Take-Home Test

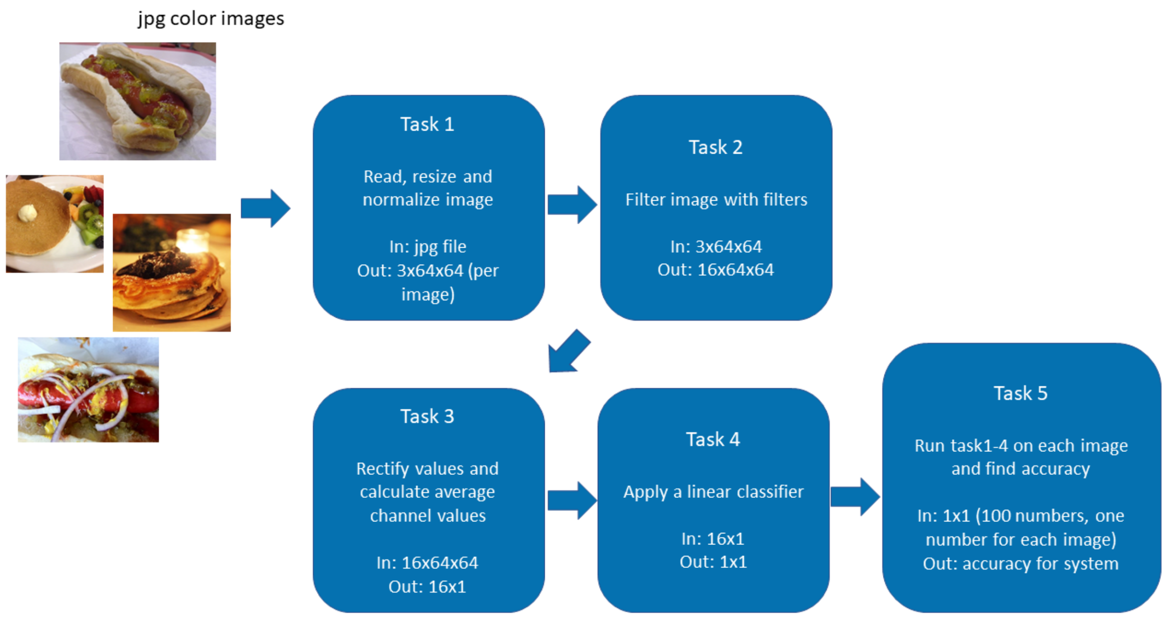
Mikael Nilsson

December 20, 2024

Introduction

In this test, you will be asked to write a short program that reads in a set of numerical values that can be used to form some filters and a linear classifier that can be used on images. You will also be provided with 100 images, 50 of hotdogs and 50 of pancakes. The files you need is in a zip file and can be downloaded (Download raw file) from this link: `https://github.com/MikaelNilssonLTH/TakeHomeTest/blob/main/TakeHomeTest.zip`

The test is divided into five tasks related to forming a classifier that tries to decide if a given input image contains a hotdog or a pancake, see an overview below.


jpg color images
Task 1
Read, resize and normalize image
In: jpg file
Out: 3x64x64 (per image)
Task 2
Filter image with filters
In: 3x64x64
Out: 16x64x64
Task 3
Rectify values and calculate average channel values
In: 16x64x64
Out: 16x1
Task 4
Apply a linear classifier
In: 16x1
Out: 1x1
Task 5
Run task1-4 on each image and find accuracy
In: 1x1 (100 numbers, one number for each image)
Out: accuracy for system

You are free to use any programming language to complete the tasks, but some Python and Matlab hints will be provided. If you think you failed some task, just move on with the next and use some random numbers as input (see sizes of arrays in the overview figure above).

Task 1 - Read, resize and normalize image

You will find 50 hotdog images (jpg) in the `data/hotdog` catalog and 50 pancake images in `data/pancakes`. The images are in color with red, green and blue channels (three channels). Make a function that reads in the color image, resize it to 64×64 (height and width) with three channels, it should contain floats in numerical range $[0, 1]$ and then re-scale this numerical range with a mean ($\mu = 0.5$) and standard deviation ($\sigma = 0.25$) according to

$$\text{output} = \frac{(\text{input} - \mu)}{\sigma} \tag{1}$$

where input here will be every value in the $C \times H \times W$ ($C = 3$ color channels, $H = 64$ the image height, $W = 64$ the image width) color image. Finally, show one (any, random) image from the hotdog or pancake dataset by using the function you implemented above (read, resize and normalize).

HINT:

It might be a good idea to scale back the function output to numerical range $[0, 1]$ by $0.25\text{output} + 0.5$ that can be used properly in many image display functions such as matplotlib.pyplot.imshow in Python or imagesc in Matlab.

Task 2 - Filter image with filters

You will find $K = 16$ filter weights (`weight<num>.csv` where `<num>` is filter number $0, 1, \ldots, 15$) and corresponding bias value (`bias<num>.csv>`). First make a function that reads in a filter and bias. The bias is a single value, b, for each filter and the weight, $\mathbf{w}$, is size $3 \times 9 \times 9$ for each filter. Note that the data you read will give you a vector of length $243 (= 3 \cdot 9 \cdot 9)$ as you read in values from the csv file. The order is in Row-Major (or C order, NumPy/Python default), as opposed to Column-Major (or Fortan order, Matlab default), and is stored as $C \times H \times W$. Note that the channels ($C = 3$) are here the red, green and blue (in that order).

HINT:

Using NumPy in Python you can read these csv files with numpy.loadtxt with proper settings followed by numpy.reshape with proper settings for shaping the weights.

HINT:

Using Matlab you can read in using csvread and using a suitable *index re-ordering* and then using reshape with proper settings.

As a test if you get the order correct you can find the 9x9 filter from weight0.csv that is mapped to the red (first channel, $w(0, 0 \ldots 8, 0 \ldots 8)$) here (with some numerical rounding):

$$\begin{bmatrix} -0.102 & -0.073 & -0.114 & -0.033 & -0.005 & -0.124 & -0.032 & -0.078 & -0.039 \\ -0.065 & -0.039 & -0.073 & -0.004 & -0.036 & -0.092 & -0.056 & -0.101 & -0.075 \\ -0.080 & -0.007 & -0.094 & -0.061 & -0.041 & -0.076 & -0.041 & -0.112 & 0.013 \\ -0.087 & 0.023 & -0.009 & -0.028 & 0.042 & 0.000 & 0.034 & -0.013 & -0.038 \\ 0.006 & -0.023 & 0.029 & 0.071 & 0.060 & -0.015 & 0.045 & 0.019 & 0.040 \\ -0.047 & -0.066 & -0.020 & -0.033 & 0.080 & 0.038 & 0.042 & 0.033 & 0.010 \\ 0.045 & -0.043 & 0.012 & -0.026 & 0.046 & -0.003 & 0.032 & -0.007 & 0.054 \\ -0.042 & -0.035 & -0.028 & 0.075 & 0.044 & 0.071 & -0.056 & -0.076 & -0.069 \\ -0.052 & 0.023 & 0.026 & 0.062 & -0.023 & -0.054 & 0.012 & -0.057 & 0.003 \end{bmatrix}$$

With these $K = 16$ ($k = 0, 1, \ldots, 15$) filters $\mathbf{w}_k \in \mathbb{R}^{3 \times 9 \times 9}$ and corresponding bias $b_k \in \mathbb{R}$ you should perform filtering on an input image from task 1. With each filter, $k = 0, 1, \ldots, 15$, you should perform filtering (cross-correlation, commonly mentioned as conv2d) as

$$\text{output}(k, i, j) = b_k + \sum_{c=1}^{C} \sum_{u=-4}^{4} \sum_{v=-4}^{4} \text{input}(c, i + u, j + v) w_k(c, u + 4, v + 4) \tag{2}$$

where input is the result from task 1 being an image of size $3 \times 64 \times 64$ that is normalized. Implement this filtering with zeros outside the image if the filter does not fit, i.e. zero padding with four zeros on each side. Hence, the output in Eq. (2) should be of size $16 \times 64 \times 64$

Task 3 - Rectify values and calculate average channel values

You should now be able to read an jpg image from Task 1, to get an image of size $3 \times 64 \times 64$ which is numerically re+scaled. Then this image can be filtered with 16 filters as described in Task 2 leading to an output of size $16 \times 64 \times 64$.

The first step here is to rectify, with a rectified linear unit, the $16 \times 64 \times 64$ result from Task 2 as

$$\text{output}(k,i,j) = \max\left(0, \text{input}(k,i,j)\right), \forall k,i,j \tag{3}$$

where $\text{input}(k,i,j)$ here is the output in Eq. (2).

The second step in this task is to average all the numbers in each channel, as

$$\text{output}(k,1,1) = \frac{1}{64 \cdot 64} \sum_{i=0}^{63} \sum_{j=0}^{63} \text{input}(k,i,j), \forall k \tag{4}$$

where input(k,i,j) now is the output from Eq. (3) and the output, $\text{output}(k,1,1), \forall k$ can be viewed as a single K-dimensional vector which we denote $\mathbf{o} \in \mathbb{R}^K$. You should be able to calculate this $\mathbf{o} \in \mathbb{R}^K$ vector, for an image, that will be used in the next task.

HINT:

Using Python there is usually a flatten operation, e.g. `numpy.ndarray.flatten`, to go from $K \times 1 \times 1$ to a K dimensional vector.

HINT:

In Matlab you can use the colon operator, (:), to form a single vector from a multi dimensional array.

Task 4 - Apply a linear classifier

You will find weights (`fc_weights.csv`), $\mathbf{w}_{fc} \in \mathbb{R}^K$, and bias (`fc_bias.csv`), $b_{fc} \in \mathbb{R}$, for a linear classifier. Read in the filter weights and the bias and calculate a score s for an image as

$$s = b_{fc} + \mathbf{o}^T \mathbf{w}_{fc} \tag{5}$$

where $\mathbf{o}$ is the output from Task 3, see Eq. (4), and then map this score (using a sigmoid function) as

$$p = \frac{1}{1 + e^{-s}} \tag{6}$$

to get the final value p that will be used to guess the class of the image in the next task.

Task 5 - Evaluate results, calculate accuracy

You should now be able to produce a value p, see Eq. (6), by running your the steps described in Task 1,2,3 and 4 on an image in a jpg file. Thus, you should now be able to produce 100 numbers, 50 for hotdogs and 50 for pancakes. As a decision you can introduce a threshold θ, on p, that says that if a number is less then then threshold ($p < \theta$) it will be considered a hotdog, and otherwise ($p \geq \theta$) it will guess pancake. Use a threshold $\theta = 0.5$ and find the number of correct guessed hotdogs (`# correct hotdogs`) (out of 50) and the number of correct guessed pancakes (`# correct pancakes`). Then calculate the **accuracy** as

$$\text{acc} = \frac{\texttt{\# correct hotdogs} + \texttt{\# correct pancakes}}{100} \tag{7}$$

and print out the result in your code.

HINT:

The accuracy should be higher than 0.6 but lower than 0.8 if you did everything correct.