Introduction

There are two parts to this assignment and the goals are:

(a) Texture matching for scene classification (60 points)

- Become familiar with extracting texture features from images using a bank of filters.
- Implement feature matching algorithm using the nearest neighbor classifier.

(b) Logistic regression implementation (40 points)

- To implement a logistic regression classifier from scratch, writing your own gradient descent technique for learning its parameters

Download the homework2.zip file from myCourses
`Contents`→`Programming_Assignments`→`Homework2`; this contains the instructions, starter code and data needed for the two parts of the assignment.

Requirements

You should perform this assignment using Python along with any image library or machine learning library of your choice. The assignment is due on **Sunday October 18th, 2020 by 11:59pm**. You are required to submit your code in two Jupyter notebooks (one for each problem) along with a single brief PDF report containing short write-ups based on the questions in the assignment. Your solutions should be zipped and uploaded to myCourses via Assignments before the deadline.

Your submitted zipped file for this assignment should be named **LastnameFirstname_hw2.zip** and should contain at least three files - *LastnameFirstname_hw2.pdf*, *LastnameFirstname_hw2a.ipynb* and *LastnameFirstname_hw2b.ipynb*. Feel free to submit any other auxiliary files as needed. We should be able to execute your code for each part of the assignment from your Jupyter notebooks. Include a `Readme` file if necessary (especially if using additional external libraries other than those given).

Failure to follow this convention could result in delays in getting your grade.

Problem 1. Texture Matching for Scene Classification (60 points)

The goal of this problem is to get you to become familiar with extracting texture features from images using filters, vector quantizing the texture responses (running k-means to cluster the responses and building feature vector histograms with the clusters) and lastly, labeling images using the nearest neighbor classification method.
We have provided some images in the folder `scenes`, and have split the data into training and testing datasets of 250 images each.

You also are provided with an auxiliary file to help create the Leung-Malik (LM) bank of filters. This LM filter set is a multi-scale, multi-orientation filter bank with 48 filters. It

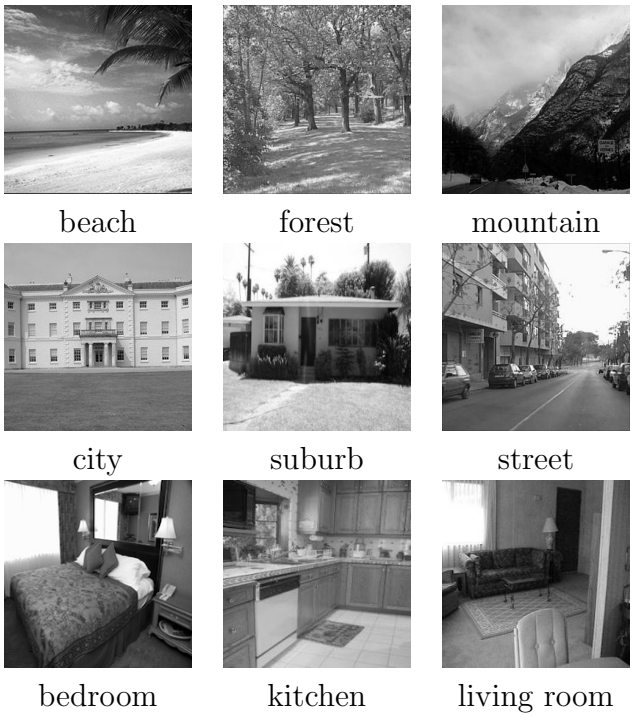

Figure 1: Sample images from 9 of the 10 classes.

consists of first and second derivatives of Gaussians at 6 orientations and 3 scales making a total of 36; 8 Laplacian-of-Gaussian (LoG) filters; and 4 Gaussians. We consider an over-complete bank where the filters occur at the basic scales $\{\sigma = \sqrt{2}, 2, 2\sqrt{2}, 4\}$. The first and second derivative filters occur at the first three scales with an elongation factor of 3 (i.e. $\sigma_x = \sigma$ and $\sigma_y = 3\sigma_x$). The Gaussians occur at the four basic scales while the 8 LoG filters occur at σ and 3σ. The filter bank is shown in Figure 2.
Use the Jupyter notebbok `Texture.ipynb` as the starting code for this Problem 1.

1) Examine the `Images` folder to understand the structure of the data. You are also provided with two files `train250.csv` and `test250.csv` containing the names and relative file paths of images. Each files contains ithe nformation for 250 images.
   Read in the files as grayscale and resize the images to 128×128

2) Apply the bank of 48 filters on the training images and similarly to the testing images. Hint: this procedure takes a really long time so test it on only a few images first before running on all images. Once you generate your filter responses on the training and testing images, take the absolute values and you might want to save the results in a python pickle file. *This step took roughly 40 mins for each set!*
   Note: *X_train shape: (250, 128, 128, 48)*; *X_test shape: (250, 128, 128, 48)*;

3) Vectorize the 250 training response data, so that
   Training dataset shape: (4096000,48);
   Max-min normalize each channel of the data

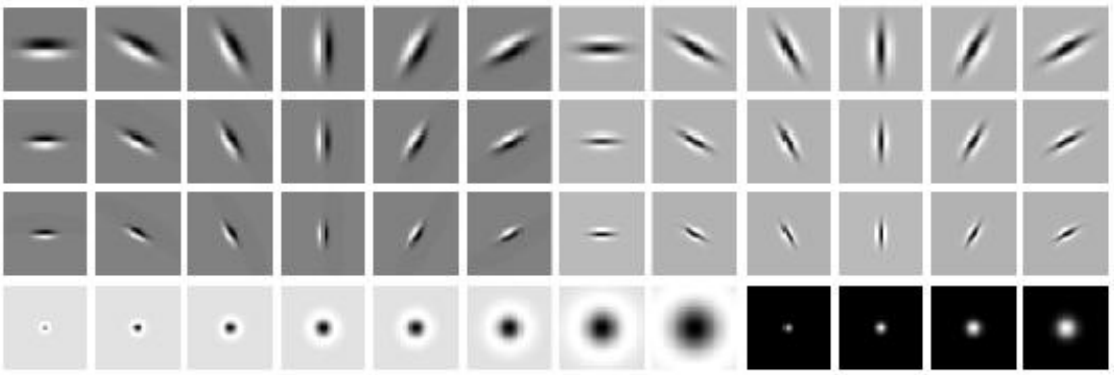

Figure 2: The LM filter bank has a mix of edge, bar and blob detectors at multiple scales and orientations. It has a total of 48 filters - 2 Gaussian derivative filters at 6 orientations and 3 scales, 8 Laplacian of Gaussian filters and 4 Gaussian filters.

Then run k-means clustering where $k = 500$ (recommended).
Again, pickle your k-means result to avoid doing this again.

4) Generate the frequency histogram (feature) for each image. Given an image response (already filtered) and the kmeans model, compute its frequency histogram. To do this, you may want to follow the steps below:

   a) Vectorize the response image

   b) Max-min normalize the response

   c) Predict clusters for each 48-dimensional pixel using the k-means predict function

   d) Generate the frequency histogram by incrementing the cluster centers where each pixel belongs

   e) Normalize the resulting frequency histogram so that it sums to 1

5) Implement the "nearest neighbor classifier" by comparing each image histogram in Testing with the reference training images, to generate a label (closest match). For comparing 2 histograms, use the chi-squared distance measure:

$$\chi^2 = \frac{1}{2}\sum_{k=1}^{K}\frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)}$$

   where K is the total number of histogram bins, $h_i(k)$ is the value at bin k for histogram h_i and $h_j(k)$ is the same definition, but for histogram h_j

6) Generate a class confusion matrix for the 10 classes and print out your accuracy.

7) In your PDF report, include your confusion matrix, your accuracy score and discuss which classes were more easily recognized by the algorithm and which ones got confused with each other. Discuss some potential reasons for this.

BONUS - 10 points: Using the same train and test data split, implement any classification technique to obtain $\geq 48\%$ accuracy. My accuracy was about 44%.

Your submission for this part should contain the all the code (including any auxilliary files) used to address this problem. Try to keep everything in the Jupyter notebook. We will run your code on our own different train/test image split. Your PDF report should contain the discussion requested above.

Problem 2. Logistic Regression Implementation (40 points)

For this second problem, we will perform binary classifications of facial expressions using a logistic regression classifier. The main goal is to successfully write the `predict` and `train` functions for the classifier. The dataset we are using is quite noisy and was obtained from the 2013 Kaggle facial expression classification competition[1]. Although the original dataset contained 7 emotional classes (six of which are shown in Figure 3), we will only use two of them (happy or positive valence and sad or negative valence) for this assignment.

Figure 3: The top row shows three examples of correctly labeled faces from the Kaggle challenge; left to right - angry, disgust and fear. The bottom row shows three incorrectly labeled faces; left to right - happy, sad and surprise. Neutral face is not shown. The dataset is quite noisy with several wrong labels or non-face images

The Data Files

You are provided with two data files, where the first file `fer3and4train.csv` contains the training data with 12,066 data samples and the second file `fer3and4test.csv` contains 2000 data samples that you will be testing your classifier on. Results should be reported on the test dataset. The files were created using `fer2013.csv` from Kaggle but have been shuffled and augmented to avoid the class imbalance problem.
All the files are stored as comma separated files with three columns each. The first column contains the label of the emotional expression, where 3=happy and 4=sad; the second column

[1] `https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-\challenge/data`

contains 2304 integer values (between 0 and 255) obtained by vectorizing 48×48 grayscale images of faces; and the last column states in which part of the development process the data should be used (i.e training or testing).

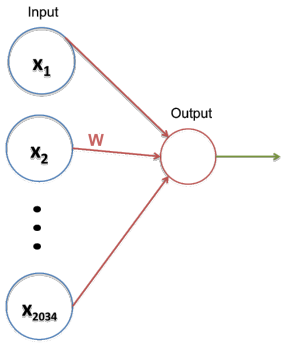

Figure 4: The logistic regression classifier

The file `question2.ipynb` contains the skeleton code for your logistic regression module. This is accompanied by an auxiliary file called `helper.py` containing the names of some helper functions useful for your module to run. In the provided Jupyter notebook, different parts of the code have been marked for your implementation. First load the training data samples $\mathbf{X}$ and their corresponding class labels Y using the helper function `getBinaryfer13Data`. Then use the class object to train the data. Call the `train` function to learn the weights and bias of the unit. The following occur within the `train` function:

i Initialize the weights W to small random numbers (variance - zero); also initialize the bias b to zero. See the helper file for examples of how to do this initialization, specifically the function `init_weight_and_bias`.

ii Create a loop over the number of epochs specified. Within the loop, the following occur:

iii Call a `forward` function to calculate the predictions referred to as pY. The `forward` function implements $\sigma(\mathbf{X} \cdot \mathbf{W} + b)$. The argument of this equation can be implemented in *numpy* as $\sigma(np.dot(X, W) + b)$ where σ is the sigmoid activation function; this is provided also as a helper function.

iv Next, learn the weights via back-propagation, by performing gradient descent using the equations below:

$$W = W - \eta \frac{\partial J}{\partial W}; \qquad W = W - \eta \cdot (\mathbf{X}^\top \cdot (pY - Y)) \tag{1}$$

Note: When doing matrix computations, the product of the vectors $X^\top Y$ can be written as $np.dot(\mathbf{X}.\mathrm{T}, \mathbf{Y})$; Also,

$$b = b - \eta \frac{\partial J}{\partial b}; \qquad b = b - \eta \cdot (pY - Y) \tag{2}$$

v Apply the forward algorithm to predict the new labels for both the training and validation data. Compute the training cost and validation cost at each epoch and append to a growing array (one training, the other validation). Keep note of the best error value on the validation data.

vi Print out the best error value on the validation data. Provide this value in your final report.

vii Plot your training and validation costs to show how the errors are changing over time. See Figure 5. Display this in your report.

viii Lastly load the test data from the file `fer3and4test.csv`. Compute and print the accuracy (or classification rate) for predicting this new dataset from your trained network. Provide this information in your report.

BONUS - 10 points: Implement the L1, L2 or ElasticNet regularizer with your code (no need to write a new `train` function, just add it on) and state your new accuracy in your PDF report.

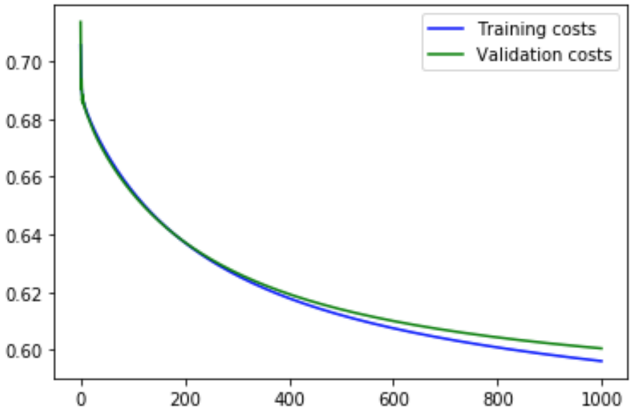

Figure 5: An example of the loss curves from the logistic regressor after 1000 epochs. My accuracy on the test set was 70.5%

As in Problem 1, your submission should contain the all the code used to address this problem. We will run the code on other test images. Your PDF report should contain your responses to any questions/discussions solicited in the problem.

This assignment is due on **Sunday October 18th, 2020 by 11:59pm**.