BeFree-2class_EUADR-GRU

September 17, 2024

1 Evaluation using the BeFree corpus

1.0.1 EUADR dataset

The EU-ADR dataset contains annotations on drugs, diseases, genes and proteins, and associations between them. In this study, we used only GDAs to evaluate the method. Each association is classified according to its level of certainty as positive association (PA), negative association (NA), speculative association (SA); or false association (FA). The EU-ADR corpus is based on 100 MEDLINE abstracts for each association set, and its annotation was conducted by three experts.

2 —————————————————————————————-

3 imports

[1]:
```
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 keras.layers import Embedding, Flatten,LSTM, GRU
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from keras.layers import Embedding, Flatten,LSTM
from keras.layers.convolutional import Conv2D, MaxPooling2D
from tensorflow.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 *
```

Using TensorFlow backend.

3.0.1 Define Callback functions to generate Mesures

[2]:
```
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 8

4.0.1 Load Prerocssed Data

[3]:
```
with open('../data/pickles/befree_EUADR_2class_PubMed-and-PMC-w2v.pickle',␣
↪'rb') as handle:

    W_train = pickle.load(handle)
    print("W_train",len(W_train))
    d1_train = pickle.load(handle)
    print("d1_train",len(d1_train))
```

```
    d2_train = pickle.load(handle)
    print("d2_train",len(d2_train))
    Y_train = pickle.load(handle)
    print("Y_train",len(d2_train))
    Tr_word_list = pickle.load(handle)
    print("Tr_word_list",len(d2_train))
    word_vectors = pickle.load(handle)
    print("word_vectors",len(word_vectors))
    word_dict = pickle.load(handle)
    print("word_dict",len(word_dict))
    d1_dict = pickle.load(handle)
    print("d1_dict",len(d1_dict))
    d2_dict = pickle.load(handle)
    print("d2_dict",len(d2_dict))
    label_dict = pickle.load(handle)
    print("label_dict",len(label_dict))
    MAX_SEQUENCE_LENGTH = pickle.load(handle)
    print("MAX_SEQUENCE_LENGTH",MAX_SEQUENCE_LENGTH)
```

```
W_train 355
d1_train 355
d2_train 355
Y_train 355
Tr_word_list 355
word_vectors 1355
word_dict 1355
d1_dict 169
d2_dict 171
label_dict 4
MAX_SEQUENCE_LENGTH 102
```

4.0.2 Create Position Embedding Vectors

[4]:
```
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, {})
model.summary()

d1_train_embedded=model.predict(d1_train)
```

```
d1_train_embedded.shape

d2_train_embedded=model.predict(d2_train)

d2_train_embedded.shape
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Pos-Embd (TrigPosEmbedding)  (None, None, 20)          0
=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
```

[4]:
```
(355, 102, 20)
```

4.0.3 Prepare Word Embedding Layer

[5]:
```
EMBEDDING_DIM=word_vectors.shape[1]
embedding_matrix=word_vectors
print(EMBEDDING_DIM)
print(len(embedding_matrix))

param='binary'

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)
```

```
200
1355
```

4.0.4 Prepare Attention Mechanism

[6]:
```
INPUT_DIM = 2
TIME_STEPS = MAX_SEQUENCE_LENGTH
```

[7]:
```
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
```

4.0.5 Create the Model

[8]:
```
dropRate=0.1
param='binary'
def build_model_positionAttention():

    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(128, 7, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.5)(x)

    x = Conv1D(64, 5, activation='relu')(x)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.5)(x)
    conv_sequence_7=GlobalMaxPooling1D()(x)    #x = Flatten()(x)




    forward = GRU(100, recurrent_dropout=0.05)(embedded_sequences)
```

```
    backward = GRU(100, go_backwards=True,recurrent_dropout=0.
↪05)(embedded_sequences)
    lstm_sequence = concatenate([forward,backward])

    merge = concatenate([conv_sequence_7,lstm_sequence])



    x = Dropout(dropRate)(x)
    x = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.
↪05))(merge)
    x = Dropout(0.1)(x)
    preds = Dense(2, activation='softmax')(x)
    model = Model(inputs=[sequence_input,␣
↪pos_embedd_1,pos_embedd_2],outputs=preds)
    model.
↪compile(loss='binary_crossentropy',optimizer='adam',metrics=['acc',f1])

    #model.summary()
    return model
```

[9]:

```
model = build_model_positionAttention()
model.summary()
```

```
Model: "model_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            (None, 102)          0
________________________________________________________________________________
__________________
embedding_1 (Embedding)         (None, 102, 200)     271000      input_1[0][0]
________________________________________________________________________________
__________________
input_2 (InputLayer)            (None, 102, 20)      0
________________________________________________________________________________
__________________
input_3 (InputLayer)            (None, 102, 20)      0
________________________________________________________________________________
__________________
concatenate_1 (Concatenate)     (None, 102, 240)     0
embedding_1[0][0]
                                                                 input_2[0][0]
                                                                 input_3[0][0]
```

```
__________________________________________________________________________________
__________________
conv1d_1 (Conv1D)              (None, 96, 128)      215168
concatenate_1[0][0]
__________________________________________________________________________________
__________________
max_pooling1d_1 (MaxPooling1D)  (None, 32, 128)      0            conv1d_1[0][0]
__________________________________________________________________________________
__________________
dropout_1 (Dropout)            (None, 32, 128)      0
max_pooling1d_1[0][0]
__________________________________________________________________________________
__________________
conv1d_2 (Conv1D)              (None, 28, 64)       41024        dropout_1[0][0]
__________________________________________________________________________________
__________________
max_pooling1d_2 (MaxPooling1D)  (None, 9, 64)        0            conv1d_2[0][0]
__________________________________________________________________________________
__________________
dropout_2 (Dropout)            (None, 9, 64)        0
max_pooling1d_2[0][0]
__________________________________________________________________________________
__________________
gru_1 (GRU)                    (None, 100)          102300
concatenate_1[0][0]
__________________________________________________________________________________
__________________
gru_2 (GRU)                    (None, 100)          102300
concatenate_1[0][0]
__________________________________________________________________________________
__________________
global_max_pooling1d_1 (GlobalM (None, 64)           0            dropout_2[0][0]
__________________________________________________________________________________
__________________
concatenate_2 (Concatenate)    (None, 200)          0            gru_1[0][0]
                                                                  gru_2[0][0]
__________________________________________________________________________________
__________________
concatenate_3 (Concatenate)    (None, 264)          0
global_max_pooling1d_1[0][0]
concatenate_2[0][0]
__________________________________________________________________________________
__________________
dense_1 (Dense)                (None, 256)          67840
concatenate_3[0][0]
__________________________________________________________________________________
__________________
dropout_4 (Dropout)            (None, 256)          0            dense_1[0][0]
```

```
--------------------------------------------------------------------------------
------------------
dense_2 (Dense)                 (None, 2)            514         dropout_4[0][0]
================================================================================
==================
Total params: 800,146
Trainable params: 529,146
Non-trainable params: 271,000
--------------------------------------------------------------------------------
------------------
```

4.0.6 Run the Evaluation using 10 fold Cross Validation

[10]:

```
validation_split_rate=0.1
skf = StratifiedKFold(n_splits=5, random_state=None)
Y = [np.argmax(y, axis=None, out=None) for y in Y_train]
#print(len(Y))
all_histories=[]
for trI, teI in skf.split(W_train,Y):
    train_index =trI
    test_index  =teI
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))
#print(train_index, test_index)
X_train, X_test = W_train[train_index], W_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)
```

