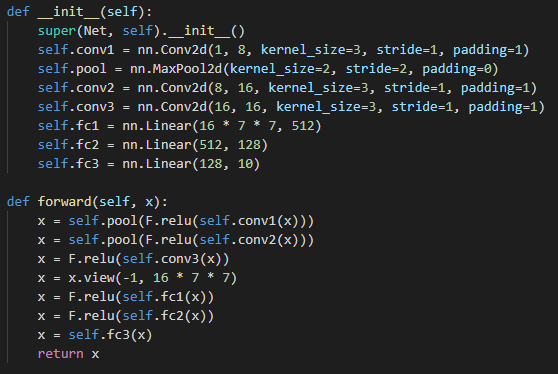
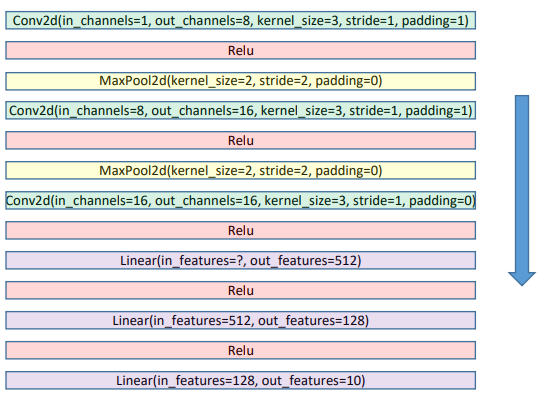
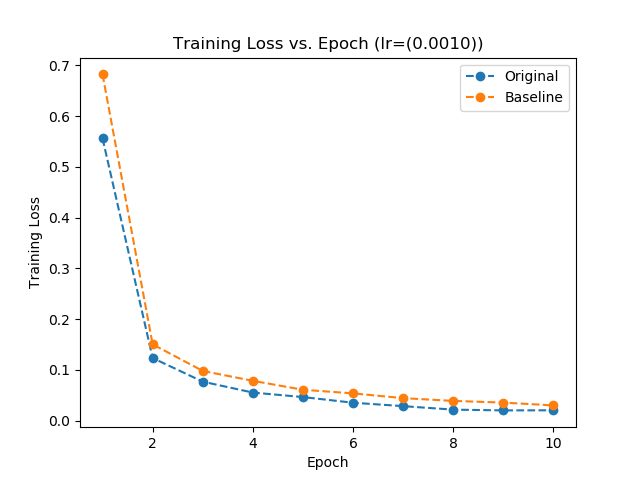
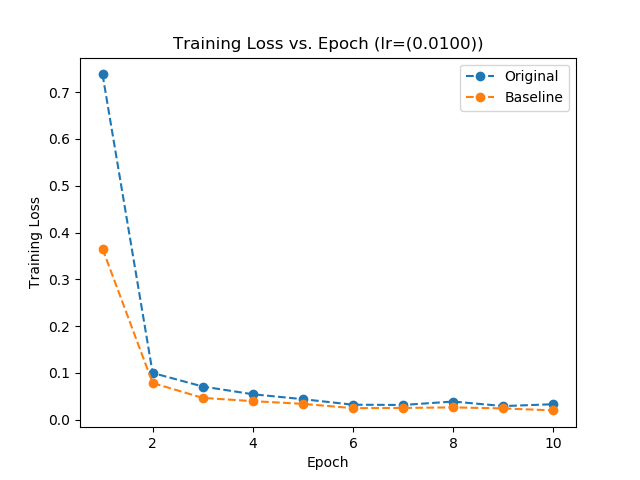
Using smaller filters (e.g. 3x3 conv) is better – because fewer parameters

ResNet best default CNN architecture

Black n’ White 🡺 use 2D conv

Colored 🡺 use 3d conv

**Part 1 – Baseline Network on MNIST Dataset**

1. *Create the network as shown in figure 1*
2. *Find a proper learning rate and plot the training loss vs. epoch*
3. *Compare the test performance with the baseline network provided to you on the MNIST dataset*

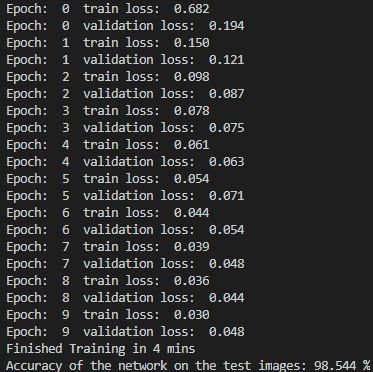
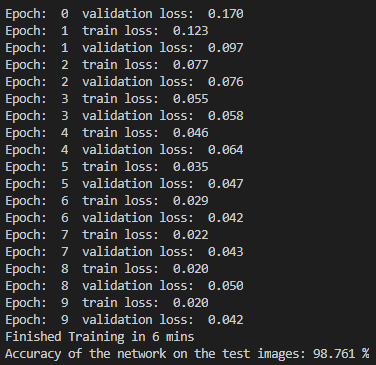
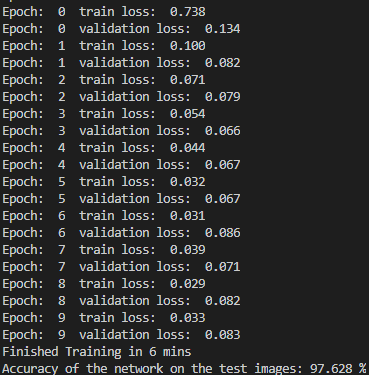
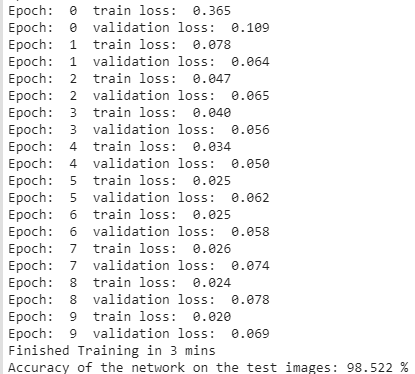
Exploring part 2; specifically, the learning rate section, lead me to a solid learning rate of 0.01. I also explored others, but for this first portion, I only compared 0.01 and 0.001, two of the best performing as seen in part 2. The comparisons are shown above in the plots.

Figure : Original - LR=0.01

Figure : Baseline - LR=0.01 (ran on google colab)

Figure : Original - LR=0.001

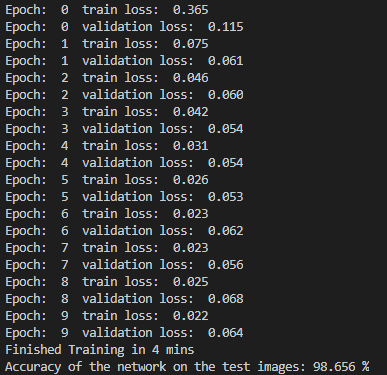
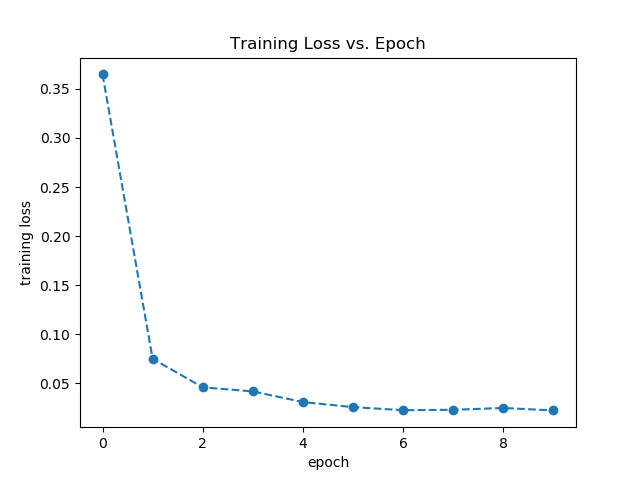
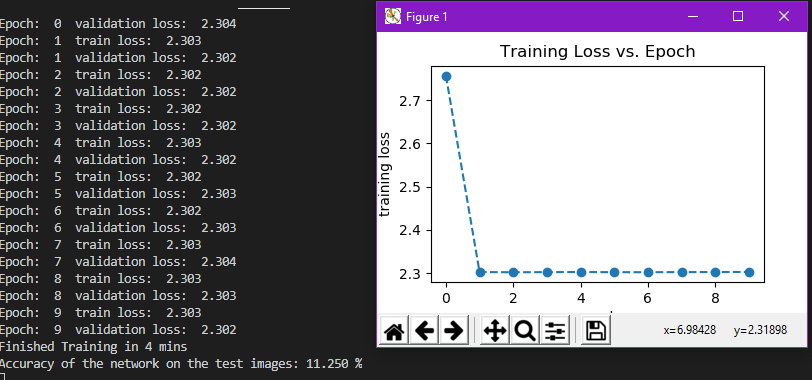
Figure : Baseline - LR=0.001

