# LAB ASSIGNMENT 7

B20BB030

## QUESTION OVERVIEW

The main goal is to Train ResNet-18 on the FashionMNIST dataset and Use Slurm to submit a job for training the model on the GPU server [15

## METRICS

### ACCURACY

Accuracy is a useful metric because it provides a straightforward measure of how well a model is performing on a given task. However, it is important to note that accuracy may not be the best metric in all cases, as it does not take into account the relative costs of different types of errors.

For example, in the context of Fashion-MNIST, misclassifying a sandal as a sneaker may not be as costly as misclassifying a coat as a T-shirt. Other metrics such as precision, recall, and F1 score may be more appropriate in such cases.

Overall, accuracy is a useful metric for evaluating the performance of machine learning models on the Fashion-MNIST dataset, but it should be used in conjunction with other metrics to provide a more complete picture of the model's performance.

## METHODOLOGY

## TO TRAIN FASHION MNIST ON RESNET

### DATA PREPROCESSING:

First, you need to preprocess the Fashion-MNIST dataset. This includes resizing the images to a uniform size (e.g., 224x224), normalizing the pixel values to be between 0 and 1, and dividing the dataset into training and validation sets.

### MODEL ARCHITECTURE:

The ResNet model architecture is a deep convolutional neural network that has been proven effective for image classification tasks. You can use an existing ResNet architecture, such as ResNet-50 or ResNet-101, or you can create a custom ResNet architecture.

## INITIALIZATION:

You can initialize the weights of the ResNet model using a pre-trained model, such as ImageNet, to speed up the training process and improve the performance of the model.

## TRAINING:

You can train the ResNet model on the Fashion-MNIST dataset using a suitable optimizer such as stochastic gradient descent (SGD) or Adam. You will also need to define the loss function, such as categorical cross-entropy, and choose an appropriate learning rate for the optimizer.

## EVALUATION:

You can evaluate the performance of the trained ResNet model on the validation set using metrics such as accuracy, precision, recall, and F1 score. You can also visualize the performance of the model using plots of the loss and accuracy over time.

## HYPERPARAMETER TUNING:

You can fine-tune the hyperparameters of the ResNet model, such as the learning rate, batch size, and a number of epochs, to improve its performance on the validation set. This can be done using techniques such as grid search or random search.

## TESTING:

Finally, you can test the performance of the trained ResNet model on a separate test set to get a final measure of its performance. This can be done by loading the saved model weights, running the model on the test set, and then evaluating its performance using metrics such as accuracy.

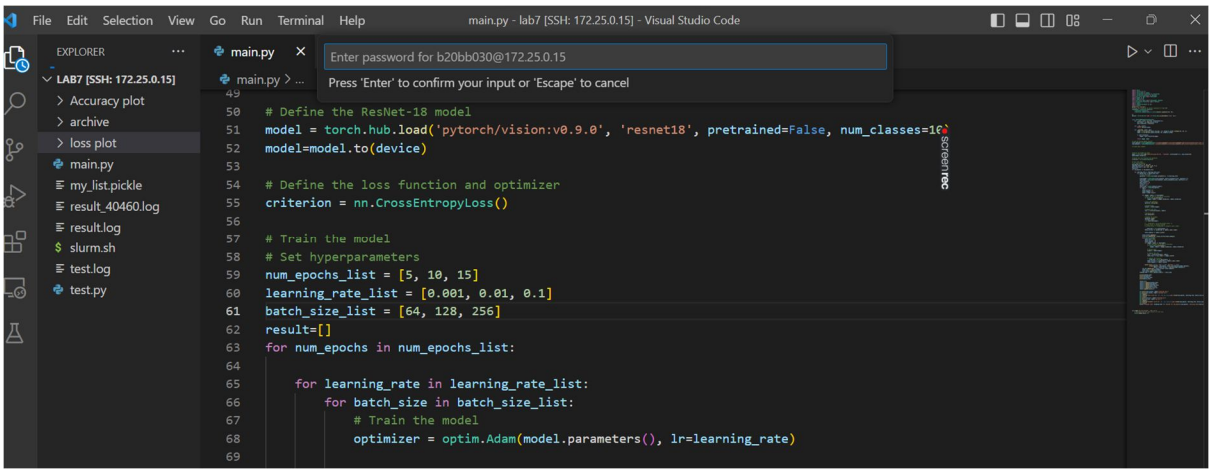
# TO USE SLURM TO SUBMIT A JOB FOR TRAINING THE MODEL ON THE GPU SERVER

## COPY FASHION MNIST DATA FROM LOCAL TO GPU SERVER USING SCP -R COMMAND

```
C:\Users\Ritam>scp -r C:\Users\Ritam\OneDrive\Documents\delete\archive b20bb030@172.25.0.15:~/scratch/data/b20bb030/lab7
b20bb030@172.25.0.15's password:
archive.zip                                                                                 100%   59MB 105.7MB/s   00:00
fashion-mnist_test.csv                                                                      100%   21MB 111.7MB/s   00:00
fashion-mnist_train.csv                                                                     100%  327MB 108.8MB/s   00:01
t10k-images-idx3-ubyte                                                                      100% 7656KB  88.8MB/s   00:00
t10k-labels-idx1-ubyte                                                                      100%   10KB   9.8KB/s   00:00
train-images-idx3-ubyte                                                                     100%   45MB 106.9MB/s   00:00
train-labels-idx1-ubyte                                                                     100%   59KB  58.6KB/s   00:00

C:\Users\Ritam>
```

## CONNECT TO THE HOST ON VSCODE BY TYPING THE PASSWORD




## GENERATE A SLURM.SH FILE AND A MAIN SCRIPT AS SHOWN

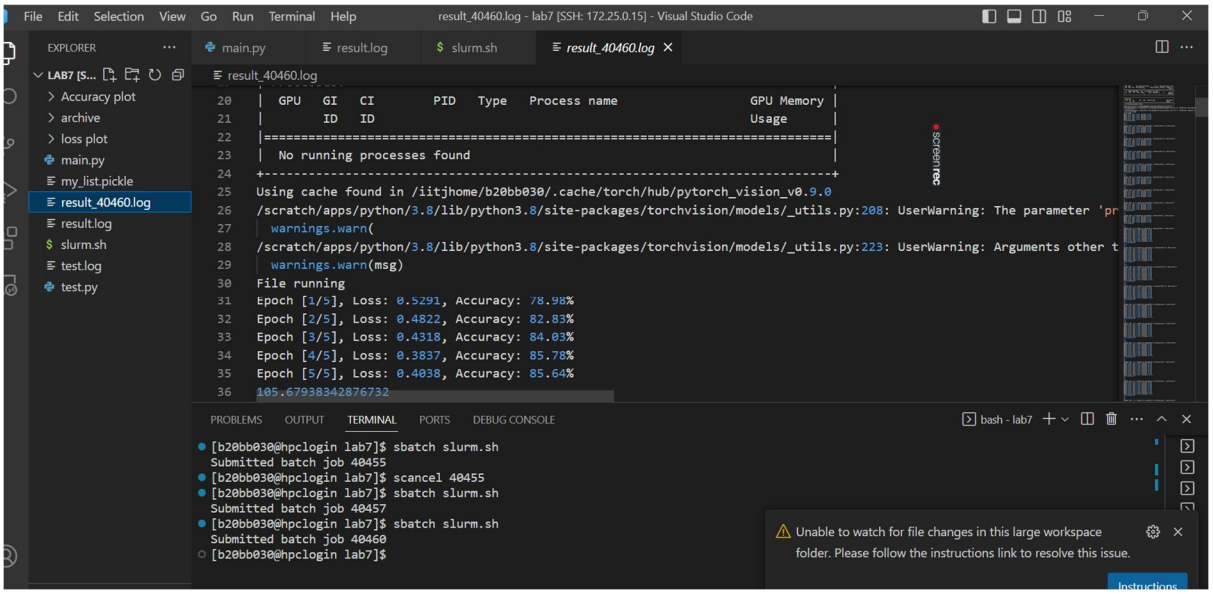
slurm.sh

```
#!/bin/bash
#SBATCH --job-name=test_job     # Job name
#SBATCH --partition=gpu2    #Partition name can be test/small/medium/large/gpu #Partition
#SBATCH --nodes=1           # Run all processes on a single node
#SBATCH --ntasks=1          # Run a single task
#SBATCH --cpus-per-task=4   # Number of CPU cores per task
#SBATCH --gres=gpu      # Include gpu for the task (only for GPU jobs)
#SBATCH --mem=6gb           # Total memory limit
#SBATCH --time=10:05:00     # Time limit hrs:min:sec
#SBATCH --output=result_%j.log # Standard output and error log

module load python/3.8

module load openmpi4
nvidia-smi
#cd /iitjhome/b20bb030/scratch/data/b20bb013
python3.8 main.py
```

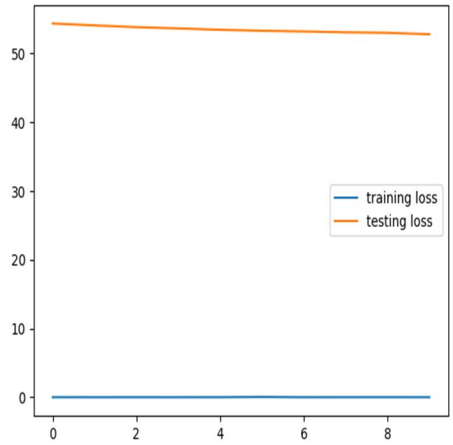
## SUBMIT THE BATCH JOB BY SBATCH SLURM.SH COMMAND AND OPEN THE RERULT.LOG



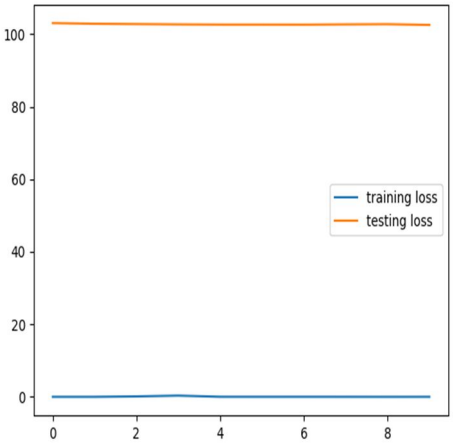
## RESULTS

In this Section, I have plotted all the loss curves and accuracy curves for the varying hyperparameters along with a table that summarizes the findings

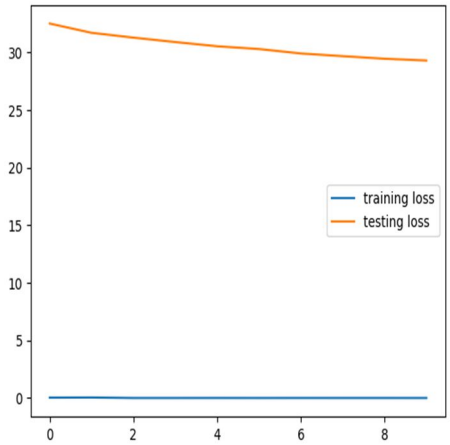
### LOSS CURVES [EPOCH NUMBER, LEARNING RATE, BATCH SIZE, ACCURACY, TIME ELAPSED IN CAPTION]



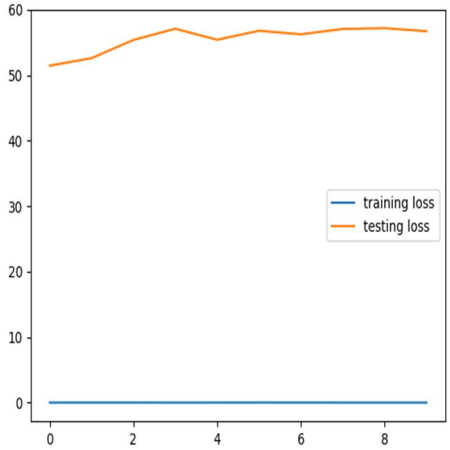

loss curve for 10 , 0.001, 128, 85.56666666666666,184.973436685279



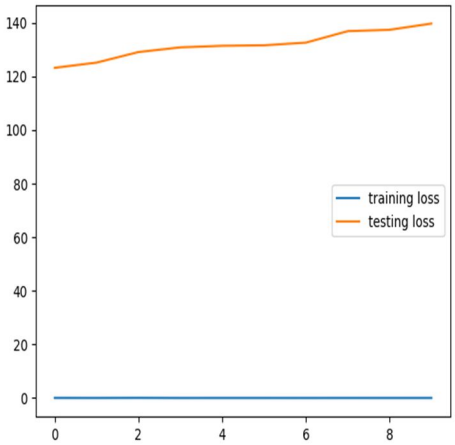

loss curve for 10 , 0.001, 256, 85.95166666666667,171.61322858929634



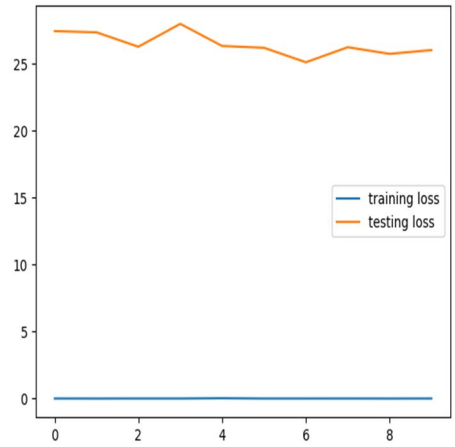

loss curve for 10 , 0.001, 64,
84.23833333333333,214.85168339684606



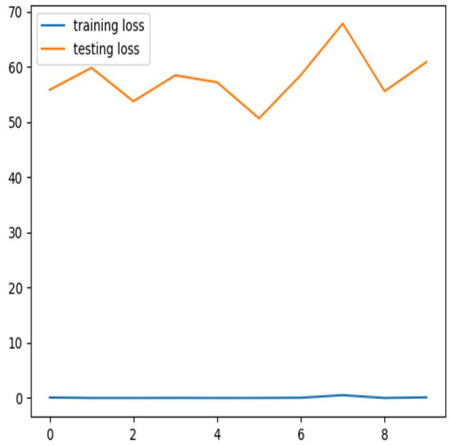

loss curve for 10 , 0.01, 128,
87.43333333333334,183.06576551962644



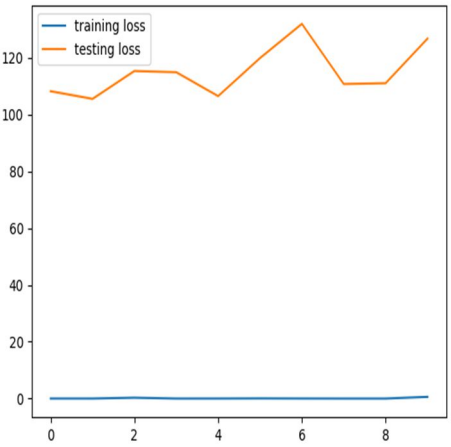

loss curve for 10 , 0.01, 256, 87.41,171.71338925231248



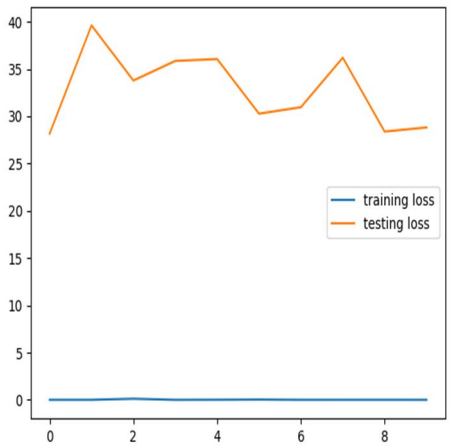

loss curve for 10 , 0.01, 64, 86.9,208.79939623270184



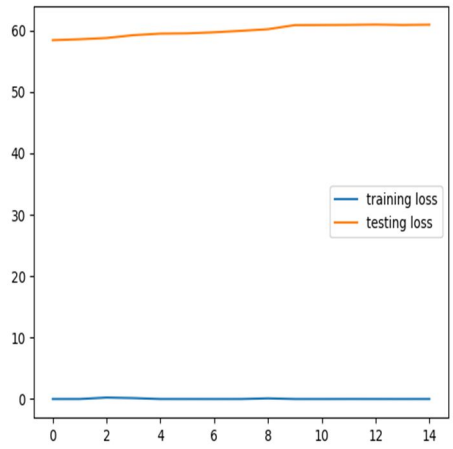

loss curve for  10 , 0.1, 128, 85.66666666666667,189.7662300132215



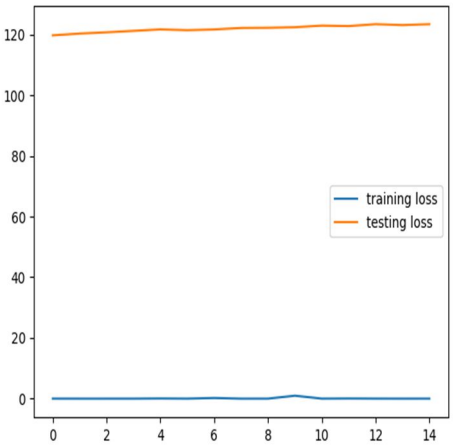

loss curve for  10 , 0.1, 256, 87.34666666666666,175.1224684547633



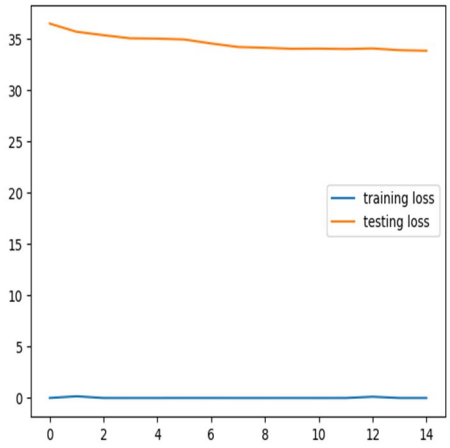
loss curve for  10 , 0.1, 64, 85.50166666666667,215.26309788413346



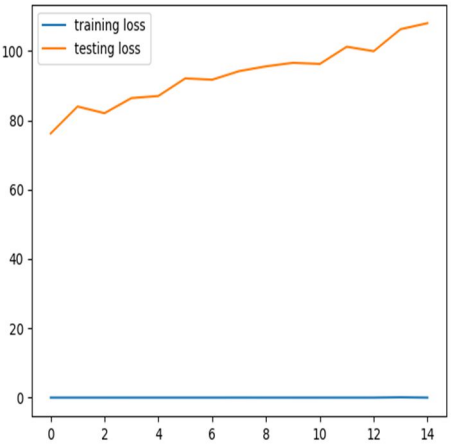

loss curve for  15 , 0.001, 128,
88.05166666666666,266.960066286847



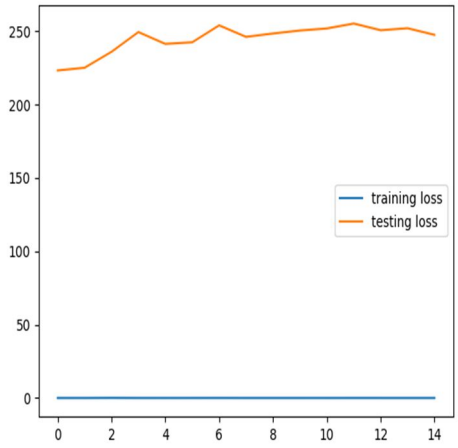

loss curve for  15 , 0.001, 256,
88.16166666666666,250.16800520848483



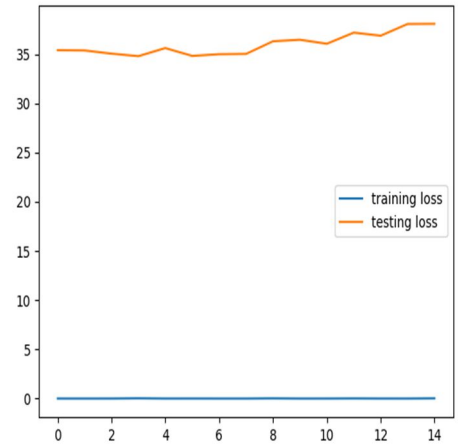

loss curve for 15 , 0.001, 64,
87.68666666666667,302.57188345771283



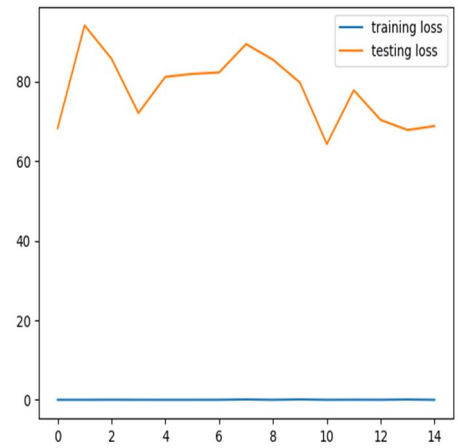

loss curve for 15 , 0.01, 128,
87.88333333333334,271.27991808764637



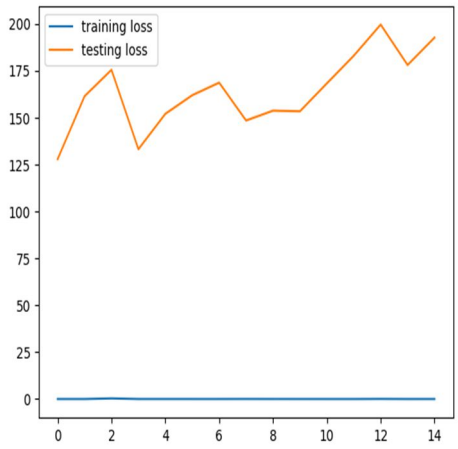

loss curve for  15 , 0.01, 256, 87.94,251.83222610410303



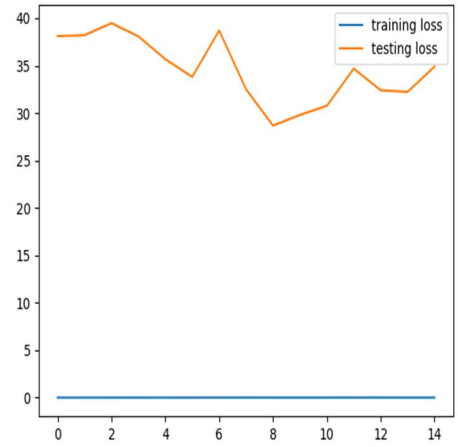

loss curve for  15 , 0.01, 64, 87.72333333333333,301.6737953508273



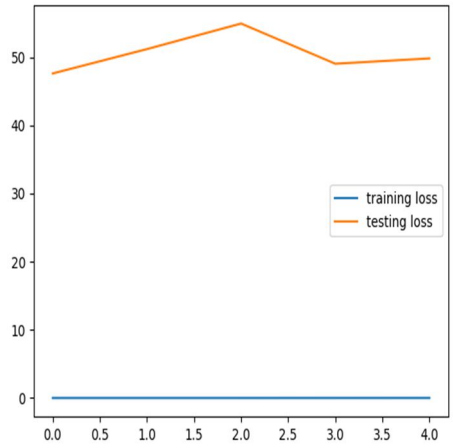
loss curve for  15 , 0.1, 128, 87.095,268.8499610843137



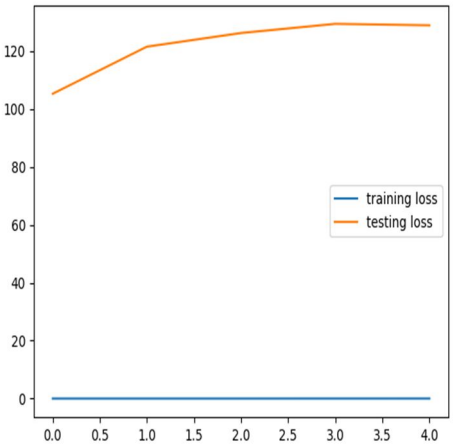

loss curve for  15 , 0.1, 256, 87.305,250.71896273549646



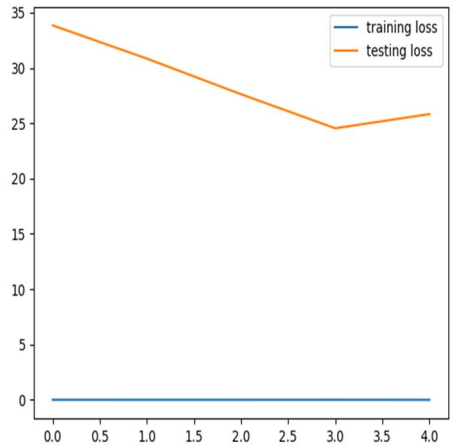

loss curve for  15 , 0.1, 64, 86.57833333333333,303.23279016278684



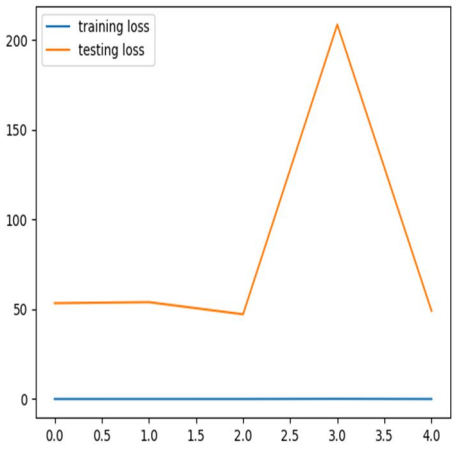

loss curve for  5 , 0.001, 128, 87.54,94.45059104356915



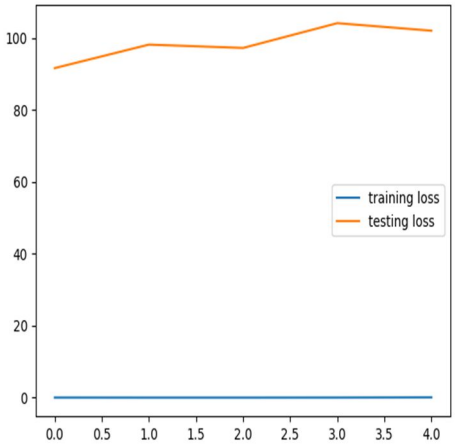

loss curve for  5, 0.001, 256, 87.45833333333333,85.90150786098093



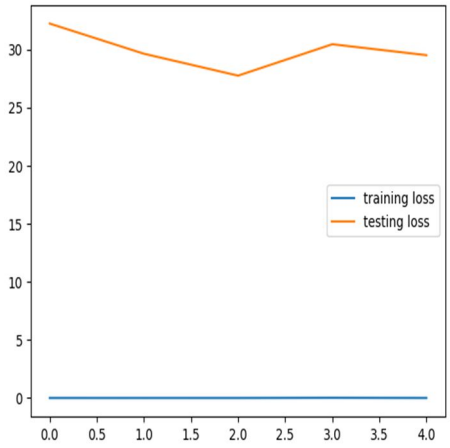
loss curve for  5, 0.001, 64, 85.63833333333334,105.67938342876732



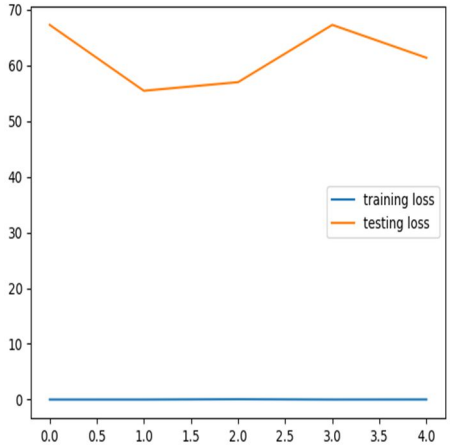

loss curve for  5, 0.01, 128, 86.265,91.20869947131723



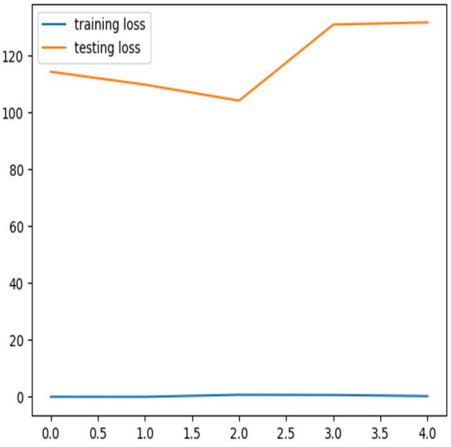

loss curve for  5, 0.01, 256, 86.90333333333334,85.85660909023136



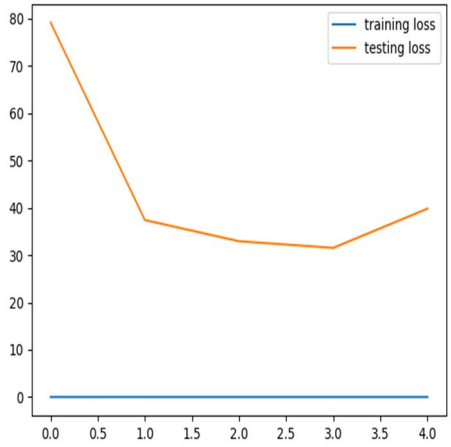

loss curve for  5, 0.01, 64, 83.21666666666667,106.19132073782384




loss curve for  5, 0.1, 128, 83.93333333333334,91.82626716885716

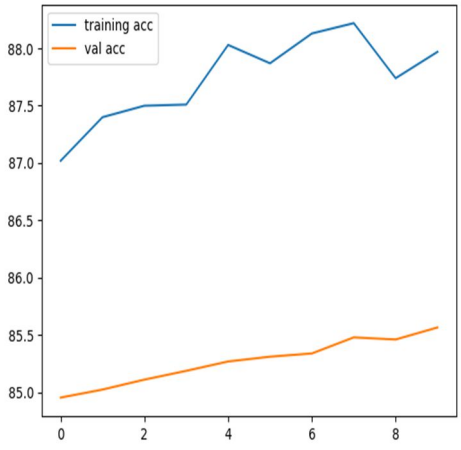


loss curve for  5, 0.1, 256, 81.855,85.9542280882597

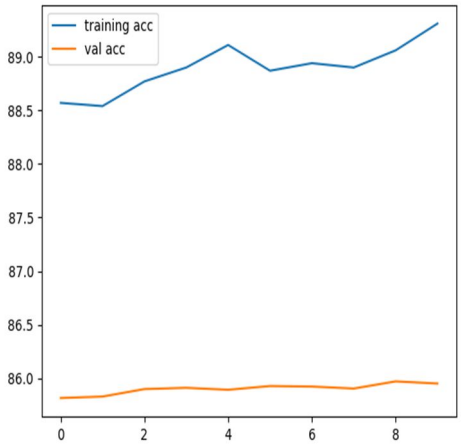



loss curve for  5, 0.1, 64, 78.30333333333333,107.80648355931044

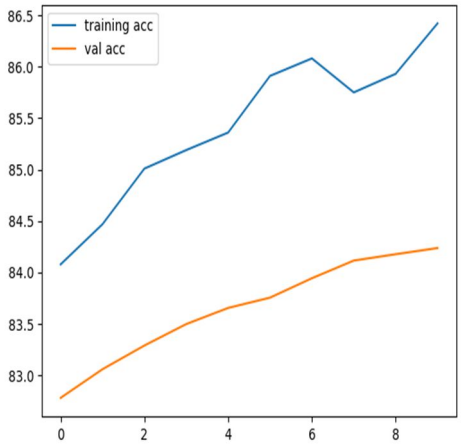
ACCURACY CURVES [EPOCH NUMBER, LEARNING RATE, BATCH SIZE, ACCURACY, TIME ELAPSED IN CAPTION]



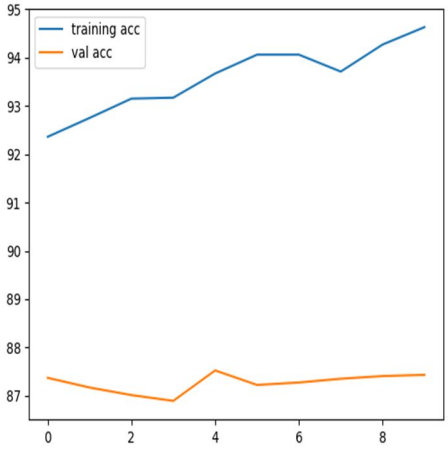

ACCURACY curve for  10, 0.001, 128,85.56666666666666.184.973436685279



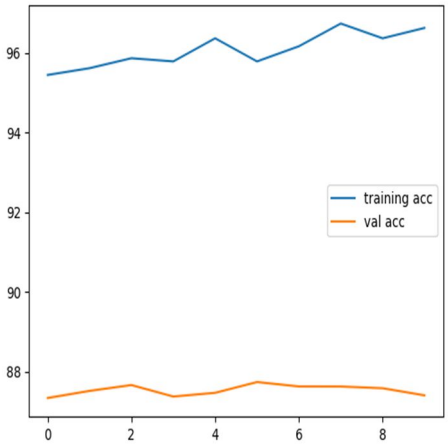

ACCURACY curve for  10, 0.001, 256,85.95166666666667.171.61322858929634



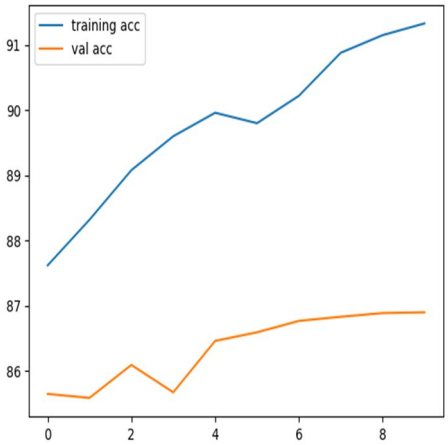

ACCURACY curve for  10 , 0.001, 64,84.23833333333333.214.85168339684606



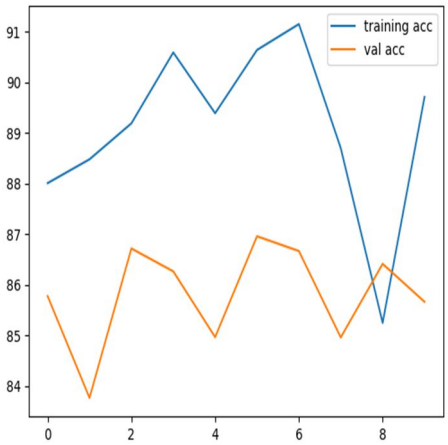

ACCURACY curve for  10 , 0.01, 128,87.43333333333334.183.06576551962644



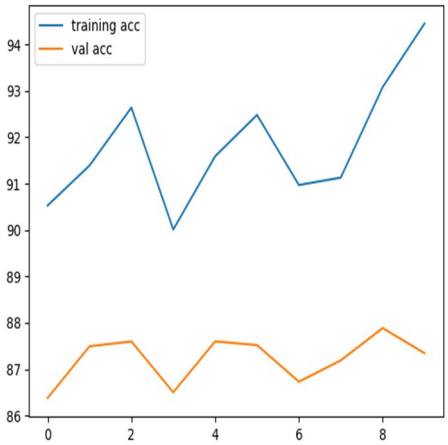

ACCURACY curve for  10 , 0.01, 256,87.41.171.71338925231248



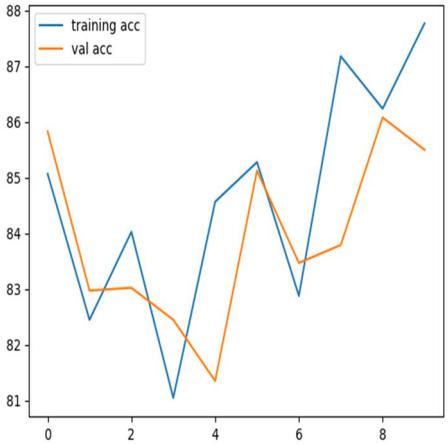

ACCURACY curve for  10 , 0.01, 64,86.9.208.79939623270184



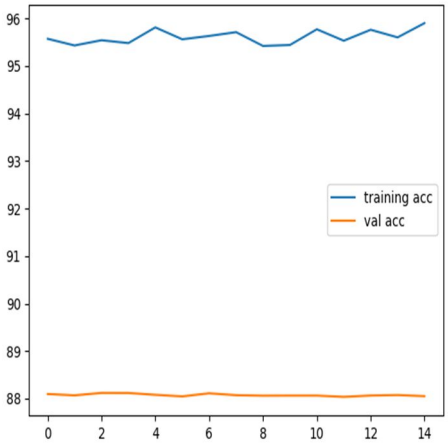

ACCURACY curve for  10 , 0.1, 128,85.66666666666667.189.7662300132215



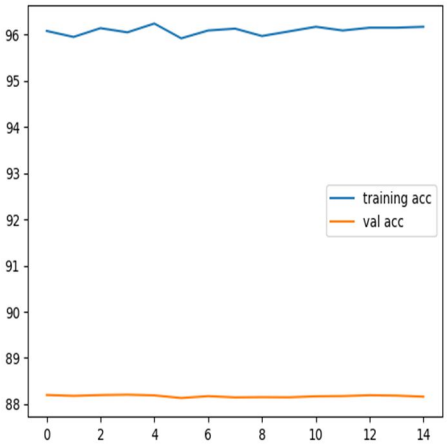

ACCURACY curve for  10 , 0.1, 256,87.34666666666666.175.1224684547633



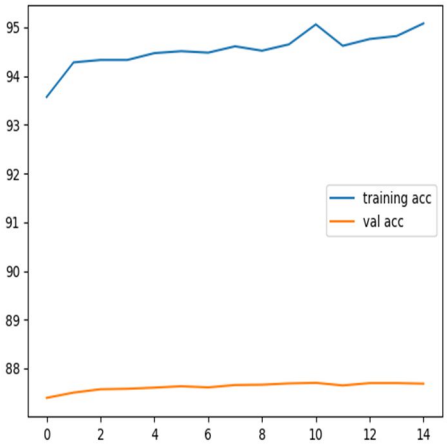

ACCURACY curve for  10 , 0.1, 64,85.50166666666667.215.26309788413346



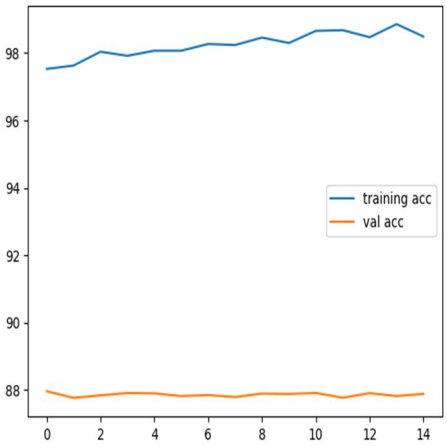

ACCURACY curve for  15 , 0.001, 128,88.05166666666666.266.960066286847



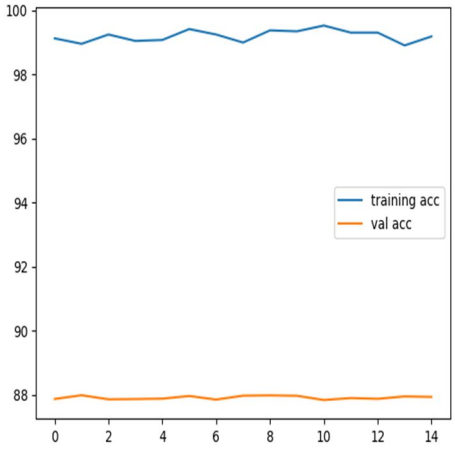

ACCURACY curve for  15 , 0.001, 256,88.16166666666666.250.16800520848483



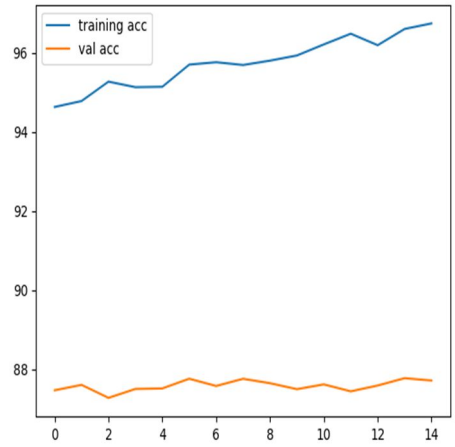

ACCURACY curve for  15 , 0.001, 64,87.68666666666667.302.57188345771283



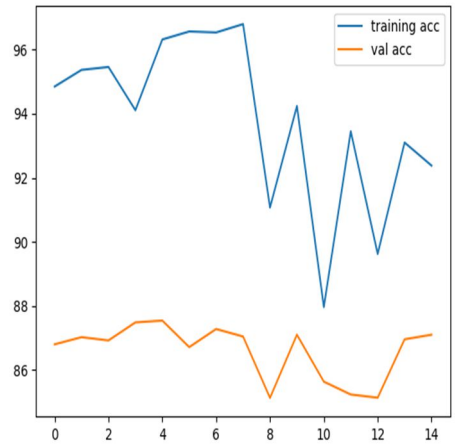

ACCURACY curve for  15 , 0.01, 128,87.88333333333334.271.27991808764637



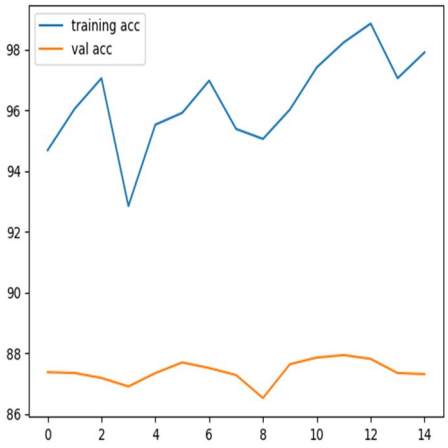

ACCURACY curve for  15 , 0.01, 256,87.94.251.83222610410303




ACCURACY curve for  15 , 0.01, 64,87.72333333333333.301.6737953508273

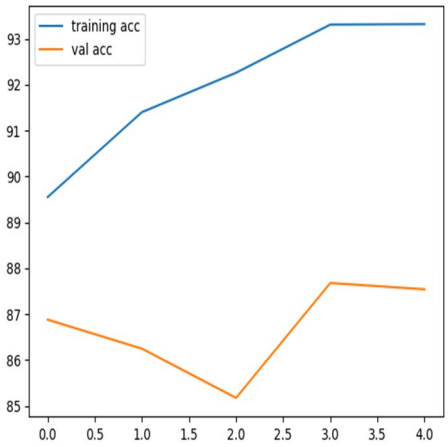



ACCURACY curve for  15 , 0.1, 128,87.095.268.8499610843137

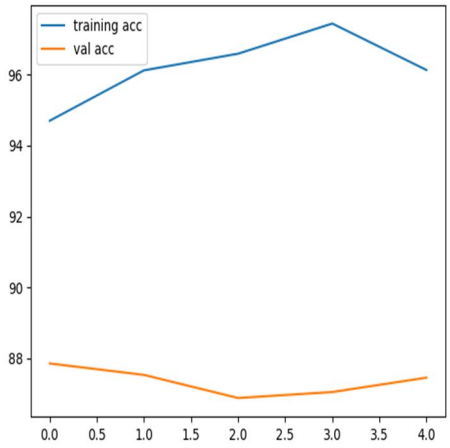



ACCURACY curve for  15 , 0.1, 256,87.305.250.71896273549646

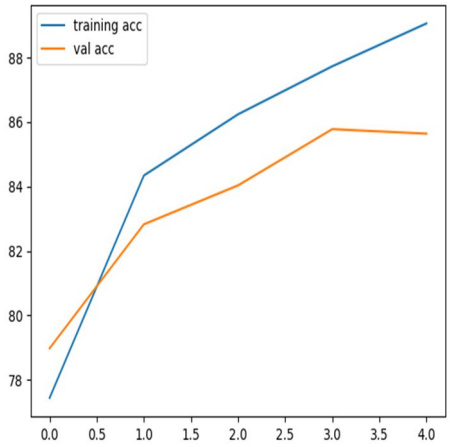

ACCURACY curve for  15 , 0.1, 64,86.57833333333333.303.23279016278684



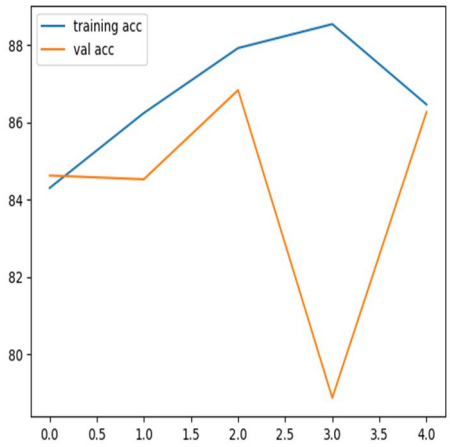

ACCURACY curve for  5 , 0.001, 128,87.54.94.45059104356915



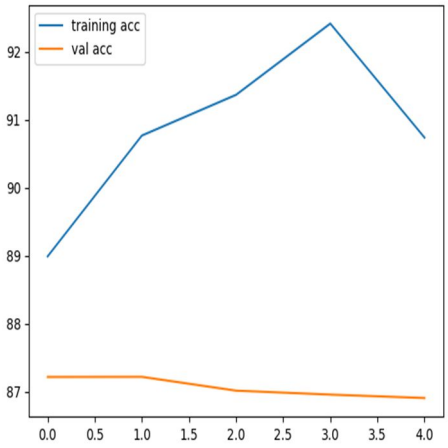

ACCURACY curve for  5 , 0.001, 256,87.45833333333333.85.90150786098093



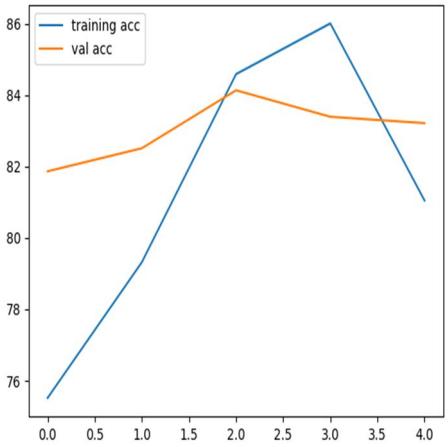

ACCURACY curve for  5 , 0.001, 64,85.63833333333334.105.67938342876732



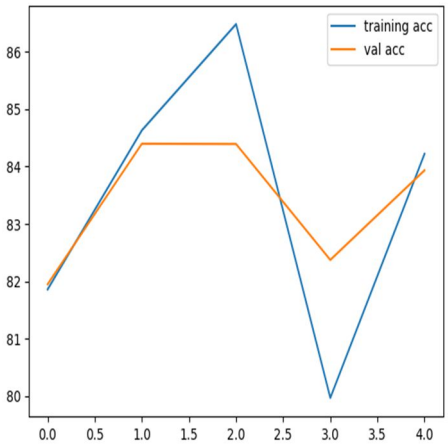

ACCURACY curve for  5 , 0.01, 128,86.265.91.20869947131723



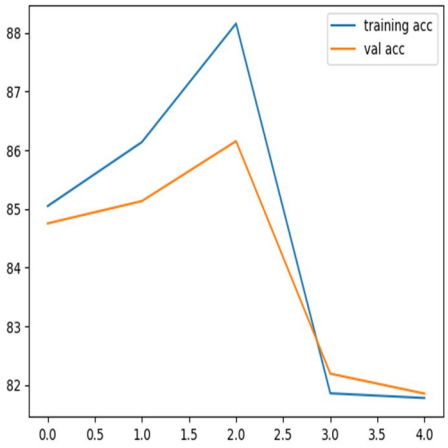

ACCURACY curve for  5 , 0.01, 256,86.90333333333334.85.85660909023136



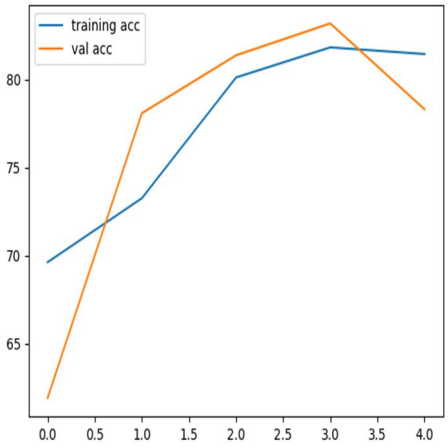

ACCURACY curve for  5 , 0.01, 64,83.21666666666667.106.19132073782384




ACCURACY curve for  5 , 0.1, 128,83.93333333333334.91.82626716885716




ACCURACY curve for 5 , 0.1, 256,81.855.85.9542280882597




ACCURACY curve for 5 , 0.1, 64,78.30333333333333.107.80648355931044

## SUMMARY OF RESULTS IN TABULAR FORM

| TIMEELAPSED(SECONDS) | EPOCH | LEARNING RATE | BATCH SIZE | ACCURACY |
|---|---|---|---|---|
| 105.679383 | 5 | 0.001 | 64 | 85.638333 |
| **94.450591** | **5** | **0.001** | **128** | **87.540000** |
| **85.901508** | **5** | **0.001** | **256** | **87.458333** |
| 106.191321 | 5 | 0.010 | 64 | 83.216667 |

| TIMEELAPSED(SECONDS) | EPOCH | LEARNING RATE | BATCH SIZE | ACCURACY |
|---|---|---|---|---|
| 91.208699 | 5 | 0.010 | 128 | 86.265000 |
| 85.856609 | 5 | 0.010 | 256 | 86.903333 |
| 107.806484 | 5 | 0.100 | 64 | 78.303333 |
| 91.826267 | 5 | 0.100 | 128 | 83.933333 |
| 85.954228 | 5 | 0.100 | 256 | 81.855000 |
| 214.851683 | 10 | 0.001 | 64 | 84.238333 |
| 184.973437 | 10 | 0.001 | 128 | 85.566667 |
| 171.613229 | 10 | 0.001 | 256 | 85.951667 |
| 208.799396 | 10 | 0.010 | 64 | 86.900000 |
| **183.065766** | **10** | **0.010** | **128** | **87.433333** |
| **171.713389** | **10** | **0.010** | **256** | **87.410000** |
| 215.263098 | 10 | 0.100 | 64 | 85.501667 |
| 189.766230 | 10 | 0.100 | 128 | 85.666667 |
| 175.122468 | 10 | 0.100 | 256 | 87.346667 |
| 302.571883 | 15 | 0.001 | 64 | 87.686667 |
| **266.960066** | **15** | **0.001** | **128** | **88.051667** |
| **250.168005** | **15** | **0.001** | **256** | **88.161667** |
| 301.673795 | 15 | 0.010 | 64 | 87.723333 |
| 271.279918 | 15 | 0.010 | 128 | 87.883333 |
| 251.832226 | 15 | 0.010 | 256 | 87.940000 |

| TIMEELAPSED(SECONDS) | EPOCH | LEARNING RATE | BATCH SIZE | ACCURACY |
|---|---|---|---|---|
| 303.232790 | 15 | 0.100 | 64 | 86.578333 |
| 268.849961 | 15 | 0.100 | 128 | 87.095000 |
| 250.718963 | 15 | 0.100 | 256 | 87.305000 |

***DISCUSSION OF THE RESULTS***

- Based on the highlighted results for epoch 5, , the model achieved an accuracy of around 87.5% (best)after 5 epochs with a learning rate of 0.001 and batch sizes of 128 and 256. The elapsed time for training the model with batch size 256 was lower than that of batch size 128., therefor a batch size of 256 seems to be optimal. Decreasing the batch size to 64 didn't improve the accuracy either.

- Similarly, in the case of 10 epochs, a batch size of 256 gave comparable results to that of 128 with only a slight decrease in accuracy (87.433333 vs 87.410000) but the time elapsed reduced by approximately 12 seconds.

- Meanwhile increasing the batch size from 128 to 256 for 15 epochs actually increased the accuracy (by 0.11%) and even decreased the time elapsed by 16 seconds !
- Therefore it is concluded from these runs , that a batch size of 256 is optimal for my model , and for the following epochs, the optimal hyperparameters are given below:

| EPOCH | LEARNING RATE | BATCH SIZE |
|---|---|---|
| **5** | **0.001** | **256** |
| **10** | **0.010** | **256** |
| **15** | **0.001** | **256** |