Assignment: 2 **Date: 15/03/24**
Reg. No.: 21BAI1533 **Subject: XAI**
Name: Zeel Mehta

XAI: DA2 Paper Review + Implementation

---

1. "Why Should I Trust You?" Explaining the Predictions of Any Classifier

Ribeiro, M.T.; Singh, S.; Guestrin, C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 1135–1144.

The paper chosen focuses on why and how should one trust an ML model on the basis of the output produced from the several inputs provided.

They have proposed a LIME (Local Interpretable Model-agnostic Explanations) technique which explains predictions of any classifier in interpretable and trustful manner but learning the model around local predictions. They have also included a method to explain the models by presenting individual predictions in non redundant way along with their explanations in their proposal.

These techniques can be tested on text and image datasets, using **random forests for textual data** and **neural networks for image data**.

After testing on the datasets in a stimulated environment we can also test it with human subjects to prove the credibility of the technique.

Since it is a model agnostic technique which can be used with any kind of classification model like text or image classification.

Dataset source: Kaggle

Post training the model using suitable Machine Learning models, the LIME technology can be implemented in python by installing the library **"pip install lime"**.

The lime library has modules which help in explaining the model like
lime_image.LimeImageExplainer() for image classification and
lime_image.LimeTextExplainer() for text classification.

This is then repeated for multiple samples to gain insights on model's decision making process and give better explanations.

Implementation:

Dataset link: https://www.kaggle.com/datasets/hrishitbhattacharya/plastic-data-one

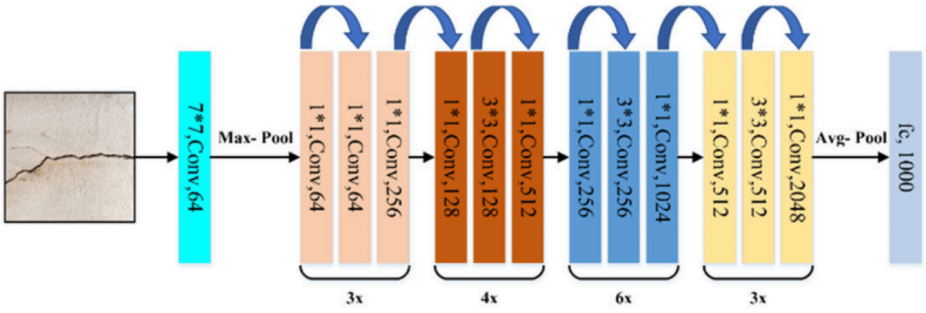
Notebook link: https://www.kaggle.com/code/zeelmehta04/21bai1533-xai-resnet-lime

Black box model used: RESNET50

ResNet-50 is a type of convolutional neural network (CNN) that has revolutionised the way we approach deep learning. **ResNet stands for residual network**, which refers to the **residual blocks** that make up the architecture of the network.

The **skip connections** used in this network allow the preservation of information from earlier layers, which help the network learn better representations of the input data. With the ResNet architecture, the makers were able to train networks with as many as 152 layers. The results of ResNet were groundbreaking, achieving a 3.57% error rate on the ImageNet dataset.

ResNet-50 consists of 50 layers that are divided into 5 blocks, each containing a set of residual blocks. The residual blocks allow for the preservation of information from earlier layers, which helps the network to learn better representations of the input data.



Source: ResearchGate

Architecture:

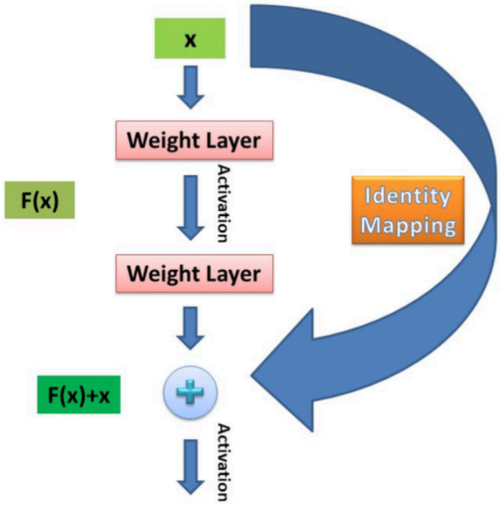
1. **Convolutional Layers**: The first layer of the network is a convolutional layer that performs convolution on the input image. This is followed by a max-pooling layer that downsamples the output of the convolutional layer. The output of the max-pooling layer is then passed through a series of residual blocks.

2. **Residual Blocks**: Each residual block consists of two convolutional layers, each followed by a batch normalisation layer and ReLU activation function. The output of the second convolutional layer is then added to the input of the residual block, which is then passed through another ReLU activation function. The output of the residual block is then passed on to the next block.

3. **Fully Connected Layer:** The final layer of the network is a fully connected layer that takes the output of the last residual block and maps it to the output classes. The number of neurons in the fully connected layer is equal to the number of output classes.

Skip Connections:

Skip connections, also known as identity connections, are a key feature of ResNet-50. They allow the preservation of information from earlier layers, which helps the network to learn better representations of the input data. Skip connections are implemented by adding the output of an earlier layer to the output of a later layer.



Source: ResearchGate

Implementation on my dataset (code blocks):

Dataset details: 6801 files belonging to 3 classes HDPE, PET, PVC.

```
[4]: dataset = tf.keras.preprocessing.image_dataset_from_directory(
         "/kaggle/input/plastic-data-one/PLASTICS_DATA",
         shuffle = True,
         image_size = (IMAGE_SIZE, IMAGE_SIZE),
         batch_size = BATCH_SIZE
     )
Found 6801 files belonging to 3 classes.
```

```
[5]: class_names = dataset.class_names
     class_names
[5]: ['HDPE', 'PET', 'PVC']
```

```
[6]: len(dataset)
[6]: 341
```

Model input specifications:

```
IMAGE_SIZE = 512
BATCH_SIZE = 20
CHANNELS = 3
EPOCHS = 10
```

+ Code + Markdown

Model summary:

[14]:
```
resnet_model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	?	23,587,712
flatten (Flatten)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

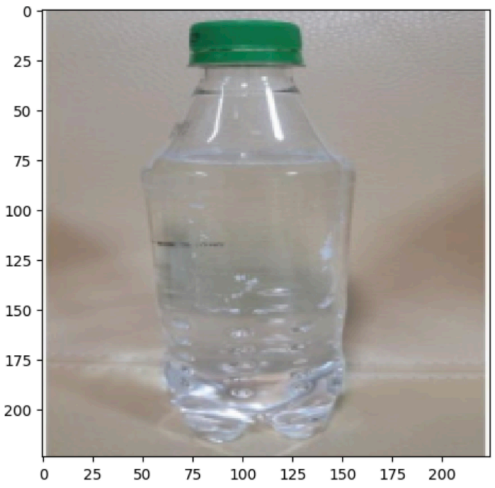
Total params: 23,587,712 (89.98 MB)
Trainable params: 0 (0.00 B)
Non-trainable params: 23,587,712 (89.98 MB)

Training dataset accuracy was found to be around 98% with a loss of 3.02%

```
scores = resnet_model.evaluate(test_ds)
scores
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 30s 127ms/step - accuracy: 0.9876 - loss: 0.0268
[0.03026057966053486, 0.989130437374115]
```

Then it was tested on a random image from a completely different dataset and it correctly predicted with 99% confidence level.



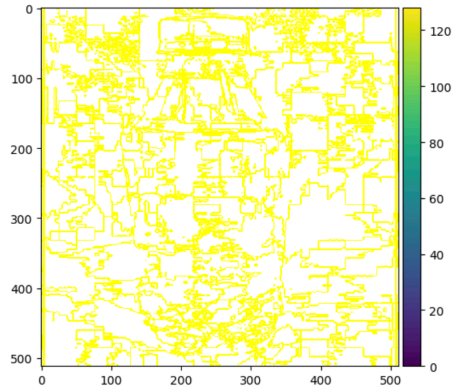
[26]:
```
unknown_img_path = "/kaggle/input/plastic-data-one/PLASTICS_DATA/PET/-141_jpg.rf.77ef1e83e0b6effabde503419854f9bd.jpg"
unknown_img = tf.keras.preprocessing.image.load_img(unknown_img_path, target_size=(512, 512))

predicted_class, confidence = predict(resnet_model, unknown_img)
print("Predicted Class:", predicted_class)
print("Confidence:", confidence, "%")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predicted Class: PET
Confidence: 94.7 %
```

```
[45]: skimage.io.imshow(skimage.segmentation.mark_boundaries(unknown_img_array/2+0.5, superpixels))
```

```
[45…] <matplotlib.image.AxesImage at 0x7854c7426920>
```



Lime Implementation:

Machine learning model predictions can be explained using a technique called **LIME (Local Interpretable Model-agnostic Explanations)**. By approximating the behaviour of the underlying model around a specific instance of interest, it offers local interpretability.

How LIME works with images:

Segmentation: To begin, LIME splits the image into manageable chunks known as superpixels. Superpixels are created by combining pixels with comparable textures and/or colours. The objective is to break the image up into segments that simplify its complexity while capturing important aspects of it.

Perturbation: Randomly turning on and off various superpixels causes the input image to be perturbed for every explanation that LIME generates. LIME produces a set of perturbed images as a result, each of which is a slightly altered copy of the original image with certain superpixels hidden.

Prediction: Following this, LIME runs these disturbed photos through the model and tracks how the changes affect the model's predictions. LIME determines the significance of various superpixels in influencing the model's choice by comparing the predictions of the perturbed images with those

```
[44]: superpixels = skimage.segmentation.quickshift(unknown_img_array, kernel_size=4,max_dist=200, ratio=0.2) #
      num_superpixels = np.unique(superpixels).shape[0]
      print("The number of super pixels generated")
      num_superpixels
```

```
The number of super pixels generated
[44…] 293
```

of the original image.

Weighting: Each superpixel is given a weight by LIME according to how much it contributes to the model's prediction. Superpixels that considerably alter the model's prediction are given larger weights, a sign of their significance in elucidating the behaviour of the model.

Local Model Fitting: Last but not least, LIME uses the altered images and the associated predictions to build a straightforward, understandable model(eg, Linear Regression). By approximating the underlying model's decision boundary around the instance of interest, this local model explains prediction.

Outputs on my model:

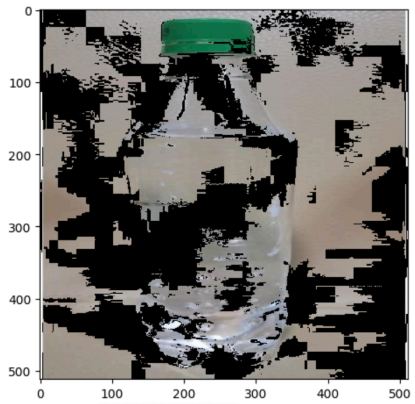
Superpixels segmentation: The code uses skimage's quickshift to segment an image (unknown_img_array) into superpixels based on colour and spatial proximity, with parameters for kernel size, max distance, and ratio. It then calculates the number of unique superpixels and prints the count.

The generated superpixels are shown in the image below using mark_boundaries.

Perturbations:
The size corresponds to the (no. of perturbations x no. of superpixels) in the image.

```
print("The perturbed image")
skimage.io.imshow(perturb_image(Xi/2+0.5,perturbations[0],superpixels))
```

```
The perturbed image
<matplotlib.image.AxesImage at 0x7854c7f888b0>
```



Weighting:

1. Distance between original image and perturbed images
2. Compute weights (importance) of each perturbed image using kernel. The distances are then mapped to a value between zero and one (weight).

```
[54]:
kernel_width = 0.25
weights = np.sqrt(np.exp(-(distances**2)/kernel_width**2)) #Kernel function
weights.shape
```

```
[54…  (150,)
```

Local Model Fitting: Linear Regression.

Extracting top predictions by fitting Linear Regression model to top predictions gained from the black box model i.e. using output of black box model to explain and bring out the output of the locally interpretable model.

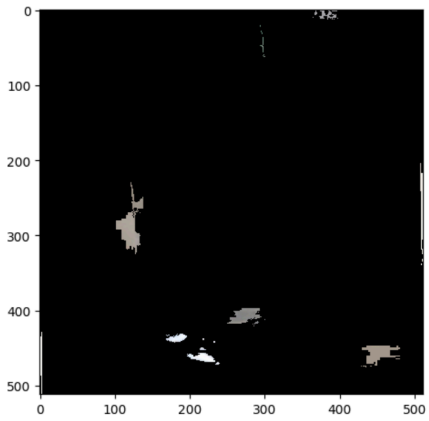
[62]:
```
simpler_model = LinearRegression()
simpler_model.fit(X=perturbations, y=predictions[:,:,top_pred_classes[0]], sample_weight=weights)
coeff = simpler_model.coef_[0]
coeff.shape
```

[62… (293,)

Computing top features extracted by LIME of Linear Regression model that it found to be the most important in predicting the class of the image. The non important features are masked as black and the important ones are displayed by LIME.

:
```
mask = np.zeros(num_superpixels)
mask[top_features]= True
skimage.io.imshow(perturb_image(Xi/2+0.5,mask,superpixels) )
```

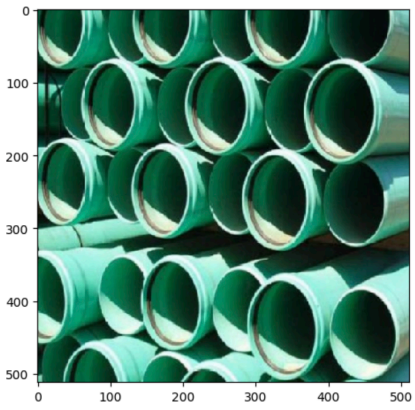
<matplotlib.image.AxesImage at 0x7854c9f70b50>



Running the same model on other classes also have given the correct predictions:

1. PVC:

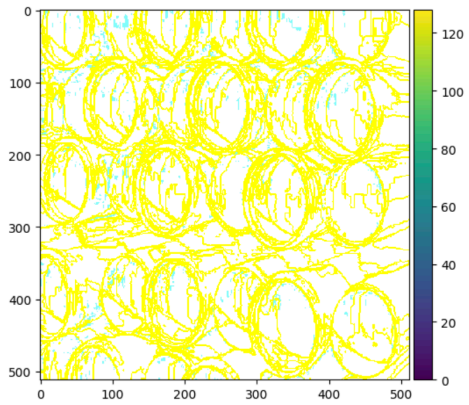
[75]:
```
Xi = skimage.io.imread("/kaggle/input/plastic-data-one/PLASTICS_DATA/PVC/Blue-Planet-Recycling-Plastics-Recycling-Plastic-Poly-Pipe-HDPE-and-PVC-Pipe_jpg.rf.78c4619
Xi = skimage.transform.resize(Xi, (512,512,3))
skimage.io.imshow(Xi)
Xi = (Xi - 0.5)*2
```



[78]:
```
skimage.io.imshow(skimage.segmentation.mark_boundaries(unknown_img_array/2+0.5, superpixels))
```

```
/opt/conda/lib/python3.10/site-packages/skimage/io/_plugins/matplotlib_plugin.py:149: UserWarning: Float image out of standard range; displaying image with stretched contrast.
  lo, hi, cmap = _get_display_range(image)
```

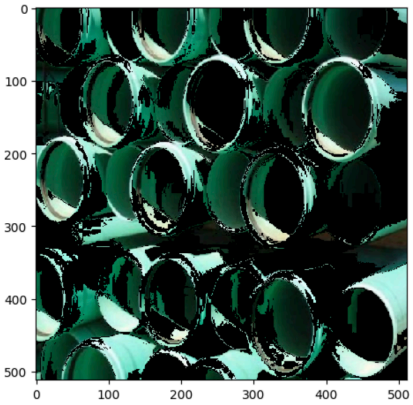
[78…] <matplotlib.image.AxesImage at 0x7854c9e3afb0>



[82]:
```
print("The perturbed image")
skimage.io.imshow(perturb_image(Xi/2+0.5,perturbations[0],superpixels))
```

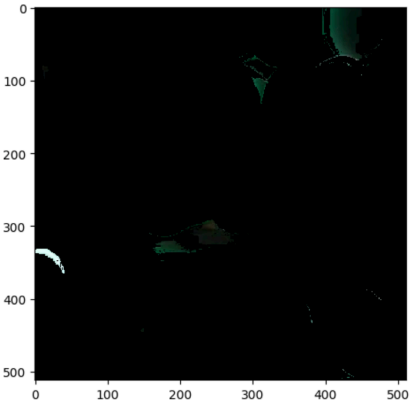
```
The perturbed image
```

[82…] <matplotlib.image.AxesImage at 0x7854c9e9bbb0>



[97]:
```
mask = np.zeros(num_superpixels)
mask[top_features]= True
skimage.io.imshow(perturb_image(Xi/2+0.5,mask,superpixels) )
```

[97…] <matplotlib.image.AxesImage at 0x7854c9d3eb30>



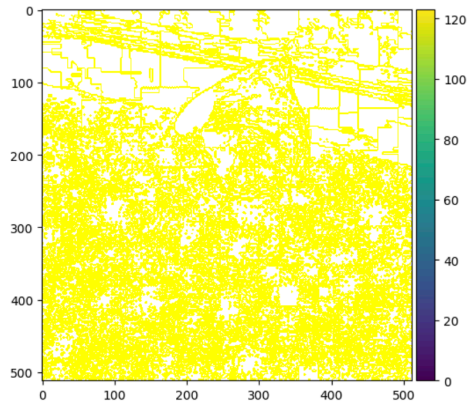
2. HDPE:

[105]:
```
Xi = skimage.io.imread("/kaggle/input/disposable-bags-classification-using-yolov5/yolov5/images/test/00001120.jpg")
Xi = skimage.transform.resize(Xi, (512,512,3))
skimage.io.imshow(Xi)
Xi = (Xi - 0.5)*2
```

[108]:
```
skimage.io.imshow(skimage.segmentation.mark_boundaries(unknown_img_array/2+0.5, superpixels))
```

```
/opt/conda/lib/python3.10/site-packages/skimage/io/_plugins/matplotlib_plugin.py:149: UserWarning: Float image out of standard range; displaying image with stretched contrast.
  lo, hi, cmap = _get_display_range(image)
```

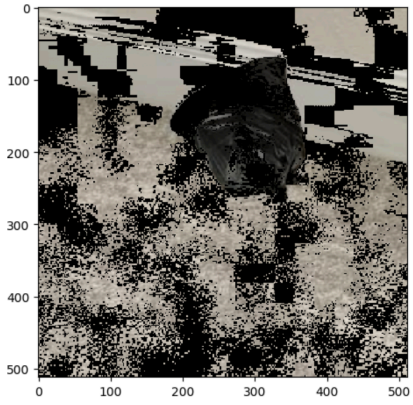
[10…] <matplotlib.image.AxesImage at 0x7854c9db9930>



[112]:
```
print("The perturbed image")
skimage.io.imshow(perturb_image(Xi/2+0.5,perturbations[0],superpixels))
```

```
The perturbed image
```

[11…] <matplotlib.image.AxesImage at 0x7854c9deda50>



[195]:
```
mask = np.zeros(num_superpixels)
mask[top_features]= True
skimage.io.imshow(perturb_image(Xi/2+0.5,mask,superpixels) )
```

[19…] <matplotlib.image.AxesImage at 0x7859e6a1ad40>