Progetto di Classificazione delle Immagini con TensorFlow e Keras

Sergio Cucinotta

9 settembre 2024

1 Abstract

L'obiettivo del test è di sviluppare un modello di machine learning che possa classificare automaticamente se un'immagine contiene un cane o un gatto, raggiungendo il 90% di accuracy. Per restare coerente all'obiettivo definito, e non essendo chiara la presenza o meno di altre specie nel dataset, ho sviluppato il progetto concentrandolo sulla distinzione tra cani e gatti all'interno di un dataset contenente queste due specie.

Il modello ottimizzato ha mostrato un'accuratezza del 93% nel riconoscere con successo le immagini di entrambe le categorie. La metodologia include un'attenta pre-elaborazione dei dati, la progettazione di un'architettura CNN con accorgimenti anti-overfitting e una valutazione dettagliata delle prestazioni sul dataset di test separato.

2 Analisi e pre-elaborazione dei Dati

Il dataset iniziale conteneva un totale di 25.000 immagini, suddivise equamente tra cani e gatti.

Durante l'analisi preliminare, è stato individuato un gruppo di 1.578 immagini corrotte, che sono state rimosse per assicurare l'integrità e la qualità del dataset.

Successivamente, le immagini sono state ridimensionate a 128x128 pixel per ridurre il carico computazionale e migliorare le prestazioni. Il dataset è stato diviso in set di addestramento (80%, 18.560 immagini), set di validazione (20%, 4.640 immagini) e set di test (200 immagini), sempre suddivise equamente tra cani e gatti.

```
new_img_size = (128,128)
data_split = 0.2
batch_size = 64
seed_value = 42

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
```

```
    pets_images,
    validation_split=data_split,
    subset="training",
    seed=seed_value,
    image_size=new_img_size,
    batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    pets_images,
    validation_split=data_split,
    subset="validation",
    seed=seed_value,
    image_size=new_img_size,
    batch_size=batch_size)

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_images,
    seed=seed_value,
    image_size=new_img_size,
    batch_size=batch_size)
```

Questa fase di preparazione dei dati è stata essenziale per garantire una solida base di immagini bilanciate e di alta qualità su cui addestrare e valutare il modello di riconoscimento.

3 Sviluppo e valutazione del Modello

Il modello implementato è un classificatore sequenziale di reti neurali convoluzionali (CNN). È composto da diversi strati che svolgono diverse operazioni, come convoluzioni, normalizzazione, max pooling e Dropout, seguiti da strati densi.

3.1 Modello Iniziale

```
model = Sequential()

model.add(Conv2D(8, (3,3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(layers.Dropout(0.2))

model.add(Conv2D(16, (3,3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(layers.Dropout(0.2))

model.add(Conv2D(32, (3,3), activation='relu', kernel_initializer='
    he_uniform', padding='same'),)
model.add(layers.BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(layers.Dropout(0.2))
```

```
17
18 model.add(Flatten())
19 model.add(Dense(128, activation='relu', kernel_initializer='
       he_uniform'))
20 model.add(Dense(1, activation='sigmoid'))
21
22 model.compile(optimizer=Adam(learning_rate=0.001), loss='
       binary_crossentropy', metrics=['acc'])
23 history = model.fit(train_ds,validation_data = val_ds,epochs = 20 )
```

Il primo modello implementato è composto da tre strati convoluzionali: il primo strato convoluzionale ha 8 filtri 3x3, il secondo 16 e il terzo 32, sempre 3x3, e ognuno di essi è accompagnato da normalizzazione, MaxPooling2d e Dropout.

Dopodichè l'output dei layer convoluzionali viene convertito in un vettore piatto e si passa per due strati densi: il primo comprende 128 neuroni con attivazione ReLU, mentre il secondo ha un singolo neurone di output con attivazione sigmoide per la classificazione binaria (cane o gatto).

Per l'addestramento, il modello è stato compilato con l'ottimizzatore Adam, utilizzando un tasso di apprendimento di 0.001 e una funzione di loss binary_crossentropy. L'addestramento del modello è stato eseguito per 20 epoche.

Nonostante l'architettura del modello abbia mostrato una buona capacità di apprendimento iniziale, sono state evidenziate alcune chiare problematiche di overfitting.

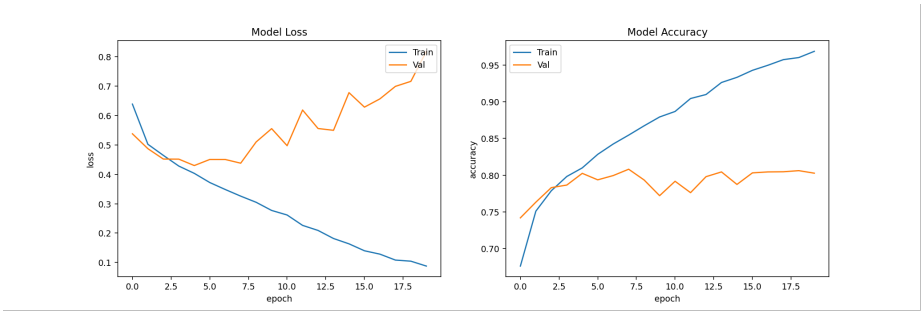

Figura 1: Perdita/Accuratezza di Allenamento e Validazione per Epoca - Primo Modello

Il valore della loss sul set di addestramento è notevolmente inferiore a quello del set di validazione. Ciò potrebbe indicare che il modello stia memorizzando i dati di addestramento senza generalizzare bene su nuovi dati. L'accuratezza sul set di addestramento è invece notevolmente maggiore rispetto a quella sul set di validazione. Ad esempio, l'accuratezza di addestramento è arrivata fino a circa il 97%, mentre quella di validazione è rimasta intorno al 80%.

Entrambi i grafici mostrano quindi un andamento molto diverso tra i set di addestramento e validazione, causato appunto da un evidente situazione di overfitting.

3.2 Ottimizzazione del modello

```
model = Sequential()

model.add(Conv2D(16, (3,3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(layers.Dropout(0.2))

model.add(Conv2D(32, (3,3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(layers.Dropout(0.2))

model.add(Conv2D(32, (3,3), activation='relu', kernel_initializer='
    he_uniform', padding='same'),)
model.add(layers.BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(layers.Dropout(0.2))

model.add(Conv2D(64, (3,3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(layers.Dropout(0.2))

model.add(Conv2D(256, (3,3), activation='relu', kernel_initializer=
    'he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(layers.Dropout(0.2))

model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_initializer='
    he_uniform'))
model.add(layers.Dropout(0.3))
model.add(layers.BatchNormalization())
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=Adam(learning_rate=0.001), loss='
    binary_crossentropy', metrics=['acc'])
history = model.fit(train_ds,validation_data = val_ds,epochs = 25 )
```

Il secondo modello è più profondo e complesso. Utilizza cinque strati convoluzionali con un numero crescente di filtri (16, 32, 32, 64, 256) per catturare e apprendere dettagli gerarchici di diversi livelli di astrazione nelle immagini. L'introduzione di una maggiore profondità e complessità nel modello può consentire una migliore estrazione delle caratteristiche distintive delle immagini, fornendo al modello una maggiore capacità di apprendimento e generalizzazione.

Inoltre, nel secondo modello è stata aggiunta una maggiore regolarizzazione con l'uso di un layer dropout più intenso (con un valore del 30%) e l'inclusione di strati aggiuntivi di normalizzazione BatchNormalization. Questi accorgimen-

ti aiutano a ridurre ulteriormente l'overfitting e a migliorare la capacità del modello di generalizzare su dati non visti durante l'addestramento.

Dopo 15 epoche, il modello ha già raggiunto un'accuratezza di circa il 92.17% sul set di addestramento e del 90.11% sul set di validazione.

```
Epoch 1/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 80s 269ms/step - loss:
    0.6493 - acc: 0.6546 - val_loss: 0.6265 - val_acc: 0.6640
Epoch 2/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 79s 270ms/step - loss:
    0.5175 - acc: 0.7457 - val_loss: 0.6588 - val_acc: 0.7259
Epoch 3/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 79s 270ms/step - loss:
    0.4584 - acc: 0.7830 - val_loss: 0.5049 - val_acc: 0.7603
Epoch 4/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 79s 271ms/step - loss:
    0.4110 - acc: 0.8105 - val_loss: 0.5494 - val_acc: 0.7457
Epoch 5/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 79s 272ms/step - loss:
    0.3784 - acc: 0.8270 - val_loss: 0.4082 - val_acc: 0.8162
Epoch 6/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 81s 277ms/step - loss:
    0.3492 - acc: 0.8452 - val_loss: 0.4305 - val_acc: 0.8009
Epoch 7/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 79s 272ms/step - loss:
    0.3212 - acc: 0.8624 - val_loss: 0.3663 - val_acc: 0.8416
Epoch 8/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 81s 280ms/step - loss:
    0.3002 - acc: 0.8688 - val_loss: 0.3460 - val_acc: 0.8502
Epoch 9/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 83s 284ms/step - loss:
    0.2834 - acc: 0.8782 - val_loss: 0.3275 - val_acc: 0.8532
Epoch 10/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 79s 273ms/step - loss:
    0.2612 - acc: 0.8876 - val_loss: 0.3183 - val_acc: 0.8597
Epoch 11/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 76s 262ms/step - loss:
    0.2380 - acc: 0.9022 - val_loss: 0.2992 - val_acc: 0.8711
Epoch 12/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 76s 262ms/step - loss:
    0.2276 - acc: 0.9030 - val_loss: 0.2965 - val_acc: 0.8763
Epoch 13/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 76s 262ms/step - loss:
    0.2154 - acc: 0.9091 - val_loss: 0.2600 - val_acc: 0.8877
Epoch 14/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 79s 271ms/step - loss:
    0.2014 - acc: 0.9148 - val_loss: 0.2946 - val_acc: 0.8733
Epoch 15/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 78s 267ms/step - loss:
    0.1906 - acc: 0.9217 - val_loss: 0.2428 - val_acc: 0.9011
Epoch 16/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 80s 275ms/step - loss:
    0.1773 - acc: 0.9276 - val_loss: 0.3691 - val_acc: 0.8584
Epoch 17/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ - 80s 276ms/step - loss:
    0.1658 - acc: 0.9325 - val_loss: 0.2610 - val_acc: 0.8938
Epoch 18/25
```

```
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79s 270ms/step - loss:
    0.1605 - acc: 0.9346 - val_loss: 0.2711 - val_acc: 0.9002
Epoch 19/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78s 270ms/step - loss:
    0.1502 - acc: 0.9386 - val_loss: 0.2561 - val_acc: 0.9028
Epoch 20/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 80s 275ms/step - loss:
    0.1384 - acc: 0.9442 - val_loss: 0.2646 - val_acc: 0.9011
Epoch 21/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 80s 276ms/step - loss:
    0.1328 - acc: 0.9481 - val_loss: 0.2671 - val_acc: 0.9011
Epoch 22/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82s 281ms/step - loss:
    0.1230 - acc: 0.9505 - val_loss: 0.2901 - val_acc: 0.8912
Epoch 23/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84s 290ms/step - loss:
    0.1153 - acc: 0.9543 - val_loss: 0.3207 - val_acc: 0.8851
Epoch 24/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82s 283ms/step - loss:
    0.1126 - acc: 0.9569 - val_loss: 0.2686 - val_acc: 0.9024
Epoch 25/25
290/290 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81s 280ms/step - loss:
    0.1070 - acc: 0.9593 - val_loss: 0.2637 - val_acc: 0.9060
```

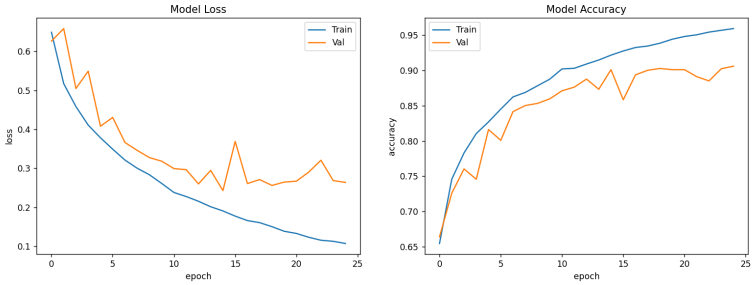

Figura 2: Perdita/Accuratezza di Allenamento e Validazione per Epoca - Secondo modello

Entrambi i grafici mostrano un buon andamento generale, con perdite decrescenti e aumenti di accuratezza nelle epoche successive. Si può notare un leggero divario tra le performance sui set di addestramento e validazione, suggerendo una potenziale leggera sovrapposizione e un livello minimo di overfitting.

4 Valutazione e risultati del modello

```python
import numpy as np

y_pred = []
y_true = []

for images, labels in test_ds:
```

```
    predictions = model.predict(images)
    y_pred.extend(predictions)
    y_true.extend(labels.numpy())

y_pred = np.array(y_pred).reshape(-1)
y_true = np.array(y_true)
y_pred_classes = np.where(y_pred < 0.5, 0, 1)

from sklearn.metrics import classification_report

print(classification_report(y_true, y_pred_classes))
```

Quest'ultima parte di codice esegue l'analisi delle prestazioni sul dataset di test del modello addestrato. Vengono effettuate previsioni sul dataset di test e valutate le performance del modello confrontando le previsioni con le etichette reali.

La funzione classification_report di sklearn fornisce una valutazione dettagliata delle metriche di precisione, richiamo e F1-score.

L'output mostra che il modello ha ottenuto un'accuratezza complessiva del 93%. Tale valutazione è supportata da metriche bilanciate di precisione e richiamo per entrambe le classi, indicando una buona capacità del modello nel riconoscere sia i cani che i gatti all'interno del dataset di test.

```
2/2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       100
           1       0.93      0.92      0.92       100

    accuracy                           0.93       200
   macro avg       0.93      0.93      0.92       200
weighted avg       0.93      0.93      0.92       200
```