**Tuesday 4/14/2020**

On this day, Connor and I decided to split tasks. Connor would work on wrapping up the retrain feature and I would work on implementing an “upload” button onto our web application, so that an user has the choice of uploading an image instead of searching one up on Google Images. I also realized that by having an “upload” feature, whenever a user tests out a link to an image, but that link does not work for a certain reason, then perhaps the user could try downloading that image instead and try it out using the “upload image” feature. Better yet, instead of having the user to manually download it, I could have Director somehow download the image for itself, and then automatically pass it onto the “upload image” feature. I thought the latter was the better choice. However, instead of thinking about that, I decided to turn my attention towards getting a basic “Upload image here” feature working first.

With “st.file\_uploader(‘upload image’),” I knew it would be straightforward to implement a file uploader feature onto my website. However, my issue had to do with code line efficiency. You see, I had 68 lines of code that dealt with processing the link input, and I did not want to copy/paste those same lines of code just to be able to handle the file uploader case. I knew there had to be some way of modifying a couple lines of code so that those 68 lines of code could work in both cases. My first thought was, “Maybe I could change the ‘if’ statement to ‘if url or uploaded\_img.’ because right now I only have ‘if url.’” However, I realized that did not make intuitive sense to do that. Instead, I decided it would make sense to put all of those 68 lines of code under a function.

However, before doing that, I wanted to test the waters by first seeing if “st.file\_uploader” worked as promised. However, initially I ended up getting this error when I tried running the web application again:



I did not realize why this error occurred. At this moment, I had Streamlit version 0.56.0, and “file\_uploader” was released on Streamlit version 0.52.0 (according to the Streamlit changelog.) I decided to update Streamlit to the latest version, but even that did not work out either.

I asked my partner if he could help since I was stressed about this error; this error seemed so trivial because of my Streamlit version. His logic was to first check out the source of the error, and navigate to “ScriptRunner.py.” He found out that we had version 0.51 for some strange reason, despite the fact that I found out we had Streamlit version 0.56.0 on the command line. My partner then tried “python -m pip install streamlit” on Director, but that did not work. Then, he tried “conda install streamlit=0.57.3” on Director that also did not work. He then planned to download the files on his computer then copy-paste them to Director. When he downloaded Streamlit on his computer, his “pip” said it was out of date and that he should do “python -m pip install --upgrade pip.” He did that because “Why not?” Then, he thought maybe this line would work on Director. So he tried “python -m pip install --upgrade streamlit” and that worked. He checked the website but still had the same error. He went through the code to make sure the method did exist and it did. He then thought maybe the server works similar to how I compile code on my computer where if I run a file then change it during the run, the output isn’t affected. Maybe all it needed was a restart. The restart worked and the problem was solved.

I sincerely thanked my partner for what he did on this day. I was trying to follow along with how my partner was trying to trace the error, but I could not due to a lack of clear communication. I thought that was understandable though since my partner was laser-focused on figuring out the error. My partner did say though that next time, he would ensure that he would communicate more often so that I could help more.

**Wednesday 4/15/2020**

With the upload button finally working, I tested it out to see if it could display the image inputted. Indeed, it did work. I ensured that the only types of files that could be inputted were “jpg” or “png” files. Frankly, I thought those were the two most common image extensions, and an internet search confirmed that thought.

At first, I thought that changing the “if” block that I mentioned earlier into a function was more work than necessary. I actually saw merit in saying “if url or uploaded\_img” because in the next line of code, where I made my “img” variable (which was the variable to be passed into the CNN for input), I could simply go from

img = user\_image(url)

to

img = user\_image(url) if url else uploaded\_img.

“There we go,” I thought, thinking that I solved the issue of handling both the upload option and link options issue through simply updating two lines of code. As I was testing, however, the problem I saw with my solution was that only one option’s results could be displayed at a time, instead of both simultaneously. For example, when the website was given an input to a link, the output runs as normal. When I try uploading an image next, however, its output overrides the output produced from the link (output being the display of the image and the auto-annotations.)

Therefore, in order to prevent overriding, I decided the best thing to do was to make checkboxes for each case (link or upload) and to put the code inside of the “if” statement into a function instead, called “img\_to\_CNN(img).” That way, each checkbox can display its own results, and the user has the choice to close out one option if he or she feels displaying both results simultaneously takes up too much space/is unnecessary for any other reason. The full-fledged code that handles the checkboxes and separate outputs was:

if st.checkbox('Insert link option'):

url = st.text\_input("Insert link to image")

img = user\_image(url)

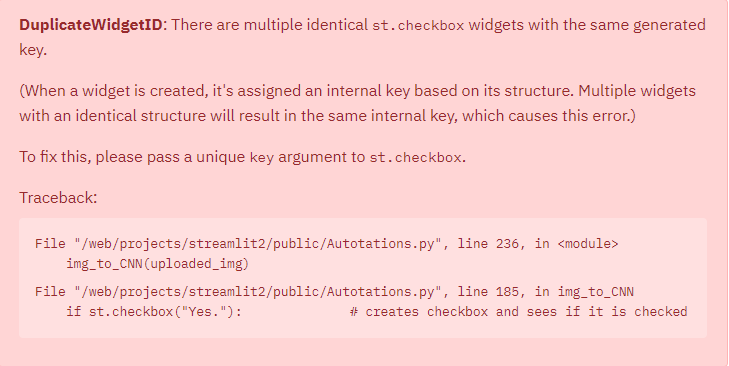
img\_to\_CNN(img)

if st.checkbox('Upload image option'):

uploaded\_img = st.file\_uploader('Upload image here', type = ['jpg', 'png'])

img\_to\_CNN(uploaded\_img)

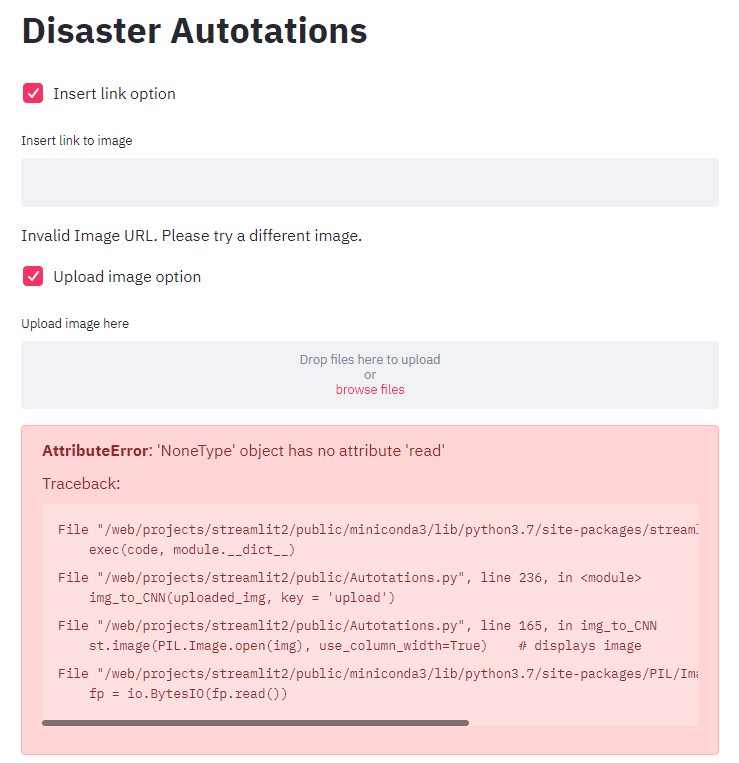
This solution seemed to work, except when I received the following error from trying the upload image after the link image:



This error, unlike the last error, was actually understandable to me. I realized I had used “st” elements (e.g. st.dataframe) in the “img\_to\_CNN” method, and because I could potentially have two calls from “img\_to\_CNN” (one for “link” and the other for “upload”), it makes sense that Streamlit would get confused with the identities of each of the “st” elements inside of the “img\_to\_CNN” method.

So, the way I handled that was by adding in a “key” argument to the “img\_to\_CNN” method, and made sure that every streamlit element (i.e. “radio”, “checkbox”, and “button”) had a “key” (e.g. st.radio(..., key = key).)

