

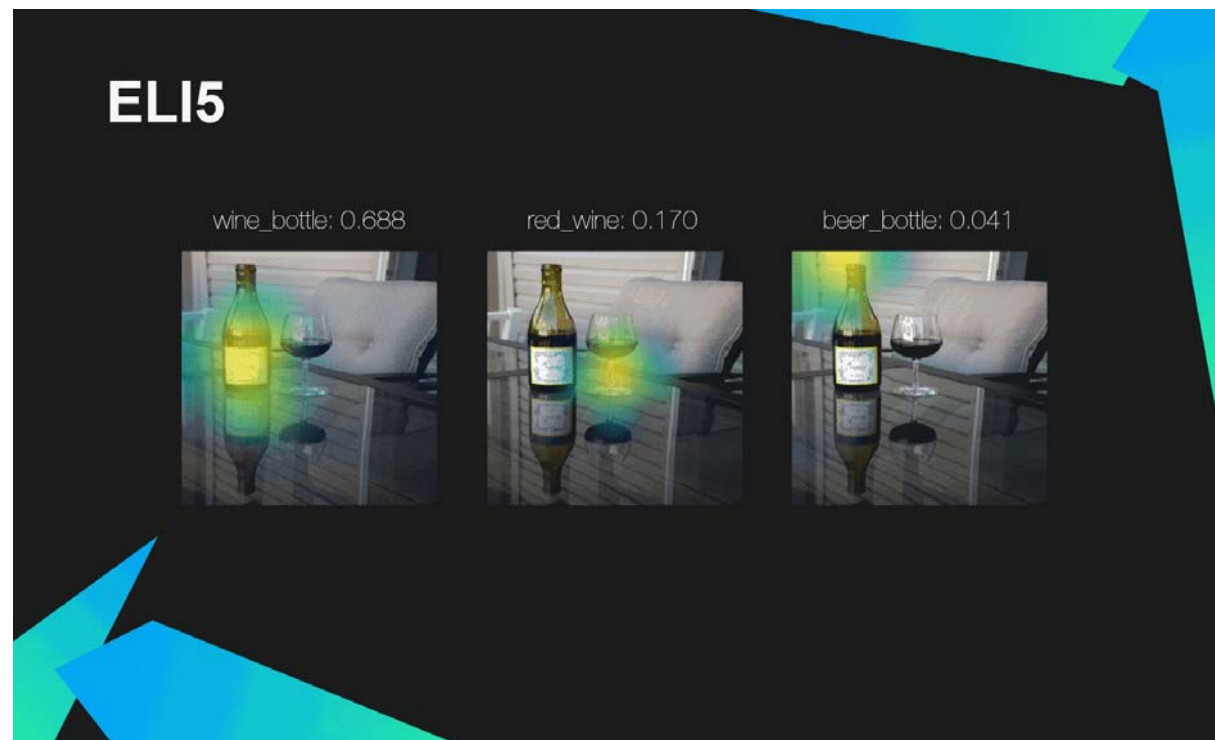
Knowing What and Why? — Explaining Image Classifier Predictions

Data Science Toolkit



Piotr Skalski [Follow](#)

Feb 20 • 9 min read



Comparison of explanations provided by tested libraries

Introduction

It is a fact that with every year Machine Learning (ML) is becoming an increasingly important part of our lives. ML helps us to perform simple daily tasks like finding the optimal route to home, accurately scanning documents with our phone or translating the text into different languages.

However, it is also used as a key element in many highly responsible systems, where human life and well-being is at stake.

- Artificial Intelligence (AI) is used in the judicial system to assess how likely it is that a convicted person will break the law once again.
- Companies are working on systems that could assist or even completely relieve doctors of the need to locate tumors on medical images

As we implement systems similar to those mentioned above, it is becoming progressively clear that **we must provide not only predictions but also explanations** as to what influenced our decision. In this post, I'll compare and benchmark the most commonly used libraries for explaining the model predictions in the field of **Image Classification** — **Eli5, LIME, and SHAP**. We are going to investigate the algorithms that they leverage, as well as compare the efficiency and quality of the provided explanations.

Note: Because I don't want to bore you with scrolling through the huge snippets of code, I've only put a tiny part of it in the article. However, if you would like to create similar visualizations, or learn how to explain your classifier's predictions, I encourage you to visit my GitHub.

Trade-Off

Before we start, let's answer the most basic — but at the same time an important — question. Why do we even need explanations? **Can't we just interpret the model's predictions** to understand what influenced them? Well, unfortunately, in many cases, **the answer is no...**

A trade-off between the potential accuracy of the model and its interpretability

We can argue that there is a **trade-off between the complexity of a model and its interpretability**. And since some problems — especially in the field of Computer Vision — are highly complicated, we have to use strong and complex models to solve them. These types of models are often referred to as Black Boxes because we do not know or understand what is happening inside them. Thus, it often happens that we sacrifice interpretability in favor of the accuracy of our model.

Image Classification

As I mentioned in the first paragraph, in this project we will deal with Image Classification models. It is one of the oldest and best-recognized Computer Vision (CV) tasks — it involves assigning a label with the name of the object class to a photo. Despite its apparent simplicity, this is a task that has caused enormous problems for researchers over the years. The breakthrough idea was AlexNet — Convolutional Neural Network (CNN), used for the first time in the ImageNet competition in 2012. AlexNet beat the other participants with an error that was about 10 percentage points lower. This solution revolutionized Computer Vision forever. Since then, CNN has become the default answer for all tasks in this branch of research. Unfortunately, CNN is an example of the Black Box — although we know the transformations taking place inside it, their full interpretation is a problem even for experts in the field.

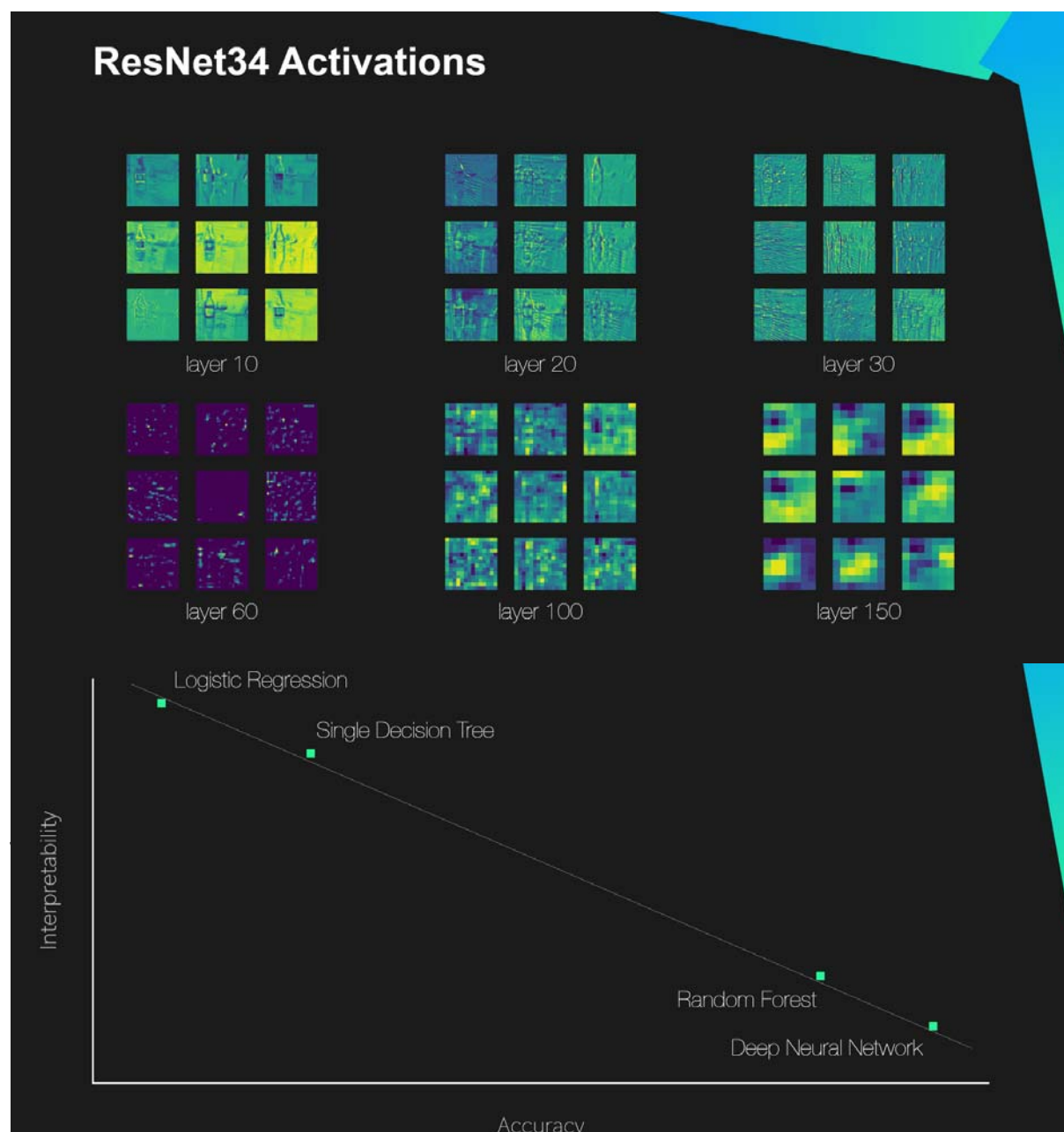
Note: If you want to explore CNN deeper, I strongly encourage you to read my other post — Gentle Dive into Math Behind Convolutional Neural Networks.

