The validation scheme is based on seq2seq-rnn-with-gru, and cleaned data is from data-without-drift and Kalman filter is from https://www.kaggle.com/teejmahal20/single-model-lgbm-kalman-filter and the added feature is from wavenet-with-1-more-feature. I also used ragnar's data in this version clean-kalman. The Wavenet is based on https://github.com/philipperemy/keras-tcn, https://github.com/peustr/wavenet and https://github.com/basveeling/wavenet and also https://www.kaggle.com/wimwim/wavenet-lstm. If any refrence is not mentioned it was not intentional, please add them in comments.

Previous versions were mainly based on https://www.kaggle.com/wimwim/wavenet-lstm

In [1]:
```
!pip install --no-warn-conflicts -q tensorflow-addons
```

In [2]:
```
from tensorflow.keras.layers import (TimeDistributed, Dropout, BatchNormalization, Flatten, Convolution1D, Activation, Input, Dense, LSTM, Lambda, Bidirectional,
                                     Add, AveragePooling1D, Multiply, GRU, GRUCell, LSTMCell, SimpleRNNCell, SimpleRNN, TimeDistributed, RNN,
                                     RepeatVector, Conv1D, MaxPooling1D, Concatenate, GlobalAveragePooling1D, UpSampling1D)
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, Callback, ReduceLROnPlateau, LearningRateScheduler
from tensorflow.keras.losses import binary_crossentropy, categorical_crossentropy, mean_squared_error
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow.keras.utils import Sequence, to_categorical
from tensorflow.keras import losses, models, optimizers
from tensorflow.keras import backend as K
import tensorflow as tf
from typing import List, NoReturn, Union, Tuple, Optional, Text, Generic, Callable, Dict
from sklearn.metrics import f1_score, cohen_kappa_score, mean_squared_error
from logging import getLogger, Formatter, StreamHandler, FileHandler, INFO
from sklearn.model_selection import KFold, GroupKFold
from tqdm import tqdm_notebook as tqdm
from contextlib import contextmanager
from joblib import Parallel, delayed
from IPython.display import display
from sklearn import preprocessing
import tensorflow_addons as tfa
import scipy.stats as stats
import random as rn
import pandas as pd
import numpy as np
import scipy as sp
import itertools
import warnings
import time
import pywt
import os
import gc


warnings.simplefilter('ignore')
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', 1000)
pd.set_option('display.max_rows', 500)
%matplotlib inline
```

In [3]:
```
EPOCHS=120
NNBATCHSIZE=20
BATCHSIZE = 4000
SEED = 321
SELECT = True
SPLITS = 5
LR = 0.001
fe_config = [
    (True, 4000),
]
```

In [4]:
```
def init_logger():
    handler = StreamHandler()
    handler.setLevel(INFO)
    handler.setFormatter(Formatter(LOGFORMAT))
    fh_handler = FileHandler('{}.log'.format(MODELNAME))
    fh_handler.setFormatter(Formatter(LOGFORMAT))
    logger.setLevel(INFO)
    logger.addHandler(handler)
    logger.addHandler(fh_handler)
```

In [5]:
```
@contextmanager
def timer(name : Text):
    t0 = time.time()
    yield
    logger.info(f'[{name}] done in {time.time() - t0:.0f} s')

COMPETITION = 'ION-Switching'
logger = getLogger(COMPETITION)
LOGFORMAT = '%(asctime)s %(levelname)s %(message)s'
MODELNAME = 'WaveNet'
```

In [6]:
```
def seed_everything(seed : int) -> NoReturn :

    rn.seed(seed)
    np.random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    tf.random.set_seed(seed)

seed_everything(SEED)
```

In [7]:
```
def read_data(base : os.path.abspath) -> Tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]:

    train = pd.read_csv('/kaggle/input/clean-kalman/train_clean_kalman.csv', dtype={'time': np.float32, 'signal': np.float32, 'open_channels':np.int32})
    test  = pd.read_csv('/kaggle/input/clean-kalman/test_clean_kalman.csv', dtype={'time': np.float32, 'signal': np.float32})
    sub  = pd.read_csv('/kaggle/input/liverpool-ion-switching/sample_submission.csv', dtype={'time': np.float32})

    return train, test, sub
```

In [8]:
```
def batching(df : pd.DataFrame,
             batch_size : int) -> pd.DataFrame :

    df['group'] = df.groupby(df.index//batch_size, sort=False)['signal'].agg(['ngroup']).values
    df['group'] = df['group'].astype(np.uint16)

    return df
```

In [9]:
```
def reduce_mem_usage(df: pd.DataFrame,
                     verbose: bool = True) -> pd.DataFrame:

    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2

    for col in df.columns:
        col_type = df[col].dtypes

        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()

            if str(col_type)[:3] == 'int':

                if (c_min > np.iinfo(np.int32).min
                      and c_max < np.iinfo(np.int32).max):
                    df[col] = df[col].astype(np.int32)
                elif (c_min > np.iinfo(np.int64).min
                      and c_max < np.iinfo(np.int64).max):
                    df[col] = df[col].astype(np.int64)
            else:
                if (c_min > np.finfo(np.float16).min
                        and c_max < np.finfo(np.float16).max):
                    df[col] = df[col].astype(np.float16)
                elif (c_min > np.finfo(np.float32).min
                      and c_max < np.finfo(np.float32).max):
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)

    end_mem = df.memory_usage().sum() / 1024**2
    reduction = (start_mem - end_mem) / start_mem

    msg = f'Mem. usage decreased to {end_mem:5.2f} MB ({reduction * 100:.1f} % reduction)'
    if verbose:
        print(msg)

    return df
```

In [10]:
```
def lag_with_pct_change(df : pd.DataFrame,
                        shift_sizes : Optional[List]=[1, 2],
                        add_pct_change : Optional[bool]=False,
                        add_pct_change_lag : Optional[bool]=False) -> pd.DataFrame:

    for shift_size in shift_sizes:
        df['signal_shift_pos_'+str(shift_size)] = df.groupby('group')['signal'].shift(shift_size).fillna(0)
        df['signal_shift_neg_'+str(shift_size)] = df.groupby('group')['signal'].shift(-1*shift_size).fillna(0)

    if add_pct_change:
        df['pct_change'] = df['signal'].pct_change()
        if add_pct_change_lag:
            for shift_size in shift_sizes:
                df['pct_change_shift_pos_'+str(shift_size)] = df.groupby('group')['pct_change'].shift(shift_size).fillna(0)
                df['pct_change_shift_neg_'+str(shift_size)] = df.groupby('group')['pct_change'].shift(-1*shift_size).fillna(0)
    return df
```

In [11]:
```
def run_feat_enginnering(df : pd.DataFrame,
                         create_all_data_feats : bool,
                         batch_size : int) -> pd.DataFrame:

    df = batching(df, batch_size=batch_size)
    if create_all_data_feats:
        df = lag_with_pct_change(df, [1, 2, 3],  add_pct_change=False, add_pct_change_lag=False)
    df['signal_2'] = df['signal'] ** 2
    return df
```

In [12]:
```
def feature_selection(df : pd.DataFrame,
                      df_test : pd.DataFrame) -> Tuple[pd.DataFrame , pd.DataFrame, List]:
    use_cols = [col for col in df.columns if col not in ['index','group', 'open_channels', 'time']]
    df = df.replace([np.inf, -np.inf], np.nan)
    df_test = df_test.replace([np.inf, -np.inf], np.nan)
    for col in use_cols:
        col_mean = pd.concat([df[col], df_test[col]], axis=0).mean()
        df[col] = df[col].fillna(col_mean)
        df_test[col] = df_test[col].fillna(col_mean)

    gc.collect()
    return df, df_test, use_cols
```

In [13]:
```
def augment(X: np.array, y:np.array) -> Tuple[np.array, np.array]:

    X = np.vstack((X, np.flip(X, axis=1)))
    y = np.vstack((y, np.flip(y, axis=1)))

    return X, y
```

In [14]:
```
def run_cv_model_by_batch(train : pd.DataFrame,
                          test : pd.DataFrame,
                          splits : int,
                          batch_col : Text,
                          feats : List,
                          sample_submission: pd.DataFrame,
                          nn_epochs : int,
                          nn_batch_size : int) -> NoReturn:
    seed_everything(SEED)
    K.clear_session()
    config = tf.compat.v1.ConfigProto(intra_op_parallelism_threads=1,inter_op_parallelism_threads=1)
    sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=config)
    tf.compat.v1.keras.backend.set_session(sess)
    oof_ = np.zeros((len(train), 11))
    preds_ = np.zeros((len(test), 11))
    target = ['open_channels']
    group = train['group']
    kf = GroupKFold(n_splits=5)
    splits = [x for x in kf.split(train, train[target], group)]

    new_splits = []
    for sp in splits:
        new_split = []
        new_split.append(np.unique(group[sp[0]]))
        new_split.append(np.unique(group[sp[1]]))
        new_split.append(sp[1])
        new_splits.append(new_split)

    tr = pd.concat([pd.get_dummies(train.open_channels), train[['group']]], axis=1)

    tr.columns = ['target_'+str(i) for i in range(11)] + ['group']
    target_cols = ['target_'+str(i) for i in range(11)]
    train_tr = np.array(list(tr.groupby('group').apply(lambda x: x[target_cols].values))).astype(np.float32)
    train = np.array(list(train.groupby('group').apply(lambda x: x[feats].values)))
    test = np.array(list(test.groupby('group').apply(lambda x: x[feats].values)))

    for n_fold, (tr_idx, val_idx, val_orig_idx) in enumerate(new_splits[0:], start=0):
        train_x, train_y = train[tr_idx], train_tr[tr_idx]
        valid_x, valid_y = train[val_idx], train_tr[val_idx]

        if n_fold < 2:
            train_x, train_y = augment(train_x, train_y)

        gc.collect()
        shape_ = (None, train_x.shape[2])
        model = Classifier(shape_)
        cb_lr_schedule = LearningRateScheduler(lr_schedule)
        cb_prg = tfa.callbacks.TQDMProgressBar(leave_epoch_progress=False,leave_overall_progress=False, show_epoch_progress=False,show_overall_progress=True)
        model.fit(train_x,train_y,
                  epochs=nn_epochs,
                  callbacks=[cb_prg, cb_lr_schedule, MacroF1(model, valid_x,valid_y)],
                  batch_size=nn_batch_size,verbose=0,
                  validation_data=(valid_x,valid_y))
        preds_f = model.predict(valid_x)
        f1_score_ = f1_score(np.argmax(valid_y, axis=2).reshape(-1),  np.argmax(preds_f, axis=2).reshape(-1), average = 'macro')
        logger.info(f'Training fold {n_fold + 1} completed. macro f1 score : {f1_score_ :1.5f}')
        preds_f = preds_f.reshape(-1, preds_f.shape[-1])
        oof_[val_orig_idx,:] += preds_f
        te_preds = model.predict(test)
        te_preds = te_preds.reshape(-1, te_preds.shape[-1])
        preds_ += te_preds / SPLITS
    f1_score_ =f1_score(np.argmax(train_tr, axis=2).reshape(-1),  np.argmax(oof_, axis=1), average = 'macro')
    logger.info(f'Training completed. oof macro f1 score : {f1_score_:1.5f}')
    sample_submission['open_channels'] = np.argmax(preds_, axis=1).astype(int)
    sample_submission.to_csv('submission.csv', index=False, float_format='%.4f')
    display(sample_submission.head())
    np.save('oof.npy', oof_)
    np.save('preds.npy', preds_)

    return
```

In [15]:
```
def lr_schedule(epoch):
    if epoch < 40:
        lr = LR
    elif epoch < 50:
        lr = LR / 3
    elif epoch < 60:
        lr = LR / 6
    elif epoch < 75:
        lr = LR / 9
    elif epoch < 85:
        lr = LR / 12
    elif epoch < 100:
        lr = LR / 15
    else:
        lr = LR / 50
    return lr
```

In [16]:
```
class Mish(tf.keras.layers.Layer):

    def __init__(self, **kwargs):
        super(Mish, self).__init__(**kwargs)
        self.supports_masking = True

    def call(self, inputs):
        return inputs * K.tanh(K.softplus(inputs))

    def get_config(self):
        base_config = super(Mish, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))

    def compute_output_shape(self, input_shape):
        return input_shape
def mish(x):
        return tf.keras.layers.Lambda(lambda x: x*K.tanh(K.softplus(x)))(x)

from tensorflow.keras.utils import get_custom_objects
from tensorflow.keras.layers import Activation
get_custom_objects().update({'mish': Activation(mish)})
```

In [17]:
```
import tensorflow as tf
from tensorflow.keras.layers import Layer
from tensorflow.keras import initializers
from tensorflow.keras import regularizers
from tensorflow.keras import constraints

class Attention(Layer):
    """Multi-headed attention layer."""

    def __init__(self, hidden_size,
                 num_heads = 8,
                 attention_dropout=.1,
                 trainable=True,
                 name='Attention'):

        if hidden_size % num_heads != 0:
            raise ValueError("Hidden size must be evenly divisible by the number of heads.")

        self.hidden_size = hidden_size
        self.num_heads = num_heads
        self.trainable = trainable
        self.attention_dropout = attention_dropout
        self.dense = tf.keras.layers.Dense(self.hidden_size, use_bias=False)
        super(Attention, self).__init__(name=name)

    def split_heads(self, x):
        """Split x into different heads, and transpose the resulting value.
        The tensor is transposed to insure the inner dimensions hold the correct
        values during the matrix multiplication.
        Args:
          x: A tensor with shape [batch_size, length, hidden_size]
        Returns:
          A tensor with shape [batch_size, num_heads, length, hidden_size/num_heads]
        """
        with tf.name_scope("split_heads"):
            batch_size = tf.shape(x)[0]
            length = tf.shape(x)[1]

            # Calculate depth of last dimension after it has been split.
            depth = (self.hidden_size // self.num_heads)

            # Split the last dimension
            x = tf.reshape(x, [batch_size, length, self.num_heads, depth])

            # Transpose the result
            return tf.transpose(x, [0, 2, 1, 3])

    def combine_heads(self, x):
        """Combine tensor that has been split.
        Args:
          x: A tensor [batch_size, num_heads, length, hidden_size/num_heads]
        Returns:
          A tensor with shape [batch_size, length, hidden_size]
        """
        with tf.name_scope("combine_heads"):
            batch_size = tf.shape(x)[0]
            length = tf.shape(x)[2]
            x = tf.transpose(x, [0, 2, 1, 3])  # --> [batch, length, num_heads, depth]
            return tf.reshape(x, [batch_size, length, self.hidden_size])

    def call(self, inputs):
        """Apply attention mechanism to inputs.
        Args:
          inputs: a tensor with shape [batch_size, length_x, hidden_size]
        Returns:
          Attention layer output with shape [batch_size, length_x, hidden_size]
        """
        # Google developper use tf.layer.Dense to linearly project the queries, keys, and values.
        q = self.dense(inputs)
        k = self.dense(inputs)
        v = self.dense(inputs)

        q = self.split_heads(q)
        k = self.split_heads(k)
        v = self.split_heads(v)

        # Scale q to prevent the dot product between q and k from growing too large.
        depth = (self.hidden_size // self.num_heads)
        q *= depth ** -0.5

        logits = tf.matmul(q, k, transpose_b=True)
        # logits += self.bias
        weights = tf.nn.softmax(logits, name="attention_weights")

        if self.trainable:
            weights = tf.nn.dropout(weights, 1.0 - self.attention_dropout)

        attention_output = tf.matmul(weights, v)
        attention_output = self.combine_heads(attention_output)
        attention_output = self.dense(attention_output)
        return attention_output

    def compute_output_shape(self, input_shape):
        return tf.TensorShape(input_shape)
```