After implementing that, I was faced with this:

I remember I had dealt with this exact same error, where “Invalid Image URL...” had popped up even before I inserted a link. Then, I realized that the way I handled the situation back then was by having an “if url” statement (which I have mentioned quite a few times in this journal report.) 

So, I modified the code snippet like so:

if st.checkbox('Insert link option'):

url = st.text\_input("Insert link to image")

if url:

img = user\_image(url)

img\_to\_CNN(img, key = 'link')

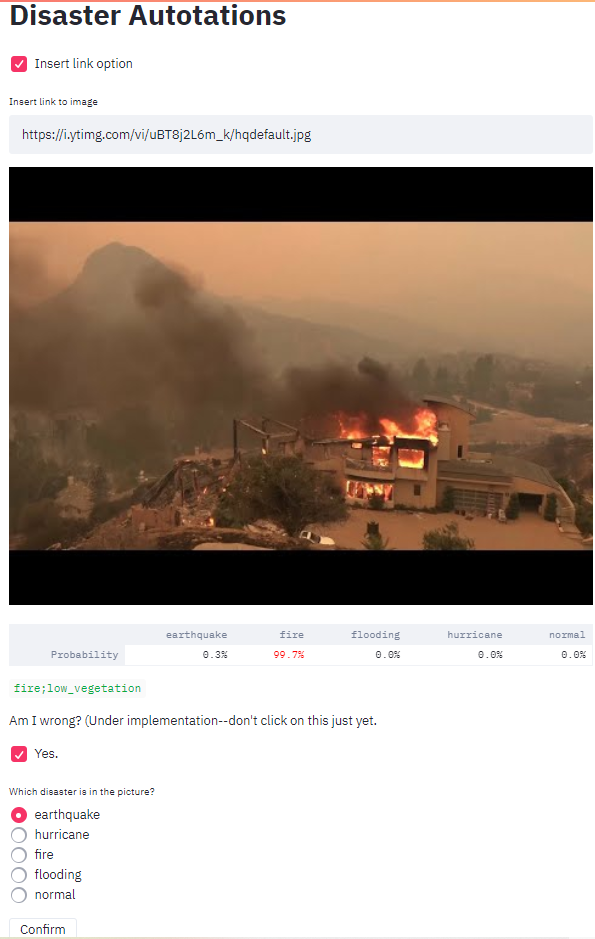
if st.checkbox('Upload image option'):

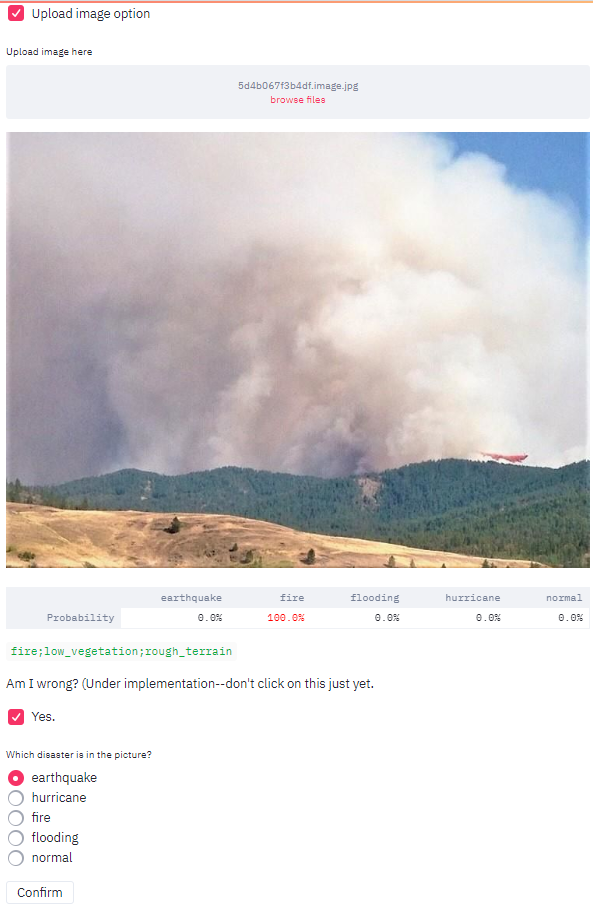
uploaded\_img = st.file\_uploader('Upload image here', type = ['jpg', 'png'])

if uploaded\_img:

img\_to\_CNN(uploaded\_img, key = 'upload')

As a result, all worked well.



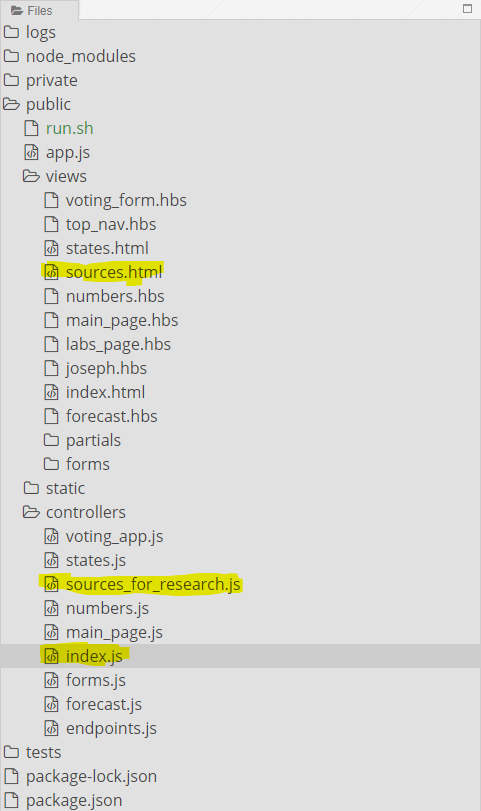


Next, I updated the sources, since the sources I had on the website were not current with all of the ones I had.

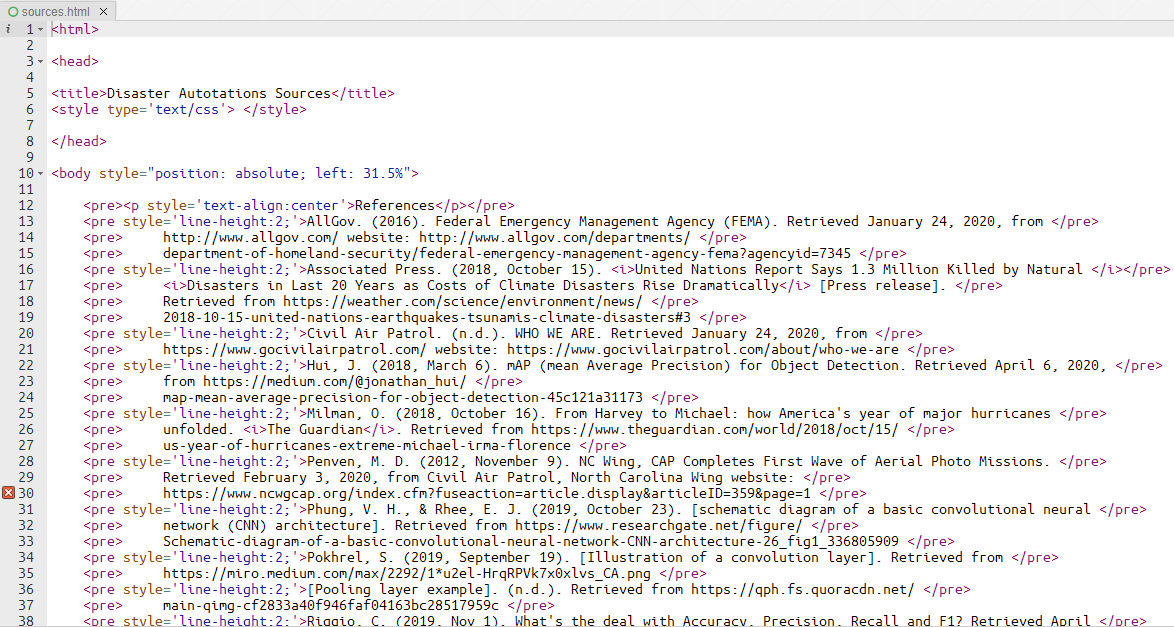
Instead of displaying a .jpg for my sources, I decided to try and figure out a way to have a sources link instead, where the user could click on “Sources” and it would take them to a different page. I thought about if there was a way to run Javascript and run Python simultaneously on the run.sh file. I decided that there probably was, but it would not be worth the time and effort to, especially since I have an easier solution. However, I did not want this possibility to escape my mind, since it is rather more intuitive to have the web page “streamlit2.sites.tjhsst.edu/sources.”

The easier solution I implemented was that I could make the sources page on my “user.tjhsst.edu/2020jlee” server instead. I had been taking “Web Application Development” with Mr. Kosek during my 2nd semester senior year, and thanks to him and NoodleTools (a web app that produces APA citations), I knew how to take the .html site that produced the sources, “view page source,” copy/paste the HTML code into a new HTML file called “sources.html” on that server, and then proceed on to make a new endpoint called “user.tjhsst.edu/2020jlee/auto\_anns\_sources.”

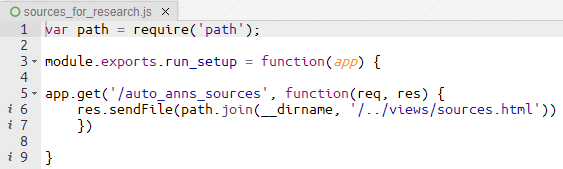
In picture, my “user.tjhsst.edu/2020jlee” directory system looked like:



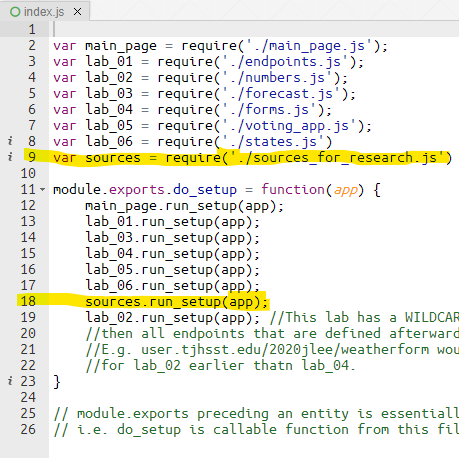
(All the highlighted files represent either modified files or newly created ones. “sources.html” and “sources\_fore\_research.js” were newly created, and “index.js” was modified.)



(This is “sources.html.”)

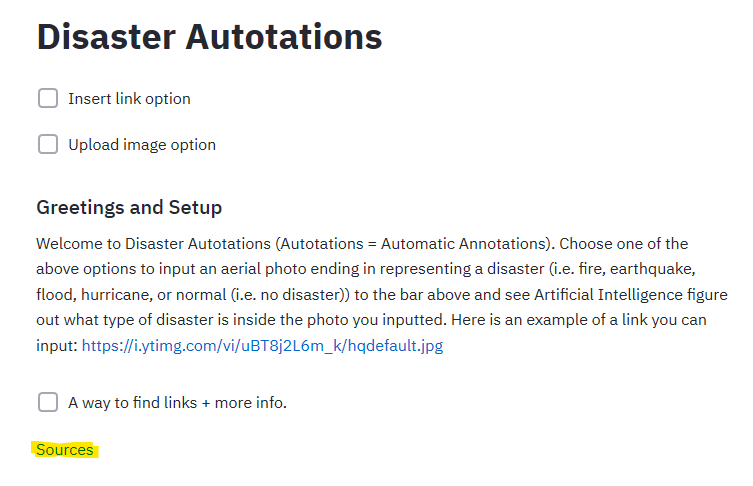


(This is “sources\_for\_research.js.”)



(This is “index.js.” The highlighted parts show what new lines of code I added to this file.)

Finally, here were images representing how users would see my new additions:



(This is what the website looked like at its finest. The highlighted part indicates where the user can find the sources. It is a hyperlink that will take the user to a new page. The code to render “Sources” was:

source\_html = "<a href = https://user.tjhsst.edu/2020jlee/auto\_anns\_sources target = \_blank> Sources </a>"

st.write(source\_html, unsafe\_allow\_html = True)

I set “target = \_blank” so that the hyperlink opens up a new browser window, instead of opening up the hyperlink on the same window displaying [streamlit2.sites.tjhsst.edu](http://streamlit2.sites.tjhsst.edu).)



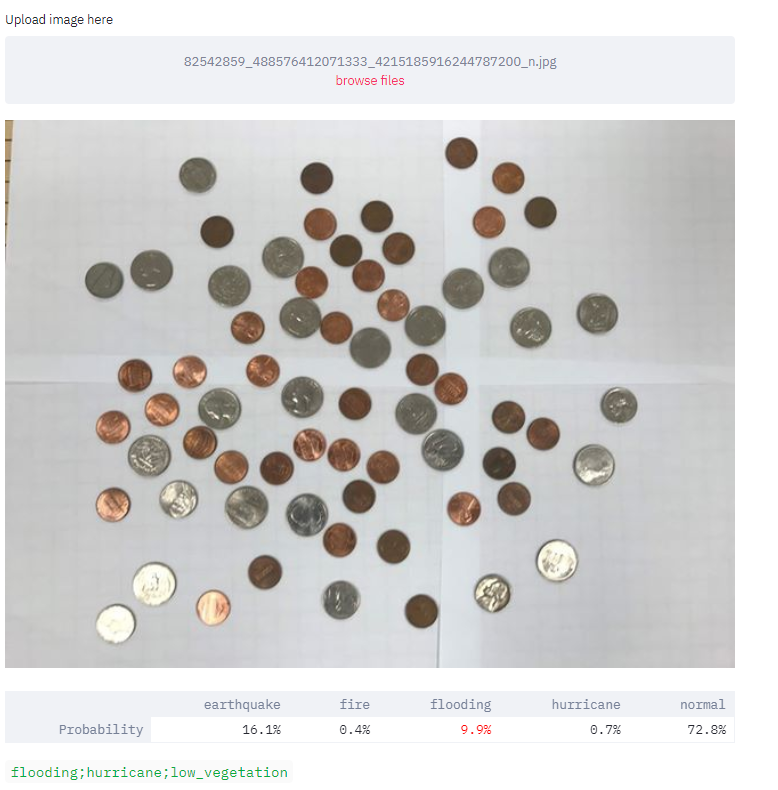
(This is where the “Sources” link points to. The address is: <https://user.tjhsst.edu/2020jlee/auto_anns_sources>.)

**Friday 4/17/2020**

My partner found an image that my Streamlit application failed on:

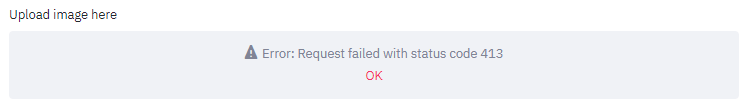


However, when I tried this image, I received proper results:



Anyhow, this image was only a test image, and had no relevance to any disaster whatsoever. However, this does raise a concern knowing that a proper image that should have worked for my partner, did not.

However, I was able to receive the error when I tried inputting a new image (any new image, that is.)



After some investigating, I came to agree with my partner that what was causing the error were the sizes of the images we tried. Streamlit failed with an image that was 3,925 KB but easily was able to process a 126 KB one. What was weird, however, was that Streamlit said it could support files up to 200 MB, and clearly the files we had tried out was much less than that size.

After seeing that no one in the Streamlit community posted about an error like mine, I made a post myself. My initial post had received a proper response from a Streamlit developer, so I was hopeful that my new post would receive a proper response as well.

Then, I decided to check over this entire journal report; this was probably my longest journal report yet (but that was mainly because of the amount of images I had put in.) As I was checking this report over, however, I also took some time to see if I could make a “streamlit2.sites.tjhsst.edu/sources” using node.js, but that attempt was unsuccessful. I tried putting in “node app.js” under the last line in “run.sh”, and I even tried “/web/projects/streamlit2/node\_modules/node run Autotations.py --server.port 12379,” but that was unsuccessful as well. This was “app.js”:

// -------------- load packages -------------- //

var express = require('express');

var app = express();

var path = require('path');

// -------------- express initialization -------------- //

app.set('port', process.env.PORT || 12379 );

app.get('/sources', function(req, res) {

res.renderFile(path.join(\_\_dirname, 'sources.html'))

console.log(3);

});

I had also checked with the Streamlit community, and there they mentioned about using Flask, but the workaround was not secure/it needed to be proceeded with one’s own caution. Overall, there was no “Streamlitic” way to create new endpoints on a web application.