VIETNAM NATIONAL UNIVERSITY

HO CHI MINH UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

NATURAL LANGUAGE PROCESSING

ASSIGNMENT REPORT

Student full name: Đặng Công Khanh   Student ID: 2053105

HO CHI MINH CITY 16/05/2024

Content

1. Using CNN for sentiment analysis

a) First of all, we have to do data preprocessing, which is to import tensorflow, pandas, numpy, pyvi, Word2Vec, and to_categorical. And also import all the necessary files like training test and the test dataset.

```
import tensorflow as tf
import pandas as pd
import numpy as np
from string import digits
from collections import Counter
from pyvi import ViTokenizer
from gensim.models.word2vec import Word2Vec
from tensorflow.keras.utils import to_categorical
%matplotlib inline
data_train = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/NLP/Lab5/vlsp_sentiment_train.csv", sep='\t')
data_train.columns =['Class', 'Data']
data_test = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/NLP/Lab5/vlsp_sentiment_test.csv", sep='\t')
data_test.columns =['Class', 'Data']
```

b) Secondly, we will extract labels and reviews from the dataset. From then, we can encode the labels and review the processed dataset

```
labels = data_train.iloc[:, 0].values
reviews = data_train.iloc[:, 1].values
encoded_labels = []
for label in labels:
    if label == -1:
        encoded_labels.append([1,0,0])
    elif label == 0:
        encoded_labels.append([0,1,0])
    else:
        encoded_labels.append([0,0,1])

encoded_labels = np.array(encoded_labels)
reviews_processed = []
unlabeled_processed = []
for review in reviews:
    review_cool_one = ''.join([char for char in review if char not in digits])
    reviews_processed.append(review_cool_one)
#Use PyVi for Vietnamese word tokenizer
```

```
word_reviews = []
all_words = []
for review in reviews_processed:
    review = ViTokenizer.tokenize(review.lower())
    word_reviews.append(review.split())
```

c) Thirdly, we do Tokenization and fit it on texts. Then, we convert texts to sequences and pad those. At the end, we prepare labels with encoded_labels and output shape information

```
EMBEDDING_DIM = 400 # how big is each word vector
MAX_VOCAB_SIZE = 10000 # how many unique words to use (i.e num rows in embedding vector)
MAX_SEQUENCE_LENGTH = 300 # max number of words in a comment to use

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical

tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE, lower=True,
char_level=False)
tokenizer.fit_on_texts(word_reviews)
sequences_train = tokenizer.texts_to_sequences(word_reviews)
word_index = tokenizer.word_index

data = pad_sequences(sequences_train, maxlen=MAX_SEQUENCE_LENGTH)
labels = encoded_labels

print('Shape of X train and X validation tensor:',data.shape)
print('Shape of label train and validation tensor:', labels.shape)
```

The result will look like this:

```
Shape of X train and X validation tensor: (5100, 300)
Shape of label train and validation tensor: (5100, 3)
```

d) Fourth, we load the pre-trained word embeddings, initialise the embedding matrix, populating it, cleaning up and finally create the embedding layer.

```
import gensim
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

from gensim.models.keyedvectors import KeyedVectors
```

```
word_vectors =
KeyedVectors.load_word2vec_format('/content/drive/MyDrive/Colab
Notebooks/NLP/Lab5/vi-model-CBOW.bin', binary=True)


vocabulary_size=min(len(word_index)+1,MAX_VOCAB_SIZE)
embedding_matrix = np.zeros((vocabulary_size, EMBEDDING_DIM))
print("vocab size", vocabulary_size)
for word, i in word_index.items():
    if i>=MAX_VOCAB_SIZE:
        continue
    try:
        embedding_vector = word_vectors[word]
        embedding_matrix[i] = embedding_vector
    except KeyError:

embedding_matrix[i]=np.random.normal(0,np.sqrt(0.25),EMBEDDING_DIM)

del(word_vectors)

from keras.layers import Embedding
embedding_layer = Embedding(vocabulary_size,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            trainable=True)
```

```
vocab size 7919
```

e) Fifth, now we use CNN for sentiment analysis

```
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Embedding
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Embedding,
Dropout,concatenate
from tensorflow.keras.layers import Reshape, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers
sequence_length = data.shape[1]
filter_sizes = [3,4,5]
num_filters = 100
```

```
drop = 0.5

inputs = Input(shape=(sequence_length,))
embedding = embedding_layer(inputs)
# reshape = Reshape((sequence_length,EMBEDDING_DIM,1))(embedding)

conv_0 = Conv1D(num_filters,
filter_sizes[0],activation='relu',kernel_regularizer=regularizers.l2(0.
01))(embedding)
conv_1 = Conv1D(num_filters,
filter_sizes[1],activation='relu',kernel_regularizer=regularizers.l2(0.
01))(embedding)
conv_2 = Conv1D(num_filters,
filter_sizes[2],activation='relu',kernel_regularizer=regularizers.l2(0.
01))(embedding)
print(conv_1)
maxpool_0 = MaxPooling1D(sequence_length - filter_sizes[0] + 1,
strides=1)(conv_0)
maxpool_1 = MaxPooling1D(sequence_length - filter_sizes[1] + 1,
strides=1)(conv_1)
maxpool_2 = MaxPooling1D(sequence_length - filter_sizes[2] + 1,
strides=1)(conv_2)

merged_tensor = concatenate([maxpool_0, maxpool_1, maxpool_2], axis=1)
flatten = Flatten()(merged_tensor)
reshape = Reshape((3*num_filters,))(flatten)
dropout = Dropout(drop)(flatten)
output = Dense(units=3,
activation='softmax',kernel_regularizer=regularizers.l2(0.01))(dropout)


# this creates a model that includes
model = Model(inputs, output)


# adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
decay=0.0)
adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
epsilon=1e-08)
model.compile(loss='categorical_crossentropy', optimizer=adam,
metrics=['accuracy'])
model.summary()
```

```
#define callbacks
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.01,
patience=4, verbose=1)
callbacks_list = [early_stopping]
```

The result of it looks like this:

```
KerasTensor(type_spec=TensorSpec(shape=(None, 297, 100), dtype=tf.float32, name=None),
name='conv1d_4/Relu:0', description="created by layer 'conv1d_4'")
Model: "model_1"
______________________________________________________________________________________________
________
 Layer (type)                Output Shape                 Param #     Connected to
==============================================================================================
=======
 input_2 (InputLayer)        [(None, 300)]                0           []

 embedding (Embedding)       (None, 300, 400)             3167600     ['input_2[0][0]']

 conv1d_3 (Conv1D)           (None, 298, 100)             120100      ['embedding[1][0]']

 conv1d_4 (Conv1D)           (None, 297, 100)             160100      ['embedding[1][0]']

 conv1d_5 (Conv1D)           (None, 296, 100)             200100      ['embedding[1][0]']

 max_pooling1d_3 (MaxPoolin  (None, 1, 100)               0           ['conv1d_3[0][0]']
 g1D)

 max_pooling1d_4 (MaxPoolin  (None, 1, 100)               0           ['conv1d_4[0][0]']
 g1D)

 max_pooling1d_5 (MaxPoolin  (None, 1, 100)               0           ['conv1d_5[0][0]']
 g1D)

 concatenate_1 (Concatenate  (None, 3, 100)               0
['max_pooling1d_3[0][0]',
 )
'max_pooling1d_4[0][0]',

'max_pooling1d_5[0][0]']

 flatten_1 (Flatten)         (None, 300)                  0           ['concatenate_1[0][0]']

 dropout_1 (Dropout)         (None, 300)                  0           ['flatten_1[0][0]']

 dense_1 (Dense)             (None, 3)                    903         ['dropout_1[0][0]']

==============================================================================================
=======
Total params: 3648803 (13.92 MB)
Trainable params: 3648803 (13.92 MB)
Non-trainable params: 0 (0.00 Byte)
```

f) Finally, we fit the model and get ready to test

```
model.fit(data, labels, validation_split=0.2,
          epochs=5, batch_size=256, callbacks=callbacks_list,
shuffle=True)
```

```python
labels_test = data_test.iloc[:, 0].values
reviews_test = data_test.iloc[:, 1].values

encoded_labels_test = []

for label_test in labels_test:
    if label_test == -1:
        encoded_labels_test.append([1,0,0])
    elif label_test == 0:
        encoded_labels_test.append([0,1,0])
    else:
        encoded_labels_test.append([0,0,1])

encoded_labels_test = np.array(encoded_labels_test)

reviews_processed_test = []
unlabeled_processed_test = []
for review_test in reviews_test:
    review_cool_one = ''.join([char for char in review_test if char not
in digits])
    reviews_processed_test.append(review_cool_one)

#Use PyVi for Vietnamese word tokenizer
word_reviews_test = []
all_words = []
for review_test in reviews_processed_test:
    review_test = ViTokenizer.tokenize(review_test.lower())
    word_reviews_test.append(review_test.split())

sequences_test = tokenizer.texts_to_sequences(word_reviews_test)
data_test = pad_sequences(sequences_test, maxlen=MAX_SEQUENCE_LENGTH)
labels_test = encoded_labels_test

print('Shape of X train and X validation tensor:',data_test.shape)
print('Shape of label train and validation tensor:', labels_test.shape)

score = model.evaluate(data_test, labels_test)

print("%s: %.2f%%" % (model.metrics_names[0], score[0]*100))
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))
```

This is our final result:

```
Epoch 1/5
16/16 [==============================] - 120s 7s/step - loss: 6.8037 - accuracy: 0.4659 -
val_loss: 5.9787 - val_accuracy: 0.3118
Epoch 2/5
16/16 [==============================] - 115s 7s/step - loss: 5.1295 - accuracy: 0.6537 -
val_loss: 6.7057 - val_accuracy: 0.0510
Epoch 3/5
16/16 [==============================] - 110s 7s/step - loss: 4.2578 - accuracy: 0.7463 -
val_loss: 5.7453 - val_accuracy: 0.0814
Epoch 4/5
16/16 [==============================] - 115s 7s/step - loss: 3.6253 - accuracy: 0.8086 -
val_loss: 5.0678 - val_accuracy: 0.1039
Epoch 5/5
16/16 [==============================] - 111s 7s/step - loss: 3.0948 - accuracy: 0.8522 -
val_loss: 4.6625 - val_accuracy: 0.1088

<keras.src.callbacks.History at 0x7b6786ac08e0>
```

```
Shape of X train and X validation tensor: (1050, 300)
Shape of label train and validation tensor: (1050, 3)
```

```
33/33 [==============================] - 10s 311ms/step - loss: 3.4751 - accuracy: 0.6048
```

```
loss: 347.51%
accuracy: 60.48%
```

g) Analysis

- After training, the model achieved an accuracy of 60.48% on the test dataset.
- Test Loss: 3.4751.
- The model achieved a high accuracy, but also comes with a high loss value.
- It struggles with generalization, as evidenced by the low validation accuracy and high test loss.
- Overfitting is a concern, given the significant difference between training and validation accuracy.

2. Using LSTM for sentiment analysis

From data preprocessing until create the embedding layer, we keep doing the same as sentiment analysis for CNN, the only difference will be on the fifth step

```
from keras.models import Model
from keras.layers import *
from keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from keras.models import Model
from keras import regularizers
sequence_length = data.shape[1]
filter_sizes = [3,4,5]
num_filters = 100
drop = 0.5

inputs = Input(shape=(sequence_length,))
embedding = embedding_layer(inputs)

################## LSTM ONLY ###############################
# reshape = Reshape((sequence_length,EMBEDDING_DIM))(embedding)

################# SINGLE LSTM ####################
# lstm_0 = LSTM(512)(reshape)

# YOU WANNA ADD MORE LSTM LAYERS? UNCOMMENT THIS #
# lstm_2 = LSTM(1024, return_sequences=True)(reshape)
# lstm_1 = LSTM(512, return_sequences=True)(lstm_2)
# lstm_0 = LSTM(256)(lstm_1)

############################################################

################## CRNN ####################################
reshape = Reshape((sequence_length,EMBEDDING_DIM))(embedding)

conv_0 = Conv1D(num_filters, (filter_sizes[0],
),padding="same",activation='relu',kernel_regularizer=regularizers.l2(0
.01))(reshape)
conv_1 = Conv1D(num_filters, (filter_sizes[1],
),padding="same",activation='relu',kernel_regularizer=regularizers.l2(0
.01))(reshape)
conv_2 = Conv1D(num_filters, (filter_sizes[2],
),padding="same",activation='relu',kernel_regularizer=regularizers.l2(0
.01))(reshape)
```

```
conv_0 = MaxPool1D(300)(conv_0)
conv_1 = MaxPool1D(300)(conv_1)
conv_2 = MaxPool1D(300)(conv_2)
# Reshape output to match RNN dimension
# conv_0 = Reshape((-1, num_filters))(conv_0)
# conv_1 = Reshape((-1, num_filters))(conv_1)
# conv_2 = Reshape((-1, num_filters))(conv_2)

concat = concatenate([conv_0, conv_1, conv_2])
concat = Flatten()(concat)
# lstm_0 = LSTM(512)(concat)

# YOU WANNA ADD MORE LSTM LAYERS? UNCOMMENT THIS #
# lstm_2 = LSTM(1024, return_sequences=True)(concat)
# lstm_1 = LSTM(512, return_sequences=True)(lstm_2)
# lstm_0 = LSTM(256)(lstm_1)

###########################################################

dropout = Dropout(drop)(concat)
output = Dense(units=3,
activation='softmax',kernel_regularizer=regularizers.l2(0.01))(dropout)

# this creates a model that includes
model = Model(inputs, output)

adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
epsilon=1e-08)
model.compile(loss='categorical_crossentropy', optimizer=adam,
metrics=['accuracy'])
model.summary()
```

The result of the code:

```
Model: "model_2"
____________________________________________________________________________________
________
 Layer (type)                Output Shape                 Param #    Connected to
====================================================================================
=======
 input_3 (InputLayer)        [(None, 300)]                0          []

 embedding (Embedding)       (None, 300, 400)             3167600    ['input_3[0][0]']

 reshape_2 (Reshape)         (None, 300, 400)             0          ['embedding[2][0]']
```

```
 conv1d_6 (Conv1D)           (None, 300, 100)            120100    ['reshape_2[0][0]']

 conv1d_7 (Conv1D)           (None, 300, 100)            160100    ['reshape_2[0][0]']

 conv1d_8 (Conv1D)           (None, 300, 100)            200100    ['reshape_2[0][0]']

 max_pooling1d_6 (MaxPoolin  (None, 1, 100)              0         ['conv1d_6[0][0]']
 g1D)

 max_pooling1d_7 (MaxPoolin  (None, 1, 100)              0         ['conv1d_7[0][0]']
 g1D)

 max_pooling1d_8 (MaxPoolin  (None, 1, 100)              0         ['conv1d_8[0][0]']
 g1D)

 concatenate_2 (Concatenate  (None, 1, 300)              0
['max_pooling1d_6[0][0]',
 )
'max_pooling1d_7[0][0]',

'max_pooling1d_8[0][0]']

 flatten_2 (Flatten)         (None, 300)                 0         ['concatenate_2[0][0]']

 dropout_2 (Dropout)         (None, 300)                 0         ['flatten_2[0][0]']

 dense_2 (Dense)             (None, 3)                   903       ['dropout_2[0][0]']

==============================================================================================
=======
Total params: 3648803 (13.92 MB)
Trainable params: 3648803 (13.92 MB)
Non-trainable params: 0 (0.00 Byte)
```

Result for testing:

```
Epoch 1/10
16/16 [==============================] - 126s 8s/step - loss: 7.2972 - accuracy: 0.4588 -
val_loss: 7.2478 - val_accuracy: 0.0824
Epoch 2/10
16/16 [==============================] - 105s 7s/step - loss: 5.4355 - accuracy: 0.6235 -
val_loss: 5.5564 - val_accuracy: 0.3078
Epoch 3/10
16/16 [==============================] - 103s 6s/step - loss: 4.5501 - accuracy: 0.6990 -
val_loss: 6.4581 - val_accuracy: 0.0529
Epoch 4/10
16/16 [==============================] - 106s 7s/step - loss: 3.8344 - accuracy: 0.7919 -
val_loss: 5.3081 - val_accuracy: 0.1333
Epoch 5/10
16/16 [==============================] - 106s 7s/step - loss: 3.3220 - accuracy: 0.8328 -
val_loss: 4.8043 - val_accuracy: 0.1608
Epoch 6/10
16/16 [==============================] - 108s 7s/step - loss: 2.9084 - accuracy: 0.8760 -
val_loss: 4.8475 - val_accuracy: 0.0843
Epoch 7/10
16/16 [==============================] - 111s 7s/step - loss: 2.5620 - accuracy: 0.8902 -
val_loss: 4.0822 - val_accuracy: 0.1853
Epoch 8/10
16/16 [==============================] - 104s 7s/step - loss: 2.2393 - accuracy: 0.9203 -
val_loss: 4.4211 - val_accuracy: 0.0833
Epoch 9/10
```

```
16/16 [==============================] - 105s 7s/step - loss: 1.9893 - accuracy: 0.9255 -
val_loss: 3.8701 - val_accuracy: 0.1235
Epoch 10/10
16/16 [==============================] - 104s 7s/step - loss: 1.7620 - accuracy: 0.9350 -
val_loss: 3.7544 - val_accuracy: 0.1157
<keras.src.callbacks.History at 0x7cdf5e085420>

Shape of X train and X validation tensor: (1050, 300)
Shape of label train and validation tensor: (1050, 3)

33/33 [==============================] - 6s 184ms/step - loss: 2.4005 - accuracy: 0.5981

loss: 2.40
accuracy: 59.81%
```

Analysis:

- After training, the model achieved an accuracy of 59.81% on the test dataset.
- Test Loss: 2.40
- The model demonstrates a strong capability to learn from the training data, achieving high accuracy.
- However, it struggles with generalization, as evidenced by the low validation accuracy and high test loss.

3. Using RNN and LSTM for sentiment analysis.

From data preprocessing until creating the embedding layer, we keep doing the same as sentiment analysis for CNN, the only difference will be on the fifth step.

```
from tensorflow.keras.layers import Dense, Input, Embedding, Dropout,
Conv1D, MaxPooling1D, LSTM, concatenate, Flatten, Reshape
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers

sequence_length = data.shape[1]
filter_sizes = [3, 4, 5]
num_filters = 100
drop = 0.5

# Input layer
inputs = Input(shape=(sequence_length,))

# Embedding layer
embedding = embedding_layer(inputs)

# CNN layers
conv_0 = Conv1D(num_filters, filter_sizes[0], activation='relu',
kernel_regularizer=regularizers.l2(0.01))(embedding)
conv_1 = Conv1D(num_filters, filter_sizes[1], activation='relu',
kernel_regularizer=regularizers.l2(0.01))(embedding)
conv_2 = Conv1D(num_filters, filter_sizes[2], activation='relu',
kernel_regularizer=regularizers.l2(0.01))(embedding)

# Max pooling layer
maxpool_0 = MaxPooling1D(sequence_length - filter_sizes[0] + 1,
strides=1)(conv_0)
maxpool_1 = MaxPooling1D(sequence_length - filter_sizes[1] + 1,
strides=1)(conv_1)
maxpool_2 = MaxPooling1D(sequence_length - filter_sizes[2] + 1,
strides=1)(conv_2)

# Concatenate pooled features
merged_tensor = concatenate([maxpool_0, maxpool_1, maxpool_2], axis=1)
flatten = Flatten()(merged_tensor)

# LSTM layer
```

```
# reshaped_tensor = Reshape((num_filters *
len(filter_sizes),))(flatten)
# lstm = LSTM(128)(reshaped_tensor)
reshaped_tensor = Reshape((1, num_filters *
len(filter_sizes)))(flatten)
lstm = LSTM(128)(reshaped_tensor)

# Dropout layer
dropout = Dropout(drop)(lstm)

# Output layer
output = Dense(units=3, activation='softmax',
kernel_regularizer=regularizers.l2(0.01))(dropout)

# Define the model
model = Model(inputs, output)

# Compile the model
adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
epsilon=1e-08)
model.compile(loss='categorical_crossentropy', optimizer=adam,
metrics=['accuracy'])

# Model summary
model.summary()
```

The summary of the code:

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_7 (InputLayer)           [(None, 300)]        0           []

 embedding (Embedding)          (None, 300, 400)     3167600     ['input_7[0][0]']

 conv1d_18 (Conv1D)             (None, 298, 100)     120100      ['embedding[6][0]']

 conv1d_19 (Conv1D)             (None, 297, 100)     160100      ['embedding[6][0]']

 conv1d_20 (Conv1D)             (None, 296, 100)     200100      ['embedding[6][0]']

 max_pooling1d_18 (MaxPooli     (None, 1, 100)       0           ['conv1d_18[0][0]']
 ng1D)

 max_pooling1d_19 (MaxPooli     (None, 1, 100)       0           ['conv1d_19[0][0]']
 ng1D)
```

```
 max_pooling1d_20 (MaxPooli  (None, 1, 100)              0         ['conv1d_20[0][0]']
 ng1D)

 concatenate_6 (Concatenate  (None, 3, 100)              0
['max_pooling1d_18[0][0]',
 )
'max_pooling1d_19[0][0]',

'max_pooling1d_20[0][0]']

 flatten_6 (Flatten)         (None, 300)                 0         ['concatenate_6[0][0]']

 reshape_3 (Reshape)         (None, 1, 300)              0         ['flatten_6[0][0]']

 lstm_3 (LSTM)               (None, 128)                 219648    ['reshape_3[0][0]']

 dropout (Dropout)           (None, 128)                 0         ['lstm_3[0][0]']

 dense (Dense)               (None, 3)                   387       ['dropout[0][0]']

===========================================================================================
=======
Total params: 3867935 (14.76 MB)
Trainable params: 3867935 (14.76 MB)
Non-trainable params: 0 (0.00 Byte)
```

Result for testing:

```
Epoch 1/5
16/16 [==============================] - 104s 6s/step - loss: 4.8533 - accuracy: 0.4679 -
val_loss: 4.5394 - val_accuracy: 0.0000e+00
Epoch 2/5
16/16 [==============================] - 98s 6s/step - loss: 2.7955 - accuracy: 0.6037 -
val_loss: 2.8256 - val_accuracy: 0.0000e+00
Epoch 3/5
16/16 [==============================] - 98s 6s/step - loss: 1.6658 - accuracy: 0.6792 -
val_loss: 2.3028 - val_accuracy: 0.0186
Epoch 4/5
16/16 [==============================] - 95s 6s/step - loss: 1.1416 - accuracy: 0.7608 -
val_loss: 1.9117 - val_accuracy: 0.1520
Epoch 5/5
16/16 [==============================] - 94s 6s/step - loss: 0.8482 - accuracy: 0.8591 -
val_loss: 2.0750 - val_accuracy: 0.1275

<keras.src.callbacks.History at 0x78e707b33910>

loss: 1.25
accuracy: 60.67%
```

Analysis:

- After training, the model achieved an accuracy of 60.67% on the test dataset.
- Test Loss: 1.25

4. Analysis

Methods	Accuracy (%)	Loss (%)
CNN	60.48	347.51
LSTM	59.81	240.0
CNN + LSTM	60.67	125.0

It appears that the combined CNN + LSTM model has the highest accuracy at 60.67%. However, when considering loss, the CNN + LSTM model has the lowest at 125.0%. This suggests that combining CNN and LSTM architectures may offer better performance in terms of both accuracy and loss compared to using either architecture individually.

While the accuracy of the CNN model alone is slightly lower at 60.48%, its loss is significantly higher at 347.51%. On the other hand, the LSTM model has a slightly lower accuracy at 59.81% but a lower loss at 240.0% compared to the CNN model.

Overall, it seems that the CNN + LSTM model provides a balance between accuracy and loss, making it a potentially more robust choice for the given dataset.