CS6910
Fundamentals of Deep Learning

Programming Assignment 1

Aditya Mallick ee21b005@smail.iitm.ac.in

Ayushman Agarwal ee21b027@smail.iitm.ac.in

Srinivasan Kidambi ee21b139@smail.iitm.ac.in

Indian Institute of Technology, Madras

Monday 25th March, 2024

Contents

1 Task 1

Comparison of optimization methods used in training of MLFFNN models for classification on Image dataset 1

1.1 Data

A brief description of the provided datasets has been given below.

1.1.1 Training Data

The training data has 36 columns representing features. Therefore, our neural network's input layer should have 36 nodes. There are 2000 rows in the dataset, thus, we have 2000 samples of training data.

There are a total of 5 classes present. Therefore, the output layer of our neural network should have 5 nodes. There is a uniform distribution of class labels over the 2000 data points, i.e. each class has 400 data points. Therefore, we have a balanced dataset which is a good thing since our model gets equal to exposure to all the 5 classes while training.

1.1.2 Validation Data

The validation data has 36 columns representing image features. There are 500 rows of such validation data. The label distribution is uniform, i.e. each class has 100 datapoints.

1.1.3 Test Data

The test data has 36 columns representing image features. There are 500 rows of such test data. The label distribution is uniform, i.e. each class has 100 datapoints.

1.2 Model

A Multilayer Feedforward Neural Network has been created with the below parameters.

- Number of nodes in input layer : 36
- Number of hidden layers : 2
- Hidden layer activation function : TanH
- Number of nodes in output layer : 5
- Normalization : Not used
- Softmax applied on output layer

1.3 Training

To implement the following weight update rules:

- Delta rule
- Generalized delta rule
- AdaGrad
- RMSProp
- AdaM

The model has been trained in the respective manner:

- Mode : Pattern mode training
- Optimizers :
  - optim.SGD(..., momentum = 0)
  - optim.SGD(..., momentum $\neq$ 0)
  - optim.Adagrad(...)
  - optim.RMSprop(...)
  - optim.Adam(...)
- Loss function : Cross entropy loss
- Stopping criterion : Change in average training error below a threshold. If no convergence within 200 epochs, end training.

1.4 Hyperparameters

Listed below are the various hyperparameters of the models. We have performed a grid search to compare model performance using a wide variety of values of these parameters. The results have been tabulated for each optimizer.

- Number of nodes in the hidden layers
- Learning rate
- Threshold for stopping criterion
- Optimizer-specific parameters

We now explain the reasoning behind choosing each of the default values for our models:

Number of nodes in the hidden layers : The number of nodes controls the network's architecture. Choosing the right number of nodes impacts how well the network learns and performs, requiring experimentation to find the optimal value. Given that the training dataset is quite small (2000 datapoints), we should expect a small number of nodes per hidden layer to perform adequately.

Learning rate : Learning rate, like the number of nodes, is set before training. It controls the step size for weight adjustments, impacting how fast and well the model learns. We need to experiment to find the optimal rate that avoids getting stuck or jumping past the best solution.

Threshold for stopping criterion : Stopping threshold is also set before training. It controls when to stop training to avoid overfitting the model and wasting resources on unnecessary training. It's a balance between enough training and efficiency.

Optimizer-specific parameters : These parameters are specific to the type of optimizer that is being used. For example, the value for momentum is a hyperparameter in the generalized delta optimizer. Similarly for RMSProp also we have tried varying the value of α. In the case of the AdaM optimizer, we have gone ahead with the values recommended by the original paper, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. (Adam: A Method for Stochastic Optimization)

1.5 Comparison of Optimization Methods

The tabulated results for the explored hyperparameter space has been presented in a series of tables below:

Optimizer	HL1	HL2	Learning Rate	Threshold	Val Error	Train Error	Test Error	Converged In
Delta	6	6	0.001	0.0010	1.604441	1.606752	1.605431	5
Delta	6	6	0.001	0.0001	1.142128	1.145656	1.174471	Not within 200
Delta	12	12	0.001	0.0010	1.179082	1.195815	1.188498	91
Delta	12	12	0.001	0.0001	1.147220	1.134483	1.160162	Not within 200
Delta	32	32	0.001	0.0010	1.176093	1.188522	1.181032	83
Delta	32	32	0.001	0.0001	1.150355	1.134161	1.160941	Not within 200
Delta	6	12	0.001	0.0010	1.215785	1.231186	1.220223	61
Delta	6	12	0.001	0.0001	1.147760	1.133313	1.153408	Not within 200
Delta	12	6	0.001	0.0010	1.603490	1.605341	1.603729	5
Delta	12	6	0.001	0.0001	1.142396	1.136730	1.163172	Not within 200
Delta	12	32	0.001	0.0010	1.153556	1.160806	1.159866	100
Delta	12	32	0.001	0.0001	1.148025	1.131442	1.165236	Not within 200
Delta	32	12	0.001	0.0010	1.603832	1.605949	1.604412	6
Delta	32	12	0.001	0.0001	1.146814	1.131481	1.162625	198
Delta	6	6	0.010	0.0010	1.166348	1.153000	1.168626	53
Delta	6	6	0.010	0.0001	1.214656	1.089458	1.250253	Not within 200
Delta	12	12	0.010	0.0010	1.306386	0.965170	1.325094	196
Delta	12	12	0.010	0.0001	1.274367	1.010110	1.285886	Not within 200
Delta	32	32	0.010	0.0010	1.152096	1.166451	1.172760	41
Delta	32	32	0.010	0.0001	1.563293	0.884367	1.510912	Not within 200
Delta	6	12	0.010	0.0010	1.149582	1.155978	1.169749	53
Delta	6	12	0.010	0.0001	1.284923	1.050819	1.273281	Not within 200
Delta	12	6	0.010	0.0010	1.159290	1.139729	1.182068	65
Delta	12	6	0.010	0.0001	1.331877	1.036595	1.267306	Not within 200
Delta	12	32	0.010	0.0010	1.154817	1.156909	1.171549	49
Delta	12	32	0.010	0.0001	1.397727	0.954511	1.264054	Not within 200
Delta	32	12	0.010	0.0010	1.314857	0.985222	1.345421	Not within 200
Delta	32	12	0.010	0.0001	1.402324	0.961095	1.418369	Not within 200
Delta	6	6	0.100	0.0010	1.277721	1.338944	1.335680	19
Delta	6	6	0.100	0.0001	1.341014	1.301738	1.377244	43
Delta	12	12	0.100	0.0010	1.264147	1.353674	1.282083	19
Delta	12	12	0.100	0.0001	1.282562	1.327850	1.303677	20
Delta	32	32	0.100	0.0010	1.256303	1.375277	1.310573	12
Delta	32	32	0.100	0.0001	1.304694	1.370707	1.365606	17
Delta	6	12	0.100	0.0010	1.277500	1.356682	1.330059	13
Delta	6	12	0.100	0.0001	1.478745	1.270944	1.415982	Not within 200
Delta	12	6	0.100	0.0010	1.252816	1.354775	1.301146	15
Delta	12	6	0.100	0.0001	1.487159	1.203926	1.490466	100
Delta	12	32	0.100	0.0010	1.247217	1.341278	1.256301	16
Delta	12	32	0.100	0.0001	1.252477	1.342356	1.292854	17
Delta	32	12	0.100	0.0010	1.287810	1.374439	1.321180	13
Delta	32	12	0.100	0.0001	1.302452	1.365252	1.326895	20

Optimizer	HL1	HL2	LR	Thresh	ValErr	TrainErr	TestErr	ConvergedIn
GeneralizedDelta ($\alpha = 0.9$)	6	6	0.001	0.0010	1.154181	1.147750	1.176055	28
GeneralizedDelta ($\alpha = 0.9$)	6	6	0.001	0.0001	1.225679	1.059443	1.224665	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	12	12	0.001	0.0010	1.146221	1.138311	1.160395	33
GeneralizedDelta ($\alpha = 0.9$)	12	12	0.001	0.0001	1.340830	0.950579	1.369773	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	32	32	0.001	0.0010	1.150801	1.146187	1.165697	29
GeneralizedDelta ($\alpha = 0.9$)	32	32	0.001	0.0001	1.499915	0.869311	1.510520	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	6	12	0.001	0.0010	1.148022	1.146165	1.164042	29
GeneralizedDelta ($\alpha = 0.9$)	6	12	0.001	0.0001	1.231873	1.033481	1.257152	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	12	6	0.001	0.0010	1.150386	1.144974	1.169038	32
GeneralizedDelta ($\alpha = 0.9$)	12	6	0.001	0.0001	1.273278	1.026818	1.223211	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	12	32	0.001	0.0010	1.151155	1.148089	1.167913	26
GeneralizedDelta ($\alpha = 0.9$)	12	32	0.001	0.0001	1.457503	0.910702	1.418349	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	32	12	0.001	0.0010	1.149072	1.146276	1.164626	27
GeneralizedDelta ($\alpha = 0.9$)	32	12	0.001	0.0001	1.402720	0.924674	1.383583	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	6	6	0.010	0.0010	1.221588	1.233519	1.260805	16
GeneralizedDelta ($\alpha = 0.9$)	6	6	0.010	0.0001	1.369476	1.218109	1.416658	46
GeneralizedDelta ($\alpha = 0.9$)	12	12	0.010	0.0010	1.222827	1.241203	1.259050	11
GeneralizedDelta ($\alpha = 0.9$)	12	12	0.010	0.0001	1.218166	1.240415	1.288532	14
GeneralizedDelta ($\alpha = 0.9$)	32	32	0.010	0.0010	1.243155	1.252562	1.302527	12
GeneralizedDelta ($\alpha = 0.9$)	32	32	0.010	0.0001	1.249929	1.255561	1.303280	13
GeneralizedDelta ($\alpha = 0.9$)	6	12	0.010	0.0010	1.231507	1.234633	1.272834	14
GeneralizedDelta ($\alpha = 0.9$)	6	12	0.010	0.0001	1.280091	1.215017	1.330488	40
GeneralizedDelta ($\alpha = 0.9$)	12	6	0.010	0.0010	1.354298	1.250285	1.413061	12
GeneralizedDelta ($\alpha = 0.9$)	12	6	0.010	0.0001	1.458813	1.108454	1.552437	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	12	32	0.010	0.0010	1.306376	1.242637	1.364272	15
GeneralizedDelta ($\alpha = 0.9$)	12	32	0.010	0.0001	2.383350	1.368939	2.374039	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	32	12	0.010	0.0010	1.231042	1.250391	1.289292	10
GeneralizedDelta ($\alpha = 0.9$)	32	12	0.010	0.0001	1.372440	1.314999	1.424538	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	6	6	0.100	0.0010	1.624348	1.943965	1.624347	4
GeneralizedDelta ($\alpha = 0.9$)	6	6	0.100	0.0001	1.624364	1.943888	1.624377	3
GeneralizedDelta ($\alpha = 0.9$)	12	12	0.100	0.0010	1.666915	2.301256	1.666922	3
GeneralizedDelta ($\alpha = 0.9$)	12	12	0.100	0.0001	1.666933	2.301250	1.666932	6
GeneralizedDelta ($\alpha = 0.9$)	32	32	0.100	0.0010	3.710337	5.422174	3.710337	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	32	32	0.100	0.0001	5.170574	5.329113	5.170574	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	6	12	0.100	0.0010	1.649002	2.174467	1.649009	6
GeneralizedDelta ($\alpha = 0.9$)	6	12	0.100	0.0001	1.629220	1.999109	1.629260	6
GeneralizedDelta ($\alpha = 0.9$)	12	6	0.100	0.0010	1.624373	1.943987	1.624373	3
GeneralizedDelta ($\alpha = 0.9$)	12	6	0.100	0.0001	1.624310	1.943958	1.624319	3
GeneralizedDelta ($\alpha = 0.9$)	12	32	0.100	0.0010	3.705663	5.362065	3.705663	73
GeneralizedDelta ($\alpha = 0.9$)	12	32	0.100	0.0001	4.387629	5.429925	4.387629	Not within 200
GeneralizedDelta ($\alpha = 0.9$)	32	12	0.100	0.0010	1.666940	2.301283	1.666940	5
GeneralizedDelta ($\alpha = 0.9$)	32	12	0.100	0.0001	1.666940	2.301281	1.666941	3

Optimizer	H1	H2	LR	Thresh	ValErr	TrainErr	TestErr	ConvergedIn
GeneralizedDelta ($\alpha = 0.99$)	6	6	0.001	0.0010	1.308064	1.254251	1.353839	6
GeneralizedDelta ($\alpha = 0.99$)	6	6	0.001	0.0001	1.400131	1.194743	1.443485	Not within 200
GeneralizedDelta ($\alpha = 0.99$)	12	12	0.001	0.0010	1.333150	1.248923	1.368170	6
GeneralizedDelta ($\alpha = 0.99$)	12	12	0.001	0.0001	1.232651	1.240731	1.286490	10
GeneralizedDelta ($\alpha = 0.99$)	32	32	0.001	0.0010	1.266020	1.245785	1.310836	10
GeneralizedDelta ($\alpha = 0.99$)	32	32	0.001	0.0001	1.337717	1.233749	1.324941	22
GeneralizedDelta ($\alpha = 0.99$)	6	12	0.001	0.0010	1.215548	1.233524	1.247915	20
GeneralizedDelta ($\alpha = 0.99$)	6	12	0.001	0.0001	1.413506	1.198328	1.451433	Not within 200
GeneralizedDelta ($\alpha = 0.99$)	12	6	0.001	0.0010	1.257736	1.267933	1.287064	5
GeneralizedDelta ($\alpha = 0.99$)	12	6	0.001	0.0001	1.505123	1.110741	1.428446	181
GeneralizedDelta ($\alpha = 0.99$)	12	32	0.001	0.0010	1.264261	1.244035	1.313999	6
GeneralizedDelta ($\alpha = 0.99$)	12	32	0.001	0.0001	1.431155	1.202131	1.539975	63
GeneralizedDelta ($\alpha = 0.99$)	32	12	0.001	0.0010	1.256952	1.248474	1.336394	6
GeneralizedDelta ($\alpha = 0.99$)	32	12	0.001	0.0001	1.394225	1.282815	1.432548	184
GeneralizedDelta ($\alpha = 0.99$)	6	6	0.010	0.0010	1.609558	1.637935	1.609558	3
GeneralizedDelta ($\alpha = 0.99$)	6	6	0.010	0.0001	1.609558	1.637933	1.609558	3
GeneralizedDelta ($\alpha = 0.99$)	12	12	0.010	0.0010	1.609858	1.662940	1.609858	3
GeneralizedDelta ($\alpha = 0.99$)	12	12	0.010	0.0001	1.609860	1.662940	1.609860	3
GeneralizedDelta ($\alpha = 0.99$)	32	32	0.010	0.0010	1.612256	1.750242	1.612256	5
GeneralizedDelta ($\alpha = 0.99$)	32	32	0.010	0.0001	1.612224	1.750230	1.612220	3
GeneralizedDelta ($\alpha = 0.99$)	6	12	0.010	0.0010	1.609821	1.662772	1.609831	5
GeneralizedDelta ($\alpha = 0.99$)	6	12	0.010	0.0001	1.609852	1.662923	1.609853	3
GeneralizedDelta ($\alpha = 0.99$)	12	6	0.010	0.0010	1.609558	1.637936	1.609558	6
GeneralizedDelta ($\alpha = 0.99$)	12	6	0.010	0.0001	1.609558	1.637936	1.609558	5
GeneralizedDelta ($\alpha = 0.99$)	12	32	0.010	0.0010	1.612255	1.750238	1.612255	5
GeneralizedDelta ($\alpha = 0.99$)	12	32	0.010	0.0001	1.612256	1.750234	1.612255	3
GeneralizedDelta ($\alpha = 0.99$)	32	12	0.010	0.0010	1.609856	1.662940	1.609857	3
GeneralizedDelta ($\alpha = 0.99$)	32	12	0.010	0.0001	1.609858	1.662941	1.609858	3
GeneralizedDelta ($\alpha = 0.99$)	6	6	0.100	0.0010	1.623041	1.928529	1.623041	2
GeneralizedDelta ($\alpha = 0.99$)	6	6	0.100	0.0001	1.623041	1.928529	1.623041	3
GeneralizedDelta ($\alpha = 0.99$)	12	12	0.100	0.0010	1.661623	2.267027	1.661623	3
GeneralizedDelta ($\alpha = 0.99$)	12	12	0.100	0.0001	1.661623	2.267027	1.661623	3
GeneralizedDelta ($\alpha = 0.99$)	32	32	0.100	0.0010	14.415029	12.400898	14.415029	38
GeneralizedDelta ($\alpha = 0.99$)	32	32	0.100	0.0001	8.698339	13.556698	8.698339	Not within 200
GeneralizedDelta ($\alpha = 0.99$)	6	12	0.100	0.0010	1.661619	2.266969	1.661619	3
GeneralizedDelta ($\alpha = 0.99$)	6	12	0.100	0.0001	1.661623	2.267027	1.661623	3
GeneralizedDelta ($\alpha = 0.99$)	12	6	0.100	0.0010	1.623041	1.928529	1.623041	3
GeneralizedDelta ($\alpha = 0.99$)	12	6	0.100	0.0001	1.623041	1.928529	1.623041	3
GeneralizedDelta ($\alpha = 0.99$)	12	32	0.100	0.0010	10.390327	13.700128	10.390327	Not within 200
GeneralizedDelta ($\alpha = 0.99$)	12	32	0.100	0.0001	8.482737	11.791562	8.482737	Not within 200
GeneralizedDelta ($\alpha = 0.99$)	32	12	0.100	0.0010	1.661622	2.267027	1.661622	3
GeneralizedDelta ($\alpha = 0.99$)	32	12	0.100	0.0001	1.661625	2.267027	1.661625	3

Optimizer	HL1	HL2	LR	Threshold	Val Error	Train Error	Test Error	Converged In
RMSProp ($\alpha = 0.9$)	6	6	0.001	0.0010	1.172044	1.193000	1.170948	21
RMSProp ($\alpha = 0.9$)	6	6	0.001	0.0001	1.194307	1.152896	1.236638	118
RMSProp ($\alpha = 0.9$)	12	12	0.001	0.0010	1.164403	1.191243	1.170192	24
RMSProp ($\alpha = 0.9$)	12	12	0.001	0.0001	1.271046	1.096007	1.296322	144
RMSProp ($\alpha = 0.9$)	32	32	0.001	0.0010	1.248971	1.067973	1.275892	85
RMSProp ($\alpha = 0.9$)	32	32	0.001	0.0001	1.599658	0.814347	1.553037	Not within 200
RMSProp ($\alpha = 0.9$)	6	12	0.001	0.0010	1.176202	1.180721	1.181484	37
RMSProp ($\alpha = 0.9$)	6	12	0.001	0.0001	1.234009	1.122085	1.230969	195
RMSProp ($\alpha = 0.9$)	12	6	0.001	0.0010	1.162144	1.193768	1.179612	40
RMSProp ($\alpha = 0.9$)	12	6	0.001	0.0001	1.245466	1.108561	1.258486	Not within 200
RMSProp ($\alpha = 0.9$)	12	32	0.001	0.0010	1.160015	1.181980	1.190921	40
RMSProp ($\alpha = 0.9$)	12	32	0.001	0.0001	1.224620	1.102953	1.214315	123
RMSProp ($\alpha = 0.9$)	32	12	0.001	0.0010	1.277120	1.052343	1.269764	125
RMSProp ($\alpha = 0.9$)	32	12	0.001	0.0001	1.327683	0.985015	1.328253	Not within 200
RMSProp ($\alpha = 0.9$)	6	6	0.010	0.0010	1.200555	1.247183	1.235001	18
RMSProp ($\alpha = 0.9$)	6	6	0.010	0.0001	1.266760	1.216494	1.272313	47
RMSProp ($\alpha = 0.9$)	12	12	0.010	0.0010	1.234623	1.267952	1.258643	15
RMSProp ($\alpha = 0.9$)	12	12	0.010	0.0001	1.328565	1.192847	1.360248	48
RMSProp ($\alpha = 0.9$)	32	32	0.010	0.0010	1.272017	1.378056	1.313360	9
RMSProp ($\alpha = 0.9$)	32	32	0.010	0.0001	1.231672	1.368527	1.301028	10
RMSProp ($\alpha = 0.9$)	6	12	0.010	0.0010	1.238005	1.238658	1.251066	28
RMSProp ($\alpha = 0.9$)	6	12	0.010	0.0001	1.255711	1.237354	1.323254	42
RMSProp ($\alpha = 0.9$)	12	6	0.010	0.0010	1.232091	1.274536	1.246155	12
RMSProp ($\alpha = 0.9$)	12	6	0.010	0.0001	1.224432	1.220235	1.241001	29
RMSProp ($\alpha = 0.9$)	12	32	0.010	0.0010	1.256658	1.301851	1.294526	13
RMSProp ($\alpha = 0.9$)	12	32	0.010	0.0001	1.351023	1.265043	1.389070	23
RMSProp ($\alpha = 0.9$)	32	12	0.010	0.0010	1.330185	1.260905	1.334309	28
RMSProp ($\alpha = 0.9$)	32	12	0.010	0.0001	1.262495	1.268759	1.294599	26
RMSProp ($\alpha = 0.9$)	6	6	0.100	0.0010	1.626330	1.958086	1.672706	23
RMSProp ($\alpha = 0.9$)	6	6	0.100	0.0001	1.535858	2.094719	1.570092	29
RMSProp ($\alpha = 0.9$)	12	12	0.100	0.0010	2.148217	2.657773	2.134471	6
RMSProp ($\alpha = 0.9$)	12	12	0.100	0.0001	2.017277	2.774116	2.017277	3
RMSProp ($\alpha = 0.9$)	32	32	0.100	0.0010	4.038336	5.193206	4.038336	3
RMSProp ($\alpha = 0.9$)	32	32	0.100	0.0001	4.038331	5.193206	4.038331	3
RMSProp ($\alpha = 0.9$)	6	12	0.100	0.0010	1.969320	2.668622	2.032084	150
RMSProp ($\alpha = 0.9$)	6	12	0.100	0.0001	1.974130	2.631473	2.000564	94
RMSProp ($\alpha = 0.9$)	12	6	0.100	0.0010	1.721194	2.257690	1.721194	65
RMSProp ($\alpha = 0.9$)	12	6	0.100	0.0001	1.721209	2.260091	1.721209	4
RMSProp ($\alpha = 0.9$)	12	32	0.100	0.0010	4.278410	4.851369	4.284009	14
RMSProp ($\alpha = 0.9$)	12	32	0.100	0.0001	3.791803	5.034964	3.835344	196
RMSProp ($\alpha = 0.9$)	32	12	0.100	0.0010	2.017277	2.774116	2.017277	3
RMSProp ($\alpha = 0.9$)	32	12	0.100	0.0001	2.017278	2.774116	2.017278	3

Optimizer	H1	H2	LR	Threshold	Val Error	Train Error	Test Error	Converged In
RMSProp ($\alpha = 0.99$)	6	6	0.001	0.0010	1.153921	1.150544	1.166969	52
RMSProp ($\alpha = 0.99$)	6	6	0.001	0.0001	1.250952	1.081249	1.215821	Not within 200
RMSProp ($\alpha = 0.99$)	12	12	0.001	0.0010	1.152998	1.160113	1.168949	28
RMSProp ($\alpha = 0.99$)	12	12	0.001	0.0001	1.344194	0.944572	1.314191	Not within 200
RMSProp ($\alpha = 0.99$)	32	32	0.001	0.0010	2.047872	0.579382	2.123614	Not within 200
RMSProp ($\alpha = 0.99$)	32	32	0.001	0.0001	2.214080	0.494391	2.164449	Not within 200
RMSProp ($\alpha = 0.99$)	6	12	0.001	0.0010	1.1435641	1.144312	1.170736	51
RMSProp ($\alpha = 0.99$)	6	12	0.001	0.0001	1.228500	1.064504	1.243645	Not within 200
RMSProp ($\alpha = 0.99$)	12	6	0.001	0.0010	1.142589	1.149610	1.174301	32
RMSProp ($\alpha = 0.99$)	12	6	0.001	0.0001	1.255740	1.031791	1.217175	Not within 200
RMSProp ($\alpha = 0.99$)	12	32	0.001	0.0010	1.409485	0.852527	1.452720	Not within 200
RMSProp ($\alpha = 0.99$)	12	32	0.001	0.0001	1.447971	0.856194	1.498746	Not within 200
RMSProp ($\alpha = 0.99$)	32	12	0.001	0.0010	1.509502	0.851445	1.499359	Not within 200
RMSProp ($\alpha = 0.99$)	32	12	0.001	0.0001	1.495662	0.852104	1.448797	Not within 200
RMSProp ($\alpha = 0.99$)	6	6	0.010	0.0010	1.219414	1.199369	1.236597	29
RMSProp ($\alpha = 0.99$)	6	6	0.010	0.0001	1.319652	1.128189	1.375502	70
RMSProp ($\alpha = 0.99$)	12	12	0.010	0.0010	1.281887	1.201936	1.317753	27
RMSProp ($\alpha = 0.99$)	12	12	0.010	0.0001	1.476538	1.003896	1.447496	Not within 200
RMSProp ($\alpha = 0.99$)	32	32	0.010	0.0010	1.353846	1.400116	1.380614	17
RMSProp ($\alpha = 0.99$)	32	32	0.010	0.0001	1.336082	1.315278	1.379795	99
RMSProp ($\alpha = 0.99$)	6	12	0.010	0.0010	1.297474	1.199242	1.268161	31
RMSProp ($\alpha = 0.99$)	6	12	0.010	0.0001	1.327170	1.155721	1.357871	109
RMSProp ($\alpha = 0.99$)	12	6	0.010	0.0010	1.309720	1.222422	1.316551	18
RMSProp ($\alpha = 0.99$)	12	6	0.010	0.0001	1.379030	1.098792	1.286795	66
RMSProp ($\alpha = 0.99$)	12	32	0.010	0.0010	1.277791	1.270238	1.305561	14
RMSProp ($\alpha = 0.99$)	12	32	0.010	0.0001	1.433491	1.187533	1.418069	43
RMSProp ($\alpha = 0.99$)	32	12	0.010	0.0010	1.243945	1.330439	1.296234	10
RMSProp ($\alpha = 0.99$)	32	12	0.010	0.0001	1.269630	1.271461	1.266009	47
RMSProp ($\alpha = 0.99$)	6	6	0.100	0.0010	1.733200	2.280335	1.733200	3
RMSProp ($\alpha = 0.99$)	6	6	0.100	0.0001	1.733200	2.280309	1.733200	3
RMSProp ($\alpha = 0.99$)	12	12	0.100	0.0010	2.056551	2.798981	2.056551	3
RMSProp ($\alpha = 0.99$)	12	12	0.100	0.0001	2.056551	2.798981	2.056551	3
RMSProp ($\alpha = 0.99$)	32	32	0.100	0.0010	4.180393	5.274473	4.180393	3
RMSProp ($\alpha = 0.99$)	32	32	0.100	0.0001	4.180396	5.274474	4.180396	3
RMSProp ($\alpha = 0.99$)	6	12	0.100	0.0010	2.056551	2.798980	2.056551	3
RMSProp ($\alpha = 0.99$)	6	12	0.100	0.0001	2.056551	2.798980	2.056551	4
RMSProp ($\alpha = 0.99$)	12	6	0.100	0.0010	1.733200	2.280309	1.733200	3
RMSProp ($\alpha = 0.99$)	12	6	0.100	0.0001	1.733200	2.280309	1.733200	3
RMSProp ($\alpha = 0.99$)	12	32	0.100	0.0010	4.180395	5.274473	4.180395	4
RMSProp ($\alpha = 0.99$)	12	32	0.100	0.0001	4.180405	5.274474	4.180405	3
RMSProp ($\alpha = 0.99$)	32	12	0.100	0.0010	2.056551	2.798981	2.056551	3
RMSProp ($\alpha = 0.99$)	32	12	0.100	0.0001	2.056551	2.798981	2.056551	3

Optimizer	HL1	HL2	Learning Rate	Threshold	Val Error	Train Error	Test Error	Converged In
AdaGrad	6	6	0.001	0.0010	1.611774	1.613640	1.613162	7
AdaGrad	6	6	0.001	0.0001	1.546724	1.551679	1.546266	Not within 200
AdaGrad	12	12	0.001	0.0010	1.602339	1.603444	1.603265	9
AdaGrad	12	12	0.001	0.0001	1.385463	1.401782	1.389667	Not within 200
AdaGrad	32	32	0.001	0.0010	1.283430	1.293806	1.281148	98
AdaGrad	32	32	0.001	0.0001	1.251197	1.265689	1.258794	Not within 200
AdaGrad	6	12	0.001	0.0010	1.609490	1.610179	1.609009	4
AdaGrad	6	12	0.001	0.0001	1.474207	1.482089	1.475436	Not within 200
AdaGrad	12	6	0.001	0.0010	1.606809	1.608088	1.607002	5
AdaGrad	12	6	0.001	0.0001	1.476550	1.480747	1.475758	Not within 200
AdaGrad	12	32	0.001	0.0010	1.343261	1.357330	1.344075	157
AdaGrad	12	32	0.001	0.0001	1.362724	1.370896	1.379294	Not within 200
AdaGrad	32	12	0.001	0.0010	1.339854	1.352319	1.360472	166
AdaGrad	32	12	0.001	0.0001	1.331008	1.350165	1.338695	Not within 200
AdaGrad	6	6	0.010	0.0010	1.230342	1.244709	1.244596	44
AdaGrad	6	6	0.010	0.0001	1.215433	1.223842	1.224996	Not within 200
AdaGrad	12	12	0.010	0.0010	1.217124	1.238210	1.231514	22
AdaGrad	12	12	0.010	0.0001	1.168137	1.182495	1.192106	Not within 200
AdaGrad	32	32	0.010	0.0010	1.168507	1.176940	1.181308	38
AdaGrad	32	32	0.010	0.0001	1.142447	1.139964	1.162770	160
AdaGrad	6	12	0.010	0.0010	1.232511	1.247701	1.240205	29
AdaGrad	6	12	0.010	0.0001	1.184789	1.199023	1.196178	Not within 200
AdaGrad	12	6	0.010	0.0010	1.248493	1.250154	1.260563	26
AdaGrad	12	6	0.010	0.0001	1.191746	1.188792	1.191028	174
AdaGrad	12	32	0.010	0.0010	1.194802	1.208782	1.213118	42
AdaGrad	12	32	0.010	0.0001	1.145423	1.151163	1.167353	Not within 200
AdaGrad	32	12	0.010	0.0010	1.212720	1.228133	1.229686	24
AdaGrad	32	12	0.010	0.0001	1.135438	1.143196	1.164031	Not within 200
AdaGrad	6	6	0.100	0.0010	1.147657	1.147102	1.168617	42
AdaGrad	6	6	0.100	0.0001	1.147693	1.115250	1.164101	146
AdaGrad	12	12	0.100	0.0010	1.142696	1.134456	1.162599	27
AdaGrad	12	12	0.100	0.0001	1.157235	1.070824	1.153010	Not within 200
AdaGrad	32	32	0.100	0.0010	1.279171	0.854284	1.308937	Not within 200
AdaGrad	32	32	0.100	0.0001	1.399284	0.818487	1.330420	Not within 200
AdaGrad	6	12	0.100	0.0010	1.143595	1.144309	1.160118	30
AdaGrad	6	12	0.100	0.0001	1.159298	1.109000	1.174462	Not within 200
AdaGrad	12	6	0.100	0.0010	1.168531	1.141246	1.158025	36
AdaGrad	12	6	0.100	0.0001	1.165794	1.084431	1.146418	Not within 200
AdaGrad	12	32	0.100	0.0010	1.146304	1.131006	1.161610	31
AdaGrad	12	32	0.100	0.0001	1.167716	1.065987	1.165852	Not within 200
AdaGrad	32	12	0.100	0.0010	1.179456	1.064000	1.189982	64
AdaGrad	32	12	0.100	0.0001	1.236737	0.991656	1.220558	Not within 200

Optimizer	HL1	HL2	Learning Rate	Threshold	Val Error	Train Error	Test Error	Converged In
Adam	6	6	0.001	0.0010	1.142660	1.143724	1.166562	23
Adam	6	6	0.001	0.0001	1.214939	1.068875	1.191415	Not within 200
Adam	12	12	0.001	0.0010	1.144244	1.140462	1.165081	25
Adam	12	12	0.001	0.0001	1.225706	0.968828	1.244357	Not within 200
Adam	32	32	0.001	0.0010	2.284846	0.509084	2.269855	Not within 200
Adam	32	32	0.001	0.0001	1.996333	0.531022	2.051144	Not within 200
Adam	6	12	0.001	0.0010	1.139021	1.138306	1.159386	26
Adam	6	12	0.001	0.0001	1.191665	1.049523	1.183219	Not within 200
Adam	12	6	0.001	0.0010	1.135437	1.135227	1.171032	28
Adam	12	6	0.001	0.0001	1.211510	1.019297	1.209107	Not within 200
Adam	12	32	0.001	0.0010	1.149509	1.142218	1.167901	24
Adam	12	32	0.001	0.0001	1.369615	0.858460	1.354960	Not within 200
Adam	32	12	0.001	0.0010	1.487589	0.864987	1.439878	Not within 200
Adam	32	12	0.001	0.0001	1.491651	0.850069	1.485392	Not within 200
Adam	6	6	0.010	0.0010	1.197720	1.149584	1.241872	23
Adam	6	6	0.010	0.0001	1.257424	1.152293	1.314113	26
Adam	12	12	0.010	0.0010	1.245428	1.123770	1.241145	39
Adam	12	12	0.010	0.0001	1.461875	0.982102	1.481067	184
Adam	32	32	0.010	0.0010	1.365297	1.260557	1.348576	80
Adam	32	32	0.010	0.0001	1.363910	1.263160	1.389755	100
Adam	6	12	0.010	0.0010	1.244495	1.154280	1.274990	23
Adam	6	12	0.010	0.0001	1.298330	1.112470	1.313913	65
Adam	12	6	0.010	0.0010	1.211176	1.167764	1.265901	20
Adam	12	6	0.010	0.0001	1.328489	1.108718	1.348194	48
Adam	12	32	0.010	0.0010	1.285239	1.189321	1.380626	18
Adam	12	32	0.010	0.0001	1.653328	0.932222	1.764475	Not within 200
Adam	32	12	0.010	0.0010	1.283127	1.266345	1.319453	7
Adam	32	12	0.010	0.0001	1.255951	1.194362	1.290213	189
Adam	6	6	0.100	0.0010	1.610251	1.684591	1.610251	3
Adam	6	6	0.100	0.0001	1.610249	1.683910	1.610249	11
Adam	12	12	0.100	0.0010	1.612231	1.750171	1.612234	3
Adam	12	12	0.100	0.0001	1.612235	1.749464	1.612235	5
Adam	32	32	0.100	0.0010	1.627501	1.979455	1.627501	3
Adam	32	32	0.100	0.0001	1.627503	1.979474	1.627503	3
Adam	6	12	0.100	0.0010	1.611920	1.742794	1.611920	4
Adam	6	12	0.100	0.0001	1.611240	1.719862	1.611240	8
Adam	12	6	0.100	0.0010	1.610249	1.683943	1.610249	4
Adam	12	6	0.100	0.0001	1.610249	1.683910	1.610249	3
Adam	12	32	0.100	0.0010	1.627489	1.980728	1.627489	4
Adam	12	32	0.100	0.0001	1.627502	1.979477	1.627502	7
Adam	32	12	0.100	0.0010	1.612235	1.749454	1.612235	3
Adam	32	12	0.100	0.0001	1.612235	1.749454	1.612235	3

All models were initialized with the same random weights.

1.5.1 Delta rule

The delta rule for weight updates is given by:

$$w(\tau + 1) = w(\tau) - \eta \cdot \frac{\partial \varepsilon}{\partial w}\Big|_{w=w(\tau)}$$

where w is the weight parameter, ε is the error as a function of weight and η is the learning rate. When paired with the pattern mode of training, it is traditionally known as **Stochastic Gradient Descent**.

Best Model Statistics

- Did not converge within 200 epochs
- Final Train Loss: 1.1333126146639698
- Final Validation Loss: 1.1477596774399281
- Final Test Loss: 1.1534081799946725
- Validation Accuracy: 53.6%
- H1 = 6, H2 = 12, LR = 0.001, Threshold = 0.0001

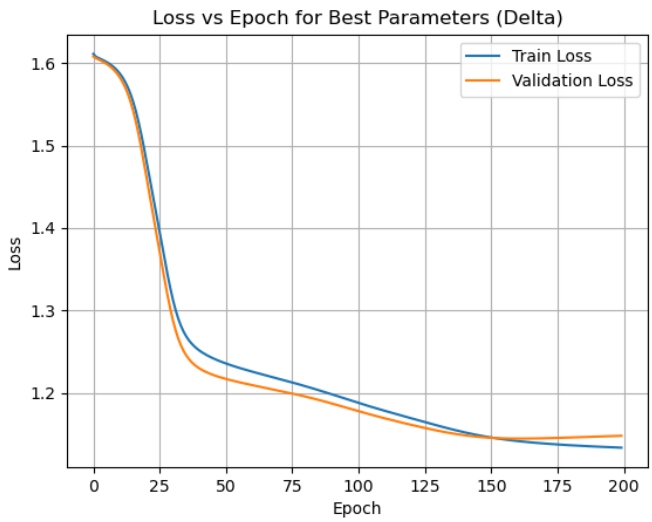
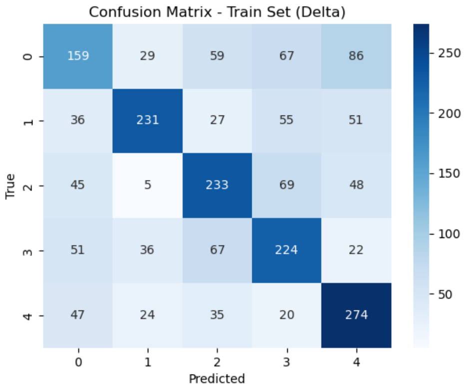
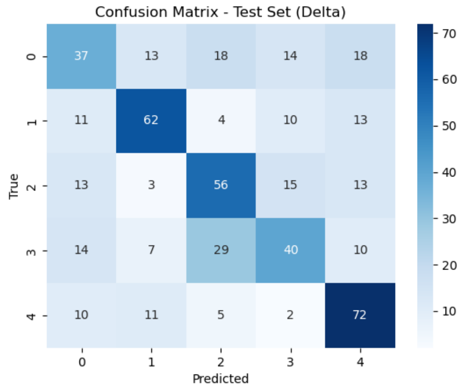


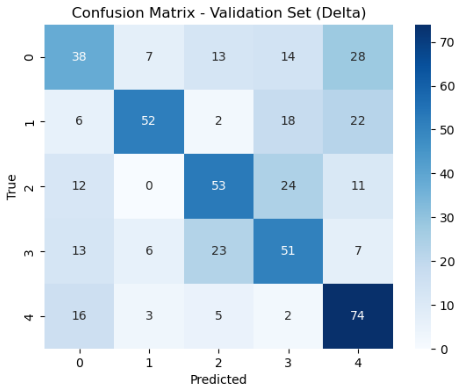
Figure 1.1: Average train and validation error plotted as a function of epochs



(a) Train



(b) Test



(c) Validation

Figure 1.2: Confusion Matrices

1.5.2 Generalized delta rule

The Generalized delta rule is a modified version of gradient descent with an added momentum term. The weight update formula is given by:

$$w(\tau + 1) = w(\tau) - \eta \cdot g_w(\tau) + \alpha \cdot \Delta w(\tau - 1)$$

$$\Delta w(\tau) = w(\tau + 1) - w(\tau)$$

We have denoted the gradient of the error as g_w for convenience, α is the momentum factor and other symbols take their usual meaning. The momentum term leads to faster convergence towards the minimum without causing divergent oscillations. α should be in the range $0 \leq \alpha \leq 1$. A typical value used in practice is $\alpha = 0.9$, and thus we have proceeded with the same.

Best Model Statistics

- Convergence achieved in 33 epochs
- Final Train Loss: 1.1383108459380455
- Final Validation Loss: 1.146220763301477,
- Final Test Loss: 1.1603947933204473

- Validation Accuracy: 53.2%
- H1 = 12, H2 = 12, LR = 0.001, Threshold = 0.001, $\alpha = 0.9$

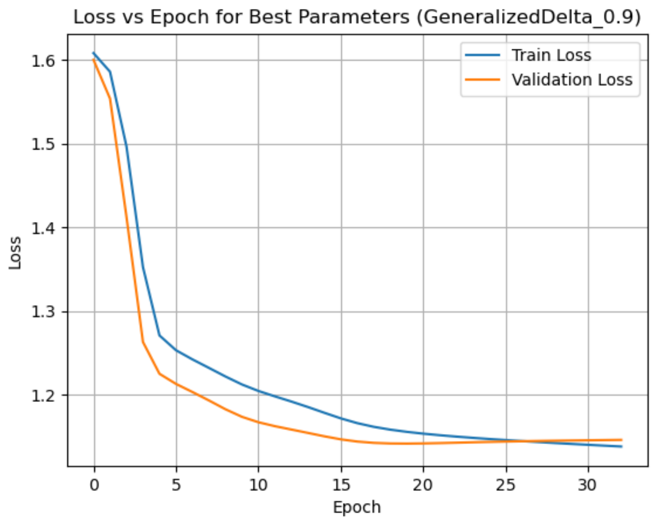
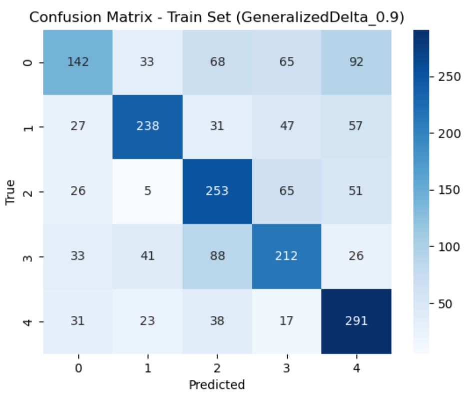
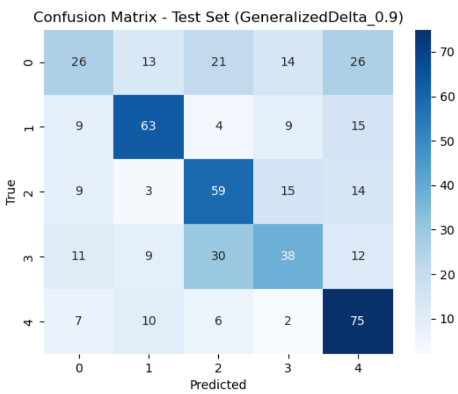


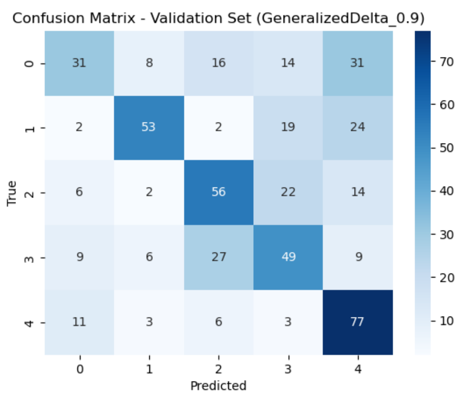
Figure 1.3: Average train and validation error plotted as a function of epochs



(a) Train



(b) Test



(c) Validation

Figure 1.4: Confusion Matrices

1.5.3 AdaGrad

AdaGrad introduces a new set of optimization algorithms that use different learning rates for each parameter in the network. AdaGrad (adaptive gradient) automatically adjusts learning rates during training. It considers past updates by keeping track of the sum of squares of gradients to slow learning for volatile parameters and accelerate it for stable ones. The formula for weight updation is as follows:

$$w(\tau+1) = w(\tau) - \frac{\eta}{\epsilon + \sqrt{\Omega_w(\tau)}} \cdot g_w(\tau)$$

$$\Omega_w(\tau) = g_w^2(0) + g_w^2(1) + ... + g_w^2(\tau-1)$$

The new terms here are Ω_w, which denotes the accumulated sum of squares upto a specific step and ϵ, which is a small constant of the order of 10^{-10} to prevent overflow when Ω_w is very small.

Best Model Statistics

- Did not converge within 200 epochs
- Final Train Loss: 1.0844308452438562
- Final Validation Loss: 1.1657941169235855
- Final Test Loss: 1.1464183717826382
- Validation Accuracy: 52.8%
- H1 = 12, H2 = 6, LR = 0.1, Threshold = 0.0001
- $\epsilon = 10^{-10}$ (default)

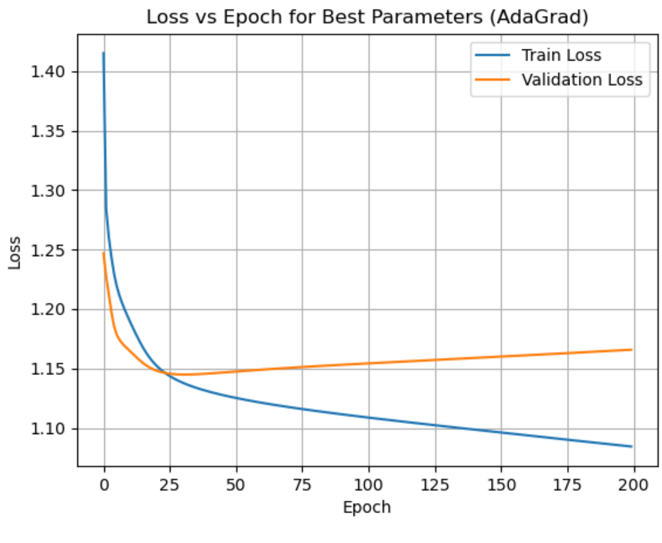
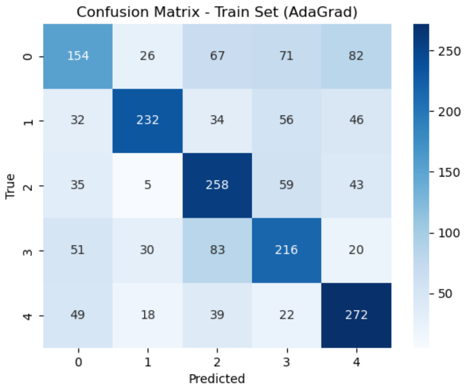
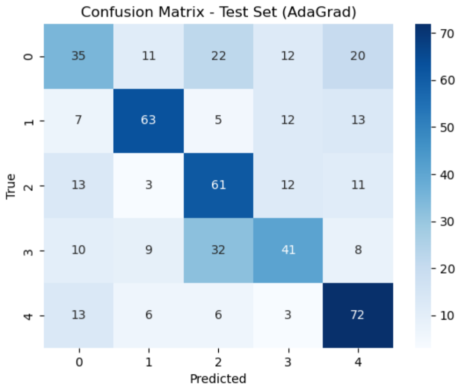


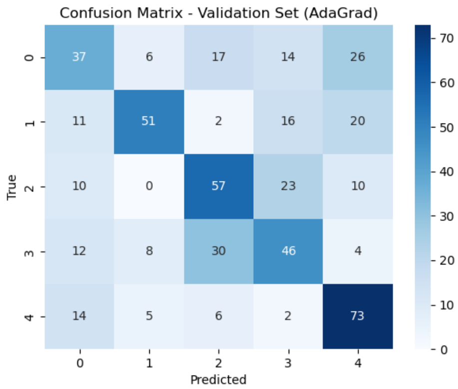
Figure 1.5: Average train and validation error plotted as a function of epochs



(a) Train



(b) Test



(c) Validation

Figure 1.6: Confusion Matrices

1.5.4 RMSProp

$$w(\tau+1) = w(\tau) - \frac{\eta}{\epsilon + \sqrt{\Omega_w(\tau)}} \cdot g_w(\tau)$$

$$\Omega_w(\tau) = \alpha \cdot \Omega_w(\tau-1) + (1-\alpha) \cdot g_w^2(\tau)$$

AdaGrad suffers from constantly accumulating past gradients, causing weight updates to shrink significantly later in training. RMSProp ('root mean square propagation') addresses this by modifying Ω_w to use an exponentially decaying average of squared gradients instead, preventing updates from becoming too small.

Best Model Statistics

- Convergence achieved in 52 epochs
- Final Train Loss: 1.1505444243238512
- Final Validation Loss: 1.1539211451235460
- Final Test Loss: 1.1669691437938210
- Validation Accuracy: 52.6%

- H1 = 6, H2 = 6, LR = 0.001, Threshold = 0.0010, $\alpha = 0.99$
- $\epsilon = 10^{-8}$ (default)

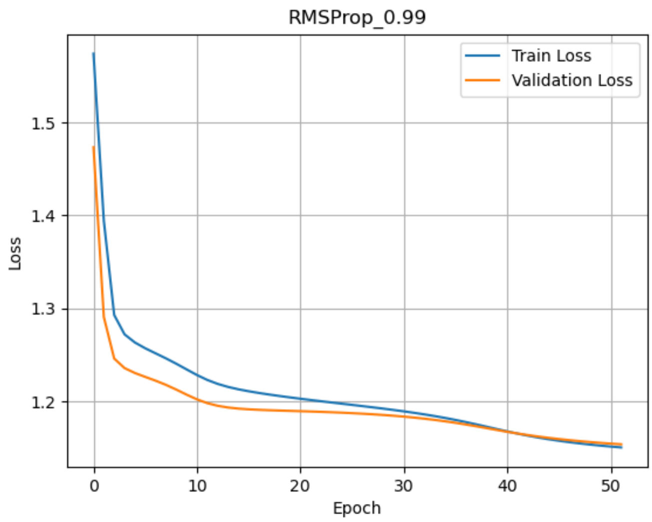
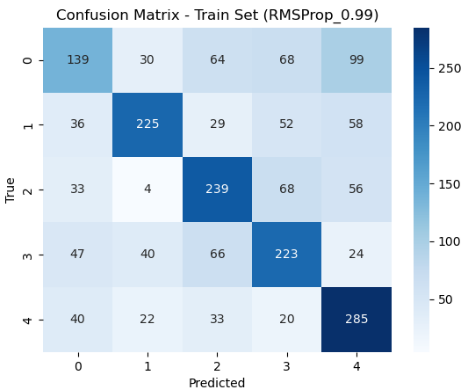
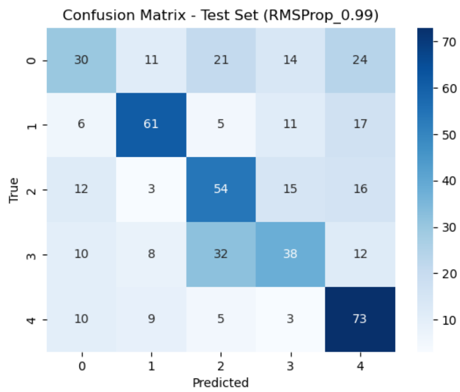


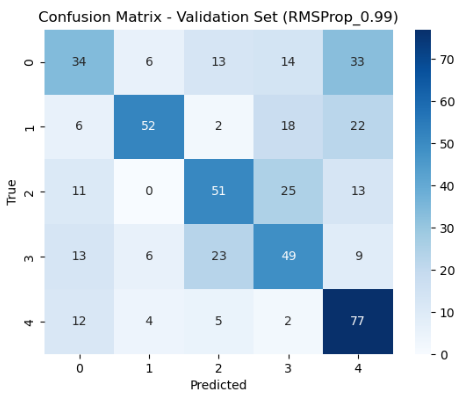
Figure 1.7: Average train and validation error plotted as a function of epochs



(a) Train



(b) Test



(c) Validation

Figure 1.8: Confusion Matrices

1.5.5 Adam

$$w(\tau + 1) = w(\tau) - \frac{\eta}{\epsilon + \sqrt{\hat{\Omega}_w(\tau)}} \cdot \hat{q}_w(\tau)$$

$$\hat{q}_w(\tau) = \frac{q_w(\tau)}{1 - \beta_1^\tau}$$

$$\hat{\Omega}_w(\tau) = \frac{\Omega_w(\tau)}{1 - \beta_2^\tau}$$

$$q_w(\tau) = \beta_1 \cdot q_w(\tau - 1) + (1 - \beta_1) \cdot g_w(\tau)$$

$$\Omega_w(\tau) = \beta_2 \cdot \Omega_w(\tau - 1) + (1 - \beta_2) \cdot g_w^2(\tau)$$

Building on RMSProp, Adam ('adaptive moments') takes things a step further. It incorporates momentum for individual parameters and keeps track of both average gradients and squared gradients using exponentially decaying moving averages. This approach allows Adam to adapt to the specific needs of each parameter during training.

Best Model Statistics

- Convergence achieved in 26 epochs
- Final Train Loss: 1.1383062913916073
- Final Validation Loss: 1.1390214284434914
- Final Test Loss: 1.1593860440086574
- H1 = 6, H2 = 12, LR = 0.001, Threshold = 0.0010
- Validation Accuracy: 54.2%
- $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ (defaults)

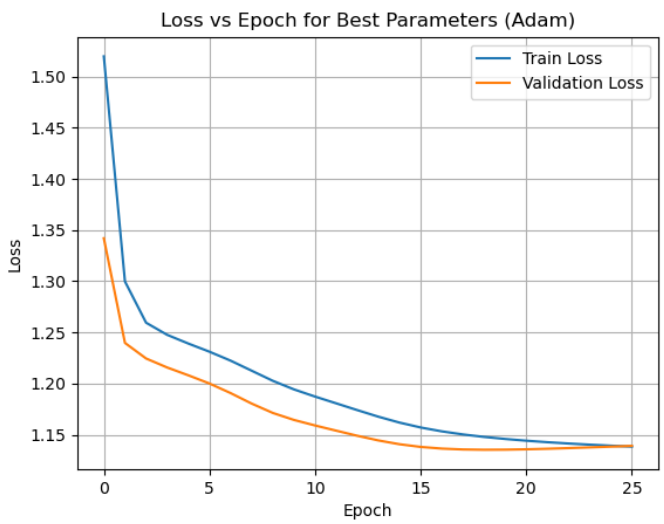
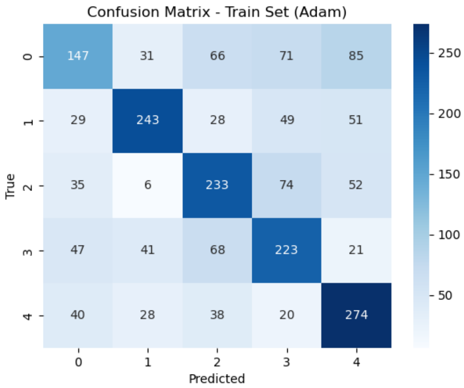
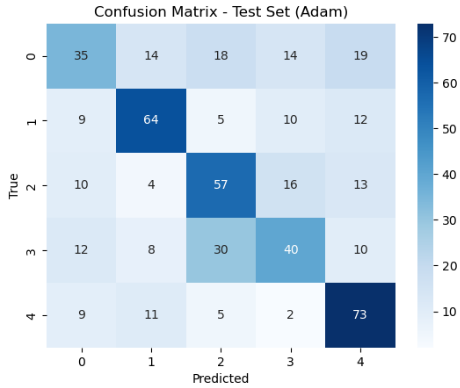


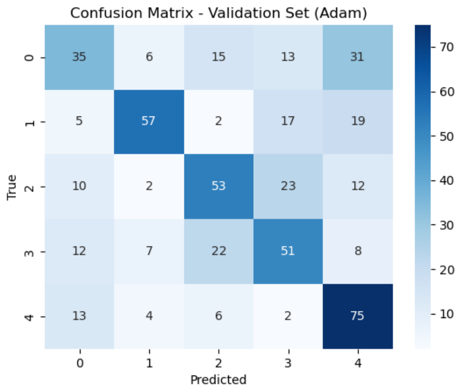
Figure 1.9: Average train and validation error plotted as a function of epochs



(a) Train



(b) Test



(c) Validation

Figure 1.10: Confusion Matrices

1.6 Inferences

From the results of the grid search, inferences can be drawn regarding the performance of models using different optimizers by analysing the number of epochs taken for convergence, validation error, test error, etc.

1.6.1 Optimal Hidden Layer Configuration

The configuration 6, 6 gives the best results as seen from 1.1. The following inferences follow:

- For our small dataset a model with lesser number of parameters tends to do better.
- Reducing the number of nodes gives faster convergence, as well as a better error due to less overfitting.

1.6.2 Optimal Threshold Value

Since we have established 6, 6 as the best hidden layer configuration, we will narrow down the best threshold by filtering models with this constraint.

- As seen in 1.2, a threshold of 10^{-3} gives faster convergence while maintaining marginally better errors.

H1	H2	Average Test Error	Average Epochs to Converge
6	6	1.386041202	60.61904762
6	12	1.411504005	75.54761905
12	6	1.397081868	67.95238095
12	12	1.424798874	67.52380952
12	32	2.149466915	93.54761905
32	12	1.457965184	85.42857143
32	32	2.323936646	90.45238095

Table 1.1: Hidden Layer Configurations vs Average Test Error

- This value makes sense because our dataset is small. A smaller threshold will cause training to continue for a long time and our model will tend to overfit.

Threshold	Average Epochs to Converge	Average Test Error	Average Val Error
0.0001	102.2857143	1.373487451	1.390402782
0.001	18.95238095	1.363025835	1.381679622

Table 1.2: Threshold vs Errors, Epochs for H1,H2 = (6,6)

1.6.3 Optimal Learning Rate and Optimizer

In this section, we analyse the results obtained on varying the learning rate and optimizer while keeping other hyperparameters the same as the optimal values noted before. Thus, we will use the configuration: H1,H2 = (6,6) and Threshold = 10^{-3}. The results are tabulated in 1.3, 1.4 and 1.5.

- **Learning Rate = 0.001**: If we look at the best performing models, Adam, RMSProp and Generalized Delta stand out. However, **Adam converges the fastest** out of the three.

- **Learning Rate = 0.01**: The best performing models are Adam and delta rule. However, Generalized Delta converges extremely fast, possibly to a premature local minima, which is why it does not perform well.

- **Learning Rate = 0.1**: AdaGrad is the best performing model. This is expected, since **AdaGrad benefits from having high learning rate early, which decays as training moves on**. Both Adam and momentum (Generalized Delta) converge very fast, but give unsatisfactory errors.

Optimizer	Epochs	Average Val Error	Average Test Error
AdaGrad	7	1.611774407	1.613162147
Adam	23	1.142660496	1.166562742
Delta	5	1.604440673	1.605431372
GeneralizedDelta ($\alpha = 0.9$)	28	1.154181332	1.176055127
GeneralizedDelta ($\alpha = 0.99$)	6	1.308064276	1.353838506
RMSProp ($\alpha = 0.9$)	21	1.17204437	1.170948014
RMSProp ($\alpha = 0.99$)	26	1.142304598	1.164377681

Table 1.3: Learning Rate = 0.001

Optimizer	Epochs	Average Val Error	Average Test Error
AdaGrad	44	1.230342499	1.244596056
Adam	23	1.197719945	1.241872212
Delta	53	1.166347986	1.16862562
GeneralizedDelta ($\alpha = 0.9$)	16	1.221587814	1.260804864
GeneralizedDelta ($\alpha = 0.99$)	3	1.609558118	1.609558184
RMSProp ($\alpha = 0.9$)	18	1.200554697	1.235001136
RMSProp ($\alpha = 0.99$)	29	1.219413506	1.236596503

Table 1.4: Learning Rate = 0.01

Optimizer	Epochs	Average Val Error	Average Test Error
AdaGrad	42	1.147657152	1.168616685
Adam	3	1.610250643	1.610250789
Delta	19	1.277721049	1.335680015
GeneralizedDelta ($\alpha = 0.9$)	4	1.624348433	1.624346818
GeneralizedDelta ($\alpha = 0.99$)	2	1.623040786	1.623041037
RMSProp ($\alpha = 0.9$)	23	1.62632966	1.672706449
RMSProp ($\alpha = 0.99$)	3	1.733200097	1.733200097

Table 1.5: Learning Rate = 0.1

1.7 Conclusion

From our findings, we can conclude the following about different optimization methods:

- In general, momentum, RMSProp and Adam converge much faster than the regular delta rule.
- AdaGrad performs poorly compared to the others when learning rate is small due to the exploding sum of squares of gradients problem.
- For higher learning rates, Adagrad does well while the others reduce in performance.

2 Task 2

Comparison of Normalization methods for classification on Image dataset 2.

2.1 Data

A brief description of the provided datasets has been given below.

2.1.1 Training Data

The training data has 36 columns representing image features. Therefore, our neural network's input layer should have 36 nodes. There are 2000 rows in the dataset, thus, we have 2000 samples of training data.

There are a total of 5 classes present. Therefore, the output layer of our neural network should have 5 nodes. There is a uniform distribution of class labels over the 2000 data points, i.e. each class has 400 data points. Therefore, we have a balanced dataset which is a good thing since our model gets equal to exposure to all the 5 classes while training.

2.1.2 Validation Data

The validation data has 36 columns representing image features. There are 500 rows of such validation data. The label distribution is uniform, i.e. each class has 100 datapoints.

2.1.3 Test Data

The test data has 36 columns representing image features. There are 500 rows of such test data. The label distribution is uniform, i.e. each class has 100 datapoints.

2.2 Models

We have built 2 Multi Feedforward Neural Networks following the stated specifications.

MLFFNN 1

- Number of nodes in input layer : 36
- Number of hidden layers : 2
- Hidden layer activation function : TanH
- Number of nodes in output layer : 5
- Normalization : Not used
- Softmax applied on output layer

MLFFNN 2

- Number of nodes in input layer : 36
- Number of hidden layers : 2
- Hidden layer activation function : TanH
- Number of nodes in output layer : 5
- Normalization : Post activation Batch Normalization
- Softmax applied on output layer

2.3 Training

Both the models have been trained in the same manner specified as below :

- Mode : Mini batch training
- Optimizer : Adam (Adaptive Moments)
- Loss function : Cross entropy loss
- Stopping criterion : Change in average error below a threshold

2.4 Hyperparameters

Listed below are the various hyperparameters of the models. We have demonstrated results and shown comparisons between the two MLFFNNs for variations in each of these:

- Batch size
- Number of nodes in the first hidden layer
- Number of nodes in the second hidden layer
- Learning rate
- Threshold for stopping criterion

A brief explanation of the various hyperparameters and their expected effects on the model has been provided below:

Batch size : The Batch size refers to the number of samples of data present in each batch during mini batch mode training. Mini-batch training allows for more frequent updates of model parameters compared to batch training, leading to faster convergence, therefore resulting in quicker training times. Mini-batch training often results in better generalization performance compared to stochastic (pattern mode) gradient descent, as it updates the parameters based on the average error of the whole batch instead of a single sample.

Mini-batches therefore provide a compromise between the high variance of stochastic gradient descent and the slow convergence of batch gradient descent.

Number of nodes in the hidden layers : The number of nodes in the hidden layers control the number of learnable parameters of the model. As per theory taught in class, the number of data points should be roughly 10 times the number of parameters in our model. Given that the training dataset has 2000 datapoints, theoretically we should consider using around 4 nodes per hidden layer, however, this is a very small number and drastically reduces the complexity(and hence, predictive capability) of our model.

We have therefore considered higher number of nodes per layer, such that model complexity is increased without making it too complex to train properly with the limited data.

Learning rate : The learning rate is a critical hyperparameter in training neural networks, as it determines the size of the update step taken during gradient descent. The choice of learning rate can significantly impact the training process and the performance of the neural network.

With a very low learning rate, the optimization algorithm progresses slowly, requiring many iterations to converge to a minimum. This can result in longer training times.
A very low learning rate may also cause the process to get stuck in local minima or saddle points. The algorithm may struggle to escape these regions of the parameter space, leading to sub-optimal solutions.

If the learning rate is too high, the gradient descent algorithm may overshoot the minimum of the loss function. This can lead to oscillations or instability in the training process.

Threshold for stopping criterion : The threshold controls the amount of convergence sought for the model. Increasing the threshold leads to early stopping in training but may converge to a sub-optimal model. A lower threshold leads to more training but can also yield an overfitted model.

Keeping the above points in mind, we have chosen appropriate learning rates for the considered models.

2.5 Batch Normalization

Batch Normalization is a technique used in Deep learning to address the internal covariate shift problem, among other benefits.

Internal Covariate Shift

Internal covariate shift refers to the phenomenon where the distribution of the input to each layer of a neural network changes as the parameters of the previous layers change during training. As the network learns, the distribution of activations in earlier layers may shift, making it difficult for later layers to adapt and learn effectively. This can slow down the training process and make it more challenging to optimize the model.

Batch Normalization:

Batch Normalization (BN) is a technique introduced by Sergey Ioffe and Christian Szegedy in their 2015 paper titled "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." BN addresses the internal covariate shift problem by normalizing the inputs to each layer across mini-batches during training.

There are two kinds of Batch Normalization:

Post Activation Batch Normalization : In this method the normalization is done on the outputs of the neuron, i.e. after the activation functions have been applied to the activation value.

Pre Activation Batch Normalization : In this method the normalization is done on the activation values of the neuron, i.e. before the activation functions have been applied to the activation value.

Here's how Batch Normalization works:

Normalization: Depending on the type of Batch Norm, for each node in a hidden layer, the activation values or the outputs are normalized to have zero mean and unit variance across the mini-batch. This is done by subtracting the mean and dividing by the standard deviation of the mini-batch for that feature.

Scaling and Shifting: After normalization, the normalized values are scaled and shifted using learnable parameters (gamma and beta). This allows the model to learn the optimal scale and shift for each feature.

Advantages and Benefits of Using Batch Normalization

Accelerate Convergence & Stabilize Training: By normalizing the inputs to each layer, BN helps to stabilize the training process and accelerate convergence. It reduces the internal covariate shift by ensuring that the distributions of activations remain more consistent across layers during training.

Regularization: BN relies on batch first and second statistical moments (mean and variance) to normalize hidden layers activations. The output values are then strongly tied to the current batch statistics. Such transformation adds some noise, depending on the input examples used in the current batch. Thus, BN also acts as a form of regularization. This noise can help prevent overfitting and improve the generalization performance of the model.

Application: Batch Normalization is applied to hidden nodes of Neural Networks. It is usually used in Convolutional Neural Network architectures where it is quite beneficial since it deals with the high internal covariant shift.

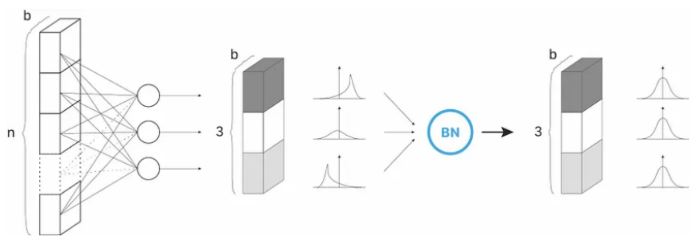


Figure 2.1: Batch norm visualized for a 3 node hidden layer

The mathematics behind Post Activation Batch Normalization is given below:

Let's consider the jth node of a layer.

Let the activation value be a_{nj}

Let the output after applying the activation function ϕ be $s_{nj} = \phi(a_{nj})$

We now perform mean variance normalization on s_{nj} for all examples in the considered batch.

Considering examples belonging to batch D_b, where each such batch has M examples, the mean and variance can be given as :

$$\mu_j = \frac{\sum_{i=1}^{M} x_i}{M}, \qquad x_i \in D_b \tag{2.1}$$

$$\sigma_j^2 = \frac{\sum_{i=1}^{M} (x_i - \mu_j)^2}{M}, \qquad x_i \in D_b \tag{2.2}$$

We now perform mean variance normalization on s_{nj} to obtain $\hat{s}_{nj}$

$$\hat{s}_{nj} = \frac{(s_{nj} - \mu_j)}{\sqrt{\sigma_j^2 + \epsilon}} \tag{2.3}$$

Here, ϵ is a small term included in the denominator to prevent division by 0.

We now perform scaling and shifting using learnable parameters γ_j and β_j as follows:

$$\hat{z}_{nj} = \gamma_j \hat{s}_{nj} + \beta_j \tag{2.4}$$

This final value can now be used to obtain the activations of the next layer.

The only change to be made for pre activation batch normalization is that the activation function ϕ is applied after the mean variance normalization and scaling & shifting operations.

2.6 Tabulated Results

Tabulated results for multiple model configurations have been presented below:

Some of the abbreviated column names have been explained below:

- BS : Batch Size
- H1 : Number of nodes in 1st hidden layer
- H2 : Number of nodes in 2nd hidden layer
- lr : Learning rate
- epochs : Number of epochs taken for convergence

Norm	BS	H1	H2	lr	Threshold	Train Err.	Val Err.	Test Err.	valAcc	testAcc	epochs
No Norm	20	60	30	0.010	0.0010	1.207754	1.277164	1.202844	47.8	50.2	8
Batch Norm	20	60	30	0.010	0.0010	1.062132	1.298281	1.238366	48.8	50.6	15
No Norm	20	60	30	0.010	0.0001	1.251272	1.302821	1.239743	47.2	48.8	4
BatchNorm	20	60	30	0.010	0.0001	0.398709	2.256665	2.337786	47.6	45.0	693
No Norm	20	60	30	0.001	0.0010	1.171193	1.263109	1.194735	51.0	51.0	16
BatchNorm	20	60	30	0.001	0.0010	0.909130	1.251527	1.181905	**51.2**	50.4	32
No Norm	20	60	30	0.001	0.0001	1.162548	1.260884	1.194134	49.0	50.2	21
Batch Norm	20	60	30	0.001	0.0001	0.772332	1.385835	1.345738	49.4	50.4	57
No Norm	20	16	16	0.010	0.0010	1.177355	1.256463	1.195875	48.8	52.6	14
BatchNorm	20	16	16	0.010	0.0010	1.146839	1.275290	1.225671	47.4	50.8	8
No Norm	20	16	16	0.010	0.0001	1.024689	1.304880	1.256950	51.0	50.2	49
Batch Norm	20	16	16	0.010	0.0001	0.755242	1.568754	1.608314	49.0	47.4	350
No Norm	20	16	16	0.001	0.0010	1.169119	1.254193	1.191518	48.6	50.6	25
Batch Norm	20	16	16	0.001	0.0010	1.144261	1.241939	1.197473	50.4	50.4	8
No Norm	20	16	16	0.001	0.0001	1.168745	1.255621	1.187404	50.0	51.6	26
Batch Norm	20	16	16	0.001	0.0001	1.074023	1.220732	1.178493	49.6	51.6	32
No Norm	20	32	32	0.010	0.0010	1.181278	1.278542	1.211011	48.8	47.4	15
Batch Norm	20	32	32	0.010	0.0010	0.821556	1.525941	1.438261	48.6	51.0	48
No Norm	20	32	32	0.010	0.0001	1.194349	1.267632	1.219429	47.4	51.6	11
Batch Norm	20	32	32	0.010	0.0001	0.384269	2.436620	2.586339	44.2	45.4	1000
No Norm	20	32	32	0.001	0.0010	1.164425	1.268434	1.199236	48.8	50.2	19
Batch Norm	20	32	32	0.001	0.0010	0.778872	1.417455	1.324280	47.8	50.8	72
No Norm	20	32	32	0.001	0.0001	1.151025	1.255606	1.196415	48.0	49.6	52
Batch Norm	20	32	32	0.001	0.0001	0.781878	1.379380	1.328276	50.6	51.8	70
No Norm	20	50	20	0.010	0.0010	1.049541	1.316417	1.284769	49.6	48.0	30
BatchNorm	20	50	20	0.010	0.0010	1.014722	1.329396	1.221580	47.6	50.2	21
No Norm	20	50	20	0.010	0.0001	0.118143	4.466846	4.327689	42.4	43.6	525
BatchNorm	20	50	20	0.010	0.0001	0.614054	1.784435	1.701317	48.4	52.0	109
No Norm	20	50	20	0.001	0.0010	1.168026	1.252137	1.189260	50.6	50.8	18
Batch Norm	20	50	20	0.001	0.0010	1.041704	1.233501	1.182289	49.8	51.4	21
No Norm	20	50	20	0.001	0.0001	1.137369	1.270201	1.224625	48.0	50.6	59
Batch Norm	20	50	20	0.001	0.0001	1.005684	1.240888	1.198281	49.8	51.6	28
No Norm	50	60	30	0.010	0.0010	1.115902	1.301283	1.251618	47.2	48.2	28
Batch Norm	50	60	30	0.010	0.0010	0.796018	1.532292	1.408160	46.2	48.6	27
No Norm	50	60	30	0.010	0.0001	0.441454	2.715300	2.398631	41.6	44.2	148
Batch Norm	50	60	30	0.010	0.0001	0.483595	2.207350	2.052321	42.0	47.2	75
No Norm	50	60	30	0.001	0.0010	1.175850	1.266841	1.193278	49.8	51.0	21
Batch Norm	50	60	30	0.001	0.0010	1.151804	1.238014	1.182398	50.8	51.8	6
No Norm	50	60	30	0.001	0.0001	1.148781	1.250115	1.193242	48.2	50.8	56
Batch Norm	50	60	30	0.001	0.0001	0.475857	2.275539	1.977589	42.4	44.2	210
No Norm	50	16	16	0.010	0.0010	1.208125	1.278099	1.209188	48.4	50.0	7
Batch Norm	50	16	16	0.010	0.0010	1.007838	1.304669	1.258320	50.8	49.0	18
No Norm	50	16	16	0.010	0.0001	0.785759	1.702008	1.524447	47.6	47.4	191
Batch Norm	50	16	16	0.010	0.0001	0.705487	1.838795	1.728772	43.6	44.6	446
No Norm	50	16	16	0.001	0.0010	1.170863	1.266066	1.194964	49.0	51.4	37
Batch Norm	50	16	16	0.001	0.0010	1.127883	1.244416	1.199843	49.4	50.6	19
No Norm	50	16	16	0.001	0.0001	1.156414	1.254215	1.189655	48.6	50.2	49
Batch Norm	50	16	16	0.001	0.0001	1.148356	1.239214	1.194174	49.4	50.4	11

Norm	BS	H1	H2	lr	Threshold	Train Err.	Val Err.	Test Err.	valAcc	testAcc	epochs
No Norm	50	32	32	0.010	0.0010	1.183324	1.280598	1.203999	46.8	51.0	10
Batch Norm	50	32	32	0.010	0.0010	0.878872	1.399010	1.426750	47.4	49.4	22
No Norm	50	32	32	0.010	0.0001	1.132428	1.281571	1.245635	47.8	50.8	32
Batch Norm	50	32	32	0.010	0.0001	0.261009	3.561026	3.461811	44.0	45.4	700
No Norm	50	32	32	0.001	0.0010	1.168788	1.266080	1.192135	49.0	50.0	24
Batch Norm	50	32	32	0.001	0.0010	1.112488	1.235129	1.170664	49.6	52.0	14
NoNorm	50	32	32	0.001	0.0001	1.152643	1.251422	1.188040	48.2	51.8	45
Batch Norm	50	32	32	0.001	0.0001	0.734895	1.439034	1.383883	46.8	50.2	79
No Norm	50	50	20	0.010	0.0010	1.023096	1.347977	1.271930	49.2	48.6	45
Batch Norm	50	50	20	0.010	0.0010	0.798794	1.453429	1.451633	48.8	48.2	41
No Norm	50	50	20	0.010	0.0001	1.036272	1.360966	1.244153	48.6	48.8	40
Batch Norm	50	50	20	0.010	0.0001	0.248883	3.207605	3.487207	45.2	44.0	364
No Norm	50	50	20	0.001	0.0010	1.160569	1.250072	1.191787	49.8	51.2	29
Batch Norm	50	50	20	0.001	0.0010	1.133583	1.220267	1.180986	**52.2**	51.4	9
No Norm	50	50	20	0.001	0.0001	1.149922	1.250370	1.188853	49.8	50.6	42
Batch Norm	50	50	20	0.001	0.0001	0.630696	1.965772	1.669783	43.0	48.4	314
No Norm	100	60	30	0.010	0.0010	1.147696	1.273219	1.213376	49.0	50.2	20
Batch Norm	100	60	30	0.010	0.0010	0.517630	2.043882	1.833265	45.0	48.4	51
No Norm	100	60	30	0.010	0.0001	0.675040	1.608164	1.488777	45.8	49.2	103
Batch Norm	100	60	30	0.010	0.0001	0.138395	4.520395	3.933847	44.4	49.6	229
No Norm	100	60	30	0.001	0.0010	1.180497	1.272788	1.193888	50.4	50.8	27
Batch Norm	100	60	30	0.001	0.0010	1.131223	1.242719	1.188234	49.6	50.2	11
No Norm	100	60	30	0.001	0.0001	1.147761	1.251128	1.187692	49.4	52.2	63
Batch Norm	100	60	30	0.001	0.0001	0.692885	1.377497	1.315155	50.0	50.8	87
No Norm	100	16	16	0.010	0.0010	1.148648	1.282320	1.206711	49.0	49.2	28
Batch Norm	100	16	16	0.010	0.0010	0.886351	1.371735	1.404883	47.2	47.6	34
No Norm	100	16	16	0.010	0.0001	1.130867	1.284975	1.220923	46.2	48.8	43
Batch Norm	100	16	16	0.010	0.0001	0.823427	1.620089	1.580509	47.8	46.8	61
No Norm	100	16	16	0.001	0.0010	1.189721	1.271549	1.205583	48.4	51.6	42
Batch Norm	100	16	16	0.001	0.0010	1.146686	1.246493	1.196553	50.8	48.8	14
No Norm	100	16	16	0.001	0.0001	1.153889	1.253503	1.189401	48.6	50.4	72
Batch Norm	100	16	16	0.001	0.0001	1.106710	1.239658	1.196463	49.6	50.6	38
No Norm	100	32	32	0.010	0.0010	1.224315	1.287412	1.209012	48.2	48.6	5
Batch Norm	100	32	32	0.010	0.0010	0.881675	1.402665	1.353303	49.0	49.2	17
No Norm	100	32	32	0.010	0.0001	0.267071	3.265985	3.234976	45.2	42.8	268
Batch Norm	100	32	32	0.010	0.0001	0.657929	1.859139	1.772046	45.2	45.4	38
No Norm	100	32	32	0.001	0.0010	1.179025	1.274243	1.199827	48.4	49.8	29
Batch Norm	100	32	32	0.001	0.0010	1.131065	1.232962	1.179465	50.0	50.2	12
No Norm	100	32	32	0.001	0.0001	1.159308	1.267950	1.195472	48.6	50.0	42
Batch Norm	100	32	32	0.001	0.0001	0.680675	1.482980	1.464494	46.0	48.0	113
No Norm	100	50	20	0.010	0.0010	1.156977	1.248272	1.200188	49.6	51.4	15
Batch Norm	100	50	20	0.010	0.0010	0.993806	1.327361	1.293867	47.8	48.2	12
No Norm	100	50	20	0.010	0.0001	0.664327	1.895096	1.668270	44.0	48.0	165
Batch Norm	100	50	20	0.010	0.0001	0.212877	3.794892	3.692150	43.6	43.0	272
No Norm	100	50	20	0.001	0.0010	1.181534	1.266025	1.202224	49.4	51.0	29
Batch Norm	100	50	20	0.001	0.0010	1.081370	1.218078	1.181239	**52.4**	52.2	25
No Norm	100	50	20	0.001	0.0001	1.157907	1.252785	1.191659	50.2	51.6	41
Batch Norm	100	50	20	0.001	0.0001	0.836653	1.387723	1.314084	47.8	49.4	93

2.7 Plots and Results for Optimal Model Configurations

We now provide the plots for average training error vs epoch as well as the confusion matrices for the best models obtained among the explored configurations.

We have considered highest validation accuracy as the criteria for choosing the best models.

After extensively exploring various configurations and different sets of values of hyperparameters, we have found that the top 3 models with the best validation accuracy all use **Batch Normalization**.

The hyperparameter configurations and performance metrics for the best 3 models as per highest validation accuracy criteria have been tabulated below:

BS	H1	H2	lr	th	trainerr	valerr	testerr	valacc	testacc	epochs
100	50	20	0.001	0.0010	1.081370	1.218078	1.181239	52.4	52.2	25
50	50	20	0.001	0.0010	1.133583	1.220267	1.180986	52.2	51.4	9
20	60	30	0.001	0.0010	0.909130	1.251527	1.181905	51.2	50.4	32

Table 2.1: Metrics for best 3 overall models, they all happen to use Batch Normalization

Thus, we can conclude that Batch Normalization indeed improves the generalization capabilities of the model and helps in stable and quicker training by dealing with the issue of internal covariant shift.

The metrics for the corresponding No Norm models have been provided in the table below. We have included the plots and confusion matrices for the same as well.

BS	H1	H2	lr	th	trainerr	valerr	testerr	valacc	testacc	epochs
100	50	20	0.001	0.0010	1.181534	1.266025	1.202224	49.4	51.0	29
50	50	20	0.001	0.0010	1.160569	1.250072	1.191787	49.8	51.2	29
20	60	30	0.001	0.0010	1.171193	1.263109	1.194735	51.0	51.0	16

Table 2.2: Metrics for corresponding model configurations without Batch Normalization

The confusion matrices as well as the error vs epoch plots for the best 3 models alongwith the corresponding No Normalization models have been provided on the following pages:

Normalization	Batch Size	h1	h2	Learning rate	Threshold
Batch Normalization	100	50	20	0.001	0.001

Table 2.3: The graphs below have been plotted for the above model configuration:

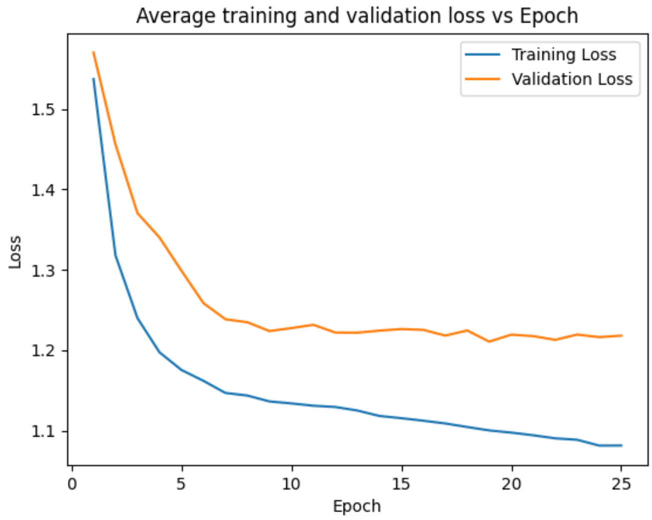
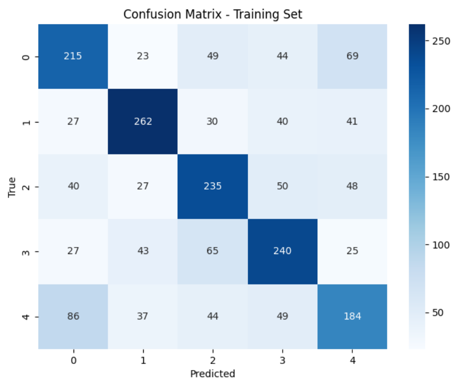
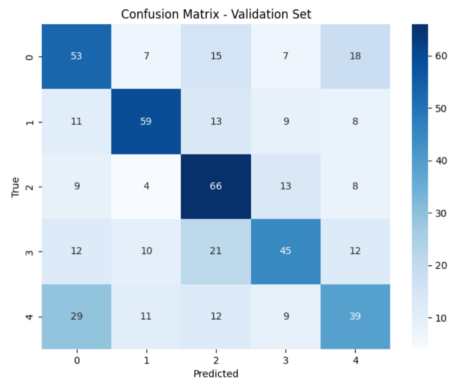


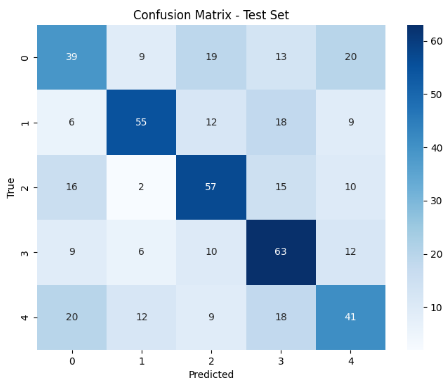
Figure 2.2: Average train and validation error plotted as a function of epochs



(a) Train



(b) Validation



(c) Test

Figure 2.3: Confusion Matrices

Normalization	Batch Size	h1	h2	Learning rate	Threshold
No Normalization	100	50	20	0.001	0.001

Table 2.4: The graphs below have been plotted for the above model configuration:

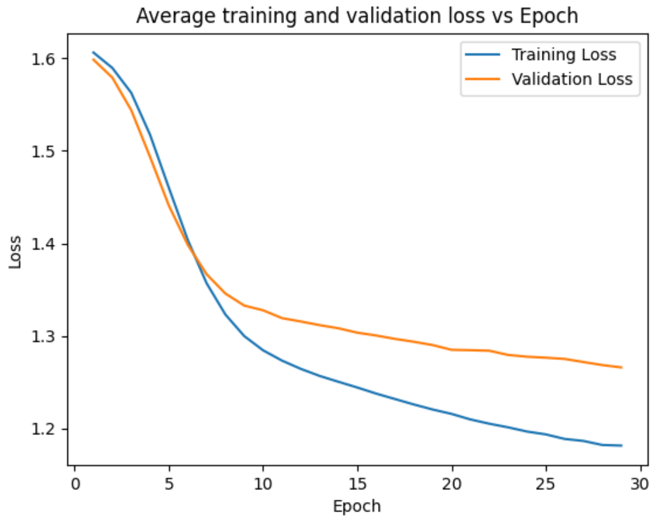
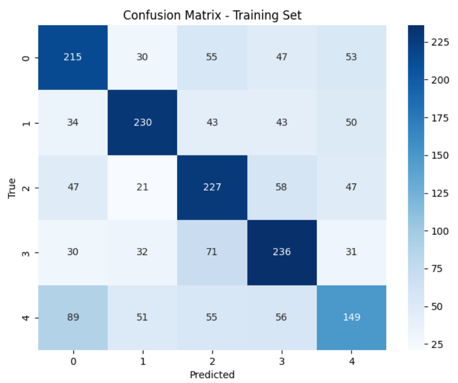
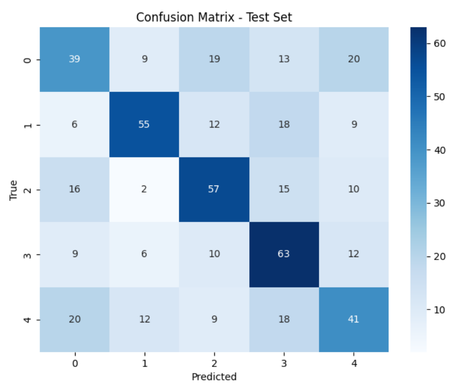


Figure 2.4: Average train and validation error plotted as a function of epochs



(a) Train

(b) Validation



(c) Test

Figure 2.5: Confusion Matrices

Normalization	Batch Size	h1	h2	Learning rate	Threshold
Batch Normalization	50	50	20	0.001	0.001

Table 2.5: The graphs below have been plotted for the above model configuration:

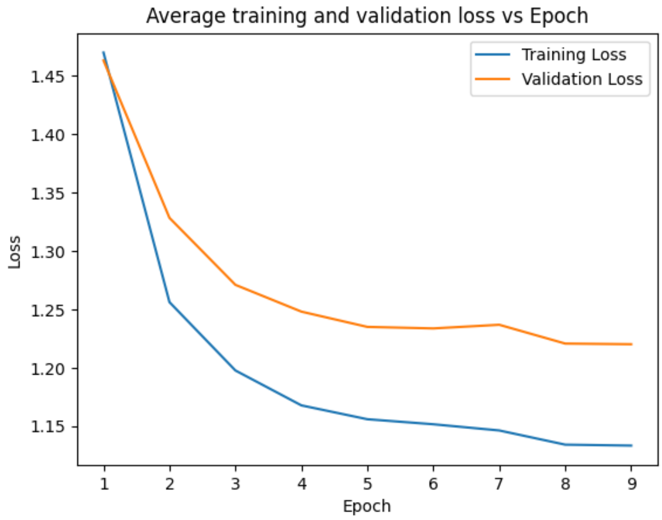
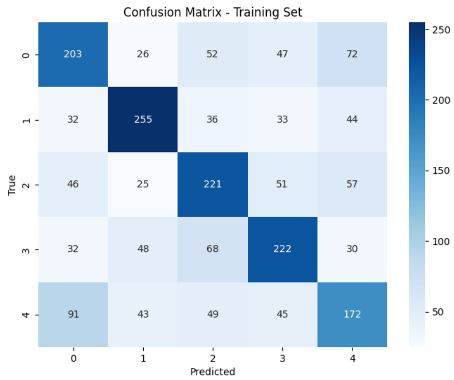
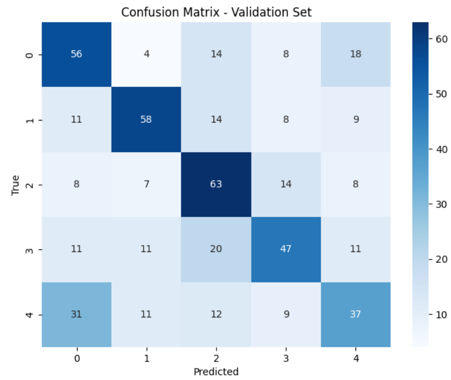


Figure 2.6: Average train and validation error plotted as a function of epochs



(a) Train



(b) Validation

(c) Test

Figure 2.7: Confusion Matrices

Normalization	Batch Size	h1	h2	Learning rate	Threshold
No Normalization	50	50	20	0.001	0.001

Table 2.6: The graphs below have been plotted for the above model configuration:

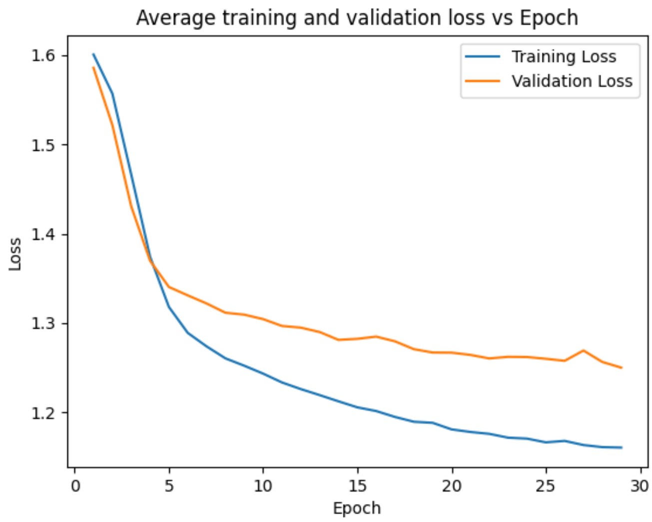
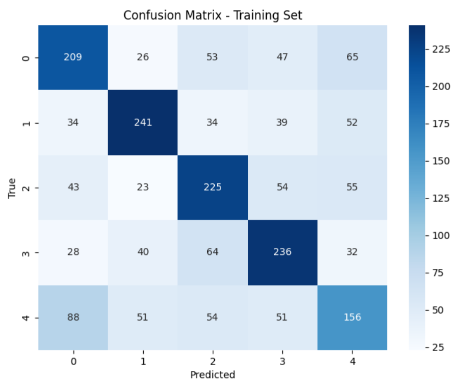
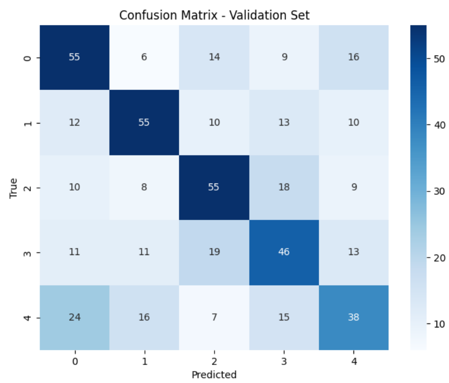


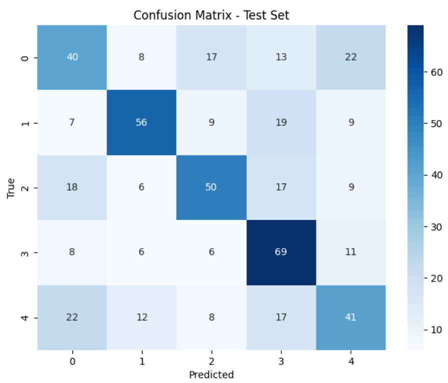
Figure 2.8: Average train and validation error plotted as a function of epochs



(a) Train



(b) Validation



(c) Test

Figure 2.9: Confusion Matrices

Normalization	Batch Size	h1	h2	Learning rate	Threshold
Batch Normalization	20	60	30	0.001	0.001

Table 2.7: The graphs below have been plotted for the above model configuration:

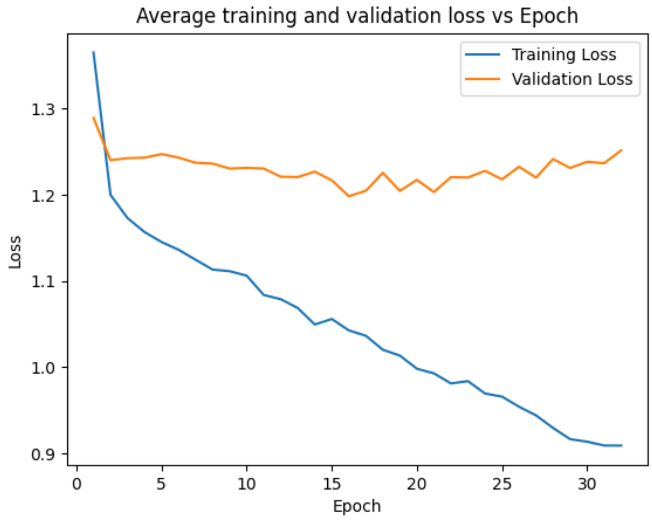
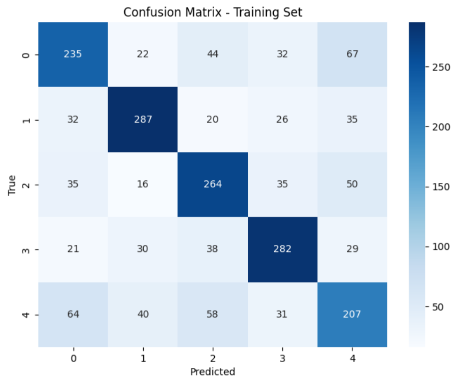
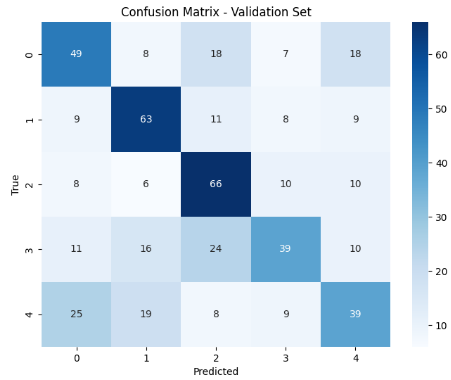


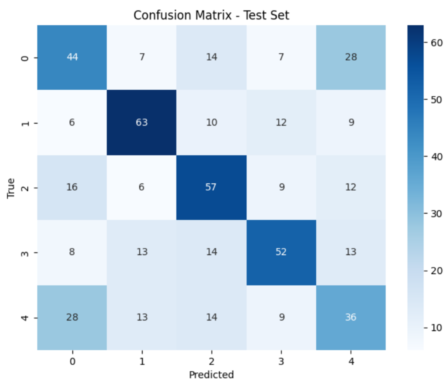
Figure 2.10: Average train and validation error plotted as a function of epochs



(a) Train



(b) Validation



(c) Test

Figure 2.11: Confusion Matrices

Normalization	Batch Size	h1	h2	Learning rate	Threshold
No Normalization	20	60	30	0.001	0.001

Table 2.8: The graphs below have been plotted for the above model configuration:

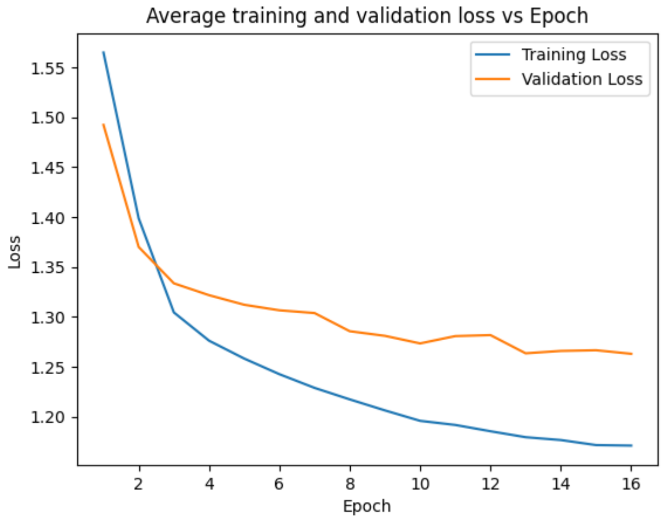
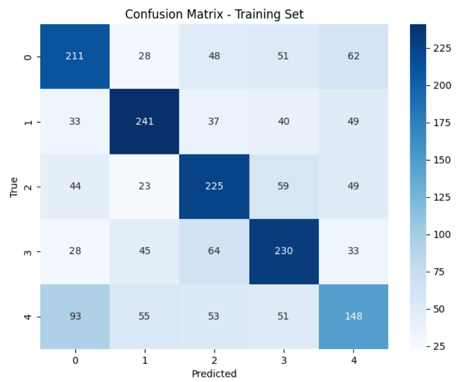
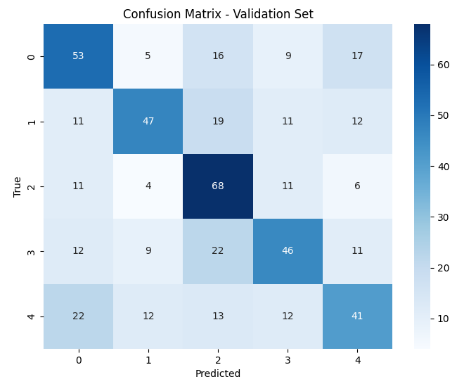


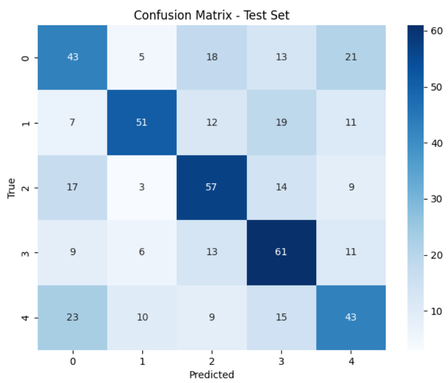
Figure 2.12: Average train and validation error plotted as a function of epochs



(a) Train



(b) Validation



(c) Test

Figure 2.13: Confusion Matrices

2.8 Inferences

Below, we observe the data and draw inferences regarding the performance of models with and without Batch Normalization by analysing Epochs for Convergence, Validation Error, Accuracy values, etc.

2.8.1 Average Number of epochs for convergence

Dependence on Batch Size and Threshold values

Table 2.9: Avg. Epochs for convergence for different combinations of Threshold, Batch Size, and Normalization methods

Threshold	Batch Size	Norm	Average Epochs for Convergence
0.0010	20	BatchNorm	21.125
		NoNorm	22.125
	50	BatchNorm	19.500
		NoNorm	25.125
	100	BatchNorm	22.000
		NoNorm	24.375
0.0001	20	BatchNorm	292.375
		NoNorm	93.375
	50	BatchNorm	274.875
		NoNorm	75.375
	100	BatchNorm	116.375
		NoNorm	99.625

Above, we have tabulated the average number of epochs required for convergence by the models with and without Batch Normalization for different batch sizes and thresholds.

- On observing the above data, it is clear that Batch Normalization gives faster convergence when an appropriate threshold is used.
- For thresholds as low as 10^{-4}, models using Batch Normalization tend to overfit and train for a very large number of epochs.
- Another inference is that, among the batch sizes considered, larger batch sizes tend to favour faster convergence when batch normalization is used.

2.8.2 Validation error

Batch Size	Norm	Val error
20	BatchNorm	1.246105
	NoNorm	1.259468
50	BatchNorm	1.234457
	NoNorm	1.262265
100	BatchNorm	1.235063
	NoNorm	1.271151

Table 2.10: Validation error for various batch sizes with lr = 0.001

From the table above, we can infer that Using Batch Normalization gives us lower validation error values, implying that using Batch Norm helps the model generalize better. This can be attributed to the regularization effect that Batch Normalization has due to introduced batch specific noise, i.e. since the statistics of each batch are different, there is noise introduced in the training process due to normalization of each such batch.

2.8.3 Validation Accuracy

H1	H2	Norm	Val Accuracy
16	16	BatchNorm	49.33
		NoNorm	48.70
32	32	BatchNorm	48.73
		NoNorm	48.33
50	20	BatchNorm	49.77
		NoNorm	49.70
60	30	BatchNorm	49.60
		NoNorm	49.20

Table 2.11: Hidden Layer configurations vs Validation Accuracies

As can be seen above, For multiple hidden layer configurations, models with batch norm tend to outperform models without it.

Batch Size	Norm	Val Accuracy
20	BatchNorm	49.80
	NoNorm	49.75
50	BatchNorm	50.50
	NoNorm	49.40
100	BatchNorm	50.70
	NoNorm	49.15

Table 2.12: Val. Acc. values for different batch sizes with lr = 0.001

After observing the data, we can infer that Batch Normalization gives better validation accuracy values. Thus, Batch Normalization improves the generalization capabilities of the model. This can be attributed to the regularization effect of Batch Normalization as explained previously.

2.8.4 Convergence for wide range of Learning Rates

Learning Rate	Norm	Epochs
0.001	BatchNorm	20.250000
0.010	BatchNorm	26.166667

Table 2.13: Epochs for Convergence vs Learning rate

As we can see, using any of the 2 learning rates(which are an order apart) results in quick convergence. Thus we can infer that Batch Normalization helps in fast convergence over a wide range of learning rates.

2.9 Conclusion

Given the above extensive analysis, we can conclude that using Batch Normalization improves our model's performance. It does so in the following ways:

- Reduces Internal Covariant Shift thus resulting in quicker convergence
- Also acts as a form or regularization by adding noise based on current batch statistics.

3 Task 3

Comparison of pre-trained and non-pretrained AANN based DFNN models for classification on Image dataset 3.

3.1 Data

A brief description of the provided datasets has been given below.

3.1.1 Training Data

The training data has 36 columns representing image features. Therefore, our neural network's input layer should have 36 nodes. There are two kinds of training data, labelled and unlabelled. The labelled data has 750 datapoints and the unlabelled version has 1750 datapoints.

There are a total of 5 classes present. Therefore, the output layer of our DFNN should have 5 nodes. There is a uniform distribution of class labels over the lebelled training data. Therefore, we have a balanced dataset which is a good thing since our model gets equal to exposure to all the 5 classes while training. We further ensure this with a custom uniform dataloader to ensure equal distribution of class examples in each mini batch.

3.1.2 Validation Data

The validation data has 36 columns representing image features. There are 250 rows of such validation data.

3.1.3 Test Data

The test data has 36 columns representing image features. There are 250 rows of such test data.

3.2 Models

We have built 2 Autoencoder based DFNN networks with a three stacked autoencoders following the stated specifications:

AANN 1 (encoder)

- Number of nodes in input layer : 36
- Number of hidden layers : 1
- Number of bottleneck layers : 1
- Hidden layer activation function : TanH
- Bottleneck layer activation function : Linear
- Number of nodes in output layer : 36
- Normalization : Not used

AANN 2 (encoder)

- Number of nodes in input layer : l1
- Number of hidden layers : 1
- Number of bottlencek layers : 1
- Hidden layer activation function : TanH
- Bottleneck layer activation function : Linear
- Number of nodes in output layer : l1
- Normalization : Not used

AANN 3 (encoder)

- Number of nodes in input layer : l2
- Number of hidden layers : 1
- Number of bottlencek layers : 1
- Hidden layer activation function : TanH
- Bottleneck layer activation function : Linear
- Number of nodes in output layer : l2
- Normalization : Not used

DFNN 1

- Number of nodes in input layer : 36
- Number of hidden layers : 3
- Hidden layer activation function : TanH
- Number of nodes in output layer : 5
- Normalization : Not used
- Softmax applied on output layer
- Pretrained : No

DFNN 2

- Number of nodes in input layer : 36
- Number of hidden layers : 3
- Hidden layer activation function : TanH
- Number of nodes in output layer : 5
- Normalization : Not used
- Softmax applied on output layer
- Pretrained : Yes

3.3 Training

Both the models have been trained in the same manner specified as below :

- Mode : Mini batch training
- Optimizer : Adam (Adaptive Moments)
- Loss function : Cross entropy loss
- Stopping criterion : Change in average training error below a threshold

3.4 Hyperparameters

Listed below are the various hyperparameters of the models. We have demonstrated results and shown comparisons between the two DFNNs for variations in each of these:

- Sizes of autoencoder hidden layers
- Sizes of autoencoder bottleneck layers
- Epochs of training
- Learning rate
- Threshold for stopping criterion

A brief explanation of the various hyperparameters and their expected effects on the models has been provided below:

Sizes of autoencoder hidden layers : The hidden layer size determines the capacity of the autoencoder to learn complex patterns in the data. Larger hidden layer sizes generally increase the model's capacity to capture intricate features, but they also increase the risk of overfitting, especially if the amount of available data is limited.

Sizes of autoencoder bottleneck layers : The bottleneck layer compresses the input data into a lower-dimensional representation. This reduction in dimensionality can help in capturing the most salient features of the input data while discarding less informative features.

Epochs of training : Increasing the maximum epochs for training can allow model configurations to reach the threshold condition with greater probability.

Learning rate : With a very low learning rate, the optimization algorithm progresses slowly. This can result in longer training times. A very low learning rate may also cause the process to get stuck in local minima or saddle points, leading to sub-optimal solutions. If the learning rate is too high, the gradient descent algorithm may overshoot the minimum of the loss function. This can lead to oscillations or instability in the training process.

Threshold for stopping criterion : The threshold controls the amount of convergence sought for the model. Increasing the threshold leads to early stopping in training but may converge to a sub-optimal model. A lower threshold leads to more training but can also yield an overfitted model.

3.5 Autoencoder (AANN) and Pre-training

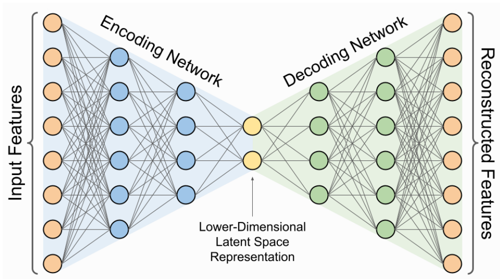


Figure 3.1: Architecture of an AANN

An auto-associative neural network, also called AANN or autoencoder, is a network trained on unlabelled data, with input and expected output being the same. This is also known as auto-associative mapping. Some salient features of an autoencoder are:

- The model learns a compressed representation of the input data in the bottleneck layer.
- This learnt dimension reduction is a non-linear compression.
- The hidden layer activation is non linear, generally TanH and the bottleneck layer has linear activation to prevent the compressed representation from being confined to a finite range space.

Autoencoders can be stacked and an output layer added beyond the stack to get a DFNN model for classification or regression tasks. The autoencoders are trained seperately on unlabelled data and then the entire DFNN is **fine-tuned on labelled data** in the pre-training process. During the construction of the DFNN, **linear layers are merged** by a multiplication of the weight matrices preceding and following them in order to reduce the number of trained parameters in the fine-tuning process.

3.6 Results

Shown below are the results of the **top 23** pretrained and non-pretrained versions of the DFNN with their counterparts. The first table has the top pretrained models and the second, the top non-pretrained ones. The best in each case is indicated in bold. (max ep = maximum epochs)

Mode	h1	b1	h2	b2	h3	b3	lr	max ep	threshold	val acc (%)	val loss	train loss
non pretrained	34	30	26	20	16	14	0.0001	250	1e-06	56.4	1.118	0.0357
pretrained	34	30	26	20	16	14	0.0001	250	1e-06	57.6	1.122	0.0342
non pretrained	34	30	24	20	18	14	0.0001	100	1e-05	50.8	1.1788	0.037
pretrained	34	30	24	20	18	14	0.0001	100	1e-05	58.4	1.1238	0.0348
non pretrained	34	30	24	20	18	14	0.0001	250	1e-06	52.0	1.1587	0.0364
pretrained	34	30	24	20	18	14	0.0001	250	1e-06	58.0	1.1198	0.0347
non pretrained	34	30	24	20	16	14	0.0001	100	1e-05	48.8	1.1516	0.0377
pretrained	34	30	24	20	16	14	0.0001	100	1e-05	58.0	1.0871	0.0344
non pretrained	34	28	26	22	18	14	0.0001	100	1e-06	51.6	1.1265	0.0373
pretrained	34	28	26	22	18	14	0.0001	100	1e-06	58.4	1.1149	0.0349
non pretrained	34	28	26	22	18	12	0.0001	500	1e-05	51.6	1.1182	0.0374
pretrained	34	28	26	22	18	12	0.0001	500	1e-05	57.6	1.1195	0.0352
non pretrained	34	28	26	20	18	14	0.0001	100	1e-06	50.0	1.1450	0.0371
pretrained	**34**	**28**	**26**	**20**	**18**	**14**	**0.0001**	**100**	**1e-06**	**59.2**	**1.119**	**0.0344**
non pretrained	34	28	26	20	18	14	0.0001	100	1e-05	51.6	1.1476	0.0373
pretrained	34	28	26	20	18	14	0.0001	100	1e-05	58.4	1.1246	0.0348
non pretrained	34	28	26	20	18	14	0.0001	500	1e-06	51.6	1.1372	0.0373
pretrained	34	28	26	20	18	14	0.0001	500	1e-06	58.0	1.1337	0.0351
non pretrained	34	28	26	20	18	12	0.0001	250	1e-05	50.8	1.1235	0.0375
pretrained	34	28	26	20	18	12	0.0001	250	1e-05	58.0	1.1279	0.0348
non pretrained	34	28	26	20	16	14	0.001	100	1e-05	54.4	1.1872	0.0295
pretrained	34	28	26	20	16	14	0.001	100	1e-05	57.6	1.1388	0.0316
non pretrained	34	28	26	20	16	12	0.0001	250	1e-06	52.4	1.1342	0.0372
pretrained	34	28	26	20	16	12	0.0001	250	1e-06	58.0	1.137	0.0347
non pretrained	34	28	24	20	16	12	0.001	250	1e-06	54.0	1.2191	0.0283
pretrained	34	28	24	20	16	12	0.001	250	1e-06	57.6	1.1831	0.031
non pretrained	32	30	26	20	16	12	0.001	100	1e-06	53.2	1.1959	0.0297
pretrained	32	30	26	20	16	12	0.001	100	1e-06	58.0	1.1349	0.0308
non pretrained	32	30	26	20	16	12	0.001	500	1e-06	54.8	1.1383	0.0301
pretrained	32	30	26	20	16	12	0.001	500	1e-06	58.4	1.1239	0.0327
non pretrained	32	30	24	22	16	14	0.0001	250	1e-06	52.8	1.1427	0.0365
pretrained	32	30	24	22	16	14	0.0001	250	1e-06	58.0	1.1379	0.0347
non pretrained	32	30	24	22	16	14	0.0001	500	1e-06	50.8	1.1023	0.0371
pretrained	32	30	24	22	16	14	0.0001	500	1e-06	58.4	1.1348	0.0348
non pretrained	32	30	24	22	16	14	0.0001	500	1e-05	51.2	1.143	0.0378
pretrained	32	30	24	22	16	14	0.0001	500	1e-05	58.8	1.1391	0.0348
non pretrained	32	30	24	22	16	12	0.0001	250	1e-06	51.6	1.1289	0.0368
pretrained	32	30	24	22	16	12	0.0001	250	1e-06	57.6	1.1528	0.0357
non pretrained	32	30	24	22	16	12	0.001	500	1e-05	53.6	1.1403	0.0366
pretrained	32	30	24	22	16	12	0.001	500	1e-05	58.0	1.1219	0.031
non pretrained	32	30	24	20	16	14	0.001	250	1e-05	54.4	1.1173	0.0359
pretrained	32	30	24	20	16	14	0.001	250	1e-05	58.8	1.0946	0.0308
non pretrained	32	28	26	20	16	14	0.001	500	1e-06	51.2	1.1206	0.0354
pretrained	32	28	26	20	16	14	0.001	500	1e-06	57.6	1.1285	0.0313
non pretrained	32	28	24	22	18	12	0.0001	100	1e-06	50.8	1.1236	0.0372
pretrained	32	28	24	22	18	12	0.0001	100	1e-06	58.8	1.0959	0.0349

Mode	h1	b1	h2	b2	h3	b3	lr	max ep	threshold	val acc (%)	val loss	train loss
non pretrained	34	30	26	20	16	12	0.001	250	1e-05	57.6	1.1095	0.0343
pretrained	34	30	26	20	16	12	0.001	250	1e-05	53.2	1.2172	0.0306
non pretrained	34	30	26	20	16	12	0.001	500	1e-06	57.6	1.1186	0.0345
pretrained	34	30	26	20	16	12	0.001	500	1e-06	54.8	1.2004	0.0302
non pretrained	34	30	26	20	16	12	0.001	500	1e-05	57.6	1.12	0.0342
pretrained	34	30	26	20	16	12	0.001	500	1e-05	54.8	1.1755	0.0295
non pretrained	**34**	**28**	**26**	**22**	**18**	**14**	**0.001**	**500**	**1e-05**	**58.0**	**1.061**	**0.035**
pretrained	34	28	26	22	18	14	0.001	500	1e-05	56.0	1.1133	0.0317
non pretrained	34	28	26	22	18	12	0.001	100	1e-05	57.6	1.116	0.0354
pretrained	34	28	26	22	18	12	0.001	100	1e-05	54.0	1.238	0.0297
non pretrained	34	28	26	22	18	12	0.001	250	1e-05	56.8	1.1129	0.0365
pretrained	34	28	26	22	18	12	0.001	250	1e-05	55.6	1.1211	0.0299
non pretrained	34	28	26	22	16	14	0.01	500	1e-05	57.2	1.2349	0.0255
pretrained	34	28	26	22	16	14	0.01	500	1e-05	48.0	2.5947	0.0023
non pretrained	34	28	26	20	18	14	0.001	100	1e-06	57.2	1.1865	0.0292
pretrained	34	28	26	20	18	14	0.001	100	1e-06	55.2	1.1748	0.0301
non pretrained	34	28	26	20	18	14	0.001	250	1e-06	57.8	1.0554	0.0326
pretrained	34	28	26	20	18	14	0.001	250	1e-06	54.8	1.1457	0.0298
non pretrained	34	28	26	20	16	14	0.001	500	1e-05	56.8	1.1193	0.0313
pretrained	34	28	26	20	16	14	0.001	500	1e-05	56.0	1.2015	0.0307
non pretrained	32	30	26	20	16	14	0.001	500	1e-05	57.6	1.0987	0.0331
pretrained	32	30	26	20	16	14	0.001	500	1e-05	53.2	1.1937	0.0265
non pretrained	32	30	24	22	18	14	0.001	250	1e-05	57.2	1.1408	0.0313
pretrained	32	30	24	22	18	14	0.001	250	1e-05	54.8	1.1346	0.0311
non pretrained	32	30	24	22	16	14	0.001	100	1e-06	57.2	1.1092	0.0337
pretrained	32	30	24	22	16	14	0.001	100	1e-06	55.2	1.0862	0.03
non pretrained	32	30	24	20	18	14	0.001	500	1e-06	57.6	1.1056	0.0319
pretrained	32	30	24	20	18	14	0.001	500	1e-06	55.6	1.1536	0.0303
non pretrained	32	30	24	20	18	12	0.001	250	1e-05	57.2	1.0948	0.0336
pretrained	32	30	24	20	18	12	0.001	250	1e-05	55.2	1.1645	0.0292
non pretrained	32	28	24	22	16	14	0.001	500	1e-05	57.2	1.1473	0.0303
pretrained	32	28	24	22	16	14	0.001	500	1e-05	55.6	1.2094	0.0296
non pretrained	32	28	24	22	16	12	0.01	100	1e-05	57.6	1.1195	0.0349
pretrained	32	28	24	22	16	12	0.01	100	1e-05	46.4	2.8847	0.0035
non pretrained	32	28	24	22	16	12	0.001	500	1e-06	57.2	1.2206	0.026
pretrained	32	28	24	22	16	12	0.001	500	1e-06	54.0	1.1812	0.028
non pretrained	32	28	24	20	18	14	0.001	250	1e-06	57.6	1.0917	0.0325
pretrained	32	28	24	20	18	14	0.001	250	1e-06	54.8	1.1398	0.0296
non pretrained	32	28	24	20	18	14	0.001	250	1e-05	56.8	1.0799	0.0323
pretrained	32	28	24	20	18	14	0.001	250	1e-05	55.6	1.1839	0.0298
non pretrained	32	28	24	20	18	14	0.001	500	1e-06	57.8	1.0848	0.034
pretrained	32	28	24	20	18	14	0.001	500	1e-06	52.8	1.1425	0.0298
non pretrained	32	28	24	20	16	14	0.001	250	1e-05	57.6	1.0953	0.0328
pretrained	32	28	24	20	16	14	0.001	250	1e-05	55.6	1.1466	0.0306
non pretrained	32	28	24	20	16	14	0.001	500	1e-05	57.6	1.0955	0.0313
pretrained	32	28	24	20	16	14	0.001	500	1e-05	53.2	1.2229	0.0282

Given the length of report required to fully encapsulate the results of the hyperparameter tuning process in the report, we have included only the top 23 model and hyperparameter configurations for each DFNN. Attached here (hyperlinked) is a google drive link to the csv with the complete training results across all the hyperparameter values considered.

The grid of hyperparameters considered were:

Table 3.1: Hyperparameter Search space

Hyerparameter	Considered Values
AANN 1 hidden layer size	34, 32
AANN 1 bottleneck layer size	30, 28
AANN 2 hidden layer size	26, 24
AANN 2 bottleneck layer size	22, 20
AANN 3 hidden layer size	18, 16
AANN 3 bottleneck layer size	14, 12
Learning rate	0.0001, 0.001, 0.01
Threshold	1e-6, 1e-5
Max Epochs	100, 250, 500

Bottleneck size for AANN 3 was decided using the PCA variance rule. The variance threshold was set to 75 and 70 % to get 14 and 12 dimensions respectively. Although 95 % was tried, the required dimension was 30 which is impractical considering an input dimension of 36. The required compressed dimensions as per the PCA rule for various values of total variance from input data are shown in the table below:

Table 3.2: Variance vs Required dimensions

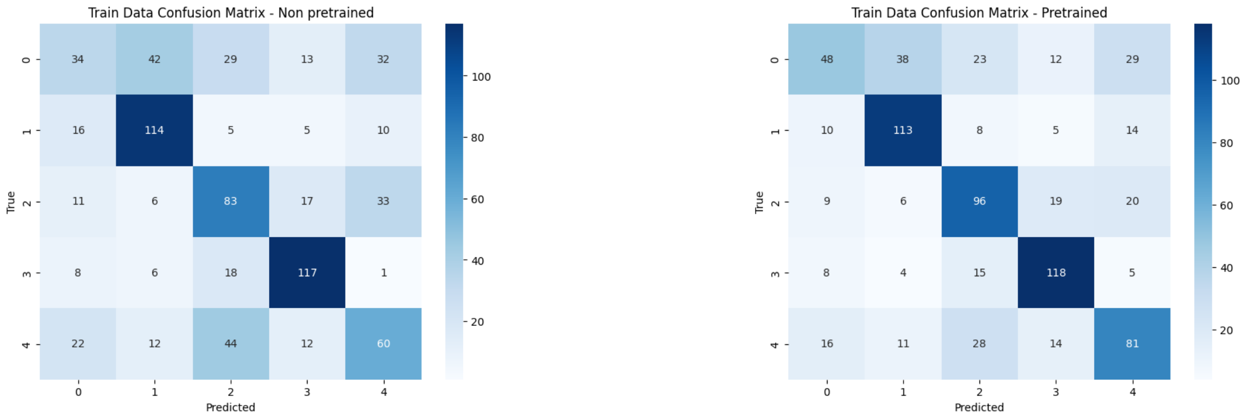
% of total variance in input	Required final dimension
95	30
90	25
85	20
80	17
75	14
70	12

We now present the confusion matrices for the best model in the pretrained and non-pretrained sets along with their counterparts:

3.6.1 Best pretrained model configuration and corresponding non pretrained model:

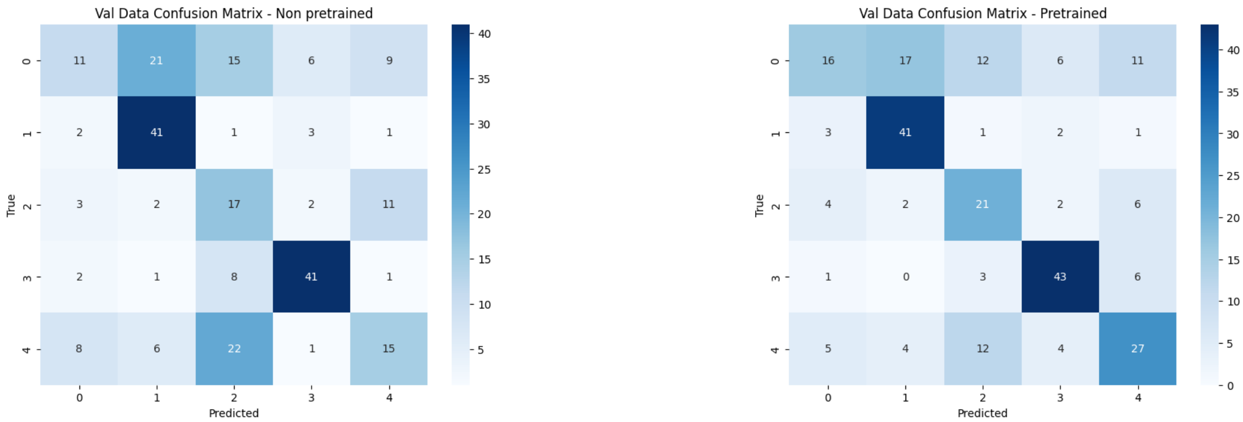
Table 3.3: Results on test data

Model	Loss	Accuracy (%)
Pretrained	1.062	57.6
Non pretrained	1.170	50.4



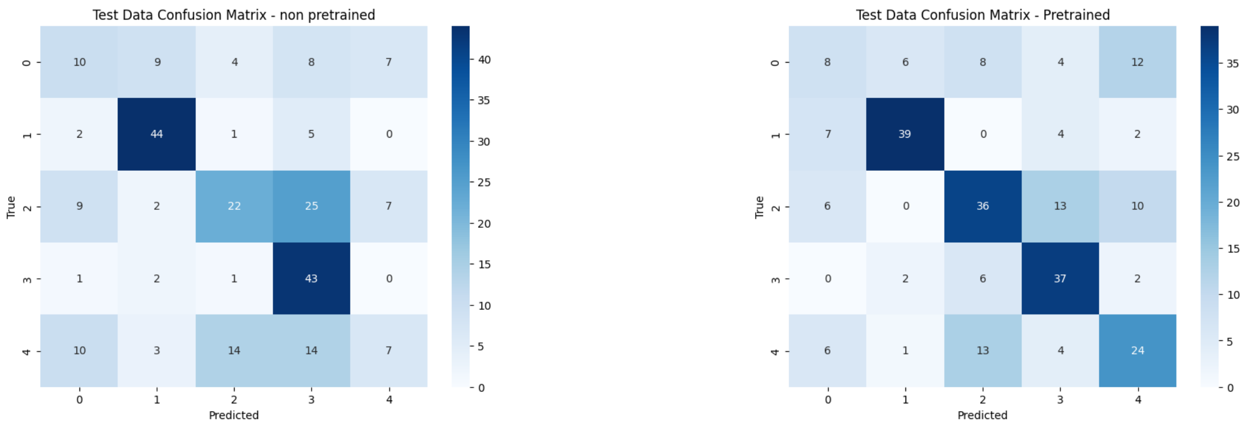
(a) Non pretrained Model (b) Pretrained model

Figure 3.2: Training data confusion matrices



(a) Non pretrained Model (b) Pretrained model

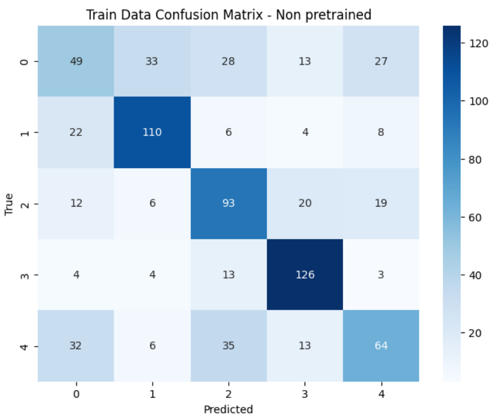
Figure 3.3: Val data confusion matrices



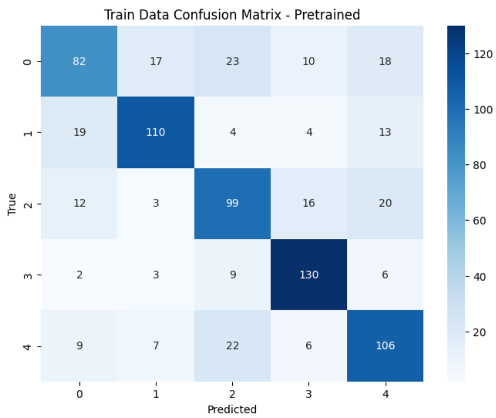
(a) Non pretrained Model (b) Pretrained model

Figure 3.4: Test data confusion matrices

3.6.2 Best non pretrained model configuration and corresponding pretrained model:

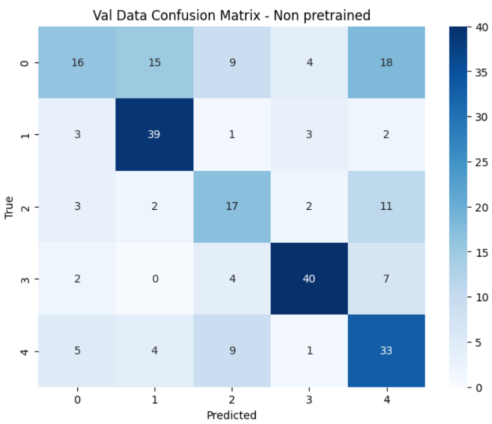


(a) Non pretrained Model

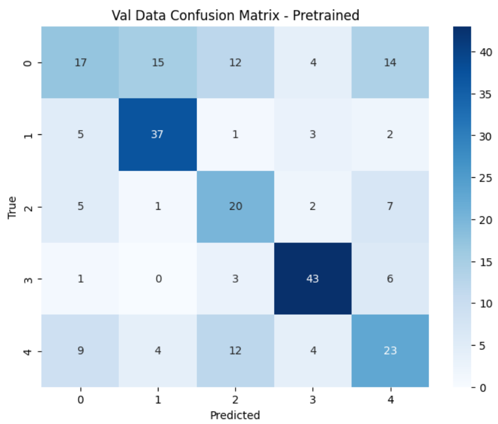


(b) Pretrained model

Figure 3.5: Training data confusion matrices

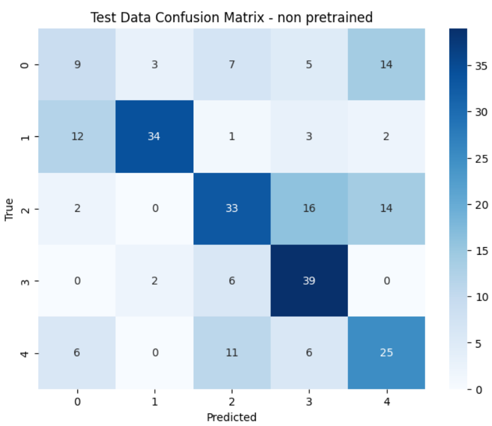


(a) Non pretrained Model

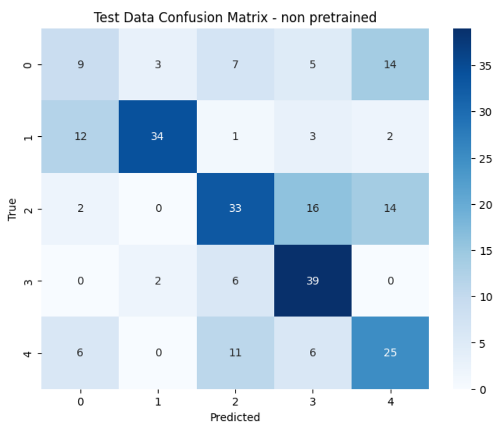


(b) Pretrained model

Figure 3.6: Val data confusion matrices



(a) Non pretrained Model



(b) Pretrained model

Figure 3.7: Test data confusion matrices

Table 3.4: Results on test data

Model	Loss	Accuracy (%)
Non pretrained	1.046	56.0
Pretrained	1.059	54.8

3.7 Inferences

From the results of the various models, we can infer the following points:

- In general, pre-training the autoencoders and then fine tuning yields better accuracy than training from scratch on only unlabelled data. The average val accuracies for both kinds of models can be seen below:

  Table 3.5: Performance of models

Model	Average val accuracies(%)
Pretrained	52.86
Non pretrained	51.10

- The performance of one or the other depends on the hyperparameters being used:
  - With a lower learning rate, the pre-trained version generally outperforms the non pre-trained version. Most of the best performing pre-trained models are on the lowest learning rate tried. This can be attributed to the fact that when fine tuning a model, using a high learning rate can lead to oscillations around the minima since the model starts relatively closer to the minima than the non pre-trained version where the weights are randomly initialized. This can be seen below (on the basis of top three average accuracies across epochs and threshold):

    Table 3.6: Performance of models across various learning rates

Learning rate	Top three val accuracies(%)	Model
0.0001	57.67	pretrained
	57.60	pretrained
	57.33	pretrained
0.001	56.33	non pretrained
	56.27	non pretrained
	56.07	non pretrained
0.01	52.20	non pretrained
	51.40	pretrained
	51.33	pretrained

  - Higher threshold is better since it prevents the pretrained model from overfitting. This can be seen below as the top accuracies are higher for 1e-5 than for 1e-6 (on the basis of top three average accuracies across epochs and learning rates):

Table 3.7: Performance of models across various thresholds

Threshold	Top three val accuracies(%)	Model
1e-6	54.40	pretrained
	54.22	pretrained
	54.18	pretrained
1e-5	54.71	pretrained
	54.62	pretrained
	54.31	pretrained

- A higher final bottleneck dimension yields higher average accuracies across other parameters as seen in the table below:

Table 3.8: Performance of models across various learning rates

Model	Bottleneck dimension	Average val accuracies(%)
Pretrained	12	52.84
	14	52.88
Non pretrained	12	51.06
	14	51.14

This can be attributed to the fact that a higher representation dimension retains more features from the image and is hence easier to classify for both kinds of models.

3.8 Conclusion

Given the above analysis, we can conclude that pretraining the autoencoders before fine tuning improves our model's performance in the following ways:

- Improves initialization of weights of the DFNN to be better than random initialization.
- Reduces the amount of the labelled data required for training the DFNN.
- Increases convergence rate as was observed during training wherein many times, the pre-trained DFNN did not need max epochs to converge as per the threshold rule.