Scrapy/Scrapinghub: Grad-CAM Neural Network Explanations for ELI5

A proposal to add Grad-CAM based neural network explanations to the ELI5 library.

About Me

1. Name: Tomas Baltrunas, @teabolt on GitHub.

Project Information

Sub-org info

- Parent organisation: Python Software Foundation
- **Sub-org: Scrapy / Scrapinghub**
- Sub-project / repo: ELI5

Project Abstract

ELI5 is a Python library for explaining machine learning (ML) models. Currently ELI5 supports scikit-learn, xgboost, and other ML libraries, taking in models such as linear classifiers and decision trees. However, explanations for neural networks are yet to be added. Thus, in this project I propose to add support for three popular neural network libraries: PyTorch, Keras, and Tensorflow. I will explain the models from these libraries using my implementation of Grad-CAM, a widely applicable and strong approach for highlighting what contributed to a prediction. In the case of image-based networks, a heat map will be created indicating where the network looked to make a prediction, of which there are many examples online. Less commonly done, in this project I will also explain text-based models via text highlighting. Finally, I will create the associated tests, documentation, and build configuration as a result of these deliverables.

Detailed description (deliverables and implementation)

New source code in **eli5/eli5** (the majority of the work is done here)

Integrate with the existing ELI5 API to explain neural network models.

1. Create a new top level Python module / package associated with each new library.
2. Allow the ELI5 API to take in neural networks from Keras, PyTorch and Tensorflow as arguments.
    a. The ELI5 API exposes two high level methods to explain a model - explain_weights (global, parameter-based explanation) and explain_prediction (local, particular prediction-based explanation)
    b. Register implementations of **explain.explain_prediction** generic function (see functools.singledispatch) in added library modules/packages, using the type of the supported models.
        i. Pytorch: **torch.nn.model**
        ii. Keras: **keras.Model**
        iii. Tensorflow:
            1. **tensorflow.keras.Model**
            2. **tensorflow.estimators.Estimator**
            3. **tensorflow.keras.layers.Layer**
    c. To keep consistent with the current API, register an “estimator not supported” implementation for the **explain.explain_weights** function.
        i. The main use of Grad-CAM is in explaining the predictions of a model (not the model’s weights).
        ii. (Optional) If there is time, use another visual explanation technique to explain the weights of a network, or apply Grad-CAM for the task.

Implement Grad-CAM to produce explanations of neural network predictions.

1. The Grad-CAM algorithm to be implemented, informally:
    a. Differentiate the network’s output with respect to a hidden layer’s output (for image-based convolutional neural networks (CNN’s) , this is the last convolutional layer. In this and other cases, the hidden layer must consist of differentiable nodes).
    b. Perform global average pooling (GAP) on the gradients. This will represent the weight for a particular node in the hidden layer.
    c. Multiply the weights by the corresponding nodes in the hidden layer, normalising the result through some activation function such as ReLU to only get positive outputs.
2. The result of Grad-CAM should be a “localisation map” (a tensor).
    a. The result should indicate how important were the corresponding hidden layer nodes (activation map nodes in the case of CNN’s) in making the prediction.

        i. Video demo of various results from the authors of Grad-CAM.
    b. The result should be coarse (approximate in quality), but robust (noise resistant) and interpretable (understandable).
3. Use the API's of associated libraries to implement the operations required by Grad-CAM.
    a. SEE FIGURE 1 for a table of ways to implement various Grad-CAM stages.
    b. The implementation can be based on many implementations already available online.
4. Handle differences between the "kind" of model when implementing Grad-CAM.
    a. Support image-based, text-based, and "other" model categories.
    b. Handle differences in picking a hidden layer for gradient calculation.
5. Ensure that the explanation implementation is extensible.
    a. (Optional) If there is time, enhance the coarse explanations of Grad-CAM by using another technique on top of or alongside Grad-CAM.
6. Store explanations in objects consistent with the **base.Explanation** and **base.TargetExplanation** classes that provide a format/API for ELI5 explanations.

Add an appropriate output format for image-based model explanations.

1. Keep consistent with the ELI5 architecture by separating the "explanation" from its "output format" (how the explanation is presented).
2. Overlay a heat map over the original image.
    a. Use the tensor produced by Grad-CAM as the basis for the heat map.
    b. SEE FIGURE 2 for a visual example.
3. Implement image manipulation using new dependencies
    a. Pillow
        i. **Image.alpha_composite** can combine two images while keeping transparency.
    b. Matplotlib
        i. Compatible with Jupyter Notebook.
        ii. **pyplot.subplots**, **pyplot.imshow** are capable of generating heat maps.

Integrate with existing output formatters for text-based model explanations.

1. Display HTML with word / character highlighting.
    a. SEE FIGURE 3 for a visual example.
    b. Formatters.html.format_as_html
    c. Example implementation that shows both text explanation and an output format can be seen here.

Integrate with existing formatters for "other type" model explanations.

1. Convert raw localisation map array to text, dictionary, dataframe formats
    a. formatters.text
    b. formatters.as_dataframe

        i. Might need to provide a new implementation for the **format_as_dataframe** generic function.
    c. **Formatters.as_dict**

Update the 'show' convenience functions to take in neural networks.

1. ELI5 API exposes the **show_weights()** and **show_prediction()** convenience functions that perform explanation and output formatting in one call, which is useful for interactive (ipython) sessions.
2. Modify the show functions in the **ipython** module.
    a. Add the image formatter.
    b. Associate default output formats for neural networks of a certain kind:
        i. Image-based -> heat map image.
        ii. Text-based -> highlighted HTML.
        iii. Other -> text / dict / dataframe (raw data).

New tests in **eli5/tests**

1. Use the **PyTest** framework for unit test cases.
    a. Models should be created, trained with sample data (test fixtures may be used here), and tested against expected output.
2. Uphold high test coverage as indicated by the **Coverage.py** tool.
3. Include **mypy** type annotations in source code (as comments to support Python 2.7), pass static checks.
4. Ensure that pull requests to be merged into master pass the Travis CI build.
5. Update test scripts in **_ci**, eg: **runtests_nodeps.sh** (run tests that don't have external library dependencies - exclude added dependencies).

Add new documentation for Sphinx/ReadTheDocs at **eli5/docs**

1. Add tutorial(s) on neural network explanations under **source/tutorials**.
2. Add detailed ELI5 API explanations for added libraries in **source/libraries**.
3. Update **overview.rst**, adding to the list of supported ML libraries.
4. Include appropriate Jupyter Notebook(s) in **eli5/notebooks**, as relevant to the tutorial(s).
5. Write docstrings in source code that can be processed by Sphinx autodoc and included in **source/autodocs**.

Update the ELI5 build with new dependencies

1. Update **setup.py** for setuptools and **requirements.txt** for pip.
2. Update tox configuration file **tox.ini**.
3. Update the Travis CI configuration file **.travis.yml**.

Workflow, other

1. When working, open GitHub pull requests and continuously integrate changes into master after a contributor's review.
2. Version support and backwards compatibility.
    a. Python 2.7 through Python 3.7 (current newest) should be supported.
    b. Tensorflow 2.0 should be targeted, supporting 1.0 as well.

Weekly timeline

● **Before Community Bonding** (April 9 - May 6)
● Contribute more pull requests to ELI5. Keep in touch with mentors. ● Read and understand more about neural networks, CNN's, text-based models. ● Play around with available Grad-CAM implementations, off-the-shelf models and datasets.

● **Community Bonding** (May 7-26): List any prepwork you want to do before coding starts.
● Re-iterate the project details with mentors. Discuss any workflow details such as GitHub conventions. ● Set up Python development environment: Python 2.7 - 3.7, pip requirements and test requirements, jupyter notebook, versions of libraries to be added. ● Read about PyTest idioms, mypy type annotations, Sphinx documentation syntax. ● Think about the possibility of keeping a blog for the duration of the programme.

● **Week 1** (May 27-31)
● (Implement new features for a single library first - Keras). ● Add support for Keras models as arguments to the ELI5 API. ● Add Grad-CAM prediction explanations for Keras image-based models. ● Test: Keras image prediction explanations.

- **Week 2** (June 3)

- Update project / build dependencies.
- Add image output formatter.
- Test: image output formatter.

- **Week 3** (June 10)

- Integrate image output with Keras prediction explanations.
- Add explanation for text-based Keras models using Grad-CAM.
- Test: text-based prediction explanations.

- **Week 4** (June 17)

- Integrate with HTML output format for text-based Keras explanations.
- Add prediction explanation for “other type” models in Keras using Grad-CAM.
- Test: “other type” prediction explanations for Keras.

- **Week 5** (June 24): **First Evaluation (June 24 - 28)**

- (By the first evaluation, the majority of features to be delivered by the project should be implemented for Keras).
- Refactor code, tests.

- **Week 6** (July 1)

- Integrate text/dict/dataframe output formatting with Keras “other type” explanations.
- (Mirror features to support PyTorch and Tensorflow libraries).
- Add support for PyTorch, Tensorflow models as arguments to ELI5 API.

- **Week 7** (July 8)

- Add explanations for predictions of PyTorch, Tensorflow models.
- Test: PyTorch, Tensorflow prediction explanations.

- **Week 8** (July 15)

- Integrate PyTorch, Tensorflow explanations with output formats.

- **Week 9** (July 22): **Second Evaluation (July 22 - 26)**

- (Most of the deliverables should now be implemented for PyTorch and Tensorflow).
- Refactor code, tests.

- **Week 10** (July 29)

- Update “show” functions to handle neural networks and output in the expected format.
- Test: “show” functions.

- **Week 11** (August 5): you may want to try to "code freeze" in week 11 and complete any tests/documentation in week 11-12.

- Final code and test refactoring.
- Docs: Add neural network tutorial(s).
- Docs: Add Jupyter Notebook(s) for the tutorial.

- **Week 12** (August 12)

- Docs: Add supported library reference for Keras, PyTorch, Tensorflow.
- Ship a new release if needed.

Final week (August 19) / **Final Evaluation (August 19 - 26)**: Submit project

FIGURES

FIGURE 1

Constructs that can be used to implement each Grad-CAM stage (horizontal) in corresponding libraries (vertical), taken from existing implementations.

	Layer access	Gradient calculation	GAP
Keras	model.get_layer()	K.gradients	K.mean
PyTorch	model.modules()	(register a hook)	torch.nn.functional.adaptive_avg_pool2d
Tensorflow	model.layers[layer]	tf.gradients	np.mean

FIGURE 2

Grad-CAM paper, highlights regions from which the model decided that the image class was ‘cat’.

FIGURE 3

Current ELI5 docs, show capability of ELI5 to highlight words/characters with HTML.

as i recall from my bout with kidney stones, there isn't any medication that can do anything about them except relieve the pain. either they pass, or they have to be broken up with sound, or they have to be extracted surgically. when i was in, the x-ray tech happened to mention that she'd had kidney stones and children, and the childbirth hurt less.