Università degli Studi di Milano-Bicocca

Advanced Machine Learning

Final Project

RECMojion

Autori:
Alessandro Borroni - 800069 - a.borroni2@campus.unimib.it
Andrea Corvaglia - 802487 - a.corvaglia3@campus.unimib.it
Massimiliano Perletti - 847548 - m.perletti2@campus.unimib.it

17 gennaio 2020

Indice

Sommario

In questo lavoro vengono proposti una serie di modelli di Deep Learning volti ad affrontare il problema del riconoscimento delle emozioni tramite espressioni facciali. Gli esperimenti sono stati condotti su un dataset da noi realizzato, aggregando immagini acquisite via webcam e immagini wild raccolte da Google, aumentate con tecniche di Data Augmentation. Per il modello migliore, risultato in una rete convoluzionale associata ad una Feature Extraction dalla VGGFace, sono state esplorate diverse architetture, seguite da una pipe di Hyper Parameter Optimization (HPO) con metodi Sequential Model Based Optimization (SMBO).

1 Introduzione

Le emozioni mediano e facilitano le interazioni tra gli esseri umani. Pertanto, comprendere l'emozione porta spesso il contesto a una comunicazione sociale apparentemente bizzarra e/o complessa.
L'emozione può essere riconosciuta attraverso una varietà di mezzi come l'intonazione della voce, il linguaggio del corpo e metodi più complessi come l'elettroencefalografia (EEG) [1].

Tuttavia, il metodo più semplice e pratico è esaminare le espressioni facciali. Esistono sette tipi di emozioni umane riconosciute universalmente nelle diverse culture [2]: *rabbia*, *disgusto*, *paura*, *felicità*, *tristezza*, *sorpresa*, *disprezzo*. È interessante notare che, anche per espressioni complesse, in cui un mix di emozioni potrebbe essere usato come descrittore, si osserva ancora un accordo interculturale [3]. Pertanto, un'utilità che rileva le emozioni dalle espressioni facciali sarebbe ampiamente applicabile. Volendo fare un esempio, un simile progresso potrebbe portare applicazioni in medicina, marketing e intrattenimento [4].

Sin da quando i computer sono stati sviluppati, scienziati e ingegneri hanno pensato a sistemi artificialmente intelligenti che sono mentalmente e/o fisicamente equivalenti agli esseri umani.
Negli ultimi decenni, l'aumento del potere computazionale generalmente disponibile ha fornito una mano per lo sviluppo di macchine per l'apprendimento rapido, mentre Internet ha fornito un'enorme quantità di dati per l'addestramento (training). Questi due sviluppi, insieme, hanno rafforzato la ricerca sui sistemi intelligenti di autoapprendimento, con le reti neurali tra le tecniche più promettenti.

Una delle principali nonché attuali applicazioni dell'intelligenza artificiale che utilizza reti neurali è il riconoscimento di volti in foto e video. La maggior parte delle tecniche in letteratura si basa sull'elaborazione dei dati visivi e sulla successiva ricerca di schemi generali presenti nei volti umani. Il riconoscimento facciale può essere utilizzato a fini di sorveglianza dalle forze dell'ordine e nella "gestione e controllo della folla". Altre applicazioni odierne comportano la sfocatura automatica dei volti sui filmati di Google Streetview e il riconoscimento automatico degli amici di Facebook nelle foto caricate sul Social. Lo sviluppo ancora più avanzato in questo campo è il **riconoscimento delle emozioni**. Oltre a identificare solo i volti, il computer utilizza la disposizione e la forma di, per esempio, sopracciglia e labbra per determinare l'espressione facciale e quindi la relativa emozione di una persona. Una possibile applicazione di ciò si trova nell'area della sorveglianza e dell'analisi comportamentale da parte delle forze dell'ordine. Ma non solo: tali tecniche sono utilizzate nelle fotocamere digitali per scattare automaticamente foto nel momento in cui l'utente sorride.
Tuttavia, le applicazioni più promettenti comportano l'umanizzazione di sistemi artificialmente intelligenti. Se i computer sono in grado di tenere traccia dello stato mentale dell'utente, i robot possono reagire e comportarsi in modo appropriato. Il riconoscimento delle emozioni, quindi, svolge un ruolo chiave nel migliorare l'interazione uomo-macchina.

Il compito del riconoscimento delle emozioni è particolarmente complicato in quanto classificare l'emozione può essere difficile a seconda che l'immagine in ingresso sia statica o un frame di transizione in un'espressione facciale. Questo problema è particolarmente difficile per il rilevamento in tempo reale in cui le espressioni facciali variano in modo dinamico.

In questa ricerca ci si concentrerà principalmente su sistemi artificialmente intelligenti basati su reti neurali in grado di derivare l'emozione di una persona attraverso foto del suo viso. Verranno sperimentati approcci diversi dalla letteratura esistente e saranno valutati i risultati di varie scelte nel processo di progettazione.

Problema: l'intento è quello di sfruttare tecniche di riconoscimento d'immagini per individuare determinate emozioni, più precisamente sette (*arrabbiato*, *disgustato*, *felice*, *impaurito*, *neutrale*, *sorpreso*, *triste*, come rappresentato in Figura 1), analizzando le espressioni facciali presenti in alcune foto di test e di fornire, insieme all'emozione individuata nelle stesse, l'emoji in grado di riassumere al meglio tale emozione. L'idea di affrontare questo task nasce originariamente dalla volontà di rappresentare con strumenti ai giorni d'oggi ampiamente utilizzati, quali le emoji o emoticon, alcuni espressioni facciali tra le più comuni.
Per capire l'utilità di tale task, partendo da questa idea si potrebbe dar vita a un modello in grado di riconoscere una determinata espressione tramite la fotocamera interna del cellulare, immettendo la relativa emoji direttamente nel testo che si sta componendo, senza doverla necessariamente cercare tra le tante disponibili.

