Loaing Necessary Libraries ¶

In [1]:

```
import re
import string
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score

import tensorflow as tf
from tensorflow.keras import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPool2D, GlobalAveragePooling2D
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Activation, Dropout
from tensorflow.keras.layers import Conv1D, Embedding, GlobalAveragePooling1D
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing import image

from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

Reading Image Info from CSV and Cleaning

In [2]:
```
df = pd.read_csv('../input/memotion-dataset-7k/memotion_dataset_7k/labels.csv')
df.drop(df.columns[df.columns.str.contains('unnamed',case = False)],axis = 1, inplace = True)
df = df.drop(columns = ['text_ocr', 'overall_sentiment'])
df.head()
```

Out[2]:

	image_name	text_corrected	humour	sarcasm	offensive	motivational
0	image_1.jpg	LOOK THERE MY FRIEND LIGHTYEAR NOW ALL SOHALIK...	hilarious	general	not_offensive	not_motivational
1	image_2.jpeg	The best of #10 YearChallenge! Completed in le...	not_funny	general	not_offensive	motivational
2	image_3.JPG	Sam Thorne @Strippin (Follow Follow Saw every...	very_funny	not_sarcastic	not_offensive	not_motivational
3	image_4.png	10 Year Challenge - Sweet Dee Edition	very_funny	twisted_meaning	very_offensive	motivational
4	image_5.png	10 YEAR CHALLENGE WITH NO FILTER 47 Hilarious ...	hilarious	very_twisted	very_offensive	not_motivational

In [3]:
```
df[df.isnull().any(axis=1)]
```

Out[3]:

	image_name	text_corrected	humour	sarcasm	offensive	motivational
119	image_120.jpg	NaN	not_funny	general	not_offensive	not_motivational
4799	image_4800.jpg	NaN	very_funny	general	slight	motivational
6781	image_6782.jpg	NaN	very_funny	twisted_meaning	not_offensive	not_motivational
6784	image_6785.jpg	NaN	hilarious	general	not_offensive	not_motivational
6786	image_6787.jpg	NaN	not_funny	not_sarcastic	very_offensive	motivational

In [4]:
```
cleaned = df.copy()
cleaned.dropna(inplace=True)
cleaned.isnull().any()
```

Out[4]:
```
image_name        False
text_corrected    False
humour            False
sarcasm           False
offensive         False
motivational      False
dtype: bool
```

Image Modelling

Loading Images

In [5]:

```
width = 100
height = 100
X = []
for i in tqdm(range(cleaned.shape[0])):
    if i in [119, 4799, 6781, 6784, 6786]:
        pass
    else:
        path = '../input/memotion-dataset-7k/memotion_dataset_7k/images/'+cleaned['image_name'][i]
        img = image.load_img(path,target_size=(width,height,3))
        img = image.img_to_array(img)
        img = img/255.0
        X.append(img)

X = np.array(X)
```

```
 93%|█████████▎ | 6511/6987 [01:59<00:09, 49.31it/s]/opt/conda/lib/python3.7/site-packages/PIL/TiffI
magePlugin.py:792: UserWarning: Corrupt EXIF data.  Expecting to read 2 bytes but only got 0.
  warnings.warn(str(msg))
 96%|█████████▌ | 6675/6987 [02:02<00:05, 52.01it/s]/opt/conda/lib/python3.7/site-packages/PIL/Imag
e.py:952: UserWarning: Palette images with Transparency expressed in bytes should be converted to R
GBA images
  "Palette images with Transparency expressed in bytes should be "
100%|██████████| 6987/6987 [02:08<00:00, 54.20it/s]
```

In [6]:
```
X.shape
```

Out[6]:
```
(6982, 100, 100, 3)
```

Dropping few rows to make shape consistent

In [7]:
```
rows_to_drop = ['image_120.jpg',
                'image_4800.jpg',
                'image_6782.jpg',
                'image_6785.jpg',
                'image_6787.jpg',
                'image_6988.jpg',
                'image_6989.jpg',
                'image_6990.png',
                'image_6991.jpg',
                'image_6992.jpg']
```

In [8]:
```
for images in rows_to_drop:
    cleaned.drop(cleaned[cleaned['image_name'] == images].index, inplace=True)
```

In [9]:
```
target = pd.get_dummies(cleaned.iloc[:,2:])
target.head()
```

Out[9]:

	humour_funny	humour_hilarious	humour_not_funny	humour_very_funny	sarcasm_general	sarcasm_not_sarcastic	sarcasm_twisted_m
0	0	1	0	0	1	0	0
1	0	0	1	0	1	0	0
2	0	0	0	1	0	1	0
3	0	0	0	1	0	0	1
4	0	1	0	0	0	0	0

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, target, test_size = 0.2)
```

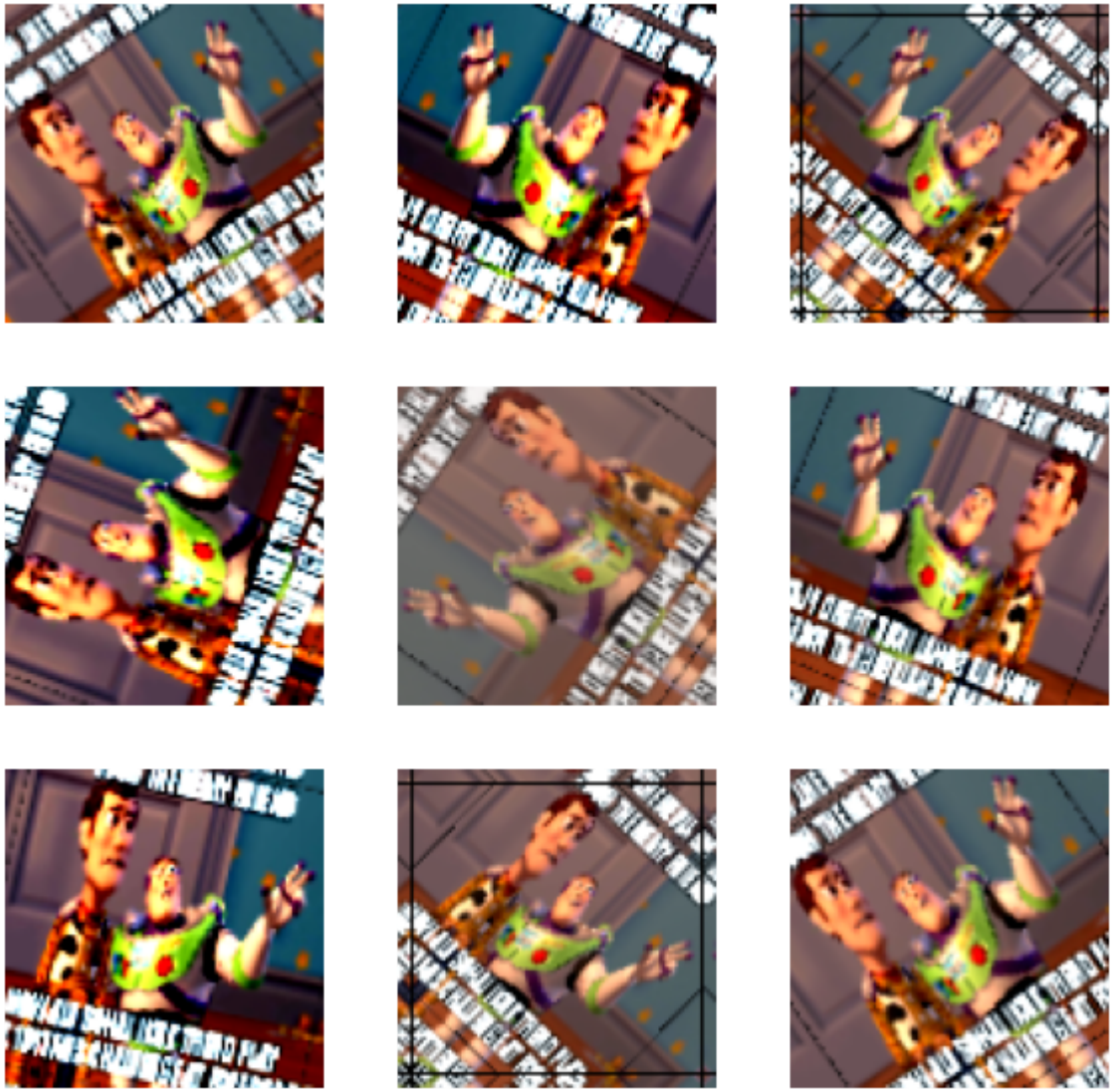
Image Preprocessing

In [11]:

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
  tf.keras.layers.experimental.preprocessing.RandomContrast([.5,2]),
  tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
  tf.keras.layers.experimental.preprocessing.RandomZoom(0.1)
])

preprocess_input = tf.keras.applications.resnet_v2.preprocess_input

rescale = tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
```

In [12]:

```
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = data_augmentation(X)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0])
  plt.axis("off")
```



Base Model

In [13]:
```
base_model_1 = tf.keras.applications.ResNet50(input_shape=X[0].shape,
                                               include_top=False,
                                               weights='imagenet')
base_model_2 = tf.keras.applications.VGG16(input_shape=X[0].shape,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_
weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weig
hts_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
```

In [14]:
```
base_model_1.trainable = False
base_model_2.trainable = False
```

Model for Image

In [15]:
```
def image_model():
    image_input = tf.keras.Input(shape=(150, 150, 3), name = 'image_input')
    image_layers = data_augmentation(image_input)
    image_layers = preprocess_input(image_layers)
    layer_bm_1 = base_model_1(image_input, training=True)
    dropout_layer = Dropout(0.2)(layer_bm_1)
    layer_bm_1 = Conv2D(2048, kernel_size=2,padding='valid')(layer_bm_1)
    dropout_layer = Dropout(0.2)(layer_bm_1)
    layer_bm_1 = Dense(512)(dropout_layer)
    dropout_layer = Dropout(0.2)(layer_bm_1)
    layer_bm_2 = base_model_2(image_input, training=True)
    dropout_layer = Dropout(0.2)(layer_bm_2)
    layer_bm_2 = Dense(512)(layer_bm_2)
    dropout_layer = Dropout(0.2)(layer_bm_2)
    layers = tf.keras.layers.concatenate([layer_bm_1, layer_bm_2])
    image_layers = GlobalAveragePooling2D()(layers)
    image_layers = Dropout(0.2, name = 'dropout_layer')(image_layers)
    return image_input, image_layers
```

In [16]:
```
image_input, image_layers = image_model()
```

Text Modelling

Standardization and Cleaning

In [17]:
```
def standardization(data):
    data = data.apply(lambda x: x.lower())
    data = data.apply(lambda x: re.sub(r'\d+', '', x))
    data = data.apply(lambda x: re.sub(r'\w*.com\w*', '', x, flags=re.MULTILINE))
    data = data.apply(lambda x: x.translate(str.maketrans('', '', string.punctuation)))
    return data

cleaned['text_corrected'] = standardization(cleaned.text_corrected)
```

Vectorizing Layers

In [18]:
```
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
vocab_size = 10000
sequence_length = 50

vectorize_layer = TextVectorization(
    max_tokens=vocab_size,
    output_mode='int',
    output_sequence_length=sequence_length)

text_ds = np.asarray(cleaned['text_corrected'])
vectorize_layer.adapt(tf.convert_to_tensor(text_ds))
```

In [19]:
```
X_text_train, X_text_test, y_text_train, y_text_test = train_test_split(cleaned.text_corrected, target
, test_size = 0.2)
```

In [20]:

```
embedding_dim=16

def text_model():
    text_input = tf.keras.Input(shape=(None,), dtype=tf.string, name='text')
    text_layers = vectorize_layer(text_input)
    text_layers = tf.keras.layers.Embedding(vocab_size, embedding_dim, name="embedding")(text_layers)
    dropout_layer = Dropout(0.2)(text_layers)

    text_layers = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(512, activation='relu', return_se
quences=True))(text_layers)
    dropout_layer = Dropout(0.2)(text_layers)
    text_layers = tf.keras.layers.BatchNormalization()(text_layers)

    text_layers = tf.keras.layers.Conv1D(128, 7, padding="valid", activation="relu", strides=3)(text_l
ayers)
    dropout_layer = Dropout(0.2)(text_layers)
    text_layers = tf.keras.layers.GlobalMaxPooling1D()(text_layers)
    dropout_layer = Dropout(0.2)(text_layers)

    text_layers = tf.keras.layers.Dense(2048, activation="relu")(text_layers)
    text_layers = tf.keras.layers.Dropout(0.5)(text_layers)
    return text_input, text_layers

text_input, text_layers = text_model()
```

Combining and Evaluating

Task A: Overall Sentiment

In [21]:
```
def model(layer_1, layer_2, image_input, text_input):
    concatenate = tf.keras.layers.concatenate([layer_1, layer_2], axis=1)
    semi_final_layer = tf.keras.layers.Dense(2048, activation='relu')(concatenate)

    output = tf.keras.layers.Dense(14, activation='softmax', name = 'humuor')(semi_final_layer)

    model = tf.keras.Model(inputs = [image_input, text_input] ,
                           outputs = output)
    return model
```

In [22]:
```
model = model(image_layers, text_layers, image_input, text_input)
```

In [23]:
```
import os
# Define the checkpoint directory to store the checkpoints
checkpoint_dir = './training_checkpoints'

# Name of the checkpoint files
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")
```

In [24]:

```
# Function for decaying the learning rate.
# You can define any decay function you need.
def decay(epoch):
  if epoch < 3:
    return 1e-1
  elif epoch >= 3 and epoch < 5:
    return 1e-2
  else:
    return 1e-5
```

In [25]:

```
# Callback for printing the LR at the end of each epoch.
class PrintLR(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    print('\nLearning rate for epoch {} is {}'.format(epoch + 1,
                                                      model.optimizer.lr.numpy()))

callbacks = [
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
    tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_prefix,
                                       save_weights_only=True),
    tf.keras.callbacks.LearningRateScheduler(decay),
    PrintLR()
]
```

In [26]:

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['binary_accuracy', 'accuracy'])
```

In [27]:

```
history = model.fit(x = {"image_input": X_train, "text_input": X_text_train},
                    y = y_train,
                    batch_size=128,
                    epochs=25,
                    callbacks=callbacks
                   )
```

```
Epoch 1/25
44/44 [==============================] - ETA: 0s - loss: 10.3724 - binary_accuracy: 0.7355 - accura
cy: 1.7905e-04
Learning rate for epoch 1 is 0.10000000149011612
44/44 [==============================] - 18s 419ms/step - loss: 10.3724 - binary_accuracy: 0.7355 -
accuracy: 1.7905e-04
Epoch 2/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 2 is 0.10000000149011612
44/44 [==============================] - 15s 352ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 3/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 3 is 0.10000000149011612
44/44 [==============================] - 17s 390ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 4/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 4 is 0.009999999776482582
44/44 [==============================] - 16s 372ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 5/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 5 is 0.009999999776482582
44/44 [==============================] - 17s 377ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 6/25
```

```
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 6 is 9.999999747378752e-06
44/44 [==============================] - 15s 350ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 7/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 7 is 9.999999747378752e-06
44/44 [==============================] - 17s 396ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 8/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 8 is 9.999999747378752e-06
44/44 [==============================] - 15s 351ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 9/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 9 is 9.999999747378752e-06
44/44 [==============================] - 16s 354ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 10/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 10 is 9.999999747378752e-06
44/44 [==============================] - 15s 351ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 11/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
```

```
Learning rate for epoch 11 is 9.999999747378752e-06
44/44 [==============================] - 18s 407ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 12/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 12 is 9.999999747378752e-06
44/44 [==============================] - 15s 345ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 13/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 13 is 9.999999747378752e-06
44/44 [==============================] - 16s 363ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 14/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 14 is 9.999999747378752e-06
44/44 [==============================] - 15s 345ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 15/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 15 is 9.999999747378752e-06
44/44 [==============================] - 17s 380ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 16/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 16 is 9.999999747378752e-06
44/44 [==============================] - 15s 344ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
```

```
accuracy: 0.0000e+00
Epoch 17/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 17 is 9.999999747378752e-06
44/44 [==============================] - 17s 376ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 18/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 18 is 9.999999747378752e-06
44/44 [==============================] - 16s 360ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 19/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 19 is 9.999999747378752e-06
44/44 [==============================] - 16s 370ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 20/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 20 is 9.999999747378752e-06
44/44 [==============================] - 15s 349ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 21/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 21 is 9.999999747378752e-06
44/44 [==============================] - 17s 377ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 22/25
```

```
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 22 is 9.999999747378752e-06
44/44 [==============================] - 17s 397ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 23/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 23 is 9.999999747378752e-06
44/44 [==============================] - 17s 378ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 24/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 24 is 9.999999747378752e-06
44/44 [==============================] - 17s 380ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
Epoch 25/25
44/44 [==============================] - ETA: 0s - loss: 10.3668 - binary_accuracy: 0.7361 - accura
cy: 0.0000e+00
Learning rate for epoch 25 is 9.999999747378752e-06
44/44 [==============================] - 15s 351ms/step - loss: 10.3668 - binary_accuracy: 0.7361 -
accuracy: 0.0000e+00
```

In [28]:

```
df_history = pd.DataFrame(history.history)
```

In [29]:

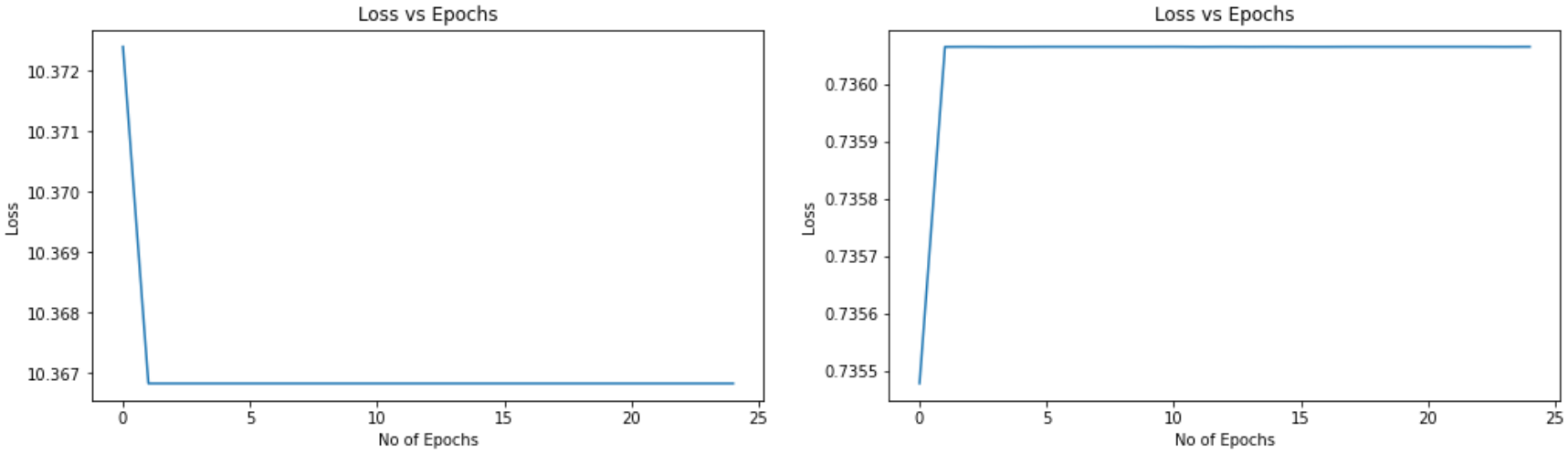
```
df_history
```

Out[29]:

	loss	binary_accuracy	accuracy	lr
0	10.372398	0.735478	0.000179	0.10000
1	10.366833	0.736066	0.000000	0.10000
2	10.366833	0.736066	0.000000	0.10000
3	10.366833	0.736066	0.000000	0.01000
4	10.366833	0.736066	0.000000	0.01000
5	10.366833	0.736066	0.000000	0.00001
6	10.366833	0.736066	0.000000	0.00001
7	10.366833	0.736066	0.000000	0.00001
8	10.366833	0.736066	0.000000	0.00001
9	10.366833	0.736066	0.000000	0.00001
10	10.366833	0.736066	0.000000	0.00001
11	10.366833	0.736066	0.000000	0.00001
12	10.366833	0.736066	0.000000	0.00001
13	10.366833	0.736066	0.000000	0.00001
14	10.366833	0.736066	0.000000	0.00001
15	10.366833	0.736066	0.000000	0.00001
16	10.366833	0.736066	0.000000	0.00001
17	10.366833	0.736066	0.000000	0.00001
18	10.366833	0.736066	0.000000	0.00001
19	10.366833	0.736066	0.000000	0.00001
20	10.366833	0.736066	0.000000	0.00001
21	10.366833	0.736066	0.000000	0.00001
22	10.366833	0.736066	0.000000	0.00001
23	10.366833	0.736066	0.000000	0.00001
24	10.366833	0.736066	0.000000	0.00001

In [30]:

```
fig, axes = plt.subplots(1,2, figsize=(15, 5))
fig.tight_layout(pad=5.0)

axes[0].plot(df_history.loss)
axes[0].set_xlabel('No of Epochs')
axes[0].set_ylabel('Loss')
axes[0].set_title('Loss vs Epochs')

axes[1].plot(df_history.binary_accuracy)
axes[1].set_xlabel('No of Epochs')
axes[1].set_ylabel('Loss')
axes[1].set_title('Loss vs Epochs')
plt.show()
```



In [31]:

```
prediction = model.predict(x = {"image_input": X_test, "text_input": X_text_test})
prediction = np.array(prediction)
prediction = np.squeeze(prediction)
prediction = 1/(1+np.exp(-np.array(prediction)))
prediction = np.where(prediction > 0.5, 1, 0)
y_true = y_test.values

micro_f1_score = f1_score(y_true[:,1], prediction[:,1], average='micro')
macro_f1_score = f1_score(y_true[:,1], prediction[:,1], average='macro')

print("Micro F1 score for Task C is ", micro_f1_score)
print("Macro F1 score for Task C is ", macro_f1_score)
```

```
Micro F1 score for Task C is  0.9141016463851109
Macro F1 score for Task C is  0.47756170531039643
```

In [32]:

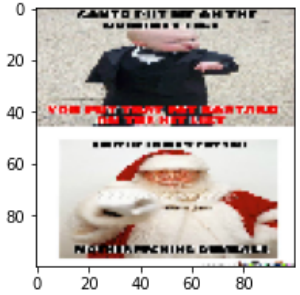
```
predictions = model.predict(x = {"image_input": X_test, "text_input": X_text_test})
predictions = np.array(predictions)
predictions.shape
```

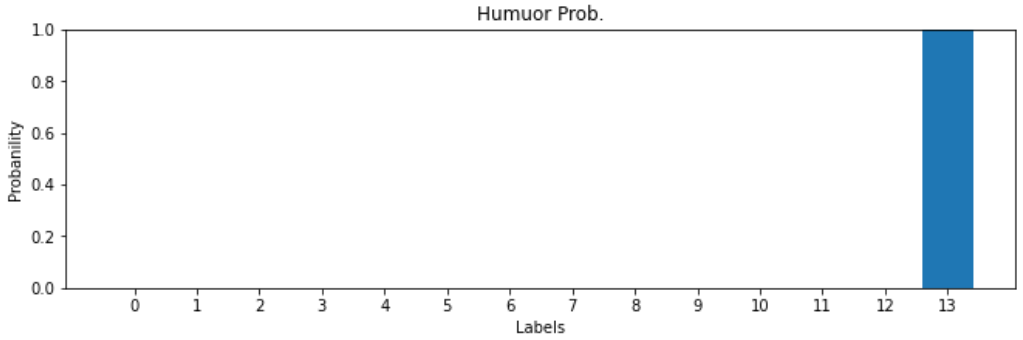
Out[32]:

```
(1397, 14)
```

In [33]:

```
import random

fig, axes = plt.subplots(1,2, figsize=(20, 4))
fig.tight_layout(pad=5.0)


x = list(range(0,14))

axes[0].imshow(X[random.randint(0,X_test.shape[0]),:,:,:])

axes[1].bar(x, predictions[random.randint(0,X_test.shape[0]),:])
axes[1].set_xlabel('Labels')
axes[1].set_ylabel('Probanility')
axes[1].set_title('Humuor Prob.')
axes[1].set_xticks(x)
axes[1].set_ylim(0,1)
plt.show()

print(predictions[random.randint(0,X_test.shape[0]),:].max(), np.where(predictions[random.randint(0,X_
test.shape[0]),:].max()))
```

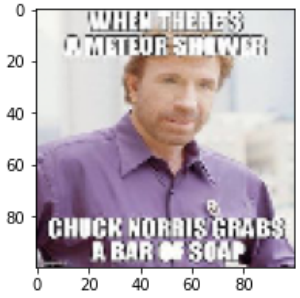


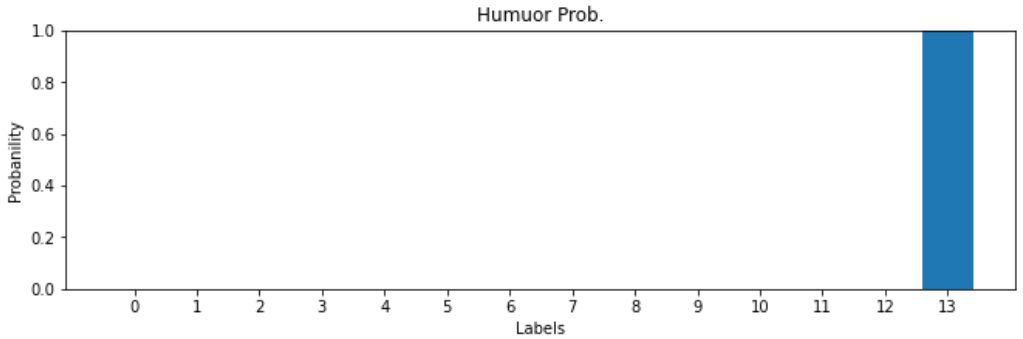

Humuor Prob.
Probaniility
Labels
0.0
0.2
0.4
0.6
0.8
1.0
0 1 2 3 4 5 6 7 8 9 10 11 12 13

```
1.0 (array([0]),)
```

In [34]:

```
fig, axes = plt.subplots(1,2, figsize=(20, 4))
fig.tight_layout(pad=5.0)


x = list(range(0,14))

axes[0].imshow(X[random.randint(0,X_test.shape[0]),:,:,:])

axes[1].bar(x, predictions[random.randint(0,X_test.shape[0]),:])
axes[1].set_xlabel('Labels')
axes[1].set_ylabel('Probanility')
axes[1].set_title('Humuor Prob.')
axes[1].set_xticks(x)
axes[1].set_ylim(0,1)
plt.show()

print(predictions[random.randint(0,X_test.shape[0]),:].max(), np.where(predictions[random.randint(0,X_
test.shape[0]),:].max()))
```

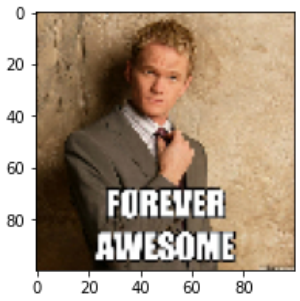


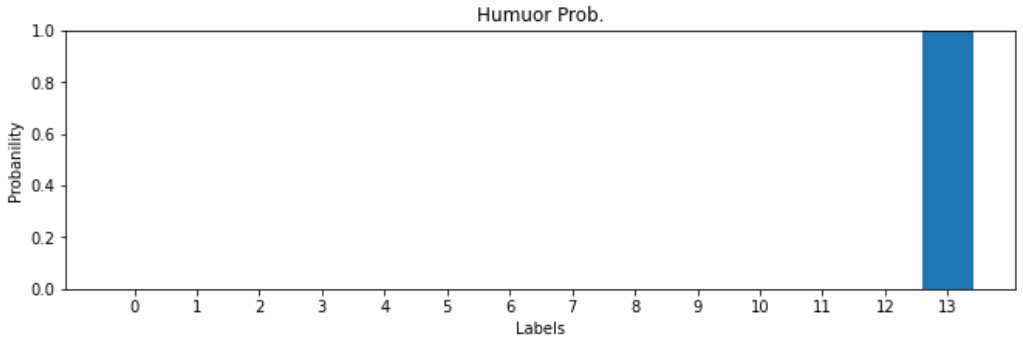



```
1.0 (array([0]),)
```

In [35]:

```
fig, axes = plt.subplots(1,2, figsize=(20, 4))
fig.tight_layout(pad=5.0)


x = list(range(0,14))

axes[0].imshow(X[random.randint(0,X_test.shape[0]),:,:,:])

axes[1].bar(x, predictions[random.randint(0,X_test.shape[0]),:])
axes[1].set_xlabel('Labels')
axes[1].set_ylabel('Probanility')
axes[1].set_title('Humuor Prob.')
axes[1].set_xticks(x)
axes[1].set_ylim(0,1)
plt.show()

print(predictions[random.randint(0,X_test.shape[0]),:].max(), np.where(predictions[random.randint(0,X_
test.shape[0]),:].max()))
```

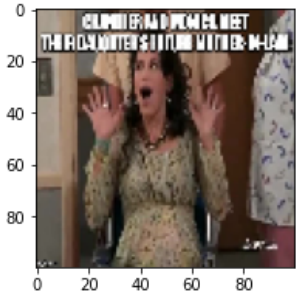


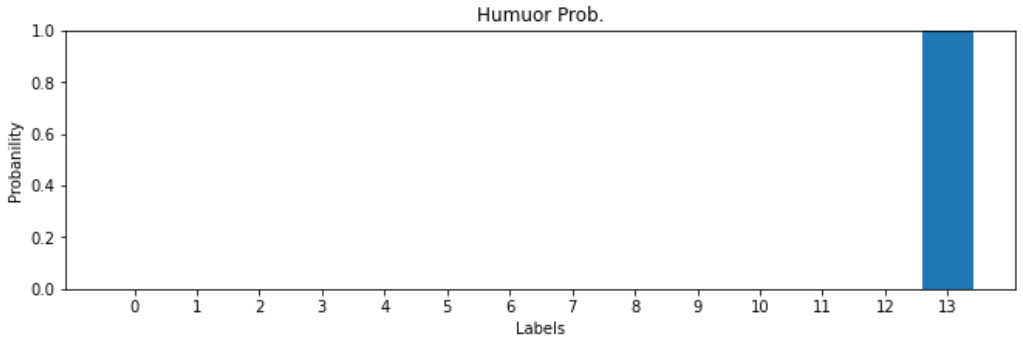



```
1.0 (array([0]),)
```

In [36]:

```
fig, axes = plt.subplots(1,2, figsize=(20, 4))
fig.tight_layout(pad=5.0)


x = list(range(0,14))

axes[0].imshow(X[random.randint(0,X_test.shape[0]),:,:,:])

axes[1].bar(x, predictions[random.randint(0,X_test.shape[0]),:])
axes[1].set_xlabel('Labels')
axes[1].set_ylabel('Probanility')
axes[1].set_title('Humuor Prob.')
axes[1].set_xticks(x)
axes[1].set_ylim(0,1)
plt.show()

print(predictions[random.randint(0,X_test.shape[0]),:].max(), np.where(predictions[random.randint(0,X_
test.shape[0]),:].max()))
```






```
1.0 (array([0]),)
```

In [ ]: