```
import re # regex
import sklearn
import pandas as pd # tables
import matplotlib.pyplot as plt # plots
import seaborn as sns # plots
import numpy as np # operations with arrays and matrices
from sklearn.model_selection import train_test_split


# reading the dataset
'''train = pd.read_csv('train.txt', header=None, sep=';', names=['Lines','Emotions'], encoding='utf-8')
test = pd.read_csv('test.txt', header=None, sep =';', names=['Lines','Emotions'], encoding='utf-8')
validation = pd.read_csv('val.txt', header=None, sep=';', names=['Lines','Emotions'], encoding='utf-8')'''
```

```
'train = pd.read_csv('train.txt', header=None, sep=';', names=['Lines','Emotions'], encoding='utf-8')\ntest = pd.read_csv
('test.txt', header=None, sep =';', names=['Lines','Emotions'], encoding='utf-8')\nvalidation = pd.read_csv('val.txt', hea
der=None, sep=';', names=['Lines','Emotions'], encoding='utf-8')'
```

K-FOLD CROSS VALIDATION

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold

# Define the emotions-to-labels mapping
emotions_to_labels = {'anger': 0, 'love': 1, 'fear': 2, 'joy': 3, 'sadness': 4,'surprise': 5}

# Read the data from the single CSV file
data = pd.read_csv('data.txt', header=None, sep=';', names=['Lines','Emotions'], encoding='utf-8')

# Shuffle the data randomly
data = data.sample(frac=1, random_state=42).reset_index(drop=True)

# Define the number of folds (e.g., 5-fold cross-validation)
num_folds = 5
skf = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=42)

# Initialize empty DataFrames for train, test, and validation
train_data = pd.DataFrame(columns=['Emotions', 'Lines', 'Labels'])
test_data = pd.DataFrame(columns=['Emotions', 'Lines', 'Labels'])
validation_data = pd.DataFrame(columns=['Emotions', 'Lines', 'Labels'])

# Iterate through the folds
for train_index, test_index in skf.split(data['Lines'], data['Emotions']):
    fold_train_data = data.iloc[train_index]
    fold_test_data = data.iloc[test_index]

    # Split the fold_train_data into train and validation sets (e.g., 80-20 split)
    fold_train_size = int(len(fold_train_data) * 0.8)
    fold_validation_data = fold_train_data.iloc[fold_train_size:]
    fold_train_data = fold_train_data.iloc[:fold_train_size]

    # Map emotions to labels for each fold
    fold_train_data['Labels'] = fold_train_data['Emotions'].replace(emotions_to_labels)
    fold_test_data['Labels'] = fold_test_data['Emotions'].replace(emotions_to_labels)
    fold_validation_data['Labels'] = fold_validation_data['Emotions'].replace(emotions_to_labels)

    # Concatenate fold data to the respective DataFrames
    train_data = pd.concat([train_data, fold_train_data], ignore_index=True)
    test_data = pd.concat([test_data, fold_test_data], ignore_index=True)
    validation_data = pd.concat([validation_data, fold_validation_data], ignore_index=True)

# Now, you have train_data, test_data, and validation_data as pandas DataFrames'''
```

```
<ipython-input-3-34b9baee02cb>:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  fold_test_data['Labels'] = fold_test_data['Emotions'].replace(emotions_to_labels)
<ipython-input-3-34b9baee02cb>:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  fold_test_data['Labels'] = fold_test_data['Emotions'].replace(emotions_to_labels)
<ipython-input-3-34b9baee02cb>:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  fold_test_data['Labels'] = fold_test_data['Emotions'].replace(emotions_to_labels)
```

```
<ipython-input-3-34b9baee02cb>:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  fold_test_data['Labels'] = fold_test_data['Emotions'].replace(emotions_to_labels)
<ipython-input-3-34b9baee02cb>:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  fold_test_data['Labels'] = fold_test_data['Emotions'].replace(emotions_to_labels)
```

```
data.head(10)
```

	Lines	Emotions
0	i feel assured that foods that are grown organ...	joy
1	i already have my christmas trees up i got two...	joy
2	i feel all betrayed and disillusioned	sadness
3	i will tell you that i am feeling quite invigo...	joy
4	i start to feel less exhausted the bits and pi...	sadness
5	i was listening to belle and sebastian feeling...	fear
6	i be able to look them in the face again witho...	sadness
7	i am thankful for feeling useful	joy
8	i woke up feeling artistic ish	joy
9	i was taunted by the ability of feeling threat...	fear

