Indian Institute of Information Technology Allahabad
B.tech sem-5

Image and Video Processing Project

Explainable Machine Learning for Image Processing
Domain: Breast Cancer

Group-12

Group members:

Shivansh Gupta	IIT2019107
Rahul Kumar	IIT2019109
Aditya Singh	IIT2019111
Sarthak	IIT2019114
Saloni	IIT2019128
Sanyam	IIT2019129
Khushi Gupta	IIT2019141

Course Instructor :

Professor Anupam Agarwal

GC Jana

***Abstract*— In this paper we have done a thorough analysis on explainable machine learning for image processing and implemented Breast Cancer Analysis.**

1. Introduction

Breast cancer is the most common type of cancer that affects females all across the world. It is distinguished by an overgrowth of a malignant tumor in the breast. The goal of breast cancer screening is to achieve an early diagnosis, which aims to discern the Malignant and Benign tumor, and the prognosis helps to put up a treatment plan. Medical image processing and machine learning for breast cancer diagnosis, prognosis and/or treatment is promising since it can help physicians, doctors and experts in detecting abnormalities quite efficiently.

Cancer is the deadliest disease in the world. Every year thousands of people die from cancer. Early stage detection of disease can reduce the risk of human lives.Because cancer cannot be diagnosed in its early stage by a physician. Sometimes cancer disease symptoms cannot be seen from outside of the body. At the second or third stage, cancer cannot be cured. Some latest techniques can cure the disease but those techniques also have some side effects. So the early stage detection of cancer can help reduce risk to patient life.

The cells that are uncontrolled and grow rapidly are given the name of cancer. Breast cancer occurs due to growth of cells in the breast. Group of extra tissues is known as a tumor. The second name of a tumor is cancer.

In this paper, we have compared the accuracy of different ML models that are used for breast cancer diagnosis after classifying whether the tumor is benign or malignant.

2. EXPLAINABLE MACHINE LEARNING

Explainable machine learning is a term that refers to approaches for deciphering the predictions of a pre-trained model. Various post-hoc approaches exist, including reason code creation, local and global visualisations of model predictions, and so on. In this paper, we employed the reason code generation as a post hoc approach. The Local Interpretable Model-agnostic Explanations (LIME) are used in particular to describe the features a machine learning model uses to generate a prediction choice.

2. Preparing Breast Cancer Histology Images Dataset

There are 5,547 50x50 pixel RGB digital photos of H&E-stained breast histopathology samples in the dataset. IDC or non-IDC pictures are labelled on these photographs. There are 2,788 IDC and 2,759 non-IDC photos in this collection. These photos have previously been converted to Numpy arrays and saved in the X.npy file. Similarly, the relevant labels are recorded in Numpy array format in the file Y.npy.

Dataset link:
https://www.kaggle.com/paultimothymooney/breast-histopathology-images

3.1 Loading Data

After downloading the X.npy and Y.npy files to a local computer, load them into memory as Numpy arrays.

Two data samples are shown here, with the picture on the left labelled as 0 (non-IDC) and the image on the right labelled as 1. (IDC).

3.2 Shuffling Data

All data samples labelled as 0 (non-IDC) are placed before data samples labelled as 1 in the original dataset files (IDC). The dataset is randomised at random to avoid spurious data patterns:

```
import random
temp =
list(zip(X,Y))r
andom.shuffle(t
emp)

X,Y =zip(*temp)
X = np.array(X)
Y = np.array(Y)
```

3.3 Transforming Dataset

A conventional deep learning model works best when the value of input data is in the range of [0, 1] or [-1, 1], but an IDC picture has pixel values in the range of [0, 255].

```
class Scale(BaseEstimator,
TransformerMixin):
   def__init__(self):
       pass

   def fit(self, X, y):
       return self

   def transform(self, X):
       X1 = X.copy()
       X1 = X1 / 255.0
       return X1
```

3.4 Dividing Dataset for Model Training and Testing

The dataset is split into three sections: 80% for model training and validation, and 20% for model testing.

```
X_train_raw, X_test_raw, y_train_raw, y_test_raw =
train_test_split(X, Y, test_size=0.2)
X_train = X_train_raw.copy() X_val
= X_train[:1000]

X_train, X_test, y_train,
y_test = train_test_split(X,
Y, test_size=0.2)


# Normalize pixel values to
be between 0 and 1
X_train = X_train
X_test = X_test

X_train.shape,X_test.shape
```

3. Training 2D ConvNet Model

The BCHI dataset is made up of images, a 2D ConvNet model was chosen for IDC prediction and later this model was further merged with an ANN model to verify the results.

