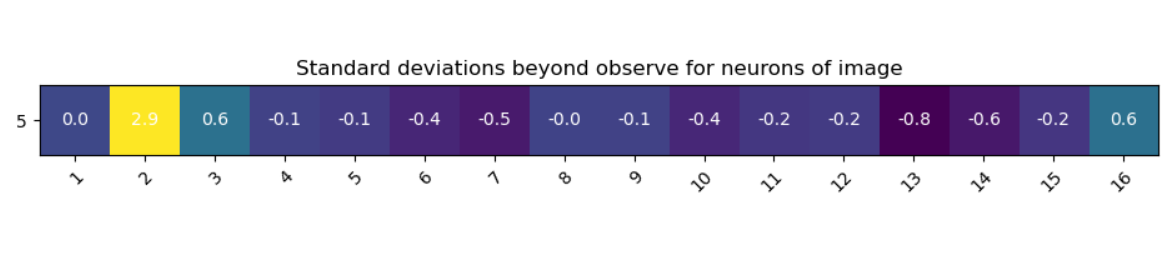
Assignment:

1. First, let’s visualize the activation differences for each sample based on the mean and stdev of classes we calculated last week. We will visualize using heatmaps. If you can hack keract’s heatmap visualization, that’s fine, or you can use matplotlib’s function (<https://matplotlib.org/3.1.1/gallery/images_contours_and_fields/image_annotated_heatmap.html>). Each pixel will represent a neuron, and its value how many standard deviations away the sample’s activation is from the class mean.



1. Let’s formalize this metric a little bit more, using better statistics. Figuring out individual stdevs is great and all, but what if some neurons are highly correlated? (Not surprising in an image—neurons are usually similar to those around them.) So if one neuron is outside the norm, others **nearby** will likely be as well. Should these be counted separately as independent events? How to adjust for correlations in the features? To address these issues, let’s explore Mahalanobis distances:
   1. <https://www.machinelearningplus.com/statistics/mahalanobis-distance/>
   2. <https://www.youtube.com/watch?v=spNpfmWZBmg>
   3. Calculate Mahalanobis distances for your adversarial examples against the class distributions, i.e instead of calculating how far away adversarial activations are from the mean in terms of the stdev, use Mahalanobis to calculate distance.
      1. So use activations from benign class and activations of the one image to output one value for the image, which will then be compared to a chi-squared distribution to output a p-value for how well that image’s activations fit the distribution typically experienced by that class. When I calculated a p-value with the non-perturbed image, the p-value was .202, meaning that the image was not rejected from the class. The p-value for the perturbed image was 0.0
   4. Hint: to calculate the covariance matrix of the training samples in each class, you can use numpy’s cov() function. However, to prevent long computation times and numerical underflow errors, you may want to **transform the data first into an uncorrelated bases** (brush off that linear algebra) through PCA. You don’t necessarily need to do the full bases; truncating the number of features essentially reduces the dimensionality of the problem space. Use sklearn’s PCA function and its explained variance parameter to pick a reasonable truncation point, e.g. if the first 10 axes explain 97% of the variance, try using just the first 10 axes.
   5. If you use PCA to transform the data, then by definition the bases are uncorrelated and the covariance matrix reduces down to a diagonal matrix of its eigenvalues, which is much easier to work with (and calculate!)
   6. If you’re rusty on your linear algebra/PCA, feel free to learn more about it! <https://www.youtube.com/watch?v=FgakZw6K1QQ>

From PCA, we see that 50 dimensions of measure explain 95% of the variability experienced by all classes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Neurons | Percent of Variability |  | Number of Neurons | Percent of Variability |
| 10 | 73.5% |  | 60 | 96.4% |
| 20 | 85.3% |  | 70 | 97.5% |
| 30 | 90.2% |  | 80 | 98.3% |
| 40 | 93.2% |  | 90 | 98.9% |
| **50** | **95.1%** |  | 100 | 99.3% |

Let’s break this down by class:

Class of fives: three of the 128 transformed neurons explain 51% of variability. 50 of the neurons explain 97.6% of the variability. Choosing to acknowledge 95% of the variability, we will use 35 dimensional measures.

We have one adversarial image that was perturbed by Deep Fool, perturbation of 0.1, to be mis-classified as a three. We will transform this image into our new PCA space then calculate the Mahalanobis distance.

1. Compare your Mahalanobis distances to the simpler mean/stdev work from last week. Do the Mahalanobis distances say anything different? Do you get more nuance? One big benefit is that Mahalanobis distances are what are known in statistics as being chi-square distributed (<https://www.khanacademy.org/math/statistics-probability/inference-categorical-data-chi-square-tests/chi-square-goodness-of-fit-tests/v/chi-square-distribution-introduction>). These distributions are great, because we can actually assign probabilities to the numbers now, much like normal distributions have a 3% chance of values falling outside of 2 stdevs of the mean. Use scipy.stats.chi2.ppf to find critical values for our degrees of freedom (dense=128? Not sure if the function goes this high, but let’s try.) We often use p values < 0.05 for significance (see tutorial in 2.a above), so try that value of p. Can we detect adversarial examples as significantly out of range of the distribution?
2. Finally, as this is our last week, make sure you document, document, document. Put all these code and answers on your github. Bonus: **create a short 5-10 slide presentation** on what you’ve done this past month; you never know when you’ll be asked to give a short technical presentation, and having one ready to go is a huge plus. Prepare it now while it’s fresh in your memory!