Importing the libraries needed

In [1]:

```
import numpy as np
import pandas as pd
import time


import matplotlib.pyplot as plt
import seaborn as sns

import re
import string

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
ore

import gensim
from gensim.models import KeyedVectors

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

import tensorflow as tf
from keras.models import Sequential
from tensorflow.keras.layers import SpatialDropout1D, Conv1D, Bidirectional, LST
M, Dense, Input, Dropout, GlobalMaxPooling1D
from keras.layers.embeddings import Embedding
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, Early
Stopping
from tensorflow.keras.optimizers import Adam

import itertools
from numpy import loadtxt
from keras.models import load_model

import warnings
warnings.filterwarnings("ignore")
```

Connecting to google drive

In [2]:

```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Mounted at /content/gdrive
```

Uploading the dataset

In [3]:

```
path_data = "/content/gdrive/MyDrive/thesis/modified.csv"

Arsas = pd.read_csv(path_data ,sep='\t')
```

In [4]:

```
data = Arsas
```

printing the first 3 rows of the data

In [5]:

```
data.head(3)
```

Out[5]:

	#Tweet_ID	Tweet_text	Sentiment_label
0	929241870508724224	... مصر الجولة الأخيرة# x المباراة القـادمة #غانا	Positive
1	928942264583376897	...هل هذه هي سياسة خارجيه لدوله تحترم نفسها والآخ	Negative
2	928615163250520065	...وزير خارجية فرنسا عن منتدى شباب العالم: شعرت ب	Positive

printing the shape of the dataset nbr of row and columns

In [6]:

```
print("Data contient {} lignes et {} colonnes.".format(data.shape[0], data.shape
[1]))
```

```
Data contient 21064 lignes et 3 colonnes.
```

printing the fiels with missed values

In [7]:

```
data.isnull().sum()
```

Out[7]:

```
#Tweet_ID          0
Tweet_text         0
Sentiment_label    0
dtype: int64
```

printing the number of the duplicated rows

In [8]:

```
print("On a  {} doublons dans Data.".format(data.duplicated().sum()))
```

```
On a  68 doublons dans Data.
```

In [9]:

```
data.drop_duplicates(inplace = True)
```

In [10]:

```
print("On a  {} doublons dans Data.".format(data.duplicated().sum()))
```

```
On a  0 doublons dans Data.
```

checking the types of the fiels in the data

In [11]:

```
data.dtypes
```

Out[11]:

```
#Tweet_ID            int64
Tweet_text          object
Sentiment_label     object
dtype: object
```

function for printing the pie

In [12]:

```
def pie(data,col):
    labels = data[col].value_counts().keys().tolist()
    n = len(labels)
    if n==2:
        colors = ['#66b3ff', '#fb3999']
    elif n==3:
        colors = ['#66b3ff', '#fb3999', '#ffcc99']
    elif n==4:
        colors = ['#66b3ff', '#fb3999', '#ffcc99',"#66f3ff"]
    elif n==5:
        colors = ['#66b3ff', '#fb3999', '#ffcc99',"#66f3ff",'#adcc99']
    elif n==6:
        colors = ['#66b3ff', '#fb3999', '#ffcc99',"#66f3ff",'#adcc99',"#db7f23"]

    fig1, f1 = plt.subplots()
    f1.pie(data[col].value_counts(), labels=labels, colors = colors, autopct='%
1.1f%%',shadow=False, startangle=60)
    f1.axis('equal')
    plt.tight_layout()
    plt.show()

def histo(data,col):
    plt.figure(figsize = (10, 8))
    sns.histplot(data=data, x=col, hue = data[col], fill=True)
```

