IML 2021 - Project report

Group E - Beiqian Liu, Jakub Zadrozny, Tony Abemonty

Introduction

The fact that we chose the scenario 4, which is helping an ML developer build unbiased and our interest on adversarial attacks leads us to explore explainable AI which is a very interesting field with numerous advantages, like helping the user build useful mental models, increase their trust in prediction and improve the training of intelligent agents, so that's why we chose to work on this topic.

Our initial purpose in this IML project was to show the fact that complicated models are not completely trustable because of their unrobustness in some cases. To emphasize this point we are doing our project through four main ideas, the first one is why do we care about explainable AI in general, the second one is how to explain current models through visualization methods, the third one is how to build explainable models comparing to black-box ML models, and finally our main focus between those points is the use of adversarial attack to fool models even the most robust and complicated ones. So what is the result shown by these robust models and does a solution exist for the adversarial attack?

Literature Review

Why do we care about XAI?

The black-box nature of machine learning systems could influence its development and application. Developers may want to know why a particular model behaves poorly at times, and end-users will question the reliability of ML systems or be confused about the prediction results. While explainable AI can help users build useful mental models and increase their understanding of the learning system by 52% [1], thus increasing their trust in not only the prediction but also the model, which is essential in safety critical or socially important applications like self-driving cars, medical diagnose, banking, etc. From the developer side, they need to be confident enough to deploy their models. What's more, XAI also provides insights into the model, it can help developers debug and improve an untrustworthy model, as well as compare and choose between models [2]. What's more, the lack of interpretability and transparency often leads to a time-consuming trial and error process [3], XAI can improve training of intelligent agents. Last but not least, under the current General Data Protection Regulation, EU citizens have rights to be explained.

How to explain current models

The most common method to explain ML models is visualization. According to Fred Hohman [4], there are six ways to visualize deep learning. 1) *Node-link Diagrams for Network Architectures*. It visualizes where data flows and the magnitude of edge weights. For example, Wang et al. [5] proposed CNN Explainer, which visualizes a CNN architecture where each neuron is encoded as a square with a heatmap representing the neuron's output, to help users more easily understand the inner workings of CNNs. 2) *Dimensionality Reduction & Scatter Plots*. Visualizing High-dimensional data in a lower-dimensional space, like Principal Component Analysis (PCA) and t-SNE. 3) *Line Charts for Temporal Metrics*. By showing a number of different metrics computed after each epoch, including the loss, accuracy, and different measure of errors, developers can track the progression of their models. This can be useful for diagnosing the long training process models. 4) *Instance-based Analysis & Exploration*. It's testing specific data instances to understand how they progress throughout a model to help interpret and debug models, which is called instance-level observation. 5) *Interactive Experimentation*. This is visual experimentation that helps users make sense of complex concepts and systems. Spinner et al. [6] designed explAIner, the system will give suggestions to refine the model, and the user can apply the refinement. After that, the system will show the difference before and after refinement for uses to compare. Another example is Tensorflow Playground, users can develop an intuition about how neural networks behave by direct manipulation. 6) *Algorithms for Attribution & Feature Visualization*. Using algorithmic techniques to present features that are representative for the results, e. g. Gradient-weighted Class Activation Mapping (Grad-CAM) [7] highlights important regions of the image as a heatmap to better expose the trustworthiness of a classifier and help identify biases in datasets. Except for images, we can also use textual explanations for model decisions. EluciDebug [1] highlights features in the context of each message which could be potentially used when making predictions, which helps users explain necessary corrections back to the learning system.

How to build explainable models

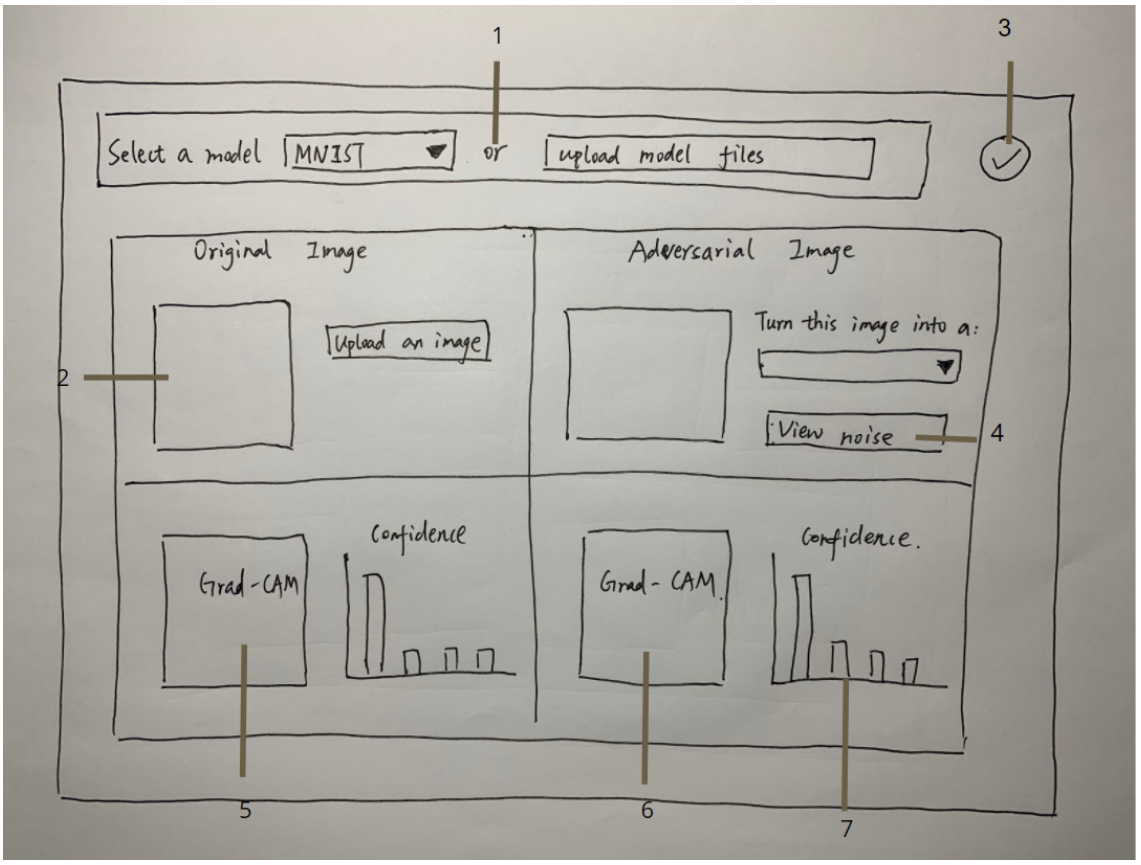
According to Cynthia Rudin [8], explaining black-box ML models is not a comprehensive solution to the problem. Most of the explanations are either not accurate enough (the explanation is always worse than the black-box as otherwise we don't need the black-box at all), not faithful to the true logic of the model (they mimic only the results and not the underlying logic) or not comprehensive enough (they explain only part of the reasoning, e.g. Grad-CAM explains only where the model is looking, but not how it's processing what it sees). Several models have been introduced (for various tasks) that provide explanations along their predictions by design and hence they mitigate the problems described above. Crucially, these models are often as accurate as the state-of-the-art black-boxes. 1) *Prototypical part network (ProtoPNet)* by C. Chen et al. [9] finds a small set of highly discriminatory patches of images (prototypes) from the training dataset. Then, given a new image, it is able to identify several parts of the image where it thinks that this part of the image looks like a prototypical part of some class, and makes its prediction based on a weighted combination of the similarity scores between parts of the image and the learned prototypes. 2) The *attention mechanism* in the *RETAIN* model by E. Choi et al. [10] for time-series prediction / forecasting selects a subset of important features in a sequence that it focuses on. This selection mechanism is quite powerful, which allows to use a simpler predictor on top of it. It's then possible to determine how different variables influence the final prediction (positive / negative influence). In the end we get a subset of important features (where the model looks) together with their influence on the model's decision (what it does with them).

Adversarial attacks

Adversarial attack is a method to add carefully crafted, small (often invisible) noise to an input image that changes its prediction from the true class to any label known to the system. It is surprisingly effective in practice: works digitally, but also physically [12], on the vast majority of models (DNNs, linear models, ensembles), it's largely transferable (the same noise vector can spoil multiple inputs, the same adversarial input will be misclassified by many different models, even different architectures, and they will often make the same mistake!) [11]. This attack 1) creates a security risk and 2) weakens the trust of users in the system. More technical details are presented in Appendix A.

Design Description

Our design targets the general ML community and aims to raise awareness about how general and easy to conduct the adversarial attacks are. Moreover we believe that through the example of adversarial attacks we can advocate for the widespread use of explanation techniques (at least) and perhaps also models explainable by design (at best). More concretely, we want to present to the users (almost anyone, including novices) a simple interface that will allow them to generate an adversarial image that will be indistinguishable from the original, but that will be misclassified as any label they specify. All with a single click of a button and done in real-time.



A sketch of our original design.

1) In our sketch design we first thought about using a pretrained model on a given dataset represented by a selection bar on the top of the sketchpad where you can either upload your model or select an existing one.
2) Image upload : display the original image you uploaded, we didn't draw the fact that you could also take an image(screenshot) by using a webcam on the sketch.
3) model preparing indication (if the model is not pretrained)
4) On the adversarial attack part, first we have a selection bar when you can choose in which label you want to turn your image and secondly a view noise button, which is supposed to be a slide button where you can add or subtract noise and then display the noise to see what happens during an adversarial attack, and finally the display of the adversarial image.
   We want to see if the user can spot the difference between the 2 images (original and adversarial), normally he wouldn't be able to.
5) GRAD-CAM original image: On the left side of the sketch, we're using GRAD-CAM on the original image to see what labels have been identified and where with map activation.
6) GRAD-CAM adversarial image: On the right side of the sketch, this time we use GRAD-CAM but on the adversarial image,we want the user to see which maps activation label are the most relevant for the model, the purpose of displaying the original and the adversarial image is to compare the map activation of each label.
7) Bar graph: this bar graph shows the confidence the model has on each label, we want to show to the user how much the model gets fooled by the adversarial attack.

So this kind of experience with the adversarial attack should raise skepticism and reduce his trust in models but maybe more trust with the use of GRAD-CAM.

Implementation[1]

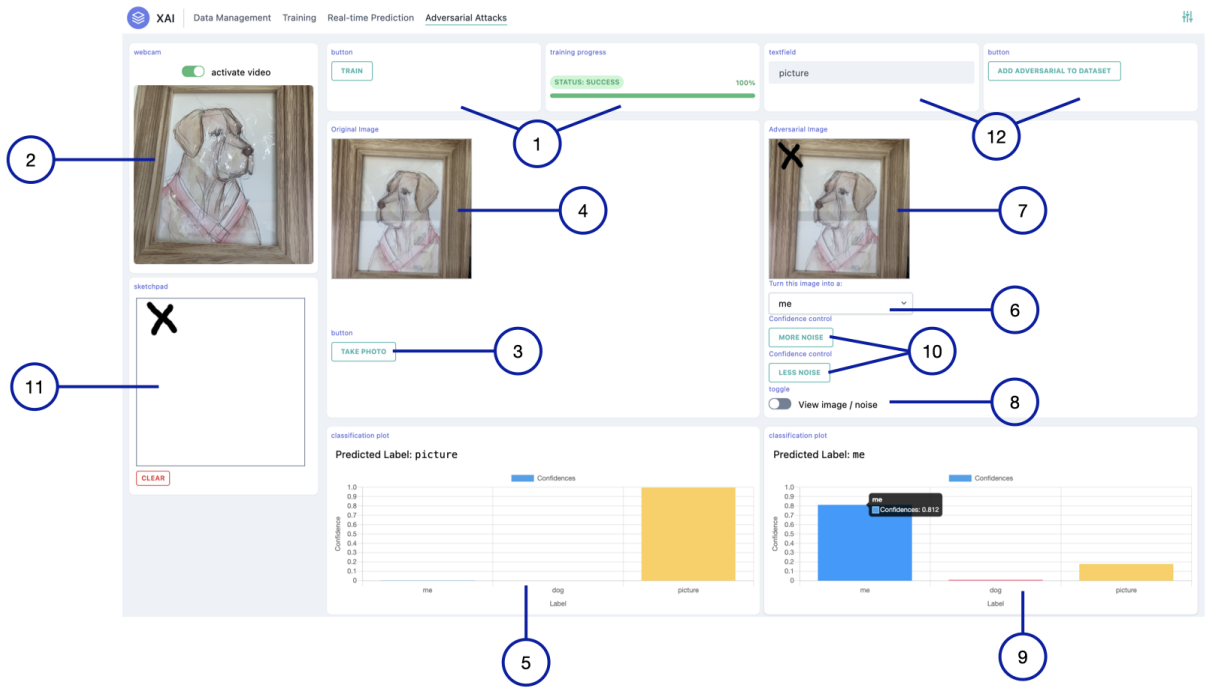
We've implemented the proposed design in the *Marcelle*[2] framework. The application consists of 2 core parts:

- **Preparing the model**: dataset creation, model training and hyperparameter tuning, real-time model evaluation (used for testing).
- **Performing the attack**: capturing the input image, selecting target label, adjusting the level of noise, painting over the image and displaying results.

Preparing the model

Training data for the model is acquired through the webcam and labeled by the user. We use a pre-trained MobileNet feature extractor with an MLP classifier on top. This part of the application is realized through 3 standard pages of the dashboard: 1) Data Management, 2) Training and 3) Real-time Prediction. All these pages are very similar to other *Marcelle* applications, so we do not describe them in detail here.

Performing the attack



The screenshot above shows the interface of the *Adversarial Attack* page of the dashboard. This is the main component of our application. The individual components are (ordered according to the suggested workflow):

[1] The code is available at https://github.com/jakubzadrozny/XAI-project
[2] https://marcelle.dev

1. Training interface: button + progress bar. This functionality is available (in expanded form) under the *Training* page, but is also provided here for convenience as it is required to train the model after each page refresh.
2. Webcam input.
3. Webcam shutter -- adversarial noise is generated only for images captured with a webcam shutter button. This is done for performance reasons and also for the user to compare the original image with the adversarial image easier and in more detail.
4. Preview of the captured image. Adversarial noise is generated for this exact image.
5. Prediction for the original image with confidence scores for each label. Ideally this should be the correct label (if the model is well-trained).
6. Target label dropdown. The user can select any label (that is known to the model) and adversarial noise will be added to the input image so that the prediction of the model changes to the specified label. If the correct label is selected, then the adversarial noise will boost the confidence of the prediction.
7. Adversarial image preview. This is the original image + the adversarial noise. This image should be (almost) indistinguishable from the original, but misclassified as the label selected in 6.
8. Image / noise view toggle. By default the adversarial image (original image + adversarial noise) is displayed, but it may be difficult to spot differences from the original in that view. The user can switch this toggle to inspect only the adversarial noise (enhanced 10x for better visibility). Gray pixels mean no noise, while a deviation from it (towards any color) means more noise.
9. Prediction for the adversarial image with confidence scores for each label. Ideally this should be the label selected in 6.
10. Noise adjustment buttons. Adversarial noise generation is implemented as an iterative procedure where each iteration adds more noise to the image, but it also increases confidence of the target prediction. Initially only one iteration is performed, which may sometimes not be enough to actually change the prediction of the classifier. In this case the user can click on the ‘More noise’ button, which will perform another iteration of the method. This can be repeated as many times as needed, each click will increase the amount (and visibility) of noise, but it will also increase the confidence of the target label (from 6.). The ‘Less noise’ button can be used to undo iterations.
11. The sketchpad can be used to paint over the image. The drawing will be overlayed with the adversarial image and passed to the classifier. This overlay does not interfere with the adversarial noise generation, it is only applied on top of the result. The prediction in 9. is actually the prediction for the overlaid images.
12. Adversarial training interface. After successfully fooling the classifier the user can add the adversarial image with the correct label (of the original) to the dataset, to help the model protect against further attacks. This process is known in the literature as adversarial training and in some cases it can provide additional regularization of the model [11].

Demo

A short demo is available at https://youtu.be/tY86PQtKBqo.

Evaluation

We want to evaluate whether providing explanations can improve users' trust in ML systems. So we designed a between-subject, single-factor experiment to evaluate our system. First we need to recruit some ML novices as participants, then divide them into two groups. The factor we varied is experiment condition: one condition (*control*) try the adversarial attacks without the explanations about the model, while the second condition (*treatment*) use our system to play adversarial attacks with Grad-CAM as explanations. In both conditions we ask participants how much they trust the ML system by Likert questionnaire before and after they try the adversarial attacks. Our hypothesis is that the trust level will decrease in both groups, but less so in the group with explanations. If so, it proves that our system can increase users' trust in ML systems by providing them Grad-CAM as explanations. However, since Grad-CAM isn't such a strong explanation to adversarial attacks, we hope in the future we could have more explanations to see a bigger difference between two groups.

Conclusion

To conclude, this project was extremely interesting and constructive, from the papers we read to the implementation of the application passing by the design of our sketch. We saw that explainable AI is really important in fields like safety critical or socially important applications and the potential lack of trust in models in these cases is a very important issue. We found that there exist six ways to explain current models through visualization and for our project we focused on *Algorithms for Attribution & Feature Visualization* and more particularly on the Grad-CAM (Gradient-weighted Class Activation Mapping) method. We also saw that methods like Grad-CAM are not comprehensive enough because it only explains where the model is looking, but not how it's processing what it sees. That's where models explainable by design, like *ProtoPNet*, really shine. We saw that these models are often as accurate as the state-of-the-art black-boxes, while providing much more insight into their inner-workings, which makes them very compelling.

As we moved to the adversarial attack, we found out that it's surprisingly effective in practice, hence it creates a security risk and also weakens the trust of users in the system.
So, through the literature and the use of our application we came to the conclusion that the most popular models used every day can be easily fooled by the adversarial attack, even if some of them are more robust than others. This really poses the question: *why are none of these models robust to such simple attacks?* and then naturally *how exactly do they work?* That's where explanation techniques and explainable models step in and provide (at least partial) answers. Of course, Grad-CAM alone is not enough to understand why the adversarial attack is so effective, but our design can be extended to incorporate more advanced explanation techniques, which would also be the line of our future work on this project.

References

[1] Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. 2015. Principles of Explanatory Debugging to Personalize Interactive Machine Learning. In Proceedings of the 20th International Conference on Intelligent User Interfaces (IUI '15). Association for Computing Machinery, New York, NY, USA, 126–137. DOI:https://doi.org/10.1145/2678025.2701399

[2] Ribeiro M T, Singh S, Guestrin C. " Why should I trust you?" Explaining the predictions of any classifier[C]//Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 2016: 1135-1144.

[3] Spinner T, Schlegel U, Schäfer H, et al. explAIner: A visual analytics framework for interactive and explainable machine learning[J]. IEEE transactions on visualization and computer graphics, 2019, 26(1): 1064-1074.

[4] Hohman F, Kahng M, Pienta R, et al. Visual analytics in deep learning: An interrogative survey for the next frontiers[J]. IEEE transactions on visualization and computer graphics, 2018, 25(8): 2674-2693.

[5] Z. J. Wang et al., "CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization," in IEEE Transactions on Visualization and Computer Graphics, vol. 27, no. 2, pp. 1396-1406, Feb. 2021, doi: 10.1109/TVCG.2020.3030418.

[6] Spinner T, Schlegel U, Schäfer H, et al. explAIner: A visual analytics framework for interactive and explainable machine learning[J]. IEEE transactions on visualization and computer graphics, 2019, 26(1): 1064-1074.

[7] Selvaraju R R, Cogswell M, Das A, et al. Grad-cam: Visual explanations from deep networks via gradient-based localization[C]//Proceedings of the IEEE international conference on computer vision. 2017: 618-626.

[8] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence, 1(5), 206-215.

[9] Chen, C., Li, O., Tao, C., Barnett, A. J., Su, J., & Rudin, C. (2018). This looks like that: deep learning for interpretable image recognition. arXiv preprint arXiv:1806.10574.

[10] Choi, E., Bahadori, M. T., Kulas, J. A., Schuetz, A., Stewart, W. F., & Sun, J. (2016). Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. arXiv preprint arXiv:1608.05745.

[11] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

[12] Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world.

Appendix A: Technical literature and implementation of the adversarial attack

Our task here is to find an image very similar to the original s.t. it will be misclassified by the model at hand.

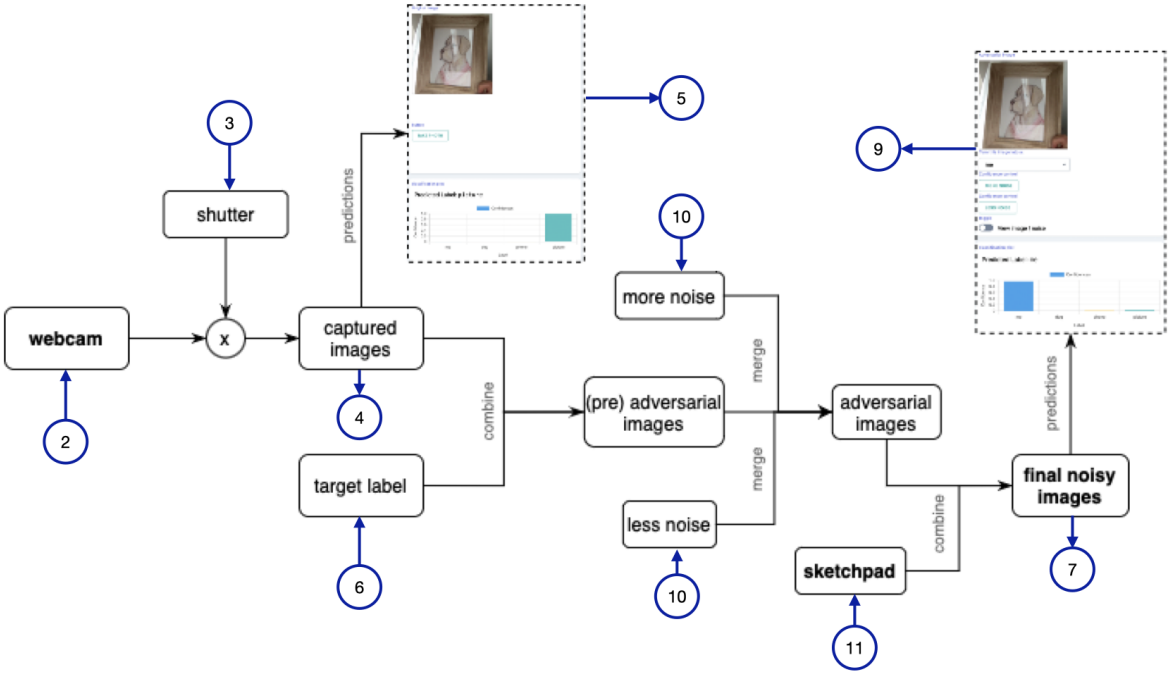
The first approach is to have the new image maximize the loss function [11]. This can be obtained by modifying the original by a (noise) vector that points the same way as the gradient of the loss function (has the same sign) and has absolute value ε on all coordinates. This method will usually produce images that will be misclassified by the model for values of ε so small that the new (adversarial) image is indistinguishable from the original. However there is no control over what exactly will be the prediction on the adversarial image.

To produce an image that will be misclassified as a pre-specified label we can maximize the model's confidence score of that label instead. This can be obtained by taking noise that is a scaled value of the gradient of the confidence score for the target label. The scale can be adjusted so that the noise has a specified magnitude, which also corresponds to the visual difference between the original and adversarial image. Our model's confidence in the wrong prediction can be maximized by iterating the step described above. We've implemented this iterative procedure in our application and it is integrated into the interface as described in the *Implementation* section.

All the gradients used above can be efficiently computed via backpropagation and they are easily accessible to the programmer via an automatic differentiation package such as `tensorflow.js`[3] (used by *Marcelle* and in this project as well). It is enough to wrap the forward pass of the model in a `tf.grad` function to access the gradients. All the iterations of the noise are also stored in an array to allow easy undo functionality.

[3] https://www.tensorflow.org/js

Appendix B: Communication between components



All the components described in the *Implementation* section communicate through `most.js`[4] streams. The hierarchy, structure and construction of those streams is illustrated in the figure above (each box is a different stream). The numbers in circles correspond to the numbers of components in the *Implementation* section. An arrow away from a circle marks an input component and an arrow towards a circle marks an output component.

[4] https://github.com/cujojs/most