Written by: Idan Kashtan

Explainable AI (XAI)

Most of the time, we need more than just the prediction of our model. We want to explain why the model thinks that way and use this explanation in several ways. First, we can take our explanation and give it to the end-users for them to understand the prediction. Second, we need it as developers and researchers so we can debug our model and make it better. Third, the decision-makers should base their decisions on that explanation and regulators. We have a few types of XAI, Global and local explanations, model-based, post-hoc, black and white box models.

In this report, I focus on LIME and how I implemented it.

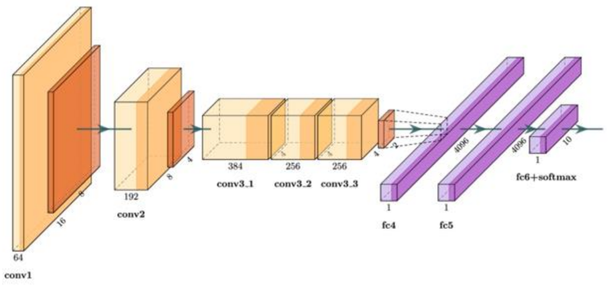
LIME

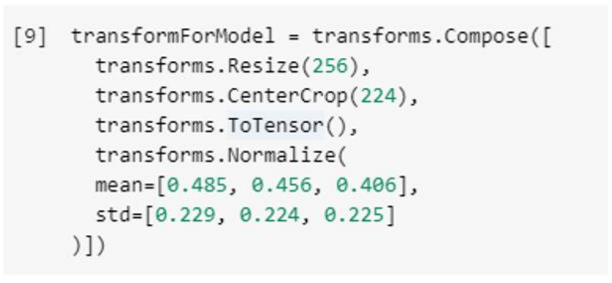
Lime stands for Local Interpretable Model-Agnostic Explanations. It uses a black-box approach, indicating we don't have access to the models' internals, such as weights and gradients. It's flexible. We can explain tabular, text, and image data. We want to explain what features were most important for this specific input, local explanation. We'll use a Perturbation-based explanation, and we'll see what it is in a bit.

The best way to learn what LIME is, is to implement it, so let's start with the implementation.

Implementation

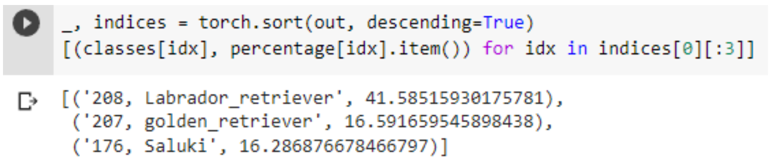
First, we need to choose a model that we want to explain its results. I choose AlexNet it has a few convolutional layers and a few FC at the end. It's important to note that we need to match our input to the training dataset that the model trained. Keep in mind that we don't know how to explain this model, we'll see what we can do to explain it later.





```
[9]  transformForModel = transforms.Compose([
       transforms.Resize(256),
       transforms.CenterCrop(224),
       transforms.ToTensor(),
       transforms.Normalize(
       mean=[0.485, 0.456, 0.406],
       std=[0.229, 0.224, 0.225]
     )])
```

We pass this input through the network, and these are the top 3 results, Labrador retriever, Golden retriever, and Saluki.

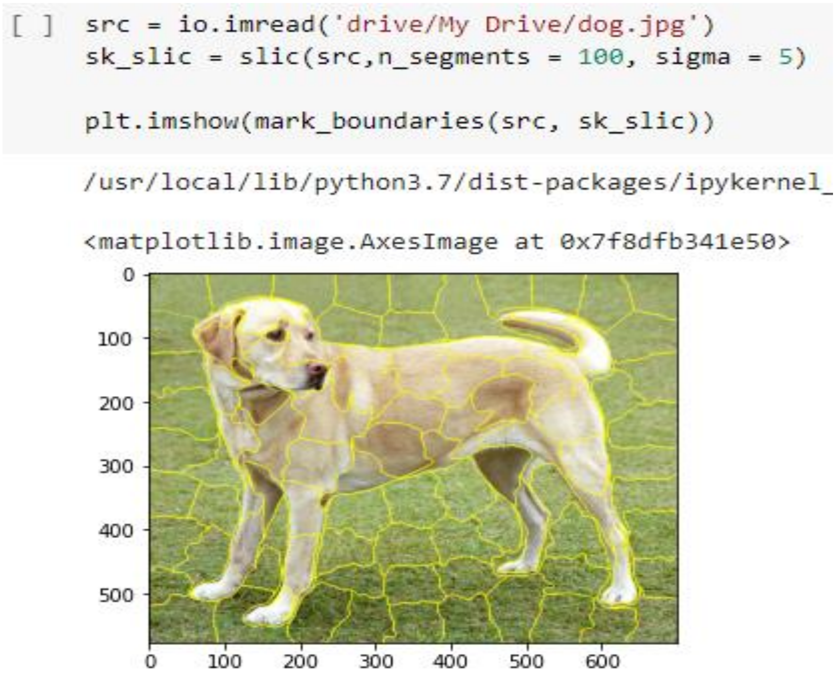


```
_, indices = torch.sort(out, descending=True)
[(classes[idx], percentage[idx].item()) for idx in indices[0][:3]]
```

```
[('208, Labrador_retriever', 41.58515930175781),
 ('207, golden_retriever', 16.591659545898438),
 ('176, Saluki', 16.286876678466797)]
```

Simplified Image

We know that the input has 256*256 pixels, so we have 65,536 features that could explain why the model thinks it's a Golden retriever. If we keep that number, the explanation will be too noisy and not informative. To reduce the number of features, we'll use superpixel. I used Slic, an algorithm to find superpixels inside an image.

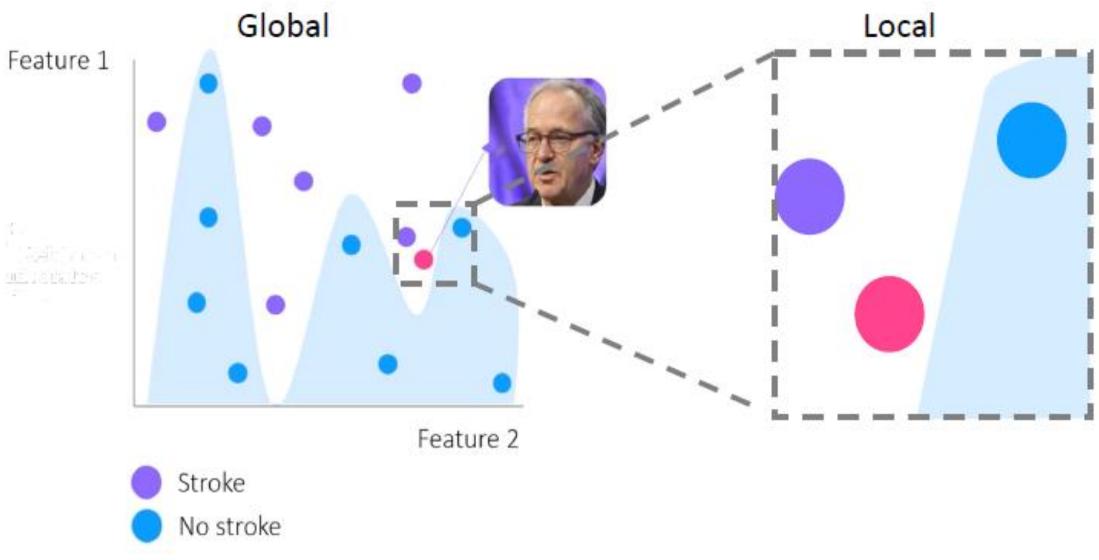


```
[ ]  src = io.imread('drive/My Drive/dog.jpg')
     sk_slic = slic(src,n_segments = 100, sigma = 5)

     plt.imshow(mark_boundaries(src, sk_slic))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_

<matplotlib.image.AxesImage at 0x7f8dfb341e50>
```

Now that we have a simplified version of the input, we can start the explanation process. Because we use a local explanation, we want to generate a new perturbed dataset. The newly generated data will be close to the original input but not identical.

We want to explain only this part around our point.

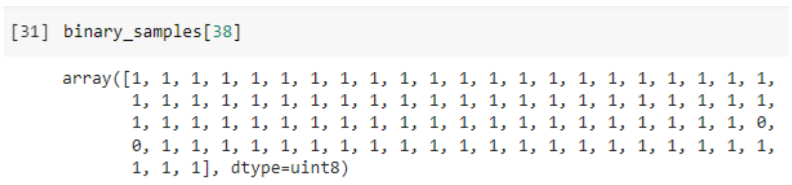


Generate Perturbed Dataset

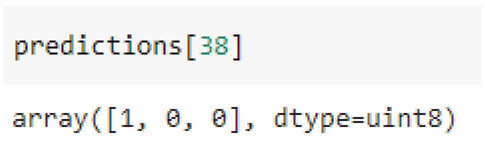
Each instance in our dataset contains four things:

1. Binary vector - indicating if a superpixel is blacked out or not. The size of the vector is the number of superpixels in the original image.
2. Corresponding image
3. Distance between the original image and itself – We need this distance to give the closest points more power over the farther ones.
4. Prediction for this instance from **AlexNet**

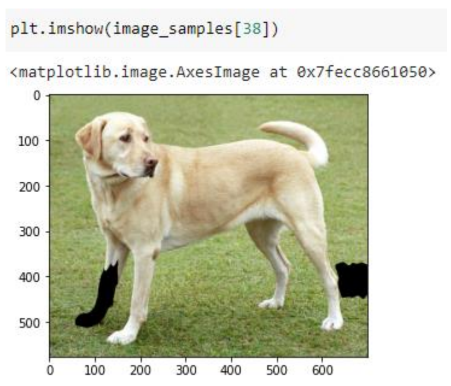
Here is an example of instance 38:



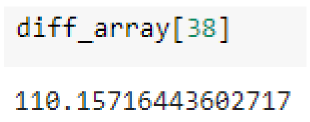
```
[31] binary_samples[38]

     array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
            0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1], dtype=uint8)
```



```
predictions[38]

array([1, 0, 0], dtype=uint8)
```



```
plt.imshow(image_samples[38])

<matplotlib.image.AxesImage at 0x7fecc8661050>
```

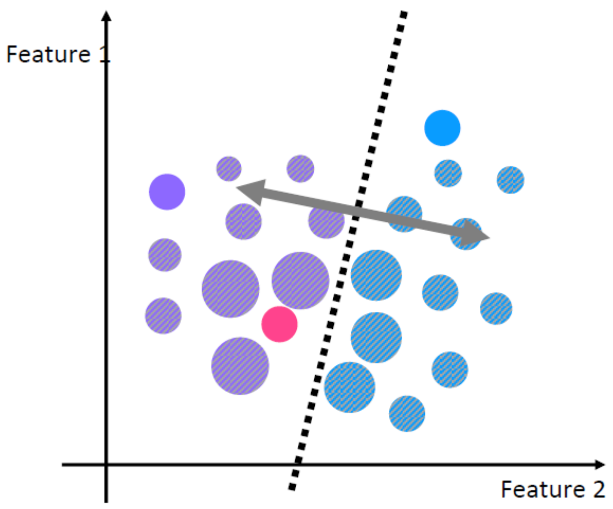


```
diff_array[38]

110.15716443602717
```

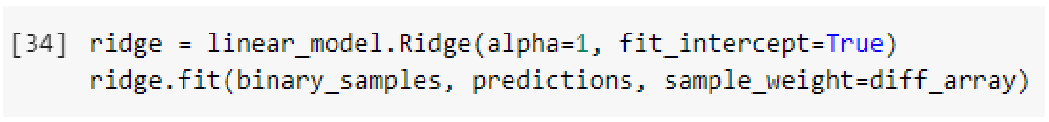
We can see that there are two blacked-out superpixels, the distance is 110, and the prediction is Labrador retriever (The zeros are the Golden and Saluki). I implemented it such that each instance has a random chance to blackout a random number of superpixels.

What now?

Unit now, we gave AlexNet an input and got its prediction that we want to explain. We simplified the original image and generated a new perturbed dataset based on that simplified image. Recall that we can't explain AlexNet? We'll use a simpler model such as Lasso, Ridge, decision trees, or whatever model that we know how to explain, and base our explanation on it. We'll take the dataset that we generated as input to train the simpler model (The bigger the point, the closest it is to the original image).



In my case, I used Ridge as my explainer:

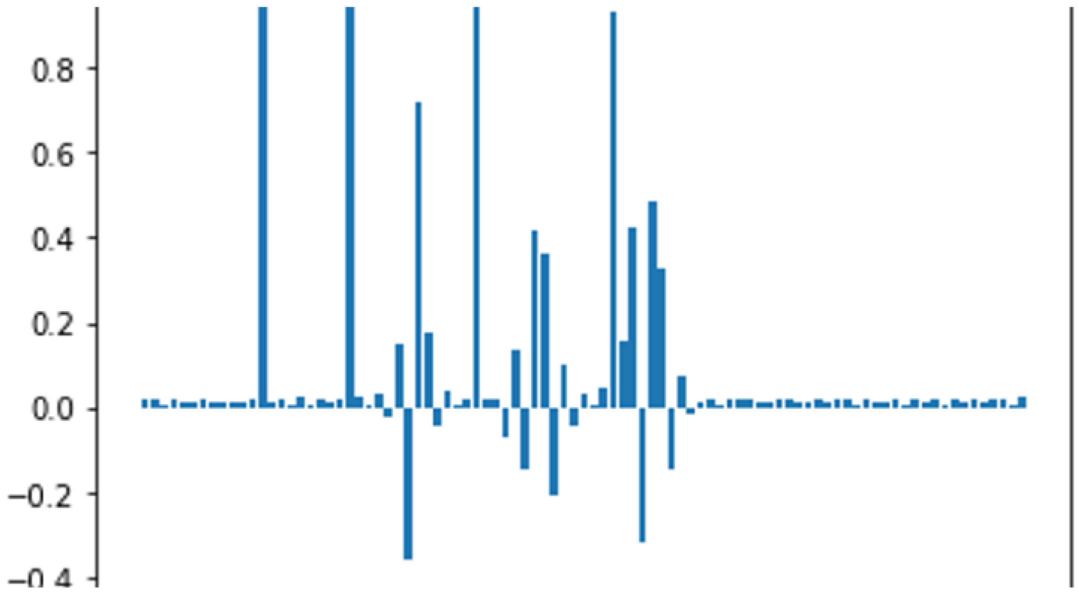


```
[34] ridge = linear_model.Ridge(alpha=1, fit_intercept=True)
     ridge.fit(binary_samples, predictions, sample_weight=diff_array)
```

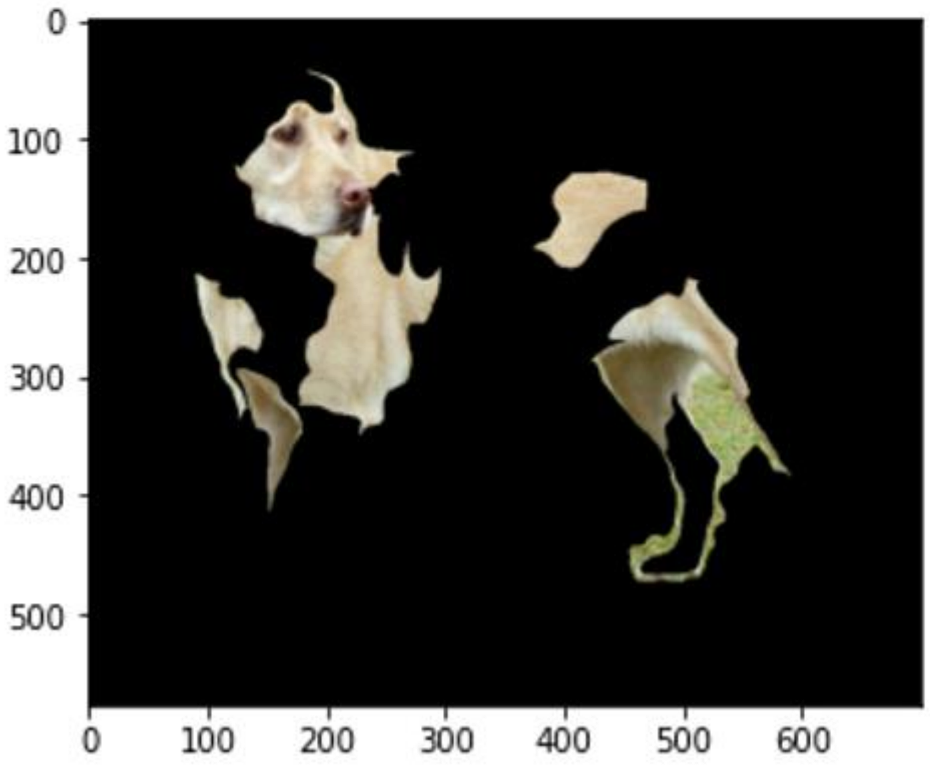
I passed the binary vector as an input, the predictions as the labels, and the distance as the weight in order to give the closest points more power on the model.

Using Ridge for the explanation

Now that we trained Ridge on the newly generated dataset, all we need to do is use its built-in explanation. Let's plot the coefficients:



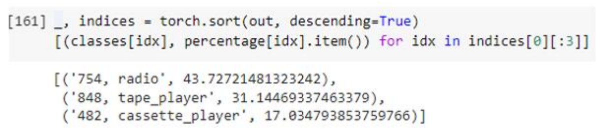
Each bar is a coefficient representing a superpixel inside the original image, the higher the bar is, the more important feature. We can see that coefficients are close to zero and not exactly zero, meaning we used L2 regularization and Ridge. Lastly, we'll take all the features that are above some threshold (I used 0.2) and plot them on the original image:



It's nice that the model thinks that our image is a Labrador because of these regions. If we would give a person to mark the important regions on the image, I think that we'd get similar results, as the face of the dog and parts of his body are the most important regions to recognize the kind of the dog.

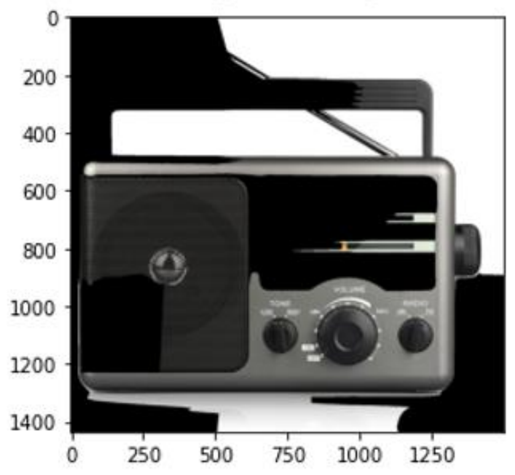
More examples

Here are the results of two more examples. One is a radio, and the other is a laptop with a hamster as wallpaper to try to trick it.



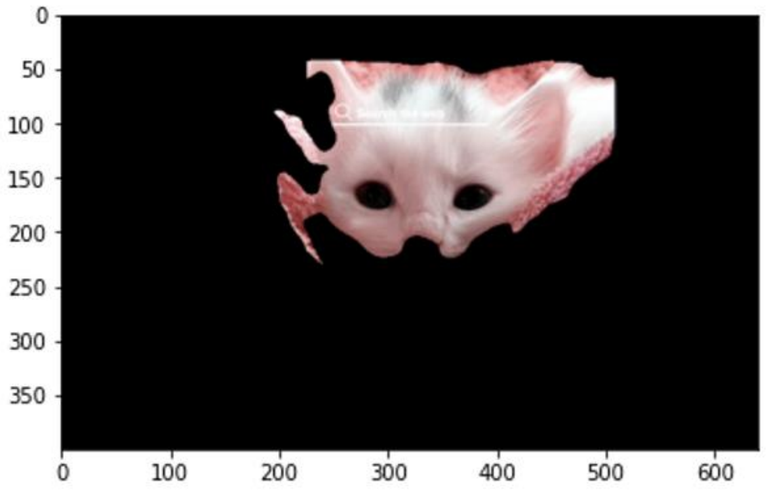
```
[161] _, indices = torch.sort(out, descending=True)
      [(classes[idx], percentage[idx].item()) for idx in indices[0][:3]]

      [('754, radio', 43.72721481323242),
       ('848, tape_player', 31.14469337463379),
       ('482, cassette_player', 17.034793853759766)]
```

```
<matplotlib.image.AxesImage at 0x7f8de3b55550>
```



We can see that the model thinks the most important regions are the speaker, buttons, and some of the background, which can indicate that our model is overfitted for this shape of a radio. Now for the last example:

```
[('333, hamster', 38.5870475769043),
 ('341, hog', 7.539167881011963),
 ('620, laptop', 6.195697784423828)]
```



The model recognized both the laptop and the hamster, but the model thinks that it's more of a hamster than a laptop. We can see why based on the regions from our explanation.

Conclusions

- We saw why we need XAI.
- We used LIME, to better understand the model's prediction. We took the original input, simplified it, generated a perturbed dataset, and used a simpler model that we know how to explain to explain the base model.
- With the explanation, we understood what features the model based its prediction on, and even we noticed that the model is a bit overfitted on some of the labels.