Figura 1: Emoji rappresentanti le emozioni prese in considerazione nel progetto

Domanda di ricerca: come possono essere utilizzate tecniche di Deep Learning per interpretare l'espressione facciale di un essere umano e carpirne l'emozione?

Approccio: per questo particolare problema si è deciso di utilizzare tecniche apprese durante il corso di Advanced Machine Learning, con l'obiettivo finale di creare un modello che fosse in grado di individuare eventuali volti presenti in alcune immagini di input, evidenziare la zona relativa al volto, riconoscere l'emozione assunta dal soggetto e associare alla stessa una emoji in grado di riassumere al meglio tale emozione. Il tutto è arricchito da diverse sperimentazioni volte a ottimizzare architettura e iper-parametri.

2 Dataset

Il dataset utilizzato è stato ottenuto grazie all'unione di tre differenti fonti di dati, in cui gli stessi dati si trovano sotto forma di immagini di diversa dimensionalità. Il metodo principale, nonché il primo a essere stato adottato, è consistito in una raccolta "manuale", attraverso l'acquisizione, tramite webcam integrata del computer portatile, di circa un centinaio di immagini per ognuna delle emozioni considerate, con l'intento di preparare la rete utilizzata al tipo di immagini che avrebbe ricevuto in input nella fase di testing. Questo metodo ha permesso di raccogliere circa 2'100 immagini in totale. In aggiunta a questa prima fonte di dati, si è deciso di ricorrere a tecniche di scraping, direttamente dal motore di ricerca Google, con il fine di integrare le immagini di partenza. Infine, si è deciso di utilizzare 144 immagini provenienti dal dataset VISGRAF, contenente volti umani di tipo animato realizzati in computer grafica, le quali immagini rappresentano sei emozioni di nostro interesse tra quelle riconosciute universalmente e citate da Paul Ekman nell'articolo summenzionato.

2.1 RECData (acquisizione delle immagini)

Per l'acquisizione si è dato vita a dei codici scritti in linguaggio Python con il fine di automatizzare il tutto. Tale script si appoggia sulla libreria OpenCV, molto diffusa in ambito di Computer Vision e

gestione delle immagini.
L'acquisizione effettiva avviene sfruttando la webcam integrata del PC portatile su cui lo script viene lanciato. Il processo prevede una unica sessione, corrispondente a un singolo lancio del codice, in cui viene scattato un numero prestabilito di foto senza interruzioni. Per evitare di acquisire immagini mosse, vista la scarsa risoluzione della camera utilizzata, è stato stabilito un time-sleep di mezzo secondo, il che significa che lo scatto avviene ogni mezzo secondo circa (bisogna considerare un ritardo quasi trascurabile dovuto all'esecuzione del codice stesso).
Tutte le foto vengono salvate nelle relative cartelle (una per ognuna delle emozioni considerate, tutte raccolte in una macro-cartella denominata "raw_images"), così da facilitare l'assegnazione delle label, corrispondenti al nome delle cartelle stesse. Le immagini ivi raccolte hanno una dimensionalità di 1280x720 pixel, come specificato dalla funzione creata in questo script, e sfruttano la scala colore Red-Green-Blue (RGB).

2.2 Scraping (integrazione)

Il dataset acquisito è stato arricchito con uno scraping da Google immagini, in particolare sono state usate delle queries che comprendessero sia sinonimi delle emozioni considerati (es. mad per angry) sia diverse espressioni per il tipo di immagini considerate (es. portrait, close-up). Questo enfatizzando che le immagini ritraessero persone scrivendo "human face" e sottraendo alla query riferimenti a disegni,loghi, etc. Dopo alcune queries per ogni emozione le immagini sono state esaminate a mano per verificare la loro appropriatezza. Si sottolinea il fatto che è stata selezionata la dimensione delle immagini in modo che fossero maggiori del target, il quale prevedeva immagini di dimensione 224x224x3 (pixel, pixel, canali). Lo scraping è stato effettuato con l'ausilio del pacchetto google_images_download. Esempio di una query: "angry mad upset human face portrait close-up -cartoon -drawing -logo -emoji".

2.3 VISGRAF (integrazione)

Il database VISGRAF (o IMPA-FACE3D) è stato creato nel 2008 per aiutare nella ricerca sull'animazione facciale. In particolare, per l'analisi e la sintesi di volti ed espressioni. A tale scopo, è stato preso, come base, il volto neutro (cioè il volto con la posizione della fotocamera frontale di zero gradi e senza espressione facciale) e le sei espressioni universalmente riconosciute proposte da Ekman: felicità, tristezza, sorpresa, rabbia, disgusto, e paura (il disprezzo è escluso). La caratteristica principale di questo set di dati è una registrazione delle informazioni geometriche con il colore (ovvero, la geometria e la trama sono correlate).
Questo set di dati include acquisizioni di 38 individui con un campione di viso neutro, campioni corrispondenti a sei espressioni facciali universali e altre espressioni che si riferiscono a 5 campioni contenenti bocca e occhi aperti e/o chiusi. Inoltre, sono stati considerati due campioni corrispondenti ai profili laterali degli individui. Complessivamente, il set di dati è composto da 22 uomini e 16 donne, con la maggior parte delle persone di età compresa tra 20 e 50 anni. Sono stati acquisiti 14 campioni per tutti gli individui, riassumendo in totale 532 campioni. Di questi, tuttavia, ne sono stati utilizzati solo 144, in quanto più idonei allo scopo del progetto raccontato in questo report.

