Rostyslav Hnatyshyn 1219465260

Akash Devdhar 1217196097

Vraj Kapadia 1217192587

Project 3

10/15/2020

We were given a working piece of software, yet it was clear that the developer paid little to no attention to the design aspects of software development. The given software did not use any of the standard architectures or design patterns. The project's original developer used a variety of different frameworks and libraries to write this application but they seem to have only working software as an end goal in mind – the code could be consolidated, the classes within the project were tightly coupled, and there was plenty of useless code entangled within the project. Unfortunately, the only patterns we found that were being used were within the packages that the original developer used – the original developer did not use any on his / her own.

As soon as we started working on the project, we immediately renamed the existing classes into ones that would better describe their functionality as well as moving everything into one package. We started by first implementing the MVC pattern, moving all of the business logic that drove the buttons into the Controller class and removing anything that didn't have to do with the GUI being displayed on the screen from the DigitRecognizerView class (formerly UI). We realized that there were quite a few classes that were not being used (AccuracyCalculator, Convolution, the training classes, the Mnist example class) that we removed. Next, we focused on cleaning up the look of the UI by removing the training components and then putting all of the functionality into a JMenu class, letting the DrawingCanvas and ResultView take up the entirety of the JFrame. We

also stripped the loading view because it did not add anything useful to the program and contributed to the “spaghettiness” of the code and slowed it down. We also removed the slf4j wrapper for the logger because it contributed to the computational overhead of the program without adding any new functionality – the training module was removed for the very same reason.

We also removed any instances the same code was being used in two different methods and consolidated it into one function. This was especially prevalent with how the Canvas view returned an array of doubles for the neural networks to predict on: in the original code, there was a complex series of operations being called directly on the canvas from the UI that we consolidated into a single method, leaving the Controller to just call one method to get the data it needs to predict on the input from the canvas. This consolidation led to making it easier to change the internal workings of the canvas for future programmers.

We implemented the observer pattern for the ResultView because it logically follows that the label would watch the values produced by the neural networks. We also ended up adding a text area that would act as the logger inside the UI so that the end user could understand what was happening behind the scenes when the neural networks were being used. We made the JLoggerArea a child of JTextArea that would listen to the logger’s handler and update its values as output was produced.

By creating a controller class, we decoupled the functionality of the UI from its presentation, which will make it much easier to work on this program in the future. Our decoupling throughout the application will facilitate the process of extending the application in the future. We also noticed that the code is now much easier to follow and understand for someone who is looking at the application for the first time. Finally, the speed of the application has appreciably increased because there is a lot less useless code being run in the background.