SAPIENZA
UNIVERSITÀ DI ROMA

Università La Sapienza, INFORMATICA

A.A. 2024 / 2025

Progetto AI Lab: Computer Vision and NLP

“Emotion Recognition & Genuineness Detection”

Emanuele Errante, 2023244

Luca Micarelli, 2061752

Introduzione

In questo progetto abbiamo deciso di esplorare due topic presenti nel corso computer vision e machine learning, applicandoli a un ambito molto interessante, l'analisi delle espressioni facciali.

Ci ha interessato molto, perché è un campo di studio interdisciplinare che applica concetti teorici di psicologia cognitiva e neuroscienze con tecniche avanzate di intelligenza artificiale e machine learning.

È inoltre un campo di ricerca che trova applicazione in numerosi scenari reali, come l'interazione uomo-macchina, la sorveglianza intelligente e supporto in ambito psicologico e psichiatrico.

Ci siamo anche imbattuti in questo paper: https://arxiv.org/pdf/1804.08348 che descrive alcuni dei progressi fatti in questo ambito spiegando come le reti neurali sono state utilizzate per migliorare l'accuratezza e la robustezza dei sistemi FER e dà un overview dei dataset più comuni, insieme ad alcune tecniche utilizzate, per pre processare le immagini e generalizzare al meglio i modelli durante il training.

Abbiamo scelto di esplorare due task diversi, ma complementari:

- **Emotion Recognition**, ovvero il riconoscimento delle emozioni espresse da un volto, come gioia, tristezza, rabbia, sorpresa, paura, disgusto o uno stato neutro, per questa task abbiamo utilizzato tecniche di computer vision e un modello di machine learning per classificare le espressioni del volto in una delle categorie predefinite.

- **Genuineness Detection**, ovvero analizzare l'autenticità delle emozioni espresse, facendo una distinzione tra le espressioni spontanee (genuine) e quelle simulate (non genuine) questo richiede l'analisi delle micro espressioni facciali, che ci permettono di analizzare i muscoli coinvolti

Proprio per questo motivo abbiamo strutturato e diviso il nostro progetto in due parti, ognuna per la rispettiva task, abbiamo poi integrato un main finale che utilizza entrambi i modelli per ottenere la previsione riguardo entrambe le task.

Abbiamo quindi due rispettive sottocartelle con il codice contenente modello, dataset e loop per il training / testing, ognuna per la rispettiva task, nella parte genuineness detection c'è anche una cartella dataset_src che contiene il codice utilizzato per scaricare e creare i dataset nel formato .csv

Tecnologie e Librerie Utilizzate

- **PyTorch**: Framework di deep learning per la costruzione, l'addestramento e l'inferenza della CNN.
- **OpenCV (cv2)**: Libreria fondamentale per la visione artificiale, utilizzata per la cattura video da webcam, il rilevamento dei volti (con classificatori Haar Cascade) e la manipolazione delle immagini.
- **NumPy (np)**: Libreria per operazioni numeriche ad alte prestazioni, essenziale per la manipolazione degli array di pixel delle immagini.
- **Pandas (pd)**: Utilizzata per la manipolazione del dataset CSV (FER-2013).
- **Torchvision transforms**: Modulo di PyTorch che fornisce trasformazioni comuni per le immagini, cruciali per la pre-elaborazione e l'aumento dei dati.

- **Matplotlib (plt)**: Utilizzata per la visualizzazione dei grafici delle metriche di addestramento, permettendo di analizzare l'andamento di loss e accuratezza.
- **tqdm**: Per visualizzare barre di progresso durante l'addestramento, migliorando l'esperienza utente.
- **OpenFace**: Tool esterno open source utilizzato per estrarre gli Action Units (AUs) dalle immagini facciali.

Emotion Recognition

Le espressioni facciali sono una delle forme più dirette e universali di comunicazione non verbale. Tuttavia, la loro interpretazione automatica presenta sfide significative. Le variazioni intra-classe (ad esempio, diverse intensità della stessa emozione) e le somiglianze inter-classe (espressioni diverse che possono apparire simili) rendono il compito complesso. Inoltre, fattori come l'illuminazione, la posa del volto, l'occlusione parziale e le differenze individuali nella morfologia facciale contribuiscono alla difficoltà.

Negli ultimi anni, l'avvento e la maturazione delle tecniche di **Deep Learning**, in particolare delle Reti Neurali Convoluzionali (CNN), hanno rivoluzionato il campo della visione artificiale, portando a progressi straordinari nel riconoscimento di pattern complessi direttamente dai dati grezzi. Le CNN sono intrinsecamente capaci di apprendere gerarchie di caratteristiche visive, partendo da bordi e texture a basso livello fino a forme e oggetti più complessi ad alto livello, rendendole particolarmente adatte per compiti come il riconoscimento facciale e delle espressioni.

Il presente progetto si inserisce in questo contesto, proponendosi di sviluppare un sistema robusto per il riconoscimento delle emozioni facciali umane. L'obiettivo primario è quello di addestrare un modello di Deep Learning, basato su una CNN, capace di classificare le espressioni facciali in una delle sette emozioni fondamentali: Rabbia (Angry), Disgusto (Disgust), Paura (Fear), Felicità (Happy), Tristezza (Sad), Sorpresa (Surprise) e Neutralità (Neutral). Per l'addestramento e la valutazione di questo modello, viene utilizzato il **dataset FER-2013** (Facial Expression Recognition 2013), ampiamente riconosciuto nella comunità di ricerca per la sua varietà e complessità.

La scelta di PyTorch come framework di riferimento è dettata dalla sua flessibilità, dalla sua natura "Pythonic" che facilita lo sviluppo rapido e l'iterazione, e dalla sua efficienza nell'esecuzione su hardware accelerato (GPU). L'integrazione con OpenCV consente la cattura e la pre-elaborazione delle immagini in tempo reale da una webcam, trasformando il sistema addestrato in uno strumento interattivo capace di analizzare le espressioni facciali in tempo reale.

Architettura del Task di Emotion_Recognition

Questo task del progetto è strutturato in diversi moduli Python, ognuno con una responsabilità specifica, garantendo modularità e manutenibilità del codice.

- main_emotion_recognition.py: Questo è il punto di ingresso principale per l'addestramento del modello di riconoscimento delle emozioni. Gestisce la configurazione dell'ambiente, il caricamento dei dati, l'inizializzazione del modello, l'avvio del processo di addestramento e la valutazione delle prestazioni finali.
- data_loader.py: Si occupa del caricamento e della pre-elaborazione del dataset FER-2013.
- model.py: Definisce l'architettura della rete neurale convoluzionale (CNN) utilizzata per la classificazione delle emozioni.
- train.py: Contiene la logica per il ciclo di addestramento e validazione del modello.

- utils.py: Fornisce funzioni di utilità per il salvataggio/caricamento dei checkpoint del modello e la visualizzazione delle metriche di addestramento.

Funzioni e Tecniche Principali

Dataset e Pre-elaborazione

Il cuore del riconoscimento delle emozioni è il dataset **FER-2013** (Facial Expression Recognition 2013). Questo dataset è composto da immagini in scala di grigi di volti umani, etichettate con una delle sette categorie di emozioni.

Il modulo data_loader.py implementa la classe FER2013Dataset che gestisce il caricamento del dataset da un file CSV (fer2013.csv). Le immagini sono rappresentate come stringhe di pixel e vengono convertite in array NumPy 48x48 pixel.

Per preparare le immagini all'input della CNN (la rete neurale convoluzionale), vengono applicate diverse trasformazioni tramite torchvision.transforms:

- **Per l'Addestramento (train_transforms)**:
    - **Conversione in immagine PIL**: Necessaria per l'applicazione delle trasformazioni di torchvision.
    - **Ridimensionamento (48x48)**: Tutte le immagini vengono uniformate a questa dimensione.
    - **RandomHorizontalFlip()**: Applica un capovolgimento orizzontale casuale, tecnica comune di **data augmentation** per aumentare la robustezza del modello alle variazioni di orientamento del volto.
    - **RandomRotation(10)**: Ruota casualmente l'immagine di un angolo compreso tra -10 e +10 gradi, contribuendo all'invarianza rotazionale.
    - **ColorJitter()**: Modifica casualmente luminosità, contrasto, saturazione e tonalità dell'immagine, rendendo il modello più robusto alle variazioni di illuminazione.
    - **ToTensor()**: Converte l'immagine PIL in un tensore PyTorch, scalando i valori dei pixel da [0, 255] a [0, 1].
    - **Normalize(mean=[0.5], std=[0.5])**: Normalizza i valori dei pixel a un intervallo di [-1, 1], una pratica che potrebbe migliorare le prestazioni della rete.
- **Per Validazione e Test (val_test_transforms)**: Vengono applicate solo le trasformazioni di ridimensionamento, conversione in tensore e normalizzazione, senza l'aumento dei dati, per garantire una valutazione coerente e senza distorsioni.

I dati vengono poi organizzati in DataLoader per consentire l'iterazione in batch, ottimizzando l'efficienza dell'addestramento e sfruttando al meglio l'accelerazione hardware (GPU), se disponibile.

Architettura del Modello (EmotionCNN)

Il modello di riconoscimento delle emozioni è una Rete Neurale Convoluzionale (CNN) denominata EmotionCNN, definita nel modulo model.py. L'architettura è composta da una sequenza di blocchi convoluzionali, seguiti da strati completamente connessi (fully connected layers).

Ogni blocco convoluzionale include:

- **Strato Convoluzionale (nn.Conv2d)**: Applica filtri all'input per estrarre caratteristiche. Il modello include tre coppie di strati convoluzionali, con un aumento progressivo del numero di canali (64, 128, 256) per catturare caratteristiche di complessità crescente. I filtri sono di dimensione 3x3 con padding 1 per mantenere le dimensioni spaziali dell'input.

- **Batch Normalization (nn.BatchNorm2d)**: Normalizza le attivazioni dei neuroni in ogni batch. Questa tecnica stabilizza e accelera il processo di addestramento, riducendo la dipendenza dall'inizializzazione dei pesi e permettendo tassi di apprendimento più elevati.
- **Funzione di Attivazione ReLU (F.relu)**: Una funzione di attivazione non lineare che introduce la capacità della rete di modellare relazioni complesse nei dati.
- **Max Pooling (nn.MaxPool2d)**: Riduce le dimensioni spaziali dell'output dei layer convoluzionali. Questo aiuta a ridurre il numero di parametri e rende il modello più robusto a piccole traslazioni nell'immagine. Vengono utilizzati kernel di dimensione 2x2 con stride 2.
- **Dropout (nn.Dropout)**: Una tecnica di regolarizzazione che disattiva casualmente un sottoinsieme di neuroni durante l'addestramento. Questo impedisce al modello di fare eccessivo affidamento su specifici neuroni, migliorando la sua capacità di generalizzare e prevenendo l'overfitting.

Dopo i blocchi convoluzionali, l'output viene appiattito e passato a due strati completamente connessi (nn.Linear):

- Il primo strato (fc1) riduce la dimensionalità delle caratteristiche estratte, seguito da Batch Normalization e Dropout.
- Il secondo strato (fc2) è lo strato di output, che mappa le caratteristiche estratte alle 7 classi di emozioni.

Processo di Addestramento

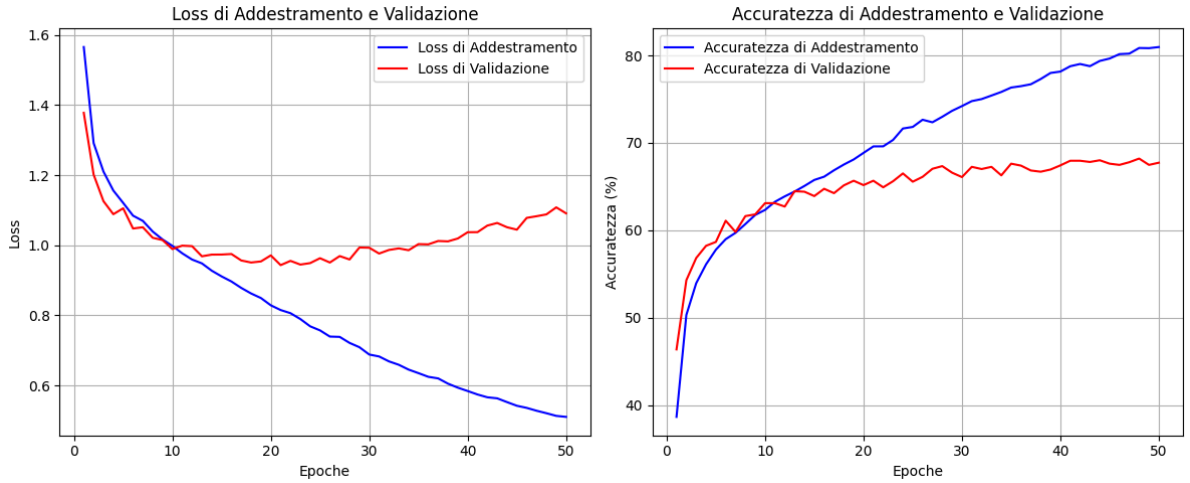
Il processo di addestramento è effettuato dalla funzione train_model nel modulo train.py, che viene richiamata da main_emotion_recognition.py.

- **Dispositivo**: Il modello viene automaticamente configurato per l'utilizzo di una GPU (CUDA) se disponibile, altrimenti ricade sulla CPU (come accade nei nostri casi di test).
- **Funzione di Loss (criterion)**: Viene utilizzata la CrossEntropyLoss, la funzione di perdita standard per i problemi di classificazione multi-classe, che combina la softmax con la negative log-likelihood loss.
- **Ottimizzatore (optimizer)**: Viene impiegato l'ottimizzatore Adam con un tasso di apprendimento iniziale di 0.001.
- **Ciclo di Addestramento**: Il modello viene addestrato per un numero predefinito di epoche.
    - **Fase di Addestramento (model.train())**: Per ogni batch di dati, i gradienti vengono azzerati, viene eseguito un forward pass per ottenere le predizioni, calcolata la loss, eseguiti un backward pass per calcolare i gradienti e un passo dell'ottimizzatore per aggiornare i pesi del modello.
    - **Fase di Validazione (model.eval())**: Dopo ogni epoca di addestramento, il modello viene valutato sul set di validazione. Durante questa fase, il calcolo dei gradienti è disabilitato (torch.no_grad()) per ottimizzare l'uso della memoria e velocizzare la valutazione, in quanto non sono necessari aggiornamenti dei pesi.
- **Monitoraggio e Salvataggio**: Vengono monitorate la loss e l'accuratezza sia per il set di addestramento che per quello di validazione. Il modello che raggiunge la migliore accuratezza sul set di validazione viene salvato come best_model.pth nella directory checkpoints/ utilizzando la funzione save_checkpoint da utils.py.

Valutazione Finale

Al termine del processo di addestramento, il modello viene valutato sul set di test (PrivateTest), che non è stato utilizzato in alcuna fase precedente (addestramento o validazione). Questo fornisce una stima imparziale delle prestazioni generalizzative del modello. Vengono calcolate e stampate la loss e l'accuratezza finali sul set di test.

Le metriche di addestramento (loss e accuratezza di addestramento e validazione per ogni epoca) vengono visualizzate graficamente e salvate come training_metrics.png nel percorso plots/ utilizzando la funzione plot_metrics fornita in utils.py. Questo permette di analizzare l'andamento dell'addestramento e individuare eventuali problemi come l'overfitting.



Genuineness Detection

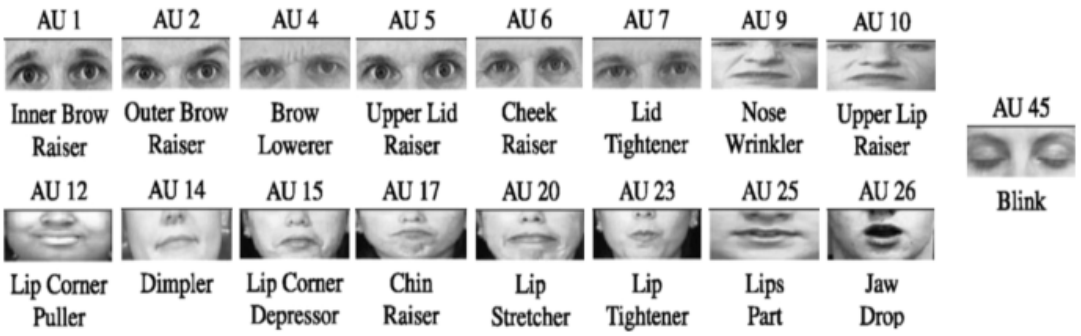
Classificare un espressione facciale in base alla genuinità non è una task facile (nemmeno per un essere umano) proprio per questo motivo siamo stati attratti da questo, l'idea era in aggiunta alla emotion recognition, di avere una feature in più che cerca di comprendere quando la nostra espressione facciale è genuina oppure no.

La prima sfida è stata capire come si possa predire se un espressione sia genuina, quindi siamo andati alla ricerca di alcuni studi che dimostrano come le microespressioni facciali involontarie diano informazioni a riguardo.

Un'espressione facciale genuina attiva i muscoli involontari, risultando spesso più simmetrica e spontanea, viceversa un'espressione finta, attiva i muscoli volontari, risultando più asimmetrica, cercando alcune informazioni in rete ci siamo imbattuti negli action units (AUs)

Gli action units sono un modo per descrivere i movimenti facciali in base ai muscoli coinvolti in un'espressione, permettendo così di analizzare il comportamento facciale in modo oggettivo e standardizzato.

Ci siamo anche imbattuti in questo paper: https://arxiv.org/pdf/2008.11353 che spiega come e perché gli action units sono un valido metodo per distinguere le espressioni facciali tra genuine / finte.

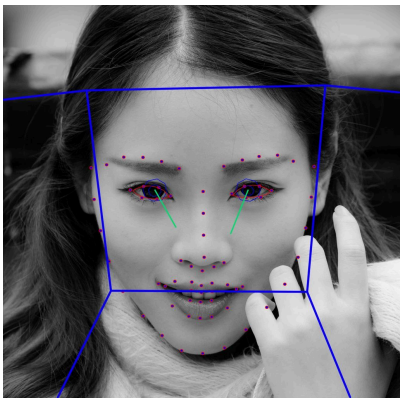


Una volta arrivati a questo punto è iniziata la vera sfida, uno dei punti più critici di questa parte è stato proprio il dataset, non abbiamo trovato in rete dataset pubblici costruiti e utilizzati per questo scopo, abbiamo trovato riferimenti ad alcuni dei dataset più usati per queste task, come BP4D-Spontaneous, SPOS (Spontaneous and Posed Facial Expression Dataset) .. ma non erano accessibili pubblicamente.

L'unica alternativa era di costruire un dataset custom, per fare questo abbiamo cercato dataset contenenti volti e espressioni facciali (spesso utilizzati per emotion recognition) ed abbiamo selezionato due differenti dataset, uno che contiene delle espressioni facciali che sembrano per lo più finte e un altro che contiene delle espressioni facciali più genuine e naturali.

