GAD-cnn-lstm

September 18, 2024

1 model 1

sara

2 ———————————————————————————————-

3 imports

[2]:
```
import tensorflow as tf
import keras
from keras.models import load_model
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras_tqdm import TQDMNotebookCallback
import numpy as np
np.random.seed(1337)
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 tensorflow.keras.utils import to_categorical
from tensorflow.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
```

3.0.1 Define Callback functions to generate Measures

[3]:
```
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()))
```

4 Experiments to reproduce the results of Table 9

4.0.1 Load pre procssed Data

[4]:
```
with open('../data/pickles/befree_3class_crawl-300d-2M.pickle', 'rb') as handle:
    gene_id_list = pickle.load(handle)
    gene_symbol_list = pickle.load(handle)
    disease_id_list = pickle.load(handle)
    X_train = pickle.load(handle)
    distance1_vectors = pickle.load(handle)
    distance2_vectors = pickle.load(handle)
    Y_train = pickle.load(handle)
    word_list = pickle.load(handle)
    word_vectors = pickle.load(handle)
    word_dict = pickle.load(handle)
```

```
    distance1_dict = pickle.load(handle)
    distance2_dict = pickle.load(handle)
    label_dict = pickle.load(handle)
    MAX_SEQUENCE_LENGTH = pickle.load(handle)
print ("word_vectors",len(word_vectors))
```

```
word_vectors 6766
```

position embedding

[5]:
```
import keras
from keras_pos_embd import TrigPosEmbedding

model = keras.models.Sequential()
model.add(TrigPosEmbedding(
    input_shape=(None,),
    output_dim=20,                      # The dimension of embeddings.
    mode=TrigPosEmbedding.MODE_EXPAND,  # Use `expand` mode
    name='Pos-Embd',
))
model.compile('adam', keras.losses.mae, {})


d1_train_embedded=model.predict(distance1_vectors)

d1_train_embedded.shape

d2_train_embedded=model.predict(distance2_vectors)

d2_train_embedded.shape
```

[5]:
```
(5330, 81, 20)
```

4.0.2 Prepare Word Embedding Layer

[6]:
```
EMBEDDING_DIM=word_vectors.shape[1]
print("EMBEDDING_DIM=",EMBEDDING_DIM)
embedding_matrix=word_vectors

def create_embedding_layer(l2_reg=0.01,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_tra
 ↪l2(l2_reg))

    else:
```

```
        return Embedding(len(word_dict)␣
↪,EMBEDDING_DIM,input_length=MAX_SEQUENCE_LENGTH)
```

EMBEDDING_DIM= 300

attention

[7]:
```
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
```

4.0.3 Create the Model

[8]:
```
# set parameter for metric calculation, 'macro' for multiclass classification
param='macro'
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)


    pos_embedd_1=Input(shape=(MAX_SEQUENCE_LENGTH,20), dtype='float32')
    pos_embedd_2=Input(shape=(MAX_SEQUENCE_LENGTH,20), dtype='float32')

    embedded_sequences =␣
↪concatenate([embedded_sequences,pos_embedd_1,pos_embedd_2])


    x = Conv1D(32, 5, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.1)(x)
    conv_sequence_w5=GlobalMaxPooling1D()(x)    #x = Flatten()(x)
```

```
    x = Conv1D(64, 3, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.1)(x)
    conv_sequence_w4=GlobalMaxPooling1D()(x)    #x = Flatten()(x)



    x = Conv1D(128, 3, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.1)(x)
    conv_sequence_w3=GlobalMaxPooling1D()(x)    #x = Flatten()(x)

    forward = LSTM(100, recurrent_dropout=0.
↪05,return_sequences=True)(embedded_sequences)
    backward = LSTM(100, go_backwards=True,recurrent_dropout=0.
↪05,return_sequences=True)(embedded_sequences)
    attention_forward=attentionNew(forward)
    attention_backward=attentionNew(backward)
    lstm_sequence = concatenate([attention_forward,attention_backward])

    lstm_sequence = Flatten()(lstm_sequence)
    merge =␣
↪concatenate([conv_sequence_w5,conv_sequence_w4,conv_sequence_w3,lstm_sequence])
    x = Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.
↪1))(merge)
    x = Dropout(0.1)(x)
    preds = Dense(3, activation='softmax')(x)
    model = Model(inputs=[sequence_input,␣
↪pos_embedd_1,pos_embedd_2],outputs=preds)
    opt=tf.keras.optimizers.Adam()
    model.
↪compile(loss='categorical_crossentropy',optimizer=opt,metrics=['acc',f1])
    return model
```

[9]:
```
model = build_model()
model.summary()
```

```
Model: "model_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            (None, 81)           0
________________________________________________________________________________
__________________
```

```
embedding_1 (Embedding)         (None, 81, 300)      2029800     input_1[0][0]
________________________________________________________________________________
__________________
input_2 (InputLayer)            (None, 81, 20)       0
________________________________________________________________________________
__________________
input_3 (InputLayer)            (None, 81, 20)       0
________________________________________________________________________________
__________________
concatenate_1 (Concatenate)     (None, 81, 340)      0
embedding_1[0][0]
                                                                 input_2[0][0]
                                                                 input_3[0][0]
________________________________________________________________________________
__________________
lstm_1 (LSTM)                   (None, 81, 100)      176400
concatenate_1[0][0]
________________________________________________________________________________
__________________
lstm_2 (LSTM)                   (None, 81, 100)      176400
concatenate_1[0][0]
________________________________________________________________________________
__________________
lambda_1 (Lambda)               (None, 81, 100)      0           lstm_1[0][0]
________________________________________________________________________________
__________________
lambda_3 (Lambda)               (None, 81, 100)      0           lstm_2[0][0]
________________________________________________________________________________
__________________
permute_1 (Permute)             (None, 100, 81)      0           lambda_1[0][0]
________________________________________________________________________________
__________________
permute_3 (Permute)             (None, 100, 81)      0           lambda_3[0][0]
________________________________________________________________________________
__________________
dense_1 (Dense)                 (None, 100, 81)      6642        permute_1[0][0]
________________________________________________________________________________
__________________
dense_2 (Dense)                 (None, 100, 81)      6642        permute_3[0][0]
________________________________________________________________________________
__________________
permute_2 (Permute)             (None, 81, 100)      0           dense_1[0][0]
________________________________________________________________________________
__________________
permute_4 (Permute)             (None, 81, 100)      0           dense_2[0][0]
________________________________________________________________________________
__________________
conv1d_1 (Conv1D)               (None, 77, 32)       54432
```

```
concatenate_1[0][0]
____________________________________________________________________________________________________
conv1d_2 (Conv1D)                (None, 79, 64)       65344
concatenate_1[0][0]
____________________________________________________________________________________________________
conv1d_3 (Conv1D)                (None, 79, 128)      130688
concatenate_1[0][0]
____________________________________________________________________________________________________
multiply_1 (Multiply)            (None, 81, 100)      0            lambda_1[0][0]
                                                                   permute_2[0][0]
____________________________________________________________________________________________________
multiply_2 (Multiply)            (None, 81, 100)      0            lambda_3[0][0]
                                                                   permute_4[0][0]
____________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)   (None, 25, 32)       0            conv1d_1[0][0]
____________________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)   (None, 26, 64)       0            conv1d_2[0][0]
____________________________________________________________________________________________________
max_pooling1d_3 (MaxPooling1D)   (None, 26, 128)      0            conv1d_3[0][0]
____________________________________________________________________________________________________
lambda_2 (Lambda)                (None, 81, 100)      0
multiply_1[0][0]
____________________________________________________________________________________________________
lambda_4 (Lambda)                (None, 81, 100)      0
multiply_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 25, 32)       0
max_pooling1d_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 26, 64)       0
max_pooling1d_2[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 26, 128)      0
max_pooling1d_3[0][0]
____________________________________________________________________________________________________
```

```
concatenate_2 (Concatenate)     (None, 81, 200)      0            lambda_2[0][0]
                                                                  lambda_4[0][0]
________________________________________________________________________________
__________________
global_max_pooling1d_1 (GlobalM (None, 32)           0            dropout_1[0][0]
________________________________________________________________________________
__________________
global_max_pooling1d_2 (GlobalM (None, 64)           0            dropout_2[0][0]
________________________________________________________________________________
__________________
global_max_pooling1d_3 (GlobalM (None, 128)          0            dropout_3[0][0]
________________________________________________________________________________
__________________
flatten_1 (Flatten)             (None, 16200)        0
concatenate_2[0][0]
________________________________________________________________________________
__________________
concatenate_3 (Concatenate)     (None, 16424)        0
global_max_pooling1d_1[0][0]
global_max_pooling1d_2[0][0]
global_max_pooling1d_3[0][0]
                                                                  flatten_1[0][0]
________________________________________________________________________________
__________________
dense_3 (Dense)                 (None, 64)           1051200
concatenate_3[0][0]
________________________________________________________________________________
__________________
dropout_4 (Dropout)             (None, 64)           0            dense_3[0][0]
________________________________________________________________________________
__________________
dense_4 (Dense)                 (None, 3)            195          dropout_4[0][0]
================================================================================
==================
Total params: 3,697,743
Trainable params: 1,667,943
Non-trainable params: 2,029,800
________________________________________________________________________________
__________________
```

[10]:

```
validation_split_rate = 0.1
skf = StratifiedKFold(n_splits=5,shuffle=True, random_state=42)
Y = [np.argmax(y, axis=None, out=None) for y in Y_train]
for tr_index, te_index in skf.split(X_train,Y):
    test_index = te_index
    train_index = tr_index
```

```
trainRate = (len(train_index)/len(Y))*100
testRate = (len(test_index)/len(Y))*100
print ("TrainRate:{:.2f}% testRate:{:.2f}% validation:{:.2f}%  ".
 ↪format(trainRate,testRate, trainRate*validation_split_rate))
X_train, X_test = X_train[train_index], X_train[test_index]
pos_train1, pos_test1 = d1_train_embedded[train_index],␣
 ↪d1_train_embedded[test_index]
pos_train2, pos_test2 = d2_train_embedded[train_index],␣
 ↪d2_train_embedded[test_index]
y_train, y_test = Y_train[train_index], Y_train[test_index]



# # Saving the training data split as a pickle file
# training_data = {
#     'X_train': X_train,
#     'pos_train1': pos_train1,
#     'pos_train2': pos_train2,
#     'y_train': y_train
# }

# with open('training_data.pkl', 'wb') as f:
#     pickle.dump(training_data, f)

# # Saving the testing data split as a pickle file
# testing_data = {
#     'X_test': X_test,
#     'pos_test1': pos_test1,
#     'pos_test2': pos_test2,
#     'y_test': y_test
# }

# with open('testing_data.pkl', 'wb') as f:
#     pickle.dump(testing_data, f)
```

```
TrainRate:80.00% testRate:20.00% validation:8.00%
```

[11]:
```
# Load the training data from the pickle file
with open('training_data.pkl', 'rb') as f:
    train_data = pickle.load(f)

# Load the testing data from the pickle file
with open('testing_data.pkl', 'rb') as f:
    test_data = pickle.load(f)

# Extract data from the loaded dictionaries
```

```
X_train = train_data['X_train']
pos_train1 = train_data['pos_train1']
pos_train2 = train_data['pos_train2']
y_train = train_data['y_train']


X_test = test_data['X_test']
pos_test1 = test_data['pos_test1']
pos_test2 = test_data['pos_test2']
y_test = test_data['y_test']
print(X_train.shape)
print(X_test.shape)
```

```
(4264, 81)
(1066, 81)
```

4.0.4 Run the Evaluation on the test dataset

[12]:
```
MaxEpochs =50
batchsize =32
validation_split_rate = 0.1
history=model.fit([X_train,pos_train1,pos_train2],␣
  ↪y_train,validation_split=validation_split_rate ,epochs=MaxEpochs,␣
  ↪batch_size=batchsize,verbose=1)
```

```
Train on 3837 samples, validate on 427 samples
Epoch 1/50
3837/3837 [==============================] - 134s 35ms/step - loss: 2780.3397 -
acc: 0.4522 - f1: 0.1557 - val_loss: 2779.6699 - val_acc: 0.4333 - val_f1:
0.2862
Epoch 2/50
3837/3837 [==============================] - 16s 4ms/step - loss: 2779.6495 -
acc: 0.4717 - f1: 0.2152 - val_loss: 2779.6760 - val_acc: 0.4379 - val_f1:
0.0000e+00
Epoch 3/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.6943 -
acc: 0.4756 - f1: 0.2328 - val_loss: 2779.6339 - val_acc: 0.4520 - val_f1:
0.1126
Epoch 4/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.6121 -
acc: 0.4782 - f1: 0.2640 - val_loss: 2779.6475 - val_acc: 0.4450 - val_f1:
0.1833
Epoch 5/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.5923 -
acc: 0.5046 - f1: 0.3184 - val_loss: 2779.6017 - val_acc: 0.4754 - val_f1:
0.3563
Epoch 6/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.5891 -
```

```
acc: 0.5207 - f1: 0.3936 - val_loss: 2779.5443 - val_acc: 0.5269 - val_f1:
0.3319
Epoch 7/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.5729 -
acc: 0.5747 - f1: 0.4789 - val_loss: 2779.4975 - val_acc: 0.5808 - val_f1:
0.4736
Epoch 8/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.4549 -
acc: 0.6044 - f1: 0.5434 - val_loss: 2779.4850 - val_acc: 0.5691 - val_f1:
0.5268
Epoch 9/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.4592 -
acc: 0.6083 - f1: 0.5556 - val_loss: 2779.4880 - val_acc: 0.6136 - val_f1:
0.6029
Epoch 10/50
3837/3837 [==============================] - 16s 4ms/step - loss: 2779.4649 -
acc: 0.6552 - f1: 0.6300 - val_loss: 2779.4806 - val_acc: 0.5902 - val_f1:
0.5760
Epoch 11/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.4065 -
acc: 0.6727 - f1: 0.6511 - val_loss: 2779.4394 - val_acc: 0.6487 - val_f1:
0.6292
Epoch 12/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.4547 -
acc: 0.6190 - f1: 0.5952 - val_loss: 2779.4847 - val_acc: 0.5644 - val_f1:
0.5329
Epoch 13/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.3941 -
acc: 0.6823 - f1: 0.6671 - val_loss: 2779.4446 - val_acc: 0.6089 - val_f1:
0.6010
Epoch 14/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.3539 -
acc: 0.6974 - f1: 0.6773 - val_loss: 2779.4530 - val_acc: 0.6206 - val_f1:
0.6101
Epoch 15/50
3837/3837 [==============================] - 16s 4ms/step - loss: 2779.4770 -
acc: 0.7081 - f1: 0.6915 - val_loss: 2779.4025 - val_acc: 0.6768 - val_f1:
0.6558
Epoch 16/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.3432 -
acc: 0.7300 - f1: 0.7194 - val_loss: 2779.3852 - val_acc: 0.6838 - val_f1:
0.6918
Epoch 17/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.2838 -
acc: 0.7482 - f1: 0.7383 - val_loss: 2779.4634 - val_acc: 0.6651 - val_f1:
0.6513
Epoch 18/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.2747 -
```

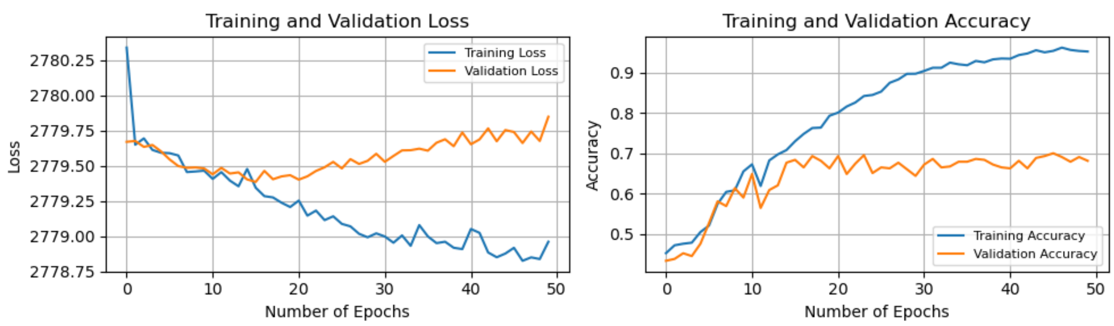
```
acc: 0.7628 - f1: 0.7544 - val_loss: 2779.4039 - val_acc: 0.6932 - val_f1:
0.6938
Epoch 19/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.2369 -
acc: 0.7639 - f1: 0.7556 - val_loss: 2779.4242 - val_acc: 0.6815 - val_f1:
0.6746
Epoch 20/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.2064 -
acc: 0.7933 - f1: 0.7838 - val_loss: 2779.4328 - val_acc: 0.6628 - val_f1:
0.6495
Epoch 21/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.2524 -
acc: 0.8011 - f1: 0.7928 - val_loss: 2779.4012 - val_acc: 0.6932 - val_f1:
0.6983
Epoch 22/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.1452 -
acc: 0.8163 - f1: 0.8143 - val_loss: 2779.4248 - val_acc: 0.6487 - val_f1:
0.6466
Epoch 23/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.1810 -
acc: 0.8262 - f1: 0.8260 - val_loss: 2779.4629 - val_acc: 0.6745 - val_f1:
0.6675
Epoch 24/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.1134 -
acc: 0.8423 - f1: 0.8412 - val_loss: 2779.4890 - val_acc: 0.6956 - val_f1:
0.6919
Epoch 25/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.1405 -
acc: 0.8449 - f1: 0.8419 - val_loss: 2779.5274 - val_acc: 0.6511 - val_f1:
0.6544
Epoch 26/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.0867 -
acc: 0.8533 - f1: 0.8522 - val_loss: 2779.4808 - val_acc: 0.6651 - val_f1:
0.6723
Epoch 27/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.0690 -
acc: 0.8752 - f1: 0.8722 - val_loss: 2779.5467 - val_acc: 0.6628 - val_f1:
0.6458
Epoch 28/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.0163 -
acc: 0.8835 - f1: 0.8817 - val_loss: 2779.5127 - val_acc: 0.6768 - val_f1:
0.6767
Epoch 29/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9911 -
acc: 0.8973 - f1: 0.8962 - val_loss: 2779.5349 - val_acc: 0.6604 - val_f1:
0.6548
Epoch 30/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.0193 -
```

```
acc: 0.8976 - f1: 0.8963 - val_loss: 2779.5848 - val_acc: 0.6440 - val_f1:
0.6381
Epoch 31/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9969 -
acc: 0.9046 - f1: 0.9042 - val_loss: 2779.5279 - val_acc: 0.6721 - val_f1:
0.6759
Epoch 32/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9527 -
acc: 0.9124 - f1: 0.9126 - val_loss: 2779.5695 - val_acc: 0.6862 - val_f1:
0.6887
Epoch 33/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.0045 -
acc: 0.9124 - f1: 0.9107 - val_loss: 2779.6091 - val_acc: 0.6651 - val_f1:
0.6684
Epoch 34/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9306 -
acc: 0.9252 - f1: 0.9258 - val_loss: 2779.6103 - val_acc: 0.6674 - val_f1:
0.6704
Epoch 35/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.0780 -
acc: 0.9210 - f1: 0.9219 - val_loss: 2779.6212 - val_acc: 0.6792 - val_f1:
0.6855
Epoch 36/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9977 -
acc: 0.9187 - f1: 0.9176 - val_loss: 2779.6071 - val_acc: 0.6792 - val_f1:
0.6884
Epoch 37/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9499 -
acc: 0.9294 - f1: 0.9294 - val_loss: 2779.6633 - val_acc: 0.6862 - val_f1:
0.6770
Epoch 38/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9597 -
acc: 0.9260 - f1: 0.9249 - val_loss: 2779.6880 - val_acc: 0.6838 - val_f1:
0.6819
Epoch 39/50
3837/3837 [==============================] - 16s 4ms/step - loss: 2778.9169 -
acc: 0.9333 - f1: 0.9329 - val_loss: 2779.6387 - val_acc: 0.6721 - val_f1:
0.6702
Epoch 40/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9074 -
acc: 0.9354 - f1: 0.9365 - val_loss: 2779.7365 - val_acc: 0.6651 - val_f1:
0.6635
Epoch 41/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.0502 -
acc: 0.9348 - f1: 0.9313 - val_loss: 2779.6527 - val_acc: 0.6628 - val_f1:
0.6724
Epoch 42/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2779.0230 -
```

```
acc: 0.9440 - f1: 0.9453 - val_loss: 2779.6870 - val_acc: 0.6815 - val_f1:
0.6756
Epoch 43/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.8834 -
acc: 0.9479 - f1: 0.9487 - val_loss: 2779.7654 - val_acc: 0.6628 - val_f1:
0.6578
Epoch 44/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.8493 -
acc: 0.9560 - f1: 0.9548 - val_loss: 2779.6741 - val_acc: 0.6885 - val_f1:
0.6798
Epoch 45/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.8757 -
acc: 0.9507 - f1: 0.9514 - val_loss: 2779.7536 - val_acc: 0.6932 - val_f1:
0.6878
Epoch 46/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9165 -
acc: 0.9544 - f1: 0.9544 - val_loss: 2779.7397 - val_acc: 0.7002 - val_f1:
0.6947
Epoch 47/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.8242 -
acc: 0.9625 - f1: 0.9620 - val_loss: 2779.6613 - val_acc: 0.6909 - val_f1:
0.6981
Epoch 48/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.8486 -
acc: 0.9567 - f1: 0.9577 - val_loss: 2779.7434 - val_acc: 0.6792 - val_f1:
0.6725
Epoch 49/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.8367 -
acc: 0.9541 - f1: 0.9539 - val_loss: 2779.6761 - val_acc: 0.6909 - val_f1:
0.6882
Epoch 50/50
3837/3837 [==============================] - 15s 4ms/step - loss: 2778.9597 -
acc: 0.9528 - f1: 0.9528 - val_loss: 2779.8483 - val_acc: 0.6815 - val_f1:
0.6845
```

[13]:
```python
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()
```



[14]:
```
import torch
from sklearn.metrics import accuracy_score, classification_report,␣
↪precision_recall_fscore_support

# Predict on the training dataset
train_predicted = np.argmax(model.predict([X_train, pos_train1, pos_train2]),␣
↪axis=1)
y_train_to_label = np.argmax(y_train, axis=1)

# Calculate accuracy, precision, recall, and F1-score for the training data
```

```
train_accuracy = accuracy_score(y_train_to_label, train_predicted)
train_prec, train_reca, train_fscore, _ =␣
↪precision_recall_fscore_support(y_train_to_label, train_predicted,␣
↪average=param)

# Print the classification report for the training data
print("Training Classification Report:")
print(classification_report(y_train_to_label, train_predicted))

# Print the precision, recall, and F1-score for the training data
print("Training Accuracy: {:.2f}%".format(train_accuracy * 100))
print("Training Precision: {:.2f}%".format(train_prec * 100))
print("Training Recall: {:.2f}%".format(train_reca * 100))
print("Training F1 Score: {:.2f}%".format(train_fscore * 100))
```

```
Training Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      2023
           1       0.97      0.91      0.94      1468
           2       0.93      0.94      0.93       773

    accuracy                           0.94      4264
   macro avg       0.94      0.94      0.94      4264
weighted avg       0.94      0.94      0.94      4264

Training Accuracy: 94.18%
Training Precision: 94.32%
Training Recall: 93.76%
Training F1 Score: 93.99%
```

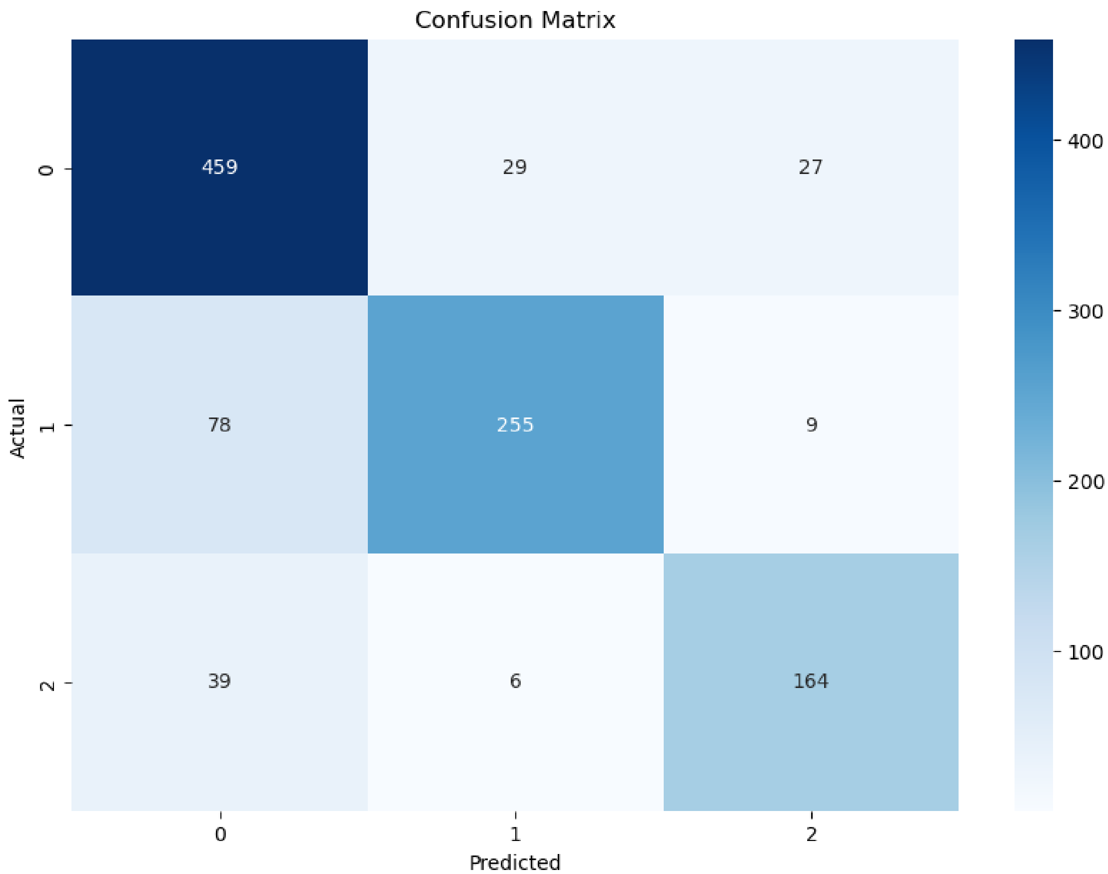
[15]:
```
predicted = np.argmax(model.predict([X_test,pos_test1,pos_test2]), axis=1)
y_test_to_label = np.argmax(y_test, axis=1)
prec, reca, fscore, sup = precision_recall_fscore_support(y_test_to_label,␣
↪predicted, average=param)
# Generate the classification report as a dictionary
report_dict = classification_report(y_test_to_label, 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     0.7969    0.8913    0.8414       515
           1     0.8793    0.7456    0.8070       342
           2     0.8200    0.7847    0.8020       209

    accuracy                         0.8236      1066
   macro avg     0.8321    0.8072    0.8168      1066
weighted avg     0.8279    0.8236    0.8226      1066

 Precision:83.21% Recall:80.72% Fscore:81.68%
```

[16]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix,␣
↪precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold
# Calculate and visualize the confusion matrix
cm = confusion_matrix(y_test_to_label, predicted)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['0', '1','2'],␣
↪yticklabels=['0', '1','2'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print precision, recall, and f-score
prec, reca, fscore, sup = precision_recall_fscore_support(y_test_to_label,␣
↪predicted, average=param)
```

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




```
Precision:83.21% Recall:80.72% Fscore:81.68%
```

```
[ ]:
```