Visualization Tool

POC for C++ backend:

Overview:

- The main aim of the backend is to write down the information to the file in an asynchronous manner.

- After referring to Tensorboards implementation. There are basically two ways in which users prefer mostly:
  - Customize logging by writing equivalent code.
  - Using predefined callbacks to write metrics for models.

- The main difference between the above two mechanisms is that in the second use, the user uses model.fit() in the second epoch in which he does not iterate on his own on the dataset.

- So basically the callbacks take care that at the end of every step or epoch the summary is automatically logged.

- The logging and writing would be the same in both of the above mechanisms.(Code reusability).

Rough Timeline

So, I am thinking of implementing the backend in the following steps.

1. Implement the backend such that the manual logging becomes possible by the user.
2. Implement callback for doing similar things at the end of epoch automatically.

3. Implement the frontend for visualizing the metrics.

Alternatively, we can also implement in the following way:

1. Implement the backend such that the manual logging becomes possible by the user.
2. Implement the frontend for visualizing the metrics.
3. Implement callback for doing similar things at the end of epoch automatically.

So according to the above two timelines we should finalize the logging design as quickly as possible.

Restrictions and Assumptions for backend design:

Some important restrictions while designing logging mechanism:

1. Asynchronous log writing:
   The log writing should be asynchronous so that model does not wait to log information into files which will remove overheads.

Some additional assumptions / restrictions while designing:

1. One model, one log.
   I am of the view that when we are logging information about a model then we should log all information which we are going to visualize in a single log file.
   I.e all the various metrics like training loss, validation loss, testing loss etc. should be done in one single file.

Trade off: Because of this restriction designing c++ backend would become quite easy. However, reading from the log file will become slightly tedious. But, in comparison it would simplify the log writing mechanism in a good way. (Especially when writing with the help of callbacks). Also using multiple log files for multiple metrics would increase some processing in reading the logs.

Probable Design:

```
namespace summary
{

  Class fileWriter
  {
      Queue <Summary> q;

      fileWriter(string logdir,
                 int maxQueueSize,
                 )
      {

      }
      void flush();
      {
      }
  };

  Class summaryType
  {
     void scalar(string name,
                 int step,
                 double value,
                 fileWriter &f)

  }

};
```

[redacted]

There are some things