```
(284, 102)
```

[12]:

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

```
Train on 255 samples, validate on 29 samples
Epoch 1/50
255/255 [==============================] - 247s 969ms/step - loss: 586.3168 -
acc: 0.6118 - f1: 0.6119 - val_loss: 584.6624 - val_acc: 0.9310 - val_f1: 0.9310
Epoch 2/50
```

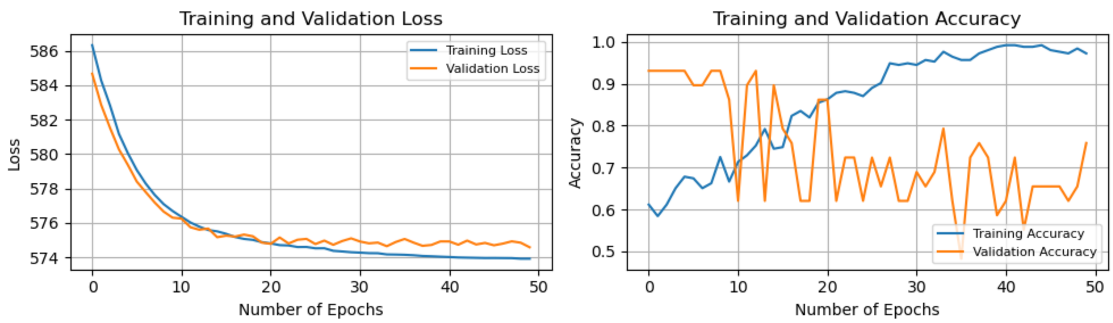
```
255/255 [==============================] - 3s 11ms/step - loss: 584.2836 - acc:
0.5843 - f1: 0.5834 - val_loss: 582.8656 - val_acc: 0.9310 - val_f1: 0.9310
Epoch 3/50
255/255 [==============================] - 3s 11ms/step - loss: 582.8166 - acc:
0.6118 - f1: 0.6114 - val_loss: 581.5160 - val_acc: 0.9310 - val_f1: 0.9310
Epoch 4/50
255/255 [==============================] - 3s 11ms/step - loss: 581.1655 - acc:
0.6510 - f1: 0.6510 - val_loss: 580.2663 - val_acc: 0.9310 - val_f1: 0.9310
Epoch 5/50
255/255 [==============================] - 3s 12ms/step - loss: 580.0452 - acc:
0.6784 - f1: 0.6779 - val_loss: 579.3665 - val_acc: 0.9310 - val_f1: 0.9310
Epoch 6/50
255/255 [==============================] - 3s 11ms/step - loss: 579.0672 - acc:
0.6745 - f1: 0.6743 - val_loss: 578.3984 - val_acc: 0.8966 - val_f1: 0.8966
Epoch 7/50
255/255 [==============================] - 3s 11ms/step - loss: 578.2824 - acc:
0.6510 - f1: 0.6513 - val_loss: 577.7879 - val_acc: 0.8966 - val_f1: 0.8966
Epoch 8/50
255/255 [==============================] - 3s 11ms/step - loss: 577.6288 - acc:
0.6627 - f1: 0.6629 - val_loss: 577.1880 - val_acc: 0.9310 - val_f1: 0.9310
Epoch 9/50
255/255 [==============================] - 3s 11ms/step - loss: 577.0930 - acc:
0.7255 - f1: 0.7254 - val_loss: 576.6606 - val_acc: 0.9310 - val_f1: 0.9310
Epoch 10/50
255/255 [==============================] - 3s 11ms/step - loss: 576.6954 - acc:
0.6667 - f1: 0.6667 - val_loss: 576.3065 - val_acc: 0.8621 - val_f1: 0.8621
Epoch 11/50
255/255 [==============================] - 3s 11ms/step - loss: 576.3565 - acc:
0.7137 - f1: 0.7140 - val_loss: 576.2506 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 12/50
255/255 [==============================] - 3s 11ms/step - loss: 576.0281 - acc:
0.7294 - f1: 0.7292 - val_loss: 575.7520 - val_acc: 0.8966 - val_f1: 0.8966
Epoch 13/50
255/255 [==============================] - 3s 11ms/step - loss: 575.7834 - acc:
0.7529 - f1: 0.7532 - val_loss: 575.6051 - val_acc: 0.9310 - val_f1: 0.9310
Epoch 14/50
255/255 [==============================] - 3s 11ms/step - loss: 575.5885 - acc:
0.7922 - f1: 0.7920 - val_loss: 575.6749 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 15/50
255/255 [==============================] - 3s 11ms/step - loss: 575.5112 - acc:
0.7451 - f1: 0.7448 - val_loss: 575.1725 - val_acc: 0.8966 - val_f1: 0.8966
Epoch 16/50
255/255 [==============================] - 3s 12ms/step - loss: 575.3687 - acc:
0.7490 - f1: 0.7486 - val_loss: 575.2613 - val_acc: 0.7931 - val_f1: 0.7931
Epoch 17/50
255/255 [==============================] - 3s 11ms/step - loss: 575.1839 - acc:
0.8235 - f1: 0.8233 - val_loss: 575.2097 - val_acc: 0.7586 - val_f1: 0.7586
Epoch 18/50
```

