COMP137 Report: Neuron Activation Pattern Monitoring on DenseNet

Ethan Oliver

Introduction

Currently most machine learning models are black boxes. A typical image classifier network takes in an input image and outputs a set of probabilities for what class that image falls into. Over the years these computer vision models have gotten more accurate but they have also gotten more complex. A human user can no longer tell how the network makes decisions and whether or not they are trustworthy decisions. These complex architectures make it easy for bias to enter the model and make it more difficult to fix errors. As machine learning models move outside the laboratory and into the real world these problems only become more pressing.

When a classification network makes a decision, an important question to ask is whether that decision was made based on information it learned from the training set about that class or if it is attempting to classify something outside the bounds of the training data. If the network is classifying something novel it is more likely to make a problematic or incorrect decision so catching these occurrences is important in safety critical systems such as self-driving cars. One way to do this is through neuron activation pattern (NAP) monitoring. The details behind the monitors architecture will be discussed later. During training the monitor learns all the activation patterns of a given layer for each class. When the network is then deployed and classifies an image, the monitor can then test whether the activations for that image fit within the activation patterns of the chosen class. This new information can be used to decide whether to accept or reject the decision made by the network.

The goal of this project is to design a NAP monitor in python and then implement this NAP monitor on an image classification network trained on Tiny Imagenet. Performance of a NAP can be measured with three metrics: trigger rate, sensitivity, and pattern probability. Trigger rate is the rate the NAP finds an image out of pattern from its class. Sensitivity is how well the NAP detects a misclassification. Pattern probability is the probability that an out of pattern image indicates that the image is misclassified. These metrics will be used to evaluate the performance of the system.

There are many hyperparameters that can affect the performance of a NAP monitor: # of neurons, negative positive split, hamming distance, thresholding. The hypotheses and results for varying these parameters can be found below. By careful tuning of these hyperparameters we hypothesize that we can create two high performance monitors. A silent monitor that triggers at less than 5% but has a sensitivity of 10% and loud monitor that triggers less than 10% but has a sensitivity of 25%. We were able to create a monitor that triggers 5.04 % with a sensitivity of 10.8%. However we were not able to find suitable hyperparameters to design a lous monitor the best result for this task was found to be a monitor with a trigger rate of 17.61% and a sensitivity of 27.97%

Related work

This project builds heavily on the work done by Cheng, Nuhrenberg, and Yasuoka in their paper "Runtime Monitoring Neuron Activations."[1] Their paper defines the concept of a NAP monitor and introduces the ideas of a gradient based selection of neurons as well as the hyper parameters of hamming distance and # neurons. The paper tests the NAP monitor on the MNIST and GTSRB datasets which have 10 and 43 classes respectively. We implement and expand their ideas to a more complicated dataset with 200 classes and a more complicated model that contains 10x as many activation neurons to select from. We also implement and test two new hyperparameters (split and thresholding) and their effects on NAP performance.

Background

Shared Reduced Ordered Binary Decision Diagrams were proposed by Randal Bryant in 1991.[2] BDD is a data structure used to represent boolean functions. It is a compressed representation of many different sets and relationships of binary variables. The data structure can be thought of as a directed acyclic graph where each node represents a variable and the end terminals are the output of the function. By changing the ordering of the variables one can reduce the size of this tree so reduced order BDD are typically smaller and more efficient then their general counterparts. The BDD implementation proposed by Bryant is used in this project to efficiently store the large sets of activation patterns as boolean functions.

DenseNet is a convolutional neural network proposed by researchers at Cornell and Tsinghua University and Facebook AI Research. [3] The network is densely connected which means the activations from each successive layer are passed on entirely to the next layer through concatenation. This allows the network to be thinner and more compact while still performing well. It also allows the error signal to propagate to earlier layers more easily as the first layer is closely connected to the output of the model. The Densenet architecture is slightly modified and used as the base classification network for this project.

Method

Data

The dataset used in this project was Tiny Imagenet. [4] The dataset contains 100,000 training images, 10,000 validation images, and 10,000 test images. As the test images are not labeled we will use the validation dataset as the test and sample the training data for validation in this experiment. The data contains no overlap between classes. Each image is 64x64 pixels and contains three channels for RGB with values ranging from 0-255. Some example classes are listed below:

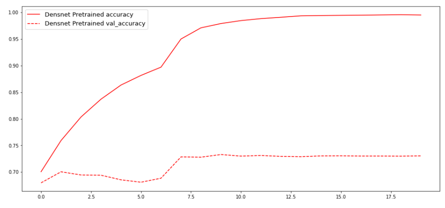
n00001930	physical entity
n00002137	abstraction, abstract entity
n00002452	thing
n00002684	object, physical object
n00003553	whole, unit
n00003993	congener
n00004258	living thing, animate thing
n00004475	organism, being
n00005787	benthos
n00005930	dwarf
n00006024	heterotroph
n00006150	parent
n00006269	life
n00006400	biont
n00006484	cell

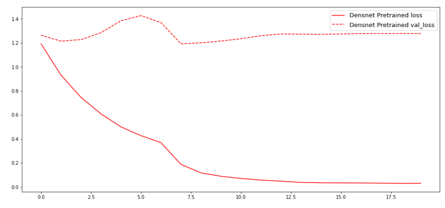
Transfer Learning

A DenseNet model pretrained on the larger imagenet was modified by removing its head and adding a fully connected 200 neuron layer to serve as the classifier for the smaller dataset. A dropout of 0.5 was used and a scaled 224x224 version of the images was used as the input. No data augmentation was performed as the training set would need to be reused when training the NAP monitor.

The model was trained for 20 epochs below are the results on the test data:

Accuracy: **0.7368**
Loss: **1.2700**





NAP Monitor architecture

After model training the construction of the NAP monitor could begin. Due to computing restraints of Google Colab it was only possible to construct monitors and run experiments for the first 50 classes of the dataset out of 200 total classes. With more computing power this concept could easily be expanded to encompass all 200 classes.

In constructing this Monitor the last feature layer with shape 1024x1 was chosen. This layer contains high level features of the image and less spatial information so it is ideal for a NAP monitor.

Each monitor had the following components associated with it:

<u>selected_indices:</u>
This array contained the indices to select for monitoring from the larger 1024 array of activations. It is unique to each class.

<u>bdd_patterns:</u>
This is a bdd that stores all the different activation patterns associated with the class.

Initialization

During initialization each class monitor is created and it's bdd_pattern set to empty. It is here that the indices are selected based on the hyperparameters. The weights between the 1024 layer and selected class are passed in during initialization. The weights are then sorted. The absolute value of the weight is an indicator for how much impact the activation has on deciding if the image is in the selected class. The number of indices selected is based on the hyperparameter *# of neurons*. The *split* hyper parameter decides how many of the neurons with the largest negative weight to select and how many of the neurons with the largest positive weight to select. Once initialized the indices remain constant throughout the training process.

Training

We only train the NAP monitor on images that are correctly classified. During training the 1024 activations are calculated using a partial model that ends at the specified layer. The activations have already passed through a ReLU function so they are either positive or zero. These activations are then converted into a binary array. The *threshold* hyperparameter decides how large the activation needs to be to be considered a “1” in the binary array. The array is then passed into the class monitor which selects the specified indices and encodes it into the bdd.

After training if the *hamming distance* hyper parameter is specified. Then the monitor's patterns are looped through and extended by adding a new case where a selected variable can be true or false and still match a pattern. This is done efficiently using the architecture of the bdd.

Testing

During runtime the 1024 activations are calculated using a partial model that ends at the specified layer. The activation is converted to a binary array and then is checked against the bdd of its predicted class. If a pattern match is found then the NAP returns true otherwise false.

Performance Evaluation

In order to evaluate the performance of the model we keep track of four data points during the evaluation process.

total: # of samples evaluated
wrong: # of samples incorrectly classified by the model
out of activation: # of samples found to be out of the activation patterns of the predicted class
out of activation and wrong: # of samples found out of activation pattern and classified incorrectly by the model

Performance of a NAP can be measured with three metrics: trigger rate, sensitivity, and pattern probability.

$$100 \; * \frac{out\ of\ activation}{total} \; = \; trigger\ rate$$

Trigger rate is the rate the NAP finds an image out of pattern from its class. A high trigger rate means that most images are found to be out of activation pattern for their respective class. While a low trigger rate indicates that the monitor does not find many images out of pattern. Keeping the trigger rate low is beneficial as we dont want the NAP questioning every decision the model makes.

$$100 * \frac{out\ of\ activation\ and\ wrong}{wrong} = sensitivity$$

Sensitivity is how well the monitor detects an out of pattern when there is a misclassification. The higher the number the better as it indicates that the monitor is detecting a large amount of the misclassified data.

$$100 * \frac{out\ of\ activation\ and\ wrong}{out\ of\ activation} = pattern\ probability$$

Pattern probability is the probability that an out of pattern image indicates that the image is misclassified. A high probability means that the NAP monitor will be a good indicator to whether or not one should trust the classification decision made by model.

Experiment

We first varied the hyperparameters to observe their effects on the NAP monitor metrics discussed above.

Hypotheses:

Increasing neuron size will increase the trigger rate and sensitivity while lowering the pattern probability as more neurons would lead to a larger chance of being out of pattern.

Neuron split will increase the trigger rate and sensitivity while lowering the pattern probability as negative neurons will vary more for the datasets within each class.

Hamming distance will decrease the trigger rate and sensitivity while increasing the pattern probability as the monitor will find way less images out of pattern by two activations and if they are, they will very likely be misclassified.

Thresholding should not have a significant impact on the metrics as it will simply lead to less activated neurons the higher the threshold. Could be an important hyper parameter to combine with others when searching the space.

Results

The model evaluated 2458 samples as within the 50 classes specified out of the 200. Of those samples it classified 554 samples incorrectly giving it a misclassification rate of: **22.5%.**

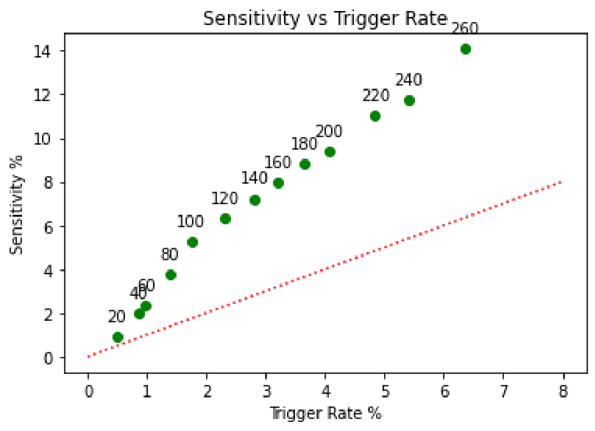
The red line in the first graph is just the line y=x. If a monitor falls below this line it is severely underperforming.

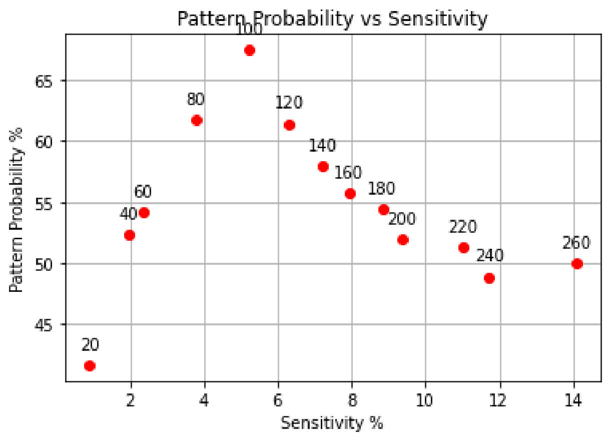
Varying neuron size:

The number of indices selected for the monitor is based on the hyperparameter *# of neurons*.

The neuron size was varied from 20-260 in increments of 20.

# of neurons	**trigger rate**	**pattern probability**	**sensitivity**
20	0.488	41.667	0.903
60	0.976	54.167	2.347
80	1.383	61.765	3.791
100	1.749	67.442	5.235
120	2.319	61.404	6.318
180	3.662	54.444	8.845
260	6.347	50.000	14.079





Contrary to the hypothesis pattern probability does not always decrease when the neuron size is increased. According to the data having a very low neuron amount actually harms the pattern probability. The strongest pattern probability was found at neuron length 100. As expected the sensitivity increases and trigger rate increases as more neurons are added to the monitor.

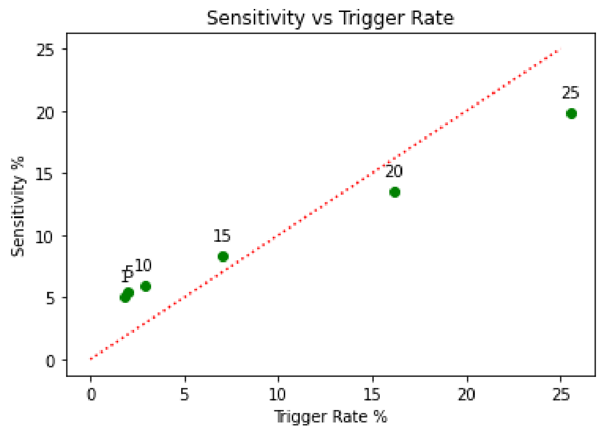
Negative Split

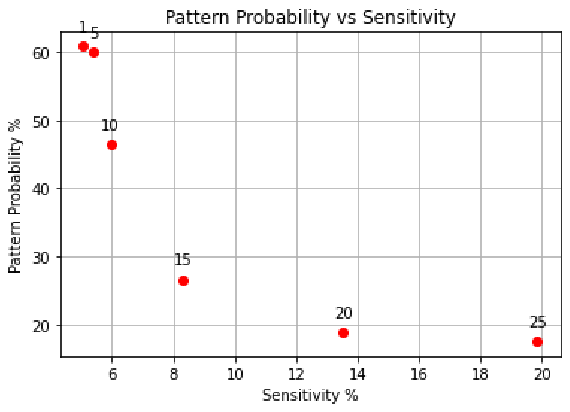
The *split* hyper parameter decides how many of the neurons with the largest negative weight to select and how many of the neurons with the largest positive weight to select.

The number of neurons was held constant at 100.

The number of negative neurons selected was reduced from the positive selection. So selecting 5 negative weight neurons implies selecting 95 positive weight neurons.

# of neurons negative	trigger rate	pattern probability	sensitivity
1	1.871	60.870	5.054
5	2.034	60.000	5.415
10	2.889	46.479	5.957
15	7.038	26.590	8.303
20	16.151	18.892	13.538
25	25.509	17.544	19.856





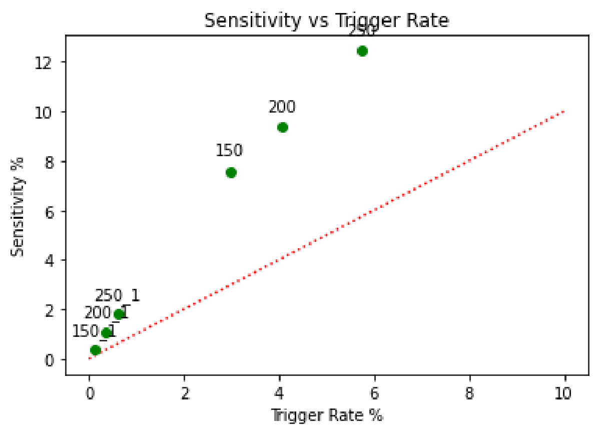
Using negative neurons in the monitor harms its performance significantly. When more than 20 neurons are added from the negative weights the trigger rate exceeds the sensitivity indicating that the monitor is firing at a higher rate then it is finding misclassified images. A low amount of negative neurons could be useful for boosting sensitivity when combined with other hyper parameters.

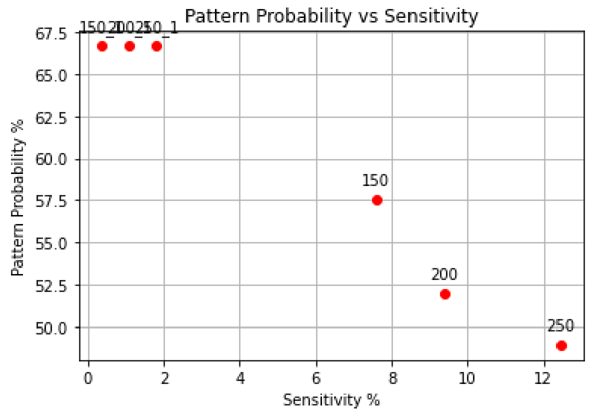
Hamming Distance

Increasing the hamming distances means that the pattern can be off by one activation and still be accepted by the monitor

The effect of implementing a hamming distance was tested at three different neuron lengths.

Monitor	**trigger rate**	**pattern probability**	**sensitivity**
150	2.970	57.534	7.581
150 + hamming	0.122	66.667	0.361
200	4.068	52.000	9.386
200 + hamming	0.366	66.667	1.083
250	5.736	48.936	12.455
250 + hamming	0.610	66.667	1.805





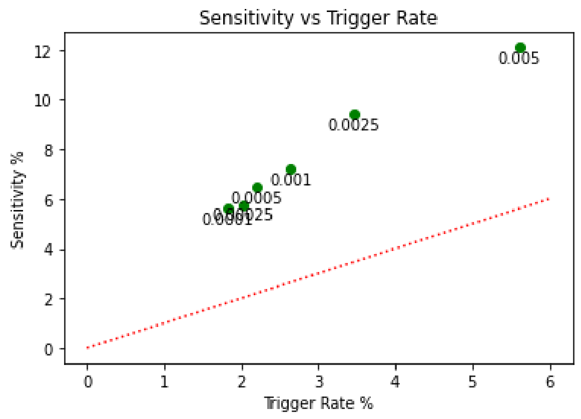
Hamming distance had a significant impact on the metrics. Increasing the hamming distance caused the trigger rate and sensitivity to drop by a huge margin from their normal values. Due to the nature of this dataset increasing the hamming distance seems unnecessary as the sensitivity is already so low in the monitor.

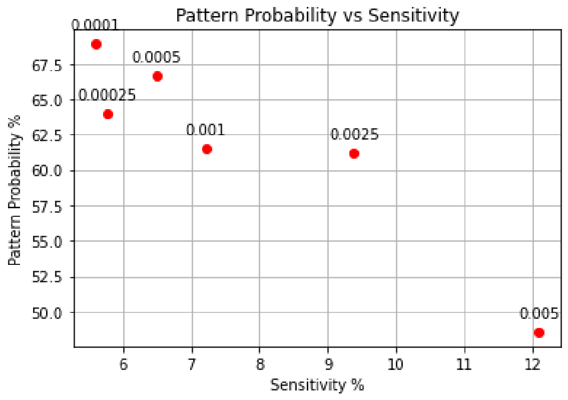
Thresholding
The *threshold* hyperparameter decides how large the activation needs to be to be considered a "1" in the binary array.

The number of neurons was held constant at 100.

The effect of changing the threshold values on monitor performance.

Threshold	trigger rate	pattern probability	sensitivity
0	1.749	67.442	5.235
0.0001	1.831	68.889	5.596
0.00025	2.034	64.000	5.776
0.0005	2.197	66.667	6.498
0.001	2.644	61.538	7.220
0.0025	3.458	61.176	9.386
0.005	5.614	48.551	12.094





Thresholding had the most interesting and significant impact on monitor performance. By varying the threshold we were able to achieve an increase in sensitivity without a reduction in pattern probability, something that was not seen in any of the other hyper parameters. Using a 0.0001 threshold resulted in a monitor that outperformed the zero threshold in both pattern probability and sensitivity. It is unclear what is behind this performance boost.

Maybe the increased threshold results in a tighter more accurate activation pattern set. More research into this is necessary.

Hyperparameter Search

By varying the neuron amount and threshold we can search through the hyperparameter space and find high performing monitors.

Some of the results of the search:

Monitor	**trigger rate**	**pattern probability**	**sensitivity**
160/0.0001	3.37	55.421	8.30
200/0.005	17.615	35.796	27.978
200/0.0005	5.044	48.387	10.830
240/0.005	6.672	48.78	14.440

Conclusion

Neuron Activation Pattern monitors are a powerful tool for creating more robust AI systems. The ability of the monitor to indicate whether or not a decision is being made based on trained data has powerful implications for designing fault tolerant systems such as self driving cars. Further hyperparameter tuning could be done to improve the performance of the NAP monitor. Another avenue of research could be looking at replacing the bdd with a system that could hold a larger range of activation values rather than 0 or 1. This could potentially improve performance of the monitor.

References

[1] Chih-Hong Cheng, Georg Nuhrenberg, and Hirotoshi Yasuoka. *Runtime Monitoring Neuron Activation Patterns*. CoRR https://arxiv.org/abs/1809.06573

[2] K. S. Brace, R. L. Rudell and R. E. Bryant, *Efficient implementation of a BDD package*, 27th ACM/IEEE Design Automation Conference https://ieeexplore.ieee.org/abstract/document/114826/

[3] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. *Densely Connected Convolutional Networks.* CoRR https://arxiv.org/abs/1608.06993

[4] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. *Tiny ImageNet Challenge*. http://cs231n.stanford.edu/reports/2017/pdfs/930.pdf