Model Logs

Dataset:

https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols/code

Approach:

1. Read an image of a linear equation.
2. Get the bounding box for each individual characters.
3. Extract the bounding box from the image.
4. Preprocess every extracted sub-image – S'
5. Load the saved model which is stored in pickle format.
6. Run every S' through model.
7. Get the predictions for each image.
8. Use the sympy for solving the equation.

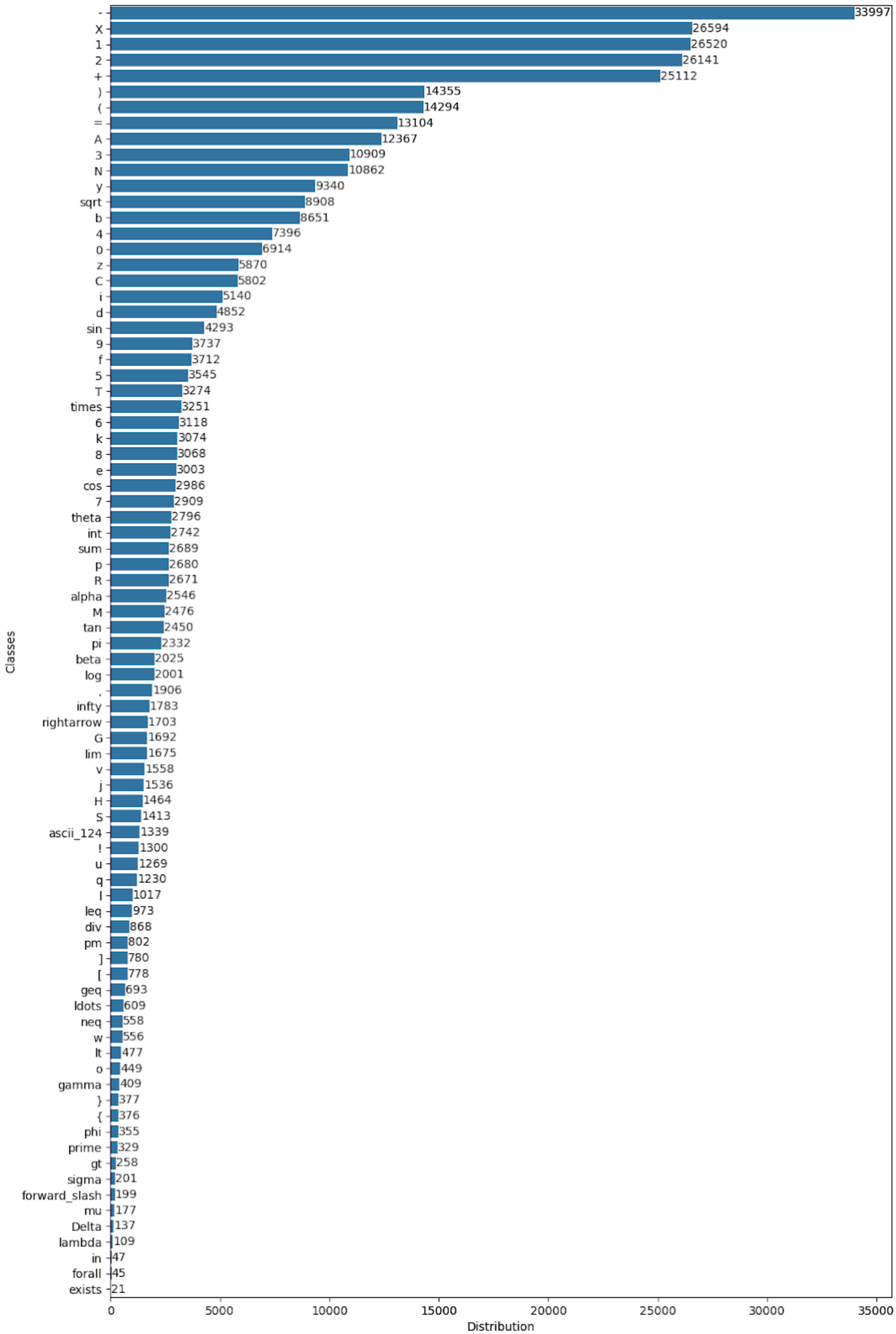
Implementation:

- Fully connect Deep neural network
- RELU activation function
- Softmax activation function
- Cross entropy loss function
- Convolutional neural network
- Flattening layer
- MaxPool layer.
- Sequential Model

Challenges:

- The biggest hurdle was implementing the convolution neural network especially the backpropagation algorithm for convolution algorithm.
- Even though the algorithm is straightforward it was tough to implement as we were frequently encountering issue in managing the gradients for each activation layer.
- Vectorizing the code was another challenge to solve, NumPy really helped in performing the vectorization and speeding up the code.
- In order to validate the correct working of each layer, we tried training each layer individually on the known dataset such as MNIST, after we validated that the layer is able to learn features by looking into it's validation accuracy during training we integrated it our sequential model. We frequently faced issues while integrating the layers together too.

Data Distribution:



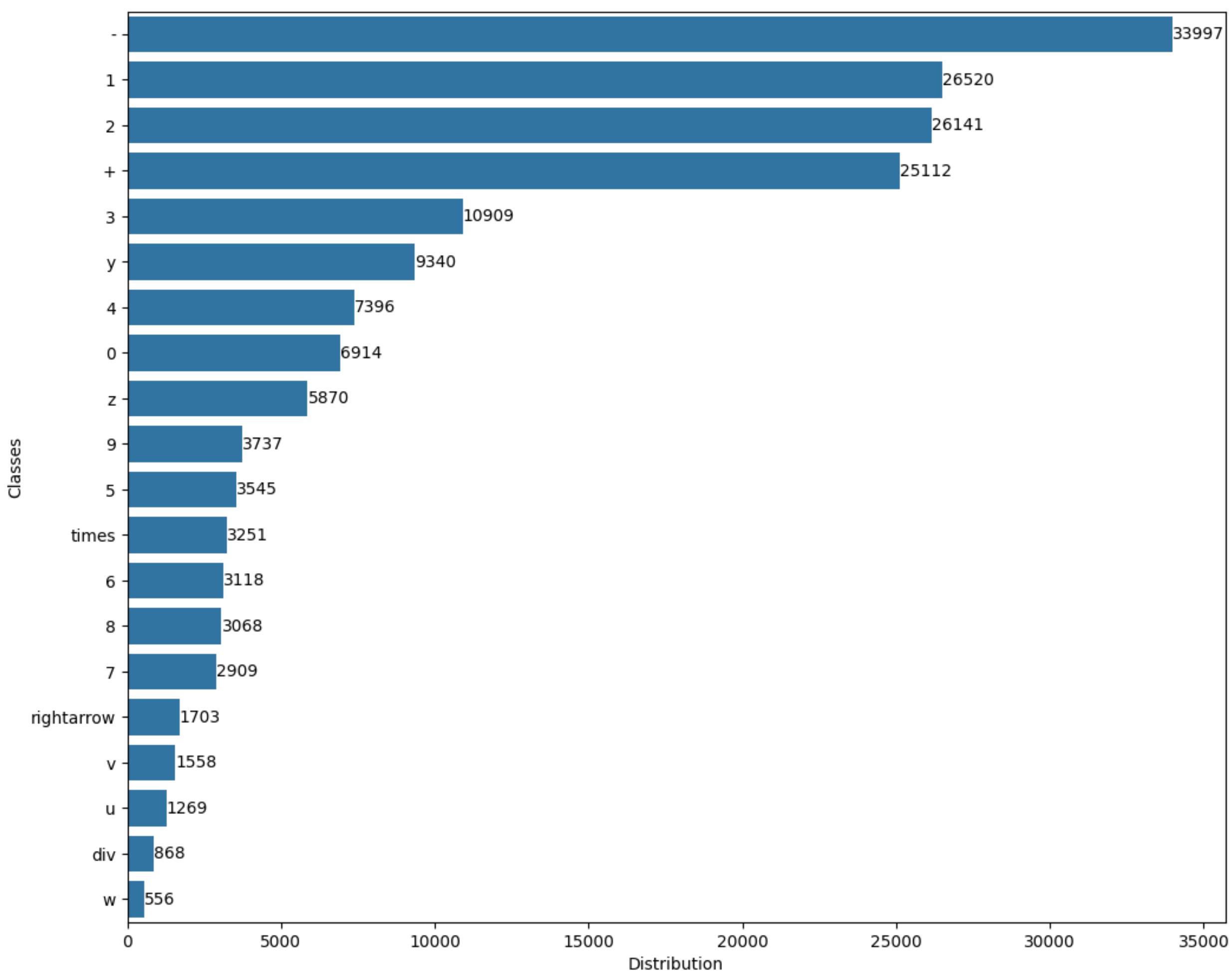
Classes
Distribution
- 33997
X 26594
1 26520
2 26141
+ 25112
) 14355
(14294
= 13104
A 12367
3 10909
N 10862
y 9340
sqrt 8908
b 8651
4 7396
0 6914
z 5870
C 5802
i 5140
d 4852
sin 4293
9 3737
f 3712
5 3545
T 3274
times 3251
6 3118
k 3074
8 3068
e 3003
cos 2986
7 2909
theta 2796
int 2742
sum 2689
p 2680
R 2671
alpha 2546
M 2476
tan 2450
pi 2332
beta 2025
log 2001
, 1906
infty 1783
rightarrow 1703
G 1692
lim 1675
v 1558
j 1536
H 1464
S 1413
ascii_124 1339
! 1300
u 1269
q 1230
l 1017
leq 973
div 868
pm 802
] 780
[778
geq 693
ldots 609
neq 558
w 556
lt 477
o 449
gamma 409
} 377
{ 376
phi 355
prime 329
gt 258
sigma 201
forward_slash 199
mu 177
Delta 137
lambda 109
in 47
forall 45
exists 21
0 5000 10000 15000 20000 25000 30000 35000

The class distribution is not balanced, so we may need to use data augmentation to balance the dataset.

Main classes include:
Digits: 0 to 9
Characters: - a - z (some include upper case characters)
Math operators: - +, -, /, *, =
It also includes basic Greek alphabet symbols like: alpha, beta, gamma, mu, sigma, phi and theta.
English alphanumeric symbols are included.
All math operators, set operators.
Basic pre-defined math functions like: log, lim, cos, sin, tan.
Number of classes - 82
Features - As it is a gray scale image dataset the features are just the number of pixels present in an image, so it will be 2025 pixels / features.

Data Preprocessing:

We didn't choose all the 82 classes from the original dataset; we chose the below 20 classes for our use case.



Results & Analysis:

We’ve implemented 4 models with different architecture & parameters:

Model 1: Flatten1, Dense1, Softmax, Cross Entropy

Model Info

Layer Name-> Flatten 1

Layer Name -> Dense_1

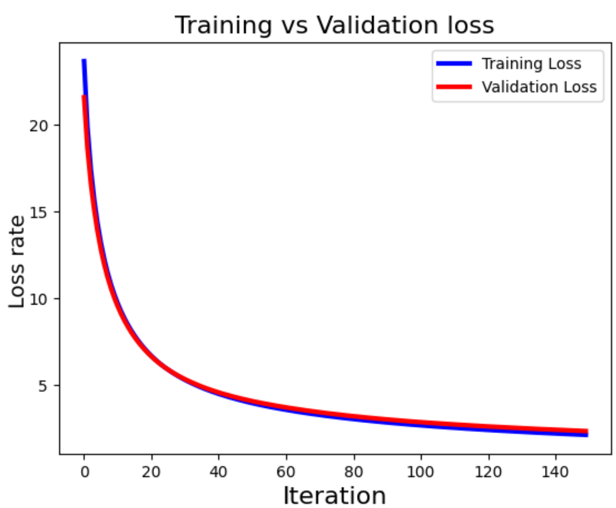
Weights shape -> (2025, 20)

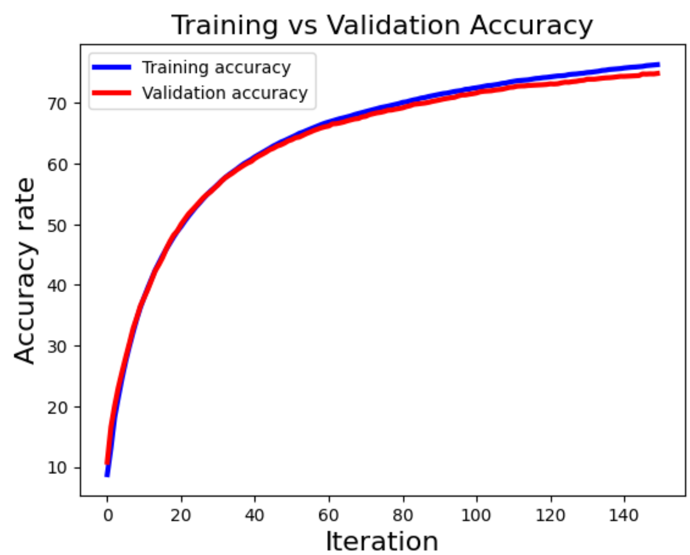
Model	Epochs	Batch Size	Learning Rate	Train Accuracy (%)	Validation Accuracy (%)	Train Loss	Validation Loss	Test Accuracy (%)	Test Loss (%)
Model 1	5	16	0.0001	4.86	5.04	28.75	28.43	-	-
Model 1	5	16	0.001	7.07	8.00	24.58	24.02	-	-
Model 1	20	16	0.001	15.70	15.99	18.19	18.38	-	-
Model 1	20	16	0.001	18.79	18.78	16.23	16.21	-	-
Model 1	30	16	0.01	55.83	56.54	5.51	5.38	-	-
Model 1	60	16	0.01	66.40	65.95	3.67	3.79	-	-
Model 1	100	16	0.01	72.45	71.55	2.70	2.89	-	-
Model 1.1	150	16	0.01	76.34	74.89	2.14	2.36	75.07	2.34
Model 1.2	20	16	0.01	84.88	85.01	0.60	0.61	84.02	0.62

Class	Precision	Recall	F1-Score	Support
0	0.92	0.95	0.93	1368
1	0.83	0.91	0.87	1385
2	0.80	0.80	0.80	1452
3	0.90	0.90	0.90	1439
4	0.76	0.83	0.79	1391
5	0.89	0.80	0.84	702

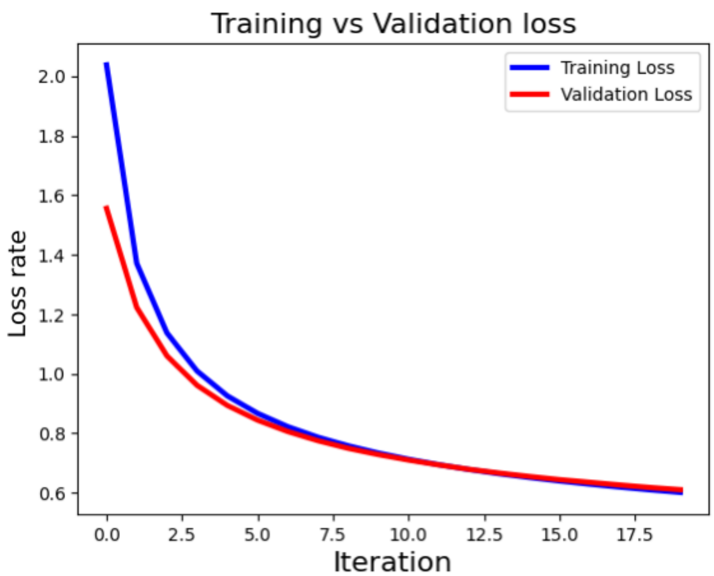
6	0.92	0.88	0.90	646
7	0.89	0.84	0.87	576
8	0.84	0.79	0.82	637
9	0.83	0.85	0.84	764
div	0.99	0.44	0.60	179
rightarrow	0.91	0.80	0.85	352
times	0.87	0.81	0.84	702
u	0.62	0.42	0.50	256
v	0.79	0.69	0.74	295
w	0.72	0.64	0.68	104
y	0.78	0.84	0.81	1329
z	0.76	0.72	0.74	1135
+	0.80	0.83	0.82	1378
-	0.94	0.99	0.96	1385
Accuracy	0.84			

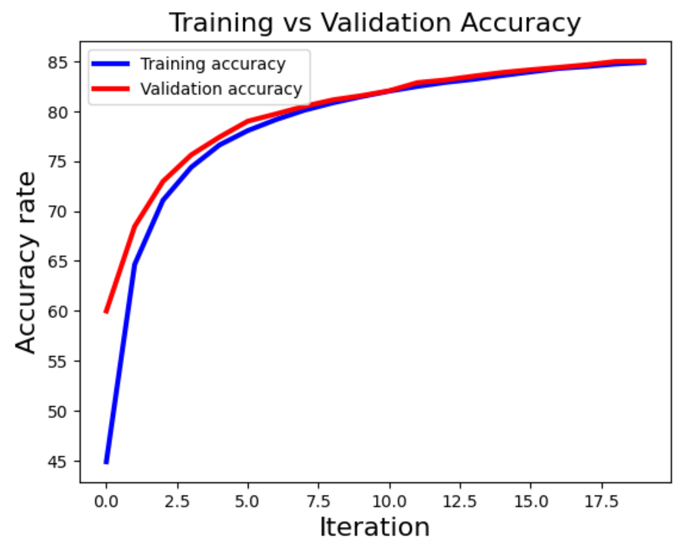
Graphs for model 1.1:

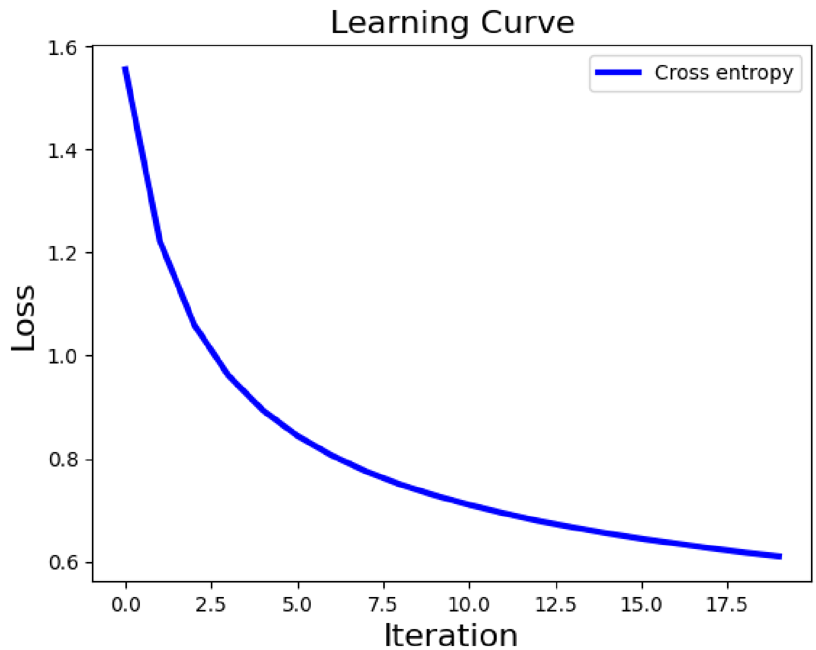


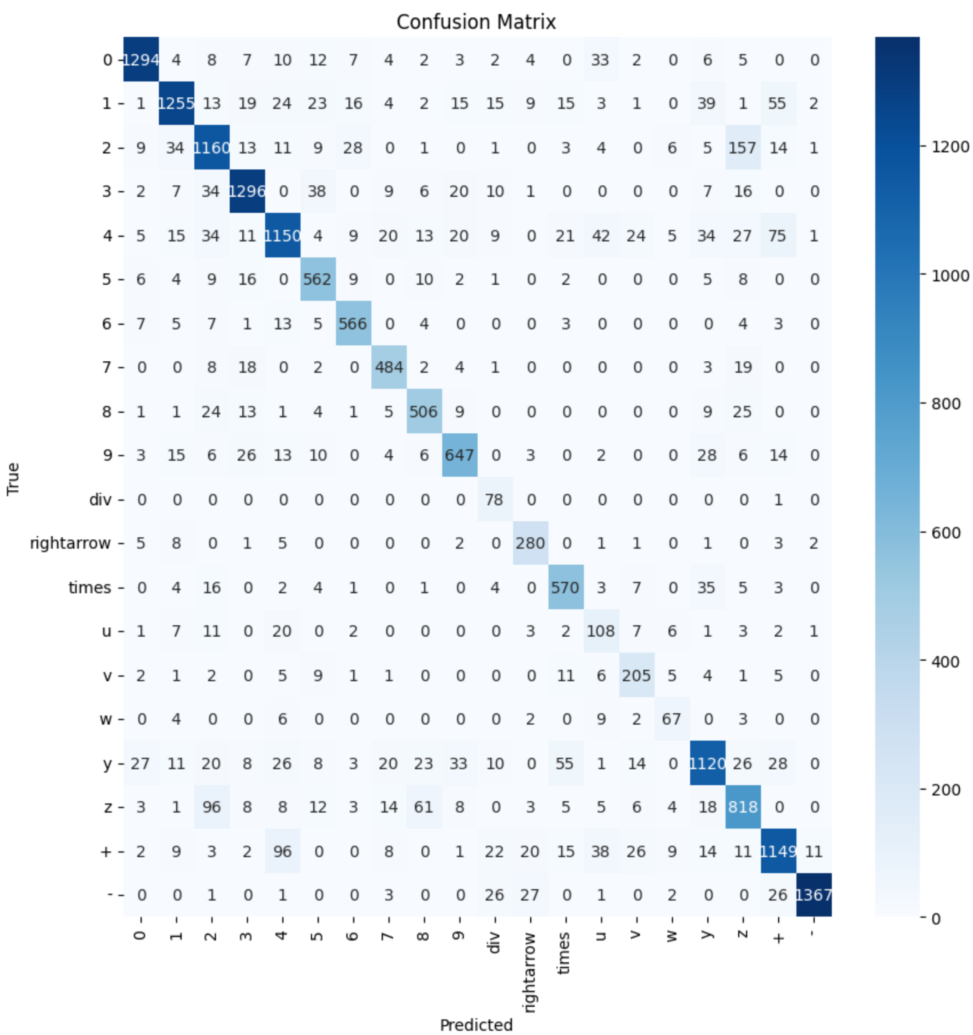


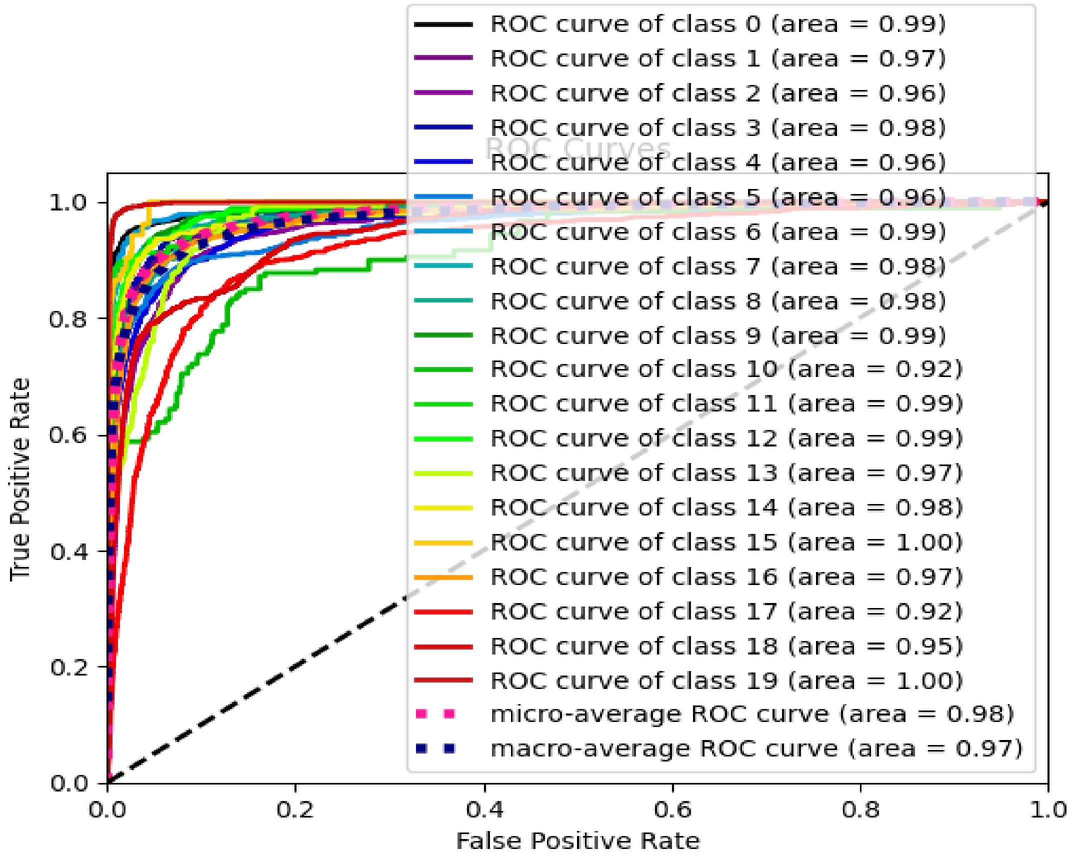
Graphs for Model 1.2 (Single layer DNN with :






Learning Curve
Cross entropy
Loss
Iteration
1.6
1.4
1.2
1.0
0.8
0.6
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5


Confusion Matrix
True
Predicted
0 1 2 3 4 5 6 7 8 9 div rightarrow times u v w y z + -
0 - 1294 4 8 7 10 12 7 4 2 3 2 4 0 33 2 0 6 5 0 0
1 - 1 1255 13 19 24 23 16 4 2 15 15 9 15 3 1 0 39 1 55 2
2 - 9 34 1160 13 11 9 28 0 1 0 1 0 3 4 0 6 5 157 14 1
3 - 2 7 34 1296 0 38 0 9 6 20 10 1 0 0 0 0 7 16 0 0
4 - 5 15 34 11 1150 4 9 20 13 20 9 0 21 42 24 5 34 27 75 1
5 - 6 4 9 16 0 562 9 0 10 2 1 0 2 0 0 0 5 8 0 0
6 - 7 5 7 1 13 5 566 0 4 0 0 0 3 0 0 0 0 4 3 0
7 - 0 0 8 18 0 2 0 484 2 4 1 0 0 0 0 0 3 19 0 0
8 - 1 1 24 13 1 4 1 5 506 9 0 0 0 0 0 0 9 25 0 0
9 - 3 15 6 26 13 10 0 4 6 647 0 3 0 2 0 0 28 6 14 0
div - 0 0 0 0 0 0 0 0 0 0 78 0 0 0 0 0 0 0 1 0
rightarrow - 5 8 0 1 5 0 0 0 0 2 0 280 0 1 1 0 1 0 3 2
times - 0 4 16 0 2 4 1 0 1 0 4 0 570 3 7 0 35 5 3 0
u - 1 7 11 0 20 0 2 0 0 0 0 3 2 108 7 6 1 3 2 1
v - 2 1 2 0 5 9 1 1 0 0 0 0 11 6 205 5 4 1 5 0
w - 0 4 0 0 6 0 0 0 0 0 0 2 0 9 2 67 0 3 0 0
y - 27 11 20 8 26 8 3 20 23 33 10 0 55 1 14 0 1120 26 28 0
z - 3 1 96 8 8 12 3 14 61 8 0 3 5 5 6 4 18 818 0 0
+ - 2 9 3 2 96 0 0 8 0 1 22 20 15 38 26 9 14 11 1149 11
- - 0 0 1 0 1 0 0 3 0 0 26 27 0 1 0 2 0 0 26 1367
1200
1000
800
600
400
200
0



Observations for model (Accuracy 84%):

1. Weight initialization: Without glorot/xavier weight initialization the model took more number of epochs for convergence compared to glorot/Xavier initialization.

2. As it is a single layer NN, it is the fastest compared to all models in training time.

3. The Model 1.2 was stuck in local minima i.e it's accuracy didn't improve above 74% after 100 epochs.

4. But the same model (Model 1.2) with glorot init converged faster at 20 epochs, with validation and testing accuracy of 84% at the end of 20 epoch.

Model 2: Dense1, ReLu, Dense2, Softmax, Cross Entropy

Layer information

Number of Layers 4

Layer 1

Layer Name -> flatten_1

Layer 2

Layer Name -> Dense_1

Weights shape -> (2025, 50)

Layer 3

Layer Name -> relu_1

Layer 4

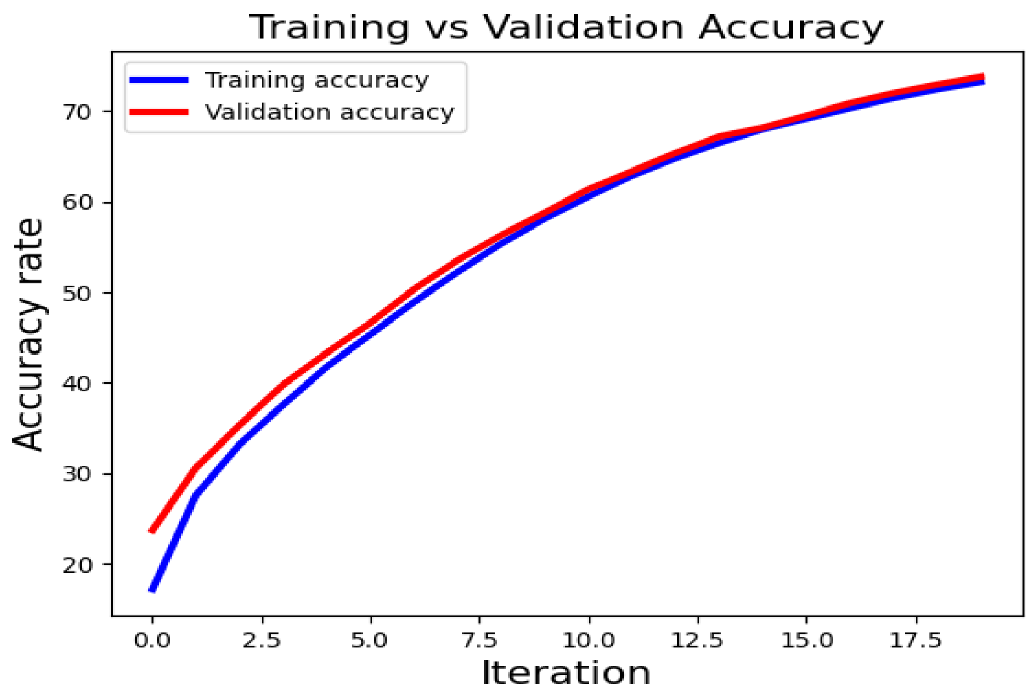
Layer Name -> Dense_2

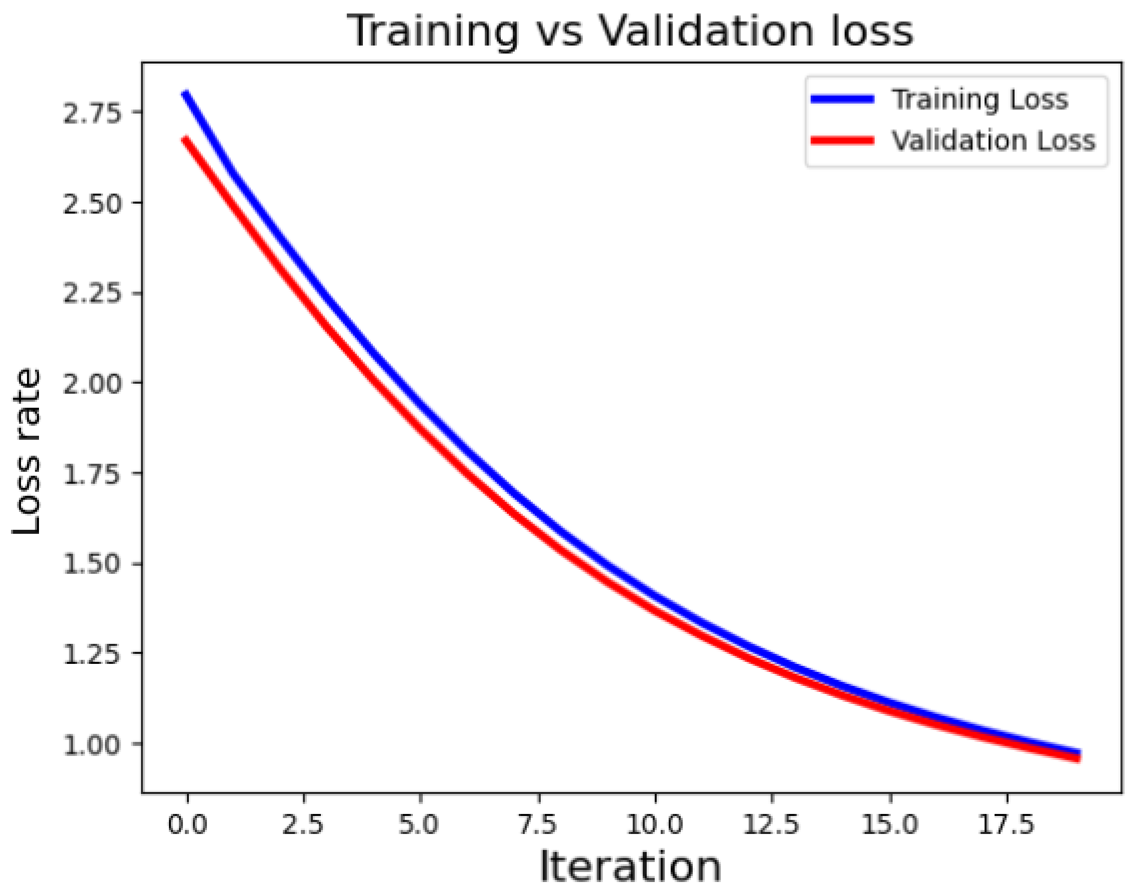
Weights shape -> (50, 20)

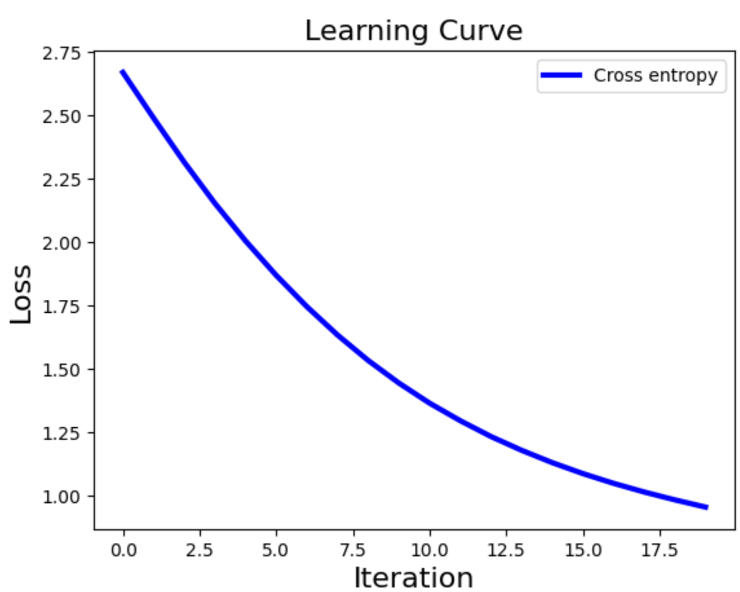
Model	Epochs	Batch Size	Learning Rate	Train Accuracy (%)	Validation Accuracy (%)	Train Loss	Validation Loss	Test Accuracy (%)	Test Loss (%)
Model 2	5	16	0.0001	4.86	5.04	28.75	28.43	-	-
Model 2.1	5	16	0.001	7.07	8.00	24.58	24.02	-	-
Model 2.1	20	16	0.001	15.70	15.99	18.19	18.38	-	-
Model 2.2	20	16	0.001	18.79	18.79	16.23	16.21	-	-

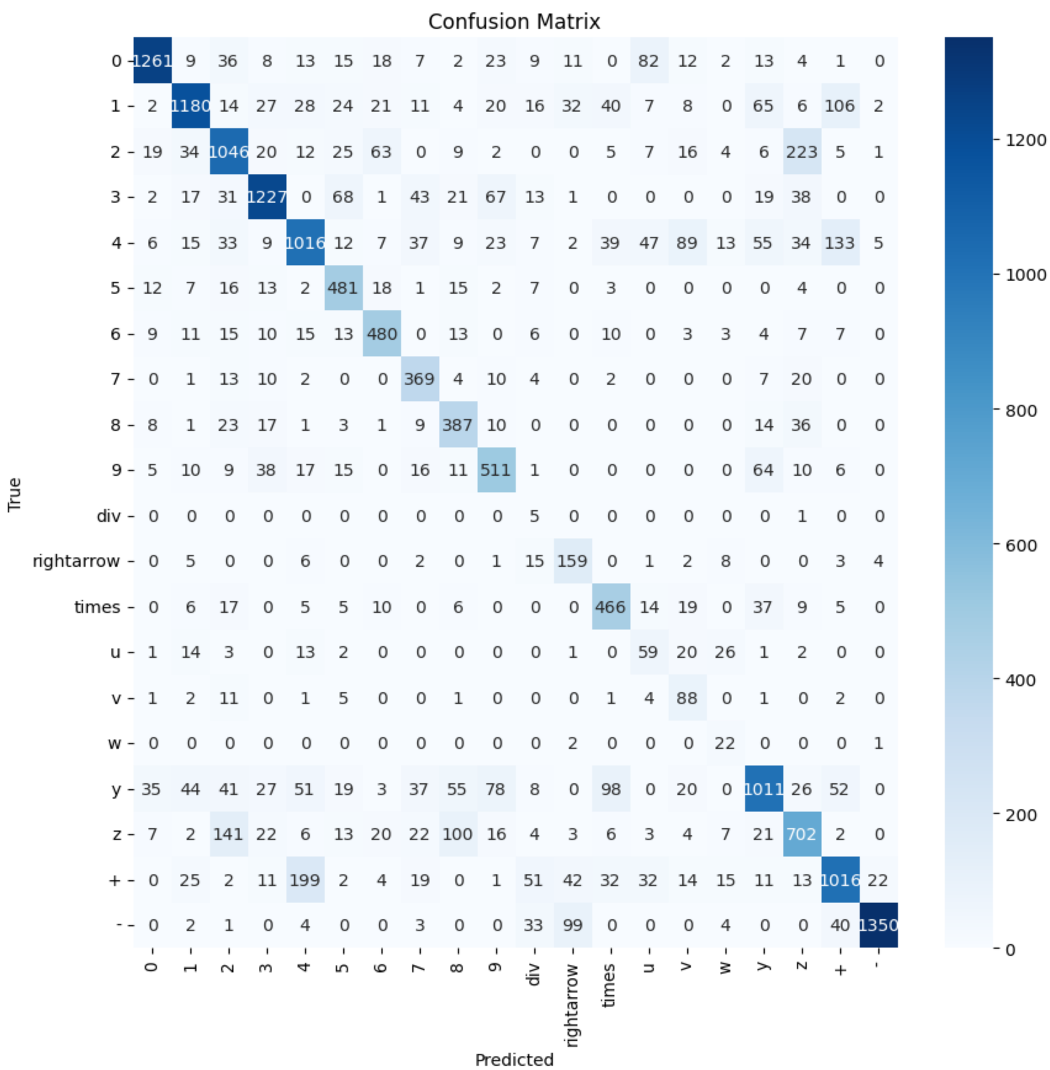
Class	Precision	Recall	F1-Score	Support
0	0.83	0.92	0.87	1368
1	0.73	0.85	0.79	1385
2	0.70	0.72	0.71	1452
3	0.79	0.85	0.82	1439
4	0.64	0.73	0.68	1391
5	0.83	0.69	0.75	702
6	0.79	0.74	0.77	646
7	0.83	0.64	0.72	576
8	0.76	0.61	0.67	637
9	0.72	0.67	0.69	764
div	0.83	0.03	0.05	179
rightarrow	0.77	0.45	0.57	352
times	0.78	0.66	0.72	702
u	0.42	0.23	0.30	256
v	0.75	0.30	0.43	295
w	0.88	0.21	0.34	104

y	0.63	0.76	0.69	1329
z	0.64	0.62	0.63	1135
+	0.67	0.74	0.70	1378
-	0.88	0.97	0.92	1385
Accuracy	0.73			
Macro Avg	0.74	0.62	0.64	17475
Weighted Avg	0.74	0.73	0.72	17475




Training vs Validation loss
Training Loss
Validation Loss
Loss rate
Iteration
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Learning Curve
Cross entropy
Loss
Iteration
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



Observations

1. Model is stuck at local minima (Model 2.1) but with glorot init (Model 2.2) it improved.

2. Model 1.2 has better metrics than Model 2.2 which has the same hyperparameters but Model 2.2 has 2 Dense layers whereas Model 1.2 has only one dense layer

Model 3: Conv1, Relu1, Maxpool1, Flatten1, Dense1, Softmax, Cross Entropy

Layer information

Layer 1

Layer Name -> conv_1

Number of filters -> 8

Filters shape -> (8, 1, 3, 3)

Stride -> 1

Layer 2

Layer Name -> relu_1

Layer 3

Layer Name -> max_pool_1

Pool size -> 2

Stride -> 2

Layer 4

Layer Name -> flatten_1

Layer 5

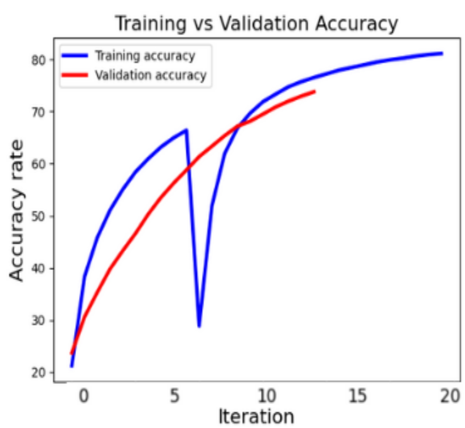
Layer Name -> Dense_1

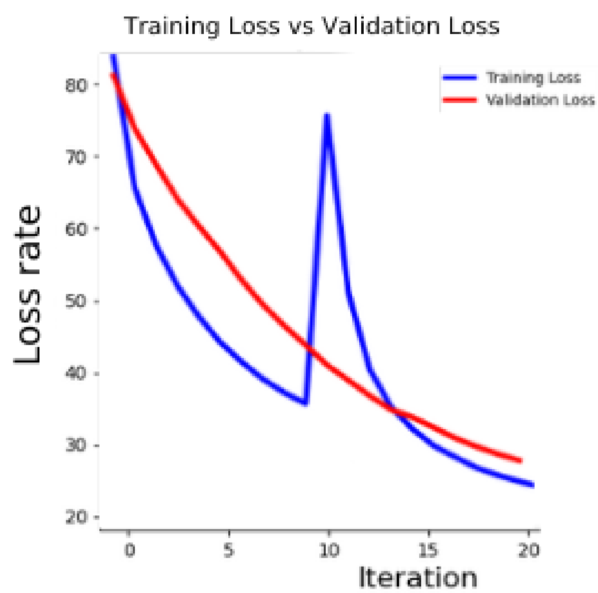
Weights shape -> (3528, 20)

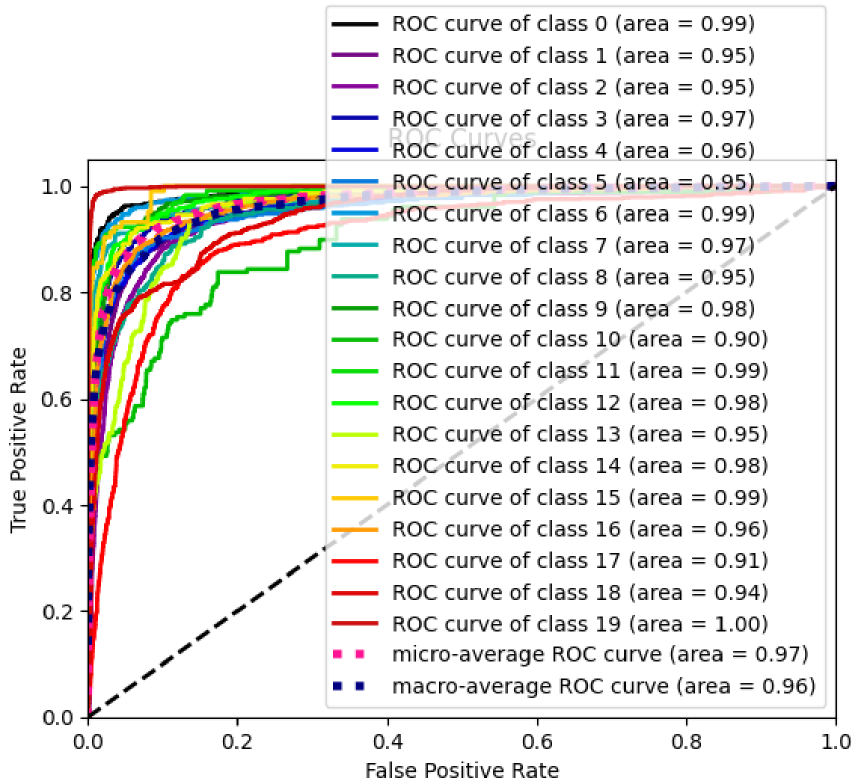
Model	Epochs	Batch Size	Learning Rate	Train Accuracy (%)	Validation Accuracy (%)	Train Loss	Validation Loss	Test Accuracy (%)	Test Loss (%)
Model 3	11	64	0.01	66.42	67.18	1.32	1.28	68.00	-
Model 3	20	64	0.01	81.17	81.40	0.73	0.73	81.00	-

Class	Precision	Recall	F1-Score	Support
0	0.89	0.93	0.91	1368
1	0.78	0.86	0.82	1385
2	0.76	0.76	0.76	1452
3	0.88	0.87	0.87	1439

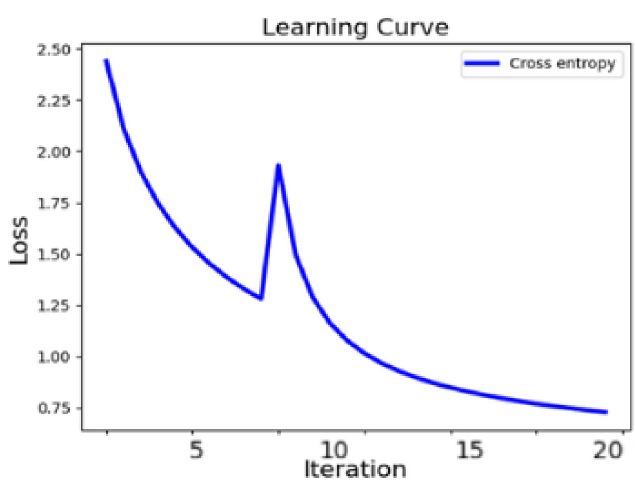
4	0.73	0.82	0.77	1391
5	0.87	0.78	0.82	702
6	0.86	0.87	0.87	646
7	0.83	0.78	0.81	576
8	0.80	0.72	0.76	637
9	0.78	0.81	0.79	764
div	0.99	0.39	0.55	179
rightarrow	0.86	0.76	0.81	352
times	0.85	0.77	0.81	702
Accuracy:	0.81			



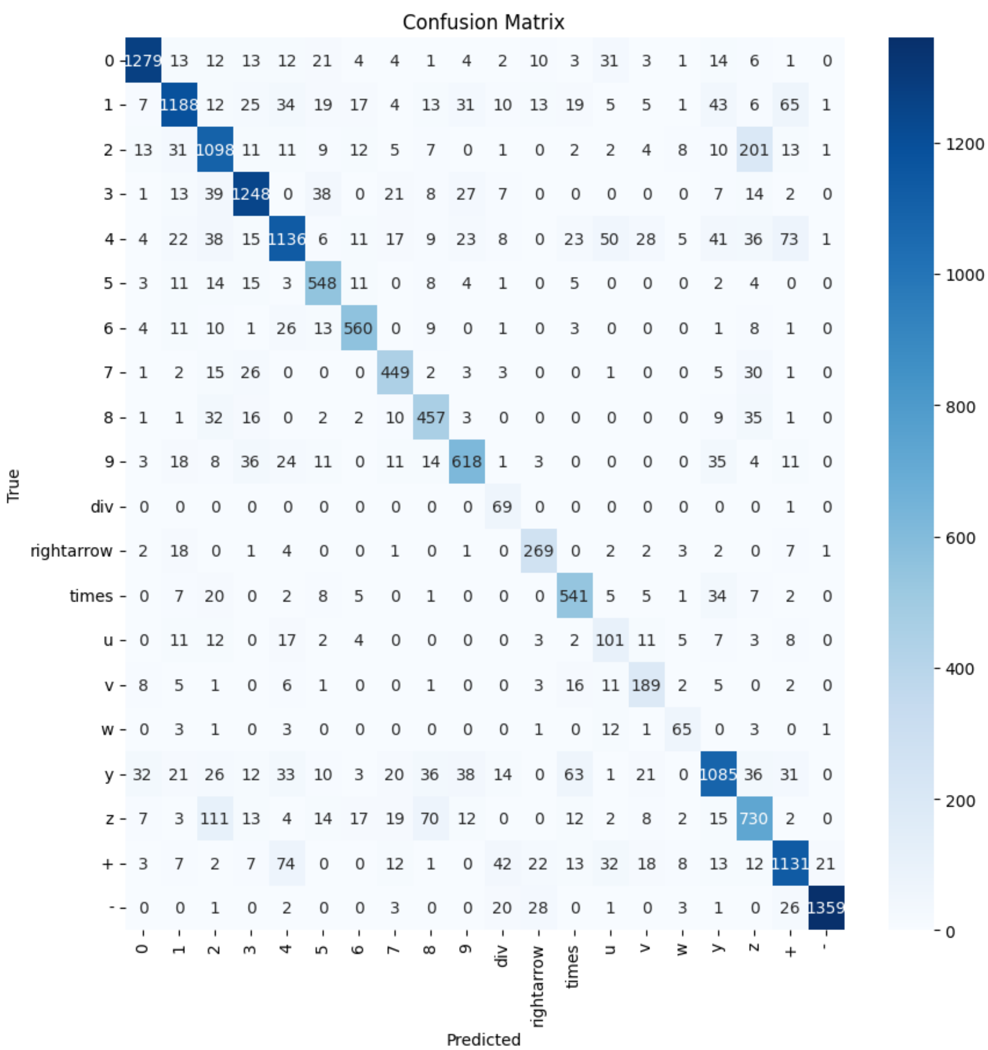




ROC Curves
ROC curve of class 0 (area = 0.99)
ROC curve of class 1 (area = 0.95)
ROC curve of class 2 (area = 0.95)
ROC curve of class 3 (area = 0.97)
ROC curve of class 4 (area = 0.96)
ROC curve of class 5 (area = 0.95)
ROC curve of class 6 (area = 0.99)
ROC curve of class 7 (area = 0.97)
ROC curve of class 8 (area = 0.95)
ROC curve of class 9 (area = 0.98)
ROC curve of class 10 (area = 0.90)
ROC curve of class 11 (area = 0.99)
ROC curve of class 12 (area = 0.98)
ROC curve of class 13 (area = 0.95)
ROC curve of class 14 (area = 0.98)
ROC curve of class 15 (area = 0.99)
ROC curve of class 16 (area = 0.96)
ROC curve of class 17 (area = 0.91)
ROC curve of class 18 (area = 0.94)
ROC curve of class 19 (area = 1.00)
micro-average ROC curve (area = 0.97)
macro-average ROC curve (area = 0.96)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0



Learning Curve
Cross entropy
Loss
Iteration
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
5
10
15
20



Observations for Model 3:

1.Took more time to train as convolution operation was involved.

2.Achieved similar metric to Model 1.2 but took longer time for each epoch.

3.Observed a spike in loss, during the 11 epoch.

Model 4: Conv2, Relu1, MaxPool1, Conv1, Relu2, MaxPool2, Flatten1, Dense1, Softmax, Cross Entropy

Layer information

Number of Layers 8

Layer 1

Layer Name -> conv_1

Number of filters -> 8

Filters shape -> (8, 1, 3, 3)

Stride -> 1

Layer 2

Layer Name -> relu_1

Layer 3

Layer Name -> max_pool_1

Pool size -> 2

Stride -> 2

Layer 4

Layer Name -> conv_2

Number of filters -> 12

Filters shape -> (12, 8, 5, 5)

Stride -> 1

Layer 5

Layer Name -> relu_2

Layer 6

Layer Name -> max_pool_2

Pool size -> 2

Stride -> 2

Layer 7

Layer Name -> flatten_1

Layer 8

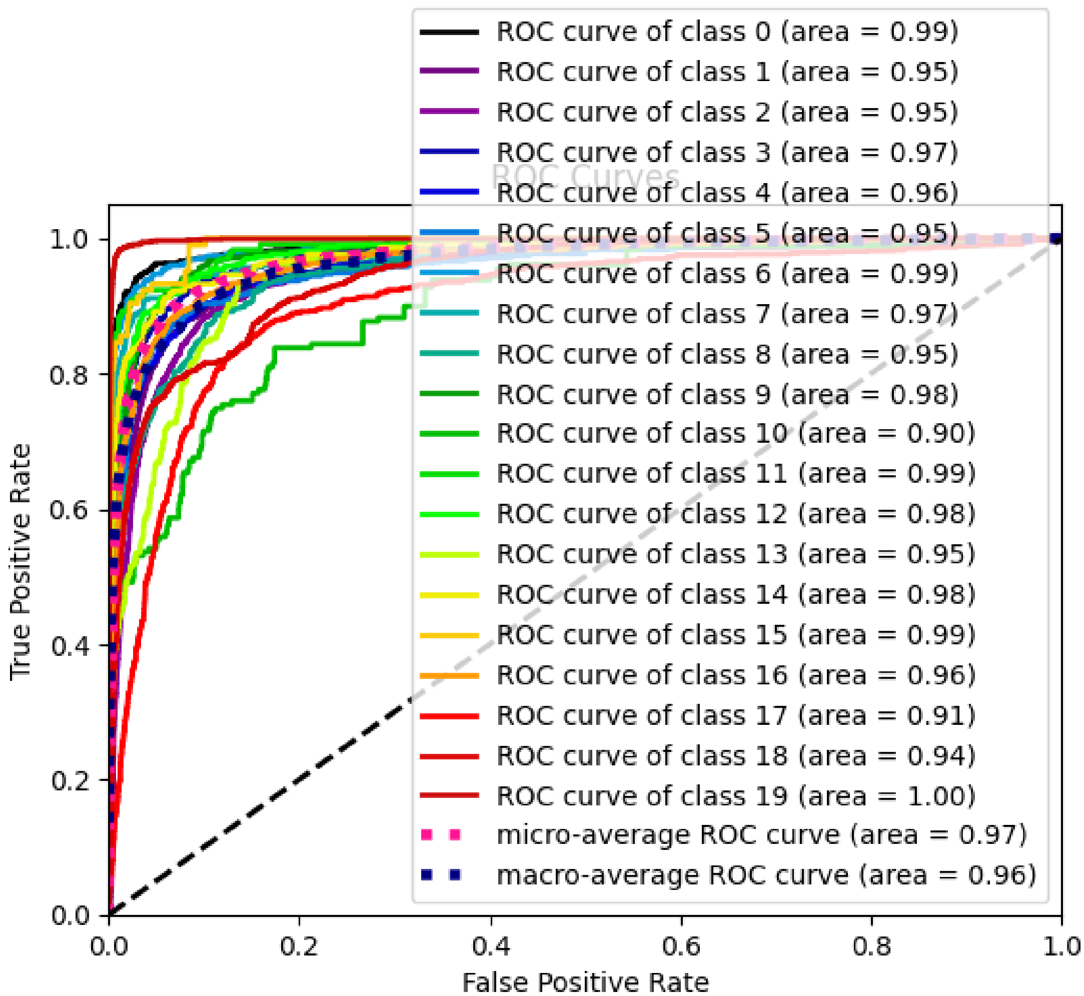
Layer Name -> Dense_1

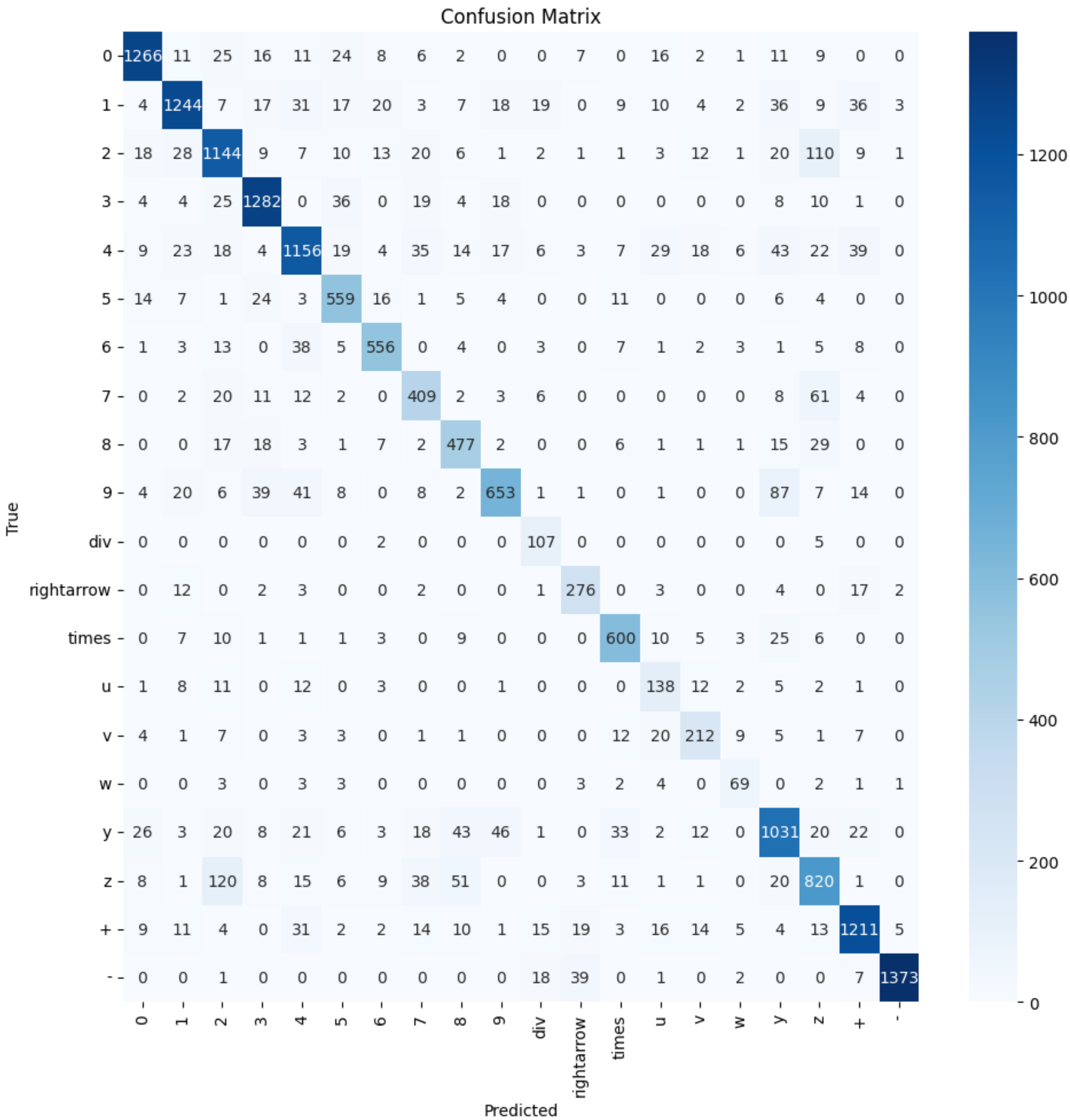
Weights shape -> (768, 20)

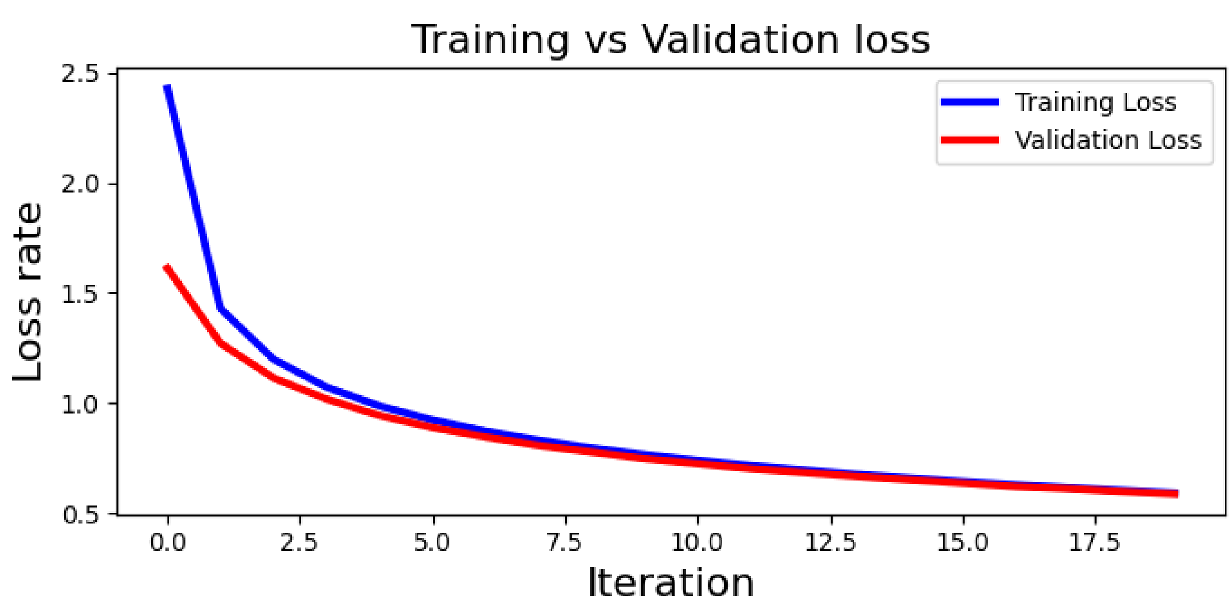
Model	Epochs	Batch Size	Learning Rate	Train Accuracy (%)	Validation Accuracy (%)	Train Loss	Validation Loss	Test Accuracy (%)	Test Loss (%)
Model 4	20	128	0.1	83.92	83.99	0.5917	0.5874	83.45	0.5988

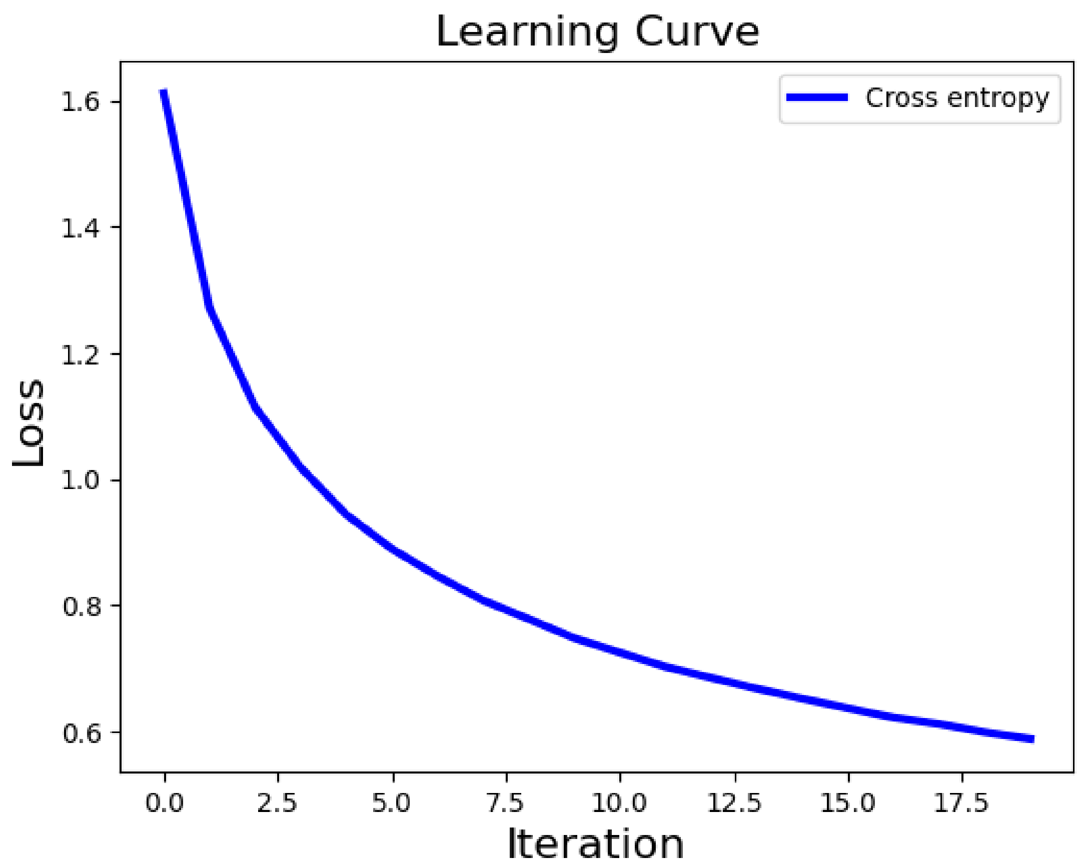
Class	Precision	Recall	F1-Score	Support
0	0.89	0.93	0.91	1368
1	0.83	0.90	0.86	1385
2	0.81	0.79	0.80	1452
3	0.91	0.89	0.90	1439
4	0.79	0.83	0.81	1391
5	0.85	0.80	0.82	702
6	0.86	0.86	0.86	646
7	0.76	0.71	0.73	576
8	0.82	0.75	0.78	637
9	0.73	0.85	0.79	764
div	0.94	0.60	0.73	179
rightarrow	0.86	0.78	0.82	352
times	0.88	0.85	0.87	702
u	0.70	0.54	0.61	256
v	0.74	0.72	0.73	295

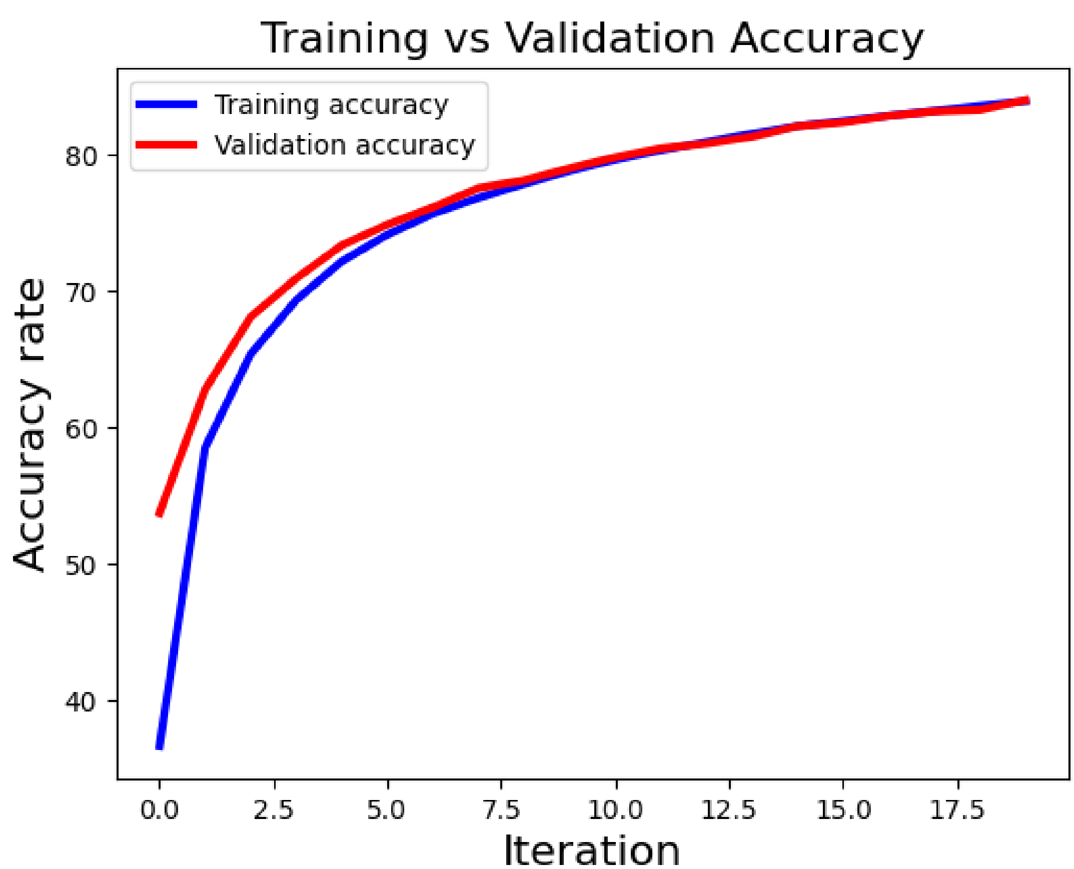
w	0.76	0.66	0.71	104
y	0.78	0.78	0.78	1329
z	0.74	0.72	0.73	1135
+	0.87	0.88	0.88	1378
-	0.95	0.99	0.97	1385
Accuracy:			0.83	17475
Macro Avg:	0.82	0.79	0.8	17475
Weighted Avg:	0.83	0.83	0.83	17475



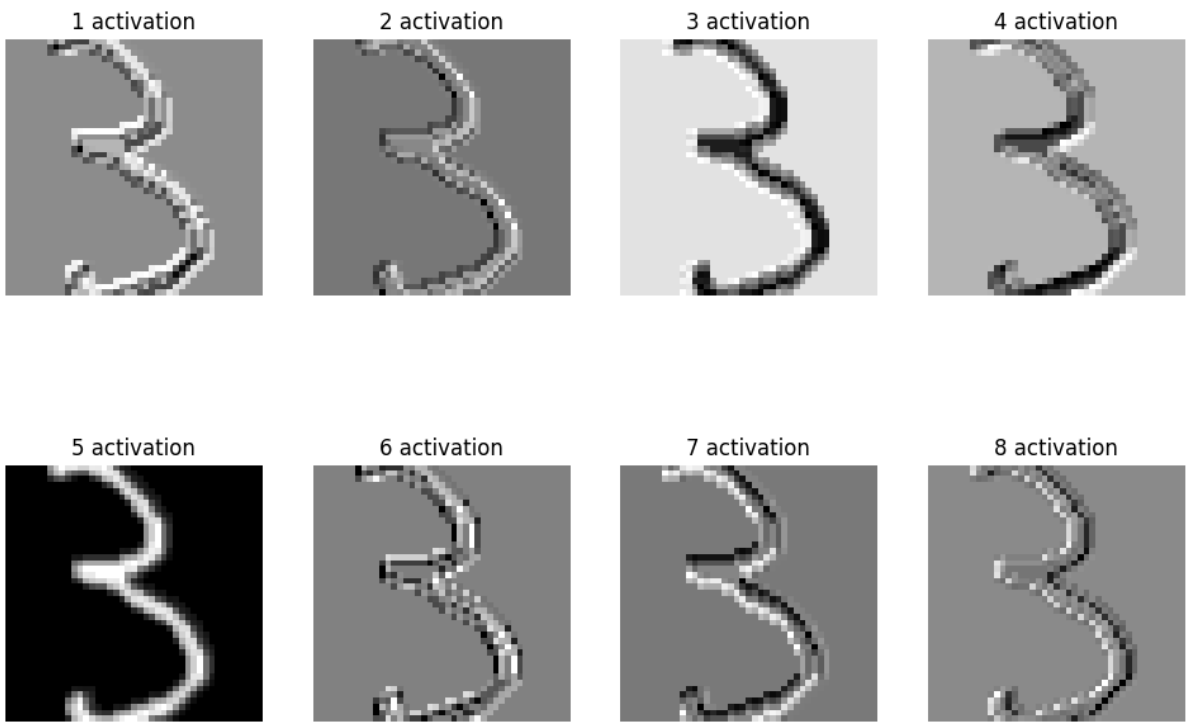

Confusion Matrix
True
Predicted

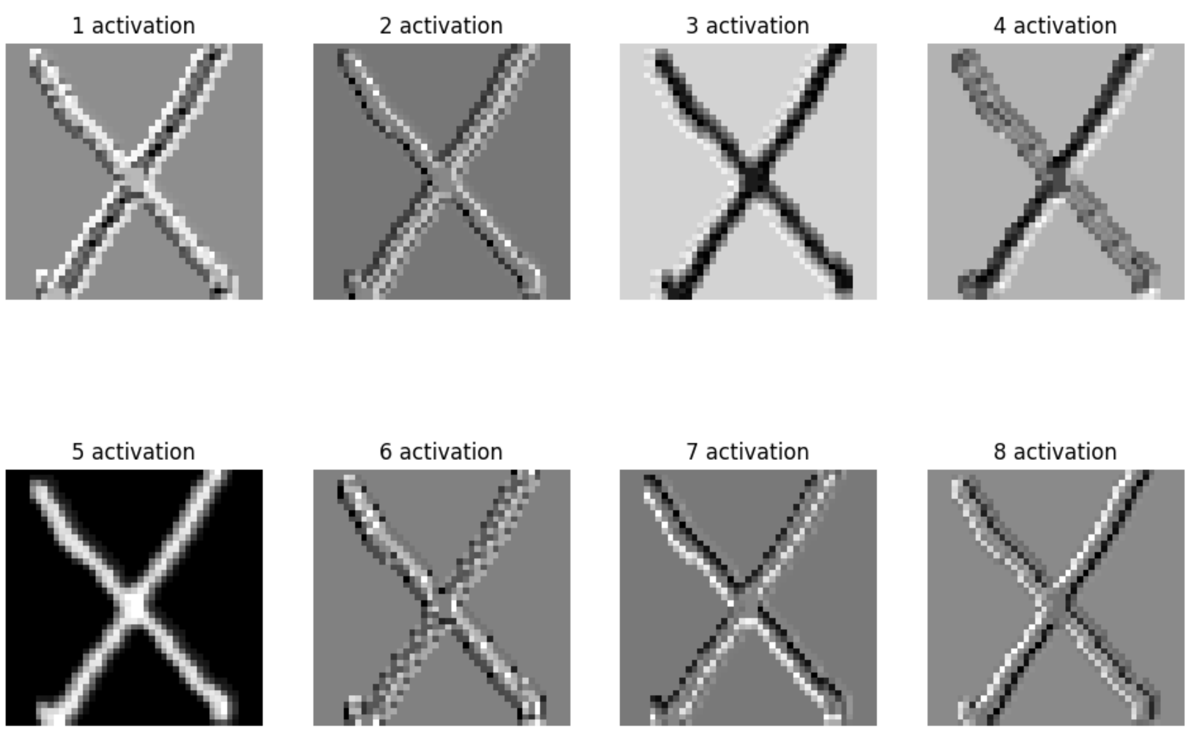

Training vs Validation loss
Training Loss
Validation Loss
Loss rate
Iteration

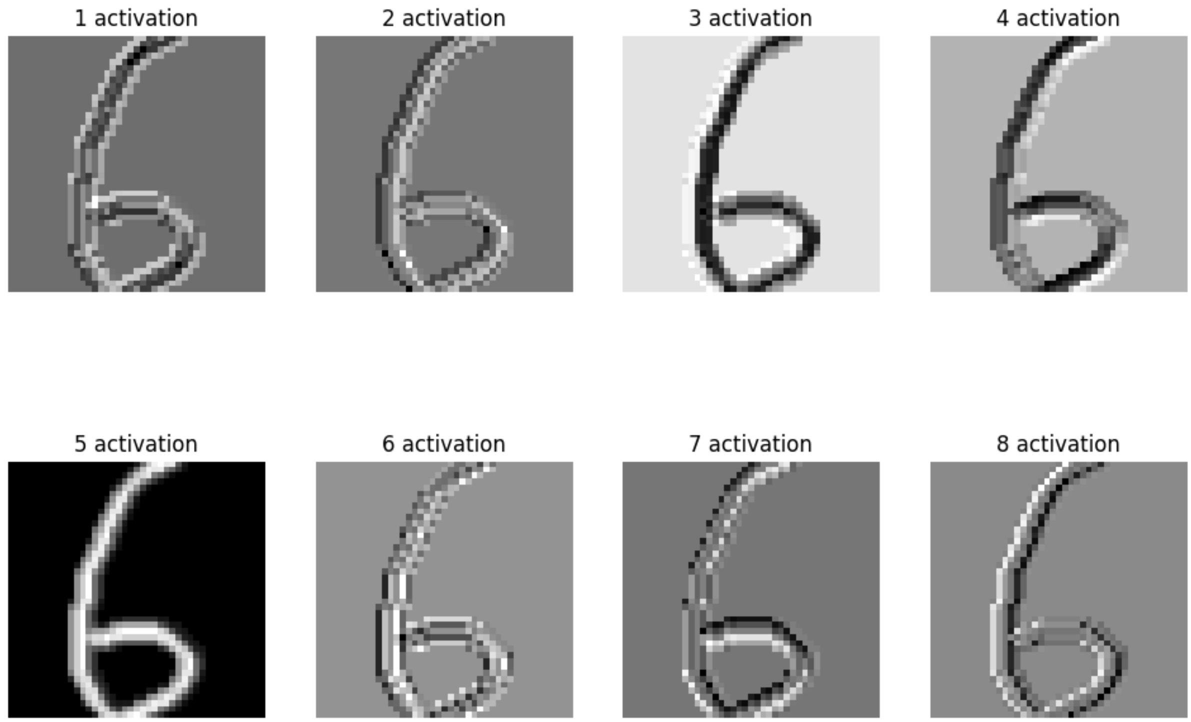

Learning Curve
Cross entropy
Loss
Iteration
1.6
1.4
1.2
1.0
0.8
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training vs Validation Accuracy
Training accuracy
Validation accuracy
Accuracy rate
Iteration
80
70
60
50
40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

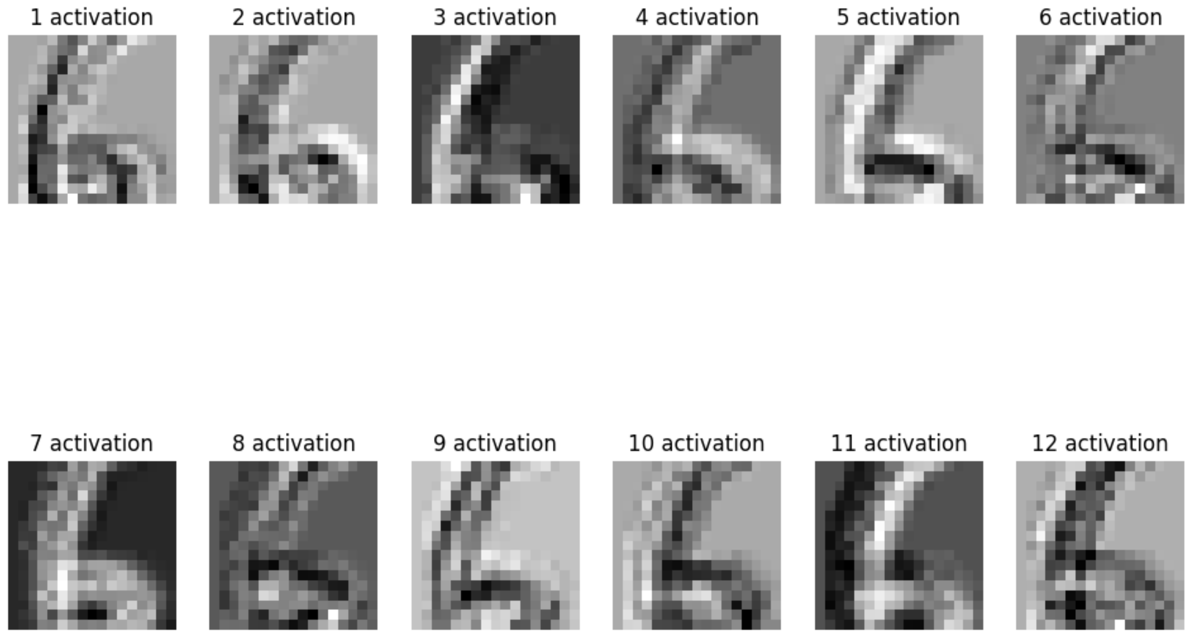
Convolutional Layer 1 Output Activations:

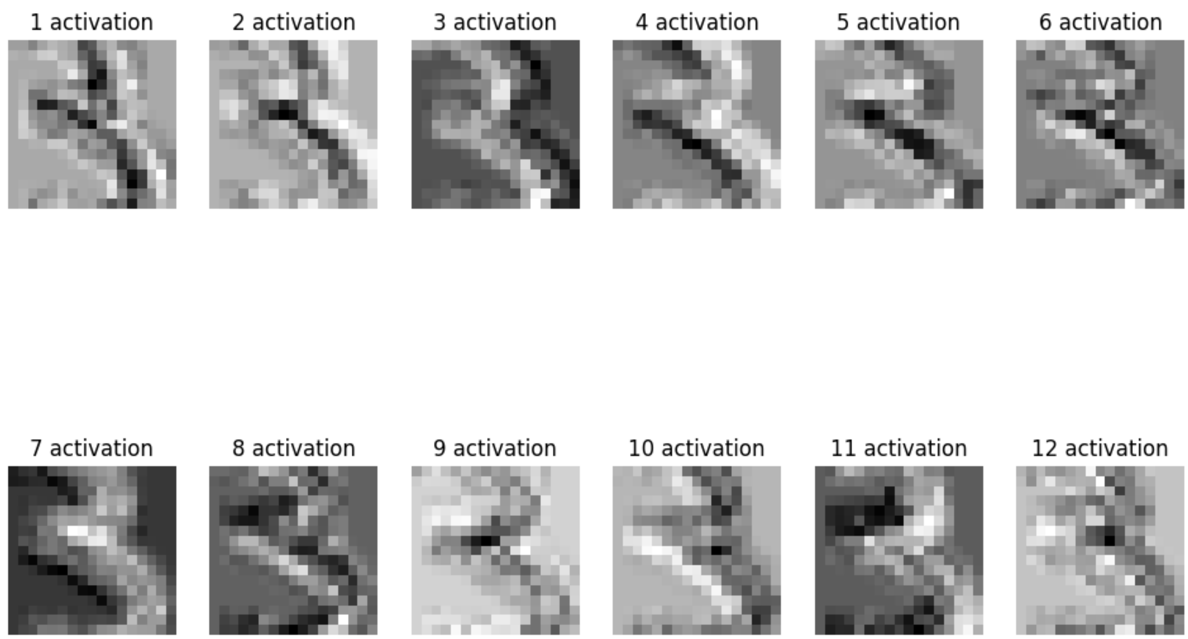


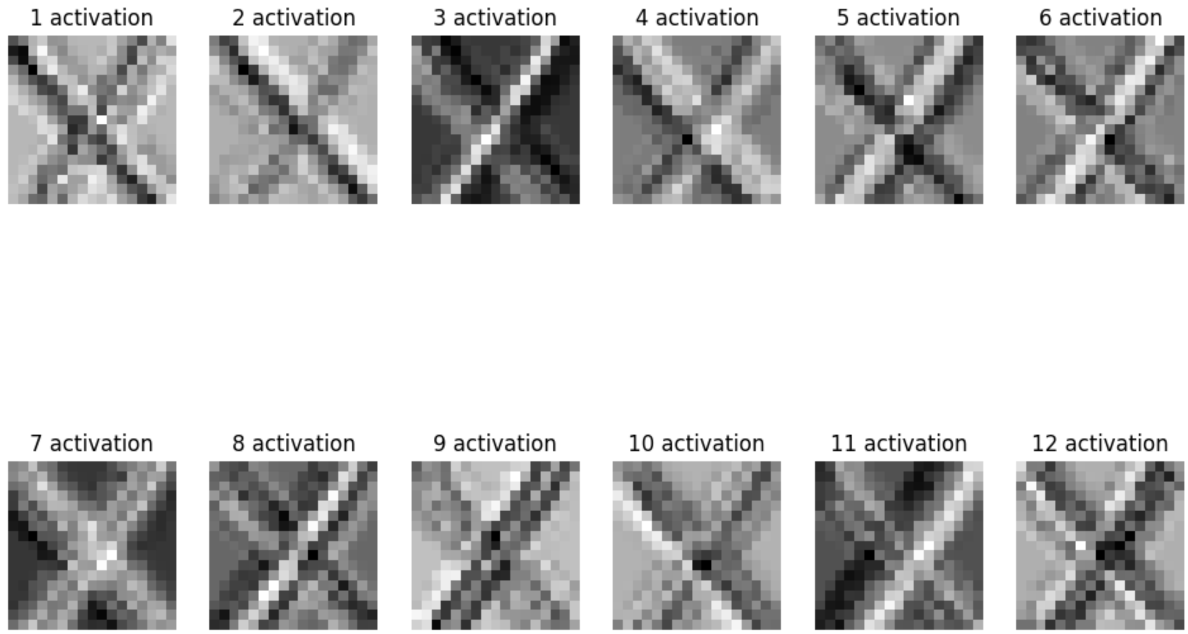



1 activation
2 activation
3 activation
4 activation
5 activation
6 activation
7 activation
8 activation

Convolutional Layer 2 Output Activations:



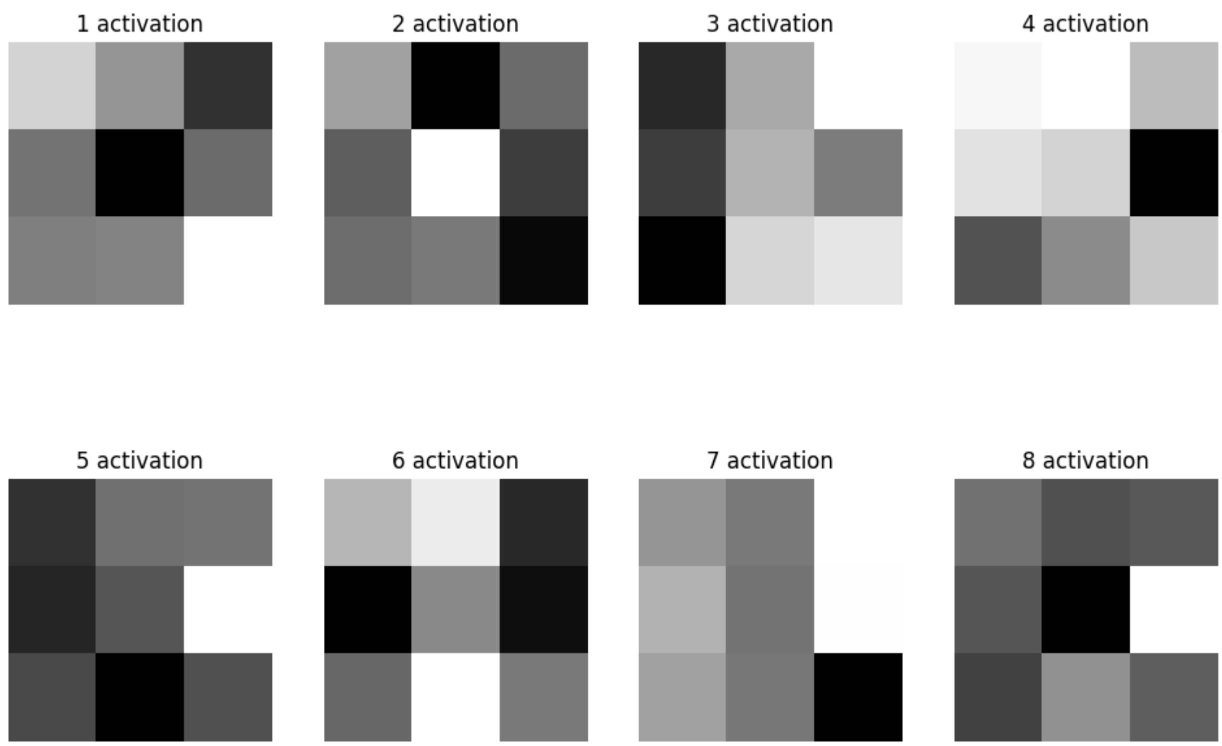




Filter Activations

Convolutional Layer 1 Filter Output

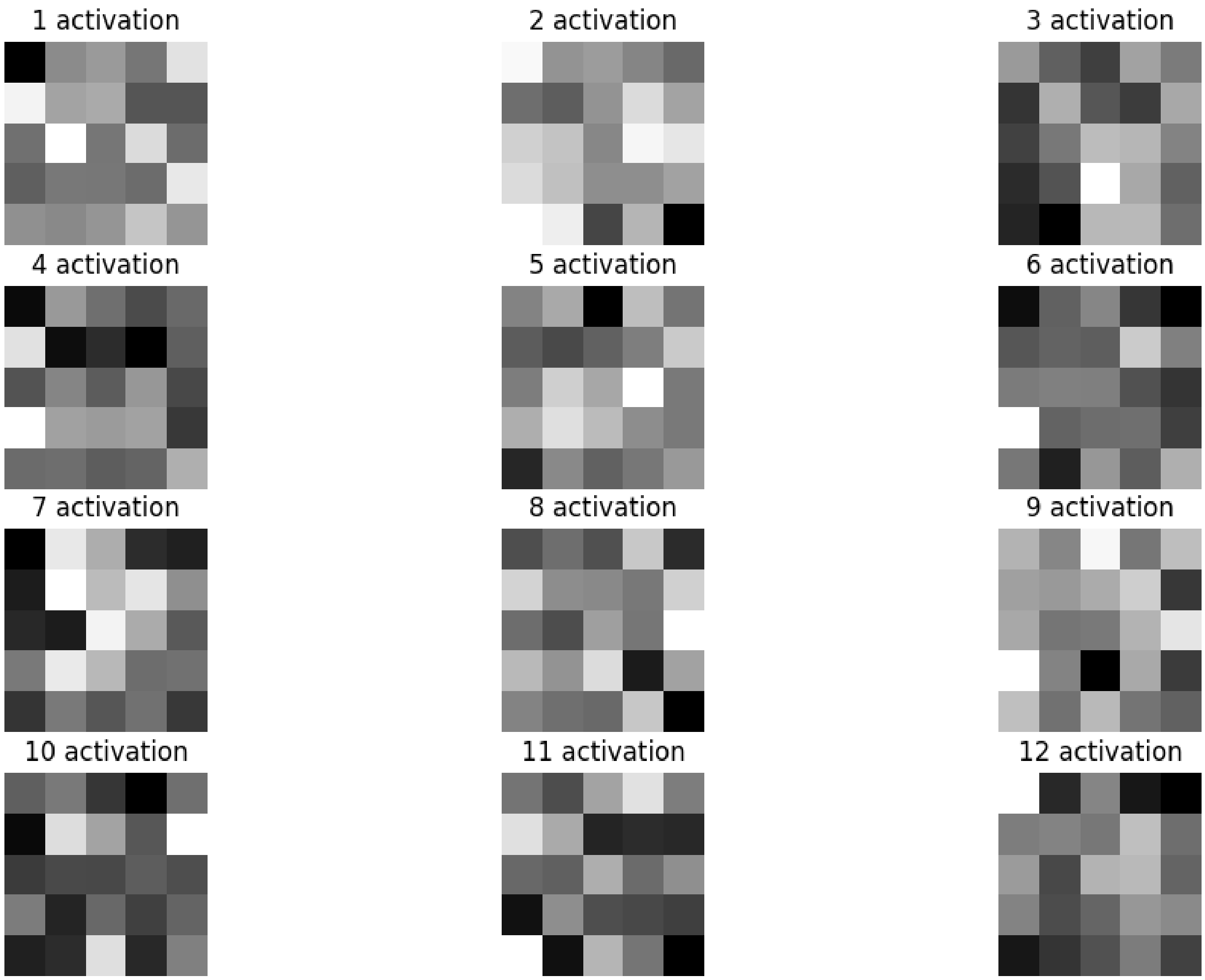
8 Filters of Size 1:3:3 (Channels, Width, Height)

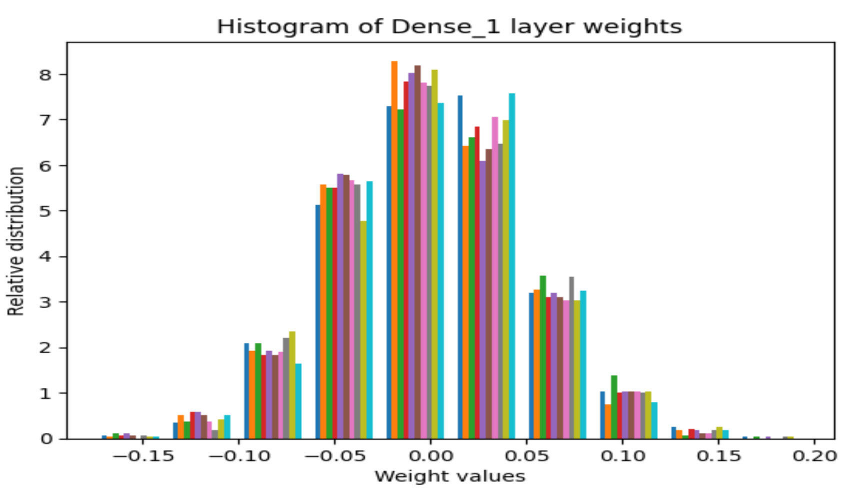


Convolutional Layer 2 Filter Output

12 Filters of size 8:5:5 (Channels, Width, Height)

Only visualizing 12:1:5:5





Conclusion and future work:

1. The best performing model on real world images didn't work as well as it did on test set. So there might be some issue while preprocessing real-images before feeding into the model. So, the model is overfitting as it's not working on real world images.

2. More work need to be done on dataset preparation as the model on real world images weren't good, so we need to augment the images for the classes which are of less numbers.

3. Forward propagation is vectorized in CNNs but during backward propagation we can vectorize the code even for calculating the gradients for each channel.

4. Currently we only solve linear equations with single variables, we can work on extending the equation solver for quadratic and trigonometric equations.

5. CNN layers are pretty slow, so we can look into improving the training speed of these layers by using Python and vectorizing the code over the image channels.

References:

1. https://www.parasdahal.com/softmax-crossentropy
2. https://jmlb.github.io/ml/2017/12/26/Calculate_Gradient_Softmax/
3. https://www.youtube.com/watch?v=5-rVLSc2XdE&ab_channel=SmartAlphaAI
4. https://pyimagesearch.com/2021/05/06/understanding-weight-initialization-for-neural-networks/
5. https://www.youtube.com/watch?v=3TdBtI9dh2I
6. https://deeplearning.cs.cmu.edu/F21/document/recitation/Recitation5/CNN_Backprop_Recitation_5_F21.pdf
7. https://www.educative.io/answers/how-to-backpropagate-through-max-pooling-layers
8. https://pyimagesearch.com/2015/04/20/sorting-contours-using-python-and-opencv/