

# Explainable AI: Explaining predictions of Tensorflow Models

By Kartik Parnami (kp2844) and Michelle Maria Roy(mr3966)

## Project Summary

Create a consolidated Python library which works with models of the user's choice as part of their training or prediction functions to explain predictions of both text based and image based deep network models specific to classification.

We hope to help remove the black box element from Deep Learning models built to large scale with high levels of accuracy by explaining their predictions

## Problem Motivation

### Explainability

The explainability problem is well known in Machine Learning. Deep Neural Networks helps us solve complex problems, usually as a "Black Box" that makes intelligent decisions. But we always lack clarity about how our model reaches a particular endpoint i.e. What are the key contributing factors (or input features)? In some application spaces like medicine, this lack of explainability is not just inconvenient, but unacceptable. We need more explanations about what was learnt for predictions.

### Lack of Consolidated Resources

Although there exist several methods to explain a model, their current implementations cannot be applied to all types of models and are scattered across libraries. Many explanation techniques have been created and are easily available for image models (especially CNNs) and the work around textual models is pretty scarce. We aim to create a consolidated model explanation library (in Python) for images and textual models (specific to classification) which is easily import-able and compatible with Tensorflow/Keras.

## Background Work

We cover the following three macro categories for a complete overview:

- **Blackbox Methods:** Assumes one cannot access the model (ie. the weights of the model). These model independent methods typically explain the predictions of any classifier by learning an interpretable model locally around the prediction. *e.g. Anchors*
- **Whitebox Methods:** Accesses the weights learned by the model and tries to use these weights to better understand what input features activated the weights while the model computed the results. *e.g. Integrated Gradients*

- **Adversarial Methods:** Use perturbations in input to “deceive” a model to predict a contrary result. The amount of perturbations give us an idea about their importance. *e.g. Counterfactual Explanations*

The methods we cover in this library are as follows:

- **Anchors:** An anchor explanation is a rule that sufficiently “anchors” the prediction locally – such that changes to the rest of the feature values of the instance do not matter. Reinforcement learning methods are used to come up with the set of feature conditions locally to explain a data point.  
*References:* [Anchors Paper](#), [author’s implementation of anchors’](#)
- **Integrated Gradients:** Use the gradients of the output with respect to the input to create approximation of the (nonlinear) deep network. Combines inputs at different scaling factors to get overall feature importance.  
*References:* [Integrated Gradients Paper](#), [Unofficial Google Data Science blog](#)
- **Counterfactual Explanations:** Look for minimal constrained change (additions), to the input for the decision of the black-box model to flip and identify important features accordingly.  
*References:* [Counterfactual Explanations Paper](#), [Pytorch Implementation of Counterfactuals](#)

## Technical Challenges

Existing work do provide various black box and white box techniques, but these are **scattered across libraries**. We were unable to find many references available for text-based models. We had to make sure our plug and play library that is **model agnostic** and allowed us to explain any text-based or image-based model in Keras/Tensorflow.

There is no way to **measure the performance of the model** explainers and decide which works better than the other. Each method comes with their own set of trade off with different amounts of bias.

## Implementation

Clone-able, lightweight [git repository](#) for easy access to any Tensorflow notebook/script. Anchors and Counterfactuals not limited to deep neural net models and can be extended to any machine learning model. We provide a uniform, easy-to-use interface for all explanation techniques:

- Initialize the explainer: `explainer = XXXExplainer()`
- Visualize the results: `explainer.explain_instance(instance, model, ...)`

## Anchors

Anchors uses “local region” to learn how to explain the model. The four main components of the approach we follow are:

1. We first generate new explanation candidates.

2. Candidate rules are to be compared to see which rule explains the best. To this end, perturbations that match the currently observed rule are created and evaluated using precision - which states how well the rule describes the instance to be explained.
3. We take more samples in case there is no statistical confidence yet or in the event that the candidate exceeds the predetermined threshold.
4. All of the above components are assembled in a beam search, which is a graph search algorithm and a variant of the breadth-first algorithm. It carries the best candidates of each round over to the next one.

The same approach is used for text and images

Real cat examples

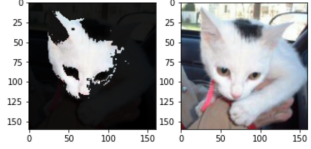
```
In [ ]: #####
# Correct Prediction #
#####

image = img[0][4].numpy()
anchor_explainer = AnchorImage()

scaled_image, segments, results = anchor_explainer.explain_instance(image, model.predict, threshold=0.99)
print(results)
fig, axes = plt.subplots(1, 2)
axes[0].imshow(scaled_image.astype('uint8'), cmap=plt.cm.gray)
axes[0].imshow(image.astype('uint8'), alpha=0.1)
axes[1].imshow(image.astype('uint8'))

True pred 0
[(49, '', 1.0, [], 0)]

Out[ ]: <matplotlib.image.AxesImage at 0x7f04a16876a0>
```



Correctly predicted cat and the pixels that serve as the anchor for the prediction.

## Integrated Gradients

Integrated Gradients assumes less bias in model explanation as it accesses the weights learnt.

The algorithm is as follows:

1. The gradients w.r.t model are computed, using the standard gradient operator, at set intervals on the interpolation path between input and it's baseline input (uninformative input).
2. We then find the integrated gradients for the input. The gradients computed at different intervals of the interpolation path form the integrated gradients by cumulating the gradients along the path.
3. Multiplying the difference in the image and the baseline with the computed integrated gradients tells us exactly what pixels were activated and contributed to the classification.

Both text and image data follow a similar implementation with the exception of baseline input used (zero word embedding vector or black image)

#### Real negative reviews example

```
In [10]: #####
# Correct Prediction #
#####

idx_to_visualize = 3
pred_dict = {1: 'Positive review', 0: 'Negative review'}
print('Predicted label = {}: {}; Real label = {}: {}'.format(predictions[idx_to_visualize],
                                                             pred_dict[predictions[idx_to_visualize]],
                                                             0 if y_test_sample[idx_to_visualize][0] > y_test_sample[idx_to_visualize][1] else 1,
                                                             pred_dict[0 if y_test_sample[idx_to_visualize][0] > y_test_sample[idx_to_visualize][1] else 1]))

words = decode_sentence(x_test_sample[idx_to_visualize], reverse_index).split()
ig_explainer.visualize(attrs[idx_to_visualize], words)

Predicted label = 0: Negative review; Real label = 0: Negative review

Out[10]: the mother in this movie is reckless with her children to the point of neglect i wish i wasn't so angry about her and her actions
because i would have otherwise enjoyed the flick what a number she was take my advice and fast forward through everything you see
her do until the end also is anyone else getting sick of watching movies that are filmed so dark anymore one can hardly see what is
being filmed as an audience we are UNK involved with the actions on the screen so then why the hell can't we have night vision

The most important features that contribute to the negative prediction as identified by the Integrated Gradients technique
```

## Counterfactual Explanations

A counterfactual explanation of a prediction describes the smallest change to the feature values that causes the prediction to flip a predefined output. The counterfactual example must fulfil the following criteria:

1. A counterfactual instance produces the predefined prediction as closely as possible.
2. A counterfactual should be very similar to the instance regarding feature values.
3. A counterfactual instance should have feature values that are likely/realistic.

While implementing counterfactual explanations for images and text we create counterfactual examples relative to an input baseline. For text-based models we have also extended the creation of counterfactual examples using antonyms.

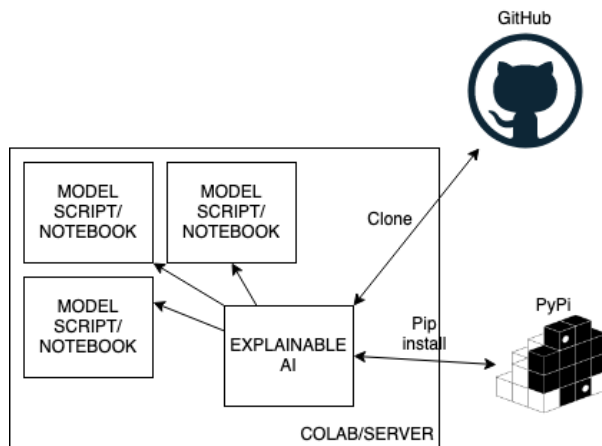
```
In [11]: #####
# Correct Prediction #
#####

text = "Cheap and delicious! I eat here about once a week because it fills me up for cheap price and the people who work there are really fun . it 's like a subway sandwich restaurant for Mexican food . I love hate the burrito w/ Cheese , Beans & rice w/ sour cream , less than $ 5 and best bad in town for that price ."
lc_explainer.explain_instance(text, predict_lr_counterfactual, with_antonyms=True)

Out[11]: Cheap and delicious displeasing! I eat here about once a week beginning because it fills me up for cheap price and the people who
work there are really fun . it 's like a subway sandwich restaurant for Mexican food . I love hate the burrito w/ Cheese , Beans & rice
w/ sour cream , less than $ 5 and best bad in town for that price .
```

The words with the most contribution towards the positive prediction are replaced one-by-one by an antonym to understand their importance to the prediction. Lime is used as the heuristic to get their relative importance.

## Architecture Diagram



## Experiment Flow

Our experiments consist of 6 colab notebooks, one covering each of the following techniques:

- Anchor explanations for text
  - Yelp polarity dataset, Custom model (Google News embedding + dense layers)
- Lime-based counterfactuals for text
  - Yelp polarity dataset, Custom model (Google News embedding + dense layers)
- Integrated Gradients for text
  - Imdb reviews dataset, Custom model (ConvNN + dense layers)
- Anchor explanations for images
  - Cats vs Dogs dataset, MobileNetV2 with Transfer Learning
- Lime-based counterfactuals for images
  - Cats vs Dogs dataset, MobileNetV2 with Transfer Learning
- Integrated Gradients for images
  - Cats vs Dogs dataset, MobileNetV2 with Transfer Learning

Each notebook covers one example each of: true positive, true negative, false positive, false negative. The following were our findings:

1. Explanations for correct predictions concur with each other especially for images
2. Integrated gradients show a much smoother attribution as compared to the local explanation techniques
3. Many times, anchors by themselves are not sufficient to explain the predictions and we have to look at further attributions.
4. Replacement of words with antonyms leads to a quicker convergence towards finding counterfactuals as compared to replacement with baseline.
5. The local blackbox explanation techniques complement each other's explanation much more than the whitebox technique.

## Limitations and future scope

- Currently we support only binary classification at the moment. We can easily extend to multi-class classification and regression problems
- We want to make a feature extension to make it pip-installable
- Create support for structured tabular data explanations.
- Beam search on anchors is erratic at times. We hope to run more test and solve this.
- More rigorous evaluation on bigger datasets and models needs to be done.
- Method to evaluate the goodness of an explanation, currently a human understanding approach.