Multimodal model

Loading the Data

```
!gdown --id 15LVlv7K31SzBs_IyZE_md0u6qEg6rby6
!unzip dataminingmtl782.zip
```

Importing Required Libraries

```
!pip install -q wordcloud
import wordcloud

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.stem.wordnet import WordNetLemmatizer
import pickle
import cv2
import re
from pylab import rcParams
from matplotlib import rc
from nltk.tokenize import word_tokenize
from textblob import TextBlob
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, plot_confusion_matrix
from tqdm import tqdm
import torch
from torch import nn, optim
from torch.utils import data
%matplotlib inline
%config InlineBackend.figure_formate = 'retina'
sns.set(style = 'whitegrid', palette = 'muted', font_scale = 1.2)
HAPPY_COLOURS_PALETTE = ['#01BEFE', '#FFDD00', '#FF7D00', '#FF006D', '#ADFF02', '#8F00FF']
sns.set_palette(sns.color_palette(HAPPY_COLOURS_PALETTE))
rcParams['figure.figsize'] = 8,8
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
```

Loading the file Data-frames

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
train_df
```

Setting Hyper Parameters

```
IMG_SCALING = (3, 3)
BATCH_SIZE = 48
GAUSSIAN_NOISE = 0.1
N_CLASSES = 3
dim = (192,192)
MAIN_PATH = ''
```

Exploratory Data Analysis and Text pre-processing

```
print(train_df['image id'].duplicated().any()) # Check if there are any duplicated images in the data
train_df['label'].value_counts().plot(kind = 'barh')# Plot the counts of each label
train_df['label_num'].value_counts().plot(kind = 'barh') # Plot the counts of each label_num
```

```
sns.countplot(train_df.label_num)
plt.xlabel('Sentiment')
```

```
train_df['text'] = train_df['text'].apply(lambda x:x.lower())
test_df['text'] = test_df['text'].apply(lambda x: x.lower())# Convert the text in lower case
print(train_df[train_df['text']=='#name?']) # Check if there are any empty text images in the dataset
print(test_df[test_df['text'] == '#name?'])
train_df['text']
```

```
train_df.isnull().sum() #Check number of null entries in the dataframe
```

```
train_df.drop(858,inplace=True)# Remove the corrupted Image
train_df[ train_df['image id']=='image_6357.jpg' ]
```

```
num = 858#Check that the corrupted image has been removed
train_df['label'].iloc[num]
img = cv2.imread(MAIN_PATH +'train_images/train_images/'+train_df['image id'].iloc[num])
plt.figure(figsize = (12*1.2,8*1.2))
plt.imshow(img)
plt.axis('off')
```

```
for i,id in enumerate(train_df['image id']):
    try:
        img = cv2.imread(MAIN_PATH +'train_images/train_images/'+id)
        if(img.shape[2]!=3):
            print(id + ' ' + img.shape)
    except:
        print(id+' '+ str(i))
# Verify that all the images that are read are of dimension(None,None,3)
```

```
for text,i in zip(train_df['text'],train_df['ID']):
    print(text)
# Go through the text and see what kind of processing is needed
```

```
# Text cleaning very important
def clean_str(string):
    string = re.sub(r"\n", " ", string)
    string = re.sub(r"\r", " ", string)
    string = re.sub(r"[0-9]", "digit", string)# Replace digits with the word digit
    string = re.sub(r"\'", " ", string)
    string = re.sub(r"\"", " ", string)
    string = re.sub(r"\?", " ", string)
    string = re.sub(r"\!", " ", string)
    string = re.sub(r"\/", " ", string)
    string = re.sub(r"\\", " ", string)
    string = re.sub(r"\.", " ", string)
    sample = re.sub(r'''(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()
<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]
{};:'".,<>?«»“”‘’]))''', " ", string)# Remove hyperlinks
    sample = re.sub(r"http\S+", "", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+.COM","", sample)# Remove the hyperlinks with www and
all the possible missing combintations of .
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"www [a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"WWW [a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"www [a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"WWW [a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"www [a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"WWW [a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"www [a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"WWW [a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"WWW[a-zA-Z0-9_@]+com","", sample)
    sample = re.sub(r"www[a-zA-Z0-9_@]+com","", sample)
    sample = re.sub(r"WWW[a-zA-Z0-9_@]+COM","", sample)
    sample = re.sub(r"www[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"WWW[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"www[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"WWW[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+COM","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+com","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+com","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+COM","", sample)
```

```
    sample = re.sub(r"http\S+", "", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+ COM","", sample)
    return sample
    # string = re.sub(r'^https?:\/\/.*[\r\n]*', '', string, flags=re.MULTILINE) # WILL REMOVE
HYPERLINKS!!!!
    return string.strip().lower()


train_df['text'] = train_df['text'].apply(clean_str) # Should be very very clean
test_df['text'] = test_df['text'].apply(clean_str)
```

```
import nltk
nltk.download('punkt')
def TOK(text):
    return word_tokenize(text.lower())
train_df['tokens'] = train_df.text.apply(TOK)

train_df
```

```
test_df['tokens'] = test_df.text.apply(TOK)

test_df
```

```
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = [word.lower() for word in stopwords.words('english')]
def remove_stopwords(tokens):
  ans = []
  for tok in tokens:
    if tok in stop_words:
      continue
    pas = True
    for s in '@!`~\'\"\\*&^%$#-=+[].,<>?':
      if s in tok:
        pas = False
        break
    if not pas:
      continue
    if 'meme' in tok:
      continue
    ans.append(tok)
  return ans
# print(train_df.tokens.iloc[599],remove_stopwords(train_df.tokens.iloc[599]))
```

```
train_df.tokens = train_df.tokens.apply(remove_stopwords)
test_df.tokens = test_df.tokens.apply(remove_stopwords)
```

```
def Tokens_to_text(tokens):
    s = ''
    for token in tokens:
        s+=token+' '
    return s.lower().strip()
def maxLen(df):
    x = 0
    for i, tokens in enumerate(df.tokens):
        x = max(x, len(tokens))
    return x
max_words = maxLen(train_df)
```

```
for i, text in enumerate(train_df.text):
  print(i, text)
```

```
train_df.text = train_df.tokens.apply(Tokens_to_text)
test_df.text = test_df.tokens.apply(Tokens_to_text)
```

```
for i, text in enumerate(train_df.text):
  print(i, text)
```

Before processing vs after processing

```
Before: 3...2...1..dont do it timmy!
After: dont do it timmy
Before: a vote for trump is a vote for putin 2009 www.protectourelections.com
After a vote for trump is a vote for putin 2009

Before: “i start where the last man left off.” thomas edison visit: www.cettechnology.com/memes
for more quotes © techsolmarketing.com - free for use without modification
After: i start where the last man left off thomas edison visit: memes for more quotes © free for
use without modification
```

Import Sentence Transformer model

```
!pip install sentence_transformers
from sentence_transformers import SentenceTransformer
text_model= SentenceTransformer('distilbert-base-nli-stsb-mean-tokens')
```

Getting Encoder for images

```
from keras import Model
from keras.applications import VGG16
from keras.preprocessing.image import ImageDataGenerator
```

```
from keras.layers import GlobalAveragePooling2D
def get_encoder(old_model: Model) -> Model:
  # Get encoder
  encoder_input: Model = Model(inputs=old_model.layers[0].input,
                               outputs=old_model.layers[10].output)

  # Create Global Average Pooling.
  encoder_output = GlobalAveragePooling2D()(encoder_input.layers[-1].output)

  # Create the encoder adding the GAP layer as output.
  encoder: Model = Model(encoder_input.input, encoder_output, name='encoder')

  return encoder
DenseNetModel = VGG16(include_top=False, input_shape=(*dim,3))
DenseNetModel = get_encoder(DenseNetModel)
```

Creating DataGenerator Class

```
import keras
args = dict(featurewise_center = False,
                  samplewise_center = False,
                  rotation_range = 45,
                  shear_range = 0.01,
                  fill_mode = 'reflect',
                   data_format = 'channels_last')
class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, labels, batch_size=32, dim=(192,192), n_channels=1,
                 n_classes=2, shuffle=True):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.labels = labels
        self.list_IDs = list_IDs
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # Find list of IDs
        list_IDs_temp = [self.list_IDs.iloc[k] for k in indexes]

        # Generate data
        X, y = self.__data_generation(list_IDs_temp)

        return X, y

    def on_epoch_end(self):
```

```
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)

    def __data_generation(self, list_IDs_temp):
        'Generates data containing batch_size samples' # X : (n_samples, *dim, n_channels)
        # Initialization
#         X = np.empty((self.batch_size, *self.dim, self.n_channels))
#         y = np.empty((self.batch_size,*self.dim,1))
        X_img = []
        X_text = []
        y = []

        # Generate data
        for i, ID in enumerate(list_IDs_temp):
            # Store sample
            # print(ID)
            img = cv2.imread( MAIN_PATH +'train_images/train_images/'  + ID['image id'] )
            img = cv2.resize(img,self.dim)

            X_img.append(img/255.0)
            X_text.append(np.array(text_model.encode(ID['text'])))

            # Store class
            y.append(self.labels[ID['image id']])

        # X = np.stack(X,0)/255.0 # [(*dim,3)] of len(batch_size) -> (batch_size,*dim,3)
        # X_text = sequence.pad_sequences(tok.texts_to_sequences(X_text),maxlen=max_len)
        datagen = ImageDataGenerator(**args)
        samples = np.array(X_img)
        it = datagen.flow(samples, batch_size = self.batch_size, shuffle = False)
        X = [np.stack(it.next()),np.stack(X_text)]
        y = np.stack(y,0)
        return X, y
params = {'dim': dim,
      'batch_size': BATCH_SIZE,
      'n_classes': 3,
      'n_channels': 3,
      'shuffle': True}
train_df,val_df = train_test_split(train_df)
# Datasets
# all_groups = list(train_data.groupby("ImageId"))
# print(all_groups[3][1]['EncodedPixels'])
partition = train_df[["image id","text"]]
labels = train_df[["image id","label_num"]].set_index("image id")["label_num"]
# # Generators
training_generator = DataGenerator(partition, labels, **params)
# validation_generator = DataGenerator(partition['validation'], labels, **params)
partition = val_df[["image id","text"]]
labels = val_df[["image id","label_num"]].set_index("image id")["label_num"]
# # Generators
validation_generator = DataGenerator(partition, labels, **params)
```

Final Model

```
from keras.layers import Input, Dense, LSTM, Dropout, Activation, Embedding
```

```
from keras.losses import SparseCategoricalCrossentropy
from keras.optimizers import Adam
max_len = 514
sequence_input = Input(name='text_inputs',shape=[768])
# embedding_layer = Embedding(max_words, 50,input_length=max_len,trainable=True)
# embedded_sequences = embedding_layer(sequence_input)
# layer = LSTM(16)(embedded_sequences)

layer = Dense(16,name='FC1')(sequence_input)
layer = Activation('relu')(layer)
layer = Dropout(0.5)(layer)
# layer = sequence_input
img_input = DenseNetModel.input
 # Add two more additional layers
x = DenseNetModel.output
x = keras.layers.Dense(16, activation='relu')(x)
x = Dropout(0.5)(x)
x = keras.layers.Dense(16, activation='relu')(x)
x = Dropout(0.5)(x)

x = keras.layers.Concatenate()([layer,x])
x = keras.layers.Dense(3, activation = 'sigmoid')(x)
# IMPORTANT : set the layers as untrainable.
for layer in DenseNetModel.layers:
    layer.trainable = False
    # Compile the model

model = Model([img_input,sequence_input], x)
model.compile(loss=SparseCategoricalCrossentropy(),optimizer= Adam(), metrics=['accuracy'])
```

Adding Callbacks

```
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping,
ReduceLROnPlateau
weight_path="{}_weights.best.hdf5".format('seg_model')

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1, save_best_only=True,
mode='min', save_weights_only=True)

reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.33,
                                   patience=1, verbose=1, mode='min',
                                   min_delta=0.0001, cooldown=0, min_lr=1e-8)

early = EarlyStopping(monitor="val_loss", mode="min", verbose=2,
                      patience=20) # probably needs to be more patient, but kaggle time is
limited

callbacks_list = [checkpoint,  reduceLROnPlat, early]
```

Training

```
n_epoch = 10
history = model.fit_generator(training_generator,epochs
=n_epoch,verbose=1,validation_data=validation_generator,callbacks=callbacks_list)
```

Plotting history

```
def plot_history(history):
    loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' not in s]
    val_loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' in s]
    acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' not in s]
    val_acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' in s]

    if len(loss_list) == 0:
        print('Loss is missing in history')
        return

    ## As loss always exists
    epochs = range(1,len(history.history[loss_list[0]]) + 1)

    ## Loss
    plt.figure(1)
    for l in loss_list:
        plt.plot(epochs, history.history[l], 'b', label='Training loss (' +
str(str(format(history.history[l][-1],'.5f'))+')'))
    for l in val_loss_list:
        plt.plot(epochs, history.history[l], 'g', label='Validation loss (' +
str(str(format(history.history[l][-1],'.5f'))+')'))

    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    ## Accuracy
    plt.figure(2)
    for l in acc_list:
        plt.plot(epochs, history.history[l], 'b', label='Training accuracy (' +
str(format(history.history[l][-1],'.5f'))+')')
    for l in val_acc_list:
        plt.plot(epochs, history.history[l], 'g', label='Validation accuracy (' +
str(format(history.history[l][-1],'.5f'))+')')

    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
plot_history(history)
```

Making Predictions

```
X_img = []
X_text = []
```

```
for i in range(len(test_df)):
    row = test_df.iloc[i]
    # Store sample
    # print(ID)
    img = cv2.imread( MAIN_PATH +'test_images/'  + row['image id'] )
    img = cv2.resize(img,dim)

    X_img.append(img/255.0)
    X_text.append(np.array(text_model.encode(row['text'])))

# X = np.stack(X,0)/255.0 # [(*dim,3)] of len(batch_size) -> (batch_size,*dim,3)
# X_text = sequence.pad_sequences(tok.texts_to_sequences(X_text),maxlen=max_len)
datagen = ImageDataGenerator(**args)
it = datagen.flow(np.array(X_img),shuffle = False,batch_size = len(X_img))
X = [np.stack(it.next()),np.stack(X_text)]
# y_pred = model.predict(X)
y_pred = model.predict(X).argmax(axis = -1)
```

Making Submission

```
submit = pd.read_csv('Sample_submission.csv')
submit['label_num'] = y_pred
submit.to_csv('Submission.csv', index = False)
```