AWS Machine Learning Blog

Create video subtitles with translation using machine learning

by Rob Dachowski | on 10 AUG 2018 | in Amazon Polly, Amazon Transcribe, Amazon Translate, Artificial Intelligence | Permalink | Comments | Share

Businesses from around the globe require fast and reliable ways to transcribe an audio or video file, and often in multiple languages. This audio and video content can range from a news broadcast, call center phone interactions, a job interview, a product demonstration, or even court proceedings. The traditional process for transcription is both expensive and lengthy, often involving the hiring of dedicated staff or services, with a high degree of manual effort. This effort is compounded when a multi-language transcript is required, often leaving customers to over-dub the original content with a new audio track.

Natural language processing (NLP) and translation are some of the hardest problems for computers to solve because of the many contextual and idiomatic elements of speech. This has historically required a native speaker of both the source and target language to perform a translation. The recently introduced neural-network based approach to machine translation has brought on unprecedented translation accuracy and fluency. While it is not perfect yet, it is now a viable solution for many more use cases than ever before, including this one.

At AWS re:Invent 2017, Andy Jassy introduced some new services in the AI/ML group targeted to solve these problems. To demonstrate some practical ways to use these services to solve the subtitling and translation problem, this blog post will focus on using Amazon Transcribe, Amazon Translate, and Amazon Polly to transcribe, translate, and overdub subtitles and alternate voice tracks with the original content.

Let's take a look at the final product, then we'll provide details of our solution.

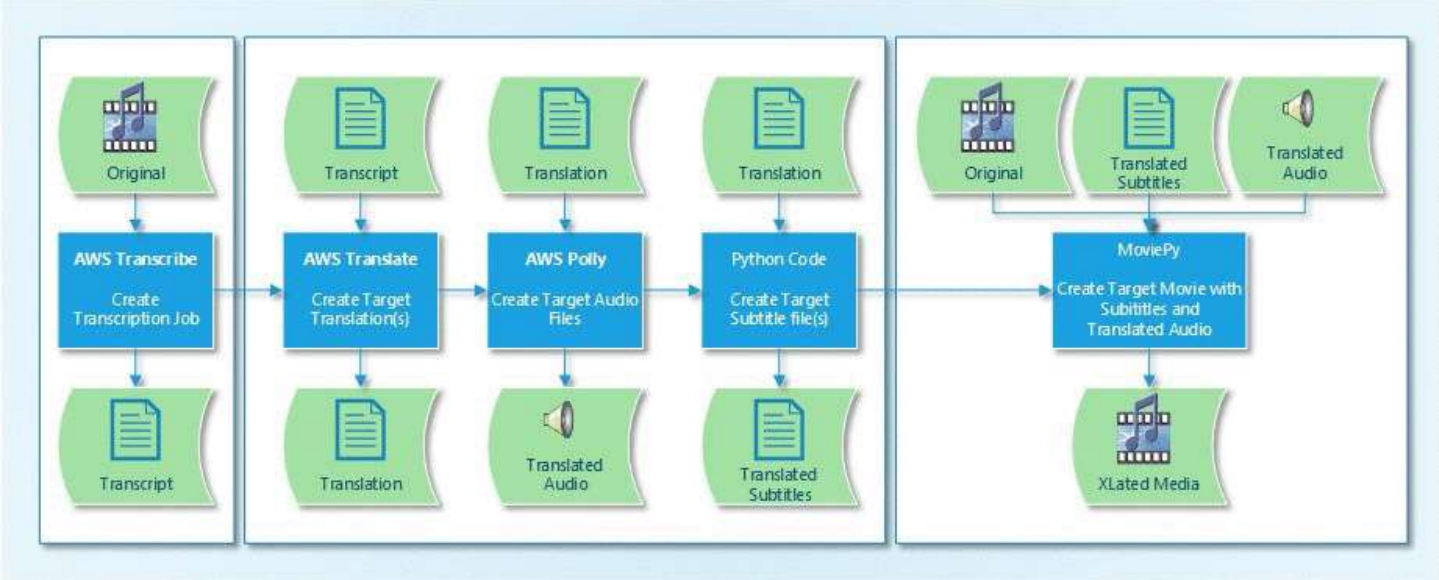
High-level approaches to solve the problem

Generally speaking, there are four high-level steps required to create subtitled, translated media:

1. Obtain a transcript of the content.

2. Translate the transcript into the target language.
3. Create the corresponding subtitles from the transcript or translation and create a subtitles file.
4. Combine all of those pieces into a finished product.

The following diagram illustrates the process in detail:



All of the source code files needed to perform these steps are included in this blog post and can be found at downloaded from here.

AWS services Overview

We'll use the following services for our use case. I'll describe the key methods and attributes needed to solve our problem.

Amazon Transcribe

Amazon Transcribe uses advanced deep learning technologies to recognize speech in audio files or video files and transcribe them into text. The output is stored in an Amazon S3 bucket that you specify or a bucket that is managed by Amazon Transcribe. The resulting transcription is a JSON file containing the full transcription of the text that includes word-level time stamping, confidence level, and punctuation. Content creators can easily use time-stamp information to sync the transcript with existing content to produce subtitles for the videos.

Amazon Transcribe can use custom vocabularies to help improve the accuracy of the transcription. You simply need to provide the customized words, jargon, and phrases to Amazon Transcribe and tell the job to use that vocabulary using the API.

In this blog post, I am using the API to launch a transcription job, set a custom vocabulary, and then use the result of the job to obtain the transcript. As you'll see, using the word-level metadata will help us to assemble the subtitles. For more information, see the Amazon Transcribe documentation.

Amazon Translate

Amazon Translate is neural machine translation service that delivers fast, high-quality, and affordable language translation. It uses deep learning models to deliver more accurate and more natural sounding translations. The Amazon Translate SDK includes TranslateText, an API action that allows us to get translations in real-time.

In this blog post, I am using the Amazon Translate API to translate the text transcription into Spanish and German. For more information, see the Amazon Translate documentation.

Amazon Polly

Amazon Polly is a cloud service that leverages AI/ML technology to synthesize text into lifelike speech. You can simplify using Amazon Polly in your applications with an API tailored to your programming language or platform, such as Java, Python, Go, and others. Amazon Polly provides the ability for you to do the following:

- Synthesize speech into an audio stream and cache and replay the speech output with no additional cost
- Choose from dozens of languages and a wide selection of natural-sounding male and female voices
- Modify your speech output using lexicons and SSML tags, such as pronunciation, volume, pitch, and speed rate

In this blog, I am using the Amazon Polly API to voice the translation in Spanish and German. I'll save the produced audio stream as an MP3 file for inclusion as the audio track in our movie. For more information, see the Amazon Polly documentation.

Preparing our development environment

To follow along with our example, you'll need to set up an environment, AWS account, download and install the tools we'll need, and, finally, install the source code.

Step 1: Establish an operating system environment

This example should run on any operating system that supports Python 2.7.15 and the ImageMagick tools. I used a Windows 10 environment running in Amazon WorkSpaces. Amazon WorkSpaces is a fully managed, secure, Desktop-as-a-Service solution powered by AWS. With Amazon WorkSpaces, you have the ability to change the virtual hardware configuration through the AWS Management Console. Early on in the process when my code was less efficient, I was running the Power configuration, but as the code stabilized, I was able to reconfigure to use the Performance configuration.

Step 2: Establish an AWS account

You'll need an Amazon Web Services account to be able to use the AWS services. The process for establishing an account can be found at https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/.

Step 3: Download and Install Python 2.7.15

Python is a programming language used by many developers for scripting batch programs, real time programs, big data, artificial intelligence, advanced analytics, and so on. There are a number of versions available. In this example, I used Python 2.7.15. Python can be downloaded from http://python.org for your target operating

system environment. The process for downloading and installing will depend on your operating system, so follow the installation instructions found on the python.org site.

After Python is successfully installed, you can enter the following command to validate that it is correctly installed:

```
python –version
```

You should see something like the following:

```
Python 2.7.15
```

Step 4: Download and Install the AWS Command Line Interface

The Amazon CLI for Windows is used so you can execute AWS commands in a Windows Command Shell. For example, to copy files to and from Amazon S3, an “aws s3” CLI command can be used. To install the AWS CLI, enter the following command

```
pip install awscli
```

You should see something like the following:

```
Installing collected packages: awscli
Successfully installed awscli-1.15.43
```

To test that the command parser is working correctly, enter the following:

```
aws –version
```

You should see something like the following:

```
aws-cli/1.15.43 Python/2.7.15 Windows/10 botocore/1.10.43
```

Step 5: Configuring the AWS CLI

Amazon Web Services takes security very seriously. No access to your account is available without the proper credentials and encrypted keys. For our programs to access AWS services, we’ll need to configure the CLI to be able to automatically sign in. It’s not good security practice to embed your AWS Identity and Access Management (IAM) access key and secret key into your program code. So, the AWS CLI provides a mechanism to associate CLI

usage with an IAM user. For an explanation of IAM, and step-by-step instructions to configure the AWS CLI, see the documentation on configuring the CLI.

Step 6: Download and install the AWS SDK for Python

The AWS SDK for Python (Boto3) is the Python SDK for access to many AWS services. It's used in the various Python code examples included with this blog post. For more information, see the documentation for Boto 3.

Generally speaking, there are two methods for connecting to an Amazon service using Boto3 that I used for this blog post. I recommend that you browse the Boto3 documentation and tutorial to get familiar with its use:

- **Resource–** This method provides a direct way to connect to and use an Amazon service. For example, to use Amazon S3, you would use the following code to establish the connection to the resource:

```
import boto3

# Let's use Amazon S3
s3 = boto3.resource('s3')
```

- **Client –** This method provides a low-level connection to an underlying service. For example, to use an Amazon Simple Queue Service (Amazon SQS) queue, you would use the following code to establish the connection to the low-level client:

```
import boto3

# Create a low-level client with the service
sqs = boto3.client('sqs')
```

To install Boto3, enter the following command:

```
pip install boto3
```

You should see something like the following:

```
Installing collected packages: boto3
Successfully installed boto3-1.7.43
```

Step 7: Install "requests" package

The requests package is an easy-to-use package designed to access URIs from within Python. For information on working with requests see the Python Quickstart. In our example, requests will be used to connect to the signed URL created by Amazon Transcribe and to download the resulting transcript JSON data.

To install requests, enterthe following command:

```
pip install requests
```

You should see something like the following:

```
Installing collected packages: requests
Successfully installed requests-2.19.1
```

Step 8: Install ImageMagick

ImageMagick is an open source package used to create, edit, compose, and convert bitmap images across a variety of image formats. MoviePy is dependent on ImageMagick to create the subtitles that are used overlay the original video.

Be aware that you need to pay special attention to the installation instructions for your platform found at the ImageMagick website. MoviePy will not work correctly if this is misconfigured.

For this blog post, I downloaded the latest version for Windows 10. To verify that ImageMagick is installed, enter the following command:

```
D:\Users\...\dev\aiml\MakeVideo>convert
```

If it is correctly installed and configured, you should see the following:

```
Version: ImageMagick 7.0.7-35 Q16 x64 2018-05-21 http://www.imagemagick.org
Copyright: Copyright (C) 1999-2018 ImageMagick Studio LLC
License: http://www.imagemagick.org/script/license.php

...
```

FFmpeg is an open source package that is used to convert audio and video into a wide variety of formats. Both ImageMagick and MoviePy are dependent on FFmpeg and will typically download it is if needed. This should be installed along with ImageMagick. Note that FFmpeg will download required additional code and updates the first time that it is run in our blog post example code.

Step 9: Install MoviePy

MoviePy is an open source Python module for video editing that can be used for basic operations (like cuts, concatenations, and title insertions), video compositing (non-linear editing), video processing, or to create advanced effects. It can read and write the most common video formats. Our example uses MoviePy for

subtitling, audio processing (adding foreign language tracks), and compositing all of the pieces together into the final videos.

To install requests, enter the following command:

```
pip install moviepy
```

MoviePy depends on a number of other libraries that should be automatically installed through this command. After all of them are installed, you should see something like the following:

```
Installing collected packages: moviepy
Running setup.py install for moviepy ... done
Successfully installed moviepy-0.2.3.5
```

Step 10: Test MoviePy

To ensure that all of the pieces are correctly installed, you might want to test MoviePy. For more information, review the short tutorial on the MoviePy site. In the tutorial, a small example program is provided that, with some minor modifications, you can use to test all of the pieces. I recommend that you take the time to read over the installation and tutorial documentation. The example code shown on the MoviePy site is:

```
from moviepy.editor import *

# Load myHolidays.mp4 and select the subclip 00:00:50 - 00:00:60
clip = VideoFileClip("myHolidays.mp4").subclip(50,60)

# Reduce the audio volume (volume x 0.8)
clip = clip.volumex(0.8)

# Generate a text clip. You can customize the font, color, etc.
txt_clip = TextClip("My Holidays 2013",fontsize=70,color='white')

# Say that you want it to appear 10s at the center of the screen
txt_clip = txt_clip.set_pos('center').set_duration(10)

# Overlay the text clip on the first video
clipvideo = CompositeVideoClip([clip, txt_clip])

# Write the result to a file (many options available !)
video.write_videofile("myHolidays_edited.webm")
```

For testing purposes, replace the “myHolidays.mp4” file with your content. Make sure to adjust the subclip parameters if your content is not at least 60 seconds long. You can also modify the message displayed in the TextClip by changing “My Holidays 2013” to your desired message. Finally, you can modify the output video file

name as required and you can also change the output type simply by specifying the extension. So, for example, instead of producing a .webm file, you can simply replace it with .mp4. See the MoviePy documentation for details.

Transcribing video using Amazon Transcribe

Our process begins by creating a transcription job with the original content. The following code is part of the main program that drives the process. Click here for the full source code.

This snippet creates the job, then loops while the status is IN_PROGRESS. After the batch job is complete, we will get the transcript response and assign it to the variable "transcript" for further processing. If we got to the "getTranscript" call, then we have successfully transcribed the content.

```
response = createTranscribeJob( args.region, args.inbucket, args.infile )

# loop until the job successfully completes
print( "\n==> Transcription Job: " + response["TranscriptionJob"]["TranscriptionJobName"]

while( response["TranscriptionJob"]["TranscriptionJobStatus"] == "IN_PROGRESS"):
    print( "."),
    time.sleep( 30 )
    response = getTranscriptionJobStatus( response["TranscriptionJob"]["TranscriptionJobN

print( "\nJob Complete")
print( "\tStart Time: " + str(response["TranscriptionJob"]["CreationTime"]) )
print( "\tEnd Time: "  + str(response["TranscriptionJob"]["CompletionTime"]) )
print( "\tTranscript URI: " + str(response["TranscriptionJob"]["Transcript"]["TranscriptF

# Now get the transcript JSON from AWS Transcribe
transcript = getTranscript( str(response["TranscriptionJob"]["Transcript"]["TranscriptFil
# print( "\n==> Transcript: \n" + transcript)
```

This snippet uses a utility function I created to place all of the Transcribe API calls in one Python file. The functions that follow leverage the Boto3 SDK for Python to invoke the Transcribe API. Detailed information about all of the API operations used can be found in the Transcribe Documentation.

```
    return response

# purpose: get and return the transcript structure given the provided uri
def getTranscript( transcriptURI ):
    # Get the resulting Transcription Job and store the JSON response in transcript
    result = requests.get( transcriptURI )

    return result.text
```

A deeper look at the response

The response back for the TranscribeURI returns a JSON structure that contains a wealth of information about the content. Looking at an excerpt of the beginning of the response that follows, we can see several key items:

- jobName
- accountId
- results – contains the full transcript, including the words and punctuation as a block of text:

```
{
    “jobName":"MY_JOB","accountId":"1234567890",
    "results":{"transcripts":
        [{"transcript":"How about language ? You know, i talked earlier about the fact
         that last year we launched both polly and relax, but there are so many other thi
         the builders want to do with language...
```

It also contains a very detailed structure for each word or punctuation element. For more information on the response structure, see the API Documentation.

```
... it."}],

"items":[
    {"start_time":"0.260","end_time":"0.480","alternatives":
        [{"confidence":"0.8977","content":"How"}],"type":"pronunciation"},
    {"start_time":"0.480","end_time":"0.750","alternatives":
        [{"confidence":"0.8431","content":"about"}],"type":"pronunciation"},
    {"start_time":"0.750","end_time":"1.440","alternatives":
        [{"confidence":"1.0000","content":"language"}],"type":"pronunciation"},
    {"alternatives":[{"content":"?"}],"type":"punctuation"},
    ...
    ]
```

Notice that each word element provides a `start_time` and an `end_time`. We'll use this in later steps to create a subtitles file that exactly aligns subtitles with the audio track. Since Amazon Transcribe uses machine learning

technologies to recognize words, it provides a confidence score for each identified word. If the content uses a lot of domain-specific terminology, then using a custom vocabulary will help to improve the accuracy of the transcription. Finally, the actual word spoken is found with the "content" tag.

The last tuple shown is for punctuation. Notice that the tuple doesn't contain timing or confidence information. This is because it is inferred, and it is assumed to be associated with the previous tuple. We'll need to address this as we process the response to prepare the subtitles file.

Translating a transcript into another language using Amazon Translate

Amazon Translate makes translating text to and from English simple. The following code snippet shows a utility function that invokes the Translate API:

```
...

def translateTranscript( transcript, sourceLangCode, targetLangCode ):
    # Get the translation in the target language.  We want to do this first so that the t
    # of what is said vs. 1 phrase at a time.  This really matters in some lanaguages

    # stringify the transcript
    ts = json.loads( transcript )

    # pull out the transcript text and put it in the txt variable
    txt = ts["results"]["transcripts"][0]["transcript"]

    # call Translate  with the text, source language code, and target language code.  The
    # translated text
    translation = translate.translate_text(Text=txt,SourceLanguageCode=sourceLangCode, Ta

    return translation
```

For more details about the Translate API, see the Translate Documentation.

Creating subtitle files

Before diving into the specifics of actually creating the subtitles, there are several design considerations we must think through.

Design consideration 1: How do we build a subtitles file?

MoviePy uses a standard subtitles file format known as "SubRip Subtitle" or SRT file. SRT files have the following format:

- Sequence number followed by a new line
- The start and end time of the subtitle to three decimal places followed by a new line

- The text to be displayed on the screen followed by two new lines

Here is an example of the output from our code:

```
1
00:00:00,260 --> 00:00:02,899
How about language? You know, i talked earlier

2
00:00:02,899 --> 00:00:05,860
about the fact that last year we launched both Polly

...
```

The Amazon Transcribe JSON structure provides the timing for each word individually. To create the subtitle timing required for the SRT file, we need to first create phrases of about 10 words. Then we write them into the SRT file format taking the start time from the first word in the phrase and the end time of the last word in the phrase to create the sequenced entry. The following function is used to create the phrases in preparation for writing them to the SRT file.

```
            # Since we are reading through each word sequentially, we'll set
            # the end_time if it is a word
            if item["type"] == "pronunciation":
                phrase["end_time"] = getTimeCode( float(item["end_time"]) )

        # in either case, append the word to the phrase...
        phrase["words"].append(item['alternatives'][0]["content"])
        x += 1

        # now add the phrase to the phrases, generate a new phrase, etc.
        if x == 10:
            #print c, phrase
            phrases.append(phrase)
            phrase = newPhrase()
            nPhrase = True
            x = 0

    return phrases
```

Note that the `getTimeCode` function is called two-thirds of the way through this snippet. Transcribe provides a start and end time based on the number of seconds that have passed since the beginning of the clip. However, the SRT format requires that time to be in an HH:MM:SS,MMM format. The following `getTimeCode` snippet does the conversion for us.

```
def getTimeCode( seconds ):
# Format and return a string that contains the converted number of seconds into SRT format

    thund = int(seconds % 1 * 1000)
     tseconds = int( seconds )
    tsecs = ((float( tseconds) / 60) % 1) * 60
    tmins = int( tseconds / 60 )
    return str( "%02d:%02d:%02d,%03d" % (00, tmins, int(tsecs), t_hund ))
```

Design consideration 2: What is the correct translation scope?

We all have seen overdubbed movies in which lips of the speaker don't line up with the audio. As we prepare subtitles, we could translate word for word, or phrase by phrase in perfect alignment with the original content. However, that would likely lead to inaccurate, incomplete, or "out of context" translations. This is because each language has differing syntax, word placement, idioms, and phrasing. Also a translation doesn't necessarily result in the same number of words in each language, hence the out-of-sync lips problem.

To illustrate the point, I used the AWS Translate console to translate the following English sentence into German and Spanish:

English

> Differing syntax, word placement, and idioms distinguish one language from another.

Spanish

> La sintaxis, la colocación de palabras y los modismos diferentes distinguen un idioma de otro.

German

> Unterschiedliche Syntax, Wortplatzierung und Idiome unterscheiden eine Sprache von einer anderen.

As you can see, neither translation has the same number of words, nor are the words in the same relative position. If we were to only translate 10 words at a time instead of a whole sentence, we would not likely arrive at a translation with the correct overall meaning.

Bottom line: We are going to have to treat subtitles and the accompanying audio for translations differently that the original transcript.

Design consideration 3: Syncing up the audio and subtitles for translations

In our example, we'll enhance the original video with translated subtitles, and we'll add an alternate spoken foreign language audio track generated by Amazon Polly. However, that leads us to a potential problem to

overcome. Unlike the original Amazon Transcribe output, we don't know the start and end time for every word because it is simply a block of translated text. Even if we translate a whole sentence at a time, that still doesn't provide timing; it only provides the translation. Moreover, unlike the original speaker, Amazon Polly will synthesize the speech at a different rate than the orginal speaker. Therefore we'll need to calculate the duration of the spoken phrases. How do we do that?

We'll let Amazon Polly and MoviePy help us. In this example, we'll take the following approach:

1. Using Amazon Translate, get the complete translation from the transcript.
2. Using our Python code, break up the block of translated text into phrases of approximately 10 words so that the text fits on the screen and can be read quickly enough by the viewer.
3. For each phrase, call Amazon Polly to get the spoken audio stream and write it to a temporary audio file.
4. Load the temporary audio file into an AudioClip object from MoviePy.
5. Return the duration of the AudioClip.

Now that we know the duration of each phrase, we can leverage the simliar logic to create the SRT file for inclusion in the final video. Note that this code differs from the transcript in that our start time value is contained in the "seconds" variable and the end time of the phrase will be calculated using the getSecondsFromTranslation function. We'll add the result to "seconds" to give us the ending time for this phrase. The code looks like this:

```
def getPhrasesFromTranslation( translation, targetLangCode ):

    # Now create phrases from the translation
    words = translation.split()

    #set up some variables for the first pass
    phrase =  newPhrase()
    phrases = []
    nPhrase = True
    x = 0
    c = 0
    seconds = 0

    print "==> Creating phrases from translation..."

    for word in words:

        # if it is a new phrase, then get the start_time of the first item
```

The code to get the duration from the phrase is below. Note that we need to translate the phrase, and request Amazon Polly to speak it to an audio stream. Then we can write the temporary audio stream to disk in order to load it as an AudioFileClip from MoviePy. After it's loaded, we can determine the exact duration and use that to calculate the timing of the subtitles relative to the translated audio track.

```
def getSecondsFromTranslation( textToTranslate, targetLangCode, audioFileName ):

    # Set up the Amazon Polly and Amazon Translate services
    client = boto3.client('polly')
    translate = boto3.client(service_name='translate', region_name="us-east-1", use_ssl=Tru

    # Use the translated text to create the synthesized speech
    response = client.synthesize_speech( OutputFormat="mp3", SampleRate="22050", Text=textT

    # Write the stream out to disk so that we can load it into an AudioClip
    writeAudioStream( response, audioFileName )

    # Load the temporary audio clip into an AudioFileClip
    audio = AudioFileClip( audioFileName)

    # return the duration
    return audio.duration
```

Creating streamed audio files from your translation using Amazon Polly

When we request Amazon Polly to synthesize text, we have several options for specifying the type, sample rate, etc., for the output. For our example, I chose an MP3 audio stream as the desired output because MoviePy can open an MP3 file as an AudioFileClip for inclusion as an alternate voice track for our final video.

After we receive a successful response from Amazon Polly, we can write the audio stream to our temporary file:

```
...

    # Use the translated text to create the synthesized speech
    response = client.synthesize_speech( OutputFormat="mp3", SampleRate="22050", Text=tra

    if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
        print( "\t==> Successfully called Polly for speech synthesis")
        writeAudioStream( response, audioFileName )
    else:
        print( "\t==> Error calling Polly for speech synthesis")

...


def writeAudioStream( response, audioFileName ):

    # Take the resulting stream and write it to an mp3 file
    if "AudioStream" in response:
```

Now we'll write the binary bytes to disk. I've created this utility method to be used in preparing the SRT file and also when creating the full audio clip for the final movie.

```
def writeAudio( output_file, stream ):

    bytes = stream.read()

    print "\t==> Writing ", len(bytes), "bytes to audio file: ", output_file
    try:
        # Open a file for writing the output as a binary stream
        with open(output_file, "wb") as file:
            file.write(bytes)

        if file.closed:
                print "\t==>", output_file, " is closed"
        else:
                print "\t==>", output_file, " is NOT closed"
    except IOError as error:
        # Could not write to file, exit gracefully
        print(error)
        sys.exit(-1)
```

Using MoviePy to create a composite video

As we discussed earlier, MoviePy is a comprehensive library for video composition and production. It leverages ImageMagick and FFmpeg (and others) for key functionality in building text titles, animation, audio, and videos. In our case, we use it for few of its features:

- Read an SRT file to create subtitles and create a subtitle clip
- Read the alternate audio track created with Amazon Polly
- Composite all of the clips together (original content, subtitles, and alternate audio track) into the finished product

To accomplish these tasks, we'll create a function called "createVideo". We will go through the key parts here, then put it all together in the next section.

```
def createVideo( originalClipName, subtitlesFileName, outputFileName, alternateAudioFileNam
    ...

    clip = VideoFileClip(originalClipName)

    ...
```

This creates a `VideoFileClip` from the video file we passed in as the original content. Next we will determine whether or not we need to replace the audio portion of the clip.

```
    if useOriginalAudio == False:
        audio = AudioFileClip(alternateAudioFileName)
        audio = audio.subclip( 0, clip.duration )
        audio.set_duration(clip.duration)
        clip = clip.set_audio( audio )
    else:
        print strftime( "\t" + "%H:%M:%S", gmtime()), "Using original audio track..."
```

If we are using the alternate audio track such as the one generated by Amazon Polly, we'll load the audio file as an AudioFileClip. The subclip and set_duration lines are ensuring that the audio clip is only as long as the video clip. If we were writing a professional package, perhaps we would want to be more sophisticated about handling the length. For our example, this works fine. Finally, we replace the audio from the video clip with the audio clip.

Next we'll create the subtitles.

```
# Create a Lambda function that will be used to generate the subtitles for each sequence
generator = lambda txt: TextClip(txt, font='Arial-Bold', fontsize=24, color='white')

# read in the subtitles files
print "\t" + strftime("%H:%M:%S", gmtime()), "Reading subtitle file: " + subtitlesFileNam
subs = SubtitlesClip(subtitlesFileName, generator)
print "\t\t==> Subtitles duration before: " + str(subs.duration)
subs = subs.subclip( 0, clip.duration - .001)
subs.set_duration( clip.duration - .001 )
print "\t\t==> Subtitles duration after: " + str(subs.duration)
print "\t" + strftime("%H:%M:%S", gmtime()), "Reading subtitle file complete: " + subtitl

print "\t" + strftime( "%H:%M:%S", gmtime()), "Creating Subtitles Track..."
annotated_clips = [annotate(clip.subclip(from_t, to_t), txt) for (from_t, to_t), txt in s

print "\t" + strftime( "%H:%M:%S", gmtime()), "Creating composited video: " + outputFileN
# Overlay the text clip on the first video clip
final = concatenate_videoclips( annotated_clips )
```

First, we need to define a Python Lambda function (not the same as an AWS Lambda Function!) that will be used to create a TextClip with the supplied font information. Then, we'll read the SRT file to create the `SubtitlesClip` call subs. Since `SubtitlesClip` is still a somewhat experimental part of MoviePy, we need to trim it to be less than the overall video clip. Otherwise, it may result in runtime errors.

After the subtitles are created, we'll create an array of subtitled clips called `annotated_clips`. Next, we concatenate all of the clips into one final clip. Finally, we'll write the subtitled video and audio out to a new video

file. Voila!

Putting it all together

Now that we've seen the parts, let's put it all together into a batch file and main driver program.

makevideo.bat

```
cls
python translatevideo.py -region us-east-1 -inbucket my-aiml-test/ -infile AWS_reInvent_201
-outfiletype mp4 -outlang es de
```

Note that in this example, I ignore the specified output bucket. This is because I wanted demonstrate that the output can be managed by Amazon Transcribe and made available via a pre-signed URL. You can optionally output the JSON transcription to a bucket you specify instead.

translatevideo.py

For this program, I used the argparse package to be able to drive the program with command line arguments.

```
print( "\nJob Complete")
print( "\tStart Time: " + str(response["TranscriptionJob"]["CreationTime"]) )
print( "\tEnd Time: "  + str(response["TranscriptionJob"]["CompletionTime"]) )
print( "\tTranscript URI: " + str(response["TranscriptionJob"]["Transcript"]["TranscriptF

# Now get the transcript JSON from AWS Transcribe
transcript = getTranscript( str(response["TranscriptionJob"]["Transcript"]["TranscriptFil
# print( "\n==> Transcript: \n" + transcript)

# Create the SRT File for the original transcript and write it out.
```

Other use cases

Now that we've seen how to use Amazon Transcribe, Amazon Translate, Amazon Polly, and Amazon S3 to creat subtitled and translated videos from a transcript generated with video content, let's think about some other wa we may be able to apply these technologies.

- Call Center Transcriptions – Since Amazon Transcribe can recognize multiple voices, it can be used to identif the caller and agent in a customer service call. Not only can all centers analyze transcripts to improve service quality, they can also use the transcripts to develop subtitles for future training videos as well.
- Translating a podcast into another language – Ever wanted to expand your podcast listener base, but didn't speak another language? Similar to the other use cases, use Amazon Transcribe to obtain a transcript of the podcast, then translate it using Amazon Translate, and "speak" it with Amazon Polly. Keep in mind that if your Podcast contains technical jargon, you can add a custom lexicon to Amazon Polly to improve the pronunciation.

Conclusion

In this blog post, we've demonstrated a working example using Amazon Transcribe, Amazon Translate, and Amazon Polly to solve the subtitling and translation problems. We've also talked through some of the key design issues you'll face when taking this example to the next level for your own projects. With the rapid progress being made in NLP technologies, it will be fun to see how far this can go in the near future.

About the Author

Rob Dachowski is a Solutions Architect aligned with the AWS National Systems Integrator team. With many years of experience architecting, designing, and implementing large scales systems, Rob has helped enterprise customers migrate to the AWS Cloud. As a professional musician and radio show producer, he has a keen interest in audio and video production tools.

Comments

Easy Anderson
5 years ago

Hi Rob,
Just want to say thank you for putting this together and making it available.
Cheers,
Andrew

2 0 Reply • Share ›

Will
4 years ago

Where does phrase = newPhrase()

newPhrase() come from???

1 0 Reply • Share ›

Will ↗ Will
4 years ago

resolved :)

0 0 Reply • Share ›

Patrick Kopitz ↗ Will
4 years ago

Hi Will, since you're the only one that posted here 10 days ago, maybe you can help on a post that I just wrote. On my side I'm stuck and I hope to find someone to help me implement that project and run it without errors. Thanks for your help if you are able to. Best Patrick

0 0 Reply • Share ›

Patrick Kopitz
4 years ago

Hi Rob (and others)
As I want to translate several videos I found your blog and installed everything step by step

(same Python version, with newer AWS and module versions). I can run your scripts fine till "Moviepy - Building video subtitledVideo-en.mp4." From that point I only have errors:

```
MoviePy - Writing audio in subtitledVideo-enTEMP_MPY_wvf_snd.mp3
chunk: 0%|1 | 73/25773 [00:00<01:50, 233.26it/s, now=None]Traceback (most recent call last):
File "translatevideo.py", line 86, in <module>
createVideo( args.infile, "subtitles-en.srt", args.outfilename + "-en." + args.outfiletype,
"audio-en.mp3", True)
File "C:\Python27\Scripts\videoUtils.py", line 109, in createVideo
final.write_videofile(outputFileName)
File "<decorator-gen-55>", line 2, in write_videofile
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 54, in requires_duration
return f(clip, *a, **k)
File "<decorator-gen-54>", line 2, in write_videofile
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 135, in use_clip_fps_by_default
return f(clip, *new_a, **new_kw)
File "<decorator-gen-53>", line 2, in write_videofile
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 22, in convert_masks_to_RGB
return f(clip, *a, **k)
File "C:\Python27\lib\site-packages\moviepy\video\VideoClip.py", line 298, in write_videofile
logger=logger)
File "<decorator-gen-45>", line 2, in write_audiofile
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 54, in requires_duration
return f(clip, *a, **k)
File "C:\Python27\lib\site-packages\moviepy\audio\AudioClip.py", line 210, in write_audiofile
logger=logger)
File "<decorator-gen-9>", line 2, in ffmpeg_audiowrite
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 54, in requires_duration
return f(clip, *a, **k)
File "C:\Python27\lib\site-packages\moviepy\audio\io\ffmpeg_audiowriter.py", line 169, in
ffmpeg_audiowrite
logger=logger):
File "C:\Python27\lib\site-packages\moviepy\audio\AudioClip.py", line 86, in iter_chunks
fps=fps, buffersize=chunksize)
File "<decorator-gen-44>", line 2, in to_soundarray
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 54, in requires_duration
return f(clip, *a, **k)
File "C:\Python27\lib\site-packages\moviepy\audio\AudioClip.py", line 127, in to_soundarray
snd_array = self.get_frame(tt)
File "<decorator-gen-11>", line 2, in get_frame
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 89, in wrapper
return f(*new_a, **new_kw)
File "C:\Python27\lib\site-packages\moviepy\Clip.py", line 93, in get_frame
return self.make_frame(t)
File "C:\Python27\lib\site-packages\moviepy\audio\AudioClip.py", line 298, in make_frame
if (part is not False)]
File "<decorator-gen-11>", line 2, in get_frame
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 89, in wrapper
return f(*new_a, **new_kw)
File "C:\Python27\lib\site-packages\moviepy\Clip.py", line 93, in get_frame
return self.make_frame(t)
```

```
File "C:\Python27\lib\site-packages\moviepy\audio\AudioClip.py", line 298, in make_frame
if (part is not False)]
File "<decorator-gen-11>", line 2, in get_frame
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 89, in wrapper
return f(*new_a, **new_kw)
File "C:\Python27\lib\site-packages\moviepy\Clip.py", line 93, in get_frame
return self.make_frame(t)
File "C:\Python27\lib\site-packages\moviepy\Clip.py", line 136, in <lambda>
newclip = self.set_make_frame(lambda t: fun(self.get_frame, t))
File "C:\Python27\lib\site-packages\moviepy\Clip.py", line 187, in <lambda>
return self.fl(lambda gf, t: gf(t_func(t)), apply_to,
File "<decorator-gen-11>", line 2, in get_frame
File "C:\Python27\lib\site-packages\moviepy\decorators.py", line 89, in wrapper
return f(*new_a, **new_kw)
File "C:\Python27\lib\site-packages\moviepy\Clip.py", line 93, in get_frame
return self.make_frame(t)
File "C:\Python27\lib\site-packages\moviepy\audio\io\AudioFileClip.py", line 77, in <lambda>
self.make_frame = lambda t: self.reader.get_frame(t)
File "C:\Python27\lib\site-packages\moviepy\audio\io\readers.py", line 185, in get_frame
self.buffer_around(fr_max)
File "C:\Python27\lib\site-packages\moviepy\audio\io\readers.py", line 239, in buffer_around
array = self.read_chunk(chunksize)
File "C:\Python27\lib\site-packages\moviepy\audio\io\readers.py", line 113, in read_chunk
s = self.proc.stdout.read(L)
AttributeError: 'NoneType' object has no attribute 'stdout'
```

Has anyone an idea how to help me/solve that issue, do I have to change some of the scripts? update the python version? probably do both? It's frustrating because the project is awsome and I was not able to find anything close but more recent, and nothing here has been updated.

And as I'm not a professional coder I'm stuck even if I can underaatand quite a bit and make the aws scripts work.

Many many many thanks for your help!
Patrick

o o Reply • Share ›

Darvin Patel

4 years ago

Hi all. Did anyone of you actually create this architecture successfully? Thanks

o o Reply • Share ›

J

Jan Gazda

4 years ago edited

Hi **@Rob Dachowski**, great article!
is there any reason for using legacy python (even in 2018)?
MoviePy seems to support python3 so it would be nice to update the article.
+ you should never assign lambda to a variable in python, it's an antipattern.

I often wonder who's reviewing the articles posted on AWS blog, they contain so much bad practices which takes me time to fix because colleagues before me just blindly copy-pasted the code from AWS blog :(

o o Reply • Share ›

M **milakunis**

4 years ago

Good Blog Actually! Thanks for sharing... Effective subtitling makes any audio, video program universally appealing. But many services fails to give flawless subtitling Dubbing services that enhances the value of content.

o o Reply • Share ›

Wes

4 years ago

Very cool! It appears that the aim of this process is to create a high-quality result, and requires a significant amount of pre-processing... significant enough to make on-the-fly video/subtitles translation either impossible or at least awkward from a UX perspective. However, what if you didn't care about the quality of the translation, i.e. you just wanted to get the gist of the audio content? Is it possible then?

o o Reply • Share ›

S **Sayak Kundu**

5 years ago

Hi Rob,
While executing the source code on a video, I frequently get errors regarding the time difference in audio and video files. Here is the error:

IOError: Error in file audio-es.mp3, Accessing time t=55.61-55.66 seconds, with clip duration=55 seconds,

Exception KeyError: KeyError(<weakref at="" 0x7f589b29c8e8;="" to="" 'tqdm'="" at="" 0x7f589b3a2c90="">,) in <object repr()="" failed=""> ignored

I thought that the code handles the time difference part before and after the editing of the video.

o o Reply • Share ›

L **Logan M** ➚ Sayak Kundu

4 years ago

Hi there. I am running into the same issue of translated audio and subsequent srt being longer than the original video clip. Did you have any luck fixing this?

o o Reply • Share ›

P **Patton Jones**

5 years ago

Hi Rob,
Thanks very much for putting up the tutorial. After getting everything setup in a jupyter notebook, I'm getting an error that I can't find a solution to. This is it:

"BadRequestException: An error occurred (BadRequestException) when calling the StartTranscriptionJob operation: The requested vocabulary couldn't be found. Check the vocabulary name and try your request again."

It seems to be directly related to the start_transcription job's "Settings parameter". Am I leaving out some necessary syntax? Thanks for your time and let me know if I can provide any more information.

Patton

o o Reply • Share ›

R **Rob Dachowski** → Patton Jones

5 years ago

Hi Patton,

Since you are using a jupyter notebook, I am assuming you are using Python with boto3, so my answer reflects that assumption.

I would check out the following:
https://boto3.amazonaws.com...

Make sure that you have defined a custom vocabulary either in the console or in your code. If you do this via the console, you should be able to click on it. It will show JSON request/response syntax validating that it is in place.

Next, Pay particular attention to the syntax for the Settings section on the StartTranscriptionJob, especially in this excerpt:

Settings={
'VocabularyName': 'yourCustomVocab'
}

The boto3 doc gives a full example of the call structure. Remember that the Settings are optional, thus they don't all need to be present if not needed. Also, make sure to remove the "," if you don't have other settings to add.

The overall doc can be found here: https://docs.aws.amazon.com....

Good luck!

o o Reply • Share ›

Jackson Pollock

4 years ago

Huge help Rob, thanks for sharing. I keep running into an issue with translate function in your srtUtils.py --> translation = translate.translate_text(Text=txt, SourceLanguageCode=sourceLangCode, TargetLanguageCode=targetLangCode)) the API states that the max bytes is 5K often txt is much greater. Any thoughts on the best way to handles this?

thanks again!

o 1 Reply • Share ›

Patrick Kopitz → Jackson Pollock

4 years ago

Hi, after diggin around with debugging my errors, I also notice that the code stops at the srtUtils.py
Any Idea? Jason, did you manage to solve that issue?

```
==> Translating from fr to es
Traceback (most recent call last):
File "translatevideo.py", line 91, in <module>
writeTranslationToSRT(transcript, 'fr', lang, "subtitles-" + lang + ".srt", args.region
)
File "C:\Python27\Scripts\srtUtils.py", line 91, in writeTranslationToSRT
translation = translateTranscript( transcript, sourceLangCode, targetLangCode, region )
File "C:\Python27\Scripts\srtUtils.py", line 248, in translateTranscript
translation = translate.translate_text(Text=txt,SourceLanguageCode=sourceLangCode,
TargetLanguageCode=targetLangCode)
File "C:\Python27\lib\site-packages\botocore\client.py", line 316, in _api_call
return self._make_api_call(operation_name, kwargs)
File "C:\Python27\lib\site-packages\botocore\client.py", line 626, in _make_api_call
raise error_class(parsed_response, operation_name)
botocore.exceptions.ClientError: <exception str()="" failed="">
```

o o Reply • Share ›

Patrick Kopitz → Patrick Kopitz

4 years ago

Ok, I just shortened my video to a 9 minutes video and that part works now