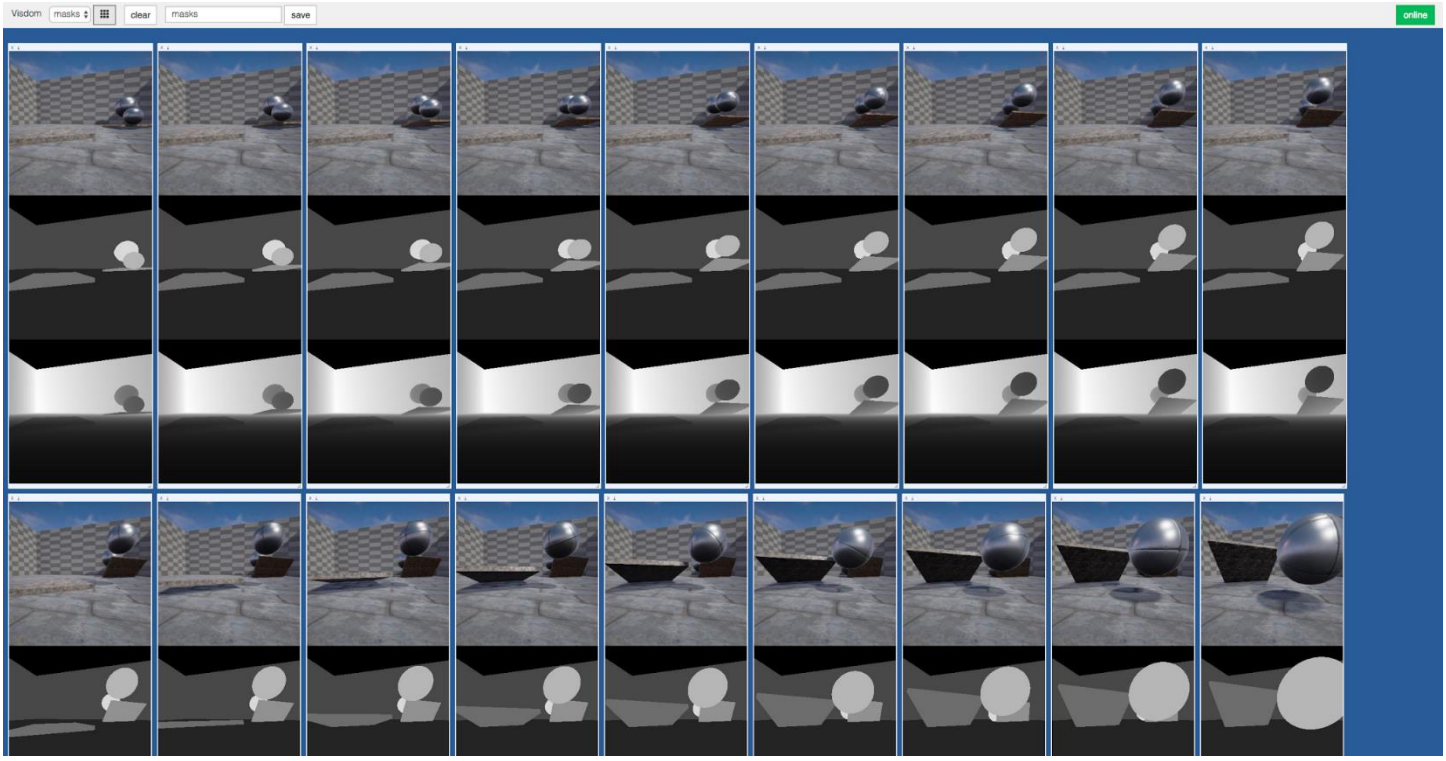
**[100 pts] STDISCM-CSC611M Problem Set: Image Classifier Dashboard**
By pairs

Instructor: Neil Del Gallego
neil.delgallego@dlsu.edu.ph

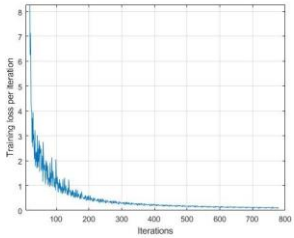
**GENERAL PROGRAMMING INSTRUCTIONS**

AI engineers often employ a variety of visualization and debugging tools when training their AI models. Most commonly used tools are either Visdom or Tensorboard. You are tasked to implement your own standalone visualizer.




**App Behavior:**

- Assuming a standard image classifier training scheme, such as using convolutional neural networks, display the input image batch, and the corresponding prediction vs ground-truth label during training.
- Show the training loss per iteration.




- The dashboard app constantly communicates with an external ML application (e.g. Pytorch or Tensorflow) during training. The intended usage is as follows: The user starts the dashboard where it will listen to an ML training application. Once the training commences, the results are displayed in real-time, on the dashboard. You are required to apply distributed computing concepts, such as RPC, and limited fault tolerance for this output.
- The dashboard is interactive and should continuously update throughout its lifetime.

- UI mockup is shown below:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Image | Image | Image | Image | Image | Image | Image | Image |
| Image | Image | Image | Image | Image | Image | Image | Image |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Predicted label | Predicted label | Predicted label | Predicted label | Predicted label | Predicted label | Predicted label | Predicted label |
| Predicted label | Predicted label | Predicted label | Predicted label | Predicted label | Predicted label | Predicted label | Predicted label |
| Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label |
| Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label | Ground-truth label |

| |
|---|
| Training loss plot |

- Given **N** image batch size, the **N** images fed to the network must be displayed as a tile. The image displays are refreshed per new image loaded.
- Following the same tile layout, display the prediction vs ground-truth label on a separate tile set.
- For standardization, **16** tiles must be displayed.
- The training loss plot can be updated anywhere between **N** to **N * 10** batches.
- Refer to the Visdom dashboard tool for reference on how your dashboard should also behave: https://github.com/fossasia/visdom

**Restrictions**

- There is no programming language restriction for this requirement. However, you are limited to deploying your software as a web application, or a standalone Windows application (.EXE) only.
- You can use any plotting library for the training loss visualization.
- Maximum image sizes are 512 x 512. Any resolution larger than 512 may no longer fit in memory. For very small image sizes (32 x 32), display and interpolate them at a viewable resolution – x2 scale.
- The minimum batch size is 16. If this cannot fit in memory during training, then a smaller batch size during training is permitted. However, you must aggregate the mini batches, when visualizing the images – 16 image tiles. For any batch larger than 16, then only display random 16 images from the current batch.

**Other Instructions**

- The frame rate must be displayed at all times.
- You can add frame delays to properly test the concurrency of your dashboard.
- Use a standard network architecture and simply follow existing training procedures (e.g. training a CNN). What's important for this output is the dashboard, and not your network/training pipeline.

**Grading Scheme**

This activity is worth 100 points with the rubric seen below.

| **Completeness** | | | |
|---|---|---|---|
| 5 pts | 10 pts | 20 pts | 30 pts |
| Only one feature is observed to be working properly. | One of the features are not functioning properly, or requires a workaround: image display, training loss plot, prediction vs ground-truth labels | The app has all features implemented, but with major issues observed: image: image display, training loss plot, prediction vs ground-truth labels | The app has all of the features implemented correctly: **image display, training loss plot, prediction vs ground-truth labels** |
| **Frame Rate Consistency and Latency** | | | |
| 2 – 10 pts | 11 - 19 pts | 21 - 30 pts | 34 - 40 pts |
| The frame rate never reached 60 FPS throughout the duration of the program.<br><br>There is a large latency between the dashboard and training application ( > 4 second delay). Sometimes, real-time updates are halted, and the dashboard doesn't attempt to reconnect to the training application.<br><br>The system can potentially suffer from a catastrophic failure. Fault-tolerance and error recovery wasn't considered in the design. | The frame rate is consistently jittering between 20 FPS – 60FPS. The application cannot maintain a steady frame rate.<br><br>There is a noticeable latency between the dashboard and training application (>3 second delay).<br><br>The system can potentially suffer from a catastrophic failure. There is an attempt to implement error recovery techniques, but it is unclear if this is working all the time. | The frame rate is consistently at least 60 FPS with occasional drops to ~30 FPS.<br><br>The latency between the dashboard and training application is moderate (<3 seconds delay).<br><br>There is a clear implementation of error recovery and fault tolerance, but occasionally, the system experiences long timeouts/failures. | The frame rate is consistently at least 60 FPS with occasional drops to ~50 FPS.<br><br>The latency between the dashboard and training application is barely noticeable. (<1.5 seconds delay).<br><br>The overall system is observed to be robust and has limited error-recovery techniques whenever there's an unreliable connection. |
| **Technical Documentation** | | | |
| 0 pts | 10 pts | 20 pts | 30 pts |
| No document was submitted, or it is very incomplete. | The technical document is around 50% complete, omitting most of the details needed to fully understand how the system is implemented. | The technical document is around 80% complete, with some details not thoroughly discussed. However, the design of the system can still be understood in the document. | The technical document is clear and fully explains the implementation and framework of the proposed system. |

NOTE: The forward and backward passes per batch don't count towards the delay. Once the network performs its backpropagation, and results are already available will the counting of delays be measured.

**Submission Instructions**

- A 1 - 5 min video demo showing the sample program in MP4.
- Submit a PPT that serves as your technical report that contains the following:
    - A general overview of how you designed your dashboard application.

    - Discuss how you perform concurrent scheduling and communication with the training pipeline. Show your RPC design and your message protocol.
    - Technical explanation on how you ensured a consistent frame rate. Discuss additional implementations you created to ensure that a constant FPS is met.
    - Show some empirical proof that your system is fault tolerant. Provide some discussion of how you ensure that your system can recover from some failure/long delays between server and client.
    - Provide code snippets supporting your technical explanation.
- Submit a Github/GDrive link containing your source code. Include a README.md file for instructions how to run your program. The program should already include any external dependencies.