```
# After concatenating the data, rename the DataFrames
train = train_data
test = test_data
validation = validation_data

# Now, you have train, test, and validation as pandas DataFrames
```

```
# adding a column with encoded emotions
emotions_to_labels = {'anger': 0, 'love': 1, 'fear': 2, 'joy': 3, 'sadness': 4,'surprise': 5}
labels_to_emotions = {j:i for i,j in emotions_to_labels.items()}

train['Labels'] = train['Emotions'].replace(emotions_to_labels)
test['Labels'] = test['Emotions'].replace(emotions_to_labels)
validation['Labels'] = validation['Emotions'].replace(emotions_to_labels)
```

```
# adding a column with encoded emotions
labels_to_emotions = {j:i for i,j in emotions_to_labels.items()}
emotions_to_labels = {'anger': 0, 'love': 1, 'fear': 2, 'joy': 3, 'sadness': 4,'surprise': 5}
train['Labels'] = train['Emotions'].replace(emotions_to_labels)
test['Labels'] = test['Emotions'].replace(emotions_to_labels)
validation['Labels'] = validation['Emotions'].replace(emotions_to_labels)
'''
emotions_to_labels = {'anger': 0, 'love': 1, 'fear': 2, 'joy': 3, 'sadness': 4,'surprise': 5}
labels_to_emotions = {j:i for i,j in emotions_to_labels.items()}

train['Labels'] = train['Emotions'].replace(emotions_to_labels)
test['Labels'] = test['Emotions'].replace(emotions_to_labels)
validation['Labels'] = validation['Emotions'].replace(emotions_to_labels)'''
```

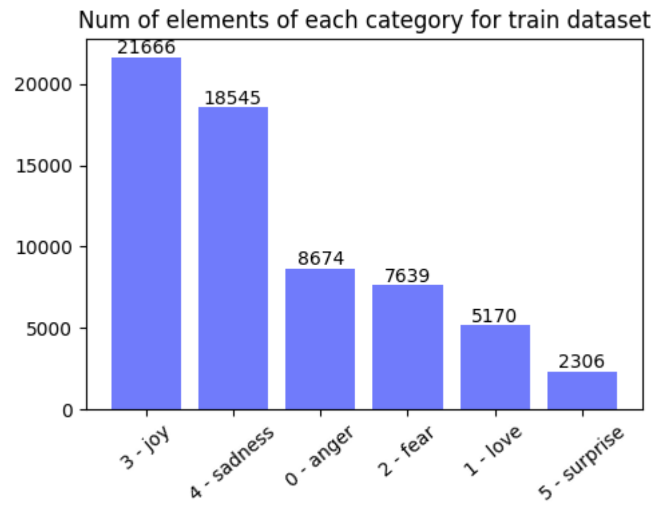
```
'\nemotions_to_labels = {'anger': 0, 'love': 1, 'fear': 2, 'joy': 3, 'sadness': 4,'surprise': 5}\nlabels_to_emotions = {j:
i for i,j in emotions_to_labels.items()}\n\ntrain['Labels'] = train['Emotions'].replace(emotions_to_labels)\ntest['Label
s'] = test['Emotions'].replace(emotions_to_labels)\nvalidation['Labels'] = validation['Emotions'].replace(emotions_to_labe
ls)'
```

```
train.head()
```

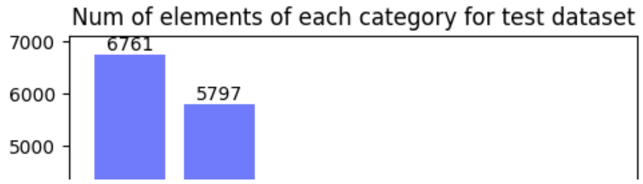
Emotions **Lines** **Labels**

```
def visualize_labels_distribution(df, title='the'):
  '''
  Accepts a dataframe with 'Emotions' column and dataset title (e.g. 'train')
  Creates bar chart with num of elements of each category
  Returns nothing

  '''
  # create a pandas series with labels and their counts
  num_labels = df['Emotions'].value_counts()

  # num of unique categories
  x_barchart = range(df['Emotions'].nunique())
  # list of labels
  x_barchart_labels = [str(emotions_to_labels[emotion]) +\
                       ' - ' + emotion for emotion in list(num_labels.index)]

  # list of counts
  y_barchart = list(num_labels.values)

  # creating bar chart
  plt.figure(figsize = (5, 4))
  plt.bar(x_barchart, y_barchart, color='#707bfb')

  # adding num of elements for each category on plot as text
  for index, data in enumerate(y_barchart):
    plt.text(x = index,
            y = data+max(y_barchart)/100,
            s = '{}'.format(data),
            fontdict = dict(fontsize=10),
            ha = 'center',)

  plt.xticks(x_barchart, x_barchart_labels, rotation=40)
  plt.title('Num of elements of each category for {} dataset'.format(title))
  plt.tight_layout()

  print('There are {} records in the dataset.\n'.format(len(df.index)))

  plt.show()


visualize_labels_distribution(train, 'train')
visualize_labels_distribution(test, 'test')
visualize_labels_distribution(validation, 'val')
```

```
There are 64000 records in the dataset.
```



```
There are 20000 records in the dataset.
```



```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords

# downloading a set of stop-words
STOPWORDS = set(stopwords.words('english'))

# tokenizer
from nltk.tokenize import word_tokenize
```

```
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]   Unzipping corpora/stopwords.zip.
```

Num of elements of each category for val dataset

```
def text_preprocess(text, stop_words=False):
  '''
  Accepts text (a single string) and
  a parameters of preprocessing
  Returns preprocessed text

  '''
  # clean text from non-words
  text = re.sub(r'\W+', ' ', text).lower()

  # tokenize the text
  tokens = word_tokenize(text)
  if stop_words:
    # delete stop_words
    tokens = [token for token in tokens if token not in STOPWORDS]

  return tokens
```

4 5

```
print('Before: ')
print(train.head())

x_train = [text_preprocess(t, stop_words=True) for t in train['Lines']]
y_train = train['Labels'].values

print('\nAfter:')
for line_and_label in list(zip(x_train[:5], y_train[:5])):
  print(line_and_label)
```

```
    Before:
      Emotions                                              Lines  Labels
    0      joy  i feel assured that foods that are grown organ...       3
    1      joy  i already have my christmas trees up i got two...       3
    2  sadness                 i feel all betrayed and disillusioned       4
```

```
     3      joy  i will tell you that i am feeling quite invigo...       3
     4     fear  i was listening to belle and sebastian feeling...       2

     After:
     (['feel', 'assured', 'foods', 'grown', 'organic', 'free', 'pesticides', 'soil', 'water', 'contaminated', 'good', 'us'], 3)
     (['already', 'christmas', 'trees', 'got', 'two', 'feeling', 'festive', 'sure', 'spurring', 'get', 'started', 'book'], 3)
     (['feel', 'betrayed', 'disillusioned'], 4)
     (['tell', 'feeling', 'quite', 'invigorated'], 3)
     (['listening', 'belle', 'sebastian', 'feeling', 'agitated'], 2)
```

```
x_test = [text_preprocess(t, stop_words=True) for t in test['Lines']]
y_test = test['Labels'].values

x_validation = [text_preprocess(t, stop_words=True) for t in validation['Lines']]
y_validation = validation['Labels'].values
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
from gensim.models import Word2Vec
model_w2v = Word2Vec(x_train + x_test + x_validation,vector_size=300,min_count = 2).wv
```

```
def create_weight_matrix(model):
  '''
  Accepts word embedding model
  and the second model, if provided
  Returns weight matrix of size m*n, where
  m - size of the dictionary
  n - size of the word embedding vector

  '''
  vector_size = model.get_vector('like').shape[0]
  w_matrix = np.zeros((DICT_SIZE, vector_size))
  skipped_words = []

  for word, index in tokenizer.word_index.items():
    if index < DICT_SIZE:
      if word in model.key_to_index:
        w_matrix[index] = model.get_vector(word)
      else:
        skipped_words.append(word)

  print(f'{len(skipped_words)} words were skipped. Some of them:')
  print(skipped_words[:50])
  return w_matrix
```

```
DICT_SIZE = 15000
tokenizer = Tokenizer(num_words=DICT_SIZE)
total = x_train + x_test + x_validation
tokenizer.fit_on_texts(total)

x_train_max_len = max([len(i) for i in x_train])
x_test_max_len = max([len(i) for i in x_test])
x_validation_max_len = max([len(i) for i in x_validation])

MAX_LEN = max(x_train_max_len, x_test_max_len, x_validation_max_len)

X_train = tokenizer.texts_to_sequences(x_train)
X_train_pad = pad_sequences(X_train, maxlen=MAX_LEN)

X_test = tokenizer.texts_to_sequences(x_test)
X_test_pad = pad_sequences(X_test, maxlen=MAX_LEN)

X_val = tokenizer.texts_to_sequences(x_validation)
X_val_pad = pad_sequences(X_val, maxlen=MAX_LEN)
```

```
DICT_SIZE = 15000
weight_matrix = create_weight_matrix(model_w2v)
print(weight_matrix.shape)
print(weight_matrix)

     0 words were skipped. Some of them:
     []
     (15000, 300)
     [[ 0.          0.          0.         ...  0.          0.
        0.        ]
      [-0.24399863  0.06068315  0.74730527 ... -0.09848853  0.29988492
        0.05449728]
```

```
 [-0.2920664   0.42695162 -0.22316186 ... -1.24302661  0.44786328
  -0.22837731]
 ...
 [-0.01635791  0.06706786 -0.011201   ... -0.00142274  0.04699629
  -0.03415344]
 [-0.01758169  0.05304477 -0.00396949 ... -0.00457234  0.05191622
  -0.03427338]
 [-0.01686442  0.04439766 -0.00197587 ... -0.00560607  0.05632621
  -0.02702906]]
```

```
# import models, layers, optimizers from tensorflow
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional, Dense, Dropout, GRU, Lambda, Input, Attention, Flatten
from tensorflow.keras.optimizers import Adam
```

BILSTM

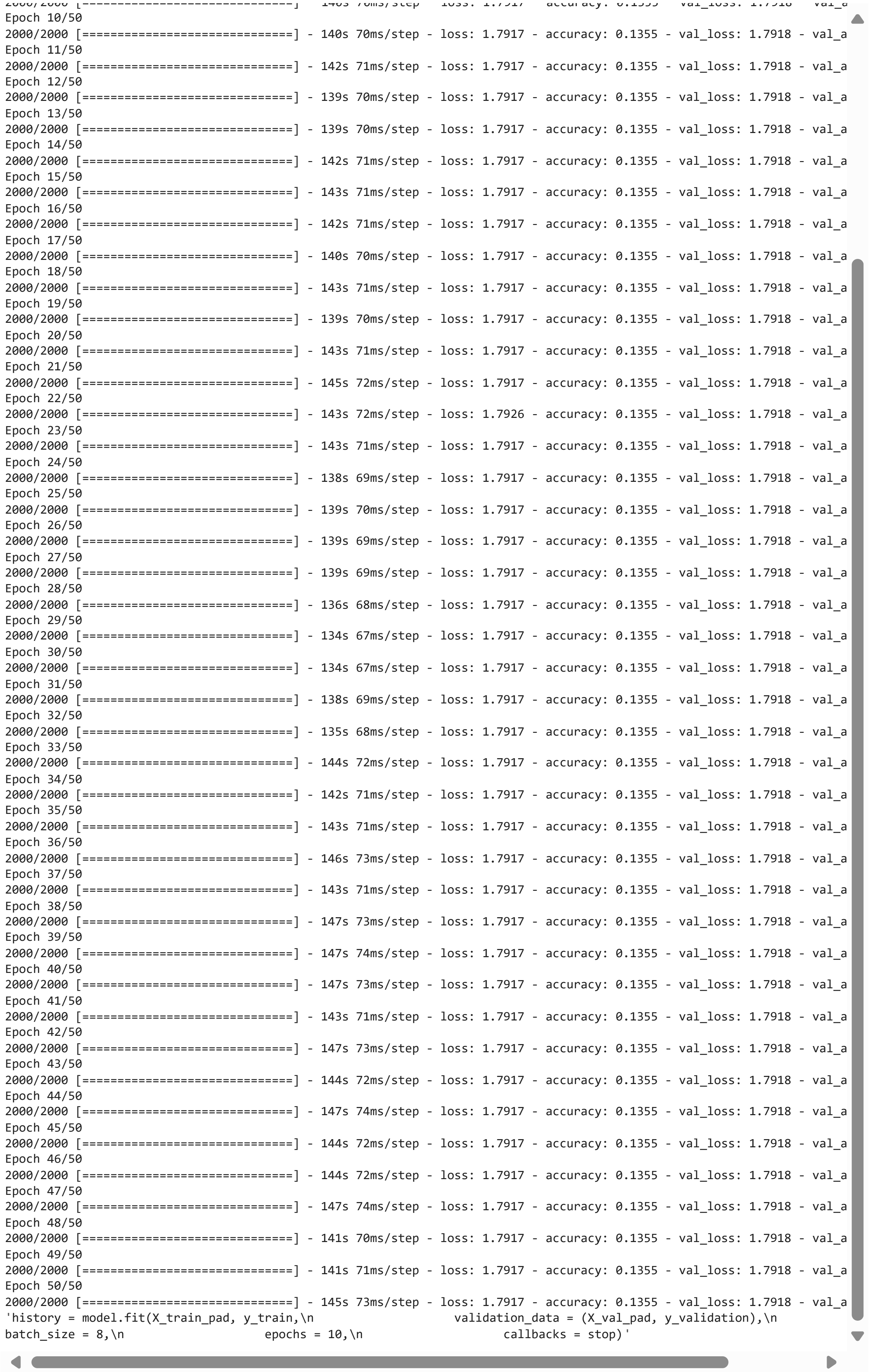
```
from keras.models import Sequential
from keras.layers import Conv1D, BatchNormalization, Embedding, Dropout

# Assuming you have defined DICT_SIZE, weight_matrix, X_train_pad

input_shape = (X_train_pad.shape[1],)  # Input shape for 1D convolution
vocab_size = 15000
embedding_dim = 300
sequence_length = MAX_LEN
units = 64
output_dim = 6
model = Sequential()

model.add(Embedding(input_dim=DICT_SIZE,
                    output_dim=weight_matrix.shape[1],
                    input_length=X_train_pad.shape[1],
                    weights=[weight_matrix],
                    trainable=False))

model.add(Conv1D(32, kernel_size=3, activation='relu', input_shape=input_shape))
model.add(BatchNormalization())
model.add(Conv1D(32, kernel_size=3, activation='relu'))
model.add(BatchNormalization())

model.add(Conv1D(32, kernel_size=5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv1D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv1D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv1D(64, kernel_size=5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv1D(128, kernel_size=4, activation='relu'))
model.add(BatchNormalization())
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(256, return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(128, return_sequences=False)))
model.add(Dense(6, activation = 'relu'))
model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(learning_rate = 0.001), metrics=['accuracy'])
model.summary()
```

```
 batch_normalization (Batch  (None, 33, 32)            128
 Normalization)

 conv1d_1 (Conv1D)           (None, 31, 32)            3104

 batch_normalization_1 (Bat  (None, 31, 32)            128
 chNormalization)

 conv1d_2 (Conv1D)           (None, 16, 32)            5152
 batch_normalization_3 (Bat  (None, 14, 64)            256
 chNormalization)

 conv1d_4 (Conv1D)           (None, 12, 64)            12352

 batch_normalization_4 (Bat  (None, 12, 64)            256
 chNormalization)

 conv1d_5 (Conv1D)           (None, 6, 64)             20544

 batch_normalization_5 (Bat  (None, 6, 64)             256
 chNormalization)

 dropout_1 (Dropout)         (None, 6, 64)             0

 conv1d_6 (Conv1D)           (None, 3, 128)            32896

 batch_normalization_6 (Bat  (None, 3, 128)            512
 chNormalization)

 bidirectional (Bidirection  (None, 3, 256)            263168
 al)

 dropout_2 (Dropout)         (None, 3, 256)            0

 bidirectional_1 (Bidirecti  (None, 3, 512)            1050624
 onal)

 dropout_3 (Dropout)         (None, 3, 512)            0

 bidirectional_2 (Bidirecti  (None, 256)               656384
 onal)

 dense (Dense)               (None, 6)                 1542

=================================================================
Total params: 6582470 (25.11 MB)
Trainable params: 2081638 (7.94 MB)
Non-trainable params: 4500832 (17.17 MB)
_________________________________________________________________
```

```
'''vocab_size = 15000
embedding_dim = 300
sequence_length = MAX_LEN
units = 64
output_dim = 6
model = Sequential()
model.add(Input(shape=(MAX_LEN,)))
model.add(Embedding(weight_matrix.shape[0], weight_matrix.shape[1], input_length=MAX_LEN, weights = [weight_matrix]))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(256, return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(128, return_sequences=False)))
model.add(Dense(6, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(learning_rate = 0.001), metrics='accuracy')
model.summary()'''
```

```
'vocab_size = 15000\nembedding_dim = 300\nsequence_length = MAX_LEN\nunits = 64\noutput_dim = 6\nmodel = Sequential()\nmod
el.add(Input(shape=(MAX_LEN,)))\nmodel.add(Embedding(weight_matrix.shape[0], weight_matrix.shape[1], input_length=MAX_LEN,
weights = [weight_matrix]))\nmodel.add(Bidirectional(LSTM(128, return_sequences=True)))\nmodel.add(Dropout(0.2))\nmodel.ad
d(Bidirectional(LSTM(256, return_sequences=True)))\nmodel.add(Dropout(0.2))\nmodel.add(Bidirectional(LSTM(128, return_sequ
ences=False)))\nmodel.add(Dense(6, activation='softmax'))\nmodel.compile(loss='sparse_categorical_crossentropy', optimizer
=Adam(learning_rate = 0.001), metrics='accuracy')\nmodel.summary()'
```

```
history=model.fit(X_train_pad, y_train,
                    validation_data = (X_val_pad, y_validation),
                    batch_size = 32,
                    epochs = 50)
'''history = model.fit(X_train_pad, y_train,
                    validation_data = (X_val_pad, y_validation),
                    batch_size = 8,
                    epochs = 10,
                    callbacks = stop)'''
```



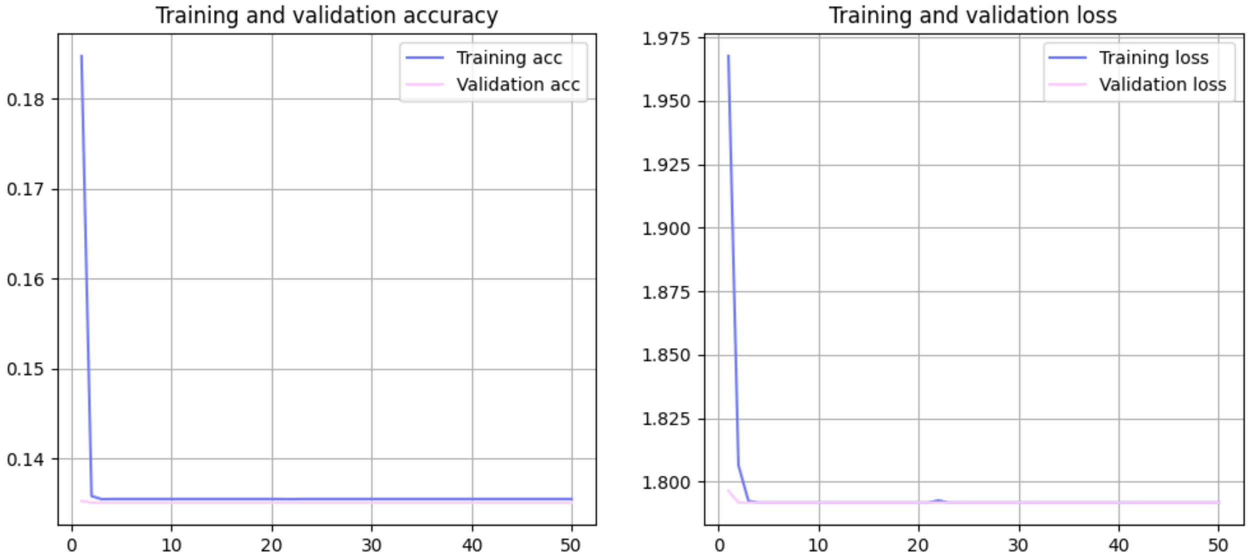
```
2000/2000 [==============================] - 140s 70ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 10/50
2000/2000 [==============================] - 140s 70ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 11/50
2000/2000 [==============================] - 142s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 12/50
2000/2000 [==============================] - 139s 70ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 13/50
2000/2000 [==============================] - 139s 70ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 14/50
2000/2000 [==============================] - 142s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 15/50
2000/2000 [==============================] - 143s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 16/50
2000/2000 [==============================] - 142s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 17/50
2000/2000 [==============================] - 140s 70ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 18/50
2000/2000 [==============================] - 143s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 19/50
2000/2000 [==============================] - 139s 70ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 20/50
2000/2000 [==============================] - 143s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 21/50
2000/2000 [==============================] - 145s 72ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 22/50
2000/2000 [==============================] - 143s 72ms/step - loss: 1.7926 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 23/50
2000/2000 [==============================] - 143s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 24/50
2000/2000 [==============================] - 138s 69ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 25/50
2000/2000 [==============================] - 139s 70ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 26/50
2000/2000 [==============================] - 139s 69ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 27/50
2000/2000 [==============================] - 139s 69ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 28/50
2000/2000 [==============================] - 136s 68ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 29/50
2000/2000 [==============================] - 134s 67ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 30/50
2000/2000 [==============================] - 134s 67ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 31/50
2000/2000 [==============================] - 138s 69ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 32/50
2000/2000 [==============================] - 135s 68ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 33/50
2000/2000 [==============================] - 144s 72ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 34/50
2000/2000 [==============================] - 142s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 35/50
2000/2000 [==============================] - 143s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 36/50
2000/2000 [==============================] - 146s 73ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 37/50
2000/2000 [==============================] - 143s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 38/50
2000/2000 [==============================] - 147s 73ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 39/50
2000/2000 [==============================] - 147s 74ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 40/50
2000/2000 [==============================] - 147s 73ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 41/50
2000/2000 [==============================] - 143s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 42/50
2000/2000 [==============================] - 147s 73ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 43/50
2000/2000 [==============================] - 144s 72ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 44/50
2000/2000 [==============================] - 147s 74ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 45/50
2000/2000 [==============================] - 144s 72ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 46/50
2000/2000 [==============================] - 144s 72ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 47/50
2000/2000 [==============================] - 147s 74ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 48/50
2000/2000 [==============================] - 141s 70ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 49/50
2000/2000 [==============================] - 141s 71ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
Epoch 50/50
2000/2000 [==============================] - 145s 73ms/step - loss: 1.7917 - accuracy: 0.1355 - val_loss: 1.7918 - val_a
'history = model.fit(X_train_pad, y_train,\n                    validation_data = (X_val_pad, y_validation),\n
batch_size = 8,\n                    epochs = 10,\n                    callbacks = stop)'
```

```
model.evaluate(X_test_pad, y_test)
```

```
625/625 [==============================] - 9s 15ms/step - loss: 1.7918 - accuracy: 0.1355
[1.7917513847351074, 0.13545000553131104]
```

```
def plot_history(history):
    '''
    Plots training and validation accuracy and loss
    Accepts a single param - history, where
    history - keras.callbacks.History object
    Returns nothing

    '''
    loss = history.history['loss']
    accuracy = history.history['accuracy']
    val_loss = history.history['val_loss']
    val_accuracy = history.history['val_accuracy']
    x = range(1, len(loss) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, accuracy, label='Training acc', color='#707bfb')
    plt.plot(x, val_accuracy, label='Validation acc', color='#fbcbff')
    plt.title('Training and validation accuracy')
    plt.grid(True)
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(x, loss, label='Training loss', color='#707bfb')
    plt.plot(x, val_loss, label='Validation loss', color='#fbcbff')
    plt.title('Training and validation loss')
    plt.grid(True)
    plt.legend()
```

```
plot_history(history)
```

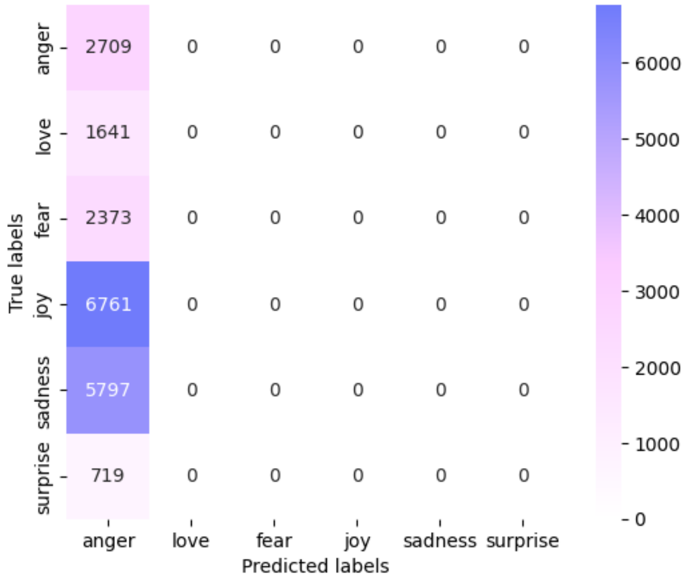


```
model.evaluate(X_test_pad, y_test)
y_pred = np.argmax(model.predict(X_test_pad), axis=1)
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred))
```

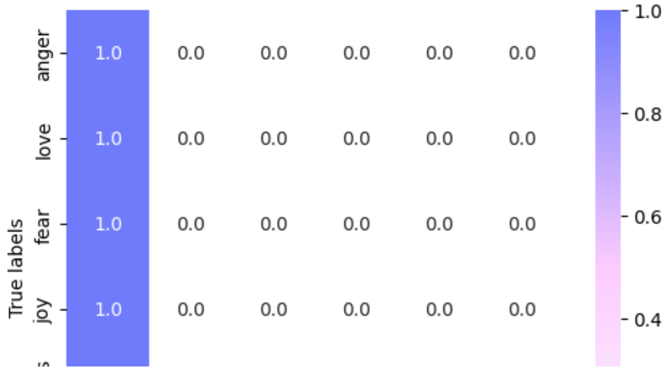
```
625/625 [==============================] - 9s 15ms/step - loss: 1.7918 - accuracy: 0.1355
625/625 [==============================] - 11s 15ms/step
              precision    recall  f1-score   support

           0       0.14      1.00      0.24      2709
           1       0.00      0.00      0.00      1641
           2       0.00      0.00      0.00      2373
           3       0.00      0.00      0.00      6761
           4       0.00      0.00      0.00      5797
           5       0.00      0.00      0.00       719

    accuracy                           0.14     20000
   macro avg       0.02      0.17      0.04     20000
```

```
    weighted avg       0.02      0.14      0.03     20000

    /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
      _warn_prf(average, modifier, msg_start, len(result))
```

```
# setting a custom colormap
from matplotlib.colors import LinearSegmentedColormap
colors = ['#ffffff', '#fbcbff', '#707bfb']
cmap = LinearSegmentedColormap.from_list('mycmap', colors)


def plot_confusion_matrix(matrix, fmt=''):
  '''
  Accepts a confusion matrix and a format param
  Plots the matrix as a heatmap
  Returns nothing

  '''
  plt.figure(figsize=(6, 5))
  sns.heatmap(matrix, annot=True,
              cmap=cmap,
              fmt=fmt,
              xticklabels=emotions_to_labels.keys(),
              yticklabels=emotions_to_labels.keys())
  plt.ylabel('True labels')
  plt.xlabel('Predicted labels')
  plt.show()


matrix = metrics.confusion_matrix(y_test, y_pred)
plot_confusion_matrix(matrix)
```



```
# create new confusion matrix
# where values are normed by row
matrix_new = np.zeros(matrix.shape)

for row in range(len(matrix)):
  sum = np.sum(matrix[row])
  for element in range(len(matrix[row])):
    matrix_new[row][element] = matrix[row][element] / sum

plot_confusion_matrix(matrix_new, fmt='.2')
```



```
def predict(texts):
  '''
  Accepts array if texts (strings)
  Prints sentence and the corresponding label (emotion)
  Returns nothing

  '''
  texts_prepr = [text_preprocess(t) for t in texts]
  sequences = tokenizer.texts_to_sequences(texts_prepr)
  pad = pad_sequences(sequences, maxlen=MAX_LEN)

  predictions = model.predict(pad)
  labels = np.argmax(predictions, axis=1)

  for i, lbl in enumerate(labels):
    print(f'\'{texts[i]}\' --> {labels_to_emotions[lbl]}')


test_texts = ['I am so happy in the way you behaved today', 'The man felt lonely', 'The guests felt satisfied']

predict(test_texts)
```

```
1/1 [==============================] - 3s 3s/step
'I am so happy in the way you behaved today' --> anger
'The man felt lonely' --> anger
'The guests felt satisfied' --> anger
```

✓ 0s completed at 3:00 PM