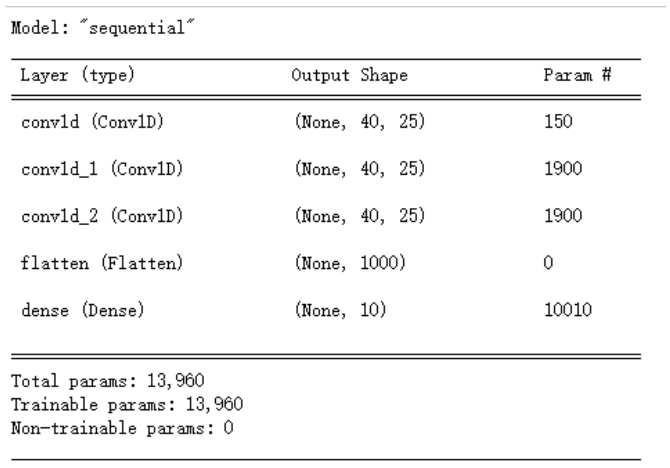
***Abstract*—This paper is a report for Project A of ECE1512 2022W, University of Toronto. In the paper, we described four assigned tasks with detail, including CNN construction and training, statistic assessment, XAI based interpretation and Quantitative evaluation. For the attribution methods, we choose Grad-CAM, Grad-CAM++ and Ablation-CAM as a group of two, and we reviewed their paper and implemented corresponding functions. Moreover, we tried to analyze the performance of the XAI methods based on quantitative evaluation and gave our explanations towards the experiment.**

I. Task 1: 1-Dimensional digit classification

A. *Question 1*

In this question, we build a ConvNet. It includes three convolutional layer, one flatten layer and one dense layer. The network is listed as Fig. 1:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv1d (Conv1D)             (None, 40, 25)            150

conv1d_1 (Conv1D)           (None, 40, 25)            1900

conv1d_2 (Conv1D)           (None, 40, 25)            1900

flatten (Flatten)           (None, 1000)              0

dense (Dense)               (None, 10)                10010

=================================================================
Total params: 13,960
Trainable params: 13,960
Non-trainable params: 0
_________________________________________________________________
```

Fig. 1. Task1-Question1: ConvNet Model

```
weight_decay = 5e-4
model = Sequential()
#Your code starts from here
model.add(Input(shape=(40,1)))
model.add(Conv1D(25, kernel_size=5, padding
    ='same', activation='relu',
    kernel_regularizer=regularizers.l2(
    weight_decay)))
model.add(Conv1D(25, kernel_size=3, padding
    ='same', activation='relu',
    kernel_regularizer=regularizers.l2(
    weight_decay)))
model.add(Conv1D(25, kernel_size=3, padding
    ='same', activation='relu',
    kernel_regularizer=regularizers.l2(
    weight_decay)))

model.add(Flatten())
model.add(Dense(10, activation='softmax',
    kernel_initializer=keras.initializers.
    RandomNormal(mean=0.0, stddev=0.5),
             bias_initializer=keras.
                 initializers.Zeros(),
                 kernel_regularizer=
                 regularizers.l2(
                 weight_decay)))
model.summary()
```

B. *Question 2*

In this section, we apply the model in question 1 to the MNIST1D dataset. The code is listed as followed:

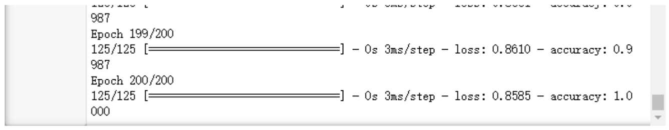
```
model.compile(loss=keras.losses.
    categorical_crossentropy,
        optimizer=tensorflow.keras.
            optimizers.SGD(),
        metrics=['accuracy'])

def lr_scheduler(epoch):
   base_ep = 15
   return 1e-3 * (.5 ** (epoch // base_ep))
lr_reduce_cb = keras.callbacks.
    LearningRateScheduler(lr_scheduler)
tensorboard_cb = keras.callbacks.
    TensorBoard(log_dir='log2', write_graph
    =True)
early_stopping_cb = keras.callbacks.
    EarlyStopping(patience=8, min_delta=0.)

# X = tensorflow.expand_dims(dataset['x'],
    axis=2)
train_x=dataset['x']
train_y=dataset['y']
train_x=train_x.reshape(4000,40,1)
train_y=tensorflow.keras.utils.
    to_categorical(train_y, num_classes=10)

# print(X.shape)
history=model.fit(x=train_x,y=train_y,
    epochs=200,
#                steps_per_epoch=len(X) //
    32,
                callbacks=[tensorboard_cb],
                shuffle = True,
                verbose=1)
model.save('MNIST1D.h5')
```

Here, we use the tensorboard to record the training procedure.First of all, we compile this model, set the loss function to cross-entropy, set the optimizer to Stochastic Gradient Descent and the metrics to accuracy. Then we define the LearningRateScheduler, the TensorBoard, the EarlyStopping for later use. After that, we handle the train data for training. At last, we will fit the data and tensorboard into the model for training and save the model into disk. The training result is shown as followed:



```
987
Epoch 199/200
125/125 [==============================] - 0s 3ms/step - loss: 0.8610 - accuracy: 0.9
987
Epoch 200/200
125/125 [==============================] - 0s 3ms/step - loss: 0.8585 - accuracy: 1.0
000
```

Fig. 2. Task1-Question2: Training Process Log

C. Question 3

1) SubQuestion a: This is the code for loss and accuracy curve.

```
   train_acc = history.history['accuracy']
   train_loss = history.history['loss']
   plt.plot(train_acc)
   plt.figure()
   plt.plot(train_loss)
```

In the code, we used the training history defined in the previous code to do the plot. The plot of loss curve3 and plot for accuracy curve4 are shown below:

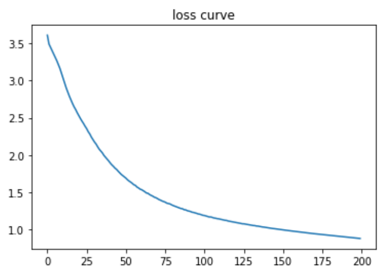


Fig. 3. Task1-Question3a-1: Loss Curve of 1-D CNN

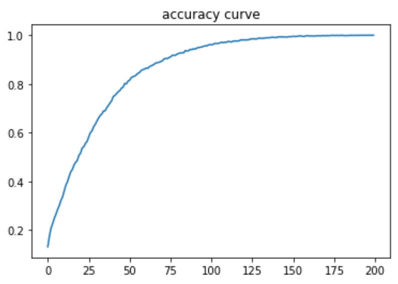


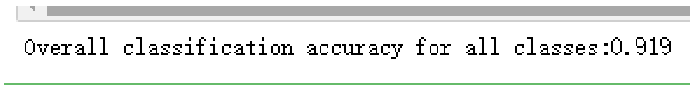
Fig. 4. Task1-Question3a-2: Accuracy Curve of 1-D CNN

2) SubQuestion b: This part talks about overall classification accuracy on the test set.

```
   # Use Scikit-learn to calculate stats
   from sklearn.metrics import accuracy_score,
        precision_score, recall_score,f1_score
   from sklearn.metrics import
       classification_report

   # Q3.b get the prediction from the test set
   x_test = dataset['x_test']
   y_test = dataset['y_test']
   x_test=x_test.reshape(1000,40,1)
   # predict_classes removed in tf 2.6.0
   y_pred = model.predict(x_test)
   y_predicted = np.argmax(y_pred,axis=1)
   print('Overall classification accuracy for
       all classes:'+str(np.sum(y_predicted==
       y_test)/y_test.shape[0]))
```

We read the test set, fit the model on the set and calculate the accuracy by deciding the max prediction in every class and compare them with the ground truth. The result is 0.9195.



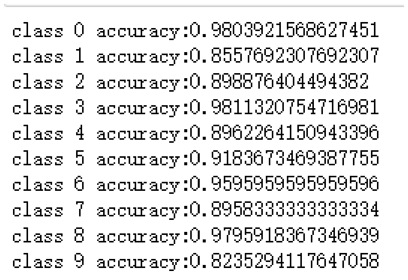
```
Overall classification accuracy for all classes:0.919
```

Fig. 5. Task1-Question3b: Overall accuracy

3) SubQuestion c: In this part, we calculated the class-wise accuracy for all classes.This is our code:

```
   # Q3.c class-wise classification
for i in range(10):
   true_i=np.where(y_test==i)[0]
   res=0
   for j in true_i:
      if y_predicted[j]==i:
         res=res+1
   print('class '+str(i)+' accuracy:'+str(res/
       true_i.shape[0]))
```

We find out the lines which belongs to class i and find the ones which is correct in these lines to calculate the accuracy. The result is shown below6:



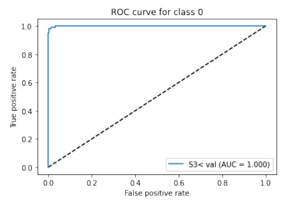
```
class 0 accuracy:0.9803921568627451
class 1 accuracy:0.8557692307692307
class 2 accuracy:0.898876404494382
class 3 accuracy:0.9811320754716981
class 4 accuracy:0.8962264150943396
class 5 accuracy:0.9183673469387755
class 6 accuracy:0.9595959595959596
class 7 accuracy:0.8958333333333334
class 8 accuracy:0.9795918367346939
class 9 accuracy:0.8235294117647058
```

Fig. 6. Task1-Question3b: class-wise accuracy

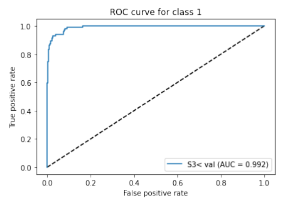
4) SubQuestion d: In this part, we will plot the ROC and AUC curves for each class. Here is the code:

