Homework 10

Math 87

due: December 1 2023

1 Hot (dog) or not (dog)?

In this exercise, you will train a classifier to classify pictures of food as either "“hot dogs" or "not hot dogs." This will involve both training a machine learning model, but also performing tasks such as ensuring that Python can read a file you downloaded. **You will be asked some questions at the end of this tutorial, your answers will be what you turn in.**

Begin by downloading

`hotdogs.zip`

in the class files (for convenience, you might want to make a folder on your Desktop and open the contents there). **If you are using Colab:** Google Colab cannot directly access the files on your computer, so you will have to upload the data sets to the Files tab in your Colab workspace. You can then perform the steps below by referencing the files you uploaded (to find the path names for these files, you can right click on the files and select "Copy path.").

1. The following code imports some of the modules you will need, and defines a function that will take in a jpeg and output an array, which you will need to process your data. Copy and paste it in to a cell:

Sampling

```
import os
from os.path import join
import tensorflow as tf
import matplotlib.pyplot as plt
from IPython.display import Image, display
import numpy as np
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import load_img, img_to_array

image_size = 224

def read_and_prep_images(img_paths,
img_height=image_size, img_width=image_size):
    imgs = [load_img(img_path, target_size=(img_height, img_width))
    for img_path in img_paths]
    img_array = np.array([img_to_array(img) for img in imgs])
    output = preprocess_input(img_array)
    return(output)
```

2. The first step will be to import the images you downloaded into your Python shell. We will start with the training images. The folder "hotdogs" contains three folders,

   `seefood, test,train`

   . The training images contains the images you will train your classifier on. Train contains two folders, one containing hot dog images and the other with not hot dog images. These folders can be referenced by a path that might look something like e.g.

   `/Users/JohnSmith/Desktop/untitled folder/hotdogs/train/hot\_dog`

   Determine the paths for the hot_dog and not_hot_dog folders, and name them as below:

Sampling

```
train_hotdog_dir = (path to train/hot_dog here)
train_notdog_dir = (path to train/not_hot_dog here)
```

   We now have a way to reference the folders in your computer from the Python shell.

3. Now use the os.listdir function to make two lists, one containing the names of all the pictures in the hot_dog and not_hot_dog folders. Using these, you must now create two *new* lists which give the *path* to each of the images in the folders. You can do this using the "join" function (which can combine two strings) and a for loop iterating over the lists of image names you just made. Name these two lists hot_dog_paths and not_hot_dog_paths.

Sampling

```
train_dog_filenames= (Use oslist.dir here)
train_notdog_filenames= (Use oslist.dir here)
hot_dog_paths =
not_hot_dog_paths =
```

   The output of this should be two lists of strings, where each entry is a filepath to a specific image.

4. The next step is to generate the training set in Python. The training set should be an pair of arrays x and y, where $x[i]$ is an image (stored as an array) and $y[i]$ is a value in $\{0, 1\}$, where $y[i] = 1$ if the image $x[i]$ is a hot dog and 0 otherwise. The function

   `read_and_prep_images`

   takes in a filepath and returns the image it indexes as an array. Using this function, and the lists hot_dog_paths and not_hot_dog_paths, create the training set. (Hint: the np.ones and np.zeros functions will be useful here). **It will be important that the objects you generate here are actually arrays, and not just lists. To be sure of this, you may pass your output through the np.asarray function**.

5. Now we have a training set of images and labels, we can define the model. How you do this is up to you. To start with, you may use the example in Class 17 materials, but feel free to modify the model as you see fit (e.g. change the type/size/number of layers, see the Keras documentation for instructions on how to do this).

- Define a model using tf.keras.model;
- Define a loss function;
- Compile the model with an optimizer

It is important that your model is able to recognize the image data. Use the np.shape function to see what format the images are stored in, and make sure that this agrees with the input_shape for the model.

6. Train the model on your training set using model.fit! The "x-values" should be images, the "y-values" should be labels. It is recommended that you set "shuffle=True" in the model.fit function (This means that the dataset will be randomly permuted every time it is put through the model. This the model won't be learning things about the order that the data is stored, which is arbitrary).

7. You are now ready to test your model on the test data sets.. Repeat steps 1-3 to create a test set, which will again comprise of a set of images and labels. Then use model.evaluate with the images as the x input and labels as the y input. (You will not be graded by the accuracy of your model on the test data).

Questions:

1. Copy and paste all of your code into a single block, and submit this as part of your homework.

2. Perform the above with at least three different model architectures (i.e. with changes to type/number of/width of layers). At least one of these models should use a 2D convolutional layer (Here is an example of how to set one up:

```
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(16, kernel_size=(3, 3), activation='relu',
input_shape=(224, 224,3)),
tf.keras.layers.Conv2D(32,kernel_size=(3,3),activation='relu'),
tf.keras.layers.Flatten(),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(2) ])
```

   This is a fairly small example. In practice, you might want more layers, higher filter counts and to intersperse dense layers).

   Record the performance of each architecture against the training data with a fixed number of epochs. What is the final classification score against the training data with the different architectures? Do you notice anything about how changes in architecture result in changes to performance? How about speed?

3. Record the performance of these architectures against the test data. Comment on your findings. How high can you get your classification score on the test data?

4. What is the purpose of "Dropout"? Do you observe any changes in your results when you use Dropout or not?