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', 'humour', 'sarcasm', 'offensive', 'motivational'])
df.head()
```

Out[2]:

	image_name	text_corrected	overall_sentiment
0	image_1.jpg	LOOK THERE MY FRIEND LIGHTYEAR NOW ALL SOHALIK...	very_positive
1	image_2.jpeg	The best of #10 YearChallenge! Completed in le...	very_positive
2	image_3.JPG	Sam Thorne @Strippin (Follow Follow Saw every...	positive
3	image_4.png	10 Year Challenge - Sweet Dee Edition	positive
4	image_5.png	10 YEAR CHALLENGE WITH NO FILTER 47 Hilarious ...	neutral

In [3]:
```
cleaned = df.copy()
cleaned.dropna(inplace=True)
cleaned.isnull().any()
```

Out[3]:
```
image_name           False
text_corrected       False
overall_sentiment    False
dtype: bool
```

Image Modelling

Loading Images

In [4]:

```
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%|█████████▎| 6506/6987 [01:33<00:05, 83.07it/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%|█████████▌| 6676/6987 [01:35<00:03, 81.68it/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 [01:40<00:00, 69.25it/s]
```

In [5]:
```
X.shape
```

Out[5]:
```
(6982, 100, 100, 3)
```

Dropping few rows to make shape consistent

In [6]:
```
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 [7]:
```
for images in rows_to_drop:
    cleaned.drop(cleaned[cleaned['image_name'] == images].index, inplace=True)
```

In [8]:
```
target = cleaned['overall_sentiment']
target = pd.get_dummies(target)
target.head()
```

Out[8]:

	negative	neutral	positive	very_negative	very_positive
0	0	0	0	0	1
1	0	0	0	0	1
2	0	0	1	0	0
3	0	0	1	0	0
4	0	1	0	0	0

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, target, test_size = 0.2, stratify=target)
```

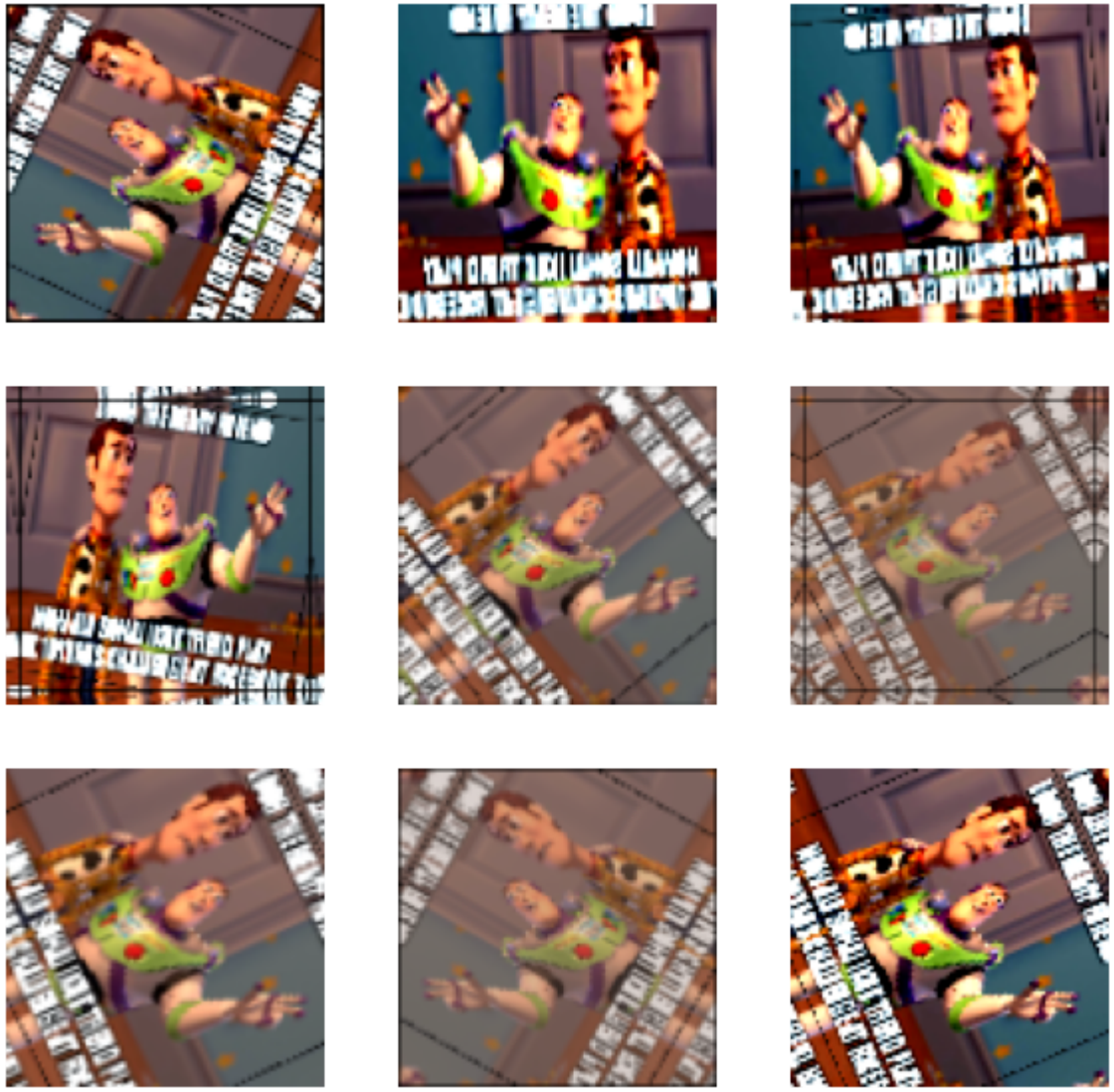
Image Preprocessing

In [10]:

```
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 [11]:

```
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 [12]:
```
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 [13]:
```
base_model_1.trainable = False
base_model_2.trainable = False
```

Model for Image

In [14]:
```
def image_model():
    image_input = tf.keras.Input(shape=(100, 100, 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=False)
    layer_bm_1 = Conv2D(2048, kernel_size=2,padding='valid')(layer_bm_1)
    layer_bm_1 = Dense(512)(layer_bm_1)
    layer_bm_2 = base_model_2(image_input, training=False)
    layer_bm_2 = Dense(512)(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 [15]:
```
image_input, image_layers = image_model()
```

Text Modelling

Standardization and Cleaning

In [16]:
```
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'.com', '', 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 [17]:
```
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 [18]:
```
X_text_train, X_text_test, y_text_train, y_text_test = train_test_split(cleaned.text_corrected, target
, test_size = 0.2, stratify=target)
```

In [19]:

```
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)

    text_layers = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(512, activation='relu', return_se
quences=True))(text_layers)
    text_layers = tf.keras.layers.BatchNormalization()(text_layers)
    text_layers = tf.keras.layers.Bidirectional(tf.keras.layers.GRU(512, activation='relu', return_seq
uences=True))(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)

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

    text_layers = tf.keras.layers.GlobalMaxPooling1D()(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 [20]:
```
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='softmax')(concatenate)

    prediction_layer = tf.keras.layers.Dense(5, activation='sigmoid', name = 'task_a')

    output = prediction_layer(semi_final_layer)

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

In [21]:
```
model = model(image_layers, text_layers, image_input, text_input)
```

In [22]:
```
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 [23]:

```
# 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 [24]:

```
# 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 [25]:

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

history = model.fit(x = {"image_input": X_train, "text_input": X_text_train},
                    y = y_train,
                    batch_size=256,
                    epochs=25,
                    callbacks=callbacks
                   )
```

```
Epoch 1/25
22/22 [==============================] - ETA: 0s - loss: 1.4386 - binary_accuracy: 0.7008 - accurac
y: 0.4469
Learning rate for epoch 1 is 0.10000000149011612
22/22 [==============================] - 20s 926ms/step - loss: 1.4386 - binary_accuracy: 0.7008 -
accuracy: 0.4469
Epoch 2/25
22/22 [==============================] - ETA: 0s - loss: 1.3726 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 2 is 0.10000000149011612
22/22 [==============================] - 16s 730ms/step - loss: 1.3726 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 3/25
22/22 [==============================] - ETA: 0s - loss: 1.3696 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 3 is 0.10000000149011612
22/22 [==============================] - 17s 773ms/step - loss: 1.3696 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 4/25
22/22 [==============================] - ETA: 0s - loss: 1.3688 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 4 is 0.009999999776482582
22/22 [==============================] - 17s 757ms/step - loss: 1.3688 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 5/25
22/22 [==============================] - ETA: 0s - loss: 1.3688 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 5 is 0.009999999776482582
22/22 [==============================] - 16s 725ms/step - loss: 1.3688 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 6/25
```

```
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 6 is 9.999999747378752e-06
22/22 [==============================] - 18s 814ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 7/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 7 is 9.999999747378752e-06
22/22 [==============================] - 16s 705ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 8/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 8 is 9.999999747378752e-06
22/22 [==============================] - 16s 740ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 9/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 9 is 9.999999747378752e-06
22/22 [==============================] - 16s 748ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 10/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 10 is 9.999999747378752e-06
22/22 [==============================] - 17s 760ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 11/25
22/22 [==============================] - ETA: 0s - loss: 1.3688 - binary_accuracy: 0.7047 - accurac
y: 0.4471
```

```
Learning rate for epoch 11 is 9.999999747378752e-06
22/22 [==============================] - 16s 728ms/step - loss: 1.3688 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 12/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 12 is 9.999999747378752e-06
22/22 [==============================] - 17s 768ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 13/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 13 is 9.999999747378752e-06
22/22 [==============================] - 17s 786ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 14/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 14 is 9.999999747378752e-06
22/22 [==============================] - 18s 817ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 15/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 15 is 9.999999747378752e-06
22/22 [==============================] - 16s 724ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 16/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 16 is 9.999999747378752e-06
22/22 [==============================] - 17s 784ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
```

```
accuracy: 0.4471
Epoch 17/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 17 is 9.999999747378752e-06
22/22 [==============================] - 16s 744ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 18/25
22/22 [==============================] - ETA: 0s - loss: 1.3688 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 18 is 9.999999747378752e-06
22/22 [==============================] - 17s 754ms/step - loss: 1.3688 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 19/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 19 is 9.999999747378752e-06
22/22 [==============================] - 16s 736ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 20/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 20 is 9.999999747378752e-06
22/22 [==============================] - 18s 797ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 21/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 21 is 9.999999747378752e-06
22/22 [==============================] - 16s 742ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 22/25
```

```
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 22 is 9.999999747378752e-06
22/22 [==============================] - 16s 723ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 23/25
22/22 [==============================] - ETA: 0s - loss: 1.3687 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 23 is 9.999999747378752e-06
22/22 [==============================] - 18s 812ms/step - loss: 1.3687 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 24/25
22/22 [==============================] - ETA: 0s - loss: 1.3688 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 24 is 9.999999747378752e-06
22/22 [==============================] - 16s 723ms/step - loss: 1.3688 - binary_accuracy: 0.7047 -
accuracy: 0.4471
Epoch 25/25
22/22 [==============================] - ETA: 0s - loss: 1.3688 - binary_accuracy: 0.7047 - accurac
y: 0.4471
Learning rate for epoch 25 is 9.999999747378752e-06
22/22 [==============================] - 17s 751ms/step - loss: 1.3688 - binary_accuracy: 0.7047 -
accuracy: 0.4471
```

In [26]:

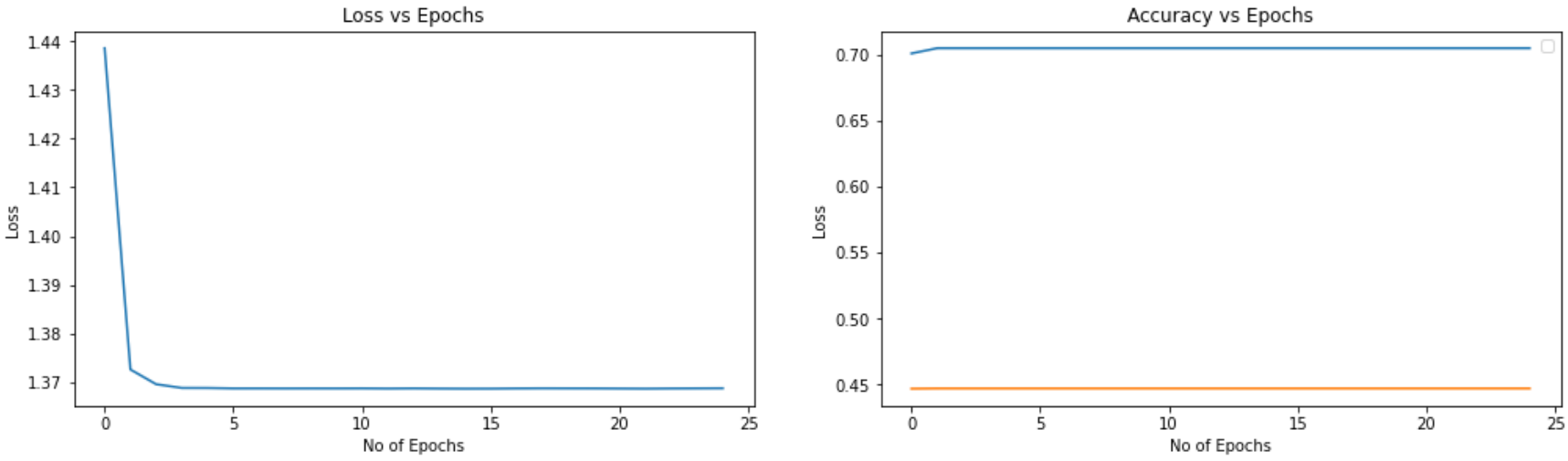
```
df_history = pd.DataFrame(history.history)
df_history
```

Out[26]:

	loss	binary_accuracy	accuracy	lr
0	1.438555	0.700770	0.446911	0.10000
1	1.372624	0.704745	0.447090	0.10000
2	1.369597	0.704745	0.447090	0.10000
3	1.368839	0.704745	0.447090	0.01000
4	1.368828	0.704745	0.447090	0.01000
5	1.368740	0.704745	0.447090	0.00001
6	1.368745	0.704745	0.447090	0.00001
7	1.368736	0.704745	0.447090	0.00001
8	1.368744	0.704745	0.447090	0.00001
9	1.368739	0.704745	0.447090	0.00001
10	1.368751	0.704745	0.447090	0.00001
11	1.368716	0.704745	0.447090	0.00001
12	1.368749	0.704745	0.447090	0.00001
13	1.368713	0.704745	0.447090	0.00001
14	1.368689	0.704745	0.447090	0.00001
15	1.368696	0.704745	0.447090	0.00001
16	1.368739	0.704745	0.447090	0.00001
17	1.368760	0.704745	0.447090	0.00001
18	1.368750	0.704745	0.447090	0.00001
19	1.368745	0.704745	0.447090	0.00001
20	1.368719	0.704745	0.447090	0.00001
21	1.368694	0.704745	0.447090	0.00001
22	1.368732	0.704745	0.447090	0.00001
23	1.368752	0.704745	0.447090	0.00001
24	1.368774	0.704745	0.447090	0.00001

In [27]:

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



In [28]:

```
eval_ = model.evaluate(x = {"image_input": X_test, "text_input": X_text_test},
                       y = y_test,
                       batch_size=32,
                       verbose=1
                      )
```

```
44/44 [==============================] - 4s 80ms/step - loss: 1.3688 - binary_accuracy: 0.7047 - ac
curacy: 0.4467
```

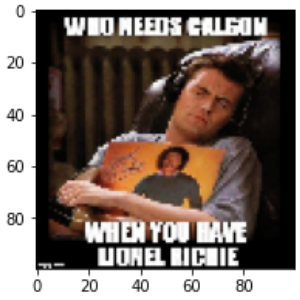
In [29]:

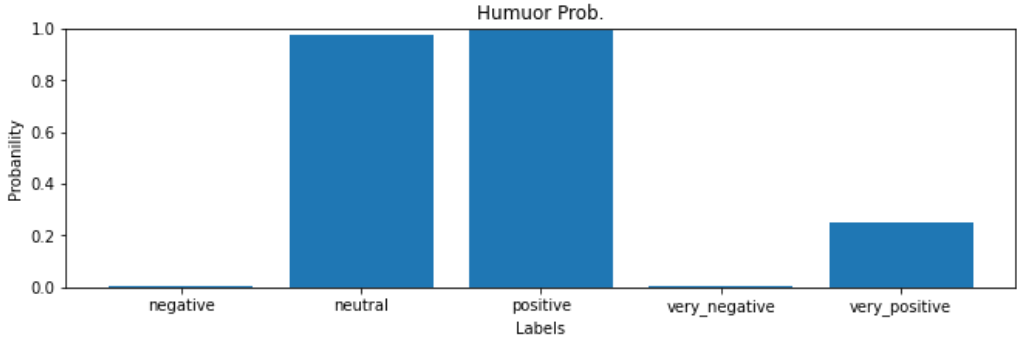
```
prediction = model.predict(x = {"image_input": X_test, "text_input": X_text_test})
prediction = np.array(prediction)
prediction = np.squeeze(prediction)
prediction = 1/(1+np.exp(-np.array(prediction)))
prediction = np.where(prediction > 0.5, 1, 0)
y_true = y_test.values

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

```
Micro F1 score for Task A is  0.31496062992125984
Macro F1 score for Task A is  0.23952095808383236
```

In [30]:

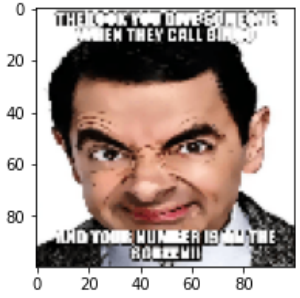
```
import random

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


x = list(y_test.columns)

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

axes[1].bar(x, model.predict(x = {"image_input": X_test, "text_input": X_text_test})[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()
```

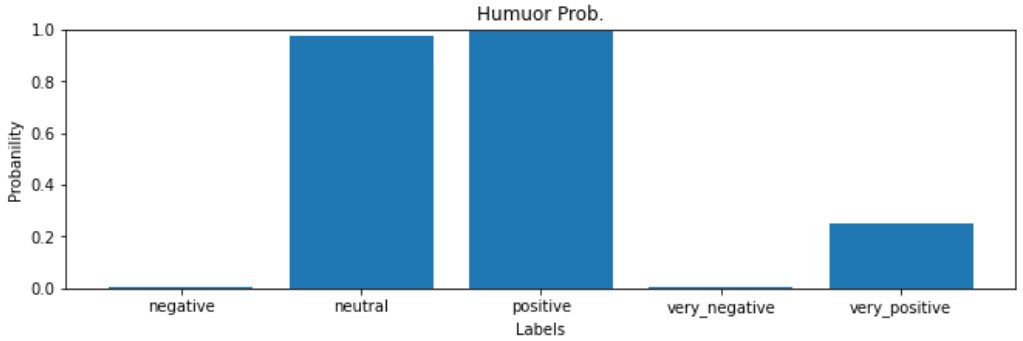




In [31]:

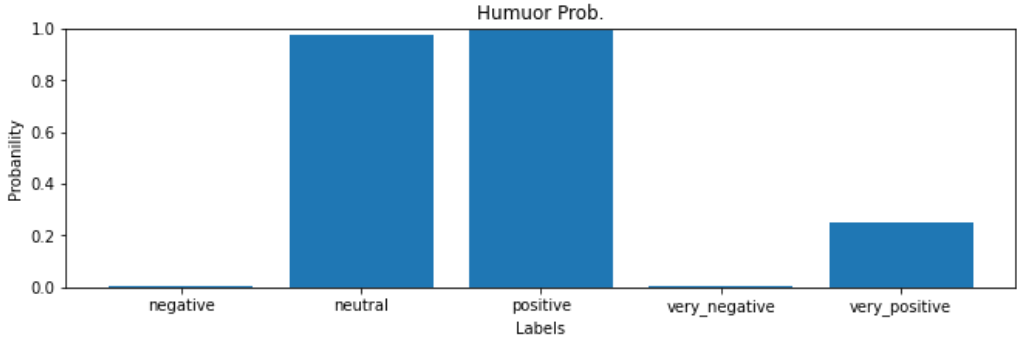
```
fig, axes = plt.subplots(1,2, figsize=(20, 4))
fig.tight_layout(pad=5.0)


x = list(y_test.columns)

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

axes[1].bar(x, model.predict(x = {"image_input": X_test, "text_input": X_text_test})[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()
```





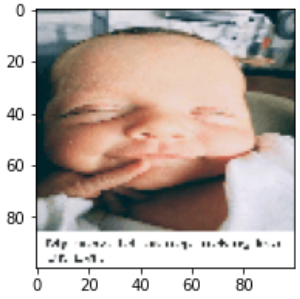
In [32]:

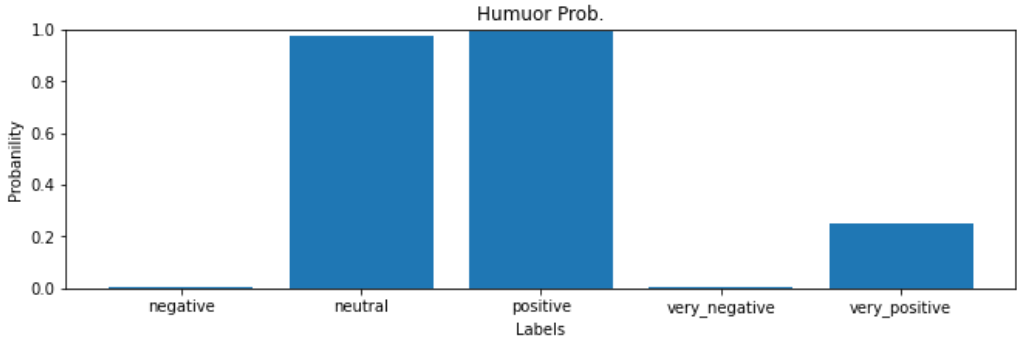
```
fig, axes = plt.subplots(1,2, figsize=(20, 4))
fig.tight_layout(pad=5.0)


x = list(y_test.columns)

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

axes[1].bar(x, model.predict(x = {"image_input": X_test, "text_input": X_text_test})[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()
```



In [33]:

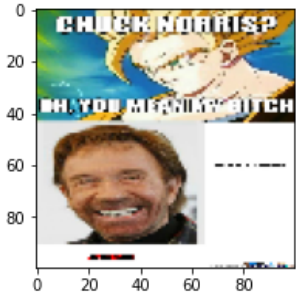
```
fig, axes = plt.subplots(1,2, figsize=(20, 4))
fig.tight_layout(pad=5.0)


x = list(y_test.columns)

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

axes[1].bar(x, model.predict(x = {"image_input": X_test, "text_input": X_text_test})[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()
```

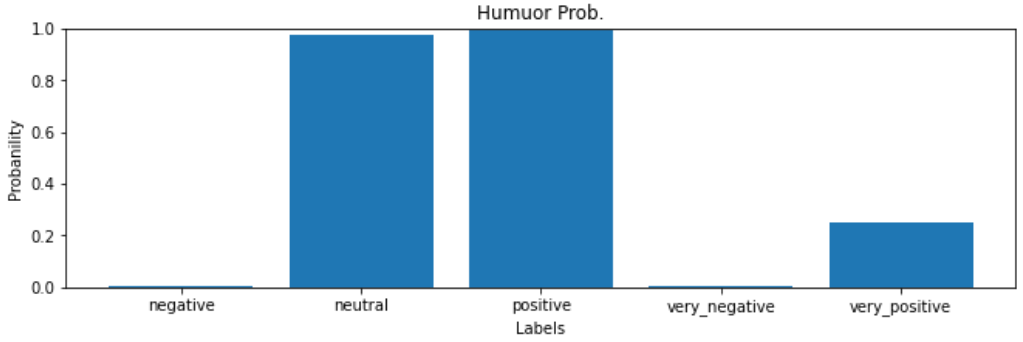




In [34]:

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


x = list(y_test.columns)

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

axes[1].bar(x, model.predict(x = {"image_input": X_test, "text_input": X_text_test})[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()
```

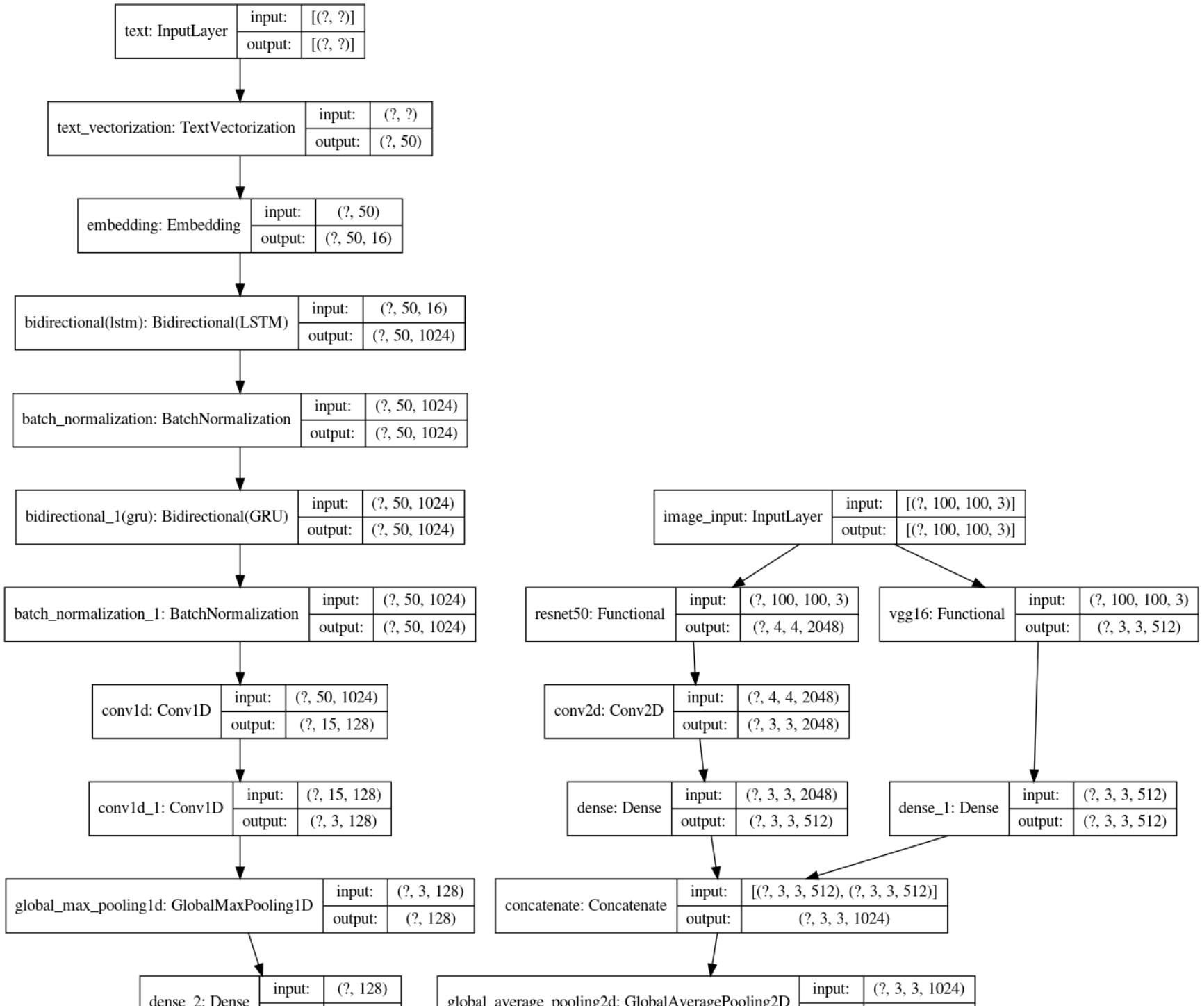


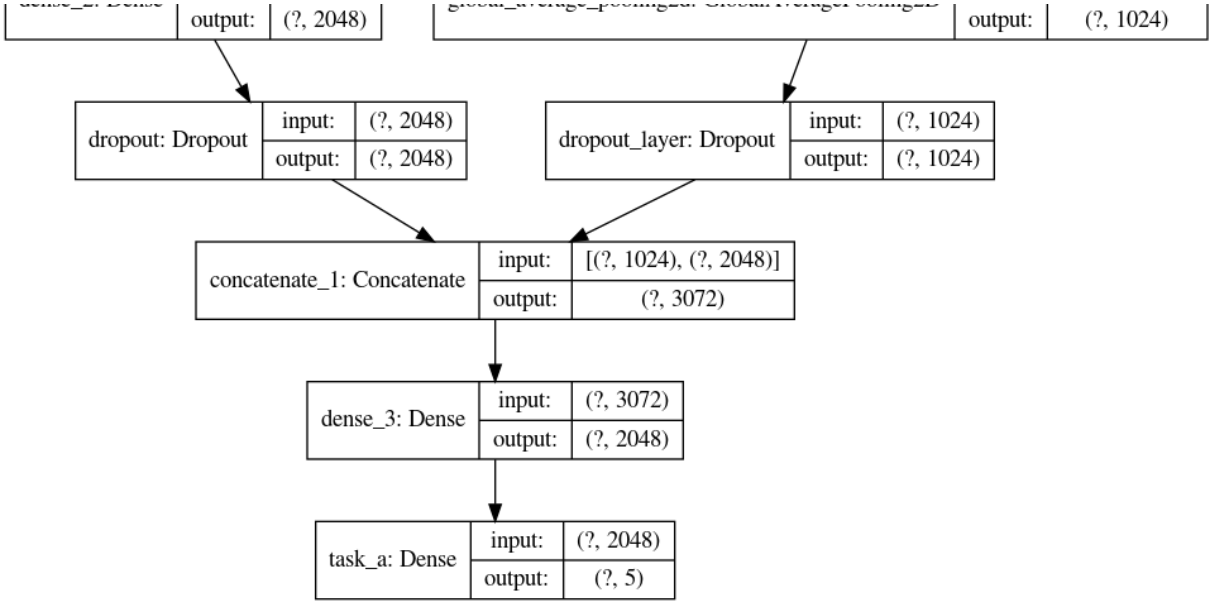


In [35]:

```
tf.keras.utils.plot_model(model, "multi_input_and_output_model.png", show_shapes=True)
```

Out[35]:


text: InputLayer
input: [(?, ?)]
output: [(?, ?)]
text_vectorization: TextVectorization
input: (?, ?)
output: (?, 50)
embedding: Embedding
input: (?, 50)
output: (?, 50, 16)
bidirectional(lstm): Bidirectional(LSTM)
input: (?, 50, 16)
output: (?, 50, 1024)
batch_normalization: BatchNormalization
input: (?, 50, 1024)
output: (?, 50, 1024)
bidirectional_1(gru): Bidirectional(GRU)
input: (?, 50, 1024)
output: (?, 50, 1024)
batch_normalization_1: BatchNormalization
input: (?, 50, 1024)
output: (?, 50, 1024)
conv1d: Conv1D
input: (?, 50, 1024)
output: (?, 15, 128)
conv1d_1: Conv1D
input: (?, 15, 128)
output: (?, 3, 128)
global_max_pooling1d: GlobalMaxPooling1D
input: (?, 3, 128)
output: (?, 128)
dense_2: Dense
input: (?, 128)
image_input: InputLayer
input: [(?, 100, 100, 3)]
output: [(?, 100, 100, 3)]
resnet50: Functional
input: (?, 100, 100, 3)
output: (?, 4, 4, 2048)
vgg16: Functional
input: (?, 100, 100, 3)
output: (?, 3, 3, 512)
conv2d: Conv2D
input: (?, 4, 4, 2048)
output: (?, 3, 3, 2048)
dense: Dense
input: (?, 3, 3, 2048)
output: (?, 3, 3, 512)
dense_1: Dense
input: (?, 3, 3, 512)
output: (?, 3, 3, 512)
concatenate: Concatenate
input: [(?, 3, 3, 512), (?, 3, 3, 512)]
output: (?, 3, 3, 1024)
global_average_pooling2d: GlobalAveragePooling2D
input: (?, 3, 3, 1024)


output: (?, 2048)
output: (?, 1024)
dropout: Dropout
input: (?, 2048)
output: (?, 2048)
dropout_layer: Dropout
input: (?, 1024)
output: (?, 1024)
concatenate_1: Concatenate
input: [(?, 1024), (?, 2048)]
output: (?, 3072)
dense_3: Dense
input: (?, 3072)
output: (?, 2048)
task_a: Dense
input: (?, 2048)
output: (?, 5)

In [ ]: