Guess The Trash

Chiara Cippitelli
Benedetta Rogato

Corso di Sistemi Digitali M - Anno Accademico 2022/2023

Indice

1 Abstract

Guess The Trash è un'applicazione Android che permette di classificare i rifiuti in base al materiale di cui sono composti e di indicare come smaltirli in maniera sostenibile.
Per la classificazione dei rifiuti è stata addestrata una rete neurale convoluzionale (CNN) attraverso l'utilizzo di un dataset di immagini di rifiuti, suddivise in sottoclassi in base al loro materiale. L'addestramento è stato effettuato mediante il framework TensorFlow.
Il modello così ottenuto è stato quindi convertito nel formato TensorFlow Lite e infine utilizzato per lo sviluppo dell'applicazione Android, mediante l'applicativo Andorid Studio.

2 Dataset

L'obiettivo del progetto consiste nel classificare le immagini di diversi rifiuti in sei possibili categorie: carta, cartone, plastica, vetro, metallo e indifferenziata. Per fare ciò è stato selezionato un dataset, precostituito e reperibile su Kaggle [Dataset], su cui allenare la rete neurale.
Il dataset è composto da approssimativamente 2500 immagini organizzate in 3 sottoinsiemi:

- Training Set: insieme di immagini utilizzato nella fase iniziale di allenamento della rete. Questo sottoinsieme contiene la maggior parte delle immagini del dataset in modo che la rete possa essere allenata su un campione molto ampio.
- Validation Set: insieme di immagini utilizzato per validare i risultati ottenuti durante la fase di training della rete.
- Testing Set: insieme di immagini utilizzato nella fase successiva all'allenamento e alla validation della rete per verificare la correttezza dei risultati forniti dalla rete.

Ognuno dei 3 set è a sua volta suddiviso nelle 6 classi a cui possono essere ricondotte le immagini dei rifiuti. Per migliorare le prestazioni della rete neurale e per controllare la correttezza del dataset, è stato necessario verificare manualmente che le immagini fossero pertinenti alla categoria assegnata e rimuovere quindi quelle non congrue.
Date le dimesioni ridotte del dataset è stato necessario utilizzare delle tecniche di data augmentation, in modo da ampliare la dimensione del dataset. Gli augmented data arricchiscono il dataset di training delle reti neurali senza raccogliere nuovi elementi, ma applicando ai dati già esistenti dei cambiamenti casuali controllati, realizzandone quindi delle copie modificate. In questo modo è stato possibile ampliare la dimensione del dataset e prevenire l'overfitting della rete.

```
train_datagen = ImageDataGenerator(validation_split= 0.1,
                                   rescale = 1./255,
                                  shear_range = 0.2,
                                  zoom_range = 0.2,
                                  horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

validation_set = train_datagen.flow_from_directory(validation_path,
                                                    target_size = (384, 384),
                                                    subset='validation')

training_set = train_datagen.flow_from_directory(train_path,
                                                  target_size = (384, 384),
                                                  batch_size = 16,
                                                  class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_path,
                                            target_size = (384, 384),
                                            batch_size = 16,
                                            class_mode = 'categorical')
```

Figure 1: Codice eseguito per effettuare la data augmentation.

3 Realizzazione e Training della rete

Per lo sviluppo e l'addestramento del modello della rete neurale sono stati utilizzati TensorFlow 2.9.2 e, come libreria di supporto, Keras. Per ottenere alte prestazioni è stata utilizzata EfficentNetV2-s, una delle reti offerte da EfficientNetV2 [EfficientNetV2], una famiglia di modelli per la classificazione di immagini. Il compito principale di questa famiglia di reti è quello di garantire una miglior efficienza parametrica, una maggiore accuratezza e di ridurre il tempo di addestramento della rete, attraverso l'utilizzo di NAS (Neural Architecture Search) e la tecnica del progressive learning.
L'idea alla base del progetto della rete, infatti, era la costruzione di un modello sequenziale, ovvero un modello basato su una serie di Layer inseriti uno dopo l'altro, attraverso i quali un dato di input fluisce in maniera sequenziale finchè non diventa output.

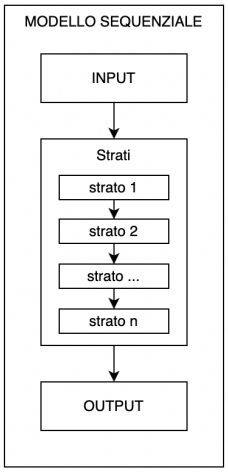

Figure 2: Struttura del modello sequenziale

Sfruttando EfficentNetV2-s, non è stato necessario aggiungere manualmente ogni layer, ma è bastato incapsulare tale rete nel modello stesso.

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=IMAGE_SIZE + (3,)),
    hub.KerasLayer(model_handle, trainable=do_fine_tuning),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(6,
                          kernel_regularizer=tf.keras.regularizers.l2(0.0001))
])
```

Figure 3: Codice utilizzato per la realizzazione del modello

Per compilare il modello è stato utilizzato come ottimizzatore SGD (Stochastic Gradient Descent) con learning rate pari a 0.005 e momentum pari a 0.9; per calcolare il valore di loss invece è stata utilizzata la funzione CategoricalCrossentropy di Keras con label smoothing pari a 0.1.

```
model.compile(
  optimizer=tf.keras.optimizers.SGD(learning_rate=0.005, momentum=0.9),
  loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True, label_smoothing=0.1),
  metrics=['accuracy'])
```

Figure 4: Codice eseguito per la compilazione

Infine il modello è stato addestrato su Google Colab per 100 epoche e con batch size pari a 16.

```
hist = model.fit(
    training_set,
    epochs=100,
    validation_data=validation_set
    )
```

Figure 5: Codice eseguito per il training della rete

Nel seguente grafico vengono mostrati i valori di accuracy del modello, in relazione all'avanzamento del training, calcolato sia sul validation set che sul training set. Si può notare che i risultati tendono a migliorare nella prima parte del training e successivamente tendono a stabilizzarsi a partire dalla quarantesima epoca. Al termine dell'addestramento entrambi i valori di accuracy sono risultati essere molto alti, aggirandosi attorno al 98%.

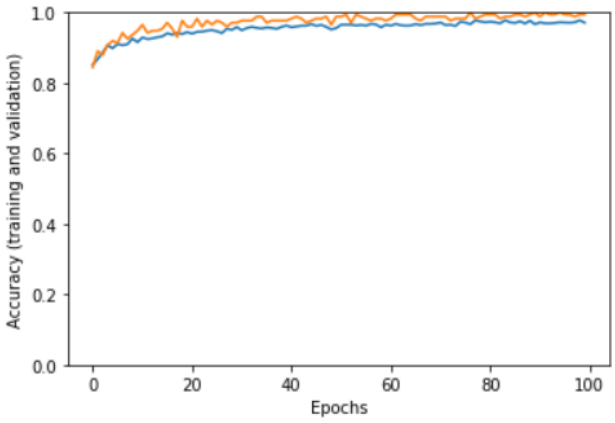

Figure 6: Grafico che mostra l'andamento del valore di accuracy calcolato sul training set (in blu) e sul validation set (in arancione).

4 Conversione del modello e quantizzazione

Per utilizzare il modello creato su una applicazione mobile, è stato necessario sfruttare il framework fornito da Tensorflow chiamato TensorFlow Lite. Quest'ultimo consente di convertire il modello in un formato FlatBuffer, ovvero un formato (estensione .tflite) efficiente, facilmente trasportabile e di dimensioni ridotte, per questo adatto a contesti distribuiti. Il modello è stato dunque compresso utilizzando come tecnica di quantizzazione la Dynamic range quantization, che converte i pesi della rete da Float32 a Int8. La dimensione finale del modello è pari a 22 MB. Di seguito è riportato il codice utilizzato.

```
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_path)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
lite_model_content = converter.convert()
with open(f"/content/gdrive/My Drive/TrainedModels/saved_model_384_.tflite", "wb") as f:
  f.write(lite_model_content)
print("Wrote %sTFLite model of %d bytes." %
      ("optimized ", len(lite_model_content)))
```

4.1 Risultati

Nonostante i vantaggi sopra descritti, il processo di conversione e quantizzazione riduce inevitabilmente l'accuracy della rete. Tuttavia, sono state analizzate le differenze tra le predizioni prodotte dal modello originale e il modello TFLite quantizzato e, come mostrato nelle figure che seguono, in realtà queste non sono significative. In entrambi i casi, infatti, il livello di accuratezza è decisamente alto.

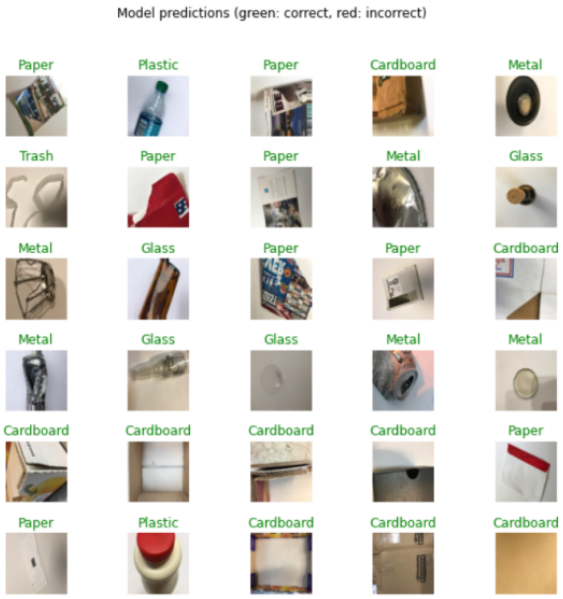

Figure 7: Predizioni generate dal modello originale.

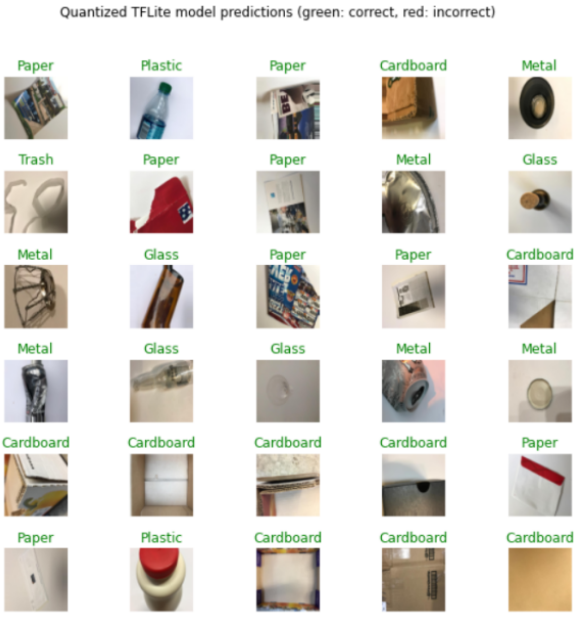

Figure 8: Predizioni generate dal modello .tflite quantizzato.

5 Applicazione Android

A partire dal modello TFLite quantizzato è stata sviluppata, mediante l'IDE Androido Studio, un'applicazione Android per dispositivi mobile , scritta in linguaggio Java.
Lo scopo dell'applicazione è permettere all'utente di ottenere una predizione sull'immagine di un rifiuto. Nello specifico, viene rilevato il materiale di cui è composto il rifiuto e in base a questo viene individuata la classe di appartenenza (Carta, Cartone, Plastica, Metallo, Indifferenziato e Vetro), dando modo all'utente di poter effettuare correttamente la raccolta differenziata.
L'immagine da fornire in input alla rete può essere catturata direttamente dalla fotocamera del dispositivo Android o, eventualmente, selezionata dalla galleria dello stesso. Di seguito si mostrano le schermate principali dell'applicazione.

Figure 9: Schermata di avvio da cui è possibile scegliere se caricare un'immagine dalla galleria o scattare una foto.

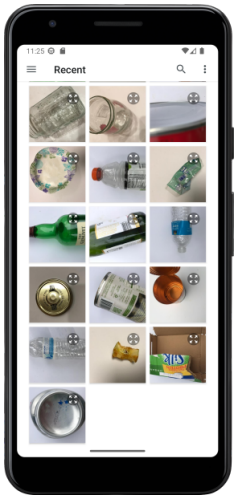

Figure 10: Schermata per scegliere un'immagine dalla galleria.

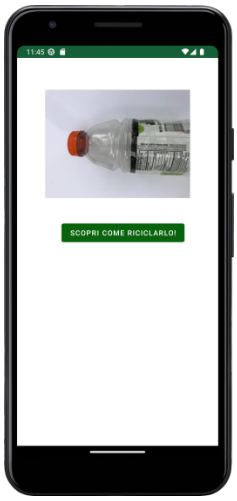

Figure 11: Una volta scelta l'immagine, viene mostrata una nuova schermata rappresentata in figura con un pulsante che permette di visualizzare la previsione.

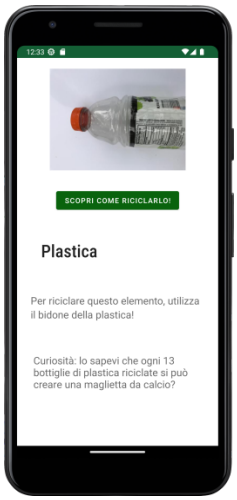

Figure 12: Schermata finale in cui viene mostrata la previsione del materiale di cui è composto il rifiuto, il modo corretto per smaltirlo e una curiosità sulle modalità di riciclo di quel rifiuto specifico.

6 Bibliografia

- [Dataset] - A. Santoro, G. Pagliara, D. B. Robin, M. Okuyar, https://www.kaggle.com/datasets/andreasantoro/split-garbage-dataset, 2019
- [EfficentNetV2] - https://github.com/google/automl/tree/master/efficientnetv2