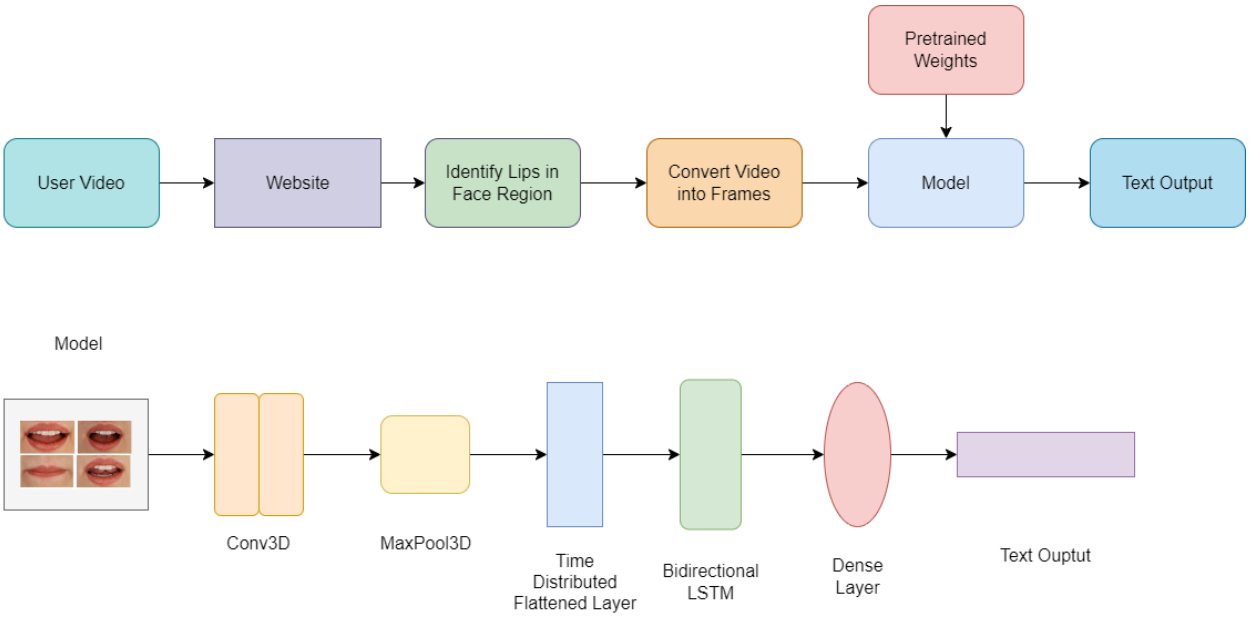
Lip Reading using Deep Learning

This project aims to develop an end-to-end machine learning solution to detect words from a video of a person speaking. The proposed solution uses Deep learning algorithms like Conv3D, MaxPooling, LSTM, and Neural Networks to predict the accurate output.

Benefits of Lip Reading using Deep Learning include:

- **Accessibility Improvement:** Lip reading using deep learning significantly improves communication for individuals with hearing impairments by relying on visual cues derived from lip movements. This technology serves as a valuable tool to enhance inclusivity and bridge communication gaps.
- **Robust Learning:** Deep learning models, trained on extensive datasets, exhibit a robust ability to recognize subtle nuances in lip movements and variations in speech. This ensures a more accurate and adaptable lip-reading system that can effectively capture the intricacies of spoken language.
- **Performance in Noisy Environments:** In contrast to traditional audio-based systems that may struggle in noisy environments, lip reading with deep learning remains effective. The visual nature of lip reading allows for reliable communication even in situations with high levels of background noise.
- **Security and Surveillance:** The application of deep learning in lip reading extends to security and surveillance, enabling automatic analysis of lip movements in video footage. This has implications for threat identification and extracting valuable information from silent videos, enhancing overall security measures.
- **Versatility:** The versatility of lip reading with deep learning is evident in its broad applicability across various scenarios. From assisting the hearing impaired to advancing security technology, this technology proves to be a multifaceted tool with diverse practical applications.

Technical Architecture of the Project:



Prior Knowledge:

To complete this project, you must require the following software's, concepts, and packages

- VS Code:
  - Refer to the link below to download VS Code.
  - Link: https://code.visualstudio.com/download
- Machine Learning Concepts
  - Deep Learning:
  - Convolutional Layer
  - LSTM
- NLP Models:
  - https://medium.com/voice-tech-podcast/an-overview-of-rnn-lstm-gru-79ed642751c6
- Web concepts:
  - Get the gist on streamlit:
  - https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/

Project Objectives:

By the end of this project, you will:

- Acquire a foundational knowledge of key concepts and techniques in Deep Learning.
- Develop a comprehensive understanding of Lip Reading principles.
- Learn efficient methods for training models.
- Acquire the skills to construct a web application utilizing the Streamlit framework.

Project Structure:

- In the process of constructing a Streamlit application, create an "app" folder to serve as the website's core.
- The "models" folder houses saved models crucial for the application's functionality.
- Within the "Training" folder, organize the code responsible for building, training, and testing the models.
- Store videos and alignments in the "Dataset" folder to facilitate seamless access to essential data for the application.

Milestone 1: Define Problem / Problem Understanding

Activity 1: The business problem

This project aims to develop an end-to-end machine learning solution to detect words from a video of a person speaking. The proposed solution uses Deep learning algorithms like CNN, LSTM, and Neural Networks to predict the accurate output.

Activity 2: Business requirements

1. **Accuracy and Reliability:** The lip reading solution must achieve a high level of accuracy in transcribing spoken words from lip movements, ensuring reliable performance across diverse accents, languages, and environmental conditions.

2. **Real-time Processing:** The system should be capable of real-time processing to enable seamless integration into various applications, such as live communication platforms, security surveillance, and assistive devices for the hearing-impaired.

3. **Scalability:** The solution should be designed to scale efficiently, accommodating an increasing volume of video data and user interactions without compromising performance. This scalability is crucial for deployment in scenarios with varying demands, such as crowded public spaces or online communication platforms.

4. **User-Friendly Interface**: The solution should offer an intuitive and user-friendly interface for easy integration into existing systems or applications. This includes clear documentation, APIs for developers, and possibly a graphical user interface for non-technical users to configure and use the lip reading capabilities effectively.

Activity 3: Literature Survey

The existing projects make use of deep learning algorithms like LSTM, and CNN. Some also use algorithms like SVM to classify the frames. We can use TensorFlow Keras to build the models. 'dlib' can be used to extract the features (lips) from the video. Many different datasets can also be found.

Activity 4: Social or Business Impact.

1. Privacy and Security:

   While audio-based speech recognition systems pose potential privacy concerns by capturing and processing audio data, lip reading systems, relying on visual information, may be perceived as less intrusive. This distinction makes lip reading an attractive alternative for applications where privacy is a top priority.

2. Use in Noisy Environments:

   In environments with high background noise, audio-based speech recognition faces challenges in maintaining accuracy. Lip reading systems offer a valuable advantage by utilizing visual cues, providing additional context that can enhance accuracy and performance in noisy scenarios.

3. Cross-Lingual Applications:

   The language-agnostic nature of lip reading is a significant strength, allowing a single model to potentially be applied across different languages without the need for language-specific training data. This versatility makes lip reading systems adaptable and effective in diverse linguistic environments.

Milestone 2: Data Collection & Preparation

Note: Use of GPU is recommended.

Activity 1: Loading the data.

Download the dataset here:
https://drive.google.com/file/d/1VNY2vZ9gtqfEb0UX80jSKto3dVRfSL0v/view?usp=sharing

Activity 2: Data Preparation

```
import os
import cv2
import tensorflow as tf
import numpy as np
from typing import List
from matplotlib import pyplot as plt
import imageio
```

We require 5 functions to process the data

- load_video()
- char_to_num
- num_to_char
- load_alignments()
- load_data()

```
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
```

```
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)
```

```
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:]
```

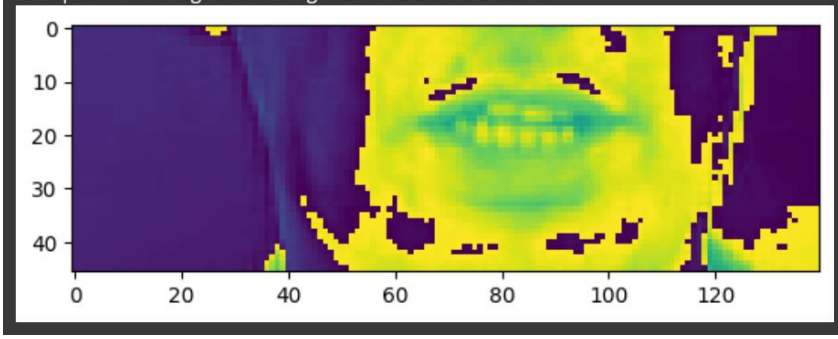
```
def load_data(path: str):
    path = bytes.decode(path.numpy())
    file_name = path.split('/')[-1].split('.')[0]
    # File name splitting for windows
    # file_name = path.split('\\')[-1].split('.')[0]
    video_path = os.path.join('data','s1',f'{file_name}.mpg')
    alignment_path = os.path.join('data','alignments','s1',f'{file_name}.align')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

    return frames, alignments
```

Milestone 3: Exploratory Data Analysis

Activity 1: Visual Analysis

```
plt.imshow(frames[40])
```

```
<matplotlib.image.AxesImage at 0x7a9f0f005de0>
```



Activity 2: Splitting data into train and test and validation sets

```
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)
```

Milestone 4: Model Building

Activity 1: Importing necessary libraries

```
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
```

Activity 2: Defining callbacks, Loss function and building the model

```
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 = 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'))
```

Activity 3: Train the models

```
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=100, callbacks=[checkpoint_callback, schedule_callback, example_callback])
```

Milestone 5: Model Deployment

Activity 1: Save the model

The model is saved. Now load the saved model with trained weights.

Activity 2: Make a Prediction

```
import gdown
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')
```

```
data = data.as_numpy_iterator()
```

```
sample = data.next()
```

```
yhat = model.predict(sample[0])
```

```
1/1 [==============================] - 14s 14s/step
```

```
print('~'*100, 'REAL TEXT')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in sample[1]]
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ REAL TEXT
[<tf.Tensor: shape=(), dtype=string, numpy=b'set blue in t three soon'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'set green by p four now'>]
```

```
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]
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PREDICTIONS
[<tf.Tensor: shape=(), dtype=string, numpy=b'set blue in t three soon'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'set green by four now'>]
```

Activity 3: Integrate with Web Framework

We will use streamlit to deploy our product.

Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps.

Activity 4.1: Create an app folder and create following .py files in it:

```
modelutil.py
streamlitapp.py
utils.py
```

Write the code for 'utils.py'

```
import tensorflow as tf
from typing import List
import cv2
import os

vocab = [x for x in "abcdefghijklmnopqrstuvwxyz'?!123456789 "]
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
# Mapping integers back to original characters
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

def load_video(path:str) -> List[float]:
    #print(path)
    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
```

```python
def load_alignments(path:str) -> List[str]:
    #print(path)
    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
```

Write code for 'modelutil.py'

```python
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D, Activation, TimeDistributed, Flatten

def load_model() -> Sequential:
    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(41, kernel_initializer='he_normal', activation='softmax'))

    model.load_weights(os.path.join('..','models','checkpoint'))

    return model
```

Write code for 'streamlitapp.py'

```python
# Import all of the dependencies
import streamlit as st
import os
import imageio

import tensorflow as tf
from utils import load_data, num_to_char
from modelutil import load_model

# Set the layout to the streamlit app as wide
st.set_page_config(layout='wide')

# Setup the sidebar
with st.sidebar:
    st.image('https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRe7q7g-97c4CPUhH4U8RHvX7BAylxkEif4hg&usqp=CAU')
    st.title('Lip Reading using Deep Learning')

st.title('Lip Reading App')
# Generating a list of options or videos
options = os.listdir(os.path.join('data', 's1'))
selected_video = st.selectbox('Choose video', options)
```

```python
# Generate two columns
col1, col2 = st.columns(2)

if options:

    # Rendering the video
    with col1:
        st.info('The video below displays the converted video in mp4 format')
        file_path = os.path.join('data','s1', selected_video)
        os.system(f'ffmpeg -i {file_path} -vcodec libx264 test_video.mp4 -y')

        # Rendering inside of the app
        video = open('test_video.mp4', 'rb')
        video_bytes = video.read()
        st.video(video_bytes)


    with col2:
        video, annotations = load_data(tf.convert_to_tensor(file_path))

        st.info('This is the output of the machine learning model as tokens')
        model = load_model()
        yhat = model.predict(tf.expand_dims(video, axis=0))
        decoder = tf.keras.backend.ctc_decode(yhat, [75], greedy=True)[0][0].numpy()
        st.text(decoder)

        # Convert prediction to text
        st.info('Words spoken in the video are:')
        converted_prediction = tf.strings.reduce_join(num_to_char(decoder)).numpy().decode('utf-8')
        st.markdown(f'## {converted_prediction}')
```

To run the website, go to the terminal and run the command:

- `streamlit run streamlitapp.py`

Make sure you have streamlit installed in your environment. If not install it using the command:

- `pip install streamlit`

This is what the website looks like: