ECE1512 Digital Image Processing and Application - Winter 2021
Project A: Visual Interpretation of Convolutional Neural Network

Zhe Su (1002113116)
Dongqi Huang (1007222989)

Handed by: Zhe Su
Handed at: February 28, 2021
Due: March 1, 2021

Part 1. 1-D Digit Classification

Task 1. 1-Dimensional digit classification

1.1 Performance evaluation

The model performance will be evaluated by multiple metrics, the detailed results are shown as following:

Overall accuracy

Overall accuracy on test set is 0.877.

```
# Task1.1.a: Overall classification accuracy
model=load_model('models/MNIST1D.h5')
dataset = make_dataset()
x_test = dataset['x_test']
y_test = dataset['y_test']
prediction = np.argmax(model(np.expand_dims(x_test, axis=-1)), axis=1)
print("Overall accuracy: {:.3f}".format(np.mean(prediction == y_test)))
>Overall accuracy: 0.877
```

Listing 1: Overall accuracy measurement on MNIST1-D

Class-wise accuracy for all classes

Accuracy for each class: digit class 0 has the highest accuracy 0.980, and digit class 9 has the lowest accuracy 0.784.

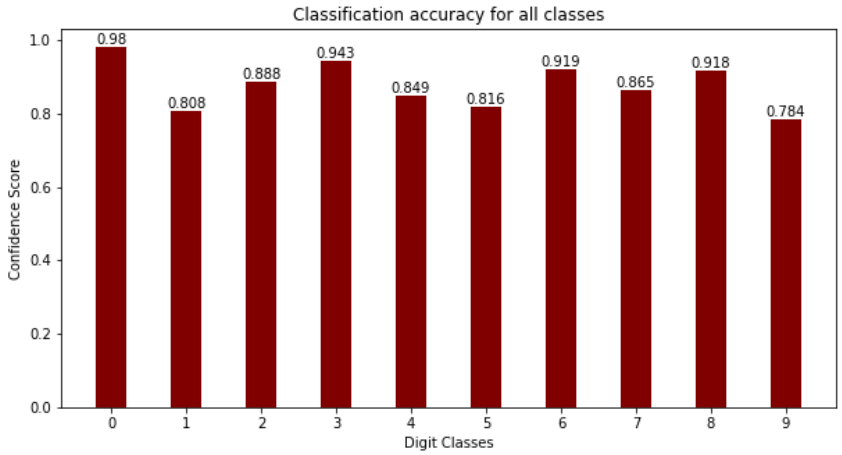
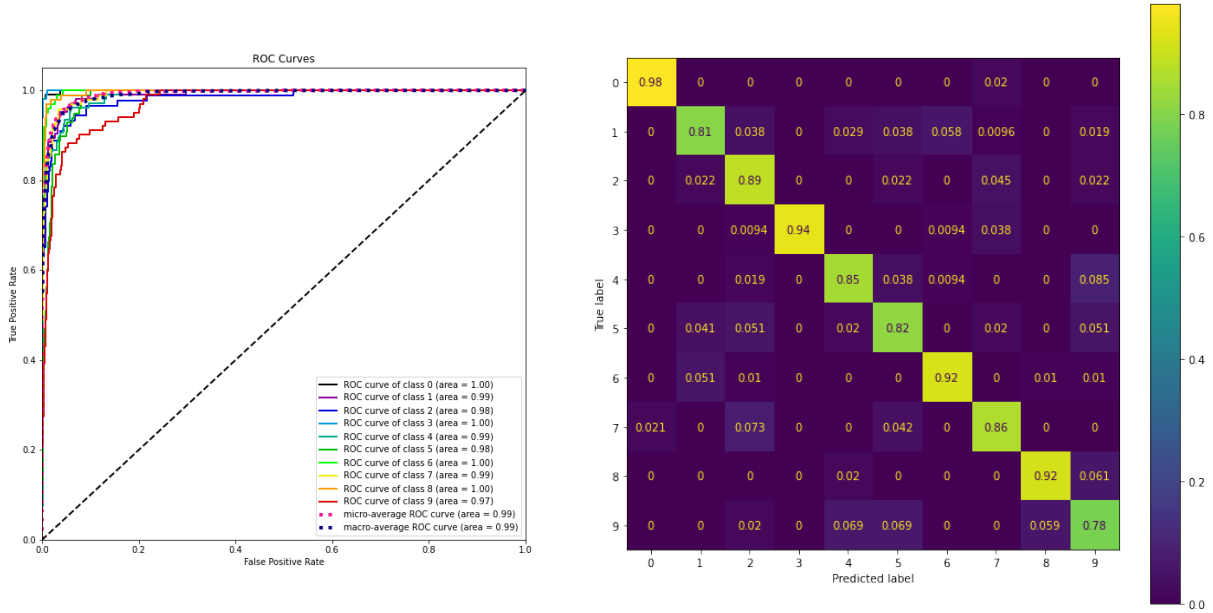


Figure 1: Accuracy for each class on MNIST1-D

ROC and AUC curves

The Receiver Operation Characteristic (ROC) curve and its corresponding area under curve (AUC) is presented by Figure 2a. In general, ROC is applied on two classes classification problem. In our case, however, MNIST1-D has 10 classes. Thus the way we covert it to a two classes classification problem is through the way that for each class we testing, all false predictions (not matter which classes the false prediction belongs



(a) ROC/AUC for each class

(b) Confusion matrix

Figure 2: ROC, AUC and Confusion Matrix of MNIST1-D

to) are all categorised to a same class. Then we are able to measure the false positive rate (FPR) and true positive rate (TPR) in the context of two classes. Open sources plotting kit scikit-plot is used to plot the ROC curve and notate the AUC values. We could observe that there are multiple digit classes achieve maximum value of 1 on AUC, and all digit classes seem to perform well on this metric.

Normalized confusion matrix

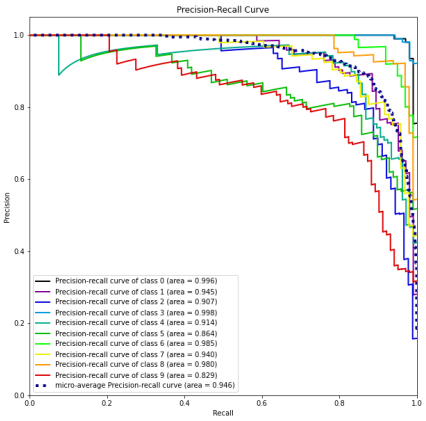
Figure 2b shows a confusion matrix on the test set. The matrix not only displays the true label accuracy, but it also presents a percentage value to show how much the system is confused in each pair of classes. We could observe that digit class 0 still has the highest prediction accuracy, and the only class it has been wrongly categorised into is digit class 7. For digit class 9, it is easy to confuse with digit class 4 and digit class 5.

Precision, Recall, and F1 scores

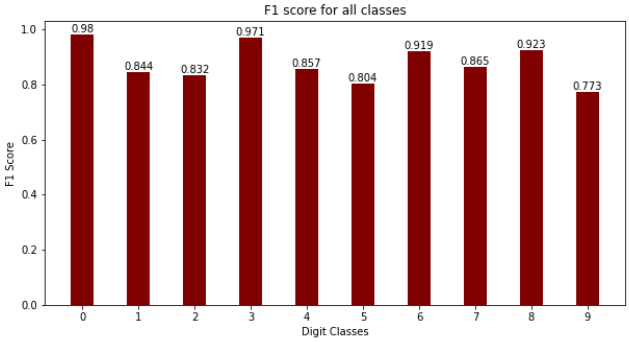
Precision-Recall curve is shown in Figure 3a and F1 score for each class is shown in Figure 3b. The results are similar to the one given by the previous metrics, which digit class 0 still achieves the highest value on both precision-recall and F1 score, whereas digit class 9 has the lowest value on both precision-recall and F1 score.

1.2. Qualitative examples analysis

Based on the result given by the previous metrics, we conclude that digit class 9 and digit class 5 are the two classes that are miss-classified the most. Performance comparison is shown in the Table 1. Specifically, from the confusion matrix, we found that digit class 9 is easily confused to class 4, 5, and 8, and digit



(a) Precision-Recall Curve



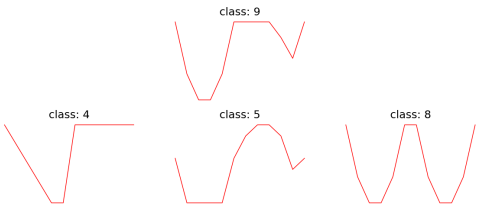
(b) F1 score on each class

Figure 3: Precision-recall curve and F1 score table

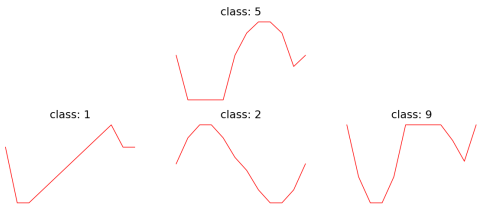
class 5 is easily miss-classified as class 1, 2, and 9. MNIST1-D dataset is generated based on the 1-D digit template. Visualizing the templates of digit class could provide us insights of why class 9 and class 5 has such a performance. Figure 4a and Figure 4b present us a template data comparison on class 9, class 5 and their corresponding confused classes given by the confusion matrix. From the shape of the templates, we could clearly see the similarity among those classes. For example, class 5's template and class 9's template are highly similar, potentially resulting in the model miss-classifying the data samples from those two classes.

Digit Class	Accuracy	ROC AUC	Precision-Recall AUC	F1 Score	Confusion to classes
9	0.784	0.970	0.829	0.773	4, 5, 8
5	0.816	0.980	0.864	0.804	1, 2, 9

Table 1: Performance of the two most miss-classification digit classes



(a) Class 9 and Class 4, 5, 8 templates



(b) Class 5 and Class 1, 2, 9 templates

Figure 4: MNIST1-D templates

Task 2. CNN Interpretation

1.1. Shapley additive explanations(SHAP)

Research gap covered and its novelty

SHAP belongs to additive feature attribution method. In most cases, it returns a local explanation of the average marginal contribution of different features to the final prediction given by the model. In addition, SHAP correlates and unifies some of the previous additive feature attribution methods as well. SHAP builds a linear explanation model for the analyzed instance. Just as the inventors stated in their paper[1] that the novelty of SHAP includes:

(1). SHAP brings game theory results that guarantee a unique solution to the entire class of additive feature attribution methods and propose SHAP values as a unified measure importance that various methods approximate.

(2). A new SHAP value estimation method is introduced that is better aligned with human intuition as measured by user studies and more effectually discriminate among model output classes than the original shapley value calculation.

(3). Give people a new view of unifying 6 existing methods with mathematical deduction.Another approximation method of Shapley values TreeSHAP is proposed by the inventor of SHAP in 2018[3], the original SHAP(2017 version) estimation consists of KernelSHAP only.

Methodologies of SHAP

We could briefly explain the methodologies of SHAP in several steps as shown below:

Step 0: All of the Shapley values could be derived by training a linear model. In other words, a modified LIME model.

Step 1: Select K sets of features.

Step 2: Mapping every one set of features to a data point, Xc. Data point Xc is generated by mixing the instance of interest Xs, and a random data point Xr. Xc derived its selected features from Xs, and the other features from the random instance Xr. Feeding the target model with these K generated Xc as inputs , we get K predictions for the generated data points.

Step 3: The weight factor for each set of features depends on the size of features contained in a combination,and that of all features. The formula is given as below:

$$Weight(z') = \frac{(M-1)}{\binom{z'}{M} z'(M-z')} \tag{1}$$

Step 4: The loss function is the classic mean square error function, but each term of the squared error, the remainder of *f(Xc) - g(Xc)*, is weighted by an aforementioned weight factor.

Pros and Cons

The KernelSHAP runs slow and thus is not a good choice for a model with thousands of features. Since SHAP highly relies on the assumption that different features of a instance are independent, it might not be able to give correct estimation for feature importance when the size of feature is large or complex dependency existing among features(i.e. images). The computation of marginal contribution involves conditional expectation, but the mapping function *h(z')* that takes padding chosen features from random instances would become less trustworthy (it would generate impractical data points). If a image is not divided to

super pixels, SHAP would not perform well since a data point has multiple features and the target model is complex(i.e. image and deep CNN). Generally, SHAP is good at giving an intuitive local explanation for feature importance (both selective and universal). However, the interpretation of the SHAP result might not be that trustworthy since we cannot quantify how much weight are assigned to abnormal data points.

1.2. Semantic Input Sampling for Explanation(SISE)

Research gap covered and its novelty

Previous models designed for visual explainable AI have their own drawbacks. Backpropagation-based methods are not able to generate global explanation. Existing perturbation-based methods are time consuming. The explanation map generated by CAM-based methods lacks of spatial resolution since they only make use of the output of a CNN's last convolution block. The novel method they come up with , SISE[2], is a perturbation based method. However, their solution gives both class discriminability and spatial resolution. Unlike RISE, the mask of a single image is not randomly generated but related to feature maps of multiple layers of the CNN. One section of their solution focuses on minimizing the number of feature maps involved, which significantly accelerates the algorithm.

Methodologies of SISE

Phase 1: Feature map extraction

Collecting feature maps in the last layers (specifically, the input of pooling layers) of all convolutional blocks for a given CNN model.

Phase 2: Feature map selection and attribution mask creation

Gradient of the model's confidence score with respect to every pixel of a feature map is averaged for all feature maps belonging to different layers. Only the feature maps with an average gradient greater than a given threshold (0 by default) are qualified for further processing. The resizing of the chosen feature maps is achieved by applying bilinear interpolation and normalization.

Phase 3: Attribution mask scoring

Similar to RISE, the process of masking is achieved by point-wise multiplication of the original image and a set of aforementioned attribution masks. The score of a particular position of is given by the weighted sum of all model predictions (feeding the target model with masked images in advance). The position specific weight factor is the proportion of the mask intensity of the particular position in the overall intensity of a mask. Apply this to every last layer of convolutional block, we obtain the saliency map for each layer.

Phase 4: Generating layer visualization map

The visualization map of generated from salient maps of two consecutive layers are given as below. Firstly, both of the salient maps are normalized and added together (not weighted). Secondly, the second layer is binarized as a filter using Ostu-based binarization to remove the absent features in the second layer. The point wise multiplication of the sum (derived from step 1) and the filter (derived from step 2) provides the merged visualization map of this two layer's salient maps.

Pros and Cons

Since it is a model specified (for CNN only) methods, SISE is estimated to have a good performance in difficult scenarios like a deep CNN. This method generate a explanation map showing the distribution of feature importance for the original model to output the confidence score in the form of a heat map. For misclassified instance, SISE generate the exact region the target model is focusing on when making the wrong classification. This would help people understanding what goes wrong in the target model.

2.2 XAI on MNIST1-D dataset

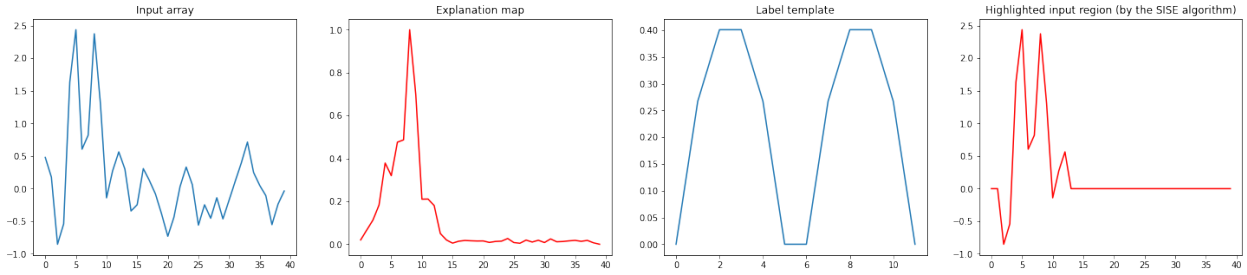
```
# Reference: using provided SISE function
from xai_utils import *
# Randomly taking one data sample from test set
index = int(np.floor(np.random.rand()*1000))
# Input and true label preparation
input = np.expand_dims(np.expand_dims(x_test[index], axis=0), axis=-1)
label = y_test[index]
# Get explanation map from SISE
explanation = SISE(input, model, label, [['conv1d_2']], grad_thr=0.)
print("SISE explanation map dim: {}".format(explanation.shape))


>SISE explanation map dim: (40, 1)
```

Listing 2: Explanation map from SISE

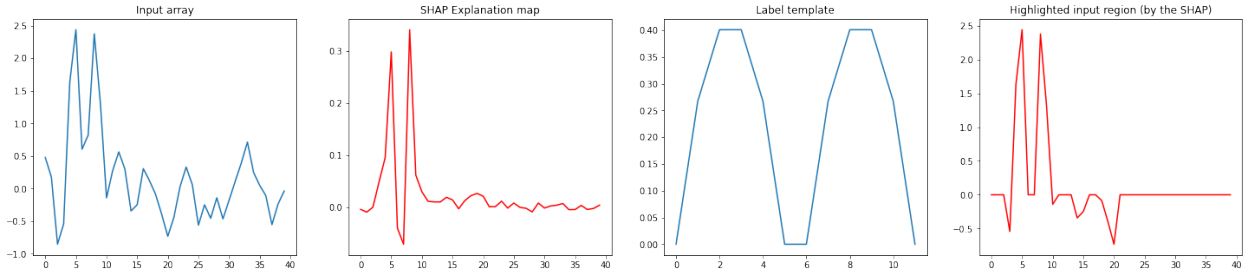
```
import shap
index = int(np.floor(np.random.rand()*1000))
input = np.expand_dims(np.expand_dims(x_test[index], axis=0), axis=-1)
label = y_test[index]
# Get explanation map from SHAP
e = shap.DeepExplainer(model, np.expand_dims(x_test, axis=-1))
elist = e.shap_values(np.expand_dims(np.expand_dims(x_test[index], axis=0), axis=-1))
shap_explaination = np.squeeze(elist[label])
print("SHAP explanation map dim: {}".format(shap_explaination.shape))


>SHAP explanation map dim: (40,)
```

Listing 3: Explanation map from SHAP

We randomly pick two data samples from the test set, their information is presented in Figure 5 and Figure 6. Each figure contains: the original data input, the explanation map given by SISE or SHAP algorithm, the MNIST1-D template data, and the highlighted input filtered by a threshold value. The input of data is generated based on the template data, and this could be clearly observed. For instance, Figure 5 template graph, representing digit class 3, shows a up-side-down W shape in a range of 0 to 10. Figure 5 input array graph also has a highly similar shape from 0 to 10 and roughly similar shapes repeatedly in the other range. The explanation map graph directly shows the weighted value of how importance the model evaluated on certain features. As we could see from explanation maps of SISE and SHAP in Figure 5, they both have high values from range 0 to 10, which could be interpreted as that the model weights the input data from this range significantly in order to make decision. Filtering out less important features data by a threshold given by the explanation map, we could get the highlighted input graphs shown in the right-most column. We could clearly see that the highlighted regions given by both SISE and SHAP show a highly similar shapes corresponding to the digit template. Then we believe that both explanation maps from SISE and SHAP well interpret the model's behaviours, and the decision making from the model is reasonable and explainable.

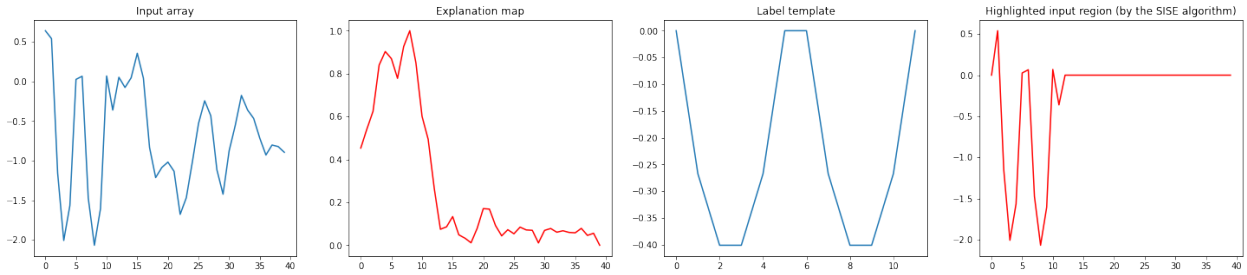


(a)

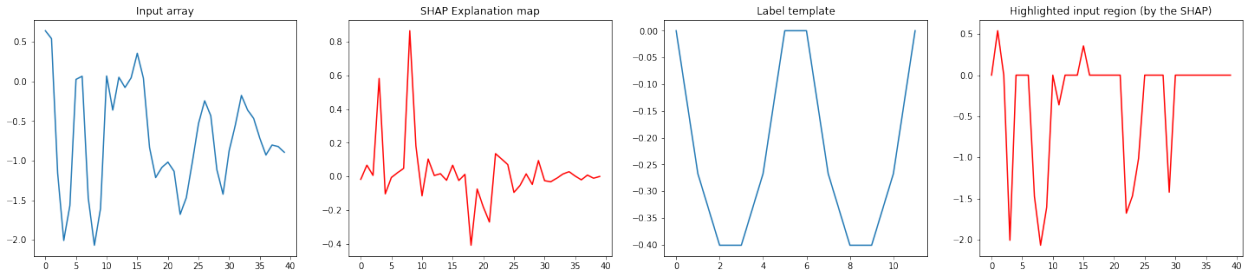


(b)

Figure 5: Qualitative example of class 3, model prediction as class 3, confidence score 1.0 for the correct label. (a) shows the explanation map and highlighted region given by SISE algorithm with threshold 0.067. (b) shows the same metrics given by SHAP algorithm with threshold 0.012.



(a)



(b)

Figure 6: Qualitative example of digit class 8, model prediction as class 8, confidence score 0.999 for the correct label. (a) shows the explanation map and highlighted region given by SISE algorithm with threshold 0.453. (b) shows the same metrics given by SHAP algorithm with threshold 0.027.

Part 2. Histopathological tissue classification

Task 3. Biomedical image classification and interpretation

3.1. Performance evaluation

Overall accuracy

Overall accuracy on test set is 0.888.

```
# Task3.1.a: Overall classification accuracy
model=load_model('models/HMT.h5')
test_generator.reset()
image_batch,label_batch=test_generator.next()
print("image_batch dim: {}, label_batch dim: {}".format(image_batch.shape, label_batch.shape))
prediction=model(image_batch)
hmt_pred = np.argmax(prediction, axis=1)
hmt_label = np.argmax(label_batch, axis=1)
print("Overall accuracy: {}".format(np.mean(hmt_pred == hmt_label)))


>image_batch dim: (496, 224, 224, 3), label_batch dim: (496, 8)
>Overall accuracy: 0.888
```

Listing 4: Overall accuracy measurement on HMT

Class-wise accuracy for all classes

Accuracy for each class: class 7 (empty) has the highest accuracy 1.0, and class 4 (Debris) has the lowest accuracy 0.661.

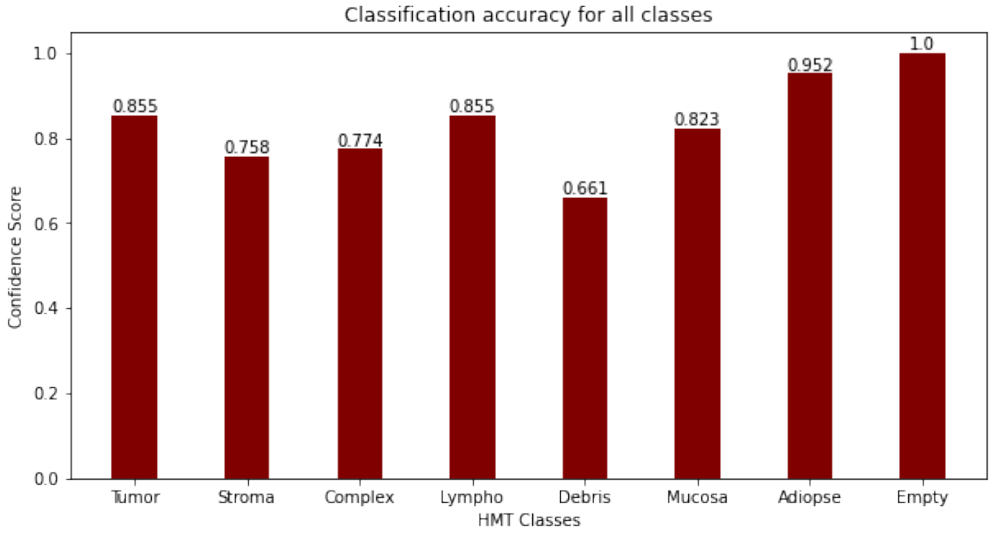
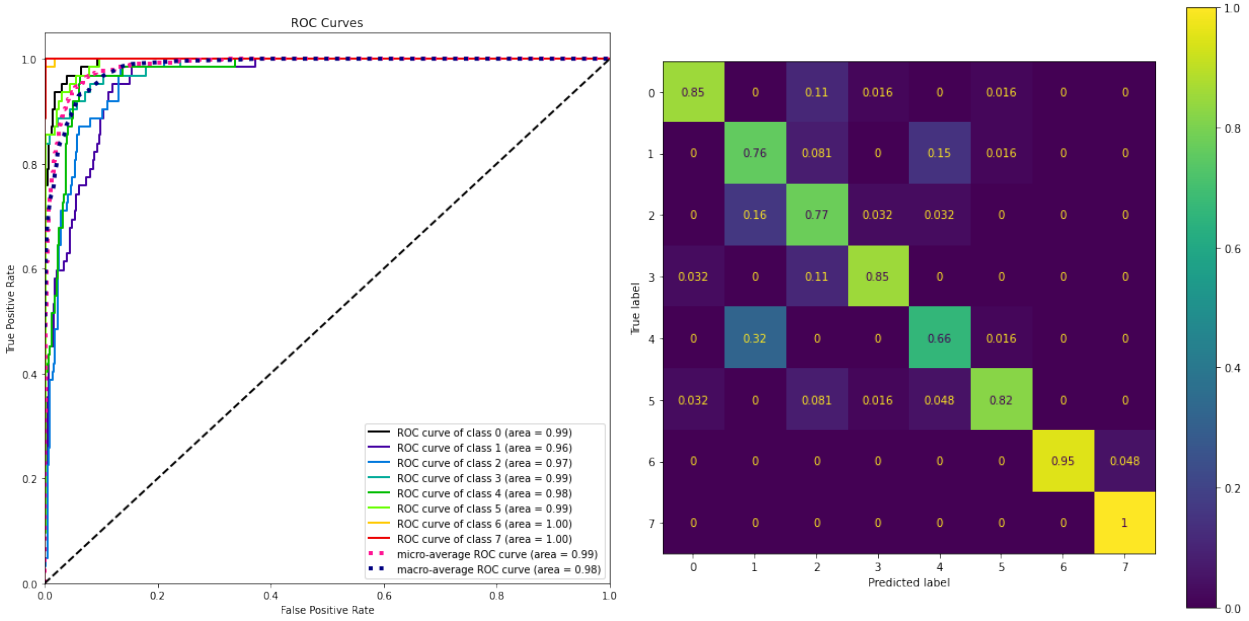


Figure 7: Accuracy for each class on HMT

ROC and AUC curves

Similar to previous section, we measure the AUC value of ROC on HMT test set. As we could see from Figure 8a, class 6 and class 7 both achieve 1.0, and the lowest AUC value among all classes is from class 1.



(a) ROC/AUC for each class

(b) Confusion matrix

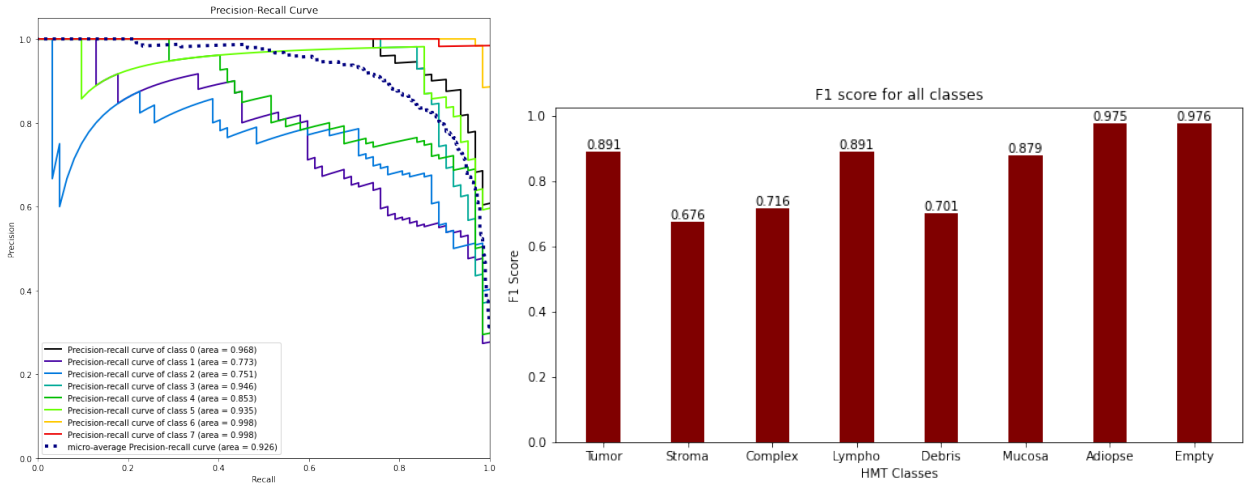
Figure 8: ROC, AUC and Confusion Matrix on HMT

Normalized confusion matrix

Figure 8b shows the confusion matrix of HMT test set where class 4 has the lowest accuracy score and it is easily miss-classified to class 1.

Precision, Recall, and F1 scores

The AUC value of precision-recall curve and corresponding F1 score are present in Figure 9a and Figure 9b. Under precision-recall curve metric, class 1 and class 2 does not perform well, which achieve a AUC value of 0.773 and 0.751 respectively. F1 scores metric shows a similar result, which class 1, class 2, and class 4 achieves the lowest score 0.676, 0.716, and 0.701 respectively.



(a) Precision-recall Curve on HMT

(b) F1 score for each class on HMT

Figure 9: Precision-recall curve and F1 score table on HMT

3.2. XAI on HMT dataset

```
# Reference: using provided SISE function
from xai_utils import *
# Randomly taking one data sample from test set
index = int(np.floor(np.random.rand()*1000))
# Get explanation map from SISE
explanation_map_SISE = SISE(np.expand_dims(image_batch[index], axis=0), model,
    layers=[['conv2d_3']], class_index=np.argmax(prediction[index]), grad_thr=0.)
print("SISE explanation map dim: {}".format(explanation_map_SISE.shape))


>SISE explanation map dim: (224, 224)
```

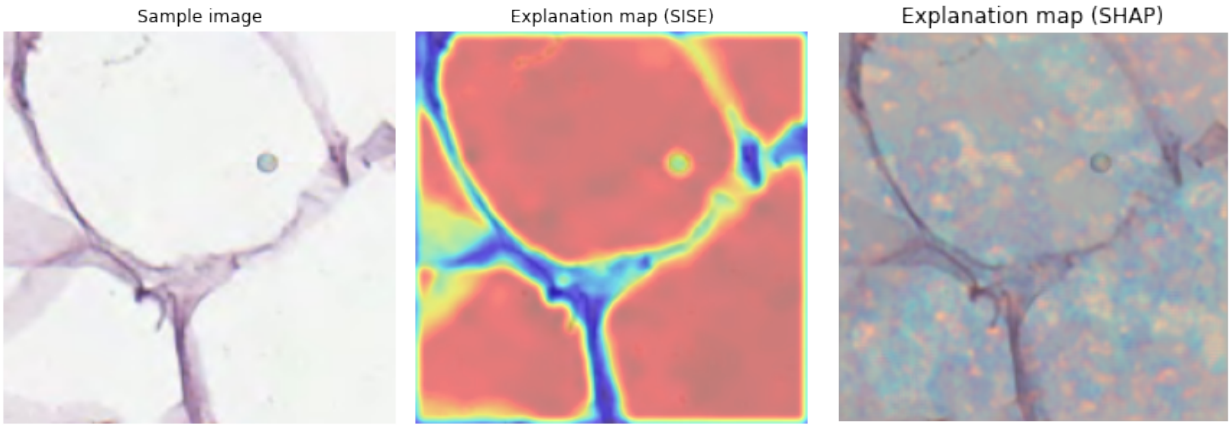
Listing 5: Explanation map from SISE on HMT

```
# Reference: using provided SISE function
import shap
# Randomly taking one data sample from test set
index = int(np.floor(np.random.rand()*1000))
# Get explanation map from SHAP
e = shap.GradientExplainer(modelHMT, image_batch)
elist = e.shap_values(np.expand_dims(image_batch[index], axis=0))
print("SHAP explanation map dim: {}".format(elist[0].shape))


>SHAP explanation map dim: (1, 224, 224, 3)
```

Listing 6: Explanation map from SHAP on HMT

Similar to previous section, we applied SISE and SHAP to evaluate model's behaviours on HMT dataset.

Figure 10 shows the original input image, image blending with SISE explanation map, and image blending with SHAP explanation map respectively. Colormap `jet` is used on plotting explanation map, which the more redness reflects on higher values in the explanation map. Figure 10b accurately presents the most important image pixels of class Adipose, whereas Figure 10c, the explanation map from SHAP, does not weight all important image pixels of class Adipose. Unlike to MNIST1-D dataset, SHAP does not perform well on this model with HMT dataset. One difference we would like to point out is that before we utilize `DeepExplainer`, but in this case we use `GradientExplainer` since it better fit to a VGG-7 network according to local trial and error experiment.



(a) (b) (c)

Figure 10: Qualitative example of class Adipose, model prediction as class Adipose, confidence score 0.986 for the correct label. (a) shows the original image. (b) shows the image blending with explanation map given by SISE algorithm. (c) shows the image blending with explanation map given by SHAP algorithm.

Task 4. Quantitative evaluation of the attribution methods

4.1. Drop and increase rate on MNIST1-D and HMT dataset

`calculate_drop_increase_mnist1d` and `calculate_drop_increase_hmt` shown in Listing 7 and Listing 8 are methods of calculating drop rate and increase rate among test set referenced from xai_utils.py. The functions require model, original test data, explanation map, and fraction as input. Parameter `frac` will be used to determine a threshold value from the explanation map, which a input mask of features dropping will be generated corresponding to this threshold value. The new input data is considered to contain the most weighted features given by the input mask. Therefore, the predictions on new data sample could be used to calculate drop rate and increase rate in order to analyze how well the explanation map interpret the model behaviours.

```
# Reference: Drop and increase rate calculation from xai_utils.py
def calculate_drop_increase_mnist1d(images, model, exmap, class_index, frac=0.15):
   predictions = model.predict(images)
   #print(images.shape)
   img=images[0,:,:]
```

```
   img = np.expand_dims(img,axis=0)
   Y=predictions[0][class_index]

   grad_array=np.reshape(exmap, (-1,))
   array_size=int(grad_array.shape[0]*frac)
   thr=np.flip(sorted(grad_array))[array_size]
   exmap1_msk=(exmap>thr)
   exmap1_thr=np.zeros(shape=(1,40,1))
   exmap1_thr=img*exmap1_msk
   ex_predictions = model.predict(exmap1_thr)[0]
   O1=ex_predictions[class_index]
   etta=(Y-O1)/(Y+1e-100)
   # Return drop, increase, original prediction, and new prediction
   return (etta*(etta>0), 1*(etta<0), Y, O1)
```

Listing 7: Drop and increase rate calculation on MNIST1-D

```
# Reference: Drop and increase rate calculation from xai_utils.py
def calculate_drop_increase_hmt(images, model, exmap, class_index, frac=0.15, shap=False):
   predictions = model.predict(images)
   img=images[0,:,:,:]
   img=img_to_array(img)
   img = np.expand_dims(img,axis=0)
   Y=predictions[0][class_index]

   grad_array=np.reshape(exmap, (-1,))
   array_size=int(grad_array.shape[0]*frac)
   thr=np.flip(sorted(grad_array))[array_size]
   exmap1_msk=(exmap>thr)
   exmap1_thr=np.zeros(shape=(1,224,224,3))
   # explanation map from shap has dim of (224, 224, 3)
   # explanation map from sise has dim of (224, 224)
   if shap:
      exmap1_thr[0,:,:,:]=img[0,:,:,:]*exmap1_msk
   else:
      exmap1_thr[0,:,:,0]=img[0,:,:,0]*exmap1_msk
      exmap1_thr[0,:,:,1]=img[0,:,:,1]*exmap1_msk
      exmap1_thr[0,:,:,2]=img[0,:,:,2]*exmap1_msk
   ex_predictions = model.predict(exmap1_thr)[0]
   O1=ex_predictions[class_index]
   etta=(Y-O1)/(Y+1e-100)
   return etta*(etta>0), 1*(etta<0), Y, O1
```

Listing 8: Drop and increase rate calculation on HMT

Here we would like to briefly mention our workaround approach for dealing with the issue of open source SHAP library being incompatible to TensorFlow 2.x. In TensorFlow 2.x environment, we load the models with disabling tensorflow 2.x behaviours and then fit them into SHAP explainer. In order to run the model prediction in TensorFlow 2.x manner while calculating drop rate and increase rate, we precompute the explanation map and store them as numpy array and directly load them from the file to bypass the issue. Listing 9 shows the code of precomputing and saving the explanation map from SHAP.

```
''' RUN ON TENSORFLOW 2.X WITH tf.compat.v1.disable_v2_behavior()'''
# precompute HMT explaination map from SHAP
eHMT = shap.GradientExplainer(modelHMT, image_batch)
hmt_expm_shap = np.zeros((496,224,224,3))
for index in range(496):
  label = np.argmax(label_batch[index])
  elist = eHMT.shap_values(np.expand_dims(image_batch[index], axis=0))
  hmt_explaination_map = np.squeeze(elist[0], axis=0)
  hmt_expm_shap[index] = hmt_explaination_map
np.save('hmt_expm_shap.npy', hmt_expm_shap)
print("hmt_expm_shap dim: {}".format(np.load("hmt_expm_shap.npy").shape))


>hmt_expm_shap dim: (496, 224, 224, 3)

# precompute MNIST1-D explaination map from SHAP
e = shap.DeepExplainer(model, np.expand_dims(x_test, axis=-1))
minist_expm_shap = np.zeros((1000,40))
for index in range(1000):
  label = y_test[index]
  elist = e.shap_values(np.expand_dims(np.expand_dims(x_test[index], axis=0), axis=-1))
  shap_explaination = np.squeeze(elist[label])
  minist_expm_shap[index] = shap_explaination
np.save('minist_expm_shap.npy', minist_expm_shap)
print("minist_expm_shap dim: {}".format(np.load("hmt_expm_shap.npy").shape))


>minist_expm_shap dim: (1000, 40)
```

Listing 9: Precompute explanation map from SHAP

The following code shows how we measure the average drop rate and increase rate on MNIST1-D with both SISE and SHAP. Results for HMT dataset is computed by the same setup. The results would be presented and analyzed in the next section.

```
explanation_map_shap = np.load('minist_expm_shap.npy')
drop_rate_sise, drop_rate_shap = 0., 0.
increase_rate_sise, increase_rate_shap = 0, 0
for index in range(1000):
   prediction=model(np.expand_dims(np.expand_dims(x_test[index], axis=0), axis=-1)).numpy()
   explanation_map_sise = np.expand_dims(explanation_map_shap[index], axis=-1)
   __drop, __increase, __true, __pred =
       calculate_drop_increase_sise(np.expand_dims(np.expand_dims(x_test[index], axis=0),
       axis=-1), model, explanation_map_sise, class_index=np.argmax(prediction[0]), frac=0.3)
   drop_rate_sise += __drop
   increase_rate_sise += __increase
   __drop, __increase, __true, __pred =
       calculate_drop_increase_sise(np.expand_dims(np.expand_dims(x_test[index], axis=0),
       axis=-1), model, explanation_map_shap[index], class_index=np.argmax(prediction[0]),
       frac=0.3)
   drop_rate_shap += __drop
   increase_rate_shap += __increase
drop_rate_sise /= 1000
```

```
increase_rate_sise /= 1000
drop_rate_shap /= 1000
increase_rate_shap /= 1000
print("drop rate sise: {}, increase rate sise: {}".format(drop_rate_sise, increase_rate_sise))
print("drop rate shap: {}, increase rate shap: {}".format(drop_rate_shap, increase_rate_shap))


> drop rate sise: 0.137, increase rate sise: 0.375
> drop rate shap: 0.459, increase rate shap: 0.2
```

Listing 10: Drop and increase rate calculation on MNIST1-D

4.2. Result Analysis

Dataset	XAI methods	Drop rate	Increase rate
MNIST1-D	SISE	0.137	0.375
MNIST1-D	SHAP	0.459	0.200
HMT	SISE	0.301	0.086
HMT	SHAP	0.439	0.057

Table 2: Average drop rate and increase rate of MNIST1-D and HMT

For both MNIST1-D and HMT datasets, SISE gives a better performance than SHAP. Specifically, when unimportant features selected by SHAP are removed, the confidence score dropped by 0.46 and 0.44 for MNIST1-D and HMT classification task respectively. When SISE is applied to refine features, the drop rate on both classification tasks are much lower(0.137 and 0.301).

There are couple reasons led to SISE out-perform SHAP. First, the vanilla version of SHAP APIs we used in previous computation does not treat images as the combination of super pixels. Therefore, the lack of analyzing dependencies among numerous features(pixels) might affect the performance on understanding model's behaviour. In addition, SHAP calculates the conditional expectation of marginal contribution by replacing absent features(pixels) with features from another random instance(image), however, most of the generated instances(images) are either similar or impractical to the original image. Moreover, due to the large size of features, the approximation of shapley values become inaccurate. On the other hand, SISE, a CNN-specified explanation method, fits both image classification tasks well due to its focused spatial resolution. Also SISE is able to make use of the dependencies among features by extracting and selecting feature maps of all convolutional blocks from the CNN model. Both SHAP and SISE achieves a lower drop rate and higher increase rate on MNIST-1D, proving that classification task on HMT dataset is generally harder than the task on MNIST-1D dataset.

Resources

Acknowledge the following resources in assist to complete the project. Hyperlinks has been bind into the text for easily access.

1. SHAP
2. Scikit-Plot
3. Evaluating a pre-trained CNN in Keras
4. ROC curves and Precision-Recall curves in Python
5. Confusion Matrix in Machine Learning

References

[1] Lundberg, Scott M., and Su-In Lee. "A unified approach to interpreting model predictions.", Advances in Neural Information Processing Systems. 2017

[2] Sam Sattarzadeh, Mahesh Sudhakar, Anthony Lem . "Explaining Convolutional Neural Networks through Attribution-Based Input Sampling and Block-Wise Feature Aggregation" 2020

[3] Lundberg, Scott M., Gabriel G. Erion, and Su-In Lee. "Consistent individualized feature attribution for tree ensembles." arXiv preprint arXiv:1802.03888 (2018)

Codes

The project is implemented by Python in jupyter notebook . Sample codes provided by the instructor are used with slight modifications. All codes could be viewed from our GitHub repository, and we will also submit a zip file of code along with the report through quercus.