Università di Catania

Machine Learning– 6 CFU

A.A. 2024/2025

Corso di Laurea Magistrale in Informatica

Dipartimento di Matematica e Informatica

Relazione Progetto

FoodVision: Algoritmo di Classificazione Multi Classe per Riconoscimento di Cibi

---

Mannino Daniele – 1000015345

Portoghese Francesco – 1000069265

Torrisi Giovanni – 1000015103

Indice

1. Introduzione

Arthur Samuel (1959), uno dei padri fondatori del Machine Learning, definì il Machine Learning come quel campo di studi scientifici che fornisce ad un computer l'abilità di "**imparare**" senza essere esplicitamente programmato.

Tale concetto venne ridefinito e perfezionato da Tom Mitchell (1997), il quale definì il Machine Learning come segue:

> *"Si dice che un Computer impari dall'esperienza **E** rispetto ad una classe di task **T** ed una misura di Performance **P**, se le sue performance rispetto al task T, misurate da P, migliorano con l'esperienza E".*

Con task ci si riferisce al problema da risolvere per il quale viene creato l'algoritmo di Machine Learning.

L'esperienza è invece la raccolta di dati (immagini, nel caso descritto di seguito) grazie al quale è possibile "allenare" il modello affinché quest'ultimo sia in grado di commettere il minor numero di errori possibile, massimizzando quindi la precisione (accuracy) e di conseguenza le performance, misurate attraverso appositi strumenti matematici e statistici.

Nel seguito della relazione verranno illustrati gli step necessari alla creazione di un algoritmo di Machine Learning in grado di risolvere un task di Classificazione Multi Classe, verranno descritte le procedure di training, validation e test, le misure di performance utilizzate per valutare le prestazioni, i dati scelti per allenare l'algoritmo e i risultati, intermedi e finali, ottenuti dallo studio e dal confronto di più architetture, con le relative performance, per la risoluzione dello stesso task.

Inoltre, in allegato alla presente relazione, verrà fornita una Demo semplice e intuitiva che permetterà all'utente di testare il corretto funzionamento dell'algoritmo.

2. Problema

In generale un task di Classificazione ha lo scopo di definire un modello parametrizzato, una funzione costo e un algoritmo di learning utili ad individuare, automaticamente, il decision boundary ottimale considerando i dati di training.

Nel caso in oggetto, il problema richiede di risolvere, attraverso gli strumenti elencati sopra, un problema di Classificazione Multi Classe relativo alla Classificazione di Cibi.

In particolare, dopo una fase di studio e analisi, i cibi sono stati raggruppati, etichettati e suddivisi in 11 classi:

- Bread;
- Dairy Product;
- Dessert;
- Egg;
- Fried Food;
- Meat;
- Noodles-Pasta;
- Rice;
- Seafood;
- Soup;
- Vegetable-Fruit;

Si noti che le classi scelte racchiudono molte varietà di cibi, ricreando uno scenario compatibile con il mondo reale.

Inoltre la suddivisione tra le classi è netta e distinta; in tal modo è possibile minimizzare errori di classificazione dovuti alla possibile appartenenza contemporanea di un dato campione fornito in input al modello a più di una classe.

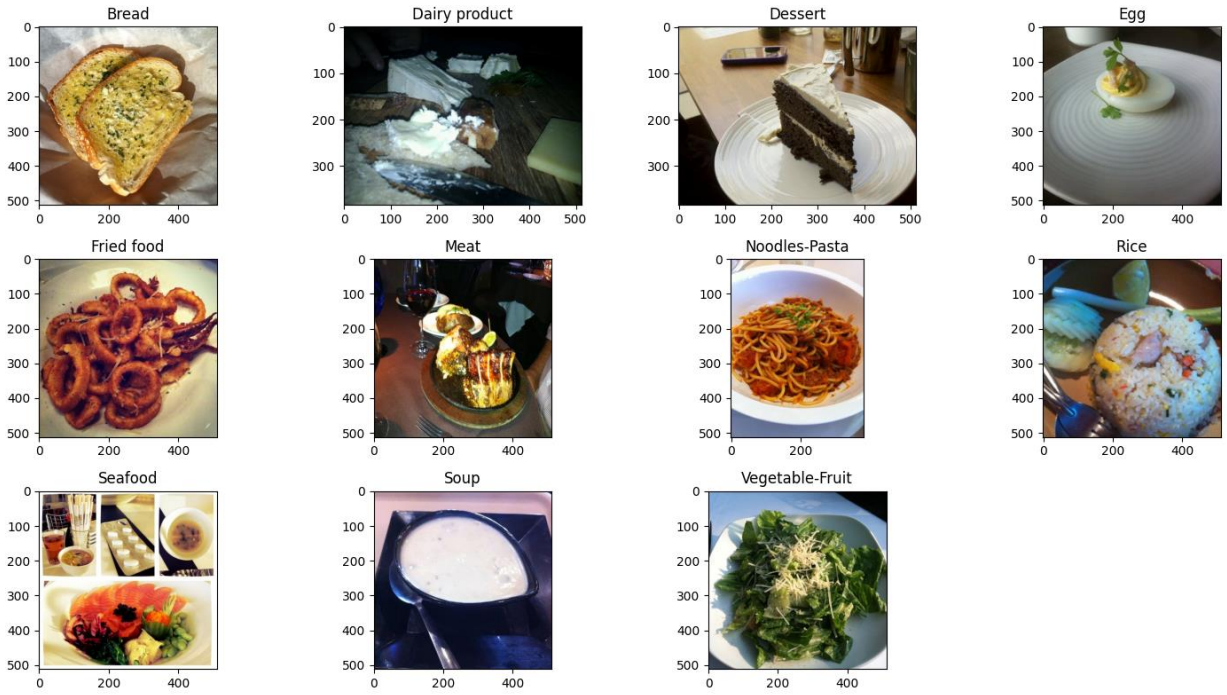


Figura 1: Suddivisione dei cibi in classi.

La figura mostra un esempio di training per ognuna delle 11 classi; come già detto si può notare la netta differenza tra le immagini appartenenti a classi differenti.

3. Dataset

Per quanto riguarda l'acquisizione del dataset la base di partenza utilizzata è stato il dataset “Food11”, pubblicato sulla nota piattaforma online “Kaggle” e raggiungibile tramite il seguente link.

Il dataset si compone di 16.643 immagini già suddivise in:

- 9.866 immagini di training;
- 3.430 immagini di validation;
- 3.347 immagini di test;

e raggruppate secondo le macro categorie individuate in fase di analisi del problema.

Tenendo conto che la consegna del progetto a noi assegnato richiedeva di costruire manualmente il dataset di test, i dati sono stati inizialmente accorpati in un unico dataset.

Osservando i campioni del dataset, sono emerse alcune criticità che, al fine di ottenere una buona generalizzazione ed evitare dei bias, abbiamo deciso di risolvere ampliando il dataset di partenza.

La prima considerazione fatta è relativa al dataset food11, nel quale è presente una certa influenza asiatica (ad esempio la classe “Noodles-Pasta” contiene solo immagini relative a Noodles e Spaghetti, che è il tipo di pasta più diffuso nel continente asiatico) a fronte dei vari tipi presenti in Occidente; qui è necessaria una integrazione per rendere la classe più eterogenea.

La seconda considerazione riguarda il bilanciamento del dataset: si nota infatti un pesante sbilanciamento tra classi; la classe “Soup”, ad esempio, formata da circa 2500 immagini totali risulta molto più rappresentata della classe “Rice” con appena 472 immagini così come anche della classe “Dairy Product” con sole 721 immagini.

Anche qui una integrazione è utile per evitare bias verso le classi dominanti, migliorare le performance su tutte le classi, e avere una maggiore stabilità nei risultati e metriche.

Sono state utilizzate più fonti per ampliare il dataset:

- “**Food101**” pubblicato sulla piattaforma “Kaggle” al seguente link: tale dataset, composto da oltre 100 mila immagini di cibi, è stato usato principalmente per integrare le classi meno rappresentate nel training e validation set, quali “Dairy Product” e “Rice”. Sono state aggiunte al dataset 800 immagini relative alla classe “Dairy Product” e 1000 immagini relative alla classe “Rice”.

- “**Multi-class food image dataset”** pubblicato sulla piattaforma “Kaggle” al seguente link è stato utilizzato per ampliare ulteriormente la classe “Dairy Product” aggiungendo 130 immagini relative al latte, elemento poco presente nel dataset iniziale.

- “**Fruit Classification Dataset**” pubblicato sulla piattaforma “Kaggle” al seguente link composto da una raccolta di immagini trovate sul web, è stato utilizzato prettamente per l’ampliamento della classe “Vegetable-Fruit”. Le immagini di questa categoria raffiguravano una buona varietà di frutta e verdura ma spesso erano fin troppo generiche e aggregate. La scelta fatta è stata quella di aggiungere immagini relative alle “sottoclassi” di frutta: Mela, Banana, Melanzana, Uva e Mandarino;

- “**Vegetable Image Dataset**” pubblicato sulla piattaforma “Kaggle” al seguente link composto anch’esso da una raccolta di immagini trovate sul web, è stato utilizzato per un ulteriore ampliamento della classe “Vegetable-Fruit”. Da tale dataset abbiamo selezionato immagini delle “sottoclassi” Carota e Broccoli;

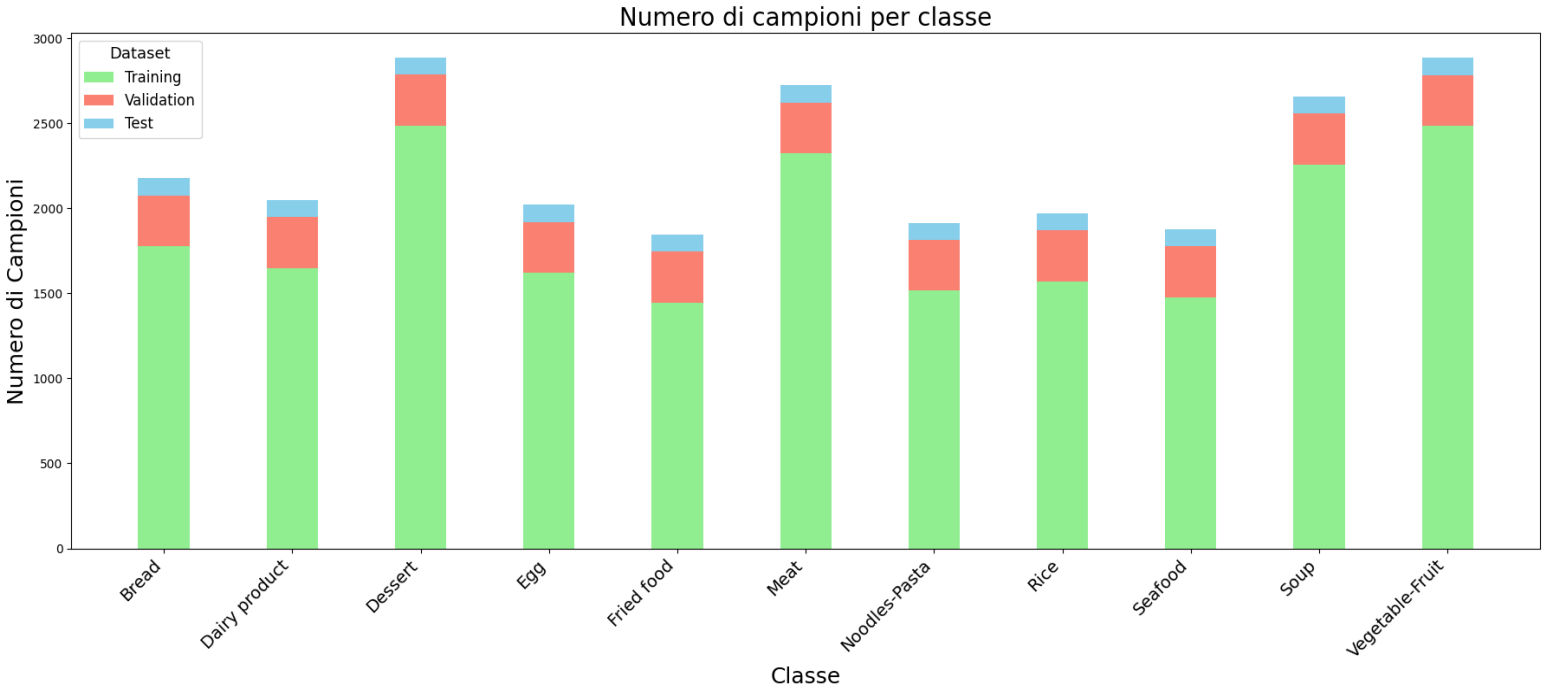
- “**icrawler**”, un tool utilizzato per effettuare il web crawling, ovvero l'automatizzazione della ricerca e del download di contenuti online. Questo strumento ci ha permesso di scaricare immagini da motori di ricerca come Google, Bing, Baidu tramite l’utilizzo di uno script Python specificando il numero di immagini da scaricare e keyword mirate affinché la ricerca fosse quanto più coerente ai nostri scopi. Abbiamo utilizzato questo tool per ampliare tutte le 11 classi del nostro dataset.