2.4 Cropping delle immagini

Per tutte le foto facenti parte del dataset finale si è deciso di optare per un ritaglio delle stesse in modo da ricondurle a una dimensionalità consona per le future elaborazioni. Si è scelto di ridurle a un formato quadrato di 224x224 pixel.
Per fare ciò è stato implementato uno script, utilizzando Python come linguaggio di programmazione, il quale si occupava di ricevere in input tutte le immagini raccolte nella cartella "raw_images", rilevare la presenza di eventuali volti nelle stesse (grazie a un classificatore utilizzato appositamente per rilevare l'oggetto per il quale è stato addestrato, in questo caso volti frontali), e, seguendo il bounding box re-

stituito tramite questo processo, ritagliare il volto restituendo un immagine quadrata della dimensione specificata. Le immagini vengono mantenute su tre canali anche in questa fase finale (RGB).

2.5 Data Augmentation

La Data Augmentation è un processo volto ad aumentare la quantità e la diversità dei dati. Il concetto basilare consiste nel non dover raccogliere nuovi dati, quanto nel trasformare quelli già presenti.
Per questo progetto si è scelto di utilizzare la libreria albumentations, la quale è una tra le più veloci librerie di potenziamento dell'immagine.

Le immagini, solo ed esclusivamente quelle di training, vengono passate in input a una funzione che si occupa di quadruplicarle. Queste foto subiscono delle trasformazioni secondo la seguente distribuzione di probabilità:

- 50% che venga ruotata (di 60° oppure di -60°);
- 50% che avvenga uno shift dei canali RGB;
- 50% che subisca una variazione in contrasto e/o luminosità;
- 50% che l'immagine sia sottoposta a una tra le seguenti trasformazioni: sfocatura, sfocatura in movimento, rumore Gaussiano)

Essendo il dataset finale a disposizione non eccessivamente grande, si è deciso di optare per questa tecnica al fine di ottenere una generalizzazione nonostante la scarsità di dati e specialmente per ridurre il rischio di overfitting nella fase di training.

3 Approccio Metodologico

3.1 Model Selection

Per poter affrontare il task si è effettuata una comparazione di diversi modelli, basati su tecniche di *Transfer Learning*. Questo per una duplice ragione: da una parte la disponibilità di reti molto performanti per il riconoscimento facciale, dall'altra il dataset di piccole dimensioni.

È stata selezionata la rete VGGFace, la quale sfrutta delle architetture originali come VGG16, ResNet50 e Senet50, addestrate, però, sul dataset *VGGFace* [5] invece che su *ImageNet*.
Da questi pesi, i quali sono stati calcolati sull'addestramento di numerosi volti, invece che su classi di animali, oggetti et similia, ci si può aspettare una performance migliore per il task preso in considerazione nel progetto.

3.1.1 Senet50 (Fine-Tuning)

Il primo metodo considerato è stato il *Fine Tuning*, nel quale viene sfruttata una rete addestrata su una base di dati molto grande e che quindi possiede già dei pesi efficaci per il task per cui è stata allenata. Questi si possono sfruttare come punto di partenza in un modello finale, il quale viene addestrato, invece, sui dati in nostro possesso, così da specializzare la rete per il nuovo task.

Si è dunque importata la rete VggFace (con versione Senet50), tagliando gli ultimi layers *fully connected* e andando a bloccare i pesi fino alla coda della rete (nello specifico fino al layer *"conv5_3_1x1_reduce/bn"*) per poter escludere la maggior parte dei pesi dal nuovo addestramento. Al contrario, gli ultimi layers rimasti "sbloccati" andranno a contribuire nell'addestramento della nostra rete, partecipando nel calcolo dei nuovi pesi sul nostro dataset. In coda alla VGGFace, sono stati applicati dei layers *Dense*, in totale tre (rispettivamente da 512, 256 e 32 neuroni), intervallati da livelli di *Dropout* (con rate 0.5) per limitare l'overfitting. Segue, infine, un livello di output con il rispettivo layer dense (con 7 neuroni, pari alle classi da predire) e attivazione *Softmax*.
Si è scelto l'Adaptive Moment Estimation (Adam) come ottimizzatore perché è una forma di SGD con

un learning rate adattivo, ma che vi accompagna anche una componente che tiene conto dei gradienti precedentemente calcolati (momento) a differenza di altri algoritmi come Adadelta e RMSprop. Vista la scelta dell'ottimizzatore si è utilizzato il learning rate di default come consigliato sulle references di keras. La funzione di Loss scelta è la categorical-crossentropy, in quanto classificazione multi-labels.

Sono stati così raggiunti circa 27 milioni di parametri, di cui solo 5 milioni addestrabili; questo comporterà un fitting del modello sicuramente più lento rispetto alle successive architetture selezionate. Infine, in fase di training, viene usata come callback il ModelCheckpoint, per poter caricare i pesi migliori ottenuti durante l'addestramento e valutarli poi sul test set. Tutta la prova è eseguita in Cross-Validation (5 folds) per poter essere più robusta rispetto ad un semplice Hold-Out e di conseguenza meno dipendente rispetto al particolare set su cui viene addestrato il modello.

3.1.2 Rete Neurale Convoluzionale (CNN)

L'idea iniziale è stata quella di creare un'alternativa al Fine Tuning, che permettesse di usare le features a medio-alto livello di una rete molto grande come *VGGFace*, ma senza allenare i layers rimanenti. Questo perché, data la loro numerosità, si avrebbe un alto numero di parametri difficilmente modificabili da un training su un dataset creato da noi e dunque di dimensioni ridotte.

