Aim

The aim of this project is to develop and train a model for lip reading using the LipNet architecture. The model should be able to recognize spoken words from video frames of a person's lips, converting visual speech patterns into textual output.

Objective

1. **Preprocess Video Data:** Extract and normalize frames from video files containing lip movements.
2. **Feature Extraction:** Extract and prepare ground truth alignments for training.
3. **Model Architecture:** Design and implement a 3D Convolutional Neural Network (CNN) combined with Bidirectional LSTMs to process the video frames and predict sequences of spoken characters.
4. **Training and Evaluation:** Train the model using the CTC (Connectionist Temporal Classification) loss function and evaluate its performance on test data.
5. **Prediction and Interpretation:** Test the model on new video data and compare predicted text with ground truth.

Summary

The project utilizes the LipNet model to transcribe lip movements in video into text. It involves downloading and processing video and alignment data, designing a deep learning model combining 3D CNNs and LSTMs, and training the model using CTC loss. The trained model is then evaluated and tested on new data to assess its performance in lip reading.

Tools and Libraries Used

- **TensorFlow**: For building and training the neural network.
- **OpenCV**: For video processing and frame extraction.
- **NumPy**: For numerical operations.
- **Matplotlib**: For visualization of data and results.
- **Imageio**: For saving GIFs from video frames.
- **Gdown**: For downloading data from Google Drive.
- **Python**: For scripting and implementing the project.

Procedure

1. **Data Preparation:**
    - **Download Data:** Use gdown to download video and alignment files from Google Drive.
    - **Load and Normalize Videos:** Extract frames from videos, convert them to grayscale, and normalize them.

- **Load Alignments:** Read and preprocess alignment files into character sequences.

2. **Data Pipeline:**
    - Create a TensorFlow Dataset to load video frames and alignments, shuffle, batch, and prefetch data for training.
3. **Model Architecture:**
    - **3D Convolutional Layers:** Extract spatiotemporal features from video frames.
    - **Bidirectional LSTMs:** Capture temporal dependencies and sequence patterns.
    - **Dense Layer:** Output character probabilities.
4. **Training:**
    - **Compile Model:** Use Adam optimizer and CTC loss function.
    - **Callbacks:** Implement learning rate scheduler and model checkpointing.
    - **Fit Model:** Train the model with training data and validate using test data.
5. **Evaluation and Prediction:**
    - **Load Model Weights:** Restore the trained model from checkpoints.
    - **Predict and Decode:** Predict text from video frames and decode it using CTC decoding.

Highlights

- **Data Normalization:** Frames are normalized to ensure consistent input for the model.
- **Custom Data Pipeline:** Efficient data loading and preprocessing using TensorFlow's Dataset API.
- **Model Architecture:** Combination of 3D CNN and Bidirectional LSTM allows the model to effectively capture both spatial and temporal features.
- **CTC Loss Function:** Handles variable-length sequences and aligns predicted and actual sequences without requiring pre-aligned data.
- **Callbacks:** Implementations for model checkpointing, learning rate scheduling, and monitoring predictions.

Conclusion

The project successfully implements a LipNet-based model for lip reading, capable of transcribing spoken words from video frames into text. The use of 3D convolutional layers combined with Bidirectional LSTMs proves effective in capturing the complex patterns of lip movements. The model's performance is evaluated through various test samples, and results demonstrate its ability to accurately predict spoken text. Future improvements could involve using larger datasets and more sophisticated model architectures to enhance accuracy and robustness.

CODE:

```
import os
import cv2
import tensorflow as tf
import numpy as np
from typing import List
from matplotlib import pyplot as plt
import imageio
#import os
os.environ['TF_XLA_FLAGS'] = '--tf_xla_auto_jit=2'
tf.config.list_physical_devices('GPU')
physical_devices = tf.config.list_physical_devices('GPU')
try:
  tf.config.experimental.set_memory_growth(physical_devices[0], True)
except:
  pass
print(tf.config.list_physical_devices())
import gdown
url = 'https://drive.google.com/uc?id=1YlvpDLix3S-U8fd-gqRwPcWXAXm8JwjL'
output = 'data.zip'
gdown.download(url, output, quiet=False)
gdown.extractall('data.zip')
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
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()})"
)
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='UTF-8'), (-1)))[1:]
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
test_path = '.\\data\\s1\\bbal6n.mpg'
tf.convert_to_tensor(test_path).numpy().decode('utf-8').split('\\')[-1].split('.')[0]
frames, alignments = load_data(tf.convert_to_tensor(test_path))
plt.imshow(frames[40])
alignments
tf.strings.reduce_join([bytes.decode(x) for x in num_to_char(alignments.numpy()).numpy()])
def mappable_function(path:str) ->List[str]:
    result = tf.py_function(load_data, [path], (tf.float32, tf.int64))
    return result
from matplotlib import pyplot as plt
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)
# Added for split
train = data.take(450)
test = data.skip(450)
len(test)
frames, alignments = data.as_numpy_iterator().next()
len(frames)
sample = data.as_numpy_iterator()
```

```
val = sample.next(); val[0]

imageio.mimsave('./animation.gif', val[0][0], fps=10)

# 0:videos, 0: 1st video out of the batch,  0: return the first frame in the video

plt.imshow(val[0][0][35])

tf.strings.reduce_join([num_to_char(word) for word in val[1][0]])

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D,
Activation, Reshape, SpatialDropout3D, BatchNormalization, TimeDistributed, Flatten

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler

data.as_numpy_iterator().next()[0][0].shape

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=True)))

model.add(Dropout(.5))
```

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


model.add(Dense(char_to_num.vocabulary_size()+1, kernel_initializer='he_normal',
activation='softmax'))
model.summary()
yhat = model.predict(val[0])
tf.strings.reduce_join([num_to_char(x) for x in tf.argmax(yhat[0],axis=1)])
tf.strings.reduce_join([num_to_char(tf.argmax(x)) for x in yhat[0]])
model.input_shape
model.output_shape
def scheduler(epoch, lr):
    if epoch < 30:
        return lr
    else:
        return lr * tf.math.exp(-0.1)
def CTCLoss(y_true, y_pred):
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

    loss = tf.keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
    return loss
class ProduceExample(tf.keras.callbacks.Callback):
    def __init__(self, dataset) -> None:
        self.dataset = dataset.as_numpy_iterator()
```

```
    def on_epoch_end(self, epoch, logs=None) -> None:
        data = self.dataset.next()
        yhat = self.model.predict(data[0])
        decoded = tf.keras.backend.ctc_decode(yhat, [75,75], greedy=False)[0][0].numpy()
        for x in range(len(yhat)):
            print('Original:', tf.strings.reduce_join(num_to_char(data[1][x])).numpy().decode('utf-8'))
            print('Prediction:', tf.strings.reduce_join(num_to_char(decoded[x])).numpy().decode('utf-8'))
            print('~'*100)
model.compile(optimizer=tf.keras.optimizers.legacy.Adam(learning_rate=0.0001), loss=CTCLoss)
checkpoint_callback = ModelCheckpoint(os.path.join('models','checkpoint'), monitor='loss',
save_weights_only=True)
schedule_callback = LearningRateScheduler(scheduler)
example_callback = ProduceExample(test)
model.fit(train, validation_data=test, epochs=3, callbacks=[checkpoint_callback, schedule_callback,
example_callback])
url = 'https://drive.google.com/uc?id=1vWscXs4Vt0a_1IH1-ct2TCgXAZT-N3_Y'
output = 'checkpoints.zip'
gdown.download(url, output, quiet=False)
gdown.extractall('checkpoints.zip', 'models')
model.load_weights('models/checkpoint')
test_data = test.as_numpy_iterator()
sample = test_data.next()
yhat = model.predict(sample[0])
print('~'*100, 'REAL TEXT')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in sample[1]]
decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75,75], greedy=True)[0][0].numpy()
print('~'*100, 'PREDICTIONS')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
```

```
sample = load_data(tf.convert_to_tensor('.\\data\\s1\\swwp2n.mpg'))
print('~'*100, 'REAL TEXT')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in [sample[1]]]
yhat = model.predict(tf.expand_dims(sample[0], axis=0))
decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75], greedy=True)[0][0].numpy()
print('~'*100, 'PREDICTIONS')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
```