Verifying robustness and explainability of neural networks with **Saimple**

École Jeunes Chercheuses et Jeunes Chercheurs en Programmation - Formal Methods for Machine Learning Pipelines

19/06/2024

1 Description of the practical (∼15 min)

The goal of this practical work is to highlight the importance of verifying a machine learning model's robustness and explainability, and learn how to handle **Saimple**, a software developped by **Numalis** that allows to do so.

The *robustness* of a machine learning model refers to its capacity to sustain optimal performance under various circumstances. In other words, a model is robust if it returns the same output when small chosen perturbations are applied to the inputs. The verification of robustness for neural networks has been the object of many recent studies, and for a detailed state of the art, the review of Caterina Urban and Antoine Miné can be consulted [5].

Another widely studied property of neural networks is its *explainability*. A model is explainable if it provides an interface summarizing the reasons behind its behavior and is able to convince of their well-foundedness. As doubts raised about the opaque nature of neural networks, explainability became a topic of importance. For a detailed state of the art, extensive reviews on the subject can be consulted [1, 3].

Both robustess and explainability are now required in order to put a model on the market for critical applications (in health, defense, autonomous driving...) in the EU, according to the AI act.

In this practical, you will first verify the robustness and explainability of a neural network that classifies pictures of two types of traffic signs: 30 speed limitation signs (class 0) and 50 speed limitation signs (class 1). Then, you will create your own neural network that classifies pictures of three types of ultrasound images of breast tissues (healthy, benign tumors, and malignant tumors) and evaluate your model.

2 Analysis of the given model

In this section, you will evaluate the neural network given in model1.onnx, that was created using the model.py file and the parameters.py file. The model is an image classifier that distinguishes between 30 and 50 speed limitation signs, built with the LeNet [4] architecture.

2.1 Classical evaluation of the model (∼15 min)

Question 1. Analyze the anatomy of the neural network given in model1.onnx, that was created using the model.py file and the parameters.py file. What are the different layers ? What are the activation functions ?

There are many ways of doing this, but we wouldS like to suggest two:

- using the **Netron** app (https://netron.app)
- look directly in the script **model.py**, inside the `leNet_model` function

Question 2. The file **classical_evaluation.py** contains a python script that allows to plot the confusion matrix of the model. Explain what a confusion matrix is.

Question 3. Run the **classical_evaluation.py** script and analyze the plotted confusion matrix. Does the model seem accurate enough ?

2.2 Explainability analysis (∼ 1h)

In this section, we will analyze the explainability of the given model by observing the most influential pixels in several inputs. To do so, we will use the Saimple app, that can help provide explainability to opaque machine learning models.

Saimple offers a visualization of the *relevance* value which indicates, for each pixel of an input image, how much a variation on this pixel impacts the output. In the case of a classifier, Saimple provides a relevance visualization for each of the classes, indicating the impact of each pixel on the score for this class. Note that when there are only two classes, the relevance values for the second class are the opposite of the ones of the first class. Saimple's interface offers different visuals where the pixels with highest (in absolute value) relevance are highlighted with colors.

To answer Question 4, you will use the graphical interface of **Saimple** (https://app.saimple.com), which facilitates results visualization and interpretation for a single evaluation. You will then use the API, which is more practical in order to launch several evaluations and integrate the results into a script. In both cases, you will need to enter your login and password, which you should have previously received by email.

Question 4. Analyze model1 for the input 30.png, which was drawn from the test data set, with the graphical interface of Saimple, with an additive noise of intensity 0.001. Find out the class with highest

score with the dominance plot: Is this input well classified ? Observe the relevance while making the visualization parameters vary. Does the relevance seem consistent ? Explain.

To analyze a given input and model on the graphical interface of Saimple, you can follow these instructions:

1. Connect onto the **Saimple** app (https://app.saimple.com/) with the login and password previously received by email
2. Click on the "+ New Evaluation" button (at the top left corner)
3. Fill in the form on the New Evaluation page:
   (a) Enter any evaluation name you like - it is useful for finding your evaluation later if you have several
   (b) Choose your model to analyse, to select a model you have to click on the "Upload Model" button (Saimple currently works only with onnx and h5 models)
   (c) Choose the input, to select an input you have to click on the "Upload Input" button (Saimple could work with other input types but here we are only interested in images)
   (d) Let the Normalize option, used to define if the input is normalized or not, to the default value ("Yes")
   (e) Let the Precision option, used to define the precision level required by your evaluation, to the default value ("Float")
   (f) Let the Noise Type option, used to define which type of noise you want to test your model against, to the default value ("Additive")
   (g) Choose the Noise Intensity option. It represents the intensity of the noise you want to test your model against
4. Click on the "Launch Evaluation" button. You should be redirected to the "Evaluation List" page
5. Wait until your result is no longer pending, and check the status (Green $\rightarrow$ everything is OK, Red $\rightarrow$ try again, or ask your teacher to see what is going on)

To visualize the results of an evaluation on the graphical interface of Saimple, you can follow these instructions:

1. In the ”Evaluation List” page, click on your evaluation.
2. To observe the classification and the robustness results, click on ”Dominance”
3. To observe the relevance results, click on ”Relevance images”
4. Once on the relevance results page, click on ”Settings” in order to make the visualization parameters vary:
   (a) Choose the class you want to plot the relevance for in the ”Class selection” section on the left.
   (b) Choose the mode in the ”Mode” section on the right (”Threshold” or ”Alpha”). In the ”Threshold” mode, the ”Threshold” value allows to color only the pixels whose relevance is higher (in absolute value) than the selected value. In the ”Alpha” mode, all the pixels are colored, but the ”Alpha” value determines the transparency of the relevance image relatively to the original image.
   (c) Select a ”Zoom” value in the ”Controls” section. A high zoom value allows to enlarge the relevance plot.
   (d) Select a ”Threshold” or an ”Alpha” value. Recall that the ”Threshold” value enables to color only the pixels with higher relevance, and the ”Alpha” value represents the relative transparency of the original image and the relevance image.
   (e) Choose a normalization type in the ”Normalization” section (”Global” or ”Local”)
   (f) Choose a color interpolation in the ”Color Interpolation” section (”CityLights” or ”Rainbow”). This determines the colors used to represent the different relevance values.

Question 5. Read the API python script, which allows to use Saimple's API in order to analyze a given model for a given input. Using this script, analyze the input 50.png, which was found on the internet, with an additive noise of intensity 0.001. Note that the visualizations are not integrated in the API, and were added in the utils python file, so that you can adapt it to your needs. Is the input well classified ? Does the relevance seem pertinent ? Explain.

To launch an evaluation, first modify the following rows in API.py by entering your login and password:

```
login = "Enter your login" #? Login
pwd = "Enter your password" #? Password
```

Then, modify the path to the input:

```
image_path_sign = '50.png' #? Input
```

And the path to the model:

```
model_path_sign = 'model1.onnx' #? Model
```

You can also change the name of the evaluation:

```
"name": "Question 5" #? Name of the evaluation
```

And its description:

```
"description": "Evaluation description", #? Description of the evaluation
```

Finally, modify the noise parameters:

```
"noise": {
            "intensity": "0.001", #? Noise intensity
            "mode": "ADDITIVE" #? Type of noise
        }
```

And run the python script. The dominance will be printed and the dominance and relevance plots, encoded in the utils python file in the scripts folder, will appear.

Question 6. You will be assigned an input in the inputs folder, which contains 30 images found on the internet. Analyze it with the API. Is it well classified ? Does the relevance seem pertinent ? We will gather the group's results and calculate the accuracy. Does it seem consistent with the accuracy given by the confusion matrix ? Formulate an hypothesis that could explain why.

Question 7. The dataset used to train the model is in the data1 folder. Observe the pictures and, if needed, analyze their relevance. Reformulate your hypothesis.

To observe the pictures of the data1 folder, stay in "grid" mode inside the folder, and zoom in to better see the pictures and try to notice something. If you struggle understanding what is expected for you to notice, use Saimple (with the graphical interface or the API) to analyze the relevance of several pictures in the folder. If you still do not notice anything, ask your teacher to help you.

Question 8. Using the model and parameters python scripts, retrain the model with the pictures in the data2 folder. Analyze the inputs 30.png and 50.png again with the API. Are the inputs well classified ? Does the relevance seem consistent ? Explain.

To retrain the model, modify the following line in the parameters.py python script:

```
DATA_FOLDER='data1'
```

Into:

```
DATA_FOLDER='data2'
```

Be careful not to change model1 into model2, or you will rewrite the model2 file provided in the zip for the next section. Run the model.py python script. A new model1.onnx file will be created. Run the API.py python script again to analyze this new model, with the two inputs.

2.3 Robustness analysis (∼1h30)

Saimple is also designed to certify the local robustness of machine learning models. A model M is *locally robust* if given an input I, and a type a noise which leads to a set S of noisy inputs, the output obtained for any input in S is identical to the output obtained for I. Saimple allows to test the robustness of a model for a chosen input and to different kinds of noises: additive (each pixel is added a value from $-c$ to c where c is a positive constant), multiplicative (each pixel is multiplied by a value from 1 to c where c is a positive constant), Bokeh blur or Gaussian blur, with chosen amplitude.

From now on, in order to all work on the same model, we will use the model2.onnx file, which correspond to a corrected unbiased version of model1.onnx.

Question 9. Write a python script that generates a noisy input by adding a random additive noise between -0.001 and 0.001 on every pixel of 50.png. Use the predict.py python file in order to predict the class of this noisy input. Does the output remain identical with the added noise ?

> You will need the numpy, Image from PIL and random librairies. To open an image, you can use the Image.open() function. To transform an image into a numpy array, you can use the numpy.asarray() function. Note that the resulting values for each pixels will be from 0 to 255, instead of from 0 to 1. To transform an array into an image, you can use the Image.fromarray() function. Note that the input values for each pixels should be from 0 to 255, and not from 0 to 1. To save an image, you can use the save() function. Since this noise is not visible to the naked eye, you can replace 0.001 by 0.1 just to make sure your script works.

Question 10. We gather the groups results. If the output remains identical for every added random noise, does that constitute a proof of robustness ? The images are of size 32 x 32 pixels, and we consider that the precision of random floats with random.uniform is 10^{-16}. How many possible noisy inputs are there ?

Due to the combinatorics, Saimple does not test any possible noisy input to verify that the output does not change. As critical applications require formal proofs of robustness, it does not use statistics either. Instead, it uses *abstract interpretation*. The abstract interpretation theory, introduced in 1977 by Cousot and Cousot [2], aims to use sound approximations of programs to prove that they verify some given properties. In the case of neural networks, the principle is to create an abstract object that contains all the noisy inputs, and observe how this object behaves when passing through the layers of the neural network. Over-approximations are made through the computations, which implies that the final abstract shape contains all the desired outputs, but can also contain space that does not correspond to the desired outputs. Thus, Saimple can be used to prove robustness, but it cannot prove a lack of robustness. We say that it is *sound* but not *complete*. Figure 1 illustrates the different steps of the robustness test performed by Saimple.

Once the final abstract shape is computed, Saimple offers in its interface a visualization in which each class is associated a score interval, that contains the scores of all the noisy inputs. Since the output class is the one with maximum score, if the interval with highest scores does not intersect any other interval, then the

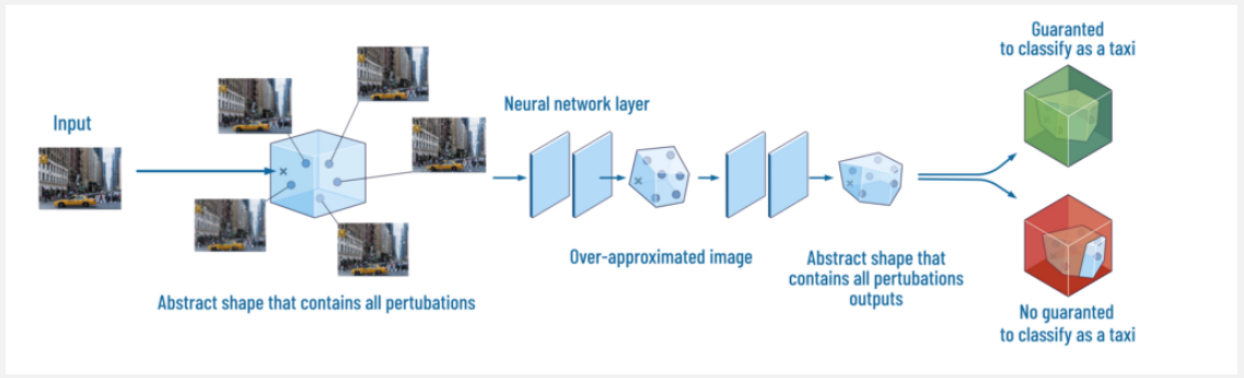

Figure 1: Saimple uses abstract interpretation to verify neural networks.

robustness is proven. We say that this class is *dominant*. However, if the intervals intersect, it does not necessarily imply that the robustness is not proven (since the scores for each class are related). For this reason, the dominance is also directly displayed.

Question 11. With the API, analyze the input 50.png with an additive noise of 0.001. Is the result robust to the chosen noise ?

> Although the dominance plot might provide some informations, you should be looking at the dominance directly printed in the terminal.

Question 12. Change the amplitude of the noise to 0.01. Is the result robust to the chosen noise ?

Question 13. The maximum value of the noise for which the result is proven robust is called the *delta max*. Create a new python script to find a tight lower and upper bound on the delta max. To do so, use dichotomy. The difference between the final upper and lower bounds should be less than 10^-5. You may observe that using dichotomy, the search for delta max is a long process even with small precision. For this reason, an optimized search functionality is available in Saimple Premium.

> Start with a lower bound at 0.001 and an upper bound at 0.01, and perform an evaluation using the API script, with a precision value in the middle. If the robustness is proven, the precision is the new lower bound, otherwise it is the new upper bound. Continue this process in a while loop until the difference between the upper and lower bound is less than 10^-5.

3 Analysis of your own model (optional)

Question 14. Create your own neural network that allows to distinguish between ultrasound images of three types: healthy, benign tumors, and malignant tumors. To that end, you will use the dataset contained in the data3 folder. When you are satisfied with your network, test its robustness to different kinds of noise, and analyze its relevance. Do you think your model is robust and explainable enough to be put on the market ?

You can modify the model.py python file to generate the neural network. Feel free to add or change layers, activation functions, or anything you think might be helpful. Do not forget to change the number of classes.

For more information about the Saimple software, you can visit the **Saimple** website (https://saimple.com). For more information about Numalis, you can visit the **Numalis** website (https://numalis.com).

References

[1] P. P. Angelov, E. A. Soares, R. Jiang, N. I. Arnold, and P. M. Atkinson. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(5):e1424, 2021.

[2] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, 1977.

[3] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] C. Urban and A. Miné. A review of formal methods applied to machine learning. *arXiv preprint arXiv:2104.02466*, 2021.