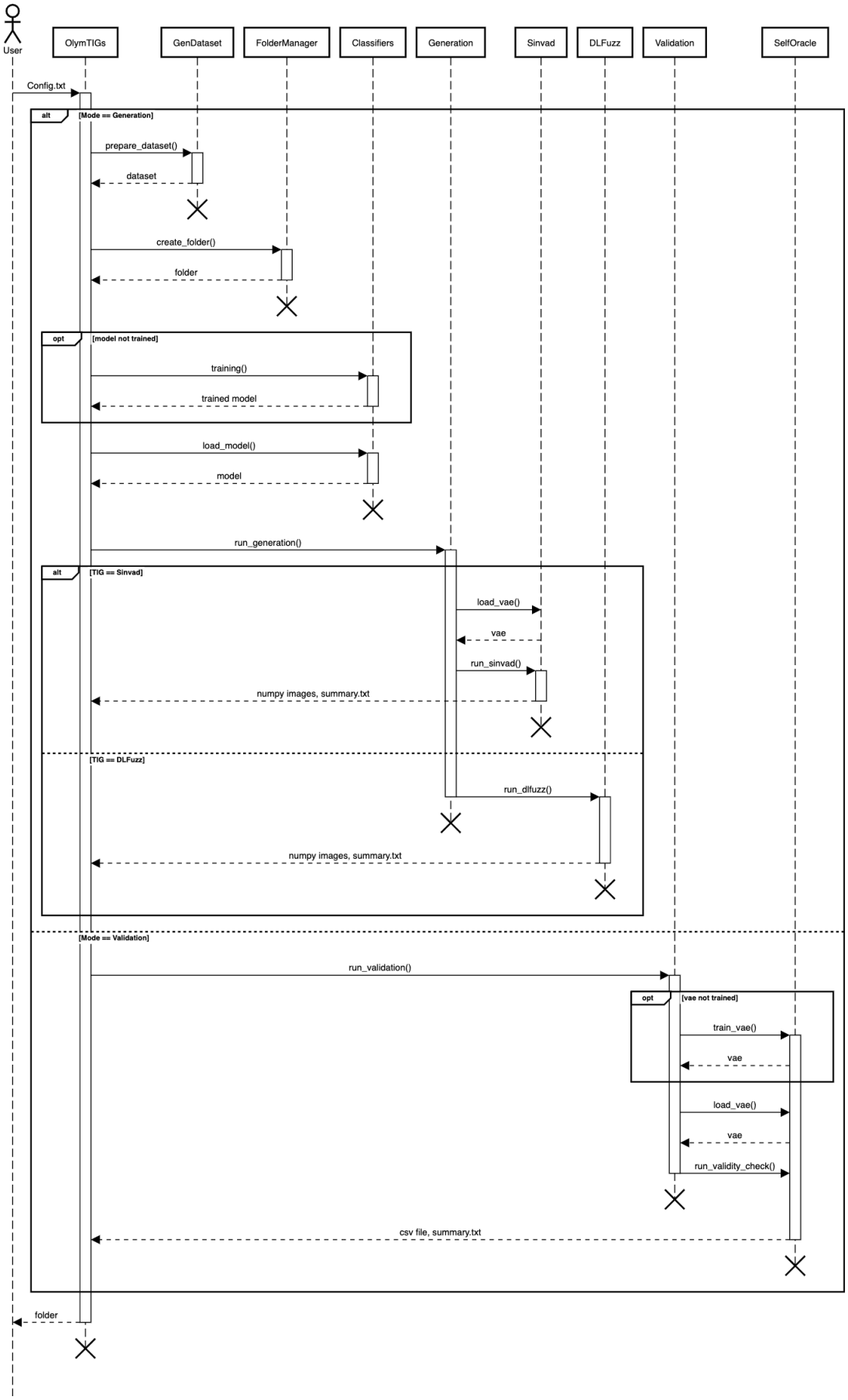

Figura 4.4: Diagramma di sequenza.

5. OlymTIGs chiama “Classifiers” per inizializzare il modello da testare e carica i pesi preaddestrati, ottenendo il modello pronto. Se non sono presenti i pesi, il modello viene addestrato automaticamente e i suoi pesi salvati.

6. OlymTIGs chiama Generation che si occuperà di raccogliere tutti i dati necessari ed avviare il test.

7. In base al TIG selezionato, vengono eseguite procedure diverse:

   - Per “SINVAD”, Generation prima chiama SINVAD per caricare il modello Variational Autoencoder (VAE) e successivamente procede con l’esecuzione dell’algoritmo. I risultati vengono salvati nella cartella di esecuzione creata da “FolderManager”.
   - Per “DLFuzz”, Generation chiama direttamente l’algoritmo DLFuzz e i risultati vengono salvati nella cartella creata da “FolderManager”.

8. Dopo l’esecuzione delle procedure del TIG selezionato, sia SINVAD che DLFuzz terminano la loro esecuzione.

Nella modalità “Validation”, OlymTIGs chiama la classe Validation che si occupa di raccogliere i dati necessari, come caricare ed eventualmente allenare il modello VAE, per poi avviare il processo di validazione utilizzando l’algoritmo indicato. Il processo di validazione si sviluppa secondo il seguente flusso:

1. L’interfaccia “OlymTIGs” chiama “Validation”

2. “Validation” sceglie l’algoritmo di validazione corretto

3. Nel caso non sia già presente il modello VAE addestrato (necessario per SelfOracle) viene effettuato l’allenamento e il salvataggio dei pesi

4. Viene caricato il modello VAE

5. Validation chiama l’esecuzione dell’algoritmo di valutazione

Per rendere più semplice da comprendere il diagramma di sequenza non è stato riportato il funzionamento interno di SelfOracle. Nello specifico la classe esegue questi passaggi:

1. “SelfOracle” e richiama la funzione “rec_losses()” per calcolare la percentuale di errore nella ricostruzione delle immagini originali nel dataset.

2. Viene chiamata la funzione “calc_thresholds” che, in base agli errori di ricostruzione, calcola diversi threshold per ampliare la gamma di risultati ottenibili dal validatore.

3. Viene eseguito l’algoritmo

4. I risultati vengono salvati in file CSV e un file di testo riassuntivo nella stessa cartella che contiene gli input in formato NumPy.

5. “SelfOracle” termina la sua esecuzione

Al completamento di ogniuna delle modalità di utilizzo OlymTIGs restituisce la cartella con i risultati all'utente prima di terminare anch'essa la sua esecuzione.

5

Valutazione sperimentale

5.1 Dataset: MNIST

Il dataset MNIST (Modified National Institute of Standards and Technology) è uno dei dataset più iconici e ampiamente utilizzati nel campo dell'apprendimento automatico e del riconoscimento di pattern. Questo dataset rappresenta una collezione di immagini di cifre scritte a mano, raccolte per scopi di ricerca e sviluppo.

Il dataset MNIST [2] è stato creato dagli scienziati del National Institute of Standards and Technology (NIST) degli Stati Uniti. Questo dataset è stato modificato e reso pubblico per promuovere la ricerca nell'ambito del riconoscimento di cifre scritte a mano. È stato ampiamente utilizzato dalla comunità scientifica e da ricercatori di tutto il mondo per sviluppare e valutare algoritmi di riconoscimento di pattern, in particolare quelli basati su reti neurali artificiali.

Le immagini nel dataset MNIST sono in scala di grigi, il che significa che ciascuna immagine è composta da 28x28 pixel, ciascuno con un valore di intensità luminosa compreso tra 0 (nero) e 255 (bianco). Queste immagini sono state preprocessate e normalizzate in modo che siano tutte della stessa dimensione e centralizzate, semplificando così l'elaborazione e l'analisi. Le immagini sono tutte caratterizzate da uno sfondo nero e dalla cifra di colore bianco in modo che sia facilitato il riconoscimento dei pattern e la distinzione tra cifra e sfondo.

Il dataset MNIST è composto da due insiemi principali:

- Set di Addestramento: Questo set contiene 60.000 immagini di cifre scritte a mano. Ogni immagine è rappresentata in scala di grigi e ha una dimensione di 28x28 pixel. Le immagini sono etichettate con la cifra corrispondente (da 0 a 9) che rappresentano.

- Set di Test: Questo set è costituito da 10.000 immagini simili a quelle del set di addestramento, anch'esse etichettate con le cifre corrispondenti.

Nonostante la sua apparente semplicità, il dataset MNIST presenta alcune sfide, come la classificazione accurata delle cifre scritte in stili diversi e l'identificazione di cifre illeggibili o distorte, è pertanto adatto alla valutazione dei classificatori.

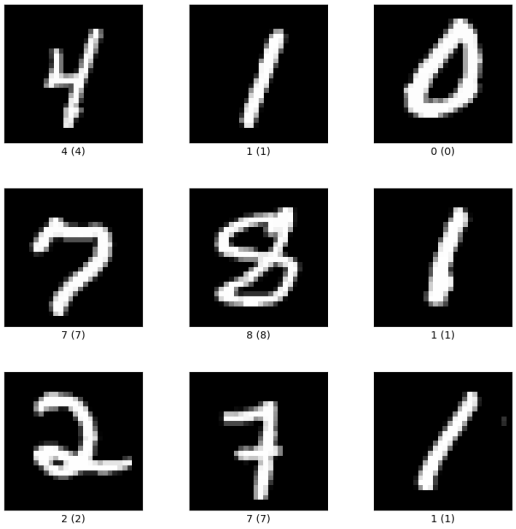

Figura 5.1: Dataset MNIST.

5.2 TIGs: DLFuzz e SINVAD

5.2.1 DLFuzz

DLFuzz [4] è un test input generator della famiglia dei Raw Input Manipulation *(sottosezione 3.3.1)*, ovvero la categoria che modifica direttamente l'immagine di partenza applicando perturbazioni e filtri per generare un nuovo input che possa essere di difficile comprensione per il modello a cui è fatto visionare.

DLFuzz adotta un approccio iterativo volto a selezionare con attenzione i neuroni ritenuti essenziali per l'attivazione al fine di migliorare la copertura logica di una rete neurale. Tale miglioramento viene ottenuto mediante l'applicazione di lievi perturbazioni agli input di test, finalizzate a guidare la rete neurale nell'evidenziare comportamenti errati. Durante il processo di mutazione, DLFuzz conserva gli input mutati dalla precedente iterazione che contribuiscono in maniera più significativa all'incremento specifico della copertura neurale per il successivo fuzzing.

Queste perturbazioni vengono attentamente create in modo da risultare impercettibili e garantire che i risultati delle previsioni rimangano invariati tra l'input originale e gli input mutati. Questa metodologia consente a DLFuzz di rilevare automaticamente input rari e individuare comportamenti errati attraverso il testing differenziale. Gli errori emergono quando il risultato delle previsioni relative agli input mutati si discosta dai risultati dell'input originale.

Architettura

L'input di test, identificato come i, è costituito da un'immagine attinta dal dataset MNIST da sottoporre a un processo di classificazione. La rete neurale convoluzionale (CNN) selezionata per l'analisi, nel caso della competizione, LeNet1, accoglie tale input. Il procedimento di mutazione applica lievi perturbazioni all'immagine i, al fine di generare i', il quale risulta essere visivamente indistinguibile dall'originale i. Nel

caso in cui la CNN classifichi i' e i in classi diverse, tale esito è considerato un comportamento erroneo, e i' viene qualificato come uno degli input avversari. La divergenza nelle previsioni di classificazione prima e dopo la mutazione indica la presenza di almeno una previsione errata, rendendo superflua qualsiasi azione manuale di etichettatura. In contrasto, se la CNN predice che entrambi gli input appartengono alla medesima classe, i' viene ulteriormente sottoposto a mutazione mediante l'algoritmo al fine di mettere alla prova la robustezza della CNN.

Il processo di ottimizzazione

Il problema di ottimizzazione in esame riguarda l'approccio basato sui gradienti nell'apprendimento profondo avversario, che risulta essere più efficace sotto vari aspetti, in particolare per quanto concerne l'efficienza temporale. Questo metodo individua perturbazioni ottimizzando l'input al fine di massimizzare l'errore di previsione, una procedura opposta all'ottimizzazione dei pesi della rete neurale per minimizzare l'errore di previsione durante il processo di addestramento. L'implementazione di questo approccio risulta agevole attraverso la personalizzazione della funzione di perdita come obiettivo, seguita dalla massimizzazione della perdita mediante la salita del gradiente.

La funzione di perdita utilizzata in DLFuzz è definita come segue:

$$obj = \sum_{i=0}^{k}(c_i - c) + \lambda \cdot \sum_{i=0}^{m} n_i$$

L'obiettivo dell'ottimizzazione consiste in due parti. La prima parte $\sum_{i=0}^{k}(c_i - c)$ considera l'etichetta di classe originale dell'input, indicata come c, e le prime k etichette di classe con una confidenza leggermente inferiore a c. Massimizzando la prima parte, l'input viene guidato oltre il confine decisionale della classe originale, collocandosi nello spazio decisionale delle prime "k" altre classi. Gli input così modificati risultano più inclini a essere classificati in modo errato. La seconda parte $\sum_{i=0}^{m} n_i$ prevede l'attivazione di specifici neuroni, rappresentati da n_i. La selezione di tali neuroni segue varie strategie al fine di migliorare la copertura neurale. Il parametro iperparametrico λ viene utilizzato per bilanciare i due obiettivi.

Il processo di fuzzing

Quando viene fornito un input di test i, DLFuzz mantiene una lista di semi per conservare gli input intermedi mutati che contribuiscono alla copertura dei neuroni. Inizialmente, questa lista dei semi contiene solo un input, che è esattamente i. Successivamente, DLFuzz esamina ogni seme i e ottiene gli elementi che compongono l'obiettivo di ottimizzazione. Successivamente, DLFuzz calcola la direzione del gradiente per le mutazioni successive. Nel processo di mutazione, DLFuzz applica in modo iterativo il gradiente elaborato come perturbazione a i e ottiene l'input intermedio i'. Dopo ciascuna mutazione, vengono acquisiti l'etichetta di classe intermedia c', le informazioni sulla copertura, la distanza L2 tra i e i'. Se la copertura del neurone è migliorata da i' e la distanza L2 è inferiore a una soglia desiderata, i' viene aggiunto alla lista dei semi. Infine, se c' è già diverso da c, il processo di mutazione per i semi i si interrompe e i' viene incluso nell'insieme di input avversari. Pertanto, DLFuzz è in grado di generare

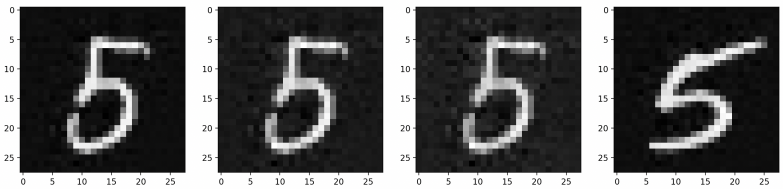

Figura 5.2: Immagini avversarie prodotte da DLFuzz. Il TIG modifica l'immagine nello spazio originale dei pixel per generare un nuovo input, alterando i valori dei pixel.

input avversari multipli per un determinato input originale ed esplorare un nuovo modo per migliorare ulteriormente la copertura dei neuroni.

Per il processo di mutazione iterativa, inizialmente sono disponibili vari metodi di elaborazione per generare perturbazioni quando vengono ottenuti i gradienti, tra cui mantenere solo il segno, imitare situazioni realistiche, ecc. Queste strategie di mutazione per l'input sono facili da estendere a DLFuzz. Inoltre, DLFuzz utilizza la distanza L2 per misurare la perturbazione con la stessa computazione. Ciò garantisce che la distanza tra i e i' sia impercettibile. Per le condizioni di conservazione dei semi nella riga 18, DLFuzz limita la distanza desiderata a un intervallo relativamente piccolo (inferiore a 0,02) per garantire l'impercettibilità. Poiché il miglioramento della copertura del neurone di un input diminuisce nel tempo, la soglia corrispondente per la conservazione dei semi diminuisce anche con il tempo di esecuzione. Inoltre, gli input e gli iperparametri configurati per DLFuzz hanno un certo impatto sulle prestazioni e richiedono sforzi di esplorazione. Inoltre, è possibile aumentare le soglie di conservazione dei semi per conservare più input mutati con una maggiore distanza.

Strategie per la selezione dei neuroni

Per massimizzare la copertura dei neuroni, proponiamo quattro strategie euristiche per la selezione dei neuroni che sono più probabili di migliorare la copertura, come elencate di seguito. Per ciascun seme xs, verranno selezionati m neuroni utilizzando una o più strategie, che possono essere personalizzate all'interno delle iterazioni dell'algoritmo.

1. Selezionare i neuroni coperti frequentemente durante i test passati. Questa strategia è ispirata dall'esperienza pratica nei test tradizionali del software, in cui i frammenti di codice spesso eseguiti o raramente eseguiti sono più suscettibili di introdurre difetti. I neuroni coperti frequentemente o raramente possono portare a una logica insolita e attivare più neuroni.

2. Selezionare i neuroni coperti raramente durante i test passati, seguendo le considerazioni precedentemente menzionate.

3. Selezionare i neuroni con i pesi più elevati. Questa strategia si basa sulla nostra assunzione che i neuroni con i pesi più alti possono avere una maggiore influenza su altri neuroni.

4. Selezionare i neuroni vicino alla soglia di attivazione. È più facile accelerare l'attivazione/disattivazione dei neuroni con valori di output leggermente inferiori/superiori alla soglia.

Queste strategie consentono di scegliere in modo intelligente i neuroni da considerare durante il processo di generazione degli input al fine di massimizzare la copertura dei neuroni.

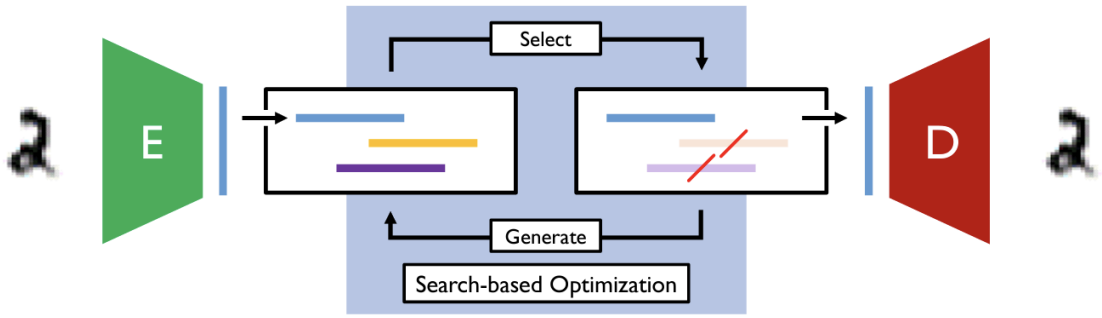

Figura 5.3: Struttura di SINVAD.

5.2.2 SINVAD

SINVAD [5] è una tecnica basata sui Generative DL Models (GDLM). In particolare, utilizza una tipologia di modelli chiamati Variational Autoencoders (VAE). Tali modelli presentano una caratteristica distintiva che li differenzia in modo fondamentale dagli autoencoder standard e che conferisce loro una notevole utilità nella modellazione generativa: i loro spazi latenti sono concepiti in maniera intrinsecamente continua. Questa specifica concezione consente una facile esecuzione di campionamenti casuali e interpolazioni, rendendo il modello particolarmente adatto per tali operazioni.

Struttura

La struttura di SINVAD può essere schematizzata in tre punti principali:

1. Attraverso il decoder, l'immagine di partenza viene codificata in uno spazio latente
2. Vengono aggiunti valori casuali campionati da una distribuzione normale standard a un singolo elemento del vettore latente.
3. Il vettore viene poi riconvertito in immagine attraverso il decoder.

L'esplorazione dello spazio latente eseguita da SINVAD è guidata o dalla probabilità di comportamenti scorretti, stimata dalla softmax layer di output, oppure dalla massimizzazione della copertura delle sorprese, partendo dal presupposto che input sorprendenti siano anche probabili di mettere in evidenza comportamenti scorretti. Per quantificare il grado di sorpresa di un input, SINVAD si basa sulle distanze tra i vettori di attivazione o sulla stima della densità del kernel.

Encoder e Decoder

I Variational Autoencoder sono composti da due componenti interconnesse: l'encoder ed il decoder. L'encoder trasforma l'input originale in uno spazio latente, che è una distribuzione multivariata normale con parametri rappresentati come medie e varianze in uno spazio z-dimensionale. Il decoder invece, opera la mappatura di un vettore z-dimensionale, che rappresenta un campione estratto dalla distribuzione normale multivariata, verso un vettore nello spazio di input. Questa operazione corrisponde a una trasformazione inversa che ricostruisce l'input originale nel suo spazio originario a partire dal vettore codificato. Sia l'encoder che il decoder costituiscono reti neurali addestrate simultaneamente al fine

di minimizzare sia l'errore di ricostruzione, che rappresenta la discrepanza tra gli input originali nel set di addestramento e le ricostruzioni generate dal VAE, sia la divergenza di Kullback-Leibler tra la distribuzione posteriore appresa e la reale distribuzione posteriore. Quest'ultima, di solito, è modellata come una distribuzione gaussiana unitaria per ciascuna dimensione dello spazio latente.

Al fine di conseguire tale obiettivo, si adotta un approccio in cui il codificatore non genera un singolo vettore di codifica di dimensione n, ma invece produce due vettori distinti, ciascuno di dimensione n. Questi due vettori sono noti come il vettore delle medie, μ, e il vettore delle deviazioni standard, σ. Essi rappresentano i parametri di una distribuzione di variabili casuali multivariata di lunghezza n, in cui l'elemento i-esimo di μ e σ indica rispettivamente la media e la deviazione standard dell' i-esima variabile casuale, X_i. Questi parametri vengono utilizzati per campionare una codifica dall'insieme di variabili casuali, e questa codifica campionata viene quindi fornita al decoder per la ricostruzione.

La generazione stocastica implicata da questo processo comporta che, anche quando l'input rimane costante e le medie insieme alle deviazioni standard mantengono invariata la loro specifica, la codifica risultante varierà in modo intrinseco ad ogni iterazione. Tale variazione è intrinseca al processo di campionamento, che introduce un elemento di casualità durante la generazione della codifica, sebbene i parametri statistici, quali le medie e le deviazioni standard, rimangano costanti.

Il vettore delle medie controlla la posizione centrale desiderata per la rappresentazione codificata di un input, mentre il vettore delle deviazioni standard influenza l'ampiezza dell'intervallo all'interno del quale la codifica può variare rispetto alla media. In quanto le codifiche vengono generate casualmente da punti qualsiasi all'interno della distribuzione definita da tali parametri, ciò implica che il decodificatore apprende a considerare non solo un singolo punto nello spazio latente come riferimento per un campione specifico di quella classe, ma anche tutti i punti prossimi a esso come rappresentativi della stessa classe. Questo permette al decodificatore di ricostruire non solo codifiche isolate e distinte nello spazio latente, ma anche quelle che presentano lievi variazioni, in quanto durante il processo di addestramento viene esposto a una serie di variazioni della codifica per lo stesso input. Questa caratteristica contribuisce a rendere il decoder in grado di affrontare non solo le codifiche discrete, ma anche quelle continue e leggermente differenti nello spazio latente, favorendo così una maggiore continuità nella decodifica.

Attualmente, il modello si espone a una certa quantità di variazione locale attraverso la modifica delle codifiche dei campioni, il che si traduce in spazi latenti che mostrano una uniformità su scala locale, in particolare tra campioni simili. L'obiettivo ideale consiste nel favorire una sovrapposizione anche tra campioni che non condividono notevoli somiglianze, al fine di consentire l'interpolazione tra le diverse classi. Tuttavia, poiché non vi sono restrizioni imposte sui valori che i vettori μ e σ possono assumere, esiste la possibilità che il codificatore apprenda a generare valori di μ molto diversi per classi distinte, raggruppandoli insieme e minimizzando le deviazioni standard σ. Tale approccio mira a ridurre al minimo la variazione nelle codifiche per campioni simili, ovvero a ridurre l'incertezza per il decodificatore. Questa strategia consente al decodificatore di effettuare una ricostruzione efficiente dei dati di addestramento.

Divergenza di Kullback-Leibler

Per promuovere questa condizione, si introduce la divergenza di Kullback-Leibler (KL) nella funzione di perdita. La divergenza KL tra due distribuzioni di probabilità rappresenta una misura quantitativa

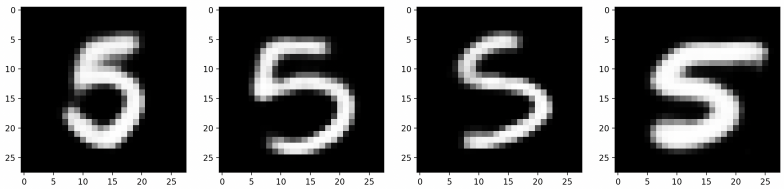

Figura 5.4: Immagini avversarie generate tramite SINVAD. Il TIG è in grado di ricostruire la distribuzione sottostante delle immagini e utilizzare tale conoscenza per generare nuovi input.

della loro discrepanza. Minimizzare la divergenza KL in questo contesto implica l'ottimizzazione dei parametri della distribuzione di probabilità, ossia μ e σ, in modo tale da renderli simili ai parametri della distribuzione target.

Per i VAE, la perdita KL è equivalente alla somma di tutte le divergenze KL tra la componente $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ in X e la normale standard. È minimizzata quando $\mu_i = 0$, $\sigma_i = 1$.

Per i Variational Autoencoders (VAE), la perdita KL corrisponde alla somma delle divergenze KL tra ciascuna delle componenti $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ in X e la distribuzione normale standard. Questa perdita è minimizzata quando i parametri μ_i sono uguali a zero e i parametri σ_i sono uguali a uno per ciascuna delle componenti.

Questa perdita incoraggia il codificatore a distribuire uniformemente le codifiche nello spazio latente intorno al suo centro. Qualora il codificatore tenti di raggruppare le codifiche in regioni specifiche lontane dall'origine, si vedrà penalizzato.

Ora, se si considerano esclusivamente i risultati della perdita KL nello spazio latente, otteniamo una disposizione densa e casuale delle codifiche, concentrate in prossimità del centro dello spazio latente, senza prestare particolare attenzione alla somiglianza tra le codifiche vicine. Tuttavia, il decoder troverà estremamente difficile generare informazioni significative da uno spazio così configurato, poiché manca di una struttura coerente.

Ottimizzazione dello spazio latente

L'ottimizzazione congiunta delle due perdite, invece, produce uno spazio latente che mantiene la somiglianza tra le codifiche vicine a livello locale tramite la formazione di cluster, ma presenta una densità significativa di punti vicino all'origine dello spazio latente a livello globale.

La configurazione sopra descritta rappresenta un equilibrio tra la tendenza della perdita di ricostruzione a formare cluster e quella della perdita KL a comprimere densamente lo spazio latente. Tale equilibrio forma cluster distinti che possono essere efficacemente decodificati dal decoder. Ciò implica che durante la generazione casuale, campionando un vettore dalla stessa distribuzione dei vettori codificati, cioè $\mathcal{N}(0, I)$, il decoder sarà in grado di eseguire una decodifica efficace. Inoltre, durante le interpolazioni, non vi sono salti bruschi tra i cluster, ma una transizione uniforme tra le caratteristiche che il decoder può comprendere.

Test su SINVAD

Da test effettuati nel paper "SINVAD: Search-based Image Space Navigation for DNN Image Classifier Test Input Generator" si evince come questo tipo di approccio basato su variational autoencoder porti

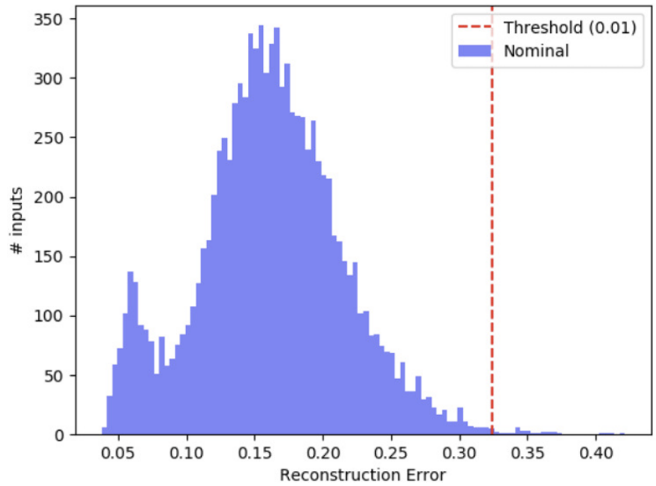

Figura 5.5: Soglie di validazione definite da SelfOracle.

alla produzione di input di test che possano effettivamente mettere alla prova la robustezza delle Deep Neural Networks quando si trovano ad affrontare test realistici. SINVAD ha ottenuto nel complesso buoni risultati per quanto concerne la fedeltà all'input originale da cui ha applicato le variazioni, la generazione di immagini affini al limite dell'indecisione per i modelli e nell'identificazione di punti deboli nei modelli utilizzati per i test.

5.3 SelfOracle

SelfOracle [11] è un rilevatore di anomalie basato su VAE (Variational Autoencoder) originariamente proposto per sistemi di guida autonoma. Questa tecnica sfrutta un VAE addestrato sugli stessi dati utilizzati per addestrare il sistema di deep learning in esame, ma non richiede la disponibilità di un insieme di anomalie. Si è adottato un VAE addestrato per minimizzare l'errore di ricostruzione, in cui l'immagine ricostruita è ottenuta da un decoder deterministico. Dopo l'addestramento, il VAE di SelfOracle avrà un errore di ricostruzione più elevato per i dati non validi (fuori distribuzione), mentre restituirà migliori ricostruzioni per i dati validi. SelfOracle sfrutta l'adattamento alla distribuzione di probabilità per trovare la soglia che discrimina gli input nominali (validi) da quelli anomali (non validi). In particolare, si adatta una distribuzione Gamma ai dati di addestramento attraverso il metodo di stima della massima verosimiglianza. L'adattamento alla distribuzione di probabilità fornisce un modello statistico per gli errori di ricostruzione restituiti dal VAE per i dati nominali. Questo modello statistico può essere utilizzato dal tester per configurare la soglia per l'errore di ricostruzione del VAE, che porta al tasso desiderato di falsi allarmi (cioè, input che appartengono all'insieme nominale ma vengono classificati come non validi). Ad esempio, un tasso di falsi allarmi del 1% significa scegliere il valore della soglia che divide la distribuzione Gamma in due parti, con la parte più a destra (associata a errori di ricostruzione più elevati) che ha un'integrale pari a 0,01. Gli input con un errore di ricostruzione superiore a tale soglia vengono considerati anomali (non validi), mentre quelli al di sotto vengono considerati nominali (validi).

5.4 Domande di ricerca

Nel corso della ricerca, sono stati comparati i due Generatori di Test Input (TIG) per valutare la loro capacità di generare input artificiali validi e il framework nel suo insieme per quanto riguarda il tempo impiegato a completare un ciclo di generazione.

5.4.1 RQ1: Confronto dei TIGs sul numero di input validi che scatenano fallimenti

Quale TIG produce, a parità di budget, il maggior numero di input validi che scatenano fallimenti?

I Generatori di Test Input (TIG) risultano particolarmente utili quando gli input induttivi di comportamento scorretto che producono sono validi. Per selezionare correttamente gli input validi, si ricorre alle tecniche di validazione automatica degli input. Questa interrogazione di ricerca mira a confrontare l'efficacia delle due implementazioni di TIG nella generazione di input validi, basandosi sulla valutazione effettuata da validatore automatico SelfOracle. Come metrica di valutazione, per ogni TIG preso in considerazione, si calcola la percentuale di input considerati validi rispetto all'insieme di tutti gli input generati.

5.4.2 RQ2: Analisi efficienza

Quale Test Input Generator completa in minor tempo un ciclo di generazione degli input?

La velocità di risposta del framework è intrinsecamente legata all'efficienza operativa e alla praticità d'uso in contesti applicativi reali. Gli obbiettivi di questa valutazione sono confrontare i due TIGs per valutare quale permette la generazione degli input nel minor tempo e un'analisi del tempo impiegato per la valdazione delle immagini e per l'allenamento dei classificatori. Come metriche di valutazione si considerano il tempo impiegato per la generazione delle immagini, per la validazione e per l'allenamento dei modelli.

5.5 Risultati sperimentali

Per ottenere i risultati sperimentali, sono state eseguite 10 iterazioni per ciascuno dei due TIG utilizzando il dataset MNIST. In ogni iterazione, sono stati generati 100 input avversari, portando il totale a 1000 immagini avversarie generate per ciascun TIG. Inoltre, al fine di limitare il costo computazionale dell'esperimento, è stata fissata l'etichetta delle immagini, specificamente il numero 5.

La valutazione della significatività statistica delle conclusioni è stata condotta attraverso l'esecuzione del test di Wilcoxon. In particolare, è stato effettuato un confronto tra i due TIG in termini di validità ed invalidità degli input generati. Un valore di $p < \alpha = 0.05$ (dove α rappresenta la probabilità di rigettare l'ipotesi nulla) è stato considerato come indicatore di significatività statistica nelle disparità osservate tra i due TIG analizzati.

5.5.1 RQ1: Confronto dei TIGs sul numero di input validi che scatenano fallimenti

Dai risultati ottenuti si evince che SINVAD ha generato un numero maggiore di input validi fino alla soglia di sicurezza del 99.9%; Oltre di essa tutti gli input di entrambi i TIGs sono stati validati al 100%.

Soglia (%)	Threshold	Input validi (%)	
		DLFuzz	SINVAD
90	0.2346	83	85.9
95	0.2616	89	**93.9**
99	0.3176	99.1	99.4
99.9	0.3888	100	100
99.99	0.4541	100	100
99.999	0.5160	100	100

Tabella 5.1: Input validi generati da DLFuzz e SINVAD per ogni percentuale di confidenza. I risultati validi dal punto di vista statistico sono riportati in grassetto.

Si osserva che tutte le immagini nel dataset originale MNIST presentano uno sfondo nero con un carattere scalato in scala di grigi posizionato centralmente. Tale caratteristica è appresa in maniera intrinseca dai modelli generativi di machine learning utilizzati da SINVAD. Al contrario, DLFuzz è in grado di apportare modifiche ai pixel delle immagini, generando spesso risultati con sfondi non completamente neri. Di conseguenza, le immagini generate da DLFuzz presentano proprietà totalmente assenti nell'insieme di dati originale, su cui è stato addestrato Variational Autoencoder di SelfOracle. Questa diversità ha condotto a valori più alti di perdita di ricostruzione.

Dal punto di vista statistico, si constata che esclusivamente la percentuale di confidenza al 95% ha generato risultati sostanzialmente distinti tra i due Test Input Generators. Nello specifico, si è evidenziata una predilezione per SINVAD di circa il 5%.

RQ1: SINVAD ha dimostrato di essere il Test Input Generator più efficace per la generazione di input avversari validi.

5.5.2 RQ2: Analisi efficienza

	Tempo medio (s)
DLFuzz	102.08
SINVAD	68.64

Tabella 5.2: Tempi medi delle esecuzioni.

La *Tabella 5.2* presenta i tempi medi impiegati da DLFuzz e SINVAD per la generazione di 100 immagini. Dai risultati emersi, si nota che DLFuzz impiega un tempo medio di circa 102 secondi a completare un ciclo di generazione mentre SINVAD ne impiega circa 68.6. La differenza di tempo è marcata, DLFuzz richiede approssimativamente il 48.25% in più di tempo per la generazione delle immagini rispetto a SINVAD.

	Tempo medio (s)
SelfOracle	6.75
Allenamento	126.78

Tabella 5.3: Tempi medi delle esecuzioni.

Per quanto concerne le prestazioni complessive del framework, il procedimento di SelfOracle registra un tempo medio di circa 6.75 secondi, mentre l'intervallo medio dedicato all'allenamento ammonta a 126.78 secondi. Di conseguenza, nell'ambito di un'intera esecuzione, che abbraccia sia la fase di addestramento del modello che la successiva validazione, il tempo totale impiegato per DLFuzz si attesta intorno a 236 secondi, pari a circa 4 minuti, mentre per SINVAD è di 202 secondi, corrispondenti a circa 3 minuti e mezzo.

RQ2: SINVAD manifesta un notevole vantaggio temporale nella generazione delle immagini, registrando un tempo inferiore del 48,25% rispetto a DLFuzz. In una esecuzione completa il framework impiega circa 202 secondi utilizzando SINVAD e circa 236 secondi utilizzando DLFuzz.

