50.008 Computational Data Science

Emotion Analysis And Image Captioning In TV Series: Context At A Glance

Ivan Christian 1003056
Ho Jin Kind 1002846
Ng Jen Yang 1003007

Abstract

A multi-modal emotional analysis model, along with image captioning model, done periodically over the span of a television series episode. Using the combination of the aforementioned model the project has combined the components to inform the consumers of the intended scene and emotion context.

Introduction

To be able to gain the context of what is going on in a tv series with minimal amount attention span is key when watching a television series. Nowadays, people choose to multi-task, and the ability to gain some context of what is going on in a show while doing other work can assist in increasing productivity while also making sure the audience does not miss major parts of the context of the episode.

Motivation

Inspired by the work of Soujanya Poria, Multimodal Multi-Party Dataset for Emotion Recognition in Conversations[1], and by making use of emotion recognition along with image captioning, We can give a user sufficient context of what is going on during a television series ata glance.

A conversational emotion recognition system can be used to generate appropriate responses by analyzing user emotions.

Furthermore, understanding emotiion in the television series also has value in that it can also help producers classify on the different television show genres which attract a higher viewership.

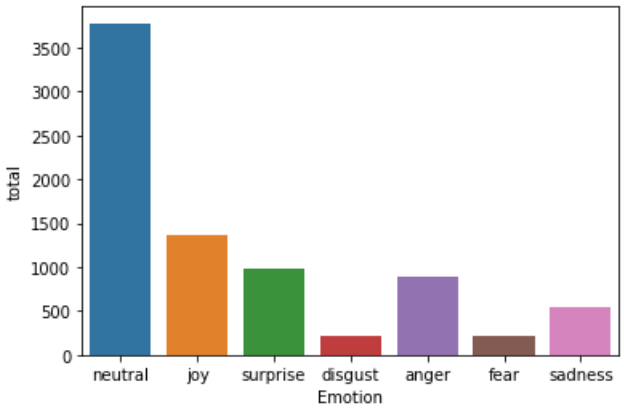
Project Overview

The project uses various model to solve the initial issue

Dataset and Collection

Emotion Analysis

For the Emotion Analysis for both Audio and text, we used the Dataset from MELD[1]. The dataset contains a total of 10,000 utterances which are labelled, the mp4 for the video of each utterance, and csv with labels of emotion.



This is a breakdown of the dataset. As there is an inbalance in the data, skewed towards a large number of neutral, this could later affect the accuracy of our model.

Word Embeddings (Text):

We used the pretrained word vectors from the GloVe dataset for the generation of word embeddings used in the model. We decided to use the 300dim word vectors, as the word vectors with lower dimensions would return a lower accuracy and F-score in our testing.

Image Captioning (Captions):

Dataset for the training is not as easily available should the the dataset is scraped and gathered by oneself. As such the project will be using the Flickr 8k dataset, which is especially designed for image captioning. The dataset consist of a folder of 8092 JPG images as well as documents containing the captions which correspond to the images. Each image has 5 corresponding captions which can be seen in the following format, where the format of every line contains the `<image name>#i <caption>`, where `0 ≤ i ≤ 4`:

```
...
1000268201_693b08cb0e.jpg#0 A child in a pink dress is climbing up a set of stairs in
1000268201_693b08cb0e.jpg#1 A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2 A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3 A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4 A little girl in a pink dress going into a wooden cabin .
...
```

The each image is given an id which corresponds to one corresponding caption which will be used for training.

While all the captions can be found in the text file: `Flickr8k.lemma.token.txt`
The dataset has already been seperated into train, test and dev images and hence cann be found seperately in `Flickr_8k.trainImages`, `Flickr_8k.testImages`, and `Flickr_8k.devImages`

Image Captioning (Images):

This dataset contains 8000 images each with 5 captions, as seen previous that an image can have multiple captions.

These images are divided witht the following ratios:

```
Training Set — 6000 images
Dev Set — 1000 images
Test Set — 1000 images
```

An example can be seen below taken from the training set can be seen below with its corresponding caption:

“A girl wear a red and multicolored bikini be lay on her back in shallow water .”

The datasets can be found in https://www.kaggle.com/ming666/flicker8k-dataset#Flickr8k_text.zip or https://forms.illinois.edu/sec/1713398

Data Pre-Processing(Text)

Pre-processing for the text involved simple cleaning of two things: the Friends dataset, as well as the pretrained word vectors dataset.
Cleaning up the Friends dataset involved removing non-ACSII characters, and processing the pretrained word vectors involved creating a dictionary to hold the word, and the associated vector. Next, each sentence was tokenized by creating a Tokenizer object, and fitting it on the utterances in the dataset. After which, each stream of token was converted into a sequence, as per :

```
tokenizer = Tokenizer(num_words= vocab_size)
tokenizer.fit_on_texts(data['Utterance'])
sequences = tokenizer.texts_to_sequences(data['Utterance'])
```

A dictionary containing a one-to-one matching of every unique word to an integer value is then created, as per:

```
word_index = tokenizer.word_index
```

Finally, the utterances were created after padding (to ensure constant length).

```
utterances = sequence.pad_sequences(sequences, padding='post', maxlen=50)
```

Encoding for emotions was done in the same manner for both text and audio; both used a dictionary to map emotions to a value.

```
dict_emotion = {'anger':0, 'neutral':1, 'surprise':2,'joy':3, 'sadness':4,
               'fear':5,'disgust':6}
def onehotemo(emo,dict_emotion):
    return(dict_emotion[emo])
data['emoenc']= data['Emotion'].apply(lambda x: onehotemo(x,dict_emotion))
labels = data['emoenc']
```

We then separated the dataset into training and testing data, then converted each emotion label from an integer into a one-hot/binary array, as per:

```
x_train, x_test, y_train, y_test = train_test_split(utterances, labels, test_size=0.2
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)
```

After that, the utterances had to have each word replaced with the respective word vector in the pretrained word vectors dataset. For the words that did not appear in the dataset, the vector representing the words was randomly generated, then guaranteed to be unique by referencing against a list of previous vectors, then storing the word and vector as a key value pair in a dictionary.
Code for random unique vector generation:

```
previous_random_vectors={}
unique_vectors = []
def create_unique_random_xdim_vector(dimensions, word,previous_vectors,unique_vectors
    flag = True
    while flag == True:
        rarray = np.random.rand(300,)
        if any(np.array_equal(rarray,x) for x in unique_vectors):
            print('array created before,creating new one again')
        else:
            flag = False
            previous_vectors[word] = rarray
            unique_vectors.append(rarray)
    return rarray,previous_vectors,unique_vectors
```

Code for embedding generation:

```
embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
    else:
        embedding_matrix[i],previous_random_vectors,unique_vectors = create_unique_ra
        print('random vector created for ' + word)
```

Now we have all data embedded, and ready to train the emotion prediction model on the samples.

Data Pre-Processing(Audio)

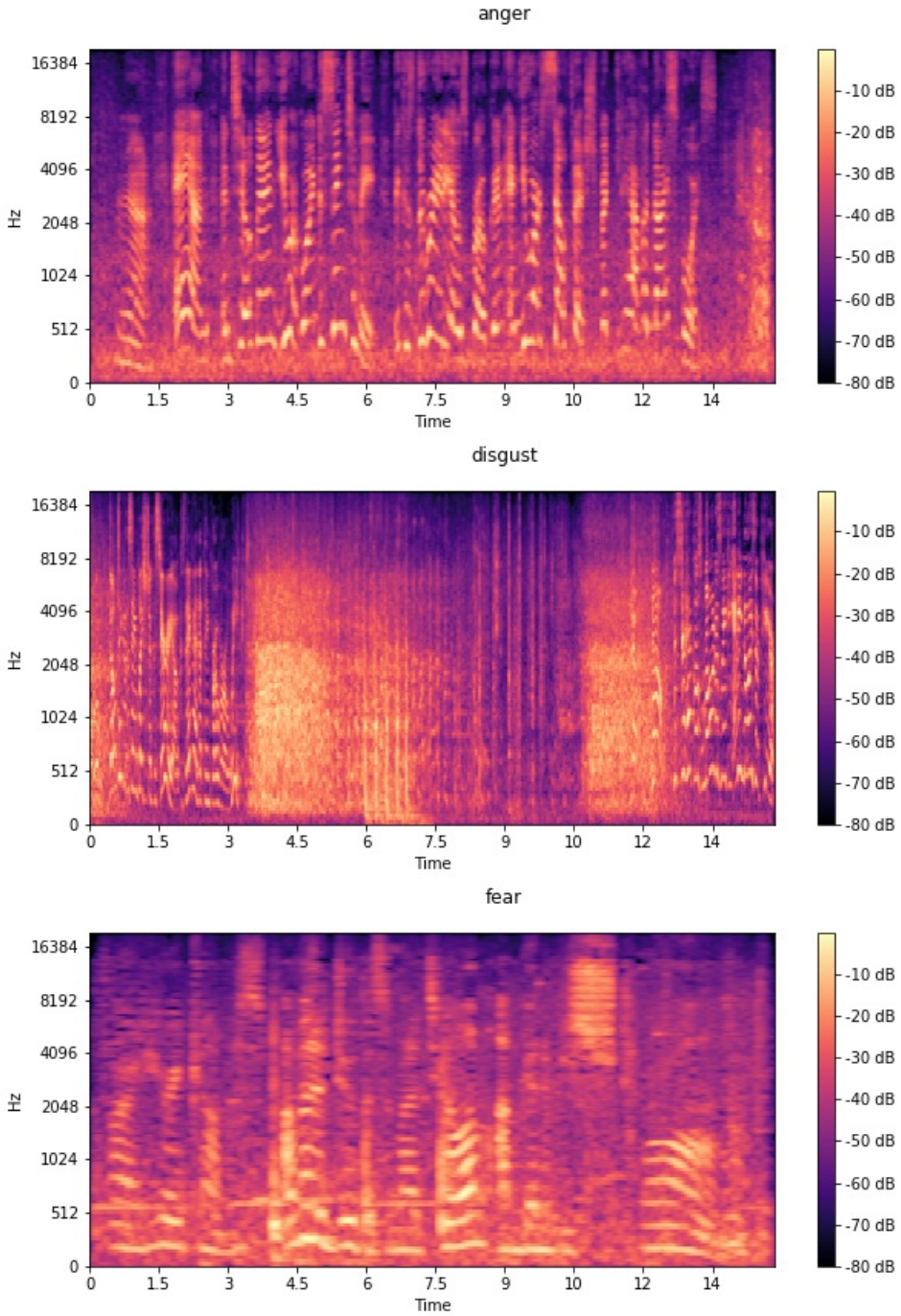
We used spectograms to analyse the audio, and with the raw mp4 files, have to convert them into mp3 format.

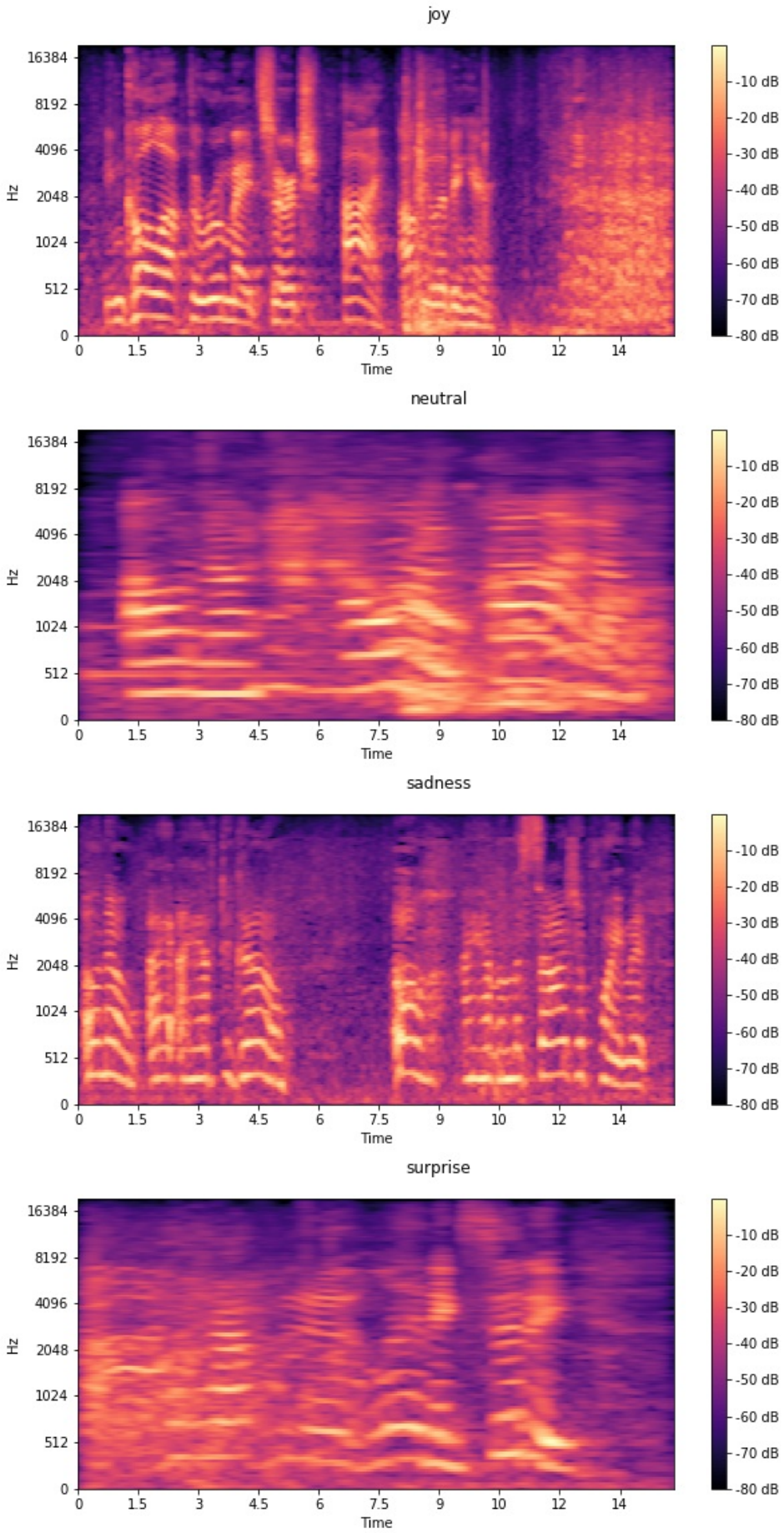
Each audio file was then converted into a spectogram which is a visual representation of the spectrum

of frequencies over time. A regular spectogram is squared magnitude of the short term Fourier transform (STFT) of the audio signal. This regular spectogram is squashed using mel scale to convert the audio frequencies into something more visually understandable.

Data Visualisation(Audio)

By analysing the spectogram, we can see if there are distinct differences between the different emotions from the mel-spectogram, and get a sense of the level of accuracy we can achieve using a CNN to do the classification.





From the plots, we can see some form of distinction between the emotions, albeit not defined. Also, as noted in the ‘disgust’ audio track, it was filled with audience laughter as evident from 3.5s to 7s and 10s to 13s. Such laughter within the dataset may result in more noise and lower accuracy when training with CNN. Due to the varying lengths of each audio file, but fixed size array that is required, we had to resize the spectograms to all be 640 by 128.

```
...
def create_spectogram(filename):
    fullpathfilename = get_audio_path(AUDIO_DIR, filename)
    y, sr = librosa.load(fullpathfilename)
    duration= librosa.get_duration(y=y, sr=sr)
    spect = librosa.feature.melspectrogram(y=y, sr=sr,n_fft=2048, hop_length=int(dura
    spect = librosa.power_to_db(spect, ref=np.max)
    return cv2.resize(spect.T,(128,640))
...
```

We then zip the numpy arrray with both the spectogram along with labels into a npz file.

Data Pre-Processing (Images):

In order to process the images for training, we use keras' preprocessing function to preprocess the image and make sure that the images are the correct dimension as the inception model needed. For the inception model the target input dimension is (width=299,height=299) while the output dimension is 2048.

```
...
WIDTH = 299
HEIGHT = 299
OUTPUT_DIM = 2048
preprocess_input = tensorflow.keras.applications.inception_v3.preprocess_input
...
```

Once the width and height are specified, the following function changes firstly the dimension of the image and preprocessed the image such that it is encoded. The following is the function statement in the code:

```
def encodeImage(img):
  img = img.resize((WIDTH, HEIGHT), Image.ANTIALIAS)
  x = tensorflow.keras.preprocessing.image.img_to_array(img)
  # Expand to 2D array
  x = np.expand_dims(x, axis=0)
  x = preprocess_input(x)
  x = encode_model.predict(x) # Get the encoding vector for the image
  x = np.reshape(x, OUTPUT_DIM )
  return x
```

This function itself mention an encoder model which will be explained in the following parts of the report but for data pre-processing the image is resized to the specified width and height, which then is immediately preprocessed to become array for further use. Once the array is created, it is expanded further into a 2D array. Using the preprocessing defined in the tensorflow library specifically for the inception model, the images are inputted to obtain the encoding vectors for for the images to be used for training and caption generation. It is to be noted that the images are antialiased so that it is smoother and lacks jaggedness during training.

Data Pre-processing (Caption Text):

Pre-processing on the caption text includes cleaning out the punctuations, splitting up the text based on `'\n'`, and making sure that everything is in lowercase as seen in the following function:

```
def clean_caption_text():
  null_punct = str.maketrans('', '', string.punctuation)
  lookup = dict()
  with open( os.path.join(root_captioning,'Text_Files','Flickr8k.token.txt'), 'r') as
    max_length = 0
    for line in fp.read().split('\n'):
      tok = line.split()
      if len(line) >= 2:
        id = tok[0].split('.')[0]
        desc = tok[1:]

        # Cleanup description
        desc = [word.lower() for word in desc]
        desc = [w.translate(null_punct) for w in desc]
        desc = [word for word in desc if len(word)>1]
        desc = [word for word in desc if word.isalpha()]
        max_length = max(max_length,len(desc))

        if id not in lookup:
          lookup[id] = list()
        lookup[id].append(' '.join(desc))

  lex = set()
  for key in lookup:
    [lex.update(d.split()) for d in lookup[key]]

  return lookup, lex, max_length
```

Once the pre-processing is done the train captions are then further processed by checking the number of times words appear in the documents. The following code makes sure that anything less than the `word_count_threshold` is rooted out:

```
word_count_threshold = 25
word_counts = {}
nsents = 0
for sent in all_train_captions:
    nsents += 1
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
print('preprocessed words %d ==> %d' % (len(word_counts), len(vocab)))
```

To improve the model, words that appear less than the `word_count_threshold` are rooted out from training. This is done so that the model does not learn to associate niche words to certain features of the image and generate incorrect captions for said image. This, in theory, will improve the accuracy of the model as the model will no longer be able to associate image features to niche words that do not appear above the threshold. Different threshold are used to see which word threshold generates the best accuracy for the model.

Once the words have finnished being filtered, they are then embedded based on the `glob.840B.300d` dataset taken from GloVe, as seen in the following:

```
#Word Embeddings from glove. Reason why we are using 840B 300d is so that the caption

glove_dir = os.path.join(root_captioning,'glob.6B')
embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.840B.300d.txt'), encoding="utf-8")
try:
  for line in tqdm(f):
      values = line.split()
      word = values[0]
      coefs = np.asarray(values[1:], dtype='float32')
      embeddings_index[word] = coefs
except ValueError:
  pass

f.close()
print(f'Found {len(embeddings_index)} word vectors.')
```

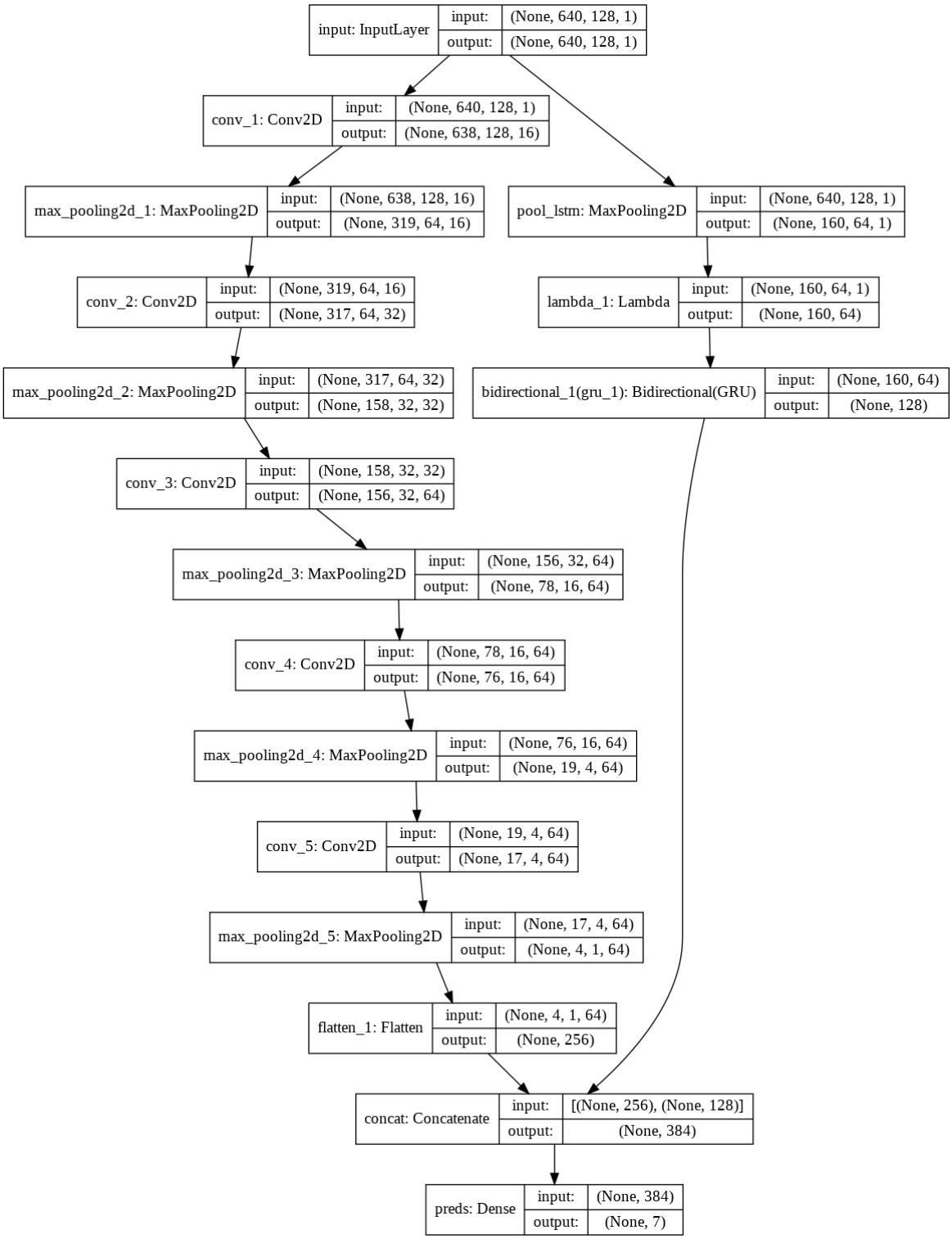
The embedding is done such that each words has its own vectors and the filtered words that do not appear in the in the dataset, which is highly unlikely due to teh sheer size and robustnes of the glove dataset, has a vector of zero. For the image captioning side of the project, the data preprocessing can now generated trainable data for the model.

Model

CNN-LSTM Static Model: Text

Training the model to predict emotion on text strings used a very simple approach: a CNN block of one convolution and MaxPooling layer, does feature extraction and down-samples the input. Then, the outputs of the CNN block are pushed into a Dropout layer. This is to prevent overfitting for the model, as the training accuracies observed are quite high, while test accuracies are significantly lower than the training accuracies. Afterward, the inputs(after dropout regularization), are passed into a LSTM layer, where we obtained the trained weights via a Dense layer of size 7, after applying softmax activation.
The model was trained using ADAM optimizer and the loss function was categorical cross entropy. The model is trained with 50 epochs.

CNN-LSTM Parallel Model: Audio



The inspiration of this model comes from the work by deep sounds from the paper by Lin Feng[2]. This paper used a parallel CRNN architecture to predict genres in music.

For the CNN block of our proposed CRNN architecture has 5 layers, including five convolutional-pooling layers. After each convolutional layer, the max-pooling operation is followed to further process the output of previous convolutional layer

The result from this block is flattened and is a tensor of shape None , 256.

The recurrent block starts with 2D max pooling layer of pool size 4,2 to reduce the size of the spectogram before LSTM operation. This feature reduction was done primarily to speed up processing.

The reduced image is sent to a bidirectional GRU with 64 units. The output from this layer is a tensor of shape None, 128.

The outputs from the convolutional and recurrent blocks are then concatenated resulting in a tensor of shape, None, 384. Finally we have a dense layer with softmax activation.

The model was trained using RMSProp optimizer with a learning rate of 0.0005 and the loss function

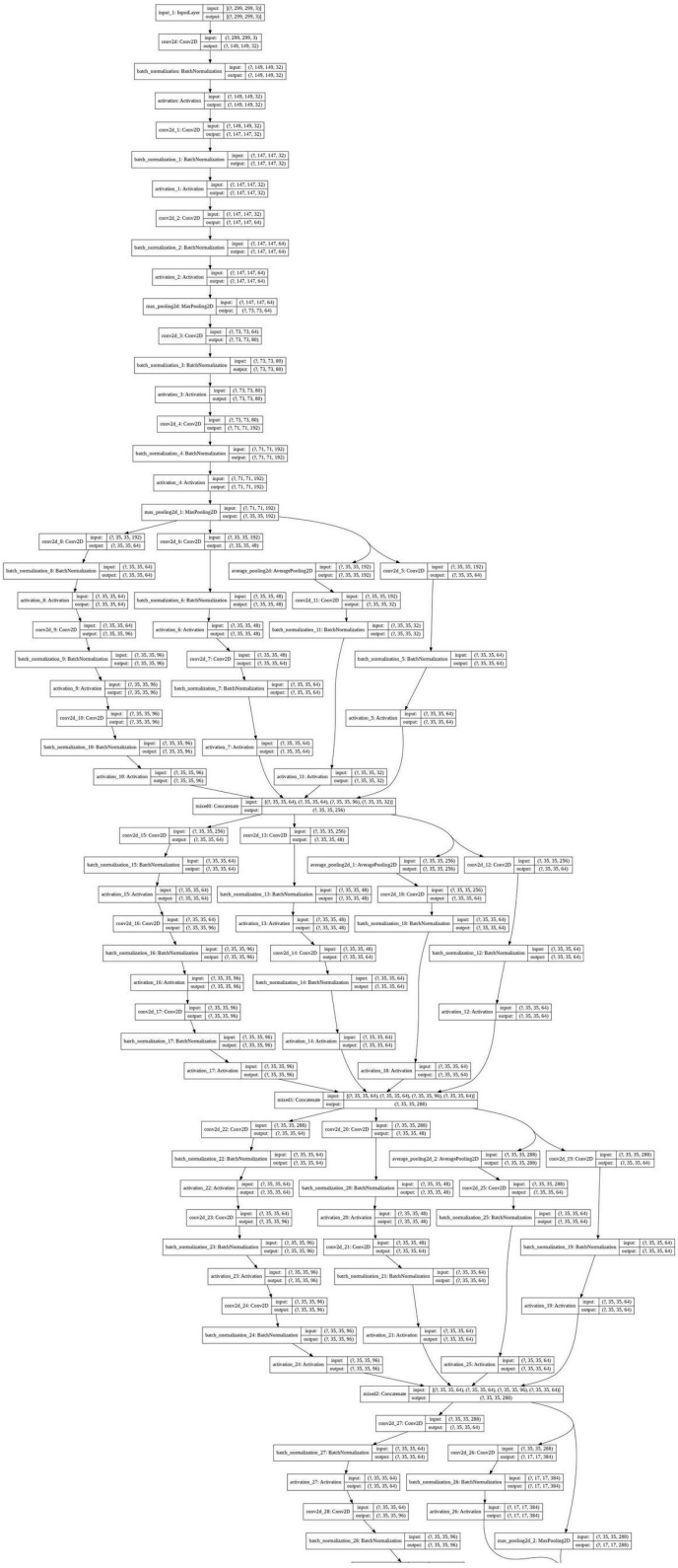
was categorical cross entropy. We also experimented with ADAM optimizer with 0.001 learning rate, but got worse results. The model is trained with 50 epochs.

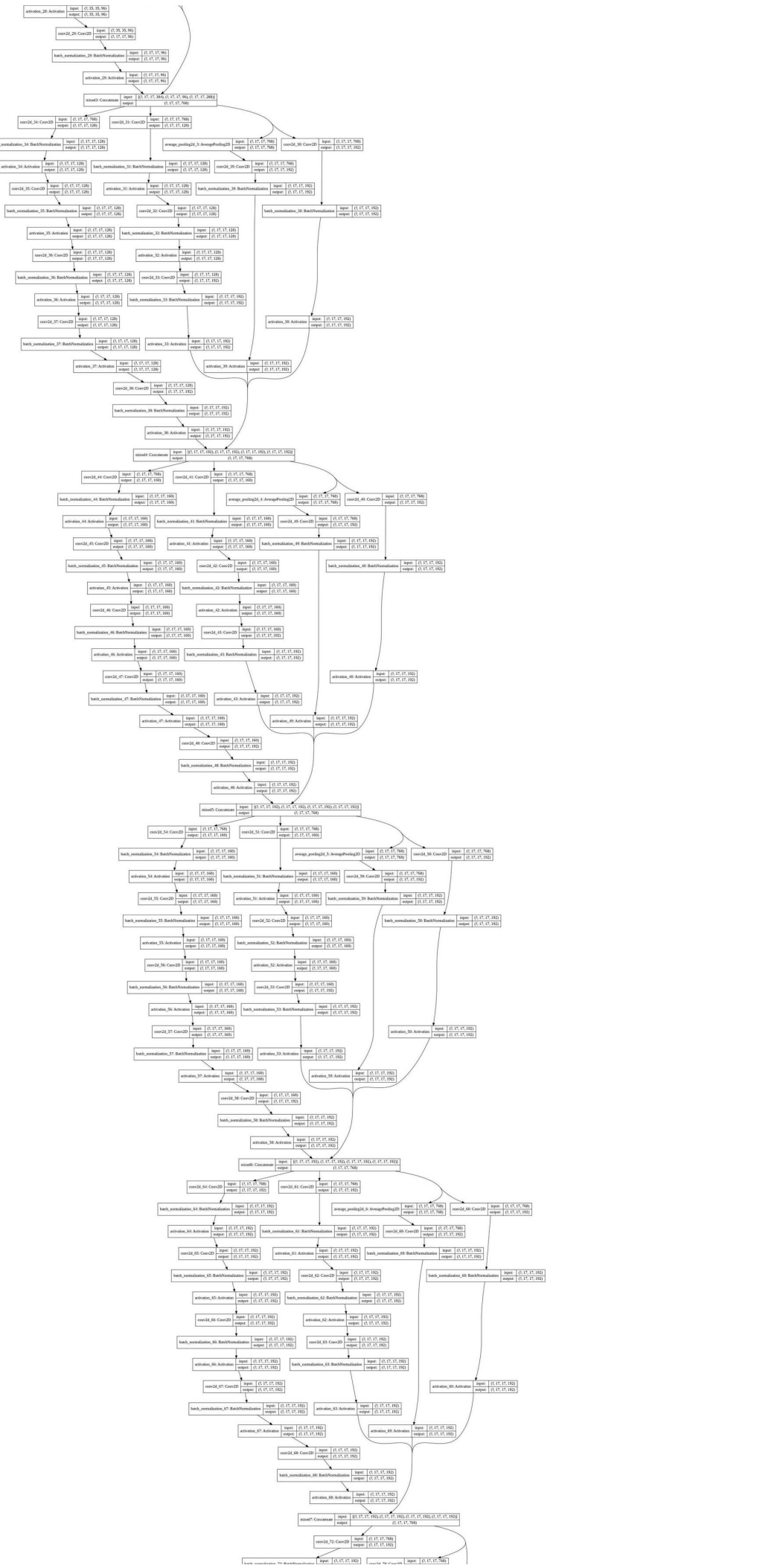
Image Captioning:

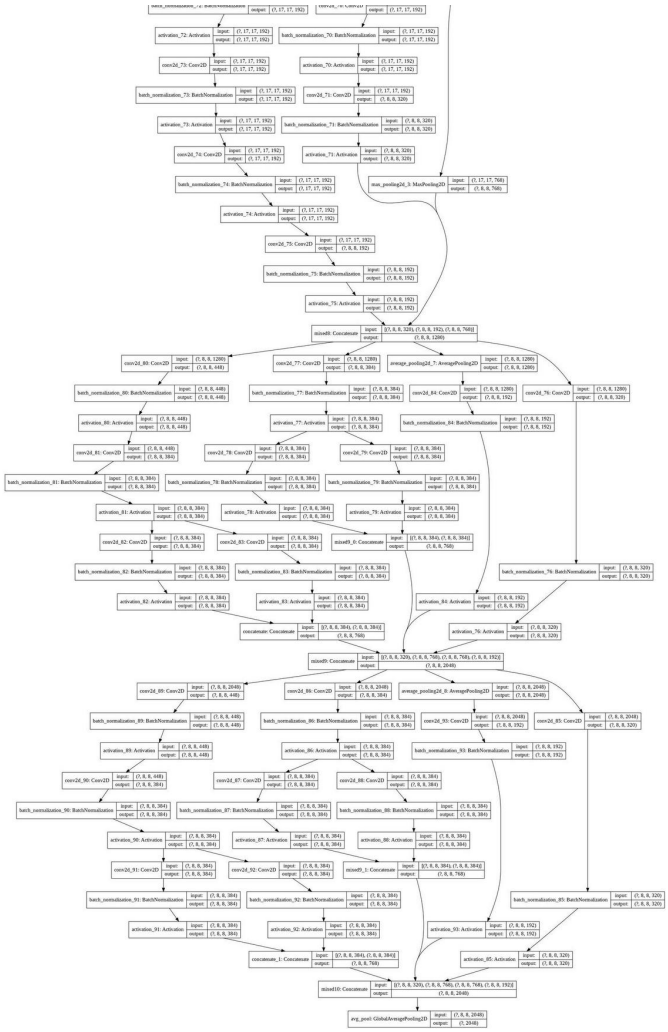
INCEPTIONV3: Feature Extractor for Images

This model is a pre-trained model available publicly in the Tensorflow library and is genreally used for feature extraction for images. There are various other models that can be used to extract the features, however the project choooses this model to use this model for the following reasons. The first reason for the choice is because the Inception model is the most lightw

A diagram of the model can be seen below:




activation_28: Activation
conv2d_29: Conv2D
batch_normalization_29: BatchNormalization
activation_29: Activation
mixed3: Concatenate
conv2d_34: Conv2D
conv2d_31: Conv2D
average_pooling2d_3: AveragePooling2D
conv2d_36: Conv2D
batch_normalization_34: BatchNormalization
activation_34: Activation
batch_normalization_31: BatchNormalization
conv2d_30: Conv2D
conv2d_35: Conv2D
activation_31: Activation
batch_normalization_39: BatchNormalization
batch_normalization_35: BatchNormalization
conv2d_32: Conv2D
batch_normalization_36: BatchNormalization
activation_30: Activation
activation_35: Activation
batch_normalization_32: BatchNormalization
conv2d_38: Conv2D
conv2d_36: Conv2D
activation_32: Activation
batch_normalization_36: BatchNormalization
conv2d_33: Conv2D
activation_36: Activation
batch_normalization_33: BatchNormalization
conv2d_37: Conv2D
batch_normalization_37: BatchNormalization
activation_33: Activation
activation_37: Activation
activation_39: Activation
conv2d_38: Conv2D
batch_normalization_38: BatchNormalization
activation_38: Activation
mixed4: Concatenate
conv2d_44: Conv2D
conv2d_41: Conv2D
batch_normalization_44: BatchNormalization
average_pooling2d_4: AveragePooling2D
conv2d_49: Conv2D
activation_44: Activation
batch_normalization_41: BatchNormalization
conv2d_39: Conv2D
conv2d_45: Conv2D
activation_41: Activation
batch_normalization_49: BatchNormalization
batch_normalization_45: BatchNormalization
conv2d_42: Conv2D
batch_normalization_40: BatchNormalization
activation_45: Activation
batch_normalization_42: BatchNormalization
conv2d_46: Conv2D
activation_42: Activation
batch_normalization_46: BatchNormalization
conv2d_43: Conv2D
activation_46: Activation
batch_normalization_43: BatchNormalization
conv2d_47: Conv2D
activation_40: Activation
batch_normalization_47: BatchNormalization
activation_43: Activation
activation_47: Activation
activation_49: Activation
conv2d_48: Conv2D
batch_normalization_48: BatchNormalization
activation_48: Activation
mixed5: Concatenate
conv2d_54: Conv2D
conv2d_51: Conv2D
batch_normalization_54: BatchNormalization
average_pooling2d_5: AveragePooling2D
conv2d_58: Conv2D
activation_54: Activation
batch_normalization_51: BatchNormalization
conv2d_59: Conv2D
conv2d_55: Conv2D
activation_51: Activation
batch_normalization_59: BatchNormalization
batch_normalization_55: BatchNormalization
conv2d_52: Conv2D
batch_normalization_50: BatchNormalization
activation_55: Activation
batch_normalization_52: BatchNormalization
conv2d_56: Conv2D
activation_52: Activation
batch_normalization_56: BatchNormalization
conv2d_53: Conv2D
activation_56: Activation
batch_normalization_53: BatchNormalization
conv2d_57: Conv2D
activation_50: Activation
batch_normalization_57: BatchNormalization
activation_53: Activation
activation_57: Activation
activation_59: Activation
conv2d_50: Conv2D
batch_normalization_58: BatchNormalization
activation_58: Activation
mixed6: Concatenate
conv2d_64: Conv2D
conv2d_61: Conv2D
batch_normalization_64: BatchNormalization
average_pooling2d_6: AveragePooling2D
conv2d_68: Conv2D
activation_64: Activation
batch_normalization_61: BatchNormalization
conv2d_69: Conv2D
conv2d_65: Conv2D
activation_61: Activation
batch_normalization_69: BatchNormalization
batch_normalization_65: BatchNormalization
conv2d_62: Conv2D
batch_normalization_60: BatchNormalization
activation_65: Activation
batch_normalization_62: BatchNormalization
conv2d_66: Conv2D
activation_62: Activation
batch_normalization_66: BatchNormalization
conv2d_63: Conv2D
activation_66: Activation
batch_normalization_63: BatchNormalization
conv2d_67: Conv2D
activation_60: Activation
batch_normalization_67: BatchNormalization
activation_63: Activation
activation_67: Activation
activation_69: Activation
conv2d_68: Conv2D
batch_normalization_68: BatchNormalization
activation_68: Activation
mixed7: Concatenate
conv2d_72: Conv2D



As seen from the diagram, the model itself is fairly complicated and would require a fair amount of explanation to explain properly. As such a summary of the use of this model and the reason why we are using this model to develop the feature extractor. More information can be found

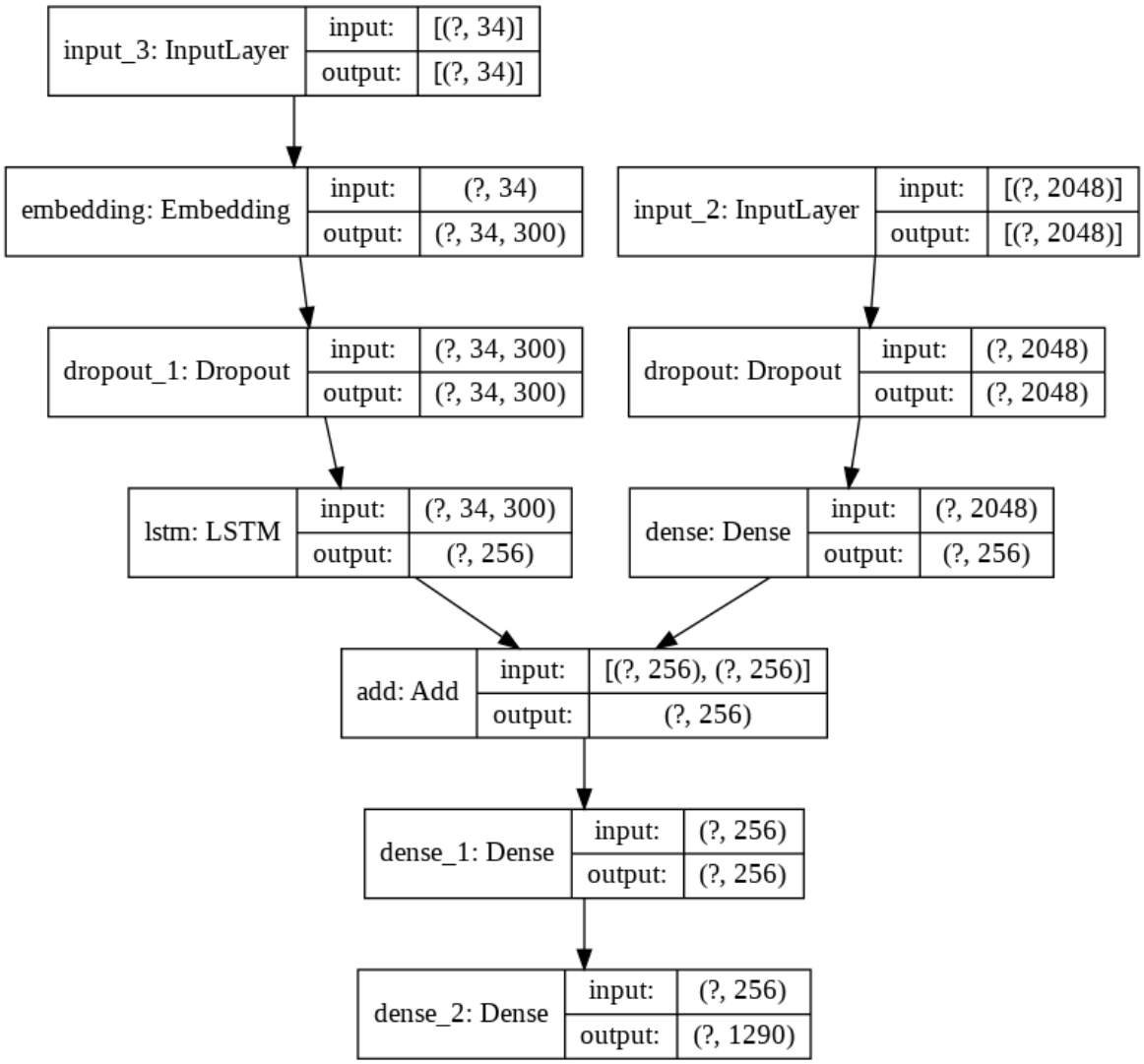
The code to use the model is as follows

```
...
# Specify the imagenet trained InceptionV3
encode_model = InceptionV3(weights='imagenet')
# Remove the last layer (output softmax layer) from the inception v3
# this is done so that the model can be combined with the LSTM for captioning.
encode_model = Model(encode_model.input, encode_model.layers[-2].output)
...
```

As seen previously, the model takes in the pre-defined input but does not output the intended pre-defined output. It outputs instead the layer before the softmax layer in order to be connected further in the combination model in the next section.

CNN-LSTM Model: Image Captioning

The following is the diagram of the CNN-LSTM model used for the image captioning training:



this is produced by this snipplet of code using the `plot_model` function:

```
...
inputs1 = Input(shape=(OUTPUT_DIM,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
inputs2 = Input(shape=(max_length,)) # max_length = 34, which comes from thr data pre
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
caption_model = Model(inputs=[inputs1, inputs2], outputs=outputs)
...
```

As seen from the diagram above the left part of the diagram processes the natural language processing part that forms the captions of the images. This part of the model is responsible for the generation of the captions based on the training data provided from the dataset. The embedded word vectors are inputted to the to be trained in the LSTM network for it to be added together with the image feature extraactor. To futher explain the network, the right side of the network is the part responsible for finding the important features of the images that have inputted to the inception network. Most of the feature extraction process has been done in the inception model, which is a transfer learning method where we use a pre trained model for our purposes, and hence the right part of the combination model is used as a layer that modifies the matrix sizes such that the images and captions can be trained

together. The model is trained for 10 epochs.

The following is the snipplet where the training is done:

```
...
model_path = os.path.join(root_captioning,f'caption-model{word_count_threshold}-300.h
if not os.path.exists(model_path):
  for i in tqdm(range(EPOCHS*2)):
      generator = data_generator(train_descriptions, encoding_train, wordtoidx, max_l
      caption_model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose

  caption_model.optimizer.lr = 1e-4
  number_pics_per_bath = 6
  steps = len(train_descriptions)//number_pics_per_bath

  for i in range(EPOCHS):
      generator = data_generator(train_descriptions, encoding_train, wordtoidx, max_l
      caption_model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose
  caption_model.save_weights(model_path)
  # print(f"\Training took: {hms_string(time()-start)}")
else:
  caption_model.load_weights(model_path)
...
```

The weights are saved to a pkl file so that there is no need to retrain the model should the project be using the same parameters for the prediction.

Evaluation

Text Emotion Analysis: Model Accuracy

After training our model, we tested it using the test data as per the train_test_split(). Validation score and accuracy was as follows:

```
Test score: 1.1070155495995875
Test accuracy: 0.7537537813186646
```

This was a marked improvement over our previous accuracies, which ranged from 0.45 to 0.65 at maximum. This significant jump was solely from the generation of random unique embeddings for each word not in the dataset, showing how much data was unable to be trained on, and the actual value of it.

Text Prediction

The models were run on utterances from dialogue01 of the dataset, meaning all utterances in that scene. This scene was used for all predictions, across all models. Hence, we can also test accuracy and F scores for these, as they were labelled as well.

```
Test score: 1.1070155495995875
Test accuracy: 0.7537537813186646
```

Model with random generation of embedding for words missing from pretrained vectors dataset:

```
[1. 1. 0. 1. 1. 1. 1. 2. 1. 1. 1. 1. 2. 3. 1. 1.]
[1 3 4 1 1 3 1 2 1 1 2 0 1 3 1 1]
['anger', 'neutral', 'surprise', 'joy', 'sadness']
              precision    recall  f1-score   support

       anger       0.00      0.00      0.00         1
     neutral       0.67      0.89      0.76         9
    surprise       0.50      0.50      0.50         2
         joy       1.00      0.33      0.50         3
     sadness       0.00      0.00      0.00         1

   micro avg       0.62      0.62      0.62        16
   macro avg       0.43      0.34      0.35        16
weighted avg       0.62      0.62      0.58        16

accuracy_score = 0.625
```

The accuracy score during the training was also relatively high, registering values of 0.85 at the 50th epoch. However, as the training and test accuracies are significantly lower, overfitting may have occurred, even with the presence of a Dense layer in the model.

What is more interesting, is when we compare the above results to the results for the model with random generation of embedding for words missing from pretrained vectors dataset, as well as stopwords cleaning:

```
[2. 1. 1. 1. 1. 1. 1. 2. 0. 1. 1. 4. 1. 3. 1. 1.]
[1 3 4 1 1 3 1 2 1 1 2 0 1 3 1 1]
['anger', 'neutral', 'surprise', 'joy', 'sadness']
              precision    recall  f1-score   support

       anger       0.00      0.00      0.00         1
     neutral       0.64      0.78      0.70         9
    surprise       0.50      0.50      0.50         2
         joy       1.00      0.33      0.50         3
     sadness       0.00      0.00      0.00         1

    accuracy                           0.56        16
   macro avg       0.43      0.32      0.34        16
weighted avg       0.61      0.56      0.55        16

accuracy_score = 0.5625
```

By comparing the results of both models, it is implied that removing stopwords actually*decreases* accuracy, even if the F1 scores remain the same. This could mean that the stopwords themselves could reveal the emotion of an utterance.
It gets even worse without the randomly generated vectors, as per:

```
[1. 1. 1. 1. 1. 1. 0. 0. 2. 1. 1. 1. 2. 5. 1. 1.]
[1 3 4 1 1 3 1 2 1 1 2 0 1 3 1 1]
['anger', 'neutral', 'surprise', 'joy', 'sadness', 'fear']
              precision    recall  f1-score   support

       anger       0.00      0.00      0.00         1
     neutral       0.55      0.67      0.60         9
    surprise       0.00      0.00      0.00         2
         joy       0.00      0.00      0.00         3
     sadness       0.00      0.00      0.00         1
        fear       0.00      0.00      0.00         0

   micro avg       0.38      0.38      0.38        16
   macro avg       0.09      0.11      0.10        16
weighted avg       0.31      0.38      0.34        16

accuracy_score = 0.375
```

Audio Emotion Analysis: Model Accuracy

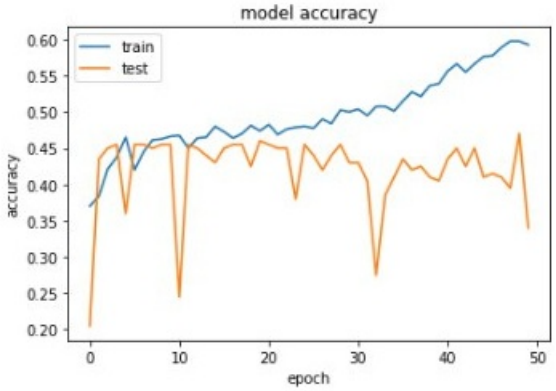
Originally, our model, which uses a learning rate of 0.0005, achieved about 0.4124 validation accuracy.

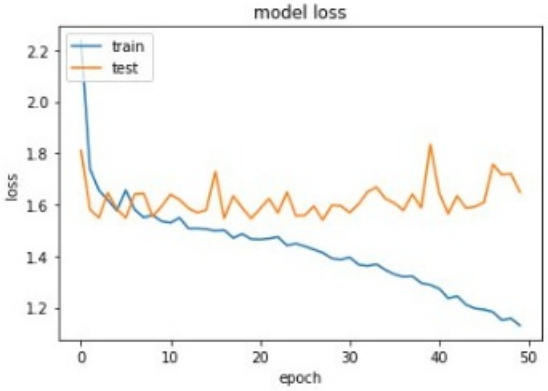
```
Epoch 00036: val_acc did not improve from 0.46847
Epoch 37/50
7990/7990 [==============================] - 83s 10ms/step - loss: 1.3208 - acc: 0.5323 - val_loss: 1.6879 - val_acc: 0.4269

Epoch 00037: val_acc did not improve from 0.46847
Epoch 38/50
7990/7990 [==============================] - 83s 10ms/step - loss: 1.3090 - acc: 0.5359 - val_loss: 1.6998 - val_acc: 0.4179

Epoch 00038: val_acc did not improve from 0.46847
Epoch 39/50
7990/7990 [==============================] - 82s 10ms/step - loss: 1.2993 - acc: 0.5397 - val_loss: 1.7271 - val_acc: 0.4244

Epoch 00039: val_acc did not improve from 0.46847
Epoch 40/50
7990/7990 [==============================] - 83s 10ms/step - loss: 1.2913 - acc: 0.5411 - val_loss: 1.7463 - val_acc: 0.4154

Epoch 00040: val_acc did not improve from 0.46847
Epoch 41/50
7990/7990 [==============================] - 82s 10ms/step - loss: 1.2763 - acc: 0.5459 - val_loss: 1.7894 - val_acc: 0.4124
```

By adding a reduceCallback function, we are able to reduce learning rate if validation accuracy does not improve for 10 epochs.

```
...
reducelr_callback = ReduceLROnPlateau(
                monitor='val_acc', factor=0.5, patience=10, min_delta=0.01,
                verbose=1
            )
...
```

```
['loss', 'val_accuracy', 'lr', 'val_loss', 'accuracy']
```





	precision	recall	f1-score	support
anger	0.35	0.09	0.14	214
neutral	0.48	0.99	0.64	933
surprise	0.00	0.00	0.00	231
joy	0.00	0.00	0.00	388
sadness	0.00	0.00	0.00	132
fear	0.00	0.00	0.00	53
disgust	0.00	0.00	0.00	47
accuracy			0.47	1998
macro avg	0.12	0.15	0.11	1998
weighted avg	0.26	0.47	0.32	1998

```
0.47297297297297297
```

Even with the modifications, the model was only able to achieve 0.47 accuracy, and a weighted average f1-score of 0.32. This could be caused by the largely skewed data set with large numbers of neutral datas, and also the laugh track which is present in some of the data set which acts causes noise in our data.

Image Captioning: Model Accuracy

Various models were tested to find the most accurate one depending on `word_count_threshold` (wct) as seen in the following:

```
wct = 15
caption-model15-300.hdf5
['loss', 'accuracy'] = [2.047430920600891, 0.46607482]

wct = 18
caption-model18-300.hdf5
['loss', 'accuracy'] = [2.028952491283417, 0.4700637]

wct = 20
caption-model20-300.hdf5
['loss', 'accuracy'] = [2.0349709510803224, 0.4765575]

wct = 23
caption-model23-300.hdf5
['loss', 'accuracy'] = [2.0612991213798524, 0.47310433]

wct = 25
caption-model25-300.hdf5
['loss', 'accuracy'] = [2.030712640285492, 0.4682231]

wct = 30
caption-model30-300.hdf5
['loss', 'accuracy'] = [2.030712640285492, 0.4682231]
```

An example of the training process which produces the above values can be seen in the following:

```
...
2000/2000 [==============================] - 242s 121ms/step - loss: 2.4482 - accurac

 95%|████████████████████████████████████████████████████████████████████████████     |

2000/2000 [==============================] - 243s 121ms/step - loss: 2.4372 - accurac

100%|████████████████████████████████████████████████████████████████████████████████|

1000/1000 [==============================] - 129s 129ms/step - loss: 2.3706 - accurac
1000/1000 [==============================] - 128s 128ms/step - loss: 2.3283 - accurac
1000/1000 [==============================] - 128s 128ms/step - loss: 2.3115 - accurac
1000/1000 [==============================] - 128s 128ms/step - loss: 2.2968 - accurac
1000/1000 [==============================] - 128s 128ms/step - loss: 2.2881 - accurac
1000/1000 [==============================] - 129s 129ms/step - loss: 2.2782 - accurac
1000/1000 [==============================] - 130s 130ms/step - loss: 2.2724 - accurac
1000/1000 [==============================] - 128s 128ms/step - loss: 2.2641 - accurac
1000/1000 [==============================] - 128s 128ms/step - loss: 2.2591 - accurac
1000/1000 [==============================] - 128s 128ms/step - loss: 2.2528 - accurac
(for 25 300)
```

The image caption generator performs less than 0.5 accuracy when it is calculated based on the loss value of the model, the general range of the accuracy can be seen to be around 0.465 - 0.479. For the following analysis the project will be using `wct = 20` as it is found to be the best among the iterations of the models as it has the highest accuracy. While it does not have the lowest loss value, this particular model is chosen because it has higher wct, meaning that there are more limited words that the model can learn such that no random niche words are trained in the model.

Image Captioning: BLEU score

As an approach to evaluating the generated caption of the Image Captioning, Bilingual Evaluation Understudy (BLEU) score is used to evaluate the accuracy of the captions with respect to the label in the dataset. In order to calculate BLEU score, the following formula can be seen in the following:

$$\mathrm{P(i)} = \frac{Matched(i)}{H(i)}$$

where i is the number of word chunking (i-gram) that is used to evaluate. H(i) is the reference where i is as mentioned previously.

Matched(i) can be calculated using the following:

$$\mathrm{Matched(i)} = \sum_{ti} min(C_h(t_i), max_j\, C_{hj}(ti))$$

For example, suppose there is a generated caption (candidate): `['this', 'is', 'a', 'test']` and the actual caption (reference) is `[['this', 'is', 'small', 'test']]`, BLEU score evaluation will calculate the accuracy for a 1-gram evaluation as the following:

$$\mathrm{BLEU\ score} = \frac{3}{4} = 0.75$$

As seen from above, the score calculates the number of appearances of words or chunks or words and compare them to the hypothesis (labelled captions) to obtain a score for the 'translation', which in this case refers to translating features of images to captions. As this is a common method to evaluate an image captioning model's accuracy, this is implemented to see how the model fares in coomparison to its labelled set. The bleu score analysis cannot be done on an unlabelled dataset as it requires a hypothesis caption to see the score for itself.

The following is the part of the code that does this analysis:

```
    candidate = generateCaption(image)
    hashtags = generateHashCaption(image)
    print(f'the caption is : {candidate}, with hashtags: {hashtags}')
    for description in ref:
        print("_____________________________________")
        print(f'Cumulative 1-gram: {sentence_bleu(description, candidate, weights=(1,
        print(f'Cumulative 2-gram: {sentence_bleu(description, candidate, weights=(0.
        print(f'Cumulative 3-gram: {sentence_bleu(description, candidate, weights=(0.
        print(f'Cumulative 4-gram: {sentence_bleu(description, candidate, weights=(0.
        print("_____________________________________")
```

A sample output for this can be seen in the following:

```
the caption is : dog runs through the snow, with hashtags: ['#dog', '#runs', '#snow']
_____________________________________
Cumulative 1-gram: 0.48, the test description is: A brown dog carries an object in it
Cumulative 2-gram: 1.0334580069279309e-154, the test description is: A brown dog carr
Cumulative 3-gram: 6.983239436165665e-204, the test description is: A brown dog carri
Cumulative 4-gram: 1.5164169595070107e-231, the test description is: A brown dog carr
```

The score is for the image:

Image Captioning: Human judgement

Although a BLEU-score gives a numerical probability to the prediction, it can be seen that the accuracy itself is quite low. However, when it is seen what the generated caption is compared to, it can be realised that the general idea of the picture is already in the captions. Using the previous example as an example to judge the model, which can be seen in the following:

```
the caption is : dog running through the snow, with hashtags: ['#dog', '#running', '#
_____
Cumulative 1-gram: 0.4642857142857143, the test description is: A brown dog carries a
Cumulative 2-gram: 1.016400514430947e-154, the test description is: A brown dog carri
Cumulative 3-gram: 6.906952680226714e-204, the test description is: A brown dog carri
Cumulative 4-gram: 1.503850462789979e-231, the test description is: A brown dog carri
```

Human judgements are able to realise that the general idea of describing ‘a dog in the snow’ is captured in the generated caption. Furthermore, the labelled caption discusses how the image is the image of ‘A brown dog carries an object in its mouth on a snowy hillside .’ While it is not accurate word for word the idea of having a dog in the snow is captured in the generated caption. As such, for this case the model has predicted te caption well enough. However, there are other examples in the test set which are not predicted well enough, which can be seen from the following:

```
the caption is : man in black jacket and cap is standing in front of large crowd of,
_____________________________________
Cumulative 1-gram: 0.2121212121212121, the test description is: A man in a lime green
Cumulative 2-gram: 6.870119095952878e-155, the test description is: A man in a lime g
Cumulative 3-gram: 5.333595525504042e-204, the test description is: A man in a lime g
Cumulative 4-gram: 1.2363867681772576e-231, the test description is: A man in a lime
```

The image of the police officer in lime is not generally predicted in the generated caption as it predicts ‘man in black jacket and cap is standing in front of large crowd of’, this as such show that the model not predicting the caption well enough as it supposedly misunderstood the picture itself. As such although teh bleu score gives a non-zero score, the model itself does not predict the caption well enough. This has created the need to create a different method of judgement as the bleu score method does not justify the accuracy of the model

A proposed naive method of judgement using the visual senses is as follows:
Suppose there is an image of a dog carrying an object through the snow, as seen in the previous example. The proposed method is a checklish method based on the following criterias:

- Object Described
- Scene Described
- Action Described

Each of the criteria carries a weight of integer 1 and depending on whether the human judges it to be true or not each criteria is given a score of 1 (when true) or 0 (when false). The sum of the score is then taken and divided by 3 to get a naive average score of the image. The average of all the images is then calculated to achieve the final score of the model.

The final score can be seen below:

```
Final Human Judgement Score: 0.6226
```

As it is a time consuming method of evaluation the score itself is only able to be gotten from the 500 test labelled data.

Image Captioning: Testing on unlabelled set of images

As there is no reference for the captions of an unlabelled images, the only reliable method of evaluation for this project is the use of human judgement and as such can only rely on how accurate the human judgement is.
Based on the aforementioned criteria, the following image results can be seen in the following:

```
Caption: man in black shirt and tie is smiling at the camera while another man is hol
#: ['#man', '#black', '#shirt', '#tie', '#smiling', '#camera', '#another', '#man', '#
```

As seen from the captions, the caption generator does detect man (Chandler) in tie and from certain angle he does seem to be looking at the camera. The caption generator also detects drinks in the background meaning it knows the shape of the mug in the background. However, the general description itself is not entirely accurate as the man (Chandler) is not seen to be smiling rather the woman (Monica) is. No man is also seen holding a drink but it does detect drink in the picture. Despite this, it can be seen that generally, it does capture the general idea of the image and hence making it acceptable to be used for the project.

Results and Discussion

Image Captioning

As of the creation fo this report, the group has not found a suitable evaluation method that can accurately capture the essence of the image captioning and as such is currently relying on human judgement to evaluate the accuracy of the model. Although Bleu-score does give insights on the accuracy of the model, it does not reflect on the actual perceived accuracy of the generated caption. Using the human judgement method, if done properly and objectively, can reflect on the perceived accuracy better.

It is also to be noted that changing the `word_count_threshold` also affects the accuracy of the model and the generated caption. This is because there are a more limited number of choices of words that can be used for training. Although the sheer size of data makes it such that it does not seem to be significant, the model accuracy and loss value presents otherwise. Changing the parameter changes the input sizes of the model and as such changes the weights of the training model, which results in a change of predictions in the end. As mentioned, the project is mainly done using `word_count_threshold = 20` as it seemed to be the most accurate out of the other values. As such there may be a possibility of creating a better model, should the `word_count_threshold` parameter be changed during preprocessing.

A limitation in the project may come from the limited variations of the captions in the dataset. Although one image is already associated with 5 captions, the captions themselves are quite similar in essence with differences in small details, as seen in the labelled examples above. This however affects the

BLEU score in a significant way as BLEU score evaluation method searches for an exact match for the words to be considered a match. As such the accuracy itself suffers and hence the less than 0.5 results in the evaluation. As such there is a probability of increasing the accuracy should the variation of captions be increased in the dataset.

Emotion

Audio Emotion Analysis

When considering a pixel of a certain color in an image, it can most often be assumed to belong to a single object. Discrete sound events do not separate into layers on a spectrogram. Instead, aeere converted from contnuous to discrete, and alot of information is lost. That means that a particular observed frequency in a spectrogram cannot be assumed to belong to a single sound as the magnitude of that frequency could have been produced by any number of accumulated sounds or even by the complex interactions between sound waves such as phase cancellation. This makes it difficult to separate simultaneous sounds in spectrogram representations, and becomes an even bigger problem with the amount of noise in the dataset.

Also, when using CNN for images , we can assume x and y axis to be the same. But for a spectogram, they are freequency and time. By moving the frequencies of a male voice upwards, it could change its meaning from man to perhaps a woman. The spatial invariance that 2D CNNs provide might not perform as well for this form of data.

Text Emotion Analysis

The model for text emotion analysis currently does not consider the speaker, or the context between utterances. For example, certain words may be used in different tones or emotional states by different speakers As a result, while the accuracy may be relatively high, it can be improved.

Combined Results

The combined result can be seen in the following:

```
Caption: ['#man', '#black', '#shirt', '#tie', '#smiling', '#camera', '#another', '#ma
Audio Analysis Sentiment: Neutral
Subtitle Analysis Sentiment: Neutral
```

Future Works and Improvements

Audio Emotion Analysis: Issue with spectogram

We should aim for a better audio feature extraction method to improve the classification performance, as using mel-spectrum does allow a more visual way to see diffeences, it may not be the optimal method of extracting the data. Also, emotions such as disgust, fear, and sadness also performed poorly, due to imbalance data.

We also aim to merge the subtitle Emotion Analysis and Audio Emotion Analysis, to see how much of an improvement we can gain by combining the unimodal models.

Furthermore, our model does consider context even though the dialogue has been sorted into ordered utterances. By making use of context, this could help improve the accuracy for both Audio and Text Analysis.

Image Captioning: Feature Extraction Model Improvement

Using a readily available pre-trained model may be the most straightforward, safest, and prove to be the most effective in extracting the features of the images, however, there is a need to improve on the feature extraction using better models. A possibility to improve the model is possibly creating a new model that can better extract features of the images.

To improve on the model, a visual attention method to see which pixels the extractor are focusing on when generating the caption may be done. This may result in modifications to model to becoming more accurate.

Image Captioning: Similarity Index using NLTK

Based on the observation made from teh results, it can be seen that the model itself do generate a somewhat similar captions to the test label. However this is not picked up by the BLEU score approach as it calculates only based on the number of exact words that appears in the test captions. An example can be seen in the following:

```
the caption is : two dogs are running through the grass, with hashtags: ['#two', '#dc

_____________________________________
Cumulative 1-gram: 0.3157894736842105, the test description is: a brown dog about to
Cumulative 2-gram: 8.382451328140937e-155, the test description is: a brown dog about
Cumulative 3-gram: 6.082029874590866e-204, the test description is: a brown dog about
Cumulative 4-gram: 1.3657076305145602e-231, the test description is: a brown dog abou
```

The example can be seen to give a relatively low accuracy as the words ‘snow’ does not appear in the test description, rather the word ‘snowy’ does. Humans would understand that ‘snow’ and ‘snowy’ are closely related words that equally describes the image (as seen in the example) in this context. As such, the evaluation method itself is not the most accurate way to evaluate the model itself. One proposed method is to use nltk to measure the word distance for each of the generated caption word. Should the word be close enough, the generated caption will be replaced with a similar word in teh test caption only for evaluation purposes. Through this improvement, the accuracy of the BLEU score may increase and give a more accurate evaluation of the model.

Future Works: Video Captioning

Since this project naively attempts to create a video analyser by combining image captioning, textual emotion analysis, and audio emotion analysis, an opportunity to combine all the models to create a

video captioner that describes the context of a scene, which in this context mainly relies on TV shows, in terms of the sentiment as well as the emotion in the scene, shows itself. A combination of the image captioning model and emotion analysis may be done using a similar method to combining the LSTM and Inception model, which is by removing the output layer and connecting it to an add layer as seen in the previous section or possibly creating a new model for this specific task itself.

Future Works: Text Emotional Analysis

In the future, a model considering the personalities and the context of the entire scene would provide for a more accurate prediction of the emotions for each utterance.
Moreover, if aiming to apply onto another series, the model would have to detect differences between the series, and change its weights accordingly. Further exploration into Generative Adversarial Networks (GAN) would be required in order to use one model for the emotional analysis of transcripts from many other series, as was the original intention of the project.

References

[1] Soujanya Poria, Devamanyu Hazarika, Navonil Majumder,Gautam Naik, Erik Cambria, Rada Mihalcea. “MELD: A Multimodal Multi-Party Dataset for Emotion Recognition in Conversations”, https://arxiv.org/pdf/1810.02508.pdf

[2]Lin Feng “Music Genre Classification with Paralleling Recurrent Convolutional Neural Network”, https://arxiv.org/pdf/1712.08370.pdf

[3]Keunwoo Choi, Gyorgy Fazekas, Mark Sandler. “convolutional recurrent neural networks for music classification” https://arxiv.org/pdf/1609.04243.pdf

[4]Daniel Rothmann, “What’s wrong with CNNs and spectrograms for audio processing?”, https://towardsdatascience.com/whats-wrong-with-spectrograms-and-cnns-for-audio-processing-311377d7ccd

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

TensorFlow. (n.d.). Image captioning with visual attention | TensorFlow Core. [online] Available at: https://www.tensorflow.org/tutorials/text/image_captioning.

Medium. (n.d.). Image Captioning with Keras—“Teaching Computers to describe pictures”. [online] Available at: https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8.

Karpathy, A. and Fei-Fei, L. (n.d.). Automated Image Captioning with ConvNets and Recurrent Nets. [online] Cs.stanford.edu. Available at: https://cs.stanford.edu/people/karpathy/sfmltalk.pdf.

Medium. (2018). Multi-Modal Methods: Image Captioning (From Translation to Attention). [online] Available at: https://medium.com/mlreview/multi-modal-methods-image-captioning-from-translation-to-attention-895b6444256e.

Papineni, K., Roukos, S., Ward, T. and Zhu, W. (2019). BLEU: a MethodforAutomaticEvaluationofMachineTranslation. [online] Aclweb.org. Available at: https://www.aclweb.org/anthology/P02-1040.pdf .

Medium. (n.d.). Evaluating Text Output in NLP: BLEU at your own risk. [online] Available at: https://towardsdatascience.com/evaluating-text-output-in-nlp-bleu-at-your-own-risk-e8609665a213.