Counting the % of each classe

In [13]:

```
data.Sentiment_label.value_counts(normalize = True)
```

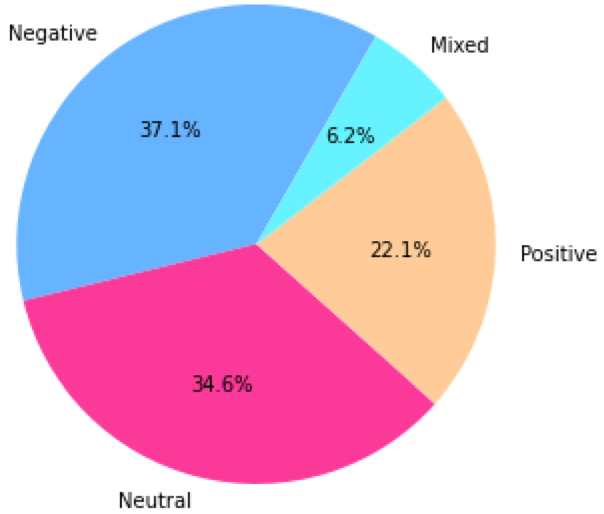
Out[13]:

```
Negative    0.371404
Neutral     0.346018
Positive    0.220566
Mixed       0.062012
Name: Sentiment_label, dtype: float64
```

Printing the distribution of the classes

In [14]:

```
pie(data, "Sentiment_label")
```



In [15]:

```
positive = data[data["Sentiment_label"] == "Positive"]
positive["sentiment"] = 1

mixed = data[data["Sentiment_label"] == "Mixed"]
mixed["sentiment"] = 2

neutral = data[data["Sentiment_label"] == "Neutral"]
neutral["sentiment"] = 3

negative = data[data["Sentiment_label"] == "Negative"]
negative["sentiment"] = 0

data = pd.concat([positive, mixed, neutral, negative], ignore_index = True)
```

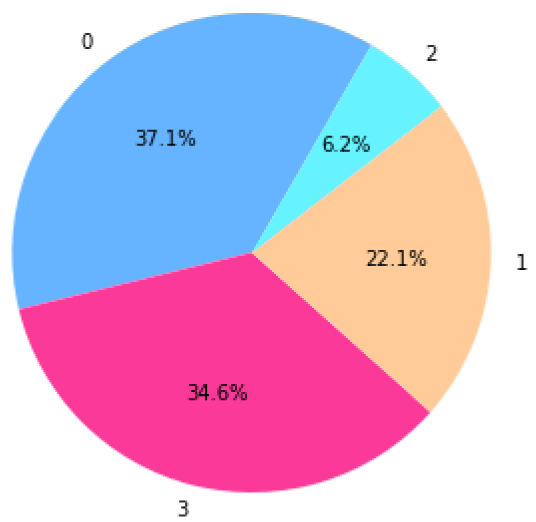
In [16]:

```
print("data contient {} lignes.".format(data.shape[0]))

print("Positive contient {} lignes.".format(positive.shape[0]))

print("Negative contient {} lignes.".format(negative.shape[0]))

print("Mixed contient {} lignes.".format(mixed.shape[0]))

print("Neutral contient {} lignes.".format(neutral.shape[0]))
```

```
data contient 20996 lignes.
Positive contient 4631 lignes.
Negative contient 7798 lignes.
Mixed contient 1302 lignes.
Neutral contient 7265 lignes.
```

