Progetto AML unificato AUG

February 21, 2025

1 Fruit multiclassification

- Cavallini Francesco (920835)
- Villa Fabio (907506)
- Perfetti Luca (919835)

Volgiamo realizzare un progetto che, letto un dataset contenente diversi istanze di frutti ripotati di 360, implementi un modello di rete neurale (basato su CNN) che permetta la multi calssificazione dei vari gruppi di frutta.fornita. Ossia, più nello specifico, vogliamo creare diverse istanze di reti neurali (sotto forma modalità trial and error) per arrivare a vedere quali sono le performmance più realistiche possibiili che possiamo ottenere addestrando una rete neurale per predirre due label; dove le label in questione rappresentano: - label 1: categoria generale di frutto (eg: “apple”) - label 2: sotto-categoria di frutto (eg: “apple golden”, “apple red”, …)

1.1 Setup

Semplice sezione di setup delle librerie e del dirve

[1]:
```
import os
from PIL import Image
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from collections import defaultdict
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import matplotlib.image as mpimg
from anytree import Node, RenderTree
from anytree.exporter import DotExporter
from PIL import ImageEnhance
import random

import logging
from numpy.random import RandomState
from sklearn import cluster, decomposition
from sklearn.datasets import fetch_lfw_people
```

```
import seaborn as sns
import tensorflow as tf
from tensorflow.data import Dataset

from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D,␣
 ↪BatchNormalization, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.applications import ResNet50
import numpy as np
from keras.layers import Input, Lambda
from tensorflow.keras.models import load_model
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from tensorflow.keras.applications import MobileNet
import cv2
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from tensorflow.keras.regularizers import l2
from tensorflow.keras.applications import ResNet101, VGG19
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow import keras

rng = RandomState(0)
```

1.2 Lettura dataset

Dalla documentazione si evice che il dataset a nostra disposizione contiene 141 istanze di diversi frutti che vengono fatti ruotare di 360 gradi. Ma si nota che i dati a nostra disposizione non sono perfettamente integri, per questo motivo i dati vengono modificati manualmente (eliminando circa 20 classi) per garantire l'integrità dei dati. Questo step verrà meglio spiegto nella relazione.

Di seguito procediamo con la lettura del dataset che abbiamo mantenuto:

[2]:
```
dataset_path = 'Dataset\\'
```

1.2.1 Setup dei dictionaries

Inizializzazione di dictionary che associa labels_2 (eg: "apple golden") ad un numero intero

[3]:
```
# Initialize dictionaries and lists
labels_2_int = {}
labels_2_array = []
labels_1_array = []

# Counter for consecutive numbering
```

```
counter = 0

# Iterate over the subdirectories using case-insensitive sorting
for folder_name in sorted(os.listdir(dataset_path + "val\\"), key=str.lower):
    # Map folder name to an integer
    folder_name = folder_name[:-2]
    if len(folder_name.split(' ')) == 1:
        folder_name = folder_name.split(' ')[0] + " Undefined"

    if folder_name not in labels_2_int:
        labels_2_int[folder_name] = counter
        labels_2_array.append(folder_name)
        counter += 1
        print(folder_name)

    if folder_name.split(' ')[0] not in labels_1_array:
        labels_1_array.append(folder_name.split(' ')[0])
```

```
Apple Undefined
Apple Braeburn
Apple Crimson Snow
Apple Golden
Apple Granny Smith
Apple hit
Apple Pink Lady
Apple Red
Apple Red Delicious
Apple Red Yellow
Apricot Undefined
Avocado Undefined
Avocado ripe
Banana Undefined
Banana Lady Finger
Banana Red
Beetroot Undefined
Blueberry Undefined
Cabbage white
Cactus fruit
Cantaloupe Undefined
Carambula Undefined
Carrot Undefined
Cauliflower Undefined
Cherry Undefined
Cherry Rainier
Cherry Wax Black
Cherry Wax Red
Cherry Wax Yellow
Chestnut Undefined
```

Clementine Undefined
Cocos Undefined
Corn Undefined
Corn Husk
Cucumber Undefined
Cucumber Ripe
Dates Undefined
Eggplant Undefined
Eggplant long
Fig Undefined
Ginger Root
Granadilla Undefined
Grape Blue
Grape Pink
Grape White
Grapefruit Pink
Grapefruit White
Guava Undefined
Hazelnut Undefined
Huckleberry Undefined
Kaki Undefined
Kiwi Undefined
Kohlrabi Undefined
Kumquats Undefined
Lemon Undefined
Lemon Meyer
Limes Undefined
Lychee Undefined
Mandarine Undefined
Mango Undefined
Mango Red
Mangostan Undefined
Maracuja Undefined
Melon Piel de Sapo
Mulberry Undefined
Nectarine Undefined
Nectarine Flat
Nut Forest
Nut Pecan
Onion Red
Onion Red Peeled
Onion White
Orange Undefined
Papaya Undefined
Passion Fruit
Peach Undefined
Peach Flat
Pear Undefined

```
Pear Abate
Pear Forelle
Pear Kaiser
Pear Monster
Pear Red
Pear Stone
Pear Williams
Pepino Undefined
Pepper Green
Pepper Orange
Pepper Red
Pepper Yellow
Physalis Undefined
Physalis with Husk
Pineapple Undefined
Pineapple Mini
Pitahaya Red
Plum Undefined
Pomegranate Undefined
Pomelo Sweetie
Potato Red
Potato Red Washed
Potato Sweet
Potato White
Quince Undefined
Rambutan Undefined
Raspberry Undefined
Redcurrant Undefined
Salak Undefined
Strawberry Undefined
Strawberry Wedge
Tamarillo Undefined
Tangelo Undefined
Tomato Undefined
Tomato Cherry Red
Tomato Heart
Tomato Maroon
Tomato not Ripened
Tomato Yellow
Walnut Undefined
Watermelon Undefined
Zucchini Undefined
Zucchini dark
```

Inizializzazione di un dictionary che associa labels_1 (eg: “appple”) ad un numero intero

[4]:
```
labels_1_int = {}
for i, fruit_name in enumerate(labels_1_array):
```

```
    # Map folder name to an integer
    labels_1_int[fruit_name] = i
```

Questi due dictionaries sono poi utili alla lettura del dataset, siccome, con questi possiamo assegnare ad ogni classe un intero

1.2.2 Creazione dei file numpy array

Funzioni di modifica immagini Siccome si ha che ogni cartella è formata da sample dello stesso oggetto frutto ma ruotato di 360 gradi, si riconosce che trainare su un dataset del genere porterebbe ad apprendere solo determinate feature di quell'ogetto ma i vari modelli non imparerebbero a generalizzare. Per questo motivo si decide di creare una seri di funzioni che inseriscono delle distorsioni dell'imagnine campione originale all'interno del dataset. Alcune di queste sono:

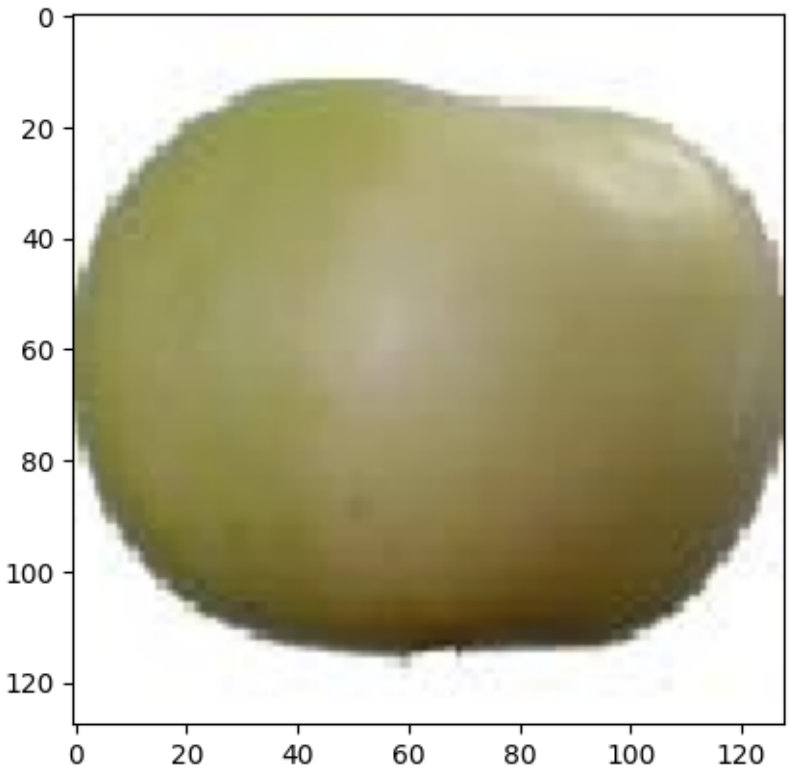
[5]:
```
img = load_img("Dataset\\val\\Apple 6\\r0_3_100.jpg", target_size=(128, 128))
```

---

[6]:
```
def random_flip(img):
    # Convert the image to a numpy array
    img_array = img_to_array(img)

    # Randomly decide whether to flip along x-axis and/or y-axis
    flip_x = np.random.choice([True, False])
    flip_y = np.random.choice([True, False])

    if flip_x:
        img_array = np.flip(img_array, axis=1)
    if flip_y:
        img_array = np.flip(img_array, axis=0)

    # Convert the numpy array back to a PIL image
    flipped_img = Image.fromarray(img_array.astype('uint8'), 'RGB')

    return flipped_img
```

[7]:
```
plt.imshow(random_flip(img))
```

[7]:
```
<matplotlib.image.AxesImage at 0x1b4d378d6c0>
```



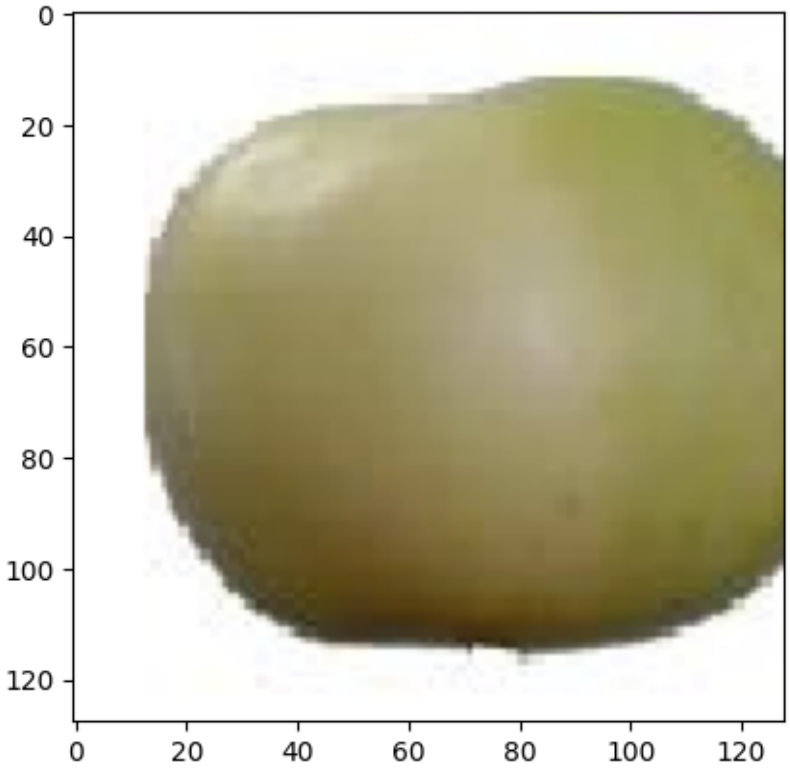
---

[8]:
```python
def translate_image(img, max_translation=60):
    # Converti l'immagine in un array numpy
    img_array = np.array(img)

    # Genera traslazioni casuali per x e y
    tx = np.random.randint(-max_translation, max_translation + 1)
    ty = np.random.randint(-max_translation, max_translation + 1)

    # Crea una nuova immagine bianca con le stesse dimensioni
    translated_img = Image.new('RGB', img.size, (255, 255, 255))

    # Calcola le coordinate di incollaggio
    x_offset = max(0, tx)
    y_offset = max(0, ty)

    # Incolla l'immagine originale nella nuova immagine traslata
    translated_img.paste(img, (x_offset, y_offset))

    # Ritaglia l'immagine per mantenere le dimensioni originali
    translated_img = translated_img.crop((0, 0, img.size[0], img.size[1]))
```

```
    return translated_img
```

[9]:
```
plt.imshow(translate_image(img))
```

[9]:
```
<matplotlib.image.AxesImage at 0x1b4d59e0100>
```



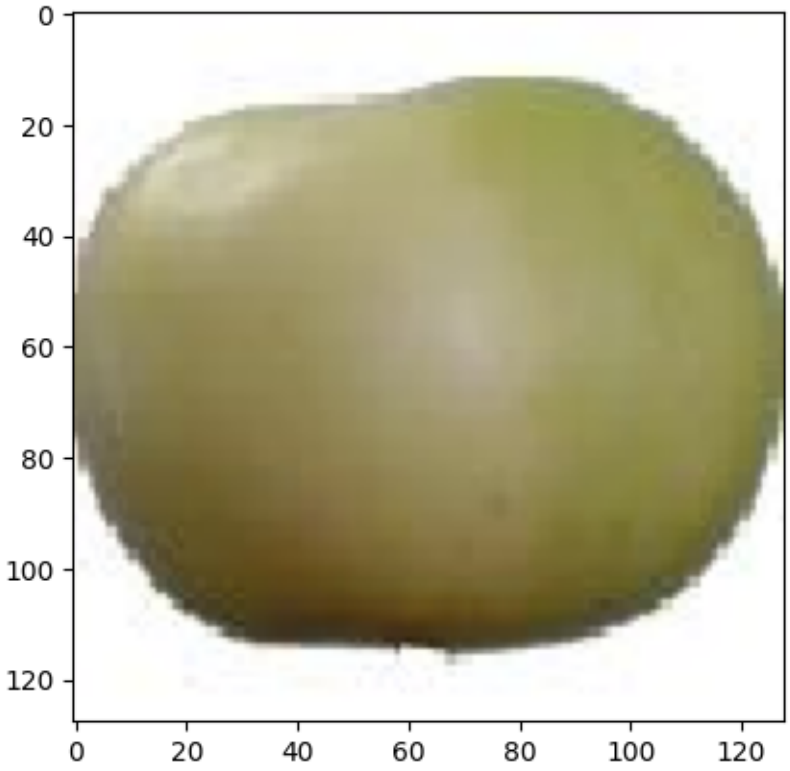
---

[10]:
```
def random_brightness(img, min_factor=0.9, max_factor=1.3):
    # Convert the image to a numpy array
    img_array = img_to_array(img)

    # Generate a random brightness factor
    brightness_factor = np.random.uniform(min_factor, max_factor)

    # Adjust the brightness, but do not modify pixels with value 255
    img_array = np.where(img_array < 24, np.clip(img_array * brightness_factor,␣
 ↪0, 255), img_array)

    # Convert the numpy array back to a PIL image
    bright_img = Image.fromarray(img_array.astype('uint8'), 'RGB')
```

```
    return bright_img
```

[11]:
```
plt.imshow(random_brightness(img))
```

[11]: <matplotlib.image.AxesImage at 0x1b4d5a23100>



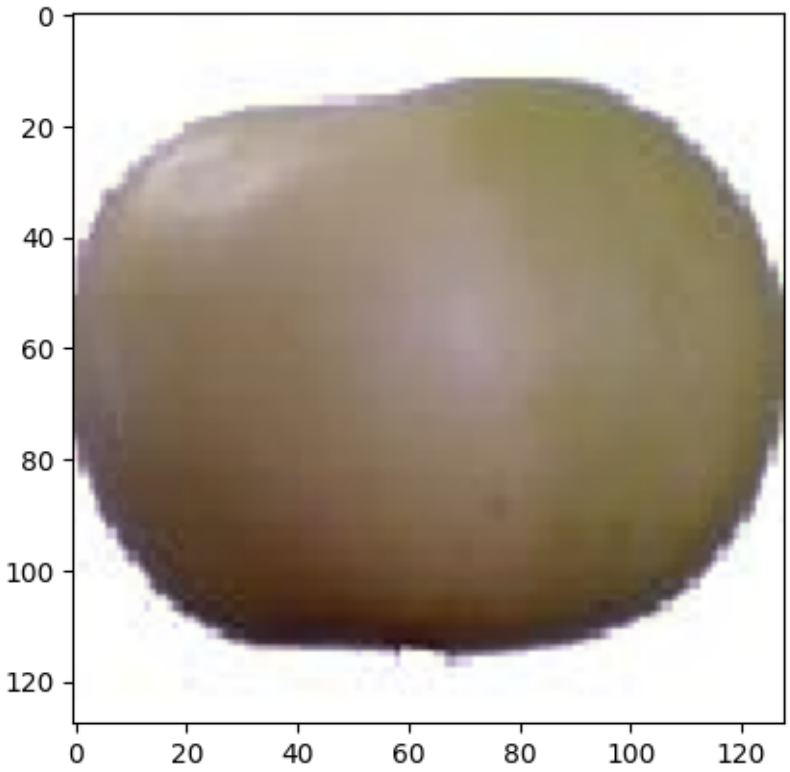
---

[12]:
```
def random_rgb_shift(img, max_shift=25):
    # Convert the image to a numpy array
    img_array = img_to_array(img)

    # Generate random shifts for each channel
    shift_r = np.random.randint(-max_shift, max_shift + 1)
    shift_g = np.random.randint(-max_shift, max_shift + 1)
    shift_b = np.random.randint(-max_shift, max_shift + 1)

    # Apply the shifts to each channel, but do not modify pixels with value 255
    img_array[..., 0] = np.where(img_array[..., 0] < 240, np.clip(img_array[...
 ↪, 0] + shift_r, 0, 255), img_array[..., 0])
```

```
    img_array[..., 1] = np.where(img_array[..., 1] < 240, np.clip(img_array[...
↪, 1] + shift_g, 0, 255), img_array[..., 1])
    img_array[..., 2] = np.where(img_array[..., 2] < 240, np.clip(img_array[...
↪, 2] + shift_b, 0, 255), img_array[..., 2])

    # Convert the numpy array back to a PIL image
    shifted_img = Image.fromarray(img_array.astype('uint8'), 'RGB')

    return shifted_img
```

[13]:
```
plt.imshow(random_rgb_shift(img))
```

[13]: <matplotlib.image.AxesImage at 0x1b4d5acb4c0>



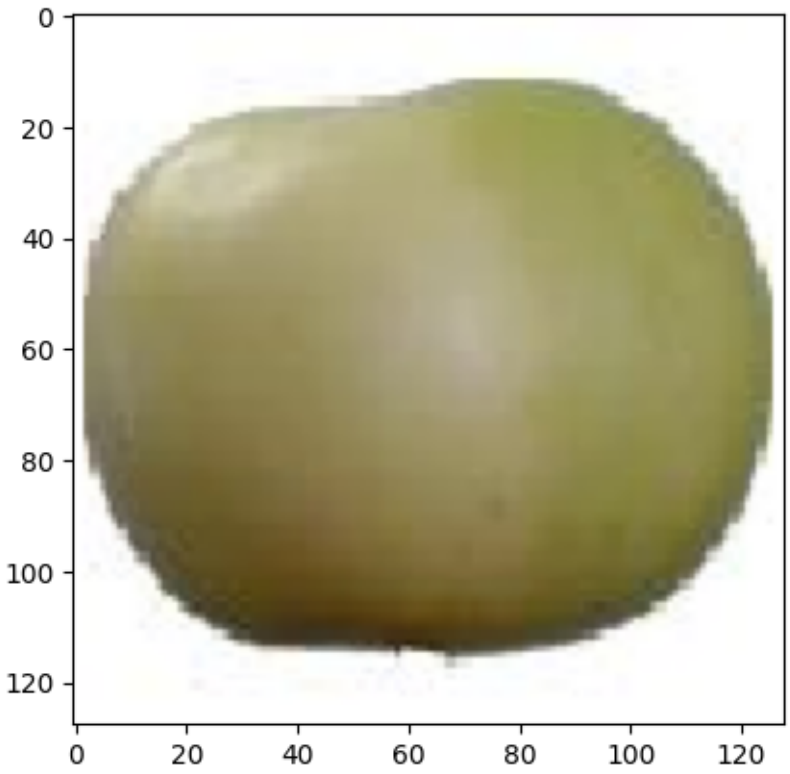
---

[14]:
```
def random_stretch(img, max_stretch=0.07):
    # Convert the image to a numpy array
    img_array = np.array(img)

    # Get the original dimensions
    original_height, original_width = img_array.shape[:2]
```

```
    # Generate random stretch factors for x and y axes
    stretch_x = 1 + np.random.uniform(-max_stretch, max_stretch)
    stretch_y = 1 + np.random.uniform(-max_stretch, max_stretch)

    # Calculate new dimensions
    new_width = int(original_width * stretch_x)
    new_height = int(original_height * stretch_y)

    # Resize the image
    stretched_img = img.resize((new_width, new_height), Image.LANCZOS)

    # Create a new white image with the original dimensions
    output_img = Image.new('RGB', (original_width, original_height), (255, 255,␣
↪255))

    # Calculate the position to paste the stretched image
    x_offset = (original_width - new_width) // 2
    y_offset = (original_height - new_height) // 2

    # Paste the stretched image onto the new white image
    output_img.paste(stretched_img, (x_offset, y_offset))

    return output_img
```

[15]:
```
plt.imshow(random_stretch(img))
```

[15]:
```
<matplotlib.image.AxesImage at 0x1b4d5b66aa0>
```



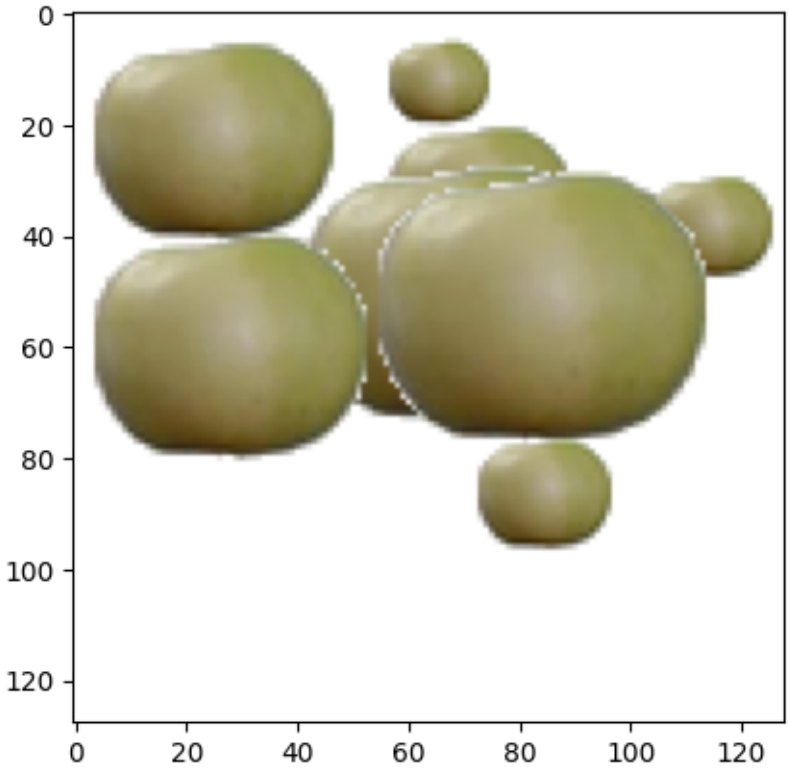
---

[16]:
```python
def random_resize_and_repeat(img, min_scale=0.1, max_scale=0.5, num_repeats=10):
    # Convert the image to a numpy array
    img_array = np.array(img)

    # Create a white background image
    output_img = Image.new('RGB', img.size, (255, 255, 255))

    for _ in range(num_repeats):
        # Generate a random scale factor
        scale_factor = np.random.uniform(min_scale, max_scale)

        # Calculate new dimensions
        new_width = int(img.size[0] * scale_factor)
        new_height = int(img.size[1] * scale_factor)

        # Resize the image
        resized_img = img.resize((new_width, new_height), Image.LANCZOS)

        # Generate random position
        x_offset = np.random.randint(0, img.size[0] - new_width + 1)
```

```
        y_offset = np.random.randint(0, img.size[1] - new_height + 1)

        # Create a mask to handle transparency
        mask = resized_img.convert("L").point(lambda x: 0 if x > 230 else 255,␣
↪mode='1')

        # Paste the resized image onto the white background using the mask
        output_img.paste(resized_img, (x_offset, y_offset), mask)

    return output_img
```

[17]:
```
plt.imshow(random_resize_and_repeat(img))
```

[17]:
```
<matplotlib.image.AxesImage at 0x1b4d5bd1210>
```



Funzione lettura dataset Si definisce la seguente funzione adibita a crare dei file .npz (numpy array) che serviranno successivamente a leggere l'intero dataset instantaneamente (invece che dover attendere lunghi tempi di attesa solo per caricare il dataset)

[18]:
```

```

```
def preprocess_and_save(input_dir, output_file, show_images=False,␣
↪augmentation=False):
    data = []
    # categoria generale (es mela)
    labels_1 = []
    # categoria specifica (es mela golden)
    labels_2 = []

    # support variables for plotting
    already_printed = []
    images_to_plot = []
    labels_to_plot = []

    class_names = sorted(os.listdir(input_dir))  # Assicura ordine costante␣
↪delle classi
    for _, class_name in enumerate(class_names):
        class_dir = os.path.join(input_dir, class_name)

        # Use os.path.split to correctly handle path separators in a␣
↪platform-independent way
        _, folder_name = os.path.split(class_dir)

        label_1 = labels_1_int[folder_name.split(' ')[0]]  # Access label_1

        folder_name = folder_name[:-2]
        if(len( folder_name.split(' ') ) == 1):
            folder_name = folder_name.split(' ')[0] + " Undefined"
        print(folder_name)
        label_2 = labels_2_int[folder_name]                # Access label_2

        for img_name in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_name)
            img = load_img(img_path)  # Ridimensiona

            if(augmentation):
            # Genera un numero casuale da uno a 10
                random_number = random.randint(1, 10)
                # Switch del numero con case vuoti
                if random_number == 1:
                    pass  # Case vuoto
                elif random_number == 2:
                    img = translate_image(img)
                elif random_number == 3:
                    pass  # nothing
                elif random_number == 4:
                    img = random_brightness(img)
                elif random_number == 5:
```

```
                    pass  # Case vuoto
                elif random_number == 6:
                    img = random_stretch(img)
                elif random_number == 7:
                    img = random_resize_and_repeat(img)
                elif random_number == 8:
                    img = random_flip(img)
                elif random_number == 9:
                    img = random_rgb_shift(img)
                elif random_number == 10:
                    pass  # Case vuoto

            img_array = img_to_array(img)  # Convert the image to a NumPy array before resizing
            #img_array = cv2.resize(img_array, (128, 128), interpolation=cv2.INTER_CUBIC)
            data.append(img_array)
            labels_1.append(label_1)
            labels_2.append(label_2)

            # Store image and label for plotting
            if label_2 not in already_printed:
                images_to_plot.append(img)
                labels_to_plot.append(folder_name)  # Use folder_name directly
                already_printed.append(label_2)

    if show_images:
        # Plot images and labels
        num_images = len(images_to_plot)
        num_cols = 10
        num_rows = (num_images + num_cols - 1) // num_cols

        plt.figure(figsize=(20, num_rows * 2))
        for i in range(num_images):
            plt.subplot(num_rows, num_cols, i + 1)
            plt.imshow(images_to_plot[i])
            plt.title(labels_to_plot[i])
            plt.axis('off')
        plt.show()
        #save image to file
        plt.savefig('plot.png')

    # Converti in array numpy
    data = np.array(data)
    labels_1 = np.array(labels_1)
    labels_2 = np.array(labels_2)
```

```
    # Controllo se è train o validation per applicare lo shuffle
    if 'train' in input_dir:
        indices = np.random.permutation(len(data))  # Genera indici casuali
        data = data[indices]
        labels_1 = labels_1[indices]
        labels_2 = labels_2[indices]
        print("shuffle eseguito")
    else:
        print("shuffle solo per il train")

    # Salva i dati
    np.savez(output_file, x=data, y1=labels_1, y2=labels_2)
```

Di seguito viene chiamata la funzione per creare i file di train/test/validation sets:

Scelta di utilizzo aumentation

[19]:
```
needs_augmentation = True
npz_prefix = "Regular_NPZ\\"
if(needs_augmentation):
    npz_prefix = "Augmented_NPZ\\"
```

File train Nota che in questo caso viene anche plottato un grafico che mostra un sample per ogni frutto disponibile nel nostro dataset. Questo plot è stato poi salvato in memoria di modo da porter comunque venire visualizzato anche quando non si esegue la funzione mostrata precedentemente:

[20]:
```
train_path = npz_prefix+'train_data_BIG_AUG.npz'
# check if npz file already exists
if not os.path.exists(train_path):
    preprocess_and_save(dataset_path+'train', train_path, show_images=True)
else:
    # plot local image using plt
    img = mpimg.imread('plot.png')
    # Create a figure with a large size
    fig, ax = plt.subplots(figsize=(12, 12))  # Adjust size as needed

    # Display the image
    ax.imshow(img)

    # Maximize the axis size by turning off the spines and ticks
    ax.set_xticks([])
    ax.set_yticks([])
    ax.spines['top'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.spines['left'].set_visible(False)
    ax.spines['right'].set_visible(False)

    # Show the image
```

```
    plt.show()
```

File validazione scrittura file validation set con la stessa funzione

[21]:
```
val_path = npz_prefix+'val_data_BIG_AUG.npz'
# check if npz file already exists
if not os.path.exists(val_path):
    preprocess_and_save(dataset_path+'val', val_path)
```

File test scrittura file test set con la stessa funzione

[22]:
```
test_path = npz_prefix+'test_data_BIG_AUG.npz'
# check if npz file already exists
if not os.path.exists(test_path):
    preprocess_and_save(dataset_path+'test', test_path)
```

1.2.3 Lettura file numpy array

Una volta che i file numpy array sono stati salvati è possibile leggerli direttamente, avere lo step precedente che ci salva tutti i dati in formato file e poi leggerli in questo step rende la lettura compilazione del file molto lenta (cosa che non ci interessa perchè dobbiamo farlo una sola volta) ma la lettura di tutto il dataset molto veloce.

[23]:
```
def load_data():
    # load train data
    data_train = np.load(npz_prefix+'train_data_BIG_AUG.npz')
    x_train, y1_train, y2_train = data_train['x'], data_train['y1'],␣
 ↪data_train['y2']

    # Load validation data
    data_val = np.load(npz_prefix+'val_data_BIG_AUG.npz')
    x_val, y1_val, y2_val = data_val['x'], data_val['y1'], data_val['y2']

    # Load test data
    data_test = np.load(npz_prefix+'test_data_BIG_AUG.npz')
    x_test, y1_test, y2_test = data_test['x'], data_test['y1'], data_test['y2']

    return x_train, y1_train, y2_train, x_val, y1_val, y2_val, x_test, y1_test,␣
 ↪y2_test
```

[24]:
```
x_train, y1_train, y2_train, x_val, y1_val, y2_val, x_test, y1_test, y2_test =␣
 ↪load_data()
```

[25]:
```
for i in range(10):
    print(f"train {i+1}: {labels_1_array[y1_train[i]]},␣
 ↪{labels_2_array[y2_train[i]]}")
```

```
train 1: Tomato, Tomato Undefined
train 2: Mulberry, Mulberry Undefined
train 3: Eggplant, Eggplant Undefined
train 4: Pear, Pear Forelle
train 5: Cherry, Cherry Undefined
```

```
train 6: Quince, Quince Undefined
train 7: Cantaloupe, Cantaloupe Undefined
train 8: Apple, Apple Golden
train 9: Physalis, Physalis Undefined
train 10: Apple, Apple Golden
```

1.2.4 Pre-Processing: one-hot-encoding delle label

Per ora le label precedentemente salvate sono state create in formato numerico (classe "apple"–>0, classe "pear"–>1, ...). In questa sezione si procede con il one-hot di tutte le label, che, in teoria, dovrebbe migliorare le perfomance di train in quanto abbiamo così che tutti i dati di label sono compresi nel range [0-1]

[ ]:
```python
def onehot_all_labels():
    # onehot label1
    y1_train = to_categorical(y1_train, num_classes=70)
    y1_val = to_categorical(y1_val, num_classes=70)
    y1_test = to_categorical(y1_test, num_classes=70)

    # onehot label2
    y2_train = to_categorical(y2_train, num_classes=121)
    y2_val = to_categorical(y2_val, num_classes=121)
    y2_test = to_categorical(y2_test, num_classes=121)

    return y1_train, y1_val, y1_test, y2_train, y2_val, y2_test
```

[ ]:
```python
y1_train, y1_val, y1_test, y2_train, y2_val, y2_test = onehot_all_labels()
```

Si sceglie di applicare direttamente questo step in fase di lettura dei dati in quanto non comporta alcuna modifica significativa al valore informativo delle informazioni, tutti i dati possono venire comunque letti come se non avesssimo fatto alcuna modifica.

1.3 Anlisi esplorativa

Prima di procedere con la preparazione dei dati si fa' una breve analisi esplorativa, per osservare se ci sono alcune incoformità nel dataset precedentemente processato e per cercare attributi particolari che magari hanno bisono di essere sistemati in fase di preparazione dati pre-train.

1.3.1 Numero di classi

Abbiamo dunque che il numero di classi per i 2 tipi di label:

[27]:
```python
num_object_label_1 = y1_train.shape[1]
num_object_label_2 = y2_train.shape[1]
print(f"Numero di classi per label_1: {num_object_label_1}")
print(f"Numero di classi per label_2: {num_object_label_2}")
```

```
Numero di classi per label_1: 70
Numero di classi per label_2: 121
```

1.3.2 Studio bilanciamento dataset

In particolare, se andiamo ad esplorare il numero di dati presenti per ogni label e sotto-label otteniamo il seguente albero:

[28]:
```
# Contiamo il numero di occorrenze per ogni etichetta secondaria
y2_counts = np.sum(y2_train, axis=0)  # Conta le occorrenze di ogni classe␣
 ↪secondaria

# Creiamo una struttura ad albero
root = Node("Frutti")  # Nodo radice
primary_nodes = {}

# Aggiungi nodi per le label primarie
for primary_label in labels_1_int:
    primary_nodes[primary_label] = Node(f"{primary_label} (0)", parent=root)

# Aggiungi nodi per le label secondarie con conteggio
for class_name, index in labels_2_int.items():
    primary_label = class_name.split(' ')[0]  # Estrarre la label primaria
    count = int(y2_counts[index])  # Numero di occorrenze per questa classe
    if primary_label in primary_nodes:
        Node(f"{class_name} ({count})", parent=primary_nodes[primary_label])

# Aggiorna i conteggi delle label primarie
for primary_label, node in primary_nodes.items():
    total_count = sum(
        int(child.name.split("(")[-1].strip(")")) for child in node.children
    )
    node.name = f"{primary_label} ({total_count})"

# Stampa l'albero con conteggi
for pre, _, node in RenderTree(root):
    print(f"{pre}{node.name}")
```

```
Frutti
  Apple (6069)
      Apple Undefined (379)
      Apple Braeburn (394)
      Apple Crimson Snow (356)
      Apple Golden (1163)
      Apple Granny Smith (394)
      Apple hit (562)
      Apple Pink Lady (365)
      Apple Red (1132)
      Apple Red Delicious (392)
      Apple Red Yellow (932)
  Apricot (394)
```

```
    Apricot Undefined (394)
Avocado (735)
    Avocado Undefined (342)
    Avocado ripe (393)
Banana (1144)
    Banana Undefined (392)
    Banana Lady Finger (360)
    Banana Red (392)
Beetroot (360)
    Beetroot Undefined (360)
Blueberry (370)
    Blueberry Undefined (370)
Cabbage (116)
    Cabbage white (116)
Cactus (392)
    Cactus fruit (392)
Cantaloupe (788)
    Cantaloupe Undefined (788)
Carambula (392)
    Carambula Undefined (392)
Carrot (121)
    Carrot Undefined (121)
Cauliflower (562)
    Cauliflower Undefined (562)
Cherry (2758)
    Cherry Undefined (985)
    Cherry Rainier (591)
    Cherry Wax Black (394)
    Cherry Wax Red (394)
    Cherry Wax Yellow (394)
Chestnut (360)
    Chestnut Undefined (360)
Clementine (392)
    Clementine Undefined (392)
Cocos (392)
    Cocos Undefined (392)
Corn (730)
    Corn Undefined (360)
    Corn Husk (370)
Cucumber (1005)
    Cucumber Undefined (316)
    Cucumber Ripe (689)
Dates (392)
    Dates Undefined (392)
Eggplant (567)
    Eggplant Undefined (375)
    Eggplant long (192)
Fig (562)
```

```
    Fig Undefined (562)
Ginger (238)
    Ginger Root (238)
Granadilla (392)
    Granadilla Undefined (392)
Grape (2737)
    Grape Blue (788)
    Grape Pink (394)
    Grape White (1555)
Grapefruit (786)
    Grapefruit Pink (392)
    Grapefruit White (394)
Guava (400)
    Guava Undefined (400)
Hazelnut (372)
    Hazelnut Undefined (372)
Huckleberry (392)
    Huckleberry Undefined (392)
Kaki (392)
    Kaki Undefined (392)
Kiwi (373)
    Kiwi Undefined (373)
Kohlrabi (377)
    Kohlrabi Undefined (377)
Kumquats (392)
    Kumquats Undefined (392)
Lemon (786)
    Lemon Undefined (394)
    Lemon Meyer (392)
Limes (392)
    Limes Undefined (392)
Lychee (392)
    Lychee Undefined (392)
Mandarine (392)
    Mandarine Undefined (392)
Mango (733)
    Mango Undefined (392)
    Mango Red (341)
Mangostan (240)
    Mangostan Undefined (240)
Maracuja (392)
    Maracuja Undefined (392)
Melon (591)
    Melon Piel de Sapo (591)
Mulberry (394)
    Mulberry Undefined (394)
Nectarine (778)
    Nectarine Undefined (394)
```

```
    Nectarine Flat (384)
Nut (952)
    Nut Forest (524)
    Nut Pecan (428)
Onion (1067)
    Onion Red (360)
    Onion Red Peeled (356)
    Onion White (351)
Orange (384)
    Orange Undefined (384)
Papaya (394)
    Papaya Undefined (394)
Passion (392)
    Passion Fruit (392)
Peach (1379)
    Peach Undefined (985)
    Peach Flat (394)
Pear (4203)
    Pear Undefined (1123)
    Pear Abate (392)
    Pear Forelle (562)
    Pear Kaiser (240)
    Pear Monster (392)
    Pear Red (533)
    Pear Stone (569)
    Pear Williams (392)
Pepino (392)
    Pepino Undefined (392)
Pepper (1984)
    Pepper Green (356)
    Pepper Orange (562)
    Pepper Red (533)
    Pepper Yellow (533)
Physalis (788)
    Physalis Undefined (394)
    Physalis with Husk (394)
Pineapple (787)
    Pineapple Undefined (392)
    Pineapple Mini (395)
Pitahaya (392)
    Pitahaya Red (392)
Plum (1414)
    Plum Undefined (1414)
Pomegranate (394)
    Pomegranate Undefined (394)
Pomelo (360)
    Pomelo Sweetie (360)
Potato (1443)
```

```
    Potato Red (360)
    Potato Red Washed (363)
    Potato Sweet (360)
    Potato White (360)
Quince (392)
    Quince Undefined (392)
Rambutan (394)
    Rambutan Undefined (394)
Raspberry (392)
    Raspberry Undefined (392)
Redcurrant (394)
    Redcurrant Undefined (394)
Salak (392)
    Salak Undefined (392)
Strawberry (985)
    Strawberry Undefined (394)
    Strawberry Wedge (591)
Tamarillo (392)
    Tamarillo Undefined (392)
Tangelo (392)
    Tangelo Undefined (392)
Tomato (4088)
    Tomato Undefined (2104)
    Tomato Cherry Red (394)
    Tomato Heart (548)
    Tomato Maroon (294)
    Tomato not Ripened (380)
    Tomato Yellow (368)
Walnut (588)
    Walnut Undefined (588)
Watermelon (380)
    Watermelon Undefined (380)
Zucchini (384)
    Zucchini Undefined (192)
    Zucchini dark (192)
```

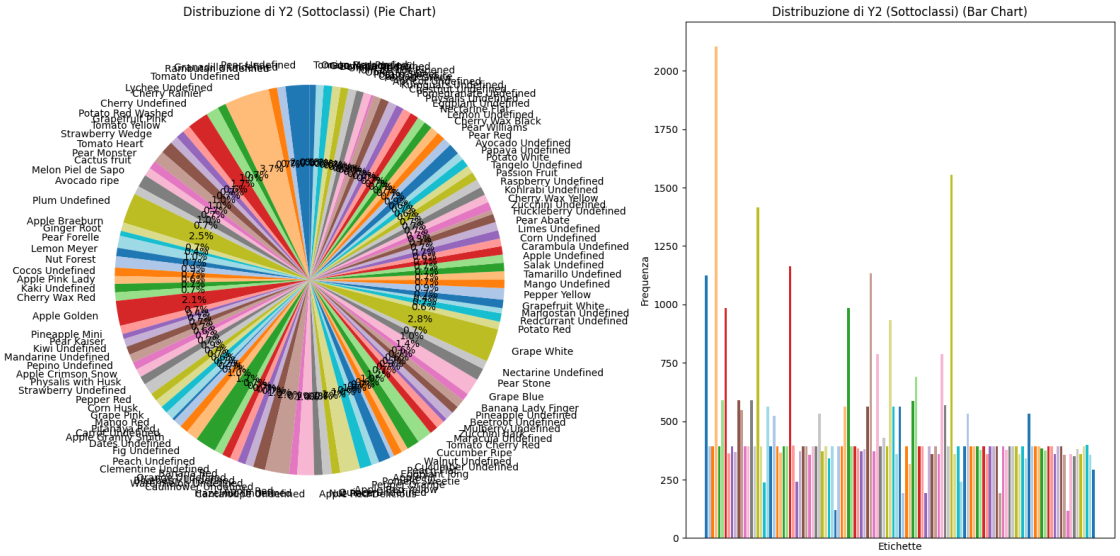
Osservando i dati euristicamente sembra che ogni sotto-classe (label_2) abbia più o meno lo stesso numero di sample (circa 300), mentre invece il numero di sample per le classi principali (label) è molto più sbilanciato. Procediamo quindi a a possiamo studiare più nel dettaglio la distrubuzione di classi e sottoclassi con la seguente funzione:

[29]:
```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

def plot_label_distribution(y_train, label_names, title="Distribuzione delle␣
 ↪etichette"):
    """
```

```
    Visualizza la distribuzione delle etichette in un dataset.

    :param y_train: array numpy (one-hot encoded)
    :param label_names: lista dei nomi delle classi
    :param title: titolo del grafico
    """
    # Converti one-hot encoding in indici delle classi
    class_indices = np.argmax(y_train, axis=1)  # Trova la classe con valore massimo

    # Conta la frequenza di ogni classe
    label_counts = Counter(class_indices)

    # Estrai etichette e frequenze
    labels = [label_names[idx] for idx in label_counts.keys()]
    values = list(label_counts.values())

    # Imposta la figura con due sottotrame
    fig, axes = plt.subplots(1, 2, figsize=(16, 8))

    # Diagramma a torta
    axes[0].pie(values, labels=labels, startangle=90, colors=plt.cm.tab20.colors, autopct='%1.1f%%')
    axes[0].set_title(f"{title} (Pie Chart)")
    axes[0].axis('equal')  # Assicura che il diagramma sia un cerchio

    # Diagramma a barre
    axes[1].bar(labels, values, color=plt.cm.tab20.colors[:len(labels)])
    axes[1].set_title(f"{title} (Bar Chart)")
    axes[1].set_xlabel("Etichette")
    axes[1].set_ylabel("Frequenza")
    axes[1].tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)  # Rimuove le etichette

    # Mostra il grafico
    plt.tight_layout()
    plt.show()
```

1.3.3 Studio delle frequenze per label_2 (sotto-classe)

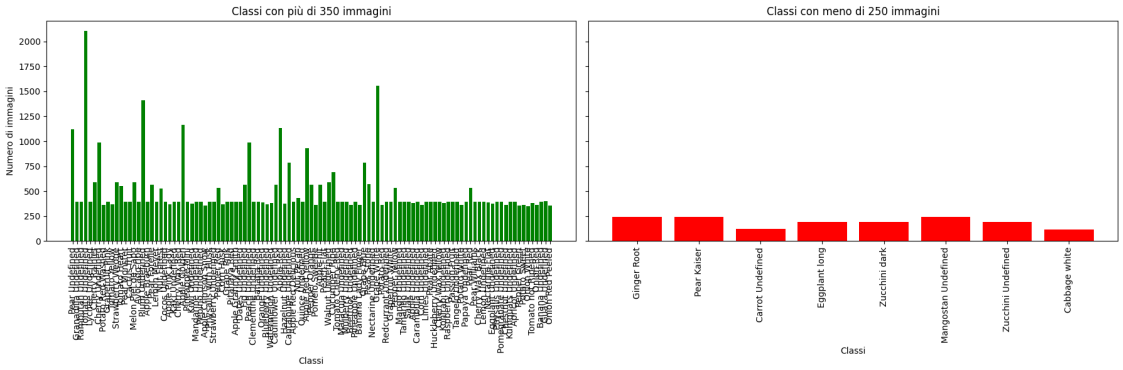
[30]:
```
# Plot distribuzione di y2_train
label_names_y2 = list(labels_2_int.keys())  # Nomi per y2
plot_label_distribution(y2_train, label_names_y2, title="Distribuzione di Y2 (Sottoclassi)")
```



Dai grafici appena mostrati ci possiamo ricredere, risulta infatti eviddente (specialmente dal diagramma a barre) che seppur la maggior parte delle classi abbia un numero vicino alle 350 istanze ci sono comunque molte altre classi che hanno o un numero minore di 350 samples o maggiore di 350. Queste venngono meglio visualizzate nel prossimo plot:

[ ]:
```
# Converti one-hot encoding in indici delle classi
class_indices = np.argmax(y2_train, axis=1)

# Conta le occorrenze di ogni classe
class_counts = Counter(class_indices)

# Filtra le classi con meno di 250 e più di 350 immagini
class_over_350 = {label_names_y2[idx]: count for idx, count in class_counts.
 ↪items() if count > 350}
class_under_250 = {label_names_y2[idx]: count for idx, count in class_counts.
 ↪items() if count < 250}

# Imposta la figura con due subplot affiancati
fig, axes = plt.subplots(1, 2, figsize=(18, 6), sharey=True)

# Grafico per le classi con più di 350 immagini
axes[0].bar(class_over_350.keys(), class_over_350.values(), color='green')
axes[0].set_xlabel('Classi')
axes[0].set_ylabel('Numero di immagini')
axes[0].set_title('Classi con più di 350 immagini')
axes[0].tick_params(axis='x', rotation=90)

# Grafico per le classi con meno di 250 immagini
axes[1].bar(class_under_250.keys(), class_under_250.values(), color='red')
```

```
axes[1].set_xlabel('Classi')
axes[1].set_title('Classi con meno di 250 immagini')
axes[1].tick_params(axis='x', rotation=90)

# Mostra il grafico
plt.tight_layout()
plt.show()
```

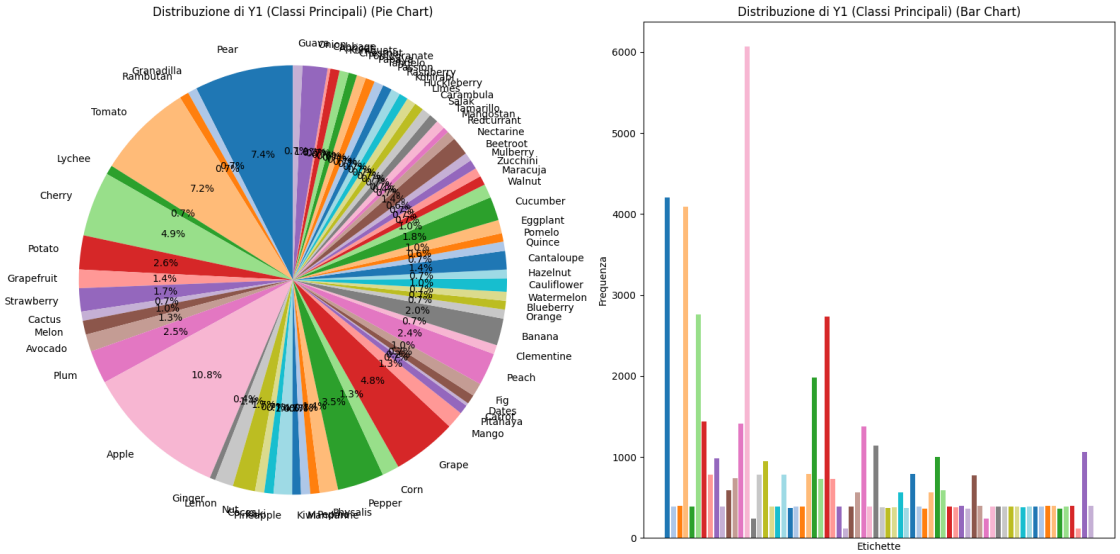


Dal grafico risulta evidente che ci siamo molte sotto-classi che hanno più di 350 sample e potrebbero portare ad una classificazione sbilanciata. Allo stesso modo, seppur in numero minore, la stessa ossrevazione vale anche per le classi undersamplate.

Per questo motivo potrebbe dunque essere sensato nella fase di preparazione dati potremmo pensare di fare il calcolo dei pesi da assegnare a ciascuna classe per evitare di avere una classificazione spilanciata

1.3.4 Studio frequenze per label_1 (classe principale)

[32]:
```
# Plot distribuzione di y1_train
label_names_y1 = list(labels_1_int.keys())  # Nomi per y1
plot_label_distribution(y1_train, label_names_y1, title="Distribuzione di Y1␣
↪(Classi Principali)")
```



In questo caso la differenza di esempi per ogni classe diventa molto più evidente, anche senza bisogno di fare un'analisi più dettagliata come per le label precedenti, per questo motivo si decide di calcolare dei pesi da assegnare a ciascuna classe anche per questa label

1.4 Preparazione dei dati

Questo step è già stato in gran parte svolto nel momento in cui si è definita la funzione che salva i file npz. Abbiamo infatti che in questa funzione ci si assicura anche che tutte le immaigni vengano salvate con la stessa dimensione standard (100,100). Inoltre è anche già stato fatto il one-hot delle labels in fase di pre-processing. Di conseguenza, non ci rimangono altre operazioni da fare se non controlloare l'uniformità di tutti i dati caricati.

1.4.1 Controllo conformità dati

[ ]:
```
def check_data_conformity(x, y1, y2, dataset_name="train"):
    errors = []

    # 1 Controllo dimensione dataset
    num_samples = x.shape[0]

    if y1.shape[0] != num_samples:
        errors.append(f" {dataset_name}: y1 ha {y1.shape[0]} campioni, ma x ne␣
↪ha {num_samples}")

    if y2.shape[0] != num_samples:
        errors.append(f" {dataset_name}: y2 ha {y2.shape[0]} campioni, ma x ne␣
↪ha {num_samples}")

    # 2 Controllo se le etichette sono one-hot encoded
```

```
    if len(y1.shape) != 2:
        errors.append(f" {dataset_name}: y1 dovrebbe essere one-hot encoded␣
↪(shape 2D), ma ha shape {y1.shape}")

    if len(y2.shape) != 2:
        errors.append(f" {dataset_name}: y2 dovrebbe essere one-hot encoded␣
↪(shape 2D), ma ha shape {y2.shape}")

    # Se non ci sono errori, tutto è conforme
    if not errors:
        print(f" {dataset_name}: Tutti i dati sono conformi!")
    else:
        for error in errors:
            print(error)
```

[ ]:
```
check_data_conformity(x_train, y1_train, y2_train,dataset_name="Train")
check_data_conformity(x_val, y1_val, y2_val, dataset_name="Validation")
check_data_conformity(x_test, y1_test, y2_test, dataset_name="Test")
```

```
Train: Tutti i dati sono conformi!
Validation: Tutti i dati sono conformi!
Test: Tutti i dati sono conformi!
```

1.4.2 Funzione relaod di tutti i dati

Siccome quando si esegue nomralizzazione tramite i metodi di keras passando i file di train e test questi vengono passati per riferimento e vengono modificati è necessario ri-caricare i corretti valori di train test e val datasets. Per precauzione aggiuntiva questa operazione viene ripetuta prima dell'esecuzione di ciascun modello.

[ ]:
```
def reaload_from_scratch():
    x_train, y1_train, y2_train, x_val, y1_val, y2_val, x_test, y1_test,␣
↪y2_test = load_data()
    y1_train, y1_val, y1_test, y2_train, y2_val, y2_test = onehot_all_labels()
    return x_train, y1_train, y2_train, x_val, y1_val, y2_val, x_test, y1_test,␣
↪y2_test
```

1.5 Costruzione dei modelli

Nota: per velocizzare il train ed evitare di overfitatre sui dati di train, si sceglie di impostare un valore di steps per epochs uguale a:

$$\frac{\text{150\% dell'intero dataset}}{\text{numero di epoche}}$$

In questo modo si ha che per ogni singola epoca si visiterà in totale solo una frazione del dataset; invece che avere che per ogni singola epoca far visitare tutto il set. Questo porta ad avere che la loss potrebbe avere comportamenti fluttuanti (scendi / sali) quando si scoprono per la prima volta sample che non sono mai stati esplorati in epoche precedenti.

1.5.1 Modello 0: CNN custom (Baseline)

Normalizzazione dati: Come vedremo ogni modello pre-trainato richiede una fase di normalizzazione a se stante, quindi per evitare di combinare erroneamente queste elaborazioni diverse sulle stesse variabili ogni modello dovrà implementare la sua normalizzazione:

[ ]:
```
# Preprocessing per train
x_train = x_train / 255.0

# Preprocessing per validation
x_val = x_val / 255.0

# Preprocessing per test
x_test = x_test / 255.0
```

Function graph

[ ]:
```
def plot_training_history(history):
    epochs = range(1, len(history.history['loss']) + 1)

    plt.figure(figsize=(12, 18))  # Aumento l'altezza per adattare più subplot

    # Grafico della Loss totale
    plt.subplot(3, 2, 1)
    plt.plot(epochs, history.history['loss'], 'r-', label='Training Loss')
    plt.plot(epochs, history.history['val_loss'], 'r--', label='Validation␣
↪Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Total Loss')

    # Grafico della Loss Y1 (Fruit)
    plt.subplot(3, 2, 2)
    plt.plot(epochs, history.history['y1_loss'], 'b-', label='Y1 (Fruit) Loss')
    plt.plot(epochs, history.history['val_y1_loss'], 'b--', label='Val Y1 Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Y1 (Fruit) Loss')

    # Grafico della Loss Y2 (Quality)
    plt.subplot(3, 2, 3)
    plt.plot(epochs, history.history['y2_loss'], 'g-', label='Y2 (Quality)␣
↪Loss')
    plt.plot(epochs, history.history['val_y2_loss'], 'g--', label='Val Y2 Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
```

```
    plt.legend()
    plt.title('Y2 (Quality) Loss')

    # Grafico della Accuracy Y1 (Fruit)
    plt.subplot(3, 2, 4)
    plt.plot(epochs, history.history['y1_accuracy'], 'b-', label='Y1 (Fruit)␣
↪Accuracy')
    plt.plot(epochs, history.history['val_y1_accuracy'], 'b--', label='Val Y1␣
↪Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('Y1 (Fruit) Accuracy')

    # Grafico della Accuracy Y2 (Quality)
    plt.subplot(3, 2, 5)
    plt.plot(epochs, history.history['y2_accuracy'], 'g-', label='Y2 (Quality)␣
↪Accuracy')
    plt.plot(epochs, history.history['val_y2_accuracy'], 'g--', label='Val Y2␣
↪Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('Y2 (Quality) Accuracy')

    # Grafico della Accuracy Totale
    total_accuracy = (np.array(history.history['y1_accuracy']) + np.
↪array(history.history['y2_accuracy'])) / 2
    val_total_accuracy = (np.array(history.history['val_y1_accuracy']) + np.
↪array(history.history['val_y2_accuracy'])) / 2

    plt.subplot(3, 2, 6)
    plt.plot(epochs, total_accuracy, 'm-', label='Total Accuracy')
    plt.plot(epochs, val_total_accuracy, 'm--', label='Val Total Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('Total Accuracy')

    plt.tight_layout()  # Migliora la disposizione dei grafici
    plt.show()
```

Calcolo steps per epochs

[ ]:
```
input_shape = x_train.shape[1:]
num_classes_1 = len(y1_train[0])
num_classes_2 = len(y2_train[0])
```

[ ]:
```
dataset_size = len(x_train)  # Numero totale di campioni
epochs_number = 100
batch_size = 32

steps_per_epoch = int(1.5 * dataset_size / batch_size / epochs_number)
steps_per_val = int(len(x_val) / batch_size / epochs_number)

print(f"Steps per epoch: {steps_per_epoch}")
print(f"Steps per val: {steps_per_val}")
```

```
Steps per epoch: 26
Steps per val: 7
```

Implementazione CNN

[ ]:
```
# Definizione della CNN
inputs = Input(shape=input_shape)

x = Conv2D(64, (9, 9), activation='relu')(inputs)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)

x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)

# Output 1: Predizione del Frutto
fruit_output = Dense(num_classes_1, activation='softmax', name='y1')(x)

# Output 2: Predizione della Qualità
quality_output = Dense(num_classes_2, activation='softmax', name='y2')(x)
```

[ ]:
```
# Modello finale
model = Model(inputs=inputs, outputs=[fruit_output, quality_output])
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100, 100, 3)	0	-

```
conv2d (Conv2D)          (None, 92, 92, 64)      15,616   input_layer[0][0]

max_pooling2d            (None, 46, 46, 64)          0   conv2d[0][0]
(MaxPooling2D)

conv2d_1 (Conv2D)        (None, 46, 46, 128)     73,856   max_pooling2d[0]…

max_pooling2d_1          (None, 23, 23, 128)         0   conv2d_1[0][0]
(MaxPooling2D)

conv2d_2 (Conv2D)        (None, 23, 23, 256)    295,168   max_pooling2d_1[…

max_pooling2d_2          (None, 11, 11, 256)         0   conv2d_2[0][0]
(MaxPooling2D)

global_average_poo…      (None, 256)                 0   max_pooling2d_2[…
(GlobalAveragePool…

dense (Dense)            (None, 128)            32,896   global_average_p…

dropout (Dropout)        (None, 128)                 0   dense[0][0]

y1 (Dense)               (None, 70)              9,030   dropout[0][0]

y2 (Dense)               (None, 121)            15,609   dropout[0][0]


Total params: 442,175 (1.69 MB)

Trainable params: 442,175 (1.69 MB)

Non-trainable params: 0 (0.00 B)
```

[ ]:
```
# Compilazione
optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(
    optimizer=optimizer,
    loss={
            'y1': 'categorical_crossentropy',
            'y2': 'categorical_crossentropy'
```

```
        },
    metrics={
        'y1': 'accuracy',
        'y2': 'accuracy'
        }
)
```

Train del modello:

[ ]:
```
if(os.path.exists('keras/model-CNN.keras')):
    model = load_model('keras/model-CNN.keras')
else:
    history = model.fit(
        x_train,
        {
            'y1': y1_train,
            'y2': y2_train
        },
        validation_data=(
            x_val,
            {
                'y1': y1_val,
                'y2': y2_val
            }
        ),
        epochs=epochs_number,
        batch_size=batch_size,
        steps_per_epoch=steps_per_epoch,
        validation_steps=steps_per_val
    )
```

```
Epoch 1/100
26/26             24s 894ms/step -
loss: 8.9519 - y1_accuracy: 0.0783 - y1_loss: 4.1210 - y2_accuracy: 0.0043 -
y2_loss: 4.8309 - val_loss: 7.8146 - val_y1_accuracy: 0.1328 - val_y1_loss:
2.8335 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.9811
Epoch 2/100
26/26             8s 322ms/step -
loss: 8.6026 - y1_accuracy: 0.0922 - y1_loss: 3.8794 - y2_accuracy: 0.0209 -
y2_loss: 4.7232 - val_loss: 7.5716 - val_y1_accuracy: 0.2734 - val_y1_loss:
2.8521 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.7195
Epoch 3/100
26/26             8s 320ms/step -
loss: 8.3837 - y1_accuracy: 0.1648 - y1_loss: 3.7783 - y2_accuracy: 0.0528 -
y2_loss: 4.6054 - val_loss: 7.6581 - val_y1_accuracy: 0.5938 - val_y1_loss:
2.7480 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.9101
Epoch 4/100
26/26             8s 320ms/step -
loss: 8.3100 - y1_accuracy: 0.1208 - y1_loss: 3.7350 - y2_accuracy: 0.0467 -
```

```
y2_loss: 4.5750 - val_loss: 7.8373 - val_y1_accuracy: 0.0078 - val_y1_loss:
3.1589 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.6783
Epoch 5/100
26/26            8s 324ms/step -
loss: 8.1653 - y1_accuracy: 0.1238 - y1_loss: 3.6834 - y2_accuracy: 0.0517 -
y2_loss: 4.4819 - val_loss: 6.7301 - val_y1_accuracy: 0.3750 - val_y1_loss:
2.2595 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.4707
Epoch 6/100
26/26            8s 324ms/step -
loss: 8.0507 - y1_accuracy: 0.1122 - y1_loss: 3.6342 - y2_accuracy: 0.0334 -
y2_loss: 4.4165 - val_loss: 7.1522 - val_y1_accuracy: 0.0391 - val_y1_loss:
2.5412 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.6110
Epoch 7/100
26/26            8s 321ms/step -
loss: 7.7782 - y1_accuracy: 0.1424 - y1_loss: 3.5267 - y2_accuracy: 0.0773 -
y2_loss: 4.2516 - val_loss: 6.8803 - val_y1_accuracy: 0.0234 - val_y1_loss:
2.5114 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.3689
Epoch 8/100
26/26            8s 321ms/step -
loss: 7.5998 - y1_accuracy: 0.1529 - y1_loss: 3.4167 - y2_accuracy: 0.0693 -
y2_loss: 4.1831 - val_loss: 6.0981 - val_y1_accuracy: 0.6289 - val_y1_loss:
1.8308 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.2673
Epoch 9/100
26/26            8s 323ms/step -
loss: 7.4795 - y1_accuracy: 0.1491 - y1_loss: 3.4246 - y2_accuracy: 0.0826 -
y2_loss: 4.0549 - val_loss: 6.4419 - val_y1_accuracy: 0.5078 - val_y1_loss:
2.1942 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.2477
Epoch 10/100
26/26            8s 322ms/step -
loss: 7.2634 - y1_accuracy: 0.1611 - y1_loss: 3.3441 - y2_accuracy: 0.0903 -
y2_loss: 3.9193 - val_loss: 6.1190 - val_y1_accuracy: 0.5430 - val_y1_loss:
1.9387 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.1804
Epoch 11/100
26/26            9s 336ms/step -
loss: 6.9156 - y1_accuracy: 0.1497 - y1_loss: 3.1965 - y2_accuracy: 0.0751 -
y2_loss: 3.7192 - val_loss: 5.8069 - val_y1_accuracy: 0.3555 - val_y1_loss:
2.0225 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 3.7843
Epoch 12/100
26/26            9s 335ms/step -
loss: 6.6713 - y1_accuracy: 0.1651 - y1_loss: 3.1049 - y2_accuracy: 0.0950 -
y2_loss: 3.5664 - val_loss: 5.5064 - val_y1_accuracy: 0.5547 - val_y1_loss:
1.6027 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 3.9037
Epoch 13/100
26/26            8s 325ms/step -
loss: 6.5475 - y1_accuracy: 0.1894 - y1_loss: 3.0808 - y2_accuracy: 0.1380 -
y2_loss: 3.4667 - val_loss: 5.4457 - val_y1_accuracy: 0.5586 - val_y1_loss:
1.6093 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 3.8363
Epoch 14/100
```

```
26/26            8s 321ms/step -
loss: 6.3168 - y1_accuracy: 0.2276 - y1_loss: 2.9581 - y2_accuracy: 0.1346 -
y2_loss: 3.3587 - val_loss: 5.6390 - val_y1_accuracy: 0.3164 - val_y1_loss:
1.8806 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 3.7584
Epoch 15/100
26/26            8s 317ms/step -
loss: 6.2056 - y1_accuracy: 0.1794 - y1_loss: 2.8663 - y2_accuracy: 0.1394 -
y2_loss: 3.3393 - val_loss: 5.2009 - val_y1_accuracy: 0.6992 - val_y1_loss:
1.2327 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 3.9683
Epoch 16/100
26/26            8s 317ms/step -
loss: 6.0814 - y1_accuracy: 0.1901 - y1_loss: 2.8465 - y2_accuracy: 0.1566 -
y2_loss: 3.2349 - val_loss: 5.0745 - val_y1_accuracy: 0.7188 - val_y1_loss:
1.5060 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 3.5685
Epoch 17/100
26/26            8s 319ms/step -
loss: 5.8058 - y1_accuracy: 0.2457 - y1_loss: 2.7249 - y2_accuracy: 0.1830 -
y2_loss: 3.0809 - val_loss: 4.7007 - val_y1_accuracy: 0.3633 - val_y1_loss:
1.6179 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 3.0828
Epoch 18/100
26/26            8s 318ms/step -
loss: 5.7003 - y1_accuracy: 0.2531 - y1_loss: 2.7075 - y2_accuracy: 0.1701 -
y2_loss: 2.9927 - val_loss: 4.5111 - val_y1_accuracy: 0.7812 - val_y1_loss:
1.5371 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 2.9740
Epoch 19/100
26/26            8s 318ms/step -
loss: 5.7141 - y1_accuracy: 0.2139 - y1_loss: 2.7611 - y2_accuracy: 0.2201 -
y2_loss: 2.9530 - val_loss: 4.5322 - val_y1_accuracy: 0.8320 - val_y1_loss:
1.2829 - val_y2_accuracy: 0.0977 - val_y2_loss: 3.2493
Epoch 20/100
26/26            8s 319ms/step -
loss: 5.4573 - y1_accuracy: 0.2151 - y1_loss: 2.6263 - y2_accuracy: 0.2107 -
y2_loss: 2.8310 - val_loss: 5.2771 - val_y1_accuracy: 0.6328 - val_y1_loss:
1.7101 - val_y2_accuracy: 0.0039 - val_y2_loss: 3.5671
Epoch 21/100
26/26            8s 318ms/step -
loss: 5.2602 - y1_accuracy: 0.2508 - y1_loss: 2.5475 - y2_accuracy: 0.2315 -
y2_loss: 2.7127 - val_loss: 3.9410 - val_y1_accuracy: 0.8516 - val_y1_loss:
1.0681 - val_y2_accuracy: 0.2305 - val_y2_loss: 2.8729
Epoch 22/100
26/26            8s 317ms/step -
loss: 5.1351 - y1_accuracy: 0.2672 - y1_loss: 2.5165 - y2_accuracy: 0.2442 -
y2_loss: 2.6186 - val_loss: 5.5378 - val_y1_accuracy: 0.4414 - val_y1_loss:
1.9356 - val_y2_accuracy: 0.1016 - val_y2_loss: 3.6022
Epoch 23/100
26/26            8s 317ms/step -
loss: 5.1851 - y1_accuracy: 0.3078 - y1_loss: 2.4624 - y2_accuracy: 0.2403 -
y2_loss: 2.7227 - val_loss: 4.9013 - val_y1_accuracy: 0.4766 - val_y1_loss:
```

```
1.6466 - val_y2_accuracy: 0.0117 - val_y2_loss: 3.2548
Epoch 24/100
26/26            8s 318ms/step -
loss: 5.0056 - y1_accuracy: 0.2519 - y1_loss: 2.4782 - y2_accuracy: 0.2924 -
y2_loss: 2.5274 - val_loss: 5.0671 - val_y1_accuracy: 0.1641 - val_y1_loss:
2.2246 - val_y2_accuracy: 0.0977 - val_y2_loss: 2.8425
Epoch 25/100
26/26            8s 318ms/step -
loss: 4.6064 - y1_accuracy: 0.3255 - y1_loss: 2.2235 - y2_accuracy: 0.3054 -
y2_loss: 2.3829 - val_loss: 4.5401 - val_y1_accuracy: 0.6680 - val_y1_loss:
1.5250 - val_y2_accuracy: 0.1133 - val_y2_loss: 3.0151
Epoch 26/100
26/26            8s 317ms/step -
loss: 4.5854 - y1_accuracy: 0.3556 - y1_loss: 2.2038 - y2_accuracy: 0.3392 -
y2_loss: 2.3816 - val_loss: 4.2134 - val_y1_accuracy: 0.8320 - val_y1_loss:
1.1723 - val_y2_accuracy: 0.1758 - val_y2_loss: 3.0411
Epoch 27/100
26/26            8s 318ms/step -
loss: 4.2946 - y1_accuracy: 0.3780 - y1_loss: 2.1095 - y2_accuracy: 0.3462 -
y2_loss: 2.1851 - val_loss: 4.6877 - val_y1_accuracy: 0.5273 - val_y1_loss:
1.8509 - val_y2_accuracy: 0.0312 - val_y2_loss: 2.8368
Epoch 28/100
26/26            8s 318ms/step -
loss: 4.3603 - y1_accuracy: 0.3877 - y1_loss: 2.1117 - y2_accuracy: 0.3481 -
y2_loss: 2.2486 - val_loss: 4.1839 - val_y1_accuracy: 0.6875 - val_y1_loss:
1.5046 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 2.6793
Epoch 29/100
26/26            8s 317ms/step -
loss: 4.3005 - y1_accuracy: 0.3885 - y1_loss: 2.1142 - y2_accuracy: 0.3589 -
y2_loss: 2.1863 - val_loss: 3.7366 - val_y1_accuracy: 0.5859 - val_y1_loss:
1.5395 - val_y2_accuracy: 0.2695 - val_y2_loss: 2.1971
Epoch 30/100
26/26            8s 319ms/step -
loss: 4.1504 - y1_accuracy: 0.3711 - y1_loss: 2.0765 - y2_accuracy: 0.3930 -
y2_loss: 2.0739 - val_loss: 3.5592 - val_y1_accuracy: 0.8047 - val_y1_loss:
1.1293 - val_y2_accuracy: 0.0078 - val_y2_loss: 2.4298
Epoch 31/100
26/26            8s 318ms/step -
loss: 4.1280 - y1_accuracy: 0.3623 - y1_loss: 2.0393 - y2_accuracy: 0.3818 -
y2_loss: 2.0887 - val_loss: 3.3095 - val_y1_accuracy: 0.9414 - val_y1_loss:
0.7685 - val_y2_accuracy: 0.0312 - val_y2_loss: 2.5410
Epoch 32/100
26/26            8s 318ms/step -
loss: 4.3726 - y1_accuracy: 0.3674 - y1_loss: 2.0451 - y2_accuracy: 0.3527 -
y2_loss: 2.3275 - val_loss: 3.1222 - val_y1_accuracy: 0.8398 - val_y1_loss:
1.0049 - val_y2_accuracy: 0.2109 - val_y2_loss: 2.1174
Epoch 33/100
26/26            8s 318ms/step -
```

```
loss: 4.0683 - y1_accuracy: 0.3959 - y1_loss: 1.9730 - y2_accuracy: 0.4038 -
y2_loss: 2.0952 - val_loss: 3.7247 - val_y1_accuracy: 0.6758 - val_y1_loss:
1.4159 - val_y2_accuracy: 0.3359 - val_y2_loss: 2.3088
Epoch 34/100
26/26            8s 317ms/step -
loss: 3.5955 - y1_accuracy: 0.4697 - y1_loss: 1.7725 - y2_accuracy: 0.4624 -
y2_loss: 1.8229 - val_loss: 3.3181 - val_y1_accuracy: 0.7969 - val_y1_loss:
1.1715 - val_y2_accuracy: 0.4297 - val_y2_loss: 2.1465
Epoch 35/100
26/26            8s 317ms/step -
loss: 3.7830 - y1_accuracy: 0.4553 - y1_loss: 1.8907 - y2_accuracy: 0.4762 -
y2_loss: 1.8923 - val_loss: 3.4329 - val_y1_accuracy: 0.7383 - val_y1_loss:
1.2429 - val_y2_accuracy: 0.4375 - val_y2_loss: 2.1900
Epoch 36/100
26/26            8s 318ms/step -
loss: 3.7750 - y1_accuracy: 0.4490 - y1_loss: 1.8447 - y2_accuracy: 0.4277 -
y2_loss: 1.9304 - val_loss: 3.0471 - val_y1_accuracy: 0.8281 - val_y1_loss:
1.0152 - val_y2_accuracy: 0.4570 - val_y2_loss: 2.0319
Epoch 37/100
26/26            8s 317ms/step -
loss: 3.6439 - y1_accuracy: 0.4470 - y1_loss: 1.8221 - y2_accuracy: 0.4749 -
y2_loss: 1.8217 - val_loss: 2.9136 - val_y1_accuracy: 0.8242 - val_y1_loss:
0.9826 - val_y2_accuracy: 0.5078 - val_y2_loss: 1.9310
Epoch 38/100
26/26            8s 317ms/step -
loss: 3.5111 - y1_accuracy: 0.4553 - y1_loss: 1.7477 - y2_accuracy: 0.4832 -
y2_loss: 1.7635 - val_loss: 2.6273 - val_y1_accuracy: 0.8242 - val_y1_loss:
1.0300 - val_y2_accuracy: 0.6602 - val_y2_loss: 1.5972
Epoch 39/100
26/26            8s 318ms/step -
loss: 3.5232 - y1_accuracy: 0.4321 - y1_loss: 1.7734 - y2_accuracy: 0.5031 -
y2_loss: 1.7498 - val_loss: 2.9966 - val_y1_accuracy: 0.8008 - val_y1_loss:
0.9598 - val_y2_accuracy: 0.3750 - val_y2_loss: 2.0369
Epoch 40/100
26/26            8s 318ms/step -
loss: 3.4990 - y1_accuracy: 0.4630 - y1_loss: 1.7656 - y2_accuracy: 0.4955 -
y2_loss: 1.7334 - val_loss: 2.9064 - val_y1_accuracy: 0.7539 - val_y1_loss:
1.0814 - val_y2_accuracy: 0.4141 - val_y2_loss: 1.8251
Epoch 41/100
26/26            8s 319ms/step -
loss: 3.2370 - y1_accuracy: 0.4817 - y1_loss: 1.6127 - y2_accuracy: 0.4921 -
y2_loss: 1.6243 - val_loss: 2.7528 - val_y1_accuracy: 0.7539 - val_y1_loss:
1.0759 - val_y2_accuracy: 0.5977 - val_y2_loss: 1.6769
Epoch 42/100
26/26            8s 318ms/step -
loss: 3.2003 - y1_accuracy: 0.5048 - y1_loss: 1.6216 - y2_accuracy: 0.5472 -
y2_loss: 1.5787 - val_loss: 2.8407 - val_y1_accuracy: 0.8086 - val_y1_loss:
0.9755 - val_y2_accuracy: 0.5039 - val_y2_loss: 1.8652
```

```
Epoch 43/100
26/26            8s 317ms/step -
loss: 3.0858 - y1_accuracy: 0.5113 - y1_loss: 1.5416 - y2_accuracy: 0.5657 -
y2_loss: 1.5442 - val_loss: 3.2861 - val_y1_accuracy: 0.6719 - val_y1_loss:
1.1395 - val_y2_accuracy: 0.1602 - val_y2_loss: 2.1466
Epoch 44/100
26/26            8s 317ms/step -
loss: 3.0692 - y1_accuracy: 0.5077 - y1_loss: 1.5375 - y2_accuracy: 0.5322 -
y2_loss: 1.5317 - val_loss: 2.8457 - val_y1_accuracy: 0.7148 - val_y1_loss:
1.0562 - val_y2_accuracy: 0.4805 - val_y2_loss: 1.7895
Epoch 45/100
26/26            8s 316ms/step -
loss: 3.1304 - y1_accuracy: 0.5291 - y1_loss: 1.5591 - y2_accuracy: 0.5682 -
y2_loss: 1.5713 - val_loss: 2.9783 - val_y1_accuracy: 0.7773 - val_y1_loss:
0.8862 - val_y2_accuracy: 0.3008 - val_y2_loss: 2.0920
Epoch 46/100
26/26            8s 317ms/step -
loss: 3.1857 - y1_accuracy: 0.5252 - y1_loss: 1.5566 - y2_accuracy: 0.5502 -
y2_loss: 1.6291 - val_loss: 2.6968 - val_y1_accuracy: 0.7422 - val_y1_loss:
0.8514 - val_y2_accuracy: 0.4023 - val_y2_loss: 1.8454
Epoch 47/100
26/26            8s 317ms/step -
loss: 3.0491 - y1_accuracy: 0.5181 - y1_loss: 1.5840 - y2_accuracy: 0.5832 -
y2_loss: 1.4651 - val_loss: 2.8362 - val_y1_accuracy: 0.7344 - val_y1_loss:
1.1971 - val_y2_accuracy: 0.6406 - val_y2_loss: 1.6390
Epoch 48/100
26/26            8s 317ms/step -
loss: 2.7570 - y1_accuracy: 0.5675 - y1_loss: 1.3984 - y2_accuracy: 0.6037 -
y2_loss: 1.3586 - val_loss: 2.6774 - val_y1_accuracy: 0.7734 - val_y1_loss:
1.1146 - val_y2_accuracy: 0.6289 - val_y2_loss: 1.5628
Epoch 49/100
26/26            8s 319ms/step -
loss: 2.6502 - y1_accuracy: 0.5903 - y1_loss: 1.3120 - y2_accuracy: 0.6042 -
y2_loss: 1.3383 - val_loss: 3.3173 - val_y1_accuracy: 0.5078 - val_y1_loss:
1.4338 - val_y2_accuracy: 0.4727 - val_y2_loss: 1.8834
Epoch 50/100
26/26            8s 321ms/step -
loss: 2.8684 - y1_accuracy: 0.5832 - y1_loss: 1.4312 - y2_accuracy: 0.6049 -
y2_loss: 1.4372 - val_loss: 2.7101 - val_y1_accuracy: 0.6523 - val_y1_loss:
1.1113 - val_y2_accuracy: 0.5312 - val_y2_loss: 1.5988
Epoch 51/100
26/26            9s 339ms/step -
loss: 2.5653 - y1_accuracy: 0.5689 - y1_loss: 1.3260 - y2_accuracy: 0.6564 -
y2_loss: 1.2393 - val_loss: 2.4103 - val_y1_accuracy: 0.7266 - val_y1_loss:
0.8630 - val_y2_accuracy: 0.4688 - val_y2_loss: 1.5473
Epoch 52/100
26/26            9s 332ms/step -
loss: 2.9250 - y1_accuracy: 0.5491 - y1_loss: 1.4401 - y2_accuracy: 0.5639 -
```

```
y2_loss: 1.4850 - val_loss: 3.2459 - val_y1_accuracy: 0.5234 - val_y1_loss:
1.0542 - val_y2_accuracy: 0.3125 - val_y2_loss: 2.1917
Epoch 53/100
26/26            8s 326ms/step -
loss: 2.9608 - y1_accuracy: 0.5778 - y1_loss: 1.4651 - y2_accuracy: 0.5864 -
y2_loss: 1.4957 - val_loss: 3.1502 - val_y1_accuracy: 0.6484 - val_y1_loss:
1.1563 - val_y2_accuracy: 0.4219 - val_y2_loss: 1.9939
Epoch 54/100
26/26            8s 320ms/step -
loss: 2.8361 - y1_accuracy: 0.5433 - y1_loss: 1.4255 - y2_accuracy: 0.6080 -
y2_loss: 1.4106 - val_loss: 3.8157 - val_y1_accuracy: 0.3711 - val_y1_loss:
1.4696 - val_y2_accuracy: 0.2852 - val_y2_loss: 2.3461
Epoch 55/100
26/26            8s 324ms/step -
loss: 2.8684 - y1_accuracy: 0.5888 - y1_loss: 1.4337 - y2_accuracy: 0.5978 -
y2_loss: 1.4347 - val_loss: 2.8054 - val_y1_accuracy: 0.7617 - val_y1_loss:
1.1013 - val_y2_accuracy: 0.4648 - val_y2_loss: 1.7041
Epoch 56/100
26/26            8s 321ms/step -
loss: 2.5166 - y1_accuracy: 0.6538 - y1_loss: 1.2599 - y2_accuracy: 0.6206 -
y2_loss: 1.2567 - val_loss: 2.8365 - val_y1_accuracy: 0.4141 - val_y1_loss:
1.4568 - val_y2_accuracy: 0.6719 - val_y2_loss: 1.3797
Epoch 57/100
26/26            8s 320ms/step -
loss: 2.8169 - y1_accuracy: 0.5421 - y1_loss: 1.4233 - y2_accuracy: 0.6016 -
y2_loss: 1.3936 - val_loss: 2.2840 - val_y1_accuracy: 0.8086 - val_y1_loss:
0.8256 - val_y2_accuracy: 0.6719 - val_y2_loss: 1.4584
Epoch 58/100
26/26            8s 322ms/step -
loss: 2.6353 - y1_accuracy: 0.5876 - y1_loss: 1.3186 - y2_accuracy: 0.6354 -
y2_loss: 1.3167 - val_loss: 2.6808 - val_y1_accuracy: 0.6172 - val_y1_loss:
1.2170 - val_y2_accuracy: 0.6211 - val_y2_loss: 1.4637
Epoch 59/100
26/26            8s 324ms/step -
loss: 2.6248 - y1_accuracy: 0.5964 - y1_loss: 1.3164 - y2_accuracy: 0.6332 -
y2_loss: 1.3084 - val_loss: 2.0996 - val_y1_accuracy: 0.7891 - val_y1_loss:
0.8856 - val_y2_accuracy: 0.7422 - val_y2_loss: 1.2140
Epoch 60/100
26/26            8s 321ms/step -
loss: 2.5531 - y1_accuracy: 0.6465 - y1_loss: 1.2521 - y2_accuracy: 0.6347 -
y2_loss: 1.3010 - val_loss: 2.4960 - val_y1_accuracy: 0.7891 - val_y1_loss:
1.0198 - val_y2_accuracy: 0.6289 - val_y2_loss: 1.4762
Epoch 61/100
26/26            8s 322ms/step -
loss: 2.5534 - y1_accuracy: 0.6126 - y1_loss: 1.2999 - y2_accuracy: 0.6329 -
y2_loss: 1.2534 - val_loss: 3.2476 - val_y1_accuracy: 0.3984 - val_y1_loss:
1.3914 - val_y2_accuracy: 0.4141 - val_y2_loss: 1.8562
Epoch 62/100
```

```
26/26            8s 322ms/step -
loss: 2.3481 - y1_accuracy: 0.6387 - y1_loss: 1.1817 - y2_accuracy: 0.6502 -
y2_loss: 1.1665 - val_loss: 2.2974 - val_y1_accuracy: 0.5898 - val_y1_loss:
1.0102 - val_y2_accuracy: 0.6992 - val_y2_loss: 1.2872
Epoch 63/100
26/26            8s 320ms/step -
loss: 2.5564 - y1_accuracy: 0.6274 - y1_loss: 1.2835 - y2_accuracy: 0.6118 -
y2_loss: 1.2729 - val_loss: 2.2705 - val_y1_accuracy: 0.7812 - val_y1_loss:
0.9071 - val_y2_accuracy: 0.6758 - val_y2_loss: 1.3634
Epoch 64/100
26/26            8s 321ms/step -
loss: 2.2656 - y1_accuracy: 0.6412 - y1_loss: 1.1247 - y2_accuracy: 0.6668 -
y2_loss: 1.1409 - val_loss: 2.5515 - val_y1_accuracy: 0.5859 - val_y1_loss:
1.3311 - val_y2_accuracy: 0.6719 - val_y2_loss: 1.2204
Epoch 65/100
26/26            8s 321ms/step -
loss: 2.5337 - y1_accuracy: 0.6183 - y1_loss: 1.2916 - y2_accuracy: 0.6380 -
y2_loss: 1.2421 - val_loss: 2.3884 - val_y1_accuracy: 0.7383 - val_y1_loss:
0.8898 - val_y2_accuracy: 0.5000 - val_y2_loss: 1.4985
Epoch 66/100
26/26            8s 322ms/step -
loss: 2.1183 - y1_accuracy: 0.6497 - y1_loss: 1.0720 - y2_accuracy: 0.7027 -
y2_loss: 1.0464 - val_loss: 2.5767 - val_y1_accuracy: 0.6523 - val_y1_loss:
1.1062 - val_y2_accuracy: 0.5898 - val_y2_loss: 1.4705
Epoch 67/100
26/26            8s 321ms/step -
loss: 2.3692 - y1_accuracy: 0.6252 - y1_loss: 1.1763 - y2_accuracy: 0.6471 -
y2_loss: 1.1929 - val_loss: 2.7212 - val_y1_accuracy: 0.6055 - val_y1_loss:
1.0726 - val_y2_accuracy: 0.5039 - val_y2_loss: 1.6486
Epoch 68/100
22/26            1s 287ms/step -
loss: 2.4951 - y1_accuracy: 0.6544 - y1_loss: 1.2294 - y2_accuracy: 0.6221 -
y2_loss: 1.2655

/Users/lucaperfetti/Desktop/università/Secondo Anno/Advanced
ML/Project/.venv/lib/python3.10/site-
packages/keras/src/trainers/epoch_iterator.py:107: UserWarning: Your input ran
out of data; interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to use the
`.repeat()` function when building your dataset.
  self._interrupted_warning()

26/26            7s 272ms/step -
loss: 2.4909 - y1_accuracy: 0.6512 - y1_loss: 1.2288 - y2_accuracy: 0.6241 -
y2_loss: 1.2619 - val_loss: 2.1217 - val_y1_accuracy: 0.8789 - val_y1_loss:
0.7421 - val_y2_accuracy: 0.6367 - val_y2_loss: 1.3797
Epoch 69/100
26/26            8s 317ms/step -
loss: 2.2063 - y1_accuracy: 0.6646 - y1_loss: 1.1293 - y2_accuracy: 0.6747 -
```

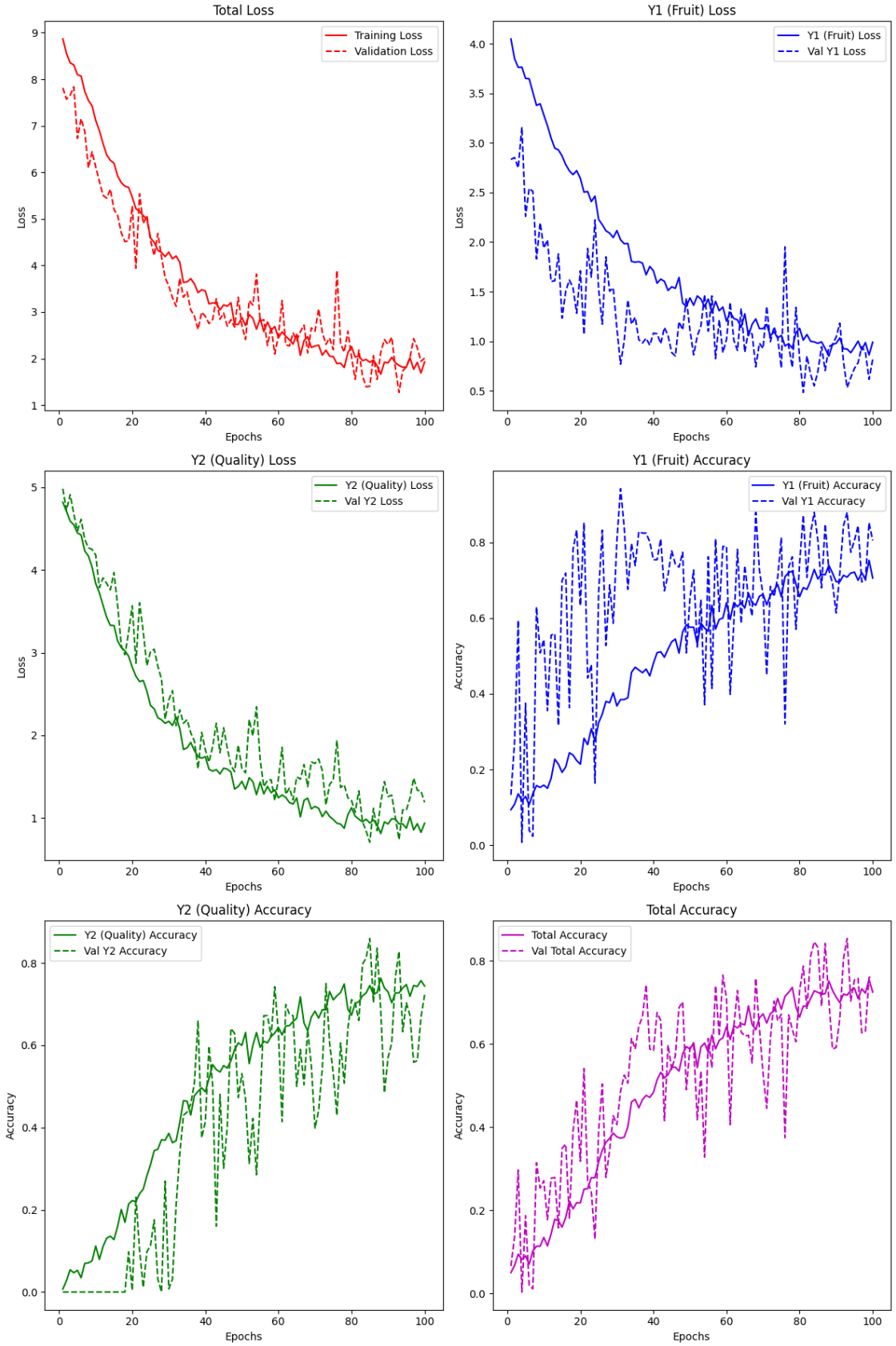
```
y2_loss: 1.0770 - val_loss: 2.6595 - val_y1_accuracy: 0.7266 - val_y1_loss:
0.9806 - val_y2_accuracy: 0.5508 - val_y2_loss: 1.6789
Epoch 70/100
26/26            8s 318ms/step -
loss: 2.1878 - y1_accuracy: 0.6791 - y1_loss: 1.0872 - y2_accuracy: 0.6805 -
y2_loss: 1.1006 - val_loss: 2.5876 - val_y1_accuracy: 0.6602 - val_y1_loss:
0.9269 - val_y2_accuracy: 0.3984 - val_y2_loss: 1.6607
Epoch 71/100
26/26            8s 317ms/step -
loss: 2.2820 - y1_accuracy: 0.6366 - y1_loss: 1.1508 - y2_accuracy: 0.6581 -
y2_loss: 1.1312 - val_loss: 3.0653 - val_y1_accuracy: 0.4492 - val_y1_loss:
1.3510 - val_y2_accuracy: 0.4414 - val_y2_loss: 1.7142
Epoch 72/100
26/26            8s 318ms/step -
loss: 2.0265 - y1_accuracy: 0.6715 - y1_loss: 1.0068 - y2_accuracy: 0.6713 -
y2_loss: 1.0196 - val_loss: 2.5706 - val_y1_accuracy: 0.6836 - val_y1_loss:
0.9955 - val_y2_accuracy: 0.5430 - val_y2_loss: 1.5751
Epoch 73/100
26/26            8s 317ms/step -
loss: 2.0367 - y1_accuracy: 0.6849 - y1_loss: 1.0145 - y2_accuracy: 0.6944 -
y2_loss: 1.0223 - val_loss: 2.3055 - val_y1_accuracy: 0.6562 - val_y1_loss:
1.1461 - val_y2_accuracy: 0.7500 - val_y2_loss: 1.1594
Epoch 74/100
26/26            8s 318ms/step -
loss: 2.0867 - y1_accuracy: 0.6972 - y1_loss: 1.0360 - y2_accuracy: 0.7176 -
y2_loss: 1.0508 - val_loss: 2.4414 - val_y1_accuracy: 0.7070 - val_y1_loss:
1.0339 - val_y2_accuracy: 0.6055 - val_y2_loss: 1.4075
Epoch 75/100
26/26            8s 317ms/step -
loss: 1.9928 - y1_accuracy: 0.6569 - y1_loss: 1.0278 - y2_accuracy: 0.7238 -
y2_loss: 0.9650 - val_loss: 2.1918 - val_y1_accuracy: 0.8125 - val_y1_loss:
0.7308 - val_y2_accuracy: 0.5312 - val_y2_loss: 1.4610
Epoch 76/100
26/26            8s 318ms/step -
loss: 1.8616 - y1_accuracy: 0.6990 - y1_loss: 0.9406 - y2_accuracy: 0.7117 -
y2_loss: 0.9211 - val_loss: 3.8903 - val_y1_accuracy: 0.3203 - val_y1_loss:
1.9514 - val_y2_accuracy: 0.4297 - val_y2_loss: 1.9390
Epoch 77/100
26/26            8s 317ms/step -
loss: 1.9651 - y1_accuracy: 0.7215 - y1_loss: 1.0034 - y2_accuracy: 0.7288 -
y2_loss: 0.9617 - val_loss: 2.3203 - val_y1_accuracy: 0.7344 - val_y1_loss:
0.9503 - val_y2_accuracy: 0.6055 - val_y2_loss: 1.3700
Epoch 78/100
26/26            8s 319ms/step -
loss: 1.7634 - y1_accuracy: 0.7462 - y1_loss: 0.8920 - y2_accuracy: 0.7584 -
y2_loss: 0.8713 - val_loss: 2.1400 - val_y1_accuracy: 0.7617 - val_y1_loss:
0.7373 - val_y2_accuracy: 0.5078 - val_y2_loss: 1.4026
Epoch 79/100
```

```
26/26            8s 318ms/step -
loss: 2.2037 - y1_accuracy: 0.6826 - y1_loss: 1.1218 - y2_accuracy: 0.6692 -
y2_loss: 1.0819 - val_loss: 2.5875 - val_y1_accuracy: 0.5703 - val_y1_loss:
1.3414 - val_y2_accuracy: 0.6406 - val_y2_loss: 1.2461
Epoch 80/100
26/26            8s 318ms/step -
loss: 2.0093 - y1_accuracy: 0.6851 - y1_loss: 1.0249 - y2_accuracy: 0.7129 -
y2_loss: 0.9843 - val_loss: 2.0348 - val_y1_accuracy: 0.7461 - val_y1_loss:
0.8225 - val_y2_accuracy: 0.7109 - val_y2_loss: 1.2123
Epoch 81/100
26/26            8s 320ms/step -
loss: 2.0864 - y1_accuracy: 0.6662 - y1_loss: 1.0565 - y2_accuracy: 0.7071 -
y2_loss: 1.0298 - val_loss: 1.5556 - val_y1_accuracy: 0.8711 - val_y1_loss:
0.4842 - val_y2_accuracy: 0.7031 - val_y2_loss: 1.0713
Epoch 82/100
26/26            8s 318ms/step -
loss: 2.0432 - y1_accuracy: 0.6758 - y1_loss: 1.0557 - y2_accuracy: 0.6951 -
y2_loss: 0.9875 - val_loss: 2.1805 - val_y1_accuracy: 0.7070 - val_y1_loss:
0.8511 - val_y2_accuracy: 0.6602 - val_y2_loss: 1.3293
Epoch 83/100
26/26            8s 318ms/step -
loss: 2.0407 - y1_accuracy: 0.6822 - y1_loss: 1.0326 - y2_accuracy: 0.7061 -
y2_loss: 1.0082 - val_loss: 1.6292 - val_y1_accuracy: 0.8242 - val_y1_loss:
0.6693 - val_y2_accuracy: 0.7969 - val_y2_loss: 0.9599
Epoch 84/100
26/26            8s 318ms/step -
loss: 1.9670 - y1_accuracy: 0.7231 - y1_loss: 0.9923 - y2_accuracy: 0.7381 -
y2_loss: 0.9747 - val_loss: 1.3936 - val_y1_accuracy: 0.8789 - val_y1_loss:
0.5503 - val_y2_accuracy: 0.8125 - val_y2_loss: 0.8432
Epoch 85/100
26/26            8s 319ms/step -
loss: 1.9470 - y1_accuracy: 0.7089 - y1_loss: 0.9953 - y2_accuracy: 0.7454 -
y2_loss: 0.9517 - val_loss: 1.4062 - val_y1_accuracy: 0.8047 - val_y1_loss:
0.6977 - val_y2_accuracy: 0.8594 - val_y2_loss: 0.7084
Epoch 86/100
26/26            8s 319ms/step -
loss: 1.9830 - y1_accuracy: 0.7238 - y1_loss: 1.0011 - y2_accuracy: 0.7143 -
y2_loss: 0.9819 - val_loss: 2.0643 - val_y1_accuracy: 0.6797 - val_y1_loss:
0.9455 - val_y2_accuracy: 0.7031 - val_y2_loss: 1.1188
Epoch 87/100
26/26            8s 319ms/step -
loss: 1.9665 - y1_accuracy: 0.6962 - y1_loss: 0.9787 - y2_accuracy: 0.6971 -
y2_loss: 0.9877 - val_loss: 1.5525 - val_y1_accuracy: 0.8477 - val_y1_loss:
0.7049 - val_y2_accuracy: 0.8359 - val_y2_loss: 0.8476
Epoch 88/100
26/26            8s 319ms/step -
loss: 1.7888 - y1_accuracy: 0.7228 - y1_loss: 0.9172 - y2_accuracy: 0.7498 -
y2_loss: 0.8716 - val_loss: 2.1328 - val_y1_accuracy: 0.7266 - val_y1_loss:
```

```
0.9447 - val_y2_accuracy: 0.6836 - val_y2_loss: 1.1881
Epoch 89/100
26/26            8s 322ms/step -
loss: 2.0177 - y1_accuracy: 0.7310 - y1_loss: 1.0012 - y2_accuracy: 0.7269 -
y2_loss: 1.0166 - val_loss: 2.4088 - val_y1_accuracy: 0.6914 - val_y1_loss:
0.9677 - val_y2_accuracy: 0.4844 - val_y2_loss: 1.4411
Epoch 90/100
26/26            8s 320ms/step -
loss: 2.0009 - y1_accuracy: 0.6899 - y1_loss: 1.0053 - y2_accuracy: 0.7223 -
y2_loss: 0.9956 - val_loss: 2.3030 - val_y1_accuracy: 0.6133 - val_y1_loss:
1.0428 - val_y2_accuracy: 0.5703 - val_y2_loss: 1.2601
Epoch 91/100
26/26            8s 318ms/step -
loss: 1.9739 - y1_accuracy: 0.6978 - y1_loss: 1.0085 - y2_accuracy: 0.7061 -
y2_loss: 0.9654 - val_loss: 2.4607 - val_y1_accuracy: 0.7188 - val_y1_loss:
1.1830 - val_y2_accuracy: 0.6055 - val_y2_loss: 1.2777
Epoch 92/100
26/26            8s 320ms/step -
loss: 2.0132 - y1_accuracy: 0.6929 - y1_loss: 0.9652 - y2_accuracy: 0.6942 -
y2_loss: 1.0480 - val_loss: 1.7996 - val_y1_accuracy: 0.8438 - val_y1_loss:
0.7935 - val_y2_accuracy: 0.7656 - val_y2_loss: 1.0061
Epoch 93/100
26/26            8s 319ms/step -
loss: 1.7638 - y1_accuracy: 0.6956 - y1_loss: 0.8745 - y2_accuracy: 0.7186 -
y2_loss: 0.8894 - val_loss: 1.2730 - val_y1_accuracy: 0.8789 - val_y1_loss:
0.5332 - val_y2_accuracy: 0.8281 - val_y2_loss: 0.7399
Epoch 94/100
26/26            8s 319ms/step -
loss: 1.9598 - y1_accuracy: 0.6874 - y1_loss: 0.9603 - y2_accuracy: 0.7252 -
y2_loss: 0.9995 - val_loss: 1.7348 - val_y1_accuracy: 0.7734 - val_y1_loss:
0.6343 - val_y2_accuracy: 0.6328 - val_y2_loss: 1.1006
Epoch 95/100
26/26            8s 320ms/step -
loss: 1.8869 - y1_accuracy: 0.7046 - y1_loss: 0.9547 - y2_accuracy: 0.7259 -
y2_loss: 0.9323 - val_loss: 1.8223 - val_y1_accuracy: 0.8047 - val_y1_loss:
0.7264 - val_y2_accuracy: 0.7031 - val_y2_loss: 1.0959
Epoch 96/100
26/26            8s 319ms/step -
loss: 1.9686 - y1_accuracy: 0.7015 - y1_loss: 0.9800 - y2_accuracy: 0.7150 -
y2_loss: 0.9886 - val_loss: 2.0263 - val_y1_accuracy: 0.8438 - val_y1_loss:
0.7839 - val_y2_accuracy: 0.6719 - val_y2_loss: 1.2424
Epoch 97/100
26/26            8s 320ms/step -
loss: 1.6880 - y1_accuracy: 0.7304 - y1_loss: 0.8820 - y2_accuracy: 0.7572 -
y2_loss: 0.8060 - val_loss: 2.4284 - val_y1_accuracy: 0.6953 - val_y1_loss:
0.9451 - val_y2_accuracy: 0.5586 - val_y2_loss: 1.4834
Epoch 98/100
26/26            8s 320ms/step -
```

```
loss: 1.8860 - y1_accuracy: 0.7024 - y1_loss: 0.9795 - y2_accuracy: 0.7552 -
y2_loss: 0.9065 - val_loss: 2.2292 - val_y1_accuracy: 0.6992 - val_y1_loss:
0.8921 - val_y2_accuracy: 0.5625 - val_y2_loss: 1.3371
Epoch 99/100
26/26             8s 318ms/step -
loss: 1.6482 - y1_accuracy: 0.7426 - y1_loss: 0.8556 - y2_accuracy: 0.7656 -
y2_loss: 0.7926 - val_loss: 1.9422 - val_y1_accuracy: 0.8516 - val_y1_loss:
0.6160 - val_y2_accuracy: 0.6680 - val_y2_loss: 1.3262
Epoch 100/100
26/26             8s 320ms/step -
loss: 1.9306 - y1_accuracy: 0.7048 - y1_loss: 0.9900 - y2_accuracy: 0.7261 -
y2_loss: 0.9406 - val_loss: 2.0099 - val_y1_accuracy: 0.8047 - val_y1_loss:
0.8187 - val_y2_accuracy: 0.7227 - val_y2_loss: 1.1911
```

[ ]:
```
if(not( os.path.exists('keras/model-CNN-AUG.keras') )):
    plot_training_history(history)
```


Total Loss
Training Loss
Validation Loss
Loss
Epochs
Y1 (Fruit) Loss
Y1 (Fruit) Loss
Val Y1 Loss
Loss
Epochs
Y2 (Quality) Loss
Y2 (Quality) Loss
Val Y2 Loss
Loss
Epochs
Y1 (Fruit) Accuracy
Y1 (Fruit) Accuracy
Val Y1 Accuracy
Accuracy
Epochs
Y2 (Quality) Accuracy
Y2 (Quality) Accuracy
Val Y2 Accuracy
Accuracy
Epochs
Total Accuracy
Total Accuracy
Val Total Accuracy
Accuracy
Epochs

Salvataggio in memoria

[ ]:
```
model.save("keras/model-CNN-AUG.keras")
```

[ ]:
```
model = load_model("keras/model-CNN-AUG.keras")
```

Valutazione metriche di classificazione

[ ]:
```
y1_pred, y2_pred = model.predict(x_test)

y1_pred_classes = np.argmax(y1_pred, axis=1)
y2_pred_classes = np.argmax(y2_pred, axis=1)

y1_test_classes = np.argmax(y1_test, axis=1)
y2_test_classes = np.argmax(y2_test, axis=1)
```

```
440/440             40s 92ms/step
```

[ ]:
```
# Valutazione del modello sul test set
loss, loss_y1, loss_y2, acc_y1, acc_y2 = model.evaluate(x_test, {"y1": y1_test,␣
 ↪"y2": y2_test}, verbose=0)

print(f"Loss Totale: {loss:.4f}")
print(f"Loss Y1 (Fruit): {loss_y1:.4f}")
print(f"Loss Y2 (Quality): {loss_y2:.4f}")
print(f"Accuracy Y1 (Fruit): {acc_y1:.4f}")
print(f"Accuracy Y2 (Quality): {acc_y2:.4f}")
```

```
Loss Totale: 1.4089
Loss Y1 (Fruit): 0.6969
Loss Y2 (Quality): 0.7130
Accuracy Y1 (Fruit): 0.8044
Accuracy Y2 (Quality): 0.8148
```

[ ]:
```
#F1-score
y_pred_m1 = (y1_pred > 0.5).astype(int)

f1 = f1_score(y1_test, y_pred_m1, average="weighted")
print(f'F1 Score: {f1}')
```

```
F1 Score: 0.779908611977167
```

[ ]:
```
#F1-score
y_pred_m2 = (y2_pred > 0.5).astype(int)

f2 = f1_score(y2_test, y_pred_m2, average="weighted")
print(f'F1 Score: {f2}')
```

```
F1 Score: 0.7817022323710353
```

[ ]:
```
print("Classification Report per Y1 (Frutto):")
print(classification_report(y1_test_classes, y1_pred_classes))

print("Classification Report per Y2 (Qualità):")
print(classification_report(y2_test_classes, y2_pred_classes))
```

```
Classification Report per Y1 (Frutto):
              precision    recall  f1-score   support

           0       0.81      0.81      0.81      1510
           1       0.75      0.92      0.83        98
           2       0.94      0.74      0.83       183
           3       0.65      0.86      0.74       286
           4       0.85      0.76      0.80        90
           5       0.84      0.71      0.77        92
           6       0.83      0.89      0.86        28
           7       0.61      0.69      0.65        98
           8       0.91      0.91      0.91       196
           9       0.63      0.68      0.66        98
          10       1.00      0.93      0.97        30
          11       0.90      0.89      0.89       140
          12       0.85      0.91      0.88       686
          13       0.78      0.63      0.70        90
          14       0.76      1.00      0.86        98
          15       0.98      0.88      0.92        98
          16       0.75      0.94      0.83       182
          17       0.73      0.92      0.82       249
          18       0.84      0.86      0.85        98
          19       0.87      0.90      0.89       141
          20       0.85      0.91      0.88       140
          21       0.86      0.32      0.47        59
          22       0.93      0.81      0.86        98
          23       0.86      0.80      0.83       682
          24       0.46      0.41      0.43       196
          25       0.87      0.80      0.83       100
          26       0.95      0.85      0.90        92
          27       0.98      1.00      0.99        98
          28       0.94      0.77      0.84        98
          29       0.94      0.72      0.82        93
          30       0.48      0.77      0.59        94
          31       0.89      0.95      0.92        98
          32       0.93      0.63      0.75       196
          33       0.73      0.92      0.81        98
          34       0.87      0.94      0.90        98
          35       0.71      0.24      0.36        98
          36       0.69      0.72      0.71       183
```

```
              37       0.63      0.83      0.72        60
              38       0.83      0.86      0.84        98
              39       0.88      0.73      0.80       147
              40       0.94      0.90      0.92        98
              41       0.49      0.40      0.44       194
              42       0.74      0.82      0.78       236
              43       0.82      0.80      0.81       266
              44       0.75      0.84      0.80        95
              45       0.77      0.88      0.82        98
              46       0.87      0.83      0.85        98
              47       0.79      0.52      0.63       343
              48       0.77      0.73      0.75      1049
              49       0.94      0.89      0.91        98
              50       0.87      0.94      0.90       494
              51       0.79      0.94      0.86       196
              52       0.93      0.79      0.85       196
              53       0.94      0.90      0.92        98
              54       0.82      0.81      0.82       353
              55       0.58      0.69      0.63        98
              56       0.93      0.79      0.86        90
              57       0.74      0.88      0.80       360
              58       0.68      0.90      0.78        98
              59       0.70      0.88      0.78        98
              60       1.00      1.00      1.00        98
              61       0.93      0.94      0.93        98
              62       0.87      0.82      0.84        98
              63       0.78      0.74      0.76       245
              64       0.95      0.56      0.71        98
              65       0.53      0.91      0.67        98
              66       0.87      0.91      0.89      1015
              67       0.91      0.91      0.91       147
              68       0.85      0.43      0.57        95
              69       0.92      0.38      0.53        96

    accuracy                             0.80     14061
   macro avg       0.81      0.79      0.79     14061
weighted avg       0.81      0.80      0.80     14061


 Classification Report per Y2 (Qualità):
              precision    recall  f1-score   support

           0       0.80      0.70      0.75        94
           1       0.70      0.64      0.67        98
           2       0.77      0.85      0.81        88
           3       0.82      0.87      0.84       290
           4       0.92      0.82      0.86        98
           5       0.78      0.86      0.82       140
```

6	0.83	0.77	0.80	91
7	0.80	0.65	0.72	281
8	0.92	0.55	0.69	98
9	0.89	0.74	0.81	232
10	0.70	0.90	0.79	98
11	0.89	0.95	0.92	85
12	0.89	0.69	0.78	98
13	0.74	0.94	0.83	98
14	0.71	0.91	0.80	90
15	0.65	0.89	0.75	98
16	0.83	0.74	0.78	90
17	0.86	0.70	0.77	92
18	0.86	0.68	0.76	28
19	0.59	0.76	0.66	98
20	0.91	0.91	0.91	196
21	0.57	0.61	0.59	98
22	1.00	0.93	0.97	30
23	0.86	0.89	0.87	140
24	0.87	0.88	0.87	245
25	0.89	0.83	0.86	147
26	0.87	0.92	0.89	98
27	0.78	0.89	0.83	98
28	0.93	0.91	0.92	98
29	0.83	0.58	0.68	90
30	0.76	0.97	0.85	98
31	0.98	0.88	0.92	98
32	0.83	0.91	0.87	90
33	0.79	0.97	0.87	92
34	0.81	0.96	0.88	78
35	0.88	0.91	0.89	171
36	0.88	0.82	0.85	98
37	0.86	0.92	0.89	93
38	0.84	0.96	0.89	48
39	0.83	0.92	0.87	140
40	0.85	0.37	0.52	59
41	0.99	0.84	0.91	98
42	0.92	0.82	0.87	196
43	0.82	0.64	0.72	98
44	0.81	0.93	0.87	388
45	0.31	0.24	0.27	98
46	0.67	0.79	0.72	98
47	0.86	0.85	0.85	100
48	0.92	0.90	0.91	92
49	0.98	1.00	0.99	98
50	0.92	0.70	0.80	98
51	0.94	0.71	0.81	93
52	0.61	0.76	0.67	94
53	0.90	0.96	0.93	98

54	0.91	0.33	0.48	98
55	0.95	0.97	0.96	98
56	0.76	0.92	0.83	98
57	0.90	0.93	0.91	98
58	0.63	0.22	0.33	98
59	0.80	0.81	0.80	98
60	0.67	0.61	0.64	85
61	0.60	0.83	0.70	60
62	0.81	0.84	0.82	98
63	0.86	0.69	0.77	147
64	0.94	0.90	0.92	98
65	0.72	0.50	0.59	98
66	0.66	0.64	0.65	96
67	0.74	0.92	0.82	130
68	0.95	0.82	0.88	106
69	0.83	0.82	0.83	90
70	0.77	0.91	0.84	89
71	0.86	0.78	0.82	87
72	0.78	0.84	0.81	95
73	0.76	0.88	0.82	98
74	0.90	0.81	0.85	98
75	0.74	0.78	0.76	245
76	0.89	0.48	0.62	98
77	0.83	0.82	0.83	280
78	0.68	0.85	0.75	98
79	0.82	0.78	0.80	140
80	0.90	0.95	0.93	60
81	0.87	0.85	0.86	98
82	0.86	0.92	0.89	133
83	0.87	0.76	0.81	142
84	0.97	0.57	0.72	98
85	0.92	0.90	0.91	98
86	0.92	0.89	0.90	88
87	0.73	0.94	0.82	140
88	0.87	0.92	0.89	133
89	0.89	0.93	0.91	133
90	0.82	0.99	0.90	98
91	0.72	0.89	0.80	98
92	0.94	0.82	0.87	98
93	1.00	0.71	0.83	98
94	0.94	0.95	0.94	98
95	0.81	0.83	0.82	353
96	0.53	0.81	0.64	98
97	0.95	0.79	0.86	90
98	0.54	0.16	0.24	90
99	0.72	0.91	0.80	90
100	0.54	0.77	0.63	90
101	0.62	0.84	0.71	90

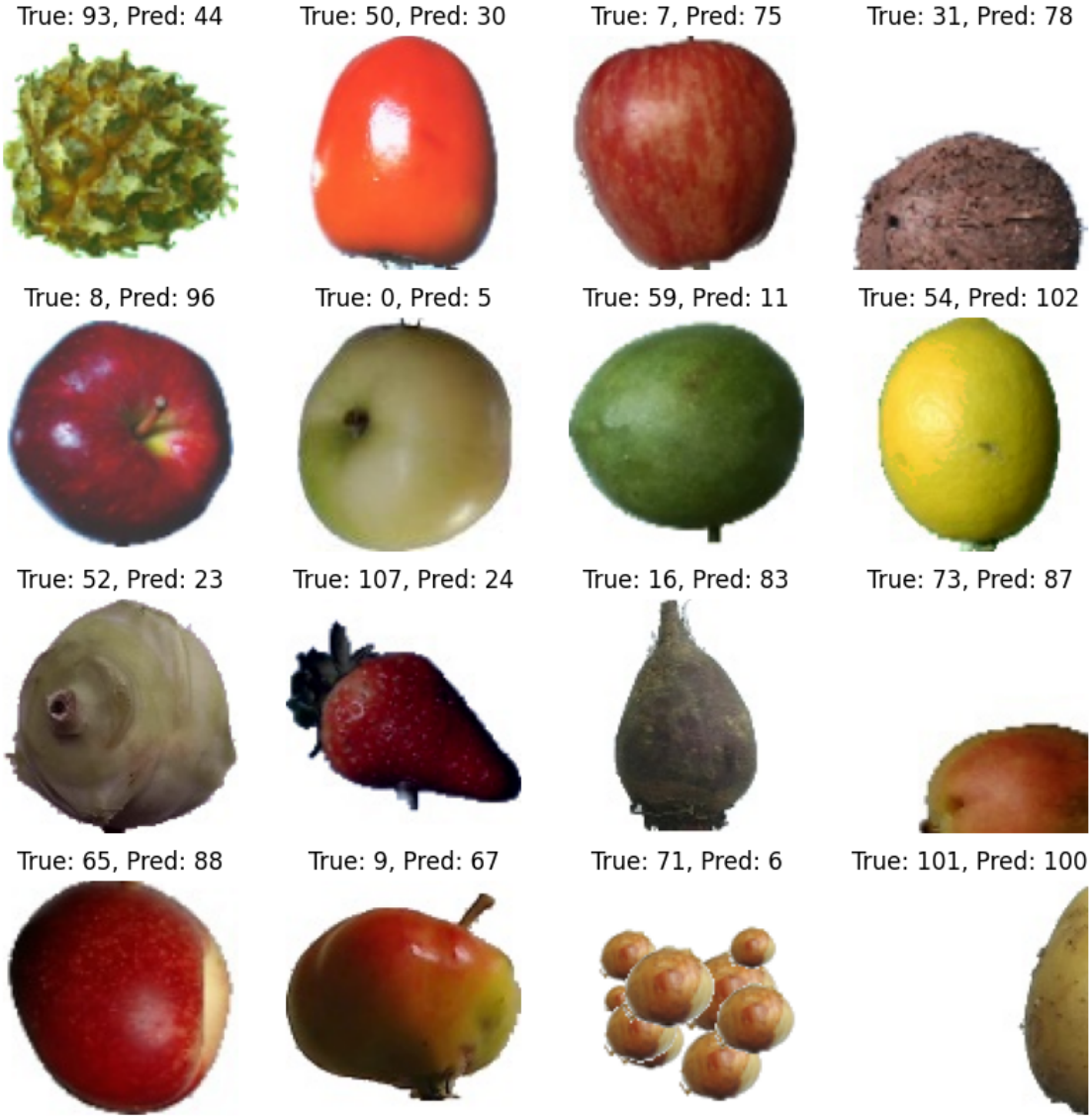
```
           102       0.87      0.87      0.87        98
           103       0.78      0.87      0.82        98
           104       1.00      1.00      1.00        98
           105       0.93      0.94      0.93        98
           106       0.95      0.80      0.87        98
           107       0.95      0.61      0.75        98
           108       0.74      0.89      0.81       147
           109       0.97      0.68      0.80        98
           110       0.52      0.94      0.67        98
           111       0.92      0.92      0.92       523
           112       0.75      0.96      0.84        98
           113       0.76      0.93      0.84       136
           114       0.78      0.88      0.83        73
           115       0.79      0.66      0.72        94
           116       0.89      0.95      0.91        91
           117       0.89      0.90      0.90       147
           118       0.86      0.57      0.68        95
           119       0.88      0.92      0.90        48
           120       0.97      0.73      0.83        48

    accuracy                           0.81     14061
   macro avg       0.82      0.81      0.80     14061
weighted avg       0.82      0.81      0.81     14061
```

Visualizzazione errori su test interno Si visualizzano su quali immagini la CNN fa' una predizione errata:

[ ]:
```
# Trova gli indici degli errori
wrong_indices = np.where(y2_pred_classes != y2_test_classes)[0]

# Seleziona fino a 16 immagini casuali dagli errori
num_samples = min(16, len(wrong_indices))  # Evita errori se ci sono meno di 16␣
 ↪errori
random_wrong_indices = np.random.choice(wrong_indices, num_samples,␣
 ↪replace=False)

# Mostra le immagini errate
fig, axes = plt.subplots(4, 4, figsize=(10, 10))

for i, ax in enumerate(axes.flat):
    if i < num_samples:
        idx = random_wrong_indices[i]

        #  Controllo formato e normalizzazione
        img = np.clip(x_test[idx], 0, 1) # Assicura che plt.imshow() funzioni␣
 ↪bene
```

```
        ax.imshow(img)
        ax.set_title(f"True: {y2_test_classes[idx]}, Pred:␣
 ↪{y2_pred_classes[idx]}")
    ax.axis("off")

plt.show()
```

```
440/440              39s 89ms/step
```



Valutazione Test esterno

[ ]:
```python
# Percorso della cartella con le immagini
cartella_immagini = "Ext_Test"
```

[ ]:
```python
# Funzione per ridimensionare l'immagine mantenendo i canali RGB iniziali
def resize_image_keep_aspect_ratio(img, target_size=(100, 100)):
    h, w = img.shape[:2]
    scale = min(target_size[0] / w, target_size[1] / h)
    new_w, new_h = int(w * scale), int(h * scale)
    resized = cv2.resize(img, (new_w, new_h), interpolation=cv2.INTER_AREA)

    # Crea un'immagine di sfondo bianca 100x100
    output = np.ones((target_size[1], target_size[0], 3), dtype=np.uint8) * 255

    # Calcola il punto di inserimento per centrare l'immagine
    x_offset = (target_size[0] - new_w) // 2
    y_offset = (target_size[1] - new_h) // 2
    output[y_offset:y_offset + new_h, x_offset:x_offset + new_w] = resized

    # Converti da BGR a RGB
    output_rgb = cv2.cvtColor(output, cv2.COLOR_BGR2RGB)

    return output_rgb
```

[ ]:
```python
# Itera su tutte le immagini nella cartella
for nome_file in os.listdir(cartella_immagini):
    percorso_file = os.path.join(cartella_immagini, nome_file)

    # Controlla che sia un file immagine
    if not (nome_file.lower().endswith((".jpg", ".png", ".jpeg"))):
        continue

    # Carica l'immagine
    img = cv2.imread(percorso_file)
    if img is None:
        print(f"Errore nel caricamento di {nome_file}")
        continue

    img_resized = resize_image_keep_aspect_ratio(img)

    # Mostra l'immagine
    plt.imshow(img_resized / 255)  # Normalizzazione per la visualizzazione
    plt.title(f"Immagine: {nome_file}")
    plt.axis("off")
    plt.show()

    # Converti in array numpy e normalizza
```

```
    img_array = np.expand_dims(img_resized, axis=0).astype("float32") / 255.0 ␣
↪# Normalizzazione 0-1

    # Predizione con il modello custom
    prediction = model.predict(img_array)

    # Se il modello ha due output (frutta e qualità)
    predicted_fruit_idx = np.argmax(prediction[0])  # Output per la categoria␣
↪frutta
    predicted_quality_idx = np.argmax(prediction[1])  # Output per la qualità

    # Recupera i nomi dai dizionari/liste delle etichette
    predicted_fruit = labels_1_array[predicted_fruit_idx]
    predicted_quality = labels_2_array[predicted_quality_idx]

    # Stampa il risultato
    print(f"Immagine: {nome_file}, Predizione: {predicted_fruit} -␣
↪{predicted_quality}")
```


```
1/1             0s 18ms/step
Immagine: Cocos.jpg, Predizione: Peach - Peach Flat
```

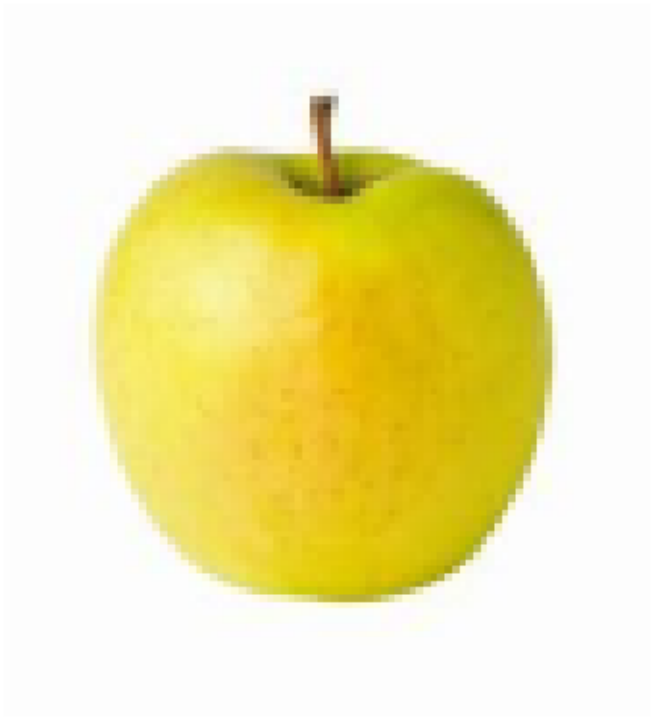
Immagine: Chestnut.JPG

```
1/1             0s 18ms/step
Immagine: Chestnut.JPG, Predizione: Apple - Apple Red Yellow
```


```
1/1             0s 19ms/step
Immagine: Lime.jpg, Predizione: Grape - Grape White
```

Immagine: Onion White.jpg

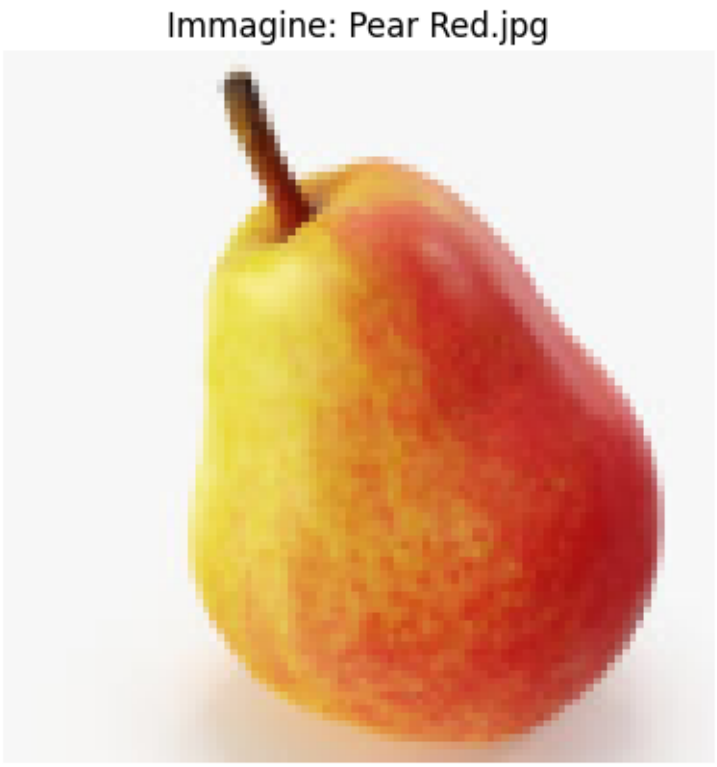
```
1/1             0s 19ms/step
Immagine: Onion White.jpg, Predizione: Physalis - Grapefruit Pink
```




```
1/1             0s 18ms/step
Immagine: Apple Golden.jpg, Predizione: Maracuja - Maracuja Undefined
```

Immagine: Onion White Peeled.jpg

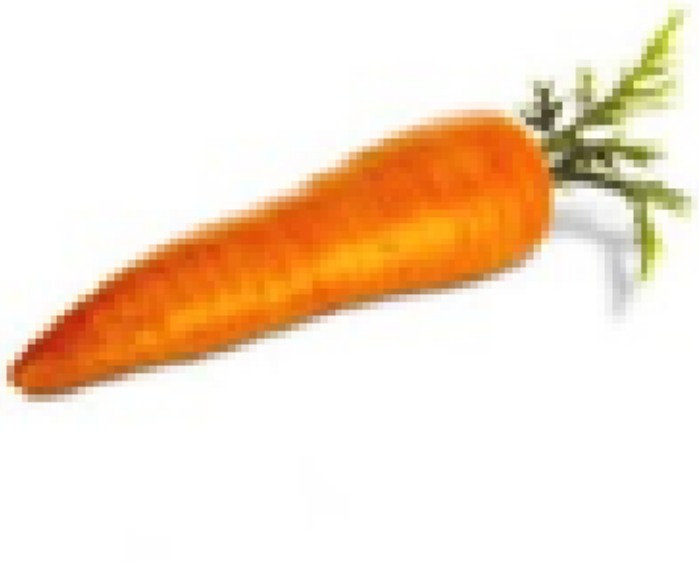
```
1/1             0s 18ms/step
Immagine: Onion White Peeled.jpg, Predizione: Physalis - Physalis Undefined
```




```
1/1             0s 18ms/step
Immagine: Pear Red.jpg, Predizione: Cherry - Pear Red
```


```
1/1             0s 18ms/step
Immagine: apple-braeburn.jpg, Predizione: Apple - Pepper Orange
```


```
1/1             0s 18ms/step
Immagine: Fig.jpg, Predizione: Apple - Apple Red Yellow
```




```
1/1             0s 17ms/step
Immagine: Carrot.jpg, Predizione: Pepper - Pepper Yellow
```

Immagine: Apple Golden con sfondo.jpeg

```
1/1             0s 18ms/step
Immagine: Apple Golden con sfondo.jpeg, Predizione: Apple - Apple Red Yellow
```


```
1/1             0s 18ms/step
Immagine: Grape White.jpg, Predizione: Maracuja - Maracuja Undefined
```


```
1/1             0s 21ms/step
Immagine: pomodoro.jpg, Predizione: Pear - Pear Red
```

Immagine: Apple Golden 2.jpeg

```
1/1             0s 18ms/step
Immagine: Apple Golden 2.jpeg, Predizione: Corn - Corn Undefined
```

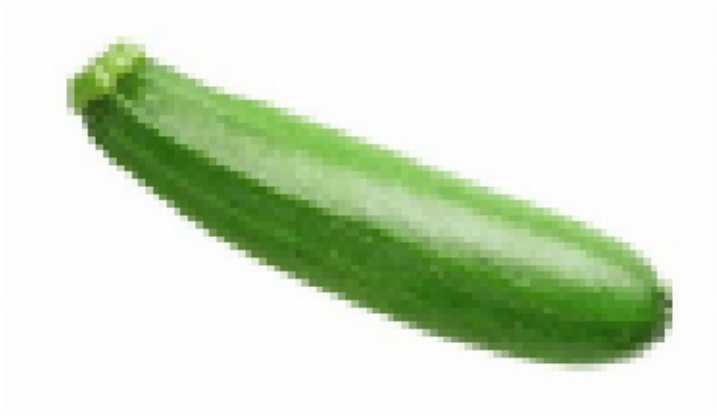

```
1/1             0s 19ms/step
Immagine: Lychee.jpg, Predizione: Redcurrant - Cherry Rainier
```


```
1/1             0s 18ms/step
Immagine: Cucumber.png, Predizione: Corn - Corn Husk
```

Immagine: Dragon Fruit.jpg

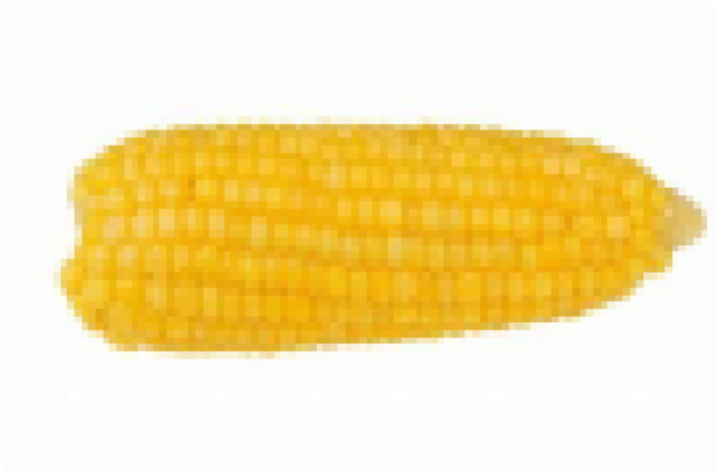
```
1/1             0s 18ms/step
Immagine: Dragon Fruit.jpg, Predizione: Apple - Apple Red Yellow
```


```
1/1             0s 18ms/step
Immagine: Lemon.jpg, Predizione: Maracuja - Maracuja Undefined
```




```
1/1             0s 44ms/step
Immagine: zucchina.jpg, Predizione: Corn - Zucchini Undefined
```


```
1/1             0s 18ms/step
Immagine: Guava.jpg, Predizione: Apple - Apple Golden
```




```
1/1             0s 18ms/step
Immagine: mais.jpg, Predizione: Corn - Corn Undefined
```


```
1/1             0s 19ms/step
Immagine: kiwi.jpg, Predizione: Potato - Potato Sweet
```

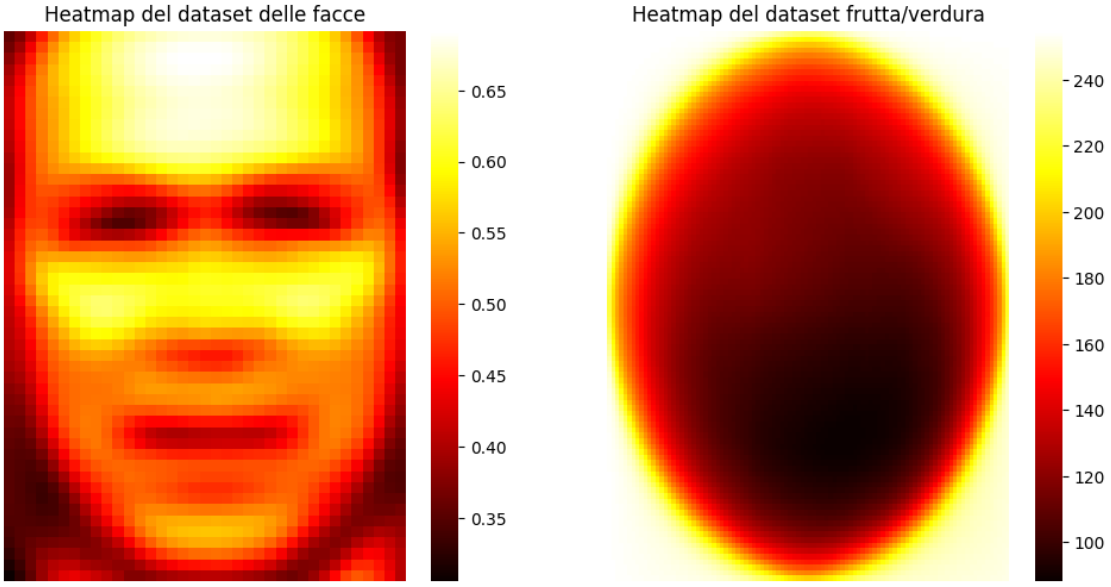
1.5.2 Modello 1: Siamese netework con tripletts loss

Motivazioni scelta del modello Si sceglie di realizzare una erte siamese con triplett loss perchè si pensa che il nostro problema di classificazione possa essere in qualche modo riconducilbile all'analisi delle feature facciali (che sarebbe il caso d'uso principale di questo tipo di rete).

[42]:
```
from sklearn.datasets import fetch_lfw_people

# Scarica il dataset (resize=0.4 rende le immagini più leggere)
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
lfw_images = lfw_people.images  # Immagini  # Array di immagini
```

[ ]:
```
# Calcola l'immagine media per entrambi i dataset
mean_face = np.mean(lfw_images, axis=0)  # Media facce
mean_custom = np.mean(x_train, axis=(0, -1))  # Media dataset personale

# Plotta le due heatmap affiancate
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
```

```
sns.heatmap(mean_face, cmap="hot", xticklabels=False, yticklabels=False,␣
 ↪ax=axes[0])
axes[0].set_title("Heatmap del dataset delle facce")

sns.heatmap(mean_custom, cmap="hot", xticklabels=False, yticklabels=False,␣
 ↪ax=axes[1])
axes[1].set_title("Heatmap del dataset frutta/verdura")

plt.show()
```



Andando a recuperare un dataset di facce è possbile confermare delle heatmap mostrate un paio di similitudini: - In entrambi i dataset si può ricostruire una forma "standard" e regolare che ogni elemento del dataset può assunmere. (dove per le facce è appunto la forma di una mashera facciale e per i frutti/verdura è un cerchio dato che molti frutti sono di questa dimensione) - In entrambi i dataset sono preseni grosse aree ad alta varianza che aiutano a distinguere l'immagine (nel dataset facciale sono principalmente bocca ed occhi; mentre nel dataset dei frutti corrisponde con la zona inferiore del frutto/verdura)

Essendo quindi i due problemi facilmente riconducibili l'uno con l'altro si pensa che l'utilizzo di questa rete possa essere uno strumento efficace per l'apprendimento di questo problema di classificazione.

Definizione steps per modello siamese è possibile creare una rete siamese con due output che gestisca sia la label1 che la label2. La struttura della rete può essere progettata in questo modo:

1. **Un backbone condiviso**: una CNN equivalente al modello-0 per estrarre le feature dalle immagini.
2. **Doppio ramo di embedding**: una testa di rete per produrre embedding separati per la

label1 e la label2.

3. **Loss triplet separata per ciascun output**: una per la label1 e una per la label2.
4. **Input multiplo**: la rete riceverà cinque immagini come input.

La funzione di loss potrebbe essere la combinazione di due triplet loss, una per ciascun output:

$$\mathcal{L} = \mathcal{L}_{\text{triplet1}} + \mathcal{L}_{\text{triplet2}}$$

Dove:

- $\mathcal{L}_{\text{triplet1}}$ usa anchor, positivo e negativo per label1.
- $\mathcal{L}_{\text{triplet2}}$ usa anchor, positivo e negativo per label2.

[44]:
```
batch_size = 32
```

Preparazione dei dati per essere processati con triplett loss nota che in questa funzione si fa anche normalizzazione dei dati:

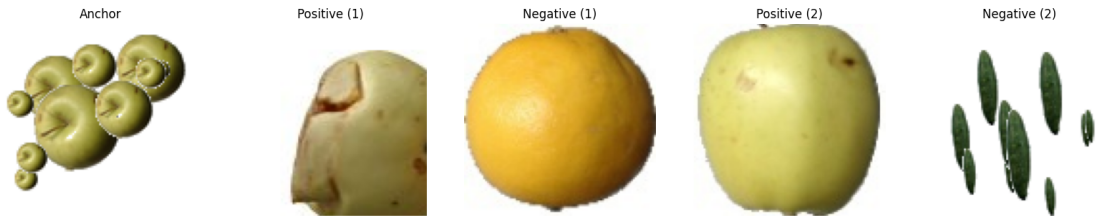
[45]:
```
def triplet_generator(x, y1, y2, batch_size=32):
    while True:
        anchor_batch, pos1_batch, neg1_batch, pos2_batch, neg2_batch, labels1,␣
↪labels2 = [], [], [], [], [],  [], []

        for _ in range(batch_size):
            idx_anchor = np.random.randint(0, len(x))
            anchor = x[idx_anchor] / 255

            label1_anchor = y1[idx_anchor]
            label2_anchor = y2[idx_anchor]
            labels1.append(label1_anchor)
            labels2.append(label2_anchor)

            # Trova un positivo e un negativo per label1
            pos1_idx = np.random.choice(np.where(np.argmax(y1, axis=1) == np.
↪argmax(label1_anchor))[0])
            neg1_idx = np.random.choice(np.where(np.argmax(y1, axis=1) != np.
↪argmax(label1_anchor))[0])

            pos1 = x[pos1_idx] / 255
            neg1 = x[neg1_idx] / 255

            # Trova un positivo e un negativo per label2
            pos2_idx = np.random.choice(np.where(np.argmax(y2, axis=1) == np.
↪argmax(label2_anchor))[0])
            neg2_idx = np.random.choice(np.where(np.argmax(y2, axis=1) != np.
↪argmax(label2_anchor))[0])

            pos2 = x[pos2_idx] / 255
```

```
            neg2 = x[neg2_idx] / 255

            anchor_batch.append(anchor)
            pos1_batch.append(pos1)
            neg1_batch.append(neg1)
            pos2_batch.append(pos2)
            neg2_batch.append(neg2)

        yield ((np.array(anchor_batch), np.array(pos1_batch), np.
 ↪array(neg1_batch),
                np.array(pos2_batch), np.array(neg2_batch)),  # 5 immagini␣
 ↪separate
               (np.array(labels1), np.array(labels2)))
```

[ ]:
```
# Creazione del dataset TensorFlow
train_dataset = tf.data.Dataset.from_generator(
    lambda: triplet_generator(x_train, y1_train, y2_train, batch_size),
    output_signature=(
        (tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32),  # Anchor
         tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32),  #␣
 ↪Positivo label1
         tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32),  #␣
 ↪Negativo label1
         tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32),  #␣
 ↪Positivo label2
         tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32)), #␣
 ↪Negativo label2
        (tf.TensorSpec(shape=((batch_size, 70)), dtype=tf.float32),  # Output 1
         tf.TensorSpec(shape=((batch_size, 121)), dtype=tf.float32))  # Output 2
    )
)

val_dataset = tf.data.Dataset.from_generator(
    lambda: triplet_generator(x_val, y1_val, y2_val, batch_size),
    output_signature=(
        (tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32),
         tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32),
         tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32),
         tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32),
         tf.TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32)),
        (tf.TensorSpec(shape=((batch_size, 70)), dtype=tf.float32),
         tf.TensorSpec(shape=((batch_size, 121)), dtype=tf.float32))
    )
)


for batch in train_dataset.take(1):
```

```
    (anchors, positives1, negatives1, positive2, negative2),(label1,label2) =␣
 ↪batch
    print("Anchor shape:", anchors.shape)
    print("Positives shape:", positives1.shape)
    print("Negatives shape:", negatives1.shape)


# Fetch one batch and display a sample
for batch in train_dataset.take(1):
    (anchors, positives1, negatives1, positive2, negative2),(label1,label2) =␣
 ↪batch

    # Convert one-hot encoded labels back to integer class values
    label1_int = np.argmax(label1, axis=1)
    label2_int = np.argmax(label2, axis=1)

    print(f"labels: {label_names_y1[label1_int[0]]} -␣
 ↪{label_names_y2[label2_int[0]]}")
    fig, axes = plt.subplots(1, 5, figsize=(20, 5))
    axes[0].imshow((anchors[0].numpy() * 255).astype("uint8"))
    axes[0].set_title("Anchor")
    axes[1].imshow((positives1[0].numpy() * 255).astype("uint8"))
    axes[1].set_title("Positive (1)")
    axes[2].imshow((negatives1[0].numpy() * 255).astype("uint8"))
    axes[2].set_title("Negative (1)")
    axes[3].imshow((positive2[0].numpy() * 255).astype("uint8"))
    axes[3].set_title("Positive (2)")
    axes[4].imshow((negative2[0].numpy() * 255).astype("uint8"))
    axes[4].set_title("Negative (2)")
    for ax in axes:
        ax.axis("off")
    plt.show()
    break
```

```
Anchor shape: (32, 100, 100, 3)
Positives shape: (32, 100, 100, 3)
Negatives shape: (32, 100, 100, 3)
labels: Apple - Apple Golden
```



Implementazione del modello: Backbone (rete embedded che processa le feature delle immagini in input):

```
[ ]: def build_custom_encoder():
         """
         Costruisce un encoder CNN per l'estrazione di feature.
         """
         inputs = Input(shape=(100, 100, 3))

         x = Conv2D(64, (9, 9), activation='relu')(inputs)
         x = MaxPooling2D((2, 2))(x)

         x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
         x = MaxPooling2D((2, 2))(x)

         x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
         x = MaxPooling2D((2, 2))(x)

         x = GlobalAveragePooling2D()(x)

         x = Dense(128, activation='relu')(x)
         x = Dropout(0.3)(x)
         x = Lambda(lambda x: tf.math.l2_normalize(x, axis=1),␣
      ↪output_shape=(1024,))(x)

         return Model(inputs, x, name="custom_encoder")
```

Modello siamese (modello wrapper che permette di prendere in input 5 immagini alla volta):

```
[ ]: def build_dual_output_siamese_model(encoder, output_shape=(3, 128)):
         """
         Costruisce una rete siamese con due output separati.
         """
         anchor_input = Input(shape=(100, 100, 3), name="anchor")
         positive1_input = Input(shape=(100, 100, 3), name="positive1")
         negative1_input = Input(shape=(100, 100, 3), name="negative1")
         positive2_input = Input(shape=(100, 100, 3), name="positive2")
         negative2_input = Input(shape=(100, 100, 3), name="negative2")

         # Estrazione feature con encoder condiviso
         anchor_emb = encoder(anchor_input)
         positive1_emb = encoder(positive1_input)
         negative1_emb = encoder(negative1_input)
         positive2_emb = encoder(positive2_input)
         negative2_emb = encoder(negative2_input)

         # Raggruppamento embedding per label1 e label2
```

```
    embeddings_label1 = Lambda(lambda x: tf.stack(x, axis=1),␣
↪output_shape=output_shape, name="triplett_label1")([anchor_emb,␣
↪positive1_emb, negative1_emb])
    embeddings_label2 = Lambda(lambda x: tf.stack(x, axis=1),␣
↪output_shape=output_shape, name="triplett_label2")([anchor_emb,␣
↪positive2_emb, negative2_emb])
    return Model(
        inputs=[anchor_input, positive1_input, negative1_input,␣
↪positive2_input, negative2_input],
        outputs=[embeddings_label1, embeddings_label2],
        name="dual_output_siamese_network"
    )
```

Definizione triplett loss custom che fa' la somma delle 2 triplett loss di label1 e label2:

[ ]:
```
from tensorflow.keras.saving import register_keras_serializable

def triplet_loss(margin=0.1):
    def loss(y_true, y_pred):
        anchor, positive, negative = y_pred[:, 0, :], y_pred[:, 1, :], y_pred[:
↪, 2, :]
        pos_dist = tf.reduce_sum(tf.square(anchor - positive), axis=1)
        neg_dist = tf.reduce_sum(tf.square(anchor - negative), axis=1)
        loss = tf.maximum(pos_dist - neg_dist + margin, 0.0)
        return tf.reduce_mean(loss)

    return loss
```

Train della rete:

[51]:
```
# Costruisci il modello
batch_size = 32
```

[52]:
```
# Costruisci l'encoder
encoder = build_custom_encoder()
encoder.summary()
```

Model: "custom_encoder"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 100, 100, 3)	0
conv2d (Conv2D)	(None, 92, 92, 64)	15,616
max_pooling2d (MaxPooling2D)	(None, 46, 46, 64)	0

conv2d_1 (Conv2D)	(None, 46, 46, 128)	73,856
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 128)	0
conv2d_2 (Conv2D)	(None, 23, 23, 256)	295,168
max_pooling2d_2 (MaxPooling2D)	(None, 11, 11, 256)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)	0
dense (Dense)	(None, 128)	32,896
dropout (Dropout)	(None, 128)	0
lambda (Lambda)	(None, 1024)	0

Total params: 417,536 (1.59 MB)

Trainable params: 417,536 (1.59 MB)

Non-trainable params: 0 (0.00 B)

[53]:
```
# Costruisci il modello siamese con doppio output
siamese_model = build_dual_output_siamese_model(encoder)
siamese_model.summary()
```

Model: "dual_output_siamese_network"

Layer (type)	Output Shape	Param #	Connected to
anchor (InputLayer)	(None, 100, 100, 3)	0	-
positive1 (InputLayer)	(None, 100, 100, 3)	0	-
negative1 (InputLayer)	(None, 100, 100, 3)	0	-
positive2 (InputLayer)	(None, 100, 100, 3)	0	-

```
negative2             (None, 100, 100,            0   -
(InputLayer)          3)

custom_encoder        (None, 1024)          417,536   anchor[0][0],
(Functional)                                          positive1[0][0],
                                                      negative1[0][0],
                                                      positive2[0][0],
                                                      negative2[0][0]

triplett_label1       (None, 3, 128)              0   custom_encoder[0…
(Lambda)                                              custom_encoder[1…
                                                      custom_encoder[2…

triplett_label2       (None, 3, 128)              0   custom_encoder[0…
(Lambda)                                              custom_encoder[3…
                                                      custom_encoder[4…
```

Total params: 417,536 (1.59 MB)

Trainable params: 417,536 (1.59 MB)

Non-trainable params: 0 (0.00 B)

[54]:
```
# Definisci due loss separate per label1 e label2
loss1 = triplet_loss(margin=0.1)
loss2 = triplet_loss(margin=0.1)

# Compila il modello con la somma delle due loss
siamese_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
                      loss={'triplett_label1': loss1, 'triplett_label2':␣
 ↪loss2})  # Associa la loss ai due output


dataset_size = len(x_train)  # Numero totale di campioni
epochs_number = 30

steps_per_epoch = int(1.5* dataset_size / batch_size / epochs_number)
steps_per_val = int(len(x_val) / batch_size / epochs_number)

print(f"Steps per epoch: {steps_per_epoch}")
print(f"Steps per val: {steps_per_val}")
```

```
Steps per epoch: 88
Steps per val: 24
```

Nota che per velocizzare il train ed evitare di overfitatre sui dati di train, si sceglie di impostare un valore di steps per epochs uguale a:

$$\frac{\text{150\% dell'intero dataset}}{\text{numero di epoche}}$$

In questo si ha che per ogni singola epoca si visiterà in totale solo una frazione del dataset; invece che avere che per ogni singola epoca far visitare tutto il set. Avendo quindi un totale di 20 epoche vuol dire che, su tutte le 20 epoche si visiteranno in totatle un quantitivo di immagini totali pari alla lunghezza di tutto tutto il dataset per 1.5. Questo perchè ogni batch viene estratto casualmente, quindi se avessimo visto solo la lunghezza di tutto il dataset si correva il rischio di non visualizzare tutte le immagini. Aggiungendo questa moltiplicazione per 1.5 questo rischio non viene azzerato ma viene alleviato di molto.

Questo porta ad avere che la loss potrebbe avere comportamenti fluttuanti (scendi / sali) quando si scoprono per la prima volta sample che non sono mai stati esplorati in epoche precedenti, ma avendo un totale di 20 epoche è altamente probabile che entro la fine del train tutti i dati siano stati visitati più di una volta. In questo modo si riescono a migliorare di moltissimo i tempi di train.

[ ]:
```
# Check if history file exists
if os.path.exists("siamese_history.npz"):
    encoder = load_model("encoder.keras")
    siamese_model = load_model("siamese_model.keras")
    data = np.load("hsiamese_history.npz")
    history = {key: data[key] for key in data.files}
    print("History loaded successfully.")
else:
    history = siamese_model.fit(
        train_dataset,
        epochs=epochs_number,
        validation_data=val_dataset,
        steps_per_epoch=steps_per_epoch,
        validation_steps=steps_per_val
    )
    encoder.save("encoder.keras")
    siamese_model.save("siamese_model.keras")
    np.savez("siamese_history.npz", **history.history)
```

```
Epoch 1/30
88/88             110s 1s/step -
loss: 0.2619 - triplett_label1_loss: 0.1317 - triplett_label2_loss: 0.1303 -
val_loss: 0.1453 - val_triplett_label1_loss: 0.0869 - val_triplett_label2_loss:
0.0584
Epoch 2/30
88/88             141s 2s/step -
loss: 0.1817 - triplett_label1_loss: 0.1049 - triplett_label2_loss: 0.0769 -
val_loss: 0.1442 - val_triplett_label1_loss: 0.0929 - val_triplett_label2_loss:
0.0513
Epoch 3/30
88/88             158s 2s/step -
```

```
loss: 0.1698 - triplett_label1_loss: 0.0903 - triplett_label2_loss: 0.0795 -
val_loss: 0.1548 - val_triplett_label1_loss: 0.1024 - val_triplett_label2_loss:
0.0524
Epoch 4/30
88/88            126s 1s/step -
loss: 0.1653 - triplett_label1_loss: 0.0942 - triplett_label2_loss: 0.0711 -
val_loss: 0.1217 - val_triplett_label1_loss: 0.0770 - val_triplett_label2_loss:
0.0447
Epoch 5/30
88/88            129s 1s/step -
loss: 0.1528 - triplett_label1_loss: 0.0907 - triplett_label2_loss: 0.0621 -
val_loss: 0.1297 - val_triplett_label1_loss: 0.0845 - val_triplett_label2_loss:
0.0453
Epoch 6/30
88/88            137s 2s/step -
loss: 0.1495 - triplett_label1_loss: 0.0867 - triplett_label2_loss: 0.0629 -
val_loss: 0.1333 - val_triplett_label1_loss: 0.0873 - val_triplett_label2_loss:
0.0460
Epoch 7/30
88/88            130s 1s/step -
loss: 0.1568 - triplett_label1_loss: 0.0957 - triplett_label2_loss: 0.0612 -
val_loss: 0.1042 - val_triplett_label1_loss: 0.0734 - val_triplett_label2_loss:
0.0308
Epoch 8/30
88/88            132s 2s/step -
loss: 0.1373 - triplett_label1_loss: 0.0865 - triplett_label2_loss: 0.0508 -
val_loss: 0.1022 - val_triplett_label1_loss: 0.0774 - val_triplett_label2_loss:
0.0248
Epoch 9/30
88/88            119s 1s/step -
loss: 0.1275 - triplett_label1_loss: 0.0833 - triplett_label2_loss: 0.0442 -
val_loss: 0.0884 - val_triplett_label1_loss: 0.0608 - val_triplett_label2_loss:
0.0276
Epoch 10/30
88/88            97s 1s/step - loss:
0.1157 - triplett_label1_loss: 0.0752 - triplett_label2_loss: 0.0405 - val_loss:
0.0930 - val_triplett_label1_loss: 0.0723 - val_triplett_label2_loss: 0.0208
Epoch 11/30
88/88            95s 1s/step - loss:
0.1094 - triplett_label1_loss: 0.0749 - triplett_label2_loss: 0.0345 - val_loss:
0.0777 - val_triplett_label1_loss: 0.0589 - val_triplett_label2_loss: 0.0188
Epoch 12/30
88/88            99s 1s/step - loss:
0.1058 - triplett_label1_loss: 0.0708 - triplett_label2_loss: 0.0350 - val_loss:
0.0768 - val_triplett_label1_loss: 0.0618 - val_triplett_label2_loss: 0.0150
Epoch 13/30
88/88            97s 1s/step - loss:
0.1077 - triplett_label1_loss: 0.0701 - triplett_label2_loss: 0.0376 - val_loss:
```

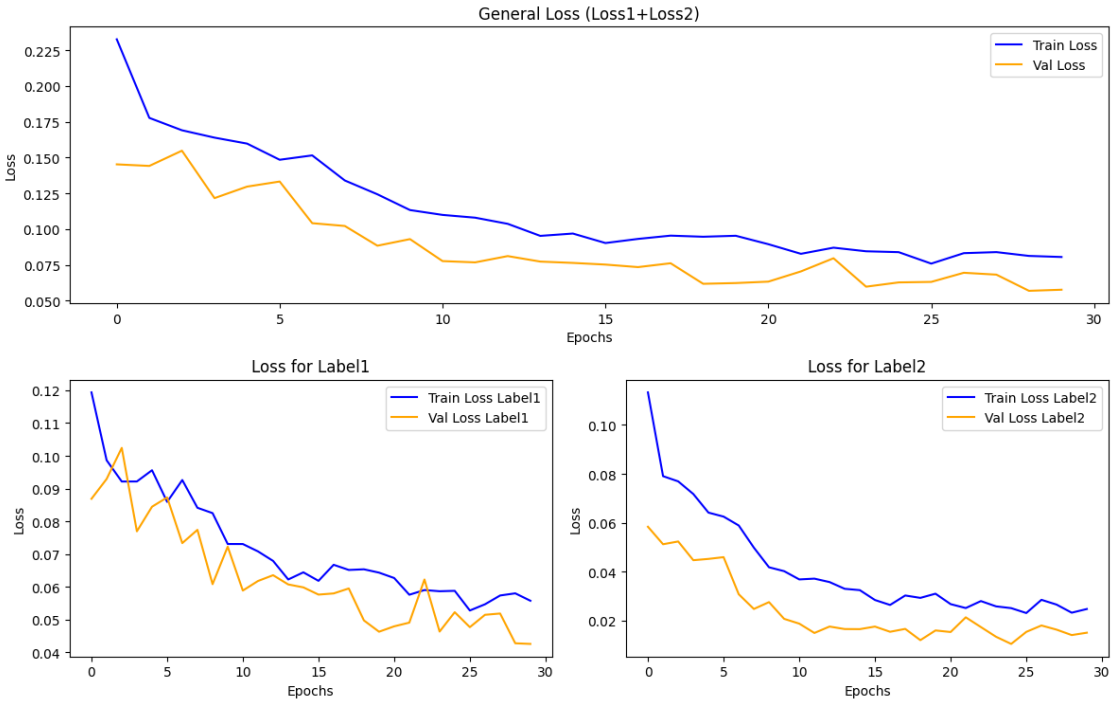
```
0.0812 - val_triplett_label1_loss: 0.0636 - val_triplett_label2_loss: 0.0176
Epoch 14/30
88/88            98s 1s/step - loss:
0.0937 - triplett_label1_loss: 0.0596 - triplett_label2_loss: 0.0340 - val_loss:
0.0773 - val_triplett_label1_loss: 0.0607 - val_triplett_label2_loss: 0.0166
Epoch 15/30
88/88            106s 1s/step -
loss: 0.0968 - triplett_label1_loss: 0.0639 - triplett_label2_loss: 0.0330 -
val_loss: 0.0764 - val_triplett_label1_loss: 0.0598 - val_triplett_label2_loss:
0.0166
Epoch 16/30
88/88            100s 1s/step -
loss: 0.0907 - triplett_label1_loss: 0.0624 - triplett_label2_loss: 0.0282 -
val_loss: 0.0752 - val_triplett_label1_loss: 0.0576 - val_triplett_label2_loss:
0.0176
Epoch 17/30
88/88            99s 1s/step - loss:
0.0964 - triplett_label1_loss: 0.0699 - triplett_label2_loss: 0.0264 - val_loss:
0.0735 - val_triplett_label1_loss: 0.0580 - val_triplett_label2_loss: 0.0155
Epoch 18/30
88/88            118s 1s/step -
loss: 0.0943 - triplett_label1_loss: 0.0652 - triplett_label2_loss: 0.0291 -
val_loss: 0.0762 - val_triplett_label1_loss: 0.0595 - val_triplett_label2_loss:
0.0167
Epoch 19/30
88/88            117s 1s/step -
loss: 0.0935 - triplett_label1_loss: 0.0643 - triplett_label2_loss: 0.0292 -
val_loss: 0.0618 - val_triplett_label1_loss: 0.0497 - val_triplett_label2_loss:
0.0121
Epoch 20/30
88/88            118s 1s/step -
loss: 0.0978 - triplett_label1_loss: 0.0651 - triplett_label2_loss: 0.0327 -
val_loss: 0.0623 - val_triplett_label1_loss: 0.0463 - val_triplett_label2_loss:
0.0160
Epoch 21/30
88/88            121s 1s/step -
loss: 0.0909 - triplett_label1_loss: 0.0635 - triplett_label2_loss: 0.0274 -
val_loss: 0.0633 - val_triplett_label1_loss: 0.0479 - val_triplett_label2_loss:
0.0154
Epoch 22/30
88/88            120s 1s/step -
loss: 0.0761 - triplett_label1_loss: 0.0505 - triplett_label2_loss: 0.0256 -
val_loss: 0.0705 - val_triplett_label1_loss: 0.0491 - val_triplett_label2_loss:
0.0214
Epoch 23/30
88/88            119s 1s/step -
loss: 0.0901 - triplett_label1_loss: 0.0595 - triplett_label2_loss: 0.0306 -
val_loss: 0.0797 - val_triplett_label1_loss: 0.0622 - val_triplett_label2_loss:
```

```
0.0174
Epoch 24/30
88/88             117s 1s/step -
loss: 0.0817 - triplett_label1_loss: 0.0560 - triplett_label2_loss: 0.0257 -
val_loss: 0.0598 - val_triplett_label1_loss: 0.0463 - val_triplett_label2_loss:
0.0134
Epoch 25/30
88/88             119s 1s/step -
loss: 0.0817 - triplett_label1_loss: 0.0570 - triplett_label2_loss: 0.0247 -
val_loss: 0.0628 - val_triplett_label1_loss: 0.0523 - val_triplett_label2_loss:
0.0105
Epoch 26/30
88/88             106s 1s/step -
loss: 0.0761 - triplett_label1_loss: 0.0527 - triplett_label2_loss: 0.0234 -
val_loss: 0.0631 - val_triplett_label1_loss: 0.0477 - val_triplett_label2_loss:
0.0154
Epoch 27/30
88/88             114s 1s/step -
loss: 0.0857 - triplett_label1_loss: 0.0573 - triplett_label2_loss: 0.0284 -
val_loss: 0.0695 - val_triplett_label1_loss: 0.0515 - val_triplett_label2_loss:
0.0181
Epoch 28/30
88/88             115s 1s/step -
loss: 0.0858 - triplett_label1_loss: 0.0605 - triplett_label2_loss: 0.0253 -
val_loss: 0.0682 - val_triplett_label1_loss: 0.0519 - val_triplett_label2_loss:
0.0163
Epoch 29/30
88/88             115s 1s/step -
loss: 0.0792 - triplett_label1_loss: 0.0547 - triplett_label2_loss: 0.0244 -
val_loss: 0.0569 - val_triplett_label1_loss: 0.0428 - val_triplett_label2_loss:
0.0141
Epoch 30/30
88/88             111s 1s/step -
loss: 0.0786 - triplett_label1_loss: 0.0554 - triplett_label2_loss: 0.0232 -
val_loss: 0.0577 - val_triplett_label1_loss: 0.0426 - val_triplett_label2_loss:
0.0151
```

[ ]:
```python
def plot_dual_siamese_history(history):
    """
    Plots the training history of the dual-output Siamese model.
    """
    fig = plt.figure(figsize=(12, 8))  # Overall figure size

    # General Loss (first row, spans 2 columns)
    ax1 = plt.subplot2grid((2, 2), (0, 0), colspan=2)  # Full-width first row
    ax1.plot(history["loss"], label="Train Loss", color="blue")
    ax1.plot(history["val_loss"], label="Val Loss", color="orange")
```

```
    ax1.set_title("General Loss (Loss1+Loss2)")
    ax1.set_xlabel("Epochs")
    ax1.set_ylabel("Loss")
    ax1.legend()

    # Loss for Label1 (second row, left)
    ax2 = plt.subplot2grid((2, 2), (1, 0))
    ax2.plot(history["triplett_label1_loss"], label="Train Loss Label1",␣
↪color="blue")
    ax2.plot(history["val_triplett_label1_loss"], label="Val Loss Label1",␣
↪color="orange")
    ax2.set_title("Loss for Label1")
    ax2.set_xlabel("Epochs")
    ax2.set_ylabel("Loss")
    ax2.legend()

    # Loss for Label2 (second row, right)
    ax3 = plt.subplot2grid((2, 2), (1, 1))
    ax3.plot(history["triplett_label2_loss"], label="Train Loss Label2",␣
↪color="blue")
    ax3.plot(history["val_triplett_label2_loss"], label="Val Loss Label2",␣
↪color="orange")
    ax3.set_title("Loss for Label2")
    ax3.set_xlabel("Epochs")
    ax3.set_ylabel("Loss")
    ax3.legend()

    plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout to fit title
    plt.show()
```

[ ]:
```
plot_dual_siamese_history(history.history)
```



Overall l'andamento della loss non è neacnhe così tanto malvagio. Seppure siano presenti evidenti flunctuation è comunque possibile notare le tendenze negative dei tutte le curve. Una possbile soluzione per migliorara l'andamento di questa curva sarebbe cambiare il learning rate ma, come potremo nontare in seguito, anche cambiando il learning rate non si otterranno grandissimi risultati.

Predizioni sul test set Una volta che il modello è addestrato, se ne estraggono le feature e si prova a trainare 2 SVM (uno per output). In questo modo si fa' model compression (utilizzando il metodo della **knowledge distillation**) e si riesce per ogni immagine di input ad assegnare una vera e propria label, in modo da poter avere 2 modelli che presi in input le feature estratte e la label riesca a predirre a sua volta una label di predizione.

Il processo sarebbe il seguente: 1. **Estrazione delle feature**: Usa la rete siamese per generare gli embedding delle immagini nel set di training. 2. **Allenamento degli SVM**: Addestra due SVM separati (uno per label1 e uno per label2) usando gli embedding come input e le etichette corrispondenti come target. 3. **Valutazione**: Calcola l'accuracy degli SVM sul set di validazione per determinare la qualità delle feature. 4. **Predizione**: Una volta addestrati, puoi passare una nuova immagine attraverso la rete per ottenere l'embedding e poi classificarla con gli SVM.

Questo approccio ti permette di avere una metrica di accuratezza ben definita e di verificare se gli embedding appresi sono discriminativi.

1. Estrazione delle feature:

[ ]:
```
# Estrai le feature passando le immagini nel modello siamese
feature_extractor = encoder
```

```
train_features = feature_extractor.predict(x_train)
val_features = feature_extractor.predict(x_val)
test_features = feature_extractor.predict(x_test)
```

```
1764/1764              140s 79ms/step
870/870              70s 81ms/step
440/440              35s 80ms/step
```

2. Allenamento degli SVM

[60]:
```
# Crea pipeline con normalizzazione + SVM
svm1 = Pipeline([('scaler', StandardScaler()), ('svm', SVC(kernel='linear',␣
 ↪probability=True))])
svm2 = Pipeline([('scaler', StandardScaler()), ('svm', SVC(kernel='linear',␣
 ↪probability=True))])

# Converte one-hot in etichette scalari
y1_train = np.argmax(y1_train, axis=1)
y2_train = np.argmax(y2_train, axis=1)

# Allena gli SVM
svm1.fit(train_features, y1_train)
svm2.fit(train_features, y2_train)
```

[60]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('svm', SVC(kernel='linear', probability=True))])
```

3. Predizione sui dati di test

[ ]:
```
# Predici le classi sulle feature di validazione
y1_pred = svm1.predict(val_features)
y2_pred = svm2.predict(val_features)

# converti da onehot a normali
y1_val = np.argmax(y1_val, axis=1)
y2_val = np.argmax(y2_val, axis=1)

# Calcola accuracy
acc1 = accuracy_score(y1_val, y1_pred)
acc2 = accuracy_score(y2_val, y2_pred)

print(f"Accuracy per Label 1: {acc1 * 100:.2f}%")
print(f"Accuracy per Label 2: {acc2 * 100:.2f}%")

f1_1 = f1_score(y1_val, y1_pred, average='macro')
f1_2 = f1_score(y2_val, y2_pred, average='macro')

print(f"F1-score per Label 1: {f1_1*100:.2f}%")
print(f"F1-score per Label 2: {f1_2*100:.2f}%")
```

```
Accuracy per Label 1: 85.37%
Accuracy per Label 2: 86.48%
F1-score per Label 1: 85.62%
F1-score per Label 2: 85.68%
```

Valutazione Test esterno

[62]:
```python
def classify_image(image_path, model, svm1, svm2):
    from tensorflow.keras.preprocessing.image import load_img, img_to_array

    # Preprocessa l'immagine
    img = load_img(image_path, target_size=(224, 224))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)  # Aggiunge batch dimension

    # Estrai le feature con il modello siamese
    features = model.predict(img_array)
    # Classifica con gli SVM
    label1_pred = svm1.predict(features)[0]
    label2_pred = svm2.predict(features)[0]

    return label1_pred, label2_pred
```

[ ]:
```python
# Directory containing the images
ext_test_dir = "Ext_Test"

# Iterate over all images in the directory
for image_file in os.listdir(ext_test_dir):
    image_path = os.path.join(ext_test_dir, image_file)
    print("------------------------------------------------------------")
    label1, label2 = classify_image(image_path, feature_extractor, svm1, svm2)
    print(f"Image: {image_file}")
    print(f"Predicted Label 1: {labels_1_array[label1]}, Predicted Label 2:␣
 ↪{labels_2_array[label2]}")
```

```
------------------------------------------------------------
1/1             0s 120ms/step
Image: Apple Golden 2.jpeg
Predicted Label 1: Banana, Predicted Label 2: Banana Lady Finger
------------------------------------------------------------
1/1             0s 47ms/step
Image: Apple Golden con sfondo.jpeg
Predicted Label 1: Pear, Predicted Label 2: Onion Red
------------------------------------------------------------
1/1             0s 51ms/step
Image: Apple Golden.jpg
Predicted Label 1: Apple, Predicted Label 2: Apple Undefined
------------------------------------------------------------
1/1             0s 40ms/step
```

```
Image: apple-braeburn.jpg
Predicted Label 1: Walnut, Predicted Label 2: Potato Red
----------------------------------------------------------
1/1             0s 44ms/step
Image: Carrot.jpg
Predicted Label 1: Cucumber, Predicted Label 2: Cucumber Ripe
----------------------------------------------------------
1/1             0s 53ms/step
Image: Chestnut.JPG
Predicted Label 1: Cocos, Predicted Label 2: Potato Sweet
----------------------------------------------------------
1/1             0s 46ms/step
Image: Cocos.jpg
Predicted Label 1: Cocos, Predicted Label 2: Cocos Undefined
----------------------------------------------------------
1/1             0s 52ms/step
Image: Cucumber.png
Predicted Label 1: Tomato, Predicted Label 2: Tomato not Ripened
----------------------------------------------------------
1/1             0s 43ms/step
Image: Dragon Fruit.jpg
Predicted Label 1: Strawberry, Predicted Label 2: Pear Undefined
----------------------------------------------------------
1/1             0s 40ms/step
Image: Fig.jpg
Predicted Label 1: Watermelon, Predicted Label 2: Watermelon Undefined
----------------------------------------------------------
1/1             0s 50ms/step
Image: Grape White.jpg
Predicted Label 1: Grape, Predicted Label 2: Grape White
----------------------------------------------------------
1/1             0s 45ms/step
Image: Guava.jpg
Predicted Label 1: Tomato, Predicted Label 2: Watermelon Undefined
----------------------------------------------------------
1/1             0s 40ms/step
Image: kiwi.jpg
Predicted Label 1: Potato, Predicted Label 2: Potato Sweet
----------------------------------------------------------
1/1             0s 48ms/step
Image: Lemon.jpg
Predicted Label 1: Maracuja, Predicted Label 2: Maracuja Undefined
----------------------------------------------------------
1/1             0s 39ms/step
Image: Lime.jpg
Predicted Label 1: Corn, Predicted Label 2: Corn Husk
----------------------------------------------------------
1/1             0s 43ms/step
```

```
Image: Lychee.jpg
Predicted Label 1: Strawberry, Predicted Label 2: Strawberry Wedge
----------------------------------------------------------
1/1             0s 45ms/step
Image: mais.jpg
Predicted Label 1: Guava, Predicted Label 2: Guava Undefined
----------------------------------------------------------
1/1             0s 33ms/step
Image: Onion White Peeled.jpg
Predicted Label 1: Potato, Predicted Label 2: Potato Red
----------------------------------------------------------
1/1             0s 48ms/step
Image: Onion White.jpg
Predicted Label 1: Potato, Predicted Label 2: Potato Sweet
----------------------------------------------------------
1/1             0s 50ms/step
Image: Pear Red.jpg
Predicted Label 1: Strawberry, Predicted Label 2: Pear Undefined
----------------------------------------------------------
1/1             0s 40ms/step
Image: pomodoro.jpg
Predicted Label 1: Strawberry, Predicted Label 2: Grape Pink
----------------------------------------------------------
1/1             0s 42ms/step
Image: zucchina.jpg
Predicted Label 1: Corn, Predicted Label 2: Corn Husk
```

1.5.3 Modello 2: ResNet101

Reload dateset

```
[ ]: x_train, y1_train, y2_train, x_val, y1_val, y2_val, x_test, y1_test, y2_test =␣
      ↪reaload_from_scratch()
```

Normalizzazione dati:

```
[ ]: x_train_preprocessed = tf.keras.applications.resnet.preprocess_input(x_train)
     x_val_preprocessed = tf.keras.applications.resnet.preprocess_input(x_val)
     x_test_preprocessed = tf.keras.applications.resnet.preprocess_input(x_test)
```

Calcolo steps per epochs

```
[ ]: dataset_size = len(x_train)  # Numero totale di campioni
     epochs_number = 100
     batch_size = 32

     steps_per_epoch = int(1.5* dataset_size / batch_size / epochs_number)
     steps_per_val = int(len(x_val) / batch_size / epochs_number)

     print(f"Steps per epoch: {steps_per_epoch}")
```

```
print(f"Steps per val: {steps_per_val}")
```

```
Steps per epoch: 26
Steps per val: 7
```

Implementazione ResNet101

[ ]:
```
input_shape = x_train.shape[1:]
print(input_shape)
num_classes_1 = y1_train.shape[1]
print(num_classes_1)
num_classes_2 = y2_train.shape[1]
print(num_classes_2)
```

```
(100, 100, 3)
70
121
```

[ ]:
```
# Carica ResNet101 senza la testa di classificazione
base_model = ResNet101(weights='imagenet', include_top=False,␣
 ↪input_shape=input_shape)
# Sblocca la ResNet101
base_model.trainable = False

# Appiattimento dell'output della ResNet101
x = GlobalAveragePooling2D()(base_model.output)

x = Dense(256, activation="relu")(x)  # Riduci la dimensionalità
x = Dropout(0.3)(x)  # Evita overfitting
x = Dense(128, activation="relu")(x)  # Ulteriore compressione

# Ramo 1 - Predizione del Frutto
fruit_output = Dense(num_classes_1, activation="softmax", name="y1")(x)

# Ramo 2 - Predizione della Qualità
quality_output = Dense(num_classes_2, activation="softmax", name="y2")(x)

# Modello finale
model = Model(inputs=base_model.input, outputs=[fruit_output, quality_output])

early_stopping = EarlyStopping(
    monitor="val_loss",  # Monitora la perdita sulla validation set
    patience=10,         # Numero di epoche senza miglioramenti prima di␣
 ↪fermarsi
    min_delta=1e-5,      # Delta minimo per considerare un miglioramento
    mode="min",          # Si ferma quando 'val_loss' smette di diminuire
    restore_best_weights=True  # Ripristina i pesi della migliore epoca
)
```

```
[ ]: model.summary()
```

```
Model: "functional_12"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_13 (InputLayer)	(None, 100, 100, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 106, 106, 3)	0	input_layer_13[0…
conv1_conv (Conv2D)	(None, 50, 50, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalizatio…	(None, 50, 50, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 50, 50, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 52, 52, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 25, 25, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 25, 25, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalizatio…	(None, 25, 25, 64)	256	conv2_block1_1_c…
conv2_block1_1_relu (Activation)	(None, 25, 25, 64)	0	conv2_block1_1_b…
conv2_block1_2_conv (Conv2D)	(None, 25, 25, 64)	36,928	conv2_block1_1_r…
conv2_block1_2_bn (BatchNormalizatio…	(None, 25, 25, 64)	256	conv2_block1_2_c…
conv2_block1_2_relu (Activation)	(None, 25, 25, 64)	0	conv2_block1_2_b…
conv2_block1_0_conv	(None, 25, 25,	16,640	pool1_pool[0][0]

```
(Conv2D)              256)

conv2_block1_3_conv   (None, 25, 25,          16,640   conv2_block1_2_r…
(Conv2D)              256)

conv2_block1_0_bn     (None, 25, 25,           1,024   conv2_block1_0_c…
(BatchNormalizatio…   256)

conv2_block1_3_bn     (None, 25, 25,           1,024   conv2_block1_3_c…
(BatchNormalizatio…   256)

conv2_block1_add      (None, 25, 25,               0   conv2_block1_0_b…
(Add)                 256)                             conv2_block1_3_b…

conv2_block1_out      (None, 25, 25,               0   conv2_block1_add…
(Activation)          256)

conv2_block2_1_conv   (None, 25, 25,          16,448   conv2_block1_out…
(Conv2D)              64)

conv2_block2_1_bn     (None, 25, 25,             256   conv2_block2_1_c…
(BatchNormalizatio…   64)

conv2_block2_1_relu   (None, 25, 25,               0   conv2_block2_1_b…
(Activation)          64)

conv2_block2_2_conv   (None, 25, 25,          36,928   conv2_block2_1_r…
(Conv2D)              64)

conv2_block2_2_bn     (None, 25, 25,             256   conv2_block2_2_c…
(BatchNormalizatio…   64)

conv2_block2_2_relu   (None, 25, 25,               0   conv2_block2_2_b…
(Activation)          64)

conv2_block2_3_conv   (None, 25, 25,          16,640   conv2_block2_2_r…
(Conv2D)              256)

conv2_block2_3_bn     (None, 25, 25,           1,024   conv2_block2_3_c…
(BatchNormalizatio…   256)

conv2_block2_add      (None, 25, 25,               0   conv2_block1_out…
(Add)                 256)                             conv2_block2_3_b…

conv2_block2_out      (None, 25, 25,               0   conv2_block2_add…
(Activation)          256)

conv2_block3_1_conv   (None, 25, 25,          16,448   conv2_block2_out…
```

```
(Conv2D)                 64)

conv2_block3_1_bn        (None, 25, 25,             256   conv2_block3_1_c…
(BatchNormalizatio…      64)

conv2_block3_1_relu      (None, 25, 25,               0   conv2_block3_1_b…
(Activation)             64)

conv2_block3_2_conv      (None, 25, 25,          36,928   conv2_block3_1_r…
(Conv2D)                 64)

conv2_block3_2_bn        (None, 25, 25,             256   conv2_block3_2_c…
(BatchNormalizatio…      64)

conv2_block3_2_relu      (None, 25, 25,               0   conv2_block3_2_b…
(Activation)             64)

conv2_block3_3_conv      (None, 25, 25,          16,640   conv2_block3_2_r…
(Conv2D)                 256)

conv2_block3_3_bn        (None, 25, 25,           1,024   conv2_block3_3_c…
(BatchNormalizatio…      256)

conv2_block3_add         (None, 25, 25,               0   conv2_block2_out…
(Add)                    256)                             conv2_block3_3_b…

conv2_block3_out         (None, 25, 25,               0   conv2_block3_add…
(Activation)             256)

conv3_block1_1_conv      (None, 13, 13,          32,896   conv2_block3_out…
(Conv2D)                 128)

conv3_block1_1_bn        (None, 13, 13,             512   conv3_block1_1_c…
(BatchNormalizatio…      128)

conv3_block1_1_relu      (None, 13, 13,               0   conv3_block1_1_b…
(Activation)             128)

conv3_block1_2_conv      (None, 13, 13,         147,584   conv3_block1_1_r…
(Conv2D)                 128)

conv3_block1_2_bn        (None, 13, 13,             512   conv3_block1_2_c…
(BatchNormalizatio…      128)

conv3_block1_2_relu      (None, 13, 13,               0   conv3_block1_2_b…
(Activation)             128)

conv3_block1_0_conv      (None, 13, 13,         131,584   conv2_block3_out…
```

```
(Conv2D)                 512)

conv3_block1_3_conv      (None, 13, 13,          66,048   conv3_block1_2_r…
(Conv2D)                 512)

conv3_block1_0_bn        (None, 13, 13,           2,048   conv3_block1_0_c…
(BatchNormalizatio…      512)

conv3_block1_3_bn        (None, 13, 13,           2,048   conv3_block1_3_c…
(BatchNormalizatio…      512)

conv3_block1_add         (None, 13, 13,               0   conv3_block1_0_b…
(Add)                    512)                             conv3_block1_3_b…

conv3_block1_out         (None, 13, 13,               0   conv3_block1_add…
(Activation)             512)

conv3_block2_1_conv      (None, 13, 13,          65,664   conv3_block1_out…
(Conv2D)                 128)

conv3_block2_1_bn        (None, 13, 13,             512   conv3_block2_1_c…
(BatchNormalizatio…      128)

conv3_block2_1_relu      (None, 13, 13,               0   conv3_block2_1_b…
(Activation)             128)

conv3_block2_2_conv      (None, 13, 13,         147,584   conv3_block2_1_r…
(Conv2D)                 128)

conv3_block2_2_bn        (None, 13, 13,             512   conv3_block2_2_c…
(BatchNormalizatio…      128)

conv3_block2_2_relu      (None, 13, 13,               0   conv3_block2_2_b…
(Activation)             128)

conv3_block2_3_conv      (None, 13, 13,          66,048   conv3_block2_2_r…
(Conv2D)                 512)

conv3_block2_3_bn        (None, 13, 13,           2,048   conv3_block2_3_c…
(BatchNormalizatio…      512)

conv3_block2_add         (None, 13, 13,               0   conv3_block1_out…
(Add)                    512)                             conv3_block2_3_b…

conv3_block2_out         (None, 13, 13,               0   conv3_block2_add…
(Activation)             512)

conv3_block3_1_conv      (None, 13, 13,          65,664   conv3_block2_out…
```

```
(Conv2D)                 128)

conv3_block3_1_bn        (None, 13, 13,             512   conv3_block3_1_c…
(BatchNormalizatio…      128)

conv3_block3_1_relu      (None, 13, 13,               0   conv3_block3_1_b…
(Activation)             128)

conv3_block3_2_conv      (None, 13, 13,         147,584   conv3_block3_1_r…
(Conv2D)                 128)

conv3_block3_2_bn        (None, 13, 13,             512   conv3_block3_2_c…
(BatchNormalizatio…      128)

conv3_block3_2_relu      (None, 13, 13,               0   conv3_block3_2_b…
(Activation)             128)

conv3_block3_3_conv      (None, 13, 13,          66,048   conv3_block3_2_r…
(Conv2D)                 512)

conv3_block3_3_bn        (None, 13, 13,           2,048   conv3_block3_3_c…
(BatchNormalizatio…      512)

conv3_block3_add         (None, 13, 13,               0   conv3_block2_out…
(Add)                    512)                             conv3_block3_3_b…

conv3_block3_out         (None, 13, 13,               0   conv3_block3_add…
(Activation)             512)

conv3_block4_1_conv      (None, 13, 13,          65,664   conv3_block3_out…
(Conv2D)                 128)

conv3_block4_1_bn        (None, 13, 13,             512   conv3_block4_1_c…
(BatchNormalizatio…      128)

conv3_block4_1_relu      (None, 13, 13,               0   conv3_block4_1_b…
(Activation)             128)

conv3_block4_2_conv      (None, 13, 13,         147,584   conv3_block4_1_r…
(Conv2D)                 128)

conv3_block4_2_bn        (None, 13, 13,             512   conv3_block4_2_c…
(BatchNormalizatio…      128)

conv3_block4_2_relu      (None, 13, 13,               0   conv3_block4_2_b…
(Activation)             128)

conv3_block4_3_conv      (None, 13, 13,          66,048   conv3_block4_2_r…
```

```
(Conv2D)                 512)

conv3_block4_3_bn        (None, 13, 13,           2,048   conv3_block4_3_c…
(BatchNormalizatio…      512)

conv3_block4_add         (None, 13, 13,               0   conv3_block3_out…
(Add)                    512)                             conv3_block4_3_b…

conv3_block4_out         (None, 13, 13,               0   conv3_block4_add…
(Activation)             512)

conv4_block1_1_conv      (None, 7, 7, 256)      131,328   conv3_block4_out…
(Conv2D)

conv4_block1_1_bn        (None, 7, 7, 256)        1,024   conv4_block1_1_c…
(BatchNormalizatio…

conv4_block1_1_relu      (None, 7, 7, 256)            0   conv4_block1_1_b…
(Activation)

conv4_block1_2_conv      (None, 7, 7, 256)      590,080   conv4_block1_1_r…
(Conv2D)

conv4_block1_2_bn        (None, 7, 7, 256)        1,024   conv4_block1_2_c…
(BatchNormalizatio…

conv4_block1_2_relu      (None, 7, 7, 256)            0   conv4_block1_2_b…
(Activation)

conv4_block1_0_conv      (None, 7, 7,           525,312   conv3_block4_out…
(Conv2D)                 1024)

conv4_block1_3_conv      (None, 7, 7,           263,168   conv4_block1_2_r…
(Conv2D)                 1024)

conv4_block1_0_bn        (None, 7, 7,             4,096   conv4_block1_0_c…
(BatchNormalizatio…      1024)

conv4_block1_3_bn        (None, 7, 7,             4,096   conv4_block1_3_c…
(BatchNormalizatio…      1024)

conv4_block1_add         (None, 7, 7,                 0   conv4_block1_0_b…
(Add)                    1024)                            conv4_block1_3_b…

conv4_block1_out         (None, 7, 7,                 0   conv4_block1_add…
(Activation)             1024)

conv4_block2_1_conv      (None, 7, 7, 256)      262,400   conv4_block1_out…
```

```
(Conv2D)

conv4_block2_1_bn     (None, 7, 7, 256)        1,024   conv4_block2_1_c…
(BatchNormalizatio…

conv4_block2_1_relu   (None, 7, 7, 256)            0   conv4_block2_1_b…
(Activation)

conv4_block2_2_conv   (None, 7, 7, 256)      590,080   conv4_block2_1_r…
(Conv2D)

conv4_block2_2_bn     (None, 7, 7, 256)        1,024   conv4_block2_2_c…
(BatchNormalizatio…

conv4_block2_2_relu   (None, 7, 7, 256)            0   conv4_block2_2_b…
(Activation)

conv4_block2_3_conv   (None, 7, 7,           263,168   conv4_block2_2_r…
(Conv2D)              1024)

conv4_block2_3_bn     (None, 7, 7,             4,096   conv4_block2_3_c…
(BatchNormalizatio…   1024)

conv4_block2_add      (None, 7, 7,                 0   conv4_block1_out…
(Add)                 1024)                            conv4_block2_3_b…

conv4_block2_out      (None, 7, 7,                 0   conv4_block2_add…
(Activation)          1024)

conv4_block3_1_conv   (None, 7, 7, 256)      262,400   conv4_block2_out…
(Conv2D)

conv4_block3_1_bn     (None, 7, 7, 256)        1,024   conv4_block3_1_c…
(BatchNormalizatio…

conv4_block3_1_relu   (None, 7, 7, 256)            0   conv4_block3_1_b…
(Activation)

conv4_block3_2_conv   (None, 7, 7, 256)      590,080   conv4_block3_1_r…
(Conv2D)

conv4_block3_2_bn     (None, 7, 7, 256)        1,024   conv4_block3_2_c…
(BatchNormalizatio…

conv4_block3_2_relu   (None, 7, 7, 256)            0   conv4_block3_2_b…
(Activation)

conv4_block3_3_conv   (None, 7, 7,           263,168   conv4_block3_2_r…
```

```
(Conv2D)                1024)

conv4_block3_3_bn       (None, 7, 7,              4,096   conv4_block3_3_c…
(BatchNormalizatio…     1024)

conv4_block3_add        (None, 7, 7,                  0   conv4_block2_out…
(Add)                   1024)                             conv4_block3_3_b…

conv4_block3_out        (None, 7, 7,                  0   conv4_block3_add…
(Activation)            1024)

conv4_block4_1_conv     (None, 7, 7, 256)       262,400   conv4_block3_out…
(Conv2D)

conv4_block4_1_bn       (None, 7, 7, 256)         1,024   conv4_block4_1_c…
(BatchNormalizatio…

conv4_block4_1_relu     (None, 7, 7, 256)             0   conv4_block4_1_b…
(Activation)

conv4_block4_2_conv     (None, 7, 7, 256)       590,080   conv4_block4_1_r…
(Conv2D)

conv4_block4_2_bn       (None, 7, 7, 256)         1,024   conv4_block4_2_c…
(BatchNormalizatio…

conv4_block4_2_relu     (None, 7, 7, 256)             0   conv4_block4_2_b…
(Activation)

conv4_block4_3_conv     (None, 7, 7,              263,168   conv4_block4_2_r…
(Conv2D)                1024)

conv4_block4_3_bn       (None, 7, 7,              4,096   conv4_block4_3_c…
(BatchNormalizatio…     1024)

conv4_block4_add        (None, 7, 7,                  0   conv4_block3_out…
(Add)                   1024)                             conv4_block4_3_b…

conv4_block4_out        (None, 7, 7,                  0   conv4_block4_add…
(Activation)            1024)

conv4_block5_1_conv     (None, 7, 7, 256)       262,400   conv4_block4_out…
(Conv2D)

conv4_block5_1_bn       (None, 7, 7, 256)         1,024   conv4_block5_1_c…
(BatchNormalizatio…

conv4_block5_1_relu     (None, 7, 7, 256)             0   conv4_block5_1_b…
```

```
(Activation)

conv4_block5_2_conv   (None, 7, 7, 256)      590,080   conv4_block5_1_r…
(Conv2D)

conv4_block5_2_bn     (None, 7, 7, 256)        1,024   conv4_block5_2_c…
(BatchNormalizatio…

conv4_block5_2_relu   (None, 7, 7, 256)            0   conv4_block5_2_b…
(Activation)

conv4_block5_3_conv   (None, 7, 7,           263,168   conv4_block5_2_r…
(Conv2D)              1024)

conv4_block5_3_bn     (None, 7, 7,             4,096   conv4_block5_3_c…
(BatchNormalizatio…   1024)

conv4_block5_add      (None, 7, 7,                 0   conv4_block4_out…
(Add)                 1024)                            conv4_block5_3_b…

conv4_block5_out      (None, 7, 7,                 0   conv4_block5_add…
(Activation)          1024)

conv4_block6_1_conv   (None, 7, 7, 256)      262,400   conv4_block5_out…
(Conv2D)

conv4_block6_1_bn     (None, 7, 7, 256)        1,024   conv4_block6_1_c…
(BatchNormalizatio…

conv4_block6_1_relu   (None, 7, 7, 256)            0   conv4_block6_1_b…
(Activation)

conv4_block6_2_conv   (None, 7, 7, 256)      590,080   conv4_block6_1_r…
(Conv2D)

conv4_block6_2_bn     (None, 7, 7, 256)        1,024   conv4_block6_2_c…
(BatchNormalizatio…

conv4_block6_2_relu   (None, 7, 7, 256)            0   conv4_block6_2_b…
(Activation)

conv4_block6_3_conv   (None, 7, 7,           263,168   conv4_block6_2_r…
(Conv2D)              1024)

conv4_block6_3_bn     (None, 7, 7,             4,096   conv4_block6_3_c…
(BatchNormalizatio…   1024)

conv4_block6_add      (None, 7, 7,                 0   conv4_block5_out…
```

```
(Add)                1024)                                  conv4_block6_3_b…

conv4_block6_out     (None, 7, 7,                 0         conv4_block6_add…
(Activation)         1024)

conv4_block7_1_conv  (None, 7, 7, 256)      262,400         conv4_block6_out…
(Conv2D)

conv4_block7_1_bn    (None, 7, 7, 256)        1,024         conv4_block7_1_c…
(BatchNormalizatio…

conv4_block7_1_relu  (None, 7, 7, 256)            0         conv4_block7_1_b…
(Activation)

conv4_block7_2_conv  (None, 7, 7, 256)      590,080         conv4_block7_1_r…
(Conv2D)

conv4_block7_2_bn    (None, 7, 7, 256)        1,024         conv4_block7_2_c…
(BatchNormalizatio…

conv4_block7_2_relu  (None, 7, 7, 256)            0         conv4_block7_2_b…
(Activation)

conv4_block7_3_conv  (None, 7, 7,           263,168         conv4_block7_2_r…
(Conv2D)             1024)

conv4_block7_3_bn    (None, 7, 7,             4,096         conv4_block7_3_c…
(BatchNormalizatio…  1024)

conv4_block7_add     (None, 7, 7,                 0         conv4_block6_out…
(Add)                1024)                                  conv4_block7_3_b…

conv4_block7_out     (None, 7, 7,                 0         conv4_block7_add…
(Activation)         1024)

conv4_block8_1_conv  (None, 7, 7, 256)      262,400         conv4_block7_out…
(Conv2D)

conv4_block8_1_bn    (None, 7, 7, 256)        1,024         conv4_block8_1_c…
(BatchNormalizatio…

conv4_block8_1_relu  (None, 7, 7, 256)            0         conv4_block8_1_b…
(Activation)

conv4_block8_2_conv  (None, 7, 7, 256)      590,080         conv4_block8_1_r…
(Conv2D)

conv4_block8_2_bn    (None, 7, 7, 256)        1,024         conv4_block8_2_c…
```

```
(BatchNormalizatio…

conv4_block8_2_relu   (None, 7, 7, 256)           0   conv4_block8_2_b…
(Activation)

conv4_block8_3_conv   (None, 7, 7,           263,168   conv4_block8_2_r…
(Conv2D)              1024)

conv4_block8_3_bn     (None, 7, 7,             4,096   conv4_block8_3_c…
(BatchNormalizatio…   1024)

conv4_block8_add      (None, 7, 7,                 0   conv4_block7_out…
(Add)                 1024)                            conv4_block8_3_b…

conv4_block8_out      (None, 7, 7,                 0   conv4_block8_add…
(Activation)          1024)

conv4_block9_1_conv   (None, 7, 7, 256)       262,400   conv4_block8_out…
(Conv2D)

conv4_block9_1_bn     (None, 7, 7, 256)         1,024   conv4_block9_1_c…
(BatchNormalizatio…

conv4_block9_1_relu   (None, 7, 7, 256)           0   conv4_block9_1_b…
(Activation)

conv4_block9_2_conv   (None, 7, 7, 256)       590,080   conv4_block9_1_r…
(Conv2D)

conv4_block9_2_bn     (None, 7, 7, 256)         1,024   conv4_block9_2_c…
(BatchNormalizatio…

conv4_block9_2_relu   (None, 7, 7, 256)           0   conv4_block9_2_b…
(Activation)

conv4_block9_3_conv   (None, 7, 7,           263,168   conv4_block9_2_r…
(Conv2D)              1024)

conv4_block9_3_bn     (None, 7, 7,             4,096   conv4_block9_3_c…
(BatchNormalizatio…   1024)

conv4_block9_add      (None, 7, 7,                 0   conv4_block8_out…
(Add)                 1024)                            conv4_block9_3_b…

conv4_block9_out      (None, 7, 7,                 0   conv4_block9_add…
(Activation)          1024)

conv4_block10_1_co…   (None, 7, 7, 256)       262,400   conv4_block9_out…
```

```
(Conv2D)

conv4_block10_1_bn    (None, 7, 7, 256)        1,024   conv4_block10_1_…
(BatchNormalizatio…

conv4_block10_1_re…   (None, 7, 7, 256)            0   conv4_block10_1_…
(Activation)

conv4_block10_2_co…   (None, 7, 7, 256)      590,080   conv4_block10_1_…
(Conv2D)

conv4_block10_2_bn    (None, 7, 7, 256)        1,024   conv4_block10_2_…
(BatchNormalizatio…

conv4_block10_2_re…   (None, 7, 7, 256)            0   conv4_block10_2_…
(Activation)

conv4_block10_3_co…   (None, 7, 7,           263,168   conv4_block10_2_…
(Conv2D)              1024)

conv4_block10_3_bn    (None, 7, 7,             4,096   conv4_block10_3_…
(BatchNormalizatio…   1024)

conv4_block10_add     (None, 7, 7,                 0   conv4_block9_out…
(Add)                 1024)                            conv4_block10_3_…

conv4_block10_out     (None, 7, 7,                 0   conv4_block10_ad…
(Activation)          1024)

conv4_block11_1_co…   (None, 7, 7, 256)      262,400   conv4_block10_ou…
(Conv2D)

conv4_block11_1_bn    (None, 7, 7, 256)        1,024   conv4_block11_1_…
(BatchNormalizatio…

conv4_block11_1_re…   (None, 7, 7, 256)            0   conv4_block11_1_…
(Activation)

conv4_block11_2_co…   (None, 7, 7, 256)      590,080   conv4_block11_1_…
(Conv2D)

conv4_block11_2_bn    (None, 7, 7, 256)        1,024   conv4_block11_2_…
(BatchNormalizatio…

conv4_block11_2_re…   (None, 7, 7, 256)            0   conv4_block11_2_…
(Activation)

conv4_block11_3_co…   (None, 7, 7,           263,168   conv4_block11_2_…
```

```
(Conv2D)                1024)

conv4_block11_3_bn      (None, 7, 7,              4,096   conv4_block11_3_…
(BatchNormalizatio…     1024)

conv4_block11_add       (None, 7, 7,                  0   conv4_block10_ou…
(Add)                   1024)                             conv4_block11_3_…

conv4_block11_out       (None, 7, 7,                  0   conv4_block11_ad…
(Activation)            1024)

conv4_block12_1_co…     (None, 7, 7, 256)       262,400   conv4_block11_ou…
(Conv2D)

conv4_block12_1_bn      (None, 7, 7, 256)         1,024   conv4_block12_1_…
(BatchNormalizatio…

conv4_block12_1_re…     (None, 7, 7, 256)             0   conv4_block12_1_…
(Activation)

conv4_block12_2_co…     (None, 7, 7, 256)       590,080   conv4_block12_1_…
(Conv2D)

conv4_block12_2_bn      (None, 7, 7, 256)         1,024   conv4_block12_2_…
(BatchNormalizatio…

conv4_block12_2_re…     (None, 7, 7, 256)             0   conv4_block12_2_…
(Activation)

conv4_block12_3_co…     (None, 7, 7,            263,168   conv4_block12_2_…
(Conv2D)                1024)

conv4_block12_3_bn      (None, 7, 7,              4,096   conv4_block12_3_…
(BatchNormalizatio…     1024)

conv4_block12_add       (None, 7, 7,                  0   conv4_block11_ou…
(Add)                   1024)                             conv4_block12_3_…

conv4_block12_out       (None, 7, 7,                  0   conv4_block12_ad…
(Activation)            1024)

conv4_block13_1_co…     (None, 7, 7, 256)       262,400   conv4_block12_ou…
(Conv2D)

conv4_block13_1_bn      (None, 7, 7, 256)         1,024   conv4_block13_1_…
(BatchNormalizatio…

conv4_block13_1_re…     (None, 7, 7, 256)             0   conv4_block13_1_…
```

```
(Activation)

conv4_block13_2_co…  (None, 7, 7, 256)     590,080   conv4_block13_1_…
(Conv2D)

conv4_block13_2_bn   (None, 7, 7, 256)       1,024   conv4_block13_2_…
(BatchNormalizatio…

conv4_block13_2_re…  (None, 7, 7, 256)           0   conv4_block13_2_…
(Activation)

conv4_block13_3_co…  (None, 7, 7,          263,168   conv4_block13_2_…
(Conv2D)             1024)

conv4_block13_3_bn   (None, 7, 7,            4,096   conv4_block13_3_…
(BatchNormalizatio…  1024)

conv4_block13_add    (None, 7, 7,                0   conv4_block12_ou…
(Add)                1024)                          conv4_block13_3_…

conv4_block13_out    (None, 7, 7,                0   conv4_block13_ad…
(Activation)         1024)

conv4_block14_1_co…  (None, 7, 7, 256)     262,400   conv4_block13_ou…
(Conv2D)

conv4_block14_1_bn   (None, 7, 7, 256)       1,024   conv4_block14_1_…
(BatchNormalizatio…

conv4_block14_1_re…  (None, 7, 7, 256)           0   conv4_block14_1_…
(Activation)

conv4_block14_2_co…  (None, 7, 7, 256)     590,080   conv4_block14_1_…
(Conv2D)

conv4_block14_2_bn   (None, 7, 7, 256)       1,024   conv4_block14_2_…
(BatchNormalizatio…

conv4_block14_2_re…  (None, 7, 7, 256)           0   conv4_block14_2_…
(Activation)

conv4_block14_3_co…  (None, 7, 7,          263,168   conv4_block14_2_…
(Conv2D)             1024)

conv4_block14_3_bn   (None, 7, 7,            4,096   conv4_block14_3_…
(BatchNormalizatio…  1024)

conv4_block14_add    (None, 7, 7,                0   conv4_block13_ou…
```

(Add)	1024)		conv4_block14_3_…
conv4_block14_out (Activation)	(None, 7, 7, 1024)	0	conv4_block14_ad…
conv4_block15_1_co… (Conv2D)	(None, 7, 7, 256)	262,400	conv4_block14_ou…
conv4_block15_1_bn (BatchNormalizatio…	(None, 7, 7, 256)	1,024	conv4_block15_1_…
conv4_block15_1_re… (Activation)	(None, 7, 7, 256)	0	conv4_block15_1_…
conv4_block15_2_co… (Conv2D)	(None, 7, 7, 256)	590,080	conv4_block15_1_…
conv4_block15_2_bn (BatchNormalizatio…	(None, 7, 7, 256)	1,024	conv4_block15_2_…
conv4_block15_2_re… (Activation)	(None, 7, 7, 256)	0	conv4_block15_2_…
conv4_block15_3_co… (Conv2D)	(None, 7, 7, 1024)	263,168	conv4_block15_2_…
conv4_block15_3_bn (BatchNormalizatio…	(None, 7, 7, 1024)	4,096	conv4_block15_3_…
conv4_block15_add (Add)	(None, 7, 7, 1024)	0	conv4_block14_ou… conv4_block15_3_…
conv4_block15_out (Activation)	(None, 7, 7, 1024)	0	conv4_block15_ad…
conv4_block16_1_co… (Conv2D)	(None, 7, 7, 256)	262,400	conv4_block15_ou…
conv4_block16_1_bn (BatchNormalizatio…	(None, 7, 7, 256)	1,024	conv4_block16_1_…
conv4_block16_1_re… (Activation)	(None, 7, 7, 256)	0	conv4_block16_1_…
conv4_block16_2_co… (Conv2D)	(None, 7, 7, 256)	590,080	conv4_block16_1_…
conv4_block16_2_bn	(None, 7, 7, 256)	1,024	conv4_block16_2_…

```
(BatchNormalizatio…

conv4_block16_2_re… (None, 7, 7, 256)           0   conv4_block16_2_…
(Activation)

conv4_block16_3_co… (None, 7, 7,           263,168   conv4_block16_2_…
(Conv2D)            1024)

conv4_block16_3_bn  (None, 7, 7,             4,096   conv4_block16_3_…
(BatchNormalizatio… 1024)

conv4_block16_add   (None, 7, 7,                 0   conv4_block15_ou…
(Add)               1024)                            conv4_block16_3_…

conv4_block16_out   (None, 7, 7,                 0   conv4_block16_ad…
(Activation)        1024)

conv4_block17_1_co… (None, 7, 7, 256)       262,400   conv4_block16_ou…
(Conv2D)

conv4_block17_1_bn  (None, 7, 7, 256)         1,024   conv4_block17_1_…
(BatchNormalizatio…

conv4_block17_1_re… (None, 7, 7, 256)           0   conv4_block17_1_…
(Activation)

conv4_block17_2_co… (None, 7, 7, 256)       590,080   conv4_block17_1_…
(Conv2D)

conv4_block17_2_bn  (None, 7, 7, 256)         1,024   conv4_block17_2_…
(BatchNormalizatio…

conv4_block17_2_re… (None, 7, 7, 256)           0   conv4_block17_2_…
(Activation)

conv4_block17_3_co… (None, 7, 7,           263,168   conv4_block17_2_…
(Conv2D)            1024)

conv4_block17_3_bn  (None, 7, 7,             4,096   conv4_block17_3_…
(BatchNormalizatio… 1024)

conv4_block17_add   (None, 7, 7,                 0   conv4_block16_ou…
(Add)               1024)                            conv4_block17_3_…

conv4_block17_out   (None, 7, 7,                 0   conv4_block17_ad…
(Activation)        1024)

conv4_block18_1_co… (None, 7, 7, 256)       262,400   conv4_block17_ou…
```

```
(Conv2D)

conv4_block18_1_bn    (None, 7, 7, 256)        1,024   conv4_block18_1_…
(BatchNormalizatio…

conv4_block18_1_re…   (None, 7, 7, 256)            0   conv4_block18_1_…
(Activation)

conv4_block18_2_co…   (None, 7, 7, 256)      590,080   conv4_block18_1_…
(Conv2D)

conv4_block18_2_bn    (None, 7, 7, 256)        1,024   conv4_block18_2_…
(BatchNormalizatio…

conv4_block18_2_re…   (None, 7, 7, 256)            0   conv4_block18_2_…
(Activation)

conv4_block18_3_co…   (None, 7, 7,           263,168   conv4_block18_2_…
(Conv2D)              1024)

conv4_block18_3_bn    (None, 7, 7,             4,096   conv4_block18_3_…
(BatchNormalizatio…   1024)

conv4_block18_add     (None, 7, 7,                 0   conv4_block17_ou…
(Add)                 1024)                            conv4_block18_3_…

conv4_block18_out     (None, 7, 7,                 0   conv4_block18_ad…
(Activation)          1024)

conv4_block19_1_co…   (None, 7, 7, 256)      262,400   conv4_block18_ou…
(Conv2D)

conv4_block19_1_bn    (None, 7, 7, 256)        1,024   conv4_block19_1_…
(BatchNormalizatio…

conv4_block19_1_re…   (None, 7, 7, 256)            0   conv4_block19_1_…
(Activation)

conv4_block19_2_co…   (None, 7, 7, 256)      590,080   conv4_block19_1_…
(Conv2D)

conv4_block19_2_bn    (None, 7, 7, 256)        1,024   conv4_block19_2_…
(BatchNormalizatio…

conv4_block19_2_re…   (None, 7, 7, 256)            0   conv4_block19_2_…
(Activation)

conv4_block19_3_co…   (None, 7, 7,           263,168   conv4_block19_2_…
```

```
(Conv2D)                1024)

conv4_block19_3_bn      (None, 7, 7,              4,096   conv4_block19_3_…
(BatchNormalizatio…     1024)

conv4_block19_add       (None, 7, 7,                  0   conv4_block18_ou…
(Add)                   1024)                             conv4_block19_3_…

conv4_block19_out       (None, 7, 7,                  0   conv4_block19_ad…
(Activation)            1024)

conv4_block20_1_co…     (None, 7, 7, 256)       262,400   conv4_block19_ou…
(Conv2D)

conv4_block20_1_bn      (None, 7, 7, 256)         1,024   conv4_block20_1_…
(BatchNormalizatio…

conv4_block20_1_re…     (None, 7, 7, 256)             0   conv4_block20_1_…
(Activation)

conv4_block20_2_co…     (None, 7, 7, 256)       590,080   conv4_block20_1_…
(Conv2D)

conv4_block20_2_bn      (None, 7, 7, 256)         1,024   conv4_block20_2_…
(BatchNormalizatio…

conv4_block20_2_re…     (None, 7, 7, 256)             0   conv4_block20_2_…
(Activation)

conv4_block20_3_co…     (None, 7, 7,              263,168   conv4_block20_2_…
(Conv2D)                1024)

conv4_block20_3_bn      (None, 7, 7,              4,096   conv4_block20_3_…
(BatchNormalizatio…     1024)

conv4_block20_add       (None, 7, 7,                  0   conv4_block19_ou…
(Add)                   1024)                             conv4_block20_3_…

conv4_block20_out       (None, 7, 7,                  0   conv4_block20_ad…
(Activation)            1024)

conv4_block21_1_co…     (None, 7, 7, 256)       262,400   conv4_block20_ou…
(Conv2D)

conv4_block21_1_bn      (None, 7, 7, 256)         1,024   conv4_block21_1_…
(BatchNormalizatio…

conv4_block21_1_re…     (None, 7, 7, 256)             0   conv4_block21_1_…
```

```
(Activation)

conv4_block21_2_co…  (None, 7, 7, 256)     590,080   conv4_block21_1_…
(Conv2D)

conv4_block21_2_bn   (None, 7, 7, 256)       1,024   conv4_block21_2_…
(BatchNormalizatio…

conv4_block21_2_re…  (None, 7, 7, 256)           0   conv4_block21_2_…
(Activation)

conv4_block21_3_co…  (None, 7, 7,          263,168   conv4_block21_2_…
(Conv2D)             1024)

conv4_block21_3_bn   (None, 7, 7,            4,096   conv4_block21_3_…
(BatchNormalizatio…  1024)

conv4_block21_add    (None, 7, 7,                0   conv4_block20_ou…
(Add)                1024)                          conv4_block21_3_…

conv4_block21_out    (None, 7, 7,                0   conv4_block21_ad…
(Activation)         1024)

conv4_block22_1_co…  (None, 7, 7, 256)     262,400   conv4_block21_ou…
(Conv2D)

conv4_block22_1_bn   (None, 7, 7, 256)       1,024   conv4_block22_1_…
(BatchNormalizatio…

conv4_block22_1_re…  (None, 7, 7, 256)           0   conv4_block22_1_…
(Activation)

conv4_block22_2_co…  (None, 7, 7, 256)     590,080   conv4_block22_1_…
(Conv2D)

conv4_block22_2_bn   (None, 7, 7, 256)       1,024   conv4_block22_2_…
(BatchNormalizatio…

conv4_block22_2_re…  (None, 7, 7, 256)           0   conv4_block22_2_…
(Activation)

conv4_block22_3_co…  (None, 7, 7,          263,168   conv4_block22_2_…
(Conv2D)             1024)

conv4_block22_3_bn   (None, 7, 7,            4,096   conv4_block22_3_…
(BatchNormalizatio…  1024)

conv4_block22_add    (None, 7, 7,                0   conv4_block21_ou…
```

```
(Add)                  1024)                                  conv4_block22_3_…

conv4_block22_out      (None, 7, 7,                 0         conv4_block22_ad…
(Activation)           1024)

conv4_block23_1_co…    (None, 7, 7, 256)      262,400         conv4_block22_ou…
(Conv2D)

conv4_block23_1_bn     (None, 7, 7, 256)        1,024         conv4_block23_1_…
(BatchNormalizatio…

conv4_block23_1_re…    (None, 7, 7, 256)            0         conv4_block23_1_…
(Activation)

conv4_block23_2_co…    (None, 7, 7, 256)      590,080         conv4_block23_1_…
(Conv2D)

conv4_block23_2_bn     (None, 7, 7, 256)        1,024         conv4_block23_2_…
(BatchNormalizatio…

conv4_block23_2_re…    (None, 7, 7, 256)            0         conv4_block23_2_…
(Activation)

conv4_block23_3_co…    (None, 7, 7,           263,168         conv4_block23_2_…
(Conv2D)               1024)

conv4_block23_3_bn     (None, 7, 7,             4,096         conv4_block23_3_…
(BatchNormalizatio…    1024)

conv4_block23_add      (None, 7, 7,                 0         conv4_block22_ou…
(Add)                  1024)                                  conv4_block23_3_…

conv4_block23_out      (None, 7, 7,                 0         conv4_block23_ad…
(Activation)           1024)

conv5_block1_1_conv    (None, 4, 4, 512)      524,800         conv4_block23_ou…
(Conv2D)

conv5_block1_1_bn      (None, 4, 4, 512)        2,048         conv5_block1_1_c…
(BatchNormalizatio…

conv5_block1_1_relu    (None, 4, 4, 512)            0         conv5_block1_1_b…
(Activation)

conv5_block1_2_conv    (None, 4, 4, 512)    2,359,808         conv5_block1_1_r…
(Conv2D)

conv5_block1_2_bn      (None, 4, 4, 512)        2,048         conv5_block1_2_c…
```

```
(BatchNormalizatio…

conv5_block1_2_relu   (None, 4, 4, 512)           0   conv5_block1_2_b…
(Activation)

conv5_block1_0_conv   (None, 4, 4,         2,099,200   conv4_block23_ou…
(Conv2D)              2048)

conv5_block1_3_conv   (None, 4, 4,         1,050,624   conv5_block1_2_r…
(Conv2D)              2048)

conv5_block1_0_bn     (None, 4, 4,             8,192   conv5_block1_0_c…
(BatchNormalizatio…   2048)

conv5_block1_3_bn     (None, 4, 4,             8,192   conv5_block1_3_c…
(BatchNormalizatio…   2048)

conv5_block1_add      (None, 4, 4,                 0   conv5_block1_0_b…
(Add)                 2048)                            conv5_block1_3_b…

conv5_block1_out      (None, 4, 4,                 0   conv5_block1_add…
(Activation)          2048)

conv5_block2_1_conv   (None, 4, 4, 512)     1,049,088   conv5_block1_out…
(Conv2D)

conv5_block2_1_bn     (None, 4, 4, 512)         2,048   conv5_block2_1_c…
(BatchNormalizatio…

conv5_block2_1_relu   (None, 4, 4, 512)           0   conv5_block2_1_b…
(Activation)

conv5_block2_2_conv   (None, 4, 4, 512)     2,359,808   conv5_block2_1_r…
(Conv2D)

conv5_block2_2_bn     (None, 4, 4, 512)         2,048   conv5_block2_2_c…
(BatchNormalizatio…

conv5_block2_2_relu   (None, 4, 4, 512)           0   conv5_block2_2_b…
(Activation)

conv5_block2_3_conv   (None, 4, 4,         1,050,624   conv5_block2_2_r…
(Conv2D)              2048)

conv5_block2_3_bn     (None, 4, 4,             8,192   conv5_block2_3_c…
(BatchNormalizatio…   2048)

conv5_block2_add      (None, 4, 4,                 0   conv5_block1_out…
```

```
(Add)                  2048)                                  conv5_block2_3_b…

conv5_block2_out       (None, 4, 4,                 0         conv5_block2_add…
(Activation)           2048)

conv5_block3_1_conv    (None, 4, 4, 512)    1,049,088         conv5_block2_out…
(Conv2D)

conv5_block3_1_bn      (None, 4, 4, 512)        2,048         conv5_block3_1_c…
(BatchNormalizatio…

conv5_block3_1_relu    (None, 4, 4, 512)            0         conv5_block3_1_b…
(Activation)

conv5_block3_2_conv    (None, 4, 4, 512)    2,359,808         conv5_block3_1_r…
(Conv2D)

conv5_block3_2_bn      (None, 4, 4, 512)        2,048         conv5_block3_2_c…
(BatchNormalizatio…

conv5_block3_2_relu    (None, 4, 4, 512)            0         conv5_block3_2_b…
(Activation)

conv5_block3_3_conv    (None, 4, 4,         1,050,624         conv5_block3_2_r…
(Conv2D)               2048)

conv5_block3_3_bn      (None, 4, 4,             8,192         conv5_block3_3_c…
(BatchNormalizatio…    2048)

conv5_block3_add       (None, 4, 4,                 0         conv5_block2_out…
(Add)                  2048)                                  conv5_block3_3_b…

conv5_block3_out       (None, 4, 4,                 0         conv5_block3_add…
(Activation)           2048)

global_average_poo…    (None, 2048)                 0         conv5_block3_out…
(GlobalAveragePool…

dense_24 (Dense)       (None, 256)            524,544         global_average_p…

dropout_12             (None, 256)                  0         dense_24[0][0]
(Dropout)

dense_25 (Dense)       (None, 128)             32,896         dropout_12[0][0]

y1 (Dense)             (None, 70)               9,030         dense_25[0][0]

y2 (Dense)             (None, 121)             15,609         dense_25[0][0]
```

```
Total params: 44,404,415 (169.39 MB)

Trainable params: 582,079 (2.22 MB)

Non-trainable params: 42,658,176 (162.73 MB)

Optimizer params: 1,164,160 (4.44 MB)
```

Train del modello

[ ]:
```python
input_shape = x_train.shape[1:]
num_classes_1 = len(y1_train[0])
num_classes_2 = len(y2_train[0])

# Compilazione del modello
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    #optimizer = "adam",
    loss={
            'y1': 'categorical_crossentropy',
            'y2': 'categorical_crossentropy'
        },
    metrics={
        'y1': 'accuracy',
        'y2': 'accuracy'
    }
)
```

[ ]:
```python
if(os.path.exists('modelLastDatasetLearningRate.keras')):
    model = keras.models.load_model("modelLastDatasetLearningRate.keras")
else:
    # Addestramento del modello
    history = model.fit(x_train_preprocessed,
                        {
                            'y1': y1_train,
                            'y2': y2_train
                        },
                        validation_data=(
                            x_val_preprocessed,
                            {
                                'y1': y1_val,
                                'y2': y2_val
                            }
```

```
                        ),
                        epochs=epochs_number,  # Numero di batch per epoca
                        batch_size=batch_size,
                        steps_per_epoch = steps_per_epoch,
                        validation_steps=steps_per_val,
                        callbacks=[early_stopping]
                        )
```

```
Epoch 1/100
26/26             62s 2s/step - loss:
10.7638 - y1_accuracy: 0.0234 - y1_loss: 5.0215 - y2_accuracy: 0.0102 - y2_loss:
5.7422 - val_loss: 7.2251 - val_y1_accuracy: 0.5848 - val_y1_loss: 2.7996 -
val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.4255
Epoch 2/100
26/26             27s 1s/step - loss:
9.0082 - y1_accuracy: 0.0770 - y1_loss: 4.0874 - y2_accuracy: 0.0280 - y2_loss:
4.9209 - val_loss: 6.8256 - val_y1_accuracy: 0.7589 - val_y1_loss: 2.4776 -
val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.3480
Epoch 3/100
26/26             24s 930ms/step -
loss: 8.4763 - y1_accuracy: 0.1607 - y1_loss: 3.7337 - y2_accuracy: 0.0524 -
y2_loss: 4.7426 - val_loss: 6.2223 - val_y1_accuracy: 0.7812 - val_y1_loss:
1.9993 - val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.2230
Epoch 4/100
26/26             26s 1s/step - loss:
8.1562 - y1_accuracy: 0.1788 - y1_loss: 3.5368 - y2_accuracy: 0.0671 - y2_loss:
4.6194 - val_loss: 5.7123 - val_y1_accuracy: 0.7232 - val_y1_loss: 1.6121 -
val_y2_accuracy: 0.0000e+00 - val_y2_loss: 4.1002
Epoch 5/100
26/26             26s 1s/step - loss:
7.8516 - y1_accuracy: 0.2475 - y1_loss: 3.3059 - y2_accuracy: 0.0445 - y2_loss:
4.5458 - val_loss: 5.2556 - val_y1_accuracy: 0.7054 - val_y1_loss: 1.4051 -
val_y2_accuracy: 0.0045 - val_y2_loss: 3.8505
Epoch 6/100
26/26             27s 1s/step - loss:
7.6241 - y1_accuracy: 0.2575 - y1_loss: 3.1871 - y2_accuracy: 0.0770 - y2_loss:
4.4371 - val_loss: 4.8040 - val_y1_accuracy: 0.7098 - val_y1_loss: 1.2025 -
val_y2_accuracy: 0.0134 - val_y2_loss: 3.6016
Epoch 7/100
26/26             29s 1s/step - loss:
7.2294 - y1_accuracy: 0.3225 - y1_loss: 3.0104 - y2_accuracy: 0.1214 - y2_loss:
4.2189 - val_loss: 5.0341 - val_y1_accuracy: 0.7009 - val_y1_loss: 1.2370 -
val_y2_accuracy: 0.0134 - val_y2_loss: 3.7971
Epoch 8/100
26/26             26s 1s/step - loss:
6.8319 - y1_accuracy: 0.3719 - y1_loss: 2.6951 - y2_accuracy: 0.1355 - y2_loss:
4.1369 - val_loss: 4.6410 - val_y1_accuracy: 0.7009 - val_y1_loss: 1.1248 -
val_y2_accuracy: 0.0045 - val_y2_loss: 3.5162
```

```
Epoch 9/100
26/26            26s 992ms/step -
loss: 6.3867 - y1_accuracy: 0.4059 - y1_loss: 2.5466 - y2_accuracy: 0.1579 -
y2_loss: 3.8401 - val_loss: 4.6567 - val_y1_accuracy: 0.7009 - val_y1_loss:
0.9962 - val_y2_accuracy: 0.0179 - val_y2_loss: 3.6605
Epoch 10/100
26/26            26s 1s/step - loss:
6.3368 - y1_accuracy: 0.4041 - y1_loss: 2.5182 - y2_accuracy: 0.1775 - y2_loss:
3.8186 - val_loss: 4.6281 - val_y1_accuracy: 0.7009 - val_y1_loss: 1.0534 -
val_y2_accuracy: 0.0223 - val_y2_loss: 3.5747
Epoch 11/100
26/26            25s 967ms/step -
loss: 5.9504 - y1_accuracy: 0.4584 - y1_loss: 2.3112 - y2_accuracy: 0.2268 -
y2_loss: 3.6392 - val_loss: 3.9882 - val_y1_accuracy: 0.7009 - val_y1_loss:
0.9244 - val_y2_accuracy: 0.0848 - val_y2_loss: 3.0639
Epoch 12/100
26/26            30s 1s/step - loss:
5.3914 - y1_accuracy: 0.5258 - y1_loss: 2.0158 - y2_accuracy: 0.2579 - y2_loss:
3.3756 - val_loss: 4.1294 - val_y1_accuracy: 0.7009 - val_y1_loss: 0.9972 -
val_y2_accuracy: 0.1652 - val_y2_loss: 3.1322
Epoch 13/100
26/26            29s 1s/step - loss:
5.1020 - y1_accuracy: 0.5228 - y1_loss: 1.9377 - y2_accuracy: 0.3116 - y2_loss:
3.1642 - val_loss: 4.1919 - val_y1_accuracy: 0.7143 - val_y1_loss: 0.7618 -
val_y2_accuracy: 0.0759 - val_y2_loss: 3.4301
Epoch 14/100
26/26            25s 977ms/step -
loss: 4.9177 - y1_accuracy: 0.5283 - y1_loss: 1.8818 - y2_accuracy: 0.3278 -
y2_loss: 3.0359 - val_loss: 4.1628 - val_y1_accuracy: 0.7009 - val_y1_loss:
0.8553 - val_y2_accuracy: 0.0759 - val_y2_loss: 3.3075
Epoch 15/100
26/26            26s 1s/step - loss:
4.4770 - y1_accuracy: 0.5669 - y1_loss: 1.6917 - y2_accuracy: 0.3946 - y2_loss:
2.7853 - val_loss: 3.9742 - val_y1_accuracy: 0.7009 - val_y1_loss: 0.8945 -
val_y2_accuracy: 0.0045 - val_y2_loss: 3.0796
Epoch 16/100
26/26            27s 1s/step - loss:
4.1196 - y1_accuracy: 0.6121 - y1_loss: 1.5455 - y2_accuracy: 0.4124 - y2_loss:
2.5741 - val_loss: 3.3318 - val_y1_accuracy: 0.7054 - val_y1_loss: 0.7533 -
val_y2_accuracy: 0.0402 - val_y2_loss: 2.5786
Epoch 17/100
26/26            25s 981ms/step -
loss: 4.1931 - y1_accuracy: 0.5831 - y1_loss: 1.6178 - y2_accuracy: 0.4010 -
y2_loss: 2.5753 - val_loss: 3.3537 - val_y1_accuracy: 0.7009 - val_y1_loss:
0.8346 - val_y2_accuracy: 0.0848 - val_y2_loss: 2.5190
Epoch 18/100
26/26            26s 1s/step - loss:
3.9491 - y1_accuracy: 0.6159 - y1_loss: 1.5517 - y2_accuracy: 0.4597 - y2_loss:
```

```
2.3974 - val_loss: 3.4515 - val_y1_accuracy: 0.7009 - val_y1_loss: 0.8332 -
val_y2_accuracy: 0.0179 - val_y2_loss: 2.6183
Epoch 19/100
26/26            27s 1s/step - loss:
4.0051 - y1_accuracy: 0.5823 - y1_loss: 1.6363 - y2_accuracy: 0.4340 - y2_loss:
2.3688 - val_loss: 3.4929 - val_y1_accuracy: 0.7098 - val_y1_loss: 0.7651 -
val_y2_accuracy: 0.0268 - val_y2_loss: 2.7278
Epoch 20/100
26/26            25s 979ms/step -
loss: 3.5652 - y1_accuracy: 0.6498 - y1_loss: 1.3505 - y2_accuracy: 0.4755 -
y2_loss: 2.2147 - val_loss: 3.4087 - val_y1_accuracy: 0.7009 - val_y1_loss:
0.7720 - val_y2_accuracy: 0.0536 - val_y2_loss: 2.6366
Epoch 21/100
26/26            25s 963ms/step -
loss: 3.5189 - y1_accuracy: 0.6181 - y1_loss: 1.4526 - y2_accuracy: 0.4893 -
y2_loss: 2.0663 - val_loss: 3.0017 - val_y1_accuracy: 0.7188 - val_y1_loss:
0.6905 - val_y2_accuracy: 0.2411 - val_y2_loss: 2.3112
Epoch 22/100
26/26            23s 896ms/step -
loss: 3.3266 - y1_accuracy: 0.6490 - y1_loss: 1.3426 - y2_accuracy: 0.5064 -
y2_loss: 1.9840 - val_loss: 3.0470 - val_y1_accuracy: 0.7098 - val_y1_loss:
0.7425 - val_y2_accuracy: 0.1964 - val_y2_loss: 2.3045
Epoch 23/100
26/26            24s 924ms/step -
loss: 2.9732 - y1_accuracy: 0.6962 - y1_loss: 1.1886 - y2_accuracy: 0.5729 -
y2_loss: 1.7846 - val_loss: 2.9384 - val_y1_accuracy: 0.7098 - val_y1_loss:
0.7243 - val_y2_accuracy: 0.2232 - val_y2_loss: 2.2141
Epoch 24/100
26/26            23s 901ms/step -
loss: 3.0044 - y1_accuracy: 0.6833 - y1_loss: 1.2103 - y2_accuracy: 0.5610 -
y2_loss: 1.7941 - val_loss: 2.6889 - val_y1_accuracy: 0.7188 - val_y1_loss:
0.6711 - val_y2_accuracy: 0.3036 - val_y2_loss: 2.0178
Epoch 25/100
26/26            24s 928ms/step -
loss: 2.6677 - y1_accuracy: 0.7240 - y1_loss: 1.1012 - y2_accuracy: 0.6163 -
y2_loss: 1.5665 - val_loss: 2.5036 - val_y1_accuracy: 0.7321 - val_y1_loss:
0.6390 - val_y2_accuracy: 0.3929 - val_y2_loss: 1.8645
Epoch 26/100
26/26            23s 906ms/step -
loss: 2.7307 - y1_accuracy: 0.7248 - y1_loss: 1.0550 - y2_accuracy: 0.5991 -
y2_loss: 1.6757 - val_loss: 2.1577 - val_y1_accuracy: 0.8214 - val_y1_loss:
0.4574 - val_y2_accuracy: 0.4821 - val_y2_loss: 1.7003
Epoch 27/100
26/26            23s 900ms/step -
loss: 2.6455 - y1_accuracy: 0.6879 - y1_loss: 1.0787 - y2_accuracy: 0.6126 -
y2_loss: 1.5668 - val_loss: 1.9235 - val_y1_accuracy: 0.7946 - val_y1_loss:
0.5026 - val_y2_accuracy: 0.5804 - val_y2_loss: 1.4210
Epoch 28/100
```

```
26/26            23s 891ms/step -
loss: 2.4444 - y1_accuracy: 0.7315 - y1_loss: 0.9731 - y2_accuracy: 0.6567 -
y2_loss: 1.4712 - val_loss: 1.7038 - val_y1_accuracy: 0.8705 - val_y1_loss:
0.3560 - val_y2_accuracy: 0.6562 - val_y2_loss: 1.3479
Epoch 29/100
26/26            23s 886ms/step -
loss: 2.1435 - y1_accuracy: 0.7429 - y1_loss: 0.8602 - y2_accuracy: 0.6886 -
y2_loss: 1.2833 - val_loss: 1.8114 - val_y1_accuracy: 0.7366 - val_y1_loss:
0.6112 - val_y2_accuracy: 0.7589 - val_y2_loss: 1.2002
Epoch 30/100
26/26            23s 889ms/step -
loss: 2.3893 - y1_accuracy: 0.7297 - y1_loss: 0.9709 - y2_accuracy: 0.6354 -
y2_loss: 1.4184 - val_loss: 1.9051 - val_y1_accuracy: 0.7277 - val_y1_loss:
0.6405 - val_y2_accuracy: 0.7143 - val_y2_loss: 1.2646
Epoch 31/100
26/26            23s 898ms/step -
loss: 2.1468 - y1_accuracy: 0.7619 - y1_loss: 0.8745 - y2_accuracy: 0.6959 -
y2_loss: 1.2723 - val_loss: 1.7280 - val_y1_accuracy: 0.8348 - val_y1_loss:
0.4584 - val_y2_accuracy: 0.7321 - val_y2_loss: 1.2697
Epoch 32/100
26/26            23s 903ms/step -
loss: 2.1375 - y1_accuracy: 0.7271 - y1_loss: 0.8874 - y2_accuracy: 0.6908 -
y2_loss: 1.2502 - val_loss: 1.9483 - val_y1_accuracy: 0.7812 - val_y1_loss:
0.5723 - val_y2_accuracy: 0.6741 - val_y2_loss: 1.3760
Epoch 33/100
26/26            23s 912ms/step -
loss: 2.2829 - y1_accuracy: 0.7258 - y1_loss: 0.9238 - y2_accuracy: 0.6635 -
y2_loss: 1.3591 - val_loss: 1.5807 - val_y1_accuracy: 0.8750 - val_y1_loss:
0.3931 - val_y2_accuracy: 0.7277 - val_y2_loss: 1.1876
Epoch 34/100
26/26            23s 912ms/step -
loss: 1.9662 - y1_accuracy: 0.7861 - y1_loss: 0.7819 - y2_accuracy: 0.7067 -
y2_loss: 1.1844 - val_loss: 1.4172 - val_y1_accuracy: 0.8482 - val_y1_loss:
0.4236 - val_y2_accuracy: 0.8125 - val_y2_loss: 0.9936
Epoch 35/100
26/26            23s 899ms/step -
loss: 2.1827 - y1_accuracy: 0.7514 - y1_loss: 0.8739 - y2_accuracy: 0.7024 -
y2_loss: 1.3088 - val_loss: 1.6098 - val_y1_accuracy: 0.8571 - val_y1_loss:
0.4472 - val_y2_accuracy: 0.6964 - val_y2_loss: 1.1626
Epoch 36/100
26/26            23s 898ms/step -
loss: 2.0933 - y1_accuracy: 0.7711 - y1_loss: 0.8301 - y2_accuracy: 0.6553 -
y2_loss: 1.2632 - val_loss: 1.6037 - val_y1_accuracy: 0.8705 - val_y1_loss:
0.3151 - val_y2_accuracy: 0.6562 - val_y2_loss: 1.2886
Epoch 37/100
26/26            23s 900ms/step -
loss: 1.8433 - y1_accuracy: 0.7949 - y1_loss: 0.7339 - y2_accuracy: 0.7261 -
y2_loss: 1.1094 - val_loss: 1.3439 - val_y1_accuracy: 0.8705 - val_y1_loss:
```

```
0.3664 - val_y2_accuracy: 0.8125 - val_y2_loss: 0.9775
Epoch 38/100
26/26            23s 901ms/step -
loss: 1.8371 - y1_accuracy: 0.7861 - y1_loss: 0.7755 - y2_accuracy: 0.7244 -
y2_loss: 1.0616 - val_loss: 1.1341 - val_y1_accuracy: 0.9107 - val_y1_loss:
0.2686 - val_y2_accuracy: 0.8214 - val_y2_loss: 0.8655
Epoch 39/100
26/26            23s 891ms/step -
loss: 1.7812 - y1_accuracy: 0.7718 - y1_loss: 0.7798 - y2_accuracy: 0.7379 -
y2_loss: 1.0014 - val_loss: 1.3181 - val_y1_accuracy: 0.8795 - val_y1_loss:
0.3160 - val_y2_accuracy: 0.7500 - val_y2_loss: 1.0020
Epoch 40/100
26/26            23s 897ms/step -
loss: 1.7310 - y1_accuracy: 0.8117 - y1_loss: 0.6805 - y2_accuracy: 0.7205 -
y2_loss: 1.0505 - val_loss: 1.3806 - val_y1_accuracy: 0.8705 - val_y1_loss:
0.4035 - val_y2_accuracy: 0.8170 - val_y2_loss: 0.9771
Epoch 41/100
26/26            23s 899ms/step -
loss: 1.6874 - y1_accuracy: 0.7888 - y1_loss: 0.7379 - y2_accuracy: 0.7495 -
y2_loss: 0.9495 - val_loss: 1.1521 - val_y1_accuracy: 0.9062 - val_y1_loss:
0.2713 - val_y2_accuracy: 0.8304 - val_y2_loss: 0.8808
Epoch 42/100
26/26            24s 918ms/step -
loss: 1.5940 - y1_accuracy: 0.8301 - y1_loss: 0.6525 - y2_accuracy: 0.7705 -
y2_loss: 0.9415 - val_loss: 1.1844 - val_y1_accuracy: 0.8750 - val_y1_loss:
0.3370 - val_y2_accuracy: 0.8705 - val_y2_loss: 0.8474
Epoch 43/100
26/26            23s 894ms/step -
loss: 1.6128 - y1_accuracy: 0.8238 - y1_loss: 0.6071 - y2_accuracy: 0.7286 -
y2_loss: 1.0057 - val_loss: 1.0527 - val_y1_accuracy: 0.9196 - val_y1_loss:
0.2177 - val_y2_accuracy: 0.8259 - val_y2_loss: 0.8350
Epoch 44/100
26/26            23s 894ms/step -
loss: 1.5911 - y1_accuracy: 0.8014 - y1_loss: 0.6511 - y2_accuracy: 0.7516 -
y2_loss: 0.9400 - val_loss: 1.0204 - val_y1_accuracy: 0.8884 - val_y1_loss:
0.2797 - val_y2_accuracy: 0.8839 - val_y2_loss: 0.7407
Epoch 45/100
26/26            23s 886ms/step -
loss: 1.4371 - y1_accuracy: 0.8333 - y1_loss: 0.5604 - y2_accuracy: 0.7726 -
y2_loss: 0.8767 - val_loss: 1.2416 - val_y1_accuracy: 0.8750 - val_y1_loss:
0.3183 - val_y2_accuracy: 0.8170 - val_y2_loss: 0.9234
Epoch 46/100
26/26            23s 894ms/step -
loss: 1.3955 - y1_accuracy: 0.8125 - y1_loss: 0.6111 - y2_accuracy: 0.7891 -
y2_loss: 0.7843 - val_loss: 1.0786 - val_y1_accuracy: 0.9018 - val_y1_loss:
0.2594 - val_y2_accuracy: 0.8661 - val_y2_loss: 0.8192
Epoch 47/100
26/26            23s 895ms/step -
```

```
loss: 1.4607 - y1_accuracy: 0.8140 - y1_loss: 0.6397 - y2_accuracy: 0.7956 -
y2_loss: 0.8210 - val_loss: 1.5229 - val_y1_accuracy: 0.8125 - val_y1_loss:
0.4793 - val_y2_accuracy: 0.6830 - val_y2_loss: 1.0436
Epoch 48/100
26/26            24s 938ms/step -
loss: 1.5940 - y1_accuracy: 0.7943 - y1_loss: 0.6814 - y2_accuracy: 0.7698 -
y2_loss: 0.9126 - val_loss: 0.7710 - val_y1_accuracy: 0.9688 - val_y1_loss:
0.1758 - val_y2_accuracy: 0.9062 - val_y2_loss: 0.5952
Epoch 49/100
26/26            23s 889ms/step -
loss: 1.3511 - y1_accuracy: 0.8299 - y1_loss: 0.5682 - y2_accuracy: 0.8006 -
y2_loss: 0.7828 - val_loss: 0.9298 - val_y1_accuracy: 0.9062 - val_y1_loss:
0.2376 - val_y2_accuracy: 0.7768 - val_y2_loss: 0.6922
Epoch 50/100
26/26            24s 918ms/step -
loss: 1.5732 - y1_accuracy: 0.8217 - y1_loss: 0.6423 - y2_accuracy: 0.7685 -
y2_loss: 0.9310 - val_loss: 0.9164 - val_y1_accuracy: 0.9062 - val_y1_loss:
0.2239 - val_y2_accuracy: 0.7946 - val_y2_loss: 0.6926
Epoch 51/100
26/26            24s 921ms/step -
loss: 1.4013 - y1_accuracy: 0.8357 - y1_loss: 0.5336 - y2_accuracy: 0.7634 -
y2_loss: 0.8677 - val_loss: 0.8667 - val_y1_accuracy: 0.9643 - val_y1_loss:
0.1677 - val_y2_accuracy: 0.8125 - val_y2_loss: 0.6990
Epoch 52/100
26/26            24s 918ms/step -
loss: 1.1725 - y1_accuracy: 0.8591 - y1_loss: 0.5091 - y2_accuracy: 0.8405 -
y2_loss: 0.6633 - val_loss: 1.0703 - val_y1_accuracy: 0.8705 - val_y1_loss:
0.3248 - val_y2_accuracy: 0.7857 - val_y2_loss: 0.7456
Epoch 53/100
26/26            24s 926ms/step -
loss: 1.3880 - y1_accuracy: 0.8168 - y1_loss: 0.5579 - y2_accuracy: 0.7766 -
y2_loss: 0.8301 - val_loss: 0.8019 - val_y1_accuracy: 0.9107 - val_y1_loss:
0.2303 - val_y2_accuracy: 0.8839 - val_y2_loss: 0.5716
Epoch 54/100
26/26            24s 922ms/step -
loss: 1.3883 - y1_accuracy: 0.8350 - y1_loss: 0.5653 - y2_accuracy: 0.7939 -
y2_loss: 0.8230 - val_loss: 0.8067 - val_y1_accuracy: 0.9018 - val_y1_loss:
0.2797 - val_y2_accuracy: 0.9152 - val_y2_loss: 0.5269
Epoch 55/100
26/26            24s 926ms/step -
loss: 1.4147 - y1_accuracy: 0.8198 - y1_loss: 0.6064 - y2_accuracy: 0.8011 -
y2_loss: 0.8083 - val_loss: 0.6986 - val_y1_accuracy: 0.9241 - val_y1_loss:
0.1713 - val_y2_accuracy: 0.9152 - val_y2_loss: 0.5273
Epoch 56/100
26/26            24s 917ms/step -
loss: 1.1697 - y1_accuracy: 0.8779 - y1_loss: 0.4798 - y2_accuracy: 0.8230 -
y2_loss: 0.6899 - val_loss: 0.7689 - val_y1_accuracy: 0.9107 - val_y1_loss:
0.2047 - val_y2_accuracy: 0.8393 - val_y2_loss: 0.5641
```

```
Epoch 57/100
26/26            23s 908ms/step -
loss: 1.3609 - y1_accuracy: 0.8402 - y1_loss: 0.5771 - y2_accuracy: 0.8125 -
y2_loss: 0.7838 - val_loss: 0.5948 - val_y1_accuracy: 0.9464 - val_y1_loss:
0.1815 - val_y2_accuracy: 0.9152 - val_y2_loss: 0.4133
Epoch 58/100
26/26            24s 928ms/step -
loss: 1.1990 - y1_accuracy: 0.8649 - y1_loss: 0.4703 - y2_accuracy: 0.7788 -
y2_loss: 0.7287 - val_loss: 0.4956 - val_y1_accuracy: 0.9821 - val_y1_loss:
0.1341 - val_y2_accuracy: 0.9732 - val_y2_loss: 0.3615
Epoch 59/100
26/26            24s 906ms/step -
loss: 1.1144 - y1_accuracy: 0.8549 - y1_loss: 0.4574 - y2_accuracy: 0.8346 -
y2_loss: 0.6570 - val_loss: 0.5447 - val_y1_accuracy: 0.9330 - val_y1_loss:
0.1854 - val_y2_accuracy: 0.9688 - val_y2_loss: 0.3593
Epoch 60/100
26/26            23s 897ms/step -
loss: 1.2423 - y1_accuracy: 0.8387 - y1_loss: 0.5519 - y2_accuracy: 0.8159 -
y2_loss: 0.6904 - val_loss: 0.5858 - val_y1_accuracy: 0.9866 - val_y1_loss:
0.1476 - val_y2_accuracy: 0.9107 - val_y2_loss: 0.4382
Epoch 61/100
26/26            23s 897ms/step -
loss: 1.0323 - y1_accuracy: 0.8769 - y1_loss: 0.4737 - y2_accuracy: 0.8544 -
y2_loss: 0.5586 - val_loss: 0.4684 - val_y1_accuracy: 0.9598 - val_y1_loss:
0.1453 - val_y2_accuracy: 0.9911 - val_y2_loss: 0.3231
Epoch 62/100
26/26            23s 908ms/step -
loss: 1.0803 - y1_accuracy: 0.8826 - y1_loss: 0.4465 - y2_accuracy: 0.8303 -
y2_loss: 0.6337 - val_loss: 0.4730 - val_y1_accuracy: 0.9375 - val_y1_loss:
0.1735 - val_y2_accuracy: 0.9821 - val_y2_loss: 0.2996
Epoch 63/100
26/26            24s 915ms/step -
loss: 1.0831 - y1_accuracy: 0.8590 - y1_loss: 0.4505 - y2_accuracy: 0.8386 -
y2_loss: 0.6326 - val_loss: 0.5342 - val_y1_accuracy: 0.9375 - val_y1_loss:
0.1428 - val_y2_accuracy: 0.9643 - val_y2_loss: 0.3915
Epoch 64/100
26/26            24s 921ms/step -
loss: 0.9809 - y1_accuracy: 0.8929 - y1_loss: 0.4006 - y2_accuracy: 0.8422 -
y2_loss: 0.5803 - val_loss: 0.4611 - val_y1_accuracy: 0.9732 - val_y1_loss:
0.1304 - val_y2_accuracy: 0.9777 - val_y2_loss: 0.3308
Epoch 65/100
26/26            23s 889ms/step -
loss: 0.9736 - y1_accuracy: 0.8934 - y1_loss: 0.3844 - y2_accuracy: 0.8350 -
y2_loss: 0.5892 - val_loss: 0.3966 - val_y1_accuracy: 0.9777 - val_y1_loss:
0.1262 - val_y2_accuracy: 0.9911 - val_y2_loss: 0.2704
Epoch 66/100
26/26            23s 891ms/step -
loss: 0.8843 - y1_accuracy: 0.9006 - y1_loss: 0.3681 - y2_accuracy: 0.8406 -
```

```
y2_loss: 0.5162 - val_loss: 0.3947 - val_y1_accuracy: 1.0000 - val_y1_loss:
0.0791 - val_y2_accuracy: 0.9777 - val_y2_loss: 0.3156
Epoch 67/100
26/26            26s 993ms/step -
loss: 1.0563 - y1_accuracy: 0.8698 - y1_loss: 0.4579 - y2_accuracy: 0.8594 -
y2_loss: 0.5983 - val_loss: 0.3469 - val_y1_accuracy: 0.9866 - val_y1_loss:
0.1163 - val_y2_accuracy: 0.9598 - val_y2_loss: 0.2307
Epoch 68/100
22/26            3s 760ms/step -
loss: 1.0021 - y1_accuracy: 0.8810 - y1_loss: 0.4201 - y2_accuracy: 0.8591 -
y2_loss: 0.5820

C:\Users\fabio\PycharmProjects\AMLProject\.venv\lib\site-
packages\keras\src\trainers\epoch_iterator.py:107: UserWarning: Your input ran
out of data; interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to use the
`.repeat()` function when building your dataset.
  self._interrupted_warning()

26/26            22s 833ms/step -
loss: 0.9986 - y1_accuracy: 0.8814 - y1_loss: 0.4185 - y2_accuracy: 0.8593 -
y2_loss: 0.5799 - val_loss: 0.4754 - val_y1_accuracy: 0.9688 - val_y1_loss:
0.1463 - val_y2_accuracy: 0.9062 - val_y2_loss: 0.3291
Epoch 69/100
26/26            23s 903ms/step -
loss: 1.0103 - y1_accuracy: 0.8887 - y1_loss: 0.4243 - y2_accuracy: 0.8442 -
y2_loss: 0.5860 - val_loss: 0.3599 - val_y1_accuracy: 0.9911 - val_y1_loss:
0.0909 - val_y2_accuracy: 0.9420 - val_y2_loss: 0.2690
Epoch 70/100
26/26            23s 908ms/step -
loss: 0.9618 - y1_accuracy: 0.8924 - y1_loss: 0.3992 - y2_accuracy: 0.8469 -
y2_loss: 0.5626 - val_loss: 0.3176 - val_y1_accuracy: 0.9911 - val_y1_loss:
0.0998 - val_y2_accuracy: 0.9955 - val_y2_loss: 0.2178
Epoch 71/100
26/26            23s 904ms/step -
loss: 0.9221 - y1_accuracy: 0.8933 - y1_loss: 0.3884 - y2_accuracy: 0.8538 -
y2_loss: 0.5338 - val_loss: 0.4230 - val_y1_accuracy: 0.9420 - val_y1_loss:
0.1810 - val_y2_accuracy: 0.9955 - val_y2_loss: 0.2420
Epoch 72/100
26/26            23s 900ms/step -
loss: 1.0399 - y1_accuracy: 0.8825 - y1_loss: 0.4371 - y2_accuracy: 0.8310 -
y2_loss: 0.6028 - val_loss: 0.2831 - val_y1_accuracy: 1.0000 - val_y1_loss:
0.0831 - val_y2_accuracy: 0.9955 - val_y2_loss: 0.2001
Epoch 73/100
26/26            23s 884ms/step -
loss: 0.8772 - y1_accuracy: 0.8889 - y1_loss: 0.3954 - y2_accuracy: 0.8712 -
y2_loss: 0.4818 - val_loss: 0.4781 - val_y1_accuracy: 0.9598 - val_y1_loss:
0.1855 - val_y2_accuracy: 0.9286 - val_y2_loss: 0.2926
Epoch 74/100
```

```
26/26            23s 908ms/step -
loss: 0.9688 - y1_accuracy: 0.8716 - y1_loss: 0.3896 - y2_accuracy: 0.8438 -
y2_loss: 0.5792 - val_loss: 0.3397 - val_y1_accuracy: 0.9955 - val_y1_loss:
0.1033 - val_y2_accuracy: 0.9777 - val_y2_loss: 0.2364
Epoch 75/100
26/26            23s 896ms/step -
loss: 0.8728 - y1_accuracy: 0.8995 - y1_loss: 0.3560 - y2_accuracy: 0.8516 -
y2_loss: 0.5168 - val_loss: 0.3445 - val_y1_accuracy: 0.9866 - val_y1_loss:
0.1158 - val_y2_accuracy: 0.9643 - val_y2_loss: 0.2287
Epoch 76/100
26/26            24s 930ms/step -
loss: 0.9442 - y1_accuracy: 0.8947 - y1_loss: 0.3725 - y2_accuracy: 0.8343 -
y2_loss: 0.5717 - val_loss: 0.2765 - val_y1_accuracy: 0.9866 - val_y1_loss:
0.1007 - val_y2_accuracy: 0.9777 - val_y2_loss: 0.1758
Epoch 77/100
26/26            23s 892ms/step -
loss: 0.8995 - y1_accuracy: 0.8881 - y1_loss: 0.3773 - y2_accuracy: 0.8615 -
y2_loss: 0.5222 - val_loss: 0.2356 - val_y1_accuracy: 0.9732 - val_y1_loss:
0.1365 - val_y2_accuracy: 1.0000 - val_y2_loss: 0.0991
Epoch 78/100
26/26            24s 915ms/step -
loss: 0.8834 - y1_accuracy: 0.8786 - y1_loss: 0.3505 - y2_accuracy: 0.8552 -
y2_loss: 0.5329 - val_loss: 0.3734 - val_y1_accuracy: 0.9732 - val_y1_loss:
0.1325 - val_y2_accuracy: 0.9598 - val_y2_loss: 0.2409
Epoch 79/100
26/26            23s 906ms/step -
loss: 0.8479 - y1_accuracy: 0.9141 - y1_loss: 0.3489 - y2_accuracy: 0.8709 -
y2_loss: 0.4990 - val_loss: 0.3229 - val_y1_accuracy: 0.9732 - val_y1_loss:
0.1171 - val_y2_accuracy: 0.9955 - val_y2_loss: 0.2058
Epoch 80/100
26/26            23s 894ms/step -
loss: 0.8416 - y1_accuracy: 0.8841 - y1_loss: 0.4034 - y2_accuracy: 0.8805 -
y2_loss: 0.4382 - val_loss: 0.4273 - val_y1_accuracy: 0.9955 - val_y1_loss:
0.0923 - val_y2_accuracy: 0.9152 - val_y2_loss: 0.3349
Epoch 81/100
26/26            24s 921ms/step -
loss: 0.8058 - y1_accuracy: 0.8892 - y1_loss: 0.3567 - y2_accuracy: 0.8885 -
y2_loss: 0.4492 - val_loss: 0.5069 - val_y1_accuracy: 0.9554 - val_y1_loss:
0.1748 - val_y2_accuracy: 0.8929 - val_y2_loss: 0.3322
Epoch 82/100
26/26            24s 911ms/step -
loss: 0.8646 - y1_accuracy: 0.8898 - y1_loss: 0.3686 - y2_accuracy: 0.8699 -
y2_loss: 0.4961 - val_loss: 0.3175 - val_y1_accuracy: 0.9955 - val_y1_loss:
0.0935 - val_y2_accuracy: 0.9688 - val_y2_loss: 0.2240
Epoch 83/100
26/26            23s 903ms/step -
loss: 0.7527 - y1_accuracy: 0.9064 - y1_loss: 0.2858 - y2_accuracy: 0.8724 -
y2_loss: 0.4669 - val_loss: 0.3703 - val_y1_accuracy: 0.9688 - val_y1_loss:
```

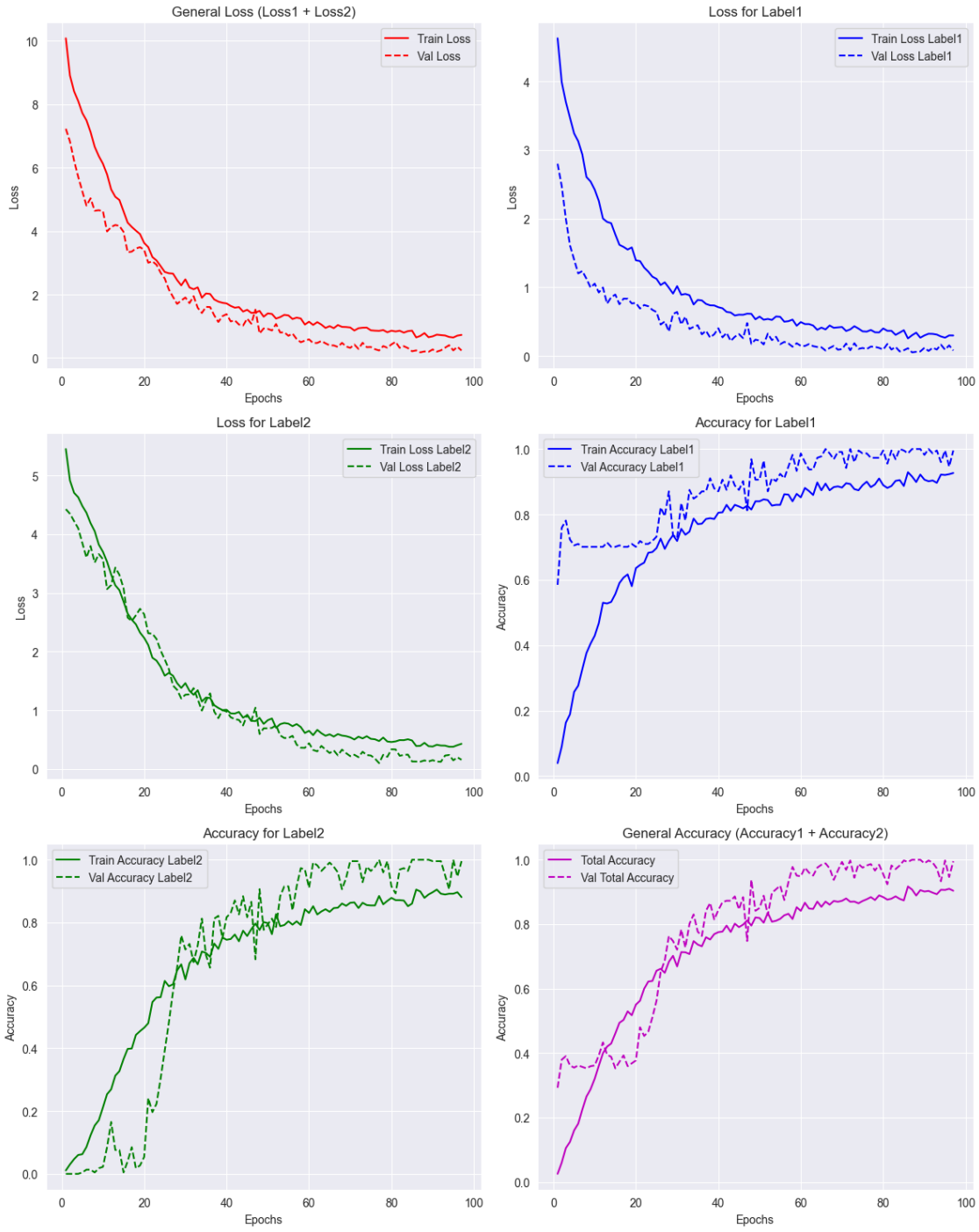
```
0.1313 - val_y2_accuracy: 0.9732 - val_y2_loss: 0.2390
Epoch 84/100
26/26            23s 891ms/step -
loss: 0.8357 - y1_accuracy: 0.9084 - y1_loss: 0.3311 - y2_accuracy: 0.8570 -
y2_loss: 0.5046 - val_loss: 0.3098 - val_y1_accuracy: 1.0000 - val_y1_loss:
0.0653 - val_y2_accuracy: 0.9643 - val_y2_loss: 0.2445
Epoch 85/100
26/26            24s 948ms/step -
loss: 0.8716 - y1_accuracy: 0.8924 - y1_loss: 0.3614 - y2_accuracy: 0.8545 -
y2_loss: 0.5101 - val_loss: 0.2105 - val_y1_accuracy: 0.9955 - val_y1_loss:
0.0833 - val_y2_accuracy: 1.0000 - val_y2_loss: 0.1272
Epoch 86/100
26/26            24s 928ms/step -
loss: 0.6569 - y1_accuracy: 0.9275 - y1_loss: 0.2537 - y2_accuracy: 0.9038 -
y2_loss: 0.4032 - val_loss: 0.2380 - val_y1_accuracy: 0.9866 - val_y1_loss:
0.1122 - val_y2_accuracy: 1.0000 - val_y2_loss: 0.1258
Epoch 87/100
26/26            24s 913ms/step -
loss: 0.6877 - y1_accuracy: 0.9220 - y1_loss: 0.3040 - y2_accuracy: 0.8939 -
y2_loss: 0.3838 - val_loss: 0.1753 - val_y1_accuracy: 1.0000 - val_y1_loss:
0.0519 - val_y2_accuracy: 1.0000 - val_y2_loss: 0.1233
Epoch 88/100
26/26            23s 902ms/step -
loss: 0.7750 - y1_accuracy: 0.8941 - y1_loss: 0.3312 - y2_accuracy: 0.8795 -
y2_loss: 0.4438 - val_loss: 0.2048 - val_y1_accuracy: 1.0000 - val_y1_loss:
0.0601 - val_y2_accuracy: 1.0000 - val_y2_loss: 0.1446
Epoch 89/100
26/26            24s 932ms/step -
loss: 0.6556 - y1_accuracy: 0.9243 - y1_loss: 0.2667 - y2_accuracy: 0.8876 -
y2_loss: 0.3889 - val_loss: 0.1928 - val_y1_accuracy: 1.0000 - val_y1_loss:
0.0658 - val_y2_accuracy: 1.0000 - val_y2_loss: 0.1270
Epoch 90/100
26/26            23s 901ms/step -
loss: 0.6601 - y1_accuracy: 0.8995 - y1_loss: 0.3076 - y2_accuracy: 0.9127 -
y2_loss: 0.3525 - val_loss: 0.2680 - val_y1_accuracy: 0.9821 - val_y1_loss:
0.1188 - val_y2_accuracy: 0.9955 - val_y2_loss: 0.1492
Epoch 91/100
26/26            24s 941ms/step -
loss: 0.7190 - y1_accuracy: 0.9052 - y1_loss: 0.3112 - y2_accuracy: 0.9029 -
y2_loss: 0.4078 - val_loss: 0.1978 - val_y1_accuracy: 1.0000 - val_y1_loss:
0.0726 - val_y2_accuracy: 0.9955 - val_y2_loss: 0.1252
Epoch 92/100
26/26            23s 903ms/step -
loss: 0.6664 - y1_accuracy: 0.9146 - y1_loss: 0.2961 - y2_accuracy: 0.9053 -
y2_loss: 0.3703 - val_loss: 0.2407 - val_y1_accuracy: 0.9866 - val_y1_loss:
0.1189 - val_y2_accuracy: 0.9955 - val_y2_loss: 0.1218
Epoch 93/100
26/26            23s 905ms/step -
```

```
loss: 0.7331 - y1_accuracy: 0.8933 - y1_loss: 0.3264 - y2_accuracy: 0.8817 -
y2_loss: 0.4067 - val_loss: 0.3216 - val_y1_accuracy: 0.9955 - val_y1_loss:
0.0936 - val_y2_accuracy: 0.9509 - val_y2_loss: 0.2280
Epoch 94/100
26/26             24s 923ms/step -
loss: 0.7096 - y1_accuracy: 0.9120 - y1_loss: 0.3012 - y2_accuracy: 0.8867 -
y2_loss: 0.4084 - val_loss: 0.4022 - val_y1_accuracy: 0.9598 - val_y1_loss:
0.1644 - val_y2_accuracy: 0.9062 - val_y2_loss: 0.2377
Epoch 95/100
26/26             24s 917ms/step -
loss: 0.6009 - y1_accuracy: 0.9385 - y1_loss: 0.2472 - y2_accuracy: 0.8986 -
y2_loss: 0.3537 - val_loss: 0.2376 - val_y1_accuracy: 0.9955 - val_y1_loss:
0.0915 - val_y2_accuracy: 1.0000 - val_y2_loss: 0.1461
Epoch 96/100
26/26             24s 916ms/step -
loss: 0.7072 - y1_accuracy: 0.9285 - y1_loss: 0.2812 - y2_accuracy: 0.8883 -
y2_loss: 0.4260 - val_loss: 0.3515 - val_y1_accuracy: 0.9464 - val_y1_loss:
0.1545 - val_y2_accuracy: 0.9464 - val_y2_loss: 0.1970
Epoch 97/100
26/26             24s 921ms/step -
loss: 0.8336 - y1_accuracy: 0.9044 - y1_loss: 0.3384 - y2_accuracy: 0.8527 -
y2_loss: 0.4952 - val_loss: 0.2358 - val_y1_accuracy: 0.9955 - val_y1_loss:
0.0799 - val_y2_accuracy: 0.9955 - val_y2_loss: 0.1559
```

[ ]:
```python
if(os.path.exists('modelLastDatasetLearningRate.keras')):
    model = keras.models.load_model("modelLastDatasetLearningRate.keras")
else:
    epochs = range(1, len(history.history['loss']) + 1)

    plt.figure(figsize=(12, 15))  # Aumento l'altezza per adattare più subplot

    # Grafico della Loss totale
    plt.subplot(3, 2, 1)
    plt.plot(epochs, history.history['loss'], 'r-', label='Train Loss')
    plt.plot(epochs, history.history['val_loss'], 'r--', label='Val Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('General Loss (Loss1 + Loss2)')

    # Grafico della Loss Y1 (Fruit)
    plt.subplot(3, 2, 2)
    plt.plot(epochs, history.history['y1_loss'], 'b-', label='Train Loss␣
 ↪Label1')
    plt.plot(epochs, history.history['val_y1_loss'], 'b--', label='Val Loss␣
 ↪Label1')
    plt.xlabel('Epochs')
```

```
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Loss for Label1')

    # Grafico della Loss Y2 (Quality)
    plt.subplot(3, 2, 3)
    plt.plot(epochs, history.history['y2_loss'], 'g-', label='Train Loss␣
↪Label2')
    plt.plot(epochs, history.history['val_y2_loss'], 'g--', label='Val Loss␣
↪Label2')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Loss for Label2')

    # Grafico della Accuracy Y1 (Fruit)
    plt.subplot(3, 2, 4)
    plt.plot(epochs, history.history['y1_accuracy'], 'b-', label='Train␣
↪Accuracy Label1')
    plt.plot(epochs, history.history['val_y1_accuracy'], 'b--', label='Val␣
↪Accuracy Label1')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('Accuracy for Label1')

    # Grafico della Accuracy Y2 (Quality)
    plt.subplot(3, 2, 5)
    plt.plot(epochs, history.history['y2_accuracy'], 'g-', label='Train␣
↪Accuracy Label2')
    plt.plot(epochs, history.history['val_y2_accuracy'], 'g--', label='Val␣
↪Accuracy Label2')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('Accuracy for Label2')

    # Grafico della Accuracy Totale
    total_accuracy = (np.array(history.history['y1_accuracy']) + np.
↪array(history.history['y2_accuracy'])) / 2
    val_total_accuracy = (np.array(history.history['val_y1_accuracy']) + np.
↪array(history.history['val_y2_accuracy'])) / 2

    plt.subplot(3, 2, 6)
    plt.plot(epochs, total_accuracy, 'm-', label='Total Accuracy')
    plt.plot(epochs, val_total_accuracy, 'm--', label='Val Total Accuracy')
```

```
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('General Accuracy (Accuracy1 + Accuracy2)')

    plt.tight_layout()  # Migliora la disposizione dei grafici
    plt.show()
```



Salvataggio in memoria

[ ]:
```
model.save("modelLastDatasetLearningRate.keras")
```

Valutazione metriche di classificazione

[ ]:
```
score = model.evaluate(x_test_preprocessed, [y1_test, y2_test], verbose=1)

print(f"Test Loss Totale: {score[0]:.4f}")
print(f"Test Loss Y1 (Frutto): {score[1]:.4f}")
print(f"Test Loss Y2 (Qualità): {score[2]:.4f}")
print(f"Test Accuracy Y1 (Frutto): {score[3]:.4f}")
print(f"Test Accuracy Y2 (Qualità): {score[4]:.4f}")
```

```
440/440            386s 867ms/step -
loss: 0.5130 - y1_accuracy: 0.9467 - y1_loss: 0.1835 - y2_accuracy: 0.9295 -
y2_loss: 0.3295
Test Loss Totale: 0.4855
Test Loss Y1 (Frutto): 0.2117
Test Loss Y2 (Qualità): 0.2732
Test Accuracy Y1 (Frutto): 0.9371
Test Accuracy Y2 (Qualità): 0.9408
```

[ ]:
```
# Previsione delle probabilità per ciascun output
y1_pred, y2_pred = model.predict(x_test_preprocessed)

# Convertiamo le probabilità in etichette (classe con probabilità più alta)
y1_pred_classes = y1_pred.argmax(axis=1)
y2_pred_classes = y2_pred.argmax(axis=1)

y1_test_classes = y1_test.argmax(axis=1)
y2_test_classes = y2_test.argmax(axis=1)

# Report per la prima task (Frutto)
print("Classification Report - Y1 (Frutto):")
print(classification_report(y1_test_classes, y1_pred_classes))

# Report per la seconda task (Qualità)
print("Classification Report - Y2 (Qualità):")
print(classification_report(y2_test_classes, y2_pred_classes))
```

```
440/440            348s 785ms/step
 Classification Report - Y1 (Frutto):
              precision    recall  f1-score   support

           0       0.83      0.97      0.89      1510
           1       0.91      0.90      0.90        98
```

2	0.96	0.97	0.96	183
3	0.96	0.96	0.96	286
4	1.00	0.90	0.95	90
5	0.98	0.91	0.94	92
6	1.00	0.86	0.92	28
7	0.98	0.85	0.91	98
8	0.97	0.99	0.98	196
9	1.00	0.92	0.96	98
10	1.00	0.93	0.97	30
11	0.97	1.00	0.98	140
12	0.93	0.95	0.94	686
13	0.97	0.84	0.90	90
14	0.86	0.99	0.92	98
15	0.94	0.98	0.96	98
16	0.89	1.00	0.94	182
17	0.96	0.94	0.95	249
18	0.99	0.98	0.98	98
19	0.99	0.94	0.97	141
20	0.99	0.94	0.96	140
21	0.95	0.98	0.97	59
22	0.99	0.90	0.94	98
23	0.98	0.96	0.97	682
24	0.91	0.90	0.90	196
25	0.93	0.98	0.96	100
26	0.98	0.92	0.95	92
27	1.00	0.99	0.99	98
28	1.00	0.88	0.93	98
29	1.00	0.96	0.98	93
30	0.98	0.96	0.97	94
31	0.94	0.96	0.95	98
32	0.93	0.86	0.90	196
33	0.96	0.87	0.91	98
34	1.00	0.98	0.99	98
35	0.97	0.95	0.96	98
36	0.95	0.75	0.84	183
37	0.93	0.95	0.94	60
38	0.89	0.92	0.90	98
39	0.96	1.00	0.98	147
40	0.99	0.99	0.99	98
41	0.88	0.59	0.71	194
42	0.89	0.94	0.92	236
43	0.98	0.94	0.96	266
44	1.00	0.94	0.97	95
45	1.00	0.76	0.86	98
46	0.99	0.96	0.97	98
47	0.94	0.79	0.86	343
48	0.93	0.93	0.93	1049
49	1.00	0.96	0.98	98

50	0.93	0.97	0.95	494
51	0.99	0.96	0.98	196
52	0.99	1.00	0.99	196
53	0.97	0.98	0.97	98
54	0.90	0.98	0.94	353
55	0.93	0.69	0.80	98
56	0.97	0.96	0.96	90
57	0.95	0.94	0.95	360
58	0.92	0.96	0.94	98
59	0.99	0.99	0.99	98
60	0.99	1.00	0.99	98
61	0.96	0.95	0.95	98
62	0.98	0.92	0.95	98
63	0.99	0.98	0.98	245
64	0.84	0.94	0.89	98
65	1.00	0.87	0.93	98
66	0.95	0.96	0.95	1015
67	1.00	0.99	0.99	147
68	0.98	0.95	0.96	95
69	0.95	0.90	0.92	96
accuracy			0.94	14061
macro avg	0.96	0.93	0.94	14061
weighted avg	0.94	0.94	0.94	14061

Classification Report - Y2 (Qualità):

	precision	recall	f1-score	support
0	0.82	0.93	0.87	94
1	0.82	0.76	0.79	98
2	0.86	0.83	0.84	88
3	0.89	0.93	0.91	290
4	0.84	0.96	0.90	98
5	0.90	0.94	0.92	140
6	0.84	0.73	0.78	91
7	0.85	0.89	0.87	281
8	0.99	0.94	0.96	98
9	0.90	0.95	0.93	232
10	0.94	0.89	0.91	98
11	0.89	0.96	0.93	85
12	0.98	0.99	0.98	98
13	0.96	0.97	0.96	98
14	0.94	0.94	0.94	90
15	0.98	0.97	0.97	98
16	0.93	0.88	0.90	90
17	0.99	0.93	0.96	92
18	1.00	1.00	1.00	28
19	0.94	0.93	0.93	98

20	0.97	0.99	0.98	196
21	1.00	0.93	0.96	98
22	1.00	0.93	0.97	30
23	0.96	1.00	0.98	140
24	0.91	0.96	0.93	245
25	0.94	0.93	0.93	147
26	0.99	0.98	0.98	98
27	0.98	0.88	0.92	98
28	0.99	0.98	0.98	98
29	0.94	0.93	0.94	90
30	0.81	0.98	0.89	98
31	0.99	0.96	0.97	98
32	0.99	0.99	0.99	90
33	0.92	1.00	0.96	92
34	1.00	0.91	0.95	78
35	0.92	0.99	0.95	171
36	0.98	0.97	0.97	98
37	0.99	0.98	0.98	93
38	0.98	1.00	0.99	48
39	0.96	0.98	0.97	140
40	0.98	0.95	0.97	59
41	0.96	0.94	0.95	98
42	0.95	0.99	0.97	196
43	0.87	0.98	0.92	98
44	0.99	0.98	0.98	388
45	0.88	0.87	0.87	98
46	0.90	0.93	0.91	98
47	0.98	0.94	0.96	100
48	0.99	0.93	0.96	92
49	0.99	1.00	0.99	98
50	0.98	0.90	0.94	98
51	0.93	0.98	0.95	93
52	1.00	0.96	0.98	94
53	0.85	0.95	0.90	98
54	0.96	0.71	0.82	98
55	0.96	0.93	0.94	98
56	0.94	0.95	0.94	98
57	1.00	0.99	0.99	98
58	0.99	0.94	0.96	98
59	0.98	0.86	0.91	98
60	0.95	0.69	0.80	85
61	0.89	0.95	0.92	60
62	0.88	0.93	0.90	98
63	0.95	0.99	0.97	147
64	1.00	0.98	0.99	98
65	0.86	0.77	0.81	98
66	0.92	0.84	0.88	96
67	0.87	0.97	0.92	130

68	0.87	0.97	0.92	106
69	0.97	0.93	0.95	90
70	0.95	0.94	0.95	89
71	1.00	0.95	0.98	87
72	0.99	0.95	0.97	95
73	0.96	0.78	0.86	98
74	0.96	0.97	0.96	98
75	0.95	0.95	0.95	245
76	0.87	0.85	0.86	98
77	0.93	0.90	0.92	280
78	0.98	1.00	0.99	98
79	0.90	0.86	0.88	140
80	0.93	0.90	0.92	60
81	0.96	0.93	0.94	98
82	0.99	0.93	0.96	133
83	0.97	0.96	0.96	142
84	0.94	0.94	0.94	98
85	0.99	0.98	0.98	98
86	1.00	0.97	0.98	88
87	0.98	0.94	0.96	140
88	0.94	0.97	0.96	133
89	0.96	0.98	0.97	133
90	0.99	0.99	0.99	98
91	0.96	0.99	0.97	98
92	0.99	0.99	0.99	98
93	0.97	1.00	0.98	98
94	0.96	1.00	0.98	98
95	0.94	0.97	0.96	353
96	0.87	0.77	0.82	98
97	1.00	0.96	0.98	90
98	0.94	0.89	0.91	90
99	0.98	0.91	0.94	90
100	0.97	0.94	0.96	90
101	0.89	0.91	0.90	90
102	0.94	0.98	0.96	98
103	0.99	1.00	0.99	98
104	1.00	1.00	1.00	98
105	0.96	0.97	0.96	98
106	0.98	0.96	0.97	98
107	0.97	0.94	0.95	98
108	0.97	0.99	0.98	147
109	0.80	0.97	0.88	98
110	0.99	0.85	0.91	98
111	0.92	0.98	0.95	523
112	0.92	0.94	0.93	98
113	0.94	0.88	0.91	136
114	1.00	0.97	0.99	73
115	0.94	0.95	0.94	94

```
          116       0.87      0.95      0.91        91
          117       0.99      1.00      0.99       147
          118       1.00      0.94      0.97        95
          119       1.00      0.88      0.93        48
          120       0.98      0.98      0.98        48

     accuracy                           0.94     14061
    macro avg       0.95      0.94      0.94     14061
 weighted avg       0.94      0.94      0.94     14061
```

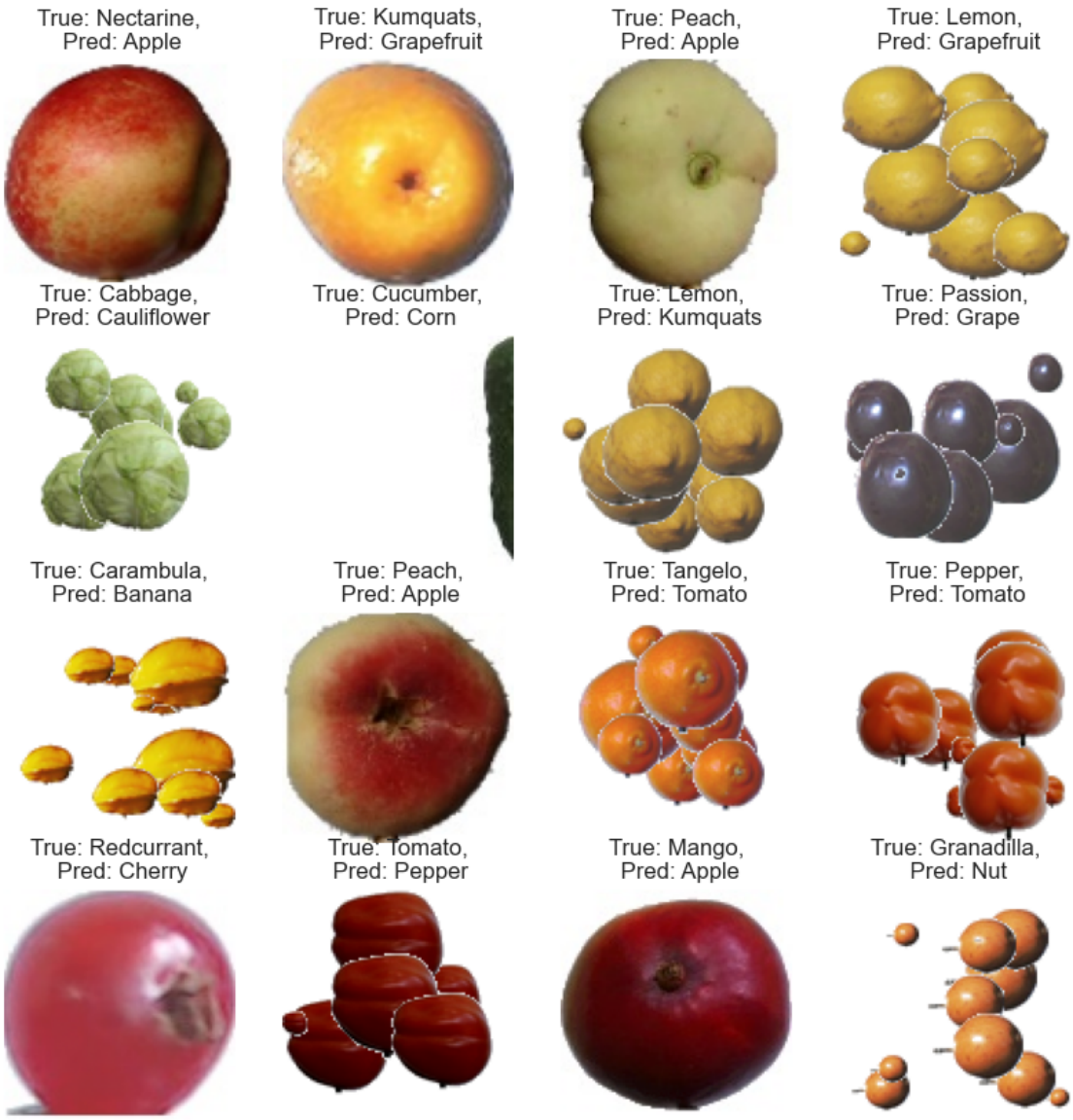
[ ]:
```python
# Report per la prima task (Frutto)
print("  Classification Report - Y1 (Frutto):")
y1_report = classification_report(y1_test_classes, y1_pred_classes,␣
 ↪output_dict=True)
print(f"Precision: {y1_report['weighted avg']['precision']:.4f}")
print(f"Recall: {y1_report['weighted avg']['recall']:.4f}")
print(f"F1-Score: {y1_report['weighted avg']['f1-score']:.4f}")

# Report per la seconda task (Qualità)
print("  Classification Report - Y2 (Qualità):")
y2_report = classification_report(y2_test_classes, y2_pred_classes,␣
 ↪output_dict=True)
print(f"Precision: {y2_report['weighted avg']['precision']:.4f}")
print(f"Recall: {y2_report['weighted avg']['recall']:.4f}")
print(f"F1-Score: {y2_report['weighted avg']['f1-score']:.4f}")
```

```
  Classification Report - Y1 (Frutto):
Precision: 0.9400
Recall: 0.9371
F1-Score: 0.9365
  Classification Report - Y2 (Qualità):
Precision: 0.9422
Recall: 0.9408
F1-Score: 0.9404
```

Visualizzazione errori su test interno

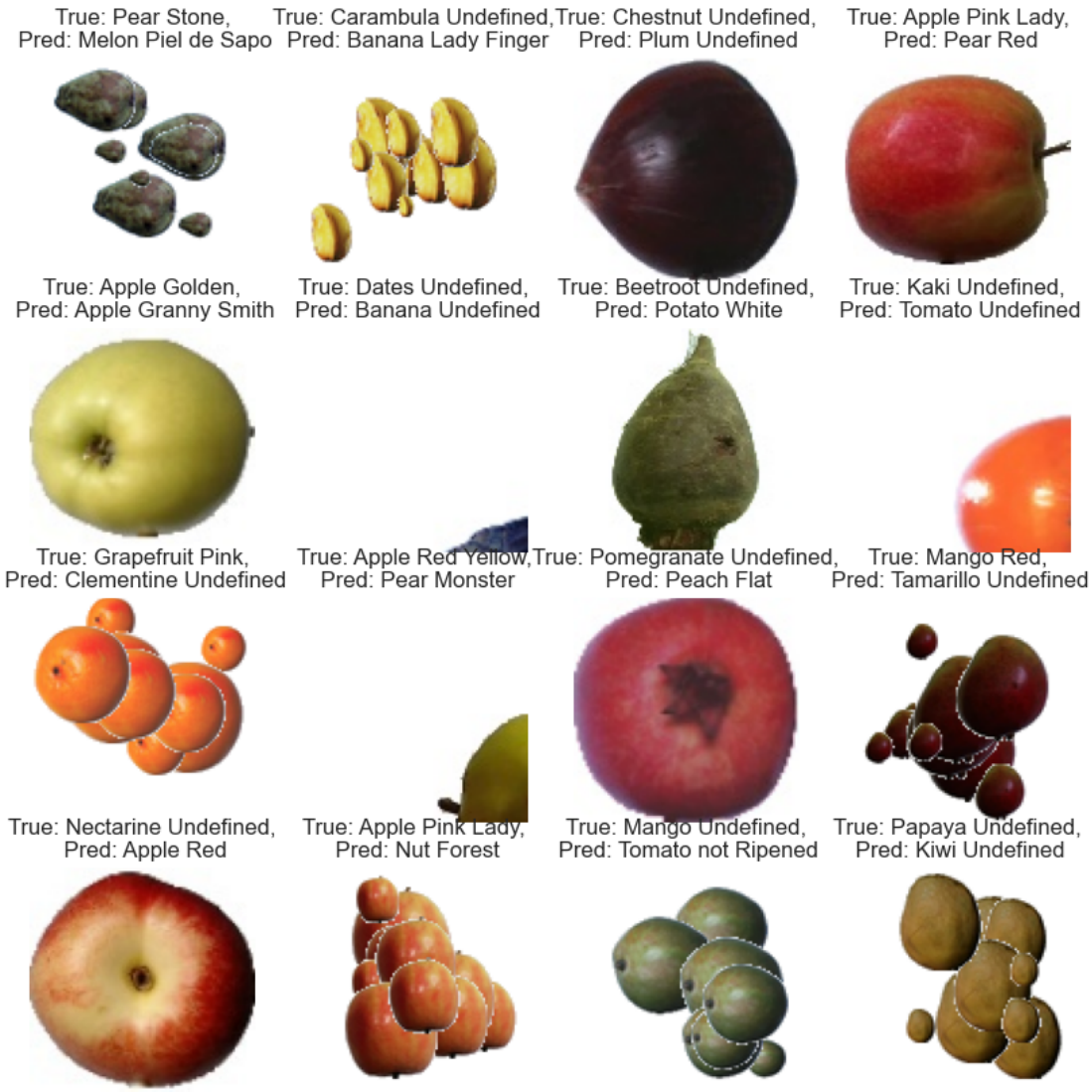
[ ]:
```python
def deprocess_resnet101(img):
    """Inverti la normalizzazione della ResNet101."""
    img = img.copy()
    img += np.array([103.939, 116.779, 123.68])  # Riaggiungi il valore medio
    img = img[..., ::-1]  # Converti da BGR a RGB
    return np.clip(img, 0, 255).astype(np.uint8)  # Clip e conversione a uint8

# Esempio di utilizzo
x_train_original = deprocess_resnet101(x_train_preprocessed)
x_val_original = deprocess_resnet101(x_val_preprocessed)
x_test_original = deprocess_resnet101(x_test_preprocessed)
```

[ ]:
```
# Trova gli indici degli errori
wrong_indices = np.where(y1_pred_classes != y1_test_classes)[0]

# Seleziona fino a 16 immagini casuali dagli errori
num_samples = min(16, len(wrong_indices))  # Evita errori se ci sono meno di 16␣
 ↪errori
random_wrong_indices = np.random.choice(wrong_indices, num_samples,␣
 ↪replace=False)

# Mostra le immagini errate
fig, axes = plt.subplots(4, 4, figsize=(10, 10))

for i, ax in enumerate(axes.flat):
    if i < num_samples:
        idx = random_wrong_indices[i]

        # Controllo formato e normalizzazione
        img = np.clip(x_test_original[idx]/255, 0, 1)  # Assicura che plt.
 ↪imshow() funzioni bene

        ax.imshow(img)
        ax.set_title(f"True: {labels_1_array[y1_test_classes[idx]]},\n Pred:␣
 ↪{labels_1_array[y1_pred_classes[idx]]}")
    ax.axis("off")  # Nasconde gli assi per estetica

plt.show()
```



[ ]:
```
# Trova gli indici degli errori
wrong_indices = np.where(y2_pred_classes != y2_test_classes)[0]

# Seleziona fino a 16 immagini casuali dagli errori
num_samples = min(16, len(wrong_indices))  # Evita errori se ci sono meno di 16␣
 ↪errori
random_wrong_indices = np.random.choice(wrong_indices, num_samples,␣
 ↪replace=False)

# Mostra le immagini errate
fig, axes = plt.subplots(4, 4, figsize=(10, 10))
```

```
for i, ax in enumerate(axes.flat):
    if i < num_samples:
        idx = random_wrong_indices[i]

        # Controllo formato e normalizzazione
        img = np.clip(x_test_original[idx] / 255, 0, 1) # Assicura che plt.
 ↪imshow() funzioni bene

        ax.imshow(img)
        ax.set_title(f"True: {labels_2_array[y2_test_classes[idx]]},\n Pred:␣
 ↪{labels_2_array[y2_pred_classes[idx]]}")
    ax.axis("off")  # Nasconde gli assi per estetica

plt.show()
```



Valutazione Test esterno

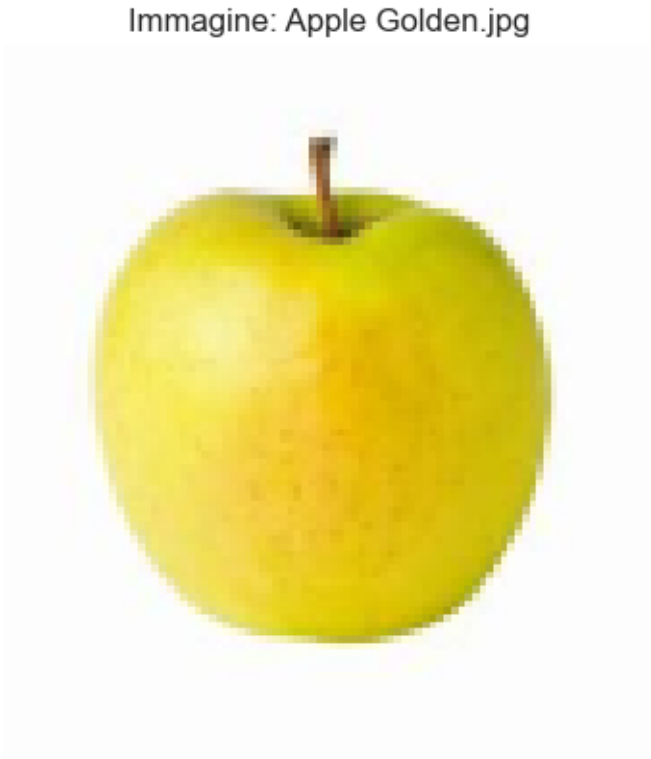
[ ]:
```
print("Ext Test: \n")
# Itera su tutte le immagini nella cartella
for nome_file in os.listdir(cartella_immagini):
    percorso_file = os.path.join(cartella_immagini, nome_file)

    # Controlla che sia un file immagine
    if not (nome_file.endswith(".jpg") or nome_file.endswith(".png") or␣
↪nome_file.endswith(".jpeg")):
        continue

    # Carica l'immagine
    img = cv2.imread(percorso_file)
    img_resized = resize_image_keep_aspect_ratio(img)
    resized_img_rgb = cv2.cvtColor(img_resized, cv2.COLOR_BGR2RGB)
    plt.imshow(img_resized / 255)
    plt.title(f"Immagine: {nome_file}")
    plt.axis("off")  # Rimuove gli assi per una visualizzazione più pulita
    plt.show()  # Mostra l'immagine immediatamente

    # Converti l'immagine in array numpy e applica il preprocessing di ResNet
    img_array = np.expand_dims(img_resized, axis=0).astype("float32")  #␣
↪Aggiungi batch dimension
    img_preprocessed = tf.keras.applications.resnet.preprocess_input(img_array)␣
↪ # Preprocess ResNet101

    # Predizione con il modello
    prediction = model.predict(img_preprocessed)
    # Supponiamo che `prediction` sia il risultato del modello

    # Estrai l'indice con la probabilità massima per entrambe le predizioni
    predicted_fruit_idx = np.argmax(prediction[0])  # Prima parte della␣
↪predizione
    predicted_quality_idx = np.argmax(prediction[1])  # Seconda parte della␣
↪predizione

    # Recupera i nomi dal dizionario o dalla lista
    predicted_fruit = labels_1_array[predicted_fruit_idx]
    predicted_quality = labels_2_array[predicted_quality_idx]

    # Stampa il risultato
    print(f"Immagine: {nome_file}, Predizione: {predicted_fruit} -␣
↪{predicted_quality}")
```

Immagine: Apple Golden 2.jpeg

```
1/1             1s 1s/step
Immagine: Apple Golden 2.jpeg, Predizione: Quince - Quince Undefined
```

Immagine: Apple Golden con sfondo.jpeg

```
1/1             0s 91ms/step
Immagine: Apple Golden con sfondo.jpeg, Predizione: Physalis - Maracuja
Undefined
```

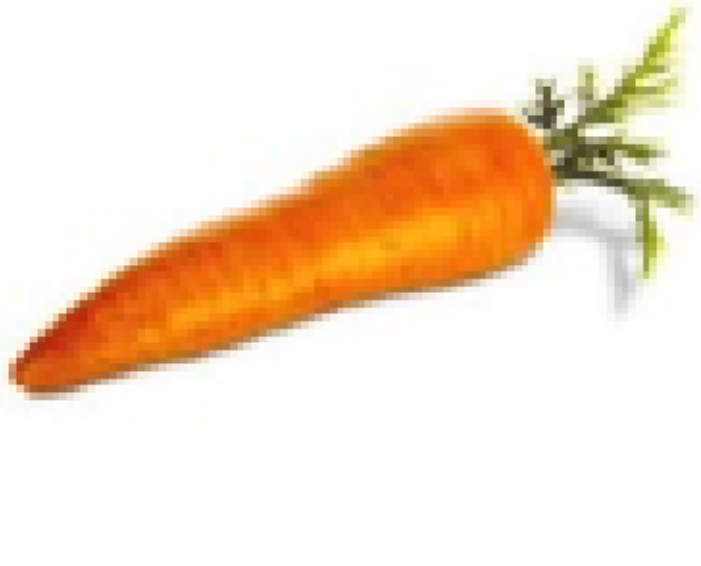



```
1/1             0s 90ms/step
Immagine: Apple Golden.jpg, Predizione: Apple - Pear Monster
```

Immagine: apple-braeburn.jpg

```
1/1             0s 101ms/step
Immagine: apple-braeburn.jpg, Predizione: Apple - Apple Red
```

Immagine: Carrot.jpg



```
1/1             0s 102ms/step
Immagine: Carrot.jpg, Predizione: Banana - Banana Undefined
```

Immagine: Cocos.jpg

```
1/1             0s 89ms/step
Immagine: Cocos.jpg, Predizione: Kiwi - Kiwi Undefined
```

Immagine: Cucumber.png

```
1/1             0s 77ms/step
Immagine: Cucumber.png, Predizione: Cucumber - Cucumber Undefined
```

Immagine: Dragon Fruit.jpg

```
1/1             0s 93ms/step
Immagine: Dragon Fruit.jpg, Predizione: Pitahaya - Pitahaya Red
```


```
1/1             0s 88ms/step
Immagine: Fig.jpg, Predizione: Apple - Apple Red Yellow
```

Immagine: Grape White.jpg

```
1/1             0s 87ms/step
Immagine: Grape White.jpg, Predizione: Apple - Apple Golden
```


```
1/1             0s 94ms/step
Immagine: Guava.jpg, Predizione: Apple - Apple Red Yellow
```

Immagine: kiwi.jpg

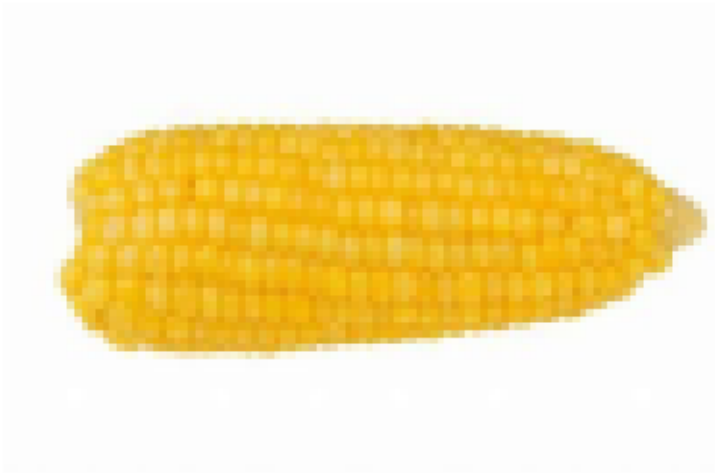
```
1/1             0s 79ms/step
Immagine: kiwi.jpg, Predizione: Pear - Pear Forelle
```

Immagine: Lemon.jpg

```
1/1             0s 81ms/step
Immagine: Lemon.jpg, Predizione: Lemon - Lemon Undefined
```


```
1/1             0s 80ms/step
Immagine: Lime.jpg, Predizione: Apple - Apple Red Yellow
```


```
1/1             0s 78ms/step
Immagine: Lychee.jpg, Predizione: Strawberry - Strawberry Wedge
```




```
1/1             0s 80ms/step
Immagine: mais.jpg, Predizione: Corn - Corn Undefined
```

Immagine: Onion White Peeled.jpg

```
1/1             0s 81ms/step
Immagine: Onion White Peeled.jpg, Predizione: Onion - Onion White
```

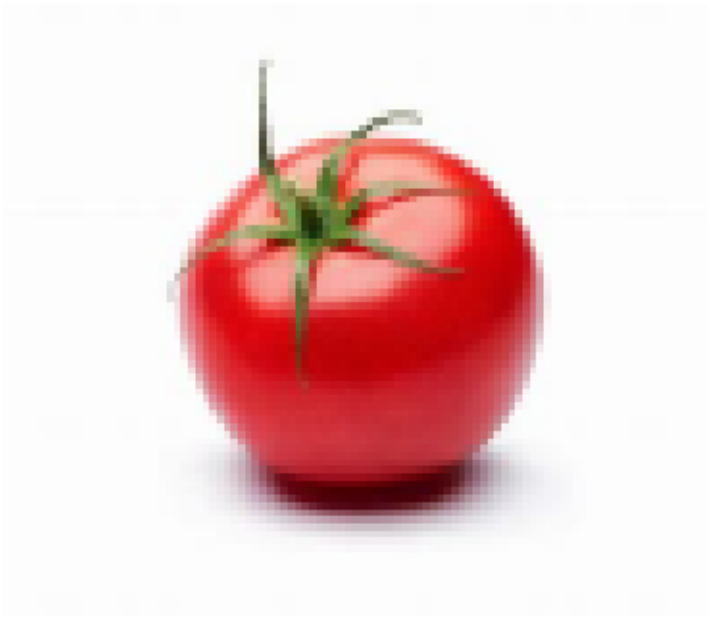
Immagine: Onion White.jpg

```
1/1             0s 83ms/step
Immagine: Onion White.jpg, Predizione: Granadilla - Clementine Undefined
```

Immagine: Pear Red.jpg

```
1/1             0s 81ms/step
Immagine: Pear Red.jpg, Predizione: Pear - Pear Red
```

Immagine: pomodoro.jpg



```
1/1             0s 79ms/step
Immagine: pomodoro.jpg, Predizione: Tomato - Tomato Undefined
```

Immagine: zucchina.jpg

```
1/1             0s 88ms/step
Immagine: zucchina.jpg, Predizione: Corn - Corn Husk
```

1.5.4 Modello 3: VGG19 + custom loss

Reload dateset

```
[ ]: x_train, y1_train, y2_train, x_val, y1_val, y2_val, x_test, y1_test, y2_test =␣
      ↪reaload_from_scratch()
```

Normalizzazione dati:

```
[ ]: # Preprocessing per train
     x_train = tf.keras.applications.vgg19.preprocess_input(x_train)

     # Preprocessing per validation
     x_val = tf.keras.applications.vgg19.preprocess_input(x_val)

     # Preprocessing per test
     x_test = tf.keras.applications.vgg19.preprocess_input(x_test)
```

Calcolo steps per epochs

```
[ ]: input_shape = x_train.shape[1:]
     num_classes_1 = len(y1_train[0])
```

```
num_classes_2 = len(y2_train[0])
```

[ ]:
```
dataset_size = len(x_train)  # Numero totale di campioni
epochs_number = 30
batch_size = 16

steps_per_epoch = int(1.5 * dataset_size / batch_size / epochs_number)
steps_per_val = int(len(x_val) / batch_size / epochs_number)

print(f"Steps per epoch: {steps_per_epoch}")
print(f"Steps per val: {steps_per_val}")
```

```
Steps per epoch: 176
Steps per val: 49
```

Implementazione VGG19

[ ]:
```
# Carica il modello VGG19 pre-addestrato
VGG_model = VGG19(
                weights='imagenet',
                include_top=False,
                input_shape=(100, 100, 3),
                input_tensor=None,
                pooling=None,
                classifier_activation="softmax"
            )
```

[ ]:
```
VGG_model.trainable = False  # Blocca i pesi del modello pre-addestrato
```

[ ]:
```
# Aggiungi strati personalizzati
x = GlobalAveragePooling2D()(VGG_model.output)
x = Dense(512, activation="relu", kernel_regularizer=l2(0.001))(x)
x = Dropout(0.2)(x)
x = Dense(256, activation="relu", kernel_regularizer=l2(0.001))(x)
x = Dropout(0.1)(x)
x = Dense(128, activation="relu", kernel_regularizer=l2(0.001))(x)

# Ramo 1 - Predizione del Frutto
fruit_output = Dense(num_classes_1, activation="softmax", name="y1")(x)

# Ramo 2 - Predizione della Qualità
quality_output = Dense(num_classes_2, activation="softmax", name="y2")(x)
```

[ ]:
```
# Crea il modello con doppia uscita
model = Model(inputs=VGG_model.input, outputs=[fruit_output, quality_output])
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100, 100, 3)	0	-
block1_conv1 (Conv2D)	(None, 100, 100, 64)	1,792	input_layer[0][0]
block1_conv2 (Conv2D)	(None, 100, 100, 64)	36,928	block1_conv1[0][…
block1_pool (MaxPooling2D)	(None, 50, 50, 64)	0	block1_conv2[0][…
block2_conv1 (Conv2D)	(None, 50, 50, 128)	73,856	block1_pool[0][0]
block2_conv2 (Conv2D)	(None, 50, 50, 128)	147,584	block2_conv1[0][…
block2_pool (MaxPooling2D)	(None, 25, 25, 128)	0	block2_conv2[0][…
block3_conv1 (Conv2D)	(None, 25, 25, 256)	295,168	block2_pool[0][0]
block3_conv2 (Conv2D)	(None, 25, 25, 256)	590,080	block3_conv1[0][…
block3_conv3 (Conv2D)	(None, 25, 25, 256)	590,080	block3_conv2[0][…
block3_conv4 (Conv2D)	(None, 25, 25, 256)	590,080	block3_conv3[0][…
block3_pool (MaxPooling2D)	(None, 12, 12, 256)	0	block3_conv4[0][…
block4_conv1 (Conv2D)	(None, 12, 12, 512)	1,180,160	block3_pool[0][0]
block4_conv2 (Conv2D)	(None, 12, 12, 512)	2,359,808	block4_conv1[0][…
block4_conv3 (Conv2D)	(None, 12, 12, 512)	2,359,808	block4_conv2[0][…

block4_conv4 (Conv2D)	(None, 12, 12, 512)	2,359,808	block4_conv3[0][…
block4_pool (MaxPooling2D)	(None, 6, 6, 512)	0	block4_conv4[0][…
block5_conv1 (Conv2D)	(None, 6, 6, 512)	2,359,808	block4_pool[0][0]
block5_conv2 (Conv2D)	(None, 6, 6, 512)	2,359,808	block5_conv1[0][…
block5_conv3 (Conv2D)	(None, 6, 6, 512)	2,359,808	block5_conv2[0][…
block5_conv4 (Conv2D)	(None, 6, 6, 512)	2,359,808	block5_conv3[0][…
block5_pool (MaxPooling2D)	(None, 3, 3, 512)	0	block5_conv4[0][…
global_average_poo… (GlobalAveragePool…	(None, 512)	0	block5_pool[0][0]
dense (Dense)	(None, 512)	262,656	global_average_p…
dropout (Dropout)	(None, 512)	0	dense[0][0]
dense_1 (Dense)	(None, 256)	131,328	dropout[0][0]
dropout_1 (Dropout)	(None, 256)	0	dense_1[0][0]
dense_2 (Dense)	(None, 128)	32,896	dropout_1[0][0]
y1 (Dense)	(None, 70)	9,030	dense_2[0][0]
y2 (Dense)	(None, 121)	15,609	dense_2[0][0]

Total params: 20,475,903 (78.11 MB)

Trainable params: 451,519 (1.72 MB)

Non-trainable params: 20,024,384 (76.39 MB)

Loss custom

[ ]:
```
from tensorflow.keras.saving import register_keras_serializable

@register_keras_serializable()
def loss_fn(y_true, y_pred, alpha=1.5):
    """
    Loss function that penalizes inconsistent predictions between label1 and␣
 ↪label2.
    """
    y1_classes = y_true.shape[1] // 2  # Supponiamo che y_true abbia tutte le␣
 ↪classi concatenate

    # Separiamo le etichette principali (y1) e secondarie (y2)
    y1_true, y2_true = y_true[:, :y1_classes], y_true[:, y1_classes:]
    y1_pred, y2_pred = y_pred[:, :y1_classes], y_pred[:, y1_classes:]

    # Standard categorical cross-entropy loss per entrambe le etichette
    loss1 = tf.keras.losses.CategoricalCrossentropy()(y1_true, y1_pred)
    loss2 = tf.keras.losses.CategoricalCrossentropy()(y2_true, y2_pred)

    # Penalità per inconsistenza tra y1 e y2
    y1_y2_mismatch = tf.cast(tf.not_equal(tf.argmax(y1_pred, axis=-1), tf.
 ↪argmax(y2_pred, axis=-1)), tf.float32)
    inconsistency_penalty = alpha * tf.reduce_mean(y1_y2_mismatch)

    # Loss finale
    return loss1 + loss2 + inconsistency_penalty
```

Train del modello

[ ]:
```
# Definisci il numero di classi
num_classes_1 = len(y1_train[0])  # Classi dei frutti
num_classes_2 = len(y2_train[0])  # Classi della qualità

# Compila il modello
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),

    loss=loss_fn,

    metrics={
        'y1': 'accuracy',
        'y2': 'accuracy'
        }
)
```

[ ]:
```
if(os.path.exists('keras/model-VGG-AUG.keras')):
    model = load_model('keras/model-VGG-AUG.keras')
```

```
else:
    history = model.fit(
        x_train,
        {'y1': y1_train,
        'y2': y2_train},
        validation_data=(
            x_val, {
                'y1': y1_val,
                'y2': y2_val}),
        epochs=epochs_number,
        batch_size=batch_size,
        steps_per_epoch=steps_per_epoch,
        validation_steps=steps_per_val
    )
```

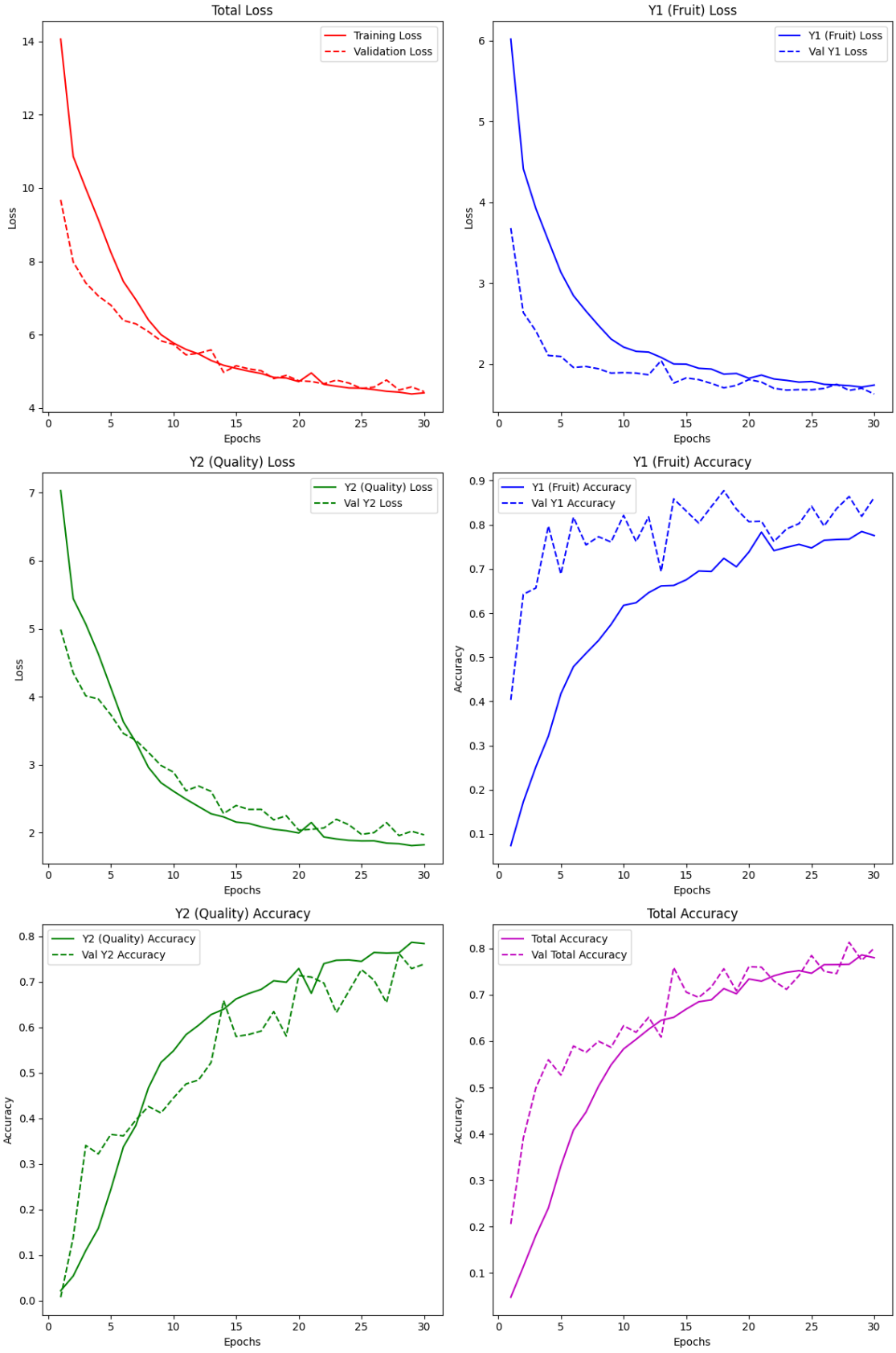
```
Epoch 1/30
176/176             96s 529ms/step -
loss: 17.0573 - y1_accuracy: 0.0469 - y1_loss: 7.5200 - y2_accuracy: 0.0153 -
y2_loss: 8.5212 - val_loss: 9.6767 - val_y1_accuracy: 0.4035 - val_y1_loss:
3.6811 - val_y2_accuracy: 0.0077 - val_y2_loss: 4.9884
Epoch 2/30
176/176             75s 429ms/step -
loss: 11.1300 - y1_accuracy: 0.1348 - y1_loss: 4.5915 - y2_accuracy: 0.0433 -
y2_loss: 5.5333 - val_loss: 7.9860 - val_y1_accuracy: 0.6425 - val_y1_loss:
2.6374 - val_y2_accuracy: 0.1393 - val_y2_loss: 4.3493
Epoch 3/30
176/176             76s 434ms/step -
loss: 10.1869 - y1_accuracy: 0.2331 - y1_loss: 4.0041 - y2_accuracy: 0.0989 -
y2_loss: 5.1853 - val_loss: 7.4142 - val_y1_accuracy: 0.6568 - val_y1_loss:
2.4097 - val_y2_accuracy: 0.3410 - val_y2_loss: 4.0123
Epoch 4/30
176/176             76s 431ms/step -
loss: 9.4381 - y1_accuracy: 0.3039 - y1_loss: 3.6267 - y2_accuracy: 0.1312 -
y2_loss: 4.8210 - val_loss: 7.0587 - val_y1_accuracy: 0.7971 - val_y1_loss:
2.1071 - val_y2_accuracy: 0.3224 - val_y2_loss: 3.9659
Epoch 5/30
176/176             75s 428ms/step -
loss: 8.4134 - y1_accuracy: 0.4105 - y1_loss: 3.2149 - y2_accuracy: 0.2142 -
y2_loss: 4.2144 - val_loss: 6.8079 - val_y1_accuracy: 0.6886 - val_y1_loss:
2.0947 - val_y2_accuracy: 0.3651 - val_y2_loss: 3.7336
Epoch 6/30
176/176             75s 428ms/step -
loss: 7.6402 - y1_accuracy: 0.4622 - y1_loss: 2.9112 - y2_accuracy: 0.3056 -
y2_loss: 3.7509 - val_loss: 6.3902 - val_y1_accuracy: 0.8169 - val_y1_loss:
1.9583 - val_y2_accuracy: 0.3618 - val_y2_loss: 3.4581
Epoch 7/30
176/176             76s 431ms/step -
loss: 7.1189 - y1_accuracy: 0.5002 - y1_loss: 2.7045 - y2_accuracy: 0.3614 -
```

```
y2_loss: 3.4419 - val_loss: 6.2978 - val_y1_accuracy: 0.7544 - val_y1_loss:
1.9710 - val_y2_accuracy: 0.3969 - val_y2_loss: 3.3584
Epoch 8/30
176/176            76s 433ms/step -
loss: 6.5454 - y1_accuracy: 0.5264 - y1_loss: 2.5399 - y2_accuracy: 0.4450 -
y2_loss: 3.0385 - val_loss: 6.0863 - val_y1_accuracy: 0.7730 - val_y1_loss:
1.9427 - val_y2_accuracy: 0.4265 - val_y2_loss: 3.1806
Epoch 9/30
176/176            76s 434ms/step -
loss: 6.0657 - y1_accuracy: 0.5604 - y1_loss: 2.3246 - y2_accuracy: 0.5164 -
y2_loss: 2.7794 - val_loss: 5.8352 - val_y1_accuracy: 0.7610 - val_y1_loss:
1.8892 - val_y2_accuracy: 0.4123 - val_y2_loss: 2.9883
Epoch 10/30
176/176            76s 433ms/step -
loss: 5.8797 - y1_accuracy: 0.6241 - y1_loss: 2.2270 - y2_accuracy: 0.5294 -
y2_loss: 2.6963 - val_loss: 5.7378 - val_y1_accuracy: 0.8213 - val_y1_loss:
1.8952 - val_y2_accuracy: 0.4452 - val_y2_loss: 2.8900
Epoch 11/30
176/176            76s 433ms/step -
loss: 5.6484 - y1_accuracy: 0.6060 - y1_loss: 2.1696 - y2_accuracy: 0.5737 -
y2_loss: 2.5276 - val_loss: 5.4523 - val_y1_accuracy: 0.7621 - val_y1_loss:
1.8895 - val_y2_accuracy: 0.4759 - val_y2_loss: 2.6156
Epoch 12/30
176/176            76s 432ms/step -
loss: 5.4703 - y1_accuracy: 0.6366 - y1_loss: 2.1452 - y2_accuracy: 0.6002 -
y2_loss: 2.3792 - val_loss: 5.4955 - val_y1_accuracy: 0.8180 - val_y1_loss:
1.8679 - val_y2_accuracy: 0.4846 - val_y2_loss: 2.6857
Epoch 13/30
176/176            76s 433ms/step -
loss: 5.2974 - y1_accuracy: 0.6463 - y1_loss: 2.0767 - y2_accuracy: 0.6165 -
y2_loss: 2.2800 - val_loss: 5.5874 - val_y1_accuracy: 0.6941 - val_y1_loss:
2.0440 - val_y2_accuracy: 0.5230 - val_y2_loss: 2.6066
Epoch 14/30
176/176            76s 433ms/step -
loss: 5.1798 - y1_accuracy: 0.6528 - y1_loss: 1.9986 - y2_accuracy: 0.6311 -
y2_loss: 2.2458 - val_loss: 4.9773 - val_y1_accuracy: 0.8586 - val_y1_loss:
1.7658 - val_y2_accuracy: 0.6590 - val_y2_loss: 2.2802
Epoch 15/30
176/176            76s 433ms/step -
loss: 5.1112 - y1_accuracy: 0.6762 - y1_loss: 2.0059 - y2_accuracy: 0.6714 -
y2_loss: 2.1751 - val_loss: 5.1555 - val_y1_accuracy: 0.8311 - val_y1_loss:
1.8297 - val_y2_accuracy: 0.5800 - val_y2_loss: 2.3996
Epoch 16/30
176/176            76s 433ms/step -
loss: 5.0292 - y1_accuracy: 0.6980 - y1_loss: 1.9565 - y2_accuracy: 0.6799 -
y2_loss: 2.1478 - val_loss: 5.0709 - val_y1_accuracy: 0.8037 - val_y1_loss:
1.8082 - val_y2_accuracy: 0.5844 - val_y2_loss: 2.3418
Epoch 17/30
```

```
176/176            76s 432ms/step -
loss: 4.8949 - y1_accuracy: 0.6944 - y1_loss: 1.9219 - y2_accuracy: 0.6946 -
y2_loss: 2.0533 - val_loss: 5.0200 - val_y1_accuracy: 0.8410 - val_y1_loss:
1.7620 - val_y2_accuracy: 0.5921 - val_y2_loss: 2.3423
Epoch 18/30
176/176            76s 430ms/step -
loss: 4.8343 - y1_accuracy: 0.7393 - y1_loss: 1.8811 - y2_accuracy: 0.7046 -
y2_loss: 2.0388 - val_loss: 4.8049 - val_y1_accuracy: 0.8772 - val_y1_loss:
1.7071 - val_y2_accuracy: 0.6349 - val_y2_loss: 2.1874
Epoch 19/30
176/176            75s 429ms/step -
loss: 4.8893 - y1_accuracy: 0.6918 - y1_loss: 1.9185 - y2_accuracy: 0.6988 -
y2_loss: 2.0618 - val_loss: 4.8906 - val_y1_accuracy: 0.8355 - val_y1_loss:
1.7370 - val_y2_accuracy: 0.5811 - val_y2_loss: 2.2485
Epoch 20/30
176/176            76s 431ms/step -
loss: 4.6871 - y1_accuracy: 0.7411 - y1_loss: 1.8210 - y2_accuracy: 0.7383 -
y2_loss: 1.9624 - val_loss: 4.7460 - val_y1_accuracy: 0.8070 - val_y1_loss:
1.8096 - val_y2_accuracy: 0.7138 - val_y2_loss: 2.0367
Epoch 21/30
  8/176            50s 301ms/step - loss:
5.2514 - y1_accuracy: 0.7565 - y1_loss: 2.0509 - y2_accuracy: 0.6589 - y2_loss:
2.2950

/Users/lucaperfetti/Desktop/università/Secondo Anno/Advanced
ML/Project/.venv/lib/python3.10/site-
packages/keras/src/trainers/epoch_iterator.py:107: UserWarning: Your input ran
out of data; interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to use the
`.repeat()` function when building your dataset.
  self._interrupted_warning()

176/176            21s 116ms/step -
loss: 4.9723 - y1_accuracy: 0.7821 - y1_loss: 1.8729 - y2_accuracy: 0.6743 -
y2_loss: 2.1557 - val_loss: 4.7265 - val_y1_accuracy: 0.8081 - val_y1_loss:
1.7784 - val_y2_accuracy: 0.7105 - val_y2_loss: 2.0487
Epoch 22/30
176/176            75s 426ms/step -
loss: 4.6851 - y1_accuracy: 0.7243 - y1_loss: 1.8329 - y2_accuracy: 0.7289 -
y2_loss: 1.9540 - val_loss: 4.6636 - val_y1_accuracy: 0.7621 - val_y1_loss:
1.7009 - val_y2_accuracy: 0.6974 - val_y2_loss: 2.0684
Epoch 23/30
176/176            75s 426ms/step -
loss: 4.5975 - y1_accuracy: 0.7550 - y1_loss: 1.7977 - y2_accuracy: 0.7443 -
y2_loss: 1.9069 - val_loss: 4.7641 - val_y1_accuracy: 0.7906 - val_y1_loss:
1.6792 - val_y2_accuracy: 0.6327 - val_y2_loss: 2.1959
Epoch 24/30
176/176            75s 427ms/step -
loss: 4.5359 - y1_accuracy: 0.7664 - y1_loss: 1.7583 - y2_accuracy: 0.7549 -
```

```
y2_loss: 1.8900 - val_loss: 4.6836 - val_y1_accuracy: 0.8026 - val_y1_loss:
1.6851 - val_y2_accuracy: 0.6798 - val_y2_loss: 2.1151
Epoch 25/30
176/176            76s 431ms/step -
loss: 4.5785 - y1_accuracy: 0.7516 - y1_loss: 1.8041 - y2_accuracy: 0.7381 -
y2_loss: 1.8922 - val_loss: 4.5361 - val_y1_accuracy: 0.8421 - val_y1_loss:
1.6829 - val_y2_accuracy: 0.7270 - val_y2_loss: 1.9749
Epoch 26/30
176/176            75s 426ms/step -
loss: 4.5521 - y1_accuracy: 0.7649 - y1_loss: 1.7669 - y2_accuracy: 0.7608 -
y2_loss: 1.9084 - val_loss: 4.5721 - val_y1_accuracy: 0.7971 - val_y1_loss:
1.6999 - val_y2_accuracy: 0.7039 - val_y2_loss: 1.9995
Epoch 27/30
176/176            74s 418ms/step -
loss: 4.4920 - y1_accuracy: 0.7584 - y1_loss: 1.7471 - y2_accuracy: 0.7481 -
y2_loss: 1.8735 - val_loss: 4.7661 - val_y1_accuracy: 0.8366 - val_y1_loss:
1.7511 - val_y2_accuracy: 0.6546 - val_y2_loss: 2.1476
Epoch 28/30
176/176            74s 423ms/step -
loss: 4.4349 - y1_accuracy: 0.7649 - y1_loss: 1.7323 - y2_accuracy: 0.7592 -
y2_loss: 1.8364 - val_loss: 4.4939 - val_y1_accuracy: 0.8640 - val_y1_loss:
1.6757 - val_y2_accuracy: 0.7621 - val_y2_loss: 1.9561
Epoch 29/30
176/176            74s 419ms/step -
loss: 4.3999 - y1_accuracy: 0.7766 - y1_loss: 1.7338 - y2_accuracy: 0.7837 -
y2_loss: 1.8054 - val_loss: 4.5784 - val_y1_accuracy: 0.8191 - val_y1_loss:
1.7018 - val_y2_accuracy: 0.7292 - val_y2_loss: 2.0199
Epoch 30/30
176/176            73s 418ms/step -
loss: 4.4001 - y1_accuracy: 0.7724 - y1_loss: 1.7416 - y2_accuracy: 0.7935 -
y2_loss: 1.8032 - val_loss: 4.4496 - val_y1_accuracy: 0.8618 - val_y1_loss:
1.6308 - val_y2_accuracy: 0.7390 - val_y2_loss: 1.9676
```

[ ]:
```
# Dopo l'addestramento del modello
plot_training_history(history)
```


Total Loss
Training Loss
Validation Loss
Loss
Epochs
Y1 (Fruit) Loss
Y1 (Fruit) Loss
Val Y1 Loss
Loss
Epochs
Y2 (Quality) Loss
Y2 (Quality) Loss
Val Y2 Loss
Loss
Epochs
Y1 (Fruit) Accuracy
Y1 (Fruit) Accuracy
Val Y1 Accuracy
Accuracy
Epochs
Y2 (Quality) Accuracy
Y2 (Quality) Accuracy
Val Y2 Accuracy
Accuracy
Epochs
Total Accuracy
Total Accuracy
Val Total Accuracy
Accuracy
Epochs

Salvataggio in memoria

[ ]:
```
model.save("keras/model-VGG-AUG.keras")
```

[ ]:
```
model = tf.keras.models.load_model(
    "keras/model-VGG-AUG.keras",
    custom_objects={"loss_fn": loss_fn}
)
```

Valutazione metriche di classificazione

[ ]:
```
y1_pred, y2_pred = model.predict(x_test)

y1_pred_classes = np.argmax(y1_pred, axis=1)
y2_pred_classes = np.argmax(y2_pred, axis=1)

y1_test_classes = np.argmax(y1_test, axis=1)
y2_test_classes = np.argmax(y2_test, axis=1)
```

```
440/440             281s 638ms/step
```

[ ]:
```
# Valutazione del modello sul test set
loss, loss_y1, loss_y2, acc_y1, acc_y2 = model.evaluate(x_test, {"y1": y1_test,␣
 ↪"y2": y2_test}, verbose=0)

print(f"Loss Totale: {loss:.4f}")
print(f"Loss Y1 (Fruit): {loss_y1:.4f}")
print(f"Loss Y2 (Quality): {loss_y2:.4f}")
print(f"Accuracy Y1 (Fruit): {acc_y1:.4f}")
print(f"Accuracy Y2 (Quality): {acc_y2:.4f}")
```

```
Loss Totale: 4.1946
Loss Y1 (Fruit): 1.6351
Loss Y2 (Quality): 1.7078
Accuracy Y1 (Fruit): 0.8452
Accuracy Y2 (Quality): 0.8489
```

[ ]:
```
#F1-score
y_pred_m1 = (y1_pred > 0.5).astype(int)

f1 = f1_score(y1_test, y_pred_m1, average="weighted")
print(f'F1 Score: {f1}')
```

```
F1 Score: 0.8432390439141705
```

[ ]:
```
#F1-score
y_pred_m2 = (y2_pred > 0.5).astype(int)
```

```
f2 = f1_score(y2_test, y_pred_m2, average="weighted")
print(f'F1 Score: {f2}')
```

```
F1 Score: 0.8432510570633491
```

[ ]:
```
print("Classification Report per Y1 (Frutto):")
print(classification_report(y1_test_classes, y1_pred_classes))

print("Classification Report per Y2 (Qualità):")
print(classification_report(y2_test_classes, y2_pred_classes))
```

```
Classification Report per Y1 (Frutto):
              precision    recall  f1-score   support

           0       0.71      0.92      0.80      1510
           1       0.83      0.96      0.89        98
           2       0.98      0.81      0.89       183
           3       0.98      0.96      0.97       286
           4       1.00      0.09      0.16        90
           5       0.85      0.25      0.39        92
           6       1.00      0.71      0.83        28
           7       1.00      0.50      0.67        98
           8       0.99      0.94      0.96       196
           9       0.90      0.88      0.89        98
          10       1.00      0.97      0.98        30
          11       0.99      0.94      0.96       140
          12       0.81      0.87      0.84       686
          13       0.71      0.88      0.78        90
          14       0.94      0.93      0.93        98
          15       1.00      0.87      0.93        98
          16       1.00      0.86      0.93       182
          17       0.92      0.87      0.89       249
          18       1.00      0.97      0.98        98
          19       0.99      0.75      0.85       141
          20       0.90      0.89      0.89       140
          21       0.94      0.75      0.83        59
          22       0.90      0.72      0.80        98
          23       0.96      0.62      0.75       682
          24       0.85      0.91      0.88       196
          25       0.99      0.70      0.82       100
          26       0.89      0.92      0.90        92
          27       1.00      0.94      0.97        98
          28       0.92      0.90      0.91        98
          29       0.93      0.72      0.81        93
          30       1.00      0.84      0.91        94
          31       0.86      0.84      0.85        98
          32       0.85      0.76      0.80       196
          33       0.96      0.78      0.86        98
```

34	0.97	1.00	0.98	98
35	1.00	0.85	0.92	98
36	0.88	0.67	0.76	183
37	0.94	0.85	0.89	60
38	0.96	0.94	0.95	98
39	0.83	0.99	0.90	147
40	0.96	0.99	0.97	98
41	0.88	0.32	0.47	194
42	0.88	0.81	0.84	236
43	0.96	0.70	0.81	266
44	0.71	0.96	0.82	95
45	0.78	0.70	0.74	98
46	0.86	0.88	0.87	98
47	0.81	0.70	0.75	343
48	0.75	0.93	0.83	1049
49	0.99	0.92	0.95	98
50	0.94	0.95	0.95	494
51	0.89	0.94	0.91	196
52	0.96	0.99	0.98	196
53	0.99	0.99	0.99	98
54	0.47	0.88	0.61	353
55	0.66	0.53	0.59	98
56	0.89	0.86	0.87	90
57	0.82	0.82	0.82	360
58	0.92	0.87	0.89	98
59	0.98	0.99	0.98	98
60	1.00	0.96	0.98	98
61	0.94	0.69	0.80	98
62	0.92	0.93	0.92	98
63	0.95	0.97	0.96	245
64	0.93	0.77	0.84	98
65	0.93	0.81	0.86	98
66	0.96	0.86	0.90	1015
67	0.91	0.98	0.94	147
68	0.89	1.00	0.94	95
69	0.70	0.98	0.81	96
accuracy			0.85	14061
macro avg	0.90	0.83	0.85	14061
weighted avg	0.87	0.85	0.84	14061

Classification Report per Y2 (Qualità):

	precision	recall	f1-score	support
0	0.97	0.76	0.85	94
1	0.70	0.78	0.74	98
2	0.72	0.84	0.77	88

3	0.89	0.91	0.90	290
4	0.91	0.87	0.89	98
5	0.84	0.85	0.85	140
6	0.58	0.76	0.66	91
7	0.56	0.90	0.69	281
8	0.70	0.89	0.78	98
9	0.66	0.88	0.76	232
10	0.88	0.83	0.85	98
11	0.97	0.81	0.88	85
12	0.99	0.98	0.98	98
13	0.96	0.90	0.93	98
14	0.93	0.92	0.93	90
15	0.85	0.96	0.90	98
16	0.95	0.77	0.85	90
17	1.00	0.36	0.53	92
18	0.96	0.89	0.93	28
19	0.94	0.81	0.87	98
20	0.98	0.88	0.92	196
21	0.98	0.83	0.90	98
22	1.00	0.97	0.98	30
23	1.00	0.39	0.56	140
24	0.84	0.79	0.81	245
25	0.95	0.88	0.91	147
26	0.91	0.98	0.95	98
27	0.91	0.88	0.89	98
28	0.82	0.90	0.86	98
29	0.68	0.82	0.74	90
30	0.96	0.93	0.94	98
31	0.98	0.90	0.94	98
32	0.99	0.93	0.96	90
33	0.97	0.70	0.81	92
34	0.94	0.86	0.90	78
35	0.92	0.86	0.89	171
36	0.72	0.96	0.82	98
37	0.96	0.88	0.92	93
38	1.00	0.94	0.97	48
39	0.75	0.92	0.82	140
40	0.94	0.56	0.70	59
41	0.95	0.73	0.83	98
42	0.90	0.57	0.70	196
43	0.92	0.86	0.89	98
44	0.97	0.93	0.95	388
45	0.85	0.84	0.84	98
46	1.00	0.62	0.77	98
47	0.93	0.86	0.90	100
48	0.95	0.78	0.86	92
49	1.00	0.87	0.93	98
50	0.97	0.85	0.90	98

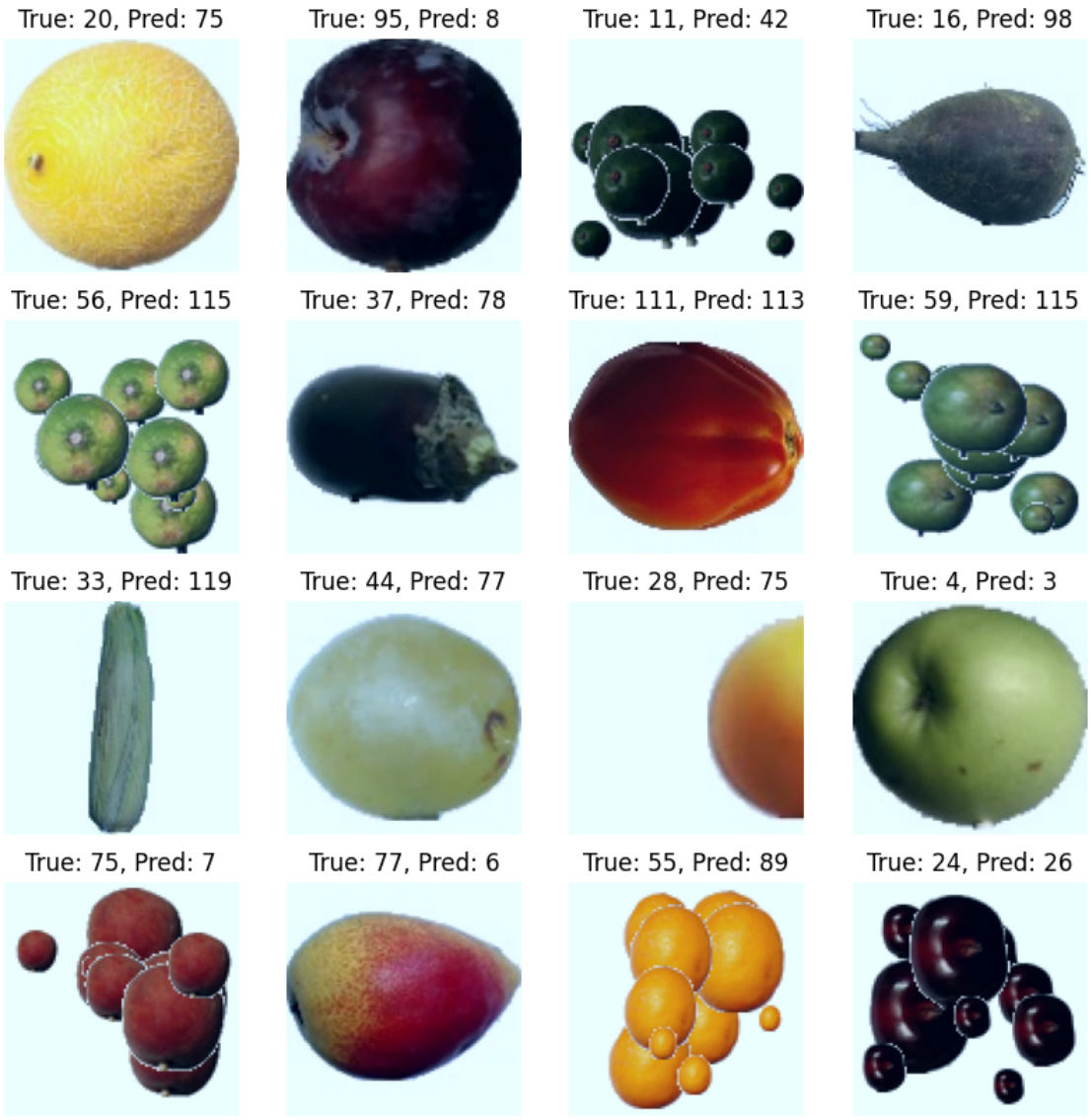
51	0.93	0.83	0.88	93
52	0.99	0.94	0.96	94
53	1.00	0.61	0.76	98
54	0.80	0.81	0.80	98
55	0.94	0.86	0.90	98
56	0.94	0.85	0.89	98
57	0.99	0.99	0.99	98
58	0.99	0.82	0.89	98
59	0.98	0.84	0.90	98
60	0.77	0.71	0.74	85
61	0.98	0.68	0.80	60
62	0.73	0.96	0.83	98
63	0.87	0.95	0.91	147
64	0.99	0.88	0.93	98
65	0.94	0.30	0.45	98
66	0.81	0.73	0.77	96
67	0.95	0.78	0.86	130
68	0.72	0.92	0.81	106
69	0.96	0.28	0.43	90
70	1.00	0.76	0.87	89
71	0.94	0.52	0.67	87
72	0.72	0.99	0.84	95
73	0.69	0.72	0.71	98
74	0.95	0.84	0.89	98
75	0.84	0.84	0.84	245
76	0.82	0.73	0.77	98
77	0.66	0.89	0.76	280
78	0.80	0.99	0.88	98
79	0.76	0.77	0.76	140
80	0.96	0.80	0.87	60
81	0.81	0.94	0.87	98
82	0.91	0.89	0.90	133
83	0.95	0.91	0.93	142
84	0.94	0.79	0.86	98
85	0.99	0.94	0.96	98
86	0.99	0.97	0.98	88
87	0.94	0.91	0.93	140
88	0.88	0.95	0.92	133
89	0.89	0.98	0.94	133
90	0.97	0.79	0.87	98
91	0.86	0.97	0.91	98
92	0.51	0.98	0.67	98
93	0.99	0.94	0.96	98
94	0.98	0.99	0.98	98
95	0.59	0.83	0.69	353
96	0.85	0.48	0.61	98
97	0.85	0.98	0.91	90
98	0.91	0.71	0.80	90

99	0.95	0.86	0.90	90
100	0.77	0.97	0.86	90
101	0.83	0.87	0.85	90
102	0.90	0.90	0.90	98
103	0.98	0.99	0.98	98
104	1.00	0.78	0.87	98
105	0.87	0.96	0.91	98
106	0.96	0.91	0.93	98
107	0.99	0.93	0.96	98
108	0.99	0.96	0.97	147
109	0.80	0.90	0.85	98
110	0.69	0.94	0.80	98
111	0.89	0.92	0.90	523
112	0.97	0.88	0.92	98
113	0.89	0.91	0.90	136
114	1.00	0.95	0.97	73
115	0.82	0.95	0.88	94
116	0.74	0.82	0.78	91
117	0.93	0.98	0.95	147
118	0.89	0.99	0.94	95
119	0.66	0.94	0.78	48
120	0.76	0.98	0.85	48
accuracy			0.85	14061
macro avg	0.89	0.84	0.85	14061
weighted avg	0.87	0.85	0.85	14061

Visualizzazione errori su test interno Si visualizzano su quali immagini la CNN fa' una predizione errata:

[ ]:
```
# Trova gli indici degli errori
wrong_indices = np.where(y2_pred_classes != y2_test_classes)[0]

# Seleziona fino a 16 immagini casuali dagli errori
num_samples = min(16, len(wrong_indices))  # Evita errori se ci sono meno di 16␣
 ↪errori
random_wrong_indices = np.random.choice(wrong_indices, num_samples,␣
 ↪replace=False)

# Mostra le immagini errate
fig, axes = plt.subplots(4, 4, figsize=(10, 10))

for i, ax in enumerate(axes.flat):
    if i < num_samples:
        idx = random_wrong_indices[i]
```

```
        # Ripristina i valori originali dell'immagine
        img = x_test[idx].copy()  # Copia per sicurezza
        img = img + [123.68, 116.78, 103.94]  # Riaggiunge la normalizzazione␣
 ↪di VGG19
        img = np.clip(img, 0, 255).astype(np.uint8)  # Riporta a valori corretti

        # Converti da BGR a RGB
        img = img[..., ::-1]

        ax.imshow(img)
        ax.set_title(f"True: {y2_test_classes[idx]}, Pred:␣
 ↪{y2_pred_classes[idx]}")
    ax.axis("off")

plt.show()
```



Valutazione Test esterno

[ ]:
```
# Itera su tutte le immagini nella cartella
for nome_file in os.listdir(cartella_immagini):
    percorso_file = os.path.join(cartella_immagini, nome_file)

    # Controlla che sia un file immagine
    if not (nome_file.lower().endswith((".jpg", ".png", ".jpeg"))):
        continue

    # Carica l'immagine
    img = cv2.imread(percorso_file)
    if img is None:
```

```
        print(f"Errore nel caricamento di {nome_file}")
        continue

    img_resized = resize_image_keep_aspect_ratio(img)

    # Mostra l'immagine
    plt.imshow(img_resized / 255)  # Normalizzazione per la visualizzazione
    plt.title(f"Immagine: {nome_file}")
    plt.axis("off")
    plt.show()

    # Converti in array numpy e normalizza
    img_array = np.expand_dims(img_resized, axis=0).astype("float32") / 255.0 ␣
↪# Normalizzazione 0-1

    # Predizione con il modello custom
    prediction = model.predict(img_array)

    # Se il modello ha due output (frutta e qualità)
    predicted_fruit_idx = np.argmax(prediction[0])  # Output per la categoria␣
↪frutta
    predicted_quality_idx = np.argmax(prediction[1])  # Output per la qualità

    # Recupera i nomi dai dizionari/liste delle etichette
    predicted_fruit = labels_1_array[predicted_fruit_idx]
    predicted_quality = labels_2_array[predicted_quality_idx]

    # Stampa il risultato
    print(f"Immagine: {nome_file}, Predizione: {predicted_fruit} -␣
↪{predicted_quality}")
```

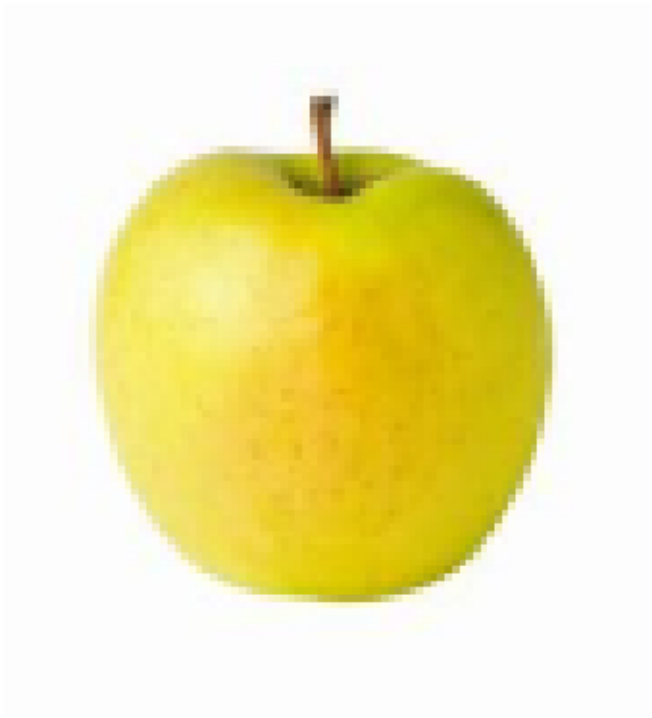

```
1/1             0s 163ms/step
Immagine: Cocos.jpg, Predizione: Peach - Peach Flat
```

Immagine: Chestnut.JPG

```
1/1             0s 51ms/step
Immagine: Chestnut.JPG, Predizione: Peach - Grape White
```


```
1/1             0s 51ms/step
Immagine: Lime.jpg, Predizione: Peach - Peach Flat
```

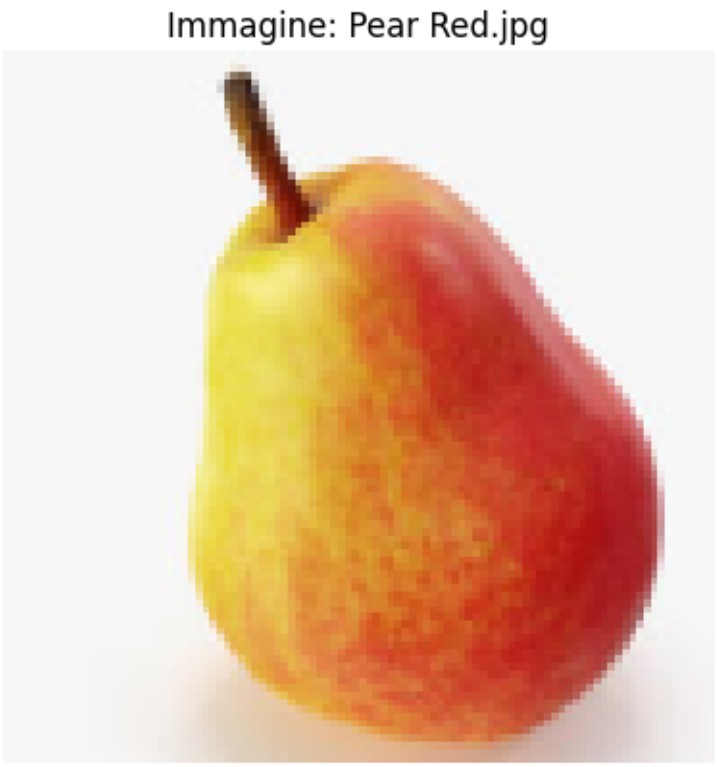

```
1/1             0s 51ms/step
Immagine: Onion White.jpg, Predizione: Peach - Peach Flat
```




```
1/1             0s 48ms/step
Immagine: Apple Golden.jpg, Predizione: Peach - Peach Flat
```

Immagine: Onion White Peeled.jpg

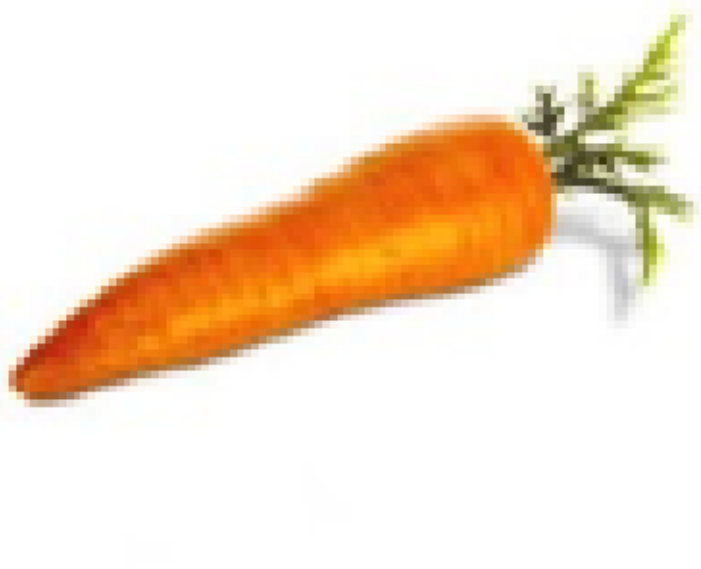
```
1/1             0s 47ms/step
Immagine: Onion White Peeled.jpg, Predizione: Peach - Peach Flat
```




```
1/1             0s 48ms/step
Immagine: Pear Red.jpg, Predizione: Peach - Peach Flat
```


```
1/1             0s 53ms/step
Immagine: apple-braeburn.jpg, Predizione: Peach - Peach Flat
```


```
1/1             0s 82ms/step
Immagine: Fig.jpg, Predizione: Peach - Peach Flat
```




```
1/1             0s 50ms/step
Immagine: Carrot.jpg, Predizione: Peach - Peach Flat
```

Immagine: Apple Golden con sfondo.jpeg

```
1/1             0s 51ms/step
Immagine: Apple Golden con sfondo.jpeg, Predizione: Peach - Grape White
```

Immagine: Grape White.jpg

```
1/1             0s 50ms/step
Immagine: Grape White.jpg, Predizione: Peach - Peach Flat
```


```
1/1             0s 49ms/step
Immagine: pomodoro.jpg, Predizione: Peach - Peach Flat
```

Immagine: Apple Golden 2.jpeg

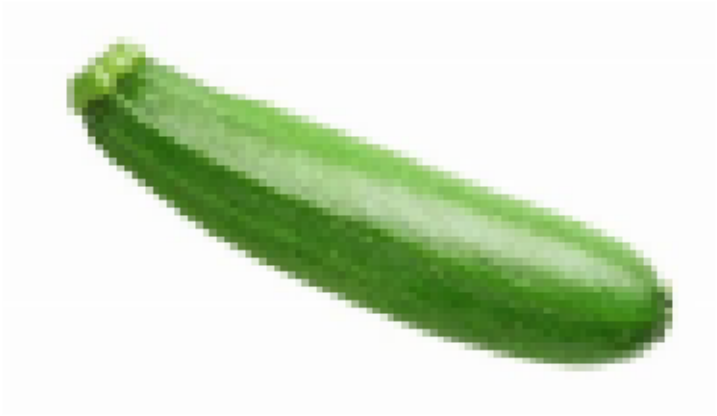
```
1/1             0s 49ms/step
Immagine: Apple Golden 2.jpeg, Predizione: Peach - Peach Flat
```


```
1/1             0s 47ms/step
Immagine: Lychee.jpg, Predizione: Peach - Peach Flat
```


```
1/1             0s 50ms/step
Immagine: Cucumber.png, Predizione: Cucumber - Peach Flat
```

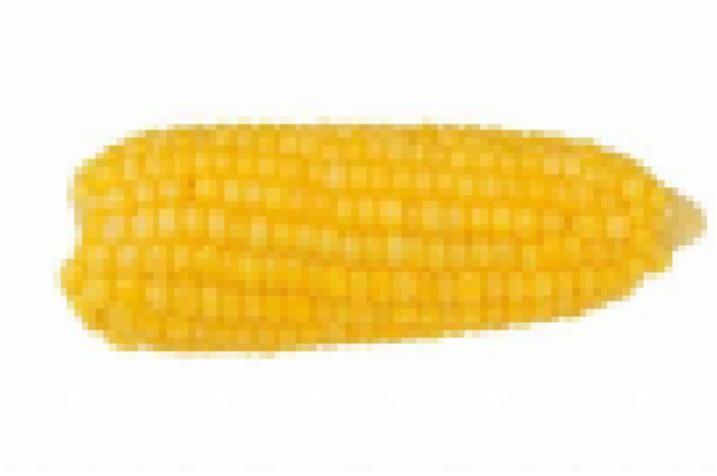

```
1/1             0s 48ms/step
Immagine: Dragon Fruit.jpg, Predizione: Peach - Peach Flat
```


```
1/1             0s 50ms/step
Immagine: Lemon.jpg, Predizione: Peach - Peach Flat
```




```
1/1             0s 49ms/step
Immagine: zucchina.jpg, Predizione: Peach - Peach Flat
```


```
1/1             0s 48ms/step
Immagine: Guava.jpg, Predizione: Peach - Peach Flat
```




```
1/1             0s 48ms/step
Immagine: mais.jpg, Predizione: Peach - Peach Flat
```


```
1/1             0s 45ms/step
Immagine: kiwi.jpg, Predizione: Peach - Peach Flat
```

1.6 Conclusioni

	Tempi di esecuzione	Accuracy	f1-score	Precisione sul test esterno
Baseline	20m	~80%	~78%	<20%
Siamese Model	1:30h	~85%	~86%	>20%
ResNet101	45m	~94%	~94%	~50%
VGG19	43m	~84%	~84%	<20%

Possiamo dunque concludere ritenendoci soddisfatti dei risultati ottenuti in quanto tutti molto inclini con le nostre aspettative. Infine possiamo decreatare il modello 2 (fine tune di resnet101 con loss tradizionale) come il più adeguato a risolvere problemi di multitask learning gerarchici.