

THIS ASSIGNMENT IS ABOUT

Developing a robust handwritten digit recognition system using PyTorch and the MNIST dataset. It requires creating a custom neural network, optimizing training loops, analyzing performance metrics, and testing various hyperparameters to enhance model accuracy.

Mohamed Hussien El zohiry 7818
Alaa Mohamed Elkhouly 7721

Assignment 2

Machine Learning

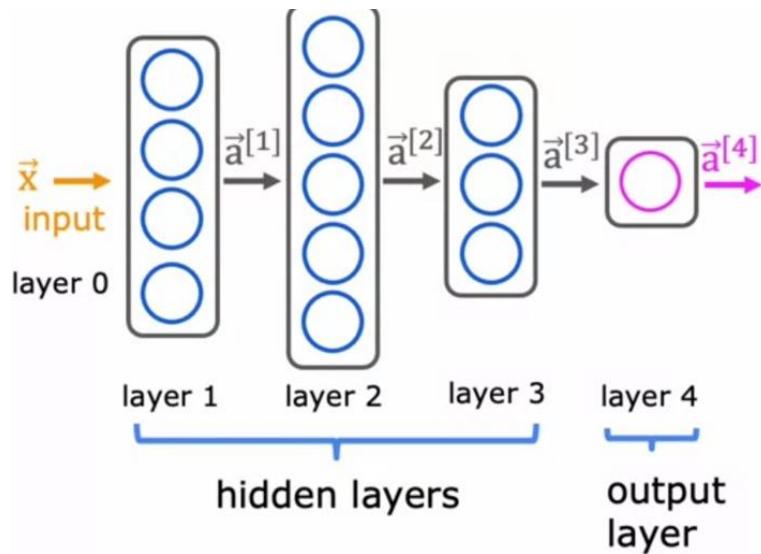
What is PyTorch?

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab. It provides tools for building and training neural networks. Key features include:

1. **Dynamic Computation Graphs:** Allows flexibility in building and modifying models during runtime.
2. **Tensor Operations:** Offers high-performance multidimensional arrays (tensors) similar to NumPy, with GPU acceleration.
3. **Deep Learning Models:** Simplifies the creation of neural networks using built-in modules like `torch.nn`.
4. **Automatic Differentiation:** Includes `autograd` for automatic calculation of gradients, essential for optimization tasks.
5. **Integration with Python:** Supports seamless integration with Python libraries and workflows.

PyTorch is widely used in research and production for tasks like computer vision, natural language processing, and reinforcement learning.

What is a neural network (NN)?



A **neural network** is a computational model inspired by the structure and function of the human brain. It is used in machine learning to solve complex problems by learning patterns from data. Neural networks consist of interconnected layers of nodes, called **neurons**, which process and transform inputs to produce outputs. Each neuron receives an activation signal from the preceding layer and computes some value that is propagated forward.

Key Components:

1. **Input Layer:** Receives the raw input data features (e.g., pixel values from an image).
2. **Hidden Layers:** Perform computations by applying weights, biases, and activation functions to transform the input into meaningful patterns.
3. **Output Layer:** Produces the final predictions or classifications (e.g., the digit class).

4. **Weights and Biases:** Parameters adjusted during training to improve model accuracy.
5. **Activation Functions:** Non-linear functions like ReLU, Sigmoid, or Tanh that determine the output of each neuron. Introducing non-linearity into the model, enables it to learn complex patterns.

Working Process:

1. **Forward Propagation:** Data flows through the network, and predictions are generated.
2. **Loss Calculation:** The difference between predictions and actual values is computed using a loss function.
3. **Backward Propagation:** Gradients of the loss are calculated and used to update weights through optimization techniques like gradient descent.

Neural networks excel in tasks like image recognition, natural language processing, and predictive analytics.

***Loss Function:** The loss function measures the error between the predicted outputs and the true labels. In this task, we'll use Cross Entropy Loss, which is commonly used for classification problems.

***The optimizer** (e.g., Stochastic Gradient Descent) updates the network's weights to minimize the loss function during training.

Equations for Neural Network (NN)

1. Forward Propagation in a Fully Connected Layer:

$$z = W \cdot x + b$$

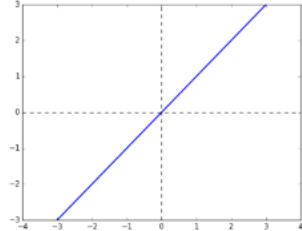
- z: Weighted sum (input to activation function).
- W: Weight matrix.
- x: Input vector.
- b: Bias vector.

2. Activation Function:

$$a = f(z) = \emptyset(W \cdot x + b)$$

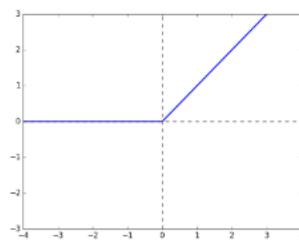
- a: Output after activation.
- f: Activation function (e.g., ReLU, Sigmoid).
- \emptyset : is applied component-wise

Examples of activation functions:



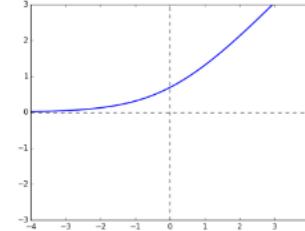
Identity

$$y = z$$



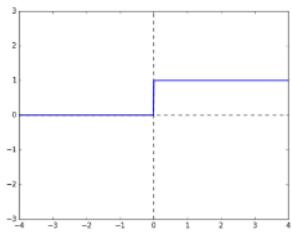
Rectified Linear Unit (ReLU)

$$y = \max(0, z)$$



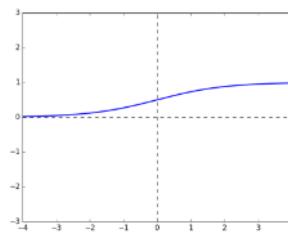
Soft ReLU

$$y = \log(1 + e^z)$$



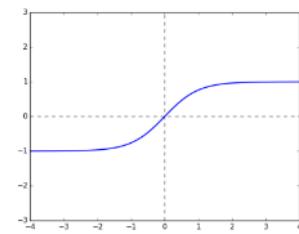
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

$$y = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

3. Output Layer:

$$\hat{y} = g(a)$$

- \hat{y} : Model prediction.
- g : Output activation (e.g., Softmax for classification).

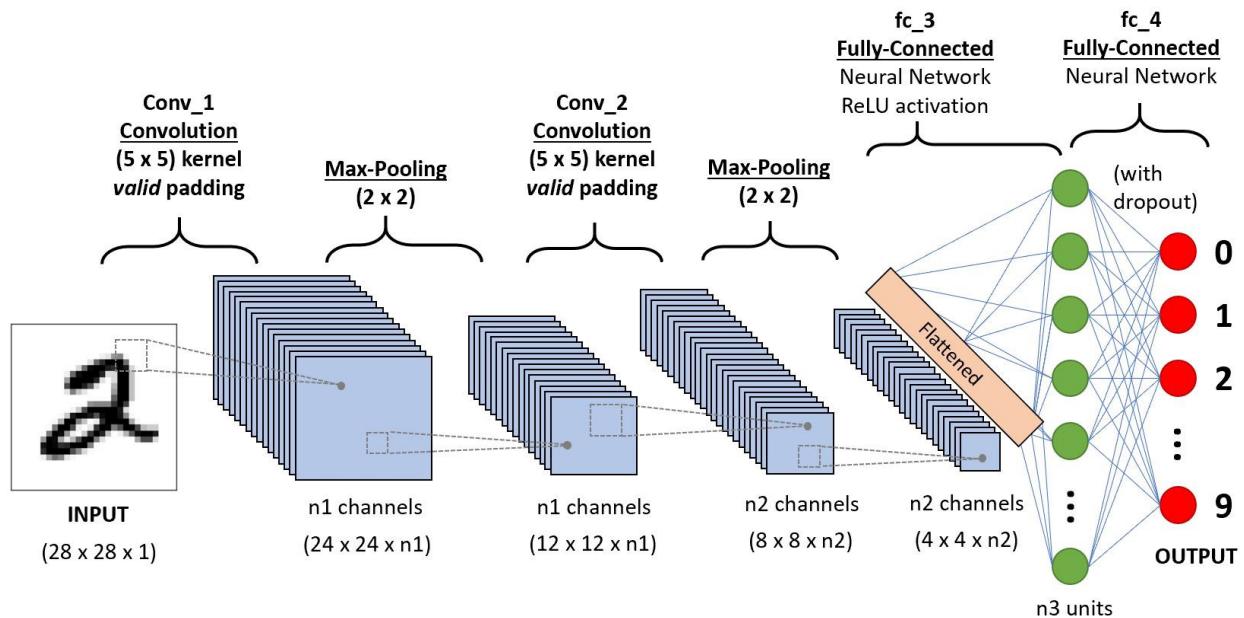
4. Loss Function: Example: Cross-Entropy Loss (for classification):

$$L = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k y_{ij} \log(\hat{y}_{ij})$$

- y : True labels.
- \hat{y} : Predicted probabilities.
- n : Number of samples.
- k : Number of classes.

NN Equations focus on matrix multiplication and activation for fully connected layers.

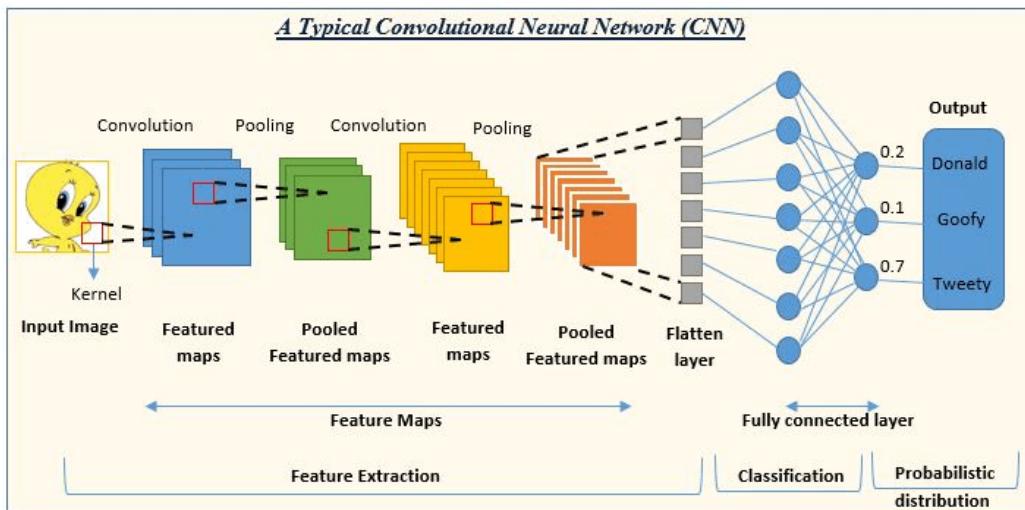
What is a Convolutional neural network (CNN)?



A **Convolutional Neural Network (CNN)** is a specialized type of neural network designed for processing structured grid data, such as images. It is widely used in tasks like image classification, object detection, and computer vision.

Key Features:

1. **Convolutional Layers:**
 - Perform convolution operations to detect patterns like edges, textures, or shapes in data.
 - Use filters (or kernels) that slide over the input data to extract features.
2. **Pooling Layers:**
 - Reduce the spatial dimensions of the data while retaining important information.
 - Common types include max pooling (takes the maximum value) and average pooling (takes the average value).
3. **Fully Connected Layers:**
 - Connect all neurons to the output, combining extracted features for final predictions.
4. **Activation Functions:**
 - Introduce non-linearity to model complex patterns (e.g., ReLU, Sigmoid).
5. **Dropout and Normalization:**
 - Dropout prevents overfitting by randomly deactivating neurons during training.
 - Normalization stabilizes and speeds up training by standardizing data.



Advantages:

- **Local Connectivity:** Focuses on small, localized regions of data to capture patterns.
- **Weight Sharing:** Reduces the number of parameters, making the model computationally efficient.
- **Hierarchical Feature Learning:** Builds simple features in earlier layers and combines them into complex features in later layers.

Common Applications:

- **Image and Video Recognition:** Classifying images or identifying objects in videos.
- **Medical Imaging:** Analyzing X-rays or MRIs for diagnostics.

Autonomous Vehicles: Detecting objects like pedestrians or traffic signs.

Equations for Convolutional Neural Network (CNN)

1. Convolution Operation:

$$z[i, j] = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} x[i+m, j+n] \cdot w[m, n] + b$$

- $z[i, j]$: Convolution output at position (i, j) .
- $x[i+m, j+n]$: Input patch.
- $w[m, n]$: Kernel/filter weights.
- b : Bias.
- k_h, k_w : Kernel height and width.

2. Activation Function:

$$a[i, j] = f(z[i, j])$$

- $a[i, j]$: Activated output.
- f : Activation function (e.g., ReLU).

3. **Pooling Operation** (e.g., Max Pooling):

$$p[i, j] = \max_{(m,n) \in R} a[i + m, j + n]$$

- $p[i, j]$: Pooling output.
- R : Region of pooling (e.g., 2×2 \times 2).

4. **Flattening and Fully Connected Layer**: After the convolutional and pooling layers:

$$z = W \cdot x + b$$

(Similar to NN fully connected layer).

CNN Equations include convolution and pooling operations in addition to fully connected layers.

Preventing Overfitting

Overfitting happens when a model performs well on training data but poorly on unseen data.

1. **Regularization**:

- Add a penalty to the loss function to reduce model complexity.
- **L2 Regularization** (Ridge): Adds $\lambda \sum W^2$ to the loss.
- **L1 Regularization** (Lasso): Adds $\lambda \sum |W|$ to the loss.

2. **Dropout**:

- Randomly deactivate neurons during training to reduce dependency on specific neurons.
- Example: Dropout rate of 0.5 deactivates 50% of neurons in a layer.

3. **Data Augmentation**:

- Expand the dataset by applying transformations like rotations, scaling, or flipping (especially for images).

4. **Early Stopping**:

- Stop training when the validation loss stops improving to avoid overtraining.

5. **Reduce Model Complexity**:

- Simplify the architecture by using fewer layers or neurons.

6. **Increase Training Data**:

- Add more samples to reduce the model's tendency to memorize patterns.

7. **Batch Normalization**:

- Stabilizes and accelerates training, reducing reliance on specific feature values.

Preventing Underfitting

Underfitting occurs when the model fails to capture patterns in the data and performs poorly on both training and validation datasets.

1. Increase Model Complexity:

- Use more layers, neurons, or advanced architectures to capture complex patterns.

2. Train Longer:

- Allow the model to train for more epochs if the training loss has not converged.

3. Use Appropriate Features:

- Ensure the input data has relevant features that reflect the problem domain.

4. Optimize Hyperparameters:

- Adjust learning rate, batch size, or weight initialization methods for better learning.

5. Reduce Regularization:

- Lower the strength of regularization techniques (e.g., smaller L2 or L1 coefficients).

6. Use Better Activation Functions:

- Use non-linear activations like ReLU, Tanh, or Leaky ReLU to model complex patterns effectively.

7. Ensure Sufficient Data Preprocessing:

- Scale or normalize inputs for better gradient flow and convergence.

Neural Networks (NNs) and Convolutional Neural Networks (CNNs) Comparison:

Feature	Neural Networks (NN)	Convolutional Neural Networks (CNN)
Architecture	Fully connected layers, where every neuron is linked to all neurons in adjacent layers.	Includes convolutional, pooling, and fully connected layers.
Input Type	Works with unstructured or structured data (e.g., tabular data).	Specialized for grid-like data such as images or time-series data.
Parameter Efficiency	Has a large number of parameters as every neuron connects to others.	Uses weight sharing in convolution layers, reducing parameters.
Feature Extraction	Relies on manually defined features or learned features in deeper layers.	Automatically extracts hierarchical features from data (e.g., edges, shapes).
Spatial Awareness	Does not account for spatial relationships in data.	Preserves spatial relationships, crucial for tasks like image recognition.
Performance on Images	Inefficient and less accurate for image-related tasks.	Highly effective for image and video processing.
Computational Complexity	Higher due to more parameters and connections.	Lower due to local connectivity and weight sharing.
Overfitting	More prone to overfitting with high-dimensional data.	Less prone to overfitting when using techniques like dropout and pooling.
Use Cases	Tabular data, time-series prediction, and non-image tasks.	Image recognition, object detection, and video analysis.

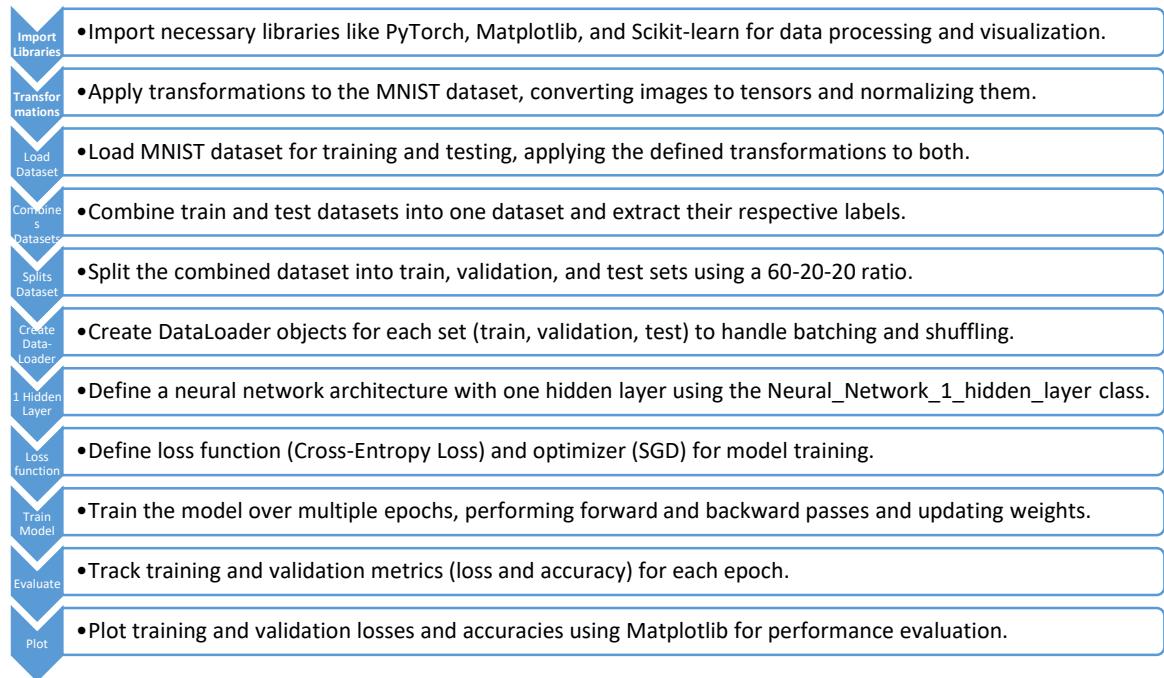
While NNs are more general-purpose and can be applied to various types of data, CNNs are specifically designed for spatially organized data like images. CNNs outperform traditional NNs in tasks requiring feature extraction from visual data.

