Loaing Necessary Libraries

In [1]:

```python
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]:

```
cleaned = df.copy()
cleaned.dropna(inplace=True)
cleaned.isnull().any()
```

Out[3]:

```
image_name        False
text_corrected    False
humour            False
sarcasm           False
offensive         False
motivational      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%|█████████▎| 6500/6987 [01:07<00:06, 70.19it/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))
 95%|█████████▌| 6671/6987 [01:08<00:02, 120.38it/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 [01:11<00:00, 97.70it/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]:

```
cleaned = cleaned.replace({'humour': {'not_funny': 0, 'funny': 1, 'very_funny': 1, 'hilarious':1},
                        'sarcasm': {'not_sarcastic': 0, 'general': 1, 'twisted_meaning': 1, 'very_twisted': 1},
                        'offensive': {'not_offensive': 0, 'slight': 1, 'very_offensive': 1, 'hateful_offensive': 1},
                        'motivational': {'not_motivational': 0, 'motivational': 1}})
```

In [9]:

```
target = cleaned.iloc[:,2:]
target.head()
```

Out[9]:

	humour	sarcasm	offensive	motivational
0	1	1	0	0
1	0	1	0	1
2	1	0	0	0
3	1	1	1	1
4	1	1	1	0

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(X, target, test_size = 0.2, stratify=target)
```

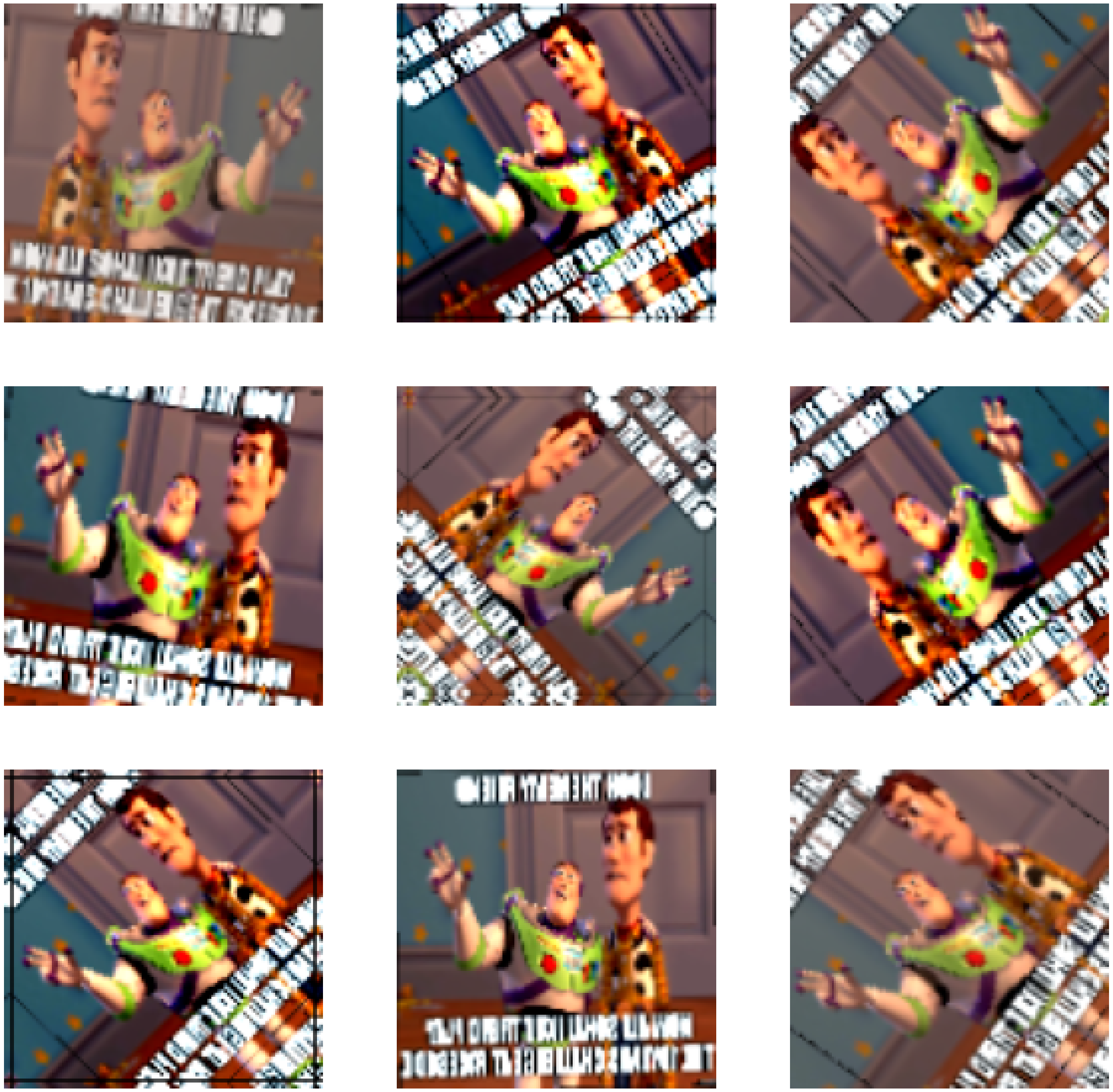
Image Preprocessing

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 [==============================] - 6s 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 [==============================] - 2s 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=False)
    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=False)
    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, stratify=target)
```

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_sequences=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_layers)
    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='softmax')(concatenate)

    prediction_layer = tf.keras.layers.Dense(4, activation='softmax', name = 'task_B_out')


    output = prediction_layer(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 < 5:
    return 1.0
  elif epoch >= 5 and epoch < 15:
    return 0.5
  else:
    return 0.1
```

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),
    tf.keras.callbacks.EarlyStopping(monitor = 'accuracy', patience=5),
    PrintLR()
]
```

In [26]:

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
                  loss = 'categorical_crossentropy',
                  metrics=['categorical_accuracy'])

history = model.fit(x = {"image_input": X_train, "text_input": X_text_train},
                        y= y_train,
                        batch_size=32,
                        epochs=25,
                        validation_data=({"image_input": X_test, "text_input": X_text_test}, y_test ),
                        callbacks=callbacks
                       )
```

```
Epoch 1/25
175/175 [==============================] - ETA: 0s - loss: 3.8728 - categorical_accuracy: 0.3311
Learning rate for epoch 1 is 1.0
175/175 [==============================] - 47s 269ms/step - loss: 3.8728 - categorical_accuracy: 0.
3311 - val_loss: 3.4244 - val_categorical_accuracy: 0.1453
Epoch 2/25
175/175 [==============================] - ETA: 0s - loss: 4.3517 - categorical_accuracy: 0.3495
Learning rate for epoch 2 is 1.0
175/175 [==============================] - 43s 244ms/step - loss: 4.3517 - categorical_accuracy: 0.
3495 - val_loss: 3.5695 - val_categorical_accuracy: 0.8218
Epoch 3/25
175/175 [==============================] - ETA: 0s - loss: 4.4181 - categorical_accuracy: 0.3608
Learning rate for epoch 3 is 1.0
175/175 [==============================] - 44s 249ms/step - loss: 4.4181 - categorical_accuracy: 0.
3608 - val_loss: 3.4761 - val_categorical_accuracy: 0.1453
Epoch 4/25
175/175 [==============================] - ETA: 0s - loss: 4.3523 - categorical_accuracy: 0.3543
Learning rate for epoch 4 is 1.0
175/175 [==============================] - 44s 254ms/step - loss: 4.3523 - categorical_accuracy: 0.
3543 - val_loss: 3.5376 - val_categorical_accuracy: 0.8218
Epoch 5/25
175/175 [==============================] - ETA: 0s - loss: 4.3686 - categorical_accuracy: 0.3859
Learning rate for epoch 5 is 1.0
175/175 [==============================] - 44s 249ms/step - loss: 4.3686 - categorical_accuracy: 0.
3859 - val_loss: 3.3991 - val_categorical_accuracy: 0.8218
Epoch 6/25
175/175 [==============================] - ETA: 0s - loss: 4.2478 - categorical_accuracy: 0.3805
Learning rate for epoch 6 is 0.5
175/175 [==============================] - 43s 247ms/step - loss: 4.2478 - categorical_accuracy: 0.
3805 - val_loss: 3.4202 - val_categorical_accuracy: 0.1453
Epoch 7/25
```

```
175/175 [==============================] - ETA: 0s - loss: 4.1166 - categorical_accuracy: 0.3375
Learning rate for epoch 7 is 0.5
175/175 [==============================] - 44s 253ms/step - loss: 4.1166 - categorical_accuracy: 0.
3375 - val_loss: 3.4676 - val_categorical_accuracy: 0.1453
Epoch 8/25
175/175 [==============================] - ETA: 0s - loss: 4.1310 - categorical_accuracy: 0.3472
Learning rate for epoch 8 is 0.5
175/175 [==============================] - 44s 251ms/step - loss: 4.1310 - categorical_accuracy: 0.
3472 - val_loss: 3.4145 - val_categorical_accuracy: 0.1453
Epoch 9/25
175/175 [==============================] - ETA: 0s - loss: 4.2730 - categorical_accuracy: 0.3400
Learning rate for epoch 9 is 0.5
175/175 [==============================] - 45s 255ms/step - loss: 4.2730 - categorical_accuracy: 0.
3400 - val_loss: 3.4834 - val_categorical_accuracy: 0.1453
Epoch 10/25
175/175 [==============================] - ETA: 0s - loss: 4.2926 - categorical_accuracy: 0.3561
Learning rate for epoch 10 is 0.5
175/175 [==============================] - 43s 244ms/step - loss: 4.2926 - categorical_accuracy: 0.
3561 - val_loss: 3.3995 - val_categorical_accuracy: 0.8218
Epoch 11/25
175/175 [==============================] - ETA: 0s - loss: 4.3280 - categorical_accuracy: 0.3586
Learning rate for epoch 11 is 0.5
175/175 [==============================] - 44s 250ms/step - loss: 4.3280 - categorical_accuracy: 0.
3586 - val_loss: 3.3935 - val_categorical_accuracy: 0.8218
Epoch 12/25
175/175 [==============================] - ETA: 0s - loss: 4.5785 - categorical_accuracy: 0.3803
Learning rate for epoch 12 is 0.5
175/175 [==============================] - 45s 258ms/step - loss: 4.5785 - categorical_accuracy: 0.
3803 - val_loss: 3.4616 - val_categorical_accuracy: 0.1453
Epoch 13/25
175/175 [==============================] - ETA: 0s - loss: 4.4257 - categorical_accuracy: 0.3540
Learning rate for epoch 13 is 0.5
```

```
175/175 [==============================] - 42s 243ms/step - loss: 4.4257 - categorical_accuracy: 0.
3540 - val_loss: 3.4895 - val_categorical_accuracy: 0.8218
Epoch 14/25
175/175 [==============================] - ETA: 0s - loss: 4.7543 - categorical_accuracy: 0.3334
Learning rate for epoch 14 is 0.5
175/175 [==============================] - 44s 251ms/step - loss: 4.7543 - categorical_accuracy: 0.
3334 - val_loss: 3.4210 - val_categorical_accuracy: 0.1453
Epoch 15/25
175/175 [==============================] - ETA: 0s - loss: 4.6492 - categorical_accuracy: 0.3667
Learning rate for epoch 15 is 0.5
175/175 [==============================] - 45s 258ms/step - loss: 4.6492 - categorical_accuracy: 0.
3667 - val_loss: 3.5386 - val_categorical_accuracy: 0.8218
Epoch 16/25
175/175 [==============================] - ETA: 0s - loss: 4.6341 - categorical_accuracy: 0.2981
Learning rate for epoch 16 is 0.10000000149011612
175/175 [==============================] - 44s 252ms/step - loss: 4.6341 - categorical_accuracy: 0.
2981 - val_loss: 3.4817 - val_categorical_accuracy: 0.1453
Epoch 17/25
175/175 [==============================] - ETA: 0s - loss: 4.4673 - categorical_accuracy: 0.2618
Learning rate for epoch 17 is 0.10000000149011612
175/175 [==============================] - 43s 247ms/step - loss: 4.4673 - categorical_accuracy: 0.
2618 - val_loss: 3.5638 - val_categorical_accuracy: 0.1453
Epoch 18/25
175/175 [==============================] - ETA: 0s - loss: 3.5338 - categorical_accuracy: 0.1810
Learning rate for epoch 18 is 0.10000000149011612
175/175 [==============================] - 45s 258ms/step - loss: 3.5338 - categorical_accuracy: 0.
1810 - val_loss: 3.6142 - val_categorical_accuracy: 0.1453
Epoch 19/25
175/175 [==============================] - ETA: 0s - loss: 3.5588 - categorical_accuracy: 0.2009
Learning rate for epoch 19 is 0.10000000149011612
175/175 [==============================] - 44s 252ms/step - loss: 3.5588 - categorical_accuracy: 0.
2009 - val_loss: 3.4959 - val_categorical_accuracy: 0.1453
```

```
Epoch 20/25
175/175 [==============================] - ETA: 0s - loss: 3.5548 - categorical_accuracy: 0.2118
Learning rate for epoch 20 is 0.10000000149011612
175/175 [==============================] - 44s 252ms/step - loss: 3.5548 - categorical_accuracy: 0.
2118 - val_loss: 3.5696 - val_categorical_accuracy: 0.1453
Epoch 21/25
175/175 [==============================] - ETA: 0s - loss: 3.5584 - categorical_accuracy: 0.1456
Learning rate for epoch 21 is 0.10000000149011612
175/175 [==============================] - 44s 250ms/step - loss: 3.5584 - categorical_accuracy: 0.
1456 - val_loss: 3.6697 - val_categorical_accuracy: 0.1453
Epoch 22/25
175/175 [==============================] - ETA: 0s - loss: 3.5913 - categorical_accuracy: 0.1563
Learning rate for epoch 22 is 0.10000000149011612
175/175 [==============================] - 44s 253ms/step - loss: 3.5913 - categorical_accuracy: 0.
1563 - val_loss: 3.6644 - val_categorical_accuracy: 0.1453
Epoch 23/25
175/175 [==============================] - ETA: 0s - loss: 3.5935 - categorical_accuracy: 0.1658
Learning rate for epoch 23 is 0.10000000149011612
175/175 [==============================] - 44s 254ms/step - loss: 3.5935 - categorical_accuracy: 0.
1658 - val_loss: 3.5762 - val_categorical_accuracy: 0.1453
Epoch 24/25
175/175 [==============================] - ETA: 0s - loss: 3.5862 - categorical_accuracy: 0.1456
Learning rate for epoch 24 is 0.10000000149011612
175/175 [==============================] - 45s 256ms/step - loss: 3.5862 - categorical_accuracy: 0.
1456 - val_loss: 3.5885 - val_categorical_accuracy: 0.1453
Epoch 25/25
175/175 [==============================] - ETA: 0s - loss: 3.5950 - categorical_accuracy: 0.1679
Learning rate for epoch 25 is 0.10000000149011612
175/175 [==============================] - 43s 244ms/step - loss: 3.5950 - categorical_accuracy: 0.
1679 - val_loss: 3.5495 - val_categorical_accuracy: 0.1453
```

In [27]:

```
prediction = model.predict(x = {"image_input": X_test, "text_input": X_text_test})
prediction = np.array(prediction)
prediction = np.squeeze(prediction).T
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[:4,1], prediction[:4,1], average='micro')
macro_f1_score = f1_score(y_true[:4,1], prediction[:4,1], average='macro')

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

```
Micro F1 score for Task B is  1.0
Macro F1 score for Task B is  1.0
```

In [28]:

```
pd.DataFrame(history.history)
```

Out[28]:

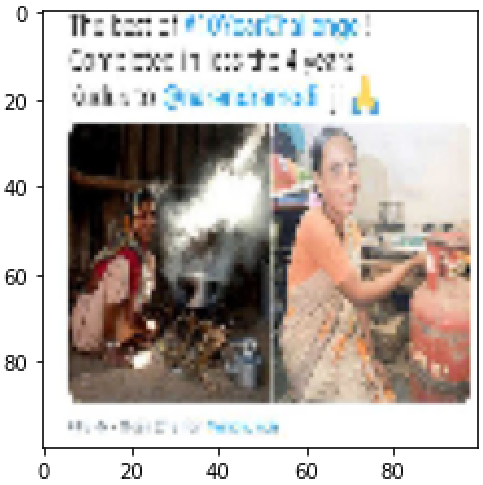
	loss	categorical_accuracy	val_loss	val_categorical_accuracy	lr
0	3.872755	0.331065	3.424437	0.145311	1.0
1	4.351704	0.349508	3.569532	0.821761	1.0
2	4.418116	0.360788	3.476143	0.145311	1.0
3	4.352265	0.354342	3.537557	0.821761	1.0
4	4.368616	0.385855	3.399115	0.821761	1.0
5	4.247776	0.380483	3.420226	0.145311	0.5
6	4.116553	0.337511	3.467620	0.145311	0.5
7	4.131020	0.347180	3.414467	0.145311	0.5
8	4.273046	0.340018	3.483443	0.145311	0.5
9	4.292561	0.356133	3.399493	0.821761	0.5
10	4.328044	0.358639	3.393518	0.821761	0.5
11	4.578453	0.380304	3.461632	0.145311	0.5
12	4.425720	0.353984	3.489510	0.821761	0.5
13	4.754329	0.333393	3.420993	0.145311	0.5
14	4.649230	0.366697	3.538564	0.821761	0.5
15	4.634129	0.298120	3.481671	0.145311	0.1
16	4.467349	0.261773	3.563781	0.145311	0.1
17	3.533846	0.181021	3.614153	0.145311	0.1
18	3.558821	0.200895	3.495899	0.145311	0.1
19	3.554817	0.211817	3.569575	0.145311	0.1
20	3.558445	0.145568	3.669712	0.145311	0.1
21	3.591279	0.156312	3.664448	0.145311	0.1
22	3.593545	0.165801	3.576198	0.145311	0.1
23	3.586208	0.145568	3.588499	0.145311	0.1
24	3.595023	0.167950	3.549543	0.145311	0.1

In [29]:

```
plt.imshow(X[1,:,:,:])
target.iloc[1,:]
```

Out[29]:

```
humour          0
sarcasm         1
offensive       0
motivational    1
Name: 1, dtype: int64
```



In [30]:

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

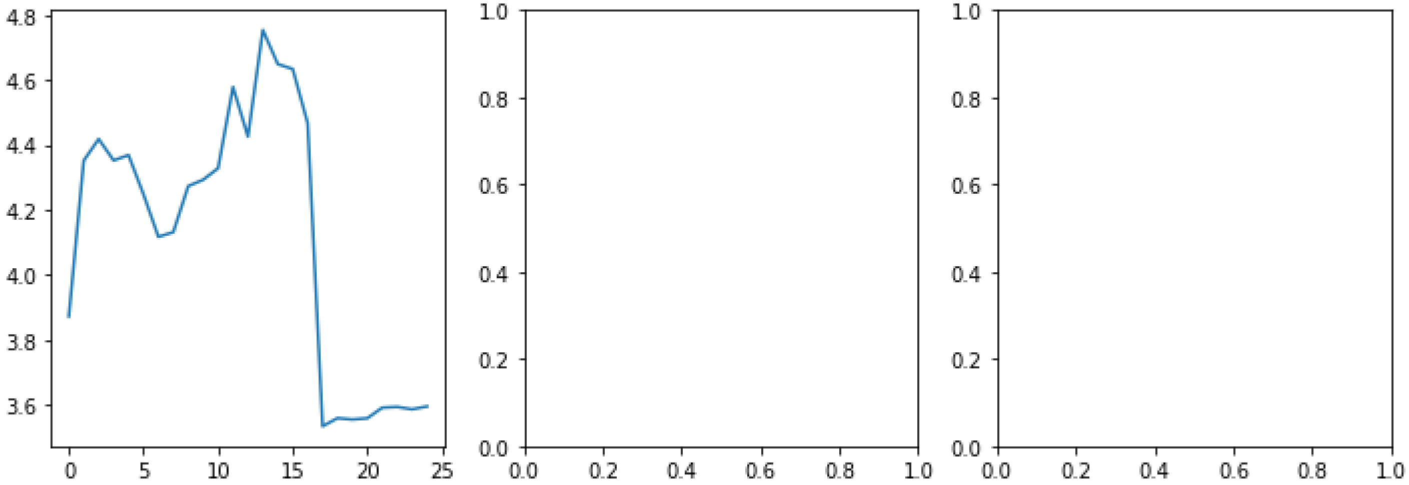
```
In [31]: plt.bar(['humuor', 'sarcasm', 'offensive', 'motivational'], np.where(prediction[:,1,0] > 0.5, 1, 0))
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-31-6fe2b4939f1b> in <module>
----> 1 plt.bar(['humuor', 'sarcasm', 'offensive', 'motivational'], np.where(prediction[:,1,0] > 0.
5, 1, 0))

IndexError: too many indices for array
```

In [32]:

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

fig, axes = plt.subplots(1,3, figsize=(12, 4))

axes[0].plot(df.loss)
axes[0].plot(df.humuor_loss)
axes[0].plot(df.sarcasm_loss)
axes[0].plot(df.offensive_loss)
axes[0].plot(df.motivational_loss)
axes[0].set_xlabel('Epochs')
axes[0].set_ylabel('Losses')
axes[0].set_title('Losses Per Epoch')
axes[0].legend(['Humuor loss', 'Sarcasm loss','Offensive loss','Motivational Loss'], loc='upper right'
)



axes[1].plot(df.humuor_accuracy)
axes[1].plot(df.sarcasm_accuracy)
axes[1].plot(df.offensive_accuracy)
axes[1].plot(df.motivational_accuracy)
axes[1].set_xlabel('Epochs')
axes[1].set_ylabel('Accuracy')
axes[1].set_title('Accuracy Per Epoch')
axes[1].legend(['Humuor Acc', 'Sarcasm Acc','Offensive Acc','Motivational Acc'], loc='lower right')




axes[2].plot(df.loss)
axes[2].set_xlabel('Epochs')
axes[2].set_ylabel('Losses')
axes[2].set_title('Losses Per Epoch')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-32-3070f1fd3cf9> in <module>
      4 
      5 axes[0].plot(df.loss)
----> 6 axes[0].plot(df.humuor_loss)
      7 axes[0].plot(df.sarcasm_loss)
      8 axes[0].plot(df.offensive_loss)

/opt/conda/lib/python3.7/site-packages/pandas/core/generic.py in __getattr__(self, name)
   5137             if self._info_axis._can_hold_identifiers_and_holds_name(name):
   5138                 return self[name]
-> 5139             return object.__getattribute__(self, name)
   5140 
   5141     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'humuor_loss'
```



In [33]:

```
test_images = X_test.shape[0]

random_index = np.random.choice(test_images, 5)
random_test_images = X_test[random_index, ...]
random_test_labels = (y_test.humour[random_index, ...],
                      y_test.sarcasm[random_index, ...],
                      y_test.offensive[random_index, ...],
                      y_test.motivational[random_index, ...])

predictions = model.predict(random_test_images)

fig, axes = plt.subplots(5, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)

for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, random_test_labels
)):
    axes[i, 0].imshow(np.squeeze(image))
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-33-6785b07a2b49> in <module>
      3 random_index = np.random.choice(test_images, 5)
      4 random_test_images = X_test[random_index, ...]
----> 5 random_test_labels = (y_test.humour[random_index, ...],
      6                       y_test.sarcasm[random_index, ...],
      7                       y_test.offensive[random_index, ...],

/opt/conda/lib/python3.7/site-packages/pandas/core/series.py in __getitem__(self, key)
    904             return self._get_values(key)
    905 
--> 906         return self._get_with(key)
    907 
    908     def _get_with(self, key):

/opt/conda/lib/python3.7/site-packages/pandas/core/series.py in _get_with(self, key)
    919             )
    920         elif isinstance(key, tuple):
--> 921             return self._get_values_tuple(key)
    922 
    923         elif not is_list_like(key):

/opt/conda/lib/python3.7/site-packages/pandas/core/series.py in _get_values_tuple(self, key)
    954 
    955         if not isinstance(self.index, MultiIndex):
--> 956             raise ValueError("key of type tuple not found and not a MultiIndex")
    957 
    958         # If key is contained, would have returned by now

ValueError: key of type tuple not found and not a MultiIndex
```

In [ ]: