CMSC 828L, Problem Set 2
Spring 2022

Assigned: Tuesday, March 1
Due: Tuesday, March 15, before midnight.

The goal of this problem set is to implement a framework for producing CNNs, and then use this framework to create CNNs that solve three classification problems. You only need to handle problems that have a 2D input with a single channel, and that have a single output value, so that they can handle two-class classification problems.

We are asking you to implement several functions, but how you organize your code is up to you. You will implement everything from scratch. We want to be able to look at and run your code, just to see that everything is your implementation. That is, you can't use Pytorch or any other deep learning framework. You can use standard numerical packages (eg., numpy) and packages to read in the images (eg., Imageio).

You will implement the following functions:

- create_model(input_size). This takes one arguments and returns a model. This allows you to do any prep that you need. The input size is a vector of two values, giving the dimensions of an input image. You can assume the input is a 2D image with 1 channel.
- model = add_conv_layer(model, num_channels, filter_size, activation, T, b)
  model is just the model you are adding a layer to.
  num_channels indicates the number of channels in the output of this layer. This is the number of new filters you will have.
  filter_size is a 2D vector that indicates the size of each filter.
  activation indicates a non-linear activation that follows convolution. You only need to consider two cases, either activation = 'relu' or activation = 'none'.
  T is a tensor that contains all the filters. It depends on the size of the filters, the number of input channels to this layer, and the number of output channels.
  b is a vector that contains a bias term for each filter.
- model = add_pooling_layer(model, dim, type)
  This function adds a pooling layer to your model.
  dim is a 2D vector that indicates the size of the pooling.
  type is either 'max' or 'avg' to indicate max or average pooling.
- model = add_FC_sigmoid_layer(model, b, T)
  Every network will have a fully connected layer at the end that will connect all units in the final layer with a single output unit. This unit will then apply a sigmoid function so that the network output is a value between 0 and 1. This indicates the probability that the input image belongs to class 1.
  b is a bias term that is applied before the sigmoid.
  T is a tensor that contains all weights from the final layer to the output layer.

Figure 1: Task 1. Left: Class 0. Right: Class 1

You will use this code to solve some classification tasks. There are three tasks given below. You are expected to solve any two of them. You may solve all three for extra credit. Task 1 has a regular and extra credit component. For each task, we will provide a set of images for each class, and a brief description of how these images are created. You will create one network for each task, providing the network weights and biases by hand. You must use the code that you have written for the above functions to create your networks. You should create a function:

perform_classification(task, images)

task should indicate which task is being performed. task should equal 1, 2, or 3 for the three tasks, or 4 for the extra credit version of task 1.
images should be a 3D array that is kxkxN, containing N images that are kxk.
The function should return a binary vector containing the predicted class of each image.

For each task, provide a brief description of how your network solves the problem, relating this to the weights you have come up with. It is not allowed to use some other framework, such as Pytorch, to train networks to solve these tasks, and then copy the weights. You must be able to provide an intuitive description of what your network is doing. You will probably want to keep your network as simple as possible.

We provide you with six image sets, named PS2_Images_Task_i_Class_j. i will be value from 1 to 3 indicating the task, j will be 0 or 1, indicating the class.

You can receive most of the credit on this problem as long as you demonstrate performance that is significantly above chance. More credit may be given to assignments that achieve high performance.

You should submit a zip file containing your code in one file and your writeup in another. Your writeup should clearly explain how you solved each task.

Task 1

In this task, we generate an image in which each pixel consists of IID Gaussian noise. Then the image is smoothed by two different amounts to create images in class 0 and class 1. The images are then randomly scaled by a value chosen from a uniform random distribution. They are also translated so that their maximum value - their minimum value is halfway between 0 and 255. Figure 3 shows sample images from each class.

In doing this task, you are allowed to preprocess the input images so that they all have the same scale. That is, you can produce $\frac{I}{\|I\|}$, where the norm is the Frobenius

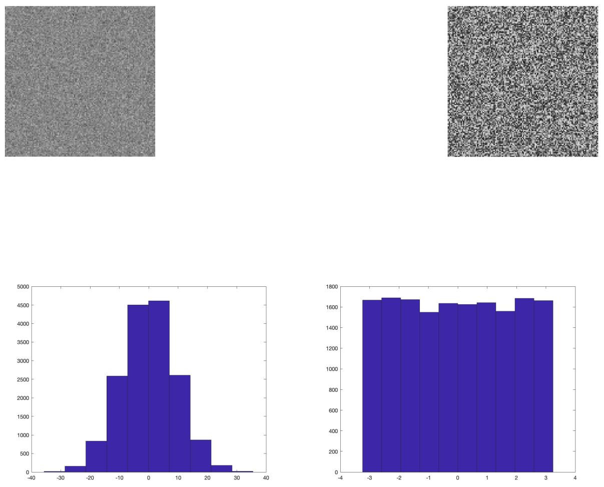


Figure 2: Task 2. Left: Class 0, image and histogram. Right: Class 1, image and histogram.

norm.

For extra credit, solve this problem without normalizing the images.

Task 2

In task 2, we also generate images in which each pixel is IID noise. In class 0, the noise has a Gaussian distribution, in class 1, the noise has a uniform distribution. In each case, the image is scaled by a scale factor drawn from a uniform distribution, just as in the previous task. Figure 2 shows an image from each class and the histogram of each image. You are not allowed to normalize the images before they enter the network.

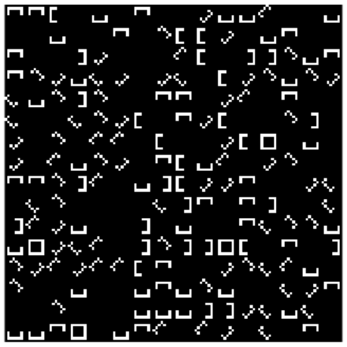
Task 3

In this task, images from each class contains some randomly distributed fragments of squares and diamonds. In addition, images in class 0 contain some random number of complete squares, while images from class 1 contain some random number of complete diamonds. Note that it is possible that the random number of squares and diamonds is 0, so it is not possible to achieve 100% performance on this task.

Hint

Just to get a sense of how you might do this, let's consider a simpler problem. Consider two classes. One has images with IID Gaussian noise that is mean $\frac{1}{2}$ and with variance 1. All values are then clipped so that they lie between 0 and 1 (that is, for every pixel value, x, take $\max(0, \min(1, x))$. The other class contains salt and pepper noise. That is, each pixel is 0 or 1 with equal probability, and the pixels are independent.

To solve this we could apply a filter that is the identity (one in the middle and 0 everywhere else). We could then give this filter a bias of -.99. This makes all values below .99 equal to 0, and all other values between 0 and .01. We then apply global average pooling, which adds up all. these values. Roughly, this allows us to count the number of 1s in the image. There will be many more 1s in the salt and pepper noise than the Gaussian noise. So, applying an appropriate bias in the final layer, we can create an output that is close to 1 for the salt and pepper class, and close to 0 for the other class.



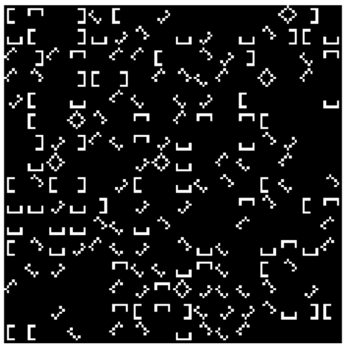


Figure 3: Task 3. Left: Class 0. Right: Class 1