La filosofia di base per la rete è stata quella di mantenere molto basso il numero di parametri, visto che parte del lavoro di interpretazione dell'immagine avviene nella Feature Extraction, così da poter spingere di più l'ottimizzazione.

L'architettura è una rivisitazione del modello VGG16, ma ridimensionata con il fine di mantenere basso il numero di parametri. Allo stesso modo, però, si è puntato sulla profondità del modello. Quest'ultimo è formato da un blocco di convoluzioni, con tre layers convoluzionali, nei quali sono stati usati filtri molto piccoli (3x3, e cioè la dimensione più piccola possibile, che cattura ancora sinistra/destra e su/giù). A questi sono stati aggiunti anche filtri di convoluzione 1x1 che agiscono come una trasformazione lineare dell'input. Lo stride è fissato a 1 pixel (valore di default) in modo che la risoluzione spaziale venga preservata dopo la convoluzione.
In particolare, i tre filtri di ogni blocco sono in sequenza 1x1, 3x3 e 1x1. In questo modo si ottiene un modello abbastanza profondo da avere una consistente non linearità fermo restante il basso numero di parametri. Questa non linearità è raggiunta aggiungendo ad ogni layer, ad eccezione dell'output layer, una funzione di attivazione non lineare, in particolare una *ReLU*.
Inoltre si è impiegato un MaxPooling alla fine del blocco di convoluzioni in modo da ridurre la dimensionalità dell'input e di conseguenza anche il numero dei parametri stessi nella parte finale della rete (Fully-Connected) ridotta ad un singolo hidden layer (210 neuroni) più l'output layer necessariamente da 7 (numero di classi).

Come ottimizzatore si è usato l'Adaptive Moment Estimation (Adam), con Learning Rate impostato al valore di default. Per quanto riguarda il numero di epoche, questo è stato scelto in modo da attivare l'*Early Stopping* basato sulla Loss function calcolata sul validation set. La dimensione del batch (128) invece è stata scelta abbastanza grande da avere un andamento smussato della curva di Loss e Accuracy, ma non così tanto da perdere la regolarizzazione data da un gradiente approssimato. La funzione di Loss scelta è la categorical-crossentropy. Il problema è di classificazione multi-classe e dunque la funzione di output usata è stata la *Softmax*. Infine come forma di regolarizzazione per evitare l'overfitting si è usato un *Dropout* tra i due layer densi e un *L2 Regularizer*.

3.1.3 Classificatori Tradizionali

Sono stati considerati tre diversi classificatori:

- Support Vector Classifier (SVC)
- Random Forest
- K-Nearest Neighbors

Ogni classificatore è stato configurato con i parametri di default implementati da scikit-learn. Per quanto riguarda il taglio della rete, questo è stato scelto in una posizione simile a quella adoperata per i precedenti modelli, in particolare al più vicino AvgPooling precedente (*global_average_pooling2d_13*), scelto perché corrispondente a una riduzione di dimensionalità dell'output che permette di eseguire il training dei modelli in tempi contenuti senza l'utilizzo di ulteriori tecniche, come per esempio una Principal Component Analysis (PCA).

3.2 Selezione Architettura

In seguito alla scelta del modello migliore (CNN), si è proceduto con l'ottimizzazione della Feature Extraction per quel modello, ovvero al confronto delle performance dello stesso su varie scelte di taglio sulla *VGGFace*. Inizialmente si sono valutati tagli coerenti, ovvero su layers dello stesso tipo (tutti Activation) in diverse regioni della rete con il fine di identificare la regione migliore in cui continuare a cercare. In ogni caso, però, non è stato possibile valutare in questa prima fase quale fosse la regione ottimale, ma solo un'area abbastanza estesa di ricerca, ovvero dalla fine del blocco conv3, fino alla coda della rete. Entro questa area è stato valutato ogni possibile taglio. È possibile vedere i risultati nella Figura 4, nel quale si nota una ciclicità nelle valutazioni, probabilmente dovuta alle diverse dimensioni degli output, a indicare che i più efficaci per l'estrazione sono i layer di attivazione. Al tempo stesso, si nota come un eccessivo avvicinamento alla parte finale della rete provochi una riduzione dell'Accuracy. Effetto causato anche da un taglio effettuato più indietro nella rete stessa.

Selezionato il taglio migliore si è proceduto con una ristretta Architecture Selection, sostanzialmente sono state confrontate nove architetture: tre diverse scelte del numero dei layer in convoluzione e tre possibilità per i layer densi, in Figura 2 è riportata la sezione di codice per la creazione dei diversi modelli. In particolare per quanto riguarda il blocco convoluzionale: C1 rappresenta la rete con i primi tre layer Conv2D, C2 aggiunge a questo un MaxPooling e un layer Conv2D e C3 aggiunge due layer Conv2D. Per quanto riguarda i layer densi, invece, ogni D aggiunge un layer Dense e un layer di Dropout. Si possono vedere i risultati nella tabella in Figura 5 e nel grafico in Figura 6.

È opportuno approfondire le modalità con le quali sono stati confrontati di volta in volta i modelli. La prima selezione tra Senet50, CNN e modelli tradizionali è avvenuta con un *K-Fold Cross Validation* a 5 fold implementata con *Scikit-Learn*; per quanto riguarda la selezione dei tagli, i modelli sono stati confrontati ripetendo tre prove per ogni training, per ridurre la variabilità dovuta all'inizializzazione randomica dei pesi.
Dalla selezione dell'architettura in poi è stata introdotta la Data Augmentation nel training. Per coniugare la Data Augmentation che deve avvenire solo sul training con una Cross Validation che fosse abbastanza veloce per eseguire qualche prova nella sezione di ottimizzazione finale, è stata adottata una strategia che seppur a discapito della RAM permette di eseguire la valutazione velocemente. Sostanzialmente è stato diviso il dataset in tre punti come una classica *3-Fold Cross Validation*, ma invece di eseguire tre volte split, Data Augmentation e successiva Feature Extraction ad ogni valutazione di un modello, sono stati salvati in memoria tre fold composti da training, validation e test set, con training aumentato e a Feature Extraction già avvenuta.

```python
def create_model(C=1,D=1):
'''
Funzione che prende in input il livello di profondità per il blocco convoluzionale "C" e per il blocco denso "D"
e restituisce il modello già compilato
'''
  # Create model
  model = Sequential()
# C
  # C1
  model.add(Conv2D(32, kernel_size=1, activation='relu', input_shape=(14, 14, 1024)))
  model.add(Conv2D(128, kernel_size=3, activation='relu'))
  model.add(Conv2D(32, kernel_size=1, activation='relu'))
  # C2
  if(C>1):
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32, kernel_size=1, activation='relu'))
  # C3
  if(C>2):
    model.add(Conv2D(32, kernel_size=3, activation='relu'))
    model.add(Conv2D(32, kernel_size=1, activation='relu'))

  model.add(Flatten())
# D
  # D1
  model.add(Dense(21, activation='relu',kernel_regularizer=regularizers.l2(0.01)))
  model.add(Dropout(0.5))
  # D2
  if(D>1):
    model.add(Dense(56, activation='relu',kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.3))
  # D3
  if(D>2):
    model.add(Dense(112, activation='relu',kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.3))

  model.add(Dense(7, activation='softmax'))

  # Compile model
  model.compile(loss=keras.losses.categorical_crossentropy,
                optimizer=keras.optimizers.Adam(),
                metrics=['acc'])
  return model
```

Figura 2: Funzione per la creazione dei modelli con le architetture da confrontare. La funzione prende in input il valore di C (profondità blocco convoluzionale) e il valore di D (profondità del blocco denso), i quali vanno da 1 a 3 e corrispondono alla profondità del modello. Come si nota dallo screen la funzione aggiunge al modello base layers in funzione dei valori di C e D scelti.

3.3 Ottimizzazione

Definita l'architettura del nostro modello, costituito da un solo blocco convoluzionale (3 layers Conv2D) e un solo layer Dense (oltre a quello finale di output), è stata definita una fase di ottimizzazione degli iper-parametri.

Per raggiungere questo obiettivo si è deciso di utilizzare il framework *Optuna*, compatibile con l'implementazione del nostro modello costruita in Keras. Si è deciso per questa soluzione per diversi motivi, tra i quali la possibilità, tramite una funzione di integrazione, di sfruttare la flessibilità e la potenza dell'ottimizzatore di *skopt* (libreria che permette di minimizzare funzioni molto costose, di scikit-learn) e la semplicità di sintassi con cui programmare lo spazio di ricerca. Questa libreria possiede molte altre caratteristiche vantaggiose, come un'implementazione di un processo di *Pruning*, che è una tecnica utile a ridurre lo spazio di ricerca, in quanto valuta alla prima epoca il valore della funzione obiettivo, e nel caso questo non sia potenzialmente valido, viene tagliato immediatamente rendendo così molto più rapido il processo. Nativamente possiede molte funzioni per realizzare grafici utili a interpretare la scelta dei parametri e assicurarsi così che si stia compiendo un'efficace combinazione tra exploration ed exploitation, interpretando dai grafici l'andamento dell'ottimizzazione.

Come *Sampler*, ossia il campionatore del set dei parametri di ricerca, si è usato il Tree-structured Parzen Estimator (TPE), che si interfaccia con l'integrazione in *skopt*, dove è stato selezionato come modello surrogato un Random Forest (RF), e come funzione di acquisizione che valuti i punti scelti dal surrogato il Lower Confidence Bound (LCB); questo perché tra i vari tentativi fatti è sembrata la configurazione che ottenesse dei minimi con un minor budget (numero di trial). Va sottolineato che il modello RF, per come è strutturato, è la scelta più naturale per la gestione di parametri categorici.

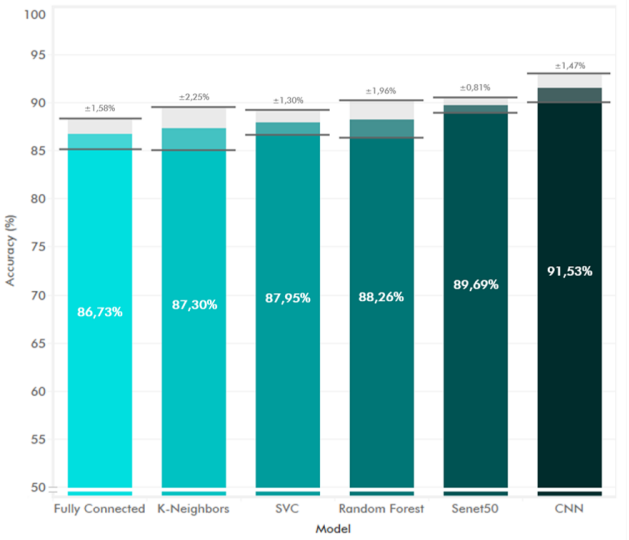
Come spazio di ricerca per i parametri, si è concentrata l'ottimizzazione sui seguenti:

- attivazione dei layers Conv2D e Dense (escluso output): "$ReLU$" oppure "$LeakyReLU$"
- numero dei filtri ney layers Conv2D: range di interi [32, 256]
- grandezza dei filtri (*kernels-size*): range di interi [1, 5]
- numero di neuroni nel layer Dense: range di interi [21, 490]
- rate del layer di Dropout: range [0, 1, con passo 0.1]
- il tipo di ottimizzatore: "$Adam$", "$rmsprop$" oppure "$nadam$"
- learning rate dell'ottimizzatore: un valore nel range continuo (1e-6, 1e-2]

Deciso lo spazio di ricerca, è stata definita la funzione obiettivo, la quale prevede la minimizzazione del valore complementare all'Accuracy (Errore). Le valutazioni sono state fatte in una Cross-Validation (3-fold) per limitare la randomicità nell'ottimizzazione. Infine il budget è stato impostato a 100 trial. Il processo di ottimizzazione è stato accompagnato da un collegamento ad un database SQLite che ha permesso di avere uno storico dei trials effettuati e dunque anche un backup per riprendere l'ottimizzazione in un secondo momento. Inoltre questo tipo di struttura permette una gestione distribuita del processo. Nel grafico in Figura 8 si vede l'andamento del Best Seen durante l'ottimizzazione e nel grafico in Figura 7, invece, è mostrata una ColorMap di esempio sulla distribuzione della funzione obiettivo al variare delle variabili indicate. In seguito alla ottimizzazione abbiamo modificato la funzione di attivazione con una LeakyReLU, abbiamo aumentato il numero di neuroni (346) e il numero di filtri (191), nonché la dimensione del kernel (4) ed è stato confermato come ottimizzatore l'Adam, ma con Learning Rate di $1.3e^{-}5$.

4 Risultati e Valutazioni

4.1 Selezione del miglior Modello



Model	Accuracy	St. Deviation
CNN	**91,53%**	1,47%
Fully Connected	86,73%	1,58%
K-Neighbors	87,30%	2,25%
Random Forest	88,26%	1,96%
Senet50	89,69%	**0,81%**
SVC	87,95%	1,30%

Figura 3: Barplot e tabella con le Accuracy degli algoritmi considerati.

4.2 Selezione del miglior Taglio

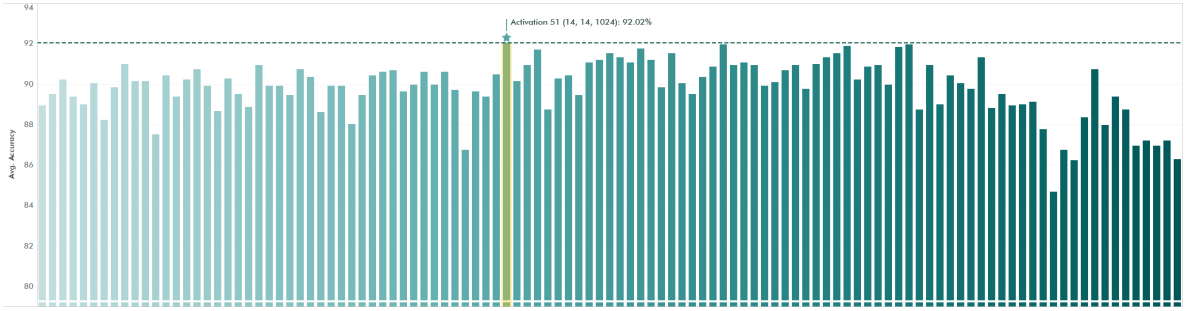


Figura 4: Barplot con le Accuracy dei tagli considerati. Si noti che sono rappresentati solo i layers effettivamente valutati e questo esclude quelli con output troppo piccolo per entrare in input alla CNN implementata nel progetto.

4.3 Selezione della migliore Architettura

Model	Accuracy	St. Dev. Acc.	Loss	St. Dev. Loss
C1-D1	**90,39%**	0,53%	**0,53**	0,03
C1-D2	89,92%	**0,09%**	0,61	0,10
C1-D3	88,11%	2,34%	0,76	0,07
C2-D1	89,73%	0,29%	0,62	0,06
C2-D2	90,35%	0,76%	0,55	0,08
C2-D3	88,73%	1,49%	0,73	0,15
C3-D1	90,04%	1,36%	0,74	0,17
C3-D2	89,85%	0,62%	0,71	**0,01**
C3-D3	87,92%	0,60%	0,76	0,12

Figura 5: Tabella con le Accuracy delle architetture considerate.

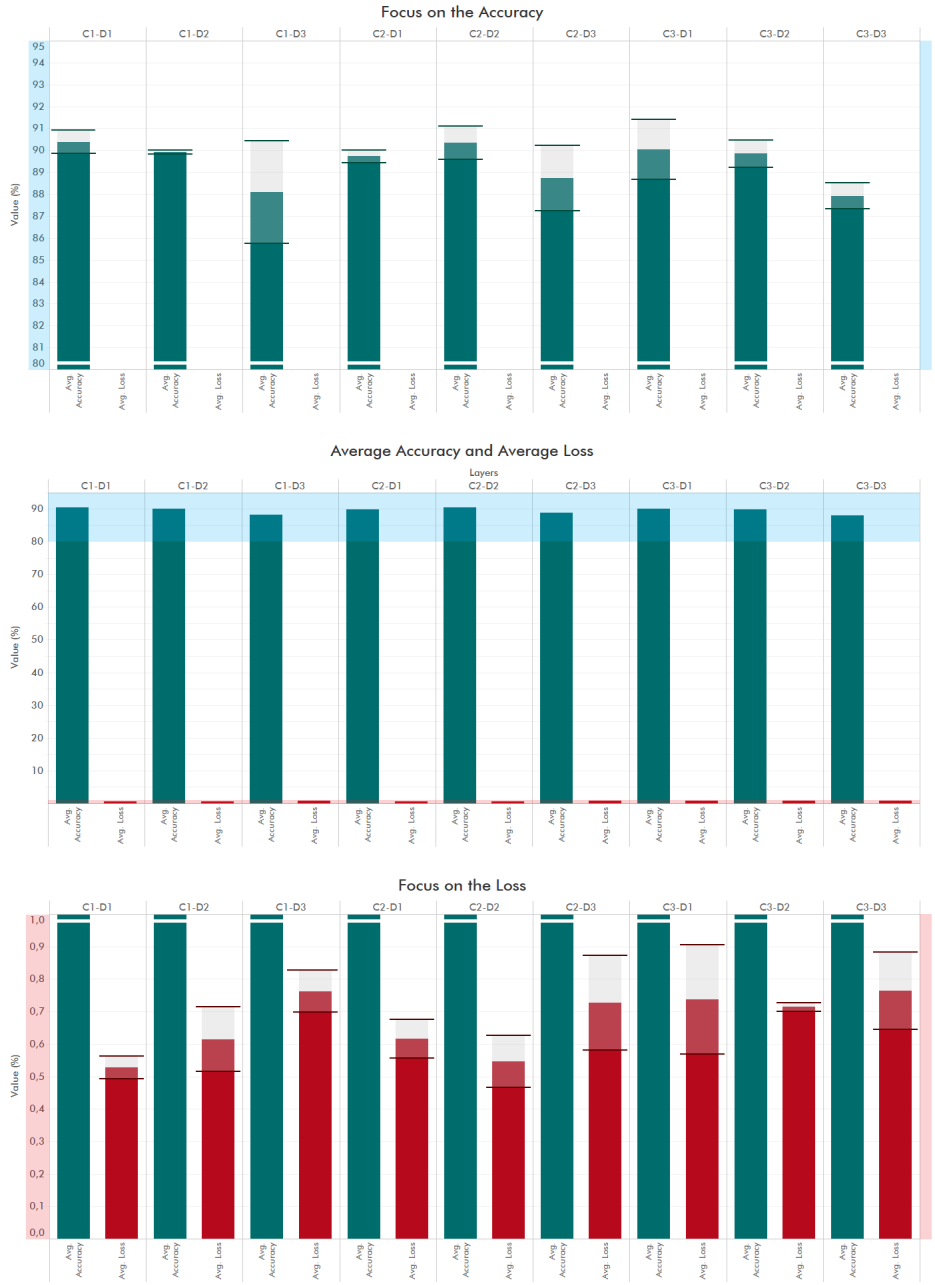


Figura 6: Barplot con le Accuracy delle architetture considerate. In particolare il primo grafico mostra solo le Accuracy, essendo l'asse verticale tagliato per enfatizzare le differenze tra le architetture e rendere comprensibili gli intervalli di confidenza evidenziati in grigio. Il secondo grafico mostra Loss e Accuracy a range intero, mentre il terzo taglia l'asse verticale a favore, questa volta ,della comprensibilità della Loss e dei suoi rispettivi intervalli di confidenza.

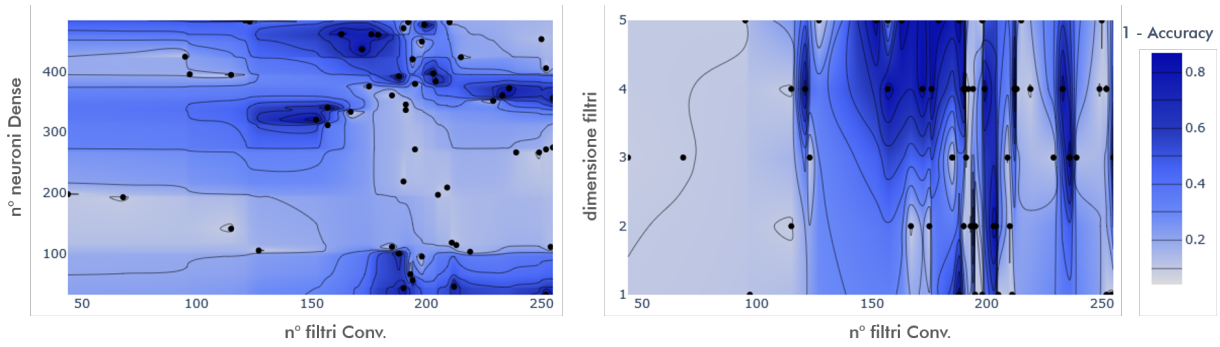


Figura 7: ColorMap che rappresenta il valore dell'Errore (1 - Accuracy) a coppie di iper-parametri. Rispettivamente filtri vs. neuroni e filtri vs. dim. filtri. Le zone in blu più scuro rappresentano valori elevati di errore e dunque configurazioni meno desiderabili.

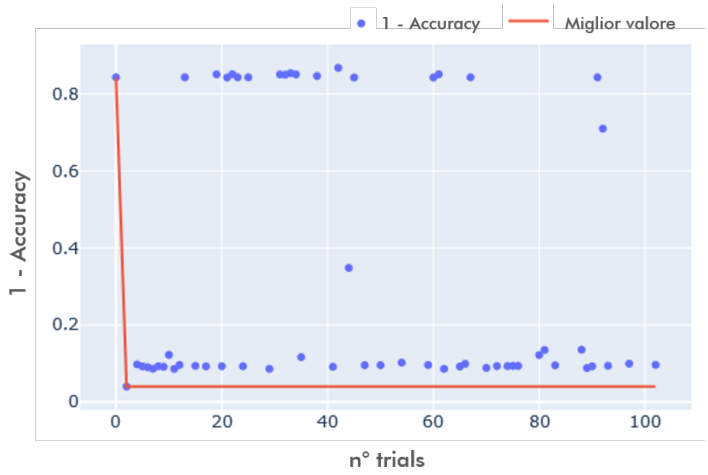


Figura 8: Grafico della curva di Best Seen per il processo di ottimizzazione. Sono rappresentati anche i valori di Accuracy di ogni trial in blu.

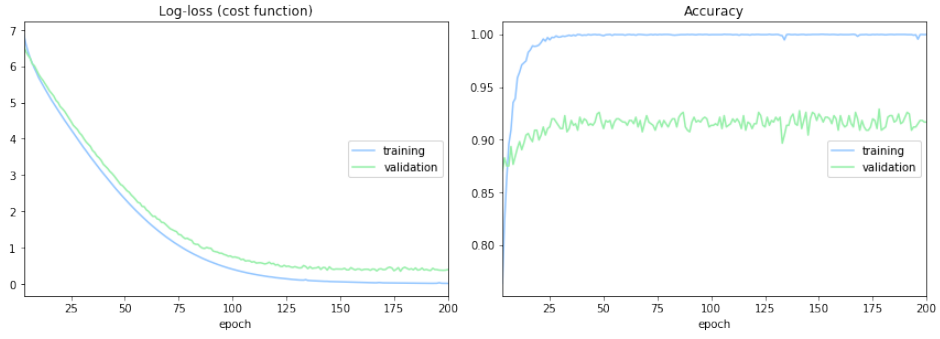


Figura 9: Nel Grafico è rappresentata la curva di Loss e Accuracy per il training set e il validation del modello finale.

5 Discussione dei Risultati

L'obiettivo di questo progetto era implementare il riconoscimento delle emozioni facciali, anche in real-time. Il miglior modello risulta essere la CNN con un'Accuracy del 91.53% (Figura 3), che sale a 92.02% (Figura 4) nell'ottimizzazione del taglio per la Feature Extraction. Il valore scende a 89.73% per via della Data Augmentation, la quale però riduce l'overfitting, probabile nel nostro dataset piccolo e su pochi soggetti. In ogni caso con l'architettura migliore sale a 90.39% (Figura 5), a significare che probabilmente il *Max-Pooling* riduceva troppo la dimensione dei dati. Il modello finale è stato trainato su tutti i dati e in validation ottiene circa 92% (Figura 9).
Possiamo quindi ritenerci soddisfatti dello score del modello, fermo restando la consapevolezza che il tutto è stato testato in un contesto controllato, con una certa monotonia delle condizioni di acquisizione delle immagini e dei soggetti ripresi. Per ottenere un'implementazione di maggiore successo sarebbe necessario progettare un set di dati molto più ampio per migliorare in questo modo la generalità del modello, e riconoscere più facilmente emozioni in condizioni non controllate di luce, distanza, orientamento e simili.

6 Conclusioni

Il lavoro ha portato ad una applicazione per il riconoscimento di emozioni a partire da espressioni facciali, anche in real-time. La pipeline di ottimizzazione ha portato a un miglioramento delle performance, tale da ritenere i risultati soddisfacenti.
Futuri sviluppi dovrebbero focalizzarsi sulla costruzione di un dataset più corposo, atto a comprendere più soggetti nelle condizioni più disparate.

Riferimenti bibliografici

[1] P. Abhang, S. Rao, B. W. Gawali, and P. Rokade (2011) *"Emotion recognition using speech and eeg signal a review"*, International Journal of Computer Applications, vol. 15, pp. 37–40

[2] P. Ekman (1971) *"Universals and cultural differences in facial expressions of emotion"*, Lincoln University of Nebraska Press, Nebraska, USA

[3] P. Ekman (1971) *"Universals and cultural differences in facial expressions of emotion"*, Lincoln University of Nebraska Press, Nebraska, USA

[4] A. Kołakowska, A. Landowska, M. Szwoch, W. Szwoch, and M. R. Wróbel (2014) *"Emotion Recognition and Its Applications"*, Cham: Springer International Publishing, pp. 51–62

[5] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, A. Zisserman (2018) *"VGGFace2: A dataset for recognising face across pose and age"*, International Conference on Automatic Face and Gesture Recognition

A Appendix: RECMojion Live Demo

Utilizzando la rete neurale appositamente addestrata con un rilevatore di volti fornito da OpenCV, abbiamo implementato con successo una Demo in cui un'emoji indicante una delle sette emozioni considerate (angry, disgust, fear, happy, neutral, sad, surprise) si accosta al volto di un utente, in tempo reale. Con l'uso del suddetto meccanismo di riconoscimento e detenzione dei volti, il volto che appare nel video viene monitorato, estratto e ridimensionato in un'immagine di dimensione 224x224. I dati così ottenuti vengono quindi inviati all'input del modello di rete neurale, il quale restituirà il nome dell'etichetta relativa (o emozione) con al suo fianco un punteggio da 0 a 1.0 esprimente l'Accuracy della classificazione. Questo output verrà collocato in alto a sinistra, sopra il bounding box racchiudente il volto.

B Appendix: Codice e Dati

L'intera struttura del progetto si è sviluppata su una repository GitHub. Pertanto, il codice completo e i risultati forniti sono disponibili all'indirizzo: `https://github.com/malborroni/RECMojion`. Si può fare riferimento al README.md della repository per ottenere informazioni sulla struttura della stessa.