```python
import os

def download_and_setup_test_dataset():
    """
    Downloading the test dataset.
    """
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/test_database.tar')

    datasets_path = "datasets"
    if not os.path.exists(datasets_path):
        os.makedirs(datasets_path)

    # put the test dataset in datasets/test
    os.system("tar xf test_database.tar -C 'datasets' --one-top-level && mv test_database.tar datasets/test")


def download_and_setup_small_dataset():
    """
    Downloading the small dataset.

    """
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/defi1certif-datasets-fire_small.tar')

    datasets_path = "datasets"
    if not os.path.exists(datasets_path):
        os.makedirs(datasets_path)

    # put the small dataset in datasets/small
    os.system(
        "tar xf defi1certif-datasets-fire_small.tar -C 'datasets' --one-top-level && mv "
        "datasets/defi1certif-datasets-fire_small datasets/small")


def download_and_setup_medium_dataset():
    """
    Downloading the medium dataset.
    """
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/defi1certif-datasets-fire_medium.tar.001')
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/defi1certif-datasets-fire_medium.tar.002')
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/defi1certif-datasets-fire_medium.tar.003')

    datasets_path = "datasets"
    if not os.path.exists(datasets_path):
        os.makedirs(datasets_path)

    # recombine the tar files
    os.system("cat  defi1certif-datasets-fire_medium.tar.001 defi1certif-datasets-fire_medium.tar.002 "
              "defi1certif-datasets-fire_medium.tar.003 >> defi1certif-datasets-fire_medium.tar")

    # put the medium dataset in datasets/medium
    os.system("tar xf defi1certif-datasets-fire_medium.tar -C 'datasets' --one-top-level && mv "
              "datasets/defi1certif-datasets-fire_medium datasets/medium")


def download_and_setup_large_dataset():
    """
    Downloading the large dataset.
    """
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/defi1certif-datasets-fire_big.tar.001')
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/defi1certif-datasets-fire_big.tar.002')
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/defi1certif-datasets-fire_big.tar.003')
    os.system(
        'wget https://github.com/belarbi2733/keras_yolov3/releases/download/1/defi1certif-datasets-fire_big.tar.004')

    datasets_path = "datasets"
    if not os.path.exists(datasets_path):
        os.makedirs(datasets_path)

    # recombine the tar files
    os.system("cat  defi1certif-datasets-fire_big.tar.001 defi1certif-datasets-fire_big.tar.002 "
              "defi1certif-datasets-fire_big.tar.003 defi1certif-datasets-fire_big.tar.004 >> "
              "defi1certif-datasets-fire_big.tar")

    # put the large dataset in datasets/large
    os.system("tar xf defi1certif-datasets-fire_big.tar -C 'datasets' --one-top-level && mv "
              "datasets/defi1certif-datasets-fire_big datasets/large")


def setup_full_dataset():
    """
    Downloads and sets up all datasets in a single folder named all.
    A folder per class is created.
    """
    download_and_setup_small_dataset()
    download_and_setup_medium_dataset()
    download_and_setup_large_dataset()

    # creating the folder to merge datasets
    if not os.path.exists("datasets/all"):
        os.makedirs("datasets/all")
    if not os.path.exists("datasets/all/fire"):
        os.makedirs("datasets/all/fire")
    if not os.path.exists("datasets/all/no_fire"):
        os.makedirs("datasets/all/no_fire")
    if not os.path.exists("datasets/all/start_fire"):
        os.makedirs("datasets/all/start_fire")

    # moving images from the small dataset to the full dataset
    os.system("find datasets/small/fire -type f -print0 | xargs -0 mv -t datasets/all/fire/")
    os.system("find datasets/small/no_fire -type f -print0 | xargs -0 mv -t datasets/all/no_fire/")
    os.system("find datasets/small/start_fire -type f -print0 | xargs -0 mv -t datasets/all/start_fire/")

    # moving images from the medium dataset to the full dataset
    os.system("find datasets/medium/fire -type f -print0 | xargs -0 mv -t datasets/all/fire/")
    os.system("find datasets/medium/no_fire -type f -print0 | xargs -0 mv -t datasets/all/no_fire/")
    os.system("find datasets/medium/start_fire -type f -print0 | xargs -0 mv -t datasets/all/start_fire/")

    # moving images from the large dataset to the full dataset
    os.system("find datasets/large/fire -type f -print0 | xargs -0 mv -t datasets/all/fire/")
    os.system("find datasets/large/no_fire -type f -print0 | xargs -0 mv -t datasets/all/no_fire/")
    os.system("find datasets/large/start_fire -type f -print0 | xargs -0 mv -t datasets/all/start_fire/")




import imghdr
import math
import os
import numpy as np
from keras import Model
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input as inception_preprocess_input
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input as vgg16_preprocess_input
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.layers import GlobalAveragePooling2D, Dense
from keras.preprocessing import image
from keras.utils import np_utils
from matplotlib import pyplot as plt


classes = ['fire', 'no_fire', 'start_fire']
nbr_classes = 3


def generate_from_paths_and_labels(images_paths, labels, batch_size, preprocessing, image_size=(224, 224)):
    """
    Generator to give to the fit function, generates batches of samples for training.
    This avoids to load the full dataset in memory. This can also be a Keras class.
    :param images_paths:
    :param labels:
    :param batch_size:
    :param image_size:
    :param preprocessing:
    :return:
    """
    number_samples = len(images_paths)
    while 1:
        perm = np.random.permutation(number_samples)  # randomize the order of the images (to be done after each epoch)

        # apply the permutations
        images_paths = images_paths[perm]
        labels = labels[perm]

        # from 0 to number_samples by batch_size increment to generate batches
        # this assumes there are number_samples / batch_size batches in an epoch
        # which ensures that each samples is only fed once to the network at each epoch
        for i in range(0, number_samples, batch_size):
            # a batch is a list of image paths : images_paths[i:i + batch_size]
            # map transforms all paths to images using keras.preprocessing.image
            inputs = list(map(
                lambda x: image.load_img(x, target_size=image_size),
                images_paths[i:i + batch_size]
            ))
            # converting the loaded images to numpy arrays
            inputs = np.array(list(map(
                lambda x: image.img_to_array(x),
                inputs
            )))

            # preprocessing the batch might notably normalize between 0 and 1 the RGB values, this is model-dependant
            inputs = preprocessing(inputs)

            # yields the image batch and corresponding labels
            yield (inputs, labels[i:i + batch_size])


def extract_dataset(dataset_path, classes_names, percentage):
    """
    Assumes that dataset_path/classes_names[0] is a folder containing all images of class classes_names[0].
    All image paths are loaded into a numpy array, corresponding labels are one-hot encoded and put into a numpy array.
    Samples are shuffled before splitting into training and validation sets to prevent problems since samples are loaded
    in order of their class.
    :param dataset_path: path to the root of the dataset.
    :param classes_names: names of the classes.
    :param percentage: percentage of samples to be used for training, the rest is for validation. Must be in [0,1].
    :return: (x_train, y_train), (x_val, y_val) a list of image paths and a list of corresponding labels for training
    and validation.
    """

    num_classes = len(classes_names)

    # putting images paths and labels in lists
    images_paths, labels = [], []
    for class_name in os.listdir(dataset_path):
        class_path = os.path.join(dataset_path, class_name)
        class_id = classes_names.index(class_name)  # class id = index of the class_name in classes_name, later o-h enc
        # here we are considering all paths for images labeled class_id
        for path in os.listdir(class_path):
            path = os.path.join(class_path, path)  # image path
            # test the image data contained in the file , and returns a string describing the image type
            if imghdr.what(path) is None:
                # this is not an image file
                continue
            images_paths.append(path)
            labels.append(class_id)

    # one-hot encode the labels
    labels_oh = np_utils.to_categorical(labels, num_classes)
    # convert images_paths to numpy array to apply permutation
    images_paths = np.array(images_paths)

    number_samples = len(images_paths)
    perm = np.random.permutation(number_samples)
    labels_oh = labels_oh[perm]
    images_paths = images_paths[perm]

    # 90% of samples used for training
    border = math.floor(percentage * len(images_paths))

    train_labels, val_labels = labels_oh[:border], labels_oh[border:]
    train_samples, val_samples = images_paths[:border], images_paths[border:]

    print("Training on %d samples" % (len(train_samples)))
    print("Validation on %d samples" % (len(val_samples)))

    return (train_samples, train_labels), (val_samples, val_labels)

def graphically_test_model(model_path, classes_names, test_image_dir, preprocess_input, image_size=(224, 224)):
    """
    Loads a model, does a prediction on each image in test_image_dir and displays the image with the class name on
    top of it.
    :param model_path:
    :param classes_names:
    :param test_image_dir:
    :param preprocess_input:
    :param image_size:
    """
    nbr_classes = len(classes_names)
    model = load_model(model_path)

    for test_image_path in os.listdir(test_image_dir):
        # load image using keras
        img = image.load_img(test_image_dir + "/" + test_image_path, target_size=image_size)

        # processed image to feed the network
        processed_img = image.img_to_array(img)
        processed_img = np.expand_dims(processed_img, axis=0)
        processed_img = preprocess_input(processed_img)

        # get prediction using the network
        predictions = model.predict(processed_img)[0]
        # transform [0,1] values into percentages and associate it to its class name (class_name order was used to
        # one-hot encode the classes)
        result = [(classes_names[i], float(predictions[i]) * 100.0) for i in range(nbr_classes)]
        # sort the result by percentage
        result.sort(reverse=True, key=lambda x: x[1])

        # load image for displaying
        img = cv2.imread(test_image_dir + "/" + test_image_path)
        # transform into RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        font = cv2.FONT_HERSHEY_COMPLEX

        # write class percentages on the image
        for i in range(nbr_classes):
            (class_name, prob) = result[i]
            textsize = cv2.getTextSize(class_name, font, 1, 2)[0]
            textX = (img.shape[1] - textsize[0]) / 2
            textY = (img.shape[0] + textsize[1]) / 2
            if (i == 0):
                cv2.putText(img, class_name, (int(textX) - 100, int(textY)), font, 5, (255, 255, 255), 6, cv2.LINE_AA)
            print("Top %d ====================" % (i + 1))
            print("Class name: %s" % (class_name))
            print("Probability: %.2f%%" % (prob))
        plt.imshow(img)
        plt.show()


def evaluate_model(model_path, classes, preprocessing, dataset_path):
    """
    Loads a model and evaluates the model (metrics) on images provided in folder a dataset.
    :param model_path:
    :param classes:
    :param preprocessing:
    :param test_dataset:
    """
    # For simplicity, the dataset is loaded using 99.9% of images
    (train_samples, train_labels), (val_samples, val_labels) = extract_dataset(dataset_path, classes, 0)
    batch_size = 16
    nbr_val_samples = len(val_samples)
    validation_sample_generator = generate_from_paths_and_labels(val_samples, val_labels, batch_size, preprocessing,
                                                               image_size=(224, 224, 3))

    model = load_model(model_path)
    return model.evaluate_generator(validation_sample_generator, steps=math.ceil(nbr_val_samples / 16),
                            max_queue_size=10, workers=1, use_multiprocessing=True, verbose=1)
```

```
download_and_setup_test_dataset()

graphically_test_model("best_trained_save.h5", classes, "datasets/test_database/Base de données de test", inception_preprocess_input, image_size=(224, 224))
```