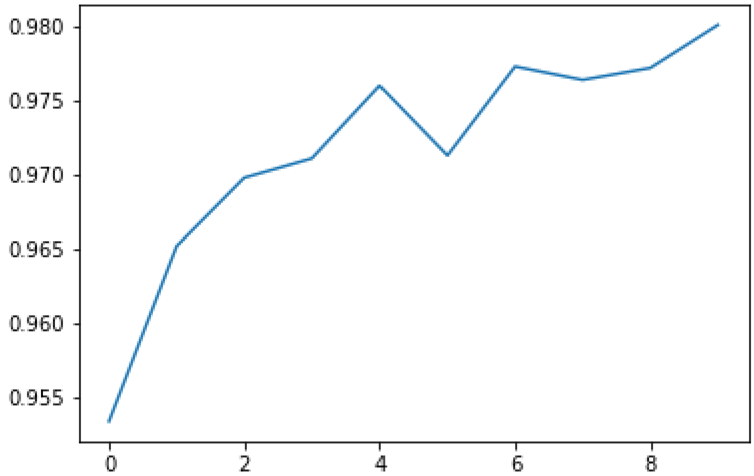
Report for question 3

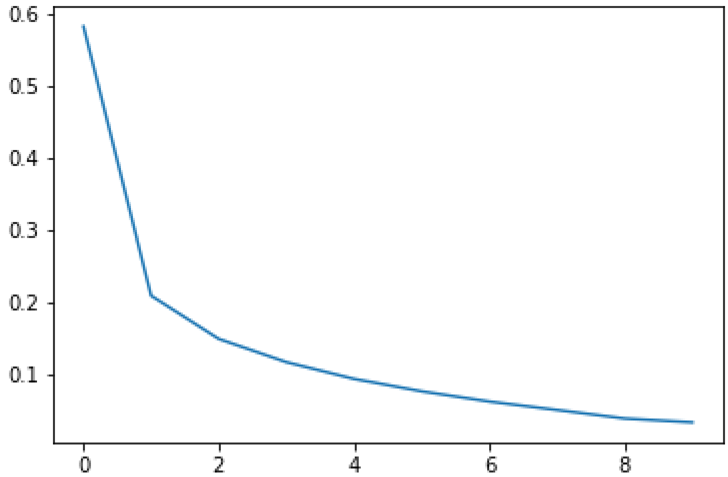
Learning Curves

MNIST classification

Accuracy

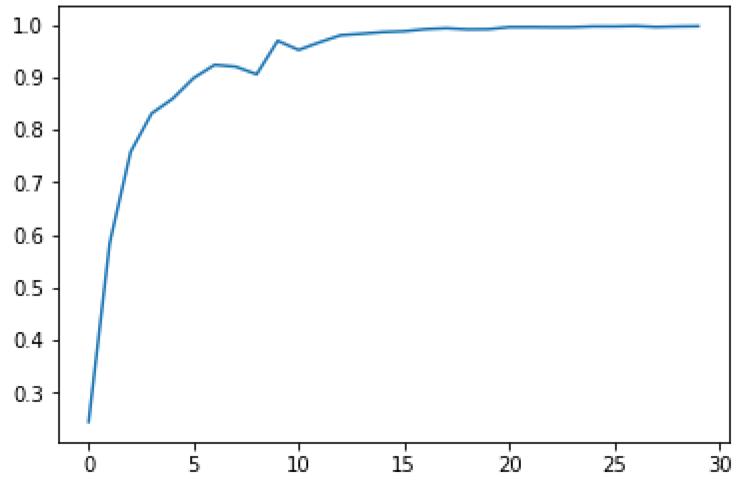


Loss

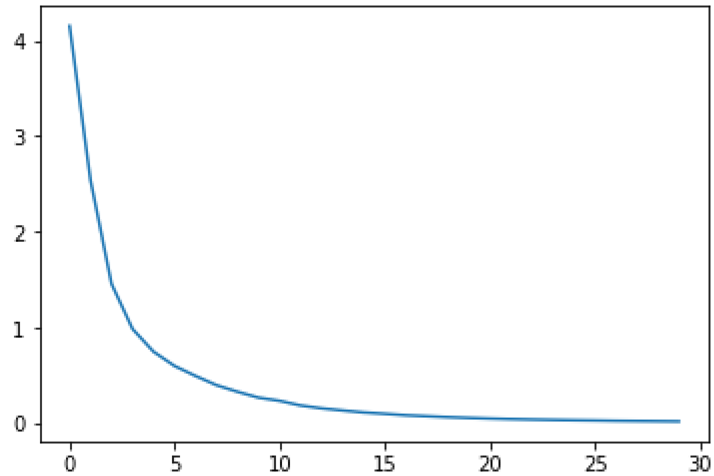


Custom Image classification

Accuracy



Loss



F-Scores

It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more. `F-score = (2 * Recall * Precision)/(Recall+Precision)`

Precision Out of all the classes, how much we predicted correctly. It should be high as possible. `Precision = TP/(TP+FP)` Recall Out of all the positive classes, how much we predicted correctly. It should be high as possible. Recall = TP/(TP+FP)

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

MNIST classification

`[0.98980632 0.99207746 0.98162476 0.97681302 0.97401936 0.9792018 0.97770701 0.98018366 0.97705252 0.9707196 ]`

Custom Image classification

```
[0.97098646 0.99474606 0.97217069 0.97902098 0.98540146 0.98194946
 1.         0.99805825 1.         1.         0.9982906  0.99424184
 0.99270073 0.98720293 0.98734177 0.98972603 0.98676749 0.98880597
 1.         0.99641577 0.96903461 0.98671727 0.95970696 0.99075786
 1.         1.         1.         1.         1.         1.
 0.99635036 1.         0.99652778 1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.        ]
```

Confusion Matrices

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).

MNIST classification

```
[[ 971    1    1    1    2    0    0    1    2    1]
 [   0 1127    3    1    0    0    0    1    3    0]
 [   1    1 1015    2    2    0    0    6    5    0]
 [   0    0    4  990    1    2    0    5    3    5]
 [   0    0    5    1  956    0    2    3    1   14]
 [   2    0    0    9    1  871    2    0    5    2]
 [   6    3    1    1   10    9  921    0    7    0]
 [   1    1    5    2    0    0    0 1014    2    3]
 [   0    1    2    5    1    1    1    2  958    3]
 [   1    3    0    5    8    4    0    9    1  978]]
```

Custom Image classification

It is of 96x96 size. So it is not possible to show it here. It is given as `confusion.npy`.

```
[[251   0   3 ...   0   0   0]
 [  0 284   0 ...   0   0   0]
```

```
 [  4   0 262 ...   0   0   0]
 ...
 [  0   0   0 ... 281   0   0]
 [  0   0   0 ...   0 251   0]
 [  0   0   0 ...   0   0 281]]
```

Variations tried

MNIST classification

We tried 2 variations:-

```
Learning_rate = 0.5
Epochs = 10
Batch_size = 10
Layers = 300 hidden
Acurr = 80.125 %
```

```
Learning_rate = 0.5
Epochs = 10
Batch_size = 100
Layers = 300 hidden
Acurr = 97.796 %
```

Custom Image classification

We tried 3 variations:-

```
Learning_rate = 0.5
Epochs = 10
Batch_size = 100
Layers = 50 hidden
Acurr = 73.125 %
```

```
Learning_rate = 0.7
Epochs = 20
Batch_size = 100
Layers = 100 hidden
Acurr = 93.125 %
```

```
Learning_rate = 0.7
Epochs = 30
```

```
Batch_size = 121
Layers = 50 hidden
Acurr = 99.69 %
```

Inferences

- As we increased the learning rate from 0.4 to 0.7 and increased the epoch from 10 to 30 the learning got increased from 70% to 99.6 %.