**Capstone – Milestone 4**

David Bui

The College of Science, Engineering, and Technology, Grand Canyon University

DSC-590 - Data Science Capstone Project

Edward Ofori

February 15, 2023

**Modular Testing**

At the start of Modular testing the base project was broken down into pieces from two different perspectives, the first in regard to the end user and user interface, and the 2nd was done from internally within the ai models. The end goal is to give access to the user of both analysis perspectives. The beginning stages of a modular test are as follows:

- Start with an AI that has already been developed through previous iterations and have it run on its own for basic tests.

- Then add the next step by having one or more humans provide input based on their role.

- Add additional AIs if needed until there's enough information available at this point so the decision making can be done effectively.

- Once sufficient data has been collected, make any changes necessary to ensure that all parts of the system are working efficiently together.

- Ensure the quality of the data through several tests and continue developing the program until no further improvements need to be made before moving onto another part of the process.

After each module has completed successfully move to the next stage in development. If problems arise during these processes, then return back to previous modules to fix them where possible instead of wasting time redoing work when new modules become available. This will speed up overall progress while reducing wasted effort. As long as the current design meets its goals then you've moved on towards the final product and should now proceed forward without worry since your model isn't likely going to break down anytime soon due to past successful iterations being tested prior to this version of the module.

|  |
| --- |
| Test Case Name: Patient loading user data to web-based application |
| Priority: High |
| Module: class User\_Interface()  def side\_bar(self,userimgs):    upload = st.sidebar.file\_uploader("Upload a data file",type=(["jpg"]),accept\_multiple\_files=True)  if upload is not None:  menu = ['Home','Patient Analysis','Data Analysis','Classification','Regression']  navigation = st.sidebar.selectbox(label="Selection Menu", options=menu)  # App pages  # Homepage  if navigation == 'Home':  with st.container():  Home\_Page.home()  # Runs 'Patient Analysis Page' app  if navigation == 'Patient Analysis':  with st.container():  Patient\_Analysis.patient\_analysis(upload)  # user starting page  else:  Home\_Page.home() |
| Test Objective: Allow the User to upload their own data into the model and return a diagnosis |
| Result: This works well. |

|  |
| --- |
| Test Case Name: The Patient has full access to the h5 trained mkkodel |
| Priority: High |
| Module: class DataObject()  stream = st.file\_uploader('Load in Model here', type='zip')  if stream is not None:  myzipfile = zipfile.ZipFile(stream)  with tempfile.TemporaryDirectory() as tmp\_dir:  myzipfile.extractall(tmp\_dir)  root\_folder = myzipfile.namelist()[0] # e.g. "model.h5py"  model\_dir = os.path.join(tmp\_dir, root\_folder)  #st.info(f'trying to load model from tmp dir {model\_dir}...')  classifier = tf.keras.models.load\_model(model\_dir) |
| Test Objective: To transfer trained Neural Network models over, so that the User doesn’t have to train them to save time. |
| Result: The h5py files sometimes result in a bug and have to be reloaded. |

|  |
| --- |
| Test Case Name: Diagnosing |
| Priority: High |
| Module: predict()  def predict(img,classifier):  shape = ((128,128,3))  #model = tf.keras.Sequential(hub[hub.KerasLayer(classifier, input\_shape=shape)])  test\_image = img.resize((128,128))  test\_image = preprocessing.image.img\_to\_array(test\_image)  test\_image = test\_image/255.0  test\_image = np.expand\_dims(test\_image,axis=0)  class\_names = ['Abnormal','AMD','Cataract','Glaucoma','Myopia','Normal']  prediction = classifier.predict(test\_image)  scores = tf.nn.softmax(prediction[0])  scores = scores.numpy()  image\_class = class\_names[np.argmax(scores)]  result = "{}",format(image\_class)  return result |
| Test Objective: Ensure that the model diagnose each eye of the patient accurately. |
| Result: This has been correctly done, but the image in the green text is too small and hard to see. |

**Requirements Testing**

Chart, line chart

Description automatically generated

The requirements of this model were to go through a f1 testing, precision, and accuracy testing. This accuracy and val\_accuracy chart can measure these terms, and through observation we can find that our model functions just under 90% accuracy. Which fits within the realm of the goals that were set from previous milestones.

Chart, line chart

Description automatically generated

Much in the same way we can wee the separation more vividly within the Loss chart. Between the Loss and Accuracy charts our testing would suggest a bit of overfitting could be taking place. The Precision and recall values seemed to follow closely behind. Our model had trouble getting good recall numbers but still achieved high precision levels which means it did well on recognizing the correct classes.

After running some training sessions using keras and tensorflow the loss value began increasing again even though the accuracy kept staying about the same throughout. It was determined that something must have changed between versions because the old code used a custom loss calculation while the newest uses cross-entropy. In order to resolve this issue I went ahead and tried rewriting my code to use the newer method but ran into some unexpected errors such as not being able to import the Keras layers, etc...

I'll keep looking for ways around this problem since this seems like a common obstacle to those starting with deep learning and machine learning. Maybe try importing other libraries? Or maybe trying adding in extra lines of code to forcefully tell keras that certain things exist even though they're undefined? Who knows! But whatever the answer is, I'm excited to see where it leads me!

|  |  |  |
| --- | --- | --- |
| **Component:**  The webpage can open without errors for the end user | | |
| **Checklist** | | |
| Type | Passed | Comments |
| Functionality | Passed | The webpage opens just fine and no errors seem to be present during naviation. |
| Performance | Passed | The initial load is a slow because the user has to upload an h5py file. |
| Usability | Passed | The user is given instructions on how to utilize the web-app |
| Accessibility | Passed | The User can download the app from GitHub with a provided link. |

|  |  |  |
| --- | --- | --- |
| **Component:**  The User is able transverse through the tabs with and without uploading data. | | |
| **Checklist** | | |
| Type | Pass | Comments |
| Functionality | Passed | Tabs are working fine and give no error. |
| Performance | Passed | The user can look at each individual image for each patient with user input just fine. |
| Usability | Passed | This part I’m quite happy with and it works smoothly. |
| Accessibility | Passed | The user can gain the data through their own sources or mining it, a link will also be provided to let them test the model. |

**System Testing**

For system testing the convolutional neural network, a scan can be done that is similar to a get function. This is done outside of the web\_app since Streamlit didn't have sources or packages to articulate this. The AI model seems to be able to grab images from a set that fit a classification input quite comfortably. An amount can also be given. Before this the dataset is always shuffled up. Testing the User's model is an entirely different approach that is currently left out of the web\_app but might be implemented later on. Another form of testing the ai model system was by

- Making sure that the output wasn't incorrect after giving the wrong inputs.

- Checking what type of results were returned and checking if they matched the expected result.

- Looking over some examples and seeing how close they are to real life cases.

**Operation and Maintenance**

The user is given instructions with the readme files and on the home webpage, there will also be a table to help the user create their own model with given parameters and link to further data resources for training. After receiving feedback regarding issues discovered, the code itself needs improvement and fixes must be made whenever applicable. This means that future versions are constantly improving upon earlier ones; however, this does not mean that newer builds don't function completely independently of older models because sometimes it just becomes difficult to find ways around bugs created due to users using their own models.

The user must upload their own fundus images, along a given Convolutional Neural Network zip file. Once the data is uploaded they will be directed to the next page tab where a patient ID number is asked of them. The fundus images they upload must have a left and right indicator written on them along with a given patient ID. If done correctly and with a functional trained model, images should be displayed of the patients eyes along with a diagnosis in green underneath.

**References**

Dresossi, T., Ghosh, S., Vincentelli, A., & Seshia, S., (2017), Systematic Testing of Convolutional Neural Networks for Autonomous Driving, Reliable Machine Learning in the Wild, <https://arxiv.org/pdf/1708.03309.pdf>

Soad, A., & Lamiaa, F., (2019), Deep Convolutional Neural Network-Based Approaches for Face Recognition, Applied Sciences, 10.3390/app9204397