In [ ]:
```
# This notebook is designed to run on Google colab.
# Submitted by : Syam Mohan
# Student ID : S3857836
```

0.0.1 Introduction

This notebook contains, experiments conducted to build a multimodal model for identifying the persuasion techniques used in memes. The challenge was introduces as Task 6 in SemEval 2021.

0.0.2 Import Libraries

In [1]:
```
!pip uninstall -y tensorflow
!pip install -q tensorflow==2.8.2
```

```
Found existing installation: tensorflow 2.9.2
Uninstalling tensorflow-2.9.2:
  Successfully uninstalled tensorflow-2.9.2
     |████████████████████████████████| 437.8 MB 35 kB/s
     |████████████████████████████████| 462 kB 33.3 MB/s
     |████████████████████████████████| 5.8 MB 63.1 MB/s
     |████████████████████████████████| 1.4 MB 61.5 MB/s
```

In [2]:
```
import os
import sys
import time
import json
import pickle
import subprocess
import pandas as pd
import numpy as np

from scipy.ndimage import rotate, shift
from collections import defaultdict

import tensorflow as tf
from tensorflow import  keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Lambda, Input
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.metrics import categorical_accuracy
from tensorflow.keras import regularizers, optimizers
from tensorflow.keras import backend as K
from tensorflow.keras.utils import plot_model

from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.preprocessing import MultiLabelBinarizer
from google.colab import files

from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_auc_score, roc_curve, f1_score, precision_recall_curve, confusion_matrix, average_precisio
```

```
import pathlib
import shutil
import tempfile
from itertools import cycle
from PIL import Image
from google.colab import files

import seaborn as sns
import matplotlib.style as style
import matplotlib.pyplot as plt
%matplotlib inline
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']

!pip install -q -U "tensorflow-text==2.8.*"
!pip install -q tf-models-official==2.7.0
!pip install -q transformers
!pip install -q tf_explain

import tensorflow_addons as tfa
import tensorflow_hub as hub
import tensorflow_text as text
from official.nlp import optimization  # to create AdamW optimizer
tf.get_logger().setLevel('ERROR')

try:
    import visualkeras
except ImportError:
    !pip install visualkeras
    import visualkeras
```

```
     |████████████████████████████████| 4.9 MB 36.2 MB/s
     |████████████████████████████████| 1.8 MB 30.8 MB/s
     |████████████████████████████████| 116 kB 65.4 MB/s
     |████████████████████████████████| 238 kB 52.2 MB/s
     |████████████████████████████████| 43 kB 1.8 MB/s
     |████████████████████████████████| 352 kB 8.5 MB/s
     |████████████████████████████████| 99 kB 8.0 MB/s
     |████████████████████████████████| 1.3 MB 47.8 MB/s
     |████████████████████████████████| 1.1 MB 51.4 MB/s
  Building wheel for py-cpuinfo (setup.py) ... done
  Building wheel for seqeval (setup.py) ... done
     |████████████████████████████████| 5.3 MB 34.8 MB/s
     |████████████████████████████████| 7.6 MB 57.5 MB/s
     |████████████████████████████████| 163 kB 64.9 MB/s
     |████████████████████████████████| 43 kB 57 kB/s
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/si
mple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting visualkeras
  Downloading visualkeras-0.0.2-py3-none-any.whl (12 kB)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.7/dist-packages (from visualkeras) (1.21.6)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/dist-packages (from visualkeras) (7.1.2)
Collecting aggdraw>=1.3.11
  Downloading aggdraw-1.3.15-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (989 kB)
     |████████████████████████████████| 989 kB 36.9 MB/s
Installing collected packages: aggdraw, visualkeras
Successfully installed aggdraw-1.3.15 visualkeras-0.0.2
```

0.0.3 Helper Functions

In [3]:
```
def get_df_from_json(txt_file_path):
  with open(txt_file_path, encoding='utf-8') as temp_file:
    _data = pd.DataFrame(json.load(temp_file))
  return _data
```

In [4]:
```
def download_files(m_histories, model_name):

    """Function to download files from google colab"""

    print('downloading history..')
    with open(model_name+'_history.json', 'wb') as file:
        pickle.dump(m_histories[model_name].history, file)
        files.download(model_name+'_history.json')

    process = subprocess.Popen(['zip', '-rq', model_name+'.zip', '/content/tensorboard_logs/models/'+model_name],
                     stdout=subprocess.PIPE,
                     stderr=subprocess.PIPE)
    time.sleep(30)
    print('downloading tensorboard logs..')
    files.download(model_name+'.zip')
```

In [5]:
```
def prediction_df(df, prediction, mlb):
  prediction_df = pd.DataFrame()
  prediction_df['actual'] = df['labels']
  prediction_df['model1_predicted'] = [inverse_transform_mlb(x, mlb) for x in prediction]
  prediction_df['actual_labels_mh'] = df['labels_mh']
  prediction_df['predicted_labels_mh'] = list(prediction)
  return prediction_df
```

In [6]:
```
def inverse_transform_mlb(list_pred, mlb):
  """
  Function to apply inverse transform and
  get the actual class names from the prediction 1-dimensional arrays

  Inputs:
    list_pred : 1 dimensional integer array with 1 or 0 values
    mlb : the multi label binarizer object
  """
  y = pd.Series(list_pred)
  y.index = mlb.classes_
  y = y[y==1].index.values
  return list(y)
```

In [7]:
```
def get_value_dict(gt, pred, target_names):
  target_actual_dict = defaultdict(list)
  target_pred_dict = defaultdict(list)
  for i, target in enumerate(target_names):
    for data in gt:
      target_actual_dict[target].append(data[i])
      target_pred_dict[target].append(data[i])
  return target_actual_dict, target_pred_dict
```

In [8]:
```
def plotter(history_hold, metric = 'binary_crossentropy', ylim=[0.0, 1.0]):
    plt.figure(figsize=(8,8))
    cycol = cycle('bgrcmk')
    for name, item in history_hold.items():
        y_train = item.history[metric]
        y_val = item.history['val_' + metric]
        x_train = np.arange(0,len(y_val))

        c=next(cycol)

        plt.plot(x_train, y_train, c+'-', label=name+'_train')
        plt.plot(x_train, y_val, c+'--', label=name+'_val')

    plt.legend()
    plt.xlim([0, max(plt.xlim())])
    plt.ylim(ylim)
    plt.xlabel('Epoch')
    plt.ylabel(metric)
    plt.grid(True)
```

In [9]:
```
def plot_roc(name, labels, predictions, **kwargs):
    fp, tp, _ = metrics.roc_curve(labels, predictions)

    plt.plot(100*fp, 100*tp, label=name, linewidth=2, **kwargs)
    plt.xlabel('False positives [%]')
    plt.ylabel('True positives [%]')
    plt.xlim([-0.5,20])
    plt.ylim([80,100.5])
    plt.grid(True)
    ax = plt.gca()
    ax.set_aspect('equal')
```

In [10]:
```
def plot_metrics(history):
    metrics = ['loss', 'prc', 'precision', 'recall']

    plt.figure(figsize=(12,5))
    for n, metric in enumerate(metrics):
        name = metric.replace("_"," ").capitalize()
        plt.subplot(2,2,n+1)
        plt.plot(history.epoch, history.history[metric], color=colors[0], label='Train')
        plt.plot(history.epoch, history.history['val_'+metric],
                 color=colors[0], linestyle="--", label='Val')
        plt.xlabel('Epoch')
        plt.ylabel(name)
        if metric == 'loss':
            plt.ylim([0, plt.ylim()[1]])
        elif metric == 'auc':
            plt.ylim([0.8,1])
        else:
            plt.ylim([0,1])

        plt.legend()
```

In [11]:
```
def plot_roc_curve(gt, pred, target_names):
    fig,axs = plt.subplots(1,len(target_names),figsize=(60,10))
    for i in range(len(target_names)):
        curve_function = roc_curve
        auc_roc = roc_auc_score(gt[:, i], pred[:, i])
        label = target_names[i] + " AUC: %.3f " % auc_roc
        xlabel = "False positive rate"
        ylabel = "True positive rate"
        a, b, _ = curve_function(gt[:, i], pred[:, i])
        #plt.figure(1, figsize=(7, 7))
        axs[i].plot([0, 1], [0, 1], 'k--')
        axs[i].plot(a, b, label=label)
        axs[i].set_xlabel(xlabel)
        axs[i].set_ylabel(ylabel)
        axs[i].legend(loc='upper center',
                      fancybox=True, ncol=1, prop={'size': 6})
    plt.savefig('ROC_Curve.png')
    plt.show()
```

In [12]:
```
def plot_labelwise_confusion_matrix_(y_true,y_pred,class_labels):
    fig, axs = plt.subplots(int(len(class_labels)/3)+1,3,figsize=(60,80))
    for n in range(len(class_labels)):
        cm = np.zeros((2,2))
        cm[0,0] = np.sum((y_true[:,n]==1) & (y_pred[:,n]>0.5))
        cm[0,1] = np.sum((y_true[:,n]==0) & (y_pred[:,n]>0.5))
        cm[1,0] = np.sum((y_true[:,n]==1) & (y_pred[:,n]<0.5))
        cm[1,1] = np.sum((y_true[:,n]==0) & (y_pred[:,n]<0.5))
        cm_sum = np.sum(cm, axis=1, keepdims=True)
        cm_perc = cm / cm_sum.astype(float) * 100
        annot = np.empty_like(cm).astype(str)
        nrows, ncols = cm.shape
        for i in range(nrows):
            for j in range(ncols):
                c = cm[i, j]
                p = cm_perc[i, j]
                if i == j:
                    s = cm_sum[i]
                    annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
                elif c == 0:
                    annot[i, j] = ''
                else:
                    annot[i, j] = '%.1f%%\n%d' % (p, c)

        class_labs = ['1','0']

        cm = pd.DataFrame(cm, index=class_labs, columns=class_labs)
        cm.index.name = 'Actual'
        cm.columns.name = 'Predicted'
        axs[int(n/3),n%3].set_ylabel('Actual', fontsize=25)
        axs[int(n/3),n%3].set_xlabel('Predicted', fontsize=25)
        #fig, ax = plt.subplots(figsize=(4,4))
        axs[int(n/3),n%3].set_title('Confusion Matrix for %s'%(class_labels[n]), fontsize=25)
        sns.heatmap(cm, cmap= "YlGnBu", annot=annot, fmt='', ax=axs[int(n/3),n%3], annot_kws={"fontsize":50})
        #fig.savefig('Confusion_Matrix.png')
    plt.show()
```

In [13]:
```
def plot_pr_curve(name, labels, predictions, **kwargs):
    precision, recall, _ = metrics.precision_recall_curve(labels, predictions)

    plt.plot(precision, recall, label=name, linewidth=2, **kwargs)
    plt.xlabel('Precision')
    plt.ylabel('Recall')
    plt.grid(True)
    ax = plt.gca()
    ax.set_aspect('equal')
```

In [14]:
```
def plot_confusion_matrix(true_value, predictions, class_labels):
    cm = metrics.confusion_matrix(true_value, predictions)
    plt.figure(figsize=(10,10))
    sns.heatmap(cm, annot=True, fmt='d',
             xticklabels=class_labels, yticklabels=class_labels, cbar=False)
    plt.title('Confusion matrix')
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
```

In [15]:
```
def f1_score(y_true, y_pred): #taken from old keras source code
    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)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

In [16]:
```
m_histories = {}
models = {}
```

0.0.4 Lauching Tensorboard

In [17]:
```
logdir = os.path.join(os.getcwd(),"tensorboard_logs")
pathlib.Path(logdir).mkdir(parents=True, exist_ok=True)
```

In [ ]:
```
# Load the TensorBoard notebook extension
%load_ext tensorboard

# Open an embedded TensorBoard viewer
%tensorboard --logdir {logdir}/models
```

0.0.5 Creating Directories and Files

In [19]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [20]:
```
!cp /content/drive/MyDrive/DeepLearning/assignment2/subtask3.zip .
```

In [21]:
```
!unzip -q -o subtask3.zip
!unzip -q -o subtask3/dev_set_task3.zip -d subtask3/
!unzip -q -o subtask3/test_set_task3.zip -d subtask3/
!unzip -q -o subtask3/training_set_task3.zip -d subtask3/

!rm subtask3.zip
!rm subtask3/dev_set_task3.zip
!rm subtask3/test_set_task3.zip
!rm subtask3/training_set_task3.zip
```

In [22]:
```
assignment_path = os.path.join(os.getcwd())
data_path = os.path.join(assignment_path, 'subtask3')
target_path = os.path.join(data_path, 'techniques_list_task3.txt')
```

In [23]:
```
training_set_info = os.path.join(data_path, 'training_set_task3', 'training_set_task3.txt')
test_set_info = os.path.join(data_path, 'test_set_task3', 'test_set_task3.txt')
dev_set_info = os.path.join(data_path, 'dev_set_task3_labeled', 'dev_set_task3_labeled.txt')
```

In [24]:
```
training_set_path = os.path.join(data_path, 'training_set_task3')
test_set_path = os.path.join(data_path, 'test_set_task3')
dev_set_path = os.path.join(data_path, 'dev_set_task3_labeled')
```

0.0.6 Loading Target Techniques List

In [25]:
```
with open(target_path, 'r') as file:
  target_labels = file.read().split('\n')
```

In [26]:
```
target_labels
```

Out[26]:
```
['Appeal to authority',
 'Appeal to fear/prejudice',
 'Black-and-white Fallacy/Dictatorship',
 'Causal Oversimplification',
 'Doubt',
 'Exaggeration/Minimisation',
 'Flag-waving',
 'Glittering generalities (Virtue)',
 'Loaded Language',
 "Misrepresentation of Someone's Position (Straw Man)",
 'Name calling/Labeling',
 'Obfuscation, Intentional vagueness, Confusion',
 'Presenting Irrelevant Data (Red Herring)',
 'Reductio ad hitlerum',
 'Repetition',
 'Slogans',
 'Smears',
 'Thought-terminating cliché',
 'Whataboutism',
 'Bandwagon',
 'Transfer',
 'Appeal to (Strong) Emotions',
 '',
 '']
```

0.0.7 Loading Train, Test, Dev Datasets

In [27]:
```
train_df = get_df_from_json(training_set_info)
test_df = get_df_from_json(test_set_info)
dev_df = get_df_from_json(dev_set_info)
```

In [28]:
```
train_df.head()
```

Out[28]:

	id	labels	text	image
0	128	[Black-and-white Fallacy/Dictatorship, Name ca...	THERE ARE ONLY TWO GENDERS\n\nFEMALE \n\nMALE\n	128_image.png
1	189	[Reductio ad hitlerum, Smears, Transfer]	This is not an accident!	189_image.png
2	96	[Appeal to fear/prejudice, Loaded Language, Na...	SO BERNIE BROS HAVEN'T COMMITTED VIOLENCE EH?\...	96_image.png
3	154	[Causal Oversimplification, Glittering general...	PATHETIC\n\nThe Cowardly Asshole\nWeak Failure...	154_image.png
4	15	[Flag-waving, Misrepresentation of Someone's P...	WHO TRUMP REPRESENTS\n\nWHO DEMOCRATS REPRESENT\n	15_image.png

In [29]:
```
test_df.head()
```

Out[29]:

	id	labels	text	image
0	705_batch_2	[Name calling/Labeling, Slogans, Smears, Trans...	The Democrats New America\n	705_image_batch_2.png
1	706_batch_2	[Appeal to (Strong) Emotions, Appeal to fear/p...	WE ARE AT WAR!\n\nThere is a complex assault o...	706_image_batch_2.png
2	710_batch_2	[Doubt, Loaded Language, Name calling/Labeling]	KILLED HIMSELF IN PRISON\n\nWON AN HONEST ELEC...	710_image_batch_2.png
3	713_batch_2	[Exaggeration/Minimisation, Loaded Language]	I will never concede!\n\nNO WAY IN HELL BIDEN ...	713_image_batch_2.png
4	715_batch_2	[Doubt]	TRYING TO FIGURE OUT HOW BIDEN WON A RECORD LO...	715_image_batch_2.png

In [30]:
```
dev_df.head()
```

Out[30]:

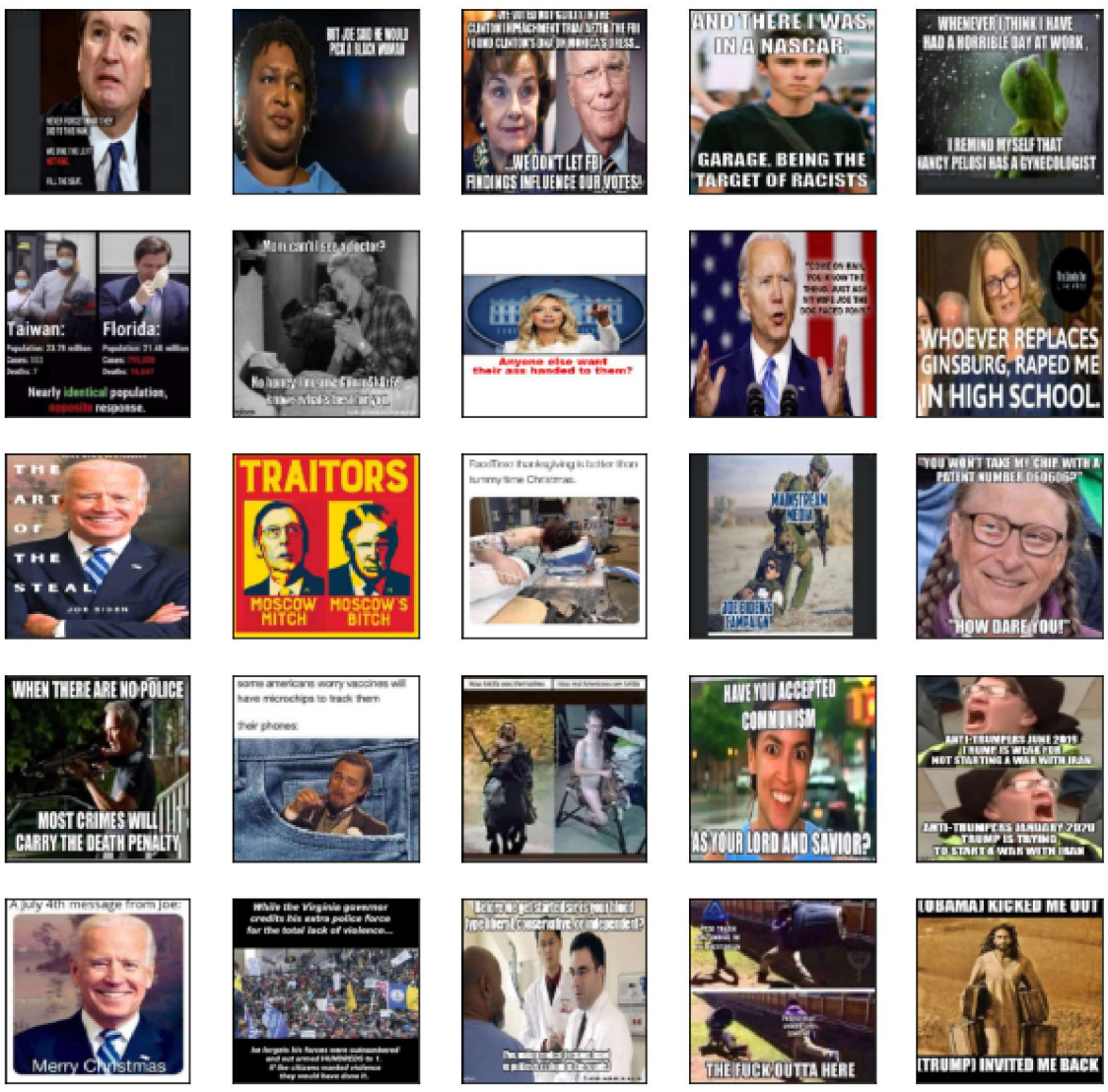
	id	labels	text	image
0	62_batch_2	[Smears, Doubt]	*President* Biden?\n\nPlease, no.\n	62_image_batch_2.png
1	111_batch_2	[Smears, Loaded Language, Name calling/Labeling]	JOE VERSUS THE VOLCANIC KREMLIN DON\n\n"WILL ...	111_image_batch_2.png
2	167_batch_2	[Smears, Transfer]	ANTI-VAXXERS BE LIKE... \n\nHANG ON A SEC - JU...	167_image_batch_2.png
3	93_batch_2	[]	VIRUS BINGO\nFREE 32 SPACE\n	93_image_batch_2.png
4	153_batch_2	[Smears, Exaggeration/Minimisation, Loaded Lan...	Never thought I'd die fighting IRRESPONSIBLY R...	153_image_batch_2.png

In [31]:
```
print("Shape of training data: ", train_df.shape)
print("Shape of testing data: ", test_df.shape)
print("Shape of dev data: ", dev_df.shape)
```

```
Shape of training data:  (687, 4)
Shape of testing data:  (200, 4)
Shape of dev data:  (63, 4)
```

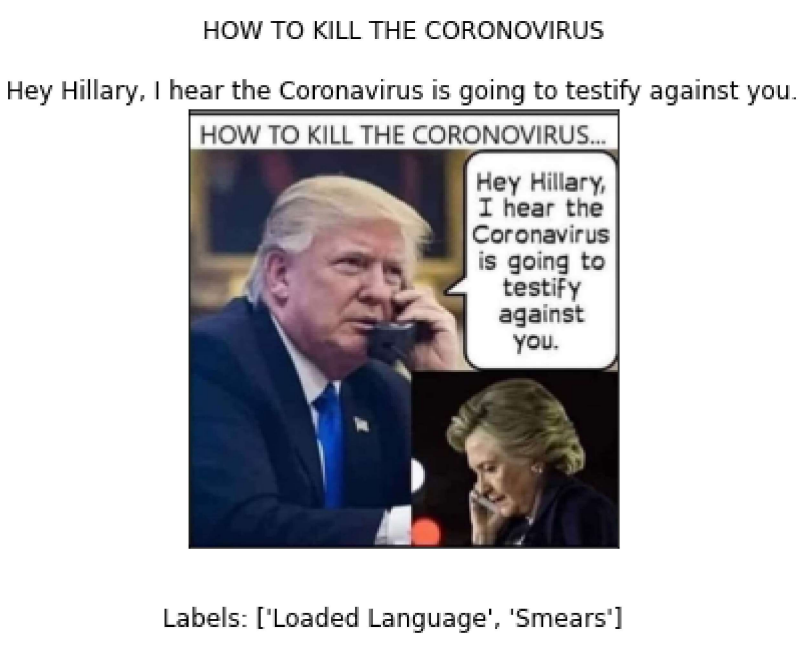
0.0.8 Sample Image Analysis

In [32]:
```
plt.figure(figsize=(10,10))
for id, (i, row) in enumerate(train_df.sample(25).iterrows()):
  plt.subplot(5, 5, id+1)
  # img = Image.open(os.path.join(training_set_path, row['image']))
  img = tf.keras.utils.load_img(os.path.join(training_set_path, row['image']), grayscale=False, color_mode='rgb', target_s
  plt.xticks([]) # removing the ticks on x-axis
  plt.yticks([])
  plt.grid(False)
  plt.imshow(img)
plt.show()
```



0.0.9 Sample Image with Label and Text

In [33]:
```
index = np.random.choice(train_df.index)
_data = train_df.iloc[index]
img = Image.open(os.path.join(training_set_path, _data['image']))
plt.title(_data['text'])
plt.xticks([]) # removing the ticks on x-axis
plt.yticks([])
plt.figtext(0.5, -0.01, "Labels: "+ str(_data['labels']), wrap=True, horizontalalignment='center', fontsize=12)
plt.imshow(img)
plt.show()
```



Labels: ['Loaded Language', 'Smears']

✅ Obeservations :

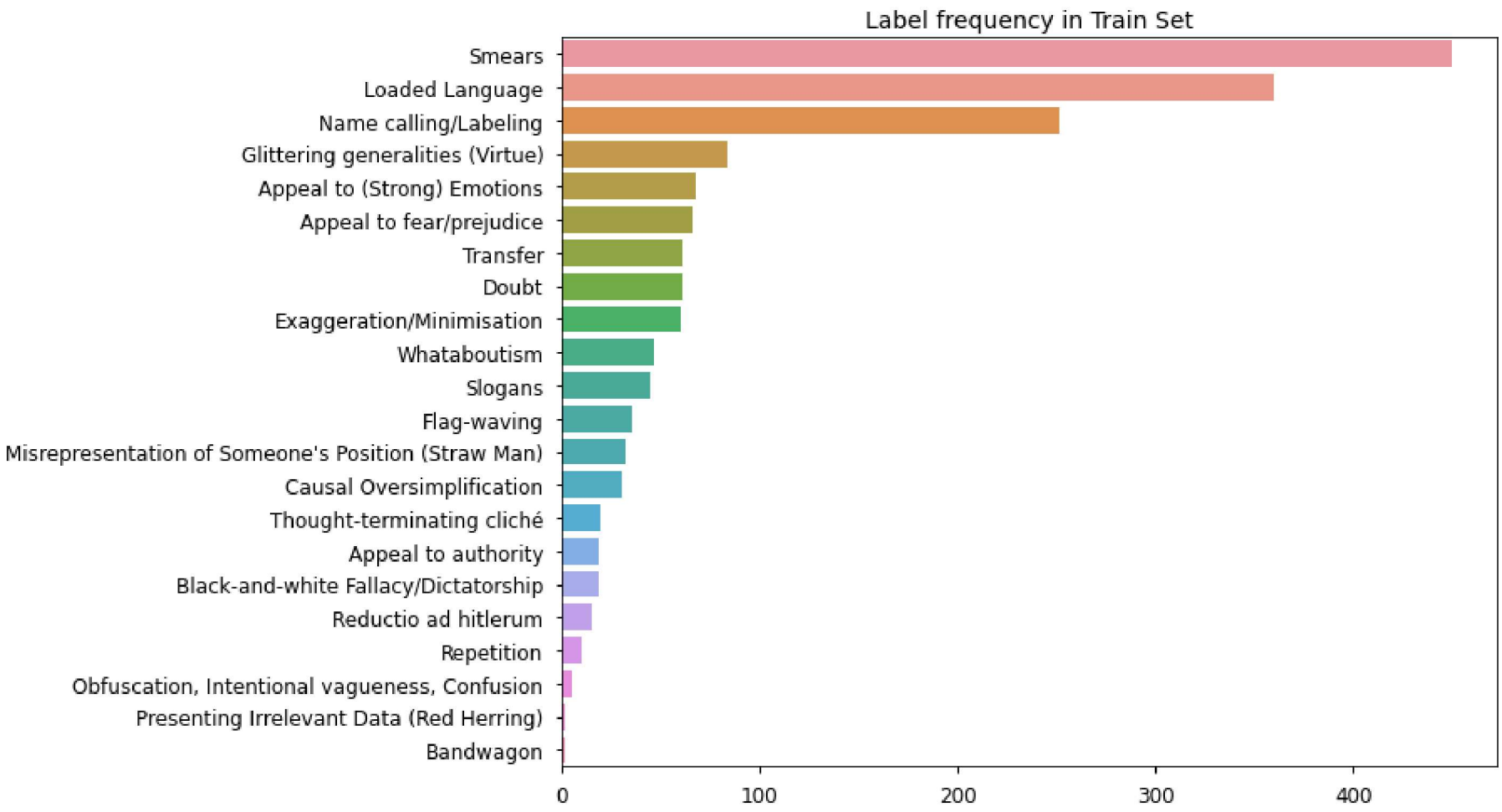
- The dataset contains 687 images and text in training set. 63 samples in the dev set and 200 samples in the test set.

- The images are of different dimensions.
- The text is an OCR extraction from the text in the meme.
- The longest sentence in the text contains 64 words.

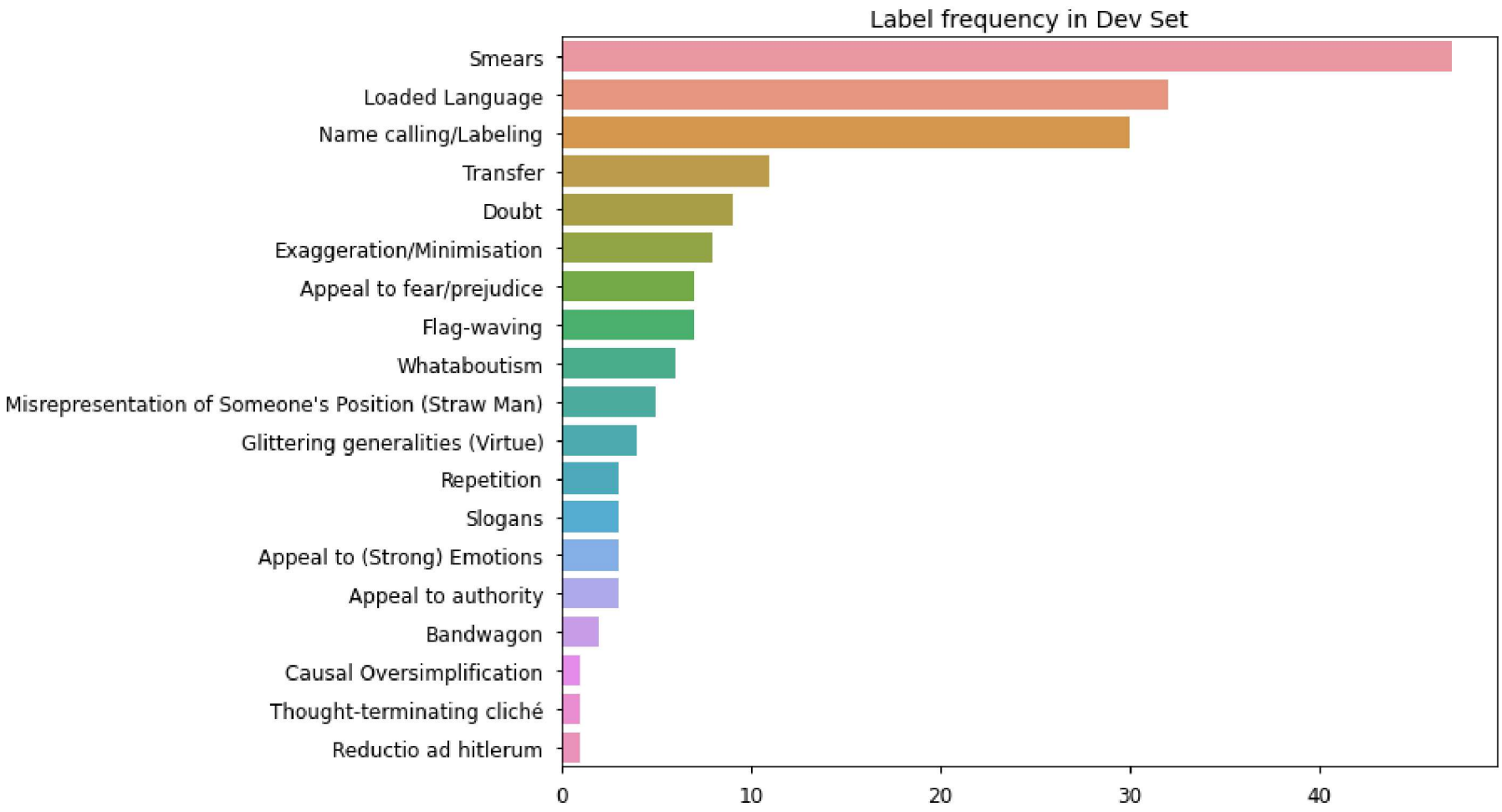
0.1 Preparing the Target Label

0.1.1 Checking the distribution of Target Label in each split of the data

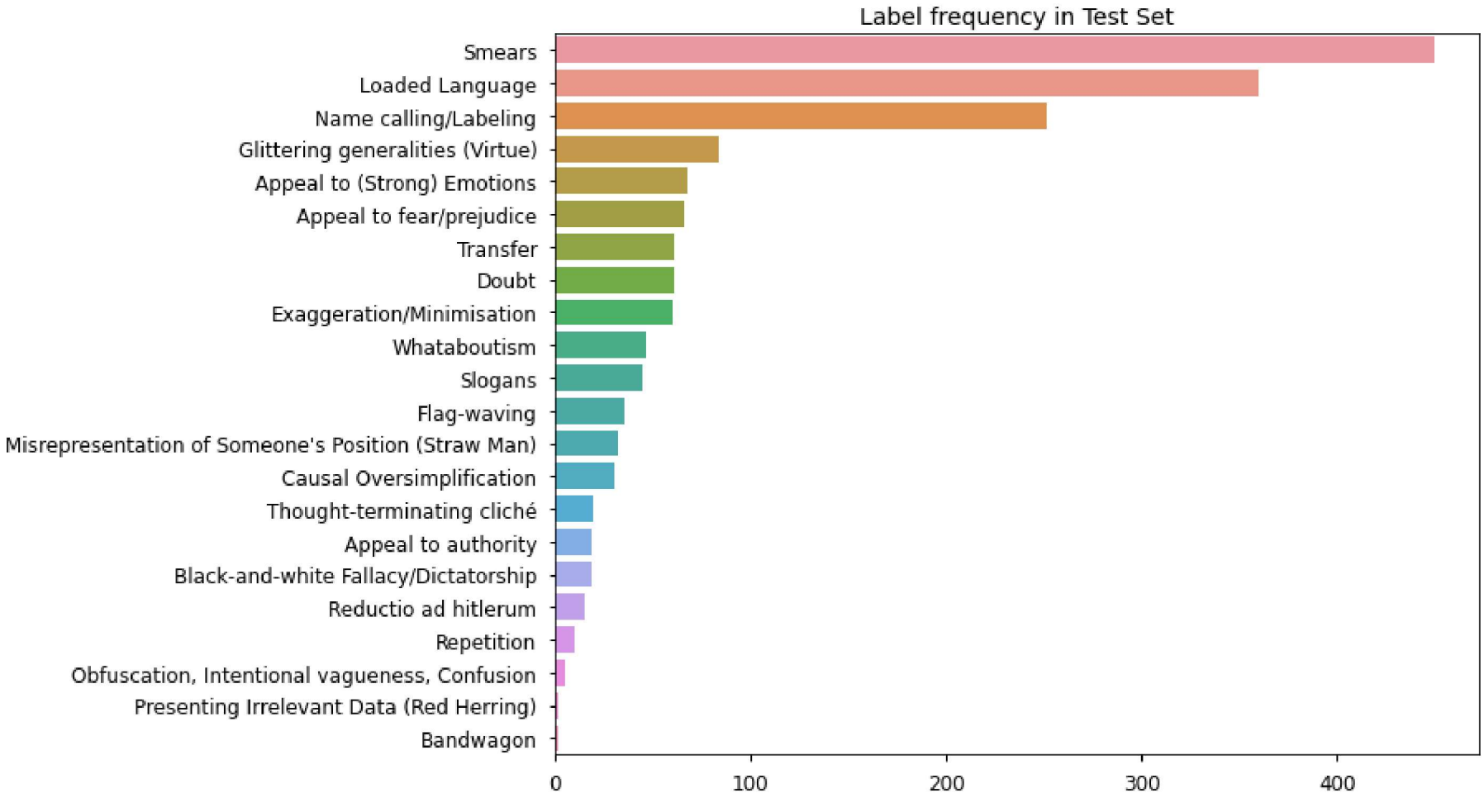
In [34]:
```
# Get label frequencies in descending order
label_freq_train = train_df['labels'].explode().value_counts().sort_values(ascending=False)

# Bar plot
style.use("seaborn-notebook")
plt.figure(figsize=(10,8))
sns.barplot(y=label_freq_train.index.values, x=label_freq_train, order=label_freq_train.index)
plt.title("Label frequency in Train Set", fontsize=14)
plt.xlabel("")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



In [35]:
```
# Get label frequencies in descending order
label_freq_dev = dev_df['labels'].explode().value_counts().sort_values(ascending=False)

# Bar plot
style.use("seaborn-notebook")
plt.figure(figsize=(10,8))
sns.barplot(y=label_freq_dev.index.values, x=label_freq_dev, order=label_freq_dev.index)
plt.title("Label frequency in Dev Set", fontsize=14)
plt.xlabel("")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



In [36]:
```
# Get label frequencies in descending order
test_freq_test = train_df['labels'].explode().value_counts().sort_values(ascending=False)

# Bar plot
style.use("seaborn-notebook")
plt.figure(figsize=(10,8))
sns.barplot(y=test_freq_test.index.values, x=test_freq_test, order=test_freq_test.index)
plt.title("Label frequency in Test Set", fontsize=14)
plt.xlabel("")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



✅ Obeservations :

- As seen from the above plot, the dataset is highly imbalanced.
- Classes like Smears, loaded language and name calling forms the majority class.

0.1.2 Creating a Multi-Hot Encoded Target Vector

- As the task is multi-label classification problem, the target labels have to be a multi-hot vector.

In [37]:
```
all_target_labels = pd.concat([train_df['labels'], dev_df['labels'], test_df['labels']], axis=0).reset_index(drop=True)
all_target_labels
```

Out[37]:
```
0      [Black-and-white Fallacy/Dictatorship, Name ca...
1               [Reductio ad hitlerum, Smears, Transfer]
2      [Appeal to fear/prejudice, Loaded Language, Na...
3      [Causal Oversimplification, Glittering general...
4      [Flag-waving, Misrepresentation of Someone's P...
                             ...
945                                           [Smears]
946                                    [Loaded Language]
947                      [Loaded Language, Smears]
948                                           [Smears]
949                        [Name calling/Labeling]
Name: labels, Length: 950, dtype: object
```

In [38]:
```
# Fit the multi-label binarizer on all the target labels
# present in the data corpus(train+dev+test)
mlb = MultiLabelBinarizer()
mlb.fit(all_target_labels)
mlb.classes_
```

Out[38]:
```
array(['Appeal to (Strong) Emotions', 'Appeal to authority',
       'Appeal to fear/prejudice', 'Bandwagon',
       'Black-and-white Fallacy/Dictatorship',
       'Causal Oversimplification', 'Doubt', 'Exaggeration/Minimisation',
       'Flag-waving', 'Glittering generalities (Virtue)',
       'Loaded Language',
       "Misrepresentation of Someone's Position (Straw Man)",
       'Name calling/Labeling',
       'Obfuscation, Intentional vagueness, Confusion',
       'Presenting Irrelevant Data (Red Herring)', 'Reductio ad hitlerum',
       'Repetition', 'Slogans', 'Smears', 'Thought-terminating cliché',
       'Transfer', 'Whataboutism'], dtype=object)
```

In [39]:
```
# Loop over all labels and corresponding integer label
N_LABELS = len(mlb.classes_)
for (i, label) in enumerate(mlb.classes_):
    print("{}. {}".format(i, label))
```

```
0. Appeal to (Strong) Emotions
1. Appeal to authority
2. Appeal to fear/prejudice
3. Bandwagon
4. Black-and-white Fallacy/Dictatorship
5. Causal Oversimplification
6. Doubt
7. Exaggeration/Minimisation
8. Flag-waving
9. Glittering generalities (Virtue)
10. Loaded Language
11. Misrepresentation of Someone's Position (Straw Man)
12. Name calling/Labeling
13. Obfuscation, Intentional vagueness, Confusion
14. Presenting Irrelevant Data (Red Herring)
15. Reductio ad hitlerum
16. Repetition
17. Slogans
18. Smears
19. Thought-terminating cliché
20. Transfer
21. Whataboutism
```

In [40]:
```
train_labels_mh = mlb.transform(list(train_df['labels']))
train_df['labels_mh'] = list(train_labels_mh)
train_df.head()
```

Out[40]:

	id	labels	text	image	labels_mh
0	128	[Black-and-white Fallacy/Dictatorship, Name ca...	THERE ARE ONLY TWO GENDERS\n\nFEMALE \n\nMALE\n	128_image.png	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
1	189	[Reductio ad hitlerum, Smears, Transfer]	This is not an accident!	189_image.png	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2	96	[Appeal to fear/prejudice, Loaded Language, Na...	SO BERNIE BROS HAVEN'T COMMITTED VIOLENCE EH? \...	96_image.png	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
3	154	[Causal Oversimplification, Glittering general...	PATHETIC\n\nThe Cowardly Asshole\nWeak Failure...	154_image.png	[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, ...
4	15	[Flag-waving, Misrepresentation of Someone's P...	WHO TRUMP REPRESENTS\n\nWHO DEMOCRATS REPRESENT\n	15_image.png	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, ...

0.1.3 Validating the Multi-Hot Vector

In [41]:
```
print("Original Label: ")
print(train_df['labels'][0], '\n')
print("Multi-Hot encoded label: ")
print(train_df['labels_mh'][0], '\n')
print("Label after applying inverse transform : ")
print(mlb.inverse_transform(np.reshape(train_df['labels_mh'][0], (1,22))))
```

```
Original Label:
['Black-and-white Fallacy/Dictatorship', 'Name calling/Labeling', 'Smears']

Multi-Hot encoded label:
[0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0]

Label after applying inverse transform :
[('Black-and-white Fallacy/Dictatorship', 'Name calling/Labeling', 'Smears')]
```

0.1.4 Applying the transform to Dev and Test sets

In [42]:
```
dev_labels_mh = mlb.transform(list(dev_df['labels']))
dev_df['labels_mh'] = list(dev_labels_mh)

test_labels_mh = mlb.transform(list(test_df['labels']))
test_df['labels_mh'] = list(test_labels_mh)
```

In [43]:
```
dev_df.head()
```

Out[43]:

	id	labels	text	image	labels_mh
0	62_batch_2	[Smears, Doubt]	*President* Biden?\n\nPlease, no.\n	62_image_batch_2.png	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	111_batch_2	[Smears, Loaded Language, Name calling/Labeling]	JOE VERSUS THE VOLCANIC KREMLIN DON\n\n"WILL ...	111_image_batch_2.png	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
2	167_batch_2	[Smears, Transfer]	ANTI-VAXXERS BE LIKE... \n\nHANG ON A SEC - JU...	167_image_batch_2.png	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	93_batch_2	[]	VIRUS BINGO\nFREE 32 SPACE\n	93_image_batch_2.png	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	153_batch_2	[Smears, Exaggeration/Minimisation, Loaded Lan...	Never thought I'd die fighting IRRESPONSIBLY R...	153_image_batch_2.png	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, ...

In [44]:
```
test_df.head()
```

Out[44]:

	id	labels	text	image	labels_mh
0	705_batch_2	[Name calling/Labeling, Slogans, Smears, Trans...	The Democrats New America\n	705_image_batch_2.png	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
1	706_batch_2	[Appeal to (Strong) Emotions, Appeal to fear/p...	WE ARE AT WAR!\n\nThere is a complex assault o...	706_image_batch_2.png	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
2	710_batch_2	[Doubt, Loaded Language, Name calling/Labeling]	KILLED HIMSELF IN PRISON\n\nWON AN HONEST ELEC...	710_image_batch_2.png	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, ...
3	713_batch_2	[Exaggeration/Minimisation, Loaded Language]	I will never concede!\n\nNO WAY IN HELL BIDEN ...	713_image_batch_2.png	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, ...
4	715_batch_2	[Doubt]	TRYING TO FIGURE OUT HOW BIDEN WON A RECORD LO...	715_image_batch_2.png	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...

0.1.5 Checking Distribution of Positive and Negative Samples

In [45]:
```
def dataset_distribution(df):
  total_count = df['labels_mh'].count()*22
  positive_sum = df['labels_mh'].sum().sum()
  negative_sum = total_count - positive_sum

  # return pd.DataFrame({'positive(%)': positive_sum/total_count * 100, 'negative(%)': negative_sum/total_count *100}, ind
  return {'positive(%)': str(positive_sum)+str('({}%)'.format(round(positive_sum/total_count * 100, 2))), 'negative(%)': s
```

In [46]:
```
distribution_df = pd.DataFrame([dataset_distribution(x) for x in [train_df, dev_df, test_df]], index=['train', 'dev', 'tes
distribution_df
```

Out[46]:

	positive(%)	negative(%)
train	1745(11.55%)	13369(88.45%)
dev	183(13.2%)	1203(86.8%)
test	523(11.89%)	3877(88.11%)

✅ Obeservations :

- As mentioned earlier the data contains more negative samples.

0.1.6 Calculating Class Weights

- Weighting the loss function of the model is one approach that can combat the issue of class imbalance in the dataset.

In [51]:
```
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
from sklearn.preprocessing import MultiLabelBinarizer


def generate_class_weights(class_series, multi_class=True, one_hot_encoded=False):
  """
  Method to generate class weights given a set of multi-class or multi-label labels, both one-hot-encoded or not.

  Some examples of different formats of class_series and their outputs are:
    - generate_class_weights(['mango', 'lemon', 'banana', 'mango'], multi_class=True, one_hot_encoded=False)
    {'banana': 1.3333333333333333, 'lemon': 1.3333333333333333, 'mango': 0.6666666666666666}
    - generate_class_weights([[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, 0]], multi_class=True, one_hot_encoded=True)
    {0: 0.6666666666666666, 1: 1.3333333333333333, 2: 1.3333333333333333}
    - generate_class_weights([['mango', 'lemon'], ['mango'], ['lemon', 'banana'], ['lemon']], multi_class=False, one_hot_e
    {'banana': 1.3333333333333333, 'lemon': 0.4444444444444444, 'mango': 0.6666666666666666}
    - generate_class_weights([[0, 1, 1], [0, 0, 1], [1, 1, 0], [0, 1, 0]], multi_class=False, one_hot_encoded=True)
    {0: 1.3333333333333333, 1: 0.4444444444444444, 2: 0.6666666666666666}

  The output is a dictionary in the format { class_label: class_weight }. In case the input is one hot encoded, the class_
  of appareance of the label when the dataset was processed.
  In multi_class this is np.unique(class_series) and in multi-label np.unique(np.concatenate(class_series)).

  Author: Angel Igareta (angel@igareta.com)
  """
  if multi_class:
    # If class is one hot encoded, transform to categorical labels to use compute_class_weight
    if one_hot_encoded:
      class_series = np.argmax(class_series, axis=1)

    # Compute class weights with sklearn method
    class_labels = np.unique(class_series)
    class_weights = compute_class_weight(class_weight='balanced', classes=class_labels, y=class_series)
    return dict(zip(class_labels, class_weights))
  else:
    # It is neccessary that the multi-label values are one-hot encoded
    mlb = None
    if not one_hot_encoded:
      mlb = MultiLabelBinarizer()
      class_series = mlb.fit_transform(class_series)

    n_samples = len(class_series)
    n_classes = len(class_series[0])

    # Count each class frequency
    class_count = [0] * n_classes
    for classes in class_series:
        for index in range(n_classes):
            if classes[index] != 0:
                class_count[index] += 1

    # Compute class weights using balanced method
    class_weights = [n_samples / (n_classes * freq) if freq > 0 else 1 for freq in class_count]
    class_labels = range(len(class_weights)) if mlb is None else mlb.classes_
    return dict(zip(class_labels, class_weights))
```

In [52]:
```
class_weights_dict = generate_class_weights(train_df['labels_mh'], one_hot_encoded=True, multi_class=False)
class_weights = class_weights_dict.values()
class_weights
```

Out[52]:
```
dict_values([0.4592245989304813, 1.6435406698564594, 0.4731404958677686, 15.613636363636363, 1.6435406698564594, 1.00733
13782991202, 0.5119225037257824, 0.5204545454545455, 0.8674242424242424, 0.3717532467532468, 0.08674242424242425, 0.9758
522727272727, 0.12391774891774891, 6.245454545454545, 15.613636363636363, 2.081818181818182, 3.1227272727272726, 0.69393
9393939394, 0.0693939393939394, 1.5613636363636363, 0.5119225037257824, 0.6644100580270793])
```

0.2 Data Generator

In [53]:
```
class DataGenerator(keras.utils.Sequence):
    def __init__(self, data_frame, images_path, batch_size=8, dim=(224, 224, 3), n_classes=22, data_mean=0.0, data_std=255
        self.dim = dim
        self.batch_size = batch_size
        self.n_classes = n_classes

        self.shuffle = shuffle
        self.augment = augment

        self.dataframe = data_frame
        self.image_text = data_frame['text'].values.tolist()
        self.image_label = data_frame['labels_mh'].values.tolist()
        self.image_ids = np.arange(len(self.image_label)).tolist()

        self.data_mean = data_mean
        self.data_std = data_std

        self.images_path = images_path

        self.on_epoch_end()

    def __len__(self):
        "Denotes the number of batches per epoch"
        return int(np.floor((len(self.image_ids)/self.batch_size)))
        # return (len(self.image_ids) + self.batch_size - 1)//self.batch_size

    def __getitem__(self, index):
        "Generate one batch of data for the given index"
        # * generate indexes of the batch
        # * if batch size is 8
        # * gets indexes from [0:8]
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # find list of IDs
        data_ids_temp = [self.image_ids[k] for k in indexes]
        image_text_temp = [self.image_text[k] for k in indexes]
        image_label_temp = [self.image_label[k] for k in indexes]

        # generate data
        X, y = self.__data_generation(data_ids_temp, image_text_temp, image_label_temp)

        return X, y

    def on_epoch_end(self):
        "Updates the indexes at the end of the epoch"
        self.indexes = np.arange(len(self.image_ids))
        if self.shuffle:
            np.random.shuffle(self.indexes)

    def __data_generation(self, data_ids_temp, image_text_temp, image_label_temp):
        # Initialization
        X_img = np.empty((self.batch_size, *self.dim))
        X_text = np.asarray([list]*self.batch_size)
        y = np.empty((self.batch_size, self.n_classes))

        # Generate data
        for i, ids in enumerate(data_ids_temp):
            X_img[i, ] = self.__read_data_instance(data_ids_temp[i])
            X_text[i] = image_text_temp[i]
            y[i, ] = list(image_label_temp[i])

        # * to categorical - creates a binary matrix of the input
        return [X_img, X_text], y # keras.utils.to_categorical(y, num_classes=self.n_classes)

    def __read_data_instance(self, pid):
        # * reading images from dataframe into tensor

        row = self.dataframe.iloc[pid]  # * take a single row of data (1 image)

        img = tf.keras.utils.load_img(os.path.join(self.images_path, row['image']), grayscale=False, color_mode='rgb', tar
        img = np.asarray(img) #/255.0 # this scaling helps visualize the image using imshow
        # input normalization
        img = (img - self.data_mean)/self.data_std

        if self.augment:
            rot = np.random.rand(1) < 0.5
            if rot:
                rot = np.random.randint(-45,45, size=1)
                img = rotate(img, angle=rot[0], reshape=False)

            shift_val = np.random.randint(-5, high=5, size=2, dtype=int).tolist() + [0,]
            img = shift(img, shift_val, order=0, mode='constant', cval=0.0, prefilter=False)

        X = img
        return X
```

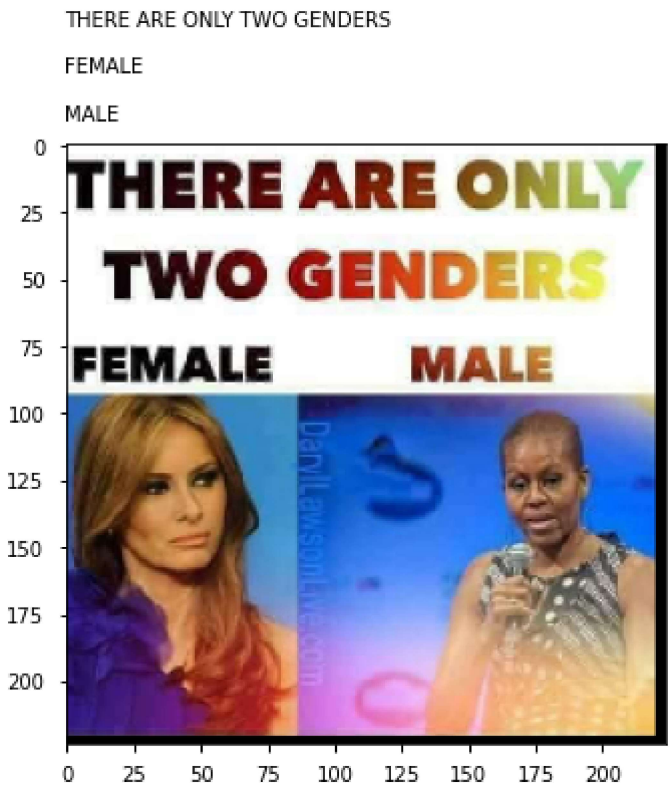
0.2.1 Testing the data generator

In [54]:
```
dg = DataGenerator(train_df, training_set_path, batch_size=2, data_std=255.0, augment=True, shuffle=False)
```

In [55]:
```
for (X, y) in dg.__iter__():
  print(X[0].shape, X[1].shape, y.shape)
  break
```

```
(2, 224, 224, 3) (2,) (2, 22)
```

In [56]:
```
for (X, y) in dg.__iter__():
  for i, (image, text) in enumerate(zip(X[0], X[1])):
    plt.imshow(image)
    plt.text(-1, -1, text)
    plt.show()
    print(y[i])
  break
```




```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```


```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0.]
```

0.3 Model Building

0.3.0.1 Define Callbacks

💡 Defining callbacks to be run while training the model. Early Stopping and model checkpointing callbacks are defined here.

- Early Stopping - is a technique to stop the model training to prevent any possible overfitting. Here, the loss of the model is monitored with a patience value=5
- Model Checkpointing - model checkpointing saves the model at different checkpoints during the training phase, here the model with the best f1 score is saved

In [57]:
```
def get_callbacks(dest_path, name, early_stop=True):
    if early_stop:
        return [
            tf.keras.callbacks.EarlyStopping(
                monitor='val_loss', patience=5, verbose=1),
            tf.keras.callbacks.TensorBoard(
                # logdir/name,
                os.path.join(dest_path, name),
                write_images=False,
                histogram_freq=60,
                embeddings_freq=60),
            tf.keras.callbacks.ModelCheckpoint(
                filepath=os.path.join(dest_path, name, name),
                save_weights_only=False,
                monitor='val_F1_macro',
                mode='max',
                verbose=1,
                save_best_only=True)
        ]
    else:
        return [tf.keras.callbacks.TensorBoard(os.path.join(dest_path, name))]
```

0.3.0.2 Instantiating the Train, Validation and Test Generators

In [58]:
```
data_mean = 0.0
data_std = 255.0 # for normalizing the data
# selecting a higher batch size, to ensure each batch includes data from
# a wide variety of classes
TRAIN_BATCH_SIZE = 8
VAL_BATCH_SIZE = 8
TEST_BATCH_SIZE = 1

training_generator = DataGenerator(
                    train_df,
                    training_set_path,
                    batch_size=TRAIN_BATCH_SIZE,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    augment=False,
                    shuffle=True
                )
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=VAL_BATCH_SIZE,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    shuffle=False
                )

test_generator = DataGenerator(
                    test_df,
                    test_set_path,
                    batch_size=TEST_BATCH_SIZE,
                    shuffle=False,
                    n_classes=22
                )
```

0.3.0.3 Setting Learning Rate Scheduler

💡 Many models train better if you gradually reduce the learning rate during training. Here we are going to decay the learning rate inversely promotional to the iteration.

In [59]:
```
N_TRAIN = len(train_df)
STEPS_PER_EPOCH = N_TRAIN//TRAIN_BATCH_SIZE

lr_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(
  0.001,
  decay_steps=STEPS_PER_EPOCH*1000,
  decay_rate=1,
  staircase=False)
```

0.3.1 Defining the Loss Function

In [60]:
```
def binary_focal_cross_entropy(y_true, y_pred):
    loss= tf.keras.losses.BinaryFocalCrossentropy(from_logits=False)
    return loss(y_true, y_pred, sample_weight=list(class_weights) )
```

In [61]:
```
def compile_and_fit(model, name, train_gen, val_gen, optimizer=None, max_epochs=10, lr_schedule=None, weights=None, num_cl

  METRICS = [
      keras.metrics.binary_crossentropy,
      keras.metrics.Precision(name='precision'),
      keras.metrics.Recall(name='recall'),
      keras.metrics.AUC(name='auc'),
      keras.metrics.AUC(name='prc', curve='PR'),  # precision-recall curve
      keras.metrics.CategoricalAccuracy(name='CategoricalAccuracy'),
      tfa.metrics.F1Score(num_classes=num_classes, average='macro', name='F1_macro'),
      tfa.metrics.F1Score(num_classes=num_classes, average='micro', name='F1_micro'),
    ]
  if optimizer is None:
    optimizer = tf.keras.optimizers.Adam(lr_schedule)

  model.compile(optimizer = optimizer,
          loss=keras.losses.BinaryFocalCrossentropy(gamma=2.0, from_logits=False),
          metrics=METRICS)

  history = model.fit(training_generator,
                      epochs=max_epochs,
                      validation_data=validation_generator,
                      verbose=1,
                      class_weight=weights,
                      callbacks=get_callbacks(os.path.join(logdir, 'models'), model_name))
  return history, model
```

0.4 Model 1: BERT + Inception -- Concatenation

💡 In this model, the text features are extracted using a BERT Transformer model and Image features are extracted using InceptionNet model. Concatenation is used as the late fusion technique.

0.4.0.1 Loading the BERT Model

In [62]:
```
# bert_model_name = 'small_bert/bert_en_uncased_L-4_H-512_A-8'
bert_model_name = 'small_bert/bert_en_uncased_L-12_H-768_A-12'
# bert_model_name = 'small_bert/bert_en_uncased_L-12_H-128_A-2'

map_name_to_handle = {
    'bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3',
    'bert_en_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_cased_L-12_H-768_A-12/3',
    'bert_multi_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/3',
    'small_bert/bert_en_uncased_L-2_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-2_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-2_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-2_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-4_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-4_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-4_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-4_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-6_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-6_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-6_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-6_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-8_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-8_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-8_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-8_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-10_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-10_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-10_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-10_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-12_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-12_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-12_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-768_A-12/1',
    'albert_en_base':
        'https://tfhub.dev/tensorflow/albert_en_base/2',
    'electra_small':
        'https://tfhub.dev/google/electra_small/2',
    'electra_base':
        'https://tfhub.dev/google/electra_base/2',
    'experts_pubmed':
        'https://tfhub.dev/google/experts/bert/pubmed/2',
    'experts_wiki_books':
        'https://tfhub.dev/google/experts/bert/wiki_books/2',
    'talking-heads_base':
        'https://tfhub.dev/tensorflow/talkheads_ggelu_bert_en_base/1',
}

map_model_to_preprocess = {
    'bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'bert_en_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_cased_preprocess/3',
    'small_bert/bert_en_uncased_L-2_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-2_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-2_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-2_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-4_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-4_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-4_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-4_H-768_A-12':
```

```
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-6_H-128_A-2':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-6_H-256_A-4':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-6_H-512_A-8':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-6_H-768_A-12':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-8_H-128_A-2':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-8_H-256_A-4':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-8_H-512_A-8':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-8_H-768_A-12':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-10_H-128_A-2':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-10_H-256_A-4':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-10_H-512_A-8':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-10_H-768_A-12':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-12_H-128_A-2':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-12_H-256_A-4':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-12_H-512_A-8':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'small_bert/bert_en_uncased_L-12_H-768_A-12':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'bert_multi_cased_L-12_H-768_A-12':
    'https://tfhub.dev/tensorflow/bert_multi_cased_preprocess/3',
'albert_en_base':
    'https://tfhub.dev/tensorflow/albert_en_preprocess/3',
'electra_small':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'electra_base':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'experts_pubmed':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'experts_wiki_books':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
'talking-heads_base':
    'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
}

tfhub_handle_encoder = map_name_to_handle[bert_model_name]
tfhub_handle_preprocess = map_model_to_preprocess[bert_model_name]

print(f'BERT model selected           : {tfhub_handle_encoder}')
print(f'Preprocess model auto-selected: {tfhub_handle_preprocess}')
```

```
BERT model selected           : https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-768_A-12/1 (https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-768_A-12/1)
Preprocess model auto-selected: https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3 (https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3)
```

0.4.1 Defining Model

In [ ]:
```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input
```

In [ ]:
```
# Defining the layers
preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
bert_encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=False, name='BERT_encoder')

# Create the base model from the pre-trained model InceptionV3
inceptionv3 = InceptionV3(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5)
87916544/87910968 [==============================] - 1s 0us/step
87924736/87910968 [==============================] - 1s 0us/step
```

In [ ]:
```
inceptionv3.trainable = False
```

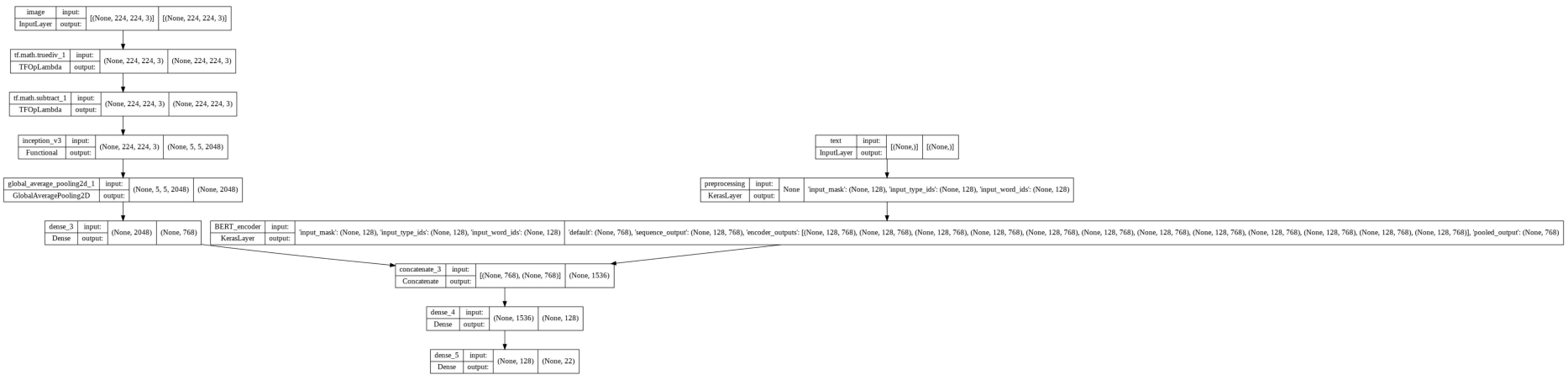
In [ ]:
```
# Defining the model graph

# text input
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
encoder_inputs = preprocessing_layer(text_input)
encoder_outputs = bert_encoder(encoder_inputs)
text_pooled_output = encoder_outputs['pooled_output']

# image input
image_input = Input(shape=(224, 224, 3), name='image')
image = preprocess_input(image_input)
image_inception = inceptionv3(image, training=False)
image_inception = tf.keras.layers.GlobalAveragePooling2D()(image_inception)
image_inception = tf.keras.layers.Dense(768, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2(0

# fusion
concatenated = tf.keras.layers.Concatenate()([text_pooled_output, image_inception])

# custom head
x = tf.keras.layers.Dense(128, kernel_regularizer=tf.keras.regularizers.l2(0.001))(concatenated)
outputs = tf.keras.layers.Dense(22, activation="sigmoid")(x)

model_inc_bert_768 = keras.Model(inputs = [image_input, text_input], outputs=[outputs], name="model_inc_bert_768")
model_inc_bert_768.summary()
```

```
Model: "model_inc_bert_768"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
image (InputLayer)              [(None, 224, 224, 3  0           []
                                )]

tf.math.truediv_1 (TFOpLambda)  (None, 224, 224, 3)  0          ['image[0][0]']

tf.math.subtract_1 (TFOpLambda  (None, 224, 224, 3)  0          ['tf.math.truediv_1[0][0]']
)

text (InputLayer)               [(None,)]            0          []

inception_v3 (Functional)       (None, 5, 5, 2048)   21802784   ['tf.math.subtract_1[0][0]']

preprocessing (KerasLayer)      {'input_mask': (Non  0          ['text[0][0]']
                                e, 128),
                                 'input_type_ids':
                                (None, 128),
                                 'input_word_ids':
                                (None, 128)}

global_average_pooling2d_1 (Gl  (None, 2048)         0          ['inception_v3[1][0]']
obalAveragePooling2D)

BERT_encoder (KerasLayer)       {'default': (None,   109482241  ['preprocessing[1][0]',
                                768),                            'preprocessing[1][1]',
                                 'sequence_output':              'preprocessing[1][2]']
                                 (None, 128, 768),
                                 'encoder_outputs':
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768)],
                                 'pooled_output': (
                                None, 768)}

 dense_3 (Dense)                (None, 768)          1573632     ['global_average_pooling2d_1[0][0
                                                                 ]']

 concatenate_3 (Concatenate)    (None, 1536)         0           ['BERT_encoder[1][13]',
                                                                  'dense_3[0][0]']

 dense_4 (Dense)                (None, 128)          196736      ['concatenate_3[0][0]']

 dense_5 (Dense)                (None, 22)           2838        ['dense_4[0][0]']

==================================================================================================
Total params: 133,058,231
Trainable params: 1,773,206
Non-trainable params: 131,285,025
__________________________________________________________________________________________________
```

0.4.2 Plotting the model

In [ ]:
```
plot_model(model_inc_bert_768, show_shapes=True)
```

Out[73]:



0.4.3 Training the model

In [ ]:
```
model_name = 'model_inc_bert_768'
```

In [ ]:
```
!rm -rf /content/tensorboard_logs/models/model_inc_bert_768
```

In [ ]:
```
epochs = 20
num_train_steps = STEPS_PER_EPOCH * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
adamw_optimizer = optimization.create_optimizer(init_lr=init_lr,
                                                num_train_steps=num_train_steps,
                                                num_warmup_steps=num_warmup_steps,
                                                optimizer_type='adamw')

adam_optimizer = tf.keras.optimizers.Adam(0.0001)
```

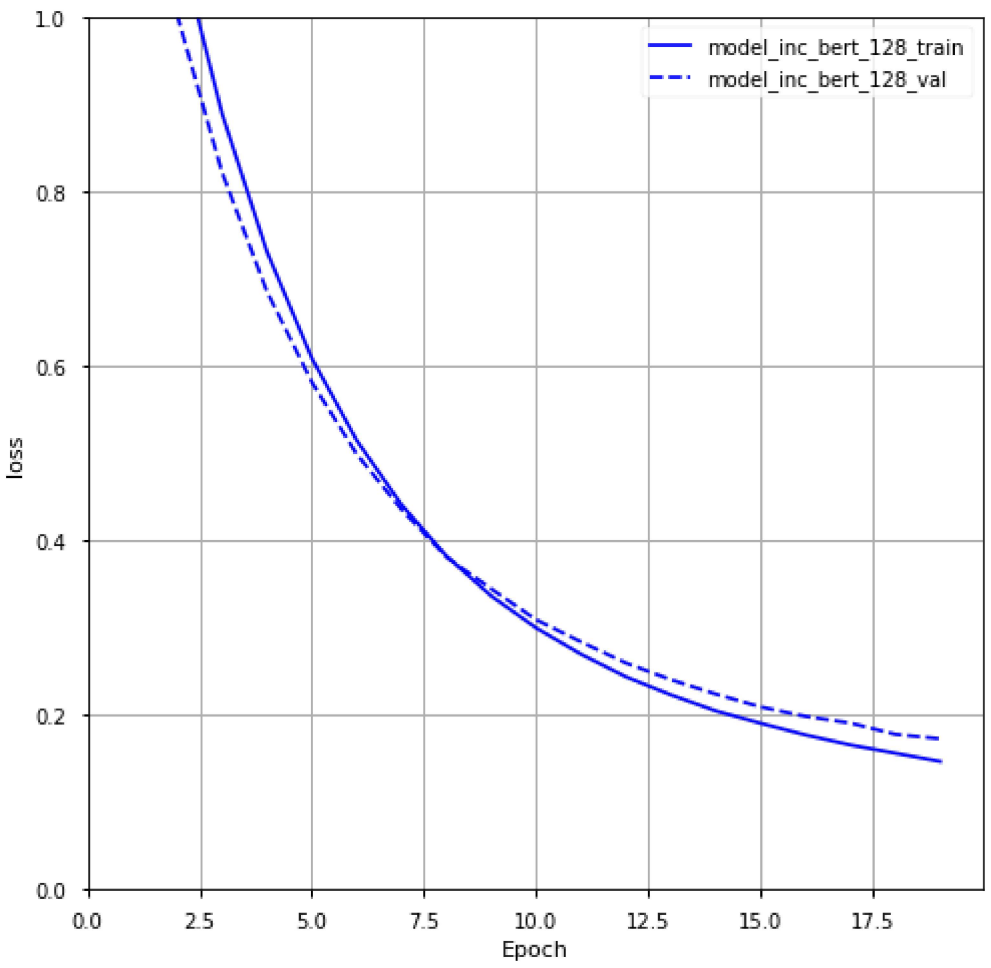
In [ ]:
```
m_histories[model_name], models[model_name] = compile_and_fit(
                model_inc_bert_768,
                model_name,
                train_gen=training_generator,
                val_gen=validation_generator,
                max_epochs=20,
                optimizer=adam_optimizer,
                weights=None

            )
```

```
Epoch 1/20
85/85 [==============================] - ETA: 0s - loss: 1.6990 - binary_crossentropy: 0.3720 - precision: 0.4432 - re
call: 0.3681 - auc: 0.7954 - prc: 0.3596 - CategoricalAccuracy: 0.1603 - F1_macro: 0.0574 - F1_micro: 0.3123
Epoch 1: val_F1_macro improved from -inf to 0.04718, saving model to /content/tensorboard_logs/models/model_inc_bert_1
28/model_inc_bert_128

WARNING:absl:Found untraced functions such as restored_function_body, restored_function_body, restored_function_body,
restored_function_body, restored_function_body while saving (showing 5 of 364). These functions will not be directly c
allable after loading.

