ALZHEIMER-DETECTION

Orlando De Bernardis
dborlando98@gmail.com

Pierantonio Carrozzini
piercarrozzini@gmail.com

https://github.com/theorly/alzheimer-detector.git

May 15, 2023

Abstract

Basandoci su uno studio svolto durante il percorso della laurea triennale, nel seguente lavoro si è pensato di realizzare un sistema di supporto alla diagnosi della malattia di Alzheimer, basato sull' analisi della scrittura a mano libera tramite Deep Learning.
Gli studi hanno infatti confermato che la scrittura a mano libera è tra le attività motorie che più vengono compromesse dal morbo di Alzheimer e sin dai primi momenti si possono registrare forme di disgrafia più o meno gravi, che ne permettono un rapido riconoscimento. La realizzazione di un sistema di supporto alla diagnosi richiede un'importante quantità e qualità di dati oltre che un'efficiente organizzazione di questi ultimi, visto il loro impatto sulle prestazioni del sistema. A tal proposito si utilizzerà un dataset proveniente da un precedente lavoro svolto durante il percorso di studi della laurea triennale in cui sono stati raccolti, tramite opportuni protocolli, campioni di scrittura a mano libera di soggetti non affetti da alcun disturbo e soggetti malati di Alzheimer appunto.

Dopo la realizzazione ed opportuna organizzazione del dataset si procederà ad addestrare una rete neurale ad hoc, e ad integrare il tutto in un applicazione Android.
Si potrebbe pensare, data la ridotta quantità di dati presente nel dataset, di ricorrere alla tecnica di *Transfer Learning*, in modo da aumentare la precisione della nostra rete neurale.
L'obiettivo come detto è quello di realizzare un sistema che, tramite la scansione di un campione di scrittura a mano libera sia in grado di predire con buona accuracy se il soggetto è affetto o meno da tale disturbo.

Keypoints:

- rielaborazione del dataset
- training della rete neurale
- implementazione del modello in Android tramite TensorFlowLite
- deploy del sistema di diagnosi

Contents

1 Introduzione

Le malattie neurodegenerative (ND) rappresentano un insieme di disturbi che afflig- gono il sistema nervoso e che colpiscono milioni di persone in tutto il mondo. Questi disturbi cognitivi alterano alcune funzioni umane essenziali quali l'attenzione, la memoria, l'abilità di giudizio, il linguaggio, la lettura e la scrittura. Le tre patologie più conosciute sono sicuramente il morbo di Alzheimer, il morbo di Parkinson e la Sclerosi Laterale Amiotrofica (SLA). Per il morbo di Alzheimer, così come per le altre patologie ND, non vi è ancora alcuna cura. A giocare un ruolo cruciale è una corretta e precoce diagnosi, che permette di contenere l'evoluzione della malattia e allungare l'aspettativa di vita di ogni singolo paziente.
L'importanza di una diagnosi tempestiva ha spinto i ricercatori a sviluppare appositi test che permettessero di visualizzare l'andamento della scrittura, al fine di fare predizioni sullo stato cognitivo del soggetto esaminato. L'intelligenza artificiale (IA) si è mostrata sin da subito un potente strumento al servizio dell'uomo, soprattutto nel campo della medicina moderna. Offre, infatti, uno strumento di supporto a medici e specialisti per diagnosi sempre più accurate e rapide, essendo in grado di cogliere dettagli impercettibili per l'essere umano.
Dunque, partendo dal lavoro svolto dal Dipartimento di Ingegneria Elettrica e dell'Informazione dell'Università di Cassino per la raccolta dei campioni di scrittura, nel seguente lavoro abbiamo preso in considerazione l'idea di costruire e addestrare un nuovo modello di rete neurale e di implementarla successivamente in un applicazione Android che tramite una scansione di un particolare campione di scrittura sia in grado di riconoscere un paziente malato di Alzheimer.

2 Dataset

2.1 Realizzazione del dataset

Per ricorrere all'IA è necessario, infatti, un ampio e organizzato dataset e dunque il protocollo deve essere ben definito. E' fondamentale per la validità del dataset che vi sia una distinzione bilanciata tra i campioni raccolti da soggetti malati e da soggetti sani. Il Protocollo si compone di 25 esercizi di scrittura a mano (denominati Tasks) eseguiti da 181 soggetti in totale, di cui 91 sani e 90 malati di morbo di Alzheimer. Gli esercizi sono stati eseguiti su fogli A4 bianchi. Le persone sane sono identificate con la sigla *Healthy Controls* e appartengono alla classe **HC**, mentre le persone malate identificate come *Patients* e appartengono alla classe **PT**. I Tasks sono disposti in ordine crescente di difficoltà e ognuno di essi mira ad analizzare alcune caratteristiche del paziente. Tenendo conto dei vari obiettivi, possono essere suddivisi:

- Task grafici , il cui obiettivo è di valutare l'abilità di ciascun paziente nel disegnare tratti elementari, unire punti, disegnare figure semplici o più complesse.
- Task di copia diretta e inversa , che mirano a verificare l'abilità a riprodurre lettere, numeri e parole di differente lunghezza e organizzazione spaziale.
- Task di memoria che mirano a testare la capacità del soggetto di riprodurre a memoria una parola o una frase

Nel seguente lavoro non sono stati presi in considerazione tutti e 25 i Task, ma ne sono stati scelti solo alcuni per motivi di disponibilità. In particolare sono stati scelti i seguenti:

- **TASK 01**: *firma* del paziente;
- **TASK 08**: il paziente deve scrivere per quattro volte la lettera "*l*";
- **TASK 09**: il paziente deve scrivere per quattro volte le lettere "*le*";
- **TASK 10**: il paziente deve scrivere la parola "*foglio*";

2.2 Pre-elaborazione del dataset

Sono necessari alcuni passaggi intermedi che permettono la perfetta riuscita dell'operazione. Uno dei passaggi fondamentali nella costruzione del dataset è l'estrazione della Region Of Interest (ROI) da ogni singola immagine. La ROI rappresenta la porzione di immagine utile per la rete neurale e, dunque, la porzione che rappresenta lo svolgimento del task in esame. L'estrazione delle ROI è stata eseguita mediante uno script scritto in Python e basato sulla libreria OpenCV che permette l'estrazione automatica delle ultime.

I passaggi fondamentali del codice utilizzato sono:

- **definizione dei path**: sono stati inseriti i path sorgente in cui prelevare immagini;
- **processing dell'immagine offline**: è stata operata una conversione in scala di grigi ed una binarizzazione mediante una Threshold ricavata dall'algoritmo di Otsu.

- **estrazione coordinate dell'immagine**: l'operazione è stata svolta sull'immagine bina- rizzata in precedenza con Otsu. L'obiettivo è quello di ricavare lo spazio delle coordinate (x,y), andando così a definire una ROI tramite i punti della traccia. L'operazione è eseguita da una funzione che, presa in input l'immagine binarizzata grayimg, fornisce in output un vettore contenente gli estremi di intervallo delle ascisse, e l'intervallo delle ordinate;
- **taglio dell'immagine**: una volta ricavate le coordinate dal punto precedente, si è utilizzata una funzione di *OpenCV* per ritagliare la sola regione di interesse dell'immagine.
- **resize dell'immagine**: una volta estratta l'immagine si verifica se questa si sviluppa in altezza o in ampiezza, quindi la dimensione maggiore viene adattata a 299, mentre quella inferiore viene modificata in modo da mantenere invariato il rapporto originale tra le due dimensioni.
- **copia dell'immagine**: infine è stata creata un'immagine bianca di dimensioni ***299x299*** su cui è stata incollata in posizione centrale l'immagine tagliata e ridimensionata;

2.3 Organizzazione finale del dataset

Infine, una volta tagliate e centrate le immagini tramite l'algoritmo precedente, è stata necessaria una riorganizzazione del dataset. Le immagini sono state suddivise in tre cartelle:

- **TRAINING SET**: contiene il 70% dei dati ed è utilizzato per l'addestramento del modello;
- **VALIDATION SET**: contiene il 20% del dataset e viene impiegato per valutare il modello in fase di training ed in base a questo interrompere l'addestramento;
- **TEST SET**: rappresenta un fold intero, contenente il 10% del dataset e viene utilizzato per valutare il modello finale e le sue prestazioni.

In conclusione, il dataset è costituito da:

- **664** campioni totali considerando le due classi di appartenenza *HC* e *PT*, sommando le scansioni di tutti e quattro i Task presi in considerazione;
- **465** campioni per il *TRAIN_SET*;
- **133** campioni per il *VALIDATION_SET*;
- **66** campioni per il *TEST_SET*;

```
data = tf.keras.utils.image_dataset_from_directory(directory = data_path, image_size=(299, 299))
data_iterator = data.as_numpy_iterator()
batch = data_iterator.next()

data = data.map(lambda x,y: (x/255,y))
scaled_iterator = data.as_numpy_iterator()

batch = scaled_iterator.next()

print(len(data))

train_size = int(len(data)*.7)
val_size = int(len(data)*.2)
test_size = int(len(data)*.1)

train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

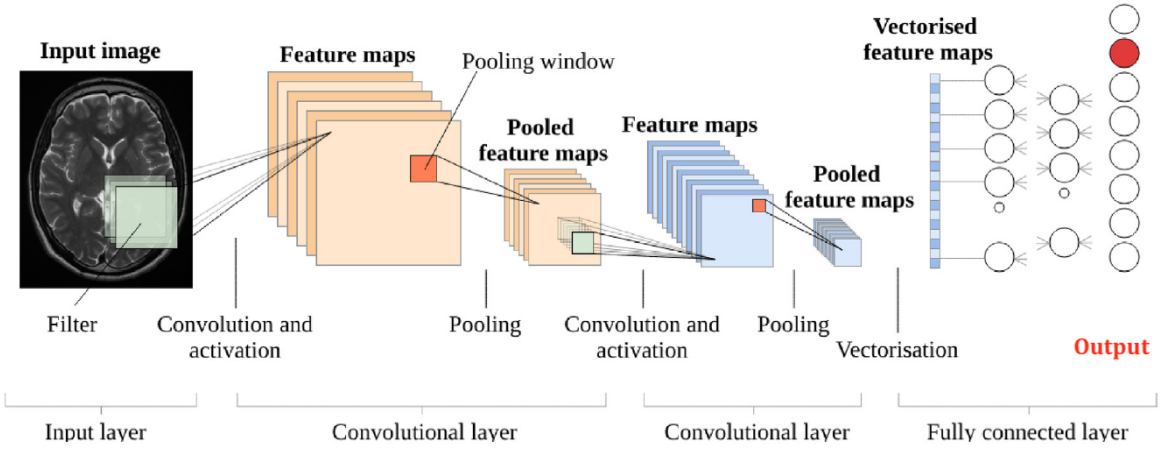
```
Found 664 files belonging to 2 classes.
21
```

Parte di codice per la suddivisione del dataset.

3 Le CNN

Le reti neurali convoluzionali, indicate con l'acronimo CNN o ConvNet, rappresentano, senza alcun dubbio, una delle più importanti categorie di reti neurali feed-forward. Le reti convoluzionali si ispirano a processi biologici, in particolare utilizzano il modello dell'organizzazione della corteccia visiva animale per realizzare la connessione tra neuroni.
Analogamente ad altre reti neurali, una rete convoluzionale è costituita da un layer di input, un layer di output e decine o centinaia di layer intermedi nascosti. Questi ultimi eseguono operazioni che alterano i dati, al fine di apprendere le features dei dati stessi. I tre layer più importanti di una CNN sono:

- **Convolutional layer**: è dove avviene la maggior parte del calcolo, sottoponendo le immagini a una serie di filtri convoluzionali, ciascuno dei quali evidenzia determinate features dalle immagini.
- **Pooling layer**: solitamente dopo un layer di convoluzione si utilizza un layer di pooling, che serve a diminuire le dimensioni del feature map in input, mantenendo però le caratteristiche dello stesso.
- **Fully connected layer**: è il livello che di fatto esegue la classificazione. Una volta ricavate le features, estratte dai livelli precedenti di convoluzione e di pooling, queste vengono mappate da un sottoinsieme di livelli completamente connessi agli output finali della rete.



Esempio di una Convolutional Neural Network.

3.1 Addestramento della rete

Per l'addestramento del modello è stato utilizzato *Google Colab* (anche chiamato Google Colaboratory). Questo è un ambiente on-line gratuito per lo sviluppo e l'esecuzione di applicazioni in Python che permette di lavorare a progetti di Data Science senza doversi preoccupare dell'installazione, configurazione e manutenzione di di un pc o server, richiedendo esclusivamente un browser e una connessione Internet.
Per compilare il modello abbiamo scelto come ottimizzatore Adam (ADAptive Moment estimation), per aggiornare i pesi della rete durante l'allenamento. E' uno degli ottimizzatore piu' robusti, tant'è che una rete allenata con Adam tende a convergere anche a fronte di piccole variazioni negli

iperparametri. Per calcolare il valore di loss abbiamo usato *BinaryCrossentropy* di Keras, che consente di utilizzare un valore intero per il label che identifica la vera classe di appartenenza. Ha il vantaggio di essere meno costosa in termini di computazione e di memoria, poich e viene utilizzato un intero per classe, anzich e un vettore. Successivamente, abbiamo utilizzato la funzione di early stopping messa a disposizione da Keras per il training del modello, in quanto risulta molto utile per fermare l'allenamento in caso di peggioramento delle performance del modello, in particolare monitorando il valore di "validation-loss" dei dati.

```
dnn_model = Sequential()

imported_model= tf.keras.applications.ResNet50(include_top=False,
input_shape=(299,299,3),
pooling='avg',classes=no_classes,
weights='imagenet')
for layer in imported_model.layers:
  layer.trainable=False

dnn_model.add(imported_model)
dnn_model.add(Flatten())
dnn_model.add(Dense(512, activation='relu'))
dnn_model.add(Dense(1, activation='sigmoid'))

dnn_model.summary()

dnn_model.compile(loss=tf.losses.BinaryCrossentropy(),
              optimizer=optimizer,
              metrics=['accuracy'])

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

La funzione di *early stopping* prende i seguenti parametri in input:

- **monitor**: come detto in precedenza il valore che è stato preso in analisi è il valore di *val_loss*, al fine di valutarne eventuali peggioramenti ad ogni epoca
- **patience**: indica quante epoche consecutive di peggioramento vengono tollerate e nel caso specifico il valore è stato impostato su *10*
- **restore_best_weights**: se impostato su *True* permette di memorizzare i pesi della rete prima delle 10 epoche di peggioramento

Tramite questa strategia è stato quindi possibile prevenire problemi di *overfitting* della rete. Infatti, nei casi in cui l'apprendimento è stato effettuato troppo a lungo o dove c'era uno scarso numero di dati per il training, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nella distribuzione tipica del resto dei casi. Perciò il modello impara le peculiarità del training set e non riesce ad adattarsi a dati nuovi. Si ha quindi overfitting quando il miglioramento delle prestazioni del modello sui dati di allenamento non implica un miglioramento delle prestazioni sui dati nuovi.[3]
Inoltre, come è possibile notare si è utilizzata la tecnica del *Transfer Learning*. Il Transfer Learning è un approccio utilizzato nell'apprendimento profondo, in cui la conoscenza viene trasferita da un modello ad un altro, tramite il quale si è in grado di risolvere un particolare task utilizzando un modello pre-addestrato su un task differente. L'obiettivo principale del transfer learning è quello di ottenere risultati migliori in tempi e costi ridotti.
E' stata infatti presa una rete di tipo **ResNet50** preaddestrata sul database pubblico ImageNet, e applicata dunque al nostro modello.
Infine, il modello inizia il training tramite la funzione **dnn_model.fit()**, con epochs=100 e batch size=32. L'allenamento, come detto, e' stato eseguito su Google Colab.

3.2 Risultati dell'addestramento

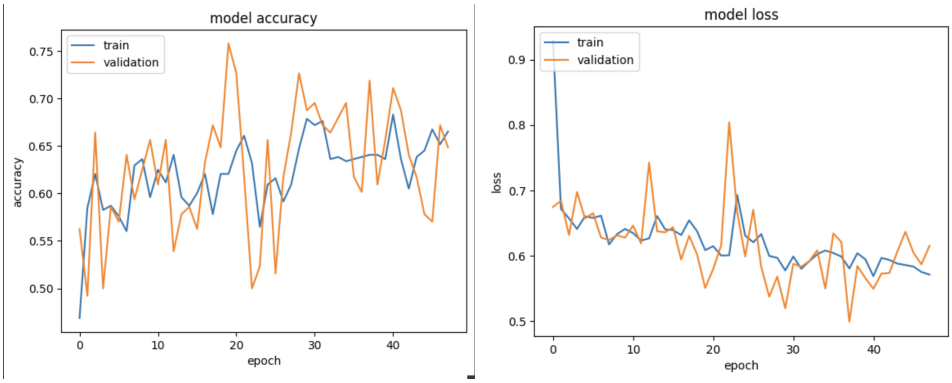
Una volta terminata la fase di training, il modello è stato valutato tramite l'apposita funzione di *evaluate*, che ha prodotto i seguenti risultati in termini di accuracy e loss:

- **accuracy** : 0.6719
- **loss** : 0.6108

```
[ ]  test_dict = dnn_model.evaluate(
         test,
         steps=test_size,
         return_dict=True
     )

     2/2 [==============================] - 3s 198ms/step - loss: 0.6108 - accuracy: 0.6719
```

Inoltre durante la fase di training tramite l'apposita libreria python *Pyplot* sono stati realizzati i seguenti grafici per valutare le metriche di accuracy e loss durante le varie epoche del train.



Le possibili classi sono:

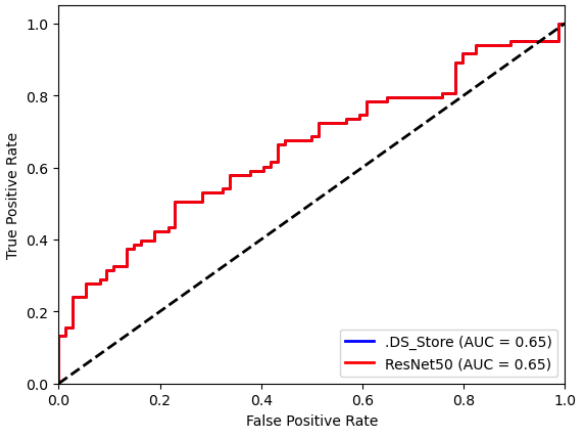
- **Positive**: si riferisce alla classe dei pazienti;
- **Negative**: si riferisce alla classe dei soggetti sani;

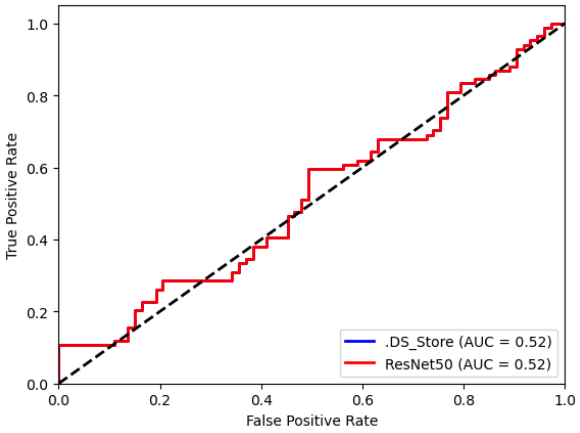
Considerando una matrice dove inseriamo classi effettive e classi previste come quella in figura, possono verificarsi quattro casi:

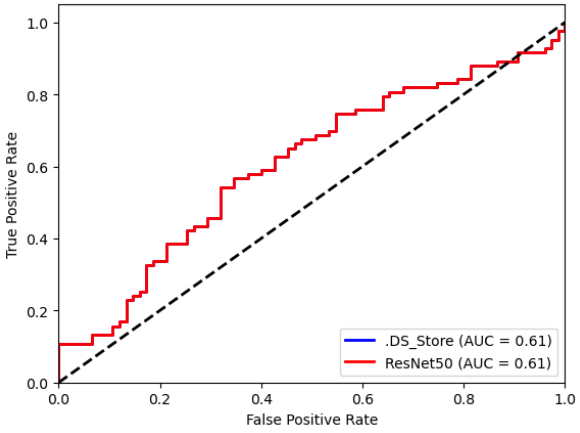
- **TP (True Positive)**: è il caso di una classificazione corretta. Viene diagnosticata una malattia ad un paziente che è realmente affetto da quella malattia;
- **FP (False Positive)**: è il caso di una classificazione non corretta. Ad esempio è diagnos- ticata ad un soggetto una certa malattia, ma il soggetto non è realmente affetto da quella malattia;
- **TN (True Negative)**: è il caso di un classificazione corretta. Nella diagnosi ad un soggetto una certa malattia viene esclusa e realmente il soggetto non soffre di quella malattia;

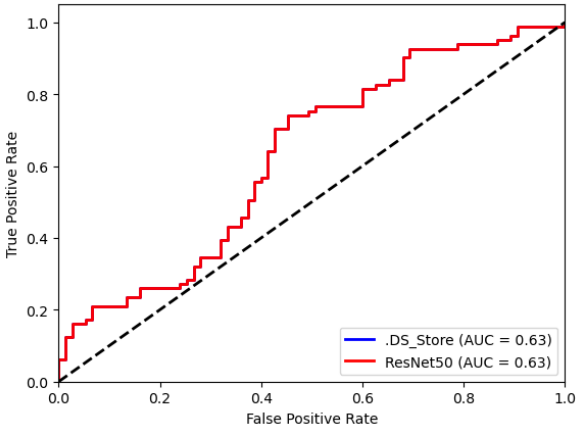
- **FN (False Negative)**: è il caso di una classificazione non corretta. Ad esempio quando non si diagnostica ad un paziente una certa malattia, ma il paziente è realmente affetto da quella malattia.

Dalla matrice è possibile ricavare alcuni parametri di notevole importanza nell'analisi di un test, rispettivamente *True Positive Rate* e *False Positive Rate*, che saranno parametri utili per effettuare un'analisi costruendo le curve ROC per i vari Task.
Una *curva ROC* è il tasso dei veri positivi in funzione del tasso dei falsi positivi. L'analisi della curva ROC di un test diagnostico permette di valutarne la bontà, di determinare il valore di cut-off più appropriato e di confrontare le performance di due, o più, diversi test. In generale un test risulta più accurato quanto più la sua curva ROC si avvicina all'angolo superiore sinistro del grafico.
Dunque, abbiamo valutato le prestazioni del nostro modello per ciascuno dei quattro task in nostro possesso e abbiamo poi realizzato le relative curve ROC presentate di seguito.









3.3 Problemi evidenziati

I risultati ottenuti attraverso questo studio rafforzano l'ipotesi di utilizzare il Deep Learning in campo medico come strumento aggiuntivo per la diagnosi di disturbi cognitivi e, considerata la limitata mole di dati a disposizione, non si può non ritenere che con l'uso di un dataset più grande le prestazioni migliorino notevolmente. Poiché le reti neurali necessitano di un grande quantitativo di dati per poter lavorare in modo efficiente, ecco che l'idea potrebbe essere quella di perfezionare il protocollo di acquisizione dei dati, al fine di acquisire ed ottenere una quantità più ampia di dati. Tuttavia, più vengono utilizzate reti complesse e alimentate da una notevole quantità di dati, maggiore saranno i costi e la potenza di calcolo richiesta.

3.4 Conversione in modello TensorflowLite

TensorFlow Lite fa parte dei mezzi messi a disposizione dal framework per far inferenza (predire con nuovi dati) dopo che un nostro modello è stato addestrato e rifinito. Lo scopo primario è quello di portare gli algoritmi di machine learning su tutti i dispositivi con limitate capacità computazionali e, dunque, anche per deployment su applicazioni Android. Il convertitore ha il compito principale di ottimizzare il modello riducendo le suo dimensioni e aumentando la suo velocità di esecuzione.

```
import tensorflow as tf
import os

converter = tf.lite.TFLiteConverter.from_keras_model(dnn_model)

tflite_noquant_model = converter.convert()
open(model_path + "model_yt.tflite", "wb").write(tflite_noquant_model)
```

Una volta salvato il nuovo modello TFLite, è stato aggiunto al progetto per l'app **Alzheimer-Detection** su Android Studio.

4 Android

A livello hardware sapevamo di aver bisogno di almeno due device per la realizzazione del progetto. 1 Computer con questi requisiti minimi:

- **Sistema operativo**: Microsoft Windows a 64 bit 8/10/11.
- **Risoluzione minima dello schermo**: 1280 x 800.
- **Architettura della CPU**: x86_64, Intel Core di seconda generazione o più recente; o CPU AMD con supporto Hypervisor.
- **RAM**: 8GB o più.
- 12 GB di spazio disponibile su disco per le varie installazioni richieste (IDE + Android SDK + Android Emulator).
- **Device**: 1 Device con SO Android che supportasse una versione API Android ¿=30 che si traduce con una versione Android ¿=11 o "Red Velvet Cake".

Per il corretto funzionamento dell'ambiente di sviluppo erano richieste le seguenti specifiche:

- Java Developer Kit, nello specifico: Java™ Platform Standard Edition 20 Development Kit - JDK™ 20.
- Android Studio SDK 2022.2.1 "Flamingo.
- Android Emulator versione 32.1.12, che ci ha permesso di testare l'app nelle prime fasi direttamente su computer.

Versioni Software utilizzate nel progetto:

- Java Version: Java 8 (1.8)
- Android Gradle Plugin: 7.4.2
- Gradle: 7.5
- JetBrains Runtime Version jbr-17 (17.0.6)
- Minimum SDK Version: 30 (Android 11).
- Compile SDK Version e Minimum SDK Version: 33(Android 13.0 o "Tiramisù".
- Virtual Device Emulator: Pixel 3a API 33 (x86_64).

La classificazione di immagini è una delle applicazioni più comuni nell'ambito dell'Intelligenza Artificiale. Consiste nell'identificare l'oggetto o la classe di un'immagine in base ai suoi tratti distintivi.
Lo scopo di questo capitolo sarà quello di illustrare quali sono state le tecniche e i metodi utilizzati nella creazione dell'applicazione Android, nonché fornire le schermate dell'applicazione in questione al fine di chiarire il funzionamento di ciascuna; infine, un link ad una breve demo.
Passi seguiti per lo sviluppo:

- *Preparazione dell'ambiente di sviluppo*: prima di iniziare a sviluppare l'applicazione, è stato necessario preparare l'ambiente di sviluppo. Con ciò si intende l'installazione e configurazione di Android Studio, del framework TensorFlow Lite e dei modelli di classificazione di immagini.
- *Creazione del layout dell'applicazione*

In generale, si è cercato di mantenere l'applicazione per quanto più possibile fluida e diretta al punto focale dell'attività legata al corso di studio. Sono state incluse solo schermate necessarie al corretto funzionamento di quest'ultima ed alcune funzionalità che non richiedessero un appesantimento notevole riportando rallentamenti nell'utilizzo.
Le classi utilizzate sono le seguenti:

- Schermata di avvio
- Schermata di scelta della modalità di utilizzo dell'app
- Schermata Main per classificare le immagini
- Schermata "New Feature" per l'attività progettuale
- Schermata "About us"
- Schermata "Email"

Per implementare la classificazione di immagini, è necessario utilizzare il modello pre-addestrato "ResNet50" fornito da @. In particolare, si deve caricare il modello e convertire l'immagine da classificare in un formato utilizzabile dal modello. Non appena l'immagine è pronta, è possibile utilizzare il modello per effettuare la classificazione.
Quest'ultimo restituirà le probabilità associate a ciascuna classe.
Infine, è possibile visualizzare il risultato della classificazione sull'interfaccia dell'applicazione. È stato utilizzato un TextView per visualizzare il nome della classe con la probabilità più alta.
La cattura dell'immagine avviene con l'utilizzo del seguente codice Java:

```
captureBtn.setOnClickListener(new View.OnClickListener() {
    PierCarrrozzini
    @Override
    public void onClick(View view) {
        if (checkSelfPermission(Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED) {
            Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            startActivityForResult(intent, requestCode: 3);
        } else {
            requestPermissions(new String[]{Manifest.permission.CAMERA}, requestCode: 100);
        }
    }
});
```

Se i permessi di accesso alla Camera dello smartphone sono stati garantiti dall'utente finale, l'app procederà ad aprire la fotocamera e permetterà all'utente di scattare una foto con la camera interna o esterna ed in seguito, se l'utente si riterrà soddisfatto dell'immagine, quest'ultima verrà importata all'interno dell'app. Se i permessi non saranno già stati concessi l'app non riuscirà ad accedere alla camera e li richiederà nuovamente all'utente.
La scelta di un'immagine dalla galleria sfrutta il seguente codice:

```
selectBtn.setOnClickListener(new View.OnClickListener() {
    PierCarrrozzini
    @Override
    public void onClick(View view) {
        Intent intent = new Intent();
        //Intent intent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        intent.setAction(Intent.ACTION_GET_CONTENT);
        intent.setType("image/*");
        startActivityForResult(intent, requestCode: 10);
    }
});
```

Tramite la pressione del pulsante select si potrà quindi scegliere un'immagine in qualsiasi formato presente sullo smartphone.
Al termine della scelta l'immagine verrà trasferita all'interno dell'ImageView per essere mostrata all'utente e processata dall'applicazione tramite l'utilizzo della rete neurale ResNet50 preaddestrata.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    if (resultCode == RESULT_OK) {
        if (requestCode == 3) {
            Bitmap image = (Bitmap) data.getExtras().get("data");
            int dimension = Math.min(image.getWidth(), image.getHeight());
            image = ThumbnailUtils.extractThumbnail(image, dimension, dimension);
            imageView.setImageBitmap(image);

            image = Bitmap.createScaledBitmap(image, imageSize, imageSize,  filter: false);
            classifyImage(image);
        } else { //if (requestCode == 10) {
            Uri uri = data.getData();
            Bitmap image = null;
            try {
                image = MediaStore.Images.Media.getBitmap(this.getContentResolver(), uri);
            } catch (IOException e) {
                e.printStackTrace();
            }
            imageView.setImageBitmap(image);
            image = Bitmap.createScaledBitmap(image, imageSize, imageSize,  filter: false);
            classifyImage(image);
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

La funzione mostrata ha lo scopo di processare i dati in ingresso dal select Button o capture Button affinché divengano disponibili come thumbnail nella cornice dell'ImageView. Dopodiché viene effettuato un rescale per permettere ai dati in ingresso nella rete neurale di essere analizzati correttamente.
Infine, l'immagine viene data in pasto alla funzione classifyImage ().

```
try {
    ResnetNoquantv2 model = ResnetNoquantv2.newInstance(getApplicationContext());

    // Creates inputs for reference.
    TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1, 299, 299, 3}, DataType.FLOAT32);
    ByteBuffer byteBuffer = ByteBuffer.allocateDirect( capacity: 4 * imageSize * imageSize * 3);
    byteBuffer.order(ByteOrder.nativeOrder());

    int[] intValues = new int[imageSize * imageSize];
    image.getPixels(intValues,  offset: 0, image.getWidth(),  x: 0,  y: 0, image.getWidth(), image.getHeight());
    int pixel = 0;

    for (int i = 0; i < imageSize; i++) {
        for (int j = 0; j < imageSize; j++) {
            int val = intValues[pixel++];
            byteBuffer.putFloat( value: ((val >> 16) & 0xFF) * (1.f / 255));
            byteBuffer.putFloat( value: ((val >> 8) & 0xFF) * (1.f / 255));
            byteBuffer.putFloat( value: (val & 0xFF) * (1.f / 255));
        }
    }
    inputFeature0.loadBuffer(byteBuffer);

    // Runs model inference and gets result.
    ResnetNoquantv2.Outputs outputs = model.process(inputFeature0);
    TensorBuffer outputFeature0 = outputs.getOutputFeature0AsTensorBuffer();
```

```
            float[] confidences;
            confidences = outputFeature0.getFloatArray();
            int maxPos;
            if (confidences[0] > 0.5) {
                maxPos = 0;
            } else {
                maxPos = 1;
            }

            String[] classes = {"PT", "HC"};

            result.setText(classes[maxPos]);

            String s = "";
            String p = "";
            s += String.format("%s: %.1f%%\n", classes[0], confidences[0] * 100);
            p += String.format("%s: %.1f%%\n", classes[1], (1 - confidences[0]) * 100);
            if (maxPos == 0) {
                confidence.setText(s + p);
            } else {
                confidence.setText(p + s);
            }

            // Releases model resources if no longer used.
            model.close();
        } catch (IOException e) {
            Log.e(TAG,  msg: "IOException " + e.getMessage());
            // TODO Handle the exception
        }
```

È qui che i dati vengono processati dalla ResNet per poi rilasciare in output base il valore 0 o 1, mentre in outputFeature0 un valore float che rappresenterà l'accuracy della predizione. Tramite un semplice if poi si decide se mostrare HC (Healtycare o paziente sano) in corrispondenza del valore 1, oppure PT (Patient o paziente malato) in corrispondenza dello 0. Nella TextView sottostante, concatenando due stringhe, verranno mostrati i valori floating point dell'accuracy della predizione ordinati dal maggiore al minore. Infine, il modello viene rilasciato e chiuso, e può essere riutilizzato per predire altre immagini.

Nella schermata di progetto viene implementato un ToggleButton che permette di scegliere quale delle due reti neurali utilizzare per la predizione. Come già riportato nei capitoli precedenti, la rete ResNet50 ha portato risultati migliori rispetto alla rete InceptionV3.

La funzione che permette di inviare una mail tramite un qualsiasi client presente sul proprio smartphone:

```
private void sendMail() {
    String recipientList = mEditTextTo.getText().toString();
    String[] recipients = recipientList.split( regex: ",");

    String subject = mEditTextSubject.getText().toString();
    String message = mEditTextMessage.getText().toString();

    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.putExtra(Intent.EXTRA_EMAIL, recipients);
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);
    intent.putExtra(Intent.EXTRA_TEXT, message);

    intent.setType("message/rfc822");
    startActivity(Intent.createChooser(intent, title: "Choose an email client"));
}
```

La funzione setType("message/rfc822") permette di aprire solo app di messaggistica riguardanti l'invio e ricezione di e-mail.

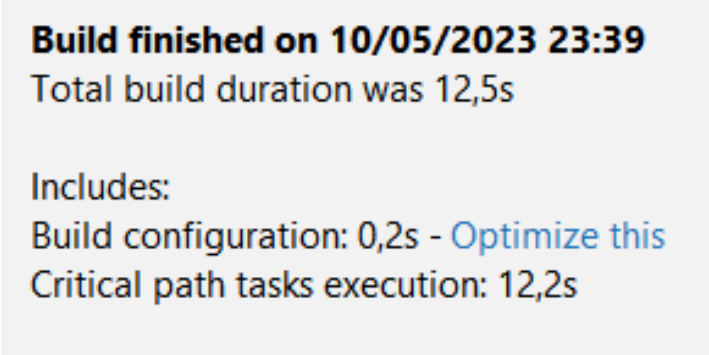
La funzione che permette di aprire i vari link presenti all'interno dell'app (es. nella sezione About Us):

```
private void gotoUrl(String s) {
    Uri uri = Uri.parse(s);
    startActivity(new Intent(Intent.ACTION_VIEW,uri));
}
```

Terminato l'import delle risorse utili al funzionamento dell'app (librerie, grafiche, assets) e la scrittura del codice, siamo passati alla fase di debugging e testing. Seguita dalla fase finale di creazione dell'APK. Quest'ultimo è stato creato sia in versione debug, sia in versione release tramite la creazione di un keystore. L'utilità della versione release per ora è limitata ma potrà essere sfruttata in seguito per estendere l'attività di progetto in quanto permette lo sviluppo di un "Android App Bundle" per rendere disponibile l'applicazione anche sul Google Play Store. Un altro importante vantaggio di questa modalità di installazione dell'APK è che in futuro si potrebbe pensare di rendere disponibili le risorse più pesanti (come le CNN) tramite download diretto in app, allegerendo così il download iniziale dal Play Store e rendendo l'app più immediata e fruibile.

4.1 Statistiche di progetto

Alcune metriche interessanti che possono risultare utili al fine di confrontare il progetto nello stato attuale, con eventuali migliorie apportabili in futuro all'app o alle reti neurali utilizzate.

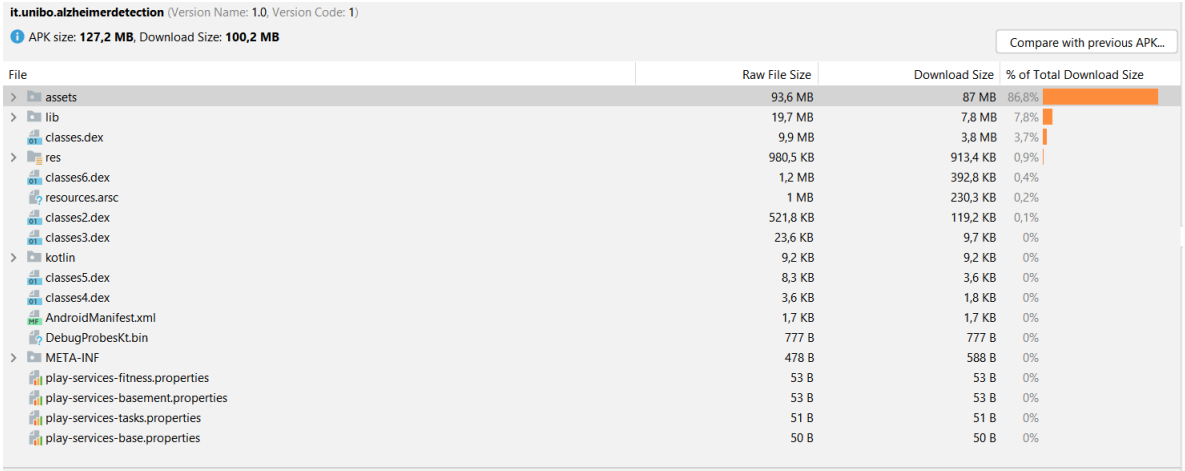


Come si può vedere la maggior parte del tempo richiesto per completare la build è impiegata dal Critical Path di conseguenza non migliorabile. Mentre il tempo per configurare la build occupa un tempo trascurabile.

```
BUILD SUCCESSFUL in 20s
38 actionable tasks: 9 executed, 29 up-to-date
```

Senza considerare i secondi che sono richiesti affinché il Daemon Gradle inizi a funzionare, o il progetto venga configurato, possiamo notare come le tempistiche vengono ulteriormente ridotte. Questo poiché molti dei task non cambiano di volta in volta risultando quindi già "UP-TO-DATE" e anche tra i task eseguiti, buona parte risulta richiedere prestazioni irrisorie a livello di calcolo.

4.2 Dimensioni



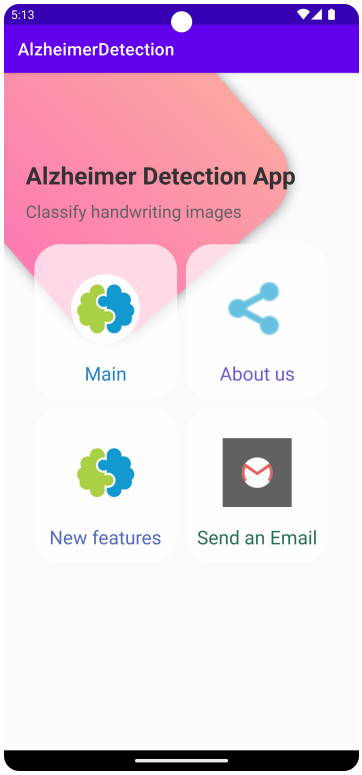
Per quanto riguarda le dimensioni, l'apk finale ha un peso che supera di poco i 100MB se consideriamo l'app inziale. Mentre arriva ad occupare poco più di 200 MegaBytes di memoria se l'app in questione è nella versione che comprende anche l'attività progettuale. Questo perché come si può vedere dai prossimi grafici, quasi la totalità del peso dell'applicativo è dato dalle due reti neurali. Ci è sembrato interessante portare anche una comparazione tra le dimensioni dei 2 progetti:

app-debug.apk (old) vs AlzheimerDetectionV2.apk (new)

File	Old Size	New Size	Diff Size
app-debug.apk	127,2 MB	212,5 MB	85,3 MB
assets/	93,6 MB	180,7 MB	87,1 MB
classes2.dex	521,8 KB	1000,5 KB	478,7 KB
res/	1,2 MB	1,3 MB	38,5 KB
AndroidManifest.xml	5,9 KB	6 KB	152 B
play-services-tasks.properties	76 B	76 B	0 B
play-services-fitness.properties	80 B	80 B	0 B
play-services-basement.properties	82 B	82 B	0 B
play-services-base.properties	74 B	74 B	0 B
kotlin/	24,1 KB	24,1 KB	0 B
DebugProbesKt.bin	1,7 KB	1,7 KB	0 B
META-INF/	489 B	489 B	0 B
lib/	19,7 MB	19,7 MB	0 B
classes4.dex	3,6 KB	0 B	-3,6 KB
classes5.dex	8,3 KB	0 B	-8,3 KB
classes3.dex	23,6 KB	0 B	-23,6 KB
resources.arsc	1 MB	1016,1 KB	-24,8 KB
classes.dex	9,9 MB	8,9 MB	-1,1 MB
classes6.dex	1,2 MB	0 B	-1,2 MB

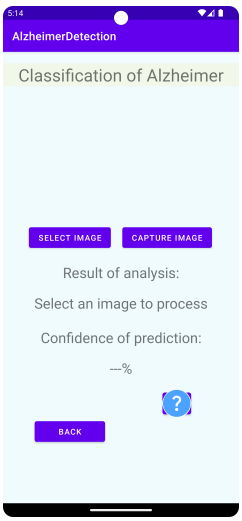
5 Demo

In questo capitolo verrà fornita una breve demo del funzionamento dell'app Alzheimer-Detection.

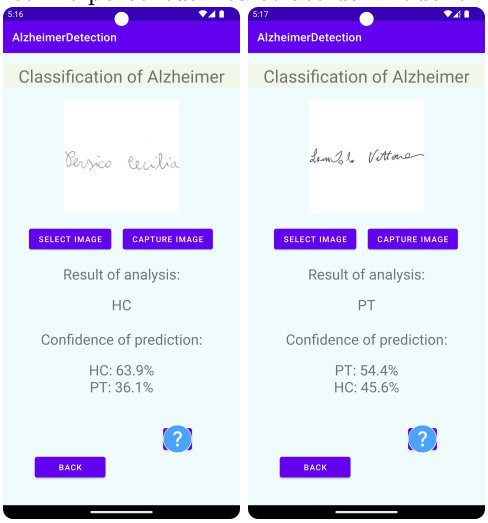


Una volta eseguita l'applicazione ci si troverà dinanzi alla seguente schermata di avvio, contenente quattro diverse possibili scelte:

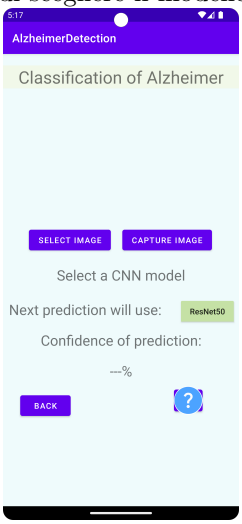
- **Main**: cliccando sull'icona del main si avvierà la schermata principale per effettuare le predizioni. In questa sezione sarà possibile effettuare le predizioni sfruttando solamente il modello basato su ResNet50.
- **About us**: contiene tutte le informazioni di contatto e i link alla repository git del progetto;
- **New features**: cliccando su questa sezione si avvierà la schermata che permette di effettuare le predizioni ed, inoltre, sarà possibile scegliere tra le due diverse architetture di rete;
- **Send an email**: tramite questa funzionalità sarà possibile inviare noi una mail. E' stata inserita per raccogliere feedback sul progetto, in modo da poter migliorare il progetto in futuro.



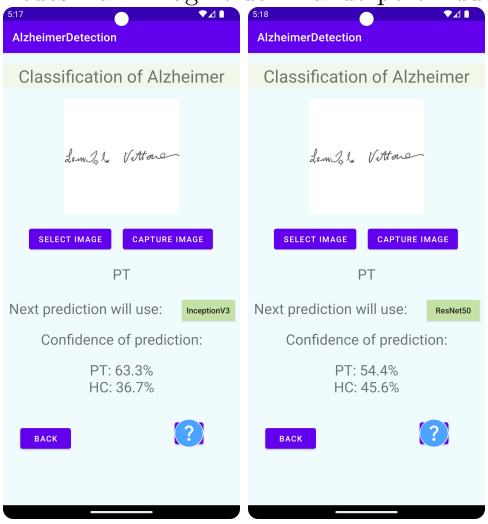
Una volta cliccato sulla sezione main ci si troverà dinanzi alla seguente schermata. A questo punto cliccando sul punto interrogativo in basso si aprirà una schermata di informazioni dove verrà spiegato come raccogliere l'immagine per ottenere una predizione migliore possibile. E' possibile catturare una nuova immagine o, in alternativa, selezionare un'immagine dalla galleria.
Una volta selezionata l'immagine verrà elaborata dall'app, verrà effettuata la predizione dal modello sottostante e stampato a schermo il risultato (HC se paziente classificato come sano, o PT se malato) con le percentuali calcolate dal modello.



Accedendo alla sezione *New features* invece, ci si troverà dinanzi alla medesima schermata tuttavia con un bottone che permetterà di scegliere il modello da utilizzare per la predizione.



A questo punto basterà ripetere il procedimento per ottenere la predizione. Ecco degli screen di una predizione della medesima immagine utilizzando però i due diversi modelli.



Infine, cliccando sulla sezione *About us* verranno visualizzati tutti i link di contatto e alla documentazione del progetto.

6 ATTIVITA' PROGETTUALE

Nel seguente capitolo si affronterà il discorso dell'attività progettuale da 3 CFU per il corso di Sistemi Digitali M.
L'idea è stata quella di addestrare un nuovo modello, utilizzando sempre la tecnica del *Transfer Learning*, implementare anche quest'ultimo nell'app Alzheimer-Detection, ed infine effettuare una nuova valutazione delle prestazioni ed un confronto con il modello precedente.

6.0.1 Addestramento del modello

Il dataset utilizzato per il training non ha subito alcuna variazione, è sempre suddiviso nelle tre cartelle di Train, Validation e Test. Come accennato in precedenza, anche in questo caso è stata adottata la tecnica del *Transfer Learning* visti i pochi campioni a disposizione, al fine di cercare un miglioramento delle performance generali e una riduzione dei tempi.
Questa volta è stata utilizzata una rete **InceptionV3**, pre-addestrata sempre sul database pubblico di ImageNet, e applicata poi al nostro caso di studio.

```
from keras.models import Sequential
from keras.models import Model
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping, ReduceLROnPlateau, TensorBoard
from keras import optimizers, losses, activations, models
from keras.layers import Convolution2D, Dense, Input, Flatten, Dropout, MaxPooling2D, BatchNormalization, GlobalAveragePooling2D, Concatenate
from keras import applications

base_model = applications.InceptionV3(weights='imagenet',
                                       classes = no_classes,
                                include_top=False,

                                input_shape=(299, 299,3))
base_model.trainable = False

add_model = Sequential()
add_model.add(base_model)
add_model.add(GlobalAveragePooling2D())
add_model.add(Dropout(0.5))
add_model.add(BatchNormalization())
add_model.add(Dense(512, activation='relu'))
add_model.add(Dense(1,
                    activation='sigmoid'))

dnn_model = add_model
dnn_model.compile(loss=tf.losses.BinaryCrossentropy(),
              optimizer=optimizers.SGD(lr=1e-4,
                                       momentum=0.9),
              #optimizer = Adam(),
              metrics=['accuracy'])

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

dnn_model.summary()
```

In questo caso è stato aggiunto un layer di *BatchNormalization*, usato per normalizzare l'output di un layer durante l'allenamento in modo da avere media nulla e varianza unitaria. Come in precedenza è stata utilizzata la funzione di *early stopping* al fine di evitare overfitting della rete. A differenza del precedente modello questa volta è stata utilizzata come ottimizzatore il SDG(Stochastic Gradient Descent) per aggiornare i pesi della rete durante l'allenamento.
Il modello è stato addestrato sempre utilizzando Google Colab, per un totale di 50 epoche. Al termine è stato valutato tramite l'apposita funzione di evaluate.

```
[ ]  test_dict = dnn_model.evaluate(
         test,
         steps=test_size,
         return_dict=True
     )
```

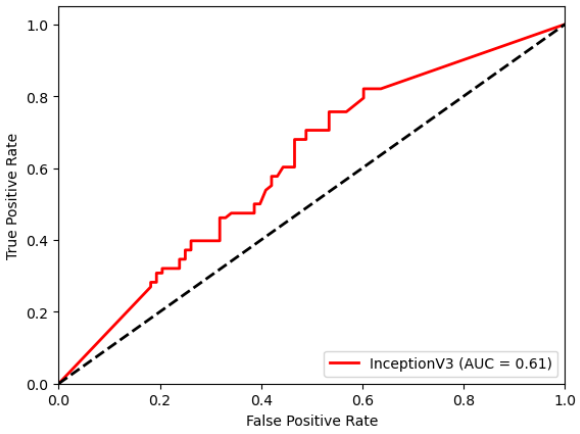
```
    2/2 [==============================] - 2s 158ms/step - loss: 0.6251 - accuracy: 0.6562
```

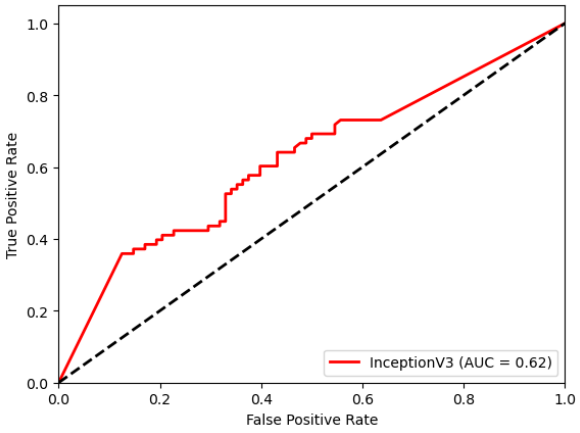
Una volta addestrato e valutato il modello, è stato convertito in un modello TFLite ed integrato in una nuova activity dell'app.

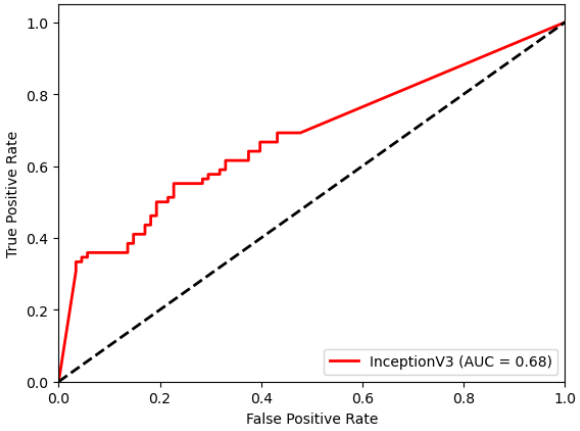
6.0.2 New Activity in Android app

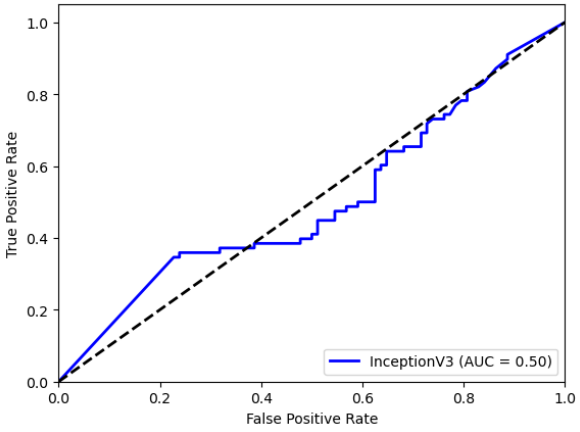
6.0.3 Risultati del nuovo modello

Come per il precedente modello le prestazioni del nuovo modello sono state valutate in termini di curve ROC, in modo da poter effettuare un confronto diretto con il precedente modello basato su ResNet50.









Dal confronto delle curve ROC, e in particolare dai valori di AUC (Area under the curve) si potrebbe dire che questo modello offre prestazioni simili e in alcuni casi anche migliori del precedente tuttavia, prove alla mano, le predizioni effettuate tramite l'app Alzheimer-Detection suggeriscono il contrario.

Bibliography

[1] Nicole Dalia Cilia, Claudio De Stefano, Francesco Fontanella, Alessandra Scotto di Freca. "An experimental Protocol to support cognitive impairment diagnosis by using hand-writing analysis". Department of Electrical and Information Engineering University of Cassino and Southern Lazio, Cassino, Italy, 2018.

[2] https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy

[3] https://it.wikipedia.org/wiki/Overfitting

[4] https://medium.com/@bravinwasike18/building-a-deep-learning-model-with-keras-and-resnet-50-9dd6f4eb3351

[5] https://keras.io/

[6] https://www.kaggle.com/code/kmader/transfer-learning-with-inceptionv3