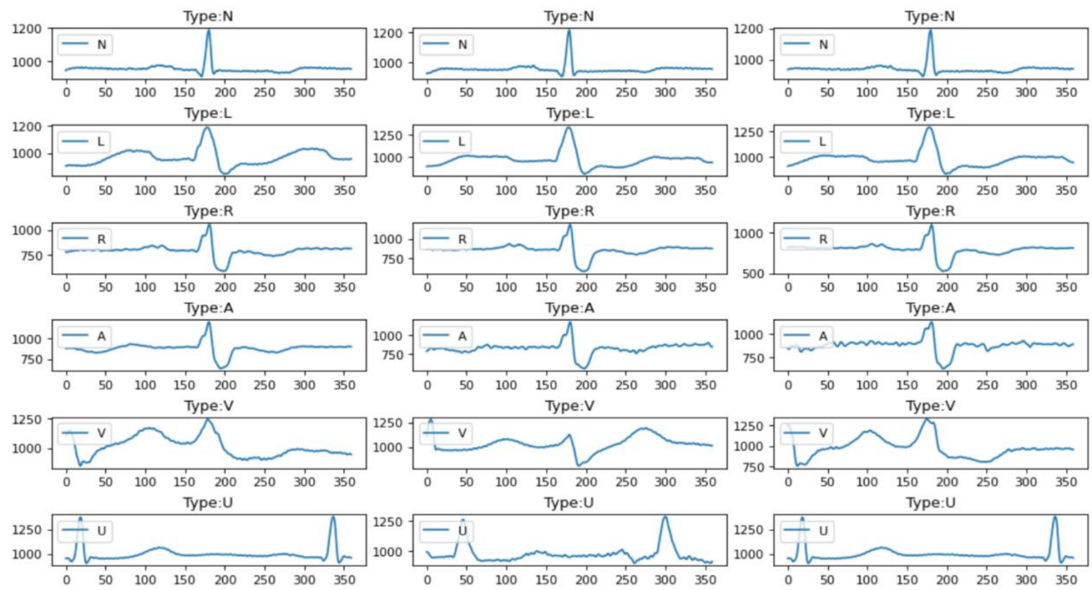
Mini Project1 Report

Xiangyu Gao & Rui Li

Answer1)



There are some differences between these plots. Type N is relatively flat and can reach a maximum of about 1200 in places with large fluctuations. Type L is not as smooth as N, with larger fluctuations during rest periods and increasing fluctuations. Type R is a bit similar to Type N, but its peak value is only around 1000, and the difference between peak and valley values is relatively large. Types A and R are similar, but A is more volatile and noisier than R. Type V is the most irregular of all types, while Type U has two peaks and is easier to identify.
And the total number of data is 97838.

```
In [9]: # Print length of dataframe you got after preprocessing.
        len(data)

Out[9]: 97838
```

Answer2)

```python
def type_modify(data):
    for i in range(len(data['Type'])):
        if data['Type'][i] == 'N':
            data['Type'][i] = 0
        if data['Type'][i] == 'A':
            data['Type'][i] = 1
        if data['Type'][i] == 'U':
            data['Type'][i] = 2
        if data['Type'][i] == 'L':
            data['Type'][i] = 3
        if data['Type'][i] == 'R':
            data['Type'][i] = 4
        if data['Type'][i] == 'V':
            data['Type'][i] = 5
    return data
data = type_modify(data)
```

I transferred the target type from letters to numbers.

```
def cleaning(df):
  # Take all the features as input, and do any data cleaning necessary.
  # YOUR CODE HERE
    df_re = pd.DataFrame()
    for i in set(df['Type']):
        df_i = df.where(df['Type'] == i)
        df_i = df_i.replace(np.inf, np.nan)
        df_i = df_i.dropna(axis=0,how='all')
        df_i = df_i.fillna(df_i.mean())
        df_re = pd.concat([df_re,df_i],axis = 0)
    return df_re.sort_index()
data_cln = cleaning(data)
```

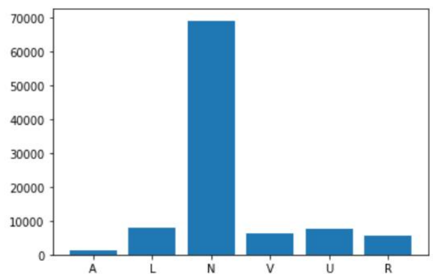
Cleaned the data by removing infinity values and none values.

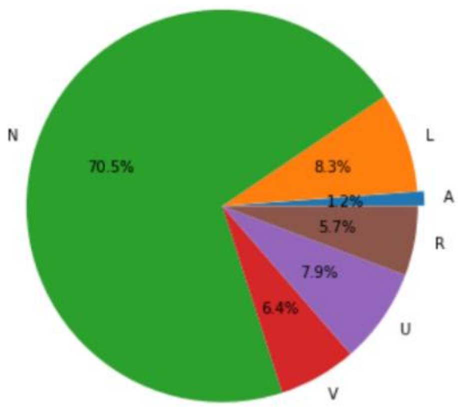
```
def normalizer(df,nor_type):
  # Taken input the output of cleaning function,
#and perform data normalization independently for all the features.
  # YOUR CODE HERE
    if nor_type == '0_1':
        data_0_1 = df.copy()
        scaler_0_1=MinMaxScaler()
        for col in df.columns.drop('Type'):
             data_0_1[col] = scaler_0_1.fit_transform(data_0_1[col].values.reshape(-1,1))
        return data_0_1
    elif nor_type == 'std':
        data_std = df.copy()
        std = StandardScaler()
        for col in df.columns.drop('Type'):
            data_std[col] = std.fit_transform(data_std[col].values.reshape(-1,1))
        return data_std
data_std = normalizer(data_cln,'0_1')
```

Then used MinMaxScaler() or StandardScaler() to normalize the data

Answer3)

```
the number of A is: 1156 , accounts for 0.012
the number of L is: 8075 , accounts for 0.083
the number of N is: 69013 , accounts for 0.705
the number of V is: 6287 , accounts for 0.064
the number of U is: 7699 , accounts for 0.079
the number of R is: 5608 , accounts for 0.057
```





The dataset is imbalanced. N is the largest type which accounts for over 70%. So an imbalanced remover is needed in this case.

Then two imbalanced_remover are implemented.

```python
def imbalance_remover_1(df):
  # Implement a method to handle class imbalance.
    under = RandomUnderSampler(random_state=42,sampling_strategy='majority')
    sm = SMOTE(random_state=42)

    x = df.copy().drop('Type',axis = 1)
    y = df['Type']

    under.fit(x, y)
    X_res, y_res = under.fit_resample(x, y)

    sm.fit(x,y)
    X_res, y_res = sm.fit_resample(X_res, y_res)

    print('Resampled dataset shape {}'.format(Counter(y_res)))

    return pd.concat([X_res,y_res],axis = 1)

def imbalance_remover_2(df):
  # Implement a method to handle class imbalance.
    sm = SMOTE(random_state=42)

    x = df.copy().drop('Type',axis = 1)
    y = df['Type']

    sm.fit(x,y)
    X_res, y_res = sm.fit_resample(x, y)

    print('Resampled dataset shape {}'.format(Counter(y_res)))

    return pd.concat([X_res,y_res],axis = 1)

data_removed = imbalance_remover_1(data_std)
```

```
Resampled dataset shape Counter({0: 8075, 1: 8075, 2: 8075, 3: 8075, 4: 8075, 5: 8075})
```

The dataset changed to 8075 for each type.

And the AutoEnconder.

```python
X = np.array(data_std.drop('Type',axis = 1))
y = np.array(data_std['Type'])


input_dim = X.shape[1]


encoding_dim = 16
num_epoch = 15
batch_size = 32

input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="tanh",
                activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")(encoder)
decoder = Dense(int(encoding_dim / 2), activation='tanh')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['mae'])
checkpointer = ModelCheckpoint(filepath="SofaSofa_model.h5",
                               verbose=0,
                               save_best_only=True)
history = autoencoder.fit(X_train, X_train,
                          epochs=num_epoch,
                          batch_size=batch_size,
                          shuffle=True,
                          validation_data=(X_test, X_test),
                          verbose=1).history
```

Before the data augment, the metrics of the test set are as follows:

```
confusion_matrix:
[[13715    10    33     5     3     7]
 [   88   116     0     0     8     5]
 [  101     1  1406     2     3    16]
 [   27     0     7  1621     0     2]
 [   14     2     1     0  1111     2]
 [   54     1    16     4     0  1187]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99     13773
           1       0.89      0.53      0.67       217
           2       0.96      0.92      0.94      1529
           3       0.99      0.98      0.99      1657
           4       0.99      0.98      0.99      1130
           5       0.97      0.94      0.96      1262

    accuracy                           0.98     19568
   macro avg       0.96      0.89      0.92     19568
weighted avg       0.98      0.98      0.98     19568

---------------------------------
accuracy_score:
0.9789452166802943
---------------------------------
f1_score:
0.9789452166802943
```

After that, they are:

```
confusion_matrix:
[[1583    1    9    0    0    0]
 [  15 1632    5    0    2    0]
 [  22   17 1567    2    7   26]
 [   1    1    7 1566    1    4]
 [   5    4    1    0 1612    2]
 [   5    1   17    4    5 1566]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1593
           1       0.99      0.99      0.99      1654
           2       0.98      0.95      0.97      1641
           3       1.00      0.99      0.99      1580
           4       0.99      0.99      0.99      1624
           5       0.98      0.98      0.98      1598

    accuracy                           0.98      9690
   macro avg       0.98      0.98      0.98      9690
weighted avg       0.98      0.98      0.98      9690

---------------------------------
accuracy_score:
0.9830753353973168
---------------------------------
f1_score:
0.9830753353973168
```

So I suppose that they work well.

Answer4)

```python
def noise_remover1(df):
    db4 = pywt.Wavelet('db4')
    for i in tqdm(range(len(df['0']))):
        coeffs = pywt.wavedec(df.iloc[i][:-1], db4)
        coeffs[len(coeffs)-1] *= 0
        coeffs[len(coeffs)-2] *= 0
        df.iloc[i][:-1] = pywt.waverec(coeffs,db4)
    return df
data_removed_1 = noise_remover1(data_removed)
```

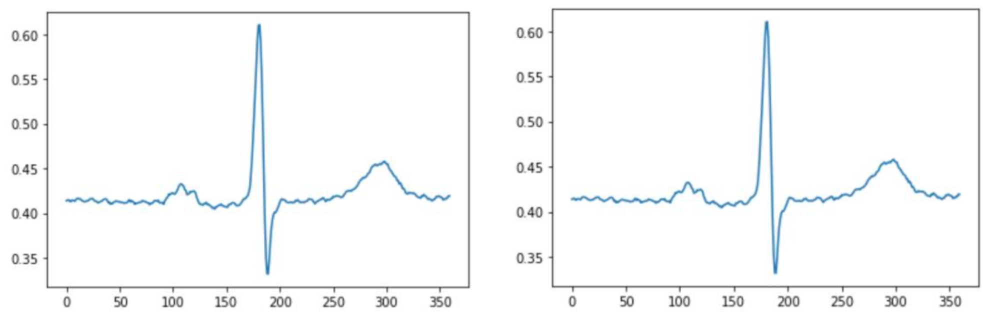
```
100%|██████████| 414078/414078 [10:39<00:00, 647.63it/s]
```

```python
def noise_remover2(df):
  # Take input the dataframe and remove noise from it.
    db8 = pywt.Wavelet('db8')
    threshold = 0.1
    maxlev = pywt.dwt_max_level(len(df.iloc[0][:-1]),db8.dec_len)
    for i in tqdm(range(len(df['0']))):
        coffs=pywt.wavedec(df.iloc[i][:-1],db8,level=maxlev)
        for i in range(1, len(coffs)):
            coffs[i] = pywt.threshold(coffs[i], threshold*max(coffs[i]))
        df.iloc[i][:-1] = pywt.waverec(coffs, db8)
    return df
data_removed_2 = noise_remover2(data_removed)
```

```
100%|██████████| 414078/414078 [11:04<00:00, 623.03it/s]
```

I used WT, named Wavelet transform, which is a new transform analysis method. It inherits and develops the idea of localization of short-time Fourier transform, and at the same time overcomes the shortcomings of window size that does not change with frequency, and can provide a "time-frequency" window that changes with frequency. , is an ideal tool for signal time-frequency analysis and processing. Its main feature is that it can fully highlight the characteristics of certain aspects of the problem through transformation, can analyze the localization of time (space) frequencies, and gradually refine the signal (function) through scaling and translation operations. Time subdivision, frequency subdivision at low frequency, can automatically adapt to the requirements of time-frequency signal analysis so that it can focus on any details of the signal.

Before and after denoising:



As we can see from the picture, The second one is more clear, and some noise has been removed, making the signal smoother, and there is not too much noise-causing over-fitting.

Answer5)

```python
def Model1(X_train, y_train):
  # You can use the sci-kit learn solver but capture any other hyper-parameter settings
  # or model settings in this method

    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.125)
    history = {}
    eval_history = lgb.record_evaluation(history)

    model = LGBMClassifier()

    model.fit(X_train, y_train,
             eval_set= [(X_train, y_train), (X_val, y_val)],
             eval_metric='auc_mu',
             callbacks= [eval_history])

    return model,history
model1,history1 = Model1(X_train, y_train)
```

I chose lightBGM to train and got awesome precision on the test set, which is over 0.98. As for the parameters, just using its default values is good.

```
confusion_matrix:
[[1583    1    9    0    0    0]
 [  15 1632    5    0    2    0]
 [  22   17 1567    2    7   26]
 [   1    1    7 1566    1    4]
 [   5    4    1    0 1612    2]
 [   5    1   17    4    5 1566]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1593
           1       0.99      0.99      0.99      1654
           2       0.98      0.95      0.97      1641
           3       1.00      0.99      0.99      1580
           4       0.99      0.99      0.99      1624
           5       0.98      0.98      0.98      1598

    accuracy                           0.98      9690
   macro avg       0.98      0.98      0.98      9690
weighted avg       0.98      0.98      0.98      9690

---------------------------------
accuracy_score:
0.9830753353973168
---------------------------------
f1_score:
0.9830753353973168
```

Answer6)

```python
def Model2(X_train, y_train):
  # Apply a feed-forward neural network along with hyper-parameter tuning for the same.
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.125)
    iter_num = 300
    history = {'training':{'auc_mu':[],'multi_logloss':[]},
               'valid_1':{'auc_mu':[],'multi_logloss':[]}}
    mlp = MLPClassifier(solver='sgd',
                        activation='relu',
                        alpha=1e-4,
                        hidden_layer_sizes=(50, 50),
                        random_state=42,
                        max_iter=1,
                        verbose=True,
                        learning_rate_init=.001,
                        validation_fraction = 0.125,)


    for _ in range(iter_num):
        mlp.partial_fit(X_train, y_train,classes=[0,1,2,3,4,5])
        history['training']['auc_mu'].append(mlp.score(X_train,y_train))
        history['valid_1']['auc_mu'].append(mlp.score(X_val,y_val))
    history['training']['multi_logloss'] = mlp.loss_curve_
    history['valid_1']['multi_logloss'] = mlp.loss_curve_
    return mlp,history
model2,history2 = Model2(X_train, y_train)
```

I used an MLPclassifier for this question and tuned its hyper-parameters, with alpha being 0.0001, and learning_rate_init being 0.001.

And the result is as follows:

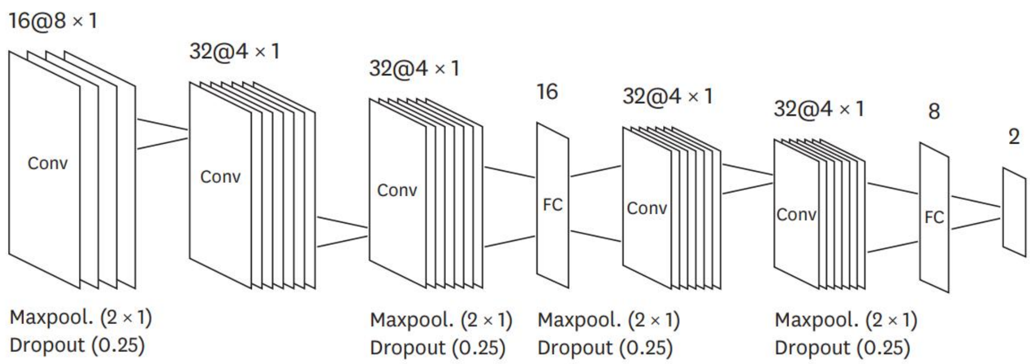
```
y_pred = model2.predict(X_test)
metrics(y_test,y_pred)
```

```
confusion_matrix:
[[1631   12   10    2    0    1]
 [   4 1551    9    0    2    1]
 [  26   21 1542   10    4   37]
 [   1    0    5 1577    2    1]
 [   0    4    4    0 1629    1]
 [   5    6   56    3    0 1533]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1656
           1       0.97      0.99      0.98      1567
           2       0.95      0.94      0.94      1640
           3       0.99      0.99      0.99      1586
           4       1.00      0.99      0.99      1638
           5       0.97      0.96      0.97      1603

    accuracy                           0.98      9690
   macro avg       0.98      0.98      0.98      9690
weighted avg       0.98      0.98      0.98      9690

---------------------------------
accuracy_score:
0.9765737874097007
---------------------------------
f1_score:
0.9765737874097007
```

