Abstract

In this project, the aim was to train and evaluate the MNIST dataset of 60000 labeled handwritten digits. This was an example of a Supervised Machine Learning system. The Anaconda platform is used to setup Keras with a TensorFlow backend, using Jupyter as a frontend to run kernels and notebooks. TensorBoard is used to view the scalars and graphs used to run the job.

An NVIDIA GPU is required so as to use the proprietary CUDA and cuDNN libraries. The relevant hardware used is:

- CPU: Ryzen 3 1200 (OC: +500MHz)
- RAM: Single Channel 8GB 2400MHz
- GPU: GTX 1060 6GB (OC: +150MHz on core, +450MHz on mem)
- Storage: SATA M.2 128GB SSD (boot) with 2 1Tb HDD in RAID 0 (mass storage).

A Sequential model (stack layers sequentially) is used, implementing a Convolutional Neural Network (CNN) – LeNet – architecture, using Python and the Keras deep learning package. This has been compiled using a Categorical Cross Entropy loss function, accuracy metric function, and the Adam optimizer (a method for stochastic optimization).

Basic model training has been done on the sample set, without using any rate annealing methods or sample synthesis (data generation). The model was fitted in 15 epochs to an accuracy of 99.45% and loss of 1.61% in under 95 seconds. Better models having 99.76% accuracy have also been achieved. A personal attempt leads to an accuracy of 98.12% which has not been submitted due to impracticality in application.

Introduction

Machine Learning (ML) is a computer science field focused on implementing algorithms capable of drawing predictive insight from static or dynamic data sources using analytic or probabilistic models and using refinement via training and feedback. It makes use of pattern recognition, artificial intelligence learning methods, and statistical data modeling.

Learning is achieved in two primary ways;

- Supervised machine learning, where the desired outcome is known and an annotated data set or measured values are used to train/fit a model to accurately predict values or labels outside of the training set. This is basically "regression" for values and "classification" for labels.
- Unsupervised learning uses "unlabeled" data and seeks to find *inherent* partitions or clusters of characteristics present in the data. ML draws from the fields of computer science, statistics, probability, applied mathematics, optimization, information theory, graph and network theory, biology, and neuroscience.

In layman terms, it is curve fitting but much more powerful than the basics learned in college mathematics.

Machine Learning has been around for a long time, with the mathematical and computation foundations laid in the 80s. The resurgence of ML is partly due to the fact that technology has been able to reach the required computation demand. Moreover, huge amounts of data have been generated in the past few years due to social networking and Internet usage by a huge number of people. India, USA, EU are the few major areas where ML has been useful to extrapolate the data generated by these countries.

Machine Learning is an area of focus in applied mathematics. ML draws on many mathematical topics, the major ones being, Probability and Statistics, Linear Algebra, Calculus, and Computer Programming. Also useful is Optimization theory, Information Theory, Graph/Network Theory, Numerical Analysis, etc.

Deep learning has revolutionized artificial intelligence by helping us build machines and systems that were only dreamt of in the past. In true essence, Deep Learning is a sub-sect of ML that uses deep, artificial neural networks to tackle the problems of Machine Learning. Deep here signifies the depth of the layers used in the model (as compared to Shallow, where one or two layers are used). A DNN is just a Neural Network with several layers stacked on top of each other (Sequential model) – greater the number of layers, deeper the network.

TensorFlow is an open-sourced library that's available on GitHub. It is one of the more famous libraries when it comes to dealing with Deep Neural Networks. The primary reason behind the popularity of TensorFlow is the sheer ease of building and deploying applications using TensorFlow. The sample projects provided in the GitHub repository are not only powerful but also written in a beginner-friendly way.

TensorFlow excels at numerical computing, which is critical for deep learning. It provides APIs in most major languages and environments needed for deep learning projects: Python, C, C++, Rust, Haskell, Go, Java, Android, iOS, Mac OS, Windows, Linux, and Raspberry Pi.

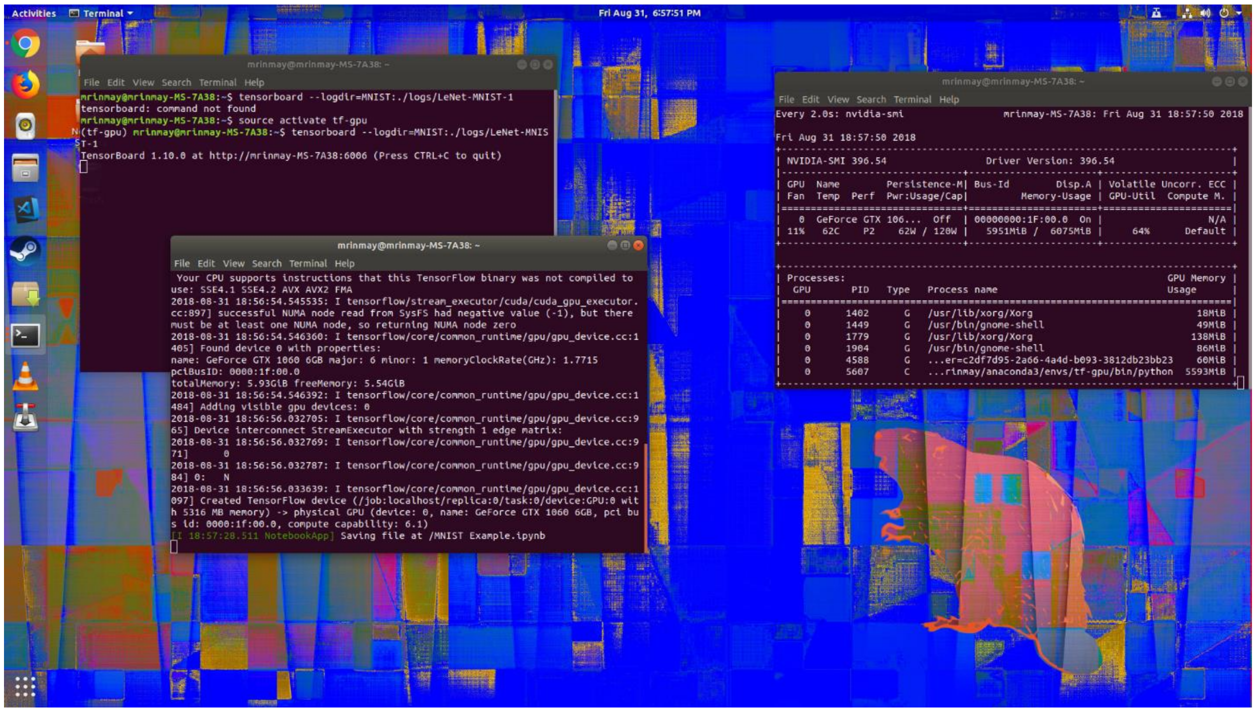
Keras is a high-level library that's built on top of TensorFlow. It provides a SciKit-learn type API (written in Python) for building Neural Networks. It is easier to handle and build ML models than compared to TensorFlow. It runs smoothly on both CPU and GPU, and supports almost all the models of a neural network – fully connected, convolutional, pooling, recurrent, embedding, etc. Furthermore, these models can be combined to build more complex models. Keras, being modular in nature, is incredibly expressive, flexible, and apt for innovative research. It is a completely Python-based framework, which makes it easy to debug and explore.

Statement of Problem

The aim was to get started with ML and Deep Learning in general. The MNIST Dataset is regarded as an entry point into ML. MNIST ("Modified National Institute of Standards and Technology") is the standard “hello world” database of computer vision. Released in 1999, this database of handwritten images has served as the basis for benchmarking classification algorithms. As new ML techniques emerge, MNIST remains a dependable asset, for both researchers and learners. In this report, the aim was to acceptably identify numbers from a database of 60k handwritten images.

Since Windows is not very developer friendly, an Ubuntu installation was preferred (due to high compatibility with CUDA). The “graphics-drivers/ppa” repository was used for driver support. CUDA was not required for the Anaconda platform, but useful for testing with native pip and Docker. A virtual environment was used to install conda (the Anaconda package manager) so as to not interfere with system package requirement. CUDA, cuDNN, TensorFlow, Jupyter, and Keras were installed using the cloud repositories. TensorBoard was used to log data that was run by TensorFlow.

The terminals from all the run has been included:


Activities
Terminal
Fri Aug 31, 6:57:51 PM
mrinmay@mrinmay-MS-7A38: ~
File Edit View Search Terminal Help

```
mrinmay@mrinmay-MS-7A38:~$ tensorboard --logdir=MNIST:./logs/LeNet-MNIST-1
tensorboard: command not found
mrinmay@mrinmay-MS-7A38:~$ source activate tf-gpu
(tf-gpu) mrinmay@mrinmay-MS-7A38:~$ tensorboard --logdir=MNIST:./logs/LeNet-MNIS
T-1
TensorBoard 1.10.0 at http://mrinmay-MS-7A38:6006 (Press CTRL+C to quit)
```

```
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: SSE4.1 SSE4.2 AVX AVX2 FMA
2018-08-31 18:56:54.545535: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:897] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero
2018-08-31 18:56:54.546360: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
405] Found device 0 with properties:
name: GeForce GTX 1060 6GB major: 6 minor: 1 memoryClockRate(GHz): 1.7715
pciBusID: 0000:1f:00.0
totalMemory: 5.93GiB freeMemory: 5.54GiB
2018-08-31 18:56:54.546392: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
484] Adding visible gpu devices: 0
2018-08-31 18:56:56.032705: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
65] Device interconnect StreamExecutor with strength 1 edge matrix:
2018-08-31 18:56:56.032769: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
71]      0
2018-08-31 18:56:56.032787: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
84] 0:   N
2018-08-31 18:56:56.033639: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
097] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 wit
h 5316 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1060 6GB, pci bu
s id: 0000:1f:00.0, compute capability: 6.1)
[I 18:57:28.511 NotebookApp] Saving file at /MNIST Example.ipynb
```

```
Every 2.0s: nvidia-smi                      mrinmay-MS-7A38: Fri Aug 31 18:57:50 2018

Fri Aug 31 18:57:50 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.54                 Driver Version: 396.54                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 106...  Off  | 00000000:1F:00.0  On |                  N/A |
| 11%   62C    P2    62W / 120W |   5951MiB /  6075MiB |     64%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1402      G   /usr/lib/xorg/Xorg                            18MiB |
|    0      1449      G   /usr/bin/gnome-shell                          49MiB |
|    0      1779      G   /usr/lib/xorg/Xorg                           138MiB |
|    0      1904      G   /usr/bin/gnome-shell                          86MiB |
|    0      4588      G   ...er=c2df7d95-2a66-4a4d-b093-3812db23bb23    60MiB |
|    0      5607      C   ...rinmay/anaconda3/envs/tf-gpu/bin/python  5593MiB |
+-----------------------------------------------------------------------------+
```

Figure 1: Terminal outputs

Methodology

The outline of the methodology used is:

- Python environment setup with Anaconda Python
    - Install Anaconda Python
- Create a Python "virtual environment" for TensorFlow using conda
- Install TensorFlow from the Anaconda Cloud Repositories
- Create a Jupyter Notebook Kernel for the TensorFlow Environment
- An Example using Keras with TensorFlow Backend
    - Install Keras
    - Launch a Jupyter Notebook
    - MNIST example
        - Import dependencies
        - Load and process the MNIST data
        - Create the LeNet-5 neural network architecture
        - Compile the model
        - Set log data to feed to TensorBoard for visual analysis
        - Train the model
    - The results
- Look at the job run with TensorBoard

It is recommended that Anaconda is used due to automated environment solving, transaction verification, and transaction execution.

To install it, the following code is used:

```
wget https: //repo.continuum.io/archive/Anaconda3-5.2.0-Linux-x86_64.sh
    expr `sha256sum -q Anaconda3-5.2.0-Linux-
x86_64.sh` = f3527d085d06f35b6aeb96be2a9253ff9ec9ced3dc913c8e27e086329f3db588
bash Anaconda3 - 5.1.0 - Linux - x86_64.sh
```

Restart the terminal to add the environment variables to PATH. Check the python version:

```
python --version
```

And update all base packages:

```
conda update conda

conda update anaconda

conda update python

conda update --all
```

To create a virtual environment, use conda. This keeps the base clean and will give TensorFlow a space for all of its dependencies. It is in general good practice to keep separate environments for projects especially when they have special package dependencies.

From a command line do,

```
conda create --name tf-gpu
```

Now activate the environment,

```
mrinmay@mrinmay-MS-7A38:~$ source activate tf-gpu

(tf-gpu) mrinmay@mrinmay-MS-7A38:~$
```

There is no good reason to do an (old) CUDA install and a pip install when using Anaconda Python. There is an up-to-date official Anaconda package for TensorFlow with GPU acceleration that includes all of the needed CUDA dependencies and it is well optimized for performance.

```
(tf-gpu) mrinmay@mrinmay-MS-7A38:~$ conda install tensorflow-gpu
```

Here's a part of the output listing:

```
The following NEW packages will be INSTALLED:

...

...

    cudatoolkit:       9.0-h13b8566_0

    cudnn:             7.1.2-cuda9.0_0

    cupti:             9.0.176-0

...

    intel-openmp:      2018.0.0-8

    mkl:               2018.0.2-1

    mkl_fft:           1.0.1-py36h3010b51_0

    mkl_random:        1.0.1-py36h629b387_0


    libgcc-ng:         7.2.0-hdf63c60_3

    libgfortran-ng:    7.2.0-hdf63c60_3

    libprotobuf:       3.5.2-h6f1eeef_0

    libstdcxx-ng:      7.2.0-hdf63c60_3

...

    numpy:             1.14.3-py36hcd700cb_1

    numpy-base:        1.14.3-py36h9be14a7_1

...
```

```
    protobuf:          3.5.2-py36hf484d3e_0
    python:            3.6.5-hc3d631a_2
...
    tensorboard:       1.8.0-py36hf484d3e_0
    tensorflow:        1.8.0-hb11d968_0
    tensorflow-base:   1.8.0-py36hc1a7637_0
    tensorflow-gpu:    1.8.0-h7b35bdc_0
```

GPU accelerated TensorFlow 1.8, CUDA 9.0, cuDNN 7.1, Intel's MKL libraries (that are linked into numpy) and TensorBoard have all been installed for the backend support.

Now to setup the frontend, Jupyter will be used.

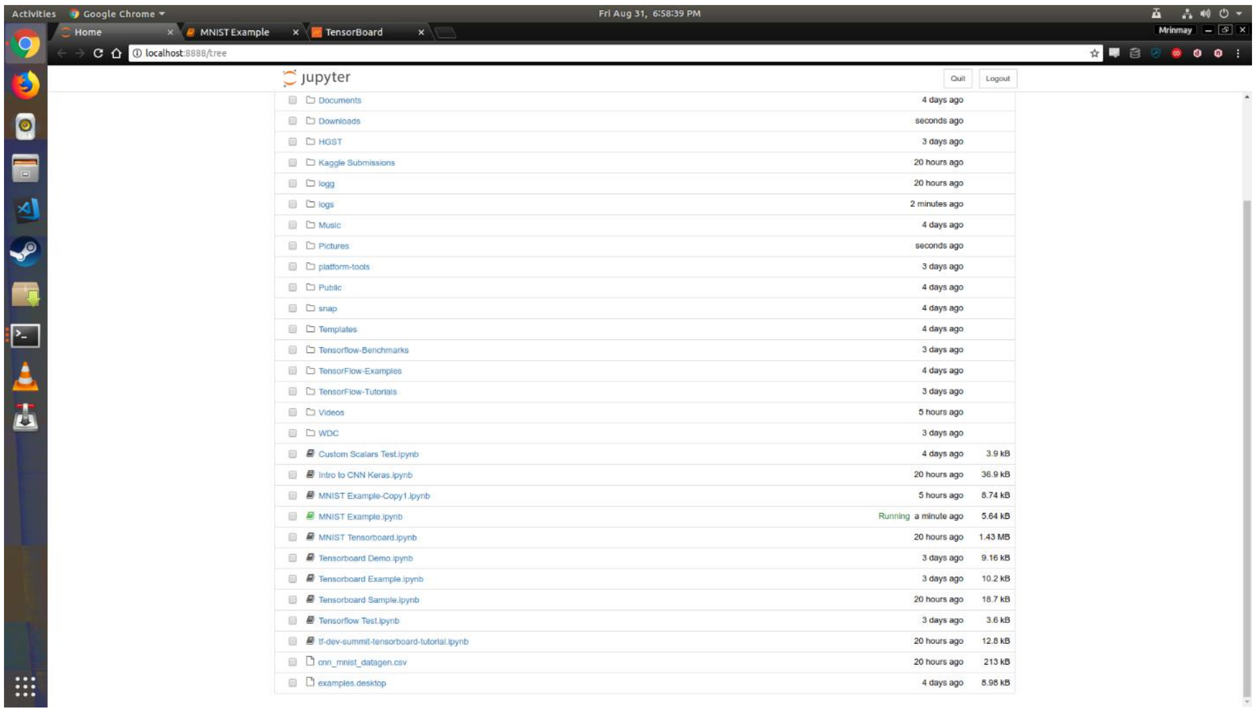
With your tf-gpu environment activated do,

```
(tf-gpu) mrinmay@mrinmay-MS-7A38:~$ conda install ipykernel
```

Creating the Jupyter kernel,

```
(tf-gpu) mrinmay@mrinmay-MS-7A38:~$ python -m ipykernel install --user --name tf-gpu
--display-name "TensorFlow-GPU"
```

This is a sample of the Jupyter frontend running in Chrome.



Documents		4 days ago	
Downloads		seconds ago	
HGST		3 days ago	
Kaggle Submissions		20 hours ago	
logg		20 hours ago	
logs		2 minutes ago	
Music		4 days ago	
Pictures		seconds ago	
platform-tools		3 days ago	
Public		4 days ago	
snap		4 days ago	
Templates		4 days ago	
Tensorflow-Benchmarks		3 days ago	
TensorFlow-Examples		4 days ago	
TensorFlow-Tutorials		3 days ago	
Videos		5 hours ago	
WDC		3 days ago	
Custom Scalars Test.ipynb		4 days ago	3.9 kB
Intro to CNN Keras.ipynb		20 hours ago	36.9 kB
MNIST Example-Copy1.ipynb		5 hours ago	8.74 kB
MNIST Example.ipynb	Running	a minute ago	5.64 kB
MNIST Tensorboard.ipynb		20 hours ago	1.43 MB
Tensorboard Demo.ipynb		3 days ago	9.16 kB
Tensorboard Example.ipynb		3 days ago	10.2 kB
Tensorboard Sample.ipynb		20 hours ago	18.7 kB
Tensorflow Test.ipynb		3 days ago	3.6 kB
tf-dev-summit-tensorboard-tutorial.ipynb		20 hours ago	12.8 kB
cnn_mnist_datagen.csv		20 hours ago	213 kB
examples.desktop		4 days ago	8.98 kB

Figure 2: Jupyter Frontend view

The setup for LeNet-5 using Keras (with TensorFlow backend), along with a Jupyter notebook and "TensorFlow-GPU" kernel is shown here. Model training will be done on the MNIST digits data-set.

With the tf-gpu environment activated do,

```
(tf-gpu) mrinmay@mrinmay-MS-7A38:~$ conda install keras
```

With the tf-gpu environment activated start Jupyter,

```
(tf-gpu) mrinmay@mrinmay-MS-7A38:~$ jupyter notebook
```

From the 'New' drop-down menu select the 'TensorFlow-GPU' kernel that was added (as seen in the image in the last section).

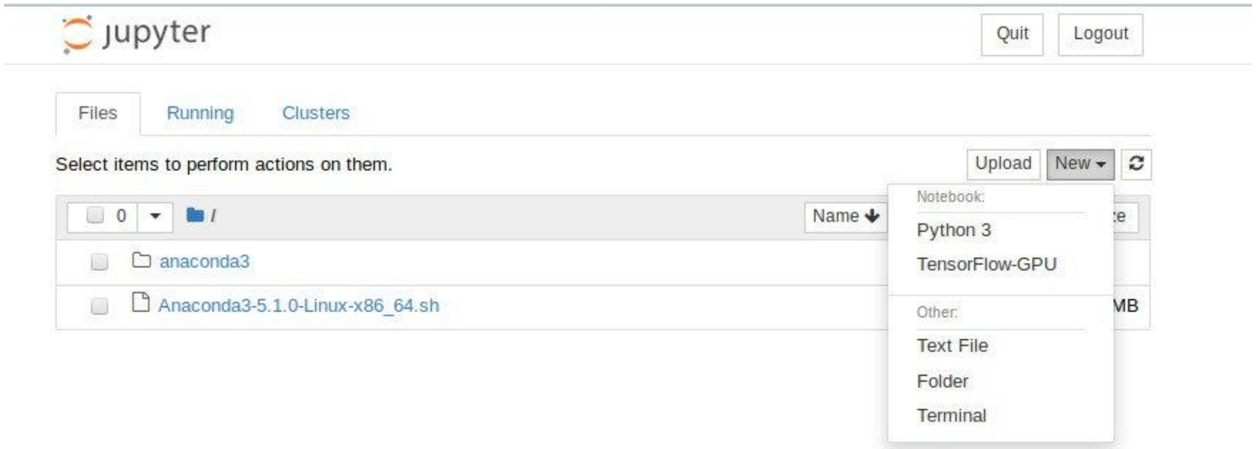


Figure 3: New notebook creation

Fabrication of Model and Prototyping

The MNIST data can be imported using the Kaggle site using the following code,

```
kaggle competitions download -c digit-recognizer
```

or by using the built-in dataset in Keras.

The model to be used will be:

1. Sequential: The `Sequential` model is a linear stack of layers.

The core layers to be used in the model will be:

1. Dense: It is a regular, densely connected Neural Network layer. It implements the operation: `output = activation(dot(input, kernel) + bias)` where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (which is only applicable if `use_bias` is `True`).
2. Dropout: Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.
3. Flatten: Flattens the input. Does not affect the batch size. Must be applied to input before Dense function is used on the input, e.g. can flatten rank-two tensors into row-major arrays.

The pooling layers that will be used are:

1. MaxPooling2D: Max Pooling operation for spatial data i.e. 2D inputs like images.

The convolutional layers that will be used are:

Conv2D: 2D convolutional layer (e.g. spatial convolution over images). This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well. When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the sample axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

Finally a callback function can be passed to `.fit()` method of the Sequential model which will then be called at each stage of the training:

1. TensorBoard: TensorBoard is a visualization tool provided with TensorFlow. This callback writes a log for TensorBoard, which helps to visualize dynamic graphs of training and test metrics, as well as activation histograms for the different layers in the model.

All of the above tasks can be written in Python as:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Flatten,  MaxPooling2D, Conv2D
from keras.callbacks import TensorBoard
```

The MNIST dataset needs to be loaded and processed before any model is created. To do so a parser will be created. The data will be loaded to a matrix having 2 tuples of the form:

- **x_train, x_test**: uint8 array of grayscale image data with shape (num_samples, 28, 28).
- **y_train, y_test**: uint8 array of digit labels (integers in range 0-9) with shape (num_samples,).

`mnist.load_data()` supplies the MNIST digits with structure `(nb_samples, 28, 28)` i.e. with 2 dimensions per example representing a greyscale image 28x28.

The Convolution2D layers in Keras however, are designed to work with 3 dimensions per example. They have 4-dimensional inputs and outputs. This covers colour images `(nb_samples, nb_channels, width, height)`, but more importantly, it covers deeper layers of the network, where each example has become a set of feature maps i.e. `(nb_samples, nb_features, width, height)`.

The greyscale image for MNIST digits input would either need a different CNN layer design (or a param to the layer constructor to accept a different shape), or the design could simply use a standard CNN and you must explicitly express the examples as 1-channel images. The Keras team chose the latter approach, which needs the re-shape.

The next step for the input data is to convert our data type to float32 and normalize our data values to the range [0, 1].

Preprocessing of the class labels is also necessary. The y_train and y_test data are not split into 10 distinct class labels, but rather are represented as a single array with the class values. Here `utils.to_categorical` is used.

The CNN input layer is declared as:

```
model.add(Convolution2D(32, 3, 3, activation='relu', input_shape=(1,28,28)))
```

The input shape parameter should be the shape of 1 sample. In this case, it's the same (1, 28, 28) that corresponds to the (depth, width, height) of each digit image. the first 3 parameters correspond to the number of convolution filters to use, the number of rows in each convolution kernel, and the number of columns in each convolution kernel, respectively.

The dropout layer is used as a method for regularizing the model in order to prevent overfitting (refers to a model that models the training data too well). Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

MaxPooling2D is a way to reduce the number of parameters in the model by sliding a 2x2 pooling filter across the previous layer and taking the max of the 4 values in the 2x2 filter.

So far, for model parameters, two Convolution layers have been added. To complete the model architecture, add a fully connected layer and then the output layer:

```
model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(n_classes, activation='softmax'))
```

For Dense layers, the first parameter is the output size of the layer. Keras automatically handles the connections between layers. The final layer has an output size of 10, corresponding to the 10 classes of digits. The weights from the Convolution layers must be flattened (made 1-dimensional) before passing them to the fully connected Dense layer.

Next, compilation and training of the model is needed. When the model is compiled, declare the loss function and the optimizer.

```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Set log data to feed to TensorBoard for visual analysis:

```
tensor_board = TensorBoard('./logs/LeNet-MNIST-1')
```

To fit the model, declare the batch size and number of epochs to train for, then pass in the training data.

```
model.fit(X_train, y_train, batch_size=128, epochs=15, verbose=1,
          validation_data=(X_test,y_test), callbacks=[tensor_board])
```

Evaluation of the model on test data can be done now:

```
score = model.evaluate(X_test, Y_test, verbose=1)
```

Although this step has been mentioned, it was not entered in the final code revision as the model fitting had a verbosity level that rendered this step ineffectual.

Results

The results from the MNIST training has been included as a set of images. This includes the scalars and graphs generated by TensorBoard and an IPython Notebook HTML.

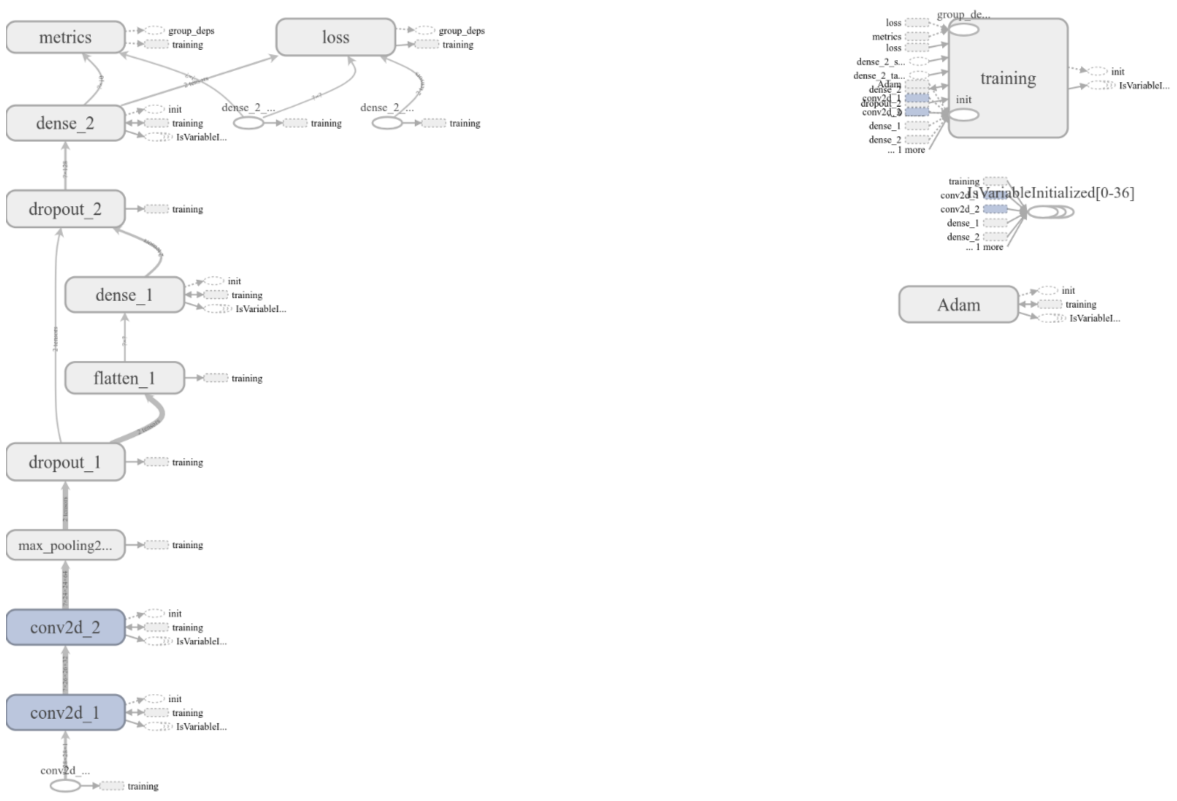
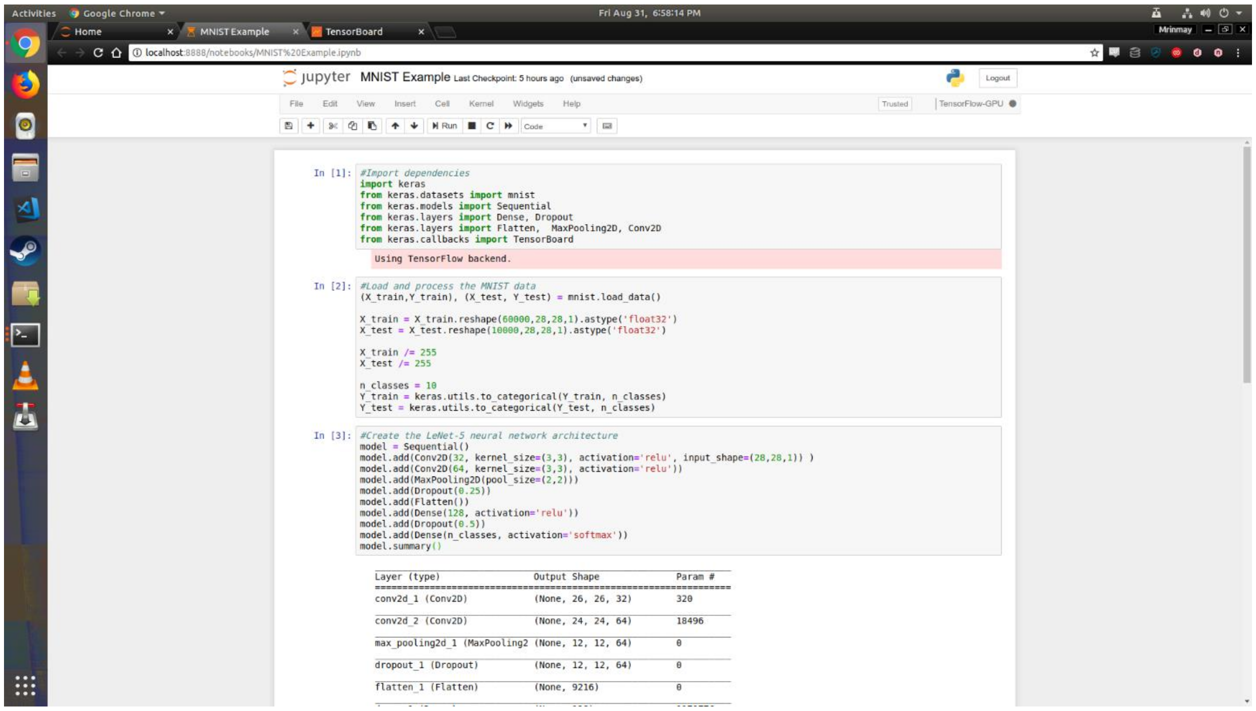
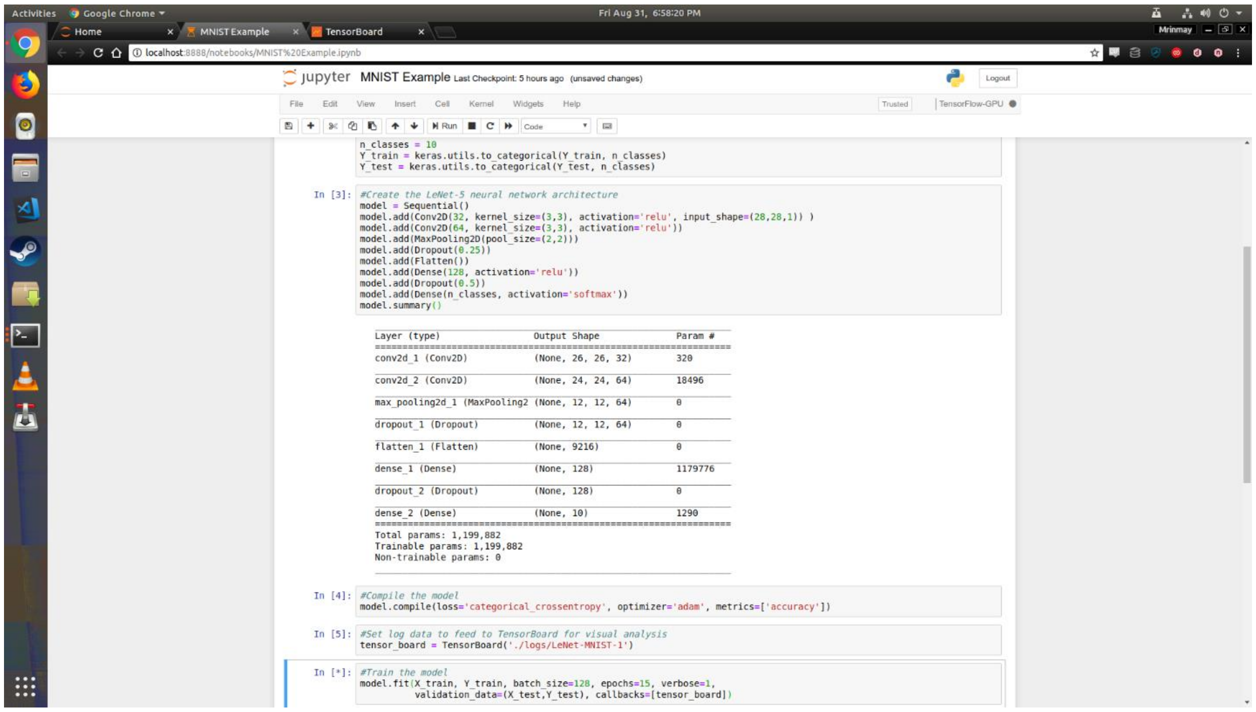


Figure 4: View of the graph generated by the model in TensorBoard



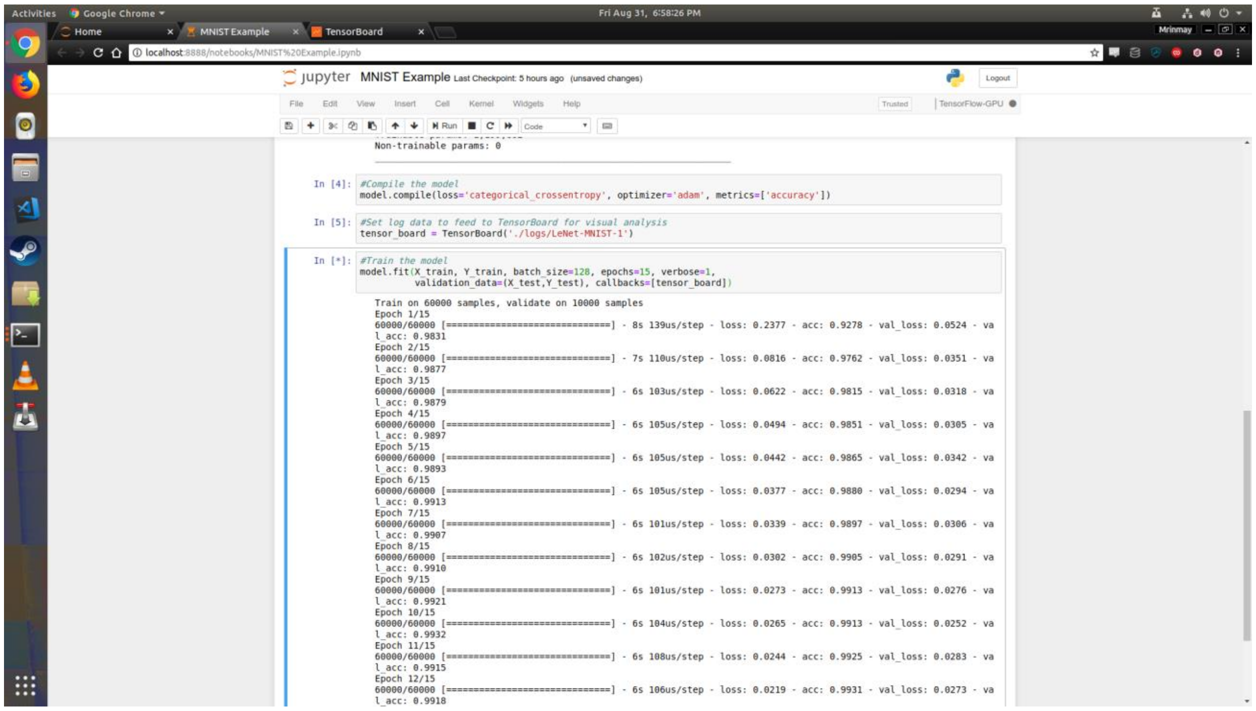
```
In [1]: #Import dependencies
        import keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers import Dense, Dropout
        from keras.layers import Flatten,  MaxPooling2D, Conv2D
        from keras.callbacks import TensorBoard

          Using TensorFlow backend.

In [2]: #Load and process the MNIST data
        (X_train,Y_train), (X_test, Y_test) = mnist.load_data()

        X_train = X_train.reshape(60000,28,28,1).astype('float32')
        X_test = X_test.reshape(10000,28,28,1).astype('float32')

        X_train /= 255
        X_test /= 255

        n_classes = 10
        Y_train = keras.utils.to_categorical(Y_train, n_classes)
        Y_test = keras.utils.to_categorical(Y_test, n_classes)

In [3]: #Create the LeNet-5 neural network architecture
        model = Sequential()
        model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)) )
        model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(n_classes, activation='softmax'))
        model.summary()

          Layer (type)                 Output Shape              Param #
          =================================================================
          conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
          _________________________________________________________________
          conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
          _________________________________________________________________
          max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
          _________________________________________________________________
          dropout_1 (Dropout)          (None, 12, 12, 64)        0
          _________________________________________________________________
          flatten_1 (Flatten)          (None, 9216)              0
          _________________________________________________________________
```

Figure 5.1: IPython File Screenshot



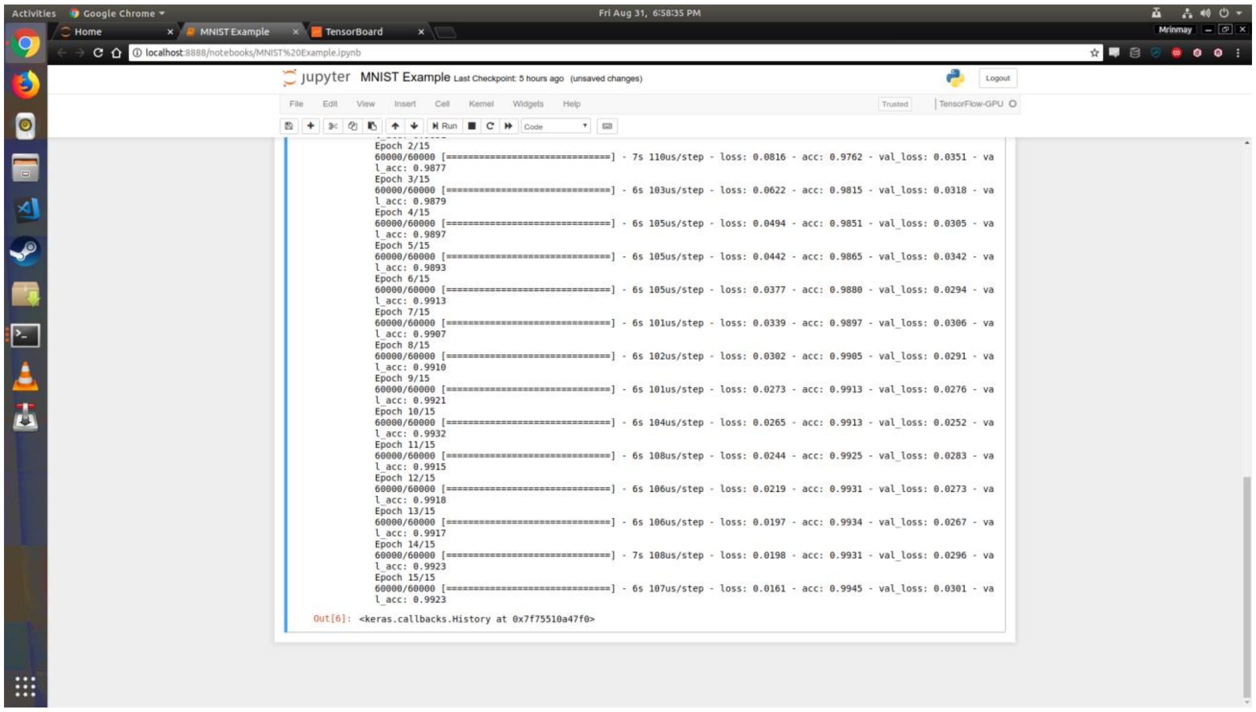
```
        n_classes = 10
        Y_train = keras.utils.to_categorical(Y_train, n_classes)
        Y_test = keras.utils.to_categorical(Y_test, n_classes)

In [3]: #Create the LeNet-5 neural network architecture
        model = Sequential()
        model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)) )
        model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(n_classes, activation='softmax'))
        model.summary()

          Layer (type)                 Output Shape              Param #
          =================================================================
          conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
          _________________________________________________________________
          conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
          _________________________________________________________________
          max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
          _________________________________________________________________
          dropout_1 (Dropout)          (None, 12, 12, 64)        0
          _________________________________________________________________
          flatten_1 (Flatten)          (None, 9216)              0
          _________________________________________________________________
          dense_1 (Dense)              (None, 128)               1179776
          _________________________________________________________________
          dropout_2 (Dropout)          (None, 128)               0
          _________________________________________________________________
          dense_2 (Dense)              (None, 10)                1290
          =================================================================
          Total params: 1,199,882
          Trainable params: 1,199,882
          Non-trainable params: 0
          _________________________________________________________________

In [4]: #Compile the model
        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

In [5]: #Set log data to feed to TensorBoard for visual analysis
        tensor_board = TensorBoard('./logs/LeNet-MNIST-1')

In [*]: #Train the model
        model.fit(X_train, Y_train, batch_size=128, epochs=15, verbose=1,
                  validation_data=(X_test,Y_test), callbacks=[tensor_board])
```

Figure 5.2: IPython File Screenshot



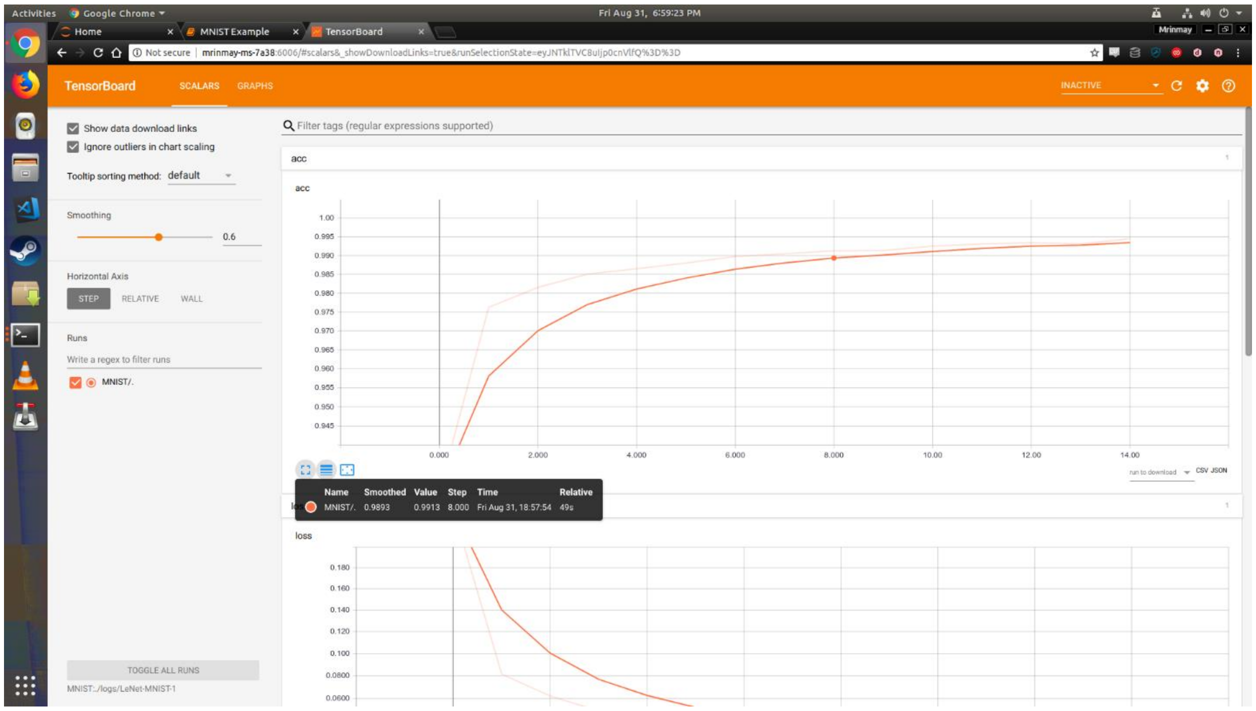
```
Non-trainable params: 0

In [4]: #Compile the model
        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

In [5]: #Set log data to feed to TensorBoard for visual analysis
        tensor_board = TensorBoard('./logs/LeNet-MNIST-1')

In [*]: #Train the model
        model.fit(X_train, Y_train, batch_size=128, epochs=15, verbose=1,
                  validation_data=(X_test,Y_test), callbacks=[tensor_board])

          Train on 60000 samples, validate on 10000 samples
          Epoch 1/15
          60000/60000 [==============================] - 8s 139us/step - loss: 0.2377 - acc: 0.9278 - val_loss: 0.0524 - va
          l_acc: 0.9831
          Epoch 2/15
          60000/60000 [==============================] - 7s 110us/step - loss: 0.0816 - acc: 0.9762 - val_loss: 0.0351 - va
          l_acc: 0.9877
          Epoch 3/15
          60000/60000 [==============================] - 6s 103us/step - loss: 0.0622 - acc: 0.9815 - val_loss: 0.0318 - va
          l_acc: 0.9879
          Epoch 4/15
          60000/60000 [==============================] - 6s 105us/step - loss: 0.0494 - acc: 0.9851 - val_loss: 0.0305 - va
          l_acc: 0.9897
          Epoch 5/15
          60000/60000 [==============================] - 6s 105us/step - loss: 0.0442 - acc: 0.9865 - val_loss: 0.0342 - va
          l_acc: 0.9893
          Epoch 6/15
          60000/60000 [==============================] - 6s 105us/step - loss: 0.0377 - acc: 0.9880 - val_loss: 0.0294 - va
          l_acc: 0.9913
          Epoch 7/15
          60000/60000 [==============================] - 6s 101us/step - loss: 0.0339 - acc: 0.9897 - val_loss: 0.0306 - va
          l_acc: 0.9907
          Epoch 8/15
          60000/60000 [==============================] - 6s 102us/step - loss: 0.0302 - acc: 0.9905 - val_loss: 0.0291 - va
          l_acc: 0.9910
          Epoch 9/15
          60000/60000 [==============================] - 6s 101us/step - loss: 0.0273 - acc: 0.9913 - val_loss: 0.0276 - va
          l_acc: 0.9921
          Epoch 10/15
          60000/60000 [==============================] - 6s 104us/step - loss: 0.0265 - acc: 0.9913 - val_loss: 0.0252 - va
          l_acc: 0.9932
          Epoch 11/15
          60000/60000 [==============================] - 6s 108us/step - loss: 0.0244 - acc: 0.9925 - val_loss: 0.0283 - va
          l_acc: 0.9915
          Epoch 12/15
          60000/60000 [==============================] - 6s 106us/step - loss: 0.0219 - acc: 0.9931 - val_loss: 0.0273 - va
          l_acc: 0.9918
```

Figure 5.3: IPython File Screenshot



```
          Epoch 2/15
          60000/60000 [==============================] - 7s 110us/step - loss: 0.0816 - acc: 0.9762 - val_loss: 0.0351 - va
          l_acc: 0.9877
          Epoch 3/15
          60000/60000 [==============================] - 6s 103us/step - loss: 0.0622 - acc: 0.9815 - val_loss: 0.0318 - va
          l_acc: 0.9879
          Epoch 4/15
          60000/60000 [==============================] - 6s 105us/step - loss: 0.0494 - acc: 0.9851 - val_loss: 0.0305 - va
          l_acc: 0.9897
          Epoch 5/15
          60000/60000 [==============================] - 6s 105us/step - loss: 0.0442 - acc: 0.9865 - val_loss: 0.0342 - va
          l_acc: 0.9893
          Epoch 6/15
          60000/60000 [==============================] - 6s 105us/step - loss: 0.0377 - acc: 0.9880 - val_loss: 0.0294 - va
          l_acc: 0.9913
          Epoch 7/15
          60000/60000 [==============================] - 6s 101us/step - loss: 0.0339 - acc: 0.9897 - val_loss: 0.0306 - va
          l_acc: 0.9907
          Epoch 8/15
          60000/60000 [==============================] - 6s 102us/step - loss: 0.0302 - acc: 0.9905 - val_loss: 0.0291 - va
          l_acc: 0.9910
          Epoch 9/15
          60000/60000 [==============================] - 6s 101us/step - loss: 0.0273 - acc: 0.9913 - val_loss: 0.0276 - va
          l_acc: 0.9921
          Epoch 10/15
          60000/60000 [==============================] - 6s 104us/step - loss: 0.0265 - acc: 0.9913 - val_loss: 0.0252 - va
          l_acc: 0.9932
          Epoch 11/15
          60000/60000 [==============================] - 6s 108us/step - loss: 0.0244 - acc: 0.9925 - val_loss: 0.0283 - va
          l_acc: 0.9915
          Epoch 12/15
          60000/60000 [==============================] - 6s 106us/step - loss: 0.0219 - acc: 0.9931 - val_loss: 0.0273 - va
          l_acc: 0.9918
          Epoch 13/15
          60000/60000 [==============================] - 6s 106us/step - loss: 0.0197 - acc: 0.9934 - val_loss: 0.0267 - va
          l_acc: 0.9917
          Epoch 14/15
          60000/60000 [==============================] - 7s 108us/step - loss: 0.0198 - acc: 0.9931 - val_loss: 0.0296 - va
          l_acc: 0.9923
          Epoch 15/15
          60000/60000 [==============================] - 6s 107us/step - loss: 0.0161 - acc: 0.9945 - val_loss: 0.0301 - va
          l_acc: 0.9923

Out[6]: <keras.callbacks.History at 0x7f75510a47f0>
```

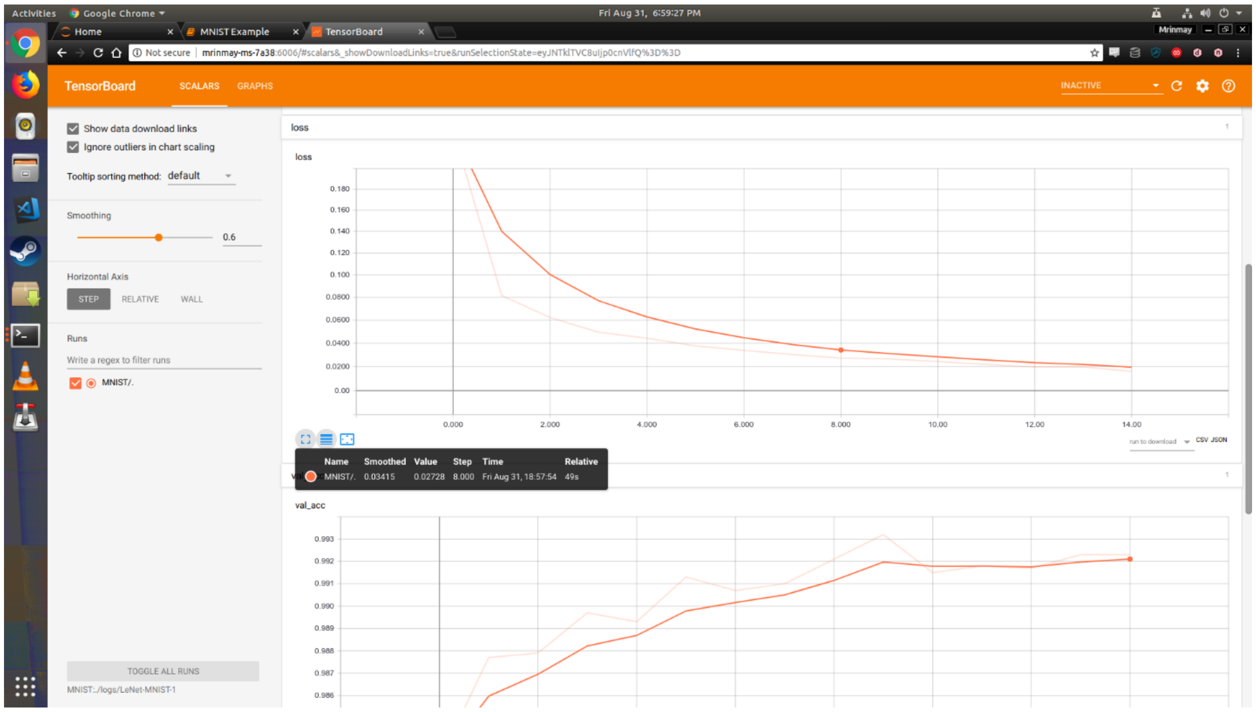
Figure 5.4: IPython File Screenshot



Show data download links ✓
Ignore outliers in chart scaling ✓
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
Runs: MNIST/.
MNIST:./logs/LeNet-MNIST-1

	Name	Smoothed	Value	Step	Time	Relative
●	MNIST/.	0.9893	0.9913	8.000	Fri Aug 31, 18:57:54	49s

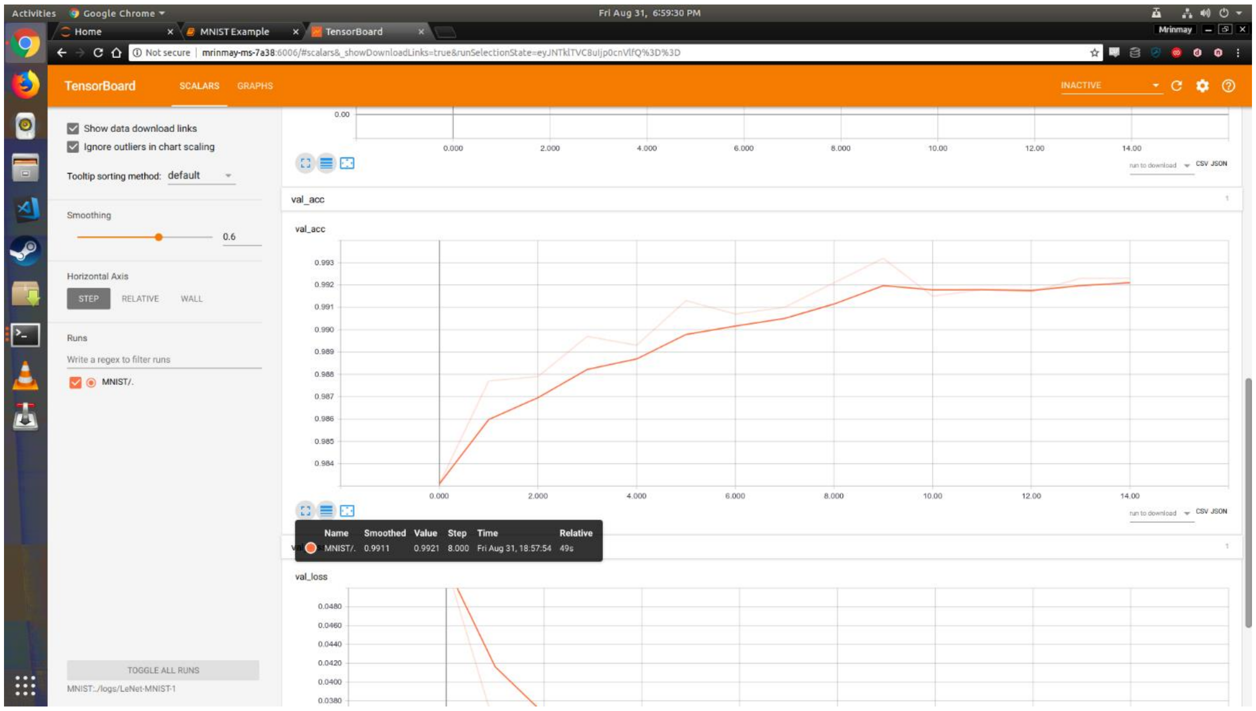
Figure 6.1: Scalars generated by TensorBoard



Show data download links ✓
Ignore outliers in chart scaling ✓
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
Runs: MNIST/.
MNIST:./logs/LeNet-MNIST-1

	Name	Smoothed	Value	Step	Time	Relative
●	MNIST/.	0.03415	0.02728	8.000	Fri Aug 31, 18:57:54	49s

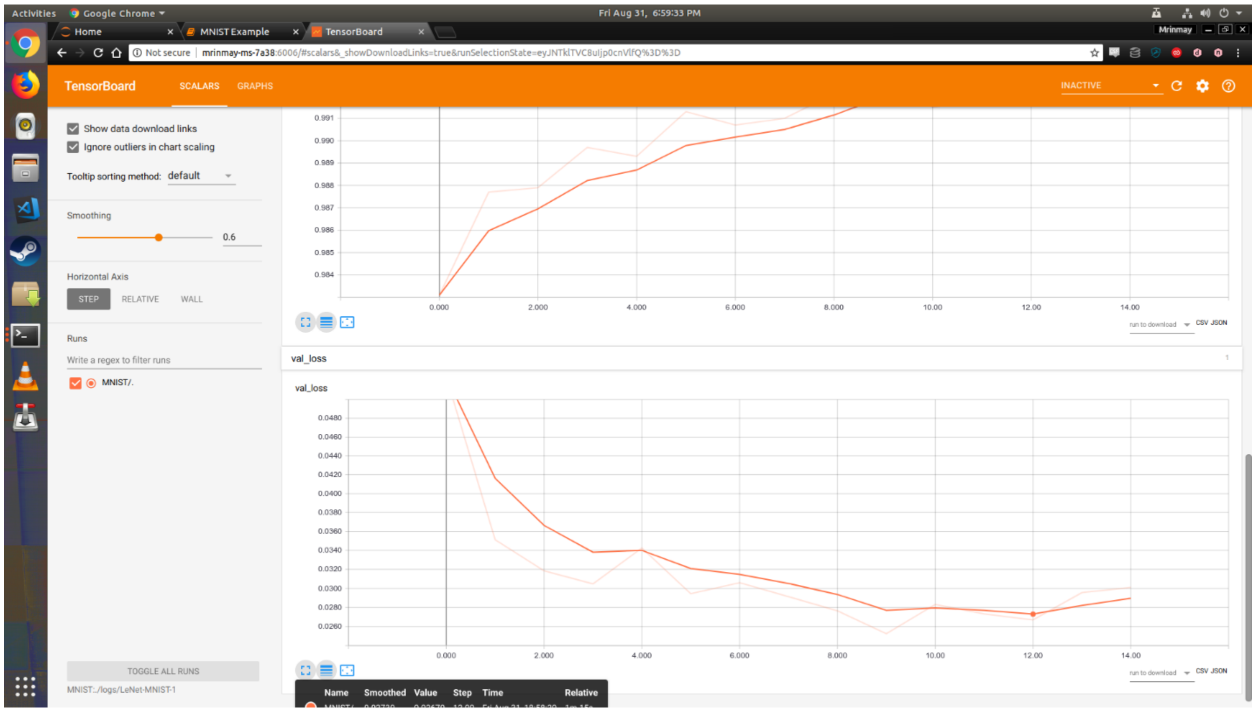
Figure 6.2: Scalars generated by TensorBoard



Show data download links ✓
Ignore outliers in chart scaling ✓
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
Runs: MNIST/. ✓
MNIST:./logs/LeNet-MNIST-1

```
Name      Smoothed  Value   Step   Time                   Relative
MNIST/.   0.9911    0.9921  8.000  Fri Aug 31, 18:57:54   49s
```

Figure 6.3: Scalars generated by TensorBoard



Show data download links ✓
Ignore outliers in chart scaling ✓
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
Runs: MNIST/. ✓
MNIST:./logs/LeNet-MNIST-1

Figure 6.4: Scalars generated by TensorBoard

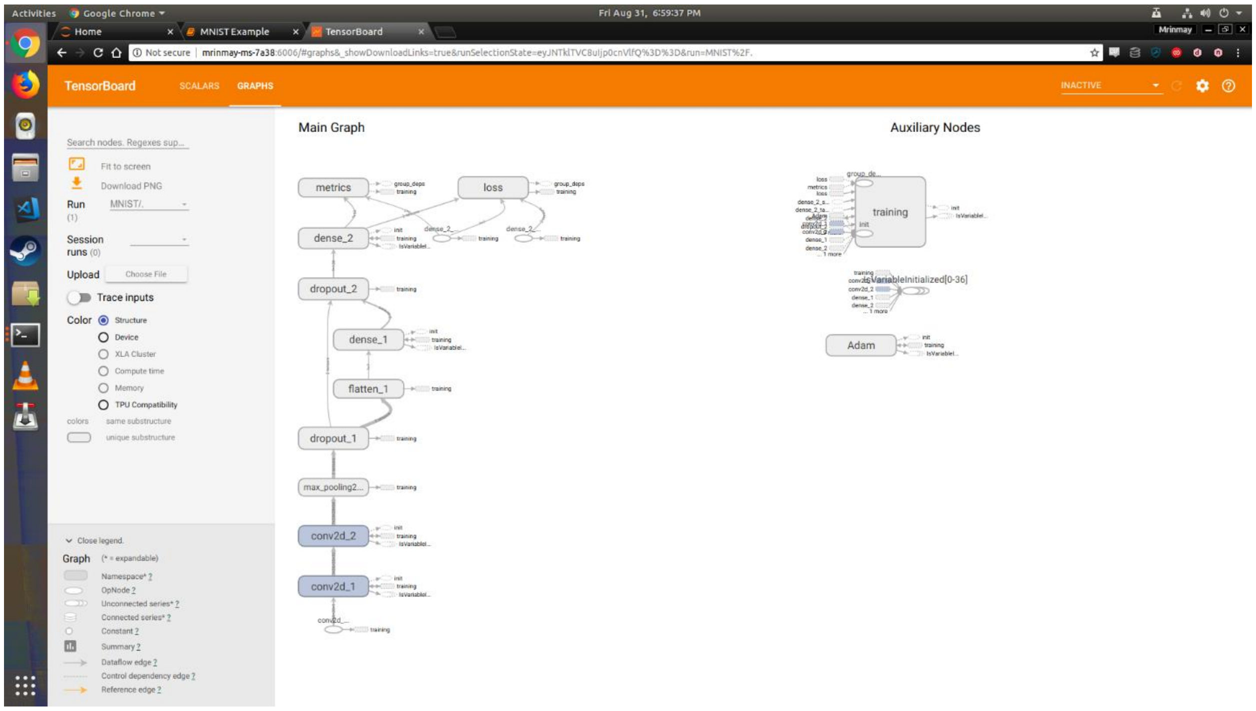


Figure 7.1: Graphs generated by TensorBoard

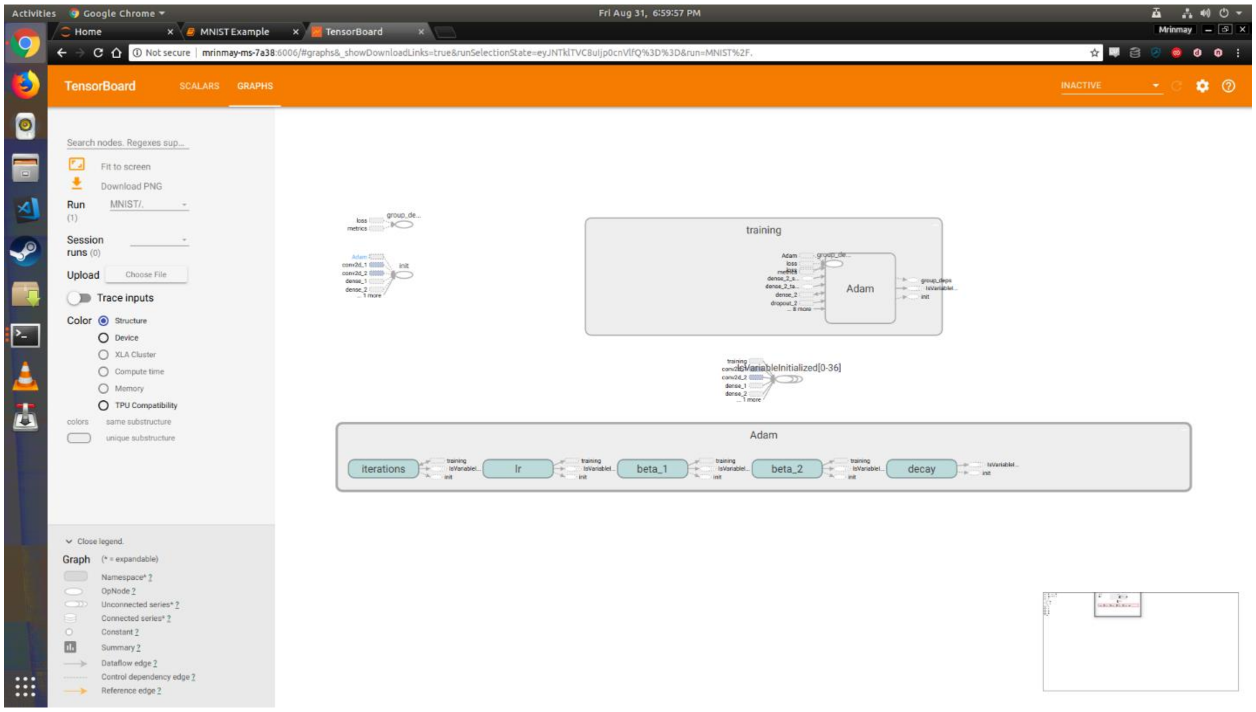


Figure 7.2: Graphs generated by TensorBoard

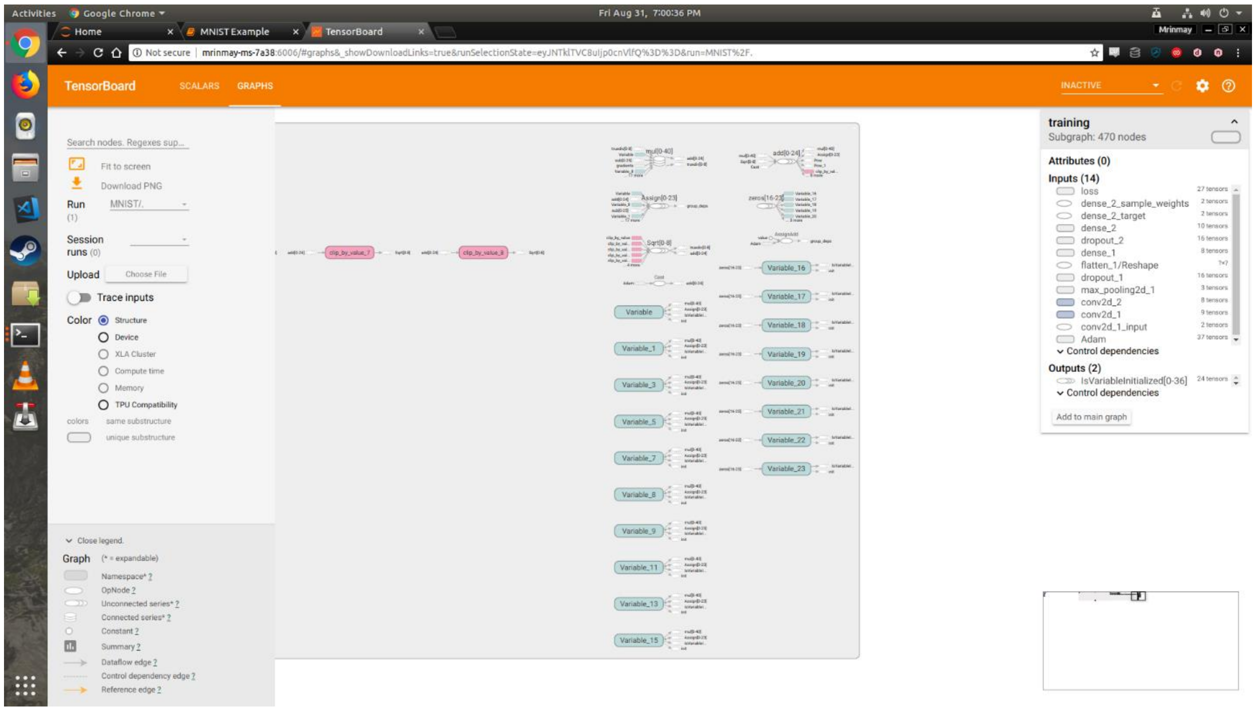


Figure 7.3: Graphs generated by TensorBoard

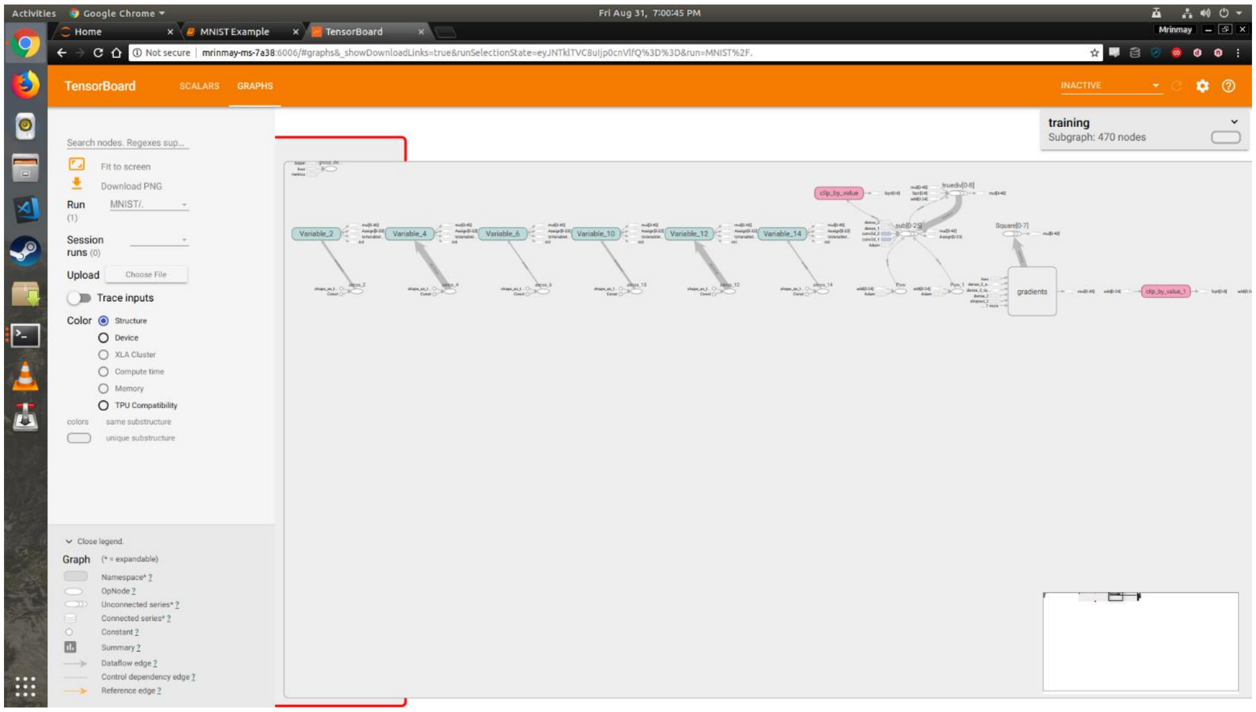


Figure 7.4: Graphs generated by TensorBoard

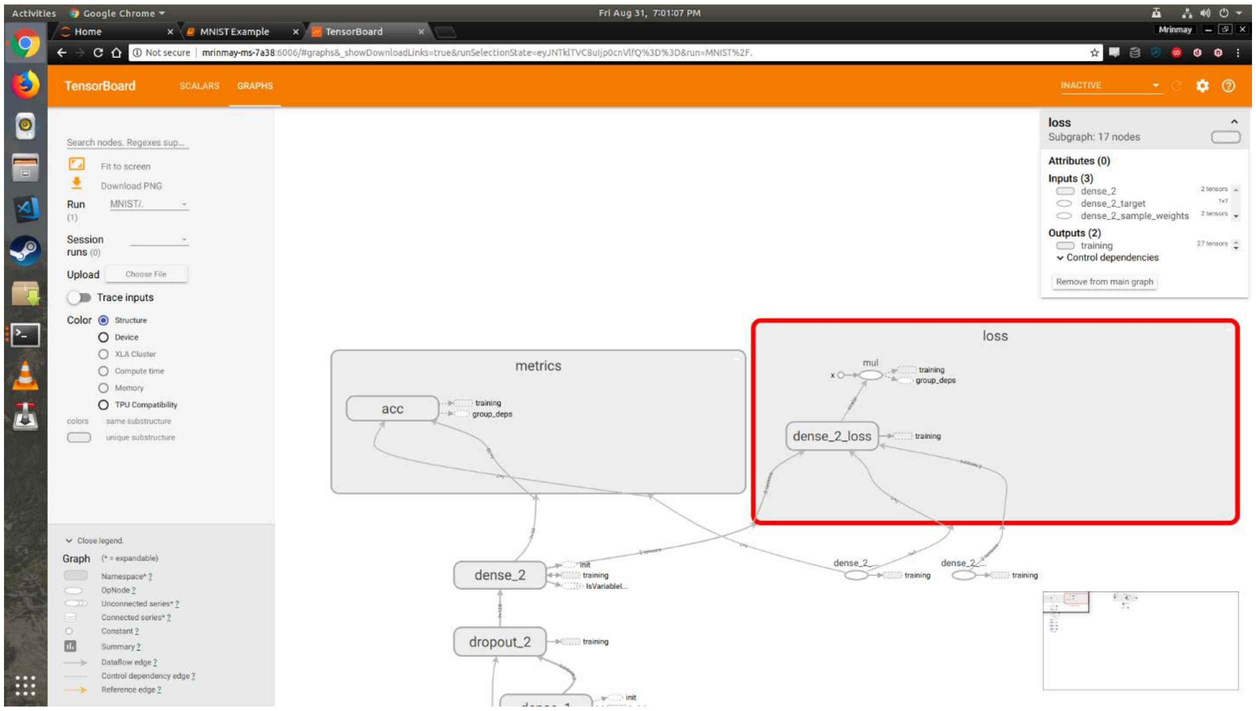


Figure 7.5: Graphs generated by TensorBoard

Here is the content of “MNIST Example.ipynb” IPython Notebook File.

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "Using TensorFlow backend.\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
"_________________________________________________________________\n",
      "Layer (type)                 Output Shape              Param #   
\n",
"=================================================================\n",
      "conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
\n",
"_________________________________________________________________\n",
      "conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496     
\n",
"_________________________________________________________________\n",
      "max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0         
\n",
"_________________________________________________________________\n",
      "dropout_1 (Dropout)          (None, 12, 12, 64)        0         
\n",
"_________________________________________________________________\n",
      "flatten_1 (Flatten)          (None, 9216)              0         
\n",
"_________________________________________________________________\n",
      "dense_1 (Dense)              (None, 128)               1179776   
\n",
"_________________________________________________________________\n",
      "dropout_2 (Dropout)          (None, 128)               0         
\n",
"_________________________________________________________________\n",
      "dense_2 (Dense)              (None, 10)                1290
\n",

"=================================================================\n",
      "Total params: 1,199,882\n",
      "Trainable params: 1,199,882\n",
      "Non-trainable params: 0\n",

"_________________________________________________________________\n",
      "Train on 60000 samples, validate on 10000 samples\n",
      "Epoch 1/15\n",
      "60000/60000 [==============================] - 7s 120us/step -
loss: 0.2297 - acc: 0.9300 - val_loss: 0.0500 - val_acc: 0.9841\n",
      "Epoch 2/15\n",
      "60000/60000 [==============================] - 6s 103us/step -
loss: 0.0819 - acc: 0.9754 - val_loss: 0.0384 - val_acc: 0.9881\n",
      "Epoch 3/15\n",
      "60000/60000 [==============================] - 6s 105us/step -
loss: 0.0593 - acc: 0.9817 - val_loss: 0.0337 - val_acc: 0.9893\n",
      "Epoch 4/15\n",
      "60000/60000 [==============================] - 6s 104us/step -
loss: 0.0488 - acc: 0.9851 - val_loss: 0.0317 - val_acc: 0.9895\n",
      "Epoch 5/15\n",
      "60000/60000 [==============================] - 6s 104us/step -
loss: 0.0423 - acc: 0.9866 - val_loss: 0.0304 - val_acc: 0.9905\n",
      "Epoch 6/15\n",
      "60000/60000 [==============================] - 6s 102us/step -
loss: 0.0350 - acc: 0.9890 - val_loss: 0.0282 - val_acc: 0.9914\n",
      "Epoch 7/15\n",
      "60000/60000 [==============================] - 6s 102us/step -
loss: 0.0336 - acc: 0.9891 - val_loss: 0.0320 - val_acc: 0.9896\n",
      "Epoch 8/15\n",
      "60000/60000 [==============================] - 6s 100us/step -
loss: 0.0290 - acc: 0.9902 - val_loss: 0.0332 - val_acc: 0.9908\n",
      "Epoch 9/15\n",
      "60000/60000 [==============================] - 6s 101us/step -
loss: 0.0250 - acc: 0.9915 - val_loss: 0.0296 - val_acc: 0.9916\n",
      "Epoch 10/15\n",
      "60000/60000 [==============================] - 6s 101us/step -
loss: 0.0230 - acc: 0.9924 - val_loss: 0.0297 - val_acc: 0.9920\n",
      "Epoch 11/15\n",
      "60000/60000 [==============================] - 6s 100us/step -
loss: 0.0216 - acc: 0.9928 - val_loss: 0.0342 - val_acc: 0.9904\n",
      "Epoch 12/15\n",
      "60000/60000 [==============================] - 6s 102us/step -
loss: 0.0213 - acc: 0.9928 - val_loss: 0.0275 - val_acc: 0.9927\n",
      "Epoch 13/15\n",
      "60000/60000 [==============================] - 6s 104us/step -
loss: 0.0185 - acc: 0.9942 - val_loss: 0.0350 - val_acc: 0.9916\n",
      "Epoch 14/15\n",
      "60000/60000 [==============================] - 6s 107us/step -
loss: 0.0174 - acc: 0.9939 - val_loss: 0.0311 - val_acc: 0.9926\n",
      "Epoch 15/15\n",
      "60000/60000 [==============================] - 6s 107us/step -
loss: 0.0159 - acc: 0.9945 - val_loss: 0.0326 - val_acc: 0.9924\n"
     ]
    },
    {
     "data": {
      "text/plain": [
       "<keras.callbacks.History at 0x7f7387cf4780>"
      ]
     },
     "execution_count": 1,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "#Import dependencies\n",
    "import keras\n",
    "from keras.datasets import mnist\n",
    "from keras.models import Sequential\n",
    "from keras.layers import Dense, Dropout\n",
    "from keras.layers import Flatten,  MaxPooling2D, Conv2D\n",
    "from keras.callbacks import TensorBoard\n",
    "\n",
    "#Load and process the MNIST data\n",
    "(X_train,Y_train), (X_test, Y_test) = mnist.load_data()\n",
    "\n",
    "X_train = X_train.reshape(60000,28,28,1).astype('float32')\n",
    "X_test = X_test.reshape(10000,28,28,1).astype('float32')\n",
    "\n",
    "X_train /= 255\n",
    "X_test /= 255\n",
    "\n",
    "n_classes = 10\n",
    "Y_train = keras.utils.to_categorical(Y_train, n_classes)\n",
    "Y_test = keras.utils.to_categorical(Y_test, n_classes)\n",
    "\n",
    "#Create the LeNet-5 neural network architecture\n",
    "model = Sequential()\n",
    "model.add(Conv2D(32, kernel_size=(3,3), activation='relu',
input_shape=(28,28,1)) )\n",
    "model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))\n",
    "model.add(MaxPooling2D(pool_size=(2,2)))\n",
    "model.add(Dropout(0.25))\n",
    "model.add(Flatten())          \n",
    "model.add(Dense(128, activation='relu'))\n",
    "model.add(Dropout(0.5))\n",
    "model.add(Dense(n_classes, activation='softmax'))\n",
    "model.summary()\n",
    "\n",
    "#Compile the model\n",
    "model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])\n",
    "\n",
    "#Set log data to feed to TensorBoard for visual analysis\n",
    "tensor_board = TensorBoard('./logs/LeNet-MNIST-1')\n",
    "\n",
    "#Train the model\n",
    "model.fit(X_train, Y_train, batch_size=128, epochs=15, verbose=1,\n",
    "          validation_data=(X_test,Y_test), callbacks=[tensor_board])"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "TensorFlow-GPU",
   "language": "python",
   "name": "tf-gpu"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.6.6"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```

Conclusion

For better accuracy, Yassine Ghouzam (http://www.dsimb.inserm.fr/~ghouzam/) over at Kaggle provides a way to achieve 99.7% accuracy. It uses a 5-layer Sequential CNN rather than 2. It also uses the RMSProp optimizer with the following values:

```
optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
```

A learning rate annealer was implemented in callbacks to reduce the learning rate by half if accuracy does not improve after three epochs (in order to make the optimizer converge faster and closest to the global minimum of the loss function).

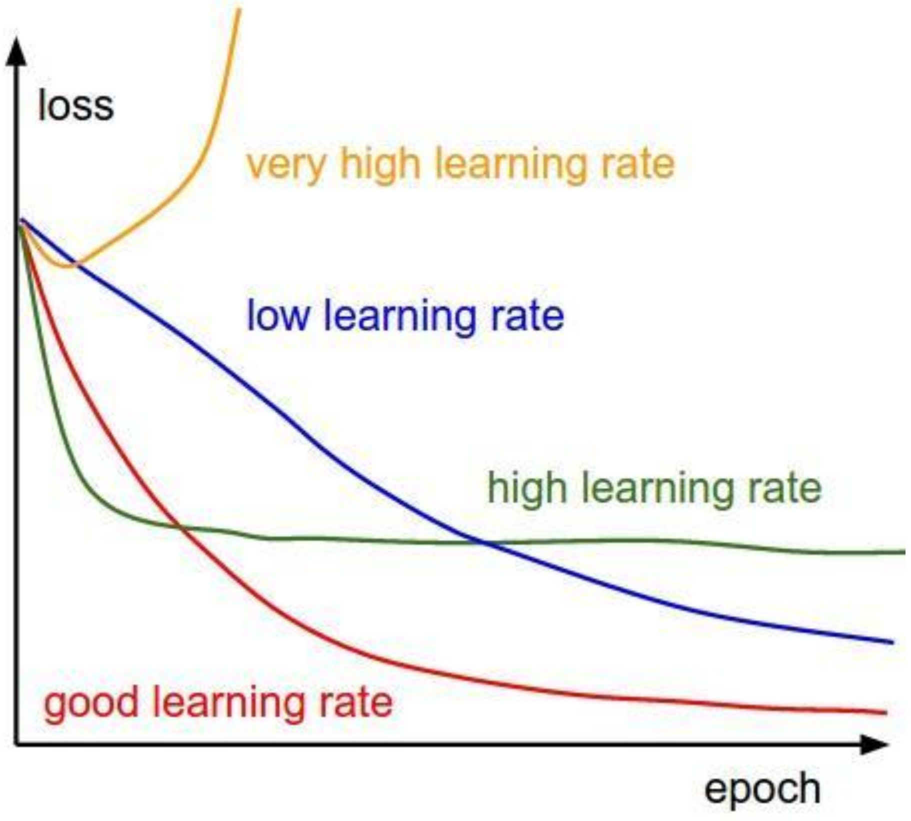


Figure 8: Examples of different learning rates

Data needed to be augmented in order to avoid the overfitting problem. Rather than calling the Dropout method, the training data was artificially expanded. The idea is to alter the training data with small transformations to reproduce the variations occurring when someone is writing a digit.

Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation techniques. Some popular augmentations used are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more.

The following techniques were applied:

- Randomly rotate some training images by 10 degrees
- Randomly Zoom by 10% some training images
- Randomly shift images horizontally by 10% of the width
- Randomly shift images vertically by 10% of the height

Applying the above techniques on the MNIST training dataset:

- Without data augmentation, an accuracy of 98.114% was obtained
- With data augmentation, 99.67% of accuracy was achieved.

Trivia: Most errors were found to be due to the fact that 9 and 4 have a similar typeface.

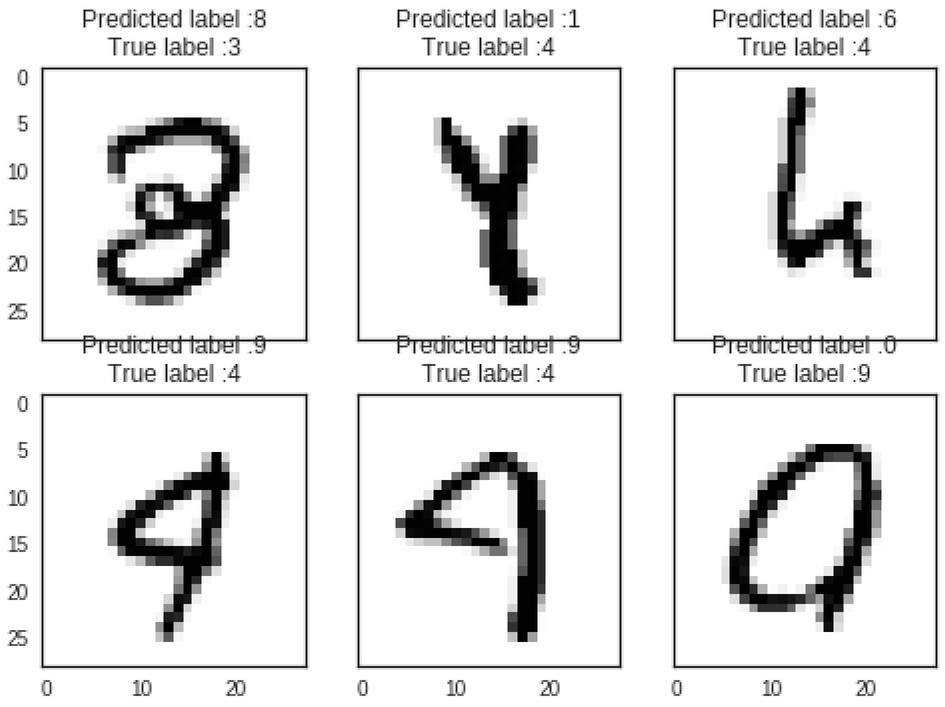


Figure 9: Top 6 errors in the model

These are some of the techniques that can be used to realize better results.

In the future, the whole platform can be expanded to improve workflow and efficiency by setting up a Docker environment. To work on the cloud, the Nvidia GPU Cloud (NGC) can be setup as a PaaS (Platform as a Service). This helps in managing multiple developers requiring GPU resources for their model training by building a single server connected to the local network. OAuth credentials and SSL can be used to setup the architecture. This method could help set Amity University as front-runner for Machine Learning avenues in the future.