I dataset sono stati presi da Kaggle, rispettivamente:

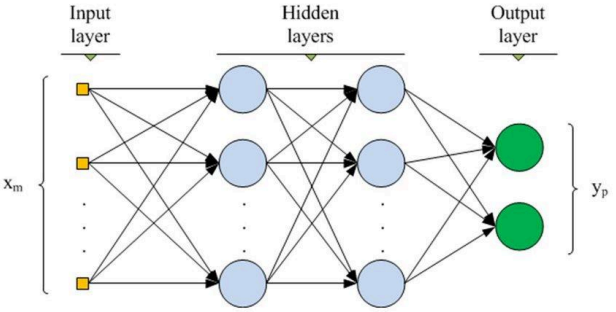
- posed: *https://www.kaggle.com/datasets/sudarshanvaidya/random-images-for-face-emotion-recognition*
- genuine: *https://www.kaggle.com/datasets/freak2209/face-data*



A questo punto il secondo step è stato trovare un modo per estrarre gli action units dalle immagini per costruire un dataset in formato csv che contenesse i vari AUs estratti e la label associata, dato la complessità di questa task ci siamo affidati a OpenFace, un tool esterno open source molto utilizzato per questo scopo.

A sinistra un esempio di come OpenFace viene applicato per estrarre gli action units da un immagine, in output avremo un file csv che contiene gli AUs estratti

Una volta creato il dataset siamo potuti passare alla costruzione del modello, per risolvere questa task abbiamo creato un Multilayer Perceptron (MLP) che prende in input il vettore poi passa attraverso 2 hidden layer e infine abbiamo l'output che produce un logits finale che sarà usato per ottenere la probabilità e poi la classificazione binaria.



Ogni layer è seguito da:

- *Batch Normalization*: per rendere il training più stabile e accelerare la convergenza
- *Leaky ReLU*: per permettere alla rete neurale di imparare pattern più complessi
- *Dropout*: per regolarizzare il training e ridurre l'overfitting

Abbiamo poi utilizzato BCEWithLogitsLoss come loss function, che combina sigmoid layer e binary cross-entropy rendendo il training più stabile ed efficiente, abbiamo usato l'optimizer Adam (uno dei più comuni e utilizzati) con l'aggiunta di un L2 penalty che applica una funzione matematica per penalizzare i pesi più grandi, con l'obiettivo di generalizzare meglio i nuovi dati e ridurre l'overfitting.

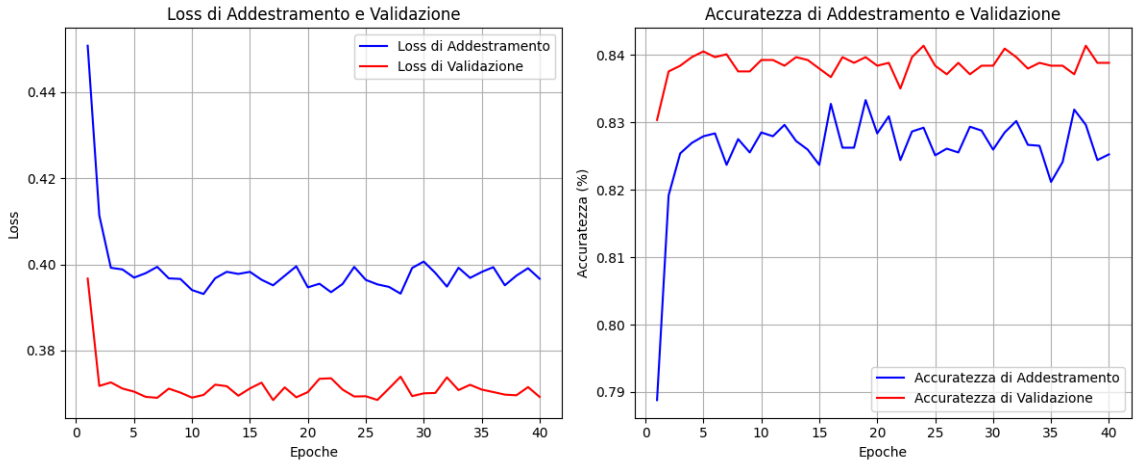
$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

- *BCEWithLogitsLoss Formula*

Abbiamo utilizzato anche uno scheduler StepLR, che ogni 2 epochs diminuisce il learning rate del 10%, con l'obiettivo di migliorare la convergenza durante il training.

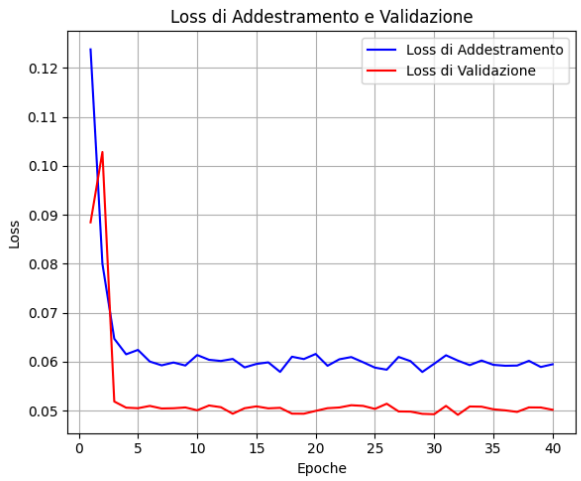
Partendo dal presupposto che il dataset non è stato costruito in maniera perfetta, e che avevamo a disposizione solamente 2000 samples, uno degli obiettivi durante il training e il testing è stato di cercare di ridurre al minimo l'overfitting.

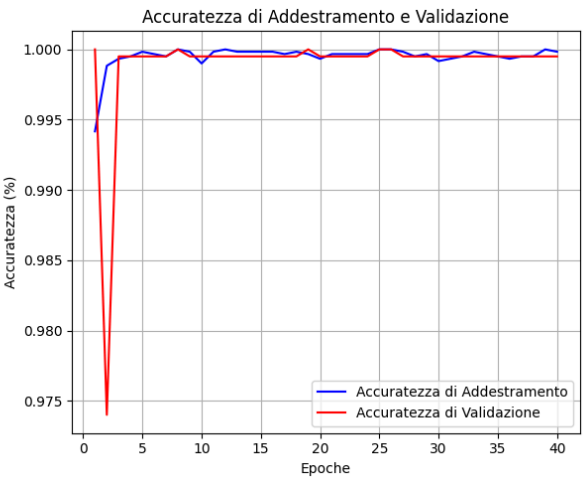
Abbiamo usato la cross validation con 4 folders per cercare di avere una valutazione più accurata e stabile, che viene applicata direttamente al dataset originale (abbiamo un unico dataset, non è stato diviso in training e testing.)



I grafici ottenuti durante il training usando 40 epochs mostrando come nelle prime 5 epochs circa otteniamo notevoli miglioramenti, mentre dopo abbiamo delle piccole oscillazioni, ma rimaniamo abbastanza stabili, raggiungiamo la stabilità molto velocemente, questo principalmente perché abbiamo pochi samples nel nostro dataset, l'accuracy è comunque molto buona, ma credo ci siano comunque dei piccoli bias che portano a un leggero overfitting dovuto proprio alla divergenza delle immagine presenti nel dataset originario.

Abbiamo provato ad utilizzare anche i landmarks invece degli action units, estraendo le coordinate dei pixel con il tool face_mesh di mediapipe, ma il modello presenta durante il training dei forti bias dovuti al dataset, raggiungendo un accuracy di circa il 100% abbiamo un chiaro overfitting, in cui il modello impara molto bene a riconoscere i pattern, ma purtroppo riconosce i pattern sbagliati





Nonostante ciò nel codice abbiamo lasciato la parte relativa ai landmarks, dando la possibilità di testare la previsione del modello basata sui landmarks, anche se risulta chiaramente sbagliata in quanto se il viso è vicino alla videocamera ci ritorna false, altrimenti se ci allontaniamo ci da true.

L'abbiamo lasciata principalmente per darci la possibilità in futuro di cambiare dataset o di provare ad applicare delle funzioni di normalizzazione sui dati prima della creazione del dataset stesso.

La costruzione del dataset è stata sicuramente la parte più critica, in quanto non avendo trovato dataset applicabili, abbiamo dovuto costruirne uno custom, che presenta comunque dei bias, e ciò ha influenzato tutto il lavoro successivo.

Proprio per questo motivo nonostante una buona accuracy non abbiamo la certezza che il modello funzioni bene, testando con la webcam funziona bene in alcune situazioni, ma non in tutti i casi, principalmente perchè una posa neutra che rileva pochi AUs viene rilevata più probabilmente come false, mentre posizioni più accentuate più probabilmente come true, ma anche in questo caso è difficile da dire se ciò è dovuto a dei bias del modello per colpa del dataset oppure se proprio a dei problemi strutturali nel rilevamenti degli AUs.

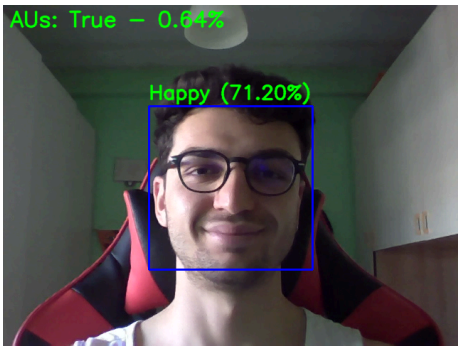
In ogni caso si presentava una task molto stimolante e divertente da affrontare che ci ha permesso di approfondire meglio molti aspetti che non conoscevamo

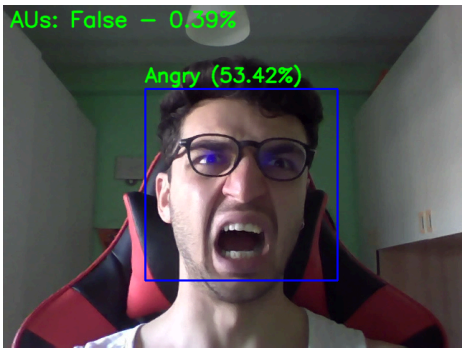
La prima immagine a sinistra è presa dal dataset posed, mentre quella a destra dal dataset genuine, nonostante siano entrambe due file .png di dimensione 224 x 224 in grayscale, risulta evidente che abbiamo delle importanti differenze nello stile e qualità delle immagini, tutti fattori che hanno sicuramente influito durante il training del modello.

Conclusione

Nonostante alcune sfide incontrate durante il percorso, siamo riusciti a creare due modelli rispettivamente un CNN e un MLP, che bene o male riescono a portare a termine il loro intento e nonostante alcuni problemi incontrati con i dataset, testando il codice abbiamo ottenuto dei risultati non perfetti, ma di cui ci possiamo ritenere soddisfatti.

Nella root del progetto è presente un file main.py, che contiene il codice per testare i modelli tramite webcam, viene fatta la parte di emotion recognition in tempo reale, mentre premendo il tasto 'A', viene salvato il frame e effettuata la genuinity detection basata sugli action units, e viene poi mostrato il frame con il risultato finale, questa parte non poteva essere fatta in tempo reale in quanto per estrarre gli AUs viene usato il tool OpenFace che impiega alcuni secondi, per girare.





Premendo il tasto 'L' abbiamo lo stesso effetto, ma usando la predizione basata sui landmarks, anche se come spiegato precedentemente, la predizione non funziona bene, basandosi sulla distanza del viso più che sugli effettivi movimenti facciali.

Ovviamente per far funzionare il main abbiamo bisogno di fare prima il training dei modelli che salva i pesi in dei checkpoint, questo ci permette poi di caricare i modelli allenati e fare le previsioni sui frame che gli vengono passati.

Di seguito il link alla repo github: https://github.com/LucaMica02/GenuEmotion

In questo link invece ci sono i dataset utilizzati per fare il training dei modelli, più le immagini originali usate per estrarre action units e landmarks:
https://drive.google.com/drive/u/0/folders/1w5s0F2jjYb_B5ut2sYBO2ZqEqnOz0aLu