It works well.

Answer7)

The paper[1] I used is called *Automatic Prediction of Atrial Fibrillation Based on Convolutional Neural Network Using a Short-term Normal Electrocardiogram Signal,* which implemented a convolutional neural network like the following graph:



And I tried to re-build it:

```python
model = Sequential()

model.add(Conv1D(filters=16, kernel_size=8, padding='same',
                 activation='relu',input_shape=(360, 1),strides=1))
model.add(MaxPool1D(pool_size = 2, strides=2))
model.add(Dropout(0.25))

model.add(Conv1D(filters=32, kernel_size=4, padding='same', activation='relu',strides=1))


model.add(Conv1D(filters=64, kernel_size=4, padding='same', activation='relu'))
model.add(MaxPool1D(pool_size = 2, strides=1))
model.add(Dropout(0.25))

model.add(Dense(35,kernel_regularizer=regularizers.l2(0.0001), bias_regularizer=regularizers.l2(0.0001)))

model.add(Conv1D(filters=128, kernel_size=2, padding='same', activation='relu',strides=1))

model.add(Conv1D(filters=256, kernel_size=2, padding='same', activation='relu',strides=1))
model.add(MaxPool1D(pool_size = 2, strides=2))

model.add(Flatten())
model.add(Dropout(0.25))

model.add(Dense(6,kernel_regularizer=regularizers.l2(0.0001), bias_regularizer=regularizers.l2(0.0001)))

model.add(Softmax())

model.summary()

adam = adam_v2.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model.compile(loss='categorical_crossentropy',  optimizer=adam,metrics=['accuracy'])
history = model.fit(train_x, train_y, batch_size=36, epochs=200, verbose=1, validation_data=(test_x, test_y))
```

```
0.9683
Epoch 195/200
667/667 [==============================] - 39s 59ms/step - loss: 0.0617 - accuracy: 0.9905 - val_loss: 0.1843 - val_accuracy:
0.9700
Epoch 196/200
667/667 [==============================] - 38s 57ms/step - loss: 0.0634 - accuracy: 0.9896 - val_loss: 0.1606 - val_accuracy:
0.9727
Epoch 197/200
667/667 [==============================] - 41s 61ms/step - loss: 0.0622 - accuracy: 0.9898 - val_loss: 0.1857 - val_accuracy:
0.9718
Epoch 198/200
667/667 [==============================] - 37s 56ms/step - loss: 0.0642 - accuracy: 0.9885 - val_loss: 0.1916 - val_accuracy:
0.9697
Epoch 199/200
667/667 [==============================] - 40s 59ms/step - loss: 0.0683 - accuracy: 0.9886 - val_loss: 0.1820 - val_accuracy:
0.9695
Epoch 200/200
667/667 [==============================] - 44s 66ms/step - loss: 0.0655 - accuracy: 0.9893 - val_loss: 0.1698 - val_accuracy:
0.9703
```

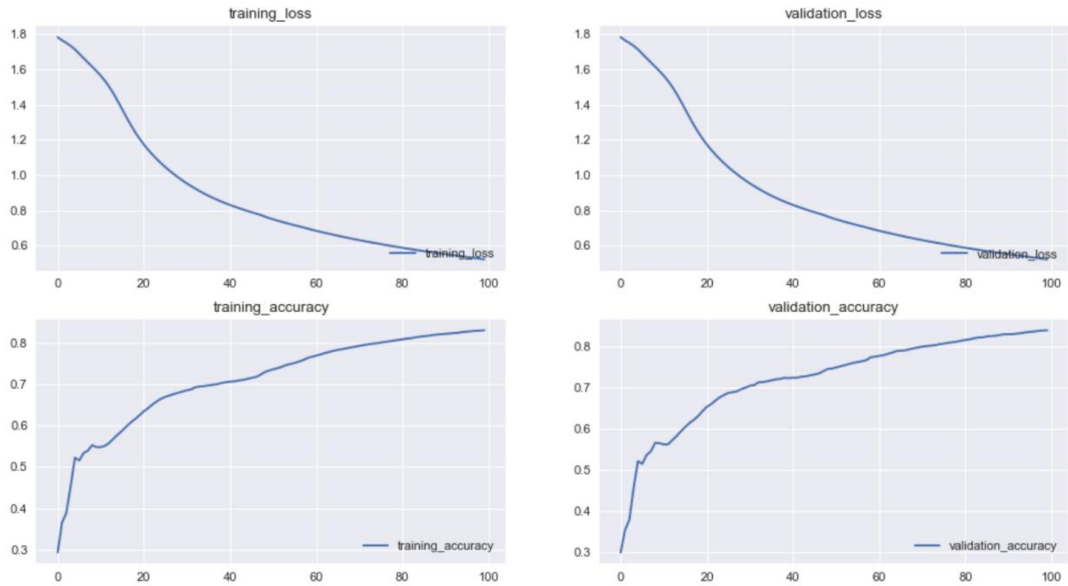
The matrics of the test set are:

```
confusion_matrix:
[[ 935    1    0   22    3   20]
 [   5  994    0    0    0    9]
 [   4    0  951    1    2    0]
 [  32    0    4  955    1    3]
 [  13    3    0    3 1017    3]
 [  26    5    1    5   12  970]]
--------------------------------
classification_report:
              precision    recall  f1-score   support

           0       0.92      0.95      0.94       981
           1       0.99      0.99      0.99      1008
           2       0.99      0.99      0.99       958
           3       0.97      0.96      0.96       995
           4       0.98      0.98      0.98      1039
           5       0.97      0.95      0.96      1019

    accuracy                           0.97      6000
   macro avg       0.97      0.97      0.97      6000
weighted avg       0.97      0.97      0.97      6000

--------------------------------
accuracy_score:
0.9703333333333334
--------------------------------
f1_score:
0.9703333333333334
```

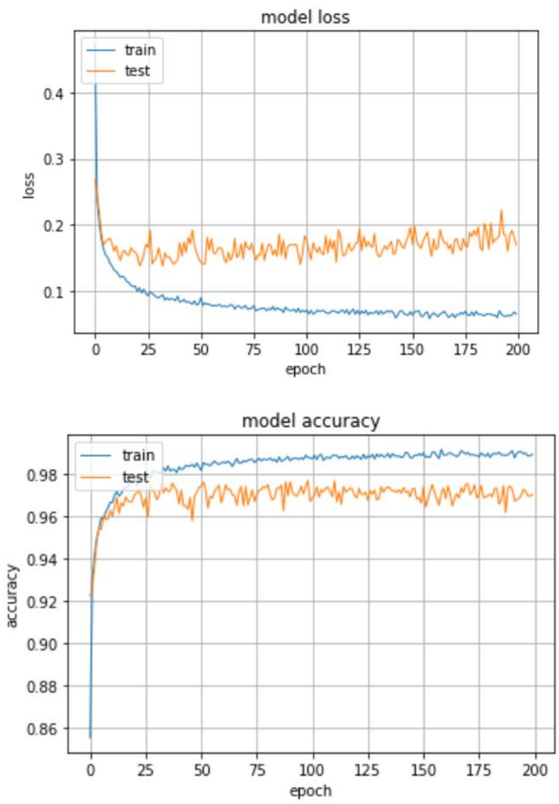
I got 0.97 which is lower than the accuracy of 98.6% in the parer. I suppose that my pre-processing methods are different than the methods referred to in the paper so my accuracy is still needed to be improved further.

Answer8)

MLPclassifier:



CNN in the paper:

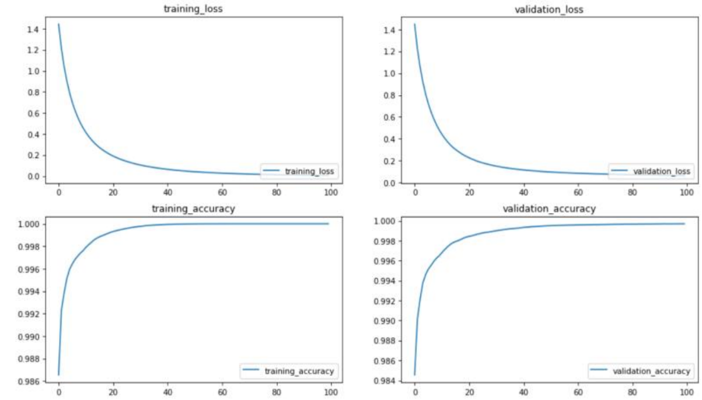


Answer9)

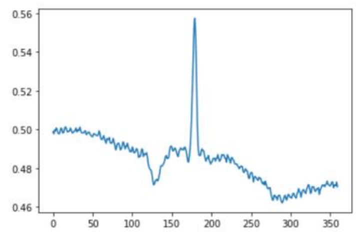
I chose V1 since I guess it is relevant to the heartbeat waves. And the V1 indicator has the largest number among other indicators. I matched V1 to those who have the value of V1 and then filled others with Nan. So for now it has 720 features.

Here is the result of training through the new dataset, it performs better than before.

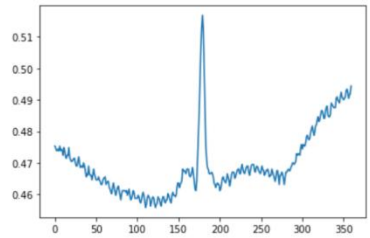
```
confusion_matrix:
[[1601    2    7    0    2    0]
 [   8 1598    3    0    0    0]
 [  19    9 1526    2    8   24]
 [   3    0    7 1587    1    2]
 [   5    0    2    0 1614    0]
 [   5    5   20    6    1 1623]]
--------------------------------
classification_report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      1612
           1       0.99      0.99      0.99      1609
           2       0.98      0.96      0.97      1588
           3       0.99      0.99      0.99      1600
           4       0.99      1.00      0.99      1621
           5       0.98      0.98      0.98      1660

    accuracy                           0.99      9690
   macro avg       0.99      0.99      0.99      9690
weighted avg       0.99      0.99      0.99      9690

--------------------------------
accuracy_score:
0.9854489164086687
--------------------------------
f1_score:
0.9854489164086687
```



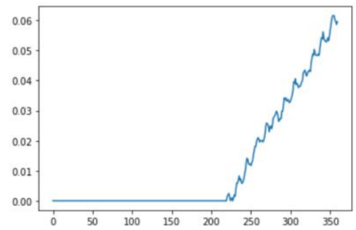
Answer10)



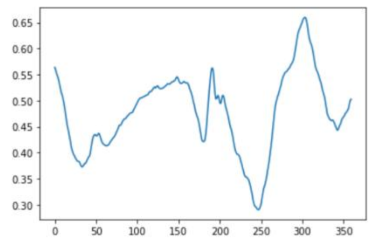
False negatives



False negatives



False positives



False positives

False-negative: similarities with negatives, such as abnormal trends and huge fluctuations, are easily identified as false negatives.

False-positive: I think it may be because the amount of positive data is not complete, the machine cannot fully recognize it, and some fake data is created during data enhancement so that the machine cannot correctly identify the true positive.

Teamwork

Our team has two members: XiangyuGao and RuiLi. We worked together and talked about the assignment. In each part of this project, we did the tasks together. All the codes are programmed after we brainstorm, and there is no division of labor. For example, you do the first part and I do the second part. So we are very good at grasping the whole. Through this mission, we got to know each other better and became more familiar with each other, which provided an emotional foundation for future study and life, and I believe that we will work together and make progress together.

References

[1]Erdenebayar U, Kim H, Park JU, Kang D, Lee KJ. Automatic Prediction of Atrial Fibrillation Based on Convolutional Neural Network Using a Short-term Normal Electrocardiogram Signal. J Korean Med Sci. 2019 Feb;34(7):e64. https://doi.org/10.3346/jkms.2019.34.e64