Question 1: Classifying Scenes

On the course homepage you will find two files: scenes.zip and scene_classifier.c. Download both files (note: scenes.zip is 100 MB large) and unzip scenes.zip in such a way that the directory “scene_categories” is in the same directory as scene_classifier.c. If you want to put it somewhere else, just change the relevant line in the function createExemplars in scene_classifier.c so that it finds the scene images.

As with the other programs, scene_classifier.c needs to be compiled together with the netpbm.c file. It implements a multi-layer neural network that uses backpropagation learning to classify scene images. As we discussed in class, the image database contains 8 scene categories, with 260 images for each category. We will use 200 images for training and 60 for testing in each category. I tried very hard to make the code readable, the variable names meaningful, and provide enough comments to explain what the code does. As you can see, implementing the network does not really require much code or many equations.

In order to be as efficient as possible, the program first loads each image, computes the image features to be fed into the network, and then stores all features in a Matrix structure named exem, which makes sense because each row holds one of the exemplars. As usual, an exemplar consists of a set of input values (features) plus the desired output value, which here is the category of the image, identified by a number from 0 to 7. Basically, each exemplar tells the network: “For these input values (image features), the correct output is this category.” The network is shown these exemplars in random order, and whenever its output deviates from the correct (desired) one, the network adapts its weights a little bit so that it improves its performance. Once the network has seen each training exemplar exactly once, we say that one epoch has been completed. After each epoch, we show the testing exemplars to the network and see how many of them it classifies incorrectly. This error rate is a good measure for the performance of the network. The program performs an adjustable number of epochs and determines the best classification performance that was reached during this process. After each epoch, the
learning rate $\eta$ is multiplied by a factor $< 1$ to slowly reduce it. Most textbooks do not mention such a reduction of $\eta$, but in my experience it helps to get better results. You can experiment with it to determine the best network performance.

Your task is to improve the classification performance of the network. You should do this in two ways: First, find appropriate image features that enable the network to do a good classification job. In the initial scene_classifier.c code, the function getFeatures computes only two features, and they are not very good. They are simply the average intensity of all image pixels and its standard deviation. Both feature values are scaled to be in the interval from 0 to 1. As you can see when running the program, after 10,000 epochs, when using these two features, the network reaches a minimal error rate of about 72%. If the features were completely uninformative with regard to classifying the scenes, we would expect the network to get 1 in 8 classifications right, just by chance. This would correspond to an error rate of 87.5%. This means that the two features do contain some useful information, i.e., there are systematic differences in intensity across scene categories. However, this is not at all informative enough to make useful classifications.

So you should compute different features, or you can still use these two and add more features. Increase the value of INPUTS accordingly, and modify the getFeatures function to put the new features into the features Matrix. You should be very creative here. For example, maybe the amount of straight lines in the image is a good feature to distinguish between categories. Or perhaps the presence of high spatial frequencies, as determined by a Fourier transform? Or the presence of a sky, i.e., of pixels being brighter at the top of the image than at the bottom? There are tons of possible features, and you can compute and include as many as you like. However, remember that adding a lot of irrelevant features could actually decrease the network performance. It would be easiest for me if you copied all functions that you use into the file scene_classifier.c right before the getFeatures function. You can use all code that is on the course homepage or write your own code. I will upload more code snippets to the homepage within the next few days. You can also ask me for code for algorithms that we discussed in class; I may have it and can send it to you after making it a bit more readable.

Second, you can play around with the network parameters that are set by the #define lines at the top of the file. In particular, the number of hidden-layer units is worth experimenting with, or the initial value or decay of the learning rate. The current start value of 4.0 is rather high but works well for the two chosen features. The number of epochs could be modified, too. When I run your code later on, I will only allow it to run on my computer, a reasonably current laptop, for a maximum of 10 minutes. If it seems that it will run longer, I will reduce the number of epochs. If your computations take a long time, you can also temporarily change the values for TRAINING and TESTING from 200 and 60, respectively, to, for example, 20 and 6, so you can get some idea how good your features are working with in only 10% of the time needed for a full run.

Please only modify the code at the top of the file, until the end of the getFeatures file. You do not need to change anything else, and it would make it easier for me to test your code.
And do not forget: The author of the program that performs the best, on a new set of testing exemplars, will win a $50 gift certificate!

Please email me any questions that you may have, and let us talk about it more on Tuesday next week. Please try to get the code to run on your computer soon so that we can make sure that you can enter the competition.

**Question 2: Signature, Please!**

Use the diagram on the right to sketch the signature of the contour on the left. It does not have to be perfectly scaled.

![Signature Diagram](image)

**Question 3: Stereo Vision with Paper and Pencil**

On the course homepage, you will find a picture named gary.jpg (under “Software”), as shown on the following page of this assignment. It is a stereo image pair. When you look at it, try to cross your eyes (your visual axes, to be precise) so that you fuse the two images into one. If you keep them fused for a second or two, you will start noticing how your perception of the image changes – you are now seeing it as three-dimensional. For example, you will see that Gary’s soprano saxophone seems to come out of the picture and point towards you. This is not surprising as this picture was taken with a stereo camera using two lenses with a certain stereo baseline between them, coplanar image planes, and identical focal lengths.

So when you let your left and right eye look at the picture taken by the left and right camera, respectively, then your visual system can reconstruct the 3D information based on the binocular disparity between the images. Notice, however, that in order to allow you to use the cross-eyed viewing technique, the images are swapped, i.e., the image taken by the left camera is shown on the right side and vice versa.
Unfortunately, we do not have the time to develop and implement a stereo matching algorithm, so you will have to do the computations yourselves, using your own visual system, paper, pencil, and a calculator. Well, and you will also need some common image editing software such as Windows Paint or GIMP. You will use it to measure the coordinates of pixels in the image in order to compute the disparity \((x_1 \text{ vs. } x_1')\).

We do not know the exact parameters of the stereo camera, but let us assume the following: The stereo baseline is similar to the separation between the human eyes, which is 6.5 cm. The width of each of the two pictures is 5 cm. Let us consider the points \(A, B, C, D,\) and \(E\) in the scene as shown in the figure below. Let us assume that point \(A\) has a depth (= distance to the image plane) of 2 meters.

a) Determine the focal length \(f\) of the camera.

b) According to this value of \(f\), determine the depth of points \(B, C, D,\) and \(E\) in the stereo image.

Please write down all the steps of your calculations.
Question 4: Trouble with the Stereo

For certain types of repetitive patterns, your stereo vision may be tricked into believing that the pattern is closer to you or further away from you than it actually is.

(a) Under what conditions can this occur?

(b) Let’s say that the actual pattern is 50 cm away from you and repeats itself every 3 cm. The optical centers of your eyeballs are 6.5 cm away from each other. Compute four other distances (two greater ones and two smaller ones) at which you could perceive the pattern to be located.

Question 5: Optical Flow

Please draw the optical flow patterns for different types of camera motions by using many small arrows whose length indicates the local speed of motion. For example, if a camera is placed on a moving train and is facing to the right, the motion pattern may look like this:
The reason for this pattern is that there is only slow motion relative to the camera for objects that are far away, and faster motion for objects close by. Please illustrate the optical flow patterns for the following situations in a similar way:

(a) The camera is moving forward (i.e., you are pointing the camera in your direction of motion while walking forward).

(b) The camera is rotating clockwise along its vertical axis (e.g., while holding the camera you are turning your body to the right as if tracking an object that is moving rightward).
(c) The camera is rotating clockwise along its visual axis (e.g., while holding the camera you are turning it clockwise so that you are switching from “landscape” to “portrait” orientation of the video).

(d) Just like (c), but now you are walking forward while rotating the camera.