**Week 3**

Play around with this to get a feel for changes in activation patterns that occur with adversarial examples. In particular:

1. Try different pairs of attacked vs benign classes
2. Play around with adding different layers
3. In particular, view the very last ‘activation\_15’ layer. Given this is the logit layer, does the change in activation values correspond with the change in classification?
   1. The change in activation values does correspond with the change in classification. Especially in the last layer you can see that the correct class’ classification is greatly decreased and the targeted class’ activation is increased.
4. What is the red tick mark in the layers vs the black one? (read Neural Divergence paper if necessary—link in website)
   1. The red tick mark in the different layers corresponds to the attacked image’s activation while the black is for the benign image.
5. Do earlier layers exhibit more or less changes than later ones?
   1. After analyzing all the layers in the model, earlier layers seem to exhibit less changes than the later layers. This applies to all of the classes and combinations of attacked/benign images.

Assignment:

1. Play around with Neural Divergence and answer the questions.
2. Download and install the Keract module:
   1. <https://github.com/philipperemy/keract>
   2. (earlier versions work with TF 1 if that’s how you’re rolling).
   3. Read through the README to get a sense of what it does and how to use it.
   4. Read up on it: <https://www.machinecurve.com/index.php/2019/12/02/visualize-layer-outputs-of-your-keras-classifier-with-keract/>
3. Try out the MNIST tutorial in that last link (point d above). Either add in the 2 Conv2d layers in the front of your model as in the tutorial, or try to get heatmaps to work on the dense layers, to see what’s being activated. It’s especially neat to use with convolutional layers, because each convolution is looking for a particular pattern. This helps to visualize what each convolution finds: a curve? A straight line? What is being triggered in the image? Save a few images of your heat map and send to me or upload to you GitHub.
   1. After following the tutorial above, two convolutional layers were added to the model giving it the following structure.
      1. model=Sequential()

model.add(Conv2D(6, kernel\_size=(5,5), activation='relu', input\_shape=input\_shape))

model.add(MaxPooling2D(pool\_size=(2,2)), name='maxpool')

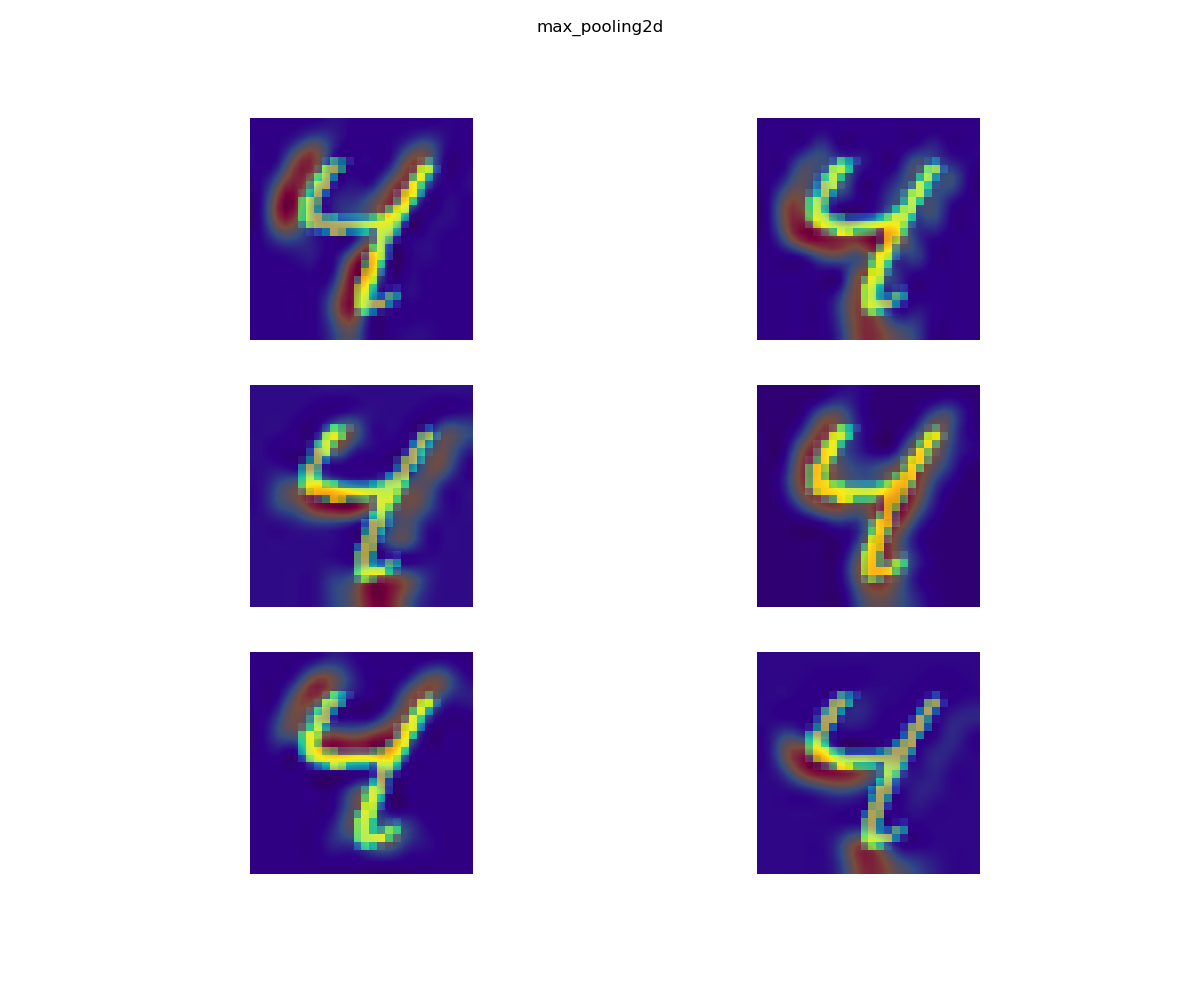
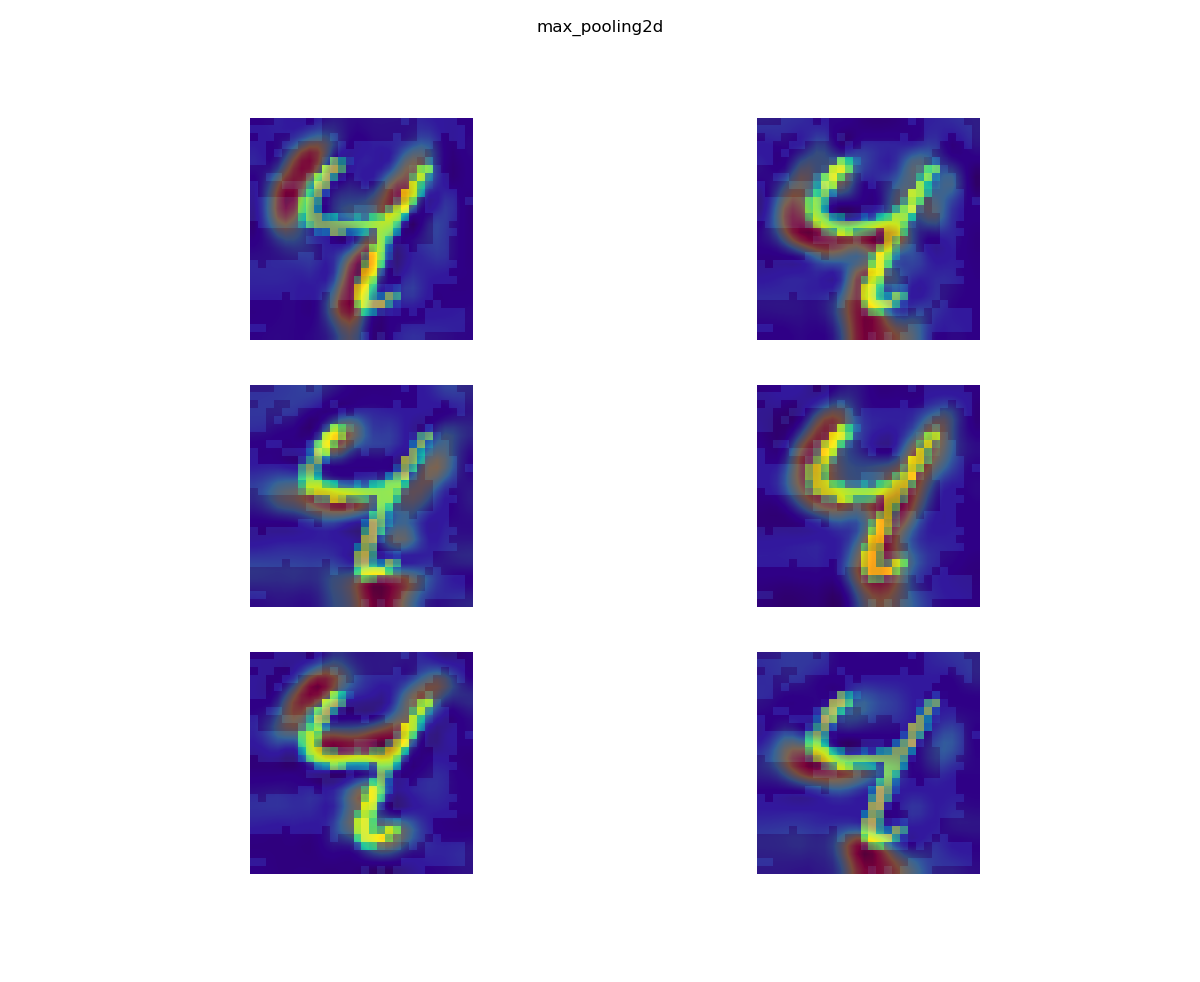
model.add(Conv2D(10, kernel\_size=(5, 5), activation='relu'))

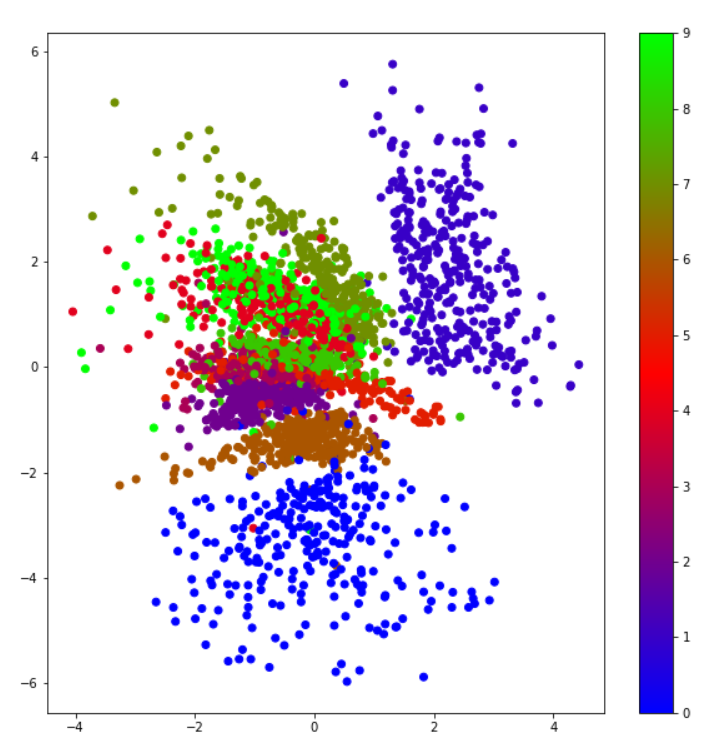
model.add(Flatten())

model.add(Dense(128, activation='relu', name='dense128'))

model.add(Dropout(0.2))

model.add(Dense(no\_classes, activation='softmax'))

1. Try to visually see differences in the heatmap activations of a benign sample vs its adversarial example. Use the adversarial examples you generated last week. What changes in the model’s detection? How does what it detects help determine the ultimate classification?
   1. After visually comparing heatmap activations for a benign 4 example and an adversarial 4 example classified as an 8, it is still difficult to see how the computer determined that classification. One difference in activation is found with the second photo in the first column. It seems that in the adversarial example, the right vertical line is not as strongly detected. It also seems that all of the curves are highly activated (8’s are clearly all curves instead of straight line segments like 4’s). This example aligns with the plot of the MNIST decision boundaries. You can see that the closest number you can perturb a 4 to is actually an 8 like in this example.
   2. 
   3. 



1. Let’s formalize this difference a little more with some quantitative analysis:
   1. Group MNIST examples by label (numpy’s where() function may be useful, as in zero\_indices = np.where(labels == 0)[0]; for ex in dataset[zero\_indices]…)
   2. For each class, run all examples through Keract and record the 128 values of the Dense layer for each.
   3. Calculate the mean and standard deviation for each neuron of the Dense layer for each class, e.g. for neuron 1 of the Dense layer for class ‘4’, the mean activation value is 3.6 with a stdev of 1.2.
      1. The mean and standard deviation for each neuron in the 128 Dense Layer were calculated using 1000 examples of each class. The results are stored in ‘mnist\_mean\_actv1000’ and ‘mnist\_stdev\_actv1000’.
   4. What’s the average number of neuron activations of a normal sample that fall outside of one standard deviation from the mean? E.g. on average for all the benign training samples, each sample can expect to have 3 neurons out of 128 of the dense layer whose activation does not occur within 1 stdev of the mean.
      1. On average for all the benign training samples, each sample can expect to have 53 neurons out of 128 in the dense layer whose activations does not fall within 1 stdev of the mean. This was calculated by summing the neurons that fell outside one standard deviation from the mean of that example’s class and then dividing by the total number of neurons. 530,920 of the total 1280000 neurons tested (10,000 samples) fell outside one stdev of the mean, or 41.478125 %.
   5. Now do this for your adversarial examples. Make at least 100 adversarial examples for MNIST (10 for each of the digits, any attack method is fine), and find the average number of neurons whose activation is outside of 1 standard dev of the mean, for BOTH the original class and the attacked class. In other words, if an image was originally ‘5’ but gets attacked to be classified as ‘3’, use both the original ‘5’ mean and stdev, plus the attacked class ‘3’ mean and stdev. Is the adversarial example have activations outside the norm more often for both classes? Just one? By how much? Does this number change with smaller perturbations (eg L2 norms?)
      1. The 100 examples were arranged consecutively with ten examples of each class in order. First, the number of neurons out of bounds for the 100 benign examples was calculated, then the number of neurons out of bounds for the adversarial examples was calculated (if a 5 was misclassified as a 3, it was with respect to the mean/stdev of the 5 class), and finally, the number of neurons out of bounds for the adversarial examples with respect to the class they were misclassified as were calculated (so if it was a 5 misclassified as a 3, the mean/stdev of the 3 class was used). We can see that the adversarial examples actually had more neurons fall within 1 standard deviation of the mean than benign which is quite interesting. As you decrease perturbation, more neurons fall outside of the first standard deviation from the mean.

|  |  |  |
| --- | --- | --- |
| 100 examples type | Total Neurons (of 12800) ‘Out of bounds/outside 1stdev’ for DeepFool eps 0.1 | Total Neurons (of 12800) ‘Out of bounds/outside 1stdev’ for DeepFool eps 0.03 |
| benign | 6016 (47%) | 6016 (47%) |
| adversarial | 5853 (45.7%) | 5987 (46.8%) |
| adversarial with respect to what they were misclassified as | 5169 (40.4%) | 5307 (41.5%) |

* 1. We’ll be using this values to calculate the Mahalanobis distance from a given point (an adversarial example) to a distribution (the range of activation values seen in the training data). But that’ll be for next week, since I think this is enough for now!
  2. Bonus: implement the MNIST model in the tutorial in 2.d above, and also calculate mean/stdev values for the Maxpooling layer in addition to the dense layer right before the 10 value logit dense layer at the end.

**Additional Experimentation**

1. While analyzing the mean activation for each class, I saw that roughly 12 neurons had zero activation for any of the classes. I wondered if this was a result of the dropout layer and calculated the mean again without the dropout layer in the model. After there were still roughly 12 neurons with zero mean activation across the board. This data was stored in ‘nodropout\_actv1000.’ I wondered if perhaps the model was able to classify images without needing all 128 neurons, so I calculated the mean activations again but this time only using 116 (assuming 12 of 128 were unused). There were 10 of the 116 with zero mean activations for all of the classes still. This data is stored in ‘lessneurons\_actv100.’ This could be a result of using Relu as the activation function. Some neurons can ‘die’ during training and will be 0 from that point on. LeakyRelu is an attempt to fix this ‘vanishing gradient’ problem, but many people still prefer normal Relu. Relu computes faster, and it is hypothesized that it can introduce ‘optimal brain damage’ in a sense (like dropout) to prevent overfitting. Given that the accuracy of the model in this experiment is still quite high, it seems to not be a highly significant issue, though the reasoning behind certain neurons with zero mean activation is an intriguing mystery.