IMPLEMENTATION CODE

Install Packeges

```
[ ]  !pip install -q keras
```

```
[ ]  import keras
```

```
[ ]  import numpy as np
     import pandas as pd
     import seaborn as sns
     from keras.layers import Dropout
     import matplotlib.pyplot as plt
     from sklearn.metrics import classification_report
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import f1_score
     from sklearn.metrics import confusion_matrix
     from keras.utils.np_utils import to_categorical
     from sklearn.utils import class_weight
     import warnings
     warnings.filterwarnings('ignore')
```

Import Dataset and View the Arch

```
[ ] #dataset
    train_df=pd.read_csv('/content/drive/MyDrive/ECG Data F/mitbih_train.csv',header=None)
    test_df=pd.read_csv('/content/drive/MyDrive/ECG Data F/mitbih_test.csv',header=None)
```

```
[ ] train_df[187]=train_df[187].astype(int)
    equilibre=train_df[187].value_counts()
    print(equilibre)
    train_df.shape
```

```
[ ] plt.figure(figsize=(20,10))
    my_circle=plt.Circle( (0,0), 0.7, color='white')
    plt.pie(equilibre, labels=['normal beat','unknown Beats','Ventricular ectopic beats','Supraventricular ectopic beats','Fusion Beats'],
            colors=['red','green','blue','skyblue','orange'],autopct='%1.1f%%')
    p=plt.gcf()
    p.gca().add_artist(my_circle)
    plt.show()
```

Resample Technique

```
[ ] #resample
    from sklearn.utils import resample
    df_1=train_df[train_df[187]==1]
    df_2=train_df[train_df[187]==2]
    df_3=train_df[train_df[187]==3]
    df_4=train_df[train_df[187]==4]
    df_0=(train_df[train_df[187]==0]).sample(n=20000,random_state=42)

    df_1_upsample=resample(df_1,replace=True,n_samples=20000,random_state=123)
    df_2_upsample=resample(df_2,replace=True,n_samples=20000,random_state=124)
    df_3_upsample=resample(df_3,replace=True,n_samples=20000,random_state=125)
    df_4_upsample=resample(df_4,replace=True,n_samples=20000,random_state=126)

    train_df=pd.concat([df_0,df_1_upsample,df_2_upsample,df_3_upsample,df_4_upsample])
```

Showing Resample Curve

```
[ ] equilibre=train_df[187].value_counts()
    print(equilibre)
```

```
[ ] plt.figure(figsize=(20,10))
    my_circle=plt.Circle( (0,0), 0.7, color='white')
    plt.pie(equilibre, labels=['normal beat','unknown Beats','Ventricular ectopic beats','Supraventricular ectopic beats','Fusion Beats'],
            colors=['red','green','blue','skyblue','orange'],autopct='%1.1f%%')
    p=plt.gcf()
    p.gca().add_artist(my_circle)
    plt.show()
```

Showing the ECG Curve after Resample

```
[ ] c=train_df.groupby(187,group_keys=False).apply(lambda train_df : train_df.sample(1))
```

```
[ ] plt.subplot(1, 5, 1)  # 1 line, 2 rows, index nr 1 (first position in the subplot)
    plt.plot(c.iloc[0,:186])
    plt.subplot(1, 5, 2)  # 1 line, 2 rows, index nr 2 (second position in the subplot)
    plt.plot(c.iloc[1,:186])
    plt.subplot(1, 5, 3)  # 1 line, 2 rows, index nr 2 (second position in the subplot)
    plt.plot(c.iloc[2,:186])
    plt.subplot(1, 5, 4)  # 1 line, 2 rows, index nr 2 (second position in the subplot)
    plt.plot(c.iloc[3,:186])
    plt.subplot(1, 5, 5)  # 1 line, 2 rows, index nr 2 (second position in the subplot)
    plt.plot(c.iloc[4,:186])

    plt.show()
```

Plot the Classes

```
[ ] def plot_hist(class_number,size,min_):
        img=train_df.loc[train_df[187]==class_number].values
        img=img[:,min_:size]
        img_flatten=img.flatten()

        final1=np.arange(min_,size)
        for i in range (img.shape[0]-1):
            tempo1=np.arange(min_,size)
            final1=np.concatenate((final1, tempo1), axis=None)
        print(len(final1))
        print(len(img_flatten))
        plt.hist2d(final1,img_flatten, bins=(80,80),cmap=plt.cm.jet)
        plt.show()
```

```
[ ] plot_hist(0,70,5)
```

```
[ ]  plt.plot(c.iloc[1,:186],color='red')
```

```
[ ] plot_hist(1,50,5)
```

```
[ ] plt.plot(c.iloc[2,:186],color='yellow')
```

```
[ ] plot_hist(2,60,30)
```

```
[ ] plt.plot(c.iloc[3,:186],color='grey')
```

```
[ ] plot_hist(3,60,25)
```

```
[ ] plt.plot(c.iloc[4,:186],color='green')
```

```
[ ] plot_hist(4,50,18)
```

Add Gaussian Noise

```
[ ]  #Gaussian Method
     def add_gaussian_noise(signal):
         noise=np.random.normal(0,0.05,186)
         return (signal+noise)
```

```
[ ]  tempo=c.iloc[0,:186]
     bruiter=add_gaussian_noise(tempo)

     plt.subplot(2,1,1)
     plt.plot(c.iloc[0,:186],color='green')

     plt.subplot(2,1,2)
     plt.plot(bruiter)

     plt.show()
```

Dividing Dataset into Train and Test set

```
[ ]  target_train=train_df[187]
     target_test=test_df[187]
     y_train=to_categorical(target_train)
     y_test=to_categorical(target_test)
```

```
[ ]  X_train=train_df.iloc[:,:186].values
     X_test=test_df.iloc[:,:186].values
     #for i in range(len(X_train)):
     #    X_train[i,:186]= add_gaussian_noise(X_train[i,:186])
     X_train = X_train.reshape(len(X_train), X_train.shape[1],1)
     X_test = X_test.reshape(len(X_test), X_test.shape[1],1)
```

```
[ ]  X_train.shape
```

```
[ ]  #train and test samples
     print(X_train.shape[0], 'training samples')
     print(X_test.shape[0], 'testing samples')
```

1D CNN Proposed Model 01

```
[ ]  #model
     def network(X_train,y_train,X_test,y_test):

         """
         #Model 01
         im_shape=(X_train.shape[1],1)
         inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
         conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
         conv1_1=BatchNormalization()(conv1_1)
         pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
         drop = Dropout(0.2)
         conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
         conv2_1=BatchNormalization()(conv2_1)
         pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
         drop = Dropout(0.2)
         conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
         conv3_1=BatchNormalization()(conv3_1)
         pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
         drop = Dropout(0.2)
         flatten=Flatten()(pool3)
         dense_end1 = Dense(64, activation='relu')(flatten)
         #drop = Dropout(0.2)
         dense_end2 = Dense(32, activation='relu')(dense_end1)
         main_output = Dense(5, activation='softmax', name='main_output')(dense_end2)

         model = Model(inputs= inputs_cnn, outputs=main_output)
         model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])
         history=model.fit(X_train, y_train,epochs=200,batch_size=16,validation_data=(X_test,y_test))
         return(model,history)
        """
```

Proposed Model 02

```
#model 02
 im_shape=(X_train.shape[1],1)
 inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
 conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
 conv1_1=BatchNormalization()(conv1_1)
 pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
 drop = Dropout(0.2)
 conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
 conv2_1=BatchNormalization()(conv2_1)
 pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
 drop = Dropout(0.2)
 conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
 conv3_1=BatchNormalization()(conv3_1)
 pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
 drop = Dropout(0.2)
 flatten=Flatten()(pool3)
 dense_end1 = Dense(64, activation='relu')(flatten)
 drop = Dropout(0.2)
 dense_end2 = Dense(32, activation='relu')(dense_end1)
 drop = Dropout(0.2)
 main_output = Dense(5, activation='softmax', name='main_output')(dense_end2)

 model = Model(inputs= inputs_cnn, outputs=main_output)
 model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])
 history=model.fit(X_train, y_train,epochs=150,batch_size=16,validation_data=(X_test,y_test))
 return(model,history)
```

Proposed Model 03

```
#model 03
 im_shape=(X_train.shape[1],1)
 inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
 conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
 conv1_1=BatchNormalization()(conv1_1)
 pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
 #drop = Dropout(0.2)
 conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
 conv2_1=BatchNormalization()(conv2_1)
 pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
 drop = Dropout(0.2)
 conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
 conv3_1=BatchNormalization()(conv3_1)
 pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
 drop = Dropout(0.2)
 flatten=Flatten()(pool3)
 dense_end1 = Dense(64, activation='relu')(flatten)
 drop = Dropout(0.2)
 dense_end2 = Dense(32, activation='relu')(dense_end1)
 drop = Dropout(0.2)
 main_output = Dense(5, activation='softmax', name='main_output')(dense_end2)

 model = Model(inputs= inputs_cnn, outputs=main_output)
 model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])
 history=model.fit(X_train, y_train,epochs=150,batch_size=16,validation_data=(X_test,y_test))
 return(model,history)
```

Proposed Final Model

```
#Model 04
 im_shape=(X_train.shape[1],1)
 inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
 conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
 conv1_1=BatchNormalization()(conv1_1)
 pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
 drop = Dropout(0.2)
 conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
 conv2_1=BatchNormalization()(conv2_1)
 pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
 drop = Dropout(0.2)
 conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
 conv3_1=BatchNormalization()(conv3_1)
 pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
 drop = Dropout(0.2)
 flatten=Flatten()(pool3)
 dense_end1 = Dense(64, activation='relu')(flatten)
 dense_end2 = Dense(32, activation='relu')(dense_end1)
 main_output = Dense(5, activation='softmax', name='main_output')(dense_end2)

 model = Model(inputs= inputs_cnn, outputs=main_output)
 model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])
 history=model.fit(X_train, y_train,epochs=150,batch_size=16,validation_data=(X_test,y_test))
 return(model,history)
```

Evolution Functions

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

    def evaluate_model(history,X_test,y_test,model):
        scores = model.evaluate((X_test),y_test, verbose=0)
        print("Accuracy: %.2f%%" % (scores[1]*100))
        print('Test Loss:', scores[0])
        print('Test Accuracy:', scores[1])
        #precision_score = precision_score((X_test),y_test)
        #print('Precision score: %f' % precison_score)
        #recall_score = recall_score((X_test),y_test)
        #print('Recall Score: %f' % recall_score)
        #f1_score = f1_score((X_test),y_test)
        #print('f1 score: %f' % f1_score)



        print(history)
        fig1, ax_acc = plt.subplots()
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.title('Model - Accuracy')
        plt.legend(['Training', 'Validation'], loc='lower right')
        plt.show()
```

```
[ ]
        fig2, ax_loss = plt.subplots()
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.title('Model- Loss')
        plt.legend(['Training', 'Validation'], loc='upper right')
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.show()
        target_names=['0','1','2','3','4']

        y_true=[]
        for element in y_test:
            y_true.append(np.argmax(element))
        prediction_proba=model.predict(X_test)
        prediction=np.argmax(prediction_proba,axis=1)
        cnf_matrix = confusion_matrix(y_true, prediction)
```

```
[ ] from keras.layers import Dense, Convolution1D, MaxPool1D, Flatten, Dropout
    from keras.layers import Input
    from keras.models import Model
    from keras.layers.normalization import BatchNormalization
    import keras
    #from keras.callbacks import EarlyStopping, ModelCheckpoint

    model,history=network(X_train,y_train,X_test,y_test)
```

```
[ ] model.summary()
```

Confusion Matrix and Accuracy

```
[ ] evaluate_model(history,X_test,y_test,model)
    y_pred=model.predict(X_test)


    print(y_pred)
    print("Result from real time data included in testing dataset:")
    Y_pred_classes = np.argmax(y_pred,axis = 1)
    print(Y_pred_classes[0])
```

```
[ ] #confusion matrix
    from sklearn.metrics import f1_score, precision_score, recall_score, classification_report, confusion_matrix
    from sklearn import metrics

    import itertools
    def plot_confusion_matrix(cm, classes,
                              normalize=False,
                              title='Confusion matrix',
                              cmap=plt.cm.Blues):
        """
        This function prints and plots the confusion matrix.
        Normalization can be applied by setting `normalize=True`.
        """
        if normalize:
            cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
            print("Normalized confusion matrix")
        else:
            print('Confusion matrix, without normalization')
```

```
[ ]
        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=45)
        plt.yticks(tick_marks, classes)

        fmt = '.2f' if normalize else 'd'
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')

    # Compute confusion matrix
    cnf_matrix = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
    np.set_printoptions(precision=2)

    #f1_score = cnf_matrix
    #print('f1 score' %f1_score)
    # Plot non-normalized confusion matrix
    plt.figure(figsize=(10, 10))
    plot_confusion_matrix(cnf_matrix, classes=['N', 'S', 'V', 'F', 'Q'],normalize=True,
                          title='Confusion matrix, with normalization')
    plt.show()
```