3.1 Create the convolutional base

Similarly , the function *getKerasCNNModel*() below creates a 2D ConvNet for the IDC image classification.

```
model =
models.Sequential()
model.add(layers.Conv2D(32
, (3, 3),
activation='relu',
input_shape=(50, 50, 3)))
model.add(layers.MaxPoolin
g2D((2, 2)))
model.add(layers.Conv2D(64
, (3, 3),
activation='relu'))
model.add(layers.MaxPoolin
g2D((2, 2)))
model.add(layers.Conv2D(64
, (3, 3),
activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64,
activation='relu'))
model.add(layers.Dense(2))
```

3.2 Compile and train model with CNN

```
model.compile(optimizer='adam',loss=tf.keras.losses.
SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_test,
y_test),batch_size=250)
```

3.3 Evaluate model (CNN)

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label =
'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(X_test,  y_test,
verbose=2)
```

4. Explaining Model Prediction Results

As described before, We use LIME to explain the model prediction results in this article.

4.1 ANN model creation

```
model.add(tf.keras.layers.Dense( units =300,
activation = 'relu', input_shape = (7500,)))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(units = 2, activation
= 'softmax'))
model.compile(optimizer = 'adam', loss =
'sparse_categorical_crossentropy',  metrics =
['accuracy'])

model.summary()
```

4.2 Compile and train model with ANN

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label =
'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

history = model.fit(X_train,
y_train,validation_data=(X_test, y_test), epochs = 10
,batch_size=250)
```

4.3 Evaluate model (CNN+ANN)

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label =
'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

history = model.fit(X_train,
y_train,validation_data=(X_test, y_test), epochs = 10
,batch_size=250)
```

We make a pipeline to wrap the ConvNet model for the integration with LIME API.

```
from sklearn.pipeline import
Pipeline simple_cnn_pipeline =
Pipeline([
   ('scale', Scale()),
   ('CNN', KerasCNN(X_val=X_val,
y_val=y_val))
   ])
```

4.4 Selecting LIME Explainer

Local surrogate models are interpretable models that are used to explain individual predictions of black box machine learning models. (Link) Surrogate models are trained to approximate the predictions of the underlying black box model.

Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions.

Mathematically, local surrogate models with interpretability constraint can be expressed as follows:

$$\textbf{explanation}(x) = \textbf{argmin}_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

where:

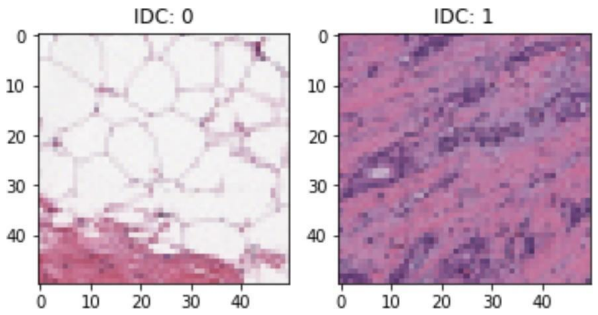
- explanation model for instance x is the model g
- loss L
- original model f
- model complexity Ω(g) and
- proximity measure π_x

For various forms of datasets, such as picture, text, tabular data, and so on, the LIME technique supports many types of machine learning model explainers. Because the dataset in this study is made up of photos, the LIME image explanation was chosen.Quickshift, a 2D image segmentation algorithm, is utilised to create LIME super pixels (i.e., segments)

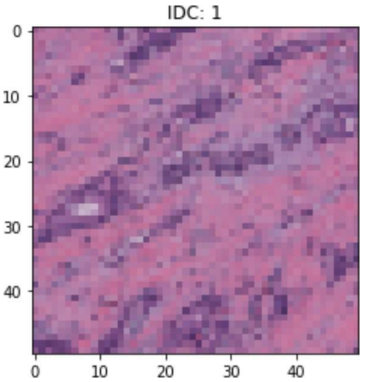
```
from lime import lime_image
from lime.wrappers.scikit_image import
SegmentationAlgorithm

explainer = lime_image.LimeImageExplainer()

segmenter = SegmentationAlgorithm('quickshift',
kernel_size=1, max_dist=200, ratio=0.2)
```

4.2 Explaining Model Prediction

After the ConvNet model has been trained, the LIME image explainer's explain instance() method may be used to provide an explanation of the model prediction given a fresh IDC picture.
A template image and a related mask image make up an image prediction explanation. These graphics can be used in a variety of ways to describe a ConvNet model prediction result.



Explanation 1: Prediction of Positive IDC (IDC: 1)



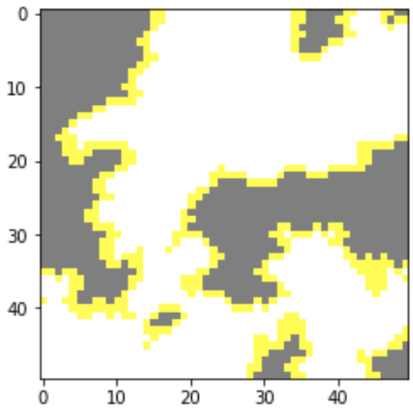
This figure shows a positive IDC image for explaining model prediction via LIME.
The code below is to generate an explanation object explanation_1 of the model prediction for the image IDC_1_sample (IDC: 1) in above figure..

In this explanation, white color is used to indicate the portion of the image that supports the model prediction (IDC: 1).

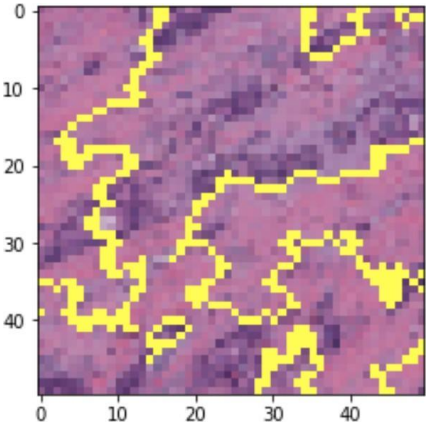
```
explanation_1 =
explainer.explain_instance(IDC_1_samp
le_test,classifier_fn =
model.predict,top_labels=2,hide_color
=0, # 0 - gray
num_samples=10000,
segmentation_fn=segmenter
)
```

Once the model prediction's explanation has been received, the procedure get image and mask() may be used to get the template image and the associated mask image (super pixels):

```
from skimage.segmentation import
mark_boundaries

temp, mask =
explanation_1.get_image_and_mask(explanat
ion_1.top_labels[0],

positive_only=True,

num_features=20,

hide_rest=True)

plt.imshow(mark_boundaries(temp / 2 +
0.5, mask))
```



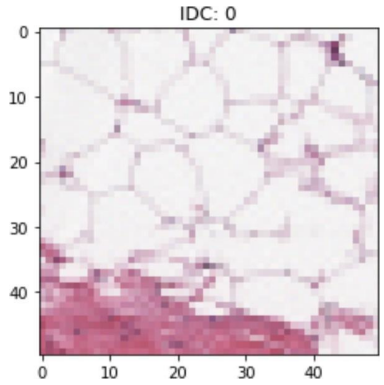
Above figure shows the hidden portion of the given IDC image in gray color. The area of the supplied IDC image that confirms the model prediction of positive IDC is indicated by the white section of the image.



The code below shows the yellow boundary of the IDC picture area that validates the model forecast of positive IDC (see above figure).

```
temp, mask =
explanation_1.get_image_and_mask(expla
nation_1.top_labels[0],

positive_only=True,

num_features=20,

hide_rest=False)

print('explanation 1 top labels[0] =
', explanation_1.top_labels[0])
plt.imshow(mark_boundaries(temp,
mask))
```

Explanation 2: Prediction of non-IDC



(IDC: 0)

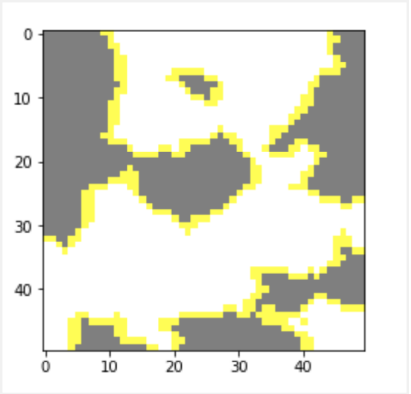
Above figure shows a non-IDC image for explaining model prediction via LIME.

For the picture IDC 0 sample in above figure, the code below generates an explanation object explanation 2 of the model prediction. White colour is used to represent the area of the picture that confirms the model prediction of non-IDC in this explanation.

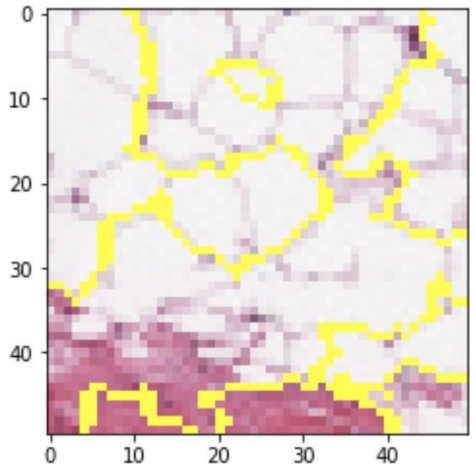
```
explanation_2 =
explainer.explain_instance(IDC_0_sample_
test,

classifier_fn = model.predict,

top_labels=2,

hide_color=0,

num_samples=10000,

segmentation_fn=segmenter

)
```

Once the model prediction's explanation has been received, the procedure get image and mask() may be used to get the template image and the associated mask image (super pixels):

```
from skimage.segmentation import
mark_boundaries

temp, mask =
explanation_2.get_image_and_mask(explana
tion_2.top_labels[0],

positive_only=True,

num_features=20,

hide_rest=True)

print('explanation 2 top labels[0] = ',
explanation_2.top_labels[0])
plt.imshow(mark_boundaries(temp / 2 +
0.5, mask))
```



Above figure depicts the non-IDC image's concealed region in grey. The area of the supplied non-IDC image that supports the model prediction of non-IDC is indicated by the white section of the image.



The code below shows the yellow boundary of the IDC picture area that supports the model prediction of non-IDC (see above figure).

```
temp, mask =
explanation_2.get_image_and_mask(expla
nation_2.top_labels[0],

positive_only=True,

num_features=20,

hide_rest=False)
print('explanation 2 top labels[0] =
', explanation_2.top_labels[0])
plt.imshow(mark_boundaries(temp,
mask))
```

5. Conclusion

We demonstrated how to utilise the LIME image explainer to explain the IDC image prediction results of a 2D ConvNet model in IDC breast cancer detection using the Kaggle BCHI dataset.

Setting the number of super-pixels/features to 20 gave explanations for both IDC and non-IDC model prediction.

We also observed that the amount of super pixels/features used affects the explanation outcomes. To obtain an acceptable model prediction explanation, domain expertise is necessary to change this parameter. For a decent outcome, the quality of the input data (in this example, pictures) is also critical. By adding additional samples, accuracy may be increased.

6. References

[1] Explainable Machine Learning for Breast Cancer Diagnosis

[2] Explainable AI: A Review of Machine Learning Interpretability Methods

[3] Explainable deep learning models in medical image analysis

[4] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?" Explaining the Predictions of Any Classifier

[5] Interpretable Machine Learning, A Guide for Making Black Box Models Explainable

[6] Predicting IDC in Breast Cancer Histology

Images

[7] EXPLAINABLE EXPLAINABLE

INTELLIGENCE

[8] Explainable Deep Learning Models in Medical

Image Analysis

[9] Generalizable and Explainable Deep Learning in

Medical Imaging with Small Data

[10] Machine learning

[11] Breast cancer