Deep Learning - CSE641

Assignment - 1 (Report)

Group 4

Name:	**Mahima Chandwani**	**Vrutti Patel**	**Suraj Pandey**
Roll No:	**(MT18007)**	**(MT18020)**	**(MT18025)**

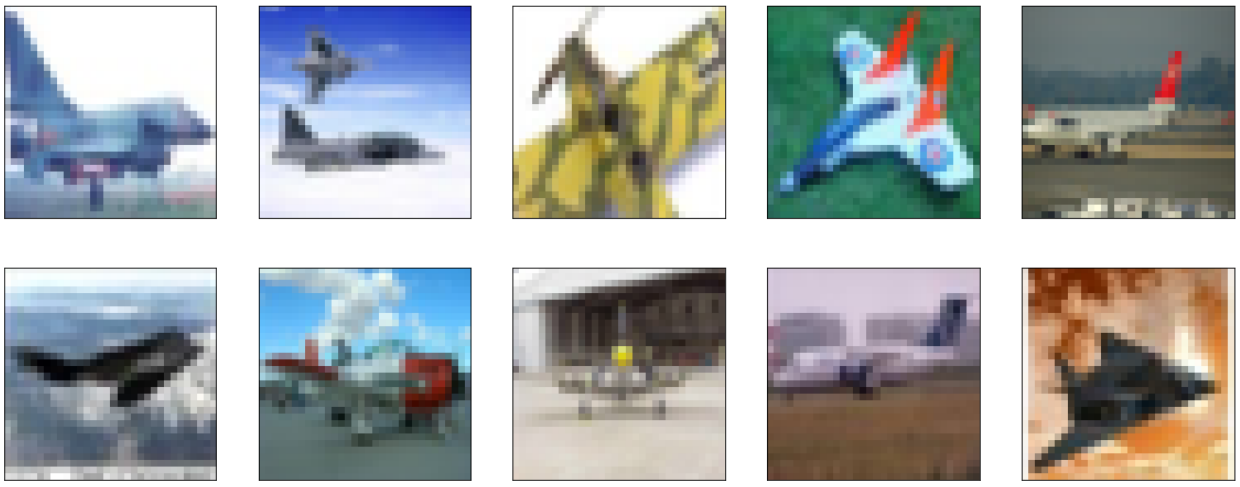
TABLE OF CONTENT

Q1) CNN architecture for given Dataset

