Deep Learning

Assignment 1 Report

**Student Name:** Aniruddha Kalburgi

**Student ID:** R00171450

**Email Contact:** [aniruddha.kalburgi@mycit.ie](mailto:aniruddha.kalburgi@mycit.ie)

**Part A - Tensorflow and the Low Level API:**

**Task (i):**

* Designed a binary classifier with 2 layers configuration as stated for task(i), with classes 3 and 8 chosen for classification.
* The model was run for 40 epochs.
* This resulted in 96% of classification accuracy.
* Total time taken to train the model was 3.42 seconds.

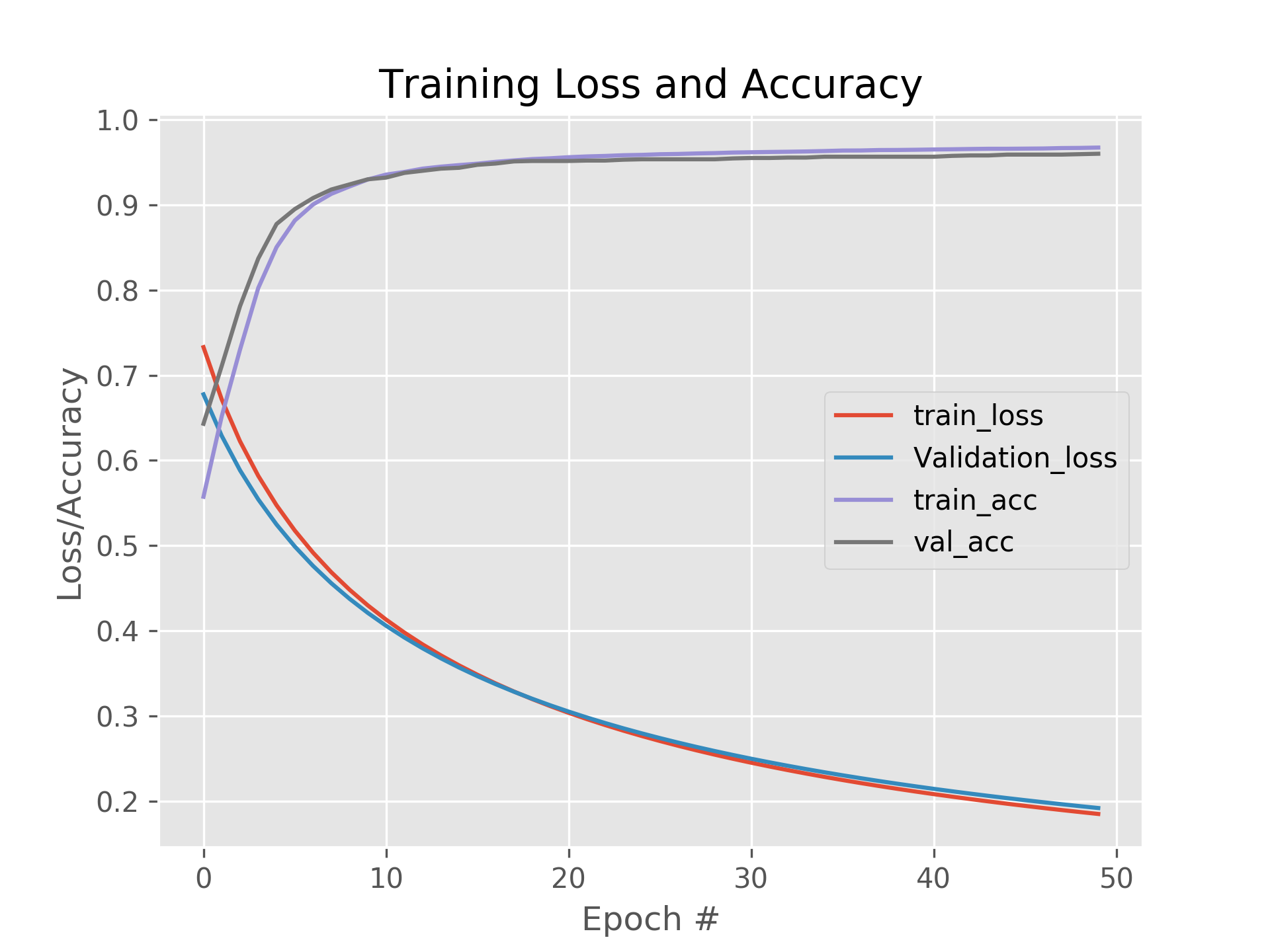


Figure: Task (i) - 40 Epochs, Accuracy: 96%, Time: 3.42 seconds

**Task (ii):**

* Designed a multi-class classifier for all 10 classes with 3 layers as stated
* for task (ii), the following results were obtained with 40 epochs for the
* training phase.
* Final Validation Accuracy: 54.47%
* Training Time: 15.57 seconds

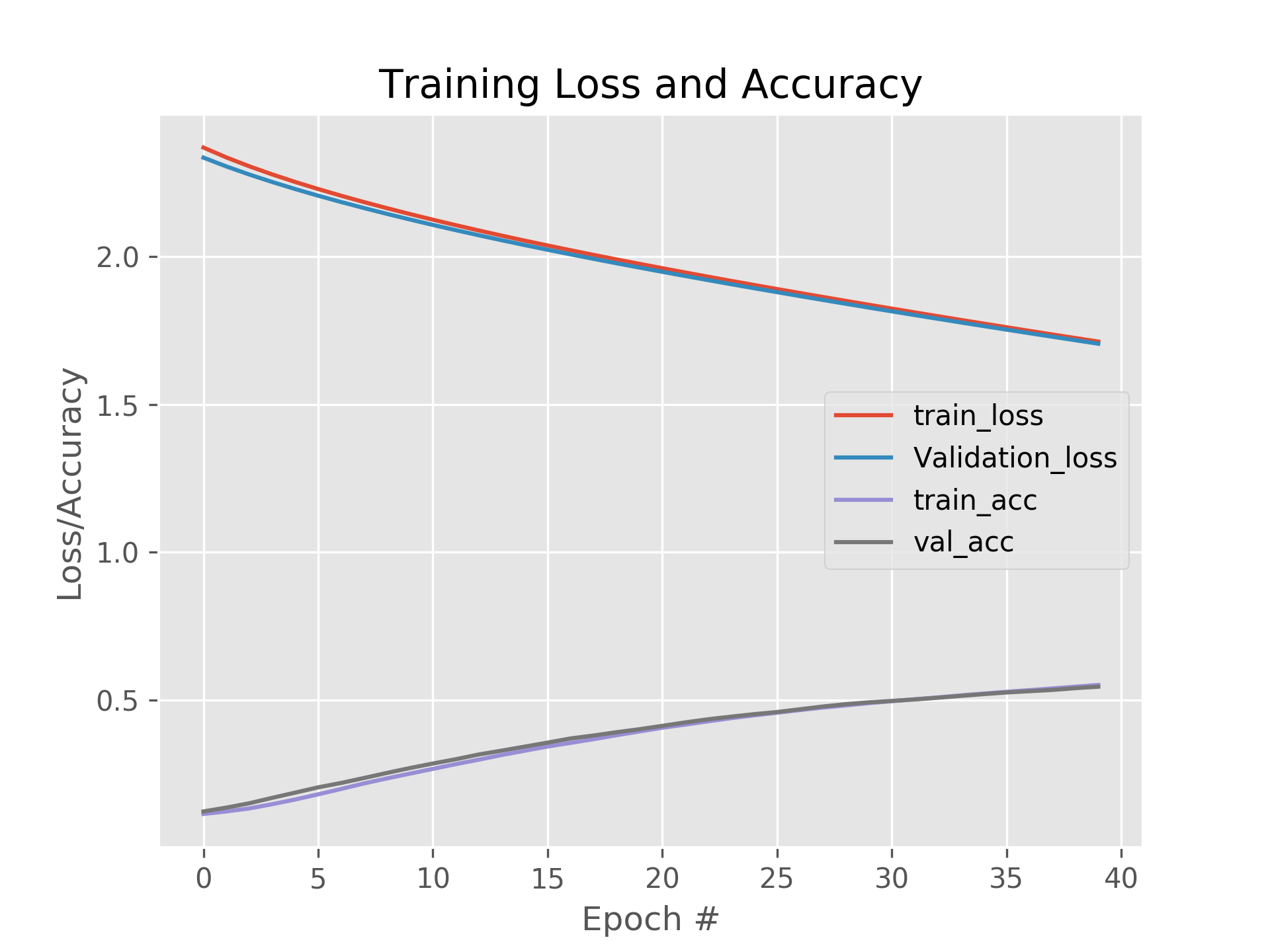


Figure: Task (ii) - 40 Epochs, Accuracy: 54.47%, Time: 15.57 seconds

**Task (iii):**

* From the data in following table below, we understand that Mini-batch size 32 had significant positive impact on the accuracy, but due to very small batch size, it takes huge training time – 59.19 seconds.
* Where, the higher batch size of 2048 finished training in 7.58 seconds but yielded only 81.57% accuracy.
* As we see, batch size 32 yield the highest accuracy, then it falls to 87.5 for batch size 64. But, the accuracy improves again for the batch sizes of 128, 256 and 512.
* This makes clear that higher batch sizes contribute to the improvement in training time, but not necessarily to the accuracy.
* As we begin to increase the batch size beyond 256, we see continuous drop in the accuracy, so we can infer that even too much higher batch sizes drops the accuracy.
* Yet, all considered mini-batch sizes outperformed in terms of accuracy over the non-mini-batch based model in task(ii).

|  |  |  |
| --- | --- | --- |
| **Mini-batch size** | **Accuracy** | **Training Time (seconds)** |
| 32 | 93.75 | 59.19 |
| 64 | 87.5 | 31.92 |
| 128 | 88.54 | 18.7 |
| 256 | 89.58 | 11.8 |
| 512 | 88.54 | 8.7 |
| 1024 | 84.03 | 8.04 |
| 2048 | 81.57 | 7.58 |

The images below shows the loss and accuracy over the 40 epochs for different batch sizes.

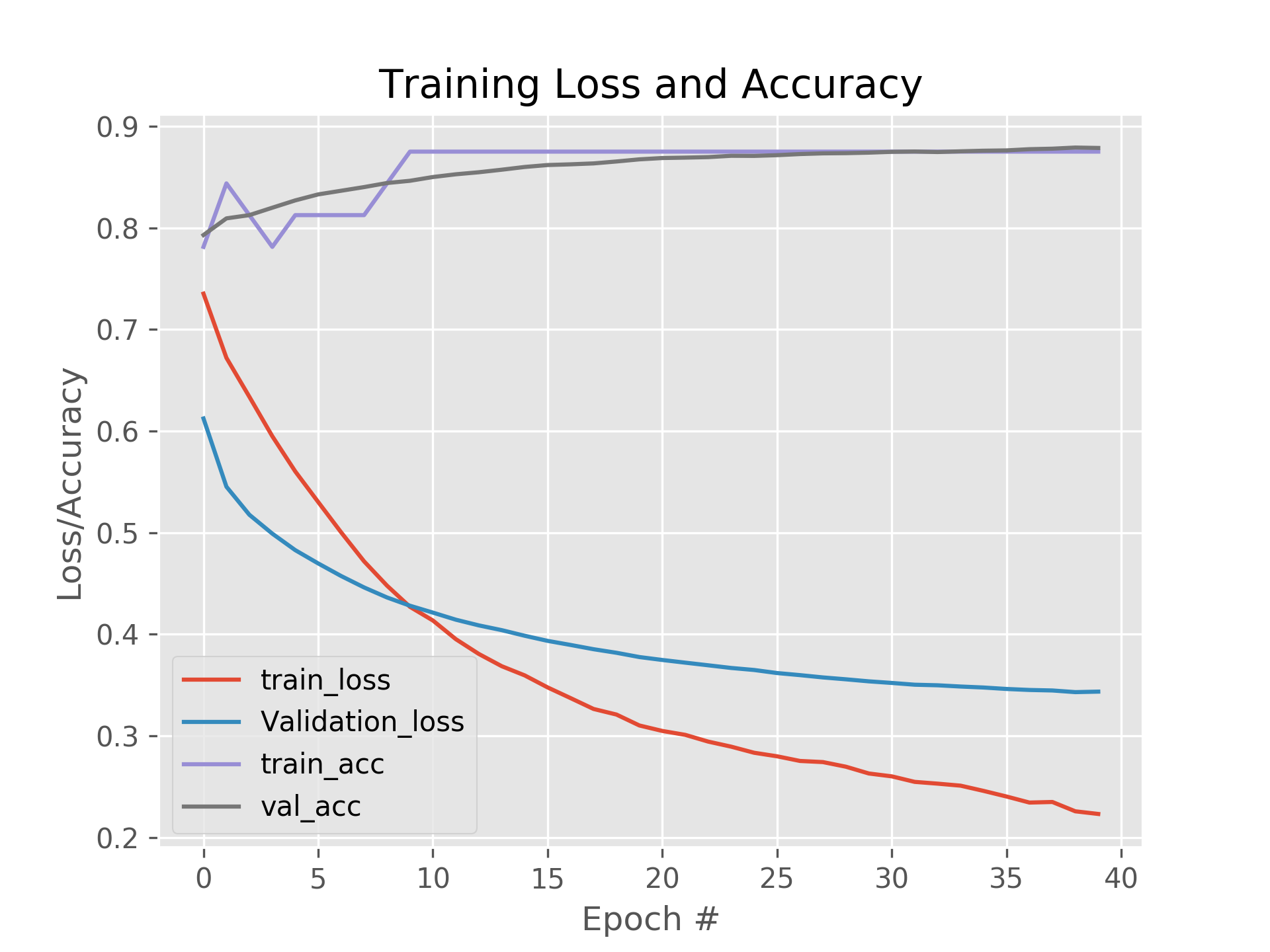
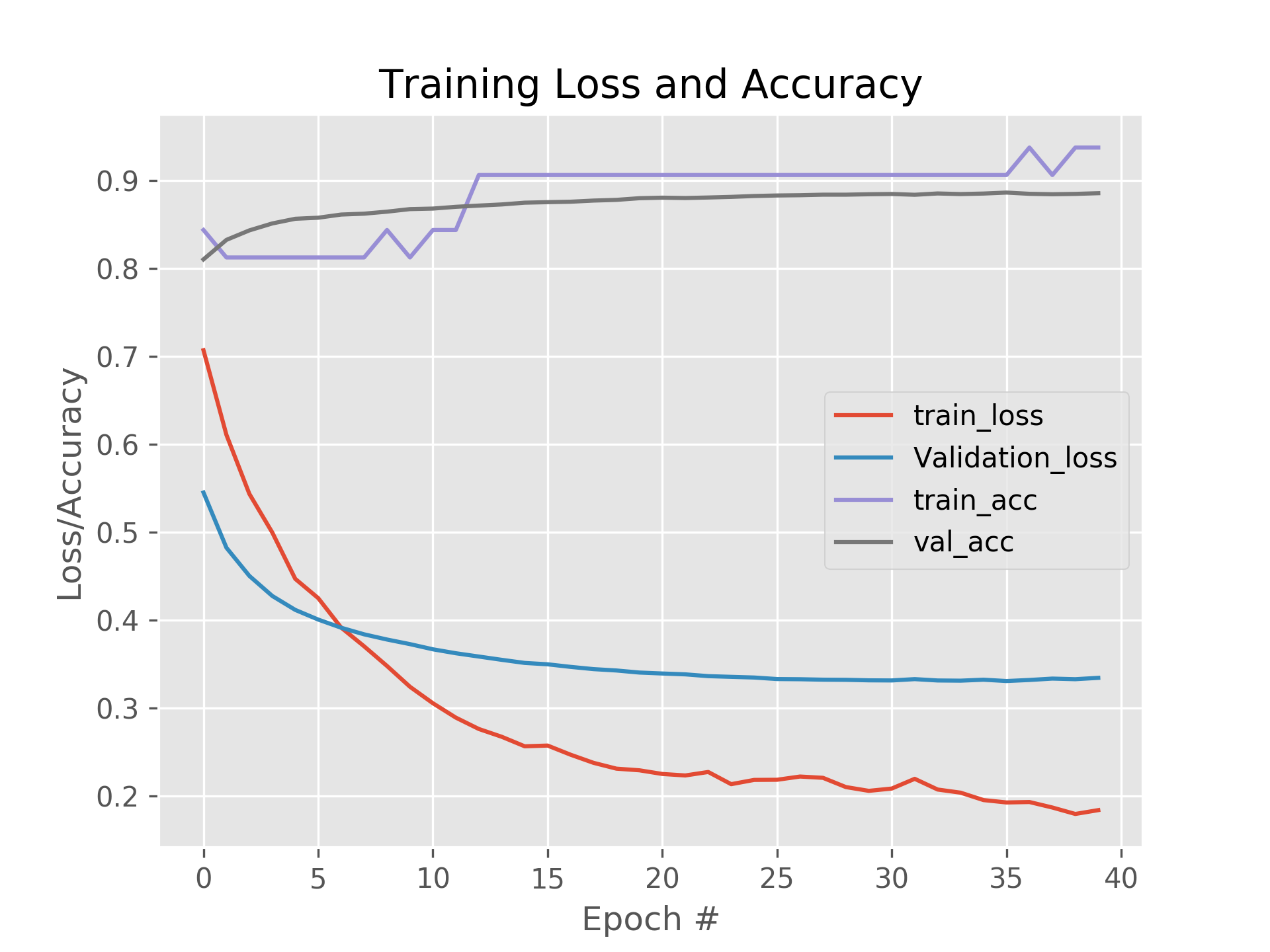
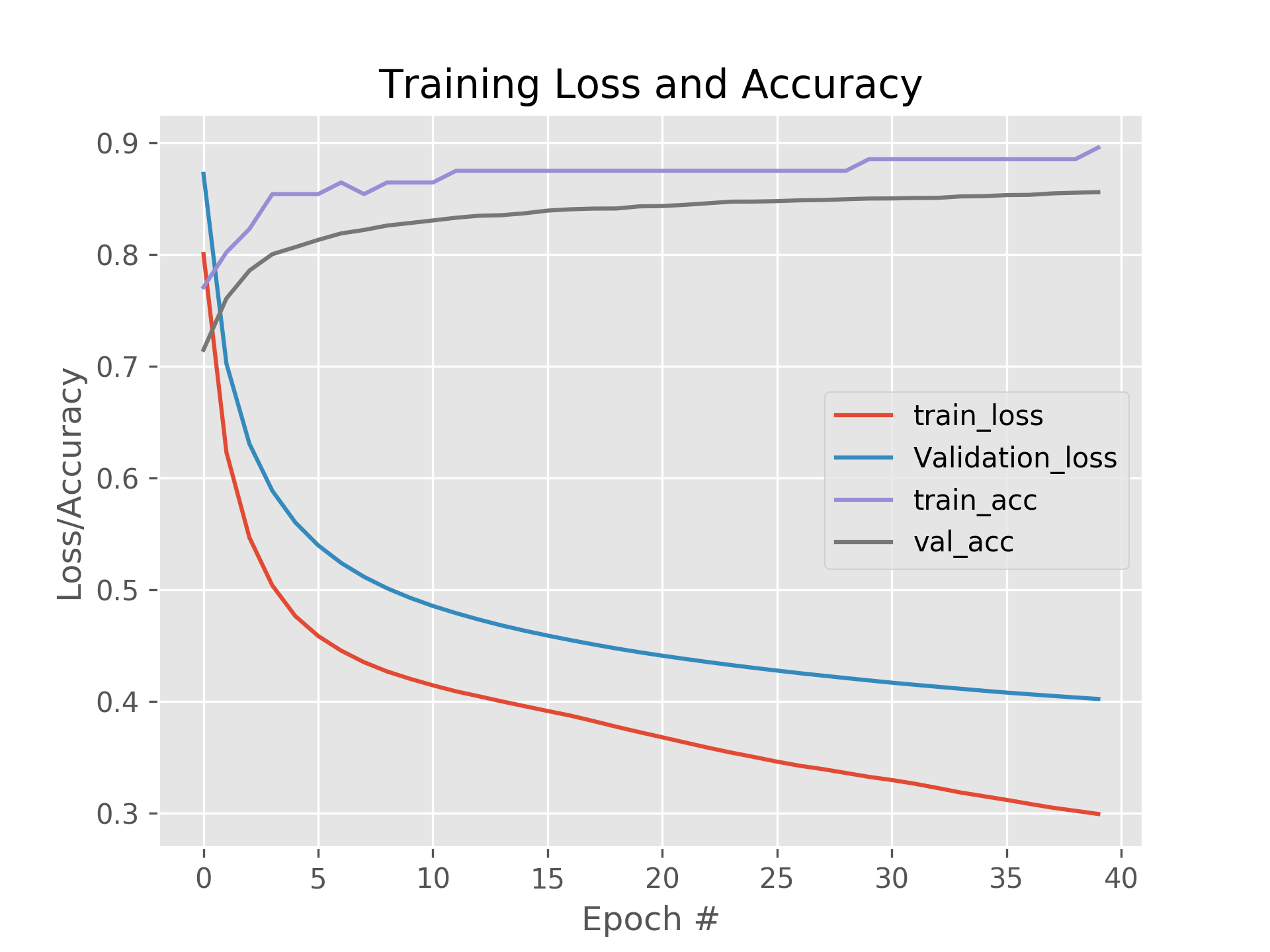


Fig: Batch size 32 Fig: Batch size 64



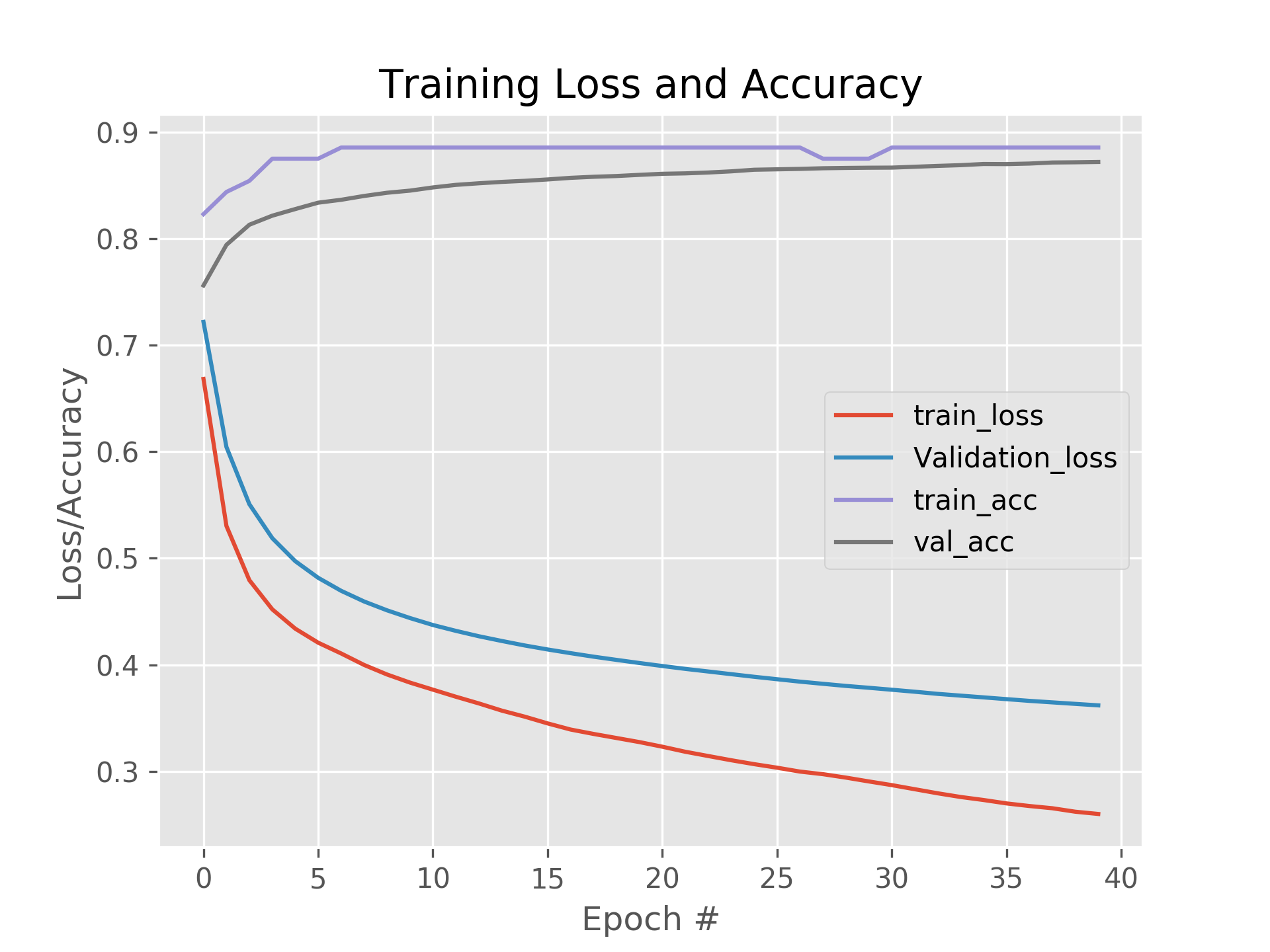


Fig: Batch size 128 Fig: Batch size 25

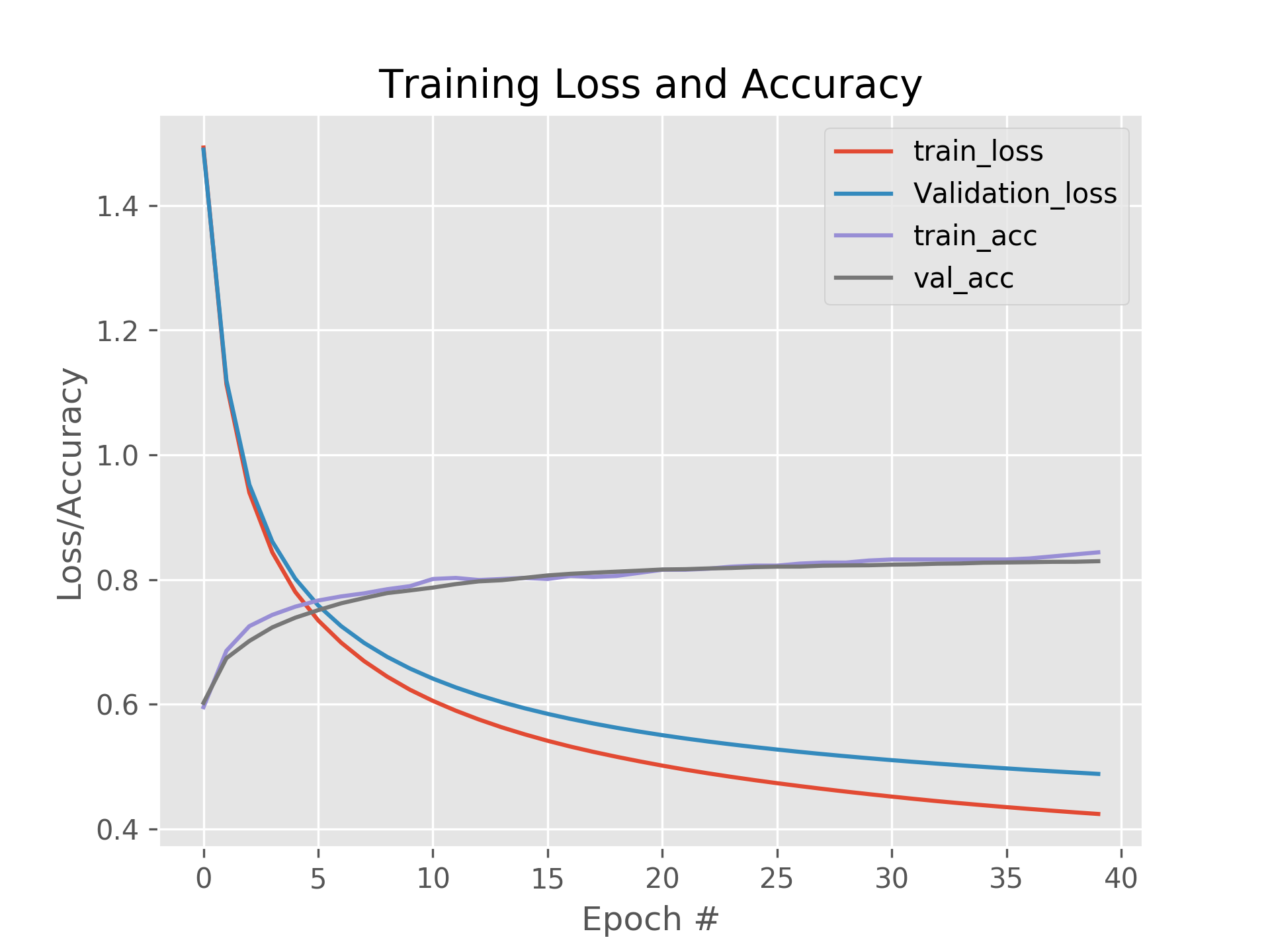
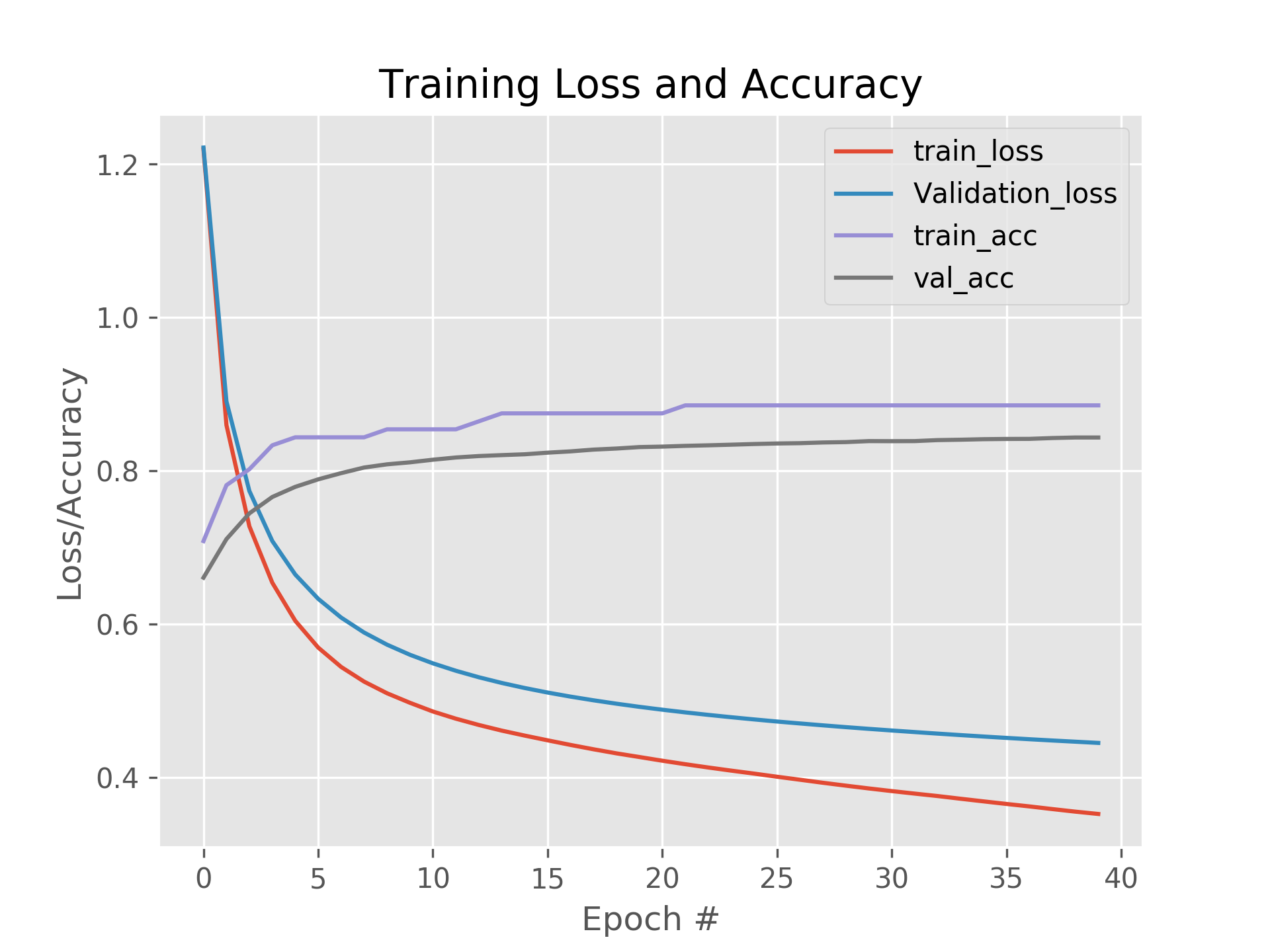


Fig: Batch size 512 Fig: Batch size 1024

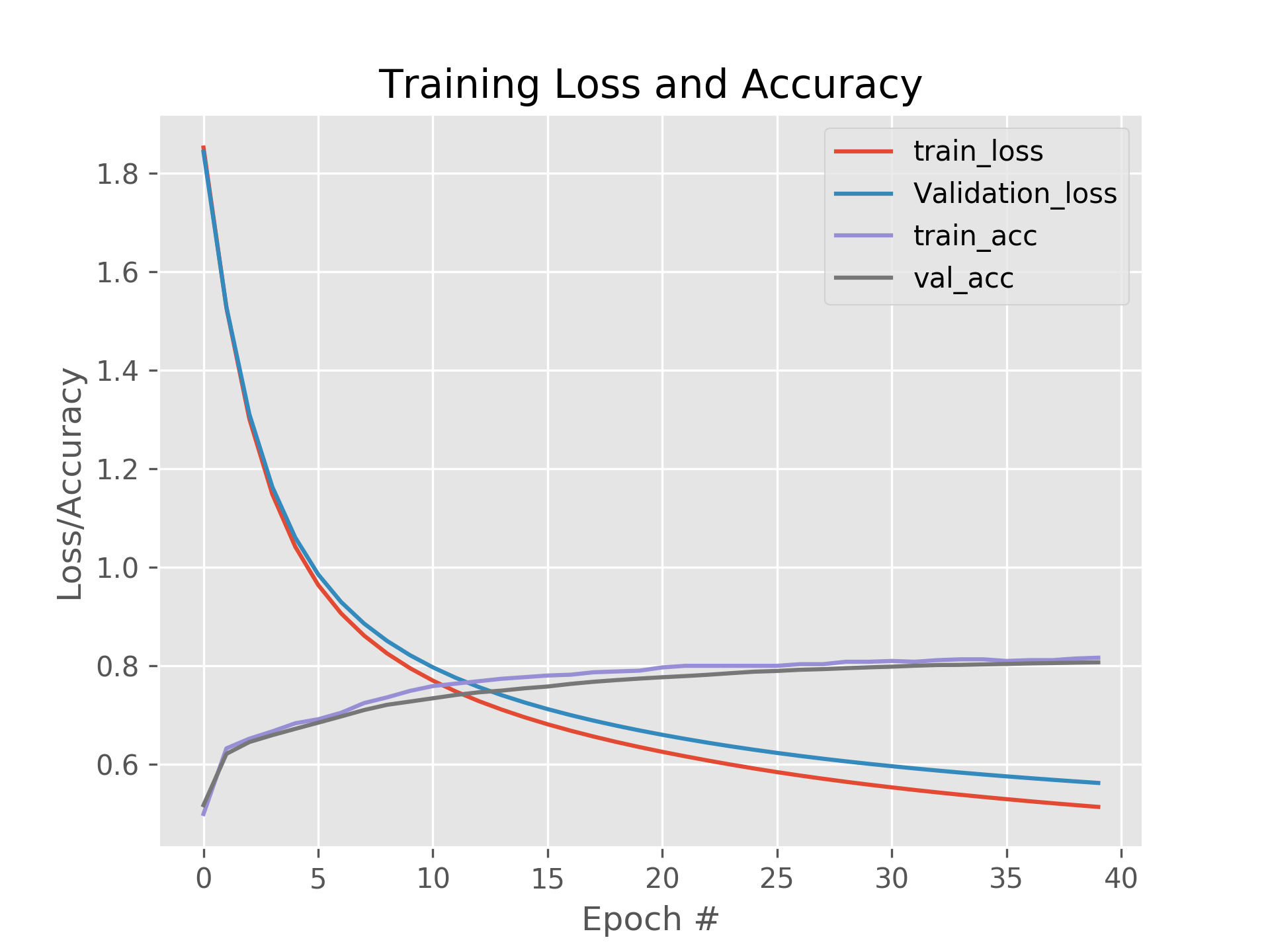


Fig: Batch size 2048

**Part B – Keras – High Level API:**

**Task (i):**

The benchmark classifier was built using 10 softmax units in the output layer, with batch-size 256 and 10% training data split for validation.

Running it for 20 epochs resulted in 85.51% accuracy.

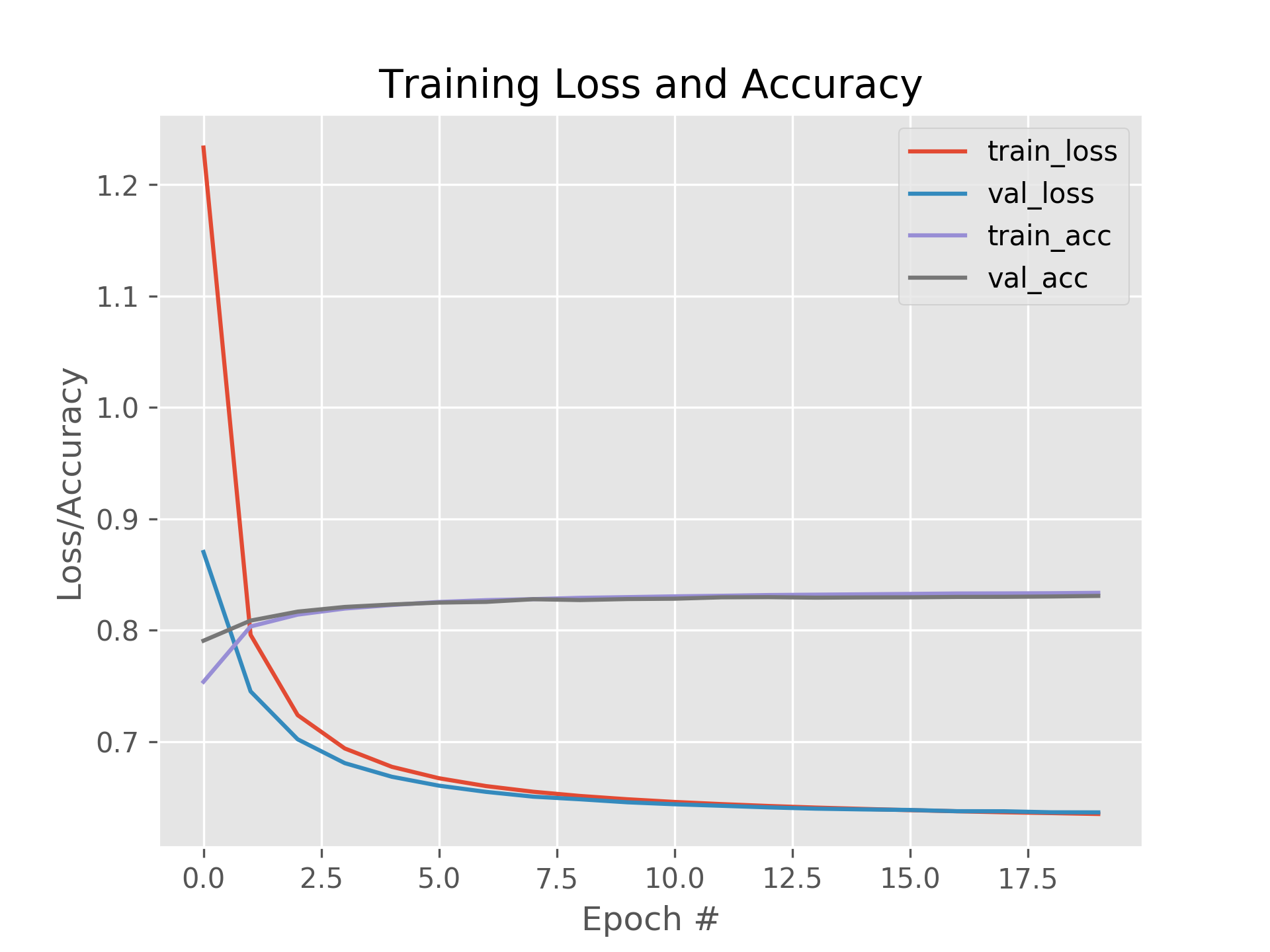


Fig: Baseline - Softmax Logistic Regression

**Task (ii):**

Configuration 1: L1 200 Neurons, L2 Softmax

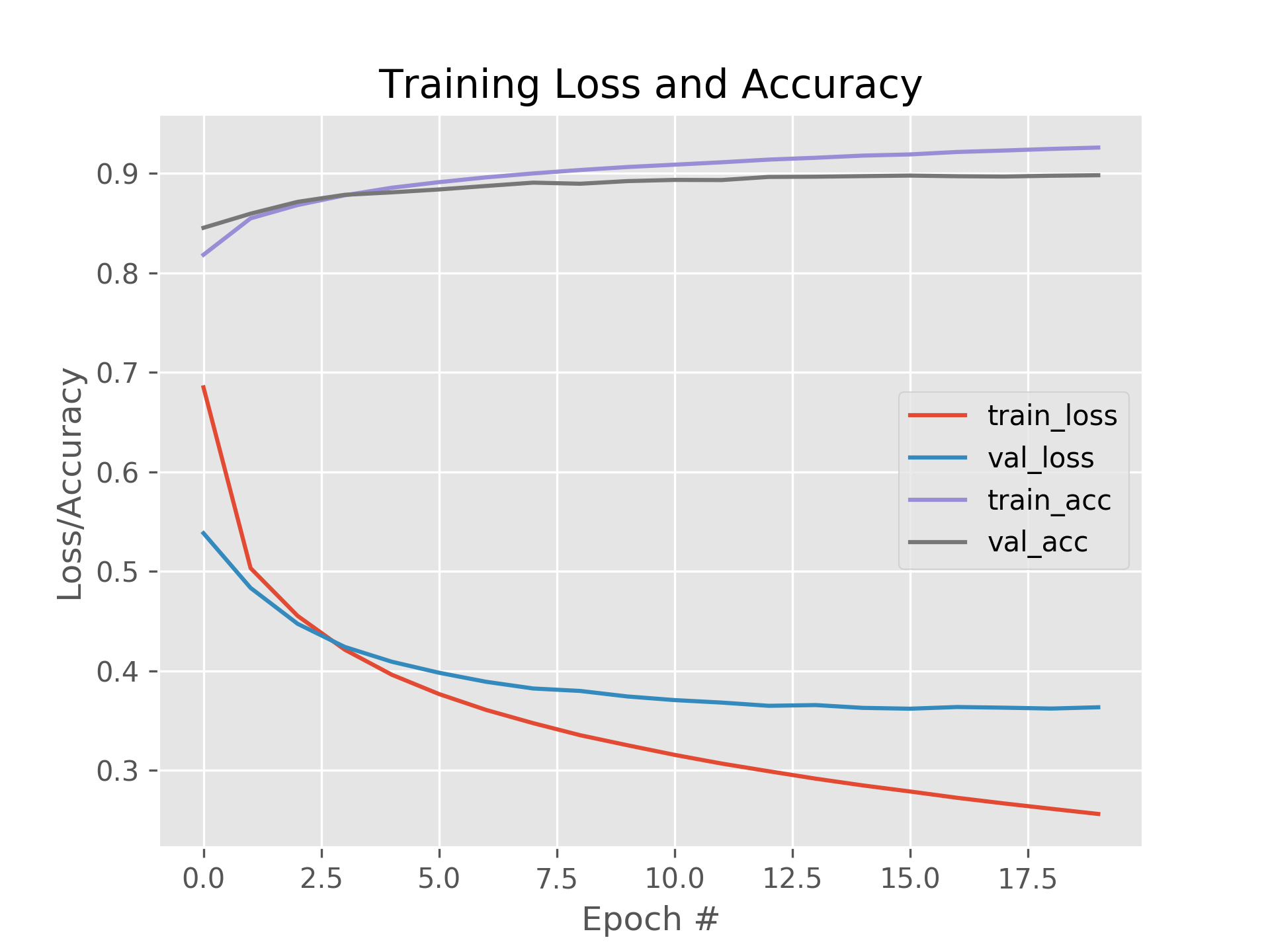
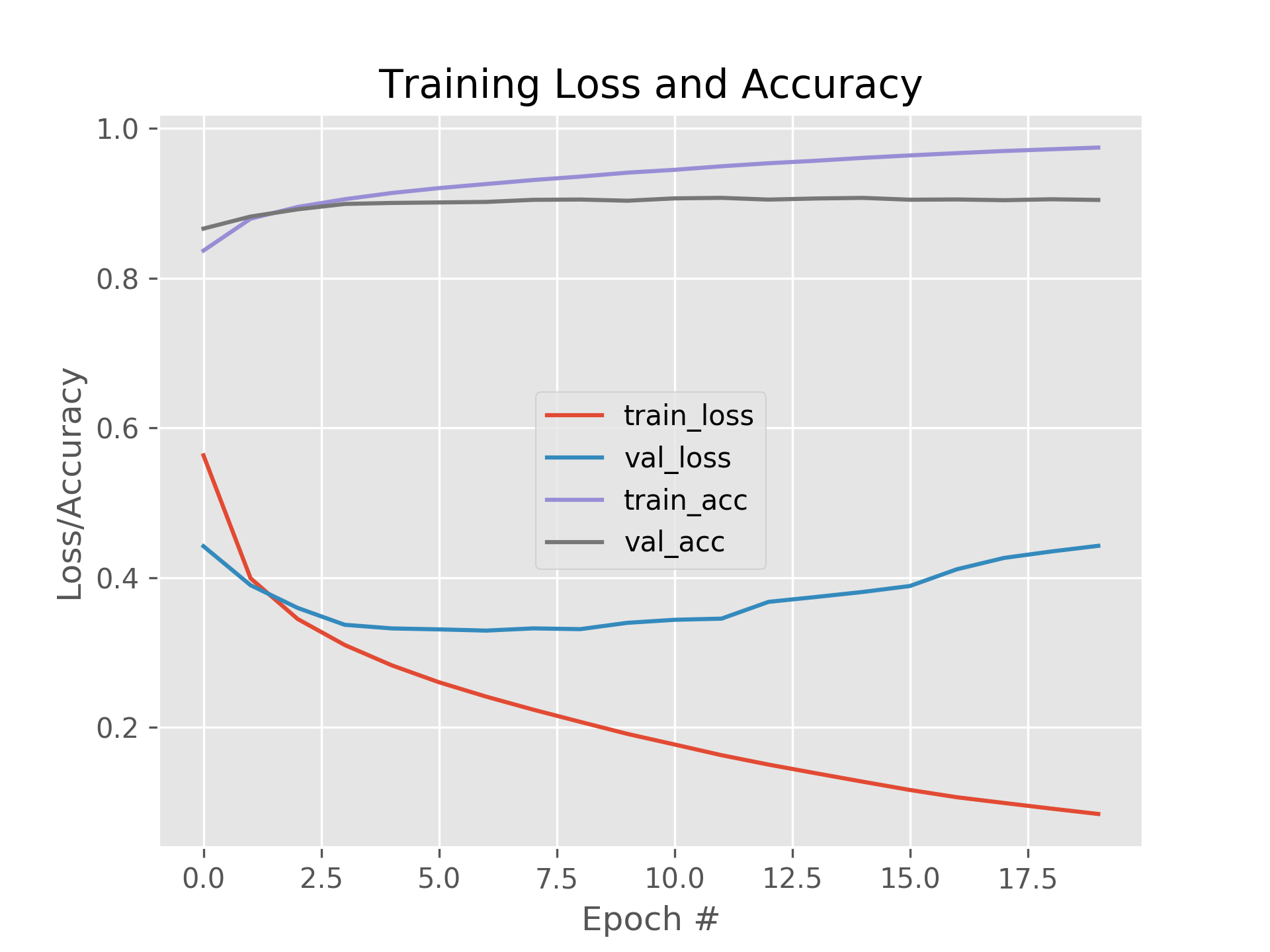


Fig: Config1, accuracy: 91.92%

Configuration 2: L1 400 Neurons L2 200 Neurons L3 Softmax, accuracy: 92.64%



Configuration 3: L1 600 Neurons L2 400 Neurons L3 200 Neurons L4 Softmax, accuracy: 92.68%

****

* With config 1, we got 91.92% accuracy, where as we got a slight improvement in accuracy with config 2: 92.64%, and 92.68% with config 3.
* But, config 1 yielded only reduction in validation loss, though it started to plateau nearby 20th epoch. Where, we see training loss and validation loss diverting in opposite directions to each other at 8th epoch in Config 2, and at 3rd epoch in config 3.
* As config 2 and 3 gives better accuracies, it’s worth investigating the effect of dropout and regularization on it.

**Task (iii)**

**Config 2: H1 400 relu, H2 200 relu, O Softmax**

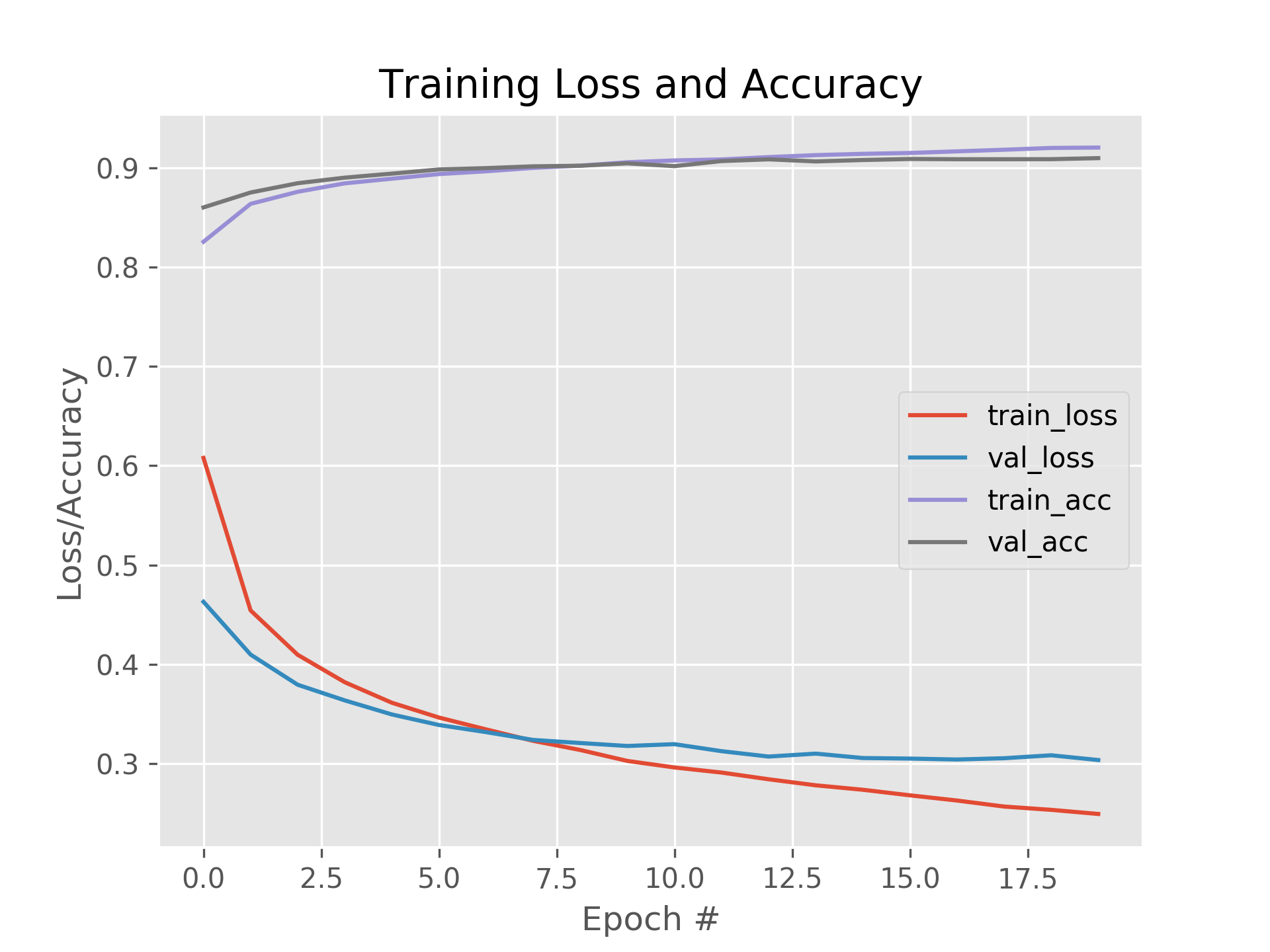


Fig: Config 2, with dropout , accuracy 92.89%

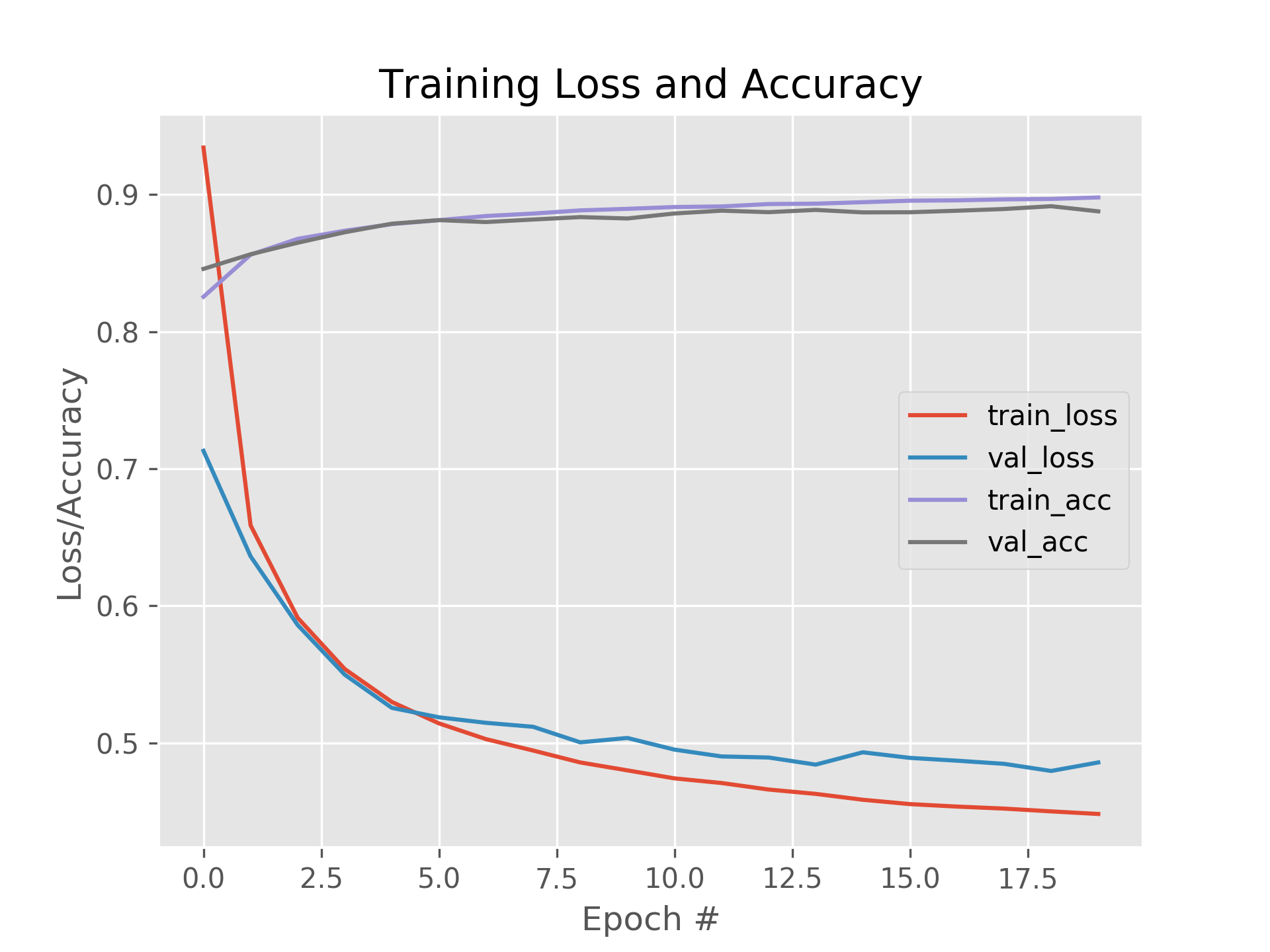
****

Fig: Config 2, with l1 regularization, accuracy 90.80%



Fig: Config 2, with L2 Regularization, accuracy: 92.19%

* For config 2, dropout resulted in better accuracy and better curves for both training and validation accuracies and losses.
* Dropout used was 0.4 for H1 and 0.1 for H2, where H is hidden layer. These values are decided based on highest accuracies offered throughout different dropout thresholds.
* Dropout was reduced for H2 because subsequent hidden layers have less number of neurons, so to consider predictions from many neurons as possible, the dropout was reduced to 0.1 and even beyond for config 3.
* **Similar strategy** was used **while choosing thresholds for L1 and L2** regularization individually for all configurations.
* With L1, we saw validation accuracy has started to drop below 90% after 17th epoch, but validation loss is still closely aligned with training loss. And the final accuracy is also around 90%, declaring L1 as the worst performer.
* With L2, we saw validation accuracy moviding down towards 90% until 20th epoch, yielding us with 92.19% of final accuracy, declaring it as a second best performer.

**Config 3: H1 600 relu, H2 400 relu, H3 200 relu, O Softmax**

****

Fig: Config 3, with Dropout, accuracy: 93.27%

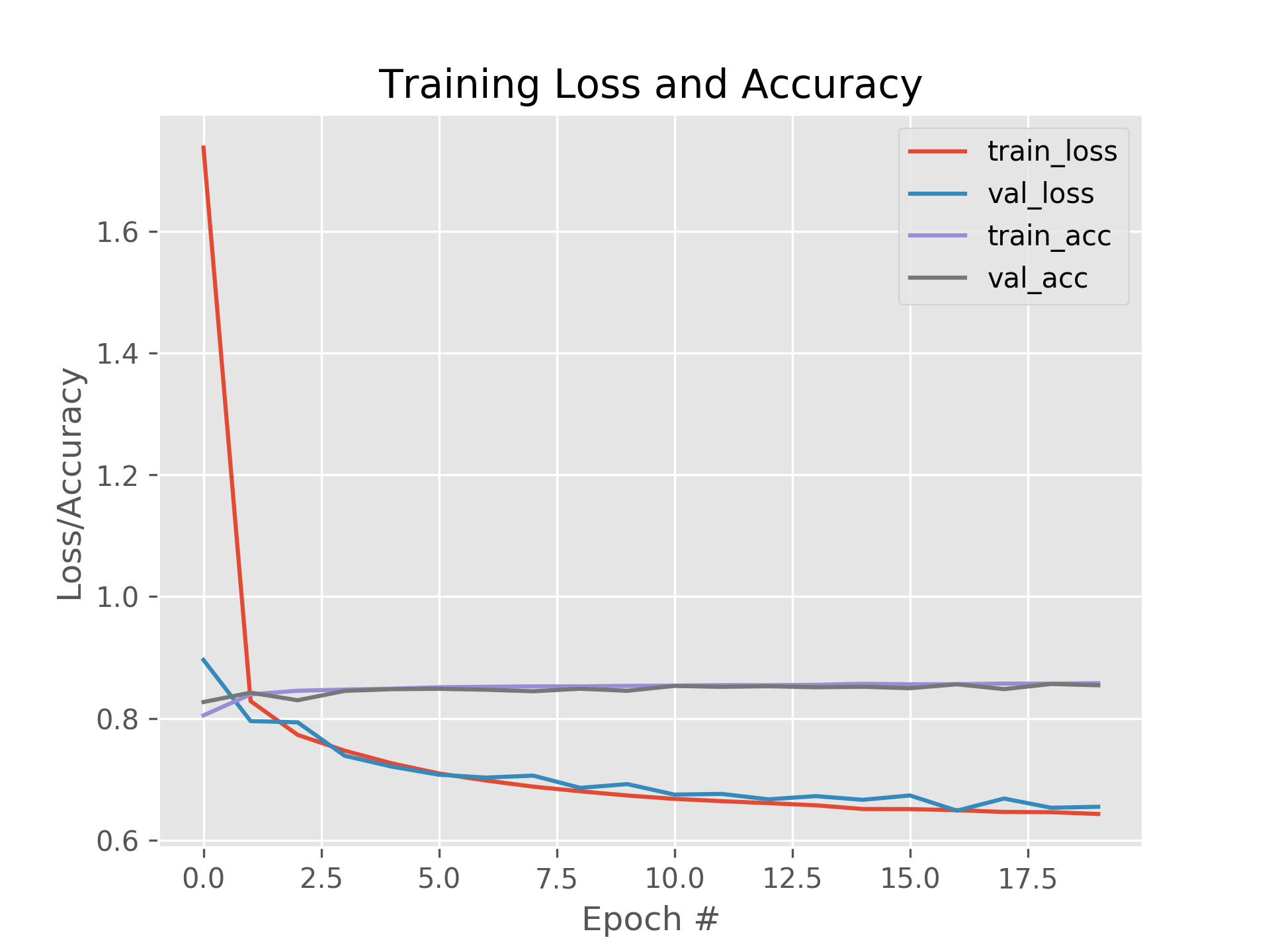
****

Fig: Config 3, with L1 Regularization, accuracy: 87.68%

****

Fig: Config 3, with L2 Regularization, accuracy: 91.15%

* For config 3, we have the same ranking for the approaches. Dropout being the highest performer yielded us with 93.27%, but, highest divergence in validation and training losses after 8th epoch.
* L2 regularization being the second best performer, yielded us with 91.15% accuracy, but, little bit divergence between training and validation losses.
* And, L1 regularization gave us the lowest accuracy of 87.68%, but, offered the consistent reduction in training and validation losses.
* Therefore, for both the deepest networks, dropout can be vouched as the best regularization method for this dataset. Although, running these configurations for different values of dropout and combinations of L1 and L2 regularization may help further improve the accuracy.

**Part C – Research:**

**Batch Normalization**

**Covariance shift** is the problem that affects the predictions made by the Deep Neural Networks. For images, it can happen in the case when there’s a change in the source of images, like when training images were taken from one camera, and the test images were taken from different camera which has different specifications. Such a difference often leads to the the shift in distributions, which demands retraining the model to adjust new distributions available in test images[2]. Because, in deep neural networks, the weights of each intermediate layer and its corresponding activation values also change in such a case, the overall retraining is very time consuming.

In logistic regression, we try to normalize the distributions to achieve the spherical contour, which allows the converge towards the local minima much faster with smoother learning curve, than compared to the erratic convergence when the contour is elongated where the distributions are spread too much away from each other. [1]

Batch normalization indepedently calculates the gamma and beta values for each mini-batch.

Such that, it allows each intermediate layer to learn on the stable input distributions. It does so by forcing activations of each layer to the mean value of 0 and variance value of 1. So, all that each deep network has to learn is gamma, and beta learning parameters, such that, these values have control on how the activation values for each layer of different networks are forced towards the mean 0 and variance 1. Hence, reducing the need to retrain the models for different covariate datasets, and rather using the pre-learned gamma and beta parameters of each layer for different networks. [1, 2].

**References:**

[1] Batch normalization: Accelerating deep network training by reducing internal covariate shift, by Ioffe, Sergey and Szegedy, Christian, year=2015

[2] http://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/