In [18]:
```
def categorical_focal_loss(gamma=2.0, alpha=0.25):
    """
    Implementation of Focal Loss from the paper in multiclass classification
    Formula:
        loss = -alpha*((1-p)^gamma)*log(p)
    Parameters:
        alpha -- the same as wighting factor in balanced cross entropy
        gamma -- focusing parameter for modulating factor (1-p)
    Default value:
        gamma -- 2.0 as mentioned in the paper
        alpha -- 0.25 as mentioned in the paper
    """
    def focal_loss(y_true, y_pred):
        # Define epsilon so that the backpropagation will not result in NaN
        # for 0 divisor case
        epsilon = K.epsilon()
        # Add the epsilon to prediction value
        #y_pred = y_pred + epsilon
        # Clip the prediction value
        y_pred = K.clip(y_pred, epsilon, 1.0-epsilon)
        # Calculate cross entropy
        cross_entropy = -y_true*K.log(y_pred)
        # Calculate weight that consists of  modulating factor and weighting factor
        weight = alpha * y_true * K.pow((1-y_pred), gamma)
        # Calculate focal loss
        loss = weight * cross_entropy
        # Sum the losses in mini_batch
        loss = K.sum(loss, axis=1)
        return loss

    return focal_loss
```

In [19]:
```
def WaveNetResidualConv1D(num_filters, kernel_size, stacked_layer):

    def build_residual_block(l_input):
        resid_input = l_input
        for dilation_rate in [2**i for i in range(stacked_layer)]:
            l_sigmoid_conv1d = Conv1D(
              num_filters, kernel_size, dilation_rate=dilation_rate,
              padding='same', activation='sigmoid')(l_input)
            l_tanh_conv1d = Conv1D(
             num_filters, kernel_size, dilation_rate=dilation_rate,
             padding='same', activation='mish')(l_input)
            l_input = Multiply()([l_sigmoid_conv1d, l_tanh_conv1d])
            l_input = Conv1D(num_filters, 1, padding='same')(l_input)
            resid_input = Add()([resid_input ,l_input])
        return resid_input
    return build_residual_block
def Classifier(shape_):
    num_filters_ = 16
    kernel_size_ = 3
    stacked_layers_ = [12, 8, 4, 1]
    l_input = Input(shape=(shape_))
    x = Conv1D(num_filters_, 1, padding='same')(l_input)
    x = WaveNetResidualConv1D(num_filters_, kernel_size_, stacked_layers_[0])(x)
    x = Conv1D(num_filters_*2, 1, padding='same')(x)
    x = WaveNetResidualConv1D(num_filters_*2, kernel_size_, stacked_layers_[1])(x)
    x = Conv1D(num_filters_*4, 1, padding='same')(x)
    x = WaveNetResidualConv1D(num_filters_*4, kernel_size_, stacked_layers_[2])(x)
    x = Conv1D(num_filters_*8, 1, padding='same')(x)
    x = WaveNetResidualConv1D(num_filters_*8, kernel_size_, stacked_layers_[3])(x)
    l_output = Dense(11, activation='softmax')(x)
    model = models.Model(inputs=[l_input], outputs=[l_output])
    opt = Adam(lr=LR)
    opt = tfa.optimizers.SWA(opt)
    model.compile(loss=losses.CategoricalCrossentropy(), optimizer=opt, metrics=['accuracy'])
    return model
```

In [20]:
```
def Classifierx(shape_):

    inp = Input(shape=(shape_))
    x = Bidirectional(GRU(256, return_sequences=True))(inp)
    x = Attention(512)(x)
    x = TimeDistributed(Dense(256, activation='mish'))(x)
    x = TimeDistributed(Dense(128, activation='mish'))(x)
    out = TimeDistributed(Dense(11, activation='softmax', name='out'))(x)

    model = models.Model(inputs=inp, outputs=out)

    opt = Adam(lr=LR)
    opt = tfa.optimizers.SWA(opt)
    model.compile(loss=losses.CategoricalCrossentropy(), optimizer=opt, metrics=['accuracy'])
    return model
```

In [21]:
```
class MacroF1(Callback):
    def __init__(self, model, inputs, targets):
        self.model = model
        self.inputs = inputs
        self.targets = np.argmax(targets, axis=2).reshape(-1)

    def on_epoch_end(self, epoch, logs):
        pred = np.argmax(self.model.predict(self.inputs), axis=2).reshape(-1)
        score = f1_score(self.targets, pred, average="macro")
        print(f' F1Macro: {score:.5f}')
```

In [22]:
```
def normalize(train, test):

    train_input_mean = train.signal.mean()
    train_input_sigma = train.signal.std()
    train['signal'] = (train.signal-train_input_mean)/train_input_sigma
    test['signal'] = (test.signal-train_input_mean)/train_input_sigma

    return train, test
```

In [23]:
```
def run_everything(fe_config : List) -> NoReturn:
    not_feats_cols = ['time']
    target_col = ['open_channels']
    init_logger()
    with timer(f'Reading Data'):
        logger.info('Reading Data Started ...')
        base = os.path.abspath('/kaggle/input/liverpool-ion-switching/')
        train, test, sample_submission = read_data(base)
        train, test = normalize(train, test)
        logger.info('Reading and Normalizing Data Completed ...')
    with timer(f'Creating Features'):
        logger.info('Feature Enginnering Started ...')
        for config in fe_config:
            train = run_feat_enginnering(train, create_all_data_feats=config[0], batch_size=config[1])
            test  = run_feat_enginnering(test,  create_all_data_feats=config[0], batch_size=config[1])
        train, test, feats = feature_selection(train, test)
        logger.info('Feature Enginnering Completed ...')

    with timer(f'Running Wavenet model'):
        logger.info(f'Training Wavenet model with {SPLITS} folds of GroupKFold Started ...')
        run_cv_model_by_batch(train, test, splits=SPLITS, batch_col='group', feats=feats, sample_submission=sample_submission, nn_epochs=EPOCHS, nn_batch_size=NNBATCHSIZE)
        logger.info(f'Training completed ...')
```

In [24]:
```
run_everything(fe_config)
```

```
2020-04-04 15:47:19,510 INFO Reading Data Started ...
2020-04-04 15:47:22,700 INFO Reading and Normalizing Data Completed ...
2020-04-04 15:47:22,701 INFO [Reading Data] done in 3 s
2020-04-04 15:47:22,703 INFO Feature Enginnering Started ...
2020-04-04 15:47:35,540 INFO Feature Enginnering Completed ...
2020-04-04 15:47:35,542 INFO [Creating Features] done in 13 s
2020-04-04 15:47:35,543 INFO Training Wavenet model with 5 folds of GroupKFold Started ...
```

```
F1Macro: 0.88717
F1Macro: 0.92537
F1Macro: 0.93041
F1Macro: 0.93231
F1Macro: 0.93446
F1Macro: 0.93602
F1Macro: 0.93300
F1Macro: 0.93556
F1Macro: 0.93413
F1Macro: 0.93416
F1Macro: 0.93263
F1Macro: 0.93659
F1Macro: 0.93185
F1Macro: 0.93569
F1Macro: 0.93742
F1Macro: 0.91941
F1Macro: 0.93689
F1Macro: 0.93730
F1Macro: 0.93569
F1Macro: 0.93411
F1Macro: 0.93578
F1Macro: 0.93499
F1Macro: 0.93756
F1Macro: 0.93739
F1Macro: 0.93156
F1Macro: 0.93788
F1Macro: 0.92603
F1Macro: 0.93644
F1Macro: 0.93475
F1Macro: 0.93781
F1Macro: 0.93754
F1Macro: 0.93808
F1Macro: 0.93602
F1Macro: 0.93753
F1Macro: 0.93798
F1Macro: 0.93829
F1Macro: 0.93832
F1Macro: 0.93045
F1Macro: 0.93829
F1Macro: 0.93776
F1Macro: 0.93852
F1Macro: 0.93844
F1Macro: 0.93848
F1Macro: 0.93849
F1Macro: 0.93863
F1Macro: 0.93850
F1Macro: 0.93833
F1Macro: 0.93836
F1Macro: 0.93817
F1Macro: 0.93852
F1Macro: 0.93862
F1Macro: 0.93868
F1Macro: 0.93867
F1Macro: 0.93864
F1Macro: 0.93866
F1Macro: 0.93867
F1Macro: 0.93884
F1Macro: 0.93870
F1Macro: 0.93844
F1Macro: 0.93810
F1Macro: 0.93882
F1Macro: 0.93865
F1Macro: 0.93836
F1Macro: 0.93862
F1Macro: 0.93873
F1Macro: 0.93880
F1Macro: 0.93875
F1Macro: 0.93872
F1Macro: 0.93871
F1Macro: 0.93868
F1Macro: 0.93871
F1Macro: 0.93860
F1Macro: 0.93891
F1Macro: 0.93882
F1Macro: 0.93882
F1Macro: 0.93881
F1Macro: 0.93881
F1Macro: 0.93874
F1Macro: 0.93874
F1Macro: 0.93887
F1Macro: 0.93865
F1Macro: 0.93885
F1Macro: 0.93870
F1Macro: 0.93858
F1Macro: 0.93865
F1Macro: 0.93889
F1Macro: 0.93885
F1Macro: 0.93870
F1Macro: 0.93881
F1Macro: 0.93891
F1Macro: 0.93872
F1Macro: 0.93888
F1Macro: 0.93877
F1Macro: 0.93886
F1Macro: 0.93880
F1Macro: 0.93891
F1Macro: 0.93881
F1Macro: 0.93856
F1Macro: 0.93881
F1Macro: 0.93886
F1Macro: 0.93876
F1Macro: 0.93894
F1Macro: 0.93877
F1Macro: 0.93887
F1Macro: 0.93892
F1Macro: 0.93878
F1Macro: 0.93891
F1Macro: 0.93883
F1Macro: 0.93885
F1Macro: 0.93878
F1Macro: 0.93892
F1Macro: 0.93878
F1Macro: 0.93880
F1Macro: 0.93878
F1Macro: 0.93879
F1Macro: 0.93882
F1Macro: 0.93887
F1Macro: 0.93862
F1Macro: 0.93881
F1Macro: 0.93881
```

```
2020-04-04 16:20:26,858 INFO Training fold 1 completed. macro f1 score : 0.93881
```

```
F1Macro: 0.89838
F1Macro: 0.92962
F1Macro: 0.93124
F1Macro: 0.93647
F1Macro: 0.93746
F1Macro: 0.93810
F1Macro: 0.93554
F1Macro: 0.93801
F1Macro: 0.93672
F1Macro: 0.93553
F1Macro: 0.93760
F1Macro: 0.93856
F1Macro: 0.93747
F1Macro: 0.93883
F1Macro: 0.93856
F1Macro: 0.84523
F1Macro: 0.84762
F1Macro: 0.84855
F1Macro: 0.84835
F1Macro: 0.84913
F1Macro: 0.84753
F1Macro: 0.84930
F1Macro: 0.84924
F1Macro: 0.84929
F1Macro: 0.84869
F1Macro: 0.84949
F1Macro: 0.84906
F1Macro: 0.84969
F1Macro: 0.84943
F1Macro: 0.92684
F1Macro: 0.93836
F1Macro: 0.93903
F1Macro: 0.93870
F1Macro: 0.93890
F1Macro: 0.93940
F1Macro: 0.93861
F1Macro: 0.93914
F1Macro: 0.93817
F1Macro: 0.93934
F1Macro: 0.93855
F1Macro: 0.93983
F1Macro: 0.93973
F1Macro: 0.93976
F1Macro: 0.93971
F1Macro: 0.93978
F1Macro: 0.93991
F1Macro: 0.93974
F1Macro: 0.93927
F1Macro: 0.93957
F1Macro: 0.93976
F1Macro: 0.93982
F1Macro: 0.93979
F1Macro: 0.93989
F1Macro: 0.93962
F1Macro: 0.93961
F1Macro: 0.93969
F1Macro: 0.93985
F1Macro: 0.93966
F1Macro: 0.93974
F1Macro: 0.93977
F1Macro: 0.93988
F1Macro: 0.93992
F1Macro: 0.93990
F1Macro: 0.94002
F1Macro: 0.93996
F1Macro: 0.93972
F1Macro: 0.93972
F1Macro: 0.93979
F1Macro: 0.93972
F1Macro: 0.93984
F1Macro: 0.93959
F1Macro: 0.93983
F1Macro: 0.93963
F1Macro: 0.93970
F1Macro: 0.93972
F1Macro: 0.93983
F1Macro: 0.93966
F1Macro: 0.93981
F1Macro: 0.93984
F1Macro: 0.93979
F1Macro: 0.93975
F1Macro: 0.93981
F1Macro: 0.93988
F1Macro: 0.93982
F1Macro: 0.93996
F1Macro: 0.93987
F1Macro: 0.93979
F1Macro: 0.94011
F1Macro: 0.93990
F1Macro: 0.93990
F1Macro: 0.93978
F1Macro: 0.93976
F1Macro: 0.93978
F1Macro: 0.93983
F1Macro: 0.93992
F1Macro: 0.93984
F1Macro: 0.93972
F1Macro: 0.93988
F1Macro: 0.93981
F1Macro: 0.93987
F1Macro: 0.93989
F1Macro: 0.93988
F1Macro: 0.93983
F1Macro: 0.93984
F1Macro: 0.93990
F1Macro: 0.93997
F1Macro: 0.93996
F1Macro: 0.94001
F1Macro: 0.94001
F1Macro: 0.93992
F1Macro: 0.94003
F1Macro: 0.93994
F1Macro: 0.93994
F1Macro: 0.93990
F1Macro: 0.93978
F1Macro: 0.93984
F1Macro: 0.93984
F1Macro: 0.93986
F1Macro: 0.93990
F1Macro: 0.93989
```

```
2020-04-04 16:53:17,552 INFO Training fold 2 completed. macro f1 score : 0.93989
```

```
F1Macro: 0.78224
F1Macro: 0.90813
F1Macro: 0.92552
F1Macro: 0.92897
F1Macro: 0.93046
F1Macro: 0.93151
F1Macro: 0.93454
F1Macro: 0.89066
F1Macro: 0.93234
F1Macro: 0.93297
F1Macro: 0.93583
F1Macro: 0.93550
F1Macro: 0.93595
F1Macro: 0.93617
F1Macro: 0.93388
F1Macro: 0.93530
F1Macro: 0.93601
F1Macro: 0.93593
F1Macro: 0.93376
F1Macro: 0.93604
F1Macro: 0.93385
F1Macro: 0.93516
F1Macro: 0.93473
F1Macro: 0.93585
F1Macro: 0.93618
F1Macro: 0.93675
F1Macro: 0.93637
F1Macro: 0.93664
F1Macro: 0.93406
F1Macro: 0.93569
F1Macro: 0.93606
F1Macro: 0.93667
F1Macro: 0.93656
F1Macro: 0.90113
F1Macro: 0.93540
F1Macro: 0.93459
F1Macro: 0.93609
F1Macro: 0.93624
F1Macro: 0.93668
F1Macro: 0.93625
F1Macro: 0.93716
F1Macro: 0.93712
F1Macro: 0.93737
F1Macro: 0.93718
F1Macro: 0.93719
F1Macro: 0.93723
F1Macro: 0.93736
F1Macro: 0.93739
F1Macro: 0.93732
F1Macro: 0.93730
F1Macro: 0.93728
F1Macro: 0.93731
F1Macro: 0.93724
F1Macro: 0.93729
F1Macro: 0.93737
F1Macro: 0.93738
F1Macro: 0.93730
F1Macro: 0.93740
F1Macro: 0.93741
F1Macro: 0.93728
F1Macro: 0.93727
F1Macro: 0.93728
F1Macro: 0.93738
F1Macro: 0.93743
F1Macro: 0.93739
F1Macro: 0.93745
F1Macro: 0.93754
F1Macro: 0.93742
F1Macro: 0.93730
F1Macro: 0.93740
F1Macro: 0.93741
F1Macro: 0.93742
F1Macro: 0.93731
F1Macro: 0.93742
F1Macro: 0.93741
F1Macro: 0.93754
F1Macro: 0.93751
F1Macro: 0.93755
F1Macro: 0.93745
F1Macro: 0.93746
F1Macro: 0.93747
F1Macro: 0.93754
F1Macro: 0.93743
F1Macro: 0.93746
F1Macro: 0.93749
F1Macro: 0.93749
F1Macro: 0.93754
F1Macro: 0.93740
F1Macro: 0.93764
F1Macro: 0.93738
F1Macro: 0.93754
F1Macro: 0.93740
F1Macro: 0.93760
F1Macro: 0.93754
F1Macro: 0.93740
F1Macro: 0.93762
F1Macro: 0.93764
F1Macro: 0.93740
F1Macro: 0.93746
F1Macro: 0.93736
F1Macro: 0.93758
F1Macro: 0.93762
F1Macro: 0.93763
F1Macro: 0.93752
F1Macro: 0.93760
F1Macro: 0.93769
F1Macro: 0.93764
F1Macro: 0.93751
F1Macro: 0.93750
F1Macro: 0.93775
F1Macro: 0.93766
F1Macro: 0.93763
F1Macro: 0.93768
F1Macro: 0.93768
F1Macro: 0.93763
F1Macro: 0.93759
F1Macro: 0.93760
F1Macro: 0.93760
F1Macro: 0.93763
F1Macro: 0.93763
```

```
2020-04-04 17:11:36,492 INFO Training fold 3 completed. macro f1 score : 0.93763
```

```
F1Macro: 0.78343
F1Macro: 0.85404
F1Macro: 0.92447
F1Macro: 0.92819
F1Macro: 0.93191
F1Macro: 0.93275
F1Macro: 0.93361
F1Macro: 0.93477
F1Macro: 0.93531
F1Macro: 0.93567
F1Macro: 0.93247
F1Macro: 0.93047
F1Macro: 0.93500
F1Macro: 0.93337
F1Macro: 0.93069
F1Macro: 0.93060
F1Macro: 0.93590
F1Macro: 0.93468
F1Macro: 0.93343
F1Macro: 0.93640
F1Macro: 0.93658
F1Macro: 0.93727
F1Macro: 0.93722
F1Macro: 0.93653
F1Macro: 0.93677
F1Macro: 0.93245
F1Macro: 0.93601
F1Macro: 0.93507
F1Macro: 0.92804
F1Macro: 0.93692
F1Macro: 0.93727
F1Macro: 0.93731
F1Macro: 0.93024
F1Macro: 0.93381
F1Macro: 0.93723
F1Macro: 0.93387
F1Macro: 0.93366
F1Macro: 0.93742
F1Macro: 0.93742
F1Macro: 0.93625
F1Macro: 0.93802
F1Macro: 0.93766
F1Macro: 0.93791
F1Macro: 0.93802
F1Macro: 0.93803
F1Macro: 0.93798
F1Macro: 0.93792
F1Macro: 0.93775
F1Macro: 0.93773
F1Macro: 0.93756
F1Macro: 0.93791
F1Macro: 0.93794
F1Macro: 0.93792
F1Macro: 0.93799
F1Macro: 0.93799
F1Macro: 0.93820
F1Macro: 0.93771
F1Macro: 0.93802
F1Macro: 0.93801
F1Macro: 0.93789
F1Macro: 0.93807
F1Macro: 0.93804
F1Macro: 0.93810
F1Macro: 0.93811
F1Macro: 0.93804
F1Macro: 0.93799
F1Macro: 0.93805
F1Macro: 0.93824
F1Macro: 0.93788
F1Macro: 0.93813
F1Macro: 0.93802
F1Macro: 0.93809
F1Macro: 0.93799
F1Macro: 0.93791
F1Macro: 0.93796
F1Macro: 0.93787
F1Macro: 0.93804
F1Macro: 0.93807
F1Macro: 0.93817
F1Macro: 0.93799
F1Macro: 0.93812
F1Macro: 0.93818
F1Macro: 0.93794
F1Macro: 0.93814
F1Macro: 0.93810
F1Macro: 0.93812
F1Macro: 0.93795
F1Macro: 0.93826
F1Macro: 0.93793
F1Macro: 0.93810
F1Macro: 0.93824
F1Macro: 0.93814
F1Macro: 0.93813
F1Macro: 0.93800
F1Macro: 0.93819
F1Macro: 0.93797
F1Macro: 0.93797
F1Macro: 0.93819
F1Macro: 0.93793
F1Macro: 0.93791
F1Macro: 0.93810
F1Macro: 0.93816
F1Macro: 0.93808
F1Macro: 0.93818
F1Macro: 0.93821
F1Macro: 0.93811
F1Macro: 0.93814
F1Macro: 0.93805
F1Macro: 0.93810
F1Macro: 0.93814
F1Macro: 0.93820
F1Macro: 0.93813
F1Macro: 0.93819
F1Macro: 0.93811
F1Macro: 0.93808
F1Macro: 0.93821
F1Macro: 0.93804
F1Macro: 0.93807
F1Macro: 0.93823
F1Macro: 0.93809
```

```
2020-04-04 17:30:01,226 INFO Training fold 4 completed. macro f1 score : 0.93809
```

```
F1Macro: 0.72163
F1Macro: 0.87476
F1Macro: 0.92538
F1Macro: 0.93057
F1Macro: 0.93177
F1Macro: 0.92589
F1Macro: 0.93438
F1Macro: 0.92904
F1Macro: 0.93569
F1Macro: 0.93543
F1Macro: 0.93618
F1Macro: 0.93543
F1Macro: 0.93387
F1Macro: 0.93642
F1Macro: 0.93595
F1Macro: 0.93528
F1Macro: 0.93519
F1Macro: 0.93704
F1Macro: 0.93698
F1Macro: 0.93725
F1Macro: 0.92647
F1Macro: 0.93681
F1Macro: 0.93687
F1Macro: 0.93725
F1Macro: 0.93447
F1Macro: 0.93565
F1Macro: 0.93742
F1Macro: 0.93704
F1Macro: 0.93720
F1Macro: 0.93678
F1Macro: 0.93721
F1Macro: 0.93548
F1Macro: 0.93535
F1Macro: 0.93737
F1Macro: 0.93711
F1Macro: 0.93646
F1Macro: 0.92626
F1Macro: 0.93773
F1Macro: 0.93776
F1Macro: 0.93646
F1Macro: 0.93808
F1Macro: 0.93734
F1Macro: 0.93800
F1Macro: 0.93794
F1Macro: 0.93819
F1Macro: 0.93815
F1Macro: 0.93810
F1Macro: 0.93781
F1Macro: 0.93816
F1Macro: 0.93806
F1Macro: 0.93821
F1Macro: 0.93830
F1Macro: 0.93812
F1Macro: 0.93820
F1Macro: 0.93799
F1Macro: 0.93779
F1Macro: 0.93813
F1Macro: 0.93815
F1Macro: 0.93815
F1Macro: 0.93816
F1Macro: 0.93818
F1Macro: 0.93760
F1Macro: 0.93830
F1Macro: 0.93818
F1Macro: 0.93811
F1Macro: 0.93826
F1Macro: 0.93822
F1Macro: 0.93828
F1Macro: 0.93821
F1Macro: 0.93826
F1Macro: 0.93828
F1Macro: 0.93816
F1Macro: 0.93787
F1Macro: 0.93824
F1Macro: 0.93809
F1Macro: 0.93825
F1Macro: 0.93826
F1Macro: 0.93822
F1Macro: 0.93837
F1Macro: 0.93829
F1Macro: 0.93831
F1Macro: 0.93825
F1Macro: 0.93819
F1Macro: 0.93835
F1Macro: 0.93827
F1Macro: 0.93814
F1Macro: 0.93834
F1Macro: 0.93808
F1Macro: 0.93831
F1Macro: 0.93828
F1Macro: 0.93829
F1Macro: 0.93834
F1Macro: 0.93827
F1Macro: 0.93785
F1Macro: 0.93836
F1Macro: 0.93797
F1Macro: 0.93815
F1Macro: 0.93818
F1Macro: 0.93820
F1Macro: 0.93835
F1Macro: 0.93838
F1Macro: 0.93828
F1Macro: 0.93833
F1Macro: 0.93833
F1Macro: 0.93838
F1Macro: 0.93836
F1Macro: 0.93833
F1Macro: 0.93833
F1Macro: 0.93832
F1Macro: 0.93828
F1Macro: 0.93832
F1Macro: 0.93830
F1Macro: 0.93829
F1Macro: 0.93831
F1Macro: 0.93832
F1Macro: 0.93836
F1Macro: 0.93838
F1Macro: 0.93837
F1Macro: 0.93824
F1Macro: 0.93841
```

```
2020-04-04 17:48:22,611 INFO Training fold 5 completed. macro f1 score : 0.93841
2020-04-04 17:48:26,248 INFO Training completed. oof macro f1 score : 0.93860
```

	time	open_channels
0	500.000092	0
1	500.000214	0
2	500.000305	0
3	500.000397	0
4	500.000488	0

```
2020-04-04 17:48:37,654 INFO Training completed ...
2020-04-04 17:48:37,656 INFO [Running Wavenet model] done in 7262 s
```

In [ ]: