Shane Staret
Charlie Darby

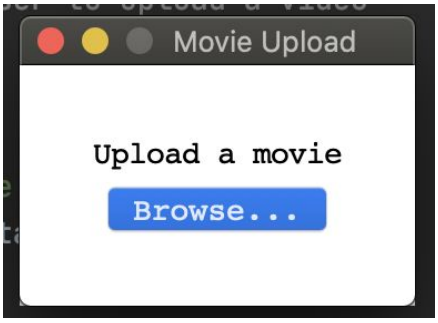
# Film Search Database Doc (Summer 2020)

## Overview

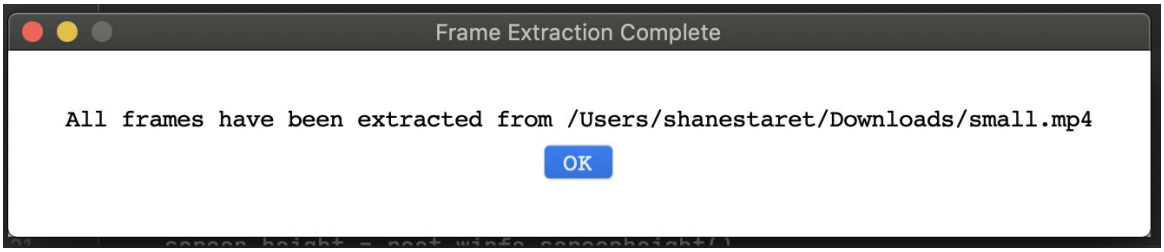
As members of the Film Search Database project for the summer of 2020, we were tasked with creating a Python tool that takes a movie file and automatically extracts each frame out of the movie into its own individual JPG. A video of the tool in action is under the "docs" directory of GitHub under the "Shane2020" branch and can also be found at this link. **We uploaded all of the code and documentation for this part of the project under the "Shane2020" branch as we did not want to push to the "master" branch yet in case there may be issues.**

## Specifics

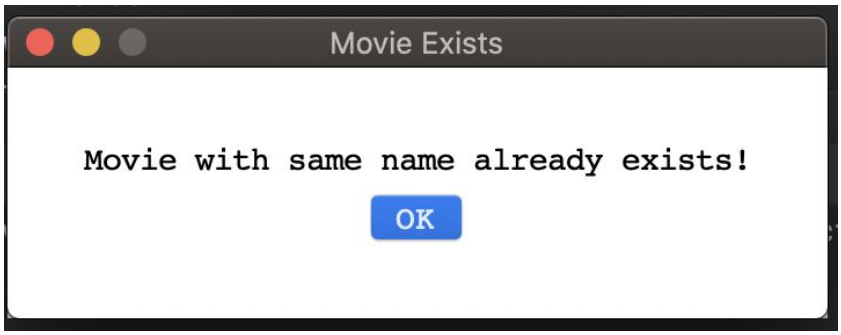
One Python file called "FrameExtraction.py" has been added to the "src" directory of the main project. This Python file contains all of the functionality for the frame extraction tool and is not dependent on any other files or folders within the main project to function. When this tool is first ran, a user is prompted with the GUI shown below:




This first GUI allows a user to upload a movie file to the tool (.avi, .mp4, .mpeg, .mov, etc.). By selecting "Browse…" the user's file directory opens on their system. After the movie is selected, the tool immediately begins to pull the frames from the file. There is no static number of frames per second that are taken. Rather, the tool looks at the metadata of the file and determines its number of frames that should be extracted per second. This ensures that there are no duplicate frames while also ensuring that there are no frames that are missed. After the tool is finished extracting all of the frames (let's say from a movie called "small.mp4"), the following prompt is shown:



Also, if it appears that a user is attempting to upload a movie whose frames have already been extracted, it will not try to extract the frames and instead will prompt the user with the below GUI and then allow them to try and select another movie:




Every movie is saved under the "movies" main directory. There is a subfolder within this directory that is the name of the movie. There is then a subfolder within this subfolder titled "frames" that contains all of the frames for that specific movie. There is no need to manually create any of these directories or files as they are all created and managed by the tool itself.

**Additional Libraries**

The OpenCV library was used to extract the individual frames from each movie file. The Tkinter library was also added to provide a GUI for the tool. The functools library was also used to allow the GUI to properly run through the code so that certain events didn't happen until the user interacted with the GUI first. The Python OS library was used to create new files/directories and to check if certain files/directories already were present.

**Issues**

We initially attempted to create a progress bar on the GUI that allowed the user to see the progress of the frame extraction tool for a particular movie. For example, if the movie had 1000 frames and the tool had extracted 800 so far, the progress bar would show 80% filled. However, there does not appear to be a Python GUI library that allows this to be properly done for our use case unless we didn't use a GUI and instead used the command line. Obviously, the command line would not be appropriate to use, so we decided to scrap this portion of the design as it seemed that Python did not have the capabilities to properly implement this.

**Further Steps to be Taken**

- Connect this to the front-end JS platform so that this can be used through the main UI
- Possibly implement a loading bar that allows the user to see the progress of the frame extraction tool as it is processing a movie. Currently, it appears this is not possible for our purposes through Python.