Monitoring Neuron Activations for Out of Distribution Detection

Sasi Kiran Gaddipati
Jaswanth Bandlamudi

Hochschule Bonn-Rhein-Sieg

January 23, 2020

Roles

- **Stakeholder:** Prof. Dr. Nico Hochgeschwender

Team Member	Roles
Jaswanth Bandlamudi	Developer Product owner
Sasi Kiran Gaddipati	Integration Engineer System Engineer

- **Developer:** Creates software and meets frequently with integration team for improving the code.
- **Product owner:** Takes all the inputs from the stakeholders and set the priorities for the work to increase the value of the deliverable.
- **Integration Engineer:** Designs and integrates the software patterns and flow.
- **System Engineer:** Identifies and resolves issues in the software.

Motivation

- Deep learning in the safety binding applications cannot explicitly derive confidence in the predictions.
- This is not only a drawback but also dangerous in real-world applications.

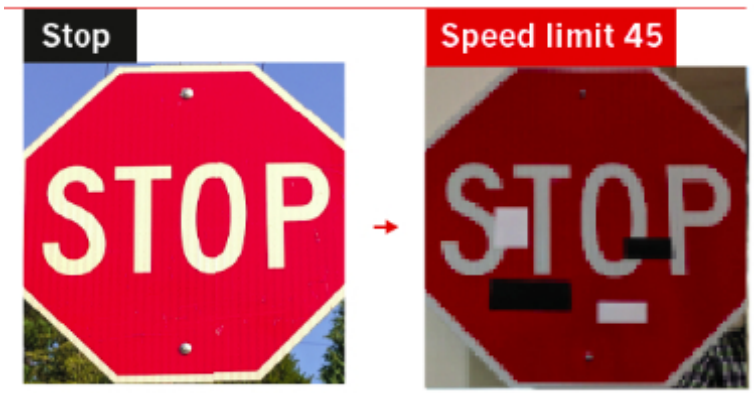

Figure: Misclassification of a traffic signal[1]

- Therefore, our objective is to "incorporate a trained deep learning module into the product and embed a monitoring system that can detect input that is not in the training distribution."

Approach

- To achieve this goal, we adhere to implement a work similar to Cheng et al.[2]
- This paper suggests to compare the output activation patterns with the saved activation patterns of a class as shown in the figure.

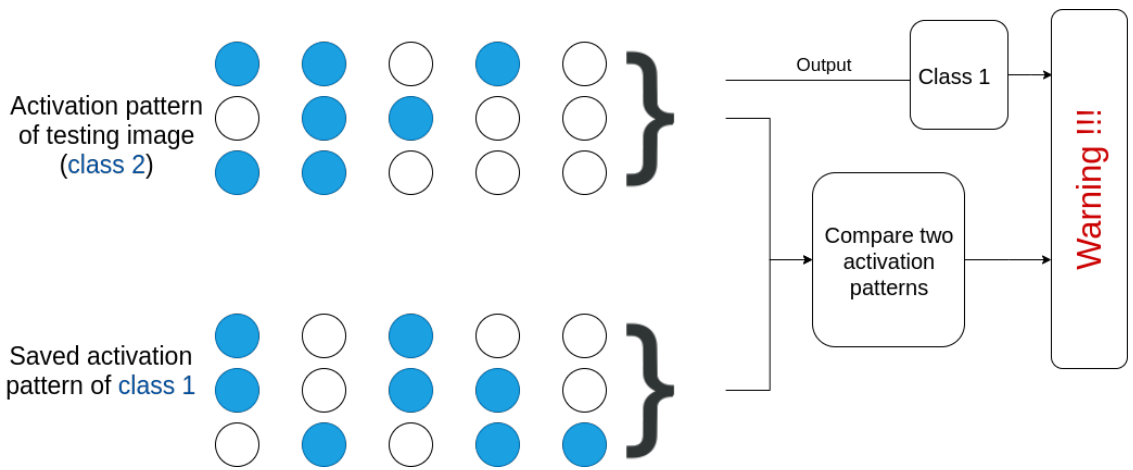

Figure: Pipeline of monitoring activations

Contrasts with Authors' Approach

- Saving activations as one-hot encoders lose much information about the generated features.
- Instead we save the mean and variance of each activation, which reduces the usage of memory and further computational cost.
- As we alter the process of saving the activation patterns, we also have to change the comparison metric from Hamming distance to KL-divergence.
- This also assists in neglecting the Binary Decision Diagrams(BDD).
- In addition, we also endeavour to visualize the activation patterns for better understanding.

Authors' approach	**Our approach**
Saving activations as one-hot encoders	Saving activations' mean and variance
Use of Hamming distance	Use of KL-Divergence
Use of Binary Decision Diagrams(BDD)	No requirement of Binary Decision Diagrams

Table: An overview of contrasts between the authors' approach and our approach

Development

- We divide the total project into two modules as follows, for the ease of implementation and division of work.
  1. Visualization of activation patterns
  2. Saving and monitoring activation patterns
- We follow the **Agile software development** for the requirement of the output at earlier stages.
- This assists in improving both the output and the frame work, according to the reviews.
- Hence, we initially created and implemented the code in *'Jupyter notebook'* for the necessity of proof of concept.
- The working code is then integrated as an object oriented programming paradigm with S.O.L.I.D principles.

Visualizing Activation Patterns

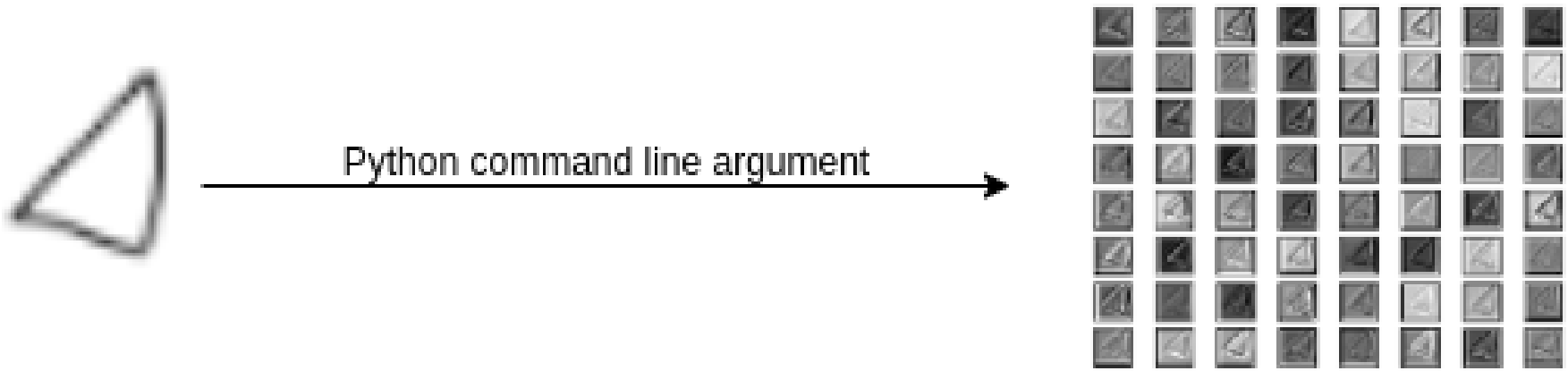

Figure: Visualizing activation patterns(I/O)

Visualizing Activation Patterns

Project Design Parameters

- Language : Python 3.7
- Coding convention : PEP08
- Data structure:
  - **Input** : Image and layer number
    - Input the image to get the activation layers
  - **Output** : Image
    - Plot the images of the activation layers
- Design pattern: Bridge pattern

Installation and Execution

- git clone https://github.com/gsasikiran/Monitoring-Neuron-Activations
- python visualize_activation.py image_path(with quotes) layer_number

Saving and Monitoring Activation Patterns

- As suggested in Cheng et al.[2], the activation maps of the penultimate layer are used for monitoring the out of distribution patterns.
- The saving of activation patterns is done in two different forms
  1. Saving the activation maps.
  2. Saving the mean and variance of the activation maps.
- The activation maps were compared using Hamming distance.
- The Gaussian distributions were compared using Kullback–Leibler divergence.
- The MNIST dataset activations were compared with activations of **MNIST triangle** which is classified as **"Eight"** by the model built.
- **Observation:** The comparison of either activation maps or the probability distributions did not give expected results.

Bayesian Neural Networks

- While exploring the probability distribution comparison, we came across a practical way for regressing confidence of the neural network in providing the output from Y. Gal[3].
- This method treats the deep learning model as a Bayesian model by modelling the weights as a Gaussian distribution.
- By performing multiple forward passes the model created the prior distribution of $P(w|x, y)$.
- While regressing the output, a posterior of $P(Y|w, x)$ is calculated and is proved to be effective in judging the neural networks uncertainty.

Observations and Conclusion

- The procedure, we followed, is not able to detect out of distribution or misclassification with confidence.
- Because of this inability to develop a proof of concept, the approach is not converted into a software package.
- However, implementing a decoy class for classification purposes helps in out-of-distribution detection.
- Bayesian neural networks are reliable in regressing the confidence score but are not real-time.
- New stochastic methods are being carried out in the research of Bayesian neural networks, to make it real-time.

References

[1] Eykholt, Kevin, et al. "Robust physical-world attacks on deep learning visual classification." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.
[2] Cheng, Chih-Hong, Georg Nührenberg, and Hirotoshi Yasuoka. "Runtime monitoring neuron activation patterns." 2019 Design, Automation and Test in Europe Conference and Exhibition (DATE). IEEE, 2019.
[3] Y. Gal. Uncertainty in Deep Learning. PhD thesis, University of Cambridge, 2016.

The End