```
255/255 [==============================] - 3s 11ms/step - loss: 575.0736 - acc:
0.8353 - f1: 0.8354 - val_loss: 575.3238 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 19/50
255/255 [==============================] - 3s 11ms/step - loss: 575.0159 - acc:
0.8196 - f1: 0.8196 - val_loss: 575.2264 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 20/50
255/255 [==============================] - 3s 11ms/step - loss: 574.8863 - acc:
0.8549 - f1: 0.8552 - val_loss: 574.8587 - val_acc: 0.8621 - val_f1: 0.8621
Epoch 21/50
255/255 [==============================] - 3s 11ms/step - loss: 574.8127 - acc:
0.8627 - f1: 0.8629 - val_loss: 574.7879 - val_acc: 0.8621 - val_f1: 0.8621
Epoch 22/50
255/255 [==============================] - 3s 12ms/step - loss: 574.7130 - acc:
0.8784 - f1: 0.8787 - val_loss: 575.1560 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 23/50
255/255 [==============================] - 3s 11ms/step - loss: 574.6934 - acc:
0.8824 - f1: 0.8826 - val_loss: 574.8069 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 24/50
255/255 [==============================] - 3s 10ms/step - loss: 574.6022 - acc:
0.8784 - f1: 0.8784 - val_loss: 575.0242 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 25/50
255/255 [==============================] - 3s 11ms/step - loss: 574.6069 - acc:
0.8706 - f1: 0.8706 - val_loss: 575.0751 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 26/50
255/255 [==============================] - 3s 10ms/step - loss: 574.5270 - acc:
0.8902 - f1: 0.8900 - val_loss: 574.7743 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 27/50
255/255 [==============================] - 3s 11ms/step - loss: 574.5318 - acc:
0.9020 - f1: 0.9017 - val_loss: 574.9752 - val_acc: 0.6552 - val_f1: 0.6552
Epoch 28/50
255/255 [==============================] - 3s 11ms/step - loss: 574.3820 - acc:
0.9490 - f1: 0.9491 - val_loss: 574.7321 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 29/50
255/255 [==============================] - 3s 11ms/step - loss: 574.3453 - acc:
0.9451 - f1: 0.9453 - val_loss: 574.9487 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 30/50
255/255 [==============================] - 3s 11ms/step - loss: 574.3013 - acc:
0.9490 - f1: 0.9488 - val_loss: 575.1059 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 31/50
255/255 [==============================] - 3s 11ms/step - loss: 574.2775 - acc:
0.9451 - f1: 0.9451 - val_loss: 574.9208 - val_acc: 0.6897 - val_f1: 0.6897
Epoch 32/50
255/255 [==============================] - 3s 11ms/step - loss: 574.2475 - acc:
0.9569 - f1: 0.9570 - val_loss: 574.8170 - val_acc: 0.6552 - val_f1: 0.6552
Epoch 33/50
255/255 [==============================] - 3s 11ms/step - loss: 574.2427 - acc:
0.9529 - f1: 0.9529 - val_loss: 574.8594 - val_acc: 0.6897 - val_f1: 0.6897
Epoch 34/50
```

```
255/255 [==============================] - 3s 11ms/step - loss: 574.1734 - acc:
0.9765 - f1: 0.9763 - val_loss: 574.6503 - val_acc: 0.7931 - val_f1: 0.7931
Epoch 35/50
255/255 [==============================] - 3s 12ms/step - loss: 574.1648 - acc:
0.9647 - f1: 0.9646 - val_loss: 574.8932 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 36/50
255/255 [==============================] - 3s 11ms/step - loss: 574.1520 - acc:
0.9569 - f1: 0.9567 - val_loss: 575.0707 - val_acc: 0.4828 - val_f1: 0.4828
Epoch 37/50
255/255 [==============================] - 3s 10ms/step - loss: 574.1258 - acc:
0.9569 - f1: 0.9570 - val_loss: 574.8579 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 38/50
255/255 [==============================] - 3s 11ms/step - loss: 574.0835 - acc:
0.9725 - f1: 0.9725 - val_loss: 574.6727 - val_acc: 0.7586 - val_f1: 0.7586
Epoch 39/50
255/255 [==============================] - 3s 11ms/step - loss: 574.0661 - acc:
0.9804 - f1: 0.9805 - val_loss: 574.7231 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 40/50
255/255 [==============================] - 3s 12ms/step - loss: 574.0421 - acc:
0.9882 - f1: 0.9882 - val_loss: 574.9271 - val_acc: 0.5862 - val_f1: 0.5862
Epoch 41/50
255/255 [==============================] - 3s 11ms/step - loss: 574.0221 - acc:
0.9922 - f1: 0.9921 - val_loss: 574.9259 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 42/50
255/255 [==============================] - 3s 11ms/step - loss: 573.9968 - acc:
0.9922 - f1: 0.9921 - val_loss: 574.7363 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 43/50
255/255 [==============================] - 3s 11ms/step - loss: 573.9852 - acc:
0.9882 - f1: 0.9882 - val_loss: 574.9750 - val_acc: 0.5517 - val_f1: 0.5517
Epoch 44/50
255/255 [==============================] - 3s 11ms/step - loss: 573.9746 - acc:
0.9882 - f1: 0.9883 - val_loss: 574.7511 - val_acc: 0.6552 - val_f1: 0.6552
Epoch 45/50
255/255 [==============================] - 3s 11ms/step - loss: 573.9641 - acc:
0.9922 - f1: 0.9922 - val_loss: 574.8412 - val_acc: 0.6552 - val_f1: 0.6552
Epoch 46/50
255/255 [==============================] - 3s 11ms/step - loss: 573.9654 - acc:
0.9804 - f1: 0.9805 - val_loss: 574.7034 - val_acc: 0.6552 - val_f1: 0.6552
Epoch 47/50
255/255 [==============================] - 3s 11ms/step - loss: 573.9612 - acc:
0.9765 - f1: 0.9764 - val_loss: 574.8004 - val_acc: 0.6552 - val_f1: 0.6552
Epoch 48/50
255/255 [==============================] - 3s 11ms/step - loss: 573.9572 - acc:
0.9725 - f1: 0.9723 - val_loss: 574.9329 - val_acc: 0.6207 - val_f1: 0.6207
Epoch 49/50

255/255 [==============================] - 3s 11ms/step - loss: 573.9252 - acc:
0.9843 - f1: 0.9844 - val_loss: 574.8527 - val_acc: 0.6552 - val_f1: 0.6552
```

```
Epoch 50/50
255/255 [==============================] - 3s 11ms/step - loss: 573.9256 - acc:
0.9725 - f1: 0.9727 - val_loss: 574.5879 - val_acc: 0.7586 - val_f1: 0.7586
```

[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]:
```python
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       0.92      0.97      0.95        89
           1       0.98      0.96      0.97       195

    accuracy                           0.96       284
   macro avg       0.95      0.97      0.96       284
weighted avg       0.97      0.96      0.96       284

Training Accuracy: 96.48%
```

```
Training Precision: 98.43%
Training Recall: 96.41%
Training F1 Score: 97.41%
```

[ ]:

```

```

[16]:

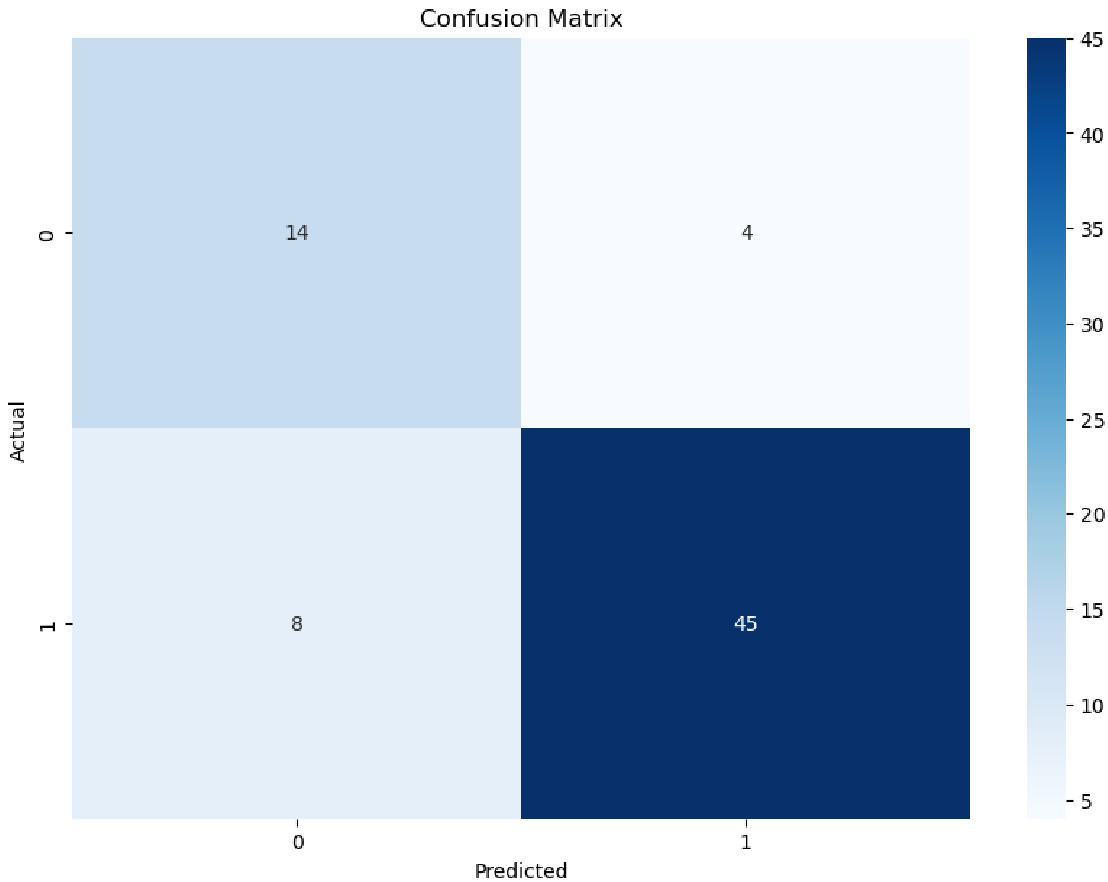
```
predicted = np.argmax(model.predict([X_test,pos_test1,pos_test2]), axis=1)
y_test_to_label= np.argmax(y_test, axis=1)
prec, reca, fscore, sup = precision_recall_fscore_support(y_test_to_label,␣
 ↪predicted, average=param)



# 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     0.6364    0.7778    0.7000        18
           1     0.9184    0.8491    0.8824        53

    accuracy                         0.8310        71
```

```
   macro avg     0.7774    0.8134    0.7912        71
weighted avg     0.8469    0.8310    0.8361        71

 Precision:91.84% Recall:84.91% Fscore:88.24%
```

[17]:

```python
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'],␣
 ↪yticklabels=['0', '1'])
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:91.84% Recall:84.91% Fscore:88.24%
```

[ ]: