**Friday 3/27/2020**

On March 22nd, I posted on the Streamlit community, asking if there was a way to have a right-side sidebar. On this day, a Streamlit developer responded, and said that the following code worked to get a right-sided sidebar:

import streamlit as st

html = """

<style>

.reportview-container {

flex-direction: row-reverse;

}

header > .toolbar {

flex-direction: row-reverse;

left: 1rem;

right: auto;

}

.sidebar .sidebar-collapse-control,

.sidebar.--collapsed .sidebar-collapse-control {

left: auto;

right: 0.5rem;

}

.sidebar .sidebar-content {

transition: margin-right .3s, box-shadow .3s;

}

.sidebar.--collapsed .sidebar-content {

margin-left: auto;

margin-right: -21rem;

}

@media (max-width: 991.98px) {

.sidebar .sidebar-content {

margin-left: auto;

}

}

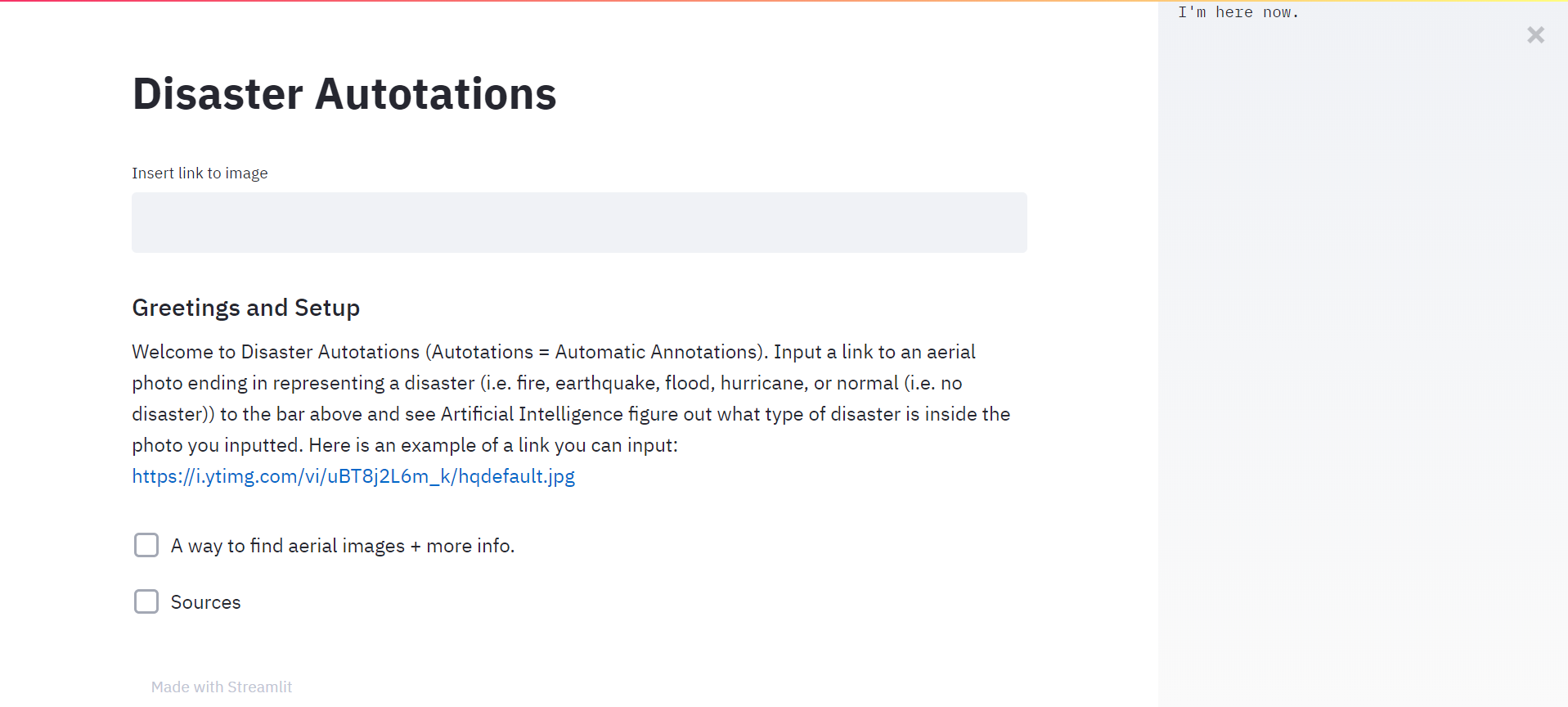
</style>

"""

st.markdown(html, unsafe\_allow\_html=True)

st.title("New Sidebar")

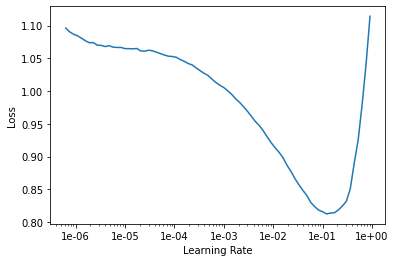
st.sidebar.text("I'm here now.")



Sure enough, there was the sidebar, as seen in the above screenshot. My issue with this, however, was that I did not know how to get two sidebars at once--a left-side one and a right-side one--from that code alone. I could only have a left-handed one or a right-handed one. I would have to think of some creative way, using HTML and CSS, to have both sidebars appear at once. I wanted one sidebar to show the instructions, and the other sidebar to show the extra information (i.e. background/motivation, solution, and progress and for the future.) Then, I realized maybe having just one sidebar to the left could also be sufficient. However, I wanted something to appear to the right-hand side to minimize white space.

I then replied back to the Streamlit developer, thanking him first and then asking if there was a possibility to display both a right-sided sidebar and a left-sided sidebar simultaneously. He said that that was not currently supported on Streamlit but he would reply back at some point if having both sidebars appear at once is possible on Streamlit.

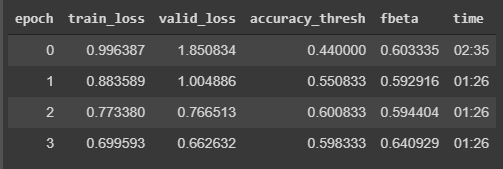
As I progressed along with the tutorial, what I found interesting was:



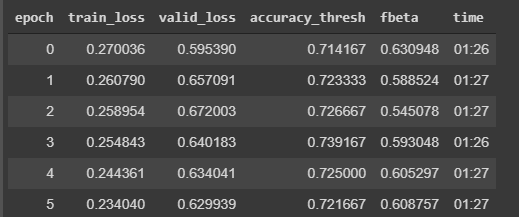
I did not know how to interpret this graph at first. As I searched online, I found out that this was a test of different learning rates and how they all stack up in minimizing the loss function. Essentially, the graph was created by neural network training sessions that started off with a learning rate of around 1e-06 (as shown in the graph), and then progressed onto a higher learning rate after each epoch. A total of 17 epochs were tried. I learned that what was essential to catch in machine learning was to not choose the learning rate that produced the minimal training loss according to the graph, because then at that point, the neural network could haywire back and forth and never reach towards a minimal loss. Instead, one would choose the learning rate whose descent rate is the highest (i.e. whose derivative is the lowest.) That way, the learning rate would be just right (not too high and now too low.) In this graph, the point of steepest descent lands around the learning rate coordinate 1e-02 (i.e. 0.01.)

**Saturday 3/28/2020**

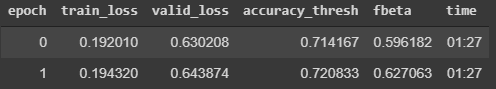
I began training the multilabel neural network. What I found interesting was the tutorial’s usage of an “fbeta” metric. The only metric I had been familiar with was mean squared error. I then learned that the “fbeta” metric essentially takes into account two other metrics, namely precision and recall. The following article helped me to understand precision and recall better: <https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c>. Recall takes a look at the total number of data points of a particular class in a dataset and shows what proportion of those data points can the model correctly identify. Precision looks at the total number of data points in which a model thinks is a particular class and shows what proportion of those data points were actually that particular class. When one trains a model to increase one of the aforementioned metrics, the other metric decreases. Thus, that is the raison d'être for “fbeta”, so that the model could have the best balance of both metrics. The higher the “fbeta” score reaches one, the better.



After a couple more cycles of four epochs each, and a couple more cycles with six epochs each, the results for the last cycle were:



What was interesting was that while the training loss is steadily decreasing, the fbeta score fluctuates between the high 0.50’s to the low 0.60’s.

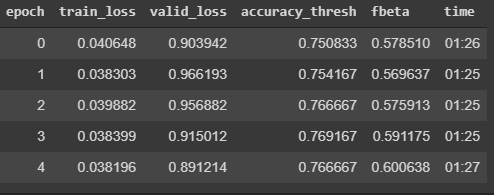


What I also noticed that was interesting was the ability to unfreeze layers. Apparently, in normal training, certain layers are freezed (which means the weights associated with those layers stay constant) so that the training process could be done quicker. In his tutorial, Tanner decided to unfreeze the layers in order to train from 128x128px images to 256x256 px.

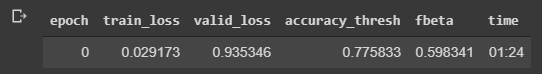
I was confused about the need to shift resolutions in the first place, since when I had tested the single-class fast.ai neural network on my user interface, the neural network did not complain about the resolution of the google image I inserted/it did the resizing on its own probably. In other words, I encountered no errors for using google images of different resolutions.

I decided to unfreeze the layers (since I was curious about the results of that action) and plotted the learning rate vs. loss graph again as I showed earlier in this journal report. I wanted to see if training five epochs with no default layer freezing would change my network for the better (or not.)

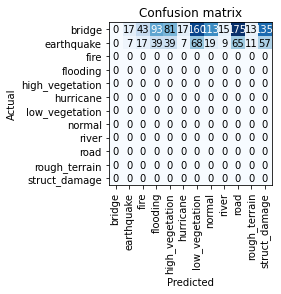
Results looked like below after around two more hours of training.



After around a half an hour later, here came my best result:



I ran a confusion matrix like I did for single-label:



I was not really sure what to make out of these results since multilabel classification is fundamentally different single-label classification. When looking at the documentation though, apparently multilabel neural net interpretation methods have not been developed yet, and that we were free to implement one ourselves. So, I decided to take the confusion matrix above with a grain of salt. I also recalled to not set my expectations too high either. Despite having achieved a great low error rate, the dataset we used was most likely not sufficient to perform well in any given photo. “Well, that’s exactly what the multi. neural net. retrain feature would be for then” I thought. Then, I exported my model to Google Drive, downloaded the .pkl file, and uploaded that file to Director.

**Sunday 3/29/2020**

On this day, I finally implemented the multi-label neural network feature onto my website. To accomplish this, I had to modify the “predict\_img()” method by adding and modifying a couple pieces of code and doing the same for the “try-catch” statement a little further below to be able to display the multi-classification results.

In detail, “predict\_img()” now looked like:

def predict\_img(img):

img = open\_image(img) # open\_image is a fast.ai built-in method

defaults.device = torch.device('cpu')

path = Path("") / 'models'

s\_learn = load\_learner(path, 'single.pkl')

m\_learn = load\_learner(path, 'multi.pkl')

return (s\_learn.predict(img), s\_learn.data.classes), (m\_learn.predict(img), m\_learn.data.classes)

And further below:

((pred\_class, pred\_idx, s\_probs), s\_classes), \

((pred\_classes, pred\_idxs, m\_probs), m\_classes) = predict\_img(img) # runs image through nn and returns various outputs

#st.write(p\_c)

#st.write(p\_i)

#st.write(o)

certainty\_df = (pd.DataFrame(np.matrix(s\_probs), columns=[x for x in s\_classes]) \* 100).round(1) #

certainty\_df = certainty\_df.astype(str) + '%'

certainty\_df.rename(index={0:"Probability"}, inplace=True) # Displaying output in table form

st.dataframe(certainty\_df.style.apply(highlight\_max, axis=1)) #

st.write(pred\_classes)

(Highlighted parts indicate new parts/modified parts.)

Then, I thought about exactly how I wanted to phrase the annotations. When a normal user sees annotations separated by a comma, they may not know how to interpret those annotations quite clearly. I wanted to include sentences that gave more insight about the annotations. An example of what I mean is:

This photo:

* was recently affected by [disaster type(s)].
* has a road/roads.
* has low vegetation/high vegetation.

The hardest part of course would be implementing an algorithm that can take in the annotations separated by semicolons and displaying a list as output (as like the example shown above.) I did not find a Streamlit method to display lists like with unordered list tags in HTML during my search. However, then I supposed that this was another instance where unsafe\_allow\_html=True came into play again.

However, displaying the ordered list was actually the least of my concerns. I was still trying to figure out the English format for my annotations. I thought to myself that it would not make much sense, for example, if I said that the photo was recently affected by flooding if flooding was immediately happening in that photo. However, then I stepped back and I realized the Civil Air Patrol would probably be taking photos post-disaster, rather than during a disaster. So, I decided that “was recently affected by” was appropriate after all.

**Thursday 4/2/2020**

I recalled that our goal was to help out the Civil Air Patrol (CAP). Therefore, when I had planned to display the annotations in bullet points and sentence fragments, those could waste time for the CAP since those would take a longer time to read than the time it would take the CAP to read the straightforward annotations separated by semicolons.

I thought then maybe I could display the English-sentence annotations for normal users and colon-separated annotations for Civil Air Patrol. On the website, I could have classification results look like:

Civil Air Patrol:

Earthquake;flooding;road;struct\_damage

This environment where this photo was taken:

* was recently affected by an earthquake
* has flooding going on inside.
* has a road/road system.
* has structural damage.

Then I decided that this format would not make much sense though because normal users may be confused about the “Civil Air Patrol” up at top and the Civil Air Patrol would waste time trying to read potentially important information (which of course is not (it is repetitive)) down at the bottom. I decided that in my tjSTAR presentation, it would be better to explain that each annotation is separated by a semicolon and each annotation describes something going on inside of the photo, or potentially an annotation that recently affected the state of the photo.