model 1

CNN + GRU + GRU

---

imports

```
import tensorflow as tf
from keras.models import load_model
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras_tqdm import TQDMNotebookCallback
import numpy as np
from keras_tqdm import TQDMNotebookCallback
import nltk
import xml.etree.ElementTree as ET
import pandas as pd
import os
import string
from nltk.tokenize import TreebankWordTokenizer
from numpy.random import random_sample
import re
import pickle
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

from keras.layers import Embedding, Flatten,LSTM,GRU
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation,  Input, merge,Conv1D,MaxPooling1D,GlobalMaxPooling1D,Convolution1D
from keras import regularizers
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
from keras.layers import Concatenate, concatenate
from keras import backend as K
from keras.layers import multiply
from keras.layers import merge
from keras.layers.core import *
from keras.layers.recurrent import LSTM
from keras.models import *
```

Define Callback functions to generate Measures

```
from keras import backend as K

def f1(y_true, y_pred):
    def recall(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

    def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Double-click (or enter) to edit

Load pre procssed Data

```
with open('../data/pickles/train_and_test_data_sentences_snp_2class.pickle', 'rb') as handle:

    W_train = pickle.load(handle)
    d1_train = pickle.load(handle)
    d2_train = pickle.load(handle)
    Y_train = pickle.load(handle)
    Tr_word_list = pickle.load(handle)

    W_test = pickle.load(handle)
    d1_test = pickle.load(handle)
    d2_test = pickle.load(handle)
    Y_test = pickle.load(handle)
    Te_word_list = pickle.load(handle)


    word_vectors = pickle.load(handle)
    word_dict = pickle.load(handle)
    d1_dict = pickle.load(handle)
    d2_dict = pickle.load(handle)
    label_dict = pickle.load(handle)
    MAX_SEQUENCE_LENGTH = pickle.load(handle)
```

Prepare Word Embedding Layer

```
EMBEDDING_DIM=word_vectors.shape[1]
embedding_matrix=word_vectors

def create_embedding_layer(l2_reg=0.1,use_pretrained=True,is_trainable=False):

    if use_pretrained:
        return Embedding(len(word_dict) ,EMBEDDING_DIM,weights=[embedding_matrix],input_length=MAX_SEQUENCE_LENGTH,trainable=is_trainable,em
    else:
        return Embedding(len(word_dict) ,EMBEDDING_DIM,input_length=MAX_SEQUENCE_LENGTH)



INPUT_DIM = 2
TIME_STEPS = MAX_SEQUENCE_LENGTH


def attentionNew(inputs):
    inputs = Lambda(lambda x: tf.keras.backend.sigmoid(x))(inputs)
    input_dim = int(inputs.shape[2])
    a = Permute((2, 1))(inputs)
    a = Dense(TIME_STEPS, activation='softmax')(a)
    a_probs = Permute((2, 1))(a)
    output_attention_mul = multiply([inputs, a_probs])
    output_attention_mul = Lambda(lambda x: tf.keras.backend.sigmoid(x))(output_attention_mul)
    return output_attention_mul
```

Create the Model

```
from keras.optimizers import Adam
def build_model():
    sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
    embedding_layer = create_embedding_layer(use_pretrained=True, is_trainable=False)
    embedded_sequences = embedding_layer(sequence_input)

    # First Conv1D Layer
    x = Conv1D(256, 7, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.4)(x)

    # Second Conv1D Layer
    x = Conv1D(128, 5, activation='relu')(x)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.4)(x)
```

```
    conv_sequence = GlobalMaxPooling1D()(x)  # GlobalMaxPooling after all CNN layers

    # Bidirectional RNN layers
    forward = GRU(100, return_sequences=True, recurrent_dropout=0.05)(embedded_sequences)
    backward = GRU(100, return_sequences=True, go_backwards=True, recurrent_dropout=0.05)(embedded_sequences)
    lstm_gru_sequence = concatenate([forward, backward], axis=-1)

    # Apply attention mechanism
    attention_output = attentionNew(lstm_gru_sequence)
    attention_pooled = GlobalMaxPooling1D()(attention_output)

    # Merge CNN and attention-enhanced RNN outputs
    merge = concatenate([conv_sequence, attention_pooled])

    # Fully connected layers
    x = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.05))(merge)
    x = Dropout(0.4)(x)
    preds = Dense(2, activation='softmax')(x)

    # Compile model
    optimizer = Adam(learning_rate=0.001)
    model = Model(sequence_input, preds)
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['acc', f1])

    return model


model = build_model()
model.summary()
```

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 91)	0	
embedding_1 (Embedding)	(None, 91, 200)	555000	input_1[0][0]
gru_1 (GRU)	(None, 91, 100)	90300	embedding_1[0][0]
gru_2 (GRU)	(None, 91, 100)	90300	embedding_1[0][0]
concatenate_1 (Concatenate)	(None, 91, 200)	0	gru_1[0][0] gru_2[0][0]
conv1d_1 (Conv1D)	(None, 85, 256)	358656	embedding_1[0][0]
lambda_1 (Lambda)	(None, 91, 200)	0	concatenate_1[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 28, 256)	0	conv1d_1[0][0]
permute_1 (Permute)	(None, 200, 91)	0	lambda_1[0][0]
dropout_1 (Dropout)	(None, 28, 256)	0	max_pooling1d_1[0][0]
dense_1 (Dense)	(None, 200, 91)	8372	permute_1[0][0]
conv1d_2 (Conv1D)	(None, 24, 128)	163968	dropout_1[0][0]
permute_2 (Permute)	(None, 91, 200)	0	dense_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 8, 128)	0	conv1d_2[0][0]
multiply_1 (Multiply)	(None, 91, 200)	0	lambda_1[0][0] permute_2[0][0]
dropout_2 (Dropout)	(None, 8, 128)	0	max_pooling1d_2[0][0]
lambda_2 (Lambda)	(None, 91, 200)	0	multiply_1[0][0]
global_max_pooling1d_1 (GlobalM	(None, 128)	0	dropout_2[0][0]
global_max_pooling1d_2 (GlobalM	(None, 200)	0	lambda_2[0][0]
concatenate_2 (Concatenate)	(None, 328)	0	global_max_pooling1d_1[0][0] global_max_pooling1d_2[0][0]
dense_2 (Dense)	(None, 256)	84224	concatenate_2[0][0]
dropout_3 (Dropout)	(None, 256)	0	dense_2[0][0]

```
dense_3 (Dense)                 (None, 2)            514         dropout_3[0][0]
==================================================================================================
Total params: 1,351,334
Trainable params: 796,334
Non-trainable params: 555,000
__________________________________________________________________________________________________
```

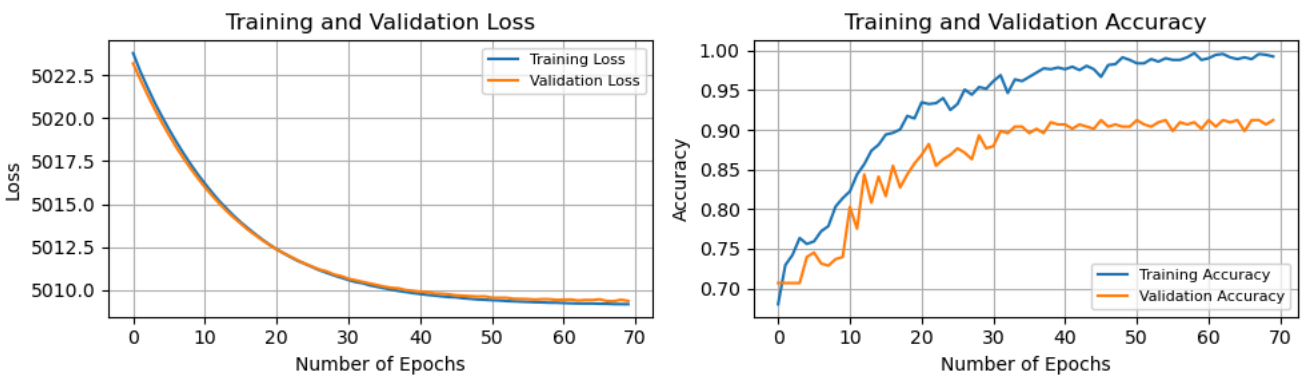
Run the Evaluation on the test dataset

```
param='macro'
epochs =70
batch_size = 32
history=model.fit(W_train, Y_train,epochs=epochs,validation_data=(W_test,Y_test), batch_size=batch_size,verbose=1)
```

```
Train on 935 samples, validate on 365 samples
Epoch 1/70
935/935 [==============================] - 4s 4ms/step - loss: 5023.7850 - acc: 0.6802 - f1: 0.6699 - val_loss: 5023.1784 - val_acc:
Epoch 2/70
935/935 [==============================] - 3s 3ms/step - loss: 5022.7198 - acc: 0.7294 - f1: 0.7365 - val_loss: 5022.2475 - val_acc:
Epoch 3/70
935/935 [==============================] - 3s 3ms/step - loss: 5021.7964 - acc: 0.7422 - f1: 0.7490 - val_loss: 5021.3641 - val_acc:
Epoch 4/70
935/935 [==============================] - 3s 3ms/step - loss: 5020.9111 - acc: 0.7636 - f1: 0.7661 - val_loss: 5020.5260 - val_acc:
Epoch 5/70
935/935 [==============================] - 3s 3ms/step - loss: 5020.1104 - acc: 0.7561 - f1: 0.7513 - val_loss: 5019.7529 - val_acc:
Epoch 6/70
935/935 [==============================] - 3s 3ms/step - loss: 5019.3556 - acc: 0.7594 - f1: 0.7656 - val_loss: 5019.0193 - val_acc:
Epoch 7/70
935/935 [==============================] - 3s 3ms/step - loss: 5018.6423 - acc: 0.7722 - f1: 0.7781 - val_loss: 5018.3180 - val_acc:
Epoch 8/70
935/935 [==============================] - 3s 3ms/step - loss: 5017.9703 - acc: 0.7786 - f1: 0.7807 - val_loss: 5017.6795 - val_acc:
Epoch 9/70
935/935 [==============================] - 3s 3ms/step - loss: 5017.3349 - acc: 0.8032 - f1: 0.8083 - val_loss: 5017.0798 - val_acc:
Epoch 10/70
935/935 [==============================] - 3s 3ms/step - loss: 5016.7391 - acc: 0.8139 - f1: 0.8150 - val_loss: 5016.5191 - val_acc:
Epoch 11/70
935/935 [==============================] - 3s 3ms/step - loss: 5016.2027 - acc: 0.8225 - f1: 0.8122 - val_loss: 5015.9883 - val_acc:
Epoch 12/70
935/935 [==============================] - 3s 3ms/step - loss: 5015.6783 - acc: 0.8439 - f1: 0.8442 - val_loss: 5015.4987 - val_acc:
Epoch 13/70
935/935 [==============================] - 3s 3ms/step - loss: 5015.1870 - acc: 0.8567 - f1: 0.8530 - val_loss: 5015.0408 - val_acc:
Epoch 14/70
935/935 [==============================] - 3s 3ms/step - loss: 5014.7513 - acc: 0.8738 - f1: 0.8734 - val_loss: 5014.6152 - val_acc:
Epoch 15/70
935/935 [==============================] - 3s 3ms/step - loss: 5014.3303 - acc: 0.8813 - f1: 0.8844 - val_loss: 5014.2215 - val_acc:
Epoch 16/70
935/935 [==============================] - 3s 3ms/step - loss: 5013.9472 - acc: 0.8941 - f1: 0.8969 - val_loss: 5013.8708 - val_acc:
Epoch 17/70
935/935 [==============================] - 3s 3ms/step - loss: 5013.5984 - acc: 0.8963 - f1: 0.8990 - val_loss: 5013.5098 - val_acc:
Epoch 18/70
935/935 [==============================] - 3s 3ms/step - loss: 5013.2617 - acc: 0.9005 - f1: 0.8957 - val_loss: 5013.2051 - val_acc:
Epoch 19/70
935/935 [==============================] - 3s 3ms/step - loss: 5012.9436 - acc: 0.9176 - f1: 0.9161 - val_loss: 5012.9040 - val_acc:
Epoch 20/70
935/935 [==============================] - 3s 3ms/step - loss: 5012.6487 - acc: 0.9144 - f1: 0.9129 - val_loss: 5012.6238 - val_acc:
Epoch 21/70
935/935 [==============================] - 3s 3ms/step - loss: 5012.3799 - acc: 0.9348 - f1: 0.9327 - val_loss: 5012.3681 - val_acc:
Epoch 22/70
935/935 [==============================] - 3s 3ms/step - loss: 5012.1395 - acc: 0.9326 - f1: 0.9344 - val_loss: 5012.1425 - val_acc:
Epoch 23/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.9100 - acc: 0.9337 - f1: 0.9354 - val_loss: 5011.9375 - val_acc:
Epoch 24/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.6948 - acc: 0.9401 - f1: 0.9379 - val_loss: 5011.7202 - val_acc:
Epoch 25/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.5232 - acc: 0.9251 - f1: 0.9271 - val_loss: 5011.5323 - val_acc:
Epoch 26/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.3321 - acc: 0.9326 - f1: 0.9307 - val_loss: 5011.3619 - val_acc:
Epoch 27/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.1548 - acc: 0.9508 - f1: 0.9521 - val_loss: 5011.1965 - val_acc:
Epoch 28/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.0044 - acc: 0.9444 - f1: 0.9458 - val_loss: 5011.0918 - val_acc:
```

```
import matplotlib.pyplot as plt

# Training & Validation accuracy
train_loss = history.history['loss']
```

```
val_loss = history.history['val_loss']
train_acc = history.history['acc']
val_acc = history.history['val_acc']
epochs = len(train_loss)

xc = range(epochs)

plt.figure(figsize=(10, 3))

# Loss subplot
plt.subplot(1, 2, 1)
plt.plot(xc, train_loss, label='Training Loss')
plt.plot(xc, val_loss, label='Validation Loss')
plt.xlabel('Number of Epochs', fontsize=10)
plt.ylabel('Loss', fontsize=10)
plt.title('Training and Validation Loss', fontsize=12)
plt.legend(fontsize=8)
plt.grid(True)

# Accuracy subplot
plt.subplot(1, 2, 2)
plt.plot(xc, train_acc, label='Training Accuracy')
plt.plot(xc, val_acc, label='Validation Accuracy')
plt.xlabel('Number of Epochs', fontsize=10)
plt.ylabel('Accuracy', fontsize=10)
plt.title('Training and Validation Accuracy', fontsize=12)
plt.legend(fontsize=8, loc='lower right')  # Change position to lower right
plt.grid(True)

plt.tight_layout()
plt.show()
```

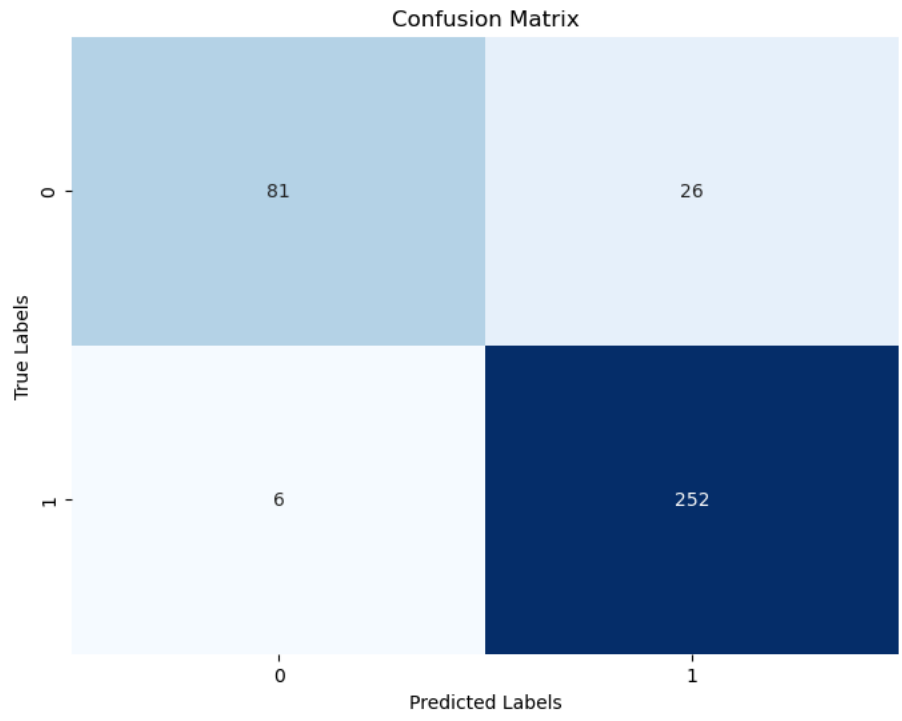



```
predicted = np.argmax(model.predict(W_test), axis=1)
y_test_to_label= np.argmax(Y_test, axis=1)
prec, reca, fscore, sup = precision_recall_fscore_support(y_test_to_label, predicted, average=param)


# Generate the classification report as a dictionary
report_dict = classification_report(y_test_to_label, predicted, output_dict=True)

# Create a new dictionary to hold the formatted values
formatted_report_dict = {}

# Iterate over the items in the report dictionary
for key, value in report_dict.items():
    if isinstance(value, dict):
        # Format the nested dictionary values
        formatted_report_dict[key] = {sub_key: f"{sub_value:.4f}" for sub_key, sub_value in value.items()}
    else:
        # Format the top-level dictionary values
        formatted_report_dict[key] = f"{value:.4f}"

# Create a string representation of the formatted dictionary
formatted_report_str = classification_report(y_test_to_label, predicted, digits=4)

# Print the formatted classification report
print(formatted_report_str)
```

```
print(" Precision:{:.2f}% Recall:{:.2f}% Fscore:{:.2f}% ".format(prec*100, reca*100, fscore*100))
```

```
              precision    recall  f1-score   support

           0     0.9310    0.7570    0.8351       107
           1     0.9065    0.9767    0.9403       258

    accuracy                         0.9123       365
   macro avg     0.9188    0.8669    0.8877       365
weighted avg     0.9137    0.9123    0.9094       365

 Precision:91.88% Recall:86.69% Fscore:88.77%
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Get the predicted labels
predicted_labels = np.argmax(model.predict(W_test), axis=1)

# Create the confusion matrix
cm = confusion_matrix(np.argmax(Y_test, axis=1), predicted_labels)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



Start coding or generate with AI.

Start coding or generate with AI.