Emanuele Spezia (mat. 486819)

Giuseppe Brescia (mat. 550718)

Breast Cancer Classification with Deep Learning

Progetto per gli esami di Machine Learning e Sistemi Intelligenti per Internet - AA 2020/2021

Abstract

Il progetto punta all'ausilio del Deep Learning, che si ispira al funzionamento del cervello umano e alle sue reti neurali biologiche. In questo caso specifico l'architettura utilizzata sarà una CNN (Convolutional Neural Network), costituita da più livelli attraverso i quali devono passare i dati prima di produrre finalmente l'output desiderato. Il Deep Learning serve a migliorare l'IA e a rendere possibili molte delle sue applicazioni pratiche.

Il lavoro su cui ci concentreremo riguarda lo studio, l'analisi e l'ottimizzazione attraverso modifiche, o implementazioni ad hoc, di una rete neurale CNN, appunto, dedita al controllo e ricerca intelligente di neoplasie maligne e benigne al seno. Nello specifico tale sistema sarà in grado di predire, con accuratezza intorno all'85-87%, se presenti o meno carcinomi duttali invasivi, che si sviluppano nel dotto (canalino che conduce il latte al capezzolo) e invadono il tessuto mammario fibroso o grasso al di fuori del dotto stesso; questa tipologia di carcinoma è la più diffusa e rappresenta circa l'80% di tutte le diagnosi di tale entità.

Obiettivi

1. Studio e analisi della CNN CancerNet proposta da Adrian Rosebrock per la Breast Cancer Classification;
2. Risoluzione problemi nel codice che ne rendono l'esecuzione obsoleta;
3. Implementazione e sviluppo di soluzioni alternative che aumentino l'accuracy e l'efficienza del sistema;
4. Studio e confronto dei risultati ottenuti attraverso i vari metodi.

Specifiche Progetto

Componente principale di questo sistema è sicuramente **TensorFlow**, una libreria software open source per l'apprendimento automatico che sfrutta il Machine Learning e fornisce moduli sperimentati e ottimizzati, utili nella realizzazione di algoritmi per diversi tipi di compiti percettivi e di comprensione del linguaggio. In questo progetto si sfrutta **Keras** per lo sviluppo di una rete CNN; Keras è una libreria per le reti neurali open source scritta in Python facile da usare, modulare ed estensibile, un'API di alto livello che può essere eseguita su TensorFlow, sopracitato, la quale consente una rapida sperimentazione e prototipazione in fase di esecuzione, indipendentemente se venga utilizzata una CPU o GPU.

L'oggetto del nostro studio è l'identificazione di IDC (Invasive Ductal Carcinoma), attraverso il training di un dataset di immagini prelevate da un database di Kaggle del peso complessivo di 3 GB. Questo set di dati contiene 277.524 patch di dimensioni 50×50 estratte da 162 immagini di vetrini di campioni di neoplasie al seno scansionati a 40x. Di questi, 198.738 sono risultati negativi e 78.786 sono risultati positivi con IDC.

Tale campione iniziale verrà poi suddiviso in tre sottocartelle chiamate rispettivamente:

- Testing
- Training
- Validation

Tale operazione verrà svolta automaticamente da un particolare file python che verrà presentato in seguito.

La fase sperimentale del progetto è poi quella che si pone come obiettivo quello di modificare o aggiungere parametri alla rete al fine di aumentare l'accuracy del sistema e la sua efficienza, nel caso migliore entrambe le cose.

Nella fase finale si produrranno i risultati delle varie combinazioni di impostazioni del modello, le quali verranno analizzate e comparate al fine di poter effettuare una scelta della miglior composizione di modifiche o upgrade che renda il sistema superiore sotto il maggior numero di punti di vista.

Specifiche Tecniche

L'analisi della rete e tutti i test che hanno interessato la stessa, sono stati eseguiti attraverso l'ausilio di 3 macchine, due portatili dove poter agevolmente e più rapidamente sperimentare cambiamenti e differenti set di configurazione della CancerNet, ed una fissa con un sistema migliore ed una scheda video dedicata molto potente, per poter diminuire il tempo di esecuzione delle varie epoche dei test. Qui di seguito le specifiche Hardware e Software delle macchine:

Specifiche Hardware Portatile 1:

OS: macOS Big Sur 11.5
CPU: Intel® Core™ i7 dual-core 3.3 GHz

RAM: 16 GB 2133 MHz LPDDR3
GPU: Intel Iris Graphics 550 1536 MB

Specifiche Hardware Portatile 2:

OS: Pop!_OS 21.04
CPU: Intel® Core™ i5-4210U 1.70GHz
RAM: 8 GB DDR3
GPU: GeForce GT 820M

Specifiche Hardware PC Fisso:

OS: Windows 10
CPU: AMD Ryzen 5 3600 3.6 GHz
RAM: 16 GB DDR4 3200 MHz
GPU: Palit RTX 2080Ti 11GB

Specifiche Software:

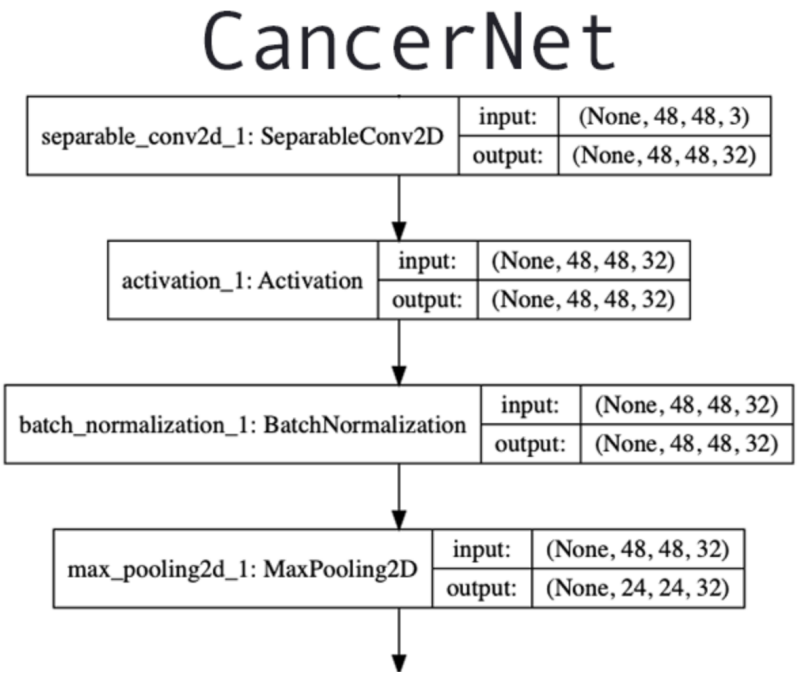
Python 3.9
TensorFlow 2.5
Keras 2.5
PyCharm Community 2021.1 (IDE)

Implementazione Progetto

L'implementazione finale del progetto consta di tre elementi principali che verranno di seguito esplicati nella loro interezza, contornati da elementi secondari che verranno discussi nel momento opportuno. Di questi tre elementi, due corrispondono alle due fasi di esecuzione del sistema, cioè alla preparazione (fase 1) e al training con successivi risultati (fase 2), l'altro elemento coincide con la definizione della struttura della rete neurale denominata "CancerNet", la quale verrà utilizzata dalla fase 2. Questi tre elementi prendono il nome rispettivamente di: build_dataset (fase 1), train_model (fase 2), cancernet.

CancerNet

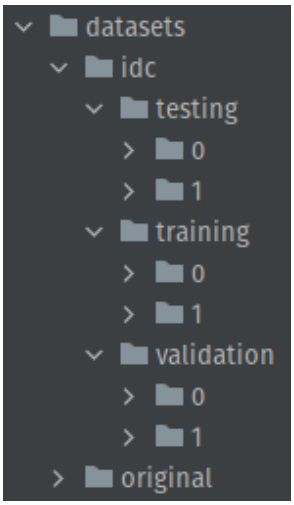
La CNN proposta da Adrian Rosebrock è formata esclusivamente da filtri CONV 3x3 impilati uno sopra l'altro prima di eseguire il max-pooling. La rete sfrutta la convoluzione separabile in profondità, attraverso SeparableConv2D dell'API Sequential di Keras, piuttosto che i livelli di convoluzione standard in quanto più efficiente, richiede meno memoria e meno calcolo. Ogni layer è quindi formato da un SeparableConv2D, una funzione d'attivazione e una MaxPooling2D. I layer hanno impilamento maggiore e numero di filtri maggiore rispetto al precedente, inoltre su di essi viene applicata batch-normalization e dropout. Infine la rete presenta un layer fully-connected e un classificatore softmax, il quale output darà le percentuali di previsione per ogni classe del modello.



L'architettura completa è disponibile al seguente link: https://github.com/EmanueleSpezia/BreastCancerClassification/blob/main/architettura_cancernet.png

Build Dataset

La prima fase del sistema serve semplicemente a dividere l'intero dataset di partenza in tre sottogruppi contenenti differenti porzioni dello stesso; questi sottogruppi verranno poi memorizzati in tre directory denominate: testing, training e validation. In ogni cartella si troveranno poi le immagini positive a idc (salvate nella directory "1") e quelle invece negative allo stesso (salvate nella directory "0").



Qui affianco è possibile prendere visione della divisione finale in directory successiva all'esecuzione di build_dataset.py, con la presenza del dataset di partenza all'interno della directory "original". In "idc", sono invece contenuti gli elementi esplicati in precedenza.
I dettagli implementativi sono generici e non dipendono dal progetto specifico, dunque non verranno discussi in dettaglio, il codice completo è comunque disponibile nella stessa repository di questa relazione.

Tutti i particolari relativi alla divisione nelle directory sopracitate e il metodo usato, sono resi noti attraverso il file di configurazione config.py, che ne contiene le generalità e che viene mostrato qui di seguito:

```
INPUT_DATASET = "datasets/original"

BASE_PATH = "datasets/idc"
TRAIN_PATH = os.path.sep.join([BASE_PATH, "training"])
VAL_PATH = os.path.sep.join([BASE_PATH, "validation"])
TEST_PATH = os.path.sep.join([BASE_PATH, "testing"])

TRAIN_SPLIT = 0.8
VAL_SPLIT = 0.1
```

Train Model

Infine il file chiamato train_model.py costituisce lo script di allenamento effettivo del modello. Al suo interno troviamo gli imports necessari al suo funzionamento dove i principali sono: matplotlib per grafici, tensorflow.keras per la data augmentation e altre utilità , sklearn per report e matrice di confusione, numpy per il numerical processing con Python.

Vengono inizializzati il numero di epoche, il learning rate e il batch size. Si tiene conto del peso delle classi per evitare eventuali squilibri. Tramite ImageDataGenerator si esegue la data augmentation, che perturba volutamente gli esempi di training prima di passarli alla rete per l'addestramento, importante per favorire la generalizzazione del modello e per ridurre la necessità di raccogliere più dati di addestramento. Perciò l'oggetto trainAug effettua modifiche casuali ai dati man mano che vengono generati.

Sempre all'interno del nostro train model vengono inizializzati i generatori di training, validation e test, che forniranno batch di immagini su richiesta. Successivamente vi è la fase di fit del modello che viene inizializzato con l'optimizer Adagrad e compilato con la loss function binary_crossentropy. Infine, una volta terminato l'addestramento, il modello viene valutato sui dati di test fornendo la confusion matrix, l'accuracy, la sensitivity e la specificity .

Modifiche apportate e problemi riscontrati

Iniziando dalle modifiche apportate alla rete è stata inizialmente presa in considerazione la porzione di codice all'interno di train_model.py. La prima caratteristica possibile da modificare è sicuramente il numero di epoche, nello specifico "NUM_EPOCHS" che caratterizza il tempo di esecuzione dell'intero processo, ma anche, da un certo numero di epoche in poi, l'accuracy del sistema. Infatti vi è stato un tentativo di rendere più veloce

ed efficiente il codice attraverso la riduzione del numero di epoche, scelta che si è però rivelata controproducente, perché porta ad una riduzione significativa dell'accuracy, ed una velocità di esecuzione che non ne giustifica tale diminuzione. Anche il processo opposto (aumento del numero di epoche), non porta ad un significativo aumento di accuracy, a danno di un aumento del tempo di esecuzione.

Dopo ciò si è lavorato alla modifica della rete neurale utilizzata, che con maggior probabilità avrebbe contribuito ad aumentare l'accuracy finale. La prima modifica alla quale si è pensato, essendo una rete neurale CNN, è stata quella di aggiungere uno o più layer per assicurare, in via ipotetica, una maggiore accuratezza nella classificazione. Purtroppo l'aggiunta di tali componenti non aumenta significativamente il valore prefissato, e porta, invece, ad una dilatazione quasi esponenziale, del tempo di esecuzione. Perciò si è scelto di non modificare il numero di livelli presenti (naturalmente, il caso opposto, di eliminazione di livelli, portava solo ad una perdita di accuracy sostanziosa e non è stato preso in considerazione).

Una successiva modifica è stata apportata al Batch size per cercare di rendere il sistema più efficiente computazionalmente, ma, come in realtà si poteva presumere, tale modifica porta anch'essa ad una riduzione troppo elevata di accuracy, anche se il tempo di esecuzione diminuisce anch'esso di molto.

La modifica che invece ha apportato i risultati più soddisfacenti è stata la funzione di attivazione utilizzata. Si sono provate le principali funzioni di attivazione disponibili in Keras, quelle che sono considerate le più efficaci, con un totale di sette differenti proposte: Sigmoid, Tanh, ReLU, Leaky ReLU, PReLU, ELU e SELU. Tutte sono state inserite a turno nella rete e la modifica che riesce ad ottenere un incremento dell'ccuracy del 2%, il più elevato ottenuto nei test, è la funzione Leaky ReLU. In questo sistema, però, non è sufficiente inserire la funzione così com'è, ma va passata come parametro ad un oggetto più generico denominato "get_custom_objects", facente parte della libreria utils di Keras,

e successivamente si procede all'aggiornamento di quest'ultimo, come riportato nel seguente frammento di codice:

```
#Aggiunta custom object e LeakyReLU
from tensorflow.keras.utils import get_custom_objects
from tensorflow.keras.layers import LeakyReLU as lr

#Aggiunta LeackyReLU come aggiornamento custom object
get_custom_objects().update({'leaky_relu': lr})
```

Per quanto riguarda i problemi riscontrati, il primo e più grave di tutti, si presenta all'avvio del train_model.py base, che riporta un errore e non viene eseguito; in poche parole il progetto così com'è non parte. Il problema era che Keras accetta come input della classe dei pesi (classWeight) un dizionario e l'elemento passato semplicemente non aveva quella struttura. Si è dunque proceduto alla creazione di un dizionario a partire dall'input base, passaggio evidenziato nel codice e riportato di seguito:

```
#MODIFICA ERRORE NEL CODICE -> KERAS HA BISOGNO DI UN DIZIONARIO!
classWeightCon = classTotals.max() / classTotals
classWeight = {i: classWeightCon[i] for i in range(2)}
```

Altro errore, anche se più semplice da correggere, è stato l'ausilio della funzione fit_generator() per il fit del modello anziché semplicemente fit(), che causava ulteriori problemi nella costruzione e fitting della CancerNet finale. Stesso identico caso per la funzione predict(), anch'esso sottolineato con commenti nel codice.

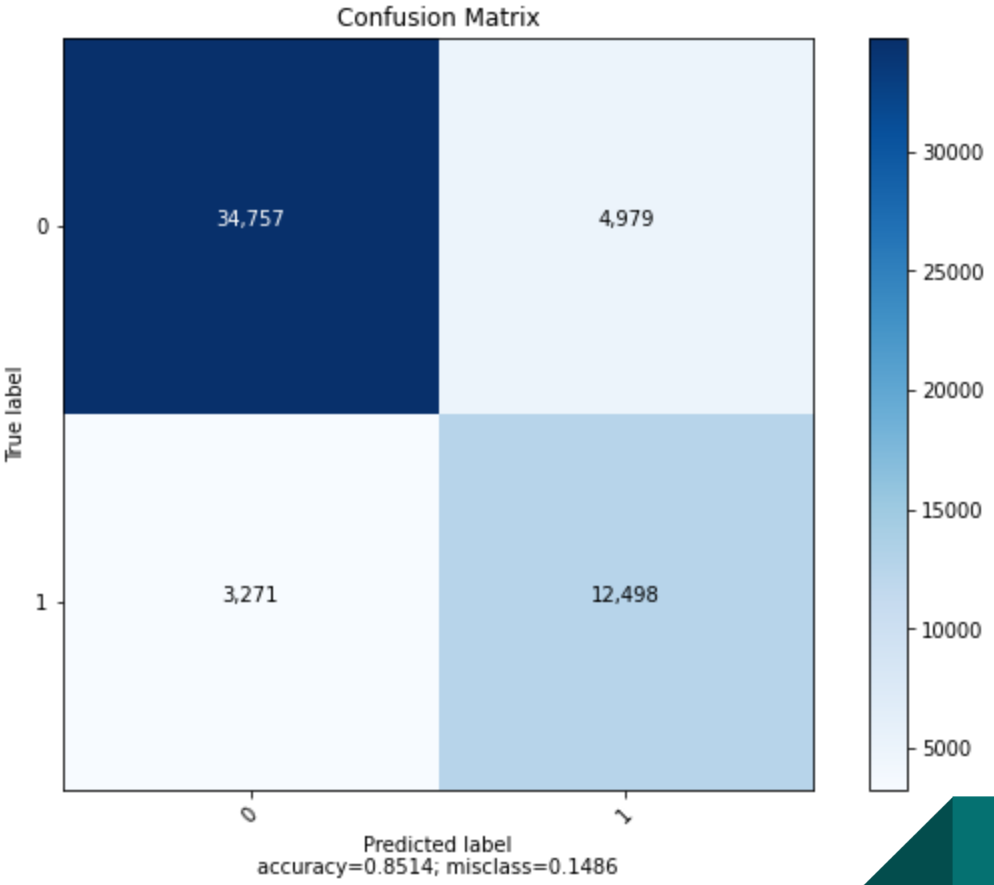
Ultimo errore presente nel codice, sempre riferito a train_model.py, si trova nella visualizzazione dei grafici finale, che non poteva avvenire a causa di un'errata inizializzazione e chiamata di alcuni parametri, che portava il programma a puntare una variabile inesistente, con conseguente non generazione dei suddetti grafici.

Risultati

Per quanto riguarda i risultati è fondamentale sfruttare le epoche di esecuzione del train_model.py nella loro interezza, portando dunque il loro numero a 40. In ogni caso l'accuracy finale del progetto privo di modifiche si attesta intorno all'85%, qui di seguito i dettagli con annessa la confusion matrix:

	precision	recall	f1-score	support
0	0.91	0.87	0.89	39736
1	0.72	0.79	0.75	15769
accuracy			0.85	55505
macro avg	0.81	0.83	0.82	55505
weighted avg	0.86	0.85	0.85	55505

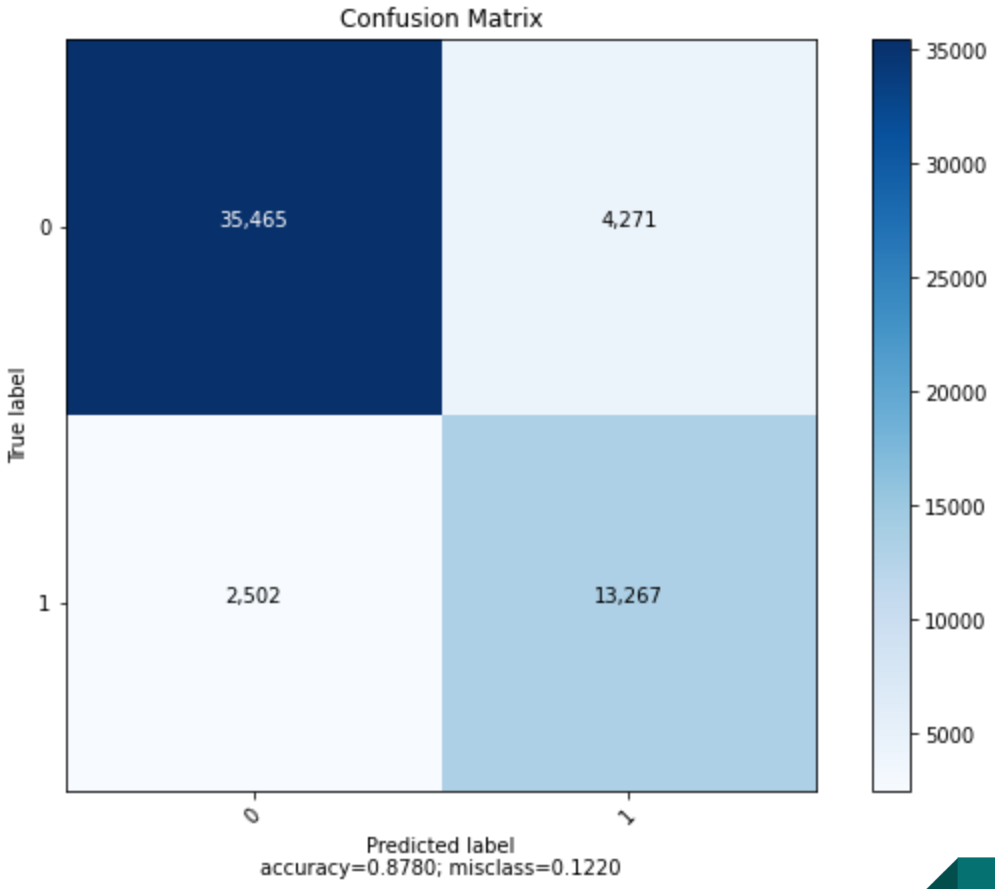
Accuracy: 0.8513
Specificity: 0.7925
Sensitivity: 0.8746



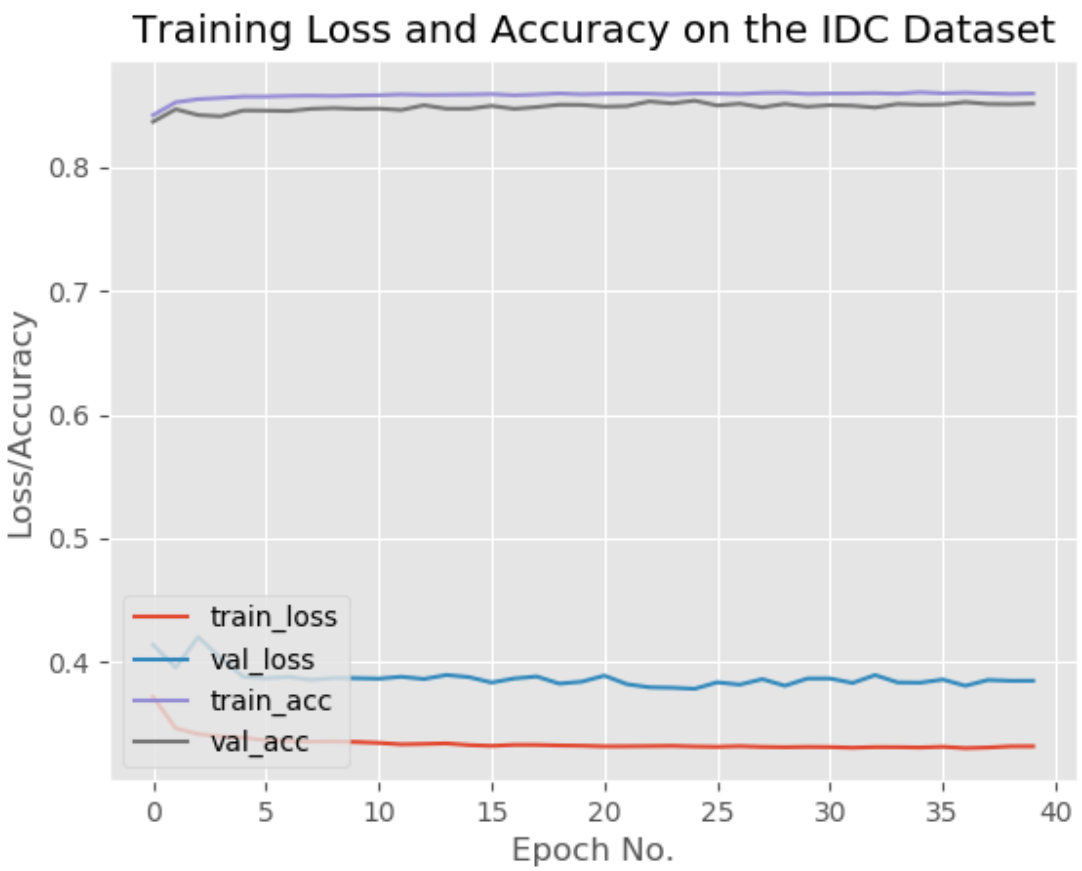
Attraverso le modifiche effettuate sia alla CancerNet che al codice di base, è stato possibile ottenere un incremento dell'accuracy di massimo 2%, sfiorando anche il 3% (nello specifico 2.8%), portandola ad un massimae di 87.8%. Qui di seguito i risultati nel dettaglio, con annessa la confusion matrix corrispondente::

	precision	recall	f1-score	support
0	0.89	0.93	0.91	39736
1	0.71	0.75	0.73	15769
accuracy			0.87	55505
macro avg	0.83	0.85	0.84	55505
weighted avg	0.88	0.87	0.87	55505

Accuracy: 0.8780
Specificity: 0.7565
Sensitivity: 0.9341



Infine viene allegato il grafico finale del Training Loss e dell'accuracy all'aumentare del numero di epoche (con massimo 40 come indicato nelle specifiche):



Bibliografia

https://data-flair.training/blogs/project-in-python-breast-cancer-classification/

https://keras.io/api/layers/activations/

https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6

https://www.pyimagesearch.com/2019/02/18/breast-cancer-classification-with-keras-and-deep-learning/