ДОДАТОК А

Програма

```
import argparse

parser = argparse.ArgumentParser()

def add_argument_group(name):
    arg = parser.add_argument_group(name)
    return arg

misc_arg = add_argument_group('misc')
misc_arg.add_argument('--input_size', type=int, default = 512)

graph_arg = add_argument_group('graph')
graph_arg.add_argument('--filter_length', type=int, default = 32)
graph_arg.add_argument('--kernel_size', type=int, default = 9)
graph_arg.add_argument('--drop_rate', type=float, default = 0.3)

train_arg = add_argument_group('train')
train_arg.add_argument('--epochs', type=int, default = 20)
train_arg.add_argument('--batch', type=int, default = 128)
train_arg.add_argument('--patience', type=int, default = 80)
train_arg.add_argument('--min_lr', type=float, default = 0.00005)
train_arg.add_argument('--checkpoint_path', type=str, default = None)
train_arg.add_argument('--resume_epoch', type=int)
train_arg.add_argument('--ensemble', type=bool, default = False)

def get_config():
    config, unparsed = parser.parse_known_args()
    return config
```

```
from __future__ import division, print_function

from keras import backend as K

from keras.layers import Input, Conv1D, Dense, add, Dropout, MaxPooling1D,
Activation, BatchNormalization, \
    Lambda
from keras.models import Model
from keras.optimizers import Adam

from config import get_config


def ECG_model(config):
    """
    reference to codes at
    https://github.com/awni/ecg/blob/master/ecg/network.py
    and
    https://github.com/physhik/ecg-mit-bih/blob/master/src/graph.py
    """


    def first_conv_block(inputs, config):
        layer = Conv1D(filters=config.filter_length,
                       kernel_size=config.kernel_size,
                       padding='same',
                       strides=1,
                       kernel_initializer='he_normal')(inputs)
        layer = BatchNormalization()(layer)
        layer = Activation('relu')(layer)

        shortcut = MaxPooling1D(pool_size=1,
                                strides=1)(layer)

        layer = Conv1D(filters=config.filter_length,
                       kernel_size=config.kernel_size,
                       padding='same',
                       strides=1,
                       kernel_initializer='he_normal')(layer)
        layer = BatchNormalization()(layer)
        layer = Activation('relu')(layer)
        layer = Dropout(config.drop_rate)(layer)
        layer = Conv1D(filters=config.filter_length,
                       kernel_size=config.kernel_size,
                       padding='same',
                       strides=1,
                       kernel_initializer='he_normal')(layer)
        layer = add([shortcut, layer])
        return layer

    def main_loop_blocks(layer, config):
        filter_length = config.filter_length
        n_blocks = 2
        for block_index in range(n_blocks):
            def zeropad(x):
                """
                zeropad and zeropad_output_shapes are from
                https://github.com/awni/ecg/blob/master/ecg/network.py
                """
                y = K.zeros_like(x)
                return K.concatenate([x, y], axis=2)
```

```
            def zeropad_output_shape(input_shape):
                shape = list(input_shape)
                assert len(shape) == 2
                shape[2] *= 2
                return tuple(shape)

            subsample_length = 16 if block_index % 2 == 0 else 8
            shortcut = MaxPooling1D(pool_size=subsample_length)(layer)

            if block_index % 4 == 0 and block_index > 0:
                shortcut = Lambda(zeropad,
output_shape=zeropad_output_shape)(shortcut)
                filter_length *= 2

            layer = BatchNormalization()(layer)
            layer = Activation('relu')(layer)
            layer = Conv1D(filters=filter_length,
                           kernel_size=config.kernel_size,
                           padding='same',
                           strides=subsample_length,
                           kernel_initializer='he_normal')(layer)
            layer = BatchNormalization()(layer)
            layer = Activation('relu')(layer)
            layer = Dropout(config.drop_rate)(layer)
            layer = Conv1D(filters=filter_length,
                           kernel_size=config.kernel_size,
                           padding='same',
                           strides=1,
                           kernel_initializer='he_normal')(layer)
            layer = add([shortcut, layer])
        return layer

    def output_block(layer, config):
        filter_length = config.filter_length
        layer = Conv1D(filters=filter_length,
                       kernel_size=16,
                       padding='same',
                       strides=4,
                       kernel_initializer='he_normal')(layer)
        from keras.layers.wrappers import TimeDistributed
        layer = BatchNormalization()(layer)
        layer = Activation('relu')(layer)
        outputs = TimeDistributed(Dense(len_classes,
activation='softmax'))(layer)
        model = Model(inputs=inputs, outputs=outputs)

        adam = Adam(lr=0.5, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False)
        model.compile(optimizer=adam,
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])
```

```
inputs = Input(shape=(config.input_size, 1), name='input')
layer = first_conv_block(inputs, config)
layer = main_loop_blocks(layer, config)
return output_block(layer, config)


    def main_loop_blocks(layer, config):
        filter_length = config.filter_length
        n_blocks = 7
        for block_index in range(n_blocks):
            def zeropad(x):
                """
                zeropad and zeropad_output_shapes are from
                https://github.com/awni/ecg/blob/master/ecg/network.py
                """
                y = K.zeros_like(x)
                return K.concatenate([x, y], axis=2)

            def zeropad_output_shape(input_shape):
                shape = list(input_shape)
                assert len(shape) == 3
                shape[2] *= 2
                return tuple(shape)

            subsample_length = 4 if block_index % 2 == 0 else 1
            shortcut = MaxPooling1D(pool_size=subsample_length)(layer)
            if block_index % 4 == 0 and block_index > 0:
                shortcut = Lambda(zeropad,
output_shape=zeropad_output_shape)(shortcut)
                filter_length *= 2

            layer = BatchNormalization()(layer)
            layer = Activation('relu')(layer)
            layer = Conv1D(filters=filter_length,
                           kernel_size=config.kernel_size,
                           padding='same',
                           strides=subsample_length,
                           kernel_initializer='he_normal')(layer)
            layer = BatchNormalization()(layer)
            layer = Activation('relu')(layer)
            layer = Dropout(config.drop_rate)(layer)
            layer = Conv1D(filters=filter_length,
                           kernel_size=config.kernel_size,
                           padding='same',
                           strides=1,
                           kernel_initializer='he_normal')(layer)
            layer = add([shortcut, layer])
        return layer

    def output_block(layer, config):
        filter_length = config.filter_length
        layer = Conv1D(filters=filter_length,
                       kernel_size=16,
                       padding='same',
                       strides=2,
                       kernel_initializer='he_normal')(layer)
        from keras.layers.wrappers import TimeDistributed
        layer = BatchNormalization()(layer)
        layer = Activation('relu')(layer)
        outputs = TimeDistributed(Dense(len_classes,
activation='softmax'))(layer)
        model = Model(inputs=inputs, outputs=outputs)
```

```
        adam = Adam(lr=0.5, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False)
        model.compile(optimizer=adam,
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])
        model.summary()
        return model

    classes = ['n', 'a', 'p', 'r', 'l']
    len_classes = len(classes)

    inputs = Input(shape=(config.input_size, 1), name='input')
    layer = first_conv_block(inputs, config)
    layer = main_loop_blocks(layer, config)
    return output_block(layer, config)
```

```
    def main_loop_blocks(layer, config):
        filter_length = config.filter_length
        n_blocks = 7
        for block_index in range(n_blocks):
            def zeropad(x):
                """
                zeropad and zeropad_output_shapes are from
                https://github.com/awni/ecg/blob/master/ecg/network.py
                """
                y = K.zeros_like(x)
                return K.concatenate([x, y], axis=2)

            def zeropad_output_shape(input_shape):
                shape = list(input_shape)
                assert len(shape) == 3
                shape[2] *= 2
                return tuple(shape)

            subsample_length = 4 if block_index % 2 == 0 else 1
            shortcut = MaxPooling1D(pool_size=subsample_length)(layer)
            if block_index % 4 == 0 and block_index > 0:
                shortcut = Lambda(zeropad,
output_shape=zeropad_output_shape)(shortcut)
                filter_length *= 2

            layer = BatchNormalization()(layer)
            layer = Activation('relu')(layer)
            layer = Conv1D(filters=filter_length,
                           kernel_size=config.kernel_size,
                           padding='same',
                           strides=subsample_length,
                           kernel_initializer='he_normal')(layer)
            layer = BatchNormalization()(layer)
            layer = Activation('relu')(layer)
            layer = Dropout(config.drop_rate)(layer)
            layer = Conv1D(filters=filter_length,
                           kernel_size=config.kernel_size,
                           padding='same',
                           strides=1,
                           kernel_initializer='he_normal')(layer)
            layer = add([shortcut, layer])
        return layer

    def output_block(layer, config):
        filter_length = config.filter_length
        layer = Conv1D(filters=filter_length,
                       kernel_size=16,
                       padding='same',
                       strides=2,
                       kernel_initializer='he_normal')(layer)
        from keras.layers.wrappers import TimeDistributed
        layer = BatchNormalization()(layer)
        layer = Activation('relu')(layer)
        outputs = TimeDistributed(Dense(len_classes,
activation='softmax'))(layer)
        model = Model(inputs=inputs, outputs=outputs)
```

```
        adam = Adam(lr=0.5, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False)
        model.compile(optimizer=adam,
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])
        model.summary()
        return model

    classes = ['n', 'a', 'p', 'r', 'l']
    len_classes = len(classes)

    inputs = Input(shape=(config.input_size, 1), name='input')
    layer = first_conv_block(inputs, config)
    layer = main_loop_blocks(layer, config)
    return output_block(layer, config)
```

```
from __future__ import division, print_function

from keras.models import Model
from keras.layers import Input, Conv1D, Dense, add, Flatten,
Dropout,MaxPooling1D, Activation, BatchNormalization, Lambda,
GlobalAveragePooling2D
from keras import backend as K
from keras.optimizers import Adam
from keras.layers.wrappers import TimeDistributed

from config import get_config


def ECG_model(config):
     """
    reference to codes at
    https://github.com/awni/ecg/blob/master/ecg/network.py
    and
    https://github.com/physhik/ecg-mit-bih/blob/master/src/graph.py
    """
    def first_conv_block(inputs, config):
        layer = Conv1D(filters=config.filter_length,
               kernel_size=config.kernel_size,
               padding='same',
               strides=1,
               kernel_initializer='he_normal')(inputs)
        layer = BatchNormalization()(layer)
        layer = Activation('relu')(layer)

        shortcut = MaxPooling1D(pool_size=1,
                     strides=1)(layer)

        layer =  Conv1D(filters=config.filter_length,
               kernel_size=config.kernel_size,
               padding='same',
               strides=1,
               kernel_initializer='he_normal')(layer)
        layer = BatchNormalization()(layer)
```

```
            layer = Activation('relu')(layer)
            layer = Dropout(config.drop_rate)(layer)
            layer =  Conv1D(filters=config.filter_length,
                            kernel_size=config.kernel_size,
                            padding='same',
                            strides=1,
                            kernel_initializer='he_normal')(layer)
            layer = add([shortcut, layer])
            return layer

        def main_loop_blocks(layer, config):
            filter_length = config.filter_length
            n_blocks = 15
            for block_index in range(n_blocks):
                def zeropad(x):
                    """
                    zeropad and zeropad_output_shapes are from
                    https://github.com/awni/ecg/blob/master/ecg/network.py
                    """
                    y = K.zeros_like(x)
                    return K.concatenate([x, y], axis=2)

                def zeropad_output_shape(input_shape):
                    shape = list(input_shape)
                    assert len(shape) == 3
                    shape[2] *= 2
                    return tuple(shape)

                subsample_length = 2 if block_index % 2 == 0 else 1
                shortcut = MaxPooling1D(pool_size=subsample_length)(layer)

                if block_index % 4 == 0 and block_index > 0 :
                    shortcut = Lambda(zeropad,
output_shape=zeropad_output_shape)(shortcut)
                    filter_length *= 2

                layer = BatchNormalization()(layer)
                layer = Activation('relu')(layer)
                layer =  Conv1D(filters= filter_length,
                                kernel_size=config.kernel_size,
                                padding='same',
                                strides=subsample_length,
                                kernel_initializer='he_normal')(layer)
                layer = BatchNormalization()(layer)
                layer = Activation('relu')(layer)
                layer = Dropout(config.drop_rate)(layer)
                layer =  Conv1D(filters= filter_length,
                                kernel_size=config.kernel_size,
                                padding='same',
                                strides= 1,
                                kernel_initializer='he_normal')(layer)
                layer = add([shortcut, layer])
            return layer

        def output_block(layer, config):
            filter_length = config.filter_length
            layer = Conv1D(filters=filter_length,
```

```
                        kernel_size=16,
                        padding='same',
                        strides=2,
                        kernel_initializer='he_normal')(layer)
        layer = BatchNormalization()(layer)
        layer = Activation('relu')(layer)
        outputs = TimeDistributed(Dense(len_classes,
activation='softmax'))(layer)
        model = Model(inputs=inputs, outputs=outputs)

        adam = Adam(lr=0.5, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False)
        model.compile(optimizer= adam,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
        model.summary()
        return model

    classes = ['n','a','p','r','l']
    len_classes = len(classes)

    inputs = Input(shape=(config.input_size, 1), name='input')
    layer = first_conv_block(inputs, config)
    layer = main_loop_blocks(layer, config)
    return output_block(layer, config)
```

```
def dataprocess(nclasses):
    import scipy.io as scio
    import numpy as np
    from sklearn.model_selection import train_test_split

    #load data
    def loaddata(label_class, datafile):
        data = scio.loadmat(datafile)
        _, _, _,  key = data.keys()
        data = data[key]
        data = data[14:526, :]
        m, n = np.shape(data)
        label = np.zeros(shape=[n, nclasses], dtype=float)

        label[:, label_class] = 1
        data = np.transpose(data)
        return data, label

    #concate data
    n_data, n_label = loaddata(0, 'dataset/normalrr1.mat')
    x, y = n_data, n_label
    a_data, a_label = loaddata(1, 'dataset/apcrr2.mat')
    x = np.concatenate((x, a_data))
    y = np.concatenate((y, a_label))
    p_data, p_label = loaddata(2, 'dataset/pvcrr2.mat')
    x = np.concatenate((x, p_data))
    y = np.concatenate((y, p_label))
    r_data, r_label = loaddata(3, 'dataset/rbbbrr1.mat')

       x = np.concatenate((x, r_data))
       y = np.concatenate((y, r_label))
       l_data, l_label = loaddata(4, 'dataset/lbbbrr1.mat')
       x = np.concatenate((x, l_data))
       y = np.concatenate((y, l_label))

       #produce 20 trainingset and a test set
       train_x, train_y = dict(), dict()
       x_val, y_val = list(), list()
       x, x_val, y, y_val = train_test_split(x, y, test_size=0.1,
  random_state=None)
       for i in range(20):
           train_x[i], train_y[i] = list(), list()
           train_x[i], x_test, train_y[i], y_test = train_test_split(x, y,
  test_size=0.1, random_state=None)
       print('datatype{},datashape{}'.format(type(a_data), np.shape(a_data)))
       print('labeltype{},labelshape{}'.format(type(a_label),
  np.shape(a_label)))
       return train_x, train_y, x_val, y_val


  if __name__ == '__main__':
       dataprocess(5)
```

```
from __future__ import division, print_function

import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
from keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard,
ReduceLROnPlateau, LearningRateScheduler
from sklearn.model_selection import train_test_split

from config import get_config
from data import dataprocess

def train(config, xtr, ytr, xte, yte, model_number):
    if model_number == 9:
        from graph_9 import ECG_model
    elif model_number == 19:
        from graph_19 import ECG_model
    else:
        from graph_34 import ECG_model

    classes = ['n','a','p','r','l']
    print('X', np.shape(xtr))
    print('y', np.shape(ytr))
    print('X type', type(xtr))
    print('y type', type(ytr))
    xtr = np.expand_dims(xtr, axis=2) # reshape data according to Keras'
requirements
    xte = np.expand_dims(xte, axis=2)
    (m, n) = ytr.shape
```

```
    ytr = ytr.reshape((m, 1, n))
    (mvl, nvl) = yte.shape
    yte = yte.reshape((mvl, 1, nvl))
    print('xte', np.shape(xte))
    print('yte', np.shape(yte))

    model = ECG_model(config)
    initial_epoch = 0
    mkdir_recursive('models')
    callbacks = [
            EarlyStopping(patience = config.patience, verbose=1),
            ReduceLROnPlateau(factor = 0.2, patience = 3, min_lr = 0.01,
verbose=1),
            TensorBoard( log_dir='./logs', histogram_freq=0, write_graph =
True, write_grads=False, write_images=True),
            ModelCheckpoint('models/{}-latest.hdf5'.format(config.feature),
monitor='val_loss', save_best_only=False, verbose=2, period=10)
    ]

    model.fit(xtr, ytr,
              validation_data=(xte, yte),
              epochs=config.epochs,
              batch_size=config.batch,
              callbacks=callbacks,
              initial_epoch=initial_epoch)


def main(config):
    print('feature:', config.feature)
    print('drop rate:', config.drop_rate)
    print('split', config.split)
    print('input_size', config.input_size)
    for i in range(20):
        xtr, ytr, xte, yte = dataprocess(5)
        print('training set', i)
        train(config, xtr[i], ytr[i], xte, yte, 9)
        train(config, xtr[i], ytr[i], xte, yte, 19)
        train(config, xtr[i], ytr[i], xte, yte, 34)


if __name__=="__main__":
    config = get_config()
    main(config)
```

ВІДОМІСТЬ АТЕСТАЦІЙНОЇ РОБОТИ

Позначення	Найменування	Дод. Відомості
	Текстові документи	
1	Пояснювальна записка	67 с.
2	Презентаційний матеріал	19 с.
	Інші документи	
3	Роздруківки програм	11 с.
4	Рецензія	2 с.
5	Відгук керівника	1 с.

					Класифікація медичних часових рядів за допомогою методів машинного навчання			
Змін	**Арк.**	**Номер докум.**	**Підп.**	**Дата**				
Розроб.		**Гайова А.Ю.**			**(Тема роботи) Відомість атестаційної роботи**		**Аркуш**	**Аркушів**
Перевір.		**Кіріченко Л.О.**						
Н. контр.		**Сидоров М.В.**				**ХНУРЕ Кафедра ПМ**		
Затв.		**Тевяшев А.Д.**						