Import libraries

In [1]:
```
import os
import cv2
import tensorflow as tf
import numpy as np
from typing import List
from matplotlib import pyplot as plt
import imageio
```

Build data loading functions

In [2]:
```
def load_video(path:str) -> List[float]:

    cap = cv2.VideoCapture(path)
    frames = []
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        frame = tf.image.rgb_to_grayscale(frame)
        frames.append(frame[190:236,80:220,:])
    cap.release()

    mean = tf.math.reduce_mean(frames)
    std = tf.math.reduce_std(tf.cast(frames, tf.float32))
    return tf.cast((frames - mean), tf.float32) / std
```

In [3]:
```
vocab = [x for x in "abcdefghijklmnopqrstuvwxyz'?!123456789 "]

char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

print(
    f"The vocabulary is: {char_to_num.get_vocabulary()} "
    f"(size ={char_to_num.vocabulary_size()})"
)
```

```
The vocabulary is: ['', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', "'", '?', '!',
'1', '2', '3', '4', '5', '6', '7', '8', '9', ' '] (size =40)
```

In [4]:
```
def load_alignments(path:str) -> List[str]:
    with open(path, 'r') as f:
        lines = f.readlines()
    tokens = []
    for line in lines:
        line = line.split()
        if line[2] != 'sil':
            tokens = [*tokens,' ',line[2]]
    return char_to_num(tf.reshape(tf.strings.unicode_split(tokens, input_encoding='
```

In [5]:
```
def load_data(path: str):
    path = bytes.decode(path.numpy())
    #file_name = path.split('/')[-1].split('.')[0]
    # File name splitting for windows
    file_name = path.split('\\')[-1].split('.')[0]
    video_path = os.path.join('data','s1',f'{file_name}.mpg')
    alignment_path = os.path.join('data','alignments','s1',f'{file_name}.align')
    frames = load_video(video_path)
    alignments = load_alignments(alignment_path)

    return frames, alignments
```

In [6]:
```
def mappable_function(path:str) ->List[str]:
    result = tf.py_function(load_data, [path], (tf.float32, tf.int64))
    return result
```

Prepare testing dataset

In [7]:
```
data = tf.data.Dataset.list_files('./data/s1/*.mpg')
data = data.shuffle(500, reshuffle_each_iteration=False)
data = data.map(mappable_function)
data = data.padded_batch(2, padded_shapes=([75,None,None,None],[40]))
data = data.prefetch(tf.data.AUTOTUNE)
test = data.skip(450)
```

Load the model

In [8]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, Ma
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
```

In [9]:
```
model = Sequential()
model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

model.add(Conv3D(256, 3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

model.add(Conv3D(75, 3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

model.add(TimeDistributed(Flatten()))

model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences
model.add(Dropout(.5))

model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences
model.add(Dropout(.5))
```

```
model.add(Dense(char_to_num.vocabulary_size()+1, kernel_initializer='he_normal', ac
```

In [10]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv3d (Conv3D)             (None, 75, 46, 140, 128)  3584

 activation (Activation)     (None, 75, 46, 140, 128)  0

 max_pooling3d (MaxPooling3D  (None, 75, 23, 70, 128)  0
 )

 conv3d_1 (Conv3D)           (None, 75, 23, 70, 256)   884992

 activation_1 (Activation)   (None, 75, 23, 70, 256)   0

 max_pooling3d_1 (MaxPooling  (None, 75, 11, 35, 256)  0
 3D)

 conv3d_2 (Conv3D)           (None, 75, 11, 35, 75)    518475

 activation_2 (Activation)   (None, 75, 11, 35, 75)    0

 max_pooling3d_2 (MaxPooling  (None, 75, 5, 17, 75)    0
 3D)

 time_distributed (TimeDistr  (None, 75, 6375)         0
 ibuted)

 bidirectional (Bidirectiona  (None, 75, 256)          6660096
 l)

 dropout (Dropout)           (None, 75, 256)           0

 bidirectional_1 (Bidirectio  (None, 75, 256)          394240
 nal)

 dropout_1 (Dropout)         (None, 75, 256)           0

 dense (Dense)               (None, 75, 41)            10537

=================================================================
Total params: 8,471,924
Trainable params: 8,471,924
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
model.load_weights('models/checkpoint')
```

Out[11]:
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x13e87472050>
```

Build evaluation functions

In [12]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [13]:
```
def evaluate_performance(actual_texts, predicted_texts):
    # Flatten the lists of strings to lists of characters for character-level metri
    actual_chars = [char for text in actual_texts for char in list(text)]
    predicted_chars = [char for text in predicted_texts for char in list(text)]

    # Calculate accuracy
    accuracy = accuracy_score(actual_chars, predicted_chars)

    # Calculate precision, recall, and F1 score
    precision = precision_score(actual_chars, predicted_chars, average='weighted')
    recall = recall_score(actual_chars, predicted_chars, average='weighted')
    f1 = f1_score(actual_chars, predicted_chars, average='weighted')

    return accuracy, precision, recall, f1
```

In [14]:
```
def add_padding(actual_texts, predicted_texts):
    for i in range(len(actual_texts)):
        len_diff = len(actual_texts[i]) - len(predicted_texts[i])
        if len_diff < 0:
            actual_texts[i] += " "*abs(len_diff)
        elif len_diff > 0:
            predicted_texts[i] += " "*abs(len_diff)
```

Run evaluation

In [15]:
```
test_data = test.as_numpy_iterator()
```

In [16]:
```
actual_texts = list()
predicted_texts = list()
```

In [17]:
```
for i in range(len(test)):
    print("iteration:", i+1)

    # move to next set
    sample = test_data.next()

    # get real text
    yhat = model.predict(sample[0])
    text_real = [tf.strings.reduce_join([num_to_char(word) for word in sentence]) f
    text_real = [text_real[0].numpy().decode(), text_real[1].numpy().decode()]
    actual_texts.extend(text_real)

    # get predicted text
    decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75,75], greedy=True)[
    text_pred = [tf.strings.reduce_join([num_to_char(word) for word in sentence]) f
    text_pred = [text_pred[0].numpy().decode(), text_pred[1].numpy().decode()]
    predicted_texts.extend(text_pred)
```

```
iteration: 1
1/1 [==============================] - 4s 4s/step
iteration: 2
1/1 [==============================] - 2s 2s/step
iteration: 3
1/1 [==============================] - 2s 2s/step
iteration: 4
1/1 [==============================] - 2s 2s/step
iteration: 5
1/1 [==============================] - 2s 2s/step
iteration: 6
1/1 [==============================] - 2s 2s/step
iteration: 7
1/1 [==============================] - 2s 2s/step
iteration: 8
1/1 [==============================] - 2s 2s/step
iteration: 9
1/1 [==============================] - 2s 2s/step
iteration: 10
1/1 [==============================] - 2s 2s/step
iteration: 11
1/1 [==============================] - 2s 2s/step
iteration: 12
1/1 [==============================] - 2s 2s/step
iteration: 13
1/1 [==============================] - 2s 2s/step
iteration: 14
1/1 [==============================] - 2s 2s/step
iteration: 15
1/1 [==============================] - 2s 2s/step
iteration: 16
1/1 [==============================] - 2s 2s/step
iteration: 17
1/1 [==============================] - 2s 2s/step
iteration: 18
1/1 [==============================] - 2s 2s/step
iteration: 19
1/1 [==============================] - 2s 2s/step
iteration: 20
1/1 [==============================] - 2s 2s/step
iteration: 21
1/1 [==============================] - 2s 2s/step
iteration: 22
1/1 [==============================] - 2s 2s/step
iteration: 23
1/1 [==============================] - 2s 2s/step
iteration: 24
1/1 [==============================] - 2s 2s/step
iteration: 25
1/1 [==============================] - 2s 2s/step
iteration: 26
1/1 [==============================] - 2s 2s/step
iteration: 27
1/1 [==============================] - 2s 2s/step
iteration: 28
1/1 [==============================] - 2s 2s/step
```

```
iteration: 29
1/1 [==============================] - 2s 2s/step
iteration: 30
1/1 [==============================] - 2s 2s/step
iteration: 31
1/1 [==============================] - 2s 2s/step
iteration: 32
1/1 [==============================] - 2s 2s/step
iteration: 33
1/1 [==============================] - 2s 2s/step
iteration: 34
1/1 [==============================] - 2s 2s/step
iteration: 35
1/1 [==============================] - 2s 2s/step
iteration: 36
1/1 [==============================] - 2s 2s/step
iteration: 37
1/1 [==============================] - 2s 2s/step
iteration: 38
1/1 [==============================] - 2s 2s/step
iteration: 39
1/1 [==============================] - 2s 2s/step
iteration: 40
1/1 [==============================] - 2s 2s/step
iteration: 41
1/1 [==============================] - 2s 2s/step
iteration: 42
1/1 [==============================] - 2s 2s/step
iteration: 43
1/1 [==============================] - 2s 2s/step
iteration: 44
1/1 [==============================] - 2s 2s/step
iteration: 45
1/1 [==============================] - 2s 2s/step
iteration: 46
1/1 [==============================] - 2s 2s/step
iteration: 47
1/1 [==============================] - 2s 2s/step
iteration: 48
1/1 [==============================] - 2s 2s/step
iteration: 49
1/1 [==============================] - 2s 2s/step
iteration: 50
1/1 [==============================] - 2s 2s/step
```

In [18]:
```
# add whitespace padding for unequal real-predicted pairs
add_padding(actual_texts, predicted_texts)
```

In [19]:
```
accuracy, precision, recall, f1 = evaluate_performance(actual_texts, predicted_text

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Accuracy: 0.9791415964701163
Precision: 0.9792040812714483
Recall: 0.9791415964701163
F1 Score: 0.9791374960470012
```