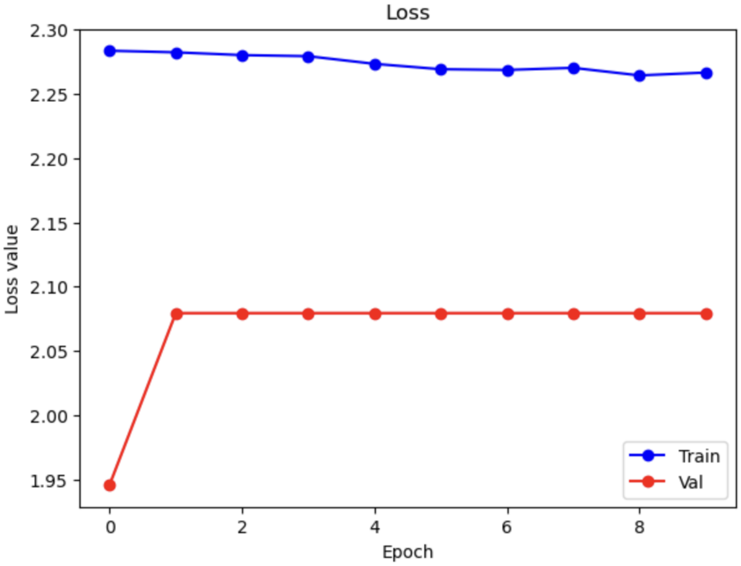
COL341 Assignment-4 Report

Amaiya Singhal (2021CS50598)

Part 1: Scratch Implementation

- In this part, I have implemented a Convolutional Neural Network consisting of the following layers:
  1. `CONV1:` Kernel size (3 × 3), In channels 3, Out channels 32.
  2. `POOL1:` Kernel size (2 × 2).
  3. `CONV2:` Kernel size (5 × 5), In channels 32, Out channels 64.
  4. `POOL2:` Kernel size (2 × 2)
  5. `CONV3:` Kernel size (3 × 3), In channels 64, Out channels 64.
  6. `POOL1:` Kernel size (2 × 2).
  7. `FC1:` Fully connected layer (also known as Linear layer) with 64 output neurons.
  8. `FC2:` Fully connected layer with 10 output neurons.
- I made separate classes for the `CONV, POOL` and `FC` layers which contain the forward and backward pass implementations for each of them.
- `ReLU` was used as the activation function for the `CONV` and the `FC1` layer.
- I used stochastic gradient descent with learning rate as 0.001 for training.
- Although I tried to optimise all the functions as much as possible, the program took a lot of time to run for each of the epochs.
- Hence I have trained my model (and made the respective observations) on a smaller set of 4800 images (480 images per class) and could run only 10 epochs.
- Due to time constraints I calculated accuracy and validation on a smaller set of 1000 test images instead of 10000.
- The learning rate of 0.001 was too small and did not lead to any noticeable improvement in the accuracy with Stochastic Gradient Descent even after 10 epochs.
- The training loss reduces but at a very slow rate not leading to any improvements in validation accuracy.

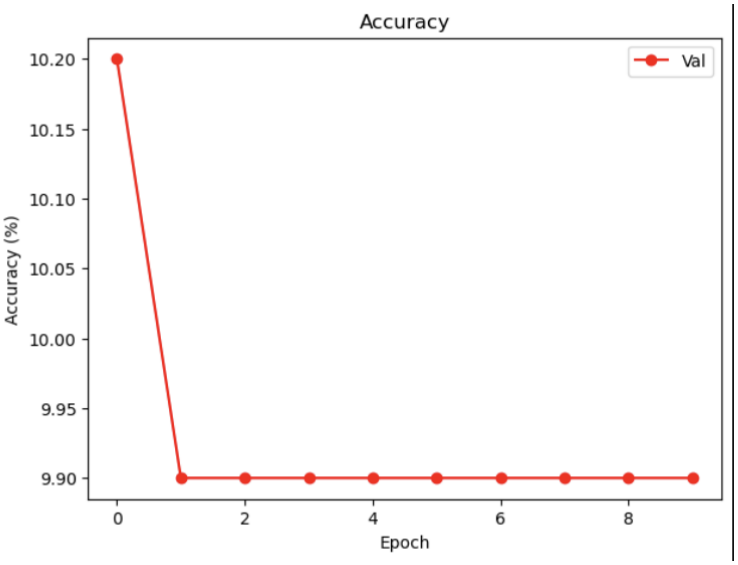
Training Epochs vs Training and Validation Losses



Variation with C

Epoch vs Overall Validation Accuracy

- The accuracy did not improve for the 10 epochs because the learning rate was just too slow to have any noticeable effect even after 10 epochs.
- The validation accuracy remains constant at around 10



Variation with C

Part 2: PyTorch Implementation

- In this part I used the `PyTorch` library for the CNN (same layers as Part 1).
- Multiple experiments were performed varying different sets of hyperparameters.
- Adam and SGD were used as the optimizers.
- Cross Entropy was taken as the loss function.
- Data Augmentation was done.

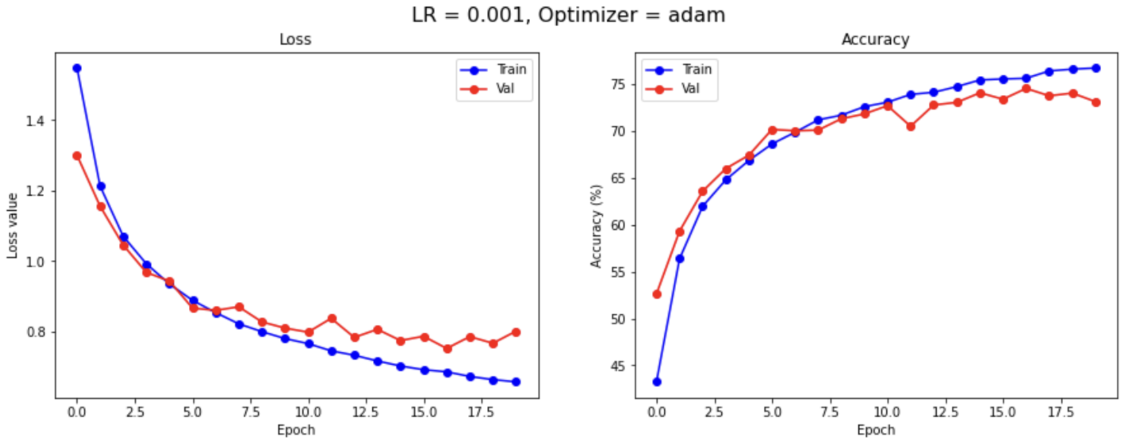
4.1 Hyper-Parameter Tuning

1) Learning Rate

LR= 0.001, Epochs= 20, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.548	0.432	1.300	0.526
2	1.213	0.564	1.156	0.593
3	1.071	0.620	1.046	0.636
4	0.993	0.648	0.968	0.660
5	0.935	0.669	0.944	0.674
6	0.889	0.686	0.867	0.702
7	0.854	0.699	0.860	0.700
8	0.822	0.712	0.871	0.701
9	0.800	0.717	0.828	0.713
10	0.781	0.726	0.810	0.718
11	0.766	0.731	0.799	0.727
12	0.746	0.739	0.838	0.705
13	0.733	0.741	0.784	0.728
14	0.717	0.748	0.807	0.731
15	0.703	0.755	0.775	0.741
16	0.692	0.756	0.787	0.734
17	0.686	0.756	0.753	0.746
18	0.673	0.764	0.786	0.738
19	0.664	0.766	0.767	0.741
20	0.658	0.767	0.802	0.732

Accuracy for class: plane	70.5 %
Accuracy for class: car	86.3 %
Accuracy for class: bird	57.6 %
Accuracy for class: cat	66.3 %
Accuracy for class: deer	70.7 %
Accuracy for class: dog	65.2 %
Accuracy for class: frog	79.4 %
Accuracy for class: horse	74.1 %
Accuracy for class: ship	77.7 %
Accuracy for class: truck	83.7 %



Variation with C

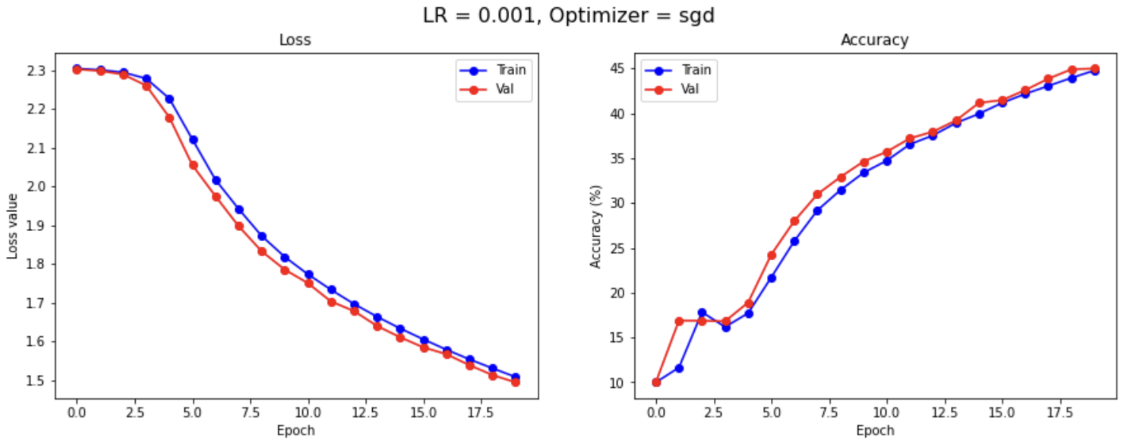
Observations

- Both train and validation losses go down with the validation loss almost taking a constant value.
- Both the train and validation accuracies go up with increase in the number of epoch.

LR= 0.001, Epochs= 20, Batch Size= 32, Optimizer= SGD

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	2.304	0.100	2.302	0.100
2	2.300	0.117	2.298	0.169
3	2.294	0.178	2.289	0.169
4	2.278	0.162	2.260	0.168
5	2.227	0.177	2.178	0.189
6	2.121	0.217	2.055	0.242
7	2.017	0.258	1.975	0.280
8	1.942	0.292	1.898	0.310
9	1.872	0.314	1.833	0.329
10	1.818	0.334	1.785	0.346
11	1.774	0.347	1.751	0.357
12	1.734	0.365	1.703	0.372
13	1.697	0.375	1.679	0.380
14	1.664	0.389	1.640	0.392
15	1.634	0.400	1.611	0.412
16	1.606	0.412	1.585	0.415
17	1.579	0.422	1.567	0.426
18	1.554	0.430	1.539	0.439
19	1.531	0.439	1.513	0.449
20	1.508	0.448	1.495	0.450

Accuracy for class: plane	48.2 %
Accuracy for class: car	68.8 %
Accuracy for class: bird	20.7 %
Accuracy for class: cat	17.7 %
Accuracy for class: deer	38.0 %
Accuracy for class: dog	48.3 %
Accuracy for class: frog	64.7 %
Accuracy for class: horse	54.9 %
Accuracy for class: ship	41.0 %
Accuracy for class: truck	47.6 %



Variation with C

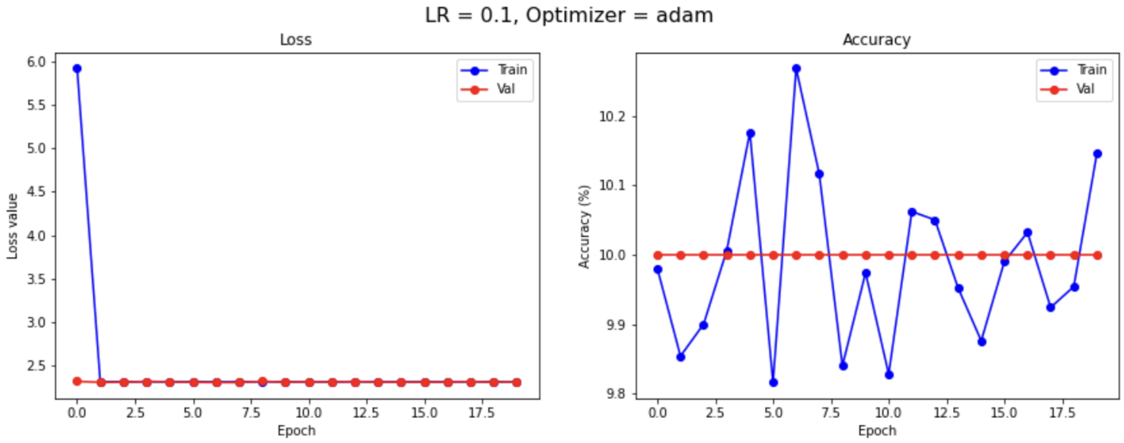
Observations

- The learning rate in this case is too slow for the stochastic gradient descent.
- Even after 20 epochs the loss is still high and the accuracy has also not yet saturated.
- We can try reducing the learning rate.

LR= 0.1, Epochs= 20, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	5.915	0.100	2.321	0.100
2	2.315	0.099	2.309	0.100
3	2.315	0.099	2.310	0.100
4	2.316	0.100	2.321	0.100
5	2.315	0.102	2.314	0.100
6	2.316	0.098	2.310	0.100
7	2.314	0.103	2.307	0.100
8	2.315	0.101	2.307	0.100
9	2.316	0.098	2.321	0.100
10	2.316	0.100	2.311	0.100
11	2.315	0.098	2.318	0.100
12	2.314	0.101	2.313	0.100
13	2.316	0.101	2.320	0.100
14	2.315	0.100	2.313	0.100
15	2.315	0.099	2.313	0.100
16	2.315	0.100	2.311	0.100
17	2.315	0.100	2.314	0.100
18	2.316	0.099	2.318	0.100
19	2.317	0.100	2.311	0.100
20	2.315	0.101	2.317	0.100

Accuracy for class: plane	0.0 %
Accuracy for class: car	0.0 %
Accuracy for class: bird	0.0 %
Accuracy for class: cat	0.0 %
Accuracy for class: deer	0.0 %
Accuracy for class: dog	0.0 %
Accuracy for class: frog	0.0 %
Accuracy for class: horse	0.0 %
Accuracy for class: ship	100.0%
Accuracy for class: truck	0.0 %



Variation with C

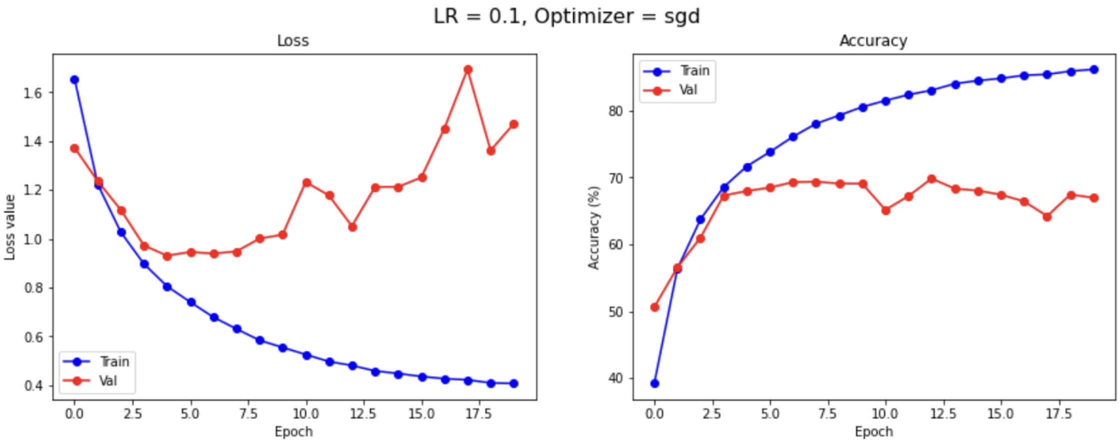
Observations

- The learning rate is too high in this case.
- The model is not able to learn anything in this case and predicts everything in the class 'ship'.
- We need to reduce the learning rate to allow the model to converge.

LR= 0.1, Epochs= 20, Batch Size= 32, Optimizer= SGD

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.655	0.392	1.374	0.506
2	1.223	0.563	1.236	0.566
3	1.027	0.638	1.119	0.610
4	0.897	0.686	0.973	0.673
5	0.806	0.717	0.931	0.680
6	0.741	0.738	0.946	0.685
7	0.680	0.761	0.940	0.693
8	0.631	0.781	0.948	0.694
9	0.585	0.793	1.001	0.691
10	0.555	0.806	1.017	0.691
11	0.526	0.815	1.233	0.652
12	0.498	0.824	1.178	0.672
13	0.481	0.831	1.052	0.699
14	0.459	0.841	1.212	0.683
15	0.449	0.845	1.213	0.680
16	0.437	0.849	1.251	0.674
17	0.427	0.853	1.452	0.664
18	0.423	0.855	1.694	0.642
19	0.410	0.859	1.362	0.675
20	0.408	0.862	1.471	0.670

Accuracy for class: plane	80.0 %
Accuracy for class: car	83.5 %
Accuracy for class: bird	37.2 %
Accuracy for class: cat	39.2 %
Accuracy for class: deer	62.1 %
Accuracy for class: dog	56.5 %
Accuracy for class: frog	90.3 %
Accuracy for class: horse	70.4 %
Accuracy for class: ship	78.7 %
Accuracy for class: truck	72.0 %



Loss and Accuracy Plot

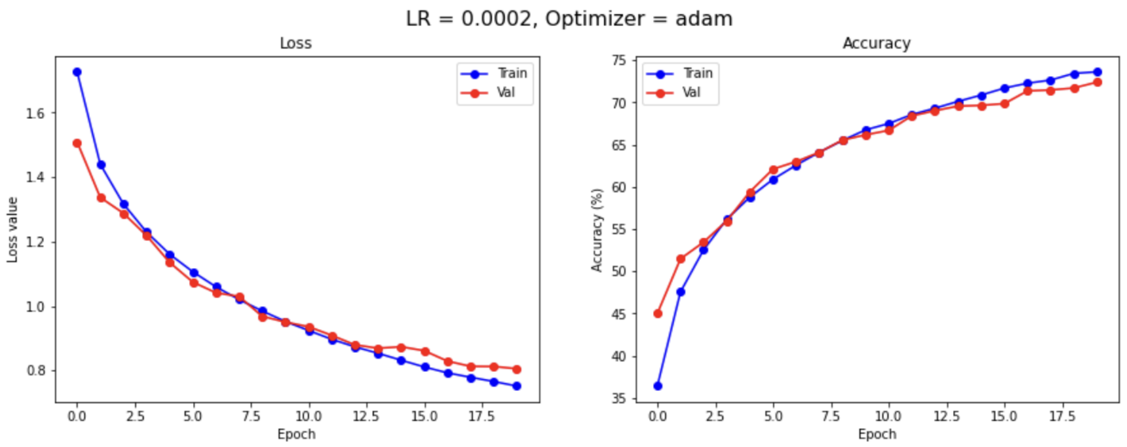
Observations

- As we had thought, increasing the Learning Rate led to better convergence of the model.
- Hence we can say Adam takes smaller learning rates and SGD requires slightly higher learning rates.

LR= 0.0002, Epochs= 20, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.727	0.364	1.509	0.450
2	1.440	0.476	1.337	0.515
3	1.317	0.525	1.288	0.534
4	1.230	0.562	1.219	0.560
5	1.161	0.588	1.135	0.594
6	1.106	0.609	1.074	0.621
7	1.060	0.626	1.041	0.630
8	1.020	0.641	1.030	0.641
9	0.985	0.655	0.968	0.655
10	0.952	0.667	0.950	0.662
11	0.924	0.675	0.936	0.667
12	0.897	0.686	0.909	0.684
13	0.873	0.693	0.880	0.690
14	0.853	0.701	0.869	0.696
15	0.832	0.709	0.873	0.697
16	0.811	0.717	0.861	0.699
17	0.793	0.723	0.829	0.714
18	0.779	0.727	0.813	0.715
19	0.766	0.734	0.812	0.717
20	0.752	0.736	0.805	0.724

Accuracy for class: plane	80.5 %
Accuracy for class: car	86.8 %
Accuracy for class: bird	60.2 %
Accuracy for class: cat	42.4 %
Accuracy for class: deer	62.7 %
Accuracy for class: dog	70.3 %
Accuracy for class: frog	78.6 %
Accuracy for class: horse	78.1 %
Accuracy for class: ship	81.9 %
Accuracy for class: truck	82.5 %



Loss and Accuracy Plot

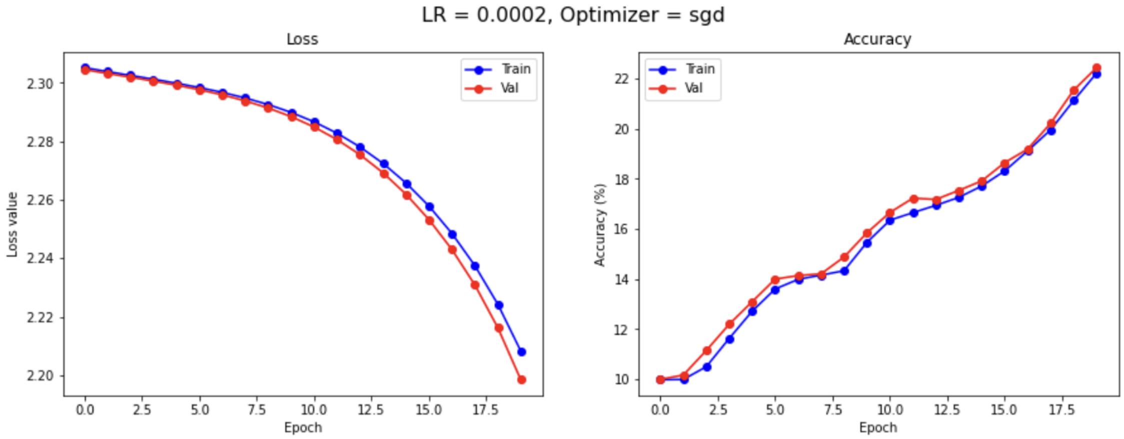
Observations

- Even though the model trains well, the learning rate is very small and hence the updates are slow.
- Both the validation and training losses are still decreasing.
- Higher learning rate would be better.

LR= 0.0002, Epochs= 20, Batch Size= 32, Optimizer= SGD

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	2.305	0.100	2.304	0.100
2	2.304	0.100	2.303	0.102
3	2.303	0.105	2.302	0.112
4	2.301	0.116	2.301	0.122
5	2.300	0.127	2.299	0.131
6	2.298	0.136	2.298	0.140
7	2.297	0.140	2.296	0.141
8	2.295	0.142	2.294	0.142
9	2.293	0.143	2.291	0.149
10	2.290	0.155	2.288	0.159
11	2.287	0.164	2.285	0.167
12	2.283	0.167	2.281	0.172
13	2.278	0.169	2.275	0.172
14	2.272	0.173	2.269	0.175
15	2.266	0.177	2.262	0.179
16	2.258	0.183	2.253	0.186
17	2.248	0.191	2.243	0.192
18	2.237	0.200	2.231	0.202
19	2.224	0.211	2.216	0.215
20	2.208	0.222	2.198	0.224

Accuracy for class: plane	11.0 %
Accuracy for class: car	21.2 %
Accuracy for class: bird	0.0 %
Accuracy for class: cat	15.4 %
Accuracy for class: deer	0.0 %
Accuracy for class: dog	1.4 %
Accuracy for class: frog	32.0 %
Accuracy for class: horse	52.7 %
Accuracy for class: ship	44.7 %
Accuracy for class: truck	46.0 %



Loss and Accuracy Plot

Observations

- As we had observed for the case of learning rate = 0.001, the learning rate of 0.0002 is very low.
- The updates are very slow and there is not a significant increase in the accuracy even after 20 epochs.

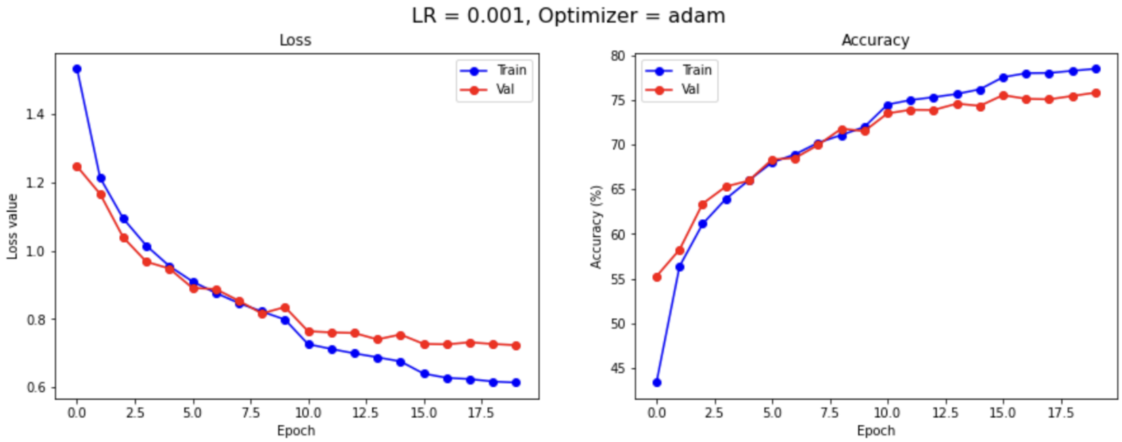
2) Variation in LR

- I have chosen the learning rate scheduler to be `MultiStepLR` with the following structure:
- scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[10, 15, 20], gamma = 0.5)
- The starting LR is chosen to be 0.001 for Adam and 0.1 for SGD
- Based on experiments in part 1, it looks like the val loss stops decreasing after epoch 10, so here I will try to reduce LR from there after every 5 epochs.
- Batch size is taken to be 32.

LR= 0.001, Epochs= 20, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc	Learning Rate
1	1.533	0.434	1.247	0.553	0.001
2	1.214	0.564	1.166	0.583	0.001
3	1.094	0.611	1.039	0.634	0.001
4	1.015	0.639	0.968	0.653	0.001
5	0.955	0.660	0.948	0.660	0.001
6	0.910	0.680	0.891	0.683	0.001
7	0.876	0.689	0.888	0.685	0.001
8	0.847	0.702	0.854	0.700	0.001
9	0.822	0.711	0.816	0.718	0.001
10	0.799	0.720	0.835	0.715	0.0005
11	0.727	0.745	0.764	0.735	0.0005
12	0.712	0.750	0.761	0.739	0.0005
13	0.700	0.753	0.759	0.739	0.0005
14	0.688	0.757	0.740	0.746	0.0005
15	0.676	0.762	0.755	0.743	0.00025
16	0.640	0.775	0.727	0.755	0.00025
17	0.628	0.780	0.726	0.751	0.00025
18	0.624	0.780	0.732	0.751	0.00025
19	0.617	0.783	0.727	0.754	0.00025
20	0.614	0.785	0.724	0.758	0.000125

Accuracy for class: plane	78.1 %
Accuracy for class: car	88.8 %
Accuracy for class: bird	66.8 %
Accuracy for class: cat	53.9 %
Accuracy for class: deer	74.8 %
Accuracy for class: dog	69.8 %
Accuracy for class: frog	82.5 %
Accuracy for class: horse	77.2 %
Accuracy for class: ship	85.5 %
Accuracy for class: truck	80.8 %



Loss and Accuracy Plot

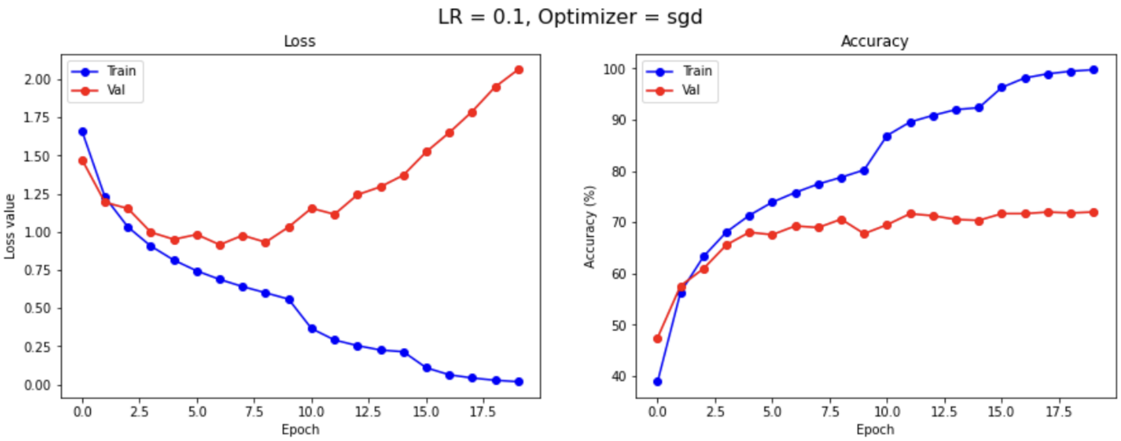
Observations

- This performs better than the constant learning rate case.
- This is because the learning rates are now being adjusted and lowered when the optimisation function reaches its minima.
- The accuracy obtained = 75.8% is the highest so far.

LR= 0.1, Epochs= 20, Batch Size= 32, Optimizer= SGD

Epoch	Train Loss	Train Acc	Val Loss	Val Acc	Learning Rate
1	1.660	0.389	1.473	0.475	0.1
2	1.228	0.564	1.194	0.576	0.1
3	1.031	0.633	1.153	0.610	0.1
4	0.907	0.681	0.997	0.656	0.1
5	0.815	0.714	0.951	0.681	0.1
6	0.744	0.739	0.983	0.676	0.1
7	0.688	0.758	0.916	0.693	0.1
8	0.641	0.775	0.977	0.690	0.1
9	0.601	0.788	0.932	0.706	0.1
10	0.559	0.802	1.031	0.678	0.05
11	0.367	0.869	1.155	0.695	0.05
12	0.292	0.896	1.115	0.717	0.05
13	0.255	0.909	1.243	0.713	0.05
14	0.226	0.920	1.295	0.706	0.05
15	0.215	0.924	1.371	0.704	0.025
16	0.110	0.963	1.524	0.717	0.025
17	0.064	0.982	1.650	0.717	0.025
18	0.043	0.990	1.788	0.720	0.025
19	0.028	0.995	1.949	0.718	0.025
20	0.018	0.998	2.063	0.720	0.0125

Accuracy for class: plane	75.6 %
Accuracy for class: car	84.0 %
Accuracy for class: bird	59.0 %
Accuracy for class: cat	53.5 %
Accuracy for class: deer	67.0 %
Accuracy for class: dog	64.4 %
Accuracy for class: frog	79.5 %
Accuracy for class: horse	74.3 %
Accuracy for class: ship	81.5 %
Accuracy for class: truck	81.4 %



Loss and Accuracy Plot

Observations

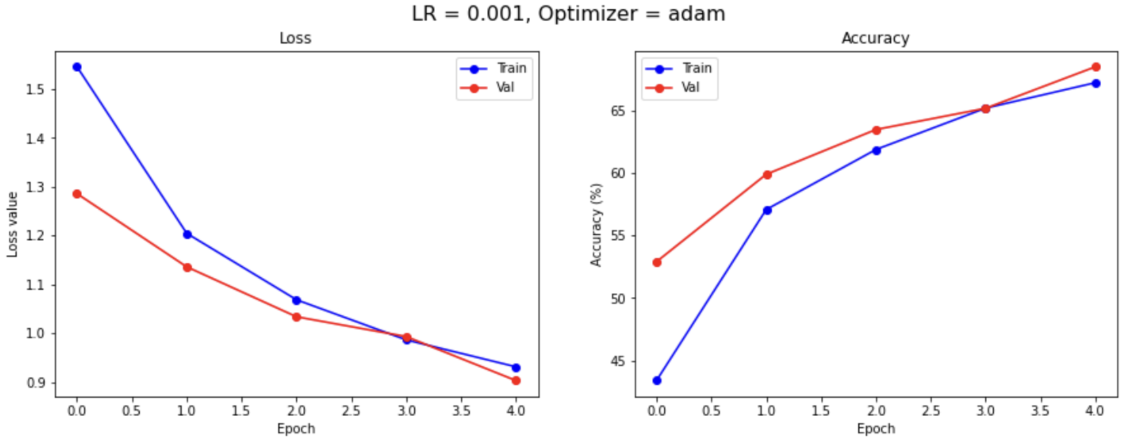
- Starting with a learning rate of 0.1 we are able to train the model well.
- Very high train accuracy is obtained indicating that overfitting has taken place.
- The results were better when we started with the learning rate = 0.001.

3) Number of Training Epochs

- Learning Rate is chosen to be 0.001
- Batch Size is 32

LR= 0.001, Epochs= 5, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.546	0.434	1.286	0.529
2	1.205	0.571	1.136	0.599
3	1.069	0.619	1.034	0.635
4	0.987	0.652	0.993	0.652
5	0.931	0.672	0.903	0.685



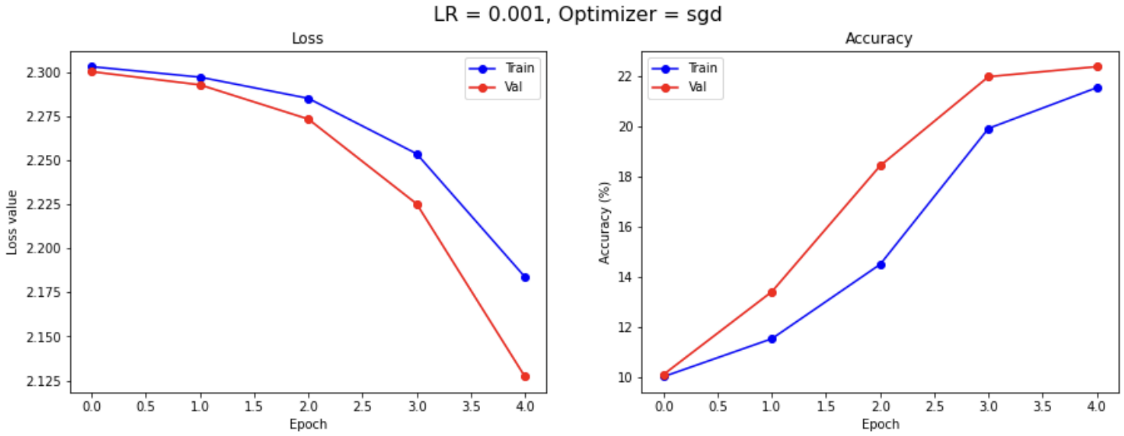
Loss and Accuracy Plot

Observations

- 5 epochs are too less to train the model enough.
- As we had observed earlier, 20 epochs was sort of the ideal number where we achieve a well trained network.
- The losses are still decreasing and accuracies still increasing in this case.

LR= 0.001, Epochs= 5, Batch Size= 32, Optimizer= SGD

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	2.303	0.100	2.300	0.101
2	2.297	0.115	2.293	0.134
3	2.285	0.145	2.273	0.184
4	2.254	0.199	2.225	0.220
5	2.184	0.215	2.127	0.224



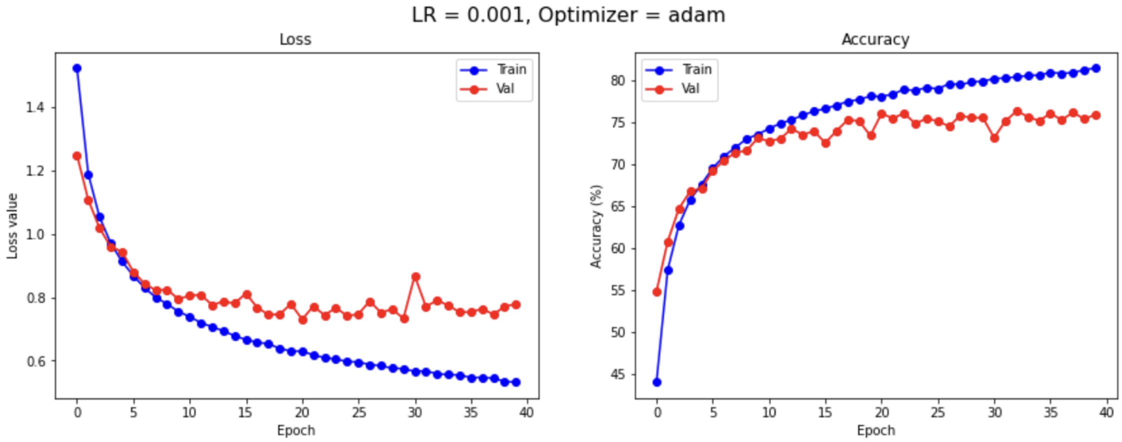
Loss and Accuracy Plot

Observations

- As the learning rate of 0.001 was already very slow for SGD, we have very little training done after 5 epochs.
- Training could not happen even after 20 epochs hence 5 epochs are pretty useless.

LR= 0.001, Epochs= 40, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.522	0.440	1.248	0.548
2	1.188	0.574	1.107	0.608
3	1.053	0.627	1.018	0.647
4	0.970	0.657	0.959	0.667
5	0.912	0.676	0.943	0.671
6	0.868	0.695	0.880	0.692
7	0.830	0.709	0.843	0.704
8	0.799	0.719	0.822	0.714
9	0.779	0.730	0.824	0.716
10	0.755	0.735	0.794	0.731
11	0.739	0.742	0.805	0.727
12	0.720	0.748	0.807	0.730
13	0.706	0.752	0.776	0.742
14	0.694	0.758	0.786	0.735
15	0.678	0.763	0.780	0.739
16	0.666	0.766	0.812	0.725
17	0.658	0.770	0.768	0.739
18	0.653	0.774	0.745	0.752
19	0.638	0.776	0.746	0.751
20	0.630	0.781	0.779	0.734
21	0.630	0.780	0.730	0.760
22	0.618	0.783	0.772	0.754
23	0.609	0.789	0.743	0.760
24	0.605	0.787	0.767	0.748
25	0.597	0.791	0.741	0.754
26	0.596	0.789	0.747	0.751
27	0.587	0.795	0.788	0.745
28	0.584	0.794	0.751	0.757
29	0.576	0.797	0.762	0.755
30	0.575	0.798	0.733	0.755
31	0.566	0.801	0.868	0.731
32	0.568	0.802	0.770	0.751
33	0.558	0.803	0.791	0.763
34	0.556	0.805	0.774	0.756
35	0.555	0.806	0.753	0.751
36	0.547	0.809	0.753	0.760
37	0.547	0.808	0.764	0.752
38	0.544	0.808	0.745	0.761
39	0.534	0.812	0.771	0.753
40	0.532	0.814	0.778	0.758



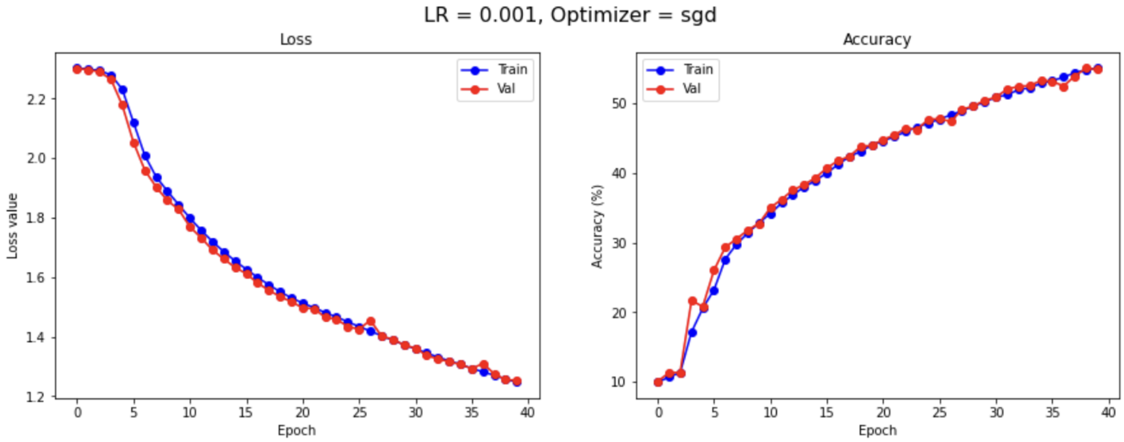
Loss and Accuracy Plot

Observations

- After 15 or so epochs the validation accuracy more or less saturates.
- The train accuracy increases and the netork is seeing the same inputs again and again so many times.
- However we do achieve any better results with respect to the validation accuracy.

LR= 0.001, Epochs= 40, Batch Size= 32, Optimizer= SGD

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	2.303	0.100	2.301	0.101
2	2.300	0.107	2.298	0.113
3	2.294	0.113	2.290	0.114
4	2.280	0.172	2.265	0.217
5	2.232	0.206	2.178	0.208
6	2.120	0.232	2.053	0.261
7	2.009	0.276	1.960	0.293
8	1.938	0.298	1.904	0.306
9	1.889	0.314	1.858	0.318
10	1.844	0.329	1.828	0.328
11	1.799	0.342	1.769	0.350
12	1.757	0.357	1.732	0.362
13	1.720	0.369	1.692	0.376
14	1.686	0.380	1.661	0.384
15	1.655	0.389	1.633	0.393
16	1.626	0.400	1.612	0.407
17	1.599	0.412	1.580	0.418
18	1.574	0.424	1.556	0.424
19	1.551	0.431	1.534	0.438
20	1.530	0.440	1.518	0.440
21	1.513	0.446	1.497	0.448
22	1.497	0.452	1.494	0.455
23	1.481	0.460	1.465	0.464
24	1.466	0.466	1.459	0.463
25	1.450	0.472	1.434	0.477
26	1.434	0.477	1.425	0.478
27	1.418	0.484	1.455	0.474
28	1.403	0.490	1.401	0.491
29	1.388	0.496	1.391	0.496
30	1.373	0.502	1.372	0.504
31	1.359	0.509	1.361	0.509
32	1.345	0.512	1.337	0.521
33	1.331	0.520	1.325	0.524
34	1.318	0.522	1.316	0.525
35	1.306	0.529	1.306	0.533
36	1.293	0.533	1.293	0.532
37	1.281	0.538	1.310	0.525
38	1.270	0.544	1.274	0.539
39	1.257	0.548	1.256	0.551
40	1.247	0.551	1.252	0.549



Loss and Accuracy Plot

Observations

- In the case of SGD, the learning rate of 0.001 is very slow as we had concluded earlier and even after 20 epochs the results are not great.
- The losses are still decreasing and the accuracies still increasing.

4) Batch Size

- Learning rate = 0.001
- Epochs = 20

LR= 0.001, Epochs= 20, Batch Size= 4, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.516	0.447	1.272	0.529
2	1.252	0.555	1.181	0.577
3	1.164	0.588	1.115	0.606
4	1.114	0.609	1.095	0.611
5	1.077	0.621	1.078	0.629
6	1.047	0.633	1.095	0.615
7	1.029	0.639	1.012	0.652
8	1.010	0.648	0.999	0.657
9	0.997	0.653	0.987	0.665
10	0.977	0.660	0.993	0.659
11	0.969	0.664	0.987	0.668
12	0.966	0.665	0.971	0.666
13	0.950	0.674	0.985	0.670
14	0.938	0.677	0.936	0.684
15	0.934	0.679	0.939	0.680
16	0.926	0.679	0.981	0.666
17	0.915	0.687	0.912	0.692
18	0.907	0.687	0.981	0.675
19	0.915	0.688	0.987	0.661
20	0.900	0.694	0.945	0.677

Observations

- For the small batch size of 4, we do not observe as good validation accuracies as we did in the previous parts.
- This tells us that smaller batch sizes are not that great.

LR= 0.001, Epochs= 20, Batch Size= 8, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.535	0.435	1.341	0.515
2	1.272	0.541	1.169	0.588
3	1.173	0.580	1.129	0.596
4	1.111	0.605	1.142	0.598
5	1.068	0.621	1.032	0.635
6	1.037	0.632	1.045	0.628
7	1.007	0.644	1.027	0.636
8	0.987	0.653	0.962	0.668
9	0.965	0.660	0.959	0.672
10	0.952	0.665	1.012	0.643
11	0.937	0.674	0.930	0.678
12	0.923	0.677	0.961	0.670
13	0.911	0.683	0.935	0.682
14	0.894	0.686	0.957	0.672
15	0.890	0.692	0.935	0.673
16	0.882	0.692	0.905	0.687
17	0.869	0.698	0.886	0.692
18	0.869	0.698	0.880	0.699
19	0.856	0.703	0.903	0.692
20	0.855	0.704	0.937	0.679

Observations

- On increasing the batch size to 8, we see an increase in the validation accuracy.
- Hence on increasing the batch size we are observing better results.

LR= 0.001, Epochs= 20, Batch Size= 16, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.488	0.454	1.214	0.563
2	1.170	0.583	1.059	0.625
3	1.035	0.635	1.042	0.635
4	0.952	0.663	0.950	0.666
5	0.904	0.683	0.929	0.682
6	0.871	0.697	0.883	0.696
7	0.833	0.707	0.875	0.694
8	0.809	0.717	0.812	0.722
9	0.783	0.725	0.817	0.714
10	0.771	0.732	0.800	0.728
11	0.753	0.737	0.808	0.729
12	0.742	0.741	0.780	0.736
13	0.722	0.749	0.804	0.727
14	0.718	0.748	0.777	0.737
15	0.706	0.754	0.823	0.722
16	0.689	0.761	0.790	0.736
17	0.682	0.763	0.768	0.740
18	0.673	0.765	0.762	0.740
19	0.671	0.767	0.798	0.737
20	0.660	0.769	0.758	0.751

Observations

- Now on increasing the batch size to 16 we see a significant improvement in the validation accuracy.
- This tells us that 16 is a good batch size.

LR= 0.001, Epochs= 20, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.518	0.441	1.232	0.557
2	1.176	0.583	1.035	0.638
3	1.032	0.634	0.988	0.653
4	0.945	0.665	0.919	0.684
5	0.888	0.688	0.834	0.711
6	0.845	0.703	0.874	0.697
7	0.803	0.717	0.828	0.718
8	0.779	0.726	0.834	0.711
9	0.759	0.736	0.780	0.739
10	0.738	0.742	0.750	0.744
11	0.719	0.749	0.803	0.731
12	0.709	0.751	0.758	0.746
13	0.696	0.757	0.777	0.745
14	0.684	0.760	0.744	0.754
15	0.675	0.764	0.737	0.754
16	0.661	0.769	0.745	0.747
17	0.653	0.770	0.730	0.752
18	0.644	0.775	0.740	0.757
19	0.632	0.778	0.756	0.747
20	0.626	0.780	0.724	0.759

Observations

- Increasing the batch size further to 32 leads to a slight increase in the accuracy compared to the case of batch size = 16.
- Hence 32 is the best batch size among the given values.

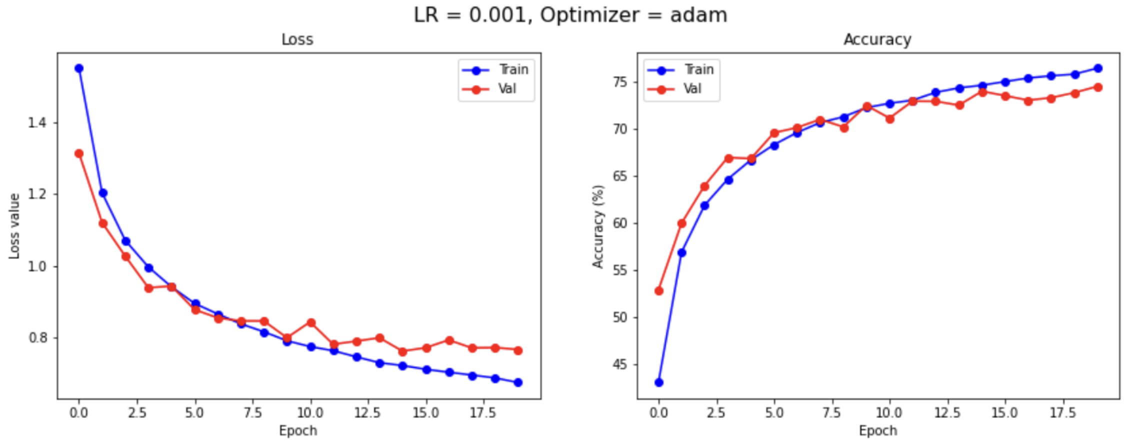
Effect of Loss Function

- Used the KL Divergence loss for this example.

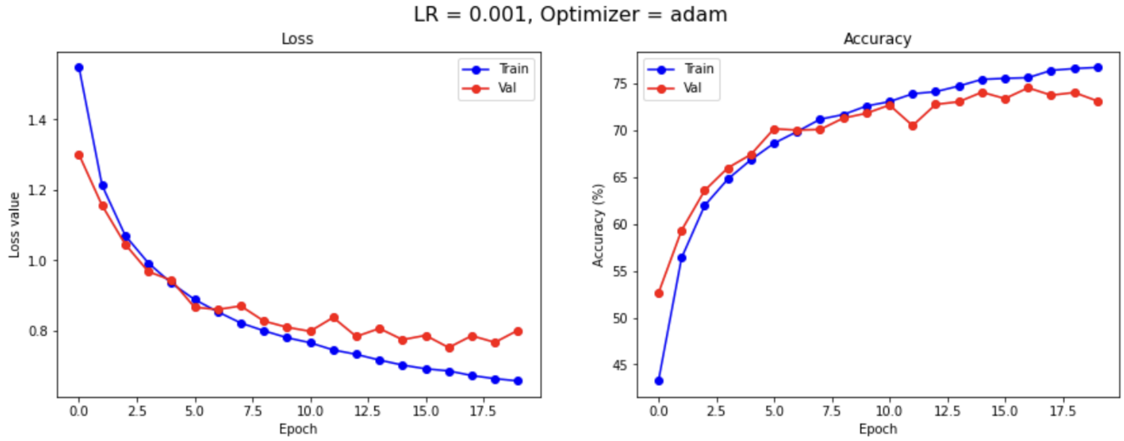
LR= 0.001, Epochs= 20, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.550	0.430	1.313	0.528
2	1.202	0.568	1.120	0.599
3	1.071	0.618	1.027	0.639
4	0.996	0.646	0.939	0.669
5	0.941	0.666	0.943	0.668
6	0.895	0.683	0.878	0.695
7	0.866	0.695	0.855	0.701
8	0.838	0.706	0.846	0.709
9	0.816	0.712	0.846	0.701
10	0.791	0.722	0.800	0.724
11	0.775	0.726	0.843	0.711
12	0.763	0.730	0.781	0.729
13	0.746	0.738	0.790	0.729
14	0.730	0.743	0.799	0.725
15	0.722	0.746	0.762	0.739
16	0.712	0.750	0.772	0.735
17	0.703	0.753	0.793	0.730
18	0.696	0.756	0.771	0.732
19	0.688	0.757	0.772	0.738
20	0.675	0.764	0.767	0.745

Accuracy for class: plane	78.3 %
Accuracy for class: car	86.0 %
Accuracy for class: bird	58.2 %
Accuracy for class: cat	59.4 %
Accuracy for class: deer	74.3 %
Accuracy for class: dog	63.3 %
Accuracy for class: frog	77.7 %
Accuracy for class: horse	75.1 %
Accuracy for class: ship	87.4 %
Accuracy for class: truck	84.8 %



Using KL Divergence



Original Plot using Cross Entropy

Observations

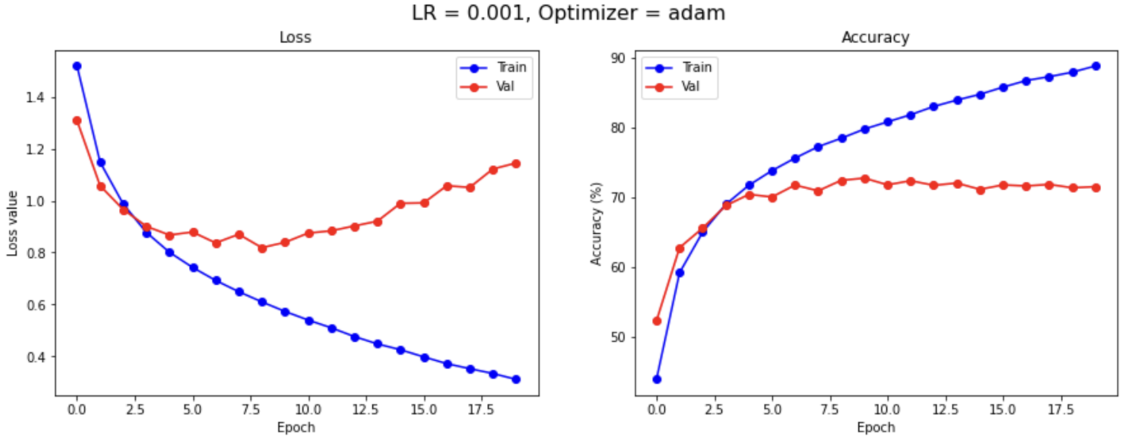
- KL Divergence is very similar to cross entropy loss.
- This is also verified when we compare their accuracies and losses.
- The plots are almost exactly the same.
- The reason for this is that KL Divergence is almost the same as cross entropy.

Effect of Data Augmentation

- Data Augmentation was turned off for this example.

LR= 0.001, Epochs= 20, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.519	0.439	1.308	0.523
2	1.148	0.592	1.057	0.628
3	0.987	0.650	0.966	0.656
4	0.878	0.690	0.901	0.688
5	0.801	0.717	0.868	0.704
6	0.743	0.738	0.879	0.700
7	0.693	0.756	0.837	0.718
8	0.649	0.772	0.870	0.709
9	0.610	0.785	0.819	0.724
10	0.573	0.798	0.839	0.727
11	0.540	0.808	0.875	0.718
12	0.511	0.818	0.884	0.724
13	0.477	0.830	0.903	0.717
14	0.448	0.839	0.921	0.720
15	0.426	0.847	0.990	0.711
16	0.399	0.858	0.991	0.718
17	0.372	0.867	1.057	0.716
18	0.353	0.873	1.051	0.718
19	0.334	0.879	1.122	0.714
20	0.312	0.888	1.145	0.715



Using KL Divergence

Observations

- Without data augmentation we observe that overfitting takes place.
- The network starts memorising the images and the validation loss quickly starts taking a U shape.
- This tells us that data augmentation is a really helpful technique that prevents overfiting.

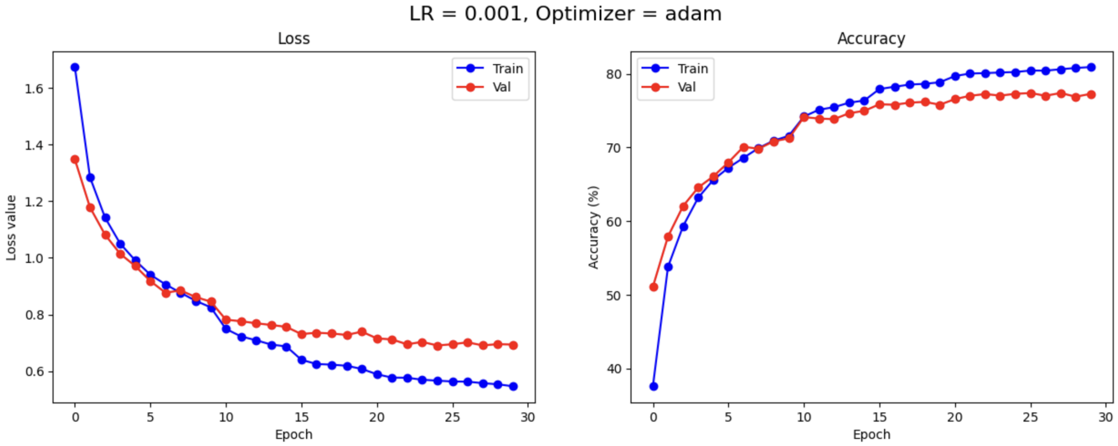
Part 3: Improving the CNN model

- I was able to slightly improve the validation accuracy by experimenting with the layers and hyperparameters.
- The setup was:
  1. `CONV1:` Kernel size (3×3), In channels 3, Out channels 32.
  2. `CONV2:` Kernel size (5×5), In channels 32, Out channels 64.
  3. `POOL1:` Kernel size (2×2)
  4. `CONV3:` Kernel size (3×3), In channels 64, Out channels 64.
  5. `POOL2:` Kernel size (2×2).
  6. `CONV4:` Kernel size (3×3), In channels 64, Out channels 64.
  7. `FC1:` Fully connected layer (also known as Linear layer) with 256 output neurons.
  8. `FC2:` Fully connected layer with 64 output neurons.
  9. `FC3:` Fully connected layer with 10 output neurons.
- The hyper-parameters that I chose were:
  1. Epochs: 30
  2. Learning Rate: 0.001
  3. Batch Size: 32
  4. Optimizer: Adam
  5. Used the LR Scheduler
- The reason for adding the layer was that the flatenned output goes from very high dimensions (576) to a lower dimension (64) so adding another layer could be beneficial.
- I also added another convolution layer but did not pool the first convolution layer. This is because pooling too many times would not be a good idea as it would significantly reduce the size of the inputs to the consecutive layers.
- I obtained an accuracy of 77.4% on the validation set with the above specified CNN.

LR= 0.0002, Epochs= 20, Batch Size= 32, Optimizer= Adam

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.675	0.376	1.348	0.511
2	1.285	0.538	1.179	0.579
3	1.142	0.593	1.082	0.620
4	1.051	0.632	1.014	0.645
5	0.991	0.656	0.971	0.661
6	0.940	0.672	0.918	0.680
7	0.906	0.686	0.877	0.701
8	0.877	0.699	0.885	0.698
9	0.848	0.709	0.861	0.708
10	0.825	0.716	0.845	0.712
11	0.749	0.742	0.781	0.741
12	0.722	0.751	0.776	0.739
13	0.709	0.755	0.769	0.738
14	0.694	0.761	0.763	0.747
15	0.687	0.764	0.756	0.750
16	0.640	0.779	0.731	0.759
17	0.625	0.782	0.735	0.758
18	0.623	0.785	0.733	0.761
19	0.619	0.786	0.728	0.762
20	0.607	0.788	0.739	0.758
21	0.589	0.797	0.715	0.766
22	0.577	0.800	0.712	0.770
23	0.576	0.801	0.694	0.772
24	0.569	0.802	0.703	0.770
25	0.566	0.802	0.690	0.773
26	0.563	0.804	0.695	0.774
27	0.562	0.804	0.701	0.770
28	0.557	0.806	0.691	0.774
29	0.553	0.808	0.695	0.769
30	0.546	0.809	0.694	0.773

Accuracy for class: plane	82.2 %
Accuracy for class: car	88.4 %
Accuracy for class: bird	71.2 %
Accuracy for class: cat	62.6 %
Accuracy for class: deer	72.9 %
Accuracy for class: dog	59.9 %
Accuracy for class: frog	83.2 %
Accuracy for class: horse	78.5 %
Accuracy for class: ship	86.6 %
Accuracy for class: truck	87.3 %



Loss and Accuracy Plot

Observations

- In part 2 the highest obtained validation accuracy was 75.8%.
- Now I have obtained a validation accuracy of 77.4% after 30 epochs.
- I only made minor changes to the network and was still able to obtain a higher accuracy, tweeking the network in some other ways that I did not explore could have led to even higher accuracies such as those obtained by ResNet.