Al termine della fase di acquisizione il nuovo dataset è stato splittato in train set e validation set prendendo in modo casuale 300 campioni per ogni classe. Il

dataset di test è stato poi costruito manualmente prendendo interamente immagini dalla rete utilizzando il tool sopra citato. Inoltre, tramite l'utilizzo di un ulteriore script python è stato fatto un controllo sulla presenza di eventuali duplicati confrontando l'hash delle immagini.

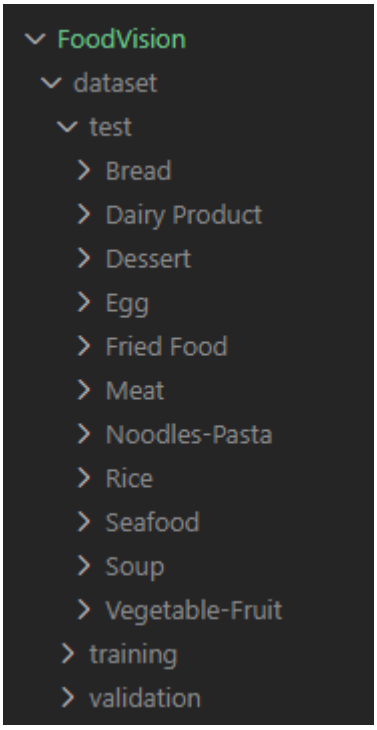
In definitiva, il dataset è composto da 25.006 immagini totali suddivise come segue:

- n° 20.606 immagini per il training set di cui:
  - n° 1776 immagini della classe "Bread";
  - n° 1650 immagini della classe "Dairy Product";
  - n° 2486 immagini della classe "Dessert";
  - n° 1620 immagini della classe "Egg";
  - n° 1446 immagini della classe "Fried Food";
  - n° 2323 immagini della classe "Meat";
  - n° 1515 immagini della classe "Noodles-Pasta";
  - n° 1572 immagini della classe "Rice";
  - n° 1477 immagini della classe "Seafood";
  - n° 2257 immagini della classe "Soup";
  - n° 2484 immagini della classe "Vegetable-Fruit";
- n° 3300 immagini per il validation set, distribuite equamente (300 immagini per classe) ;
- n° 1100 immagini per il test set, distribuite equamente (100 immagini per classe) ;



L'istogramma mostra chiaramente come la maggior parte dei campioni sia destinata al training set (82%), mentre le porzioni relative al validation set e test set occupano rispettivamente il 13% e 5% dei dati. Questa divisione garantisce al modello un'ampia base di dati su cui apprendere e dati a sufficienza per monitorare la capacità di generalizzazione del modello.

Il dataset è strutturato secondo la seguente gerarchia di cartelle:



Tale struttura rende ottimale l'implementazione dei dataset e la relativa etichettatura tramite l'utilizzo della classe ImageFolder della libreria Torchvision.

4. Metodi

Dopo aver ultimato le procedure di acquisizione del dataset da utilizzare per allenare il modello, si è passati alla fase di training.

Poiché la classificazione multiclasse è un problema noto alla comunità scientifica e già affrontato e risolto utilizzando molteplici metodi, sarebbe stato dispendioso e complesso creare una nuova architettura per il modello avendo a disposizione diverse soluzioni già documentate, funzionanti ed efficienti, che però non sono state sviluppate appositamente per risolvere il nostro specifico task, ma più in generale per risolvere task di classificazione multiclasse.

È stato quindi deciso di utilizzare come base di partenza per il nostro algoritmo alcuni dei modelli più noti in tale ambito. I risultati ottenuti sul validation set hanno permesso di poter individuare il valore appropriato per ognuno degli iperparametri: learning rate, numero di epoche, weight decay. L'architettura più adatta è stata scelta tramite tecniche di **finetuning** e **fixed feature extractor**. L'utilizzo di entrambe le tecniche ha permesso di poter monitorare la differenza di prestazioni tra i due metodi ed eventualmente agire proprio sull'architettura qualora si fosse presentato overfitting, trovando un compromesso tra complessità e prestazioni, poiché è noto che se ci sono meno layer allenabili ci sono meno parametri allenabili e di conseguenza minore complessità e meno probabilità di overfitting.

Per risolvere in modo mirato il task della classificazione di cibi, sono stati inizializzati i modelli con i relativi pesi pre-addestrati su ImagetNet, sfruttando così feature generali già apprese un vasto dataset di immagini naturali.

4.1. Squeezenet

SqueezeNet, introdotta nel 2016 da Forrest Iandola, nasce con l'obiettivo di risolvere i problemi legati alle elevate quantità di memoria e capacità computazionale richiesti per risolvere task di classificazione di immagini. L'idea alla base è mantenere delle prestazioni paragonabili a quelle di AlexNet sul dataset ImageNet, ma riducendo drasticamente il numero di parametri, e conseguentemente la dimensione del modello stesso.

La rete è composta principalmente da una sequenza di moduli chiamati **Fire modules** intervallati da alcuni strati di pooling per la ridurre le dimensioni. Al termine della rete, per produrre le classi di output troviamo uno strato di **convoluzione finale** e un **global average pooling**.

4.1.1. Il Fire Module

Il cuore dell'architettura di SqueezeNet che consente di ridurre drasticamente il numero di parametri è il **Fire Module**.

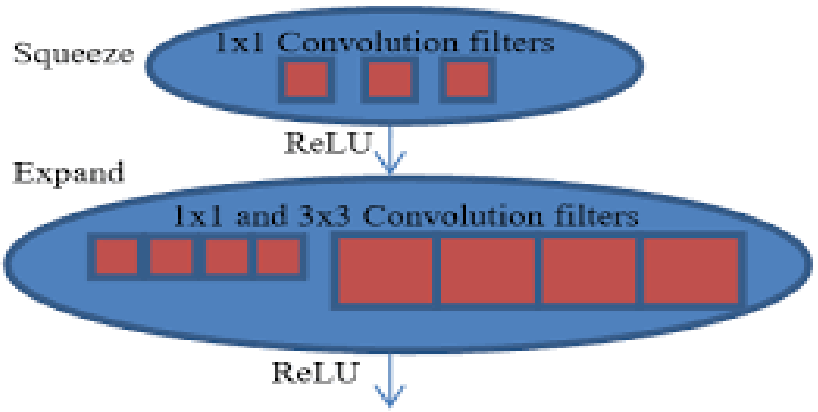
Un Fire Module si compone di due parti principali:

1. **Squeeze Layer**:
    - Utilizza convoluzioni 1×1
    - Comprime il numero di canali in ingresso
    - Funzione di attivazione: ReLU

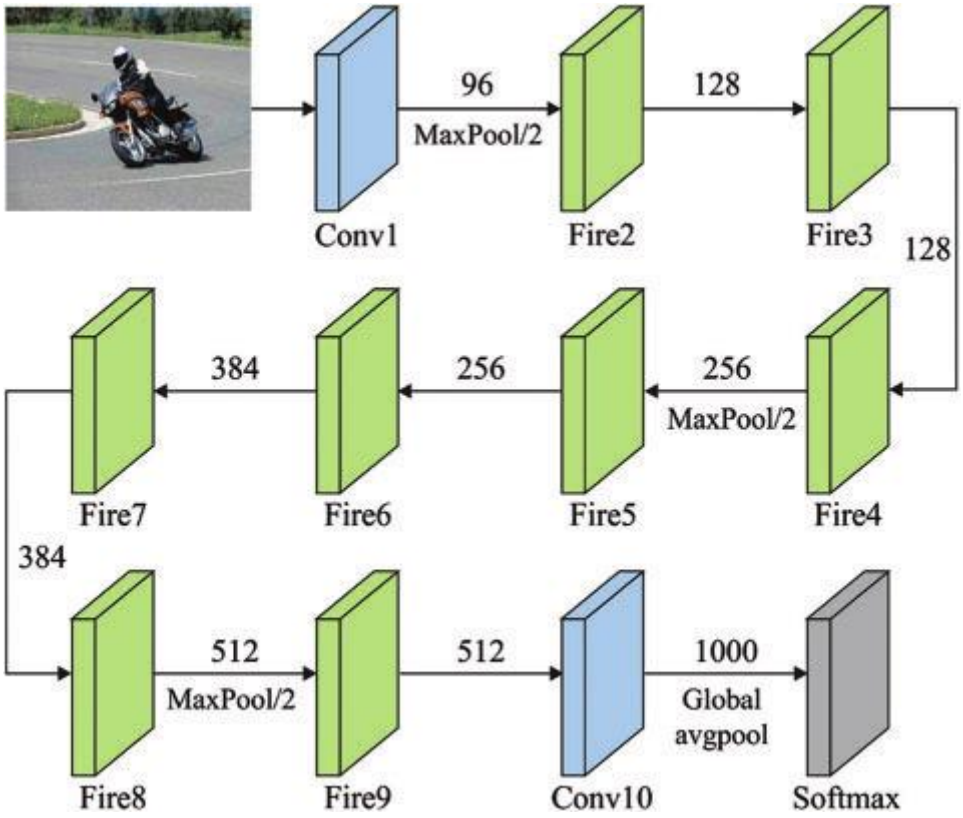
    Riduce i canali in ingresso al layer expand, facendo sì che le convoluzioni 3×3 successive lavorino su meno canali, risparmiando parametri.

2. **Expand Layer**:
    - Costituito da due branche parallele:
        - convoluzioni 1×1
        - convoluzioni 3×3 (con padding per mantenere dimensioni)
    - I due output vengono concatenati lungo la dimensione dei canali
    - Funzione di attivazione: ReLU

Di seguito lo schema della struttura di un fire module:



Di seguito un diagramma dell'intera architettura:



Abbiamo deciso di iniziare la sperimentazione con questo modello, sfruttando le sue caratteristiche di efficienza e ridotta necessità di capacità computazionale e memoria, per effettuare una prima valutazione dell'algoritmo riguardo tempi necessari a completare l'allenamento di una singola epoca e numero di epoche necessarie affinché i dati raccolti fossero sufficienti a concludere l'allenamento con un risultato soddisfacente in termini di accuracy e consistenza dei dati.

Una volta identificato il numero di epoche totali necessari all'allenamento e aver settato in modo ottimale i vari iperparametri, essendo i valori di accuracy non ancora adeguati ai valori minimi prefissati in fase di analisi del problema, è stato ripetuto il training utilizzando un modello più complesso in termini di capacità computazionale richiesta e numero di parametri da settare.

4.2. ResNet

ResNet (Residual Network), introdotta nel 2015 da Kaiming He e colleghi di Microsoft Research, nasce con l'obiettivo di affrontare il problema del degradation nelle reti neurali profonde: quando si aumenta il numero di strati, l'accuratezza sui dataset di training peggiora invece di migliorare, a causa di difficoltà nell'ottimizzazione e fenomeni di vanishing/exploding gradients.

L'idea chiave alla base di ResNet è l'introduzione dei **residual connections** (o *skip connections*), che permettono di apprendere *funzioni residue* anziché mappature dirette, facilitando il flusso dell'informazione e del gradiente attraverso la rete.

Questa strategia ha permesso di costruire reti con centinaia o migliaia di strati senza incorrere negli stessi problemi di addestramento delle architetture precedenti.

L'architettura base di ResNet è composta da:

- Uno strato iniziale di convoluzione e pooling per ridurre la dimensione spaziale dell'immagine;
- Una sequenza di blocchi residui (Residual Blocks) organizzati in *stage* con profondità crescente.
- Un *global average pooling* seguito da un fully-connected per produrre l'output delle classi.

I modelli ResNet sono disponibili in diverse varianti (ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152) che differiscono per numero di blocchi e per l'uso di Bottleneck Blocks nei modelli più profondi.

4.2.1. Il Residual Block

Il cuore dell'architettura ResNet è il Residual Block, il modulo che implementa le connessioni residue.

Un Residual Block si compone di due rami:

- **Main Path**:
    - Sequenza di strati convoluzionali (2 o 3 a seconda della variante).
    - Ogni convoluzione è seguita da Batch Normalization e funzione di attivazione ReLU.
    - Nel caso dei Bottleneck Blocks (ResNet-50 e oltre) si utilizzano convoluzioni 1×1 per ridurre e poi ripristinare il numero di canali, diminuendo il costo computazionale delle convoluzioni 3×3 centrali.

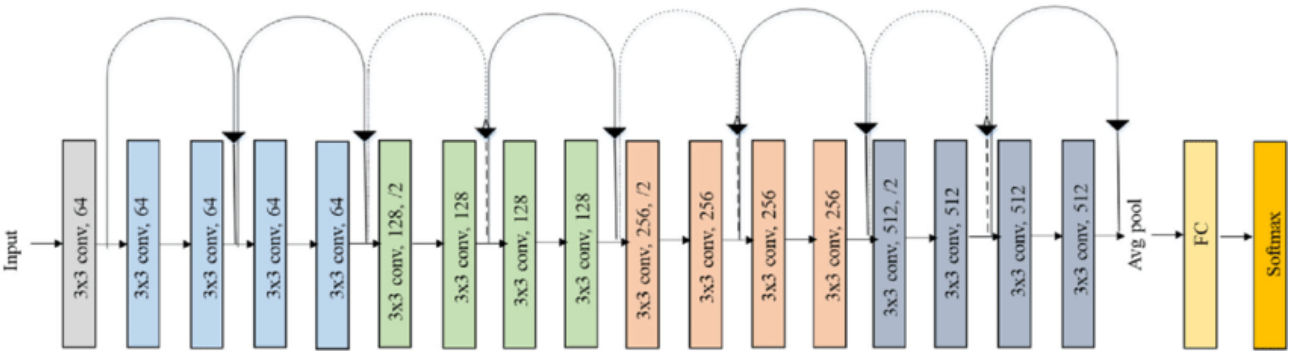
- **Shortcut Path**:
    - Riporta l'input del blocco direttamente in uscita, saltando le convoluzioni del main path.
    - Può essere:
        - **Identity mapping**: usato quando input e output hanno la stessa dimensione.
        - **Proiezione (1×1 conv)**: usata per adattare dimensione spaziale o numero di canali in caso di variazione tra input e output.

Il funzionamento del residual block si basa su tale principio:
Il blocco calcola:

$$y = F(x, W) + x$$

dove $F(x, W)$ rappresenta la trasformazione appresa dal main path e xè l'input originale. L'operazione di somma avviene elemento per elemento.

L'architettura completa del modello è riportata di seguito:



È stato scelto ResNet per la fase successiva di sperimentazione per via della sua comprovata capacità di raggiungere alti livelli di accuratezza su dataset complessi come ImageNet e alla stabilità dell'allenamento anche su profondità elevate.

Pur richiedendo più memoria e capacità computazionale rispetto a SqueezeNet, il design modulare dei blocchi residui consente un'ottima scalabilità e la possibilità di sfruttare varianti più leggere o più profonde in base alle risorse disponibili. Nel nostro caso è stato scelto ResNet18.

Inoltre, gli iperparametri settati durante il training con il modello Squeeznet (learning rate, numero di epoche totali per l'allenamento), si sono rivelati ottimali anche per ResNet, generando così un notevole miglioramento di prestazioni in termini di accuracy.

Tuttavia, per una maggiore completezza del compito, è stato scelto di proseguire la fase di sperimentazione adottando un terzo modello ancora più complesso, con l'intento di migliorare ancora le performance dell'algoritmo.

4.3. EfficientNet

EfficientNet, introdotta da Mingxing Tan e Quoc V. Le di Google nel 2019, nasce con l'obiettivo di trovare un **bilanciamento ottimale tra accuratezza e efficienza computazionale** nelle reti neurali convoluzionali per la classificazione di immagini.

L'idea chiave è il **compound scaling**, un approccio sistematico che aumenta simultaneamente profondità (*depth*), larghezza (*width*) e risoluzione dell'immagine (*resolution*) utilizzando un unico set di coefficienti di scalatura.

Questo metodo supera le strategie precedenti, che aumentano solo uno di questi fattori alla volta, evitando sprechi di capacità e garantendo un uso più efficiente delle risorse di calcolo.

La famiglia **EfficientNet** comprende diversi modelli (da EfficientNet-B0 a EfficientNet-B7), dove B0 è la versione base ottimizzata tramite ricerca architetturale automatica (*neural architecture search*), e le versioni successive applicano il compound scaling per ottenere modelli più grandi e accurati.

L'architettura di EfficientNet-B0 si basa su tre elementi principali:

- **MBConv** (Mobile Inverted Bottleneck Convolution): blocchi a basso costo computazionale derivati da MobileNetV2.
- **Squeeze-and-Excitation** (SE): meccanismo di attenzione sui canali per enfatizzare feature importanti.
- **Swish activation**: funzione di attivazione non lineare che migliora la capacità di apprendimento rispetto alla ReLU.

4.3.1. Il blocco MBConv con Squeeze-and-Excitation

Il cuore dell'architettura di EfficientNet è il MBConv Block arricchito con il meccanismo SE.

Un **MBConv** è composto da più fasi:

- **1. Expansion phase**:
  - Convoluzione 1×1 per espandere il numero di canali.
  - Funzione di attivazione: Swish

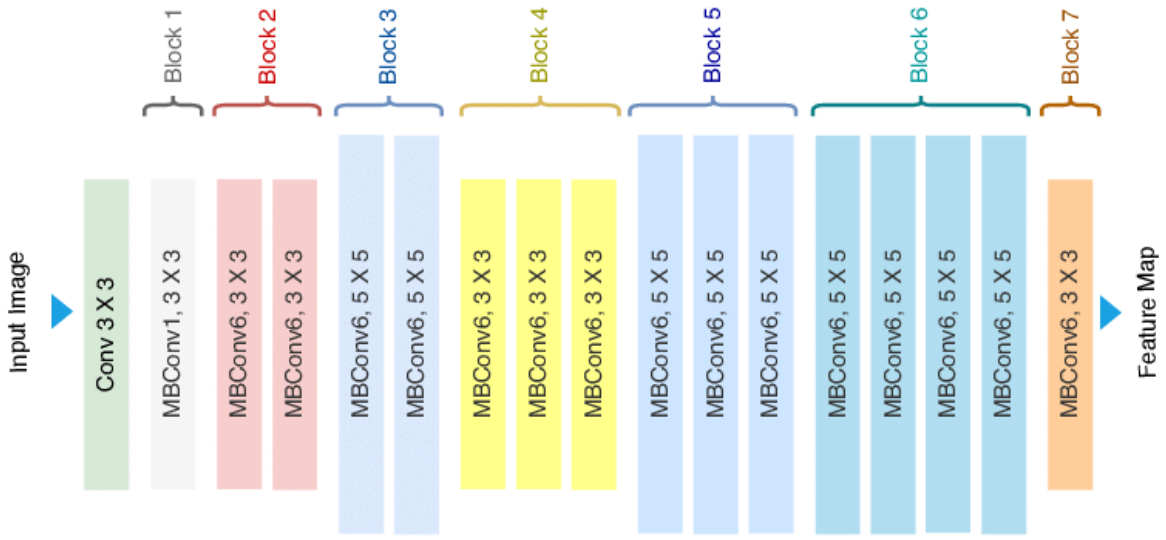
- **2. Depthwise convolution**:
  - Convoluzione separabile sui canali (*depthwise*) con kernel tipicamente 3×3 o 5×5.
  - Riduce drasticamente il numero di parametri e operazioni.
  - Seguita da Batch Normalization e Swish.

- **3. Squeeze-and-Excitation**:
  - Riduce temporaneamente il vettore dei canali a una dimensione inferiore (*squeeze*).

- Applica una rete completamente connessa per calcolare pesi di attenzione per ciascun canale (*excitation*).
- I pesi così ottenuti vengono moltiplicati per i canali originali per enfatizzare le feature rilevanti.

- **4. Projection phase**:
  - Convoluzione 1×1 per ridurre il numero di canali di nuovo alla dimensione desiderata.
  - Batch Normalization.

- **5. Skip connection** (opzionale):
  - Se dimensioni di input e output combaciano, l'output del blocco viene sommato all'input originale (residual connection).



Per i nostri esperimenti è stato scelto il modello EfficientNet-B0; pur essendo il meno complesso della famiglia, si sono notati leggeri miglioramenti rispetto ai modelli utilizzati in precedenza, grazie alle tecniche di compund scaling adottate intrinsecamente dal modello.

Essendo a questo punto soddisfatti dei risultati ottenuti si è conclusa la fase di sperimentazione e training dell'algoritmo con i risultati che saranno illustrati in dettaglio con grafici e tabelle nei capitoli successivi.

5. Valutazione

Tutte le considerazioni effettuate fino ad ora circa le performance dei modelli sono basate su delle metriche specifiche che permettono di valutare quanto preciso sia il nostro algoritmo.

Le principali misure utilizzare in fase di validation e successivamente di test sono state:

- **Precision**: risponde alla domanda: "Tra tutti i casi che il modello ha classificato come positivi, quanti erano effettivamente positivi?". Misura la qualità delle previsioni positive. È una metrica cruciale in scenari dove un falso positivo ha un costo elevato.

  La precisione è alta quando il numero di falsi positivi è basso.

  La sua formula è:

  $$Precision = \frac{TP}{TP + FP}$$

  dove con TP si indicano i veri positivi e con FP i falsi positivi.

- **Recall**: Il richiamo (o sensibilità) risponde alla domanda: "Tra tutti i casi che erano effettivamente positivi, quanti il modello è riuscito a identificare correttamente?". Misura la copertur**a** delle previsioni positive. È fondamentale in scenari dove un falso negativo ha un costo elevato.

  Il richiamo è alto quando il numero di falsi negativi è basso.

  La sua formula è:

  $$Recall = \frac{TP}{TP + FN}$$

  dove con TP si indicano i veri positivi e con FN i falsi negativi.

- **F1-Score**: è la media armonica di precisione e richiamo. È una metrica singola che bilancia entrambe le misure e fornisce un quadro più completo delle performance del modello, specialmente quando c'è un compromesso tra precisione e richiamo.

  Un valore alto di F1-score indica che il modello ha sia una buona precisione che un buon richiamo.

  Viene usata in contesti dove è importante che il modello sia equilibrato e non si focalizzi solo su una delle due metriche.

  La sua formula è:

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- **Accuracy**: L'accuracy (accuratezza) è la percentuale di predizioni corrette fatte dal modello rispetto al numero totale di predizioni.

  È una metrica globale: misura quante volte il modello indovina, senza distinguere le classi.

  È utile in contesti dove le classi sono bilanciate (cioè ogni classe ha più o meno lo stesso numero di esempi) e quando tutte le classi sono ugualmente importanti.

  La sua formula è:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Tali metriche, ottenute tramite funzione classification_report della libreria Python sklearn, contestualmente con la matrice di confusione ci hanno permesso di ottenere un quadro generale dell'andamento dell'algoritmo, fornendoci informazioni utili per attuare implementazioni e accorgimenti su elementi delle classi che non venivano correttamente classificate.

6. Esperimenti

Gli esperimenti sono stati condotti seguendo l'ordine dei modelli stabilito nella fase di analisi del problema e illustrati nel capitolo 4 (Metodi).

Durante la fase di validation, oltre alla ricerca dei migliori iperparametri, ci siamo concentrati anche sulla selezione dell'architettura più adatta. Poiché stavamo lavorando con un modello pre-addestrato abbiamo verificato fino a che punto fosse opportuno continuare ad allenare la rete, applicando le tecniche sopracitate di fine tuning e fixed feature extraction.

Per la scelta di quali layers congelare sono state osservate le implementazioni dei modelli utilizzati ed essi presentano tutti la stessa struttura generale, composta da due parti principali:

- **Features**: sezione convoluzionale che estrae le features dalle immagini;
- **Classifier**: sezione Fully Connected che effettua la classificazione finale.

Si è così deciso di allenare solo il modulo classifier per la fixed feature extraction e di tenere congelato il modulo features. Il fine tuning è stato chiaramente fatto allenando l'intera rete.

6.1. Esperimenti su SqueezeNet

Data la relativa leggerezza di squeezenet in termini di numero di parametri e considerata la potenza computazionale dei calcolatori a nostra disposizione, abbiamo addestrato il modello in ogni esperimento per 30 epoche.

- **Learning Rate: 0.01:**

  L'accuracy sul validation set, allenando tutti i layer del modello, si fermava intorno a 0.09 all'epoca 15, senza segni di miglioramento. Il training è stato a questo punto interrotto ipotizzando che un learning rate così elevato abbia probabilmente alterato in modo troppo consistente i pesi pre-addestrati, impedendo al modello di sfruttare efficacemente le feature già apprese dal pre-training.

Conclusione: Il learning rate 0.01 è troppo alto per fare training, per questo nelle prove successive si è deciso di abbassarlo.

- **Learning Rate: 0.001:**

  Con un learning rate fissato a 0.001, abbiamo ottenuto i seguenti risultati:

  - **Learning Rate: 0.001 (Finetuning):**

    Allenando tutta la rete, abbiamo ottenuto risultati migliori, ma comunque discreti e non ancora completamenti soddisfacenti (valore di accuracy sul validation set: 0.75).

  - **Learning Rate: 0.001 (Features Extractor):**

    Allenando solamente la sezione **"Classifier"**, abbiamo ottenuto risultati leggermente peggiori, poiché il modello non è riuscito ad apprendere rappresentazioni specifiche del nostro task. (valore di accuracy sul validation set: 0.71).

- **Learning Rate: 0.0001:**

  Abbiamo deciso di ridurre ulteriormente il learning rate, portandolo a 0.0001, per verificare se fosse possibile ottenere migliori performance. I risultati sono stati i seguenti:

  - **Learning Rate: 0.0001 (Finetuning):**

    Allenando tutta la rete, abbiamo ottenuto risultati migliori rispetto ai precedenti, arrivando ad un valore di accuracy di circa 0.84.

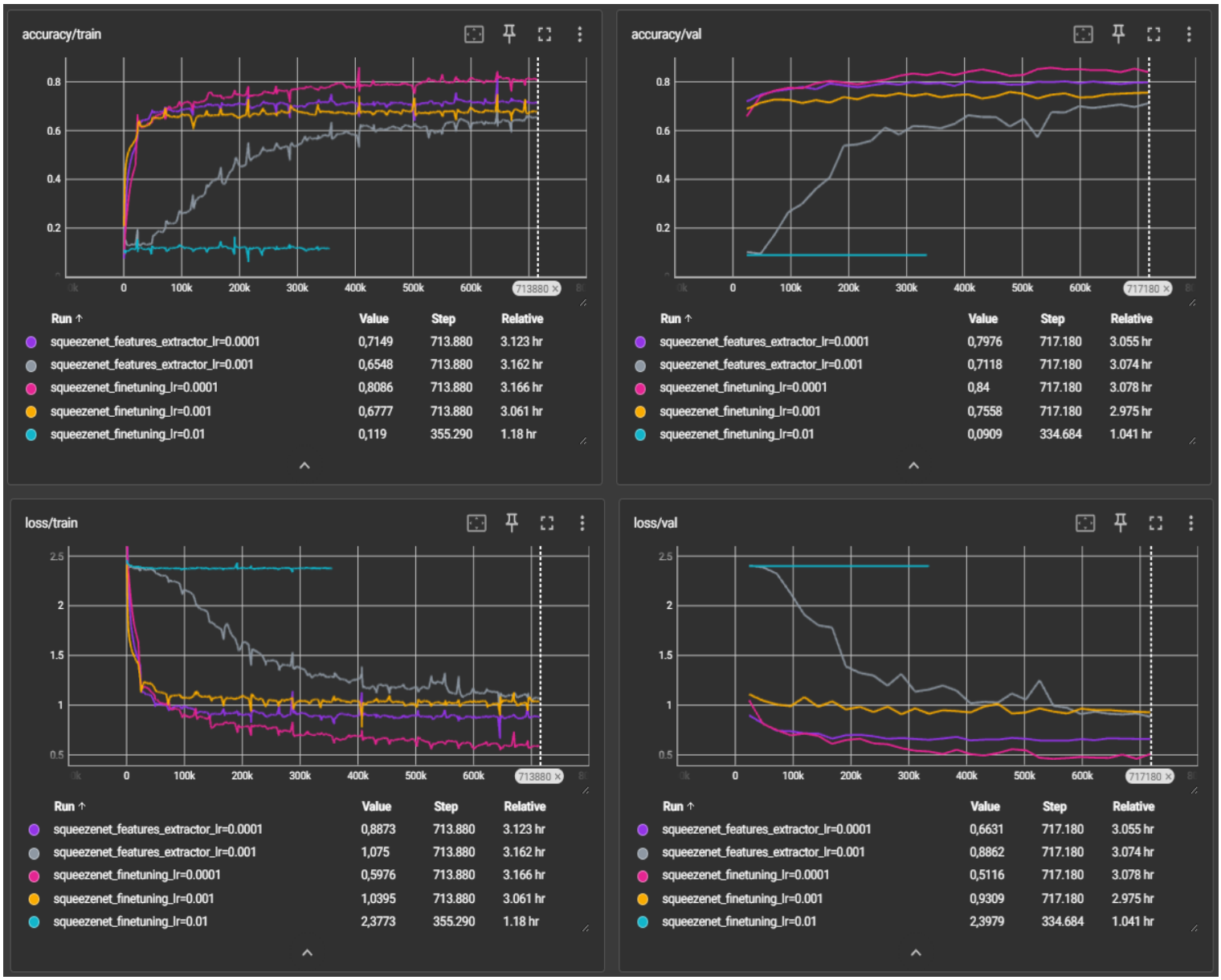
  - **Learning Rate: 0.0001 (Features Extractor):**

    Allenando solamente la sezione **"Classifier"**, come nel caso precedente, i risultati sono stati leggermente peggiori, sempre a causa della troppa genericità delle feature estratte. L'esperimento si è concluso con un valore di accuracy pari a 0.79.

La riduzione del learning rate tra i vari esperimenti utilizzando il modello SqueezeNet ha portato miglioramenti significativi in entrambi gli scenari, con un

vantaggio più evidente quando è stata allenata l'intera rete (come prevedibile che accadesse).

A seguire i grafici dei vari esperimenti, generati tramite il tool Tensorboard, che permettono di visualizzare graficamente l'andamento delle curve di loss e di accuracy durante le fasi di training e validation.



Come si evince dai grafici delle funzioni di loss, non sono presenti segnali che possano indicare un possibile overfitting. Le prestazioni migliori di Squeezenet sono state ottenute utilizzando l'opzione di fine tuning del modello con un valore di learning rate pari a 0.0001.

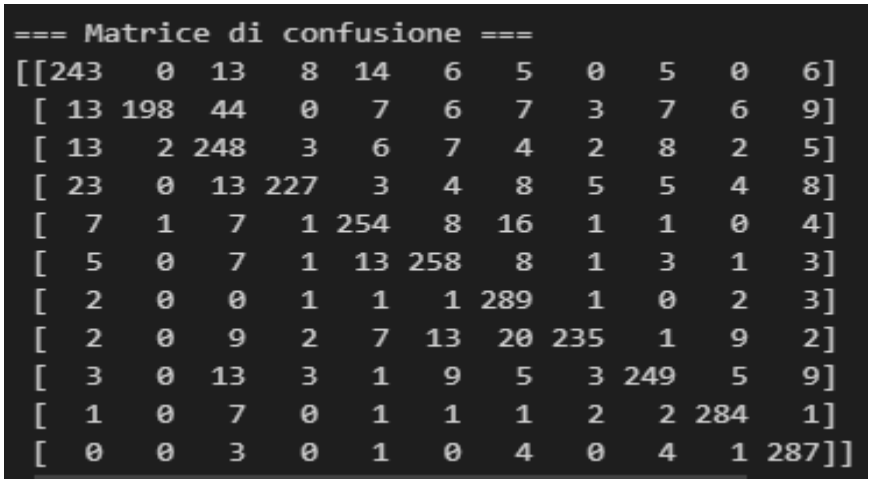
Un resoconto dettagliato circa i valori di precision recall ed f1-score sul validation set per le singole classi ed i valori totali di accuracy dell'esperimento migliore con il modello Squeezenet (fine tuning e lr = 0.0001) è mostrato a seguire:

```
=== Classification Report ===
                precision    recall  f1-score   support

         Bread       0.78      0.81      0.79       300
  Dairy product      0.99      0.66      0.79       300
       Dessert       0.68      0.83      0.75       300
           Egg       0.92      0.76      0.83       300
    Fried food       0.82      0.85      0.84       300
          Meat       0.82      0.86      0.84       300
  Noodles-Pasta      0.79      0.96      0.87       300
          Rice       0.93      0.78      0.85       300
       Seafood       0.87      0.83      0.85       300
          Soup       0.90      0.95      0.93       300
Vegetable-Fruit      0.85      0.96      0.90       300

      accuracy                           0.84      3300
     macro avg       0.85      0.84      0.84      3300
  weighted avg       0.85      0.84      0.84      3300
```

Osserviamo che Bread, Dairy Product e Dessert hanno valori di F1-Score minori probabilmente dovuti all'elevata diversità al loro interno che causa ambiguità visiva.

Ad esempio Bread include al suo interno anche panini, hamburger e pizza; e dessert è una classe che comprende dolci dalle diverse forme e colori come biscotti macarons, gelati, torte ecc… Da notare che Dairy Product ha una precision altissima, e vuol che quando il modello predice "Dairy Product" ha quasi sempre ragione, tuttavia il recall è molto basso , segno che ci sono tanti falsi negativi e quindi il modello tende a classificare gli elementi della classe come altro.

A conferma delle osservazioni precedenti è utile analizzare la matrice di confusione.



La seconda riga è quella relativa ai latticini e come si vede dall'immagine la maggioranza dei falsi negativi sono concentrati sui dessert. Queste considerazioni trovano conferma anche osservando le statistiche di dessert: essi non vengono quasi mai scambiati per dairy product, quindi dessert ha recall alto ma precision più bassa di tutti.

Per quanto riguarda le restanti categorie il modello mostra prestazioni solide, con valori di f1-score costantemente maggiori o uguali a 0.83 e due classi che superano la soglia di 0.93, evidenziando una classificazione complessivamente affidabile.

Il modello squeezenet è veloce ma ha capacità limitata, quindi probabilmente adottando un modello più potente si possono catturare dettagli più fini e aumentare le performance anche sulle classi più "critiche" come quelle sopra citate.

6.2. Esperimenti su ResNet

Avendo ottenuto risultati discreti con il modello SqueezeNet effettuando esperimenti con 30 epoche totali, ed essendo ResNet un modello più complesso, si è deciso di mantenere fisso il numero di epoche (anche e soprattutto considerando le limitate capacità computazionali disponibili), variando invece il parametro del learning rate.

- **Learning Rate: 0.0001:**

  Avendo come riferimento il modello SqueezeNet, le cui migliori prestazioni sono state ottenute con un learning rate pari a 0.0001, si è

deciso di iniziare a sperimentare con il nuovo modello partendo proprio da questo valore, adottando lo stesso piano d'azione (verificare il comportamento del modello sia utilizzando fine tuning, sia feature extraction).

- **Learning Rate: 0.0001 (Finetuning):**

  Allenando tutta la rete, abbiamo ottenuto risultati ottimi, nettamente migliori, a parità di learning rate e numero di epoche totali, di quelli ottenuti con SqueezeNet, arrivando ad un valore di accuracy di circa 0.925.

- **Learning Rate: 0.0001 (Features Extractor):**

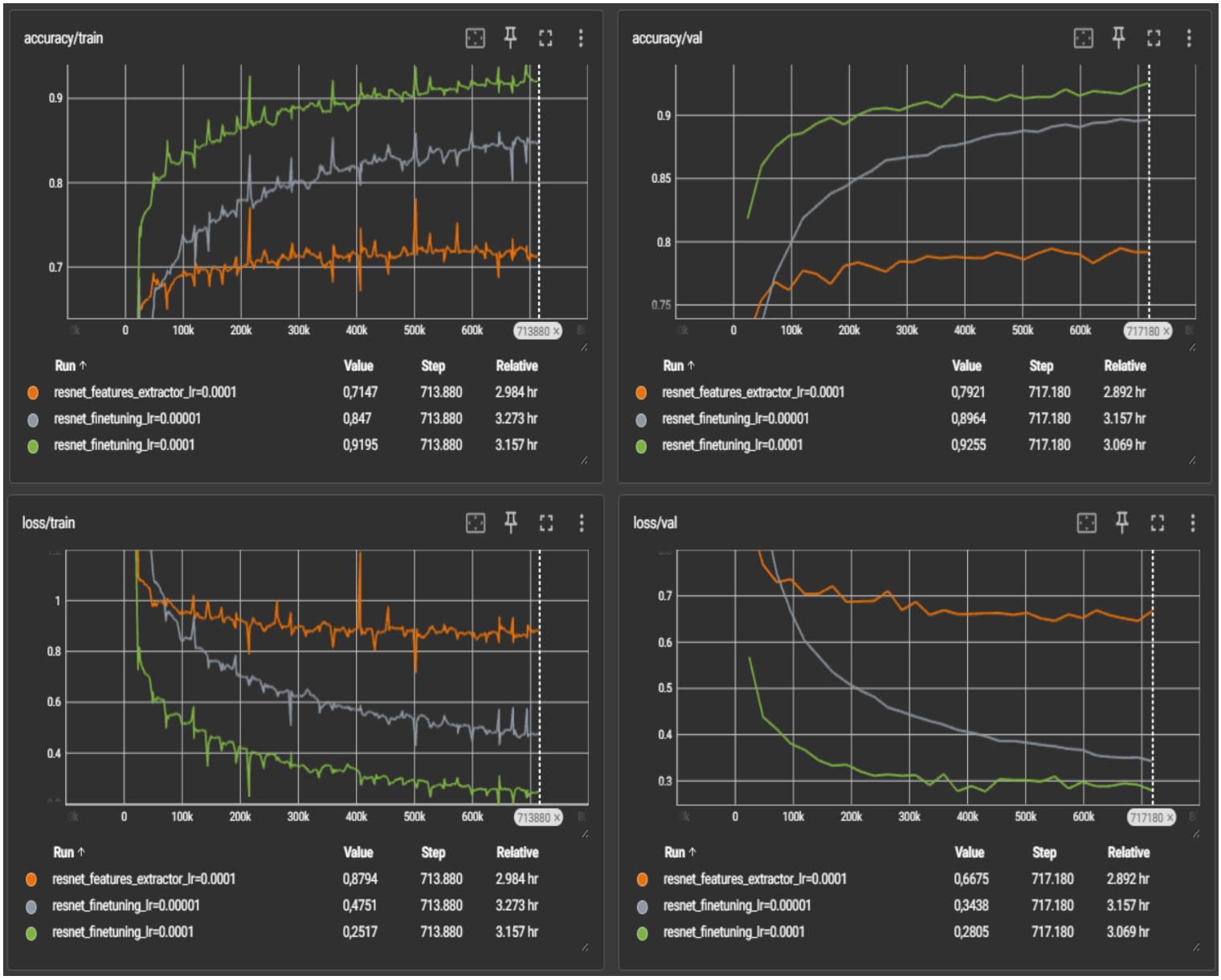
  Si è confermato invece il trend per cui, allenando solo la sezione "Classifier", per il nostro specifico task, si ottengono risultati peggiori, sempre a causa della troppa genericità delle feature estratte. L'esperimento si è concluso con un valore di accuracy pari a 0.79.

- **Learning Rate: 0.00001:**

  Poiché ResNet è un modello decisamente più complesso di SqueezeNet, si è deciso di effettuare un ulteriore esperimento, allenando tutta la rete e abbassando ancor di più il learning rate, per sfruttare maggiormente i pesi del modello pre-allenato. Tuttavia, i risultati si sono dimostrati leggermente inferiori rispetto ai precedenti, poiché il modello non è riuscito a generalizzare abbastanza sugli esempi di input. Il valore di accuracy finale si è attestato intorno al 0.896.

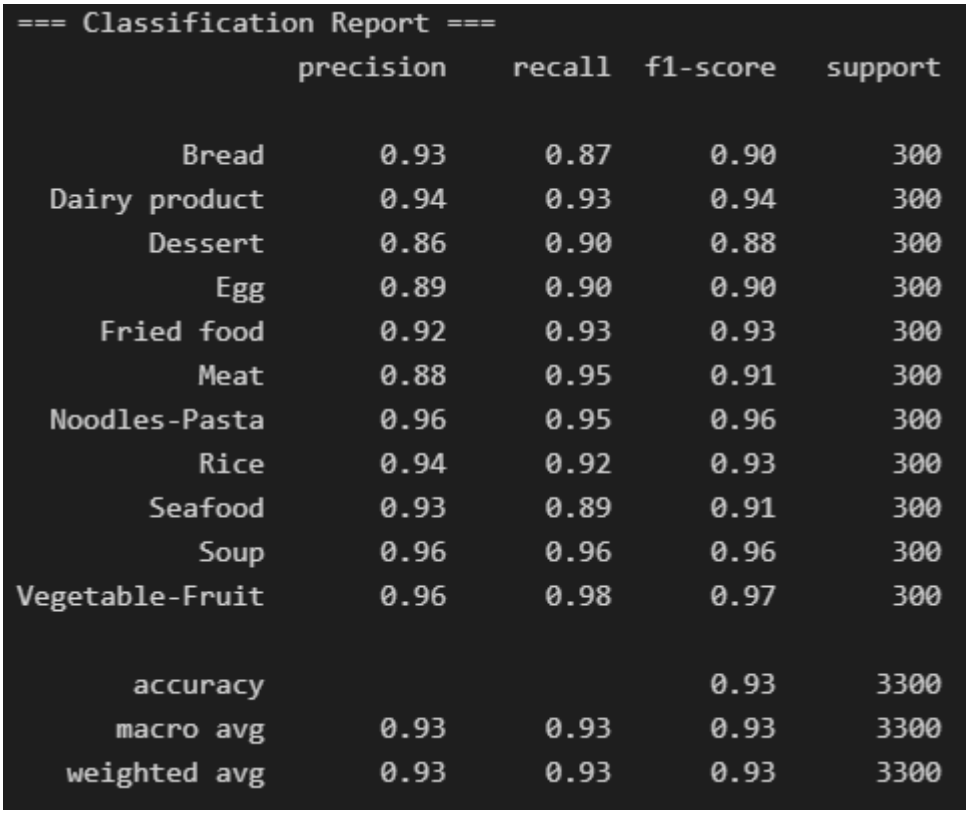
Come già anticipato, i pronostici riguardanti i possibili miglioramenti relativi ai valori di accuracy utilizzando questo modello più potente sono stati pienamente rispettati.

A seguire la rappresentazione grafica delle curve di loss e dell'andamento dei valori di accuracy durante la procedura di training.



Come si evince dai grafici delle funzioni di loss, nessun segnale che indichi un possibile overfitting. Le prestazioni migliori di ResNet sono state ottenute utilizzando l'opzione di finetuning del modello con un valore di learning rate pari a 0.0001.

Un resoconto dettagliato circa i valori di precision recall ed f1-score sul validation set per le singole classi ed i valori totali di accuracy dell'esperimento migliore con il modello ResNet è mostrato a seguire:



```
=== Classification Report ===
                precision    recall  f1-score   support

         Bread       0.93      0.87      0.90       300
  Dairy product       0.94      0.93      0.94       300
       Dessert       0.86      0.90      0.88       300
           Egg       0.89      0.90      0.90       300
    Fried food       0.92      0.93      0.93       300
          Meat       0.88      0.95      0.91       300
  Noodles-Pasta       0.96      0.95      0.96       300
          Rice       0.94      0.92      0.93       300
       Seafood       0.93      0.89      0.91       300
          Soup       0.96      0.96      0.96       300
Vegetable-Fruit       0.96      0.98      0.97       300

      accuracy                           0.93      3300
     macro avg       0.93      0.93      0.93      3300
  weighted avg       0.93      0.93      0.93      3300
```

Come si evince dall'immagine anche le 'criticità' presenti nel modello precedente riguardanti Bread, Dairy product e Dessert sono state risolte, e adesso anche le performance su queste classi sono soddisfacenti.

6.3. Esperimenti su EfficientNet

Per completare il task assegnato, si è deciso di effettuare ulteriori esperimenti, nonostante gli ottimi risultati già ottenuti, con un modello ancor più complesso. Per quanto riguarda le sperimentazioni con questo modello, è stato deciso di non effettuare l'esperimento di training utilizzando il metodo di features extraction, dati i riscontri ottenuti nei due precedenti modelli.

- **Learning Rate: 0.0001:**

  Lo schema d'azione è rimasto invariato, utilizzando, come di consueto, lo stesso numero di epoche e lo stesso learning rate (0.0001) che si è dimostrato essere il più efficace negli esperimenti con i precedenti modelli.

Come da previsione, seppur con un tempo di training superiore, i risultati sono stati leggermente migliori rispetto al medesimo esperimento effettuato con il modello ResNet.

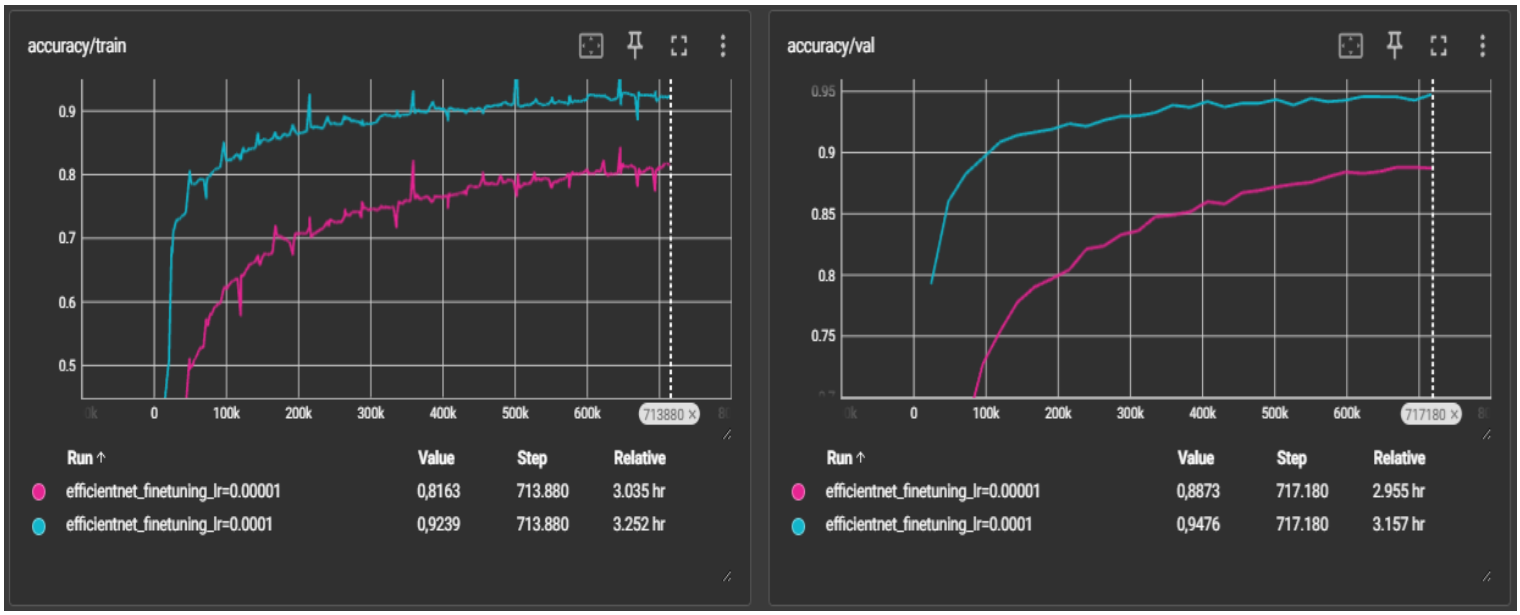
Il valore di accuracy ottenuto alla fine della procedura di training è stato di 0.947.

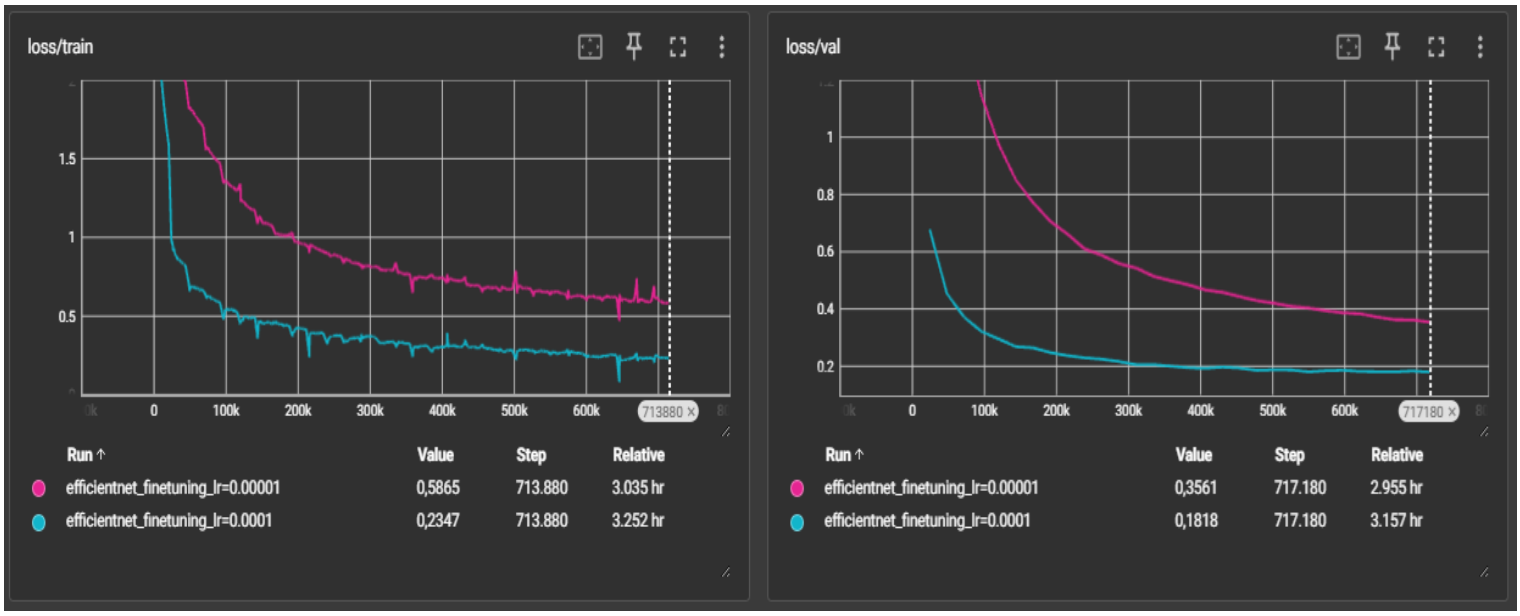
- **Learning Rate: 0.00001:**

  Anche in questo caso, data la maggior complessità del modello, si è deciso di effettuare un ulteriore esperimento, allenando tutta la rete e abbassando ancor di più il learning rate. I risultati, come ci si aspettava, si sono dimostrati leggermente inferiori rispetto ai precedenti, poiché il modello non è riuscito a generalizzare abbastanza sugli esempi di input. Il valore di accuracy finale si è attestato intorno al 0.887.

I miglioramenti ottenuti utilizzando quest'ultimo modello sono stati minimi, quindi, ritenendo i risultati ottenuti soddisfacenti, sulla base dell'esperienza maturata con gli esperimenti precedenti e a causa dell'elevata richiesta di tempo e risorse computazionali, si è deciso di concludere la fase di sperimentazione.

I dati citati sopra, sono di seguito riportati graficamente:





Anche in questo caso, i risultati migliori sono stati ottenuti con un numero di epoche totali di training fissato a 30, e un valore di learning rate pari a 0.0001.

Un resoconto dettagliato circa i valori di precision recall ed f1-score sul validation set per le singole classi ed i valori totali di accuracy dell'esperimento migliore con il modello EfficientNet è mostrato a seguire:

```
=== Classification Report ===
                precision    recall  f1-score   support

         Bread       0.93      0.94      0.94       300
  Dairy product      0.94      0.93      0.93       300
       Dessert       0.91      0.91      0.91       300
           Egg       0.93      0.93      0.93       300
    Fried food       0.94      0.94      0.94       300
          Meat       0.92      0.97      0.94       300
  Noodles-Pasta      0.97      0.97      0.97       300
          Rice       0.96      0.96      0.96       300
       Seafood       0.98      0.92      0.95       300
          Soup       0.97      0.98      0.98       300
Vegetable-Fruit      0.98      0.98      0.98       300

      accuracy                           0.95      3300
     macro avg       0.95      0.95      0.95      3300
  weighted avg       0.95      0.95      0.95      3300
```

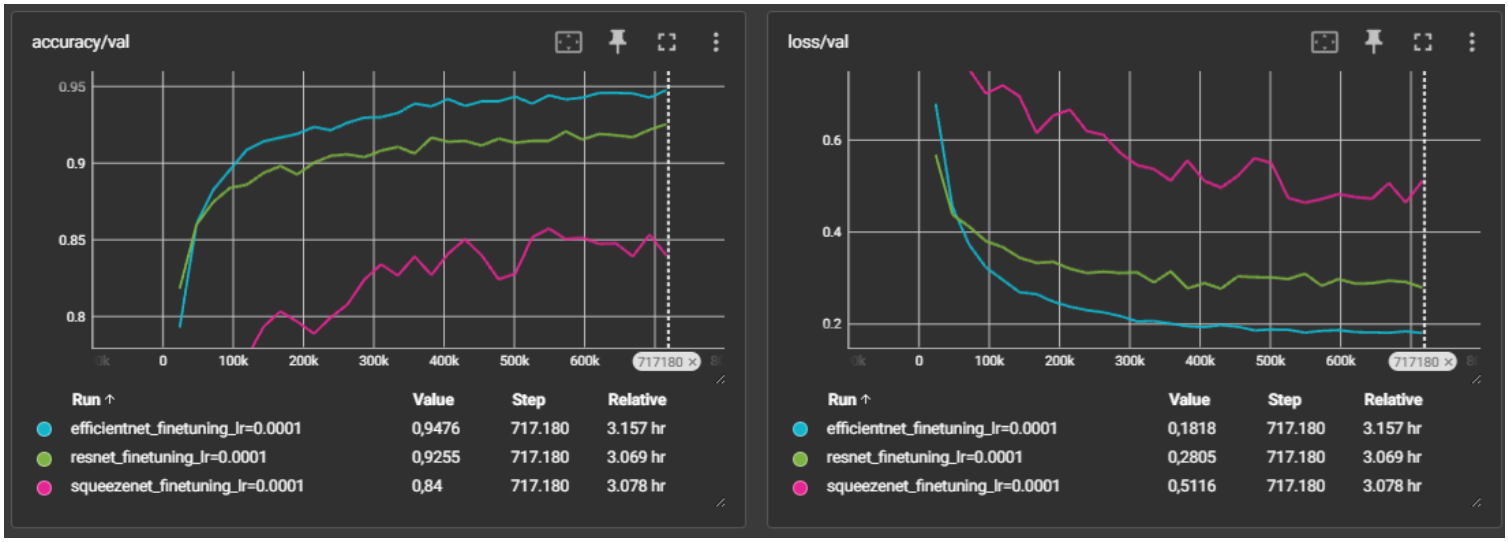
Tutti gli esperimenti condotti hanno utilizzato come valore dell' iper-parametro weight decay dell'ottimizzare lo 0. L'idea era di iniziare proprio con il valore 0 ed eventualmente aumentarlo qualora si fossero presentati segni di overfitting, ma siccome ciò non è accaduto è stato lasciato invariato.

Una tabella riassuntiva di tutti gli esperimenti condotti, per una maggiore chiarezza, è riportata di seguito (evidenziato il miglior esperimento per ogni modello):

Nome Modello	Tipo di Train	Numero Epoche	Learning Rate	Valore di Accuracy
Squeezenet	Finetuning	15	0.01	0.090
SqueezeNet	Finetuning	30	0.001	0.755
SqueezeNet	Features Extractor	30	0.001	0.711
SqueezeNet	Finetuning	30	0.0001	0.840
SqueezeNet	Features Extractor	30	0.0001	0.797
ResNet18	Finetuning	30	0.0001	0.925
ResNet18	Features Extractor	30	0.0001	0.792
ResNet18	Finetuning	30	0.00001	0.896
EfficientNet-B0	Finetuning	30	0.0001	0.947
EfficientNet-B0	Finetuning	30	0.00001	0.887

Table 1: Tabella con modelli di training ed accuratezza.

Il comportamento dei tre migliori modelli su training e validation set, visualizzato tramite utilizzo di grafici realizzati con Tensorboard, è riportato a seguire:



Infine, sono state effettuate delle prove sul test set, utilizzando i tre modelli riportati nel grafico sopra, ottenendo i seguenti risultati:

Nome Modello	Tipo di Train	Learning Rate	Valore di Accuracy
SqueezeNet	Finetuning	0.0001	0.840
ResNet18	Finetuning	0.0001	0.920
EfficientNet-B0	Finetuning	0.0001	0.940

Table 2: Tabella con risultati ottenuti sul test set.

Di seguito sono presentati in dettaglio i valori ottenuti per ogni singola classe:

- **SqueezeNet**:

```
=== Classification Report ===
                precision    recall  f1-score   support

         Bread       0.84      0.91      0.88       100
  Dairy product      0.94      0.66      0.78       100
       Dessert       0.85      0.88      0.86       100
           Egg       0.93      0.90      0.91       100
    Fried food       0.95      0.73      0.82       100
          Meat       0.88      0.89      0.89       100
  Noodles-Pasta      0.58      0.93      0.72       100
          Rice       0.96      0.88      0.92       100
       Seafood       0.88      0.72      0.79       100
          Soup       0.96      0.82      0.89       100
Vegetable-Fruit      0.71      0.89      0.79       100

      accuracy                           0.84      1100
     macro avg       0.86      0.84      0.84      1100
  weighted avg       0.86      0.84      0.84      1100
```

- **ResNet**:

```
=== Classification Report ===
                precision    recall  f1-score   support

         Bread       0.92      0.95      0.94       100
  Dairy product       0.95      0.93      0.94       100
       Dessert       0.96      0.97      0.97       100
           Egg       0.88      0.98      0.92       100
    Fried food       0.92      0.92      0.92       100
          Meat       0.97      0.95      0.96       100
  Noodles-Pasta       0.89      0.89      0.89       100
          Rice       0.89      0.95      0.92       100
       Seafood       0.87      0.86      0.86       100
          Soup       0.98      0.83      0.90       100
Vegetable-Fruit       0.94      0.91      0.92       100

      accuracy                           0.92      1100
     macro avg       0.92      0.92      0.92      1100
  weighted avg       0.92      0.92      0.92      1100
```

- **EfficientNet**:

```
=== Classification Report ===
                precision    recall  f1-score   support

         Bread       0.95      0.97      0.96       100
  Dairy product       0.94      0.96      0.95       100
       Dessert       0.97      0.95      0.96       100
           Egg       0.92      0.99      0.95       100
    Fried food       0.95      0.96      0.96       100
          Meat       0.99      0.98      0.98       100
  Noodles-Pasta       0.86      0.91      0.88       100
          Rice       0.94      0.97      0.96       100
       Seafood       0.90      0.86      0.88       100
          Soup       1.00      0.92      0.96       100
Vegetable-Fruit       0.95      0.88      0.91       100

      accuracy                           0.94      1100
     macro avg       0.94      0.94      0.94      1100
  weighted avg       0.94      0.94      0.94      1100
```

Il comportamento dei modelli osservato durante le fasi di training e validation è stato completamente rispettato anche su dati di test, a conferma della bontà del lavoro svolto in tutte le fasi del progetto.

7. Codice

Il notebook “FoodVision_Notebook.ipynb” che si trova all’interno della cartella “notebooks” è stato organizzato in varie sezioni logiche, ognuna con un ruolo preciso nello sviluppo del progetto.

- **Pre-processing dei dati**: in questa sezione vengono gestiti i dati in input per preparare i dataset. Sono incluse operazioni come la normalizzazione, suddivisione nei tre dataset (training, validation, test), trasformazioni specifiche per le fasi di allenamento e valutazione, definizione dei data loader per lavorare a blocchi di batch.

- **Definizione delle procedure di train, test e valutazione:** qui sono definite le funzioni coinvolte nel ciclo di allenamento e valutazione. Nella funzione di training sono specificati i criteri di ottimizzazione, il calcolo della loss, l’aggiornamento dei gradienti e la registrazione su tensorboard dei log di accuracy e macro f1-score per monitorare l’andamento del training. La funzione di test ha lo scopo di ottenere le predizioni sul task dato un modello già allenato e un data loader relativo ad un certo dataset. Infine la funzione di evaluation prende in input le predizioni del modello e le etichette reali e stampa in output un report con le varie metriche di valutazione e la matrice di confusione rappresentata sia in forma numerica che in forma grafica a colori.

- **Adattamento dei modelli pre-addestrati:** per una maggiore leggibilità ci sono tre sezioni separate, una per ogni modello (Squeezenet, Resnet, Efficientnet). Ogni modello viene caricato e adattato per risolvere il nostro specifico task di classificazione multiclasse. In particolare, l’ultimo layer di ogni rete viene modificato per gestire 11 classi.

- **Definizione della funzione generale per gli esperimenti:** per evitare duplicazioni è stata implementata una funzione generale che incapsula la procedura comune di un esperimento: costruzione del modello e applicazioni della funzione di training e valutazione. Questo approccio modulare consente di parametrizzare gli esperimenti (diversa architettura, learning rate, numero di epoche) e renderli facilmente ripetibili.

- **Esperimenti:** le tre sezioni finali contengono gli esperimenti applicati concretamente ai tre modelli adottati. Ogni esperimento esegue la funzione generale descritta sopra.

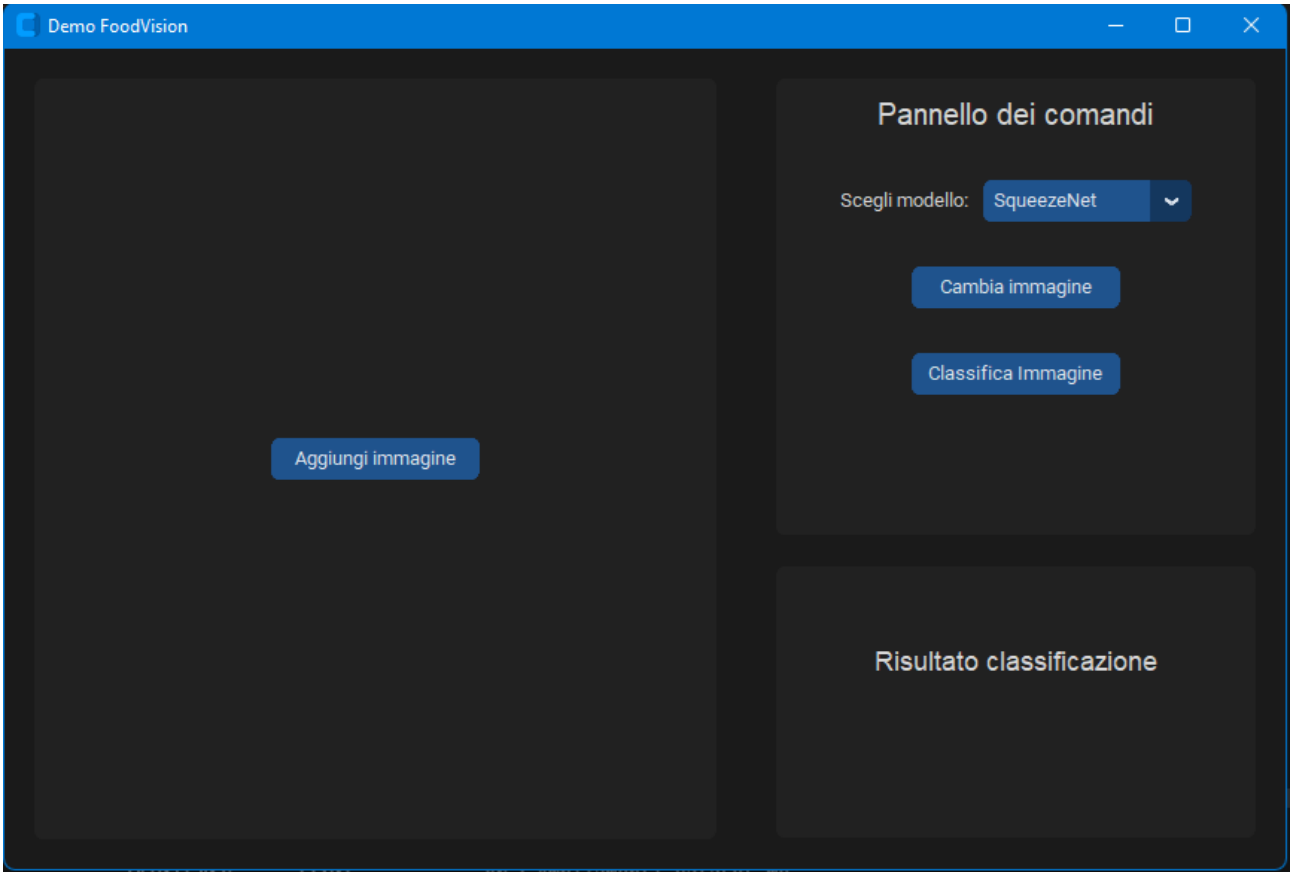
8. Demo

La demo che mostra il funzionamento dell'algoritmo è stata sviluppata in Python, utilizzando la libreria grafica Tkinter per creare finestre di dialogo e rendere più semplice ed intuitiva possibile l'esperienza dell'utente.

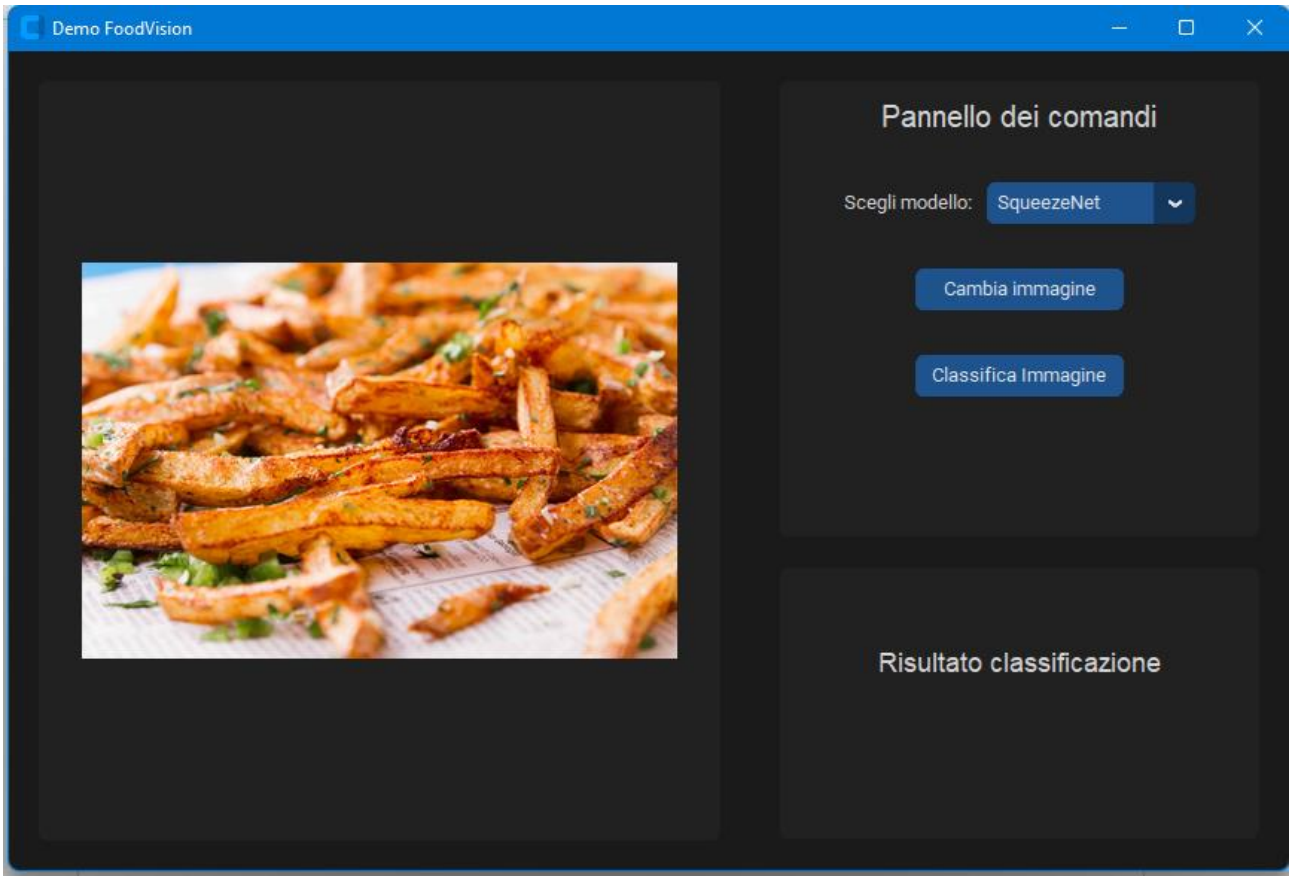
Prima di avviare la demo, è necessario spostarsi nella directory "demo" del progetto ed eseguire da terminale il comando "***pip install -r requirements.txt***" utile all'installazione delle dipendenze necessarie, inserite nel file di testo *requirements.txt*, per il corretto funzionamento dell'applicazione.

Una volta fatto ciò, è possibile avviare l'applicazione tramite riga di comando utilizzando il comando "***python .\demo.py***". La prima volta che viene eseguita l'applicazione potrebbe essere necessario un po 'di tempo in più perchè qualora i modelli pre allenati di base non fossero già presenti in cache saranno scaricati automaticamente dall'hub di pytorch.

L'applicazione presenterà all'utente una finestra di dialogo divisa in sezioni, come mostrato di seguito:



Il pannello sinistro permette, tramite click del pulsante “Aggiungi immagine”, di caricare l'immagine desiderata per testare l’algoritmo. Una volta scelta, l’immagine verrà visualizzata nel pannello come mostrato a seguire:

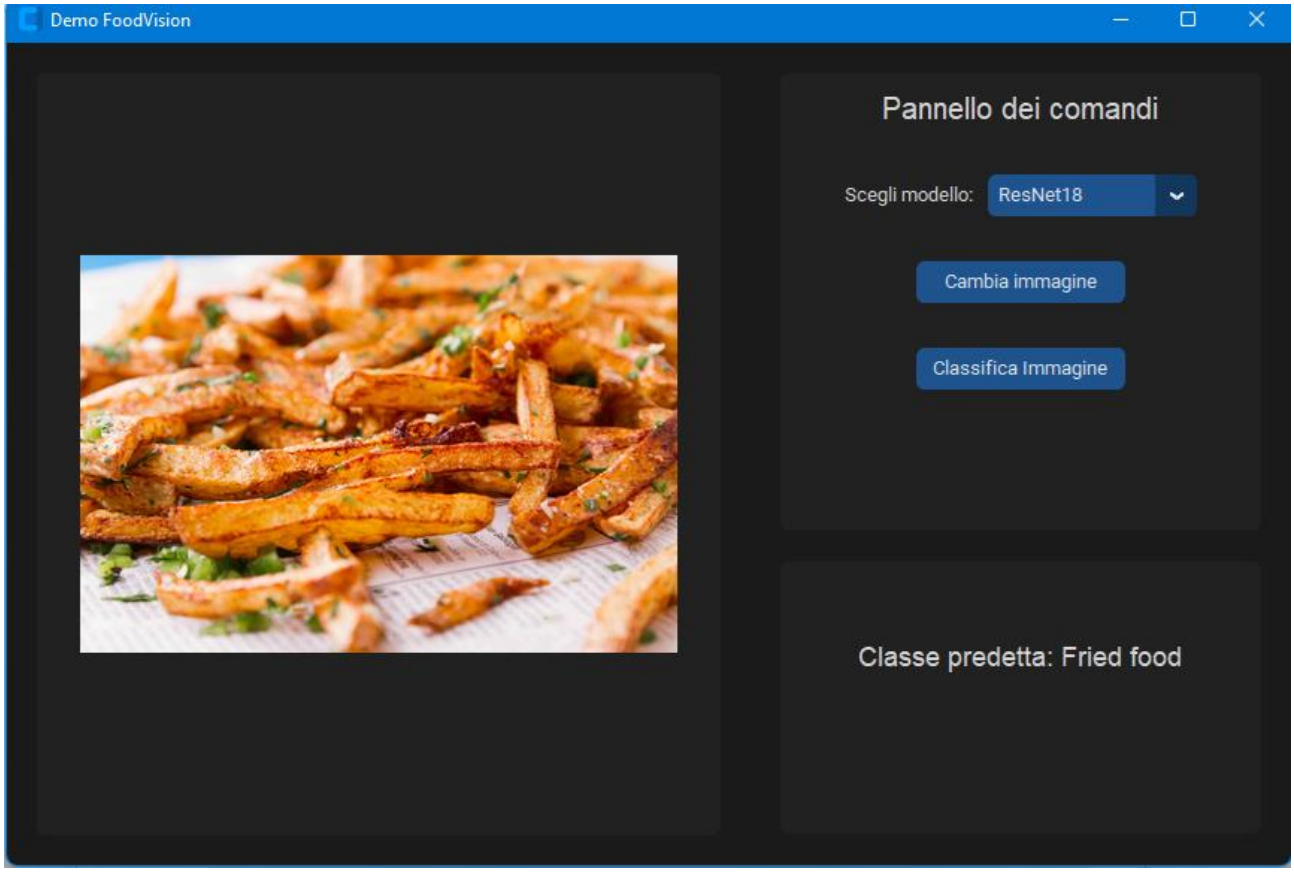


Dopo il caricamento dell'immagine, tramite il “Pannello dei comandi” sulla parte destra in alto della finestra, è possibile:

- **Scegliere il modello da utilizzare per effettuare la classificazione**: l’utente può scegliere, tramite l’utilizzo dell’apposito menù a tendina, tra le versioni migliori (sopra evidenziate nella tabella 1) dei tre modelli già discussi.
- **Cambiare l’immagine da classificare**: è possibile per l’utente cambiare in qualsiasi momento l’immagine scelta, tramite click sul bottone “Cambia immagine", senza dover quindi riavviare l’applicazione ad ogni classificazione.
- **Testare l’algoritmo di classificazione**: al click del bottone “Classifica immagine” verrà eseguito il task di classificazione utilizzando il modello scelto l'immagine di input.

Il risultato della classificazione verrà mostrato nella parte inferiore destra della finestra, fornendo all'utente la classe, tra le 11 possibili, alla quale è stata associata l'immagine di input.

Il risultato dopo un'esecuzione completa è mostrato di seguito:



È doveroso ricordare che il risultato potrebbe variare sulla stessa immagine di input in base al modello scelto, e che il modello SqueezeNet potrebbe commettere più errori rispetto ai due modelli più complessi proprio in virtù degli esperimenti condotti e illustrati in dettaglio in precedenza.

N.B. Per il corretto funzionamento dell'applicazione, è importante che la cartella di progetto venga scaricata in tutte le sue componenti e non venga alterata in alcun modo, poiché il modello da utilizzare nella demo, scelto dall'utente, è ricavato tramite riferimento al file .pth del modello corrispondente inserito nella directory "modelli" del progetto.

In allegato alla relazione verrà fornito un breve video guida dimostrativo che illustra il corretto utilizzo della demo e le operazioni che è possibile eseguire.

9. Conclusioni

In questo progetto è stato sviluppato un classificatore di immagini di cibi basato su reti convoluzionali. La prima fase è stata quella di acquisizione del dataset. Esso è costituito da 11 classi ed è stato ottenuto dall'unione di più dataset di riferimento dello stato dell'arte filtrando opportunamente le parti da aggiungere, e integrando le classi carenti con immagini raccolte da internet. Inoltre, il dataset di test è stato creato manualmente, così come richiesto dalla consegna, e per farlo sono state selezionate immagini online.

La fase successiva è stata quella in cui abbiamo allenato tre architetture di complessità crescente - Squeezenet, Resnet ed Efficientnet - valutando e confrontando le performance sul nostro dataset.

Tra le lezioni apprese durante lo sviluppo del progetto troviamo:

- **Importanza della qualità e del bilanciamento del dataset**. All'inizio nelle primissime fasi sono stati osservati risultati di accuracy poco affidabili a causa della presenza di classi sbilanciate: questo ci ha portato a fare delle integrazioni di immagini per le classi sotto-rappresentate e a prestare maggiore attenzione allo split di train/val/test set. In questo modo sono migliorati sensibilmente i valori delle metriche adottate e la stabilità delle curve.
- **Transfer learning e data augmentation sono molto efficaci**. L'uso di pesi pre-allenati è importante perché permette di riutilizzare delle rappresentazioni utili generiche già apprese precedentemente, permette una convergenza più veloce riducendo il numero di epoche necessarie per raggiungere buone prestazioni, e permette di migliorare la generalizzazione quando si hanno a disposizione relativamente pochi dati. Le trasformazioni di data augmentation sono fondamentali per aumentare la robustezza del modello e fornire un meccanismo di regolarizzazione. Tra le varie prove effettuate nelle primissime fasi di sviluppo ci sono stati anche degli esperimenti non andati a buon fine in cui la data augmentation era completamente assente o troppo 'pesante'. Nel primo caso è stato riscontrato un overfitting evidente dopo poche epoche, mentre nel secondo caso è stato riscontrato un sostanziale peggioramento delle performance perché le immagini venivano distorte

troppo introducendo caratteristiche non reali e rumore eccessivo che non consentivano di discriminare bene.

- **Più metriche per maggiore robustezza**. Monitorare altre metriche oltre all'accuracy si è rivelato molto utile, infatti f1-score macro/micro e matrice di confusione ci ha permesso di individuare facilmente le classi su cui era necessario intervenire.
- **I vincoli computazionali condizionano le scelte di training.** La limitata potenza delle macchine ci ha imposto un batch size massimo di 32 (perchè altrimenti non bastava la memoria), tempi di training relativamente lunghi e una selezione più conservativa dei modelli da sperimentare.

Per quanto riguarda i possibili miglioramenti futuri apportabili al metodo adottato, potrebbe essere interessante utilizzare modelli di ultima generazione come ibridi Transformer/CNN, o ampliare il dataset usando dei dati sintetici generati tramite GAN per verificare eventuali cambiamenti nei risultati.

10. Appendice

10.1. Struttura delle directory

L'organizzazione del progetto segue una struttura a cartelle pensata per separare chiaramente i diversi componenti: dati, modelli, log, codice di sviluppo, applicazione e documentazione. Di seguito la struttura e il contenuto di ciascuna directory:

- **dataset/** : I dati sono organizzati in apposite cartelle chiamate training,validation e test. Ognuna di queste cartelle contiene le relative immagini organizzate in sottocartelle corrispondenti alle classi.

- **modelli/** : raccoglie i modelli restituiti in output dagli esperimenti eseguiti. Ogni sottocartella porta il nome dell'esperimento e contiene il file .pth con i pesi del modello addestrato.

- **logs/** : contiene i log generati da Tensorboard durante l'addestramento. Ogni esperimento ha una propria sottocartella denominata coerentemente, contenente il relativo file di log.

- **notebooks/** : include il materiale di sviluppo. In particolare il file 'FoodVision_Notebook.ipynb' documenta tutte le fasi di sviluppo del progetto (preprocessing dei dati, training, valutazione, esperimenti, ecc.); mentre il file 'requirements.txt include le dipendenze necessarie all'esecuzione del notebook.

- **demo/** : contiene l'applicazione eseguibile che permette di provare il software sviluppato. In particolare il file 'demo.py' contiene il codice dell'applicazione , mentre il file 'requirements.txt' è dedicato alle sole dipendenze necessarie per l'esecuzione della demo.

- **relazione/** : contiene i file .doc e .pdf relativi alla relazione del progetto.