6

Conclusioni e Sviluppi futuri

6.1 Conclusioni

La presente dissertazione ha condotto un approfondito esame delle reti neurali, con particolare enfasi sulla comprensione della loro struttura e sottolineando il ruolo cruciale delle reti neurali convoluzionali (CNN) nell'ambito dell'Intelligenza Artificiale. L'analisi dettagliata delle componenti fondamentali di tali reti ha fornito una chiara panoramica sulle intricate dinamiche sottese al processo di apprendimento automatico.

Il capitolo dedicato al testing delle reti neurali ha sottolineato l'importanza critica dell'accurata valutazione delle prestazioni di tali modelli, tenendo conto della complessità delle relazioni apprese durante la fase di addestramento. L'approfondimento delle metodologie e delle metriche impiegate nel testing ha contribuito a delineare un quadro esaustivo per garantire la robustezza e l'affidabilità delle reti neurali in diverse applicazioni.

Il nucleo centrale di questo lavoro ha presentato il framework OlymTIGs come un innovativo contributo nel campo della sicurezza delle reti neurali. L'introduzione di OlymTIGs ha rappresentato una soluzione avanzata per la generazione di input avversari, sfidando la resistenza dei modelli di deep learning. Questo strumento, sviluppato miratamente per esplorare le vulnerabilità dei modelli, apre prospettive significative per migliorare la sicurezza e l'affidabilità delle reti neurali in contesti reali.

In conclusione, questa tesi non solo ha offerto una panoramica completa delle reti neurali e delle reti neurali convoluzionali, ma ha anche affrontato la critica sfida della sicurezza attraverso il framework OlymTIGs. L'avanzamento continuo in questo dinamico campo richiede un costante impegno per comprendere, sviluppare e implementare approcci innovativi. Si auspica che questo lavoro possa ispirare ulteriori ricerche e progressi nel campo delle reti neurali e della sicurezza dei modelli di deep learning.

6.2 Sviluppi futuri

Alla luce dei risultati e delle scoperte presentate in questa tesi, è cruciale esplorare una serie di sviluppi futuri che possano ulteriormente arricchire il campo delle reti neurali e la sicurezza dei modelli di deep learning.

In primo luogo, l'estensione del framework OlymTIGs per includere non solo il dataset MNIST, ma anche altri dataset rappresentativi di diverse sfide nel contesto dell'elaborazione delle immagini, costituirebbe un passo significativo. L'integrazione di dataset più complessi e diversificati consentirà una valutazione più completa delle capacità e delle vulnerabilità dei modelli, rendendo OlymTIGs un'applicazione più versatile.

Inoltre, l'espansione dei Test Input Generator all'interno di OlymTIGs costituisce un obiettivo chiave per ampliare la gamma di scenari di attacco possibili. L'integrazione di nuovi generatori di input avversari, oltre a SINVAD e DLFuzz, potrebbe fornire un'analisi più approfondita delle varie tipologie di attacchi che i modelli di deep learning potrebbero subire.

Infine, la diversificazione dei validatori utilizzati da OlymTIGs è fondamentale per una valutazione più accurata della robustezza dei modelli. L'integrazione di nuovi validatori, oltre a SelfOracle, potrebbe aprire la strada a una migliore comprensione delle possibili falle nei modelli e migliorare la capacità del framework di identificare e mitigare potenziali minacce.

Riassumendo, gli sviluppi futuri di OlymTIGs mirano a consolidare il framework come uno strumento completo e affidabile per la generazione di input avversari, estendendo la sua portata a diversi contesti e scenari. Tali miglioramenti contribuiranno a rafforzare la sicurezza delle reti neurali e promuoveranno ulteriori progressi nella ricerca sulla robustezza dei modelli di deep learning.

Bibliografia

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, e Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[3] Alessio Gambi, Giovani Guizzo, Sebastiano Panichella, Abhishek Aryan, Dongge Liu, Gunel Jahangirova, Jarkko Peltomäki, Jonathan Metzman, Marcel Böhme, Matteo Biagiola, Oliver Chang, Stefan Klikovits, Valerio Terragni, Vincenzo Riccio, Rebecca Moussa, Christian Birchler, Giovani Guizzo, e Sajad Khatiri. Search-based and fuzz testing ieee/acm international workshop, 2023.

[4] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, e Jiaguang Sun. Dlfuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 739–743, 2018.

[5] Sungmin Kang, Robert Feldt, e Shin Yoo. Sinvad: Search-based image space navigation for dnn image classifier test input generation. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pp. 521–528, 2020.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, e Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library In *Advances in Neural Information Processing Systems 32*. A cura di H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, e R. Garnett, pp. 8024–8035. Curran Associates, Inc., 2019.

[7] Vincenzo Riccio e Paolo Tonella. When and why test generators for deep learning produce invalid inputs: an empirical study. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1161–1173. IEEE, 2023.

[8] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J. Anders, e Klaus-Robert Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278, 2021.

[9] Sagar Sharma, Simone Sharma, e Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[10] Charles F. Stevens. The neuron. *Scientific American*, 241(3):54–65, 1979.

[11] Andrea Stocco, Michael Weiss, Marco Calzana, e Paolo Tonella. Misbehaviour prediction for autonomous driving systems. In *Proceedings of 42nd International Conference on Software Engineering*, ICSE '20, p. 12 pages. ACM, 2020.