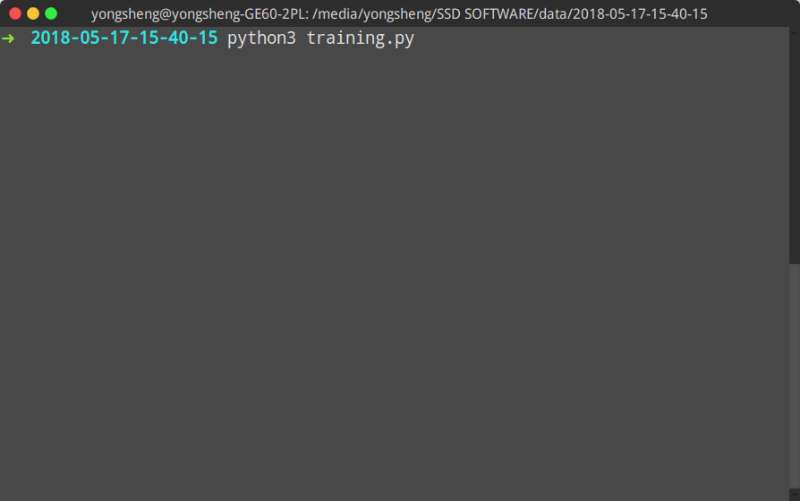
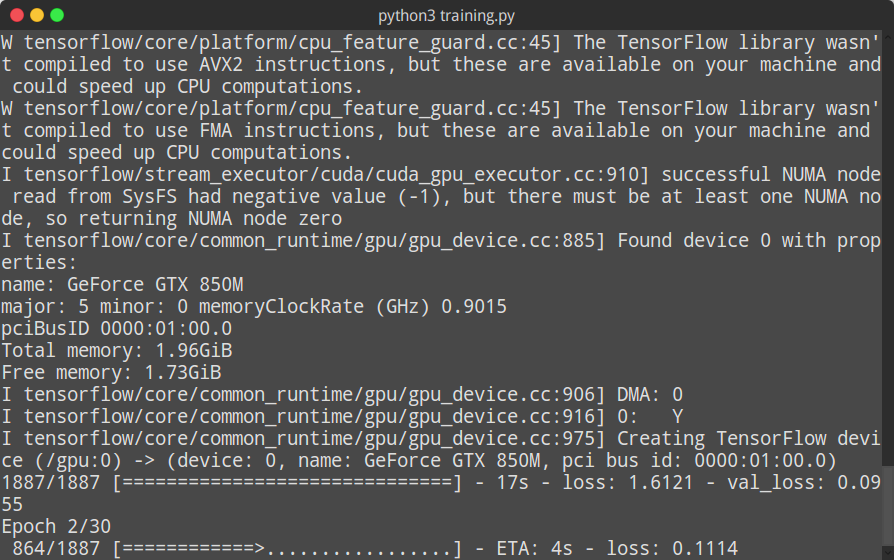
**训练：**

训练的大致流程是读入采集的驾驶图片数据集（大量jpg文件），以及相应的转角数据

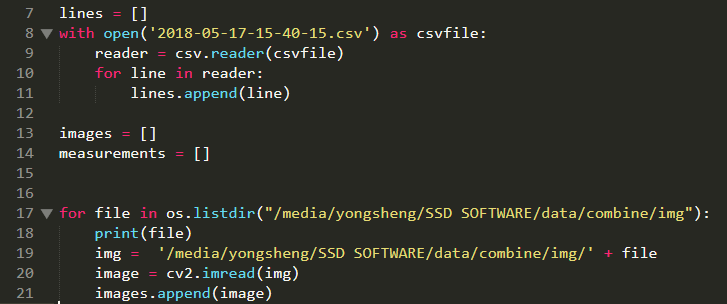
（csv文档）去训练网络。网络将学习如何根据一张驾驶图片去生成正确的转角数据。 训练后生成的网络将以hdf5的形式保存在于Training.py相同目录下。Python环境下需要额外安装名为h5py的库来对hdf5文件进行读写，可以直接通过指令pip3 install h5py进行安装。在确保keras, TensorFlow，hdf5等库安装成功后。若想执行训练任务，可直接在终端下以python3运行Training.py.



若成功运行，开始训练模型，终端将显示如下画面：



Training.py代码中具体定义了训练集数据的路径。



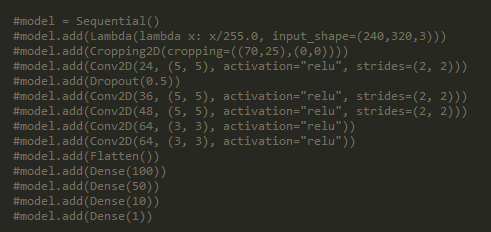
第八行定义了训练转角数据csv文件的路径以及文件名，第十七行定义了训练图片集的路径。可以在代码里面更改路径，从而达到读取任意自定义路径训练集的目的。



第25行处的代码定义了取csv文件的第几列作为转角数据(在从小车导出的csv文档中，第54列问转角数据，故此我们填53)。具体的列数可以根据不同的csv文档进行修改。



Training.py文件中，还具体定义了网络的模型结构。截图为Nvidia的网络结构，大家可以通过改动代码去调整网络结构，model.add()函数可以向神经网络里面添加不同的层。关于不同层网络的参数，具体可参照keras官方文档。Model.compile()函数里可定义loss function和optimizer。 Model.fit()函数里，X\_train为图片集，y\_train为label(转角数据),有需要可以另外调整。此外, epochs定义了训练次数， validation\_split定义了验证集数据的比例。Model.fit()函数也是整个训练流程的开始，整个训练过程就是这一句代码在起作用。在训练过程中，Loss的值会直接输出到终端上，这个值会变得下降收敛。当我们看到这个loss已经很小，而且很难进一步下降时，我们就可以认为这次的训练完成，可以保存网络了。Model.save()函数将会在训练完成之后的神经网络保存下来，默认是储存在跟Training.py相同目录下。注意，如果该目录下已存在同名的网络模型，再一次运行Training.py所生成的新模型，会将已存在的模型覆盖掉。所以，如果不想覆盖掉原来已经存在的网络模型，请务必将旧模型或新模型重命名。上图搭建的网络为nvidia提供的网络，若想替换为comma.ai的网络，可用以下模型进行替换：



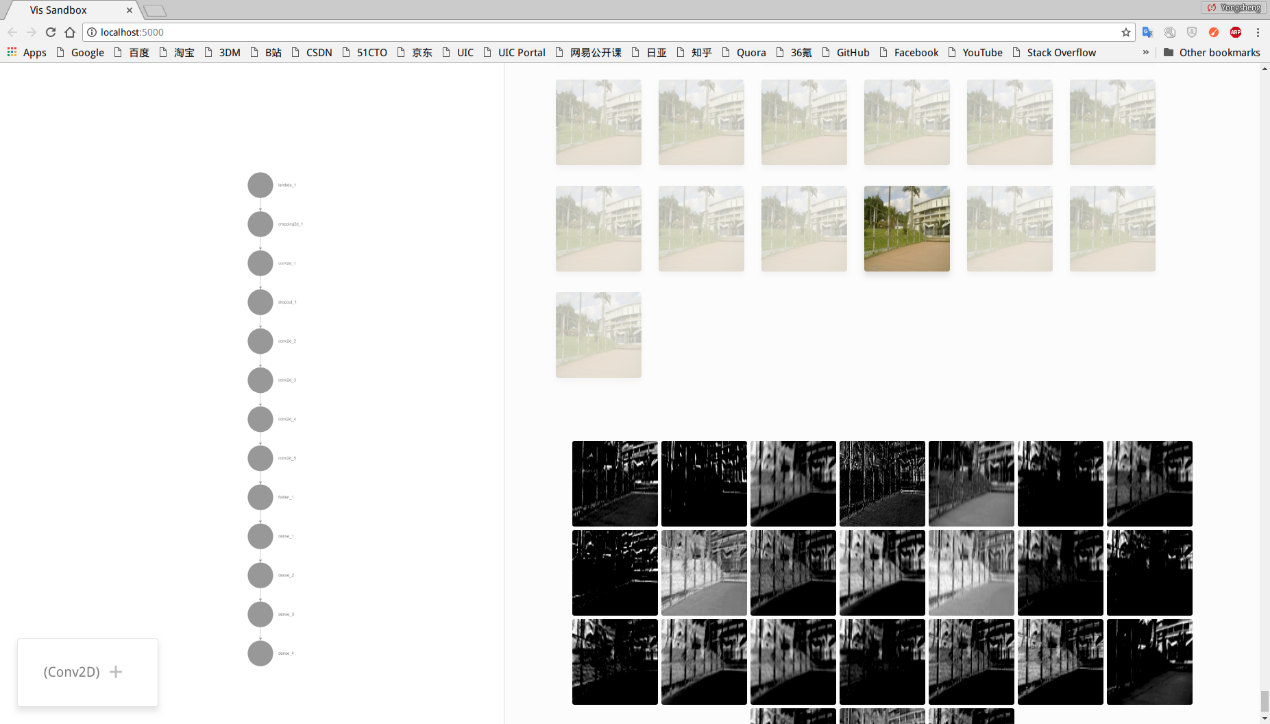
**网络输出可视化：**

若想将网络输出可视化，可在终端下以python3运行visualization.py + model名。在运行该文件之前，需先安装额外的库（quiver\_engine）。可以直接通过命令pip3 install quiver\_engine进行安装。此外，运行之前，还需要在visulization.py的相同目录下创建一个命名为“features”的文件夹，该文件夹将用于临时储存网络输出图像。

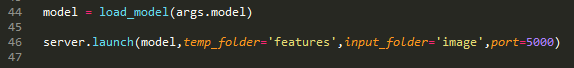


此处的model.h5为之前训练完成得到的网络，在运行visualization.py时，需要在终端上以参数的形式传递给visualization.py。实际执行时，请在visualizatio.py+空格后输入与你的模型一致的名称。

Visualization.py的运行结果如下图所示：



输出结果会在localhost的5000端口呈现，浏览器会自动跳转至该页面。窗口左侧为当前网络的结构，你可以清晰地看到网络有哪些层。窗口右侧上方为原始图片输入，下方为网络输出。当你在右侧上方选中特定的原始输入图片，在左侧分别选中不同的层后，可以看到右侧下方的网络输出会随着你选中的不同层而产生变化。



在代码的46行处，定义了原始输入图片文件夹（image），临时网络输出文件夹（features），以及服务器端口。这些可根据自己的需求进行调整。