85/85 [==============================] - 132s 1s/step - loss: 1.6990 - binary_crossentropy: 0.3720 - precision: 0.4432
- recall: 0.3681 - auc: 0.7954 - prc: 0.3596 - CategoricalAccuracy: 0.1603 - F1_macro: 0.0574 - F1_micro: 0.3123 - val
_loss: 1.5287 - val_binary_crossentropy: 0.3779 - val_precision: 0.6216 - val_recall: 0.2893 - val_auc: 0.7784 - val_p
rc: 0.4042 - val_CategoricalAccuracy: 0.1607 - val_F1_macro: 0.0472 - val_F1_micro: 0.3814
Epoch 2/20
85/85 [==============================] - ETA: 0s - loss: 1.3582 - binary_crossentropy: 0.3517 - precision: 0.6003 - re
call: 0.4021 - auc: 0.8437 - prc: 0.4928 - CategoricalAccuracy: 0.2044 - F1_macro: 0.0604 - F1_micro: 0.3724
Epoch 2: val_F1_macro improved from 0.04718 to 0.06435, saving model to /content/tensorboard_logs/models/model_inc_ber
t_128/model_inc_bert_128
```

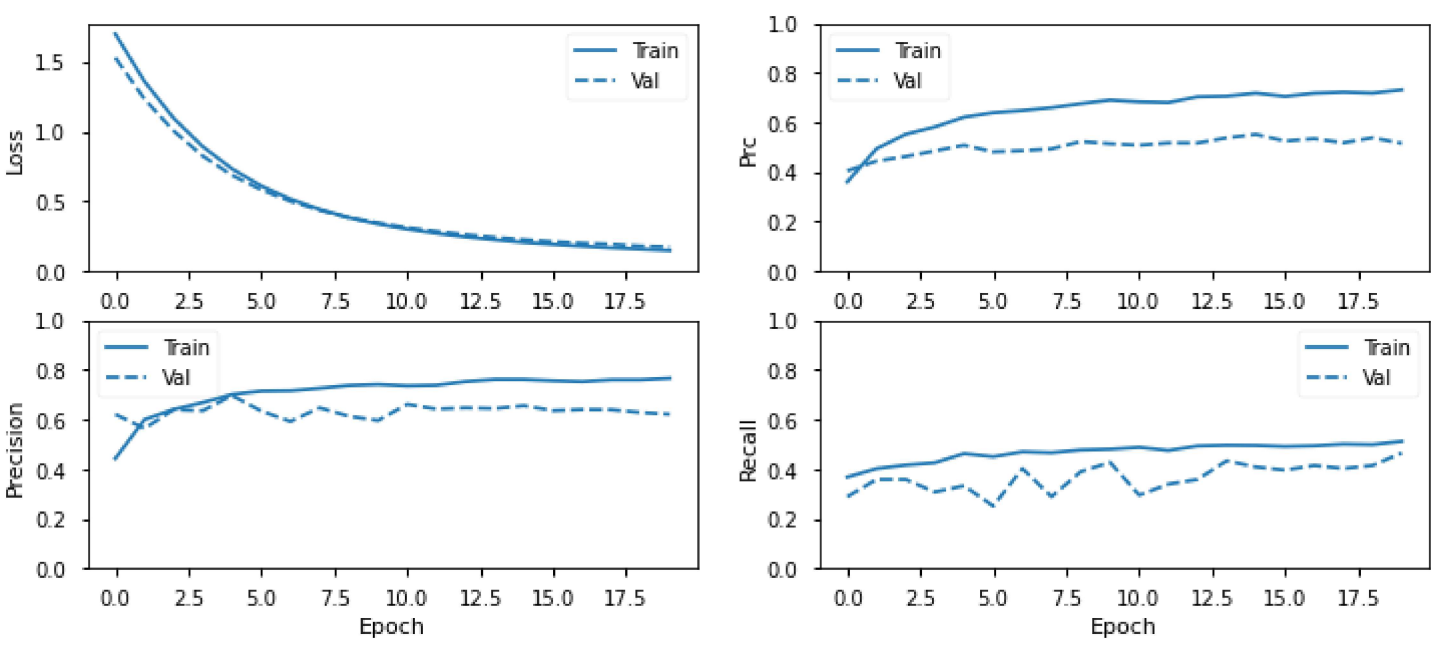
In [ ]:
```
download_files(m_histories, model_name)
```

0.4.4 Plot Metrics

In [ ]:
```
plotter(m_histories, metric="loss", ylim=[0, 1.0])
```



In [ ]:
```
plot_metrics(m_histories[model_name])
```



0.4.5 Evaluation on Validation Set

In [ ]:
```
# Applying a threshold and converting the probability values in the
# output to binary boolean values (1 or 0)
threshold = 0.5
```

0.4.5.1 Storing the predictions to dataframe for plotting

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_inc_bert_768/model_inc_bert_768')
```

In [ ]:
```
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=1,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    shuffle=False
                )
dev_pred = loaded_model.predict(validation_generator, verbose=1)
dev_prediction = (dev_pred > threshold).astype('int')
dev_prediction_df = prediction_df(dev_df, dev_prediction, mlb)
dev_prediction_df.head()
```

```
63/63 [==============================] - 7s 67ms/step
```

Out[84]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Smears, Doubt]	[Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	[Smears, Loaded Language, Name calling/Labeling]	[Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
2	[Smears, Transfer]	[Name calling/Labeling]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
3	[]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
4	[Smears, Exaggeration/Minimisation, Loaded Lan...	[Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

In [ ]:
```
print("F1 macro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
print("F1 micro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
```

```
F1 macro on Dev Set:  0.09
F1 micro on Dev Set:  0.42
```

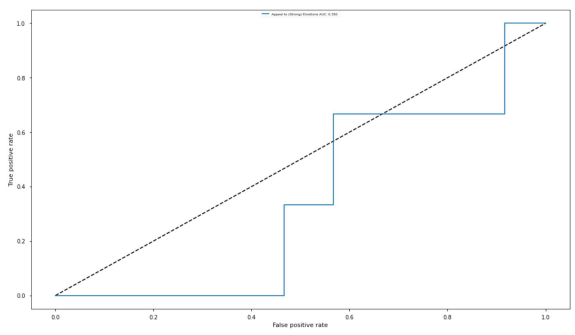
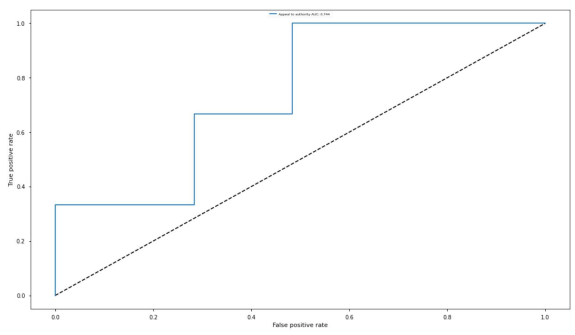
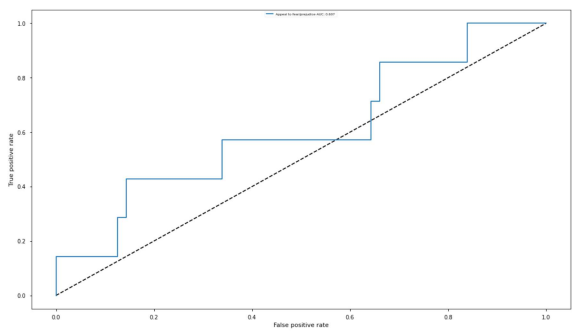
In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(dev_prediction_df['actual_labels_mh'].values.tolist(), de
perf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
perf_table.index = mlb.classes_
perf_table
```

Out[86]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	3
Appeal to authority	0.000000	0.000000	0.000000	3
Appeal to fear/prejudice	0.000000	0.000000	0.000000	7
Bandwagon	0.000000	0.000000	0.000000	2
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	0
Causal Oversimplification	0.000000	0.000000	0.000000	1
Doubt	0.000000	0.000000	0.000000	9
Exaggeration/Minimisation	0.000000	0.000000	0.000000	8
Flag-waving	1.000000	0.142857	0.250000	7
Glittering generalities (Virtue)	0.000000	0.000000	0.000000	4
Loaded Language	0.647059	0.343750	0.448980	32
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	5
Name calling/Labeling	0.541667	0.433333	0.481481	30
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	0
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	0
Reductio ad hitlerum	0.000000	0.000000	0.000000	1
Repetition	0.000000	0.000000	0.000000	3
Slogans	0.000000	0.000000	0.000000	3
Smears	0.769231	0.638298	0.697674	47
Thought-terminating cliché	0.000000	0.000000	0.000000	1
Transfer	0.000000	0.000000	0.000000	11
Whataboutism	0.000000	0.000000	0.000000	6

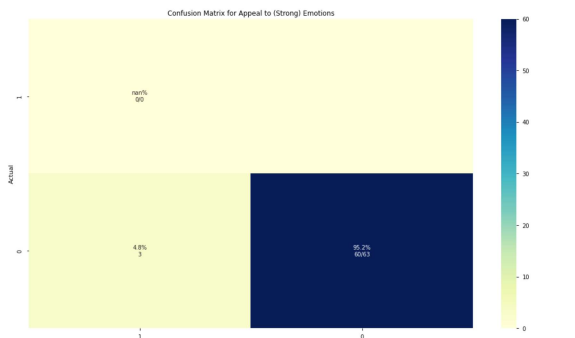
In [ ]:
```
val_preds = np.array([]).reshape((0,22))
val_true = np.array([]).reshape((0,22))
for i in range(validation_generator.__len__()):
    x,y = validation_generator.__getitem__(i)
    val_true = np.append(val_true,y,axis=0)
    val_preds = np.append(val_preds,models[model_name](x).numpy().reshape((1,22)),axis=0)
```

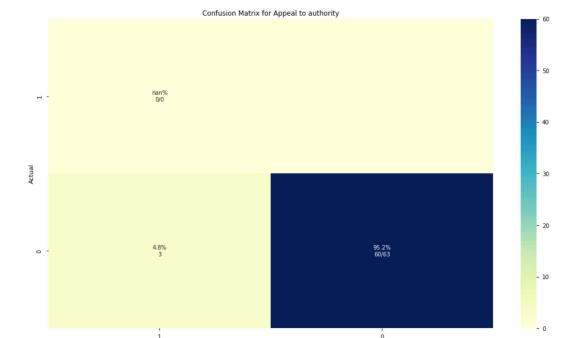
In [ ]:
```
plot_roc_curve(val_true, val_preds, mlb.classes_[0:3])
```

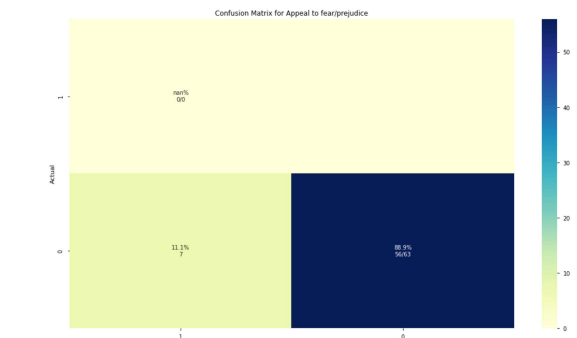




In [ ]:
```
plot_labelwise_confusion_matrix_(val_true, val_preds, mlb.classes_[0:3])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: invalid value encountered in true_divid
e
  # Remove the CWD from sys.path while we load stuff.
```







0.5 Model 2: BERT + ResNet -- Concatenation

💡 In this model, the text features are extracted using a BERT Transformer model and Image features are extracted using ResNet50 model. Concatenation is used as the late fusion technique.

0.5.1 Defining Model

In [ ]:
```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_input_resnet
```

In [ ]:
```
# Defining the layers
preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
bert_encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=False, name='BERT_encoder')

# Create the base model from the pre-trained model Resnet50
resnet50 = ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
)
```

In [ ]:
```
resnet50.trainable = False
```

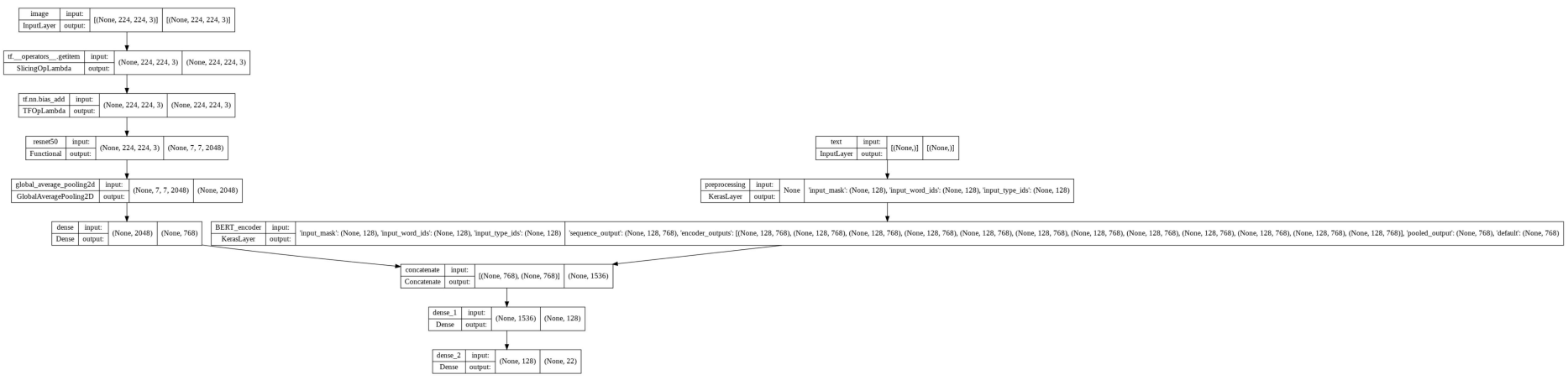
In [ ]:
```
# Defining the model graph

# text input
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
encoder_inputs = preprocessing_layer(text_input)
encoder_outputs = bert_encoder(encoder_inputs)
text_pooled_output = encoder_outputs['pooled_output']

# image input
image_input = Input(shape=(224, 224, 3), name='image')
image = preprocess_input_resnet(image_input)
image_features = resnet50(image, training=False)
image_features = tf.keras.layers.GlobalAveragePooling2D()(image_features)
image_features = tf.keras.layers.Dense(768, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2(0.

# fusion
concatenated = tf.keras.layers.Concatenate()([text_pooled_output, image_features])

# custom head
x = tf.keras.layers.Dense(128, kernel_regularizer=tf.keras.regularizers.l2(0.001))(concatenated)
outputs = tf.keras.layers.Dense(22, activation="sigmoid")(x)

model_res_bert_concat = keras.Model(inputs = [image_input, text_input], outputs=[outputs], name="model_res_bert_concat")
model_res_bert_concat.summary()
```

```
Model: "model_res_bert_concat"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
image (InputLayer)              [(None, 224, 224, 3  0           []
                                )]

tf.__operators__.getitem (Slic  (None, 224, 224, 3)  0          ['image[0][0]']
ingOpLambda)

tf.nn.bias_add (TFOpLambda)     (None, 224, 224, 3)  0           ['tf.__operators__.getitem[0][0]'
                                                                 ]

text (InputLayer)               [(None,)]            0           []

resnet50 (Functional)           (None, 7, 7, 2048)   23587712    ['tf.nn.bias_add[0][0]']

preprocessing (KerasLayer)      {'input_mask': (Non  0           ['text[0][0]']
                                e, 128),
                                 'input_word_ids':
                                (None, 128),
                                 'input_type_ids':
                                (None, 128)}

global_average_pooling2d (Glob  (None, 2048)         0           ['resnet50[0][0]']
alAveragePooling2D)

BERT_encoder (KerasLayer)       {'sequence_output':  109482241   ['preprocessing[0][0]',
                                 (None, 128, 768),               'preprocessing[0][1]',
                                 'encoder_outputs':              'preprocessing[0][2]']
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768)],
                                 'pooled_output': (
                                None, 768),
                                 'default': (None,
                                768)}

 dense (Dense)                  (None, 768)          1573632     ['global_average_pooling2d[0][0]'
                                                                 ]

 concatenate (Concatenate)      (None, 1536)         0           ['BERT_encoder[0][13]',
                                                                  'dense[0][0]']

 dense_1 (Dense)                (None, 128)          196736      ['concatenate[0][0]']

 dense_2 (Dense)                (None, 22)           2838        ['dense_1[0][0]']

==================================================================================================
Total params: 134,843,159
Trainable params: 1,773,206
Non-trainable params: 133,069,953
__________________________________________________________________________________________________
```

0.5.2 Plotting the model

In [ ]:
```
plot_model(model_res_bert_concat, show_shapes=True)
```

Out[71]:



0.5.3 Training the model

In [ ]:
```
model_name = 'model_res_bert_concat'
```

In [ ]:
```
!rm -rf /content/tensorboard_logs/models/model_res_bert_concat
```

In [ ]:
```
epochs = 20
num_train_steps = STEPS_PER_EPOCH * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
adamw_optimizer = optimization.create_optimizer(init_lr=init_lr,
                                                num_train_steps=num_train_steps,
                                                num_warmup_steps=num_warmup_steps,
                                                optimizer_type='adamw')

adam_optimizer = tf.keras.optimizers.Adam(0.0001)
```

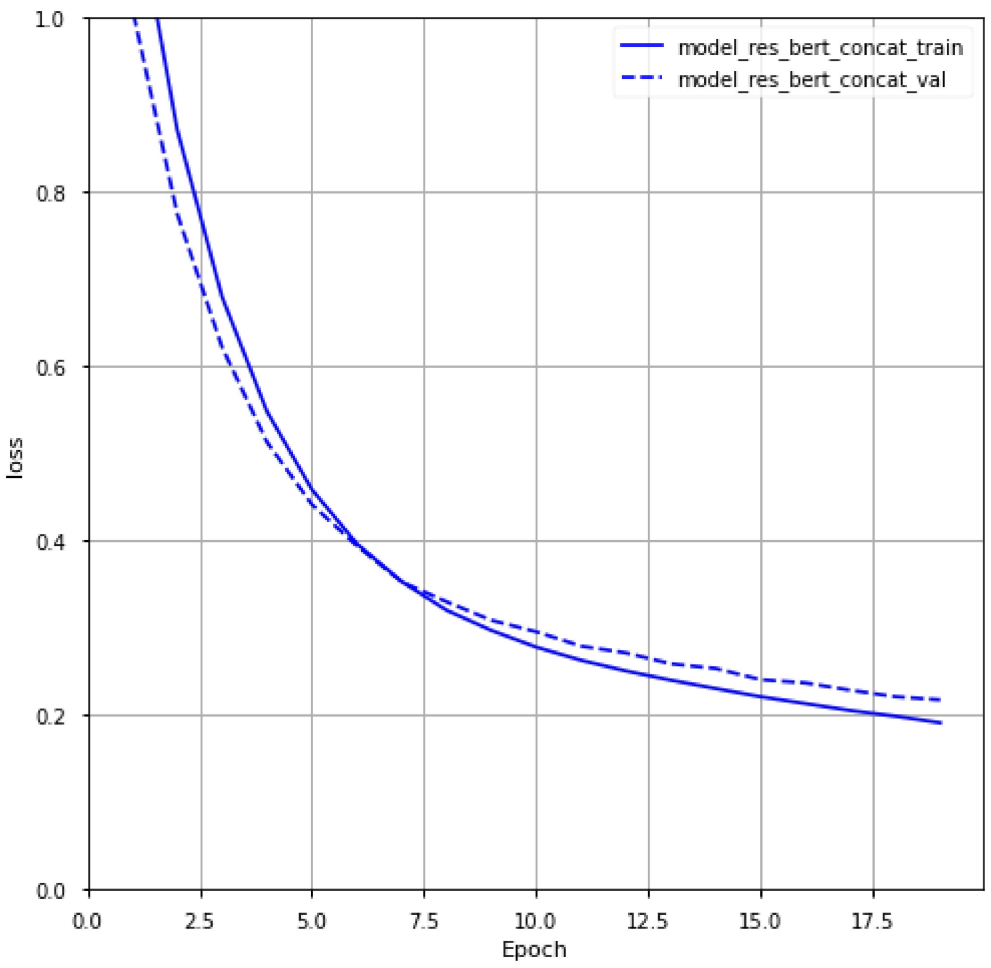
In [ ]:
```
m_histories[model_name], models[model_name] = compile_and_fit(
                model_res_bert_concat,
                model_name,
                train_gen=training_generator,
                val_gen=validation_generator,
                max_epochs=20,
                optimizer=adam_optimizer,
                weights=None
            )
```

```
Epoch 1/20
85/85 [==============================] - ETA: 0s - loss: 1.5969 - binary_crossentropy: 0.3610 - precision: 0.4813 - re
call: 0.3656 - auc: 0.7943 - prc: 0.3687 - CategoricalAccuracy: 0.1706 - F1_macro: 0.0574 - F1_micro: 0.3126
Epoch 1: val_F1_macro improved from -inf to 0.05301, saving model to /content/tensorboard_logs/models/model_res_bert_c
oncat/model_res_bert_concat

WARNING:absl:Found untraced functions such as restored_function_body, restored_function_body, restored_function_body,
restored_function_body, restored_function_body while saving (showing 5 of 364). These functions will not be directly c
allable after loading.

85/85 [==============================] - 96s 932ms/step - loss: 1.5969 - binary_crossentropy: 0.3610 - precision: 0.48
13 - recall: 0.3656 - auc: 0.7943 - prc: 0.3687 - CategoricalAccuracy: 0.1706 - F1_macro: 0.0574 - F1_micro: 0.3126 -
val_loss: 1.3514 - val_binary_crossentropy: 0.3673 - val_precision: 0.5800 - val_recall: 0.3648 - val_auc: 0.8058 - va
l_prc: 0.4186 - val_CategoricalAccuracy: 0.1071 - val_F1_macro: 0.0530 - val_F1_micro: 0.3070
Epoch 2/20
85/85 [==============================] - ETA: 0s - loss: 1.1587 - binary_crossentropy: 0.3526 - precision: 0.5794 - re
call: 0.3847 - auc: 0.8339 - prc: 0.4729 - CategoricalAccuracy: 0.1956 - F1_macro: 0.0579 - F1_micro: 0.3541
Epoch 2: val_F1_macro did not improve from 0.05301
85/85 [==============================] - 14s 164ms/step - loss: 1.1587 - binary_crossentropy: 0.3526 - precision: 0.57
94 - recall: 0.3847 - auc: 0.8339 - prc: 0.4729 - CategoricalAccuracy: 0.1956 - F1_macro: 0.0579 - F1_micro: 0.3541 -
```

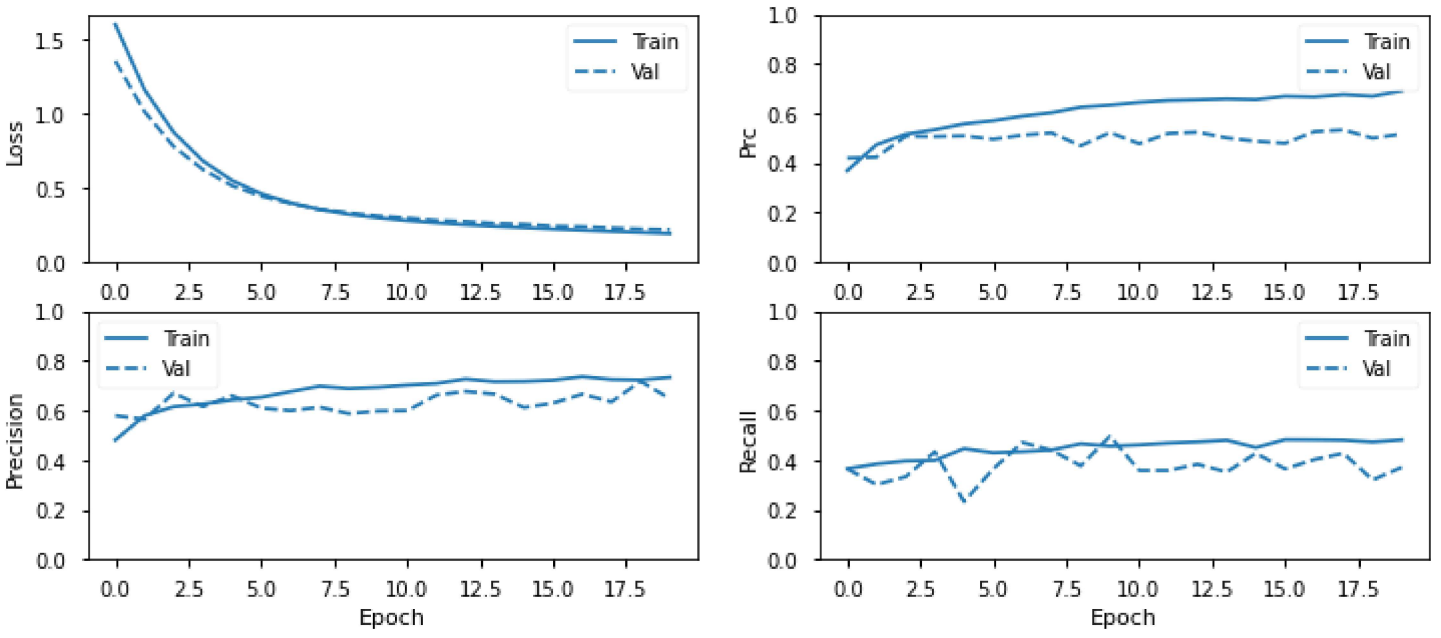
In [ ]:
```
download_files(m_histories, model_name)
```

0.5.4 Plot Metrics

In [ ]:
```
plotter(m_histories, metric="loss", ylim=[0, 1.0])
```



In [ ]:
```
plot_metrics(m_histories[model_name])
```



0.5.5 Evaluation on Validation Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert_concat/model_res_bert_concat')
```

In [ ]:
```
# Applying a threshold and converting the probability values in the
# output to binary boolean values (1 or 0)
threshold = 0.5
```

0.5.5.1 Storing the predictions to dataframe for plotting

In [ ]:
```
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=1,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    shuffle=False
                )
dev_pred = models[model_name].predict(validation_generator, verbose=1)
dev_prediction = (dev_pred > threshold).astype('int')
dev_prediction_df = prediction_df(dev_df, dev_prediction, mlb)
dev_prediction_df.head()
```

```
63/63 [==============================] - 2s 36ms/step
```

Out[82]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Smears, Doubt]	[Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	[Smears, Loaded Language, Name calling/Labeling]	[Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2	[Smears, Transfer]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
3	[]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
4	[Smears, Exaggeration/Minimisation, Loaded Lan...	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...

In [ ]:
```
print("F1 macro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
print("F1 micro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
```

```
F1 macro on Dev Set:  0.133
F1 micro on Dev Set:  0.493
```

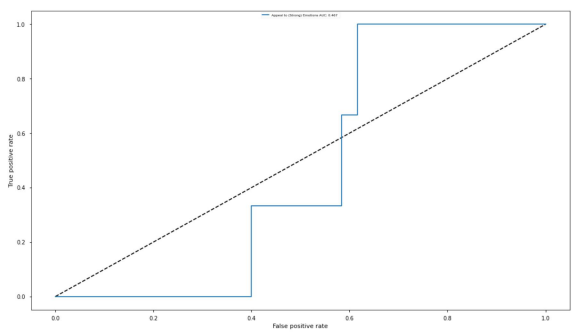
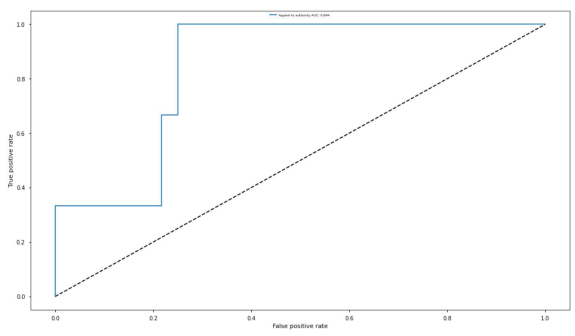
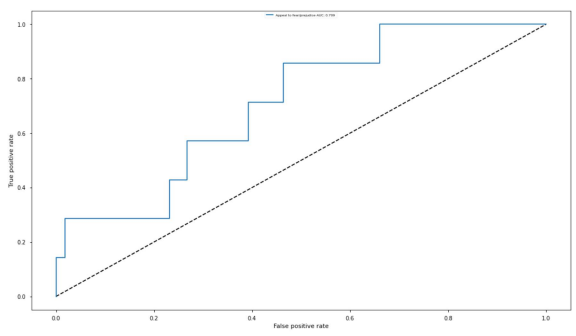
In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(dev_prediction_df['actual_labels_mh'].values.tolist(), de
perf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
perf_table.index = mlb.classes_
perf_table
```

Out[84]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	3
Appeal to authority	0.000000	0.000000	0.000000	3
Appeal to fear/prejudice	0.000000	0.000000	0.000000	7
Bandwagon	0.000000	0.000000	0.000000	2
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	0
Causal Oversimplification	0.000000	0.000000	0.000000	1
Doubt	0.000000	0.000000	0.000000	9
Exaggeration/Minimisation	0.000000	0.000000	0.000000	8
Flag-waving	1.000000	0.142857	0.250000	7
Glittering generalities (Virtue)	0.300000	0.750000	0.428571	4
Loaded Language	0.653846	0.531250	0.586207	32
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	5
Name calling/Labeling	0.583333	0.233333	0.333333	30
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	0
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	0
Reductio ad hitlerum	0.000000	0.000000	0.000000	1
Repetition	0.000000	0.000000	0.000000	3
Slogans	1.000000	0.333333	0.500000	3
Smears	0.763636	0.893617	0.823529	47
Thought-terminating cliché	0.000000	0.000000	0.000000	1
Transfer	0.000000	0.000000	0.000000	11
Whataboutism	0.000000	0.000000	0.000000	6

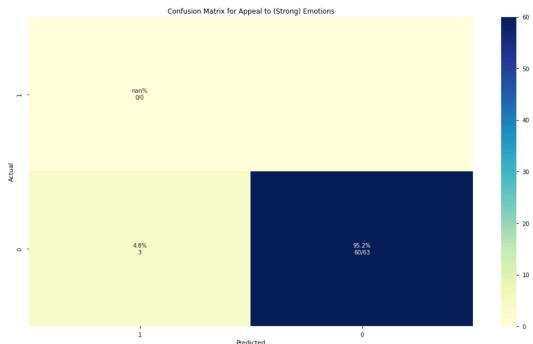
In [ ]:
```
val_preds = np.array([]).reshape((0,22))
val_true = np.array([]).reshape((0,22))
for i in range(validation_generator.__len__()):
    x,y = validation_generator.__getitem__(i)
    val_true = np.append(val_true,y,axis=0)
    val_preds = np.append(val_preds,models[model_name](x).numpy().reshape((1,22)),axis=0)
```

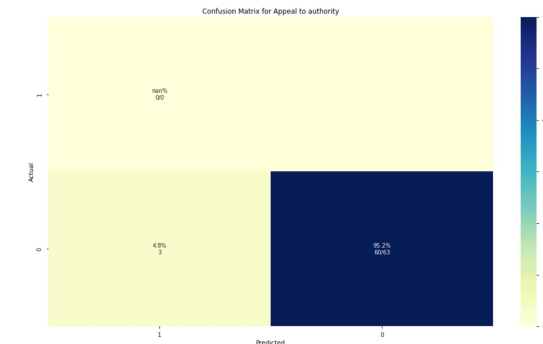
In [ ]:
```
plot_roc_curve(val_true, val_preds, mlb.classes_[0:3])
```

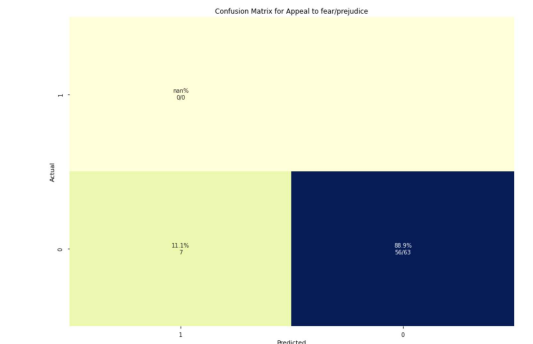




In [ ]:
```
plot_labelwise_confusion_matrix_(val_true, val_preds, mlb.classes_[0:3])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: invalid value encountered in true_divid
e
  # Remove the CWD from sys.path while we load stuff.
```







0.6 Model 3: BERT + ResNet -- Cross-Attention

💡 In this model, the text features are extracted using a BERT Transformer model and Image features are extracted using ResNet model.

- Cross Attention is used in the fusion mechanism.
- Cross-attention allows attention masks from one modality to highlight the extracted features in another modality.
- Here, the text features are used as query to the image values.

0.6.1 Defining Model

In [ ]:
```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_input_resnet
```

In [ ]:
```
# Defining the layers
preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
bert_encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=False, name='BERT_encoder')

# Create the base model from the pre-trained model Resnet50
resnet50 = ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
)
```

In [ ]:
```
resnet50.trainable = False
```

In [ ]:
```
# Defining the model graph

# text input
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
encoder_inputs = preprocessing_layer(text_input)
encoder_outputs = bert_encoder(encoder_inputs)
text_pooled_output = encoder_outputs['pooled_output']

# image input
image_input = Input(shape=(224, 224, 3), name='image')
image = preprocess_input_resnet(image_input)
image_features = resnet50(image, training=False)
image_features = tf.keras.layers.GlobalAveragePooling2D()(image_features)
image_features = tf.keras.layers.Dense(768, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2(0.

# fusion
# Cross-attention.
query_value_attention_seq = keras.layers.Attention(use_scale=True, dropout=0.2)(
    [text_pooled_output, image_features]
)

# Concatenate the projections and pass through the classification layer.
concatenated = keras.layers.Concatenate()([image_features, text_pooled_output])
concatenated = keras.layers.Concatenate()([concatenated, query_value_attention_seq])

# custom head
x = tf.keras.layers.Dense(128, kernel_regularizer=tf.keras.regularizers.l2(0.001))(concatenated)
outputs = tf.keras.layers.Dense(22, activation="sigmoid")(x)

model_res_bert = keras.Model(inputs = [image_input, text_input], outputs=[outputs], name="model_res_bert")
model_res_bert.summary()
```

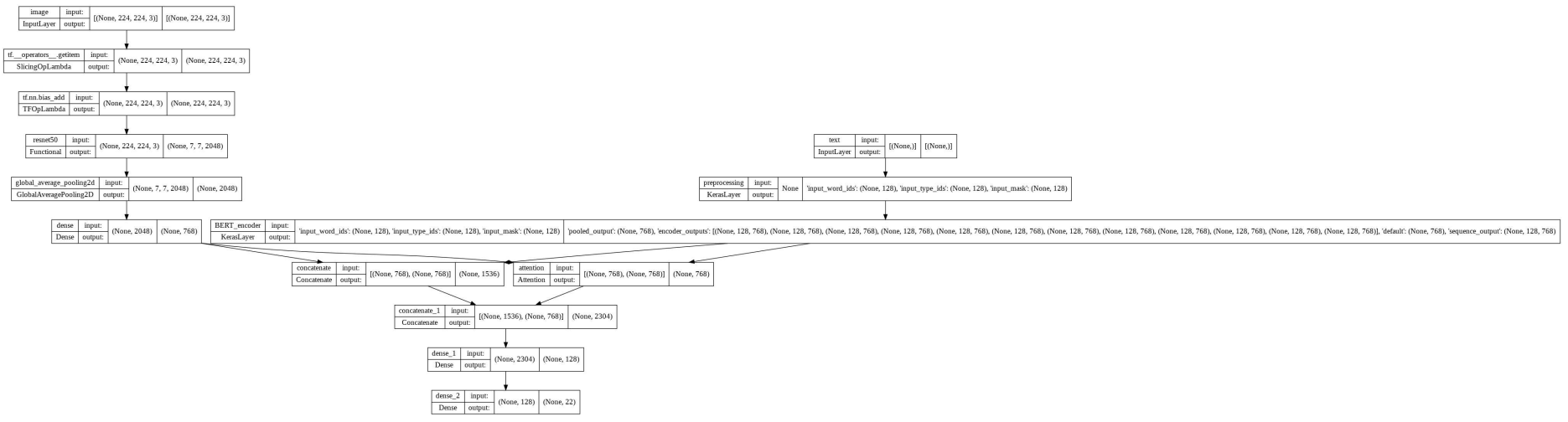
```
Model: "model_res_bert"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 image (InputLayer)             [(None, 224, 224, 3  0           []
                                )]

 tf.__operators__.getitem (Slic  (None, 224, 224, 3)  0          ['image[0][0]']
 ingOpLambda)

 tf.nn.bias_add (TFOpLambda)    (None, 224, 224, 3)  0           ['tf.__operators__.getitem[0][0]'
                                                                 ]

 resnet50 (Functional)          (None, 7, 7, 2048)   23587712    ['tf.nn.bias_add[0][0]']

 text (InputLayer)              [(None,)]            0           []

 global_average_pooling2d (Glob  (None, 2048)        0           ['resnet50[0][0]']
 alAveragePooling2D)

 preprocessing (KerasLayer)     {'input_word_ids':   0           ['text[0][0]']
                                (None, 128),
                                 'input_type_ids':
                                (None, 128),
                                 'input_mask': (Non
                                e, 128)}

 dense (Dense)                  (None, 768)          1573632     ['global_average_pooling2d[0][0]'
                                                                 ]

 BERT_encoder (KerasLayer)      {'pooled_output': (  109482241   ['preprocessing[0][0]',
                                None, 768),                       'preprocessing[0][1]',
                                 'encoder_outputs':               'preprocessing[0][2]']
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768)],
                                 'default': (None,
                                768),
                                 'sequence_output':
                                 (None, 128, 768)}

 concatenate (Concatenate)      (None, 1536)         0           ['dense[0][0]',
                                                                  'BERT_encoder[0][13]']

 attention (Attention)          (None, 768)          1           ['BERT_encoder[0][13]',
                                                                  'dense[0][0]']

 concatenate_1 (Concatenate)    (None, 2304)         0           ['concatenate[0][0]',
                                                                  'attention[0][0]']
```

```
 dense_1 (Dense)                (None, 128)          295040      ['concatenate_1[0][0]']

 dense_2 (Dense)                (None, 22)           2838        ['dense_1[0][0]']

==================================================================================================
Total params: 134,941,464
Trainable params: 1,871,511
Non-trainable params: 133,069,953
__________________________________________________________________________________________________
```

0.6.2 Plotting the model

In [ ]:
```
plot_model(model_res_bert, show_shapes=True)
```

Out[71]:



0.6.3 Training the model

In [ ]:
```
model_name = 'model_res_bert'
```

In [ ]:
```
!rm -rf /content/tensorboard_logs/models/model_res_bert
```

In [ ]:
```
epochs = 20
num_train_steps = STEPS_PER_EPOCH * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
adamw_optimizer = optimization.create_optimizer(init_lr=init_lr,
                                                num_train_steps=num_train_steps,
                                                num_warmup_steps=num_warmup_steps,
                                                optimizer_type='adamw')

adam_optimizer = tf.keras.optimizers.Adam(0.0001)
```

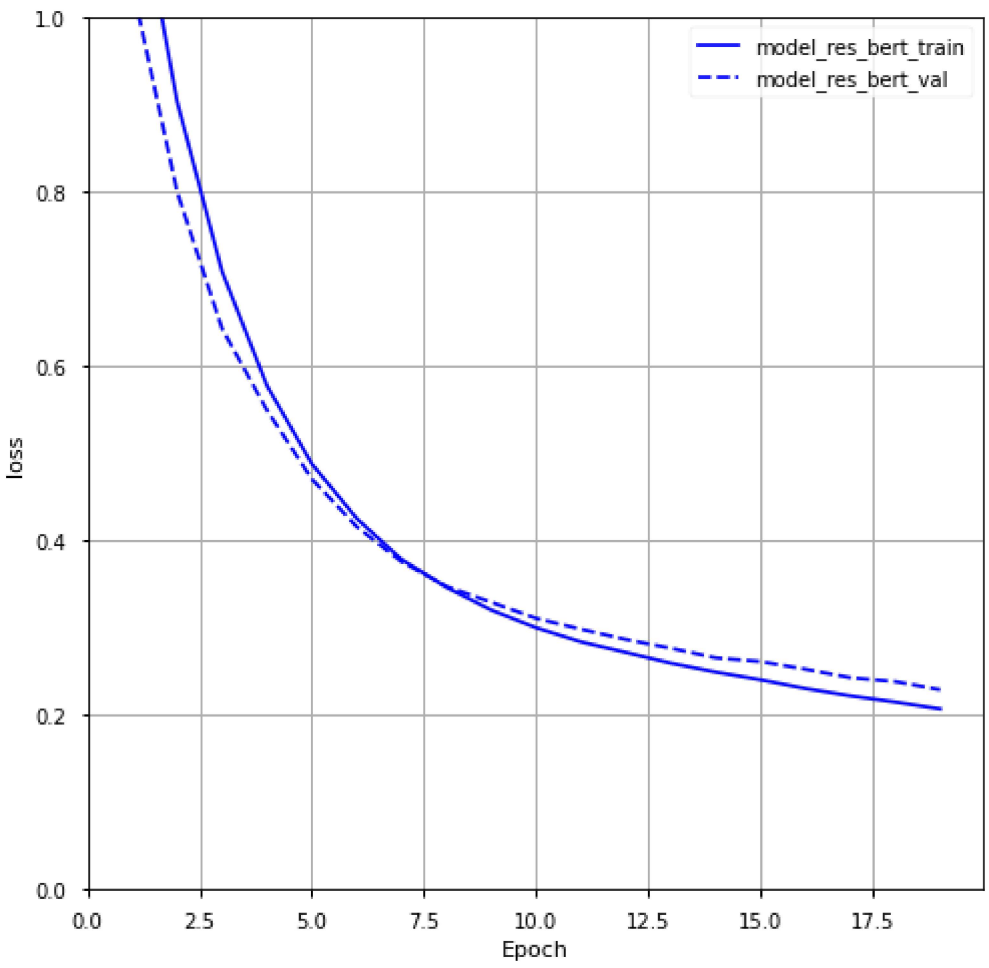
In [ ]:
```
m_histories[model_name], models[model_name] = compile_and_fit(
                model_res_bert,
                model_name,
                train_gen=training_generator,
                val_gen=validation_generator,
                max_epochs=20,
                optimizer=adam_optimizer,
                weights=None
            )
```

```
Epoch 1/20
85/85 [==============================] - ETA: 0s - loss: 1.6240 - binary_crossentropy: 0.3679 - precision: 0.4327 - re
call: 0.3382 - auc: 0.7891 - prc: 0.3189 - CategoricalAccuracy: 0.1471 - F1_macro: 0.0523 - F1_micro: 0.2974
Epoch 1: val_F1_macro improved from -inf to 0.05547, saving model to /content/tensorboard_logs/models/model_res_bert/m
odel_res_bert

WARNING:absl:Found untraced functions such as restored_function_body, restored_function_body, restored_function_body,
restored_function_body, restored_function_body while saving (showing 5 of 364). These functions will not be directly c
allable after loading.

85/85 [==============================] - 99s 956ms/step - loss: 1.6240 - binary_crossentropy: 0.3679 - precision: 0.43
27 - recall: 0.3382 - auc: 0.7891 - prc: 0.3189 - CategoricalAccuracy: 0.1471 - F1_macro: 0.0523 - F1_micro: 0.2974 -
val_loss: 1.3865 - val_binary_crossentropy: 0.3886 - val_precision: 0.5051 - val_recall: 0.3145 - val_auc: 0.7972 - va
l_prc: 0.3827 - val_CategoricalAccuracy: 0.1964 - val_F1_macro: 0.0555 - val_F1_micro: 0.3256
Epoch 2/20
85/85 [==============================] - ETA: 0s - loss: 1.1865 - binary_crossentropy: 0.3503 - precision: 0.5684 - re
call: 0.3782 - auc: 0.8289 - prc: 0.4668 - CategoricalAccuracy: 0.1706 - F1_macro: 0.0620 - F1_micro: 0.3419
Epoch 2: val_F1_macro improved from 0.05547 to 0.06300, saving model to /content/tensorboard_logs/models/model_res_ber
t/model_res_bert
```

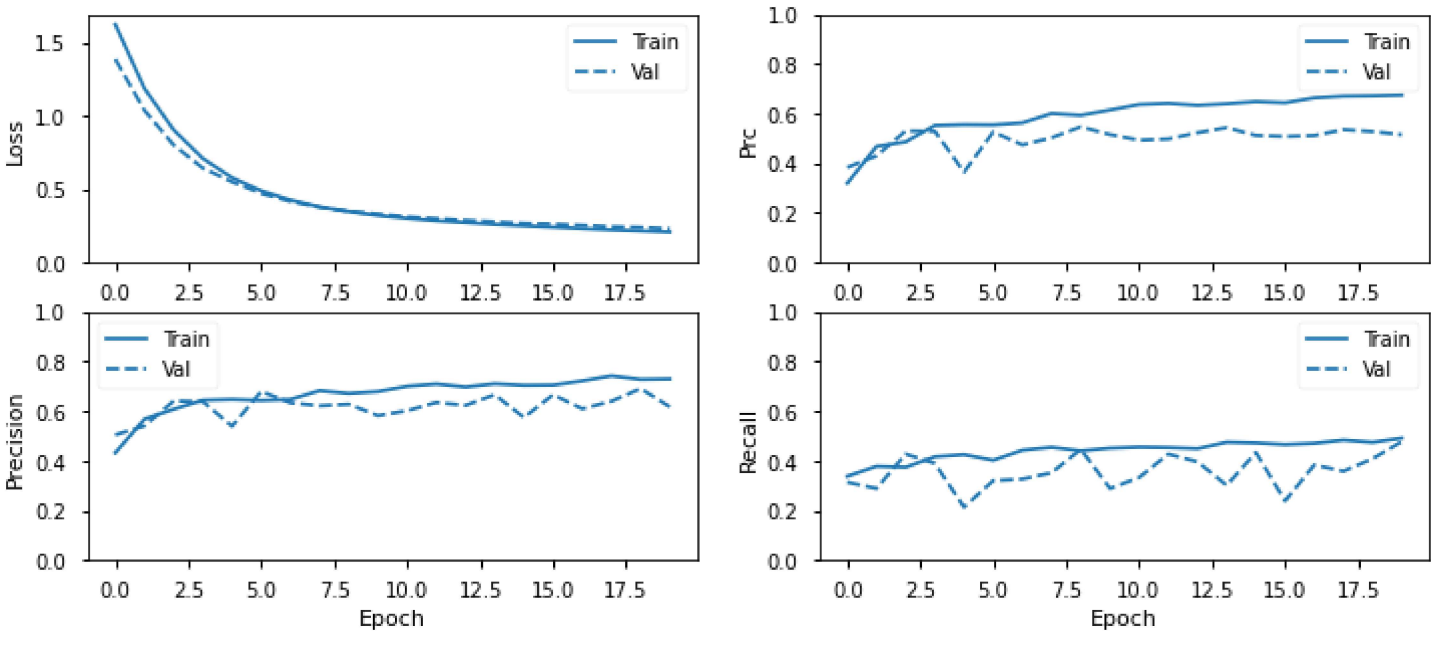
In [ ]:
```
download_files(m_histories, model_name)
```

0.6.4 Plot Metrics

In [ ]:
```
plotter(m_histories, metric="loss", ylim=[0, 1.0])
```



In [ ]:
```
plot_metrics(m_histories[model_name])
```



0.6.5 Evaluation on Validation Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert/model_res_bert')
```

In [ ]:
```
# Applying a threshold and converting the probability values in the
# output to binary boolean values (1 or 0)
threshold = 0.5
```

0.6.5.1 Storing the predictions to dataframe for plotting

In [ ]:
```
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=1,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    shuffle=False
                )
dev_pred = models[model_name].predict(validation_generator, verbose=1)
dev_prediction = (dev_pred > threshold).astype('int')
dev_prediction_df = prediction_df(dev_df, dev_prediction, mlb)
dev_prediction_df.head()
```

```
63/63 [==============================] - 4s 34ms/step
```

Out[81]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Smears, Doubt]	[Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	[Smears, Loaded Language, Name calling/Labeling]	[Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
2	[Smears, Transfer]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
3	[]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
4	[Smears, Exaggeration/Minimisation, Loaded Lan...	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...

In [ ]:
```
print("F1 macro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
print("F1 micro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
```

```
F1 macro on Dev Set:  0.11
F1 micro on Dev Set:  0.562
```

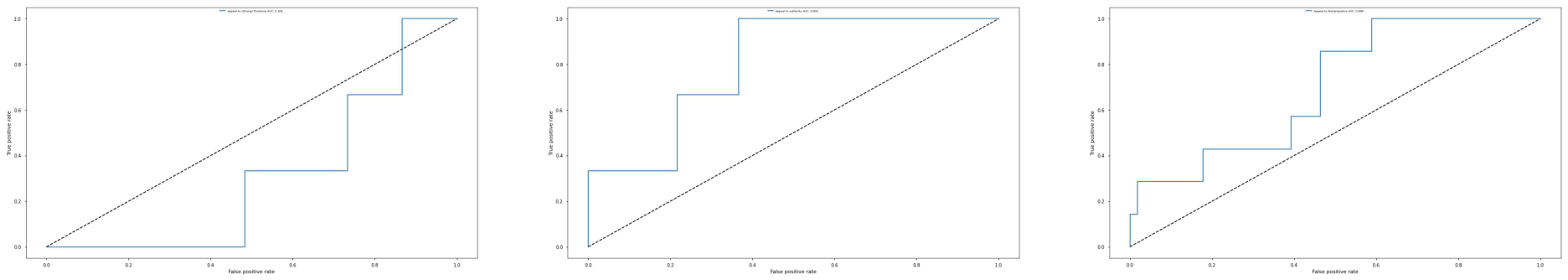
In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(dev_prediction_df['actual_labels_mh'].values.tolist(), de
perf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
perf_table.index = mlb.classes_
perf_table
```

Out[83]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	3
Appeal to authority	0.000000	0.000000	0.000000	3
Appeal to fear/prejudice	1.000000	0.142857	0.250000	7
Bandwagon	0.000000	0.000000	0.000000	2
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	0
Causal Oversimplification	0.000000	0.000000	0.000000	1
Doubt	0.000000	0.000000	0.000000	9
Exaggeration/Minimisation	0.000000	0.000000	0.000000	8
Flag-waving	0.000000	0.000000	0.000000	7
Glittering generalities (Virtue)	0.000000	0.000000	0.000000	4
Loaded Language	0.613636	0.843750	0.710526	32
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	5
Name calling/Labeling	0.558140	0.800000	0.657534	30
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	0
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	0
Reductio ad hitlerum	0.000000	0.000000	0.000000	1
Repetition	0.000000	0.000000	0.000000	3
Slogans	0.000000	0.000000	0.000000	3
Smears	0.780000	0.829787	0.804124	47
Thought-terminating cliché	0.000000	0.000000	0.000000	1
Transfer	0.000000	0.000000	0.000000	11
Whataboutism	0.000000	0.000000	0.000000	6

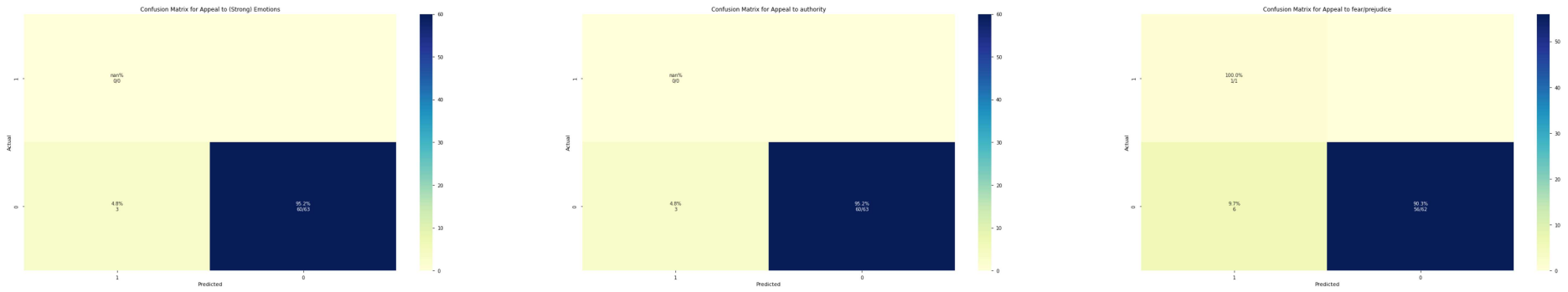
In [ ]:
```
val_preds = np.array([]).reshape((0,22))
val_true = np.array([]).reshape((0,22))
for i in range(validation_generator.__len__()):
    x,y = validation_generator.__getitem__(i)
    val_true = np.append(val_true,y,axis=0)
    val_preds = np.append(val_preds,models[model_name](x).numpy().reshape((1,22)),axis=0)
```

In [ ]:
```
plot_roc_curve(val_true, val_preds, mlb.classes_[0:3])
```



In [ ]:
```
plot_labelwise_confusion_matrix_(val_true, val_preds, mlb.classes_[0:3])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: invalid value encountered in true_divid
e
  # Remove the CWD from sys.path while we load stuff.
```



0.7 Model 4: BERT + ResNet -- Cross-Attention

💡 In this model, the text features are extracted using a BERT Transformer model and Image features are extracted using ResNet model.

- Cross Attention is used in the fusion mechanism.
- Here, the image features are used as query to the text values.

0.7.1 Defining Model

In [ ]:
```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_input_resnet
```

In [ ]:
```
# Defining the layers
preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
bert_encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=False, name='BERT_encoder')

# Create the base model from the pre-trained model Resnet50
resnet50 = ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
)
```

In [ ]:
```
resnet50.trainable = False
```

In [ ]:
```
# Defining the model graph

# text input
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
encoder_inputs = preprocessing_layer(text_input)
encoder_outputs = bert_encoder(encoder_inputs)
text_pooled_output = encoder_outputs['pooled_output']

# image input
image_input = Input(shape=(224, 224, 3), name='image')
image = preprocess_input_resnet(image_input)
image_features = resnet50(image, training=False)
image_features = tf.keras.layers.GlobalAveragePooling2D()(image_features)
image_features = tf.keras.layers.Dense(768, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2(0.

# Cross-attention.
query_value_attention_seq = keras.layers.Attention(use_scale=True, dropout=0.2)(
    [image_features, text_pooled_output]
)

# Concatenate the projections and pass through the classification layer.
concatenated = keras.layers.Concatenate()([image_features, text_pooled_output])
concatenated = keras.layers.Concatenate()([concatenated, query_value_attention_seq])

# custom head
x = tf.keras.layers.Dense(128, kernel_regularizer=tf.keras.regularizers.l2(0.001))(concatenated)
outputs = tf.keras.layers.Dense(22, activation="sigmoid")(x)

model_res_bert = keras.Model(inputs = [image_input, text_input], outputs=[outputs], name="model_res_bert")
model_res_bert.summary()
```

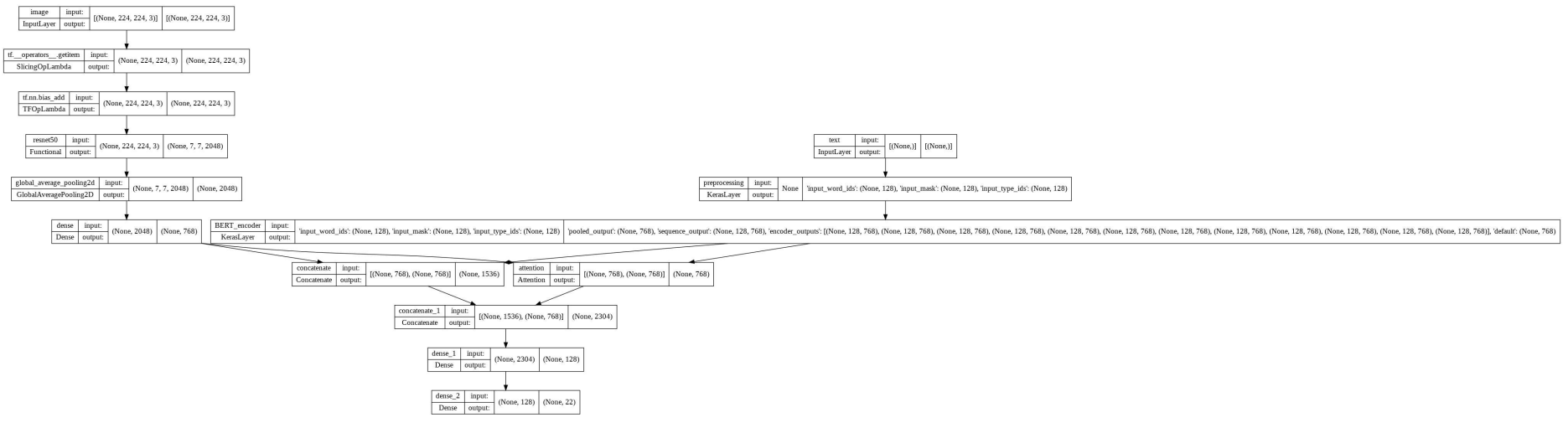
```
Model: "model_res_bert"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 image (InputLayer)             [(None, 224, 224, 3  0           []
                                )]

 tf.__operators__.getitem (Slic  (None, 224, 224, 3)  0          ['image[0][0]']
 ingOpLambda)

 tf.nn.bias_add (TFOpLambda)    (None, 224, 224, 3)  0           ['tf.__operators__.getitem[0][0]'
                                                                 ]

 resnet50 (Functional)          (None, 7, 7, 2048)   23587712    ['tf.nn.bias_add[0][0]']

 text (InputLayer)              [(None,)]            0           []

 global_average_pooling2d (Glob  (None, 2048)        0           ['resnet50[0][0]']
 alAveragePooling2D)

 preprocessing (KerasLayer)     {'input_word_ids':   0           ['text[0][0]']
                                (None, 128),
                                 'input_mask': (Non
                                e, 128),
                                 'input_type_ids':
                                (None, 128)}

 dense (Dense)                  (None, 768)          1573632     ['global_average_pooling2d[0][0]'
                                                                 ]

 BERT_encoder (KerasLayer)      {'pooled_output': (  109482241   ['preprocessing[0][0]',
                                None, 768),                       'preprocessing[0][1]',
                                 'sequence_output':               'preprocessing[0][2]']
                                 (None, 128, 768),
                                 'encoder_outputs':
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768)],
                                 'default': (None,
                                768)}

 concatenate (Concatenate)      (None, 1536)         0           ['dense[0][0]',
                                                                  'BERT_encoder[0][13]']

 attention (Attention)          (None, 768)          1           ['dense[0][0]',
                                                                  'BERT_encoder[0][13]']

 concatenate_1 (Concatenate)    (None, 2304)         0           ['concatenate[0][0]',
                                                                  'attention[0][0]']

 dense_1 (Dense)                (None, 128)          295040      ['concatenate_1[0][0]']
```

```
 dense_2 (Dense)                (None, 22)           2838        ['dense_1[0][0]']

==================================================================================================
Total params: 134,941,464
Trainable params: 1,871,511
Non-trainable params: 133,069,953
__________________________________________________________________________________________________
```

0.7.2 Plotting the model

In [ ]:
```
plot_model(model_res_bert, show_shapes=True)
```

Out[66]:



0.7.3 Training the model

In [ ]:
```
model_name = 'model_res_bert'
```

In [ ]:
```
!rm -rf /content/tensorboard_logs/models/model_res_bert
```

In [ ]:
```
epochs = 20
num_train_steps = STEPS_PER_EPOCH * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
adamw_optimizer = optimization.create_optimizer(init_lr=init_lr,
                                                num_train_steps=num_train_steps,
                                                num_warmup_steps=num_warmup_steps,
                                                optimizer_type='adamw')

adam_optimizer = tf.keras.optimizers.Adam(0.0001)
```

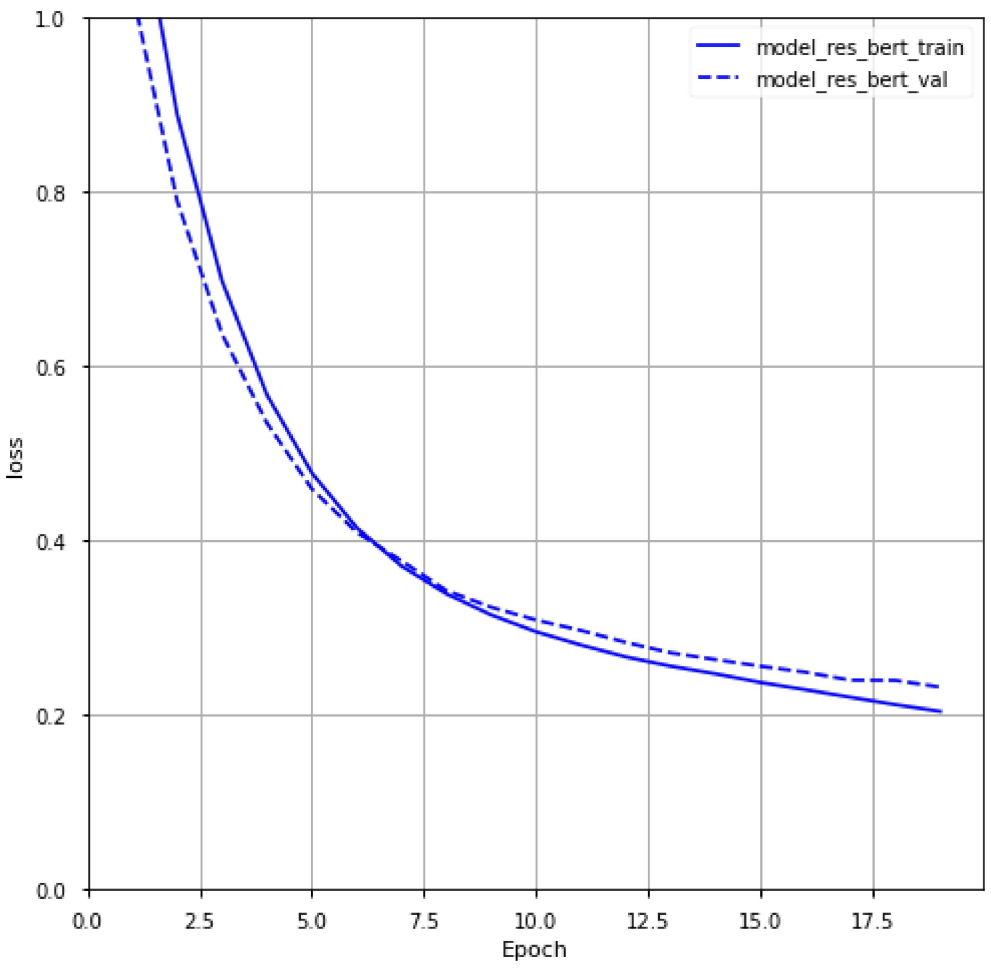
In [ ]:
```
m_histories[model_name], models[model_name] = compile_and_fit(
                model_res_bert,
                model_name,
                train_gen=training_generator,
                val_gen=validation_generator,
                max_epochs=20,
                optimizer=adam_optimizer,
                weights=None
            )
```

```
Epoch 1/20
85/85 [==============================] - ETA: 0s - loss: 1.6121 - binary_crossentropy: 0.3645 - precision: 0.4365 - re
call: 0.3463 - auc: 0.7875 - prc: 0.3436 - CategoricalAccuracy: 0.1676 - F1_macro: 0.0563 - F1_micro: 0.3099
Epoch 1: val_F1_macro improved from -inf to 0.05061, saving model to /content/tensorboard_logs/models/model_res_bert/m
odel_res_bert

WARNING:absl:Found untraced functions such as restored_function_body, restored_function_body, restored_function_body,
restored_function_body, restored_function_body while saving (showing 5 of 364). These functions will not be directly c
allable after loading.

85/85 [==============================] - 108s 1s/step - loss: 1.6121 - binary_crossentropy: 0.3645 - precision: 0.4365
- recall: 0.3463 - auc: 0.7875 - prc: 0.3436 - CategoricalAccuracy: 0.1676 - F1_macro: 0.0563 - F1_micro: 0.3099 - val
_loss: 1.3690 - val_binary_crossentropy: 0.3706 - val_precision: 0.5556 - val_recall: 0.3145 - val_auc: 0.7912 - val_p
rc: 0.4154 - val_CategoricalAccuracy: 0.1071 - val_F1_macro: 0.0506 - val_F1_micro: 0.3070
Epoch 2/20
85/85 [==============================] - ETA: 0s - loss: 1.1741 - binary_crossentropy: 0.3488 - precision: 0.5614 - re
call: 0.3749 - auc: 0.8267 - prc: 0.4562 - CategoricalAccuracy: 0.1853 - F1_macro: 0.0610 - F1_micro: 0.3492
Epoch 2: val_F1_macro did not improve from 0.05061
85/85 [==============================] - 16s 183ms/step - loss: 1.1741 - binary_crossentropy: 0.3488 - precision: 0.56
14 - recall: 0.3749 - auc: 0.8267 - prc: 0.4562 - CategoricalAccuracy: 0.1853 - F1_macro: 0.0610 - F1_micro: 0.3492 -
```

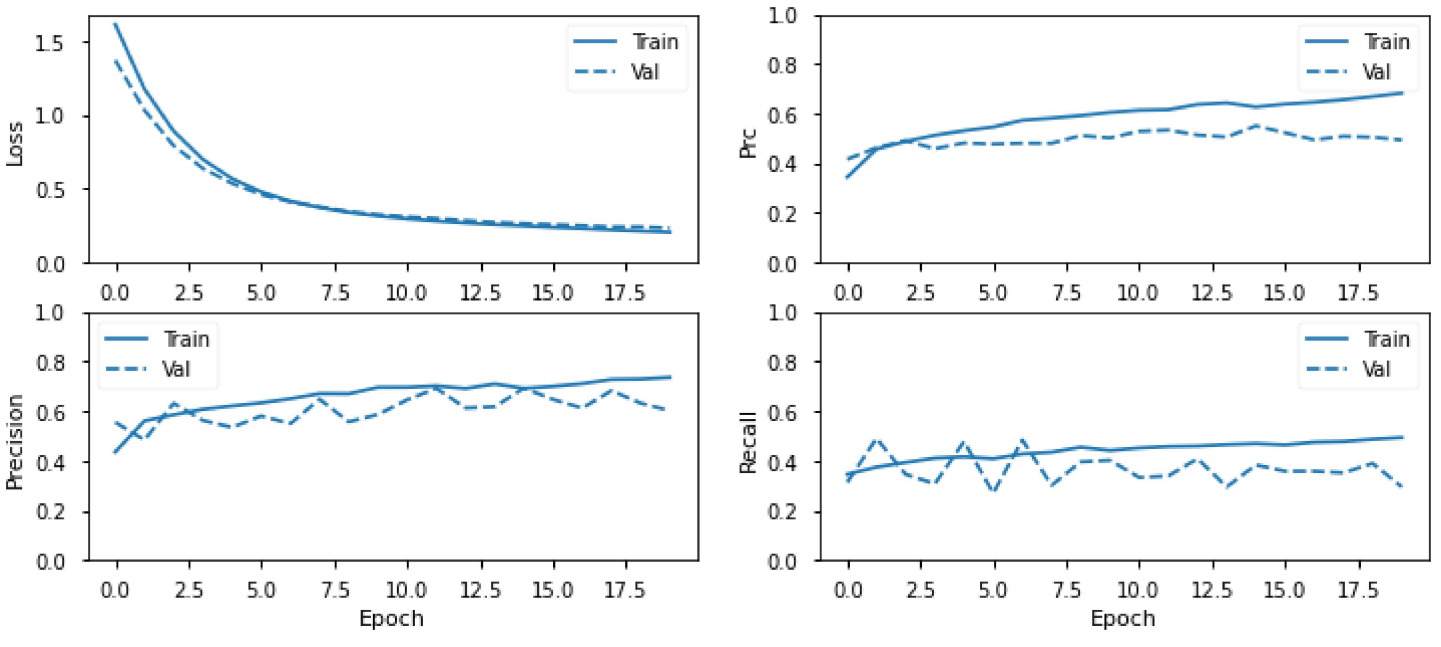
In [ ]:
```
download_files(m_histories, model_name)
```

0.7.4 Plot Metrics

In [ ]:
```
plotter(m_histories, metric="loss", ylim=[0, 1.0])
```



In [ ]:
```
plot_metrics(m_histories[model_name])
```



0.7.5 Evaluation on Validation Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert/model_res_bert')
```

In [ ]:
```
# Applying a threshold and converting the probability values in the
# output to binary boolean values (1 or 0)
threshold = 0.5
```

0.7.5.1 Storing the predictions to dataframe for plotting

In [ ]:
```
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=1,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    shuffle=False
                )
dev_pred = models[model_name].predict(validation_generator, verbose=1)
dev_prediction = (dev_pred > threshold).astype('int')
dev_prediction_df = prediction_df(dev_df, dev_prediction, mlb)
dev_prediction_df.head()
```

```
63/63 [==============================] - 8s 58ms/step
```

Out[87]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Smears, Doubt]	[Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	[Smears, Loaded Language, Name calling/Labeling]	[Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
2	[Smears, Transfer]	[Name calling/Labeling]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
3	[]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
4	[Smears, Exaggeration/Minimisation, Loaded Lan...	[Name calling/Labeling]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...

In [ ]:
```
print("F1 macro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
print("F1 micro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
```

```
F1 macro on Dev Set:  0.141
F1 micro on Dev Set:  0.454
```

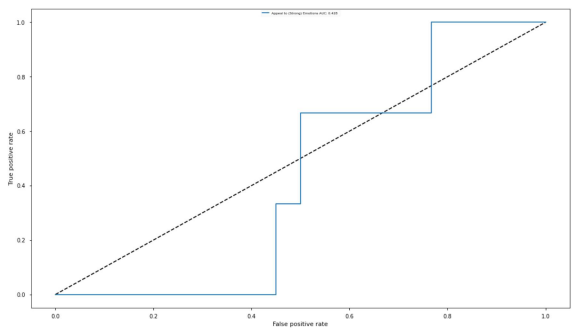
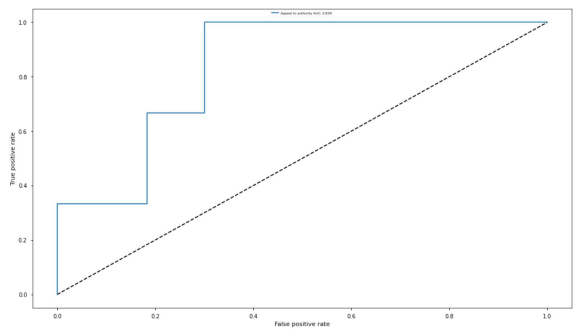
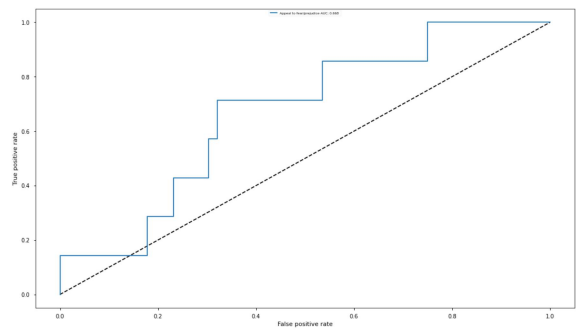
In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(dev_prediction_df['actual_labels_mh'].values.tolist(), de
perf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
perf_table.index = mlb.classes_
perf_table
```

Out[89]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	3
Appeal to authority	1.000000	0.333333	0.500000	3
Appeal to fear/prejudice	0.000000	0.000000	0.000000	7
Bandwagon	0.000000	0.000000	0.000000	2
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	0
Causal Oversimplification	0.000000	0.000000	0.000000	1
Doubt	0.000000	0.000000	0.000000	9
Exaggeration/Minimisation	0.000000	0.000000	0.000000	8
Flag-waving	0.000000	0.000000	0.000000	7
Glittering generalities (Virtue)	0.000000	0.000000	0.000000	4
Loaded Language	0.652174	0.468750	0.545455	32
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	5
Name calling/Labeling	0.513514	0.633333	0.567164	30
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	0
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	0
Reductio ad hitlerum	0.000000	0.000000	0.000000	1
Repetition	0.666667	0.666667	0.666667	3
Slogans	0.000000	0.000000	0.000000	3
Smears	0.787879	0.553191	0.650000	47
Thought-terminating cliché	0.000000	0.000000	0.000000	1
Transfer	1.000000	0.090909	0.166667	11
Whataboutism	0.000000	0.000000	0.000000	6

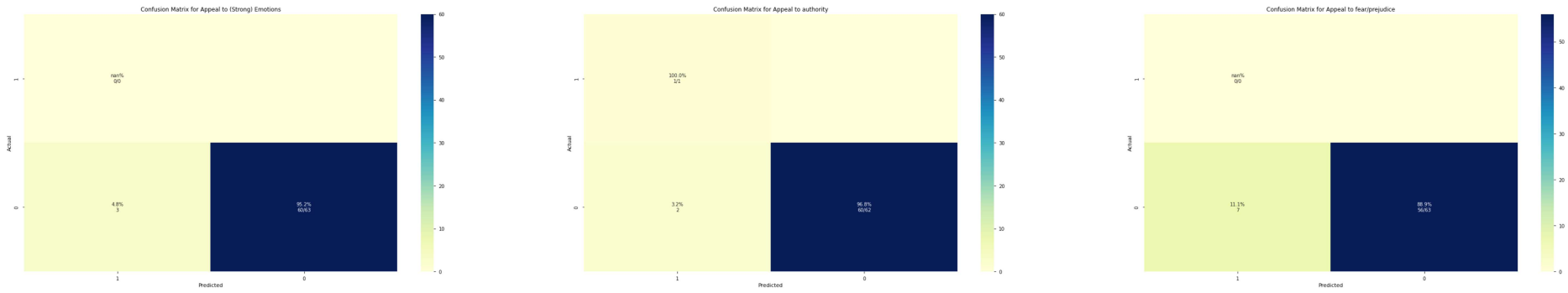
In [ ]:
```
val_preds = np.array([]).reshape((0,22))
val_true = np.array([]).reshape((0,22))
for i in range(validation_generator.__len__()):
    x,y = validation_generator.__getitem__(i)
    val_true = np.append(val_true,y,axis=0)
    val_preds = np.append(val_preds,models[model_name](x).numpy().reshape((1,22)),axis=0)
```

In [ ]:
```
plot_roc_curve(val_true, val_preds, mlb.classes_[0:3])
```

In [ ]:
```
plot_labelwise_confusion_matrix_(val_true, val_preds, mlb.classes_[0:3])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: invalid value encountered in true_divid
e
  # Remove the CWD from sys.path while we load stuff.
```



0.8 Model 5: BERT + ResNet + Stack + BiGRU

💡 In this model, the text features are extracted using a BERT Transformer model and Image features are extracted using ResNet model.

- Cross Attention is used in the fusion mechanism.
- The output from cross-attention is stack with the original image and text embeddings.
- The stacked output is passed through a bidirectional GRU layer.

0.8.1 Defining Model

In [ ]:
```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_input_resnet
```

In [ ]:
```
# Defining the layers
preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
bert_encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=False, name='BERT_encoder')

# Create the base model from the pre-trained model Resnet50
resnet50 = ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
)
```

In [ ]:
```
resnet50.trainable = False
```

In [ ]:
```python
# Defining the model graph

# text input
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
encoder_inputs = preprocessing_layer(text_input)
encoder_outputs = bert_encoder(encoder_inputs)
text_pooled_output = encoder_outputs['pooled_output']

# image input
image_input = Input(shape=(224, 224, 3), name='image')
image = preprocess_input_resnet(image_input)
image_features = resnet50(image, training=False)
image_features = tf.keras.layers.GlobalAveragePooling2D()(image_features)
image_features = tf.keras.layers.Dense(768, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2(0.


# Cross-attention.
query_value_attention_seq = keras.layers.Attention(use_scale=True, dropout=0.2)(
    [image_features, text_pooled_output]
)

# Concatenate the projections and pass through the classification layer.
stacking_attention_concat = tf.stack([image_features, query_value_attention_seq, text_pooled_output])
transpose = tf.transpose(stacking_attention_concat, [1,0,2])
bidirectional_gru = tf.keras.layers.Bidirectional(tf.keras.layers.GRU(64))(transpose)

# custom head
x = tf.keras.layers.Dense(128, kernel_regularizer=tf.keras.regularizers.l2(0.001))(bidirectional_gru)
outputs = tf.keras.layers.Dense(22, activation="sigmoid")(x)

model_res_bert_gru = keras.Model(inputs = [image_input, text_input], outputs=[outputs], name="model_res_bert_gru")
model_res_bert_gru.summary()
```

```
(None, 3, 768)
Model: "model_res_bert_gru"
```

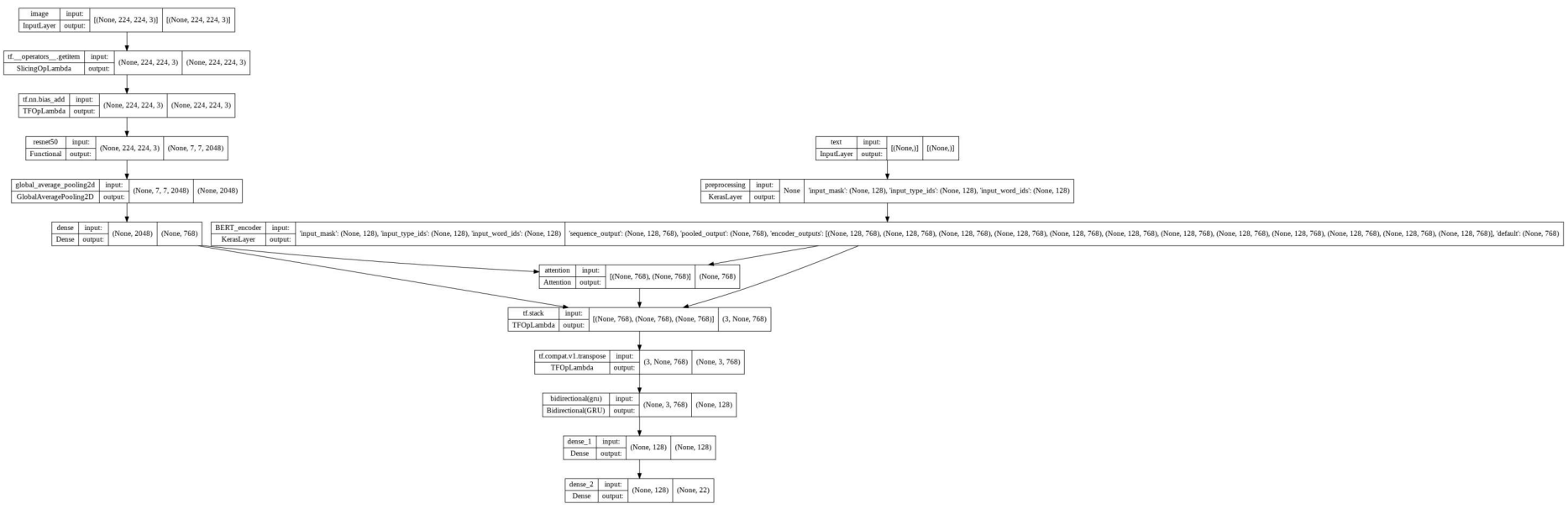
Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	[(None, 224, 224, 3)]	0	[]
tf.__operators__.getitem (SlicingOpLambda)	(None, 224, 224, 3)	0	['image[0][0]']
tf.nn.bias_add (TFOpLambda)	(None, 224, 224, 3)	0	['tf.__operators__.getitem[0][0]']
resnet50 (Functional)	(None, 7, 7, 2048)	23587712	['tf.nn.bias_add[0][0]']
text (InputLayer)	[(None,)]	0	[]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	['resnet50[0][0]']
preprocessing (KerasLayer)	{'input_mask': (None, 128), 'input_type_ids': (None, 128), 'input_word_ids': (None, 128)}	0	['text[0][0]']
dense (Dense)	(None, 768)	1573632	['global_average_pooling2d[0][0]']
BERT_encoder (KerasLayer)	{'sequence_output': (None, 128, 768), 'pooled_output': (None, 768), 'encoder_outputs': [(None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768)], 'default': (None, 768)}	109482241	['preprocessing[0][0]', 'preprocessing[0][1]', 'preprocessing[0][2]']
attention (Attention)	(None, 768)	1	['dense[0][0]', 'BERT_encoder[0][13]']
tf.stack (TFOpLambda)	(3, None, 768)	0	['dense[0][0]', 'attention[0][0]', 'BERT_encoder[0][13]']

```
tf.compat.v1.transpose (TFOpLa  (None, 3, 768)      0           ['tf.stack[0][0]']
mbda)

bidirectional (Bidirectional)  (None, 128)          320256      ['tf.compat.v1.transpose[0][0]']

dense_1 (Dense)                (None, 128)          16512       ['bidirectional[0][0]']

dense_2 (Dense)                (None, 22)           2838        ['dense_1[0][0]']

==================================================================================================
Total params: 134,983,192
Trainable params: 1,913,239
Non-trainable params: 133,069,953
__________________________________________________________________________________________________
```

0.8.2 Plotting the model

In [ ]:
```
plot_model(model_res_bert_gru, show_shapes=True)
```

Out[66]:



0.8.3 Training the model

In [ ]:
```
model_name = 'model_res_bert_gru'
```

In [ ]:
```
!rm -rf /content/tensorboard_logs/models/model_res_bert_gru
```

In [ ]:
```
epochs = 20
num_train_steps = STEPS_PER_EPOCH * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
adamw_optimizer = optimization.create_optimizer(init_lr=init_lr,
                                                num_train_steps=num_train_steps,
                                                num_warmup_steps=num_warmup_steps,
                                                optimizer_type='adamw')

adam_optimizer = tf.keras.optimizers.Adam(0.0001)
```

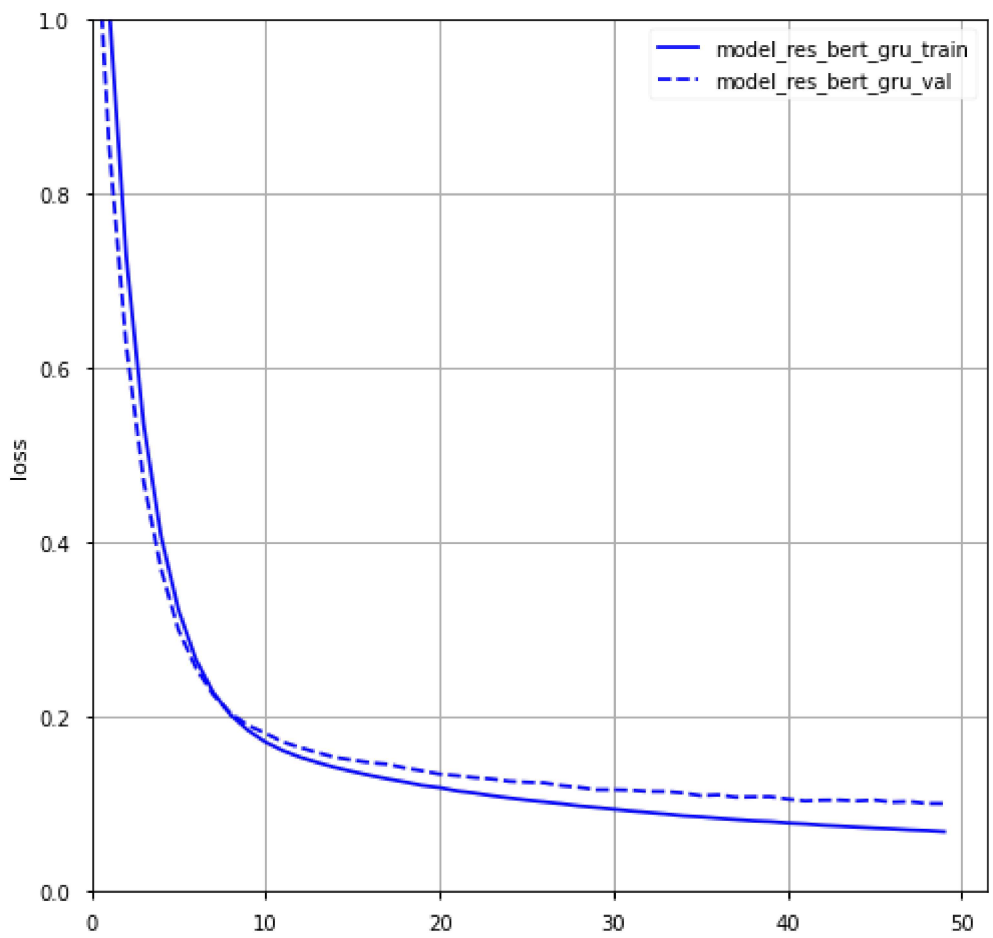
In [ ]:
```
m_histories[model_name], models[model_name] = compile_and_fit(
                model_res_bert_gru,
                model_name,
                train_gen=training_generator,
                val_gen=validation_generator,
                max_epochs=50,
                optimizer=adam_optimizer,
                weights=None
            )
```

```
Epoch 1/50
85/85 [==============================] - ETA: 0s - loss: 1.4682 - binary_crossentropy: 0.4086 - precision: 0.4733 - re
call: 0.3959 - auc: 0.7984 - prc: 0.3673 - CategoricalAccuracy: 0.1721 - F1_macro: 0.0551 - F1_micro: 0.3268
Epoch 1: val_F1_macro improved from -inf to 0.04551, saving model to /content/tensorboard_logs/models/model_res_bert_g
ru/model_res_bert_gru

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn, gru_cell_1_layer_call_and_return_conditional_l
osses, gru_cell_2_layer_call_fn, gru_cell_2_layer_call_and_return_conditional_losses, restored_function_body while sav
ing (showing 5 of 368). These functions will not be directly callable after loading.
WARNING:absl:<keras.layers.recurrent.GRUCell object at 0x7f9ebcfe35d0> has the same name 'GRUCell' as a built-in Keras
object. Consider renaming <class 'keras.layers.recurrent.GRUCell'> to avoid naming conflicts when loading with `tf.ker
as.models.load_model`. If renaming is not possible, pass the object in the `custom_objects` parameter of the load func
tion.
WARNING:absl:<keras.layers.recurrent.GRUCell object at 0x7f9ebd26c310> has the same name 'GRUCell' as a built-in Keras
object. Consider renaming <class 'keras.layers.recurrent.GRUCell'> to avoid naming conflicts when loading with `tf.ker
as.models.load_model`. If renaming is not possible, pass the object in the `custom_objects` parameter of the load func
tion.
```

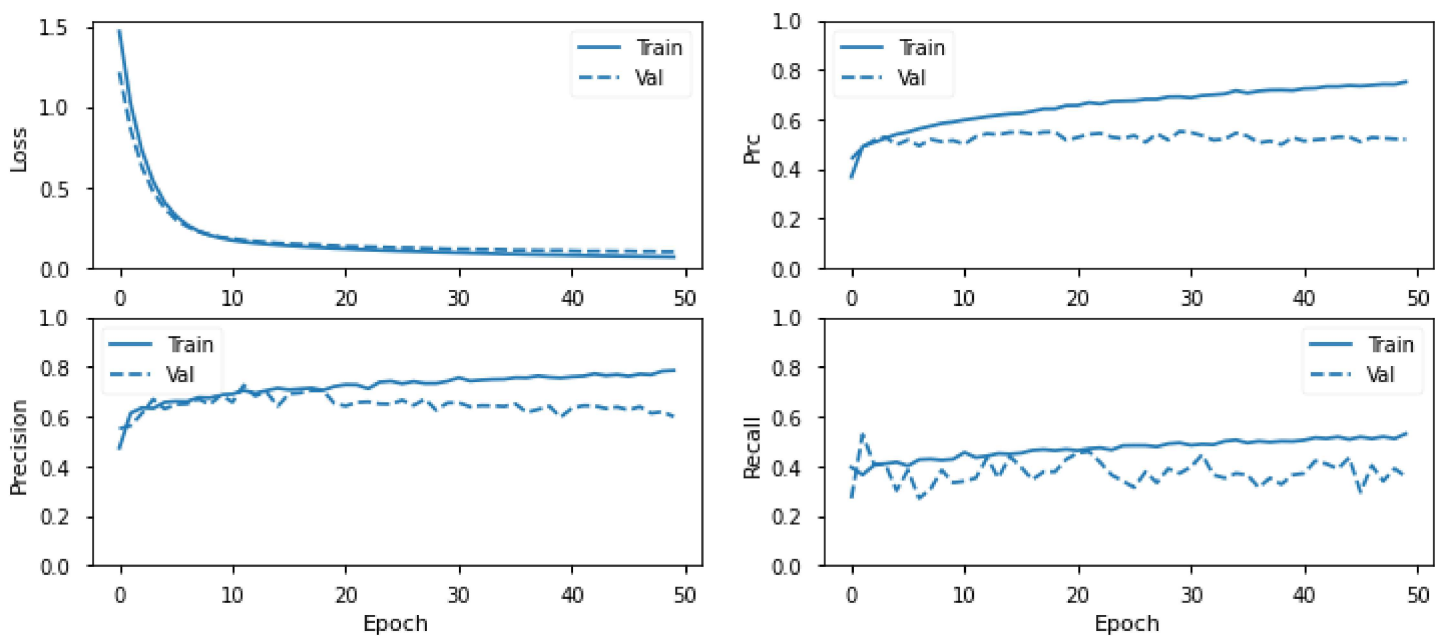
In [ ]:
```
download_files(m_histories, model_name)
```

0.8.4 Plot Metrics

In [ ]:
```
plotter(m_histories, metric="loss", ylim=[0, 1.0])
```



In [ ]:
```
plot_metrics(m_histories[model_name])
```



0.8.5 Evaluation on Validation Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert_gru/model_res_bert_gru')
```

In [ ]:
```
# Applying a threshold and converting the probability values in the
# output to binary boolean values (1 or 0)
threshold = 0.5
```

0.8.5.1 Storing the predictions to dataframe for plotting

In [ ]:
```
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=1,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    shuffle=False
                )
dev_pred = loaded_model.predict(validation_generator, verbose=1)
dev_prediction = (dev_pred > threshold).astype('int')
dev_prediction_df = prediction_df(dev_df, dev_prediction, mlb)
dev_prediction_df.head()
```

```
63/63 [==============================] - 6s 45ms/step
```

Out[75]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Smears, Doubt]	[Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	[Smears, Loaded Language, Name calling/Labeling]	[Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
2	[Smears, Transfer]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
3	[]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
4	[Smears, Exaggeration/Minimisation, Loaded Lan...	[Loaded Language]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...

In [ ]:
```
print("F1 macro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
print("F1 micro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
```

```
F1 macro on Dev Set:  0.137
F1 micro on Dev Set:  0.478
```

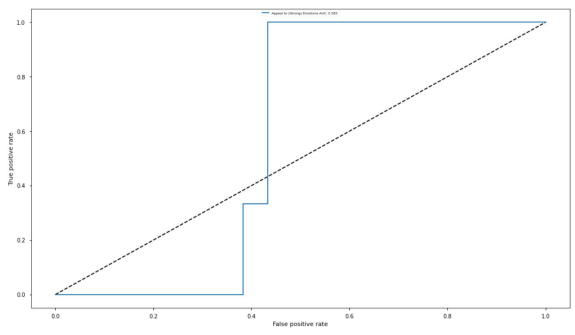
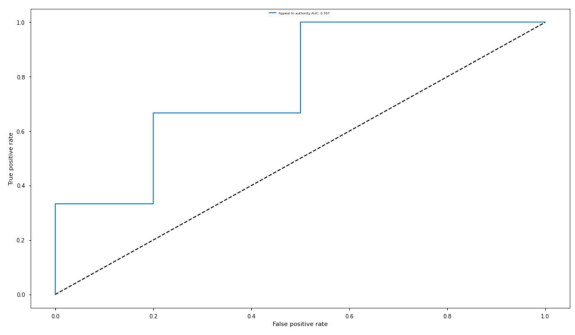
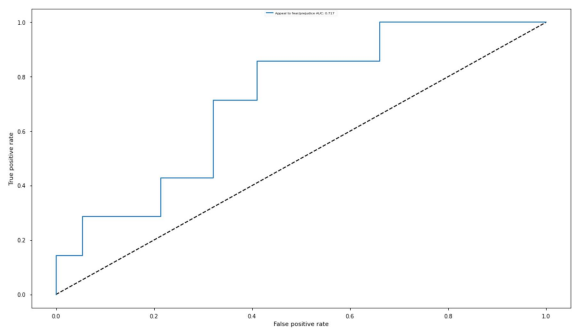
In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(dev_prediction_df['actual_labels_mh'].values.tolist(), de
perf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
perf_table.index = mlb.classes_
perf_table
```

Out[77]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	3
Appeal to authority	1.000000	0.333333	0.500000	3
Appeal to fear/prejudice	0.000000	0.000000	0.000000	7
Bandwagon	0.000000	0.000000	0.000000	2
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	0
Causal Oversimplification	0.000000	0.000000	0.000000	1
Doubt	0.000000	0.000000	0.000000	9
Exaggeration/Minimisation	0.000000	0.000000	0.000000	8
Flag-waving	0.000000	0.000000	0.000000	7
Glittering generalities (Virtue)	0.000000	0.000000	0.000000	4
Loaded Language	0.615385	0.750000	0.676056	32
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	5
Name calling/Labeling	0.521739	0.400000	0.452830	30
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	0
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	0
Reductio ad hitlerum	0.000000	0.000000	0.000000	1
Repetition	0.000000	0.000000	0.000000	3
Slogans	1.000000	0.333333	0.500000	3
Smears	0.794872	0.659574	0.720930	47
Thought-terminating cliché	0.000000	0.000000	0.000000	1
Transfer	1.000000	0.090909	0.166667	11
Whataboutism	0.000000	0.000000	0.000000	6

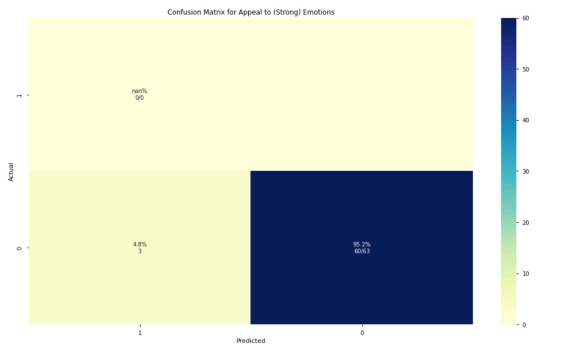
In [ ]:
```
val_preds = np.array([]).reshape((0,22))
val_true = np.array([]).reshape((0,22))
for i in range(validation_generator.__len__()):
    x,y = validation_generator.__getitem__(i)
    val_true = np.append(val_true,y,axis=0)
    val_preds = np.append(val_preds,models[model_name](x).numpy().reshape((1,22)),axis=0)
```

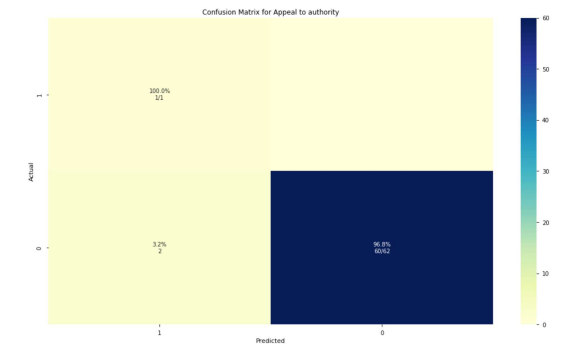
In [ ]:
```
plot_roc_curve(val_true, val_preds, mlb.classes_[0:3])
```

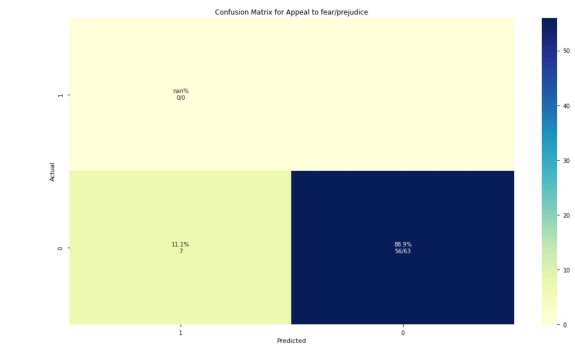




In [ ]:
```
plot_labelwise_confusion_matrix_(val_true, val_preds, mlb.classes_[0:3])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: invalid value encountered in true_divid
e
  # Remove the CWD from sys.path while we load stuff.
```







0.9 Model 6: BERT Fine Tuned + ResNet + Stack + BiGRU

💡 In this model, the text features are extracted using a BERT Transformer model and Image features are extracted using ResNet model.

- Cross Attention is used in the fusion mechanism.
- The output from cross-attention is stack with the original image and text embeddings.
- The stacked output is passed through a bidirectional GRU layer.
- Compared to all the models above, here the BERT model is fine-tuned using the given data.

0.9.1 Defining Model

In [63]:
```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_input_resnet
```

In [64]:
```
# Defining the layers
preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
bert_encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')

# Create the base model from the pre-trained model Resnet50
resnet50 = ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_orderi
ng_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_orde
ring_tf_kernels_notop.h5)
94773248/94765736 [==============================] - 0s 0us/step
94781440/94765736 [==============================] - 0s 0us/step
```

In [65]:
```
resnet50.trainable = False
```

In [66]:
```
# Defining the model graph

# text input
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
encoder_inputs = preprocessing_layer(text_input)
encoder_outputs = bert_encoder(encoder_inputs)
text_pooled_output = encoder_outputs['pooled_output']

# image input
image_input = Input(shape=(224, 224, 3), name='image')
image = preprocess_input_resnet(image_input)
image_features = resnet50(image, training=False)
image_features = tf.keras.layers.GlobalAveragePooling2D()(image_features)
image_features = tf.keras.layers.Dense(768, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2(0.

# Cross-attention.
query_value_attention_seq = keras.layers.Attention(use_scale=True, dropout=0.2)(
    [image_features, text_pooled_output]
)

# Concatenate the projections and pass through the classification layer.
stacking_attention_concat = tf.stack([image_features, query_value_attention_seq, text_pooled_output])
transpose = tf.transpose(stacking_attention_concat, [1,0,2])
bidirectional_gru = tf.keras.layers.Bidirectional(tf.keras.layers.GRU(64))(transpose)

# custom head
x = tf.keras.layers.Dense(128, kernel_regularizer=tf.keras.regularizers.l2(0.001))(bidirectional_gru)
outputs = tf.keras.layers.Dense(22, activation="sigmoid")(x)

model_res_bertft_gru = keras.Model(inputs = [image_input, text_input], outputs=[outputs], name="model_res_bertft_gru")
```

```
model_res_bertft_gru.summary()
```

```
Model: "model_res_bertft_gru"
```

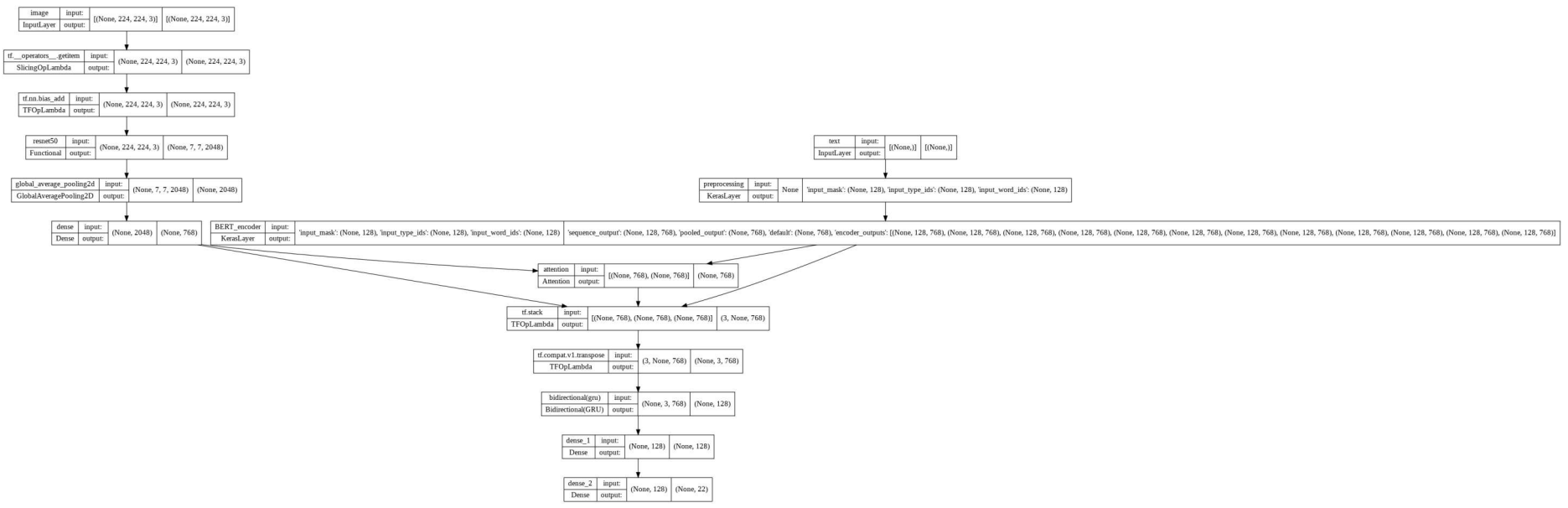
Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	[(None, 224, 224, 3)]	0	[]
tf.__operators__.getitem (SlicingOpLambda)	(None, 224, 224, 3)	0	['image[0][0]']
tf.nn.bias_add (TFOpLambda)	(None, 224, 224, 3)	0	['tf.__operators__.getitem[0][0]']
resnet50 (Functional)	(None, 7, 7, 2048)	23587712	['tf.nn.bias_add[0][0]']
text (InputLayer)	[(None,)]	0	[]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	['resnet50[0][0]']
preprocessing (KerasLayer)	{'input_mask': (None, 128), 'input_type_ids': (None, 128), 'input_word_ids': (None, 128)}	0	['text[0][0]']
dense (Dense)	(None, 768)	1573632	['global_average_pooling2d[0][0]']
BERT_encoder (KerasLayer)	{'sequence_output': (None, 128, 768), 'pooled_output': (None, 768), 'default': (None, 768), 'encoder_outputs': [(None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768)]}	109482241	['preprocessing[0][0]', 'preprocessing[0][1]', 'preprocessing[0][2]']
attention (Attention)	(None, 768)	1	['dense[0][0]', 'BERT_encoder[0][13]']
tf.stack (TFOpLambda)	(3, None, 768)	0	['dense[0][0]', 'attention[0][0]', 'BERT_encoder[0][13]']
tf.compat.v1.transpose (TFOpLambda)	(None, 3, 768)	0	['tf.stack[0][0]']

```
 bidirectional (Bidirectional)  (None, 128)          320256      ['tf.compat.v1.transpose[0][0]']

 dense_1 (Dense)                (None, 128)          16512       ['bidirectional[0][0]']

 dense_2 (Dense)                (None, 22)           2838        ['dense_1[0][0]']

==================================================================================================
Total params: 134,983,192
Trainable params: 111,395,479
Non-trainable params: 23,587,713
__________________________________________________________________________________________________
```

0.9.2 Plotting the model

In [67]:
```
plot_model(model_res_bertft_gru, show_shapes=True)
```

Out[67]:



0.9.3 Training the model

In [68]:
```
model_name = 'model_res_bertft_gru'
```

In [69]:
```
!rm -rf /content/tensorboard_logs/models/model_res_bertft_gru
```

In [70]:
```
epochs = 20
num_train_steps = STEPS_PER_EPOCH * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
adamw_optimizer = optimization.create_optimizer(init_lr=init_lr,
                                                num_train_steps=num_train_steps,
                                                num_warmup_steps=num_warmup_steps,
                                                optimizer_type='adamw')

adam_optimizer = tf.keras.optimizers.Adam(3e-5)
```

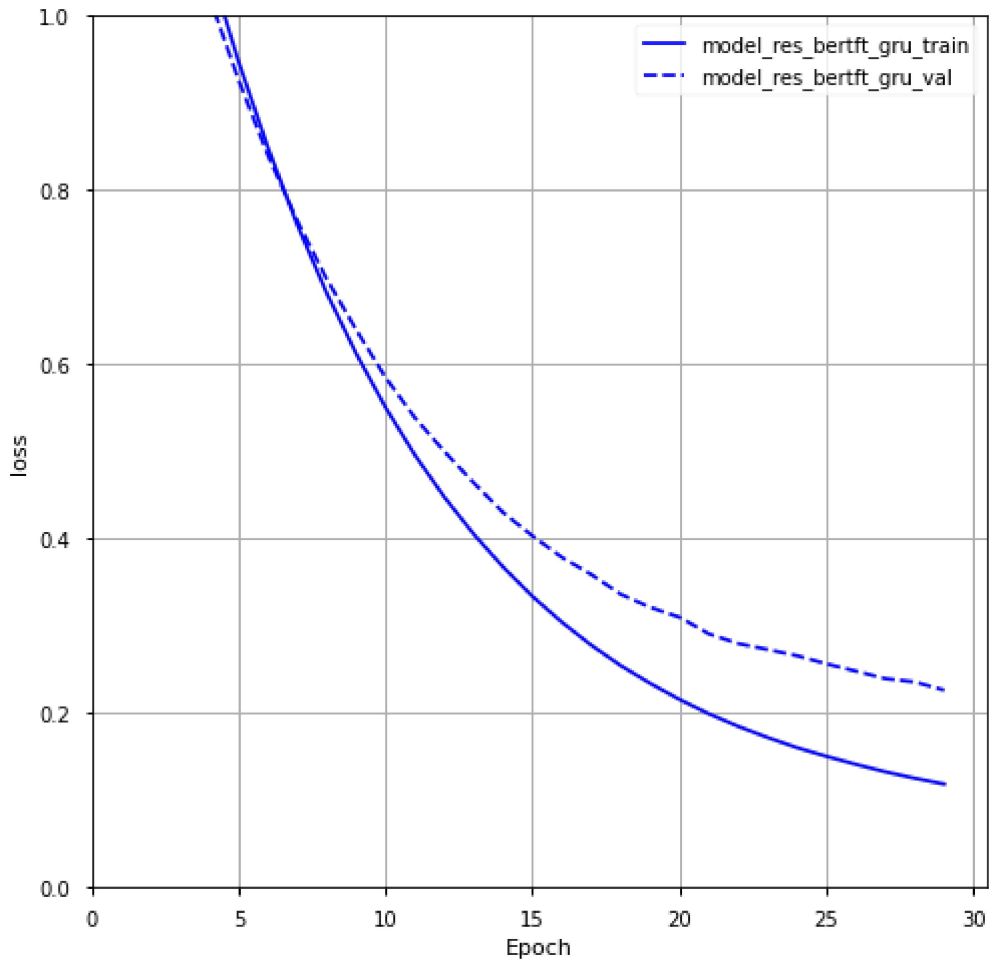
In [71]:
```
m_histories[model_name], models[model_name] = compile_and_fit(
                model_res_bertft_gru,
                model_name,
                train_gen=training_generator,
                val_gen=validation_generator,
                max_epochs=30,
                optimizer=adam_optimizer,
                weights=None
            )
```

```
Epoch 1/30
85/85 [==============================] - ETA: 0s - loss: 1.6613 - binary_crossentropy: 0.4331 - precision: 0.4038 - re
call: 0.3335 - auc: 0.7860 - prc: 0.3192 - CategoricalAccuracy: 0.1544 - F1_macro: 0.0585 - F1_micro: 0.3411
Epoch 1: val_F1_macro improved from -inf to 0.03843, saving model to /content/tensorboard_logs/models/model_res_bertft
_gru/model_res_bertft_gru

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn, gru_cell_1_layer_call_and_return_conditional_l
osses, gru_cell_2_layer_call_fn, gru_cell_2_layer_call_and_return_conditional_losses, restored_function_body while sav
ing (showing 5 of 368). These functions will not be directly callable after loading.
WARNING:absl:<keras.layers.recurrent.GRUCell object at 0x7f92c80c1390> has the same name 'GRUCell' as a built-in Keras
object. Consider renaming <class 'keras.layers.recurrent.GRUCell'> to avoid naming conflicts when loading with `tf.ker
as.models.load_model`. If renaming is not possible, pass the object in the `custom_objects` parameter of the load func
tion.
WARNING:absl:<keras.layers.recurrent.GRUCell object at 0x7f92c813b610> has the same name 'GRUCell' as a built-in Keras
object. Consider renaming <class 'keras.layers.recurrent.GRUCell'> to avoid naming conflicts when loading with `tf.ker
as.models.load_model`. If renaming is not possible, pass the object in the `custom_objects` parameter of the load func
tion.
```

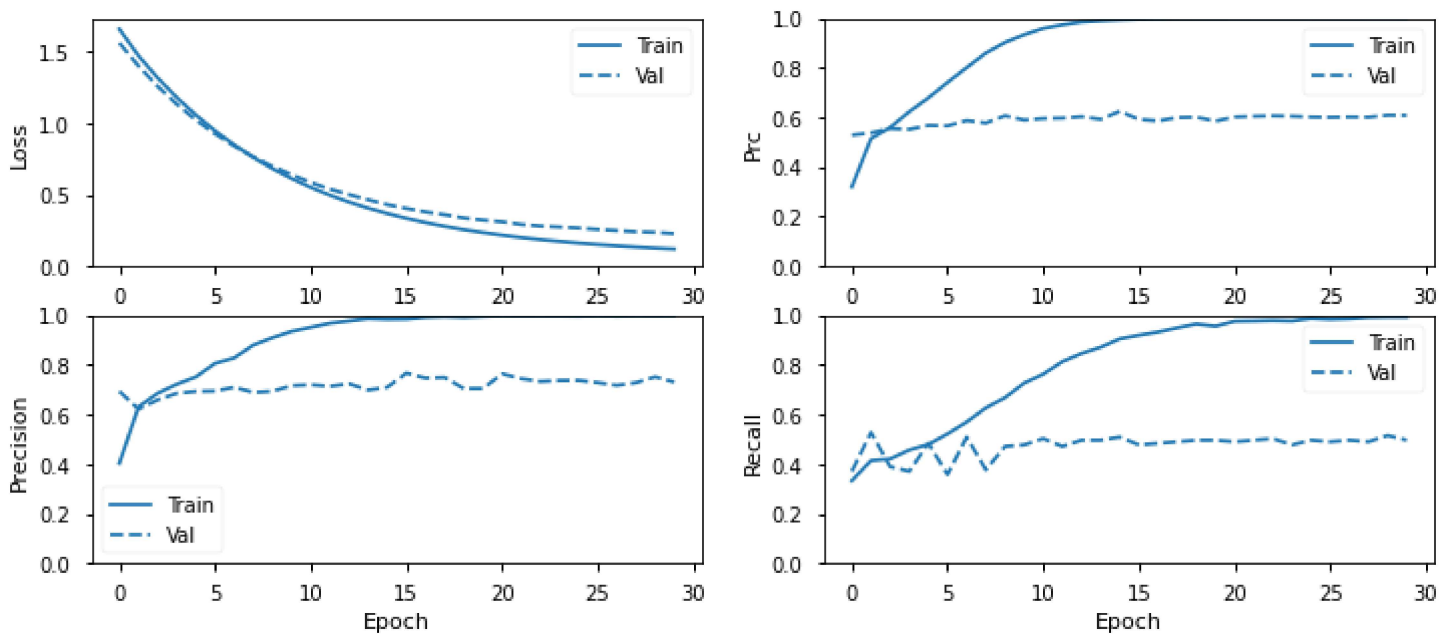
In [ ]:
```
download_files(m_histories, model_name)
```

0.9.4 Plot Metrics

In [73]:
```
plotter(m_histories, metric="loss", ylim=[0, 1.0])
```



In [74]:
```
plot_metrics(m_histories[model_name])
```



0.9.5 Evaluation on Validation Set

In [75]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bertft_gru/model_res_bertft_gru')
```

In [76]:
```
# Applying a threshold and converting the probability values in the
# output to binary boolean values (1 or 0)
threshold = 0.5
```

0.9.5.1 Storing the predictions to dataframe for plotting

In [77]:
```
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=1,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    shuffle=False
                )
dev_pred = loaded_model.predict(validation_generator, verbose=1)
dev_prediction = (dev_pred > threshold).astype('int')
dev_prediction_df = prediction_df(dev_df, dev_prediction, mlb)
dev_prediction_df.head()
```

```
63/63 [==============================] - 8s 49ms/step
```

Out[77]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Smears, Doubt]	[Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	[Smears, Loaded Language, Name calling/Labeling]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
2	[Smears, Transfer]	[Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	[]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
4	[Smears, Exaggeration/Minimisation, Loaded Lan...	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...

In [78]:
```
print("F1 macro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
print("F1 micro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
```

```
F1 macro on Dev Set:  0.216
F1 micro on Dev Set:  0.608
```

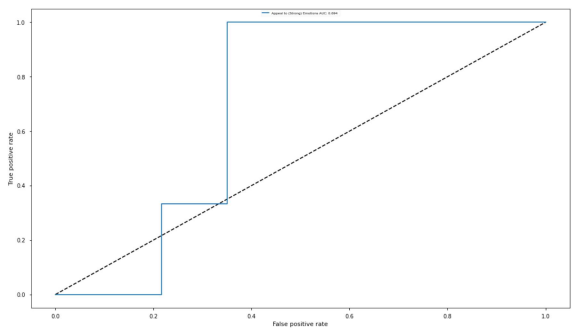
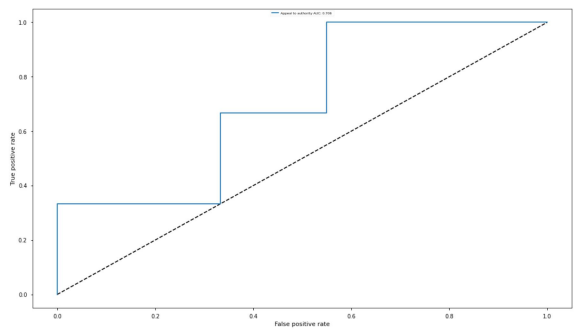
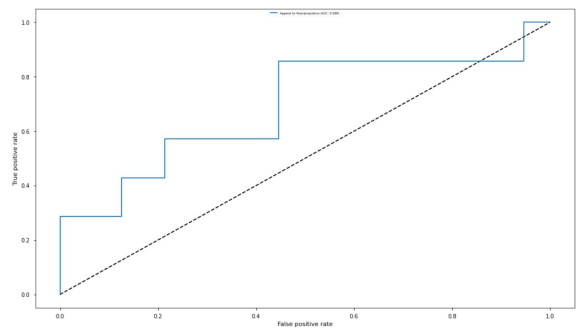
In [79]:
```
precision, recall, f1, support = precision_recall_fscore_support(dev_prediction_df['actual_labels_mh'].values.tolist(), de
perf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
perf_table.index = mlb.classes_
perf_table
```

Out[79]:

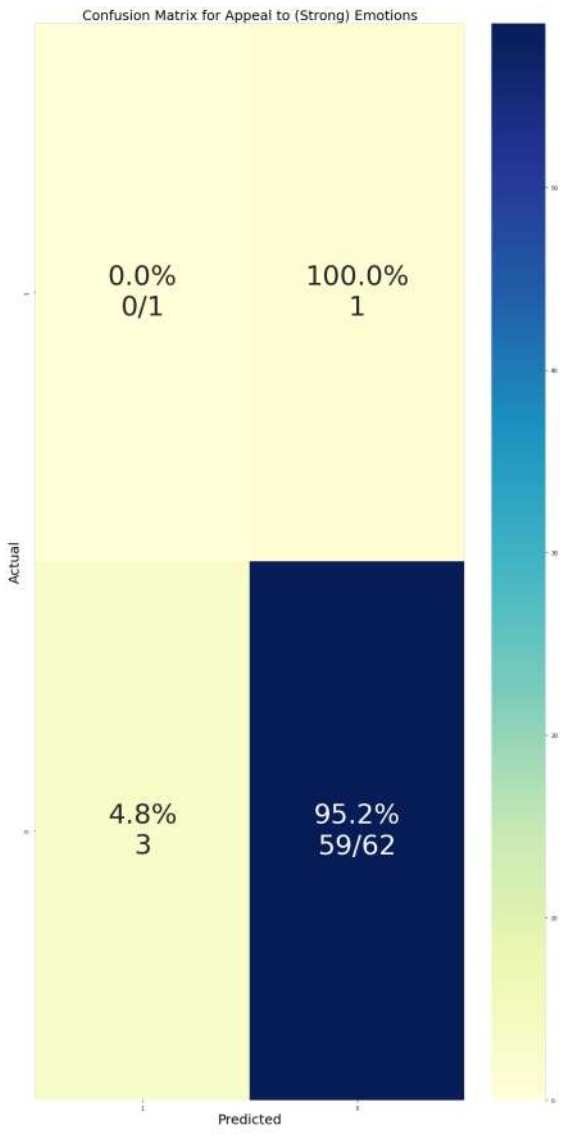
	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	3
Appeal to authority	1.000000	0.333333	0.500000	3
Appeal to fear/prejudice	1.000000	0.142857	0.250000	7
Bandwagon	0.000000	0.000000	0.000000	2
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	0
Causal Oversimplification	0.000000	0.000000	0.000000	1
Doubt	0.000000	0.000000	0.000000	9
Exaggeration/Minimisation	0.000000	0.000000	0.000000	8
Flag-waving	1.000000	0.285714	0.444444	7
Glittering generalities (Virtue)	0.500000	0.250000	0.333333	4
Loaded Language	0.743590	0.906250	0.816901	32
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	5
Name calling/Labeling	0.642857	0.600000	0.620690	30
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	0
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	0
Reductio ad hitlerum	0.000000	0.000000	0.000000	1
Repetition	0.000000	0.000000	0.000000	3
Slogans	1.000000	0.666667	0.800000	3
Smears	0.844444	0.808511	0.826087	47
Thought-terminating cliché	0.000000	0.000000	0.000000	1
Transfer	1.000000	0.090909	0.166667	11
Whataboutism	0.000000	0.000000	0.000000	6

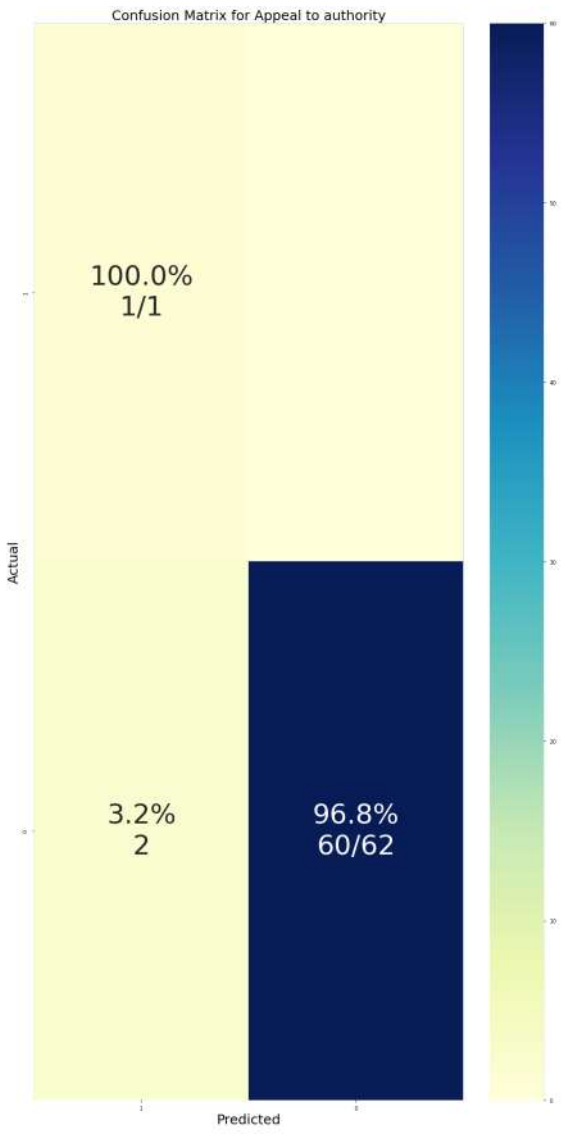
In [80]:
```
val_preds = np.array([]).reshape((0,22))
val_true = np.array([]).reshape((0,22))
for i in range(validation_generator.__len__()):
    x,y = validation_generator.__getitem__(i)
    val_true = np.append(val_true,y,axis=0)
    val_preds = np.append(val_preds,models[model_name](x).numpy().reshape((1,22)),axis=0)
```

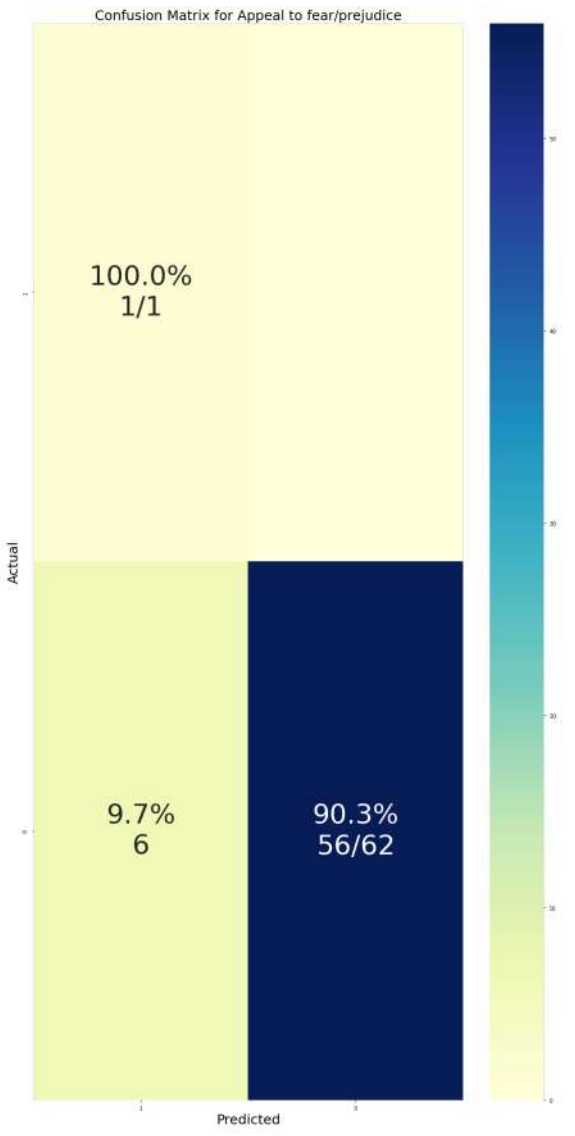
In [81]:
```
plot_roc_curve(val_true, val_preds, mlb.classes_[0:3])
```

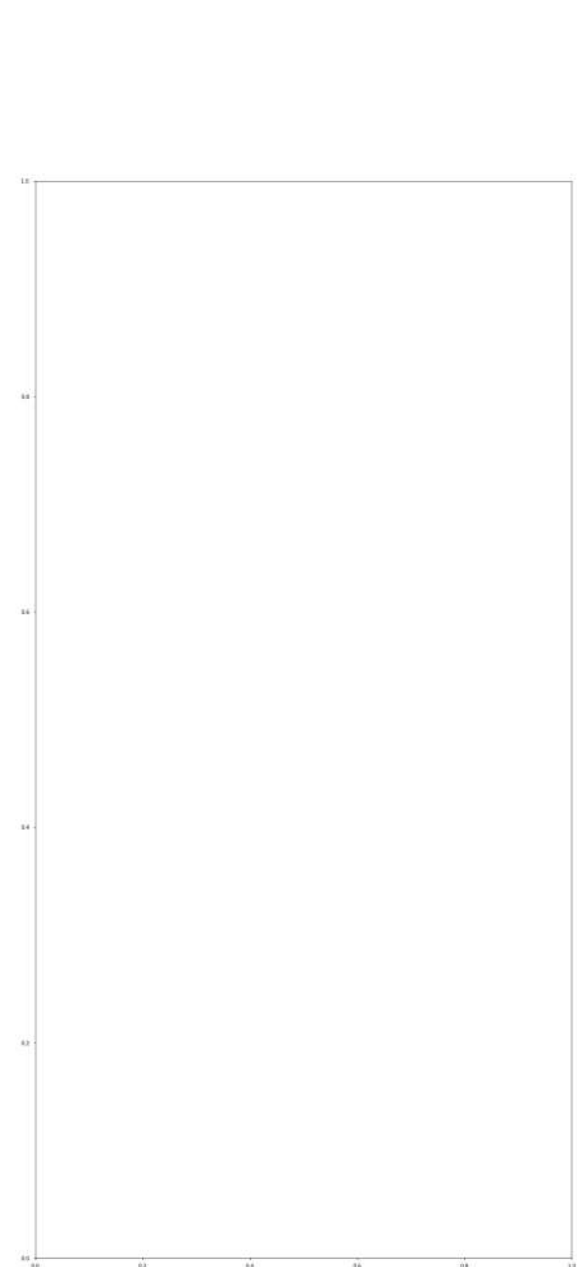




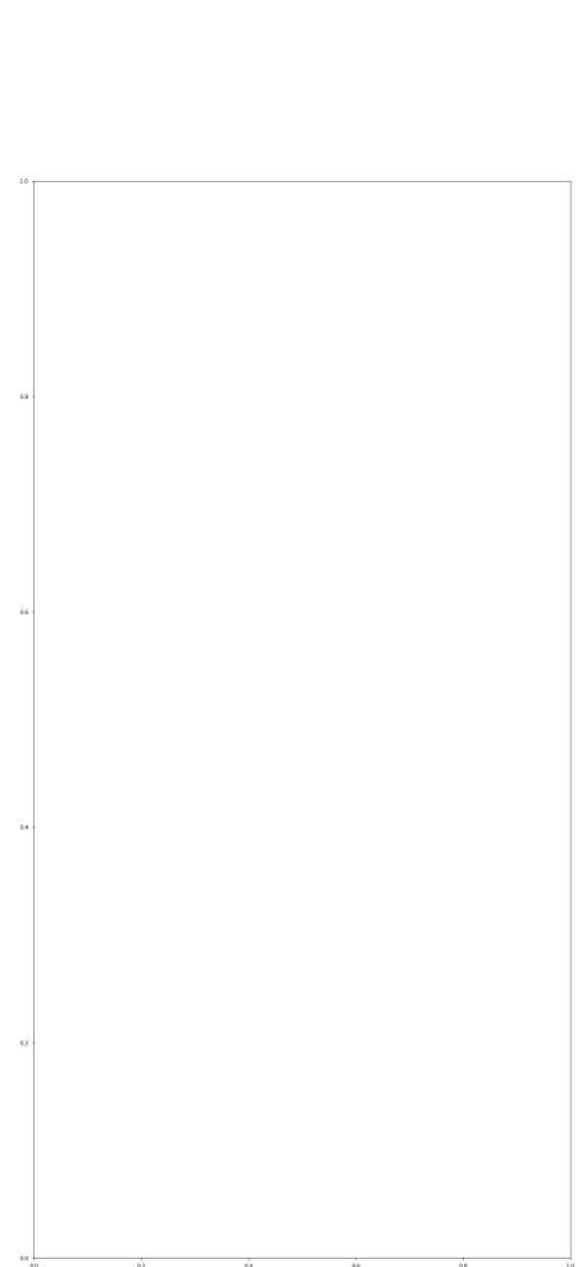
In [82]:
```
plot_labelwise_confusion_matrix_(val_true, val_preds, mlb.classes_[0:3])
```

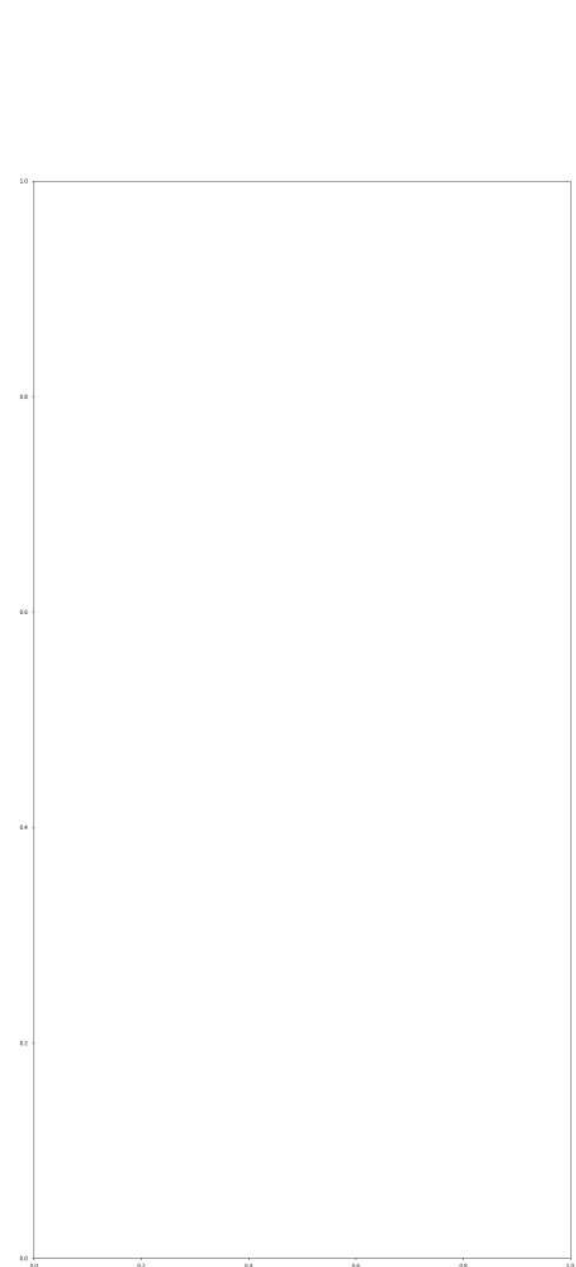












1 Additional Models Tried

1.1 Model 7: BERT Fine Tuned + ResNet + Class weights

1.1.1 Defining Model

In [ ]:
```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_input_resnet
```

In [ ]:
```
# Defining the layers
preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
bert_encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')

# Create the base model from the pre-trained model Resnet50
resnet50 = ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
)
```

In [ ]:
```
resnet50.trainable = False
```

In [ ]:
```
# Defining the model graph

# text input
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
encoder_inputs = preprocessing_layer(text_input)
encoder_outputs = bert_encoder(encoder_inputs)
text_pooled_output = encoder_outputs['pooled_output']

# image input
image_input = Input(shape=(224, 224, 3), name='image')
image = preprocess_input_resnet(image_input)
image_features = resnet50(image, training=False)
image_features = tf.keras.layers.GlobalAveragePooling2D()(image_features)
image_features = tf.keras.layers.Dense(768, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2(0.

# Cross-attention.
query_value_attention_seq = keras.layers.Attention(use_scale=True, dropout=0.2)(
    [image_features, text_pooled_output]
)

# Concatenate the projections and pass through the classification layer.
concatenated = keras.layers.Concatenate()([image_features, text_pooled_output])
concatenated = keras.layers.Concatenate()([concatenated, query_value_attention_seq])

# custom head
x = tf.keras.layers.Dense(128, kernel_regularizer=tf.keras.regularizers.l2(0.001))(concatenated)
outputs = tf.keras.layers.Dense(22, activation="sigmoid")(x)

model_res_bert_ft_cw = keras.Model(inputs = [image_input, text_input], outputs=[outputs], name="model_res_bert_ft_cw")
model_res_bert_ft_cw.summary()
```

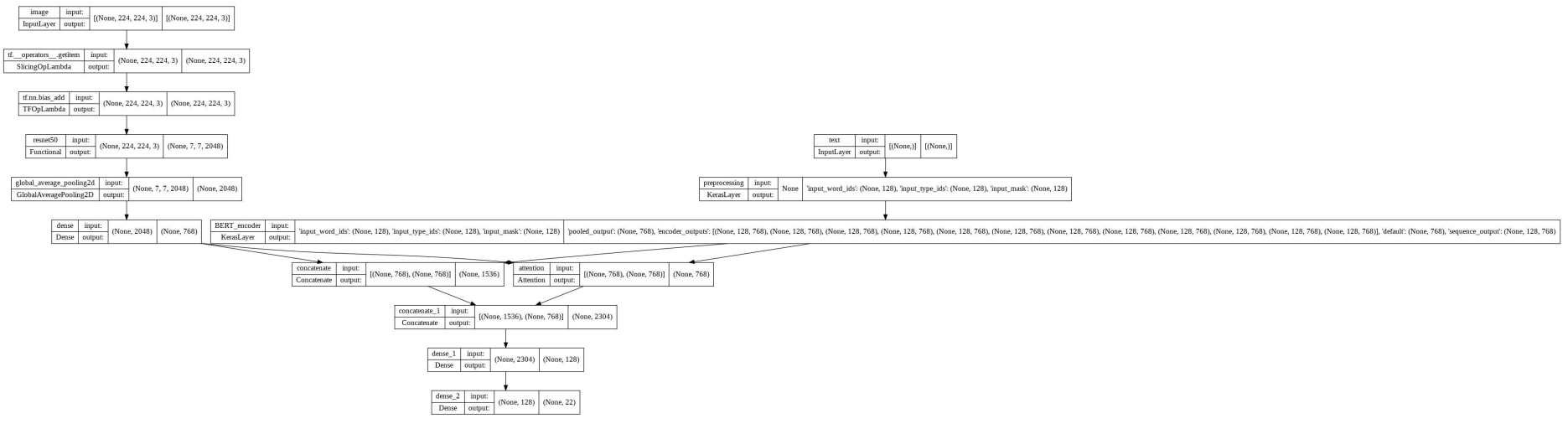
```
Model: "model_res_bert_ft_cw"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 image (InputLayer)             [(None, 224, 224, 3  0           []
                                )]

 tf.__operators__.getitem (Slic  (None, 224, 224, 3)  0          ['image[0][0]']
 ingOpLambda)

 tf.nn.bias_add (TFOpLambda)    (None, 224, 224, 3)  0           ['tf.__operators__.getitem[0][0]'
                                                                 ]

 resnet50 (Functional)          (None, 7, 7, 2048)   23587712    ['tf.nn.bias_add[0][0]']

 text (InputLayer)              [(None,)]            0           []

 global_average_pooling2d (Glob  (None, 2048)        0           ['resnet50[0][0]']
 alAveragePooling2D)

 preprocessing (KerasLayer)     {'input_word_ids':   0           ['text[0][0]']
                                (None, 128),
                                 'input_type_ids':
                                (None, 128),
                                 'input_mask': (Non
                                e, 128)}

 dense (Dense)                  (None, 768)          1573632     ['global_average_pooling2d[0][0]'
                                                                 ]

 BERT_encoder (KerasLayer)      {'pooled_output': (  109482241   ['preprocessing[0][0]',
                                None, 768),                       'preprocessing[0][1]',
                                 'encoder_outputs':               'preprocessing[0][2]']
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768)],
                                 'default': (None,
                                768),
                                 'sequence_output':
                                 (None, 128, 768)}

 concatenate (Concatenate)      (None, 1536)         0           ['dense[0][0]',
                                                                  'BERT_encoder[0][13]']

 attention (Attention)          (None, 768)          1           ['dense[0][0]',
                                                                  'BERT_encoder[0][13]']

 concatenate_1 (Concatenate)    (None, 2304)         0           ['concatenate[0][0]',
                                                                  'attention[0][0]']

 dense_1 (Dense)                (None, 128)          295040      ['concatenate_1[0][0]']
```

```
 dense_2 (Dense)                (None, 22)           2838        ['dense_1[0][0]']

==================================================================================================
Total params: 134,941,464
Trainable params: 111,353,751
Non-trainable params: 23,587,713
__________________________________________________________________________________________________
```

1.1.2 Plotting the model

In [ ]:
```
plot_model(model_res_bert_ft_cw, show_shapes=True)
```

Out[66]:



1.1.3 Training the model

In [ ]:
```
model_name = 'model_res_bert_ft_cw'
```

In [ ]:
```
!rm -rf /content/tensorboard_logs/models/model_res_bert_ft_cw
```

In [ ]:
```
epochs = 20
num_train_steps = STEPS_PER_EPOCH * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
adamw_optimizer = optimization.create_optimizer(init_lr=init_lr,
                                                num_train_steps=num_train_steps,
                                                num_warmup_steps=num_warmup_steps,
                                                optimizer_type='adamw')

adam_optimizer = tf.keras.optimizers.Adam(0.0001)
```

In [ ]:
```
data_mean = 0.0
data_std = 255.0 # for normalizing the data
TRAIN_BATCH_SIZE = 4
VAL_BATCH_SIZE = 4
TEST_BATCH_SIZE = 1

training_generator = DataGenerator(
                    train_df,
                    training_set_path,
                    batch_size=TRAIN_BATCH_SIZE,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    augment=False,
                    shuffle=True
                )
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=VAL_BATCH_SIZE,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22
                )

test_generator = DataGenerator(
                    test_df,
                    test_set_path,
                    batch_size=TEST_BATCH_SIZE,
                    shuffle=False,
                    n_classes=22
                )
```

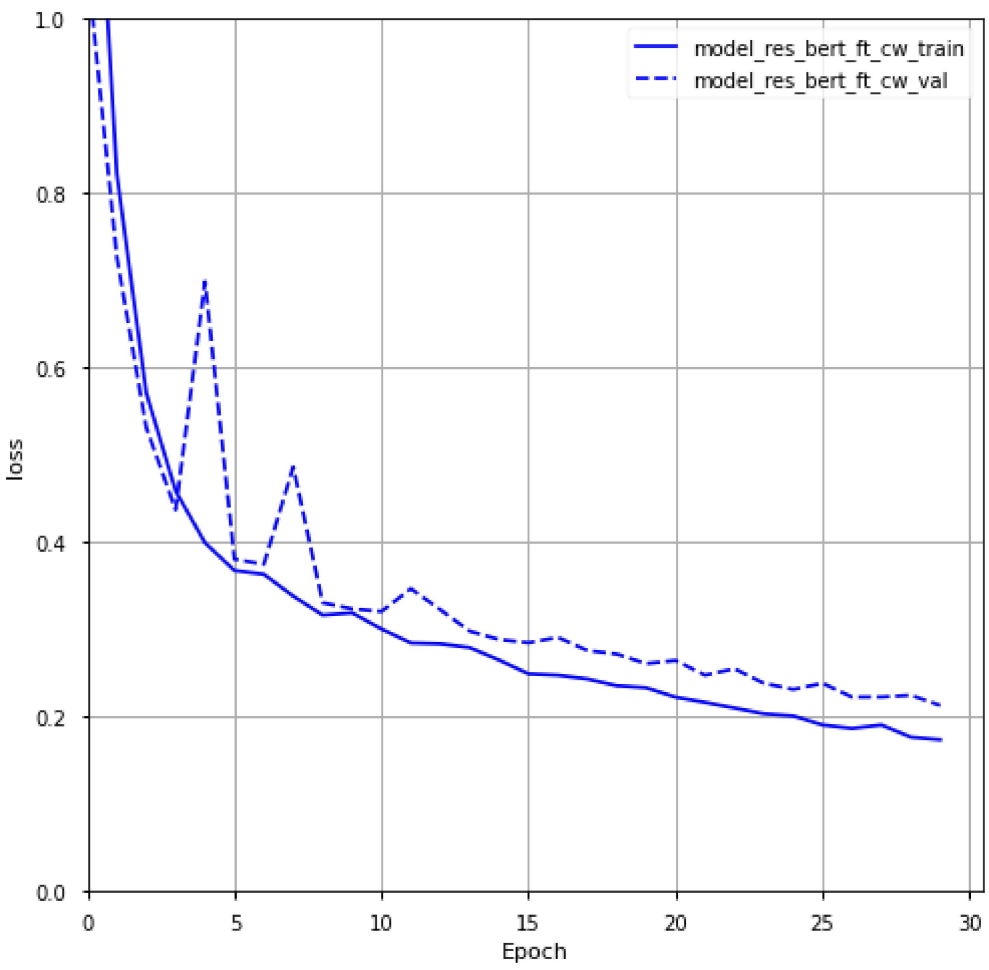
In [ ]:
```
m_histories[model_name], models[model_name] = compile_and_fit(
                model_res_bert_ft_cw,
                model_name,
                train_gen=training_generator,
                val_gen=validation_generator,
                max_epochs=30,
                optimizer=adam_optimizer,
                weights=class_weights_dict
            )
```

```
Epoch 1/30
171/171 [==============================] - ETA: 0s - loss: 1.3923 - binary_crossentropy: 0.4259 - precision: 0.2931 -
recall: 0.2774 - auc: 0.6953 - prc: 0.2384 - CategoricalAccuracy: 0.1067 - F1_macro: 0.0522 - F1_micro: 0.2026
Epoch 1: val_F1_macro improved from -inf to 0.03097, saving model to /content/tensorboard_logs/models/model_res_bert_f
t_cw/model_res_bert_ft_cw

WARNING:absl:Found untraced functions such as restored_function_body, restored_function_body, restored_function_body,
restored_function_body, restored_function_body while saving (showing 5 of 364). These functions will not be directly c
allable after loading.

171/171 [==============================] - 142s 655ms/step - loss: 1.3923 - binary_crossentropy: 0.4259 - precision:
0.2931 - recall: 0.2774 - auc: 0.6953 - prc: 0.2384 - CategoricalAccuracy: 0.1067 - F1_macro: 0.0522 - F1_micro: 0.202
6 - val_loss: 1.0604 - val_binary_crossentropy: 0.4071 - val_precision: 0.5255 - val_recall: 0.5954 - val_auc: 0.7850
- val_prc: 0.4084 - val_CategoricalAccuracy: 0.2667 - val_F1_macro: 0.0310 - val_F1_micro: 0.2661
Epoch 2/30
171/171 [==============================] - ETA: 0s - loss: 0.8224 - binary_crossentropy: 0.4002 - precision: 0.3429 -
recall: 0.2957 - auc: 0.7307 - prc: 0.2565 - CategoricalAccuracy: 0.1213 - F1_macro: 0.0632 - F1_micro: 0.2353
Epoch 2: val_F1_macro did not improve from 0.03097
171/171 [==============================] - 30s 174ms/step - loss: 0.8224 - binary_crossentropy: 0.4002 - precision: 0.
3429 - recall: 0.2957 - auc: 0.7307 - prc: 0.2565 - CategoricalAccuracy: 0.1213 - F1_macro: 0.0632 - F1_micro: 0.2353
```

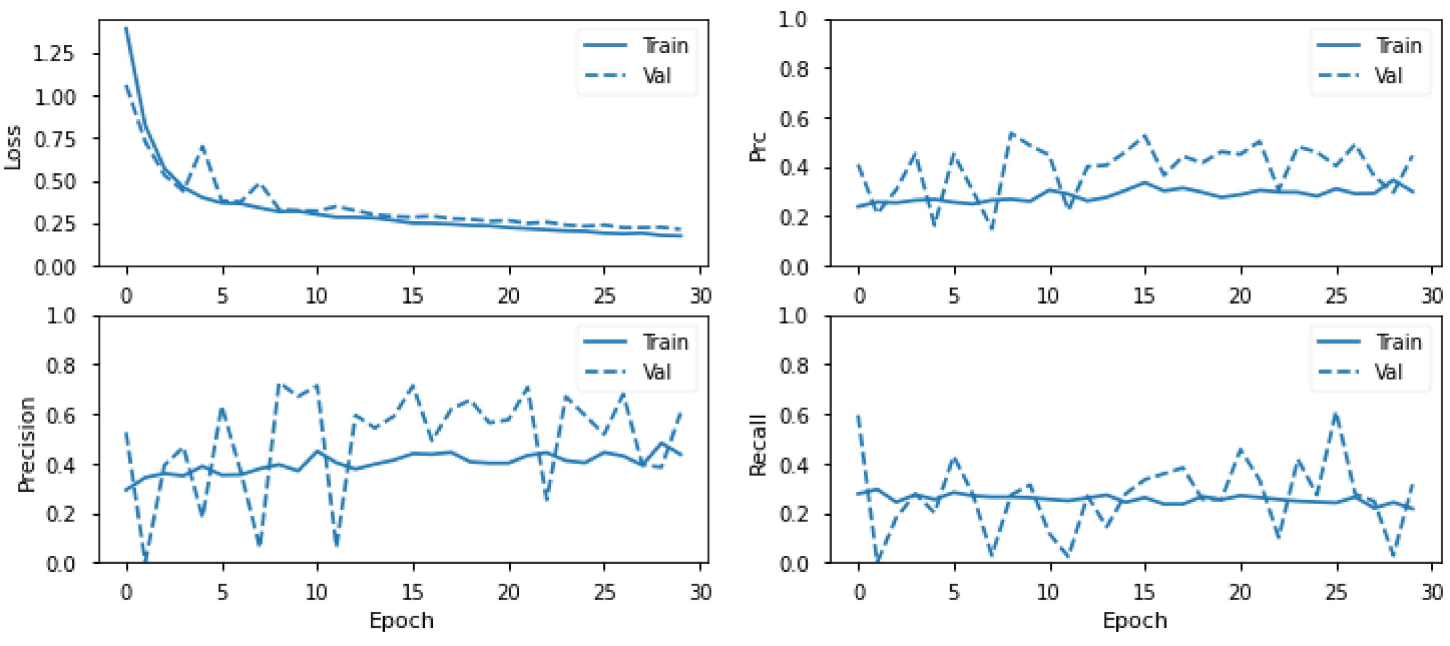
In [ ]:
```
download_files(m_histories, model_name)
```

1.1.4 Plot Metrics

In [ ]:
```
plotter(m_histories, metric="loss", ylim=[0, 1.0])
```



In [ ]:
```
plot_metrics(m_histories[model_name])
```



1.1.5 Evaluation on Validation Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert_ft_cw/model_res_bert_ft_cw')
```

In [ ]:
```
# Applying a threshold and converting the probability values in the
# output to binary boolean values (1 or 0)
threshold = 0.5
```

1.1.5.1 Storing the predictions to dataframe for plotting

In [ ]:
```
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=1,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22
                )
dev_pred = loaded_model.predict(validation_generator, verbose=1)
dev_prediction = (dev_pred > threshold).astype('int')
dev_prediction_df = prediction_df(dev_df, dev_prediction, mlb)
dev_prediction_df.head()
```

```
63/63 [==============================] - 8s 49ms/step
```

Out[78]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Smears, Doubt]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
1	[Smears, Loaded Language, Name calling/Labeling]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
2	[Smears, Transfer]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
3	[]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
4	[Smears, Exaggeration/Minimisation, Loaded Lan...	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...

In [ ]:
```
print("F1 macro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
print("F1 micro on Dev Set: ", round(metrics.f1_score(dev_prediction_df['actual_labels_mh'].values.tolist(), dev_predictio
```

```
F1 macro on Dev Set:  0.069
F1 micro on Dev Set:  0.511
```

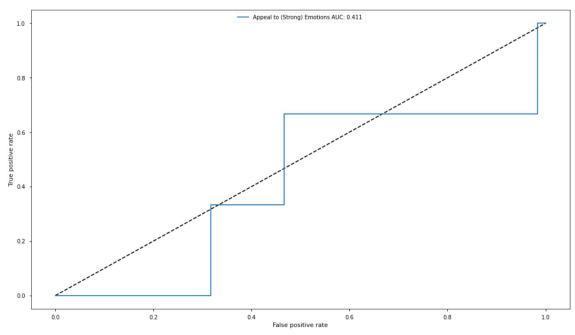
In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(dev_prediction_df['actual_labels_mh'].values.tolist(), de
perf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
perf_table.index = mlb.classes_
perf_table
```

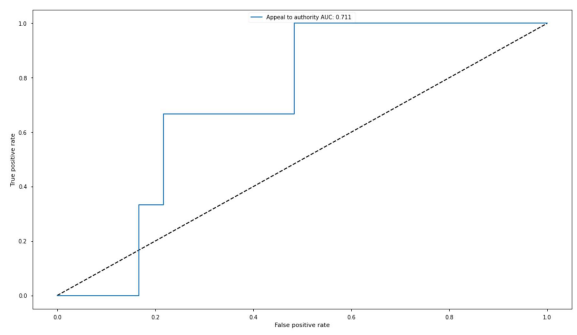
Out[80]:

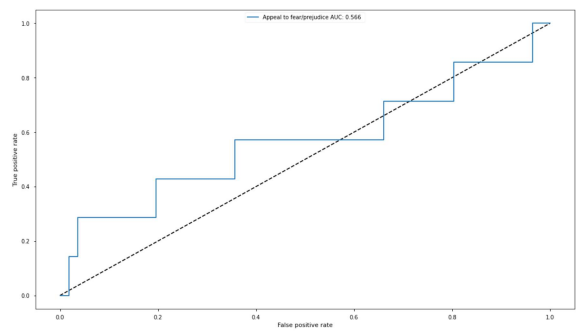
	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.0	0.000000	3
Appeal to authority	0.000000	0.0	0.000000	3
Appeal to fear/prejudice	0.000000	0.0	0.000000	7
Bandwagon	0.000000	0.0	0.000000	2
Black-and-white Fallacy/Dictatorship	0.000000	0.0	0.000000	0
Causal Oversimplification	0.000000	0.0	0.000000	1
Doubt	0.000000	0.0	0.000000	9
Exaggeration/Minimisation	0.000000	0.0	0.000000	8
Flag-waving	0.000000	0.0	0.000000	7
Glittering generalities (Virtue)	0.000000	0.0	0.000000	4
Loaded Language	0.507937	1.0	0.673684	32
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.0	0.000000	5
Name calling/Labeling	0.000000	0.0	0.000000	30
Obfuscation, Intentional vagueness, Confusion	0.000000	0.0	0.000000	0
Presenting Irrelevant Data (Red Herring)	0.000000	0.0	0.000000	0
Reductio ad hitlerum	0.000000	0.0	0.000000	1
Repetition	0.000000	0.0	0.000000	3
Slogans	0.000000	0.0	0.000000	3
Smears	0.746032	1.0	0.854545	47
Thought-terminating cliché	0.000000	0.0	0.000000	1
Transfer	0.000000	0.0	0.000000	11
Whataboutism	0.000000	0.0	0.000000	6

In [ ]:
```
val_preds = np.array([]).reshape((0,22))
val_true = np.array([]).reshape((0,22))
for i in range(validation_generator.__len__()):
    x,y = validation_generator.__getitem__(i)
    val_true = np.append(val_true,y,axis=0)
    val_preds = np.append(val_preds,models[model_name](x).numpy().reshape((1,22)),axis=0)
```

In [ ]:
```
plot_roc_curve(val_true, val_preds, mlb.classes_[0:3])
```

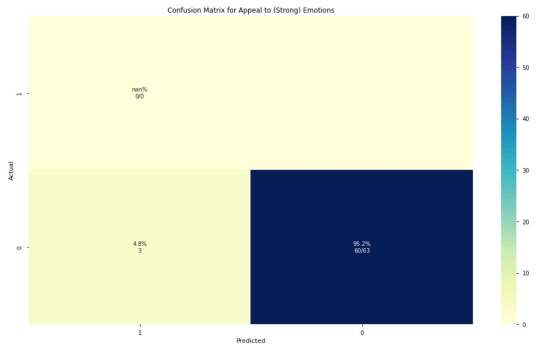


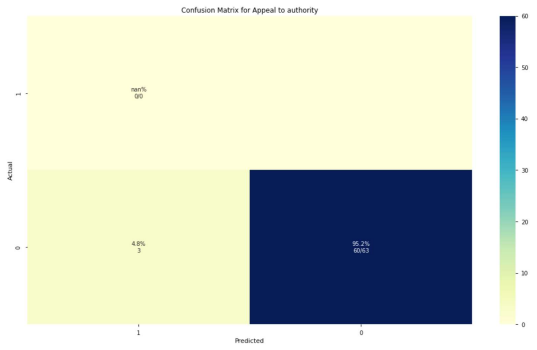


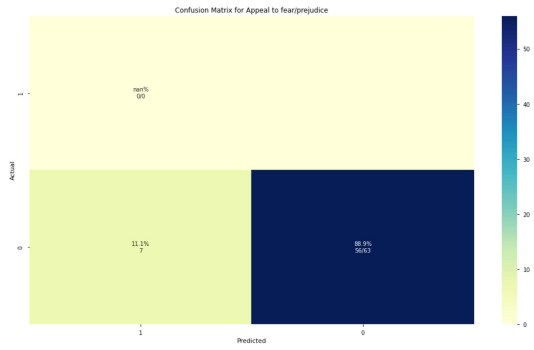


In [ ]:
```
plot_labelwise_confusion_matrix_(val_true, val_preds, mlb.classes_[0:3])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: invalid value encountered in true_divid
e
  # Remove the CWD from sys.path while we load stuff.
```







1.2 Model 7: DeBERTa + ResNet50

1.2.1 Defining Model

In [ ]:
```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input

from transformers import DebertaTokenizer, TFDebertaModel
```

In [ ]:
```
# Defining the layers

deberta_tokenizer = DebertaTokenizer.from_pretrained("microsoft/deberta-large", return_tensors="tf")
deberta_model = TFDebertaModel.from_pretrained("microsoft/deberta-large")

# Create the base model from the pre-trained model Resnet50
resnet50 = ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
)
```

```
All model checkpoint layers were used when initializing TFDebertaModel.

All the layers of TFDebertaModel were initialized from the model checkpoint at microsoft/deberta-large.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFDebertaModel for p
redictions without further training.
```

In [ ]:
```
# Make deberta_model layers untrainable
# for layer in deberta_model.layers:
#     layer.trainable = False

resnet50.trainable = False
```

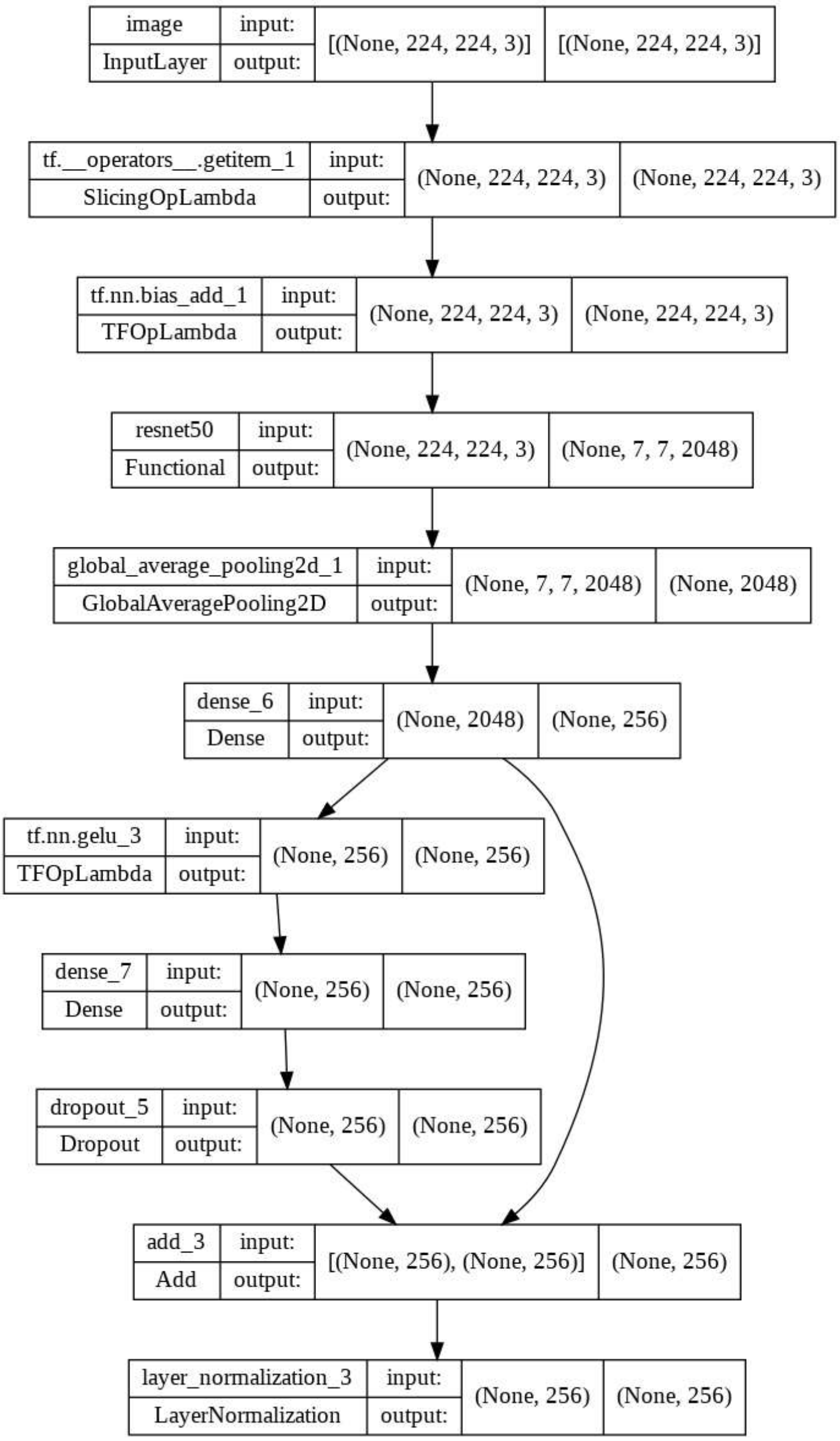
In [ ]:
```
def project_embeddings(embeddings, num_projection_layers, projection_dims, dropout_rate):

    projected_embeddings = keras.layers.Dense(units=projection_dims)(embeddings)
    for _ in range(num_projection_layers):
        x = tf.nn.gelu(projected_embeddings)
        x = keras.layers.Dense(projection_dims, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l
        x = keras.layers.Dropout(dropout_rate)(x)
        x = keras.layers.Add()([projected_embeddings, x])
        projected_embeddings = keras.layers.LayerNormalization()(x)

    return projected_embeddings
```

In [ ]:
```python
def encode_text(text):
  """
    Function to convert the outputs of hugging face tokenizer
    to tensors
  """
  inputs = [tf.compat.as_str(x) for x in text.numpy().tolist()]
  tokenized = deberta_tokenizer(inputs,
    return_tensors='tf',
    add_special_tokens=True,
    max_length=110,
    padding='max_length',
    truncation=True)
  return tokenized['input_ids'], tokenized['attention_mask']

def initialize_text_encoder(deberta_model, num_projection_layers, projection_dims, dropout_rate):

  text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
  input_ids, attention_mask = tf.py_function(encode_text, inp=[text_input], Tout=[tf.int32, tf.int32])

  input_ids = tf.ensure_shape(input_ids, [None, 110])
  attention_mask = tf.ensure_shape(attention_mask, [None, 110])

  encoder_outputs = deberta_model(input_ids, attention_mask)
  sequence_output = encoder_outputs.last_hidden_state

  bi_lstm = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True))(sequence_output)
  avg_pool = tf.keras.layers.GlobalAveragePooling1D()(bi_lstm)
  max_pool = tf.keras.layers.GlobalMaxPool1D()(bi_lstm)
  concat = tf.keras.layers.concatenate([avg_pool, max_pool])
  dropout = tf.keras.layers.Dropout(0.3)(concat)


  outputs = project_embeddings(dropout, num_projection_layers, projection_dims, dropout_rate)

  return keras.Model(text_input, outputs, name='text_encoder')

def initialize_image_encoder(resnet50, num_projection_layers, projection_dims, dropout_rate):

  image_input = Input(shape=(224, 224, 3), name='image')
  image = preprocess_input(image_input)
  image_features = resnet50(image, training=False)
  image_features = tf.keras.layers.GlobalAveragePooling2D()(image_features)

  outputs = project_embeddings(image_features, num_projection_layers, projection_dims, dropout_rate)

  return keras.Model(image_input, outputs, name='image_encoder')

def multimodal_model(resnet50, deberta_model, num_projection_layers, projection_dims, dropout_rate):
  text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
  image_input = Input(shape=(224, 224, 3), name='image')

  text_encoder = initialize_text_encoder(deberta_model, num_projection_layers=1, projection_dims=256, dropout_rate=0.2)
  image_encoder = initialize_image_encoder(resnet50, num_projection_layers=1, projection_dims=256, dropout_rate=0.2)

  text_embedding = text_encoder(text_input)
  image_embedding = image_encoder(image_input)

  query_value_attention_seq = keras.layers.Attention(use_scale=True, dropout=0.2)(
    [image_embedding, text_embedding])

  # Concatenate the projections and pass through the classification layer.
  concatenated = keras.layers.Concatenate()([image_embedding, text_embedding])
  feature_fusion = keras.layers.Concatenate()([concatenated, query_value_attention_seq])

  outputs = tf.keras.layers.Dense(22, activation="sigmoid")(feature_fusion)

  return keras.Model(inputs = [image_input, text_input], outputs=[outputs], name="model_res_deberta")
```

In [ ]:
```python
text_encoder = initialize_text_encoder(deberta_model, num_projection_layers=1, projection_dims=256, dropout_rate=0.2)
# plot_model(text_encoder, show_shapes=True)
```

In [ ]:
```
image_encoder = initialize_image_encoder(resnet50, num_projection_layers=1, projection_dims=256, dropout_rate=0.2)
plot_model(image_encoder, show_shapes=True)
```

Out[70]:



In [ ]:
```
model_res_deberta = multimodal_model(resnet50, deberta_model, num_projection_layers=1, projection_dims=256, dropout_rate=0
model_res_deberta.summary()
```

```
Model: "model_res_deberta"
```

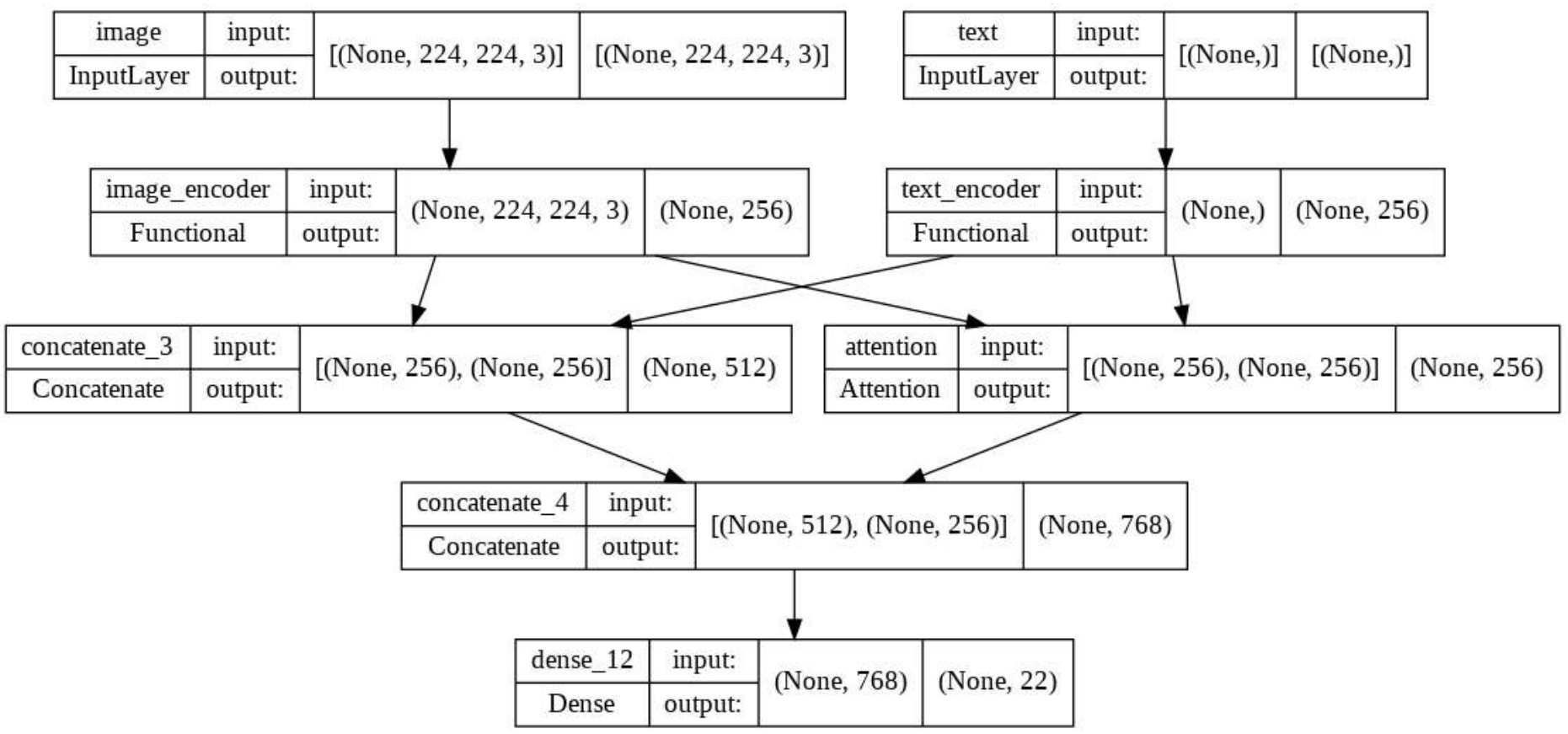
Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	[(None, 224, 224, 3)]	0	[]
text (InputLayer)	[(None,)]	0	[]
image_encoder (Functional)	(None, 256)	24178560	['image[0][0]']
text_encoder (Functional)	(None, 256)	405852672	['text[0][0]']
concatenate_3 (Concatenate)	(None, 512)	0	['image_encoder[0][0]', 'text_encoder[0][0]']
attention (Attention)	(None, 256)	1	['image_encoder[0][0]', 'text_encoder[0][0]']
concatenate_4 (Concatenate)	(None, 768)	0	['concatenate_3[0][0]', 'attention[0][0]']
dense_12 (Dense)	(None, 22)	16918	['concatenate_4[0][0]']

```
Total params: 430,048,151
Trainable params: 406,460,439
Non-trainable params: 23,587,712
```

1.2.2 Plotting the model

In [ ]:
```
plot_model(model_res_deberta, show_shapes=True)
```

Out[72]:



1.2.3 Training the model

In [ ]:
```
model_name = 'model_res_deberta'
```

In [ ]:
```
!rm -rf /content/tensorboard_logs/models/model_res_deberta
```

In [ ]:
```
data_mean = 0.0
data_std = 255.0 # for normalizing the data
# selecting a higher batch size, to ensure each batch includes data from
# a wide variety of classes
TRAIN_BATCH_SIZE = 4
VAL_BATCH_SIZE = 4
TEST_BATCH_SIZE = 1

training_generator = DataGenerator(
                    train_df,
                    training_set_path,
                    batch_size=TRAIN_BATCH_SIZE,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22,
                    augment=False,
                    shuffle=True
                )
validation_generator = DataGenerator(
                    dev_df,
                    dev_set_path,
                    batch_size=VAL_BATCH_SIZE,
                    data_mean=data_mean,
                    data_std=data_std,
                    n_classes=22
                )

test_generator = DataGenerator(
                    test_df,
                    test_set_path,
                    batch_size=TEST_BATCH_SIZE,
                    shuffle=False,
                    n_classes=22
                )
```

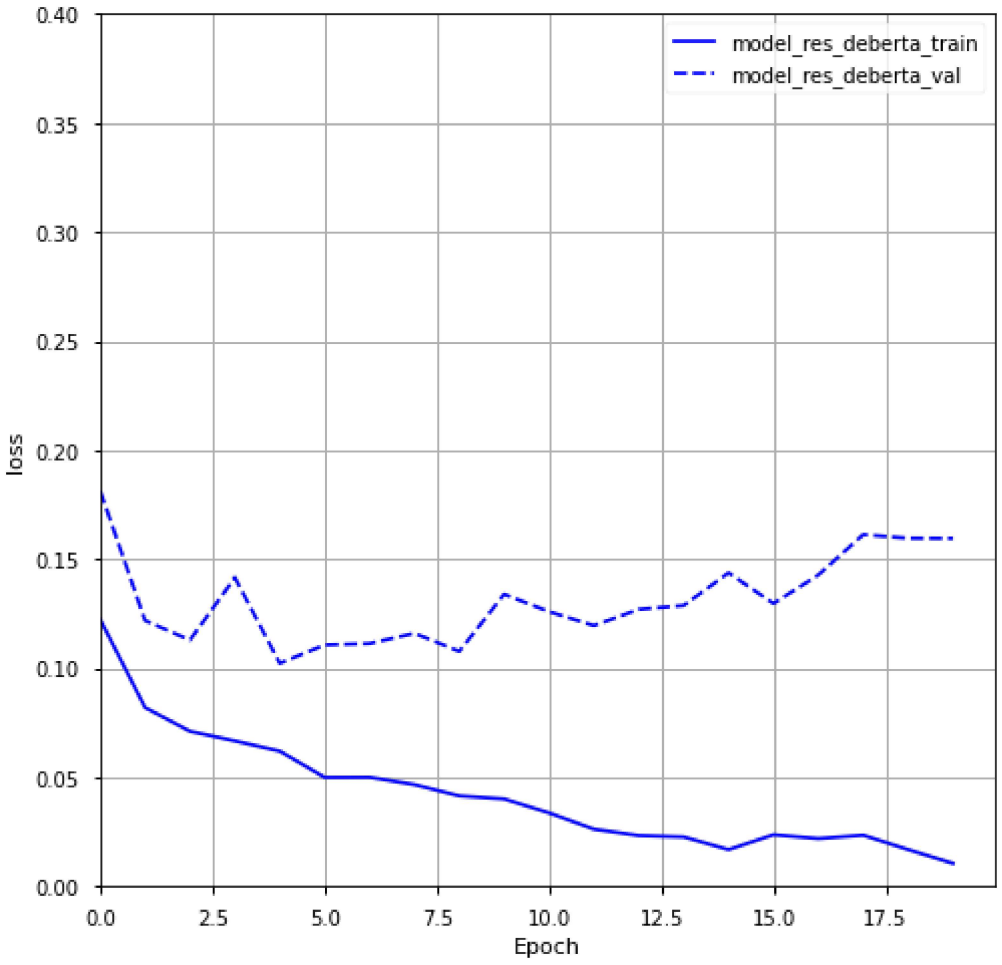
In [ ]:
```
optimizer = tf.keras.optimizers.Adam(3e-5)
```

In [ ]:
```
m_histories[model_name], models[model_name] = compile_and_fit(
                model_res_deberta,
                model_name,
                train_gen=training_generator,
                val_gen=validation_generator,
                max_epochs=20,
                optimizer=optimizer,
                weights=None
            )
```

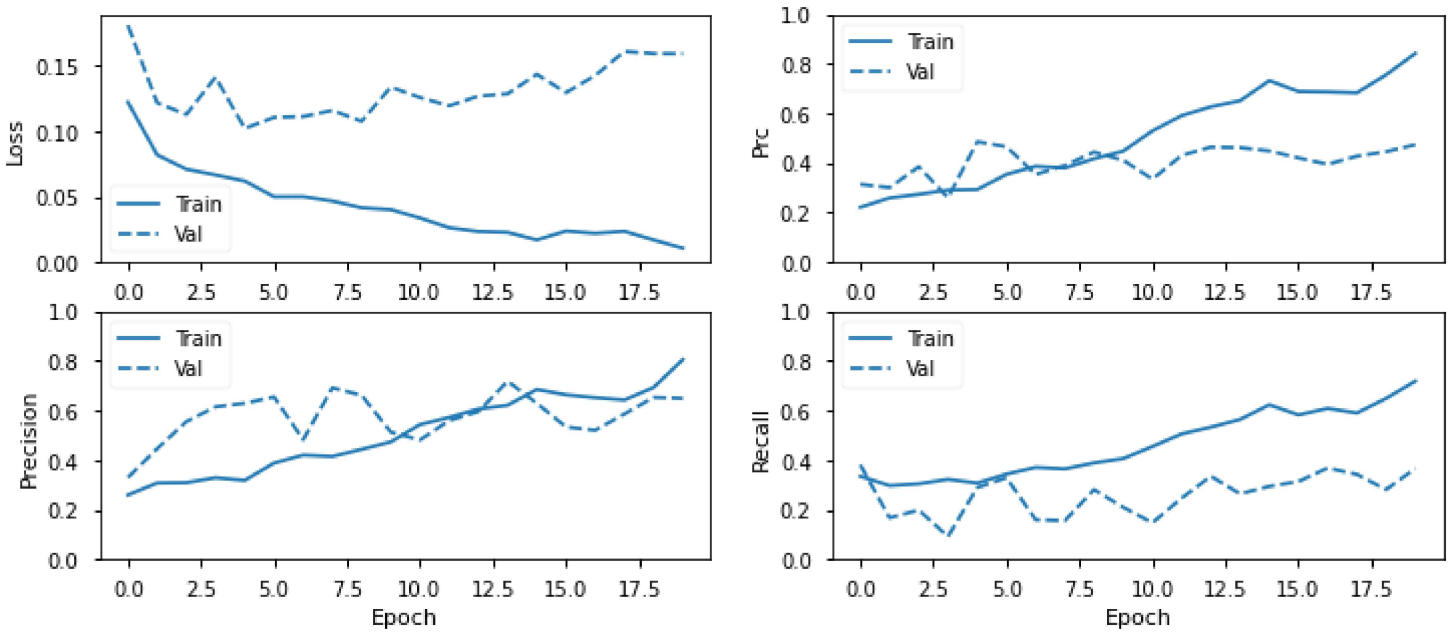
```
Epoch 1/20
171/171 [==============================] - ETA: 0s - loss: 1.1258 - binary_crossentropy: 0.4079 - precision: 0.3433 - re
call: 0.3814 - auc: 0.7362 - prc: 0.2848 - CategoricalAccuracy: 0.1608 - F1_macro: 0.0601 - F1_micro: 0.2722
```

1.2.4 Plot Metrics

In [ ]:
```
plotter(m_histories, metric="loss", ylim=[0, 0.4])
```



In [ ]:
```
plot_metrics(m_histories[model_name])
```



2 Test Predictions

2.0.1 Model 1 - Predictions on Test Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_inc_bert_768/model_inc_bert_768')
test_pred = loaded_model.predict(test_generator)
```

In [ ]:
```
threshold = 0.5
prediction = (test_pred > threshold).astype('int')
```

In [ ]:
```
test_prediction_df = prediction_df(test_df, prediction, mlb)
test_prediction_df.head()
```

Out[92]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Name calling/Labeling, Slogans, Smears, Trans...	[Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
1	[Appeal to (Strong) Emotions, Appeal to fear/p...	[Loaded Language, Name calling/Labeling, Smears]	[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
2	[Doubt, Loaded Language, Name calling/Labeling]	[Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	[Exaggeration/Minimisation, Loaded Language]	[Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	[Doubt]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...

In [ ]:
```
print("F1 macro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
print("F1 micro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
```

```
F1 macro on Test Set:  0.073
F1 micro on Test Set:  0.365
```

In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(test_prediction_df['actual_labels_mh'].values.tolist(), t
prf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
prf_table.index = mlb.classes_
prf_table
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[94]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	19
Appeal to authority	0.000000	0.000000	0.000000	13
Appeal to fear/prejudice	0.000000	0.000000	0.000000	18
Bandwagon	0.000000	0.000000	0.000000	1
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	7
Causal Oversimplification	0.000000	0.000000	0.000000	4
Doubt	0.000000	0.000000	0.000000	41
Exaggeration/Minimisation	0.000000	0.000000	0.000000	31
Flag-waving	0.000000	0.000000	0.000000	12
Glittering generalities (Virtue)	0.000000	0.000000	0.000000	24
Loaded Language	0.795455	0.350000	0.486111	100
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	3
Name calling/Labeling	0.545455	0.461538	0.500000	65
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	2
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	5
Reductio ad hitlerum	0.000000	0.000000	0.000000	7
Repetition	0.000000	0.000000	0.000000	1
Slogans	0.000000	0.000000	0.000000	22
Smears	0.540146	0.704762	0.611570	105
Thought-terminating cliché	0.000000	0.000000	0.000000	6
Transfer	0.000000	0.000000	0.000000	23
Whataboutism	0.000000	0.000000	0.000000	14

2.0.2 Model 2 - Predictions on Test Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert_concat/model_res_bert_concat')
test_pred = models[model_name].predict(test_generator)
```

In [ ]:
```
threshold = 0.5
prediction = (test_pred > threshold).astype('int')
```

In [ ]:
```
test_prediction_df = prediction_df(test_df, prediction, mlb)
test_prediction_df.head()
```

Out[90]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Name calling/Labeling, Slogans, Smears, Trans...	[Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	[Appeal to (Strong) Emotions, Appeal to fear/p...	[Loaded Language, Smears]	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
2	[Doubt, Loaded Language, Name calling/Labeling]	[Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	[Exaggeration/Minimisation, Loaded Language]	[Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	[Doubt]	[Glittering generalities (Virtue), Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...

In [ ]:
```
print("F1 macro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
print("F1 micro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
```

```
F1 macro on Test Set:  0.094
F1 micro on Test Set:  0.395
```

In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(test_prediction_df['actual_labels_mh'].values.tolist(), t
prf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
prf_table.index = mlb.classes_
prf_table
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[92]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	19
Appeal to authority	0.000000	0.000000	0.000000	13
Appeal to fear/prejudice	0.000000	0.000000	0.000000	18
Bandwagon	0.000000	0.000000	0.000000	1
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	7
Causal Oversimplification	0.000000	0.000000	0.000000	4
Doubt	1.000000	0.024390	0.047619	41
Exaggeration/Minimisation	0.000000	0.000000	0.000000	31
Flag-waving	1.000000	0.083333	0.153846	12
Glittering generalities (Virtue)	0.190476	0.166667	0.177778	24
Loaded Language	0.762712	0.450000	0.566038	100
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	3
Name calling/Labeling	0.681818	0.230769	0.344828	65
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	2
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	5
Reductio ad hitlerum	0.000000	0.000000	0.000000	7
Repetition	0.000000	0.000000	0.000000	1
Slogans	0.000000	0.000000	0.000000	22
Smears	0.541176	0.876190	0.669091	105
Thought-terminating cliché	0.000000	0.000000	0.000000	6
Transfer	0.181818	0.086957	0.117647	23
Whataboutism	0.000000	0.000000	0.000000	14

2.0.3 Model 3 - Predictions on Test Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert/model_res_bert')
test_pred = models[model_name].predict(test_generator)
```

In [ ]:
```
threshold = 0.5
prediction = (test_pred > threshold).astype('int')
```

In [ ]:
```
test_prediction_df = prediction_df(test_df, prediction, mlb)
test_prediction_df.head()
```

Out[89]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Name calling/Labeling, Slogans, Smears, Trans...	[Appeal to (Strong) Emotions, Name calling/Lab...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
1	[Appeal to (Strong) Emotions, Appeal to fear/p...	[Loaded Language, Name calling/Labeling, Smears]	[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
2	[Doubt, Loaded Language, Name calling/Labeling]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
3	[Exaggeration/Minimisation, Loaded Language]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
4	[Doubt]	[Loaded Language, Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...

In [ ]:
```
print("F1 macro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
print("F1 micro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
```

```
F1 macro on Test Set:  0.105
F1 micro on Test Set:  0.478
```

In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(test_prediction_df['actual_labels_mh'].values.tolist(), t
prf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
prf_table.index = mlb.classes_
prf_table
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[91]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.200000	0.052632	0.083333	19
Appeal to authority	0.000000	0.000000	0.000000	13
Appeal to fear/prejudice	0.666667	0.111111	0.190476	18
Bandwagon	0.000000	0.000000	0.000000	1
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	7
Causal Oversimplification	0.000000	0.000000	0.000000	4
Doubt	0.600000	0.073171	0.130435	41
Exaggeration/Minimisation	0.000000	0.000000	0.000000	31
Flag-waving	0.000000	0.000000	0.000000	12
Glittering generalities (Virtue)	0.000000	0.000000	0.000000	24
Loaded Language	0.605634	0.860000	0.710744	100
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	3
Name calling/Labeling	0.409091	0.830769	0.548223	65
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	2
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	5
Reductio ad hitlerum	0.000000	0.000000	0.000000	7
Repetition	0.000000	0.000000	0.000000	1
Slogans	0.000000	0.000000	0.000000	22
Smears	0.544872	0.809524	0.651341	105
Thought-terminating cliché	0.000000	0.000000	0.000000	6
Transfer	0.000000	0.000000	0.000000	23
Whataboutism	0.000000	0.000000	0.000000	14

2.0.4 Model 4 - Predictions on Test Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert/model_res_bert')
test_pred = models[model_name].predict(test_generator)
```

In [ ]:
```
threshold = 0.5
prediction = (test_pred > threshold).astype('int')
```

In [ ]:
```
test_prediction_df = prediction_df(test_df, prediction, mlb)
test_prediction_df.head()
```

Out[92]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Name calling/Labeling, Slogans, Smears, Trans...	[Name calling/Labeling]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
1	[Appeal to (Strong) Emotions, Appeal to fear/p...	[Loaded Language, Name calling/Labeling, Smears]	[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
2	[Doubt, Loaded Language, Name calling/Labeling]	[Transfer]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	[Exaggeration/Minimisation, Loaded Language]	[Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	[Doubt]	[Name calling/Labeling, Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...

In [ ]:
```
print("F1 macro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
print("F1 micro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
```

```
F1 macro on Test Set:  0.087
F1 micro on Test Set:  0.401
```

In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(test_prediction_df['actual_labels_mh'].values.tolist(), t
prf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
prf_table.index = mlb.classes_
prf_table
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[95]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.200000	0.052632	0.083333	19
Appeal to authority	0.000000	0.000000	0.000000	13
Appeal to fear/prejudice	0.000000	0.000000	0.000000	18
Bandwagon	0.000000	0.000000	0.000000	1
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	7
Causal Oversimplification	0.000000	0.000000	0.000000	4
Doubt	0.000000	0.000000	0.000000	41
Exaggeration/Minimisation	0.000000	0.000000	0.000000	31
Flag-waving	0.000000	0.000000	0.000000	12
Glittering generalities (Virtue)	0.000000	0.000000	0.000000	24
Loaded Language	0.781250	0.500000	0.609756	100
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	3
Name calling/Labeling	0.428571	0.646154	0.515337	65
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	2
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	5
Reductio ad hitlerum	0.000000	0.000000	0.000000	7
Repetition	0.000000	0.000000	0.000000	1
Slogans	0.000000	0.000000	0.000000	22
Smears	0.616071	0.657143	0.635945	105
Thought-terminating cliché	0.000000	0.000000	0.000000	6
Transfer	0.166667	0.043478	0.068966	23
Whataboutism	0.000000	0.000000	0.000000	14

2.0.5 Model 5 - Predictions on Test Set

In [ ]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bert_gru/model_res_bert_gru')
test_pred = loaded_model.predict(test_generator)
```

In [ ]:
```
threshold = 0.5
prediction = (test_pred > threshold).astype('int')
```

In [ ]:
```
test_prediction_df = prediction_df(test_df, prediction, mlb)
test_prediction_df.head()
```

Out[83]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Name calling/Labeling, Slogans, Smears, Trans...	[Smears, Transfer]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	[Appeal to (Strong) Emotions, Appeal to fear/p...	[Loaded Language, Name calling/Labeling, Smears]	[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
2	[Doubt, Loaded Language, Name calling/Labeling]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
3	[Exaggeration/Minimisation, Loaded Language]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
4	[Doubt]	[Loaded Language, Name calling/Labeling]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...

In [ ]:
```
print("F1 macro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
print("F1 micro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
```

```
F1 macro on Test Set:  0.107
F1 micro on Test Set:  0.457
```

In [ ]:
```
precision, recall, f1, support = precision_recall_fscore_support(test_prediction_df['actual_labels_mh'].values.tolist(), t
prf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
prf_table.index = mlb.classes_
prf_table
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-
score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[85]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	19
Appeal to authority	0.000000	0.000000	0.000000	13
Appeal to fear/prejudice	0.000000	0.000000	0.000000	18
Bandwagon	0.000000	0.000000	0.000000	1
Black-and-white Fallacy/Dictatorship	0.000000	0.000000	0.000000	7
Causal Oversimplification	0.000000	0.000000	0.000000	4
Doubt	0.666667	0.048780	0.090909	41
Exaggeration/Minimisation	0.500000	0.032258	0.060606	31
Flag-waving	0.000000	0.000000	0.000000	12
Glittering generalities (Virtue)	0.400000	0.083333	0.137931	24
Loaded Language	0.666667	0.800000	0.727273	100
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	3
Name calling/Labeling	0.533333	0.492308	0.512000	65
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	2
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	5
Reductio ad hitlerum	0.000000	0.000000	0.000000	7
Repetition	0.000000	0.000000	0.000000	1
Slogans	0.000000	0.000000	0.000000	22
Smears	0.580153	0.723810	0.644068	105
Thought-terminating cliché	0.000000	0.000000	0.000000	6
Transfer	0.272727	0.130435	0.176471	23
Whataboutism	0.000000	0.000000	0.000000	14

2.0.6 Model 6 - Predictions on Test Set (Final Model)

In [84]:
```
loaded_model = tf.keras.models.load_model('/content/tensorboard_logs/models/model_res_bertft_gru/model_res_bertft_gru')
test_pred = loaded_model.predict(test_generator)
```

In [85]:
```
threshold = 0.5
prediction = (test_pred > threshold).astype('int')
```

In [86]:
```
test_prediction_df = prediction_df(test_df, prediction, mlb)
test_prediction_df.head()
```

Out[86]:

	actual	model1_predicted	actual_labels_mh	predicted_labels_mh
0	[Name calling/Labeling, Slogans, Smears, Trans...	[Flag-waving, Smears]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ...
1	[Appeal to (Strong) Emotions, Appeal to fear/p...	[Loaded Language, Name calling/Labeling]	[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
2	[Doubt, Loaded Language, Name calling/Labeling]	[Loaded Language, Name calling/Labeling, Trans...	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, ...
3	[Exaggeration/Minimisation, Loaded Language]	[Loaded Language, Smears]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
4	[Doubt]	[Glittering generalities (Virtue), Smears]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...

In [87]:
```
print("F1 macro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
print("F1 micro on Test Set: ", round(metrics.f1_score(test_prediction_df['actual_labels_mh'].values.tolist(), test_predic
```

```
F1 macro on Test Set:  0.212
F1 micro on Test Set:  0.52
```

In [88]:
```
precision, recall, f1, support = precision_recall_fscore_support(test_prediction_df['actual_labels_mh'].values.tolist(), t
prf_table = pd.DataFrame({'precision': precision, 'recall': recall, 'f1': f1, 'support': support})
prf_table.index = mlb.classes_
prf_table
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-
score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

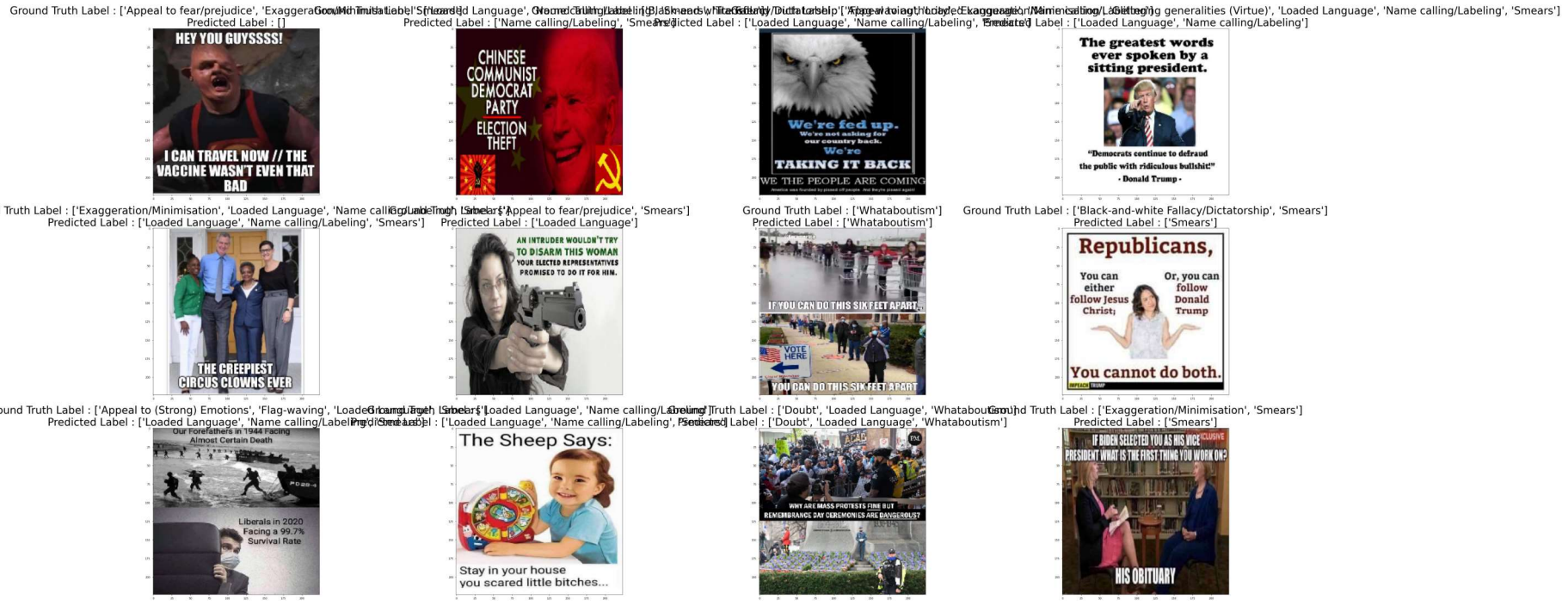
Out[88]:

	precision	recall	f1	support
Appeal to (Strong) Emotions	0.000000	0.000000	0.000000	19
Appeal to authority	0.600000	0.230769	0.333333	13
Appeal to fear/prejudice	0.285714	0.111111	0.160000	18
Bandwagon	0.000000	0.000000	0.000000	1
Black-and-white Fallacy/Dictatorship	1.000000	0.142857	0.250000	7
Causal Oversimplification	0.400000	0.500000	0.444444	4
Doubt	0.500000	0.073171	0.127660	41
Exaggeration/Minimisation	0.400000	0.064516	0.111111	31
Flag-waving	0.428571	0.250000	0.315789	12
Glittering generalities (Virtue)	0.375000	0.125000	0.187500	24
Loaded Language	0.813725	0.830000	0.821782	100
Misrepresentation of Someone's Position (Straw Man)	0.000000	0.000000	0.000000	3
Name calling/Labeling	0.625000	0.692308	0.656934	65
Obfuscation, Intentional vagueness, Confusion	0.000000	0.000000	0.000000	2
Presenting Irrelevant Data (Red Herring)	0.000000	0.000000	0.000000	5
Reductio ad hitlerum	0.000000	0.000000	0.000000	7
Repetition	0.000000	0.000000	0.000000	1
Slogans	0.428571	0.136364	0.206897	22
Smears	0.576642	0.752381	0.652893	105
Thought-terminating cliché	0.000000	0.000000	0.000000	6
Transfer	0.200000	0.043478	0.071429	23
Whataboutism	0.600000	0.214286	0.315789	14

3 Visualizing Sample Predictions on the Test Set

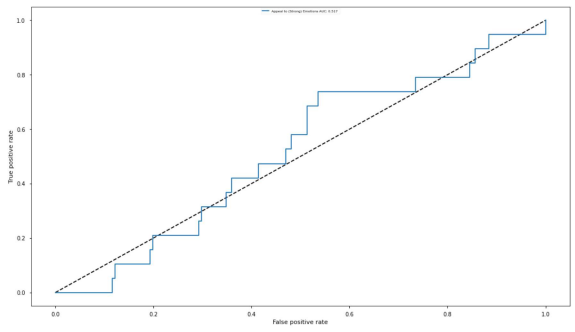
In [89]:
```
rand_ind = np.random.randint(0, len(test_df), 12)
```

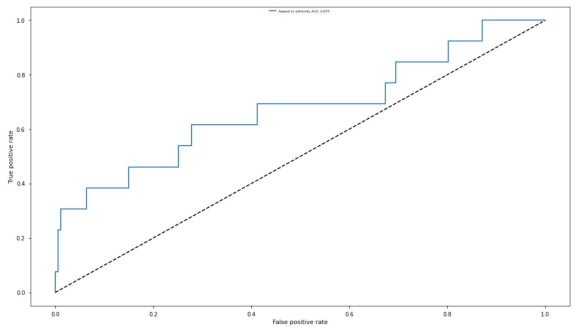
In [90]:
```
i = 0
count = 0
fig,axs = plt.subplots(3,4,figsize=(80,40))
for i, ind in enumerate(rand_ind):
  pred = test_prediction_df.iloc[ind]['model1_predicted']
  actual = test_prediction_df.iloc[ind]['actual']
  img = tf.keras.utils.load_img(os.path.join(test_set_path, test_df.iloc[ind]['image']), grayscale=False, color_mode='rgb')
  axs[int(i/4),i%4].imshow(img)

  axs[int(i/4),i%4].set_title("Ground Truth Label : %s\nPredicted Label : %s"%(actual, pred),fontsize=40)
```

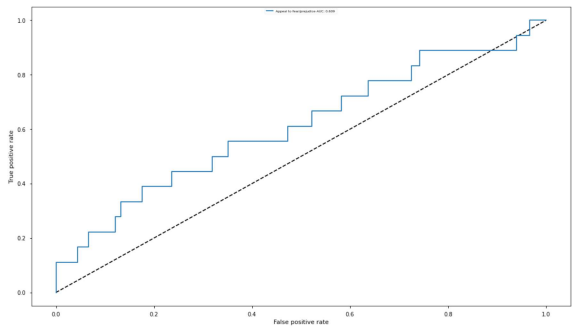


In [91]:
```
val_preds = np.array([]).reshape((0,22))
val_true = np.array([]).reshape((0,22))
for i in range(test_generator.__len__()):
    x,y = test_generator.__getitem__(i)
    val_true = np.append(val_true,y,axis=0)
    val_preds = np.append(val_preds,loaded_model(x).numpy().reshape((1,22)),axis=0)
```

In [92]:
```
plot_roc_curve(val_true, val_preds, mlb.classes_[0:3])
```

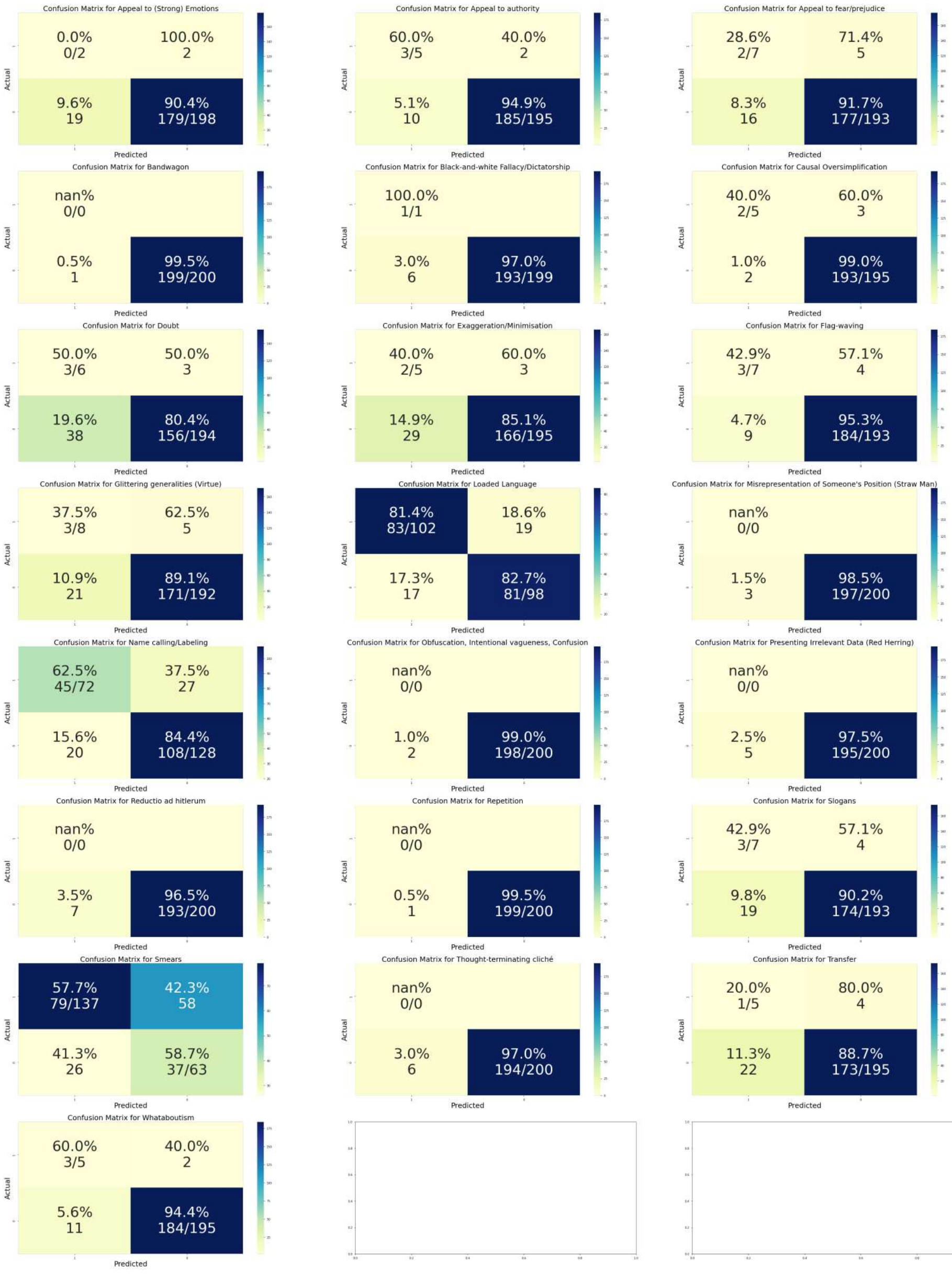






In [93]:
```
plot_labelwise_confusion_matrix_(val_true, val_preds, mlb.classes_)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: invalid value encountered in true_divid
e
  # Remove the CWD from sys.path while we load stuff.
```



4 Evaluation on Independent Data

In [112]:
```
zero_arr = [0]*22
```

In [113]:
```
ind_data_path = "/content/independent_data/"
independent_df = pd.DataFrame({'image':['01247.png', '01256.png', '01269.png']})
independent_df['text'] = ["you can't be racist if there is no other race", "when your debit card decline at the abortion c
independent_df['labels_mh'] = [zero_arr, zero_arr, zero_arr]
```

In [114]:
```
independent_df
```

Out[114]:

	image	text	labels_mh
0	01247.png	you can't be racist if there is no other race	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	01256.png	when your debit card decline at the abortion c...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2	01269.png	kermit the frog definitely not a muslim	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

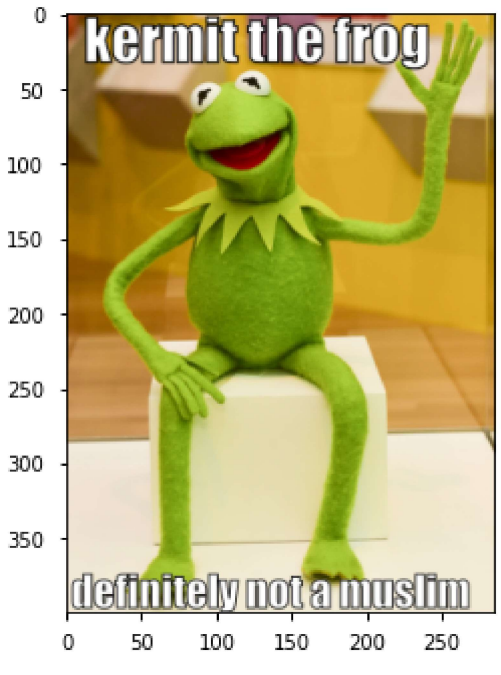
In [115]:
```
ind_generator = DataGenerator(
                    independent_df,
                    ind_data_path,
                    batch_size=TEST_BATCH_SIZE,
                    shuffle=False,
                    n_classes=22
                )
```

In [118]:
```
ind_predictions = loaded_model.predict(ind_generator)
threshold = 0.5
ind_predictions = (ind_predictions > threshold).astype('int')
```

In [127]:
```
for pred, img in zip(ind_predictions, independent_df['image']):
  image = Image.open(os.path.join(ind_data_path, img))
  plt.imshow(image)
  plt.show()
  print("Prediction : ", inverse_transform_mlb(pred,mlb))
```


```
Prediction :  ['Loaded Language', 'Name calling/Labeling']
```


```
Prediction :  ['Smears']
```



```
Prediction :  ['Loaded Language', 'Smears']
```

4.1 Conclusion

✅ From the experiments conducted, the fine tuned BERT model with cross attention and GRU layers gives a high f1-score.

✅ The sample predictions on the test set show that the model is able perform well on classes with high support value in the dataset. While the performance is poor when it comes to minority classes.

✅ Independent dataset taken from the internet shows similar results.

✅ Future: The model can be improved using better variants of BERT models and model interpretability techniques can be used to understand how the model interprets the relation between text and image features for different inputs

4.2 REFERENCES

- https://medium.com/deep-learning-with-keras/how-to-solve-multi-label-classification-problems-in-deep-learning-with-tensorflow-keras-7fb933243595 (https://medium.com/deep-learning-with-keras/how-to-solve-multi-label-classification-problems-in-deep-learning-with-tensorflow-keras-7fb933243595)
- https://machinelearningmastery.com/ (https://machinelearningmastery.com/) multi-label-classification-with-deep-learning/