```
   # Q3.d ROC and AUC curve for every class
   from sklearn.metrics import roc_curve
   from sklearn.metrics import auc
   # print(dataset['y_test'].shape)
   real_y=np.zeros((dataset['y_test'].size,10)
       )
   for i in range(y_test.size):
      real_y[i,y_test[i]]=1
   for i in range(10):
      FPR, TPR, thresholds_keras = roc_curve(
          real_y[:,i], y_pred[:,i])
      AUC = auc(FPR, TPR)
      print("AUC : ", AUC)
      plt.figure()
      plt.plot([0, 1], [0, 1], 'k--')
      plt.plot(FPR, TPR, label='S3< val (AUC =
          {:.3f})'.format(AUC))
      plt.xlabel('False positive rate')
      plt.ylabel('True positive rate')
      plt.title('ROC curve for class '+str(i))
      plt.legend(loc='best')
      plt.show()
```

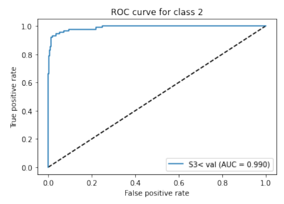
In the code, we first reshape the test set to 2D set. Then we apply the test set and prediction set to roc_curve. Then we will calculate the auc number and plot the curve.The AUC and ROC curve is listed below:



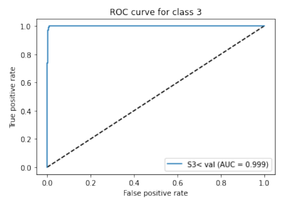
(a) Class 0



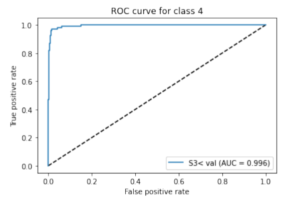
(b) Class 1



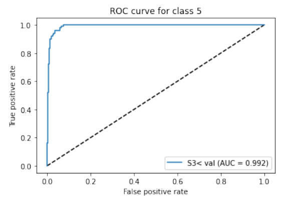
(c) Class 2



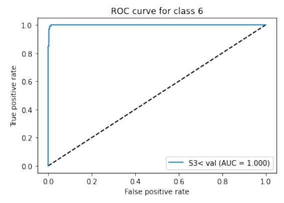
(d) Class 3



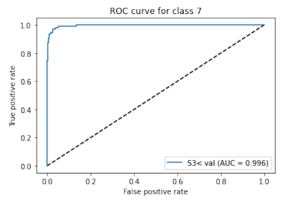
(e) Class 4



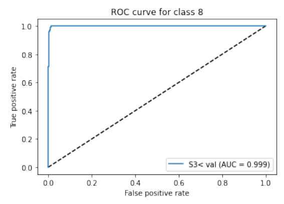
(f) Class 5



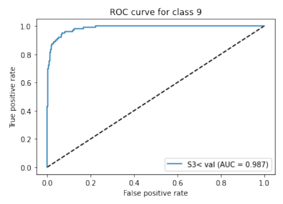
(g) Class 6



(h) Class 7



(i) Class 8



(j) Class 9

Fig. 7. Class-wise ROC Curve Plots on MNIST-1D dataset

5) SubQuestion e: This is the code for normalized confusion matrix.

```
from sklearn.metrics import
    confusion_matrix
cm=confusion_matrix(list(y_test),list(
    y_predicted))
cm_normalized = cm.astype('float') / cm.sum
    (axis=1)[:, np.newaxis]
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest',
     cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
classes=[0,1,2,3,4,5,6,7,8,9]
# print(classes)
# We want to show all ticks...
ax.set(xticks=np.arange(cm_normalized.shape
    [1]),
   yticks=np.arange(cm_normalized.shape[0])
       ,
   # ... and label them with the respective
        list entries
   xticklabels=classes, yticklabels=classes
       ,
   ylabel='True label',
   xlabel='Predicted label')
plt.setp(ax.get_xticklabels(), ha="right",
         rotation_mode="anchor")
thresh = cm_normalized.max() / 2.
for i in range(cm_normalized.shape[0]):
   for j in range(cm_normalized.shape[1]):
      ax.text(j, i, format(cm_normalized[i,
          j], '.2f'),
           ha="center", va="center",
           color="white" if cm_normalized[
               i, j] > thresh else "black"
               )
fig.tight_layout()
```

The result plot is listed in Fig.8.

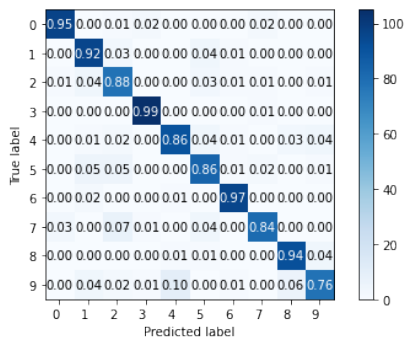


Fig. 8. normalized confusion matrix plot

6) SubQuestion f: This is the code for calculating Precision, Recall, and F-1 score on the test set.

```
# Q3.f
print(classification_report(y_true=y_test,
    y_pred=y_predicted))
```

This is the result for the plot9:

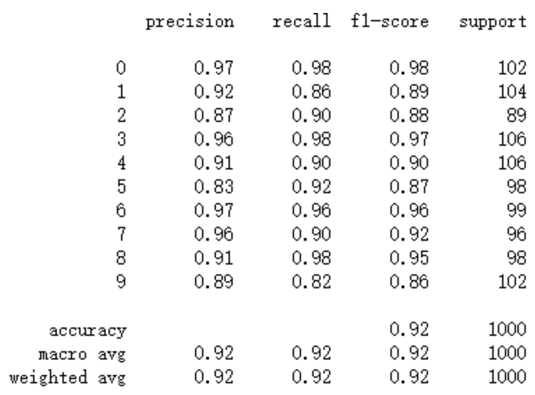
D. Question 4

To do this question, we did some coding in our program, the codes are listed below:

Success Examples:

```
index 10 true class: 0 predicted class: 0
index 412 true class: 1 predicted class: 1
index 729 true class: 2 predicted class: 2
```

Failure Examples:



```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       102
           1       0.92      0.86      0.89       104
           2       0.87      0.90      0.88        89
           3       0.96      0.98      0.97       106
           4       0.91      0.90      0.90       106
           5       0.83      0.92      0.87        98
           6       0.97      0.96      0.96        99
           7       0.96      0.90      0.92        96
           8       0.91      0.98      0.95        98
           9       0.89      0.82      0.86       102

    accuracy                           0.92      1000
   macro avg       0.92      0.92      0.92      1000
weighted avg       0.92      0.92      0.92      1000
```

Fig. 9. Precision,Recall and F-1 score

```
index 236 true class: 0 predicted class: 7
index 6 true class: 1 predicted class: 2
index 25 true class: 1 predicted class: 4
```

The concrete Success/Failure cases are recorded in 'MNIST1D.ipynb', if you want to refer to more examples. The most misclassification lies in class 1 and 4. The reason for that is TODOOO: analyze

II. Task 2: CNN interprectation

This section introduces our interpretation of 1-D CNN model based on MNIST-1D dataset using 3 different attribution methods, including our literature review, discussion and implementation of the XAI attribute algorithms.

A. *Grad-CAM*

In the previous study, researchers introduced CAM to explain the CNN. However, CAM requires to modify the structure of original training models. It limits the usage of CAM greatly, because the cost of retraining the model is quite high for the published model. It is almost impossible to retrain them. To solve this problem, Grad-CAM was proposed, which is similar to CAM. It also gets the weight of every feature maps and calculate the results accordingly. The difference lies in the calculation of weights. CAM replaced full-connection layer with GAP layer and retrain this whole model. In contrast to that, Grad-CAM put forward a new way to do it. It calculates the weight by averaging the gradients which is equivalent to the CAM. It makes CAM applicable to all the existing models.

The core algorithm of Grad-CAM is based on CAM which proposed: class c gets the final classification score Y_c with a linear combination of its global average pooled last convolutional layer feature maps A_k.

$$Y^c = \Sigma_k w_k^c \Sigma_i \Sigma_j A_{ij}^k \tag{1}$$

So, first, it defines the weight of feature map k to class c as α_k^c, and it calculates the weight using the global gradients2:

$$\alpha_k^c = \frac{1}{Z} * \Sigma_i \Sigma_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{2}$$

In which, y_c is the gradient of the score for class c, A_k is the feature map activation. $\frac{\partial y^c}{\partial A_{ij}^k}$ is the gradient achieved by backpropagating. $\frac{1}{Z}$ is global averaging. After calculating the α_k^c, the algorithm computed the weighted combination of forward activation maps, followed by a ReLU function:

$$L_{Grad-CAM}^c = ReLU(\Sigma_k \alpha_k^c A^k)$$

The advantages for Grad-CAM is quite obvious: First, compared with CAM, it does not need to change the model and retrain it which costs a lot of time and money. The novelty is the usage of propagation and gradients to compute the weights of feature maps.It makes the visualization of published models possible. Second is that the algorithm is quite easy to code and maintain.

The disadvantages for Grad-CAM is that: it considers the global features for all classes. As a result, Grad-CAM cannot localize the targets when faced with many targets of the same class. If there are many objects of the same class in an image, it cannot localize the targets well or it can only locate part of them.

B. *Grad-CAM++*

Grad-CAM++ is an algorithm based on Grad-CAM, aiming to get a better explanation for CNN models. It improves the performance of Grad-CAM when facing targets of the same class, and helps to better localize the targets.

It introduced a weighted combination of gradients of the output in the pixel level, which provides a measure of importance of every pixels to the feature maps. It introduced a closed-form solution for the pixels weights, thus making the improvement of explanation possible.

The core algorithm is shown below: First, as we have introduced above, class c gets the final classification score Y_c with a linear combination of its global average pooled last convolutional layer feature maps Ak in equation2. Grad-CAM calculates the weight w_k^c with equation**??**. To improve the performance, Grad-CAM++ changed this equation. Combine the equation2 and **??** together, we will get the equation3:

$$Y^c = \Sigma_k[\Sigma_i \Sigma_j (\Sigma_a \Sigma_b \alpha_{ab}^{kc} relu(\frac{\partial Y_c}{\partial A_{ab}^k})) A_{ij}^k] \tag{3}$$

Take partial derivative w.r.t. A_{ij}^k on both sides:

$$\frac{\partial Y_c}{\partial A_{ij}^k} = \Sigma_a \Sigma b \alpha_{ab}^{kc} \frac{\partial Y_c}{\partial A_{ab}^k} + \Sigma_a \Sigma b A_{ab}^k \{\alpha_{ij}^{kc} \frac{\partial^2 Y_c}{(\partial A_{ij}^k)^2}\} \tag{4}$$

Take a further partial derivative w.r.t. A_{ij}^k on both sides:

$$\frac{\partial^2 Y_c}{(\partial A_{ij}^k)^2} = 2\alpha_{ij}^{kc} \frac{\partial^2 Y_c}{(\partial A_{ij}^k)^2} + \Sigma_a \Sigma b A_{ab}^k \{\alpha_{ij}^{kc} \frac{\partial^3 Y_c}{(\partial A_{ij}^k)^3}\} \tag{5}$$

Rearrage equation5, we get:

$$\alpha_{ij}^{kc} = \frac{\frac{\partial^2 Y_c}{(\partial A_{ij}^k)^2}}{2\frac{\partial^2 Y_c}{(\partial A_{ij}^k)^2} + \Sigma_a \Sigma b A_{ab}^k \{\frac{\partial^3 Y_c}{(\partial A_{ij}^k)^3}\}} \tag{6}$$

So, we are able to get weight w_k with:

$$w_k^c = \Sigma_i \Sigma_j \alpha_{ij}^{kc} relu(\frac{\partial Y_c}{\partial A_{ij}^k}) \tag{7}$$

Applying equation7 to **??**, we will get final results. There are many advantages for Grad-CAM++: First, it clearly improved the performance of Grad-CAM when facing with many objects of the same class in an image. It helps us to localize the objects more accurately. Second is that it only adds tow partial derivative to the computation, and it uses backpropagation only once. So it does not increase the computation time a lot.

However, it still remains to be revised in some aspects: First, although it improved the performance of Grad-CAM in some degrees, its performance is far from perfect. The same problem remains and connot be solved completelly. Second, from my perspective, ignoring the relu function in the alpha calculation will decrease the precision.

Our implementation code is as followed:

```
ef grad_cam_plus_plus(input_model, image,
    layer_name, class_index=None):
# if class_index is None:
#    class_index=np.argmax(input_model.
    predict(np.array([image])), axis=-1)[0]
"""GradCAM method for visualizing input
    saliency."""
# cls = np.argmax(input_model.predict(image
    ))
y_c = input_model.output
conv_output = input_model.get_layer(
    layer_name).output
feedforward1 = keras.models.Model([
    input_model.input], [conv_output, y_c])
# with tf.GradientTape() as tape1:
#    with tf.GradientTape() as tape2:
#       with tf.GradientTape() as tape3:
#          ff_results = feedforward1([image
    ])
#          all_fmap_masks, predictions =
    ff_results[0], ff_results[-1]
if class_index==None:
   cls=np.argmax(input_model.predict(image)
       )
else:
   cls=class_index
#          loss = predictions[:, cls]
#       grads_val = tape3.gradient(loss,
    all_fmap_masks)
#    grads_val2 = tape2.gradient(grads_val,
    all_fmap_masks)
# grads_val3 = tape1.gradient(grads_val2,
    all_fmap_masks)
with tf.GradientTape() as tape:
   ff_results=feedforward1([image])
   all_fmap_masks, predictions = ff_results
       [0], ff_results[-1]
   loss = predictions[:, cls]
grads_val = tape.gradient(loss,
    all_fmap_masks)
grads_val2=grads_val**2
grads_val3=grads_val2*grads_val
if len(image.shape) == 3:
   axis = (0, 1)
elif len(image.shape) == 4:
   axis = (0, 1, 2)
alpha_div=(2.0 * grads_val2 + grads_val3 *
    np.sum(all_fmap_masks, axis))
alpha_div = np.where(alpha_div != 0.0,
    alpha_div, 0)
alpha = grads_val2 / alpha_div
weights = np.maximum(grads_val, 0.0) *
    alpha
weights = np.sum(weights, axis=axis)
# weights = np.mean(grads_val, axis=axis)
cam = np.dot(all_fmap_masks[0], weights)
# print (cam)
H, W = image.shape[1:3]
cam = np.maximum(cam, 0)
# cam = resize(cam, (H, W))
cam = zoom(cam, H / cam.shape[0])
# cam = np.maximum(cam, 0)
cam = cam / cam.max()
return cam
```

C. Ablation-CAM

The Ablation-CAM creatively uses ablation analysis to determine the importance of individual feature map units for different classes. It proposes a novel “gradient-free” visualization approach which avoids use of gradients and at the same time, produce high quality class-discriminative localization maps.

The core algorithm of Ablation-CAM is not complex: it uses the value of slope to describe the effect of ablation of individual unit k by the following formula:

$$slope = \frac{y^c - y_k^c}{||A_k||}$$

In the formula, y^c stands for activation score of class c, which represent the entire class activation status. y_k^c indicates the value of the function for absence of unit k, where A_k is the baseline. Those concepts lead us to ablation study, which is the basic principle of the method.

Ablation study is a method to distribute the influencing importance of different factors by controlling the variable while switching the combination of potential factors, and also their standalone. For example, if we’d like to know whether A or B component of medicine could improve the effect of an old medicine C. We could compare $C + A$, $C + B$ and also $C+A+B$ with the baseline of C. We could know if the A or B or they together are able to improve the effect. In the instance of Ablation-CAM, different unit k is the ”component”, and the whole feature map is so-called baseline, A_k. Thus, using slope described in the previous formula could represent the importance of a single unit to the feature map.

However practically, norm $||A_k||$ is hard to compute due to its large size and hence the slope could be approximately presented by the following formula, assuming a very small value.

$$w_k^c = \frac{y^c - y_k^c}{y^c}$$

As the algorithm, Ablation-CAM can then be obtained as weighted linear combination of activation maps and corresponding weights from the formula above, which is somehow similar to that of Grad-CAM.

$$L^C_{Ablation-CAM} = ReLU(\sum_k w^c_k A_k)$$

There are a number of advantages and features of Ablation-CAM. Firstly, a significant contribution and novelty of the Ablation-CAM is the ablation analysis it used to decide the weights of feature map units. Secondly, it could produce a coarse localization map highlighting the regions in the image for prediction. Thirdly, compare to other CAM methods, this approach works essentially better when it is full connected to obtain the result, which is known as final linear classifier, and have as good performance as other gradient-based CAM methods when evaluating other CNNs. Last but not the least, the approach introduce a gradient-free principle which avoids use of gradient as Grad-CAM does and produce a high-quality class-wise localization maps, which helps it to adapt into any CNN based architecture.

However, the approach have some limitations as well. First of all, the computational time required to generate a single Ablation-CAM is much grater than the required for Grad-CAM, as it has to iterate over each feature map to ablate it and check the drop in class activation score correspondingly. On the hand, the Ablation-CAM only benefits the interpretation where last convolutional layer is not followed immediately by decision nodes, yet show the same performance statistically as other CAM methods.

Our implementation code is as followed.

```
def extract_feature_map(img, model,
    class_index=None, layer_name="conv1d_2"):
   # Get gradients for the class on the last
       conv layer
   gradModel = tf.keras.models.Model([model.
       inputs],[model.get_layer(layer_name).
       output, model.output])
   print("gradModel = ")
   print(gradModel)
   # Get Activation Map on the last conv layer
   with tf.GradientTape() as tape:
      # Get Prediction on the last conv layer
      convOutputs, predictions = gradModel(np.
          array([img]))
      output = convOutputs[0]
      print("#prediction#")
      print(predictions)
      print("OUTPUT")
      print(output)

   if class_index is None:
      class_index = np.argmax(model.predict(np
          .array([img])), axis = -1)[0]
      y_class = np.max(model.predict(np.array
          ([img])))
   else:
      y_class = model.predict(np.array([img]))
          [0][class_index]
   # Get Weights on the layer
   weights = np.zeros(model.get_layer(
       layer_name).get_weights()[1].shape)
   # Get Weights for the maps
   allWeights = model.get_layer(layer_name).
       get_weights().copy()
   zeroWeight = allWeights[0][:,:,:,0]*0
   localWeight = [np.zeros(allWeights[0].shape
       )]
   localWeight.append(np.zeros(allWeights[1].
       shape))

   for i in range(weights.shape[0]):
      localWeight[0] = allWeights[0].copy()
      localWeight[0][:,:,:,i] = zeroWeight
      model.get_layer(layer_name).set_weights(
          localWeight)
      y_pred = model.predict(np.array([img]))
          [0][class_index]
      weights[i] = (y_class - y_pred)/y_class
          # Simplified Formula
      model.get_layer(layer_name).set_weights(
          allWeights)

   outputMean = np.mean([output[:,:,i] for i
       in range(output.shape[2])], axis = 0)
   outputMean = np.maximum(outputMean, 0.0)
   outMeanMask = np.zeros(output.shape[0:2],
       dtype = np.float32)
   for i in range(output.shape[0]):
      for j in range(output.shape[1]):
         if outputMean[i][j] < np.mean(
             outputMean[:,:]):
            outMeanMask[i][j] = 255
         else:
            outMeanMask[i][j] = 0
   return weights, output, outputMean,
       outMeanMask

def ablation_cam(weights, output):
   ablationMap = weights * output
   ablationCam = np.sum(ablationMap, axis=(2))

   ablationMask = np.zeros(ablationMap.shape
       [0:2], dtype = np.float32)
   for i in range(ablationMap.shape[0]):
      for j in range(ablationMap.shape[1]):
         if ablationCam[i][j] < np.mean(
             ablationCam[:,:]):
            ablationMask[i][j] = 255
         else:
            ablationMask[i][j] = 0

   return ablationCam, ablationMask
```

III. Task 3: Biomedical image classification and interpretation

A. Question 1

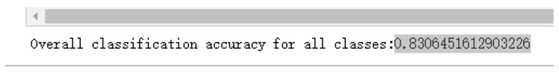
1) Overall classification accuracy: Here we calculate the overall classification accuracy, here is the code:

```
   from sklearn.metrics import accuracy_score,
        precision_score, recall_score,f1_score
   from sklearn.metrics import
       classification_report
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import
    confusion_matrix

test_generator.reset()
y_test=test_generator.classes
y_pred=model.predict(test_generator)
y_predicted=np.argmax(y_pred, axis=1)
print('Overall classification accuracy for
    all classes:'+str(np.sum(y_predicted==
    y_test)/y_test.shape[0]))
```

The result is 83.06%, as is shown in Fig10:



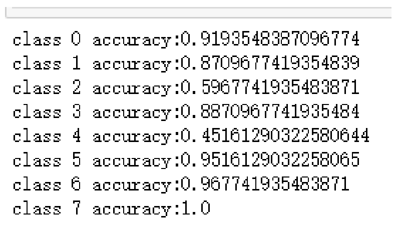
```
Overall classification accuracy for all classes:0.8306451612903226
```

Fig. 10. Overall classification accuracy

2) class-wise classification accuracy: In this part, we discuss the classification accuracy for every class.Here is the code:

```
for i in range(8):
true_i=np.where(y_test==i)[0]
res=0
for j in true_i:
    if y_predicted[j]==i:
        res=res+1
print('class '+str(i)+' accuracy:'+str(res/
    true_i.shape[0]))
```

The results are shown in Fig11:



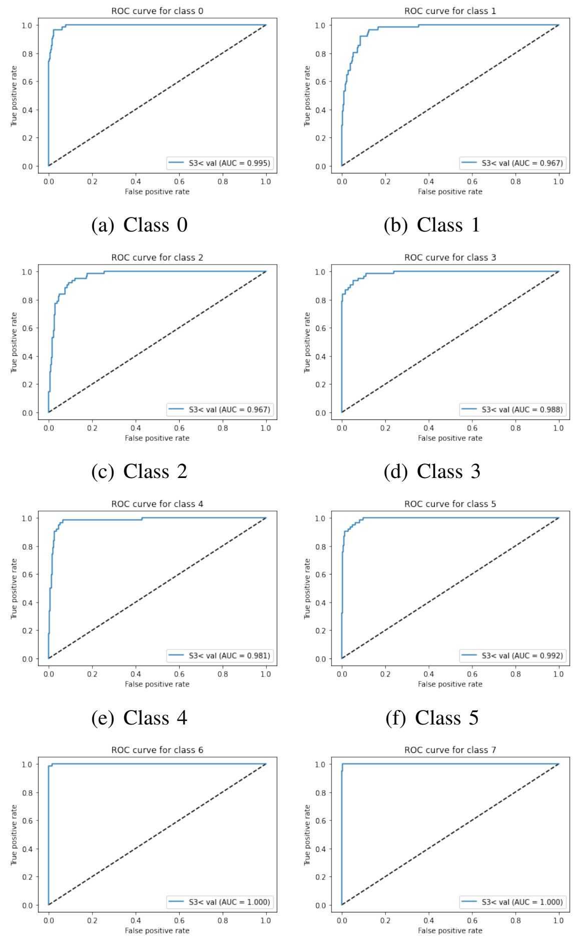
```
class 0 accuracy:0.9193548387096774
class 1 accuracy:0.8709677419354839
class 2 accuracy:0.5967741935483871
class 3 accuracy:0.8870967741935484
class 4 accuracy:0.45161290322580644
class 5 accuracy:0.9516129032258065
class 6 accuracy:0.967741935483871
class 7 accuracy:1.0
```

Fig. 11. class-wise classification accuracy

3) AUC-ROC curve: In this part, we discuss the AUC and ROC curve.Here is the code:

```
real_y=np.zeros((y_test.size,8))
for i in range(y_test.size):
    real_y[i,y_test[i]]=1
for i in range(8):
    FPR, TPR, thresholds_keras = roc_curve(
        real_y[:,i], y_pred[:,i])
    AUC = auc(FPR, TPR)
    print("AUC for class "+str(i)+": ", AUC)
    plt.figure()
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(FPR, TPR, label='S3< val (AUC =
        {:.3f})'.format(AUC))
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC curve for class '+str(i))
    plt.legend(loc='best')
    plt.show()
```

The plots are shown below:



(a) Class 0 (b) Class 1

(c) Class 2 (d) Class 3

(e) Class 4 (f) Class 5

(g) Class 6 (h) Class 7

Fig. 12. Class-wise ROC Curve Plots of HMT dataset

4) normalized confusion matrix: In this part, we discuss the normalized confusion matrix plot.Here is the code:

```
cm=confusion_matrix(list(y_test),list(
    y_predicted))
cm_normalized = cm.astype('float') / cm.sum
    (axis=1)[:, np.newaxis]
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest',
     cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
classes=[0,1,2,3,4,5,6,7]
# print(classes)
# We want to show all ticks...
ax.set(xticks=np.arange(cm_normalized.shape
    [1]),
    yticks=np.arange(cm_normalized.shape[0])
       ,
    # ... and label them with the respective
         list entries
    xticklabels=classes, yticklabels=classes
       ,
      ylabel='True label',
      xlabel='Predicted label')
plt.setp(ax.get_xticklabels(), ha="right",
         rotation_mode="anchor")
thresh = cm_normalized.max() / 2.
for i in range(cm_normalized.shape[0]):
   for j in range(cm_normalized.shape[1]):
      ax.text(j, i, format(cm_normalized[i,
           j], '.2f'),
            ha="center", va="center",
            color="white" if cm_normalized[
                i, j] > thresh else "black"
                )
fig.tight_layout()
```

The result plot is shown in Fig.13.

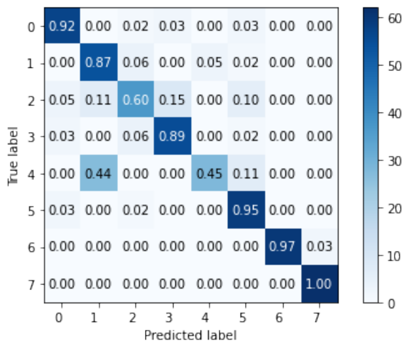
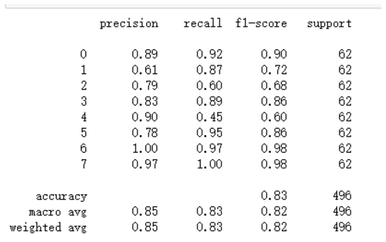


Fig. 13. normalized confusion matrix plot

5) Precision, Recall, and F-1 score: In this part, we discuss the normalized confusion matrix plot.Here is the code:

```
print(classification_report(y_true=y_test,
    y_pred=y_predicted))
```

The result plot is shown in Fig.14.



```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90        62
           1       0.61      0.87      0.72        62
           2       0.79      0.60      0.68        62
           3       0.83      0.89      0.86        62
           4       0.90      0.45      0.60        62
           5       0.78      0.95      0.86        62
           6       1.00      0.97      0.98        62
           7       0.97      1.00      0.98        62

    accuracy                           0.83       496
   macro avg       0.85      0.83      0.82       496
weighted avg       0.85      0.83      0.82       496
```

Fig. 14. Precision, Recall, and F-1 score

B. Question 2

implement 2d capable functions.
codes
modifications

IV. Task 4: Quantitative Evaluation of the Attribution Methods

A. Experiment

apply ways, codes,
table of drop/increase rate both MNIST/HMT (TODO)

B. Discussion

data analysis
successful? compare to benchmark
fail?
Two methods comparasion.

References

[1] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling Byzantine Agreements for Cryptocurrencies,” in Proceedings of the 26th Symposium on operating systems principles, 2017, pp. 51–68. doi: 10.1145/3132747.3132757.

[2] King, Sunny, and Scott Nadal. ”Ppcoin: Peer-to-peer crypto-currency with proof-of-stake.” self-published paper, August 19.1, 2012.