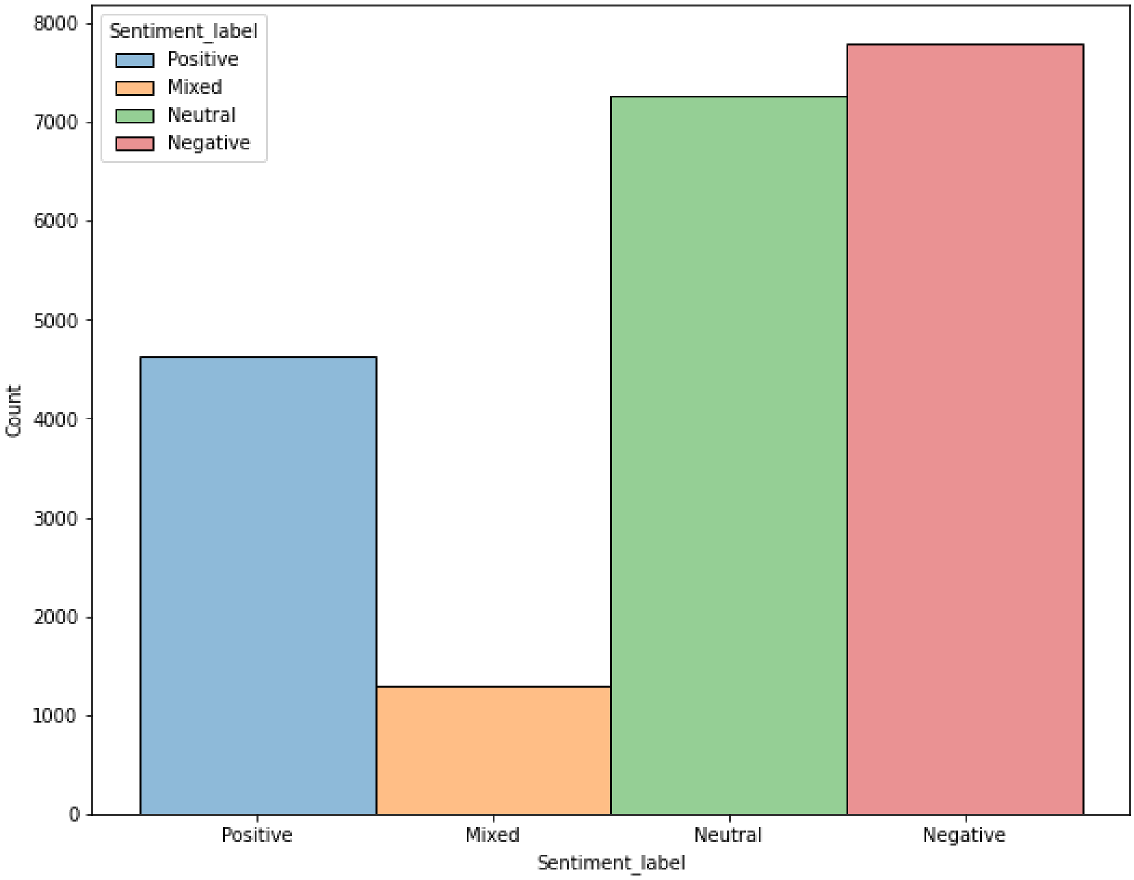
In [17]:

```
pie(data,"sentiment")
```



In [18]:

```
histo(data,"Sentiment_label")
```



function to count the length of reviews

In [19]:

```
def compte_mots(phrase):
    return len(phrase.split())

data["len_review"] = data["Tweet_text"].apply(compte_mots)
```

printing the max length of the positive and negative reviews

In [20]:

```
print("Le maximum de mots utilisé dans les reviews  est :", max(data['len_revie
w']))
print("Le moyen de mots utilisé dans les reviews est :", np.mean(data['len_revie
w']))
```

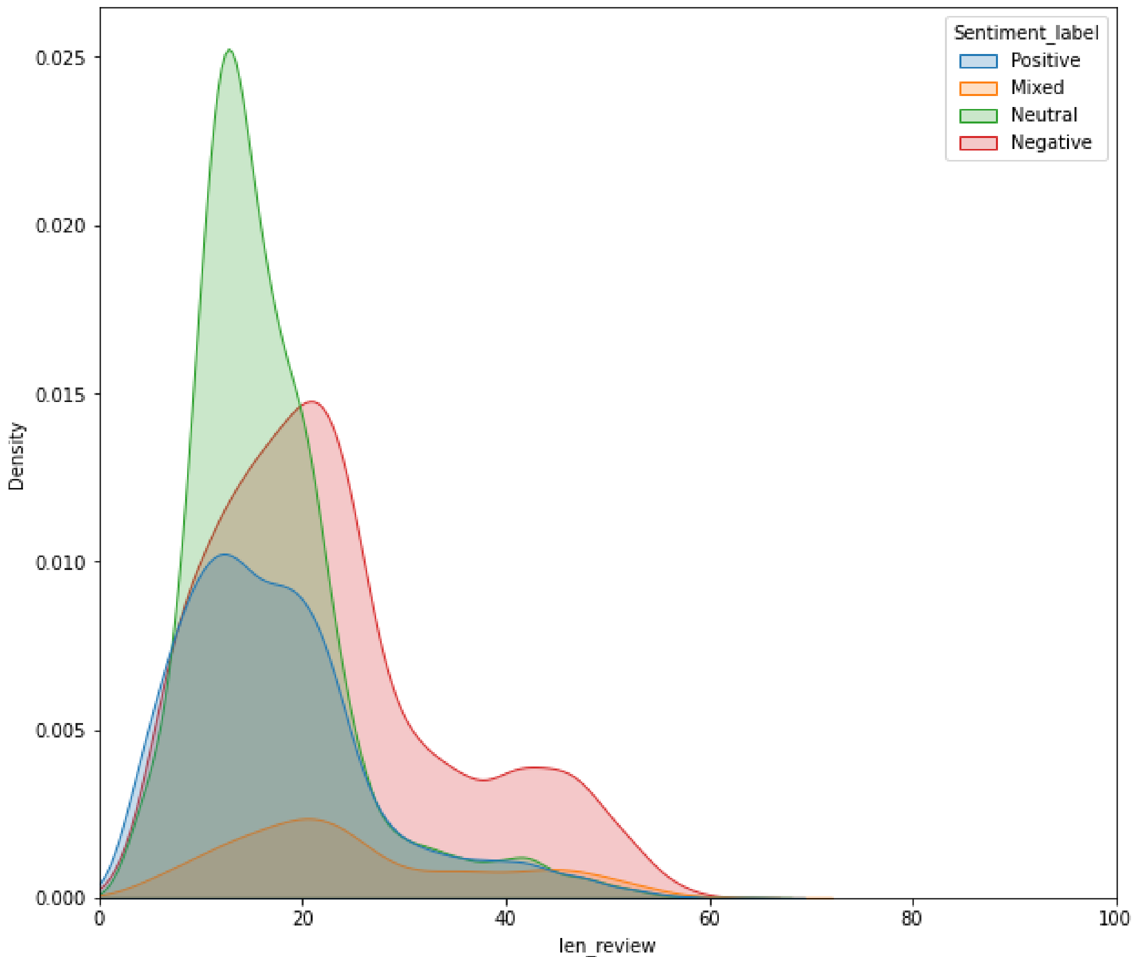
```
Le maximum de mots utilisé dans les reviews  est : 64
Le moyen de mots utilisé dans les reviews est : 19.701657458563535
```

In [21]:

```
plt.figure(figsize=(10,9))

p1=sns.kdeplot(data['len_review'], hue = data['Sentiment_label'],  shade=True, c
olor="r")

plt.xlim(0, 100)
```

Out[21]:

```
(0.0, 100.0)
```



Deleting unused fields

In [22]:

```
data.drop(['#Tweet_ID'], axis = 1, inplace = True)
data.head(3)
```

Out[22]:

	Tweet_text	Sentiment_label	sentiment	len_review
0	... مصر الجولة الأخيرة# x المباراة القـادمة #غانا	Positive	1	45
1	...وزير خارجية فرنسا عن منتدى شباب العالم: شعرت ب	Positive	1	16
2	بسم الله نبدأ 👏 نغرد علي وسم 👇 👇 👇 👇 👇 ↩ #شباب...	Positive	1	27

In [23]:

```
df = data
df.dtypes
```

Out[23]:

```
Tweet_text          object
Sentiment_label     object
sentiment            int64
len_review           int64
dtype: object
```

the function of the preprocessing

In [24]:

```
def preprocessing(text):

    # ref: https://github.com/bakrianoo/aravec
    search = ["ى","'","ـ",'"'," يا "," و ","،",".","/","-","_","ة","آ","إ","أ",
              "\\",'\n', '\t','&quot;','?','؟','!']
    replace = ["يا ","و ","","",""," "," ","ه","ا","ا","ا",
               "","","","؟ ",' ? ',' ',' ' ,' ',"","ي ', ' ! ']

    tashkeel = re.compile(r'[\u0617-\u061A\u064B-\u0652]')
    text = re.sub(tashkeel,"", text)

    longation = re.compile(r'(.)\1+')
    subst = r"\1\1"
    text = re.sub(longation, subst, text)

    text = re.sub(r"[^\w\s]", '', text)
    text = re.sub(r"[a-zA-Z]", '', text)
    text = re.sub(r"\d+", ' ', text)
    text = re.sub(r"\n+", ' ', text)
    text = re.sub(r"\t+", ' ', text)
    text = re.sub(r"\r+", ' ', text)
    text = re.sub(r"\s+", ' ', text)
    text = text.replace('و' ,'وو')
    text = text.replace('ي' ,'يي')
    text = text.replace('ا' ,'اا')

    for i in range(0, len(search)):
        text = text.replace(search[i], replace[i])

    text = text.strip()

    return text
```

preprocessing the reviews and printing the time spent

In [25]:

```
%%time
data["Clean_reviews"] = data.Tweet_text.apply(lambda x: preprocessing(x))
```

```
CPU times: user 1 s, sys: 0 ns, total: 1 s
Wall time: 999 ms
```

printing a review before and after preprocessing

In [26]:

```
print('- Avant le prétraitement \n\n',data["Tweet_text"][4])
print("\n-----------------------------------------------\n")
print('- Après le prétraitement \n\n',data["Clean_reviews"][4])
```

```
- Avant le prétraitement

 شارك بتصويت «BBC» بأفريقيا الأفضل جائزة على للحصول صلاح محمد لدعم htt
ps://t.co/t1Q0l0UlP

-----------------------------------------------

- Après le prétraitement

 شارك بتصويت لدعم محمد صلاح للحصول علي جائزه الافضل بافريقيا
```

Saving the cleaned data in a csv file

In [27]:

```
data.to_csv("cleaned_Arsas.csv")
```

asigning the reviews and classes to a new variables

In [28]:

```
X = data.Clean_reviews
y=pd.get_dummies(data.sentiment)
# y = data.sentiment
```

spliting the data to train and test set

In [29]:

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.20,
                                                    random_state = 42)
```

printing the number of the train set and the test set

In [30]:

```
print('Train set', X_train.shape)
print('Test set', X_test.shape)
```

```
Train set (16796,)
Test set (4200,)
```

In [31]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly rem
ount, call drive.mount("/content/gdrive", force_remount=True).
```

Uploading the fsttext pretrained word embedding with 150 dimension

In [32]:

```
%%time
target_word_vec = KeyedVectors.load_word2vec_format("/content/gdrive/MyDrive/the
sis/cc.ar.150.vec", binary = False)
```

```
CPU times: user 2min 24s, sys: 3.73 s, total: 2min 28s
Wall time: 2min 33s
```

tokenization of the reviews

In [33]:

```
%%time
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
```

```
CPU times: user 489 ms, sys: 8.04 ms, total: 497 ms
Wall time: 497 ms
```

In [34]:

```
word_index = tokenizer.word_index
vocab_size = len(tokenizer.word_index) + 1
```

making all reviews of the same length 70

In [35]:

```
%%time
MAX_SEQUENCE_LENGTH = 70

X_train = pad_sequences(tokenizer.texts_to_sequences(X_train),
                        maxlen = MAX_SEQUENCE_LENGTH)
X_test = pad_sequences(tokenizer.texts_to_sequences(X_test),
                       maxlen = MAX_SEQUENCE_LENGTH)

print("Training X Shape:", X_train.shape)
print("Testing X Shape:", X_test.shape)
```

```
Training X Shape: (16796, 70)
Testing X Shape: (4200, 70)
CPU times: user 914 ms, sys: 10 ms, total: 924 ms
Wall time: 920 ms
```

Construction of the embedding matrix

In [36]:

```
%%time
embedding_matrix = np.zeros((vocab_size, 150))

for word, i in word_index.items():
    if word in target_word_vec :
        embedding_vector = target_word_vec[word]
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

```
CPU times: user 150 ms, sys: 25.1 ms, total: 176 ms
Wall time: 178 ms
```

In [37]:

```
embedding_matrix.shape[0] == vocab_size
```

Out[37]:

```
True
```

Creating the model

In [38]:

```
model = Sequential()
embedding_layer = Embedding(vocab_size,
                            150,
                            weights = [embedding_matrix],
                            input_length = MAX_SEQUENCE_LENGTH,
                            trainable=False)
model.add(embedding_layer)
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax'))
model.compile(optimizer = Adam(learning_rate=0.001),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'])

# es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 70, 150)           7419000

 lstm (LSTM)                 (None, 100)               100400

 dropout (Dropout)           (None, 100)               0

 dense (Dense)               (None, 4)                 404

=================================================================
Total params: 7,519,804
Trainable params: 100,804
Non-trainable params: 7,419,000
_________________________________________________________________
None
```

fitting the model to the dataset

In [39]:

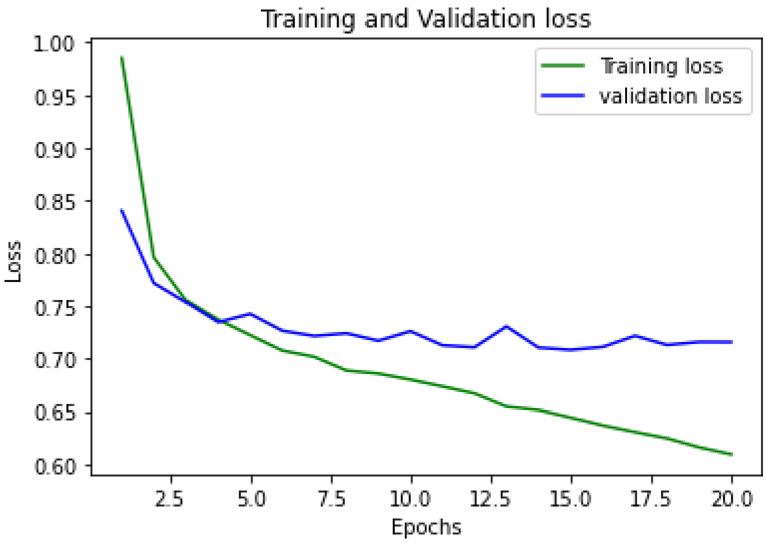
```
history = model.fit(X_train, y_train, validation_split=0.15, batch_size = 128, e
pochs=20, verbose=1)
```

```
Epoch 1/20
112/112 [==============================] - 8s 13ms/step - loss: 0.98
51 - accuracy: 0.6077 - val_loss: 0.8406 - val_accuracy: 0.6774
Epoch 2/20
112/112 [==============================] - 1s 8ms/step - loss: 0.796
2 - accuracy: 0.6991 - val_loss: 0.7718 - val_accuracy: 0.7020
Epoch 3/20
112/112 [==============================] - 1s 8ms/step - loss: 0.755
4 - accuracy: 0.7159 - val_loss: 0.7536 - val_accuracy: 0.7095
Epoch 4/20
112/112 [==============================] - 1s 10ms/step - loss: 0.73
71 - accuracy: 0.7232 - val_loss: 0.7347 - val_accuracy: 0.7139
Epoch 5/20
112/112 [==============================] - 1s 11ms/step - loss: 0.72
23 - accuracy: 0.7270 - val_loss: 0.7425 - val_accuracy: 0.7115
Epoch 6/20
112/112 [==============================] - 1s 11ms/step - loss: 0.70
77 - accuracy: 0.7357 - val_loss: 0.7265 - val_accuracy: 0.7147
Epoch 7/20
112/112 [==============================] - 1s 9ms/step - loss: 0.701
7 - accuracy: 0.7381 - val_loss: 0.7216 - val_accuracy: 0.7214
Epoch 8/20
112/112 [==============================] - 1s 7ms/step - loss: 0.688
8 - accuracy: 0.7408 - val_loss: 0.7241 - val_accuracy: 0.7159
Epoch 9/20
112/112 [==============================] - 1s 7ms/step - loss: 0.685
9 - accuracy: 0.7418 - val_loss: 0.7170 - val_accuracy: 0.7238
Epoch 10/20
112/112 [==============================] - 1s 7ms/step - loss: 0.680
2 - accuracy: 0.7455 - val_loss: 0.7261 - val_accuracy: 0.7107
Epoch 11/20
112/112 [==============================] - 1s 7ms/step - loss: 0.673
8 - accuracy: 0.7461 - val_loss: 0.7127 - val_accuracy: 0.7250
Epoch 12/20
112/112 [==============================] - 1s 7ms/step - loss: 0.667
1 - accuracy: 0.7485 - val_loss: 0.7108 - val_accuracy: 0.7175
Epoch 13/20
112/112 [==============================] - 1s 8ms/step - loss: 0.654
7 - accuracy: 0.7557 - val_loss: 0.7306 - val_accuracy: 0.7218
Epoch 14/20
112/112 [==============================] - 1s 7ms/step - loss: 0.651
4 - accuracy: 0.7550 - val_loss: 0.7105 - val_accuracy: 0.7226
Epoch 15/20
112/112 [==============================] - 1s 8ms/step - loss: 0.644
3 - accuracy: 0.7589 - val_loss: 0.7084 - val_accuracy: 0.7274
Epoch 16/20
112/112 [==============================] - 1s 8ms/step - loss: 0.636
8 - accuracy: 0.7578 - val_loss: 0.7113 - val_accuracy: 0.7206
Epoch 17/20
112/112 [==============================] - 1s 12ms/step - loss: 0.63
06 - accuracy: 0.7616 - val_loss: 0.7217 - val_accuracy: 0.7206
Epoch 18/20
112/112 [==============================] - 1s 12ms/step - loss: 0.62
47 - accuracy: 0.7647 - val_loss: 0.7130 - val_accuracy: 0.7218
Epoch 19/20
112/112 [==============================] - 1s 7ms/step - loss: 0.616
2 - accuracy: 0.7642 - val_loss: 0.7159 - val_accuracy: 0.7226
Epoch 20/20
112/112 [==============================] - 1s 7ms/step - loss: 0.609
6 - accuracy: 0.7670 - val_loss: 0.7158 - val_accuracy: 0.7306
```

Evaluating the model

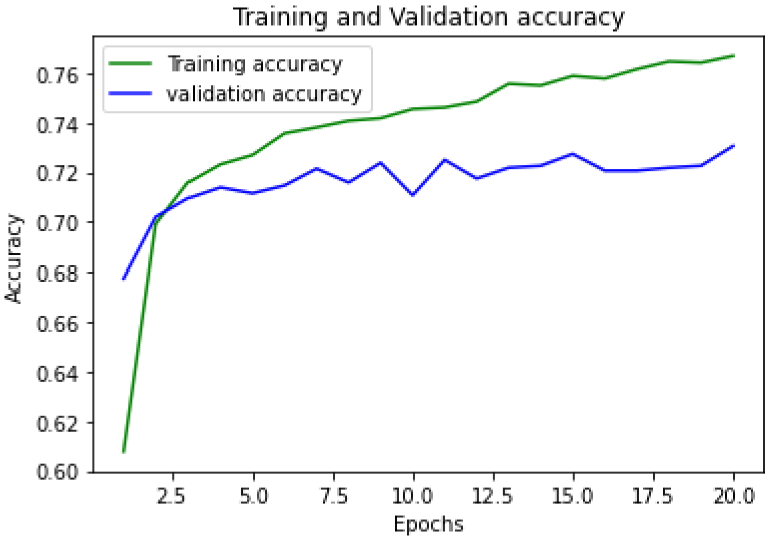
In [40]:

```
loss_train = history.history['loss']
loss_val = history.history['val_loss']
epochs = range(1,21)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [41]:

```
loss_train = history.history['accuracy']
loss_val = history.history['val_accuracy']
epochs = range(1,21)
plt.plot(epochs, loss_train, 'g', label='Training accuracy')
plt.plot(epochs, loss_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [42]:

```
score = model.evaluate(X_test, y_test, verbose=1)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))
```

```
132/132 [==============================] - 1s 5ms/step - loss: 0.752
0 - accuracy: 0.7157
accuracy: 71.57%
```

In [43]:

```
y_pred = model.predict(X_test)

y_pred = (y_pred > 0.5)
```

In [44]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.74      0.74      1529
           1       0.69      0.55      0.61       923
           2       0.00      0.00      0.00       275
           3       0.79      0.80      0.79      1473

   micro avg       0.75      0.67      0.71      4200
   macro avg       0.56      0.52      0.54      4200
weighted avg       0.70      0.67      0.68      4200
 samples avg       0.67      0.67      0.67      4200
```

function for creating confusion matrix

In [45]:

```
def print_confusion_matrix(confusion_matrix, class_names, title='Confusion matri
x', figsize = (6,6), fontsize=14):
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha=
'right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha
='right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title(title, fontsize=20)

    return fig
```

printing the confusion matrix

In [46]:

```
from sklearn.metrics import multilabel_confusion_matrix

cnf_matrix = multilabel_confusion_matrix(y_test, y_pred).reshape(4*1, -1)
classes = [str(x) for x in list(y_test.columns.values.tolist())]

print_confusion_matrix(cnf_matrix, classes);
```