ELI5

The first library we will look into is Eli5 — it's a simple but reliable tool designed to visualize, inspect and debug ML models. The library allows, among other things, to interpret the predictions of Image Classifiers written in Keras. To do it, Eli5 leverages Gradient-weighted Class Activation Mapping (Grad-CAM) algorithm. It is worth noting that this is not a general approach and it applies only to CNN solutions.

Grad-CAM

Grad-CAM is the next iteration of the CAM, which was one of the first ideas for visualizing the predictions of CNN. It's using the values of activation maps as well as backpropagation to understand the source of the prediction.



CNN light up in response to the presence of specific, complex patterns.

Besides, most often the activation maps in deep CNN layers have a lower resolution than those at the beginning of the graph.

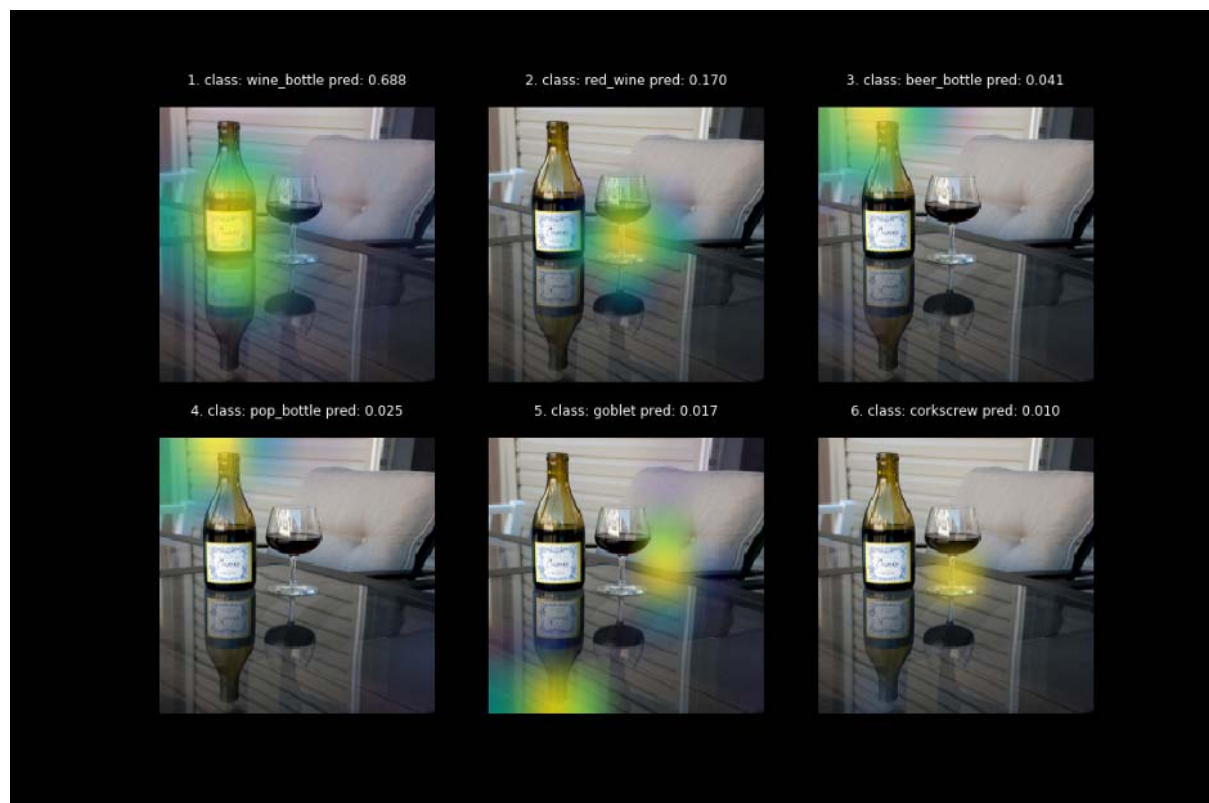
Grad-CAM explanations are obtained by performing a linear weighted combination of forward-activation maps followed by a ReLU. Coefficients of the linear equation are calculated by averaging the gradient of score obtained for given classes (before Softmax), in relation to the feature map activations.

$$L_{Grad-CAM}^c = ReLU(\sum_k \alpha_k^c A^k)$$

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

Example

A code snippet that allows explaining any Keras based CNN prediction using Eli5 — full example



Explanations obtained with the Eli5 for the ResNet34 model

Summary

- **[+]** The idea behind Eli5 is simple and computationally cheap, therefore, the algorithm is both simple to understand and **very quick** to execute.
- **[−]** The method is intended **only to explain CNN predictions**.
- **[−]** Currently, the library **only supports Keras-based models**.
- **[−]** The precision of the method is limited — it depends on **activation map resolution**.

Visualization showing how the resolution of CNN activation affects the precision of explanations

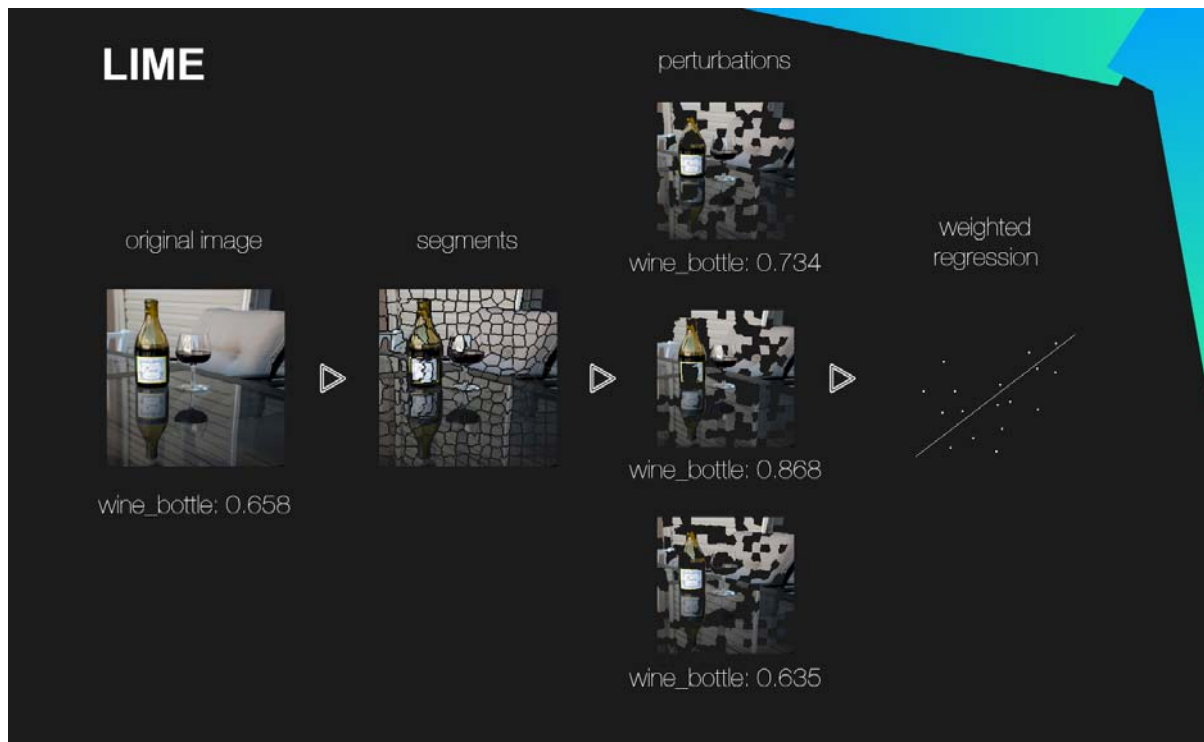
LIME

Why Should I Trust You?: Explaining the Predictions of Any Classifier is an article that underlies the entire branch of research aimed at explaining ML models. The ideas included in this paper became the foundation for the most popular of the interpretation libraries — Local interpretable model-agnostic explanations (LIME). This algorithm is completely different from Grad-CAM — tries to understand the model by perturbing the input data and understanding how these changes affect the predictions.

Surrogate Models

This time our goal is to replace the complex Black Box model with a simple, interpretable, linear one. Of course, the new model is only an approximation of the original but chosen so that it faithfully represents it locally. The first step in this direction is to divide the image into super-pixels — groups of neighboring pixels with similar color and brightness. Such an approach makes sense because the classification of a photo is probably determined by many pixels, so perturbation of single one would have a marginal impact on the prediction. Then we create a collection of artificial

photos created by replacing random super-pixels of the original photo with grey color.



LIME algorithm

We also define a function π that will allow us to determine the level of similarity between the original image and a sample, where D is cosine distance between images and σ is the width of the image. To calculate this metric, we flatten our digital photo and treat it as a vector. Then we calculate the cosine of the angle between the vectors representing the two images being compared.

$$\pi_z(z') = \exp(-D(z, z')^2 / \sigma^2)$$

Now all you need to do is find the prediction of the original model for the input image $f(x)$ as well as the predictions of a linear model for perturbed images $g(z')$ and solve the weighted regression defined by L .

$$L(f, g, \pi_z) = \sum_{z, z' \in Z} \pi_z(f(z) - g(z'))^2$$

Robustness

One of the key features we certainly expect from our dream explanation library is robustness. We want two almost identical images to give a very close explanation. It turns out, however, that **LIME is highly unstable** in this aspect.

To verify this, I conducted an experiment — I explained the predictions of the Image Classifier, for two almost identical images. The second picture was created by adding a tiny amount of Gauss noise to the original one — so small that the probability obtained from the model almost does not change, and the images are indistinguishable to the naked eye. However, the difference in the explanations turned out to be significant.

Example

```

1 def plot_eli5_top_explanations(
A code snippet that allows explaining any Image Classifier prediction using LIME — full example
    model: Model,
    image: np.array,
    class_names_mapping: Dict[int, str]

```

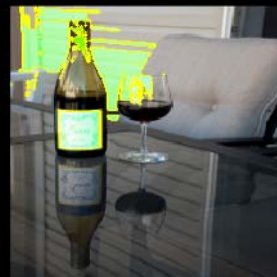
1. class: wine_bottle pred: 0.688



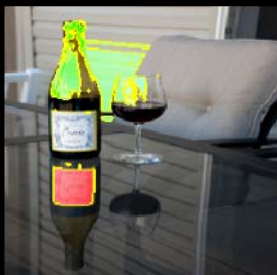
2. class: red_wine pred: 0.170



3. class: beer_bottle pred: 0.041



4. class: pop_bottle pred: 0.025



5. class: goblet pred: 0.017



6. class: corkscrew pred: 0.010



```

22
23     Explanations obtained with the LIME for the ResNet34 model
24     class_grad_cam = eli5.show_prediction(model, image, targets=[int(index)])
25     subplot_title = "{}. class: {} pred: {:.3f}".format(i + 1, name, value)
26     ax.imshow(class_grad_cam)
27     ax.set_title(subplot_title, pad=20)
28

```

Summary

- **[+] Model-agnostic method** — fully independent of the class or architecture of the ML model.
- **[+] Library** has a ready-made implementation that allows you to explain predictions of any Image Classifier.
- **[–] Time-consuming** calculations, that depending on the selected hyperparameters, can last up to several minutes for a single image.
- **[–] Stability problems** — even tiny change of image, leads to significantly different explanations

SHAP

Shapley Additive exPlanations (SHAP) and LIME are quite similar — both are additive and model-agnostic methods of explaining individual predictions. SHAP however, aims to explain model prediction for given input by computing the contribution of each feature to this prediction. To achieve this goal, SHAP uses Shapley Values, which originally comes from game theory.

Shapley Values

First of all, to better understand the analyzed algorithm, let us explain what is Shapley Value. It is a method described by Shapley (1953) as a way of assigning a reward to game players based on their contribution to total gain. The idea has been transferred to SHAP as a way of assessing which of the features has the greatest contribution to the resulting prediction of the model. It is described as the average marginal contribution of a feature value across all possible coalitions. Coalition vector \mathbf{z}' assigns a value of 0 or 1 to each feature, defining whether it is present in the coalition. These vectors are mapped to the feature space using the function \mathbf{h} — for images, the function fills in grey super-pixels, for which the corresponding vector value is 0.

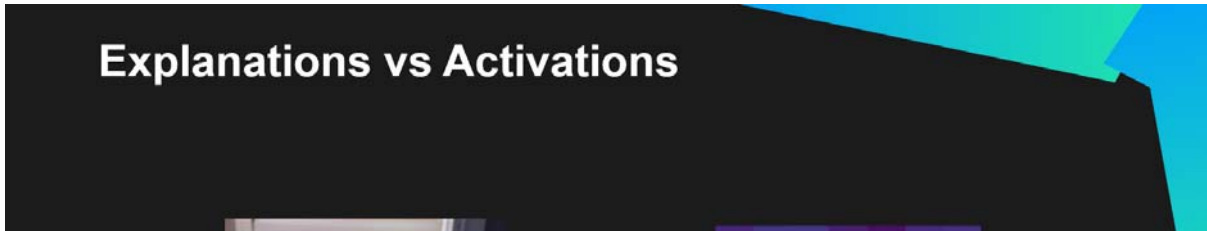
KernelSHAP it's a combination of ideas, known from LIME but with the use of Shapley Values. Similarly, as with LIME, we use super-pixels — to limit the

number of features, combining pixels of similar color and brightness. We also use a weighted regression to build a linear model, where each super-pixel represents a single feature of our model.

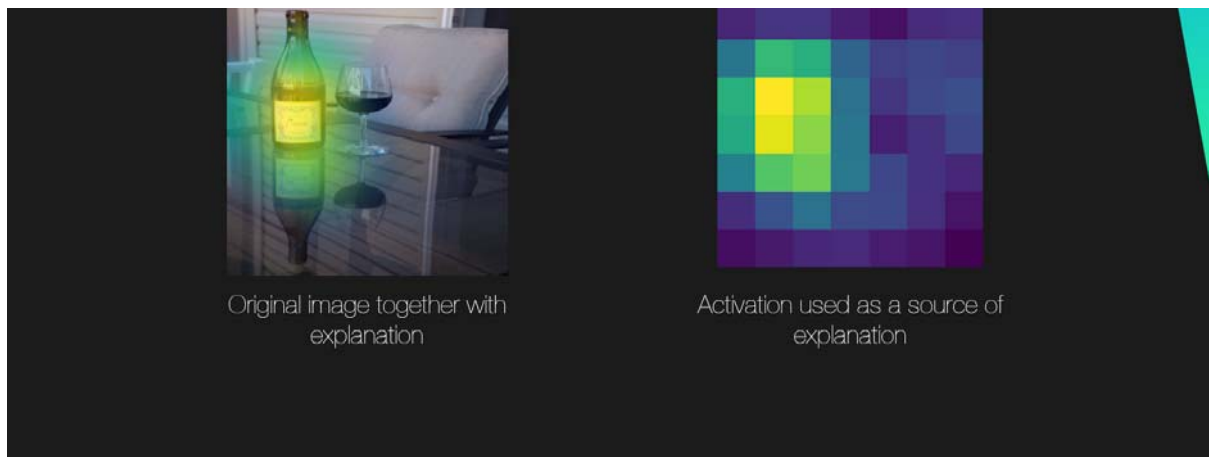
The key difference lies in the selection of the regression model weights. For LIME, it was a cosine measure between the original and the perturbed image, while for KernelSHAP, the weights are determined using the following formula, where M is the maximum coalition and $|z'|$ is the number of features in a considered coalition.

$$\pi_z(z') = \frac{(M - 1)}{\binom{M}{|z'|} |z'| (M - |z'|)}$$

Example



Explanations vs Activations



A code snippet that allows explaining any Image Classifier prediction using SHAP — full example



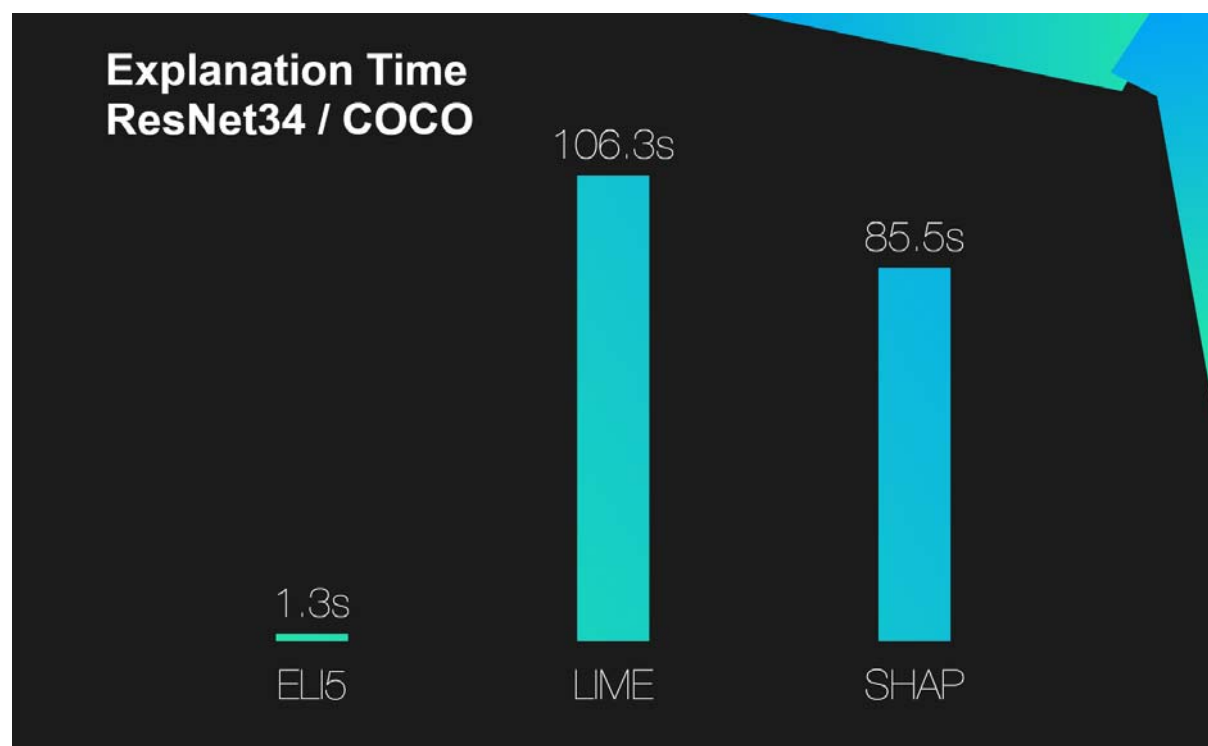
Explanations obtained with the SHAP for the ResNet34 model

Summary

- **[+]** Solid mathematical basis derived from game theory, which makes it highly interpretable.
- **[+]** Model-agnostic approach
- **[−]** Extremely slow as it requires the calculation of many Shapley Values
- **[−]** KernelSHAP is a general algorithm and if we want to use it to explain predictions of CV model we need to define our function h mapping the coalition to image features.

Benchmarks

We will use the wall time, necessary to perform the model explanation, as our metric of library performance. To avoid the influence of factors unrelated to the interpretation of the prediction, we will rely on the **average time for 100 individual cases**. The obtained values are listed below. Tests were performed on a computer without a GPU, equipped with a processor Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz.



Average Wall Time of explanation for tested libraries

Conclusions

Congratulations if you managed to get here. Big thanks for the time spent reading this article. If you liked the post, consider sharing it with your friend, or two friends or five friends. I hope this article turned out to be helpful and now you know how to explain the predictions of your Image Classifier.

This article is another part of the “Data Science Toolkit” series, if you haven’t had the opportunity yet, read the other articles. Also, if you like my job so far, follow me on Twitter and Medium and see other projects I’m working on, on GitHub and Kaggle. Stay curious!

. . .



[Deep Learning](#)

[Computer Vision](#)

[Machine Learning](#)

[Neural Networks](#)

[Towards Data Science](#)

[About](#) [Help](#) [Legal](#)


```

181 def plot_lime_top_explanations(
182     model: Model,
183     image: np.array,
184     class_names_mapping: Dict[int, str],
185     top_preds_count: int = 3,
186     fig_name: Optional[str] = None
187 ) -> None:
188
189     image_columns = 3
190     image_rows = math.ceil(top_preds_count / image_columns)
191
192     explanation = explainer.explain_instance(
193         image,
194         classifier_fn = model.predict,
195         top_labels=100,
196         hide_color=0,
197         num_samples=1000
198     )
199
200     preds = model.predict(np.expand_dims(image, axis=0))
201     top_preds_indexes = np.flip(np.argsort(preds))[0,:top_preds_count]
202     top_preds_values = preds.take(top_preds_indexes)
203     top_preds_names = np.vectorize(lambda x: class_names[x])(top_preds_indexes)
204
205     plt.style.use('dark_background')
206     fig, axes = plt.subplots(image_rows, image_columns, figsize=(image_columns * 5, image_rows * 5),
207                             [ax.set_axis_off() for ax in axes.flat])
208
209     for i, (index, value, name, ax) in \
210         enumerate(zip(top_preds_indexes, top_preds_values, top_preds_names, axes.flat):

```



```

184 def plot_shap_top_explanations(
185     model: Model,
186     image: np.array,
187     class_names_mapping: Dict[int, str],
188     top_preds_count: int = 3,
189     fig_name: Optional[str] = None
190 ) -> None:
191
192     image_columns = 3
193     image_rows = math.ceil(top_preds_count / image_columns)
194
195     segments_slic = slic(image, n_segments=100, compactness=30, sigma=3)
196
197     def _h(z):
198         return model.predict(preprocess_input(mask_image(z, segments_slic, image, 255
199
200
201     explainer = shap.KernelExplainer(_h, np.zeros((1,100)))
202     shap_values = explainer.shap_values(np.ones((1,100)), nsamples=1000)
203
204     preds = model.predict(np.expand_dims(image, axis=0))
205     top_preds_indexes = np.flip(np.argsort(preds))[0,:top_preds_count]
206     top_preds_values = preds.take(top_preds_indexes)
207     top_preds_names = np.vectorize(lambda x: class_names[x])(top_preds_indexes)
208
209     plt.style.use('dark_background')
210     fig, axes = plt.subplots(image_rows, image_columns, figsize=(image_columns * 5, image_rows * 5),
211                             [ax.set_axis_off() for ax in axes.flat])
212
213     max_val = np.max([np.max(np.abs(shap_values[i][:,:-1])) for i in range(len(shap_v
214     color_map = get_colormap()

```