

PROGETTO FIA

Il nostro progetto ha lo scopo di creare una rete neurale che sia in grado di riconoscere il genere di un libro data la sua copertina. In particolare, lavoriamo su cinque generi:

- Libri per bambini
- Libri di informatica
- Libri thriller
- Libri di storia naturale
- Libri sul mondo dei trasporti

Per usufruire della rete neurale abbiamo implementato una Web Application basilare tramite la quale, con l'aiuto di un form, un utente può caricare l'immagine di una copertina. Successivamente la Web App richiede l'esecuzione della rete che restituirà come risultato il genere del libro della copertina inviatagli. Infine, tale risultato sarà mostrato all'utente.

Di seguito mostriamo le documentazioni dei moduli creati per la realizzazione del progetto.

CREAZIONE DATASET

In questo notebook lavoriamo su un dataset di copertine di libri scaricato da kaggle (<https://www.kaggle.com/lukaanicin/book-covers-dataset>), le cui copertine sono già divise in tante cartelle quante sono le categorie di libri in esame. Noi ne effettuiamo un'ulteriore divisione tramite la creazione di una cartella contenente copertine per il test ed una per il train; la seconda con l'80% di copertine per ogni genere, la prima con il rimanente 20% delle immagini.

```
import matplotlib.pyplot as plt
import numpy as np
from progressbar import ProgressBar
import random
import cv2
import os
import sys

pbar= ProgressBar()
```

Colleghiamo il notebook al nostro google drive per permettergli accesso alla cartella book-covers con il dataset non lavorato

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

 Mounted at /content/gdrive/

Ci assicuriamo che il mounting sia avvenuto correttamente facendoci restituire i nomi delle cartelle relative al dataset scaricato. Ai nomi delle cartelle abbiamo aggiunto un valore numerico per facilitarne la categorizzazione

```
!ls 'gdrive/MyDrive/ProgettoFIA/book-covers'
```

Art-Photography_0	Natural-History_17
Biography_1	Personal-Development_18
Business-Finance-Law_2	Poetry-Drama_19
Childrens-Books_3	Reference_20
Computing_4	Religion_21
Crafts-Hobbies_5	Romance_22
Crime-Thriller_6	Science-Fiction-Fantasy-Horror_23
Dictionaries-Languages_7	Science-Geography_24
Entertainment_8	Society-Social-Sciences_25
Food-Drink_9	Sport_26
Graphic-Novels-Anime-Manga_10	Stationery_27
Health_11	Teaching-Resources-Education_28
History-Archaeology_12	Technology-Engineering_29
Home-Garden_13	Teen-Young-Adult_30
Humour_14	Transport_31
Medical_15	Travel-Holiday-Guides_32
Mind-Body-Spirit_16	

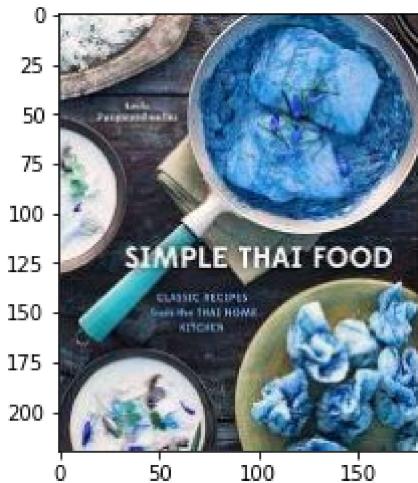
Dopo aver creato una cartella "data", con al suo interno le cartelle train e test, creiamo la variabile DATADIR con il path della cartella da cui prenderemo le immagini e le due variabili TargetDir che rappresentano i path in cui invece le salveremo.

```
DATADIR = 'gdrive/MyDrive/ProgettoFIA/book-covers'
TargetDir1='gdrive/MyDrive/ProgettoFIA/data/train'
TargetDir2='gdrive/MyDrive/ProgettoFIA/data/test'
Categories = os.listdir(DATADIR)
```

Indichiamo di creare una cartella per ogni categoria all' interno di DATADIR anche nelle altre due cartelle

```
for category in Categories:
    os.makedirs('gdrive/MyDrive/ProgettoFIA/data/train/' + category)
    os.makedirs('gdrive/MyDrive/ProgettoFIA/data/test/' + category)
```

```
for category in Categories:
    path = os.path.join(DATADIR, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, '0000988.jpg'))
        plt.imshow(img_array)
        plt.show()
        break
    break
```



```
img_array.shape
```

```
(220, 182, 3)
```

La funzione resize effettua prima il resize di tutte le immagini del dataset alle dimensioni "100x100", successivamente le salva all'interno delle cartelle di test e di train come previsto.

```
from PIL import Image
```

```
def resize(source_dir,target_dir1,target_dir2,perc):
    files= os.listdir(source_dir)
    dim=len(files)
    dim1=int(perc*dim)
    i= source_dir.split('_')[1]
    for item in files[:dim1]:
        if os.path.isfile(os.path.join(source_dir,item)):
            im = Image.open(os.path.join(source_dir,item))
            f, e = os.path.splitext(os.path.join(target_dir1,item))
            rgb_im=im.convert('RGB')
            imResize = rgb_im.resize((100,100), Image.ANTIALIAS)
            imResize.save(f + '_' + str(i) +'.jpg', 'JPEG', quality=90)

    for item in files[dim1:]:
        if os.path.isfile(os.path.join(source_dir,item)):
            im = Image.open(os.path.join(source_dir,item))
            f, e = os.path.splitext(os.path.join(target_dir2,item))
            rgb_im=im.convert('RGB')
            imResize = rgb_im.resize((100,100), Image.ANTIALIAS)
            imResize.save(f + '_' + str(i) +'.jpg', 'JPEG', quality=90)

for category in Categories:
    resize(os.path.join(DATADIR,category),
           os.path.join(TargetDir1,category),
           os.path.join(TargetDir2,category),0.8)
```

PROGETTO DI FONDAMENTI DI INTELLIGENZA ARTIFICIALE

Questo progetto si pone come obiettivo la creazione di una rete neurale con lo scopo di distinguere il genere dei libri dalla loro copertina. Il dataset utilizzato per la creazione del modello della rete neurale è stato preso da kaggle (<https://www.kaggle.com/lukaanicin/book-covers-dataset>) dandoci a disposizione 33 generi diversi di libri e circa 900 copertine per ognuno di essi consentendoci di avere un dataset di una mole più che adeguata su cui lavorare.

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from glob import glob

from tensorflow.keras.applications.vgg16 import VGG16 as PretrainedModel, \
preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Naturalmente per poter lavorare tramite i notebook di google colab abbiamo caricato il dataset su Google Drive ed effettuato il mounting del drive all' interno del notebook per poter far vedere le immagini alla rete neurale

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

Mounted at /content/gdrive/

Prendiamo i path che conducono alle cartelle apposite che abbiamo creato in precedenza rispettando la proporzione di 80% degli elementi che formano il training set ed il 20% per il validation set

```
train_path = 'gdrive/MyDrive/ProgettoFIA/data/train'
valid_path = 'gdrive/MyDrive/ProgettoFIA/data/test'
```

E dal path delle cartelle prendiamo il path per i singoli elementi

```
image_files = glob(train_path + '/*/*.jpg')
valid_image_files = glob(valid_path + '/*/*.jpg')
```

Ma per evitare l'utilizzo di troppe risorse che non avevamo a nostra disposizione e per via della mole di immagini che rendeva troppo lento il caricamento del dataset abbiamo deciso di selezionare 5 generi.

```
folders = glob(train_path + '/*')
folders

['gdrive/MyDrive/ProgettoFIA/data/train/Childrens-Books_0',
 'gdrive/MyDrive/ProgettoFIA/data/train/Computing_1',
 'gdrive/MyDrive/ProgettoFIA/data/train/Crime-Thriller_2',
 'gdrive/MyDrive/ProgettoFIA/data/train/Natural-History_3',
 'gdrive/MyDrive/ProgettoFIA/data/train/Transport_4']
```

Tutte le immagini sono state ridimensionate per avere una dimensione unica per tutte quante le immagini e permettere di avere uno shape unico di input da dare alla rete neurale

```
IMAGE_SIZE = [100, 100]
```

Sfortunatamente in precedenti test su una rete neurale completamente di nostra creazione i risultati ottenuti erano insoddisfacenti e quindi per ottimizzare i risultati abbiamo deciso di sfruttare il transfer learning.

Cioè usare una rete neurale convoluzionale preaddestrata, la rete VGG16, inclusa nella libreria di keras.

Abbiamo diviso il modello in due parti per poter allenare solo gli ultimi layer che permettono la classificazione in generi e ridurre il carico computazionale.

```
ptm = PretrainedModel(
    include_top = False,
    weights='imagenet',
    input_shape=IMAGE_SIZE + [3])

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/v1/58892288/58889256 [=====] - 0s 0us/step
```



Ma avere solo gli strati di convoluzione non ci aiuta, per cui dobbiamo aggiungere uno strato di Flatten per poter prendere l'output di questo modello ed unirlo ad una serie di strati Dense per poter avere una predizione riguardante il genere dei libri le cui copertine verranno usate come input

```
x = Flatten()(ptm.output)
```

Per cui indichiamo al modello di prendere in input le immagini delle copertine e di restituire in output il risultato del Flatten

```
model = Model(inputs=ptm.input, outputs=x)
```

```
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[(None, 100, 100, 3)]	0
block1_conv1 (Conv2D)	(None, 100, 100, 64)	1792
block1_conv2 (Conv2D)	(None, 100, 100, 64)	36928
block1_pool (MaxPooling2D)	(None, 50, 50, 64)	0
block2_conv1 (Conv2D)	(None, 50, 50, 128)	73856
block2_conv2 (Conv2D)	(None, 50, 50, 128)	147584
block2_pool (MaxPooling2D)	(None, 25, 25, 128)	0
block3_conv1 (Conv2D)	(None, 25, 25, 256)	295168
block3_conv2 (Conv2D)	(None, 25, 25, 256)	590080
block3_conv3 (Conv2D)	(None, 25, 25, 256)	590080
block3_pool (MaxPooling2D)	(None, 12, 12, 256)	0
block4_conv1 (Conv2D)	(None, 12, 12, 512)	1180160
block4_conv2 (Conv2D)	(None, 12, 12, 512)	2359808
block4_conv3 (Conv2D)	(None, 12, 12, 512)	2359808
block4_pool (MaxPooling2D)	(None, 6, 6, 512)	0
block5_conv1 (Conv2D)	(None, 6, 6, 512)	2359808
block5_conv2 (Conv2D)	(None, 6, 6, 512)	2359808
block5_conv3 (Conv2D)	(None, 6, 6, 512)	2359808
block5_pool (MaxPooling2D)	(None, 3, 3, 512)	0
flatten (Flatten)	(None, 4608)	0
<hr/>		
Total params:	14,714,688	
Trainable params:	14,714,688	
Non-trainable params:	0	

Per dare le informazioni delle copertine nella seconda parte della rete neurale utilizziamo ImageDataGenerator per creare i "generatori" di immagini dal train e dal validation set, che caricheranno i dati dalle cartelle corrispondenti dove il batch size corrisponde a quanti elementi verranno presi ad ogni elaborazione

```
gen = ImageDataGenerator(
    preprocessing_function=preprocess_input)

batch_size = 300

train_generator = gen.flow_from_directory(
    train_path,
    target_size=IMAGE_SIZE,
    batch_size=batch_size,
    class_mode='categorical'
)

valid_generator = gen.flow_from_directory(
    valid_path,
    target_size = IMAGE_SIZE,
    batch_size = batch_size,
    class_mode = 'categorical'
)

Found 3949 images belonging to 5 classes.
Found 990 images belonging to 5 classes.
```

Qui prendiamo il numero delle immagini del training e del validation set e lo shape delle loro feature

```
Ntrain = len(image_files)
Nvalid = len(valid_image_files)

feat = model.predict(np.random.random([1] + IMAGE_SIZE + [3]))
D = feat.shape[1]
```

Questa funzione crea degli array pieni di zeri prendendo come input il numero di immagini e lo shape delle feature per creare gli array delle x, mentre l'array delle y richiede il numero delle immagini ed il numero di generi su cui lavoreremo

```
def create_Xy (Nset,D,i):

    X=np.zeros((Nset,D))
    y=np.zeros((Nset,i))

    return X,y
```

```
X_train, Y_train =create_Xy(Ntrain,D,5)
```

```
X_valid, Y_valid =create_Xy(Nvalid,D,5)
```

Quest'altra funzione prende gli array creati precedentemente e li sovrascrive con i dati delle copertine fino a quando non abbiamo tutte le copertine

```
def create_set(generator,X_set,Y_set,Nset):
    i = 0
    for x, y in generator:
        features = model.predict(x)
        sz = len(y)
        X_set[i:i + sz] = features
        Y_set[i:i + sz] = y

        i += sz
        print(i)
        if i >= Nset:
            print('breaking now')
            break

    print(i)
```

```
create_set(train_generator,X_train,Y_train,Ntrain)
```

```
create_set(valid_generator, X_valid, Y_valid,Nvalid)
```

L'input di questo modello corrisponde allo shape delle feature ed all'output dell'Flatten del modello precedente.

In questo secondo modello non usiamo solamente layer di Dense ma anche layer di Dropout per avere delle predizioni più precise del 3% e dei valori di validation loss non eccessivamente al di sopra il training loss.

```
i = Input(shape=(D,))

x=Dropout(0.5)(i)
x = Dense(256, activation='relu')(x)
x=Dropout(0.1)(x)
x = Dense(256, activation='relu')(x)
x=Dropout(0.1)(x)
x = Dense(256, activation='relu')(x)
x=Dropout(0.1)(x)
x = Dense(512, activation='relu')(x)
x=Dropout(0.1)(x)
x = Dense(512, activation='relu')(x)
x=Dropout(0.1)(x)
```

```

x = Dense(1024, activation='relu')(x)
x=Dropout(0.1)(x)
x = Dense(1024, activation='relu')(x)
x=Dropout(0.1)(x)
x = Dense(5, activation='softmax')(x)

linear_model = Model(i, x)

linear_model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

r = linear_model.fit(
    X_train,
    Y_train,
    validation_data = (X_valid, Y_valid),
    epochs = 20,
    batch_size = 300
)

```

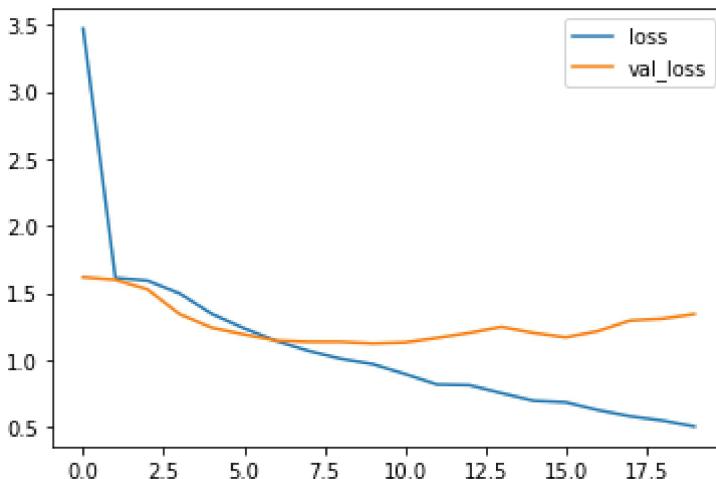
Di seguito mostriamo i grafici di loss ed accuratezza con e senza Dropout della rete neurale

```

#con Dropout
#Epoch 1/20
# loss: 4.9555 - val_loss: 1.6170
#Epoch 16/20
# loss: 0.6642 - val_loss: 1.1679
plt.plot(r.history["loss"], label="loss")
plt.plot(r.history["val_loss"], label="val_loss")
plt.legend()

```

<matplotlib.legend.Legend at 0x7f691203f0f0>



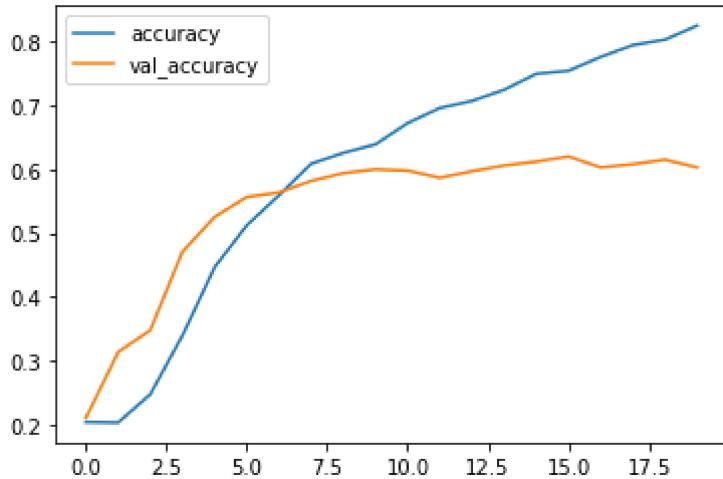
```

#con Dropout
#Epoch 1/20
# accuracy: 0.2025 - val_accuracy: 0.2101
#Epoch 16/20

```

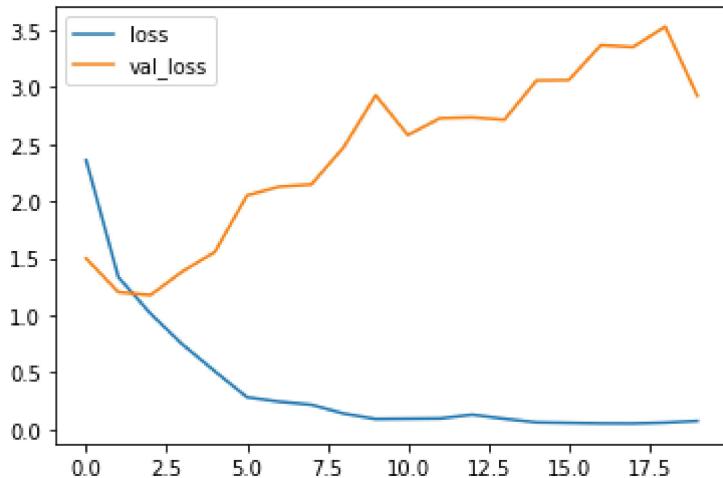
```
# accuracy: 0.7575 - val_accuracy: 0.6202
plt.plot(r.history["accuracy"], label="accuracy")
plt.plot(r.history["val_accuracy"], label="val_accuracy")
plt.legend()
```

<matplotlib.legend.Legend at 0x7f6912081438>



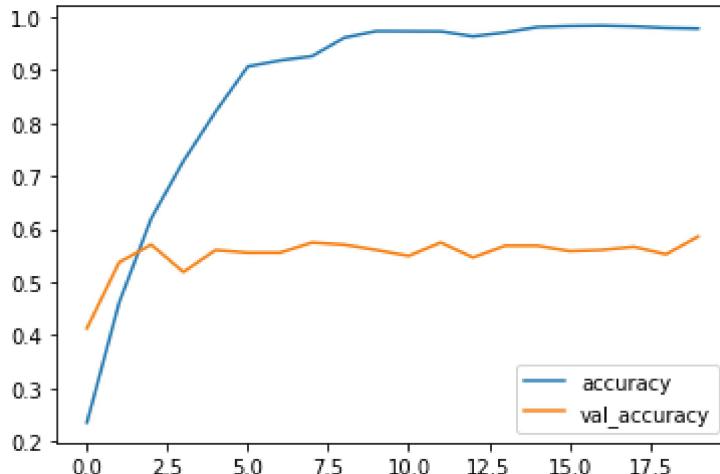
```
#senza Dropout
#Epoch 1/20
# loss: 2.9654 - val_loss: 1.4973
#Epoch 20/20
# loss: 0.0614 - val_loss: 2.9242
plt.plot(r.history["loss"], label="loss")
plt.plot(r.history["val_loss"], label="val_loss")
plt.legend()
```

<matplotlib.legend.Legend at 0x7fe22224ff98>



```
#senza Dropout
#Epoch 1/20
# accuracy: 0.2203 - val_accuracy: 0.4131
#Epoch 20/20
# accuracy: 0.9809 - val_accuracy: 0.5859
plt.plot(r.history["accuracy"], label="accuracy")
plt.plot(r.history["val_accuracy"], label="val_accuracy")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fe2222a2198>
```



Questa rappresentazione grafica rappresenta la confusion matrix, una matrice che permette di esaminare quante immagini sono state categorizzate in maniera precisa e quanti elementi di un singolo genere siano stati erroneamente categorizzati in un altro, dove gli indici che usiamo corrispondono ai numeri scelti nei nomi delle cartelle dei generi.

```
from sklearn.metrics import confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                         normalize=False,
                         title='Confusion matrix',
                         cmap=plt.cm.Blues):
    """
    Normalizzazione può essere applicata dal setting `normalize=True`.
    """

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
```

```

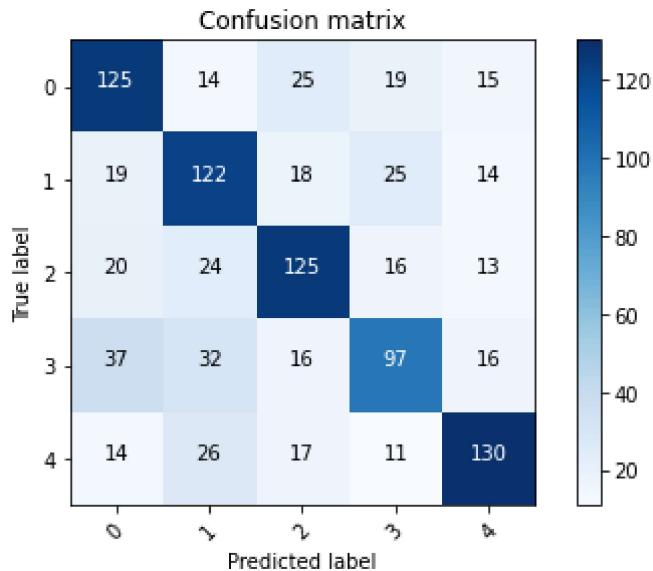
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

p_test = linear_model.predict(X_valid).argmax(axis=1)
Y=Y_valid.argmax(axis=1)
cm = confusion_matrix(Y, p_test)
plot_confusion_matrix(cm, folders)

```

Confusion matrix, without normalization

[[125 14 25 19 15]
[19 122 18 25 14]
[20 24 125 16 13]
[37 32 16 97 16]
[14 26 17 11 130]]



folders

```

['gdrive/MyDrive/ProgettoFIA/data/train/Childrens-Books_0',
 'gdrive/MyDrive/ProgettoFIA/data/train/Computing_1',
 'gdrive/MyDrive/ProgettoFIA/data/train/Crime-Thriller_2',
 'gdrive/MyDrive/ProgettoFIA/data/train/Natural-History_3',
 'gdrive/MyDrive/ProgettoFIA/data/train/Transport_4']

```

Alla fine di tutto salviamo i nostri due modelli per poterli riutilizzare nella web app creata per testare la rete neurale

```
model.save('gdrive/MyDrive/ProgettoFIA/Modello5generi')
```

```
INFO:tensorflow:Assets written to: gdrive/MyDrive/ProgettoFIA/Modello5generi/assets
```

```
linear_model.save('gdrive/MyDrive/ProgettoFIA/Modello5generi/modello_linear')
```

```
INFO:tensorflow:Assets written to: gdrive/MyDrive/ProgettoFIA/Modello5generi/modello_
```

Di seguito un piccolo esempio che prende un'immagine casuale dal nostro dataset

```
Y_valid[0]
```

```
array([0., 0., 0., 0., 1.])
```

```
img_sel=np.array(image.load_img(np.random.choice(image_files)))
```

```
array=[img_sel]
```

```
array=np.array(array)
```

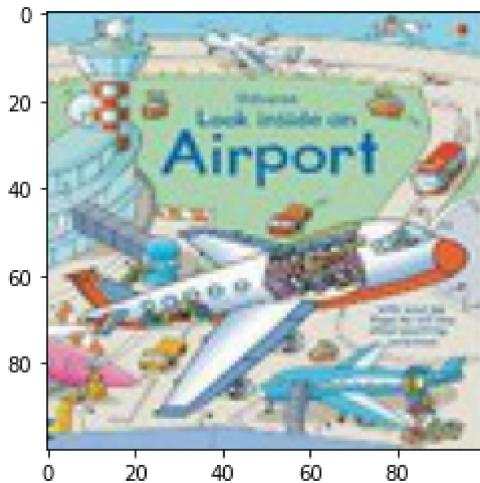
```
array.shape
```

```
(1, 100, 100, 3)
```

```
plt.imshow(img_sel)
```



```
<matplotlib.image.AxesImage at 0x7fcdf76164a8>
```



```
sel=model.predict(array)
```

```
sel.shape
```

```
(1, 4608)
```

```
linear_sel=linear_model.predict(sel)
```

```
linear_sel
```

```
array([[9.9999833e-01, 3.2477294e-07, 1.1112187e-06, 2.0387871e-07,
       3.5841989e-08]], dtype=float32)
```


WEB APPLICATION RETE NEURALE

Implementiamo qui una Web Application molto basilare tramite l'utilizzo del modulo Flask di Python.

Per prima cosa installiamo ngrok per Flask che ci permetterà di esporre il server in locale all'esterno. In questo modo potremo raggiungere la nostra Web App senza installare alcun server su Google Drive.

```
!pip install flask-ngrok
```

```
Collecting flask-ngrok
  Downloading https://files.pythonhosted.org/packages/af/6c/f54cb686ad1129e27d125d18/
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: itsdangerous>=0.24 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: Werkzeug>=0.15 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: Jinja2>=2.10.1 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: click>=5.1 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.6/dist-pac
Installing collected packages: flask-ngrok
Successfully installed flask-ngrok-0.0.25
```

Importiamo tutte le librerie che ci serviranno per l'implementazione della Web App e l'utilizzo della rete neurale

```
from flask_ngrok import run_with_ngrok
from flask import Flask, render_template, request, redirect, url_for
import numpy as np
import tensorflow
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
from werkzeug.utils import secure_filename
import os
from PIL import Image
```

Effettuiamo il montaggio della cartella di Drive

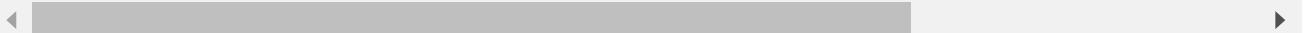
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Otteniamo le componenti della rete neurale che useremo in seguito

```
model = tensorflow.keras.models.load_model('drive/MyDrive/ProgettoFIA/Modello5generi')
linear_model = tensorflow.keras.models.load_model('drive/MyDrive/ProgettoFIA/modello_linea
```

WARNING:tensorflow:No training configuration found in save file, so the model was *not* compiled



Avviamo la Web App e teniamo traccia della cartella in cui salveremo tutte le immagini che analizzeremo

```
upload_folder = 'drive/MyDrive/ProgettoFIA/WebApp/images'
app = Flask(__name__, template_folder='drive/MyDrive/ProgettoFIA/WebApp/templates')
run_with_ngrok(app)
app.config["upload_folder"] = upload_folder
```

Quando accediamo alla pagina "home" tramite una richiesta di tipo "Get", effettuiamo il rendering di "home.html" che contiene semplicemente un form nel quale allegare l'immagine da analizzare, per poi inviarla tramite una richiesta "Post" alla pagina "home" stessa. Se la richiesta è "Post" prendiamo l'immagine allegata nel form, ne effettuiamo il resize alle dimensioni 100x100 (che è la dimensione che si aspetta di ricevere la nostra rete neurale), ne richiediamo l'analisi da parte della rete neurale ed inviamo il valore di probabilità associato alla categoria più promettente di appartenenza dell'immagine alla pagina maxIndex.

```
@app.route("/home", methods=["GET", "POST"])
def home():
    if request.method == "POST":
        #resize dell'immagine a 100x100
        print(request.files["img"])
        imageFile = request.files["img"]
        imageFile.save(os.path.join(app.config["upload_folder"]), secure_filename(imageFile.filename))
        resizeImage = Image.open(upload_folder+"/"+imageFile.filename)
        resizeImage = resizeImage.resize((100,100), Image.ANTIALIAS)
        resizeImage.save(upload_folder+"/2"+imageFile.filename)
        img_sel = np.array(image.load_img(upload_folder+"/2"+imageFile.filename))
        array = [img_sel]
        array = np.array(array)
        print(array.shape)
        sel = model.predict(array)
        linear_sel = linear_model.predict(sel)
        print(linear_sel)
        result = [i for i, j in enumerate(linear_sel[0]) if j == max(linear_sel[0])][0]
        print(result)
        return redirect(url_for("maxIndex", maxIndex=result))
    else:
        return render_template("home.html")
```

La pagina maxIndex converte il valore di probabilità associato alla categoria più promettente calcolata dalla rete neurale in una stringa corrispondente al nome di tale categoria. Infine mostra tale nome.

```
@app.route("/<maxIndex>")
def maxIndex(maxIndex):
    maxIndex = int(maxIndex)
    if maxIndex == 0:
        print("ciao")
        return f"<h1>Libro per bambini</h1>"
    elif maxIndex == 1:
        return f"<h1>Libro di informatica</h1>"
    elif maxIndex == 2:
        return f"<h1>Libro thriller</h1>"
    elif maxIndex == 3:
        return f"<h1>Libro di storia naturale</h1>"
    elif maxIndex == 4:
        return f"<h1>Libro di trasporto</h1>"
    return f"<h1>Immagine non valida</h1>"
```

Qui avviamo il server contenente la piccola Web App.

```
if __name__ == "__main__":
    app.run()

    * Serving Flask app "__main__" (lazy loading)
    * Environment: production
      WARNING: This is a development server. Do not use it in a production deployment
      Use a production WSGI server instead.
    * Debug mode: off
    * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
    * Running on http://6358d08d378a.ngrok.io
    * Traffic stats available on http://127.0.0.1:4040
127.0.0.1 - - [10/Feb/2021 08:55:44] "GET / HTTP/1.1" 404 -
[2021-02-10 08:55:44,626] ERROR in app: Exception on /favicon.ico [GET]
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 2447, in wsgi_app
    response = self.full_dispatch_request()
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1952, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1821, in handle_user_exception
    reraise(exc_type, exc_value, tb)
  File "/usr/local/lib/python3.6/dist-packages/flask/_compat.py", line 39, in reraise
    raise value
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1950, in full_dispatch_request
    rv = self.dispatch_request()
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1936, in dispatch_request
    return self.view_functions[rule.endpoint](**req.view_args)
  File "<ipython-input-7-1a9f49dd593a>", line 3, in maxIndex
    maxIndex = int(maxIndex)
ValueError: invalid literal for int() with base 10: 'favicon.ico'
127.0.0.1 - - [10/Feb/2021 08:55:44] "GET /favicon.ico HTTP/1.1" 500 -
127.0.0.1 - - [10/Feb/2021 08:55:52] "GET /home HTTP/1.1" 200 -
[2021-02-10 08:55:53,046] ERROR in app: Exception on /favicon.ico [GET]
```

```
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 2447, in wsgi_app
    response = self.full_dispatch_request()
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1952, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1821, in handle_user_exception
    reraise(exc_type, exc_value, tb)
  File "/usr/local/lib/python3.6/dist-packages/flask/_compat.py", line 39, in reraise
    raise value
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1950, in full_dispatch_request
    rv = self.dispatch_request()
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1936, in dispatch_request
    return self.view_functions[rule.endpoint](**req.view_args)
  File "<ipython-input-7-1a9f49dd593a>", line 3, in maxIndex
    maxIndex = int(maxIndex)
ValueError: invalid literal for int() with base 10: 'favicon.ico'
127.0.0.1 - - [10/Feb/2021 08:55:53] "GET /favicon.ico HTTP/1.1" 500 -
<FileStorage: 'freight-train.jpg' ('image/jpeg')>
(1, 100, 100, 3)
127.0.0.1 - - [10/Feb/2021 08:56:04] "POST /home HTTP/1.1" 302 -
127.0.0.1 - - [10/Feb/2021 08:56:04] "GET /1 HTTP/1.1" 200 -
[[0.00298549 0.9804083 0.00217622 0.0133664 0.0010635 ]]
1
[2021-02-10 08:56:04,505] ERROR in app: Exception on /favicon.ico [GET]
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 2447, in wsgi_app
    response = self.full_dispatch_request()
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1952, in full_dispatch_request
    rv = self.handle_user_exception(e)
```