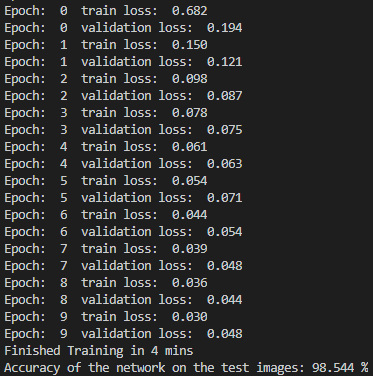
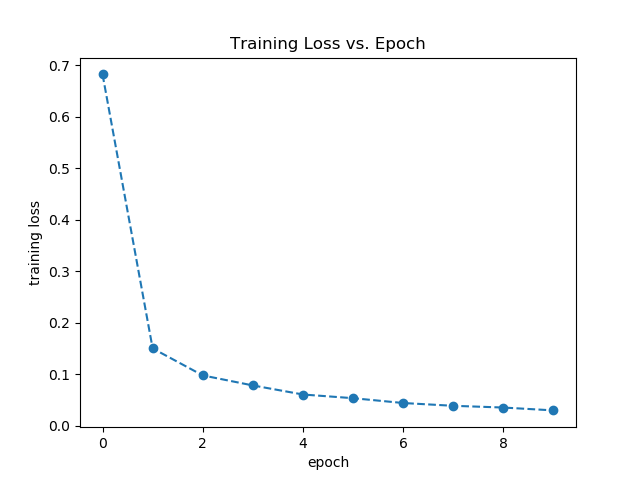
You can see in the above results that both baseline runs had a significantly quicker runtime, >=2 minutes quicker. Neither CNN necessarily performed better given my restricted testing, but you can see the original had the high of 98.761%, and low of 97.628% on the validation set, whereas baseline had a greater average overall. In this case, it also seems that a learning rate of 0.001 performs better overall.

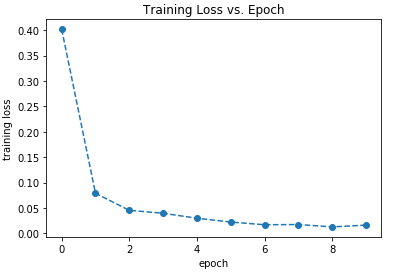
**Part 2 – Model Exploration**

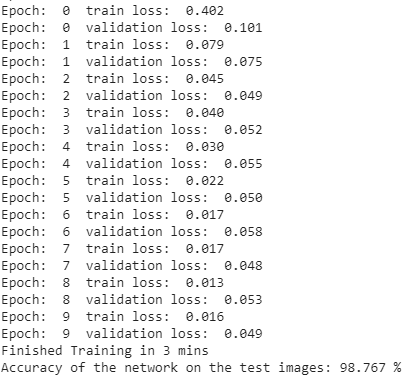
**Part 2.1**

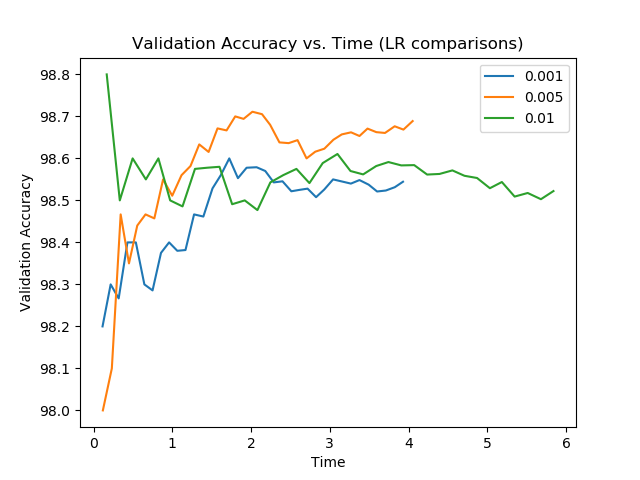
*The goal of this section is to understand the impact of the following hyperparameters and algorithmic choices on the performance of the system.*

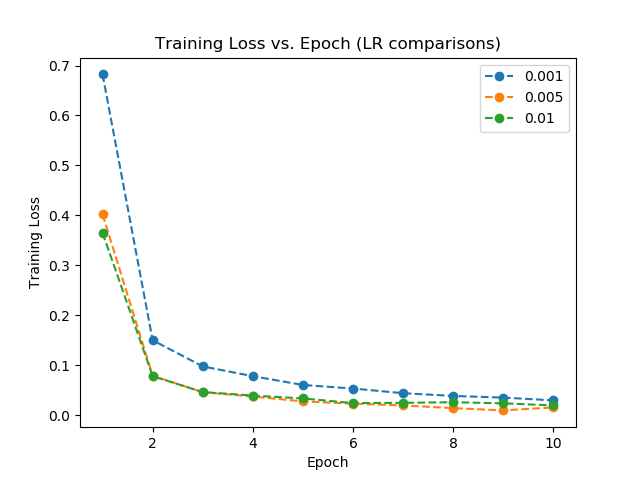
* Learning Rate (LR) & Optimizer: Adam or SGD
  + Notes on readings:
    - Majority of learning rates fail to train the specified model
    - LR’s too low never progress, and too high causes instability/no convergence
    - Training time can be greatly affected by learning rate
    - Hyper parameters are not invalidated by linear scaling the model
    - Adam is essentially a combo of RMSprop and Stochastic Gradient Descent w/ m
    - Adam is closing in on SGD w/ m to become the best optimization algo
    - As referenced in a [data science article](https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c), they mention a paper shows the optimal value for weight decay depends on number of iterations during training.
    - Though adaptive optimizers have better training performance, it doesn’t imply higher accuracy, or, better generalization in valid data
    - In general, adam has the lowest training error & loss, but not validation
  + *Find proper LR for Adam and SGD, plot training loss vs training epoch number, and compare the convergence speed of the two optimizers and their respective test classification accuracies*
  + *Find proper LR for Adam and SGD, plot training loss vs training epoch number, and compare the convergence speed of the two optimizers and their respective test classification accuracies*
    - Adam
    - Learning Rate of 0.01 - **good**
    - Learning Rate of 0.05… you can see a bigger learning rate had hugely negative, or **poor**, results with regards to both loss and accuracy. Also, technically it converges, but the training loss is huge.
    - Learning rate of 0.001 – still worse than 0.01, but ever so slightly, and only ran with one random seed value, so they are likely very similar -- **good**.

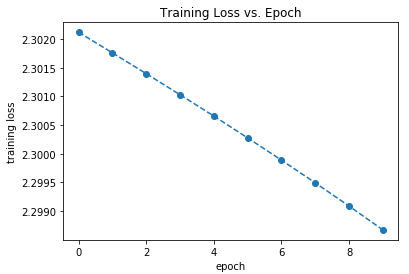
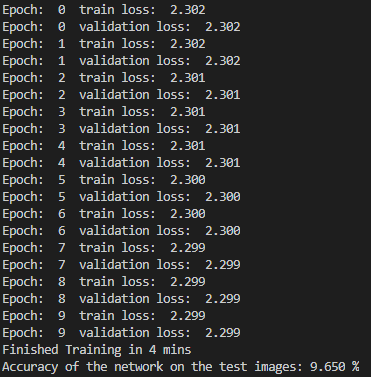


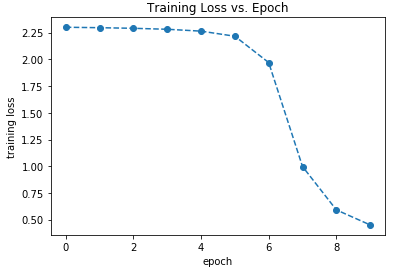
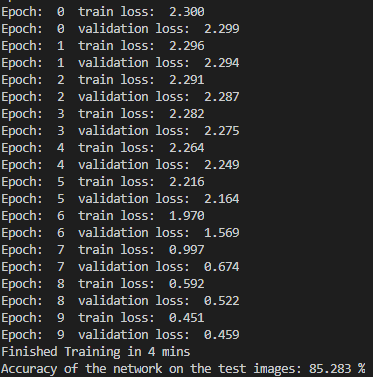
* + - ****Learning Rate of 0.005 – **great**

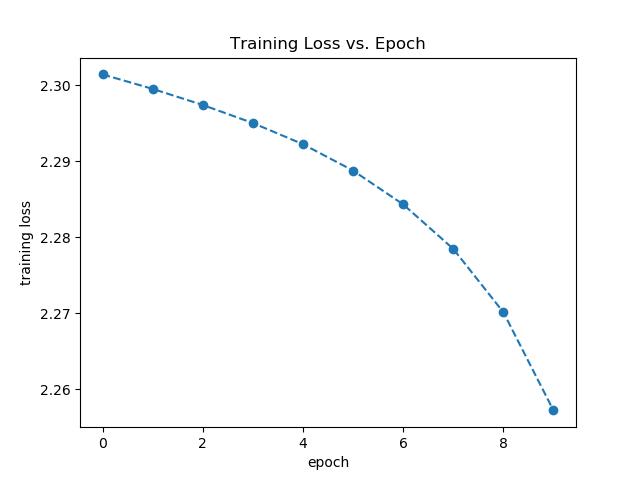
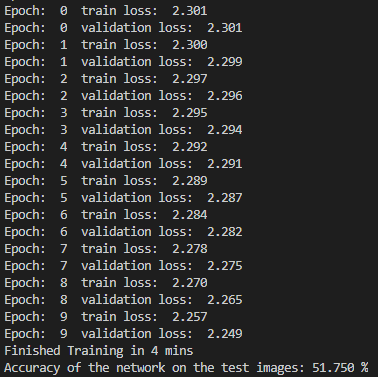
****

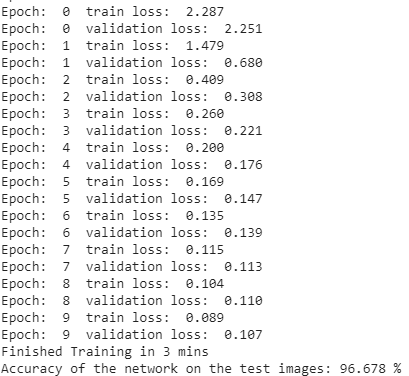
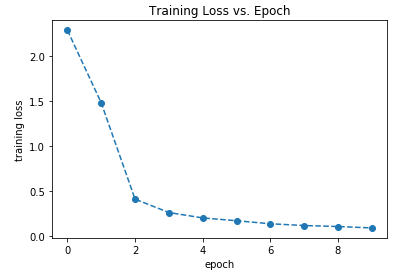
* + - ****Adam Comparisons – below you can see the comparisons between different Adam optimizer learning rates.

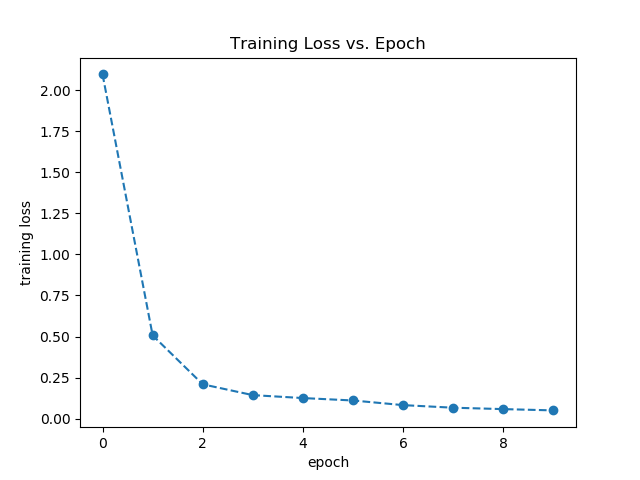
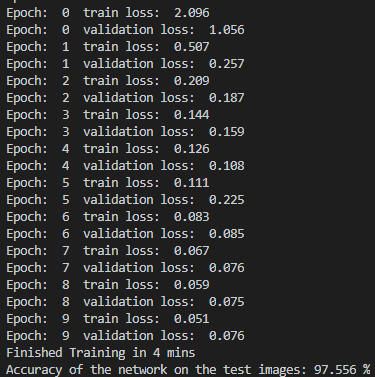
****

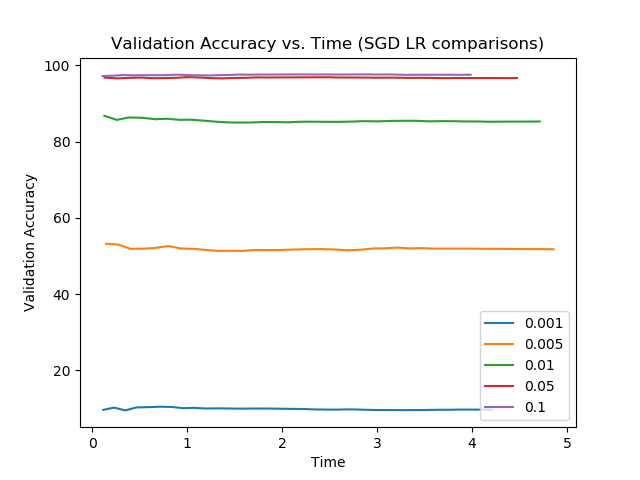
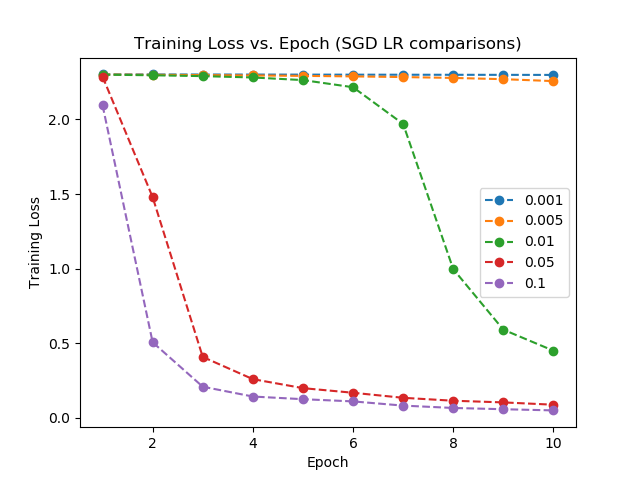
* + SGD
    - Learning Rate of 0.001 -- **poor**
    - Learning Rate of 0.01 -- **ok**

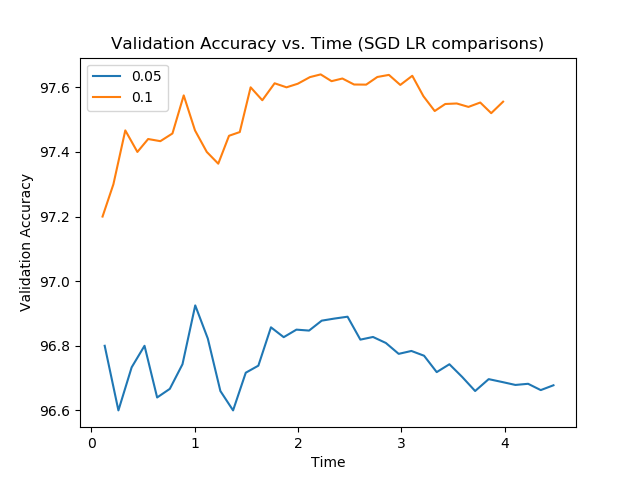


* + - Learning Rate of 0.005 – **poor**, maybe with higher iterations it would perform better
    - Learning Rate of 0.05 -- **good**

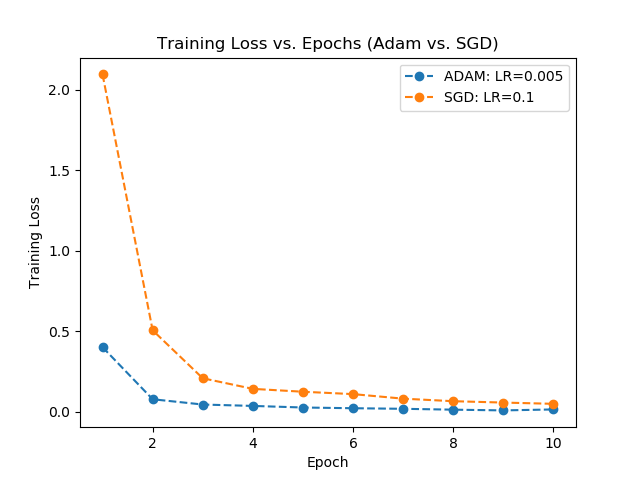


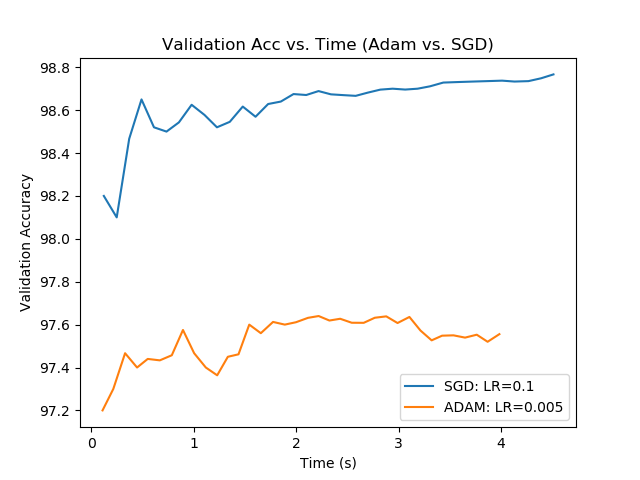
* + - Learning Rate of 0.1 -- **good**
    - SGD Comparisons – below you can see the comparisons between different SGD optimizer learning rates.

****



* + - After laborious analyses, I found the proper/best learning rate for both optimizers: Adam and SGD. Though random number seeds lead me to be wary within 0.5% or so, I decided to go ahead and use the highest working accuracy on the validation set. **Adam** came out to hit 98.767%, which held a learning rate of **0.005**. **SGD**, on the other hand, hit 97.556%, which held a learning rate of **0.1.**





*Describe the lessons you learn from the experiments*

***Part 2.2***

* Activation Functions
  + Notes on readings:
    - Sigmoid function saturate, and are only sensitive to changes around the input midpoint
      * It becomes hard to adapt the weight/improve performance after saturated
    - Deep CNN’s ~~are~~ were difficult for sigmoid (vanishing gradient problem)
      * ~~Though ReLU can in some cases fix the vanishing gradient problem~~
      * ReLU DOES fix the vanishing gradient problem in most cases
    - ReLU
      * Default activation function for CNN and MP
      * Nonlinear activation function… but behaves like a linear one
      * has become the default activation function for most CNN
      * is trivial to implement, whereas sigmoid uses exponential calculation
      * capable of outputting a true zero value 🡪 can lead to acceleration of learning
      * A limitation is where large weight updates lead to the summed input being (-) always
        + See Leaky ReLU (LRel), ELU, PReLU, for a patch/fix to this issue
    - …consider setting bias to a small value (e.g. 0.1)
    - Weights of a NN must be initialized to small random values
      * Modification to Xavier initialization 🡪 He initialization +/-sqrt(2/n)
    - Good practice to scale input data (e.g. standardizing variables to have…
      * zero mean
      * Unit variance / normalizing each value to a 0 – 1 scale
    - May be a good idea to use a form of weight regularization, [L1/L2 vector norm](https://machinelearningmastery.com/vector-norms-machine-learning/)

**Part 2.3 – Early Stopping**

* Early Stopping
  + Little training leads to underfitting of training and test sets
  + Too much training leads to overfitting of training, and poor performance on test set
  + SOLUTION 🡪 train and stop when performance on validation dataset starts to degrade (ES)
  + Simple case 🡪 training stopped when validation test set decreases (i.e. increase in loss)
    - In practice, however, fluctuations are common, so be careful…
      * Generally, ‘slower’ stops lead to improved generalization

**Part 2.4 – Data Augmentation**

* Data Augmentation
  + Invariance – a CNN that can robustly classify objects even placed in different orientations
    - A CNN can be invariant to translation, viewpoint, size, illumination
  + Offline augmentation – good for small datasets
  + Online augmentation – good for large datasets
    - Transformations on mini-batches then feed to model
  + Use it smartly, not just to increase data (no need for irrelevant data)
  + For this project, using *torchvision.transforms* will allow augmentation on dataset

**Part 2.5 – Network Depth vs. Network Width**

* Network { Depth vs. Width }
  + Network depth is number of layers
  + Network width is the max number of nodes in a layer
    - Corresponds to different color channels of an image . . .?
    - Depth of a conv. Layer 🡪 number of filters
  + <http://proceedings.mlr.press/v33/pandey14.pdf>