What does the code do?

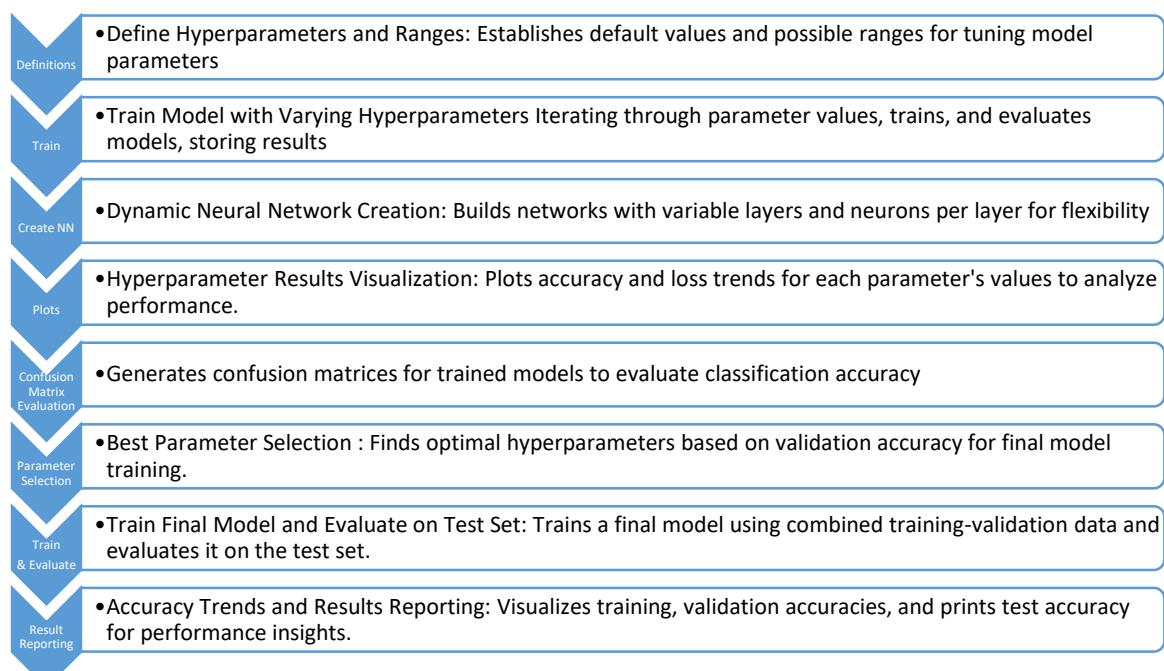
Part 1:

trains a neural network on the MNIST dataset and evaluates its performance by plotting loss and accuracy curves.

Part 1 flow code:

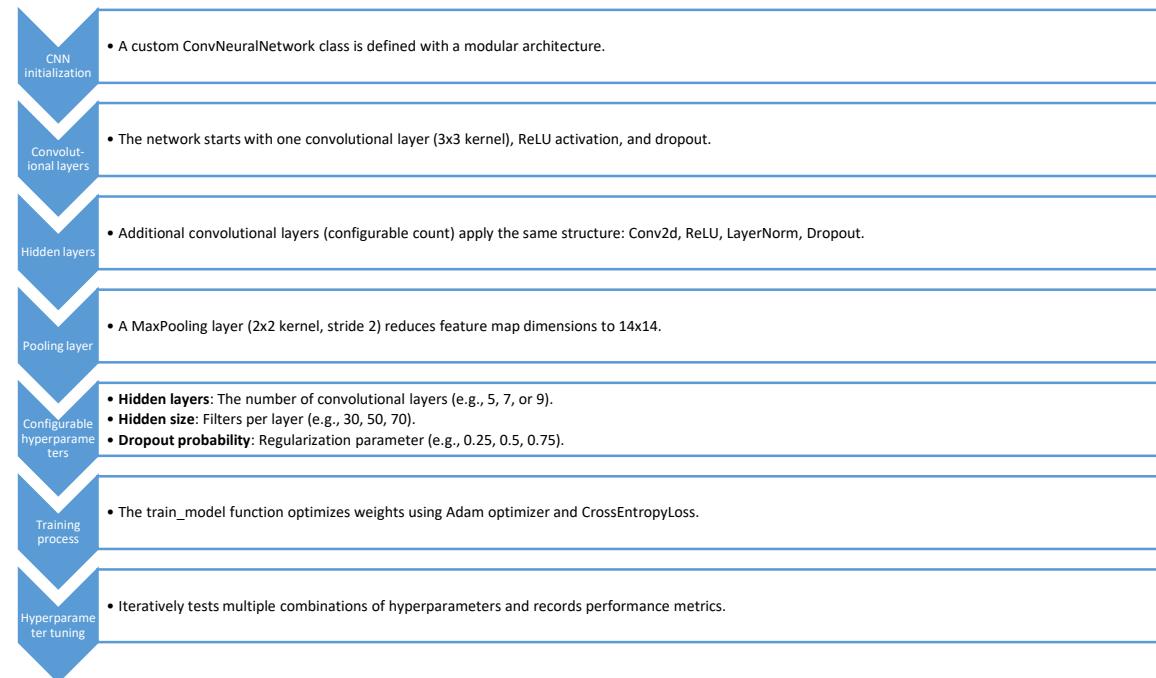
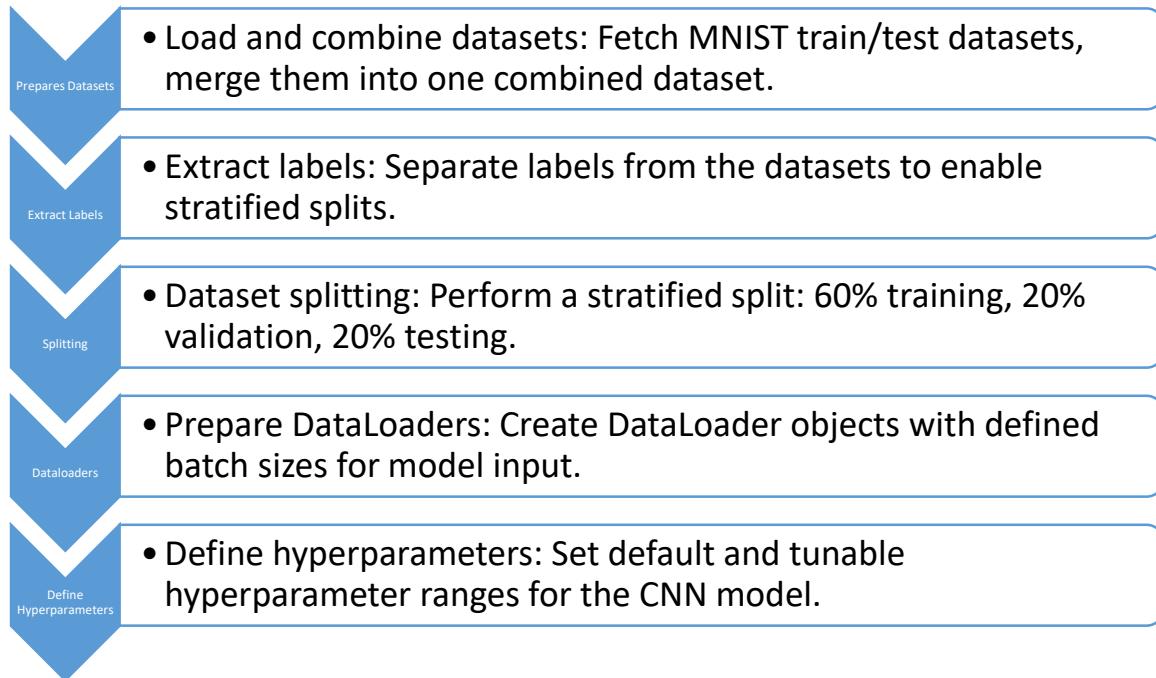


Code Continuation:

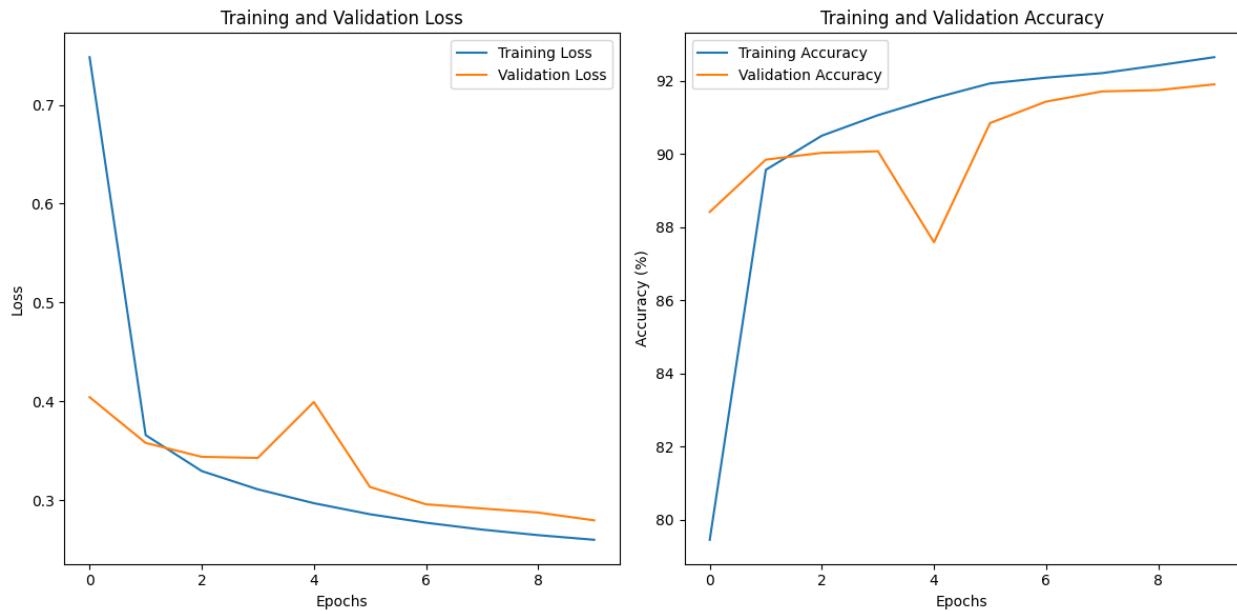


Bonus part of code: (CNN)

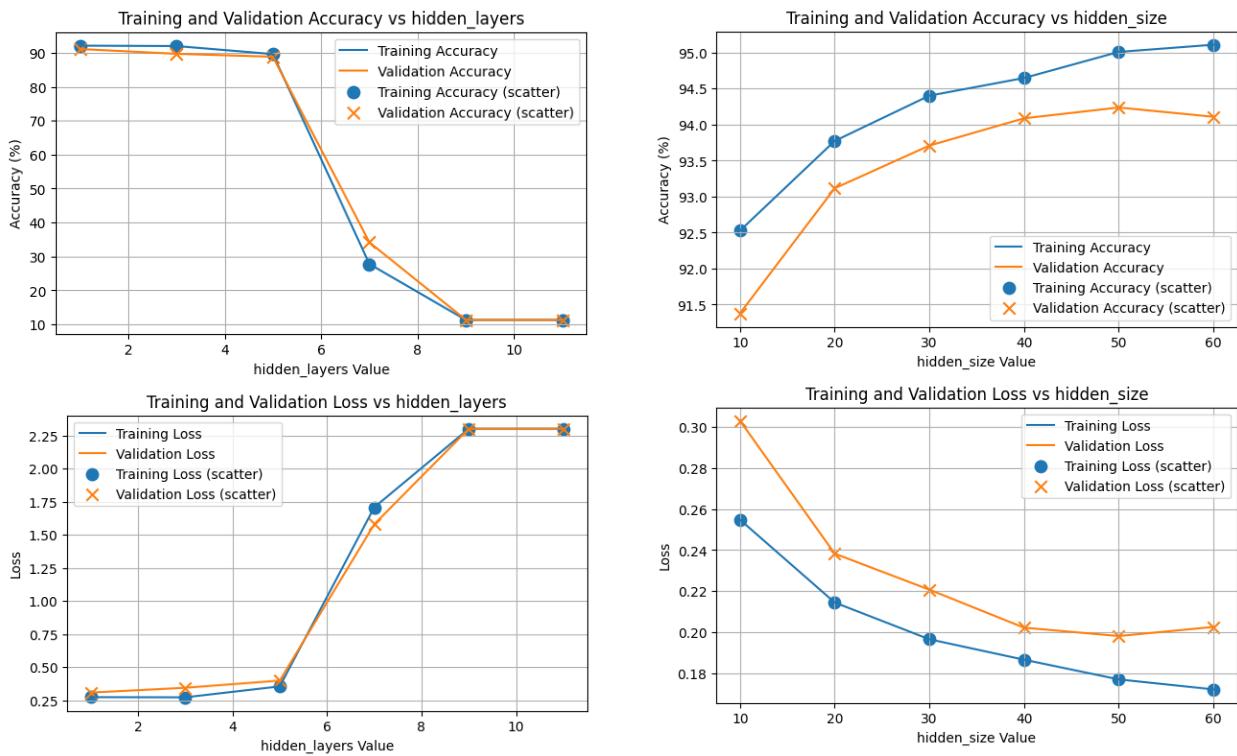
Here unlike Neural Networks (NN) there is no need to flatten the matrix first



Graphs and their explanation:

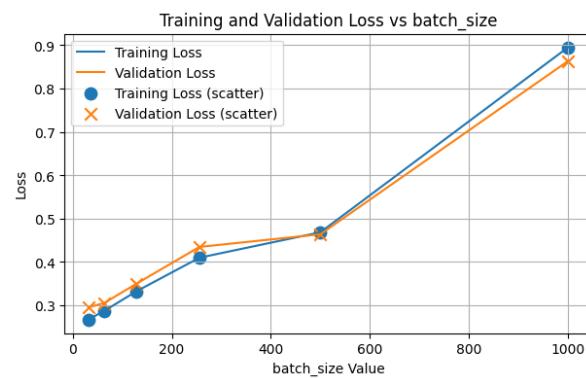
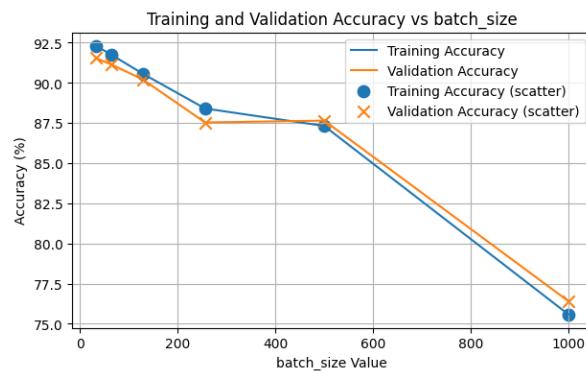


This was before tuning any parameters (base case)

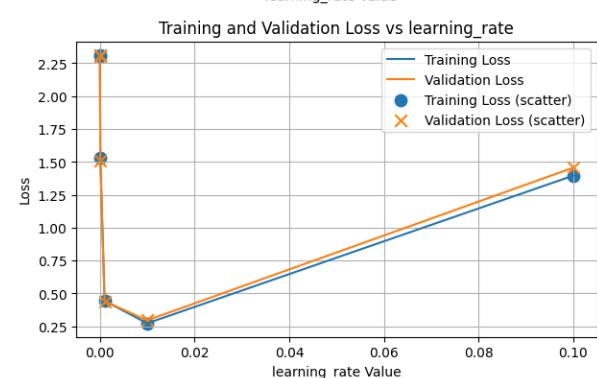
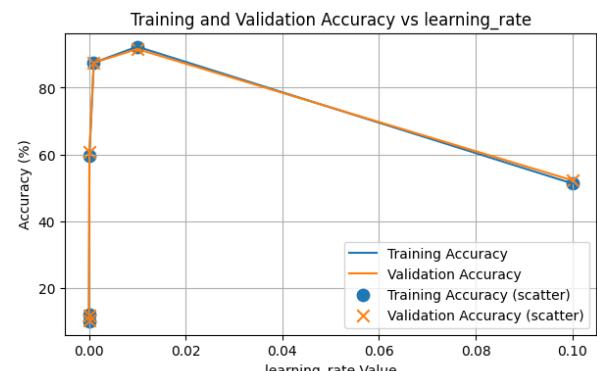


Accuracy and loss graphs above show low variance and high bias indicating underfit. Optimal range from 1 to 5 for number of hidden layers

Accuracy and loss graphs above show high variance and low bias indicating overfit, optimal range from 30 to 50 for number of neurons



Accuracy and loss graphs above show low variance and high bias indicating underfit ,optimal range here from 250 to 500 batch size.



Accuracy and loss graphs above show low variance and high bias indicating underfit ,optimal range here from 0.00001 to 0.01 for learning rate

In the confusion matrices below we are hoping the negative diagonal have the greatest values possible (darkest shades of blue)

Confusion Matrix for hidden_layers = 1.0											
True	0	1	2	3	4	5	6	7	8	9	Predicted
0	13358	0	34	40	21	105	127	23	92	10	
1	- 0	15214	91	87	11	71	18	83	168	17	
2	- 115	173	12297	211	289	53	220	199	381	42	
3	- 70	67	407	12591	4	451	33	190	330	137	
True	4	- 24	81	93	10	12538	8	151	45	130	570
5	- 212	95	159	559	187	10466	206	123	406	207	
6	- 89	45	153	6	203	156	12996	14	77	11	
7	- 46	85	188	37	141	38	0	13640	28	377	
8	- 83	310	172	451	65	338	135	71	11776	249	
9	- 68	113	71	172	597	108	5	597	196	11993	
	0	1	2	3	4	5	6	7	8	9	

Confusion Matrix for hidden_layers = 9.0											
True	0	1	2	3	4	5	6	7	8	9	Predicted
0	0	13810	0	0	0	0	0	0	0	0	
1	- 0	15760	0	0	0	0	0	0	0	0	
2	- 0	13980	0	0	0	0	0	0	0	0	
3	- 0	14280	0	0	0	0	0	0	0	0	
True	4	- 0	13650	0	0	0	0	0	0	0	
5	- 0	12620	0	0	0	0	0	0	0	0	
6	- 0	13750	0	0	0	0	0	0	0	0	
7	- 0	14580	0	0	0	0	0	0	0	0	
8	- 0	13650	0	0	0	0	0	0	0	0	
9	- 0	13920	0	0	0	0	0	0	0	0	
	0	1	2	3	4	5	6	7	8	9	

Confusion Matrix for hidden_layers = 3.0											
True	0	1	2	3	4	5	6	7	8	9	Predicted
0	- 13009	12	98	29	22	427	86	68	48	11	
1	- 4	15132	141	77	10	32	17	105	208	34	
2	- 199	308	11924	292	221	260	259	204	287	26	
3	- 55	132	419	11851	14	783	14	295	610	107	
True	4	- 26	59	47	5	11872	66	220	100	75	1180
5	- 418	60	400	765	193	9521	280	174	461	348	
6	- 166	50	355	3	288	286	12479	2	118	3	
7	- 72	192	123	142	97	42	1	13248	15	648	
8	- 68	805	179	696	52	914	166	89	10396	285	
9	- 68	82	24	170	720	191	6	1019	170	11470	
	0	1	2	3	4	5	6	7	8	9	

Confusion Matrix for hidden_layers = 11.0											
True	0	1	2	3	4	5	6	7	8	9	Predicted
0	0	13810	0	0	0	0	0	0	0	0	
1	- 0	15760	0	0	0	0	0	0	0	0	
2	- 0	13980	0	0	0	0	0	0	0	0	
3	- 0	14280	0	0	0	0	0	0	0	0	
True	4	- 0	13650	0	0	0	0	0	0	0	
5	- 0	12620	0	0	0	0	0	0	0	0	
6	- 0	13750	0	0	0	0	0	0	0	0	
7	- 0	14580	0	0	0	0	0	0	0	0	
8	- 0	13650	0	0	0	0	0	0	0	0	
9	- 0	13920	0	0	0	0	0	0	0	0	
	0	1	2	3	4	5	6	7	8	9	

Confusion Matrix for hidden_layers = 5.0											
True	0	1	2	3	4	5	6	7	8	9	Predicted
0	- 10287	76	154	106	24	232	2024	2	875	30	
1	- 3	15070	113	144	96	10	10	109	104	101	
2	- 280	861	9103	1021	381	52	1212	85	782	203	
3	- 210	873	1623	8382	385	301	876	194	1005	431	
True	4	- 87	969	888	407	8886	19	567	176	430	1221
5	- 357	434	564	699	146	6810	2085	47	1145	333	
6	- 213	95	479	67	92	232	11586	0	971	15	
7	- 2	3029	52	140	120	17	13	10735	12	460	
8	- 199	1037	775	1088	655	435	950	34	8103	374	
9	- 29	2809	118	263	1107	103	88	1097	228	8078	
	0	1	2	3	4	5	6	7	8	9	

Confusion Matrix for hidden_size = 10.0											
True	0	1	2	3	4	5	6	7	8	9	Predicted
0	- 13366	0	65	31	9	123	98	26	86	6	
1	- 0	15227	90	111	6	61	13	66	169	17	
2	- 99	143	12583	141	219	73	234	137	310	41	
3	- 60	48	460	12615	2	470	44	141	365	75	
True	4	- 27	62	84	30	12308	20	206	22	138	753
5	- 182	88	200	557	195	10605	205	78	341	169	
6	- 104	47	172	0	109	209	13053	0	55	1	
7	- 62	99	254	40	129	79	0	13386	41	490	
8	- 80	325	213	510	41	371	137	62	11768	143	
9	- 132	110	54	256	686	92	14	458	165	11953	
	0	1	2	3	4	5	6	7	8	9	

Confusion Matrix for hidden_layers = 7.0											
True	0	1	2	3	4	5	6	7	8	9	Predicted
0	- 2555	11162	0	10	1	38	36	5	0	3	
1	- 1544	13722	0	34	1	11	441	6	0	1	
2	- 2120	11446	0	97	32	32	170	43	0	40	
3	- 2336	11678	0	50	13	34	131	17	0	21	
True	4	- 22	10949	0	16	578	0	151	235	0	1699
5	- 1552	10645	0	40	5	94	255	18	0	11	
6	- 726	11307	0	203	104	12	1186	165	0	47	
7	- 68	11728	0	35	590	0	303	344	0	1512	
8	- 1072	11555	0	168	46	45	656	83	0	25	
9	- 64	11190	0	40	569	1	174	229	0	1653	
	0	1	2	3	4	5	6	7	8	9	

Confusion Matrix for hidden_size = 20.0											
True	0	1	2	3	4	5	6	7	8	9	Predicted
0	- 13407	0	34	44	17	95	115	8	88	2	
1	- 0	15276	74	89	11	71	23</td				

Confusion Matrix for hidden_size = 30.0										
True	0	1	2	3	4	5	6	7	8	
0	13436	1	27	50	20	72	112	16	73	3
1	0	15332	75	78	10	78	20	59	85	23
2	119	170	12516	183	263	64	209	167	253	36
3	74	74	386	12797	6	422	26	152	212	131
4	33	65	52	18	12618	8	195	22	91	548
5	155	89	119	440	127	11047	187	86	206	164
6	80	50	89	1	158	170	13157	7	37	1
7	52	109	210	43	160	38	0	13570	19	379
8	77	320	96	432	45	387	134	50	11865	244
9	82	123	52	195	491	113	7	352	179	12326
	0	1	2	3	4	5	6	7	8	9
	Predicted									

Confusion Matrix for batch_size = 32.0										
True	0	1	2	3	4	5	6	7	8	
0	13405	0	46	44	13	73	104	39	86	0
1	0	15147	105	88	10	75	21	95	198	21
2	135	127	12294	230	302	40	267	185	368	32
3	104	47	392	12505	4	473	35	220	368	132
4	30	57	79	30	12458	10	183	39	133	631
5	208	87	132	590	157	10539	242	123	372	170
6	106	34	154	5	192	183	13005	6	59	6
7	54	43	217	16	147	35	0	13607	42	419
8	87	236	163	498	43	402	133	56	11781	251
9	119	63	60	218	524	110	11	633	186	11996
	0	1	2	3	4	5	6	7	8	9
	Predicted									

Confusion Matrix for hidden_size = 40.0										
True	0	1	2	3	4	5	6	7	8	
0	13370	0	39	37	16	91	133	22	94	8
1	0	15296	67	104	10	69	20	53	120	21
2	91	177	12555	194	223	44	230	174	252	40
3	67	64	291	12846	3	423	37	147	274	128
4	29	85	60	29	12616	8	167	40	119	497
5	125	107	88	428	121	11070	189	87	244	161
6	59	41	73	5	119	173	13219	5	54	2
7	36	109	199	46	135	43	0	13668	25	319
8	70	324	69	393	32	310	133	68	12052	199
9	62	131	63	230	461	122	12	446	179	12214
	0	1	2	3	4	5	6	7	8	9
	Predicted									

Confusion Matrix for batch_size = 64.0										
True	0	1	2	3	4	5	6	7	8	
0	13205	0	57	47	22	212	129	45	92	1
1	1	15228	86	94	16	82	27	67	136	23
2	161	253	12031	261	302	61	257	198	402	54
3	96	71	433	12285	6	625	41	199	355	169
4	28	65	77	12	12309	39	197	36	110	777
5	218	136	129	523	198	10562	206	124	323	201
6	96	53	166	1	221	234	12884	13	73	9
7	56	170	180	29	174	73	2	13420	41	435
8	72	344	177	502	57	558	134	66	11442	298
9	103	127	72	205	694	156	13	558	149	11843
	0	1	2	3	4	5	6	7	8	9
	Predicted									

Confusion Matrix for hidden_size = 50.0										
True	0	1	2	3	4	5	6	7	8	
0	13399	0	28	34	19	69	128	39	82	12
1	0	15326	66	77	10	63	29	61	104	24
2	89	196	12525	194	244	50	193	191	252	46
3	71	74	252	12925	8	349	33	167	245	156
4	25	84	60	13	12553	2	159	38	104	612
5	120	86	96	413	120	11085	178	95	249	178
6	74	58	53	8	127	148	13194	12	73	3
7	22	89	154	35	120	38	0	13855	27	240
8	78	363	49	403	40	361	131	77	11925	223
9	65	122	50	194	416	112	10	417	161	12373
	0	1	2	3	4	5	6	7	8	9
	Predicted									

Confusion Matrix for batch_size = 128.0										
True	0	1	2	3	4	5	6	7	8	
0	13203	0	66	64	18	233	97	47	82	0
1	0	15260	85	99	7	64	21	54	147	23
2	173	512	11529	249	301	64	432	304	375	41
3	84	91	569	12147	28	623	37	227	313	161
4	36	87	91	10	11828	19	258	12	180	1129
5	225	163	244	705	103	9997	322	138	464	259
6	115	154	197	8	208	307	12668	16	76	1
7	102	286	188	27	160	46	1	13171	52	547
8	57	578	173	575	74	497	153	97	11182	264
9	97	153	109	246	929	117	15	646	220	11388
	0	1	2	3	4	5	6	7	8	9
	Predicted									

Confusion Matrix for hidden_size = 60.0										
True	0	1	2	3	4	5	6	7	8	
0	13441	1	16	46	11	58	118	32	80	7
1	0	15307	68	95	10	66	24	70	100	20
2	108	137	12569	201	252	59	177	201	235	41
3	91	67	306	12838	4	384	46	178	228	138
4	25	78	65	14	12617	10	184	31	95	531
5	134	94	109	415	119	11141	177	99	157	175
6	73	49	53	4	142	189	13172	17	48	3
7	36	92	146	28	125	33	0	13829	26	265
8	71	300	86	392	26	295	138	95	12042	205
9	88	126	53	207	367	89	7	49		

Confusion Matrix for batch_size = 500.0										
True	0	1	2	3	4	5	6	7	8	
0	13127	6	61	76	9	152	244	81	54	0
1	0	15339	91	105	0	59	15	37	104	10
2	374	796	10430	525	275	39	414	601	494	32
3	268	644	845	10714	36	775	57	455	255	231
4	189	334	27	13	8604	23	374	278	698	3110
5	647	633	205	966	270	8569	208	100	409	613
6	886	506	188	354	94	220	10836	258	407	1
7	142	607	165	59	146	14	8	11729	29	1681
8	385	1207	285	483	157	994	77	220	9431	411
9	230	357	71	145	972	174	51	1154	153	10613
	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9
	True	0	1	2	3	4	5	6	7	8
	0	1	2	3	4	5	6	7	8	9

Confusion Matrix for learning_rate = 0.001										
True	0	1	2	3	4	5	6	7	8	
0	13044	0	41	1	37	208	150	96	230	3
1	1	13649	247	136	9	472	49	28	963	206
2	308	654	10080	65	688	90	457	340	1236	62
3	116	309	250	11025	85	749	74	211	1303	158
4	4	19	81	51	49	10987	50	328	34	1975
5	436	463	251	1015	264	8289	306	137	1085	374
6	218	226	605	5	802	775	10758	86	99	176
7	107	285	96	21	300	36	2	12890	138	705
8	96	470	350	692	169	642	170	105	10686	270
9	141	123	54	386	1716	225	78	817	164	10216
	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9
	True	0	1	2	3	4	5	6	7	8
	0	1	2	3	4	5	6	7	8	9

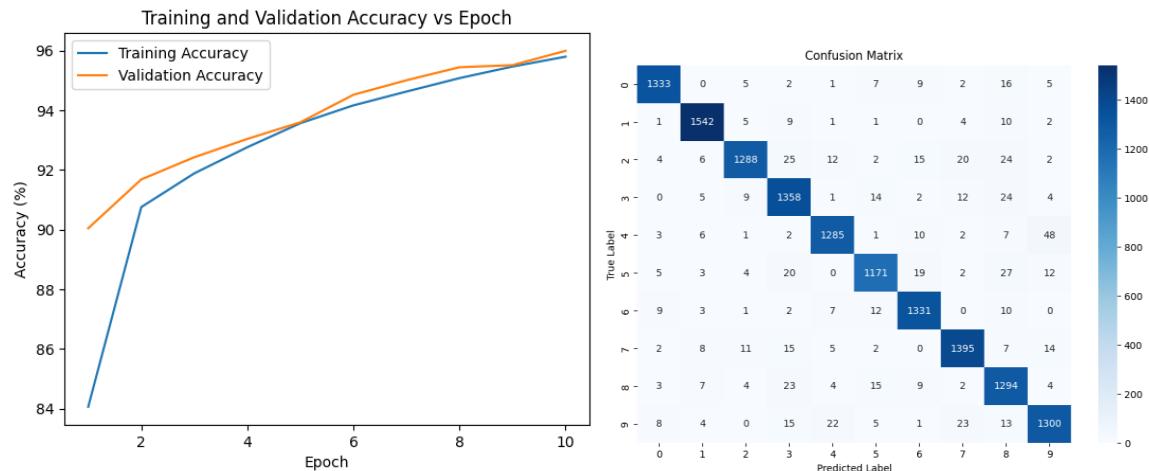
Confusion Matrix for learning_rate = 0.0001										
True	0	1	2	3	4	5	6	7	8	
0	0	169	184	232	2186	5861	2446	641	1370	721
1	0	15085	258	0	16	3	111	42	5	240
2	0	1818	8975	5	169	35	1834	293	636	215
3	0	4428	1081	1321	843	665	969	159	2499	2315
4	0	232	1247	0	6183	585	816	1470	166	2951
5	0	1481	188	42	1725	4280	1172	563	1186	1983
6	0	755	847	0	102	20	11724	70	196	36
7	0	820	566	1	908	133	329	10662	86	1075
8	0	2676	704	10	1354	241	965	267	5724	1709
9	0	439	319	4	3743	785	381	1503	172	6574
	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9
	True	0	1	2	3	4	5	6	7	8
	0	1	2	3	4	5	6	7	8	9

Confusion Matrix for learning_rate = 0.1										
True	0	1	2	3	4	5	6	7	8	
0	9981	14	1306	904	26	1100	358	26	85	10
1	3	13688	63	31	165	51	14	607	85	1053
2	629	646	2452	773	1806	2599	1700	1116	1644	615
3	402	260	377	6218	108	1865	36	4419	421	174
4	5	3216	676	2	6703	64	1570	79	259	1076
5	1000	115	2034	1795	336	4274	647	1104	1142	173
6	403	100	1217	14	1819	206	9716	2	248	25
7	9	1673	146	2431	152	168	15	8917	158	911
8	162	1201	1975	627	1744	2157	796	2202	1668	1118
9	32	6539	338	174	1666	198	102	2196	328	2347
	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9
	True	0	1	2	3	4	5	6	7	8
	0	1	2	3	4	5	6	7	8	9

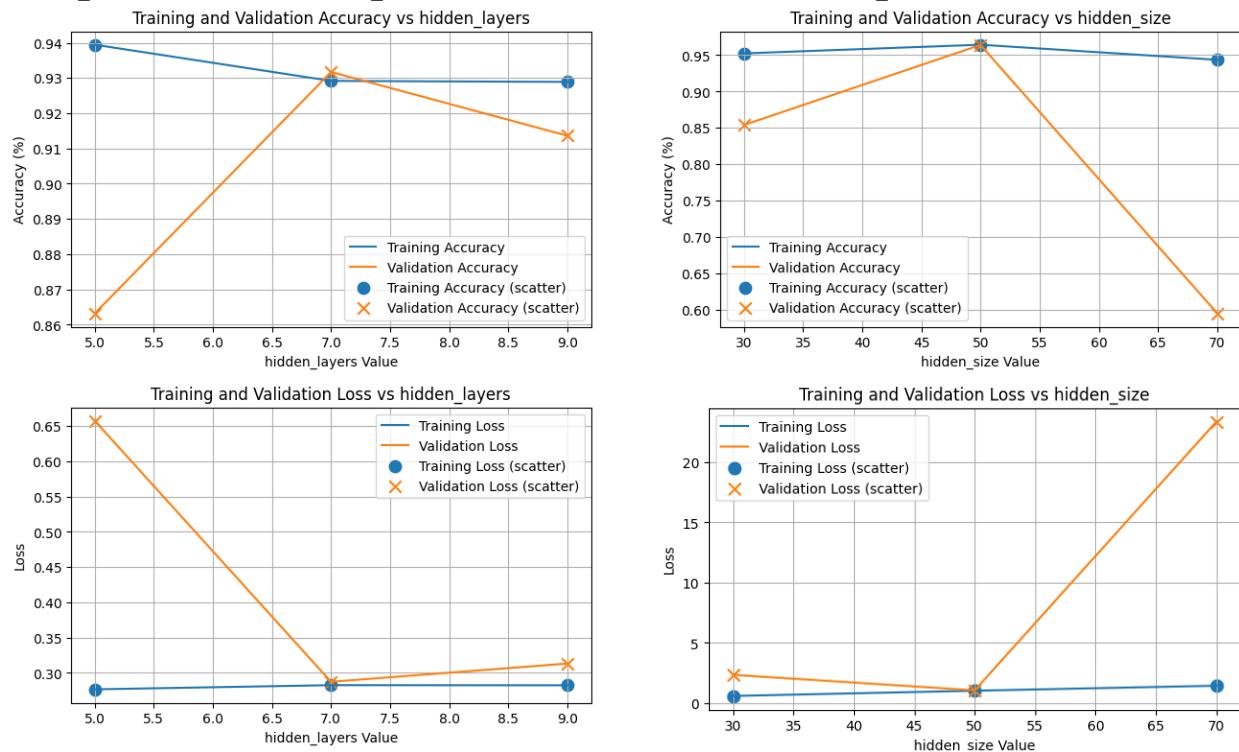
Confusion Matrix for learning_rate = 1e-05										
True	0	1	2	3	4	5	6	7	8	
0	0	0	380	0	0	0	0	0	0	13430
1	0	0	1457	0	0	0	0	0	0	14303
2	0	0	4135	528	93	0	0	0	0	9224
3	0	0	2283	6	0	0	0	0	0	11991
4	0	0	2954	3	0	0	0	0	0	10693
5	0	0	1285	1	9	0	0	0	0	11325
6	0	0	1786	8	152	0	0	0	0	11804
7	0	0	2684	94	4	0	0	0	0	11798
8	0	0	1620	0	9	0	0	0	0	12021
9	0	0	1713	38	0	0	0	0	0	12169
	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9
	True	0	1	2	3	4	5	6	7	8
	0	1	2	3	4	5	6	7	8	9

Confusion Matrix for learning_rate = 0.01										
True	0	1	2	3	4	5	6	7	8	
0	13202	0	73	60	21	222	122	10	77	23
1	0	15160	104	89	21	64	32	57	204	29
2	119	194	12339	243	295	80	291	152	236	31
3	88	68	432	12312	3	680	21	136	435	105
4	25	70	96	15	12546	26	198	22	126	526
5	147	25	182	568	192	10743	182	84	335	162
6	121	110	231	0	226	231	12750	1	66	14
7	50	102	275	49	174	95	0	13378	35	422
8	58	357	204	470	39	458	125	41	11686	212
9	146	108	70	207	870	145	5	369	204	11796
	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9
	True	0	1	2	3	4	5	6	7	8
	0	1	2	3	4	5	6	7	8	9

Confusion Matrix for learning_rate = 1e-06										
True	0	1	2	3	4	5	6	7	8	
0	0	27	909	0	0	1005	11000	0	0	869
1	0	0	9920	0	0	8	5832	0	0	0

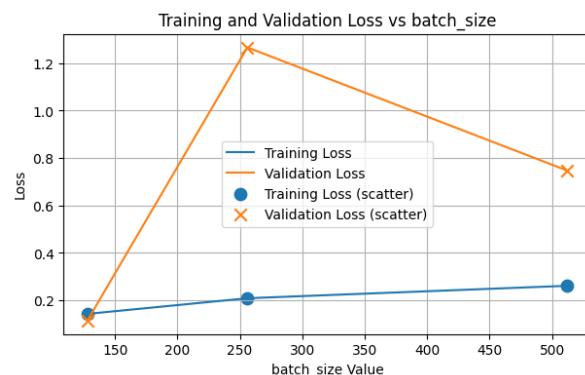
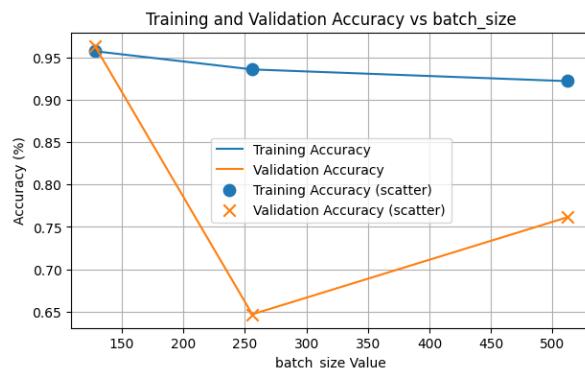


Graphs and their explanation for Bonus (CNN) part:

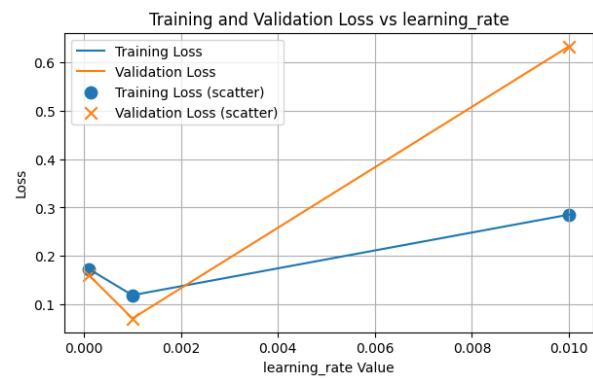
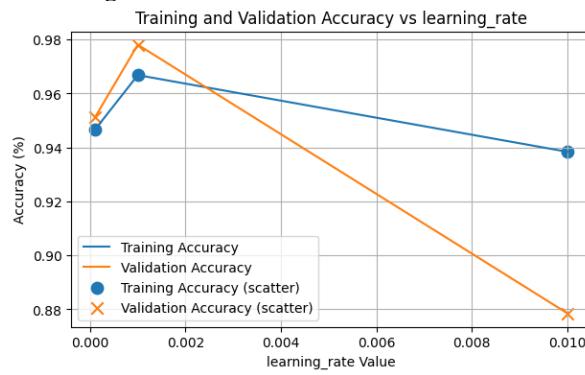


Accuracy and loss graphs above show increasing variance and low bias indicating overfit ,optimal range here from 7 to 8 hidden layers.

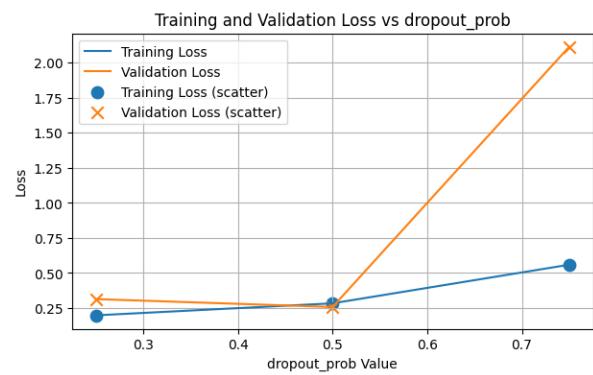
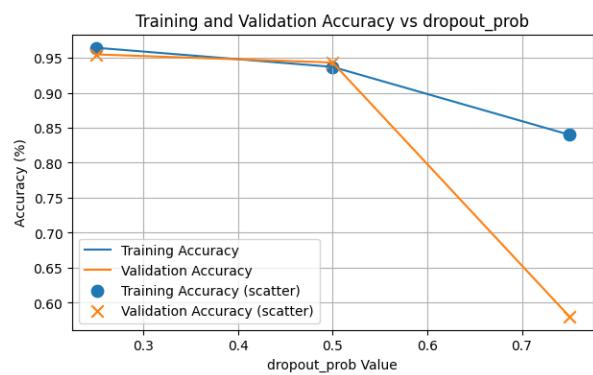
The previous accuracy and loss graphs show increasing variance and low bias indicating overfit ,optimal range here from 30 to 50 for number of neuron networks.



Accuracy and loss graphs above show decreasing variance and increasing bias indicating underfit.

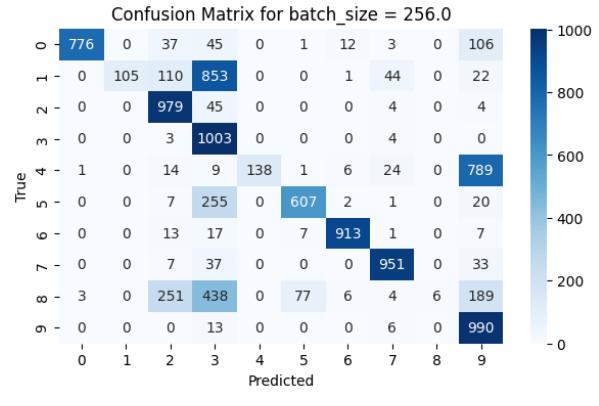
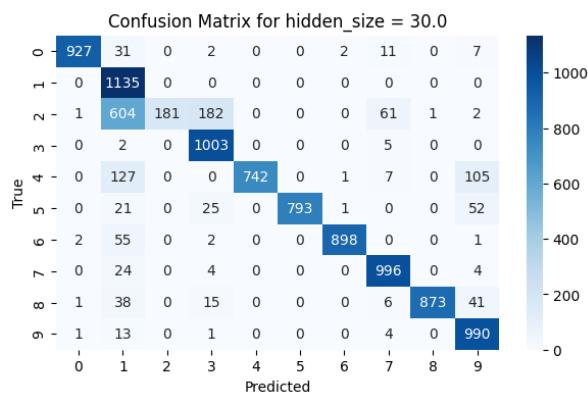
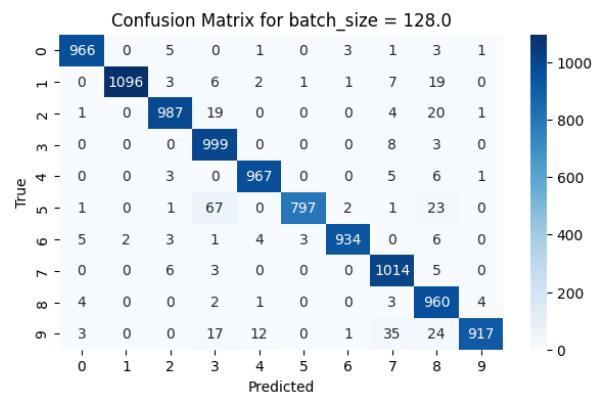
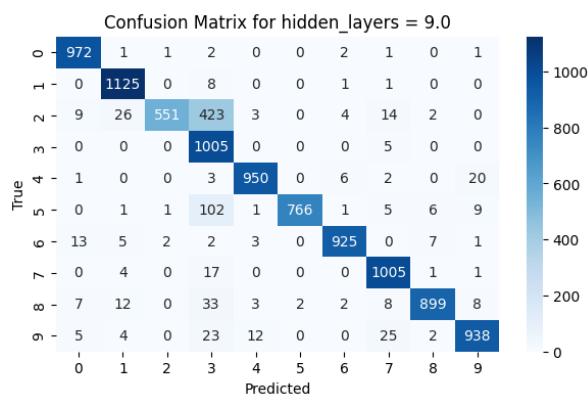
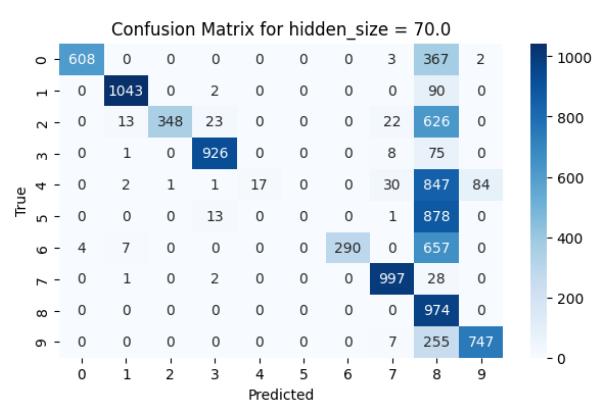
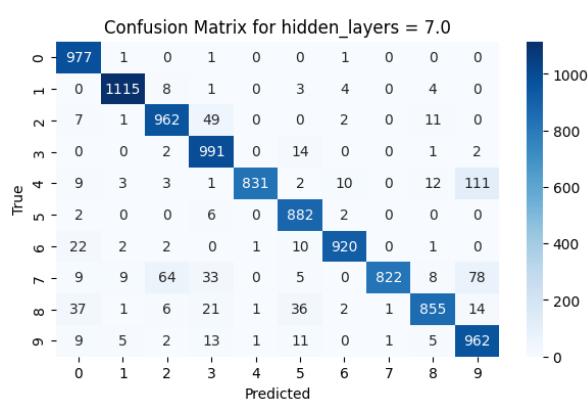
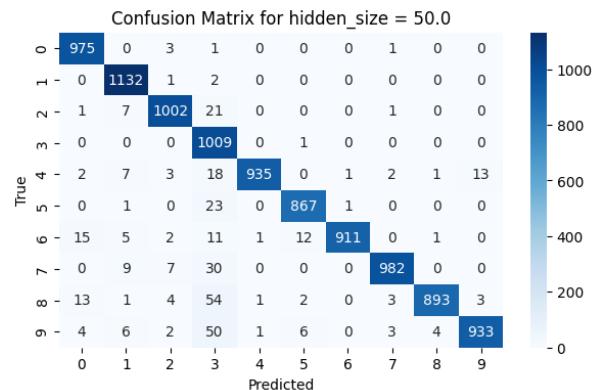
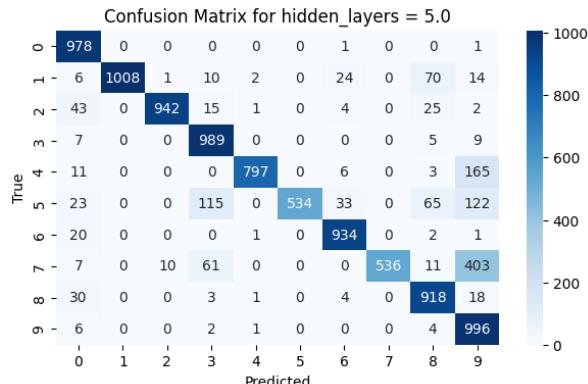


Accuracy and loss graphs above show increasing variance and increasing bias.

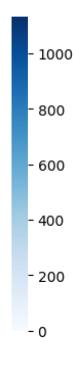


Accuracy and loss graphs above show decreasing variance and decreasing bias indicating in the optimal range 0.1 to 0.5 for drop out probability.

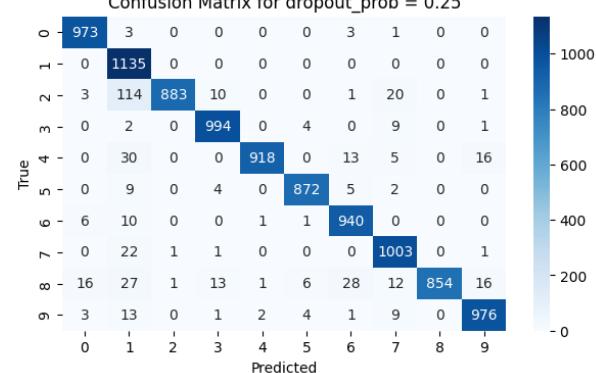
In the confusion matrices below we are hoping the negative diagonal have the greatest values possible (darkest shades of blue)



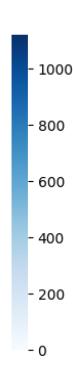
Confusion Matrix for batch_size = 512.0										
True	0	1	2	3	4	5	6	7	8	9
0	973	4	0	0	0	0	1	0	0	2
1	0	1134	0	0	0	1	0	0	0	0
2	30	337	630	0	3	1	3	6	2	20
3	38	539	10	241	0	66	0	6	2	108
4	8	77	0	0	781	0	3	0	0	113
5	5	48	1	0	1	824	6	2	0	5
6	16	52	0	0	4	5	880	0	0	1
7	1	160	5	0	8	0	0	626	0	228
8	8	60	276	0	0	8	6	3	0	552
9	6	27	0	0	2	2	0	0	0	972
0	1	2	3	4	5	6	7	8	9	9
Predicted										



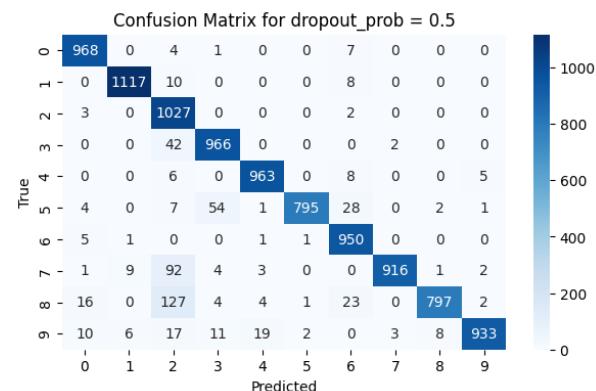
Confusion Matrix for dropout_prob = 0.25										
True	0	1	2	3	4	5	6	7	8	9
0	973	3	0	0	0	0	3	1	0	0
1	0	1135	0	0	0	0	0	0	0	0
2	3	114	883	10	0	0	0	1	20	0
3	0	2	0	994	0	4	0	9	0	1
4	0	30	0	0	918	0	13	5	0	16
5	0	9	0	4	0	872	5	2	0	0
6	6	10	0	0	1	1	940	0	0	0
7	0	22	1	1	0	0	0	1003	0	1
8	16	27	1	13	1	6	28	12	854	16
9	3	13	0	1	2	4	1	9	0	976
0	1	2	3	4	5	6	7	8	9	9
Predicted										



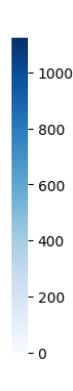
Confusion Matrix for learning_rate = 0.01										
True	0	1	2	3	4	5	6	7	8	9
0	940	0	2	2	2	26	4	1	0	3
1	0	1122	1	6	0	2	1	3	0	0
2	0	15	845	135	3	5	0	29	0	0
3	0	0	0	990	0	12	0	8	0	0
4	0	1	4	0	942	5	1	7	0	22
5	0	0	0	3	0	889	0	0	0	0
6	3	5	0	0	10	93	847	0	0	0
7	0	0	1	8	0	2	0	1017	0	0
8	0	3	1	150	4	527	1	11	250	27
9	2	3	0	7	3	36	0	15	0	943
0	1	2	3	4	5	6	7	8	9	9
Predicted										



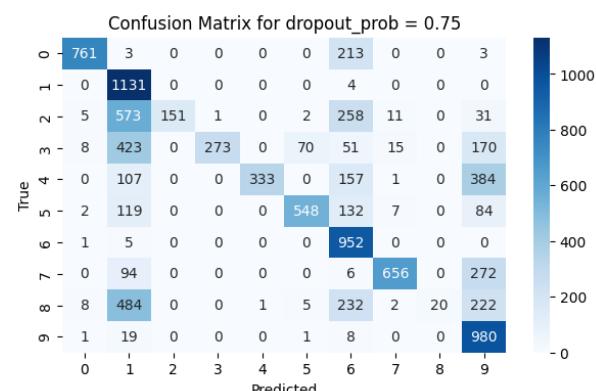
Confusion Matrix for dropout_prob = 0.5										
True	0	1	2	3	4	5	6	7	8	9
0	968	0	4	1	0	0	7	0	0	0
1	0	1117	10	0	0	0	8	0	0	0
2	3	0	1027	0	0	0	2	0	0	0
3	0	0	42	966	0	0	0	2	0	0
4	0	0	6	0	963	0	8	0	0	5
5	4	0	7	54	1	795	28	0	2	1
6	5	1	0	0	1	1	950	0	0	0
7	1	9	92	4	3	0	0	916	1	2
8	16	0	127	4	4	1	23	0	797	2
9	10	6	17	11	19	2	0	3	8	933
0	1	2	3	4	5	6	7	8	9	9
Predicted										



Confusion Matrix for learning_rate = 0.001										
True	0	1	2	3	4	5	6	7	8	9
0	979	0	0	0	0	0	0	0	0	1
1	0	1125	6	0	1	0	3	0	0	0
2	5	0	1018	0	2	0	0	0	4	3
3	1	0	2	982	0	2	0	3	11	9
4	1	0	0	0	973	0	0	0	1	7
5	0	1	4	0	844	8	1	9	20	0
6	8	3	0	0	4	0	941	0	1	1
7	2	3	11	2	0	0	0	986	2	22
8	9	0	4	1	4	1	3	0	934	18
9	2	0	0	0	2	1	0	4	1	999
0	1	2	3	4	5	6	7	8	9	9
Predicted										



Confusion Matrix for dropout_prob = 0.75										
True	0	1	2	3	4	5	6	7	8	9
0	761	3	0	0	0	0	213	0	0	3
1	0	1131	0	0	0	0	4	0	0	0
2	5	573	151	1	0	2	258	11	0	31
3	8	423	0	273	0	70	51	15	0	170
4	0	107	0	0	333	0	157	1	0	384
5	2	119	0	0	0	0	548	132	7	0
6	1	5	0	0	0	0	952	0	0	0
7	0	94	0	0	0	0	6	656	0	272
8	8	484	0	0	1	5	232	2	20	222
9	1	19	0	0	0	1	8	0	0	980
0	1	2	3	4	5	6	7	8	9	9
Predicted										



Confusion Matrix for learning_rate = 0.0001										
True	0	1	2	3	4	5	6	7	8	9
0	975	0	1	0	0	0	2	1	1	0
1	0	1129	2	1	0	0	1	0	2	0
2	17	24	964	1	6	0	3	11	5	1
3	2	5	10	959	2	3	0	8	7	14
4	6	8	0	0	948	0	4	1	0	15
5	21	17	1	5	0	805	8	2	12	21
6	17	5	0	0	2	1	931	0	2	0
7	2	22	16	4	3	0	0	944	0	37
8	32	13	4	1	12	0	4	4	881	23
9	9	10	0	1	9	1	0	4	0	975
0	1	2	3	4	5	6	7	8	9	9
Predicted										

