model 1

CNN + GRU +LSTM

---

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 *
random_seed=1337
```

```
Using TensorFlow backend.
```

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()))
```

Experiments to reproduce the results of Table 9

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.tanh(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.tanh(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 = LSTM(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]
lstm_1 (LSTM)	(None, 91, 100)	120400	embedding_1[0][0]
concatenate_1 (Concatenate)	(None, 91, 200)	0	gru_1[0][0] lstm_1[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: 1,381,434
Trainable params: 826,434
Non-trainable params: 555,000
__________________________________________________________________________________________________
```

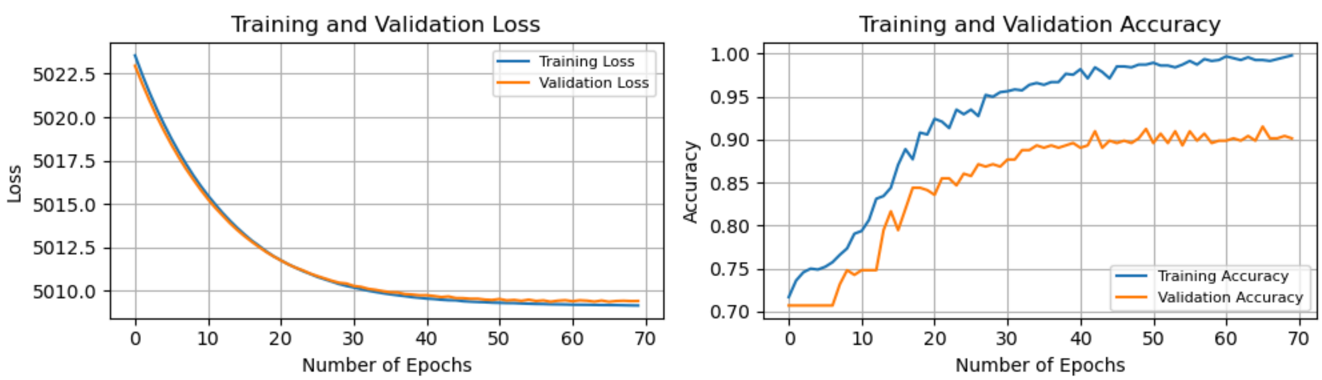
Run the Evaluation on the test dataset

```
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: 5023.5520 - acc: 0.7166 - f1: 0.7128 - val_loss: 5022.9566 - val_acc:
Epoch 2/70
935/935 [==============================] - 3s 3ms/step - loss: 5022.4352 - acc: 0.7358 - f1: 0.7353 - val_loss: 5021.9146 - val_acc:
Epoch 3/70
935/935 [==============================] - 3s 3ms/step - loss: 5021.4010 - acc: 0.7455 - f1: 0.7446 - val_loss: 5020.9427 - val_acc:
Epoch 4/70
935/935 [==============================] - 3s 3ms/step - loss: 5020.4493 - acc: 0.7497 - f1: 0.7414 - val_loss: 5020.0310 - val_acc:
Epoch 5/70
935/935 [==============================] - 3s 3ms/step - loss: 5019.5718 - acc: 0.7487 - f1: 0.7403 - val_loss: 5019.1887 - val_acc:
Epoch 6/70
935/935 [==============================] - 3s 3ms/step - loss: 5018.7545 - acc: 0.7519 - f1: 0.7509 - val_loss: 5018.4069 - val_acc:
Epoch 7/70
935/935 [==============================] - 3s 3ms/step - loss: 5017.9983 - acc: 0.7572 - f1: 0.7561 - val_loss: 5017.6847 - val_acc:
Epoch 8/70
935/935 [==============================] - 3s 3ms/step - loss: 5017.3011 - acc: 0.7658 - f1: 0.7682 - val_loss: 5017.0141 - val_acc:
Epoch 9/70
935/935 [==============================] - 3s 3ms/step - loss: 5016.6433 - acc: 0.7733 - f1: 0.7643 - val_loss: 5016.4019 - val_acc:
Epoch 10/70
935/935 [==============================] - 3s 3ms/step - loss: 5016.0444 - acc: 0.7904 - f1: 0.7884 - val_loss: 5015.8171 - val_acc:
Epoch 11/70
935/935 [==============================] - 3s 3ms/step - loss: 5015.4893 - acc: 0.7936 - f1: 0.7952 - val_loss: 5015.2792 - val_acc:
Epoch 12/70
935/935 [==============================] - 3s 3ms/step - loss: 5014.9776 - acc: 0.8064 - f1: 0.8040 - val_loss: 5014.7840 - val_acc:
Epoch 13/70
935/935 [==============================] - 3s 3ms/step - loss: 5014.4873 - acc: 0.8310 - f1: 0.8317 - val_loss: 5014.3363 - val_acc:
Epoch 14/70
935/935 [==============================] - 3s 3ms/step - loss: 5014.0459 - acc: 0.8342 - f1: 0.8385 - val_loss: 5013.8989 - val_acc:
Epoch 15/70
935/935 [==============================] - 3s 3ms/step - loss: 5013.6307 - acc: 0.8439 - f1: 0.8442 - val_loss: 5013.5073 - val_acc:
Epoch 16/70
935/935 [==============================] - 3s 3ms/step - loss: 5013.2513 - acc: 0.8706 - f1: 0.8740 - val_loss: 5013.1509 - val_acc:
Epoch 17/70
935/935 [==============================] - 3s 3ms/step - loss: 5012.8936 - acc: 0.8888 - f1: 0.8917 - val_loss: 5012.8252 - val_acc:
Epoch 18/70
935/935 [==============================] - 3s 3ms/step - loss: 5012.5920 - acc: 0.8770 - f1: 0.8802 - val_loss: 5012.5220 - val_acc:
Epoch 19/70
935/935 [==============================] - 3s 3ms/step - loss: 5012.2733 - acc: 0.9080 - f1: 0.9104 - val_loss: 5012.2323 - val_acc:
Epoch 20/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.9920 - acc: 0.9059 - f1: 0.9046 - val_loss: 5011.9806 - val_acc:
Epoch 21/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.7597 - acc: 0.9241 - f1: 0.9223 - val_loss: 5011.7489 - val_acc:
Epoch 22/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.5298 - acc: 0.9209 - f1: 0.9192 - val_loss: 5011.5293 - val_acc:
Epoch 23/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.3365 - acc: 0.9134 - f1: 0.9156 - val_loss: 5011.3343 - val_acc:
Epoch 24/70
935/935 [==============================] - 3s 3ms/step - loss: 5011.1421 - acc: 0.9348 - f1: 0.9365 - val_loss: 5011.1813 - val_acc:
Epoch 25/70
935/935 [==============================] - 3s 3ms/step - loss: 5010.9761 - acc: 0.9294 - f1: 0.9275 - val_loss: 5010.9962 - val_acc:
Epoch 26/70
935/935 [==============================] - 3s 3ms/step - loss: 5010.8015 - acc: 0.9348 - f1: 0.9327 - val_loss: 5010.8472 - val_acc:
Epoch 27/70
935/935 [==============================] - 3s 3ms/step - loss: 5010.6647 - acc: 0.9273 - f1: 0.9254 - val_loss: 5010.7155 - val_acc:
Epoch 28/70
935/935 [==============================] - 3s 3ms/step - loss: 5010.5292 - acc: 0.9519 - f1: 0.9531 - val_loss: 5010.5779 - 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()
```

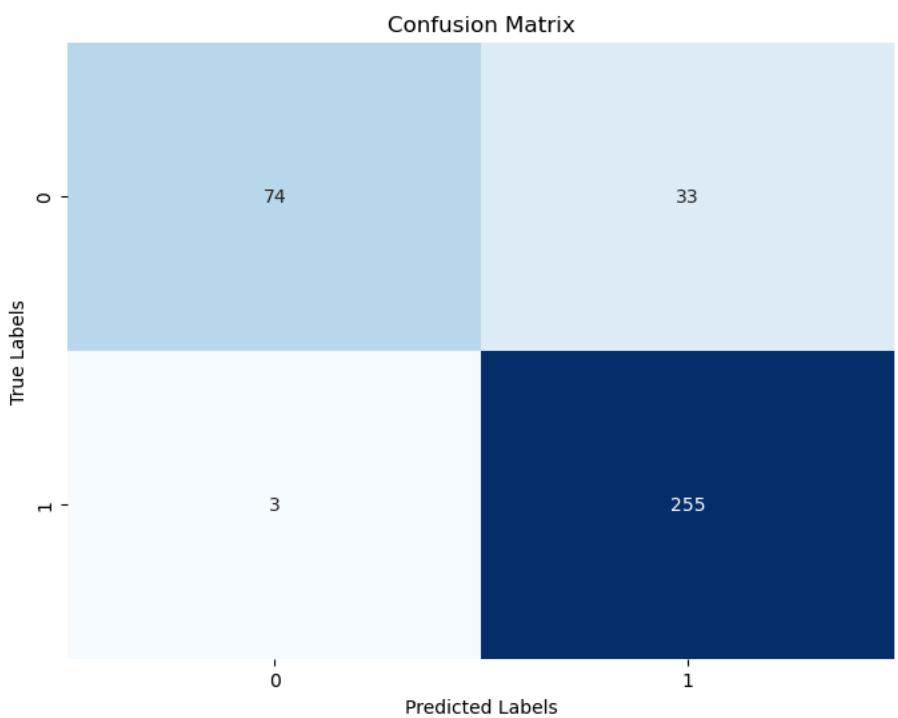



```
predicted = np.argmax(model.predict(W_test), axis=1)
y_test_to_label= np.argmax(Y_test, axis=1)
prec, reca, fscore, sup = precision_recall_fscore_support(y_test_to_label, predicted, average=param)


# 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     0.9610    0.6916    0.8043       107
           1     0.8854    0.9884    0.9341       258

    accuracy                         0.9014       365
   macro avg     0.9232    0.8400    0.8692       365
weighted avg     0.9076    0.9014    0.8960       365

 Precision:92.32% Recall:84.00% Fscore:86.92%
```

```
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.