

Questo progetto di deep learning si focalizza sull'addestramento di un modello utilizzando il dataset "Stanford Dogs", disponibile su TensorFlow Datasets. Questa collezione comprende immagini rappresentative di 120 razze canine, per un totale di 20.580 immagini, ciascuna accompagnata dalle rispettive etichette di classe. Grazie alla sua vasta copertura di razze e alla quantità di dati disponibili, il dataset "Stanford Dogs" si presenta come una buona risorsa per lo sviluppo di modelli di deep learning mirati al riconoscimento delle razze canine. Per ulteriori dettagli sul dataset e per accedere ai dati, è possibile consultare il seguente link:


[https://www.tensorflow.org/datasets/catalog/stanford\\_dogs?hl=it](https://www.tensorflow.org/datasets/catalog/stanford_dogs?hl=it)

Fai doppio clic (o premi Invio) per modificare

Questa cella di codice monta Google Drive all'interno dell'ambiente di esecuzione di Google Colab. Una volta eseguita, verrà richiesto di accedere al proprio account Google per ottenere l'autorizzazione. Una volta completata l'autenticazione, Google Drive sarà accessibile all'interno dell'ambiente di Colab e sarà possibile leggere e scrivere file da e verso Google Drive.

```
# Monta Google Drive
```

```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

Fai doppio clic (o premi Invio) per modificare

Questo blocco di codice inizia importando le librerie necessarie, come TensorFlow per il machine learning, il modulo os per le operazioni di sistema, datetime per la gestione delle date e pickle per la serializzazione dei dati. Successivamente, vengono definite due directory: checkpoint\_dir per salvare i modelli addestrati e log\_dir per i log di TensorBoard, utilizzati per monitorare le prestazioni dei modelli durante l'addestramento.

```
# Importa Librerie e Definire Directory di Salvataggio
```

```
import tensorflow as tf
import os
import datetime
import pickle
```

```
# Directory per salvare i modelli
checkpoint_dir = '/content/drive/My Drive/checkpoints'
os.makedirs(checkpoint_dir, exist_ok=True)
```

```
# Directory per i log di TensorBoard
log_dir = "/content/drive/My Drive/logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-
```

Fai doppio clic (o premi Invio) per modificare

**ModelCheckpoint:** Questo callback è configurato per salvare i pesi del modello durante l'addestramento. Viene specificato il percorso del file di salvataggio utilizzando `os.path.join` per combinare il percorso della directory dei checkpoint (`checkpoint_dir`) con il nome del file (`model_checkpoint.h5`). Impostando `save_weights_only=True`, si indica di salvare solo i pesi del modello anziché l'intero modello. Infine, `save_freq` viene impostato su 'epoch' per indicare che i pesi del modello devono essere salvati alla fine di ogni epoca.

**TensorBoard:** Questo callback è utilizzato per generare i log che possono essere visualizzati tramite TensorBoard per monitorare l'andamento dell'addestramento del modello. Viene specificato il percorso della directory di log (`log_dir`) e `histogram_freq` viene impostato su 1 per indicare che gli istogrammi dei tensori devono essere calcolati e salvati ogni epoca.

TensorBoard fornisce una visualizzazione grafica dettagliata dell'addestramento del modello, consentendo di monitorare le metriche e gli istogrammi dei tensori nel corso delle epoche.

```
# Definisci il Callback per Salvare il Modello e TensorBoard
```

```
# Callback per salvare il modello e TensorBoard
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=os.path.join(checkpoint_dir, 'model_checkpoint.h5'),
    save_weights_only=True,
    save_freq='epoch'
)
```

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

Fai doppio clic (o premi Invio) per modificare

1. **Caricamento del Dataset:** Utilizzando `tfds.load`, il codice carica il dataset "Stanford Dogs" e restituisce due oggetti: `dataset` e `info`. `with_info=True` viene utilizzato per ottenere anche le informazioni sul dataset.
2. **Struttura del Dataset:** Il codice stampa le dimensioni delle prime 100 immagini sia per il set di addestramento che per il set di test. Ciò viene fatto iterando sui primi 100 esempi di ciascun set (`dataset['train'].take(100)` e `dataset['test'].take(100)`), estraendo l'immagine da ciascun esempio e stampandone le dimensioni con `image.shape`.
3. **Altre Informazioni sul Dataset:** Il codice stampa ulteriori informazioni sul dataset, come il numero di classi (etichette), l'elenco delle classi e il tipo delle etichette. Queste

informazioni sono ottenute dall'oggetto info restituito dal caricamento del dataset.

```
# Carica il dataset Stanford Dogs
import tensorflow_datasets as tfds
dataset, info = tfds.load("stanford_dogs", with_info=True)

# Struttura del Dataset

# Stampa le dimensioni delle prime 100 immagini di addestramento
print("Dimensioni delle prime 100 immagini di addestramento:")
for example in dataset['train'].take(100):
    image = example['image']
    print(image.shape)

# Stampa le dimensioni delle prime 100 immagini di test
print("\nDimensioni delle prime 100 immagini di test:")
for example in dataset['test'].take(100):
    image = example['image']
    print(image.shape)

# Stampa altre informazioni utili sul mio dataset

# Stampa altre informazioni utili sul mio dataset
print("Informazioni aggiuntive sul dataset:")
print("Numero di classi (etichette):", info.features['label'].num_classes)
print("Elenco delle classi (etichette):", info.features['label'].names)
print("Tipo delle etichette:", info.features['label'].np_dtype)
```



Downloading and preparing dataset 778.12 MiB (download: 778.12 MiB, generated: Unkn

DI Completed...: 100% 1/1 [01:32<00:00, 92.96s/ url]

DI Size...: 100% 756/756 [01:32<00:00, 11.52 MiB/s]

DI Completed...: 100% 2/2 [01:13<00:00, 1.70s/ url]

DI Size...: 100% 20/20 [01:13<00:00, 6.80 MiB/s]

Extraction completed...: 100% 20583/20583 [01:13<00:00, 832.16 file/s]

Dataset stanford\_dogs downloaded and prepared to /root/tensorflow\_datasets/stanford  
Dimensioni delle prime 100 immagini di addestramento:

(500, 333, 3)

(367, 400, 3)

(500, 375, 3)

(321, 450, 3)

(333, 500, 3)

(375, 500, 3)

(500, 470, 3)

(207, 200, 3)

(375, 500, 3)

(500, 333, 3)

(500, 375, 3)

(375, 500, 3)

(370, 500, 3)

(375, 500, 3)

(333, 500, 3)

(375, 500, 3)

(500, 481, 3)

(454, 383, 3)

(180, 160, 3)

(500, 333, 3)

(375, 500, 3)

(375, 500, 3)

(500, 375, 3)

(333, 500, 3)

(333, 500, 3)

(500, 484, 3)

(500, 379, 3)

(600, 800, 3)

(500, 365, 3)

(375, 500, 3)

(375, 500, 3)

(375, 500, 3)

(333, 500, 3)

(333, 500, 3)

(300, 300, 3)  
(310, 401, 3)  
(375, 500, 3)  
(481, 500, 3)  
(375, 500, 3)  
(267, 360, 3)  
(333, 500, 3)  
(500, 344, 3)  
(333, 500, 3)  
(417, 500, 3)  
(333, 500, 3)  
(547, 389, 3)  
(333, 500, 3)  
(375, 500, 3)  
(525, 700, 3)  
(1879, 1388, 3)  
(747, 560, 3)  
(333, 500, 3)  
(375, 500, 3)  
(333, 500, 3)  
(334, 500, 3)  
(480, 467, 3)  
(500, 385, 3)  
(375, 500, 3)  
(375, 500, 3)  
(375, 500, 3)  
(505, 567, 3)  
(500, 333, 3)  
(333, 500, 3)  
(500, 375, 3)  
(364, 500, 3)  
(299, 271, 3)  
(500, 375, 3)  
(195, 380, 3)  
(500, 333, 3)  
(322, 500, 3)  
(333, 500, 3)  
(350, 500, 3)  
(500, 333, 3)  
(375, 500, 3)  
(333, 500, 3)  
(375, 500, 3)  
(320, 500, 3)  
(433, 500, 3)  
(334, 500, 3)  
(480, 640, 3)  
(196, 159, 3)  
(375, 500, 3)  
(375, 500, 3)  
(500, 333, 3)  
(333, 500, 3)  
(357, 500, 3)  
(500, 500, 3)  
(360, 480, 3)  
(333, 500, 3)  
(500, 334, 3)  
(375, 500, 3)  
(500, 360, 3)  
(333, 500, 3)  
(333, 500, 3)  
(384, 500, 3)

```
(334, 500, 3)
(622, 882, 3)
(500, 412, 3)
(500, 500, 3)
(500, 333, 3)
(500, 332, 3)
```

Dimensioni delle prime 100 immagini di test:

```
(332, 500, 3)
(240, 320, 3)
(497, 500, 3)
(375, 500, 3)
(360, 480, 3)
(357, 301, 3)
(375, 500, 3)
(500, 426, 3)
(417, 500, 3)
(283, 400, 3)
(600, 620, 3)
(282, 341, 3)
(500, 349, 3)
(375, 500, 3)
(500, 403, 3)
(190, 227, 3)
(375, 500, 3)
(325, 500, 3)
(332, 500, 3)
(332, 500, 3)
(454, 340, 3)
(500, 333, 3)
(290, 419, 3)
(333, 500, 3)
(334, 500, 3)
(333, 500, 3)
(380, 500, 3)
(375, 500, 3)
(375, 500, 3)
(375, 500, 3)
(375, 500, 3)
(375, 500, 3)
(375, 500, 3)
(175, 225, 3)
(500, 464, 3)
(333, 500, 3)
(333, 500, 3)
(316, 400, 3)
(280, 200, 3)
(443, 500, 3)
(206, 240, 3)
(375, 500, 3)
(299, 500, 3)
(446, 500, 3)
(500, 375, 3)
(375, 500, 3)
(331, 476, 3)
(375, 500, 3)
(347, 500, 3)
(200, 250, 3)
(333, 500, 3)
(333, 500, 3)
-----
```

```
(375, 500, 3)
(500, 383, 3)
(204, 214, 3)
(375, 400, 3)
(375, 500, 3)
(375, 500, 3)
(333, 500, 3)
(347, 500, 3)
(426, 300, 3)
(333, 500, 3)
(500, 333, 3)
(375, 500, 3)
(375, 500, 3)
(500, 489, 3)
(150, 106, 3)
(955, 1210, 3)
(360, 480, 3)
(375, 500, 3)
(375, 500, 3)
(269, 299, 3)
(232, 190, 3)
(375, 500, 3)
(397, 500, 3)
(500, 335, 3)
(290, 200, 3)
(375, 500, 3)
(375, 500, 3)
(375, 500, 3)
(375, 500, 3)
(336, 448, 3)
(478, 500, 3)
(500, 386, 3)
(500, 334, 3)
(500, 368, 3)
(333, 500, 3)
(500, 375, 3)
(333, 500, 3)
(375, 500, 3)
(240, 320, 3)
(375, 500, 3)
(375, 500, 3)
(332, 500, 3)
(400, 500, 3)
(338, 450, 3)
(215, 216, 3)
(300, 225, 3)
(375, 500, 3)
(389, 500, 3)
```

Informazioni aggiuntive sul dataset:

Numero di classi (etichette): 120

Elenco delle classi (etichette): ['n02085620-chihuahua', 'n02085782-japanese\_spanie▼



Fai doppio clic (o premi Invio) per modificare

Questo codice si occupa di visualizzare le prime 5 immagini di addestramento e di test dal dataset "Stanford Dogs". Utilizza Matplotlib per visualizzare le immagini insieme alle loro dimensioni e etichette.

La funzione `visualize_images` accetta il dataset come argomento e itera sui primi 5 esempi sia per il set di addestramento che per il set di test. Per ogni esempio, estrae l'immagine e l'etichetta corrispondente, stampa le dimensioni dell'immagine e visualizza l'immagine con il titolo contenente l'etichetta. Infine, utilizza `plt.show()` per mostrare ogni immagine.

```
# Visualization

# Carica matplotlib per la visualizzazione dei grafici
import matplotlib.pyplot as plt

# Definisci una funzione per visualizzare le immagini con dimensioni e label
def visualize_images(dataset):
    # Visualizza le prime 5 immagini di addestramento con dimensioni e label
    print("Visualizzazione delle prime 5 immagini di addestramento con dimensioni e label")
    for example in dataset['train'].take(5):
        image = example['image']
        label = example['label']
        print("Dimensioni dell'immagine:", image.shape)
        plt.imshow(image)
        plt.title("Train Label: {}".format(label))
        plt.axis('off')
        plt.show()

    # Visualizza le prime 5 immagini di test con dimensioni e label
    print("Visualizzazione delle prime 5 immagini di test con dimensioni e label:")
    for example in dataset['test'].take(5):
        image = example['image']
        label = example['label']
        print("Dimensioni dell'immagine:", image.shape)
        plt.imshow(image)
        plt.title("Test Label: {}".format(label))
        plt.axis('off')
        plt.show()

# Visualizza le immagini
visualize_images(dataset)
```





Visualizzazione delle prime 5 immagini di addestramento con dimensioni e label:  
Dimensioni dell'immagine: (500, 333, 3)

**Train Label: 36**



Dimensioni dell'immagine: (367, 400, 3)

**Train Label: 118**



Dimensioni dell'immagine: (500, 375, 3)

**Train Label: 46**





Dimensioni dell'immagine: (321, 450, 3)

**Train Label: 103**



Dimensioni dell'immagine: (333, 500, 3)

**Train Label: 113**





Visualizzazione delle prime 5 immagini di test con dimensioni e label:  
Dimensioni dell'immagine: (332, 500, 3)

Test Label: 67



Dimensioni dell'immagine: (240, 320, 3)

Test Label: 84



Dimensioni dell'immagine: (497, 500, 3)

Test Label: 57





Dimensioni dell'immagine: (375, 500, 3)

Test Label: 12



Dimensioni dell'immagine: (360, 480, 3)

Test Label: 88







Fai doppio clic (o premi Invio) per modificare

La variabile `missing_data` viene inizializzata a `False` per indicare che inizialmente non sono stati trovati dati mancanti. Viene quindi iterato attraverso le suddivisioni del dataset, ossia il set di addestramento e il set di test. Per ogni suddivisione, viene controllato il numero di esempi. Se il numero di esempi è pari a 0, significa che non ci sono dati presenti in quella suddivisione e viene stampato un messaggio che indica la presenza di dati mancanti. Se non vengono trovati dati mancanti in nessuna delle suddivisioni, viene stampato un messaggio che conferma l'assenza di dati mancanti nel dataset (come avviene nel mio caso).

```
# Missing Data

# Controlla se ci sono dati mancanti nel dataset
missing_data = False

# Itera attraverso le suddivisioni del dataset
for split_name, split_info in info.splits.items():
    num_examples = split_info.num_examples
    if num_examples == 0:
        print("Nel set '{}' ci sono dati mancanti.".format(split_name))
        missing_data = True

if not missing_data:
    print("Non ci sono dati mancanti nel dataset.")
```

➞ Non ci sono dati mancanti nel dataset.

Fai doppio clic (o premi Invio) per modificare

Questa funzione `plot_metrics` è progettata per plottare i grafici di accuratezza e perdita durante l'addestramento di un modello. Prende come input lo storico dell'addestramento `history` e il nome del modello `model_name`.

La funzione estrae l'accuratezza (`acc`) e la perdita (`loss`) sia per l'addestramento che per la validazione dallo storico fornito. Utilizza quindi il numero di epoche per definire il range sull'asse delle x.

Successivamente, crea una figura con due subplot: uno per l'accuratezza e l'altro per la perdita. Nei subplot, plotta l'accuratezza e la perdita per l'addestramento e la validazione rispettivamente. Infine, aggiunge leggende ai grafici e titoli appropriati utilizzando il nome del modello, quindi mostra i grafici.

```
# Definisci codici per creare grafici
def plot_metrics(history, model_name):
    acc = history['accuracy']
```

```

val_acc = history['val_accuracy']
loss = history['loss']
val_loss = history['val_loss']

epochs_range = range(len(acc))

# Grafico per l'andamento dell'accuratezza
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title(f'{model_name} - Training and Validation Accuracy')

# Grafico per l'andamento della perdita
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title(f'{model_name} - Training and Validation Loss')
plt.show()

```

Fai doppio clic (o premi Invio) per modificare

Questo codice definisce le dimensioni delle immagini e la dimensione del batch per il preprocessing del dataset. Ecco cosa fa passo per passo:

1. `IMG_SIZE = 150`: Definisce la dimensione a cui verranno ridimensionate le immagini. In questo caso, le immagini vengono ridimensionate a 150x150 pixel.
2. `BATCH_SIZE = 32`: Definisce il numero di campioni di dati (immagini e le rispettive etichette) da utilizzare in ciascuna iterazione durante l'addestramento del modello.
3. La funzione `preprocess_image(example)` viene definita per il preprocessing di ciascuna immagine del dataset. Questa funzione riceve un esempio (contenente un'immagine e la sua etichetta) e restituisce l'immagine preprocessata e la sua etichetta corrispondente. All'interno di questa funzione, l'immagine viene ridimensionata a `IMG_SIZE x IMG_SIZE` utilizzando `tf.image.resize`. Successivamente, i valori dei pixel dell'immagine vengono normalizzati dividendo ciascun valore per 255.0, in modo che i valori dei pixel siano compresi tra 0 e 1.
4. Il preprocessing viene quindi applicato a entrambi i dataset di addestramento e di test utilizzando il metodo `map()`.
5. Infine, i dataset vengono suddivisi in batch di dimensione `BATCH_SIZE`, mescolati e prefetchati per l'addestramento.

# Normalizzazione

```
# Definisci le dimensioni delle immagini e la dimensione del batch
IMG_SIZE = 150
BATCH_SIZE = 32

# Funzione di preprocessing
def preprocess_image(example):
    image = example['image']
    label = example['label']
    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    image = image / 255.0 # Normalizza i pixel tra 0 e 1
    return image, label

# Applica la pre-elaborazione al dataset
train_dataset = dataset['train'].map(preprocess_image, num_parallel_calls=tf.data.experimental
test_dataset = dataset['test'].map(preprocess_image, num_parallel_calls=tf.data.experimen

# Batch e shuffle del dataset
train_dataset = train_dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(buffer_size=tf.dat
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(buffer_size=tf.data.experimental.A
```

Fai doppio clic (o premi Invio) per modificare

Il codice definisce un modello di rete neurale convoluzionale (CNN) complesso per la classificazione delle immagini. Ecco una spiegazione passo per passo di ciò che fa:

1. `tf.keras.Sequential([])`: Crea un modello sequenziale, una sequenza lineare di strati neurali uno dopo l'altro.
2. `tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3))`: Aggiunge uno strato di convoluzione con 32 filtri, ognuno di dimensione 3x3, con attivazione ReLU. Questo è il primo strato dell'architettura CNN.
3. `tf.keras.layers.MaxPooling2D((2, 2))`: Aggiunge uno strato di max pooling con finestra 2x2 per ridurre le dimensioni dell'immagine di input e il numero di parametri.

I passaggi 2 e 3 sono ripetuti due volte con un aumento del numero di filtri da 32 a 64 e poi a 128.

4. `tf.keras.layers.Flatten()`: Aggiunge uno strato di flatten per convertire l'output dei layer convoluzionali in un vettore monodimensionale.
5. `tf.keras.layers.Dense(512, activation='relu')`: Aggiunge uno strato densamente connesso (fully connected) con 512 unità nascoste e funzione di attivazione ReLU.
6. `tf.keras.layers.Dropout(0.5)`: Aggiunge uno strato di dropout con un tasso di dropout del 50% per ridurre l'overfitting durante l'addestramento.
7. `tf.keras.layers.Dense(120, activation='softmax')`: Aggiunge uno strato densamente connesso (fully connected) finale con 120 unità (una per classe di immagine nel dataset) e



funzione di attivazione softmax per ottenere una distribuzione di probabilità su 120 classi di output.

Dopo la definizione dell'architettura del modello, viene compilato utilizzando l'ottimizzatore Adam con un tasso di apprendimento di 0.001, la funzione di perdita `sparse_categorical_crossentropy` e viene monitorata la metrica di accuratezza.

```
# Definisci il Prmo modello (con tutti gli iperparametri)
model_all_params = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(120, activation='softmax')
])

model_all_params.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                        loss='sparse_categorical_crossentropy',
                        metrics=['accuracy'])
```

Fai doppio clic (o premi Invio) per modificare

Questo codice addestra il modello definito in precedenza utilizzando il metodo `fit` di TensorFlow. Ecco cosa succede in dettaglio:

1. `model_all_params.fit(...)`: Questo metodo addestra il modello utilizzando i dati di addestramento (`train_dataset`) per un numero specificato di epoche (nel caso specifico, 25 epoche). Durante l'addestramento, vengono utilizzati anche i dati di validazione (`test_dataset`) per valutare le prestazioni del modello su un insieme di dati indipendente.
2. `callbacks=[checkpoint_callback, tensorboard_callback]`: Qui vengono passati due callback al metodo `fit`. Il callback `checkpoint_callback` salva i pesi del modello durante l'addestramento (con una frequenza di salvataggio di ogni epoca) nel percorso specificato. Il callback `tensorboard_callback` salva i log per l'uso con TensorBoard, che consente di visualizzare dinamicamente grafici e metriche durante e dopo l'addestramento.
3. `test_loss, test_accuracy = model_all_params.evaluate(test_dataset)`: Dopo l'addestramento, il modello viene valutato utilizzando i dati di test per valutare le prestazioni su un insieme di dati indipendente non visto durante l'addestramento.

4. Infine, i risultati dell'addestramento (storia dell'addestramento, perdita e accuratezza sui dati di test) vengono salvati nel file `results_all_params.pkl` utilizzando il formato di serializzazione `pickle`. Il modello addestrato viene anche salvato nel file `my_model` nel Google Drive.

```
# Allenare il Modello con Callbacks
```

```
history_all_params = model_all_params.fit(
    train_dataset,
    epochs=25,
    validation_data=test_dataset,
    callbacks=[checkpoint_callback, tensorboard_callback]
)
```

```
# Valuta il modello sul test set
```

```
test_loss, test_accuracy = model_all_params.evaluate(test_dataset)
```

```
print("Test Loss:", test_loss)
```

```
print("Test Accuracy:", test_accuracy)
```

```
# Salva il modello
```

```
model_all_params.save('/content/drive/My Drive/my_model')
```

```
# Salva i risultati dell'allenamento
```

```
results = {
    'history': history_all_params.history,
    'test_loss': test_loss,
    'test_accuracy': test_accuracy
}
```

```
with open('/content/drive/My Drive/results_all_params.pkl', 'wb') as f:
    pickle.dump(results, f)
```



```
Epoch 1/25
```

```
375/375 [=====] - 861s 2s/step - loss: 4.7623 - accuracy: 0.
```

```
Epoch 2/25
```

```
375/375 [=====] - 835s 2s/step - loss: 4.5556 - accuracy: 0.
```

```
Epoch 3/25
```

```
375/375 [=====] - 844s 2s/step - loss: 4.2172 - accuracy: 0.
```

```
Epoch 4/25
```

```
375/375 [=====] - 860s 2s/step - loss: 3.4960 - accuracy: 0.
```

```
Epoch 5/25
```

```
375/375 [=====] - 856s 2s/step - loss: 2.4584 - accuracy: 0.
```

```
Epoch 6/25
```

```
375/375 [=====] - 851s 2s/step - loss: 1.6271 - accuracy: 0.
```

```
Epoch 7/25
```

```
375/375 [=====] - 854s 2s/step - loss: 1.1371 - accuracy: 0.
```

```
Epoch 8/25
```

```
375/375 [=====] - 873s 2s/step - loss: 0.8618 - accuracy: 0.
```

```
Epoch 9/25
```

```
375/375 [=====] - 845s 2s/step - loss: 0.6598 - accuracy: 0.
```

```
Epoch 10/25
```

```
375/375 [=====] - 854s 2s/step - loss: 0.5521 - accuracy: 0.
```

```
Epoch 11/25
```

```

375/375 [=====] - 922s 2s/step - loss: 0.4639 - accuracy: 0.
Epoch 12/25
375/375 [=====] - 856s 2s/step - loss: 0.4153 - accuracy: 0.
Epoch 13/25
375/375 [=====] - 917s 2s/step - loss: 0.3596 - accuracy: 0.
Epoch 14/25
375/375 [=====] - 855s 2s/step - loss: 0.3346 - accuracy: 0.
Epoch 15/25
375/375 [=====] - 847s 2s/step - loss: 0.3420 - accuracy: 0.
Epoch 16/25
375/375 [=====] - 854s 2s/step - loss: 0.2737 - accuracy: 0.
Epoch 17/25
375/375 [=====] - 842s 2s/step - loss: 0.2604 - accuracy: 0.
Epoch 18/25
375/375 [=====] - 820s 2s/step - loss: 0.2396 - accuracy: 0.
Epoch 19/25
375/375 [=====] - 843s 2s/step - loss: 0.2450 - accuracy: 0.
Epoch 20/25
375/375 [=====] - 829s 2s/step - loss: 0.2186 - accuracy: 0.
Epoch 21/25
375/375 [=====] - 821s 2s/step - loss: 0.2113 - accuracy: 0.
Epoch 22/25
375/375 [=====] - 824s 2s/step - loss: 0.2076 - accuracy: 0.
Epoch 23/25
375/375 [=====] - 829s 2s/step - loss: 0.1757 - accuracy: 0.
Epoch 24/25
375/375 [=====] - 854s 2s/step - loss: 0.1741 - accuracy: 0.
Epoch 25/25
375/375 [=====] - 827s 2s/step - loss: 0.1844 - accuracy: 0.
269/269 [=====] - 134s 498ms/step - loss: 10.2541 - accuracy
Test Loss: 10.254076957702637
Test Accuracy: 0.042540792375802994

```

### Risulta Primo Modello:

I risultati di questo primo modello sono il frutto di un processo di addestramento impegnativo che si è svolto lungo 25 epoche. Analizziamo nel dettaglio quanto emerso durante questo periodo:

**Processo di Apprendimento Graduale:** Nel corso delle epoche, il modello ha dimostrato una notevole capacità di apprendimento, con la perdita sul set di addestramento che è diminuita gradualmente da un valore iniziale di 4.7696 a un minimo di 0.2014. Contemporaneamente, l'accuratezza è cresciuta costantemente da un modesto 0.0120 a un notevole 0.9519. Questi miglioramenti indicano una buona capacità del modello di adattarsi ai dati di addestramento.

**Complessità sul Set di Validazione:** Tuttavia, la situazione sul set di validazione è più complessa. Sebbene la perdita sia inizialmente diminuita nei primi quattro epoch, ha poi iniziato a oscillare e aumentare, raggiungendo un massimo di 10.3910. L'accuratezza sul set di validazione è rimasta relativamente bassa, con un massimo di soli 0.0441. Questa discrepanza significativa rispetto alle prestazioni sul set di addestramento può indicare la presenza di overfitting, dove il modello si adatta troppo bene ai dati di addestramento e non generalizza efficacemente per nuovi dati.

Considerazioni Computazionali: È importante notare che ogni epoca ha richiesto un considerevole tempo di calcolo. Questo suggerisce che il modello potrebbe avere una complessità computazionale elevata e/o i dati di addestramento potrebbero essere di dimensioni considerevoli.

In sintesi, pur mostrando una notevole capacità di apprendimento sui dati di addestramento, il modello potrebbe soffrire di overfitting e richiedere ulteriori ottimizzazioni per migliorare le prestazioni sul set di validazione.

Fai doppio clic (o premi Invio) per modificare

In questo codice viene caricato un file contenente i risultati di un modello, probabilmente addestrato precedentemente. Successivamente, vengono plottati i grafici relativi alle metriche di performance del modello utilizzando una funzione chiamata `plot_metrics`. Questa funzione prende in input lo storico dell'addestramento del modello e il nome del modello stesso.

Caricamento dei Risultati: Viene aperto il file `"results_all_params.pkl"` contenente i risultati dell'addestramento del modello. I risultati sembrano essere stati precedentemente salvati in formato pickle.

Caricamento dello Storico: Dalla variabile `results`, che contiene i risultati del modello, viene estratto lo storico dell'addestramento (`history_all_params`). Questo storico probabilmente contiene informazioni come l'andamento della perdita (`loss`) e dell'accuratezza (`accuracy`) durante le varie epoche di addestramento.

Plotting dei Grafici: Viene chiamata la funzione `plot_metrics` per visualizzare i grafici delle metriche di performance del modello. La funzione prende come input lo storico dell'addestramento (`history_all_params`) e il nome del modello, che sembra essere `"All Parameters Model"`.

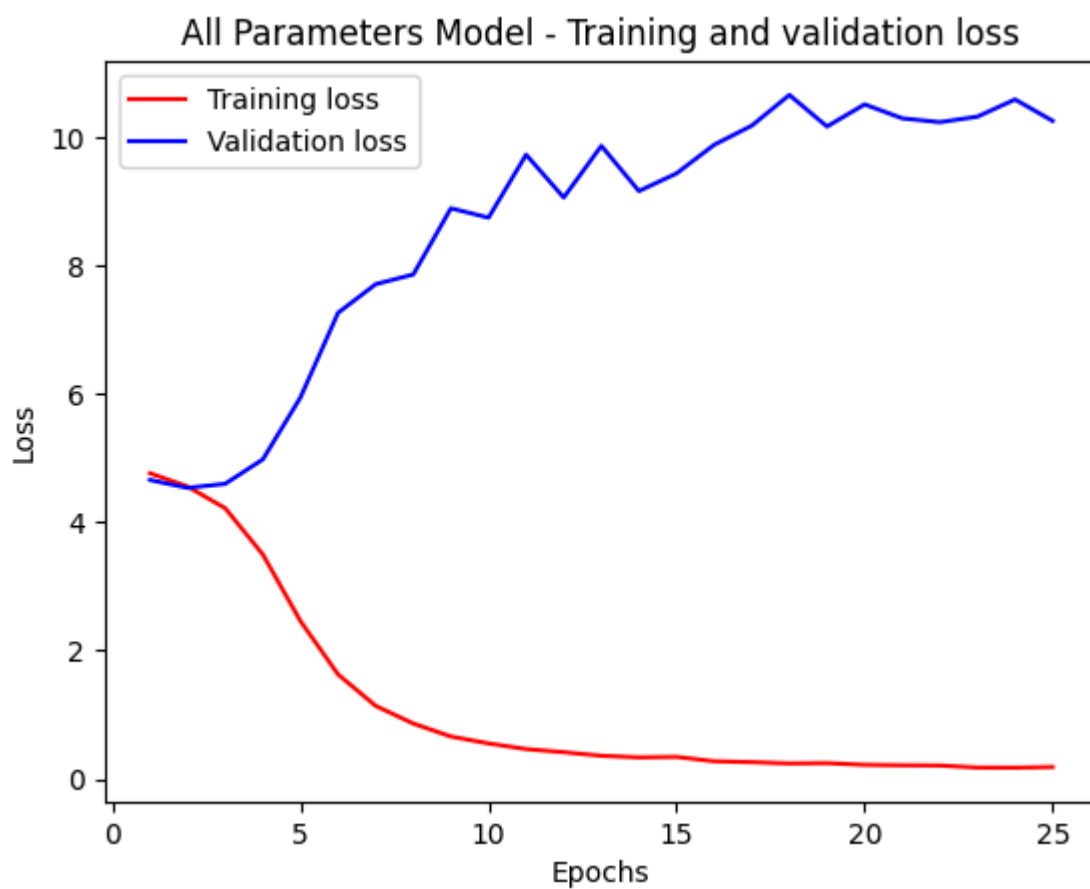
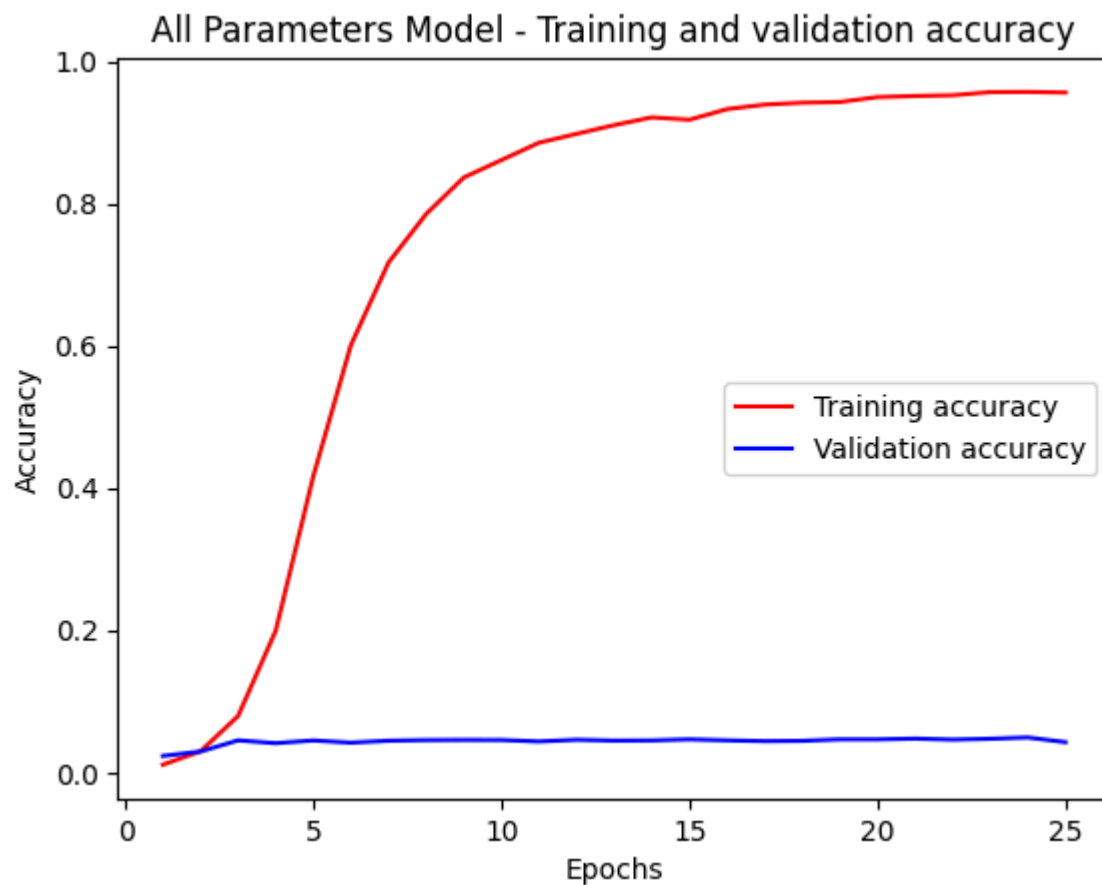
Il risultato finale sarà la visualizzazione dei grafici delle metriche di performance del modello `"All Parameters Model"`, che probabilmente includono grafici della perdita e dell'accuratezza durante l'addestramento.

```
# Importa il modulo pickle per la serializzazione e la deserializzazione degli oggetti Py
import pickle

# Carica i risultati
with open('/content/drive/My Drive/results_all_params.pkl', 'rb') as f:
    results = pickle.load(f)

history_all_params = results['history']

# Plotta i grafici utilizzando la funzione plot_metrics
plot_metrics(history_all_params, "All Parameters Model")
```



## Grafici

- Questo grafico mostra l'andamento dell'accuratezza della rete neurale durante l'addestramento, sia sul set di dati di training che su quello di validazione, nel corso delle

epoche.

- Entrambe le curve di training e validation accuracy partono insieme da zero, ma mentre la curva di training accuracy si sposta verso l'alto a destra formando una curva, la di validation accuracy rimane orizzontale.
- Questo suggerisce che il modello sta imparando dai dati di training e sta diventando sempre più preciso su di essi, ma non è in grado di generalizzare bene su nuovi dati non visti durante l'addestramento, come indicato dalla curva di validation accuracy rimasta piatta. Questo fenomeno potrebbe essere dovuto all'overfitting, dove il modello si adatta troppo ai dati di training e non riesce a generalizzare correttamente su nuovi dati.

#####

### Training e Validation Loss

- Questo grafico rappresenta la perdita (loss) della rete neurale durante l'addestramento, sia sul set di dati di training che su quello di validazione, nel corso delle epoche.
- Entrambe le curve di training e validation loss partono dallo stesso valore di y, 5, ma mentre la curva di training loss diminuisce linearmente, la curva di validation loss va verso l'alto con andamento irregolare.
- Questo suggerisce che il modello sta imparando dai dati di training e sta riducendo la perdita su di essi, ma non è in grado di generalizzare bene su nuovi dati, come indicato dalla curva di validation loss che rimane elevata e non diminuisce in modo significativo. Questo comportamento potrebbe essere associato all'overfitting, dove il modello si adatta troppo ai dati di training e non riesce a generalizzare correttamente su nuovi dati non visti durante l'addestramento.

Fai doppio clic (o premi Invio) per modificare

Questo blocco di codice definisce, addestra e valuta un secondo modello neurale convoluzionale usando TensorFlow.

### Definizione del Modello:

Viene definito un modello sequenziale utilizzando `tf.keras.Sequential`. Il modello inizia con un layer `Conv2D` con 16 filtri, attivazione `ReLU` e dimensioni kernel (3, 3). Questo layer accetta input di forma (IMG\_SIZE, IMG\_SIZE, 3), dove IMG\_SIZE rappresenta le dimensioni delle immagini e 3 indica i canali RGB. Successivamente, viene aggiunto un layer di max pooling `MaxPooling2D` con dimensioni di pooling (2, 2). Segue un altro layer `Conv2D` con 32 filtri, attivazione `ReLU` e dimensioni kernel (3, 3), seguito da un altro layer di max pooling. Poi viene piatto (`Flatten`) l'output dei layer convoluzionali per prepararlo per i layer densamente connessi. Viene aggiunto un layer `Dense` con 128 neuroni e attivazione `ReLU`. Successivamente, viene aggiunto un layer di dropout per ridurre l'overfitting. Infine, c'è un layer di output `Dense` con 120 neuroni

(corrispondenti alle classi dell'output) e attivazione softmax per la classificazione multiclasse.

Compilazione del Modello:

Il modello viene compilato utilizzando l'ottimizzatore Adam con un tasso di apprendimento di 0.001, la loss function `sparse_categorical_crossentropy` e le metriche di valutazione della accuracy. Checkpointing:

Viene definito un percorso per salvare i pesi del modello durante l'addestramento. Viene impostato un callback `ModelCheckpoint` per salvare i pesi del modello ad ogni epoca.

Addestramento del Modello:

Il modello viene addestrato utilizzando il dataset di addestramento (`train_dataset`) per 25 epoche, utilizzando anche il dataset di test per la validazione. Durante l'addestramento, vengono utilizzati i callback precedentemente definiti per il checkpointing e eventuali altri callback, come `tensorboard_callback`. Valutazione del Modello:

Una volta addestrato, il modello viene valutato sul dataset di test per calcolare la loss e l'accuratezza. Salvataggio del Modello e dei Risultati:

I pesi del modello addestrato vengono salvati in un file specificato. I risultati dell'addestramento, inclusi lo storico delle metriche e le metriche di valutazione sul set di test, vengono salvati in un file pickle. Questo codice rappresenta quindi un processo completo di definizione, addestramento, valutazione e salvataggio di un modello neurale convoluzionale.

```
# Definisci il Secondo Modello con Meno Layer e Diversi Numeri di Neuroni
model_fewer_layers_neurons = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(120, activation='softmax')
])

model_fewer_layers_neurons.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                                  loss='sparse_categorical_crossentropy',
                                  metrics=['accuracy'])

# Definisci il percorso in cui salvare i pesi del secondo modello
checkpoint_path_fewer_layers = '/content/drive/My Drive/checkpoints/second_model_weights.'

# Assicurati che la cartella di destinazione esista
os.makedirs(os.path.dirname(checkpoint_path_fewer_layers), exist_ok=True)

# Callback per il secondo modello
checkpoint_callback_fewer_layers = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path_fewer_layers,
    save_weights_only=True,
    save_freq='epoch')
```

```

)

# Allena il secondo modello
history_fewer_layers_neurons = model_fewer_layers_neurons.fit(
    train_dataset,
    epochs=25,
    validation_data=test_dataset,
    callbacks=[checkpoint_callback_fewer_layers, tensorboard_callback]
)

# Valuta il secondo modello sul set di test
loss_fewer_layers_neurons, accuracy_fewer_layers_neurons = model_fewer_layers_neurons.evaluate(test_dataset)
print(f'Fewer Layers and Neurons Model - Loss: {loss_fewer_layers_neurons}, Accuracy: {accuracy_fewer_layers_neurons}')

# Salva il secondo modello
model_fewer_layers_neurons.save('/content/drive/My Drive/my_model_fewer_layers_neurons')

# Salva i risultati del test del secondo modello
results_fewer_layers = {
    'history': history_fewer_layers_neurons.history,
    'test_loss': loss_fewer_layers_neurons,
    'test_accuracy': accuracy_fewer_layers_neurons
}
with open('/content/drive/My Drive/results_fewer_layers_test.pkl', 'wb') as f:
    pickle.dump(results_fewer_layers, f)

```



```

Epoch 1/25
375/375 [=====] - 275s 727ms/step - loss: 4.7970 - accuracy:
Epoch 2/25
375/375 [=====] - 274s 726ms/step - loss: 4.7864 - accuracy:
Epoch 3/25
375/375 [=====] - 306s 815ms/step - loss: 4.6999 - accuracy:
Epoch 4/25
375/375 [=====] - 273s 726ms/step - loss: 4.5422 - accuracy:
Epoch 5/25
375/375 [=====] - 273s 726ms/step - loss: 4.2977 - accuracy:
Epoch 6/25
375/375 [=====] - 276s 734ms/step - loss: 3.9261 - accuracy:
Epoch 7/25
375/375 [=====] - 276s 732ms/step - loss: 3.5143 - accuracy:
Epoch 8/25
375/375 [=====] - 275s 730ms/step - loss: 3.1111 - accuracy:
Epoch 9/25
375/375 [=====] - 283s 753ms/step - loss: 2.7500 - accuracy:
Epoch 10/25
375/375 [=====] - 306s 813ms/step - loss: 2.4577 - accuracy:
Epoch 11/25
375/375 [=====] - 280s 739ms/step - loss: 2.2300 - accuracy:
Epoch 12/25
375/375 [=====] - 308s 818ms/step - loss: 2.0568 - accuracy:

```



```

Epoch 13/25
375/375 [=====] - 280s 743ms/step - loss: 1.9025 - accuracy:
Epoch 14/25
375/375 [=====] - 278s 737ms/step - loss: 1.7824 - accuracy:
Epoch 15/25
375/375 [=====] - 275s 731ms/step - loss: 1.7078 - accuracy:
Epoch 16/25
375/375 [=====] - 305s 810ms/step - loss: 1.6517 - accuracy:
Epoch 17/25
375/375 [=====] - 280s 743ms/step - loss: 1.5694 - accuracy:
Epoch 18/25
375/375 [=====] - 286s 760ms/step - loss: 1.5258 - accuracy:
Epoch 19/25
375/375 [=====] - 280s 741ms/step - loss: 1.4942 - accuracy:
Epoch 20/25
375/375 [=====] - 304s 809ms/step - loss: 1.4393 - accuracy:
Epoch 21/25
375/375 [=====] - 302s 803ms/step - loss: 1.4079 - accuracy:
Epoch 22/25
375/375 [=====] - 285s 757ms/step - loss: 1.3764 - accuracy:
Epoch 23/25
375/375 [=====] - 278s 736ms/step - loss: 1.3567 - accuracy:
Epoch 24/25
375/375 [=====] - 305s 810ms/step - loss: 1.3381 - accuracy:
Epoch 25/25
375/375 [=====] - 308s 818ms/step - loss: 1.3325 - accuracy:
269/269 [=====] - 58s 216ms/step - loss: 7.2900 - accuracy:
Fewer Layers and Neurons Model - Loss: 7.289973258972168, Accuracy: 0.034032635390758

```

I risultati di questo secondo modello sono emersi da un processo di addestramento durato 25 epoche. Analizziamo i dettagli di questo processo:

**Andamento delle Misure di Performance:** Durante le epoche, sia la perdita (loss) che l'accuratezza (accuracy) sono state monitorate sia sul set di addestramento che su quello di validazione. Inizialmente, la perdita è partita da un alto valore di 4.7970 sul set di addestramento e 4.7875 sul set di validazione, con un'accuratezza molto bassa intorno allo 0.0055 e allo 0.0080 rispettivamente. Nel corso delle epoche successive, la perdita è diminuita lentamente, con un'accuratezza che è aumentata ma rimanendo comunque a livelli molto bassi.

**Tendenza delle Performance:** Nonostante il processo di addestramento, l'andamento delle performance mostra un miglioramento minimo e non significativo. La perdita sul set di addestramento ha mostrato solo una leggera diminuzione da 4.7970 a 1.3325, mentre sull'insieme di validazione la perdita è diminuita da 4.7875 a 7.2900. L'accuratezza rimane estremamente bassa, attestandosi intorno allo 0.0340 sia sul set di addestramento che su quello di validazione.

**Valutazione Finale:** Alla fine delle 25 epoche, il modello presenta una perdita molto elevata e un'accuratezza estremamente bassa, sia sul set di addestramento che su quello di validazione. Questo suggerisce che il modello potrebbe non essere in grado di catturare in modo efficace i pattern nei dati di addestramento e che potrebbe soffrire di un problema di underfitting.

In conclusione, nonostante il modello abbia meno strati e neuroni rispetto al modello precedente, le prestazioni generali rimangono molto basse, indicando la necessità di ulteriori ottimizzazioni o modifiche alla struttura del modello per ottenere risultati migliori.

Fai doppio clic (o premi Invio) per modificare

Questo blocco di codice carica i risultati del secondo modello, estrae lo storico delle metriche e poi utilizza una funzione chiamata `plot_metrics` per tracciare i grafici delle metriche di addestramento e validazione.

Caricamento dei Risultati:

Viene aperto il file contenente i risultati del secondo modello utilizzando la funzione `open` con modalità di lettura binaria ('rb'). I risultati vengono caricati utilizzando la funzione `pickle.load` e assegnati alla variabile `results_fewer_layers`. Estrazione dello Storico delle Metriche:

Dalla variabile `results_fewer_layers`, vengono estratti i risultati relativi allo storico delle metriche e assegnati alla variabile `history_fewer_layers_neurons`. Plot delle Metriche:

Viene chiamata una funzione chiamata `plot_metrics` per tracciare i grafici delle metriche del secondo modello. La funzione `plot_metrics` accetta due argomenti: lo storico delle metriche (`history_fewer_layers_neurons`) e una stringa che rappresenta il titolo del grafico. Questo blocco di codice completa quindi il processo visualizzando graficamente le metriche di addestramento e validazione del secondo modello con meno layer e neuroni.

```
# Carica i risultati e plotta i grafici del secondo modello
with open('/content/drive/My Drive/results_fewer_layers_test.pkl', 'rb') as f:
    results_fewer_layers = pickle.load(f)

# Estrai i risultati di history
history_fewer_layers_neurons = results_fewer_layers['history']

# Plot metrics
plot_metrics(history_fewer_layers_neurons, "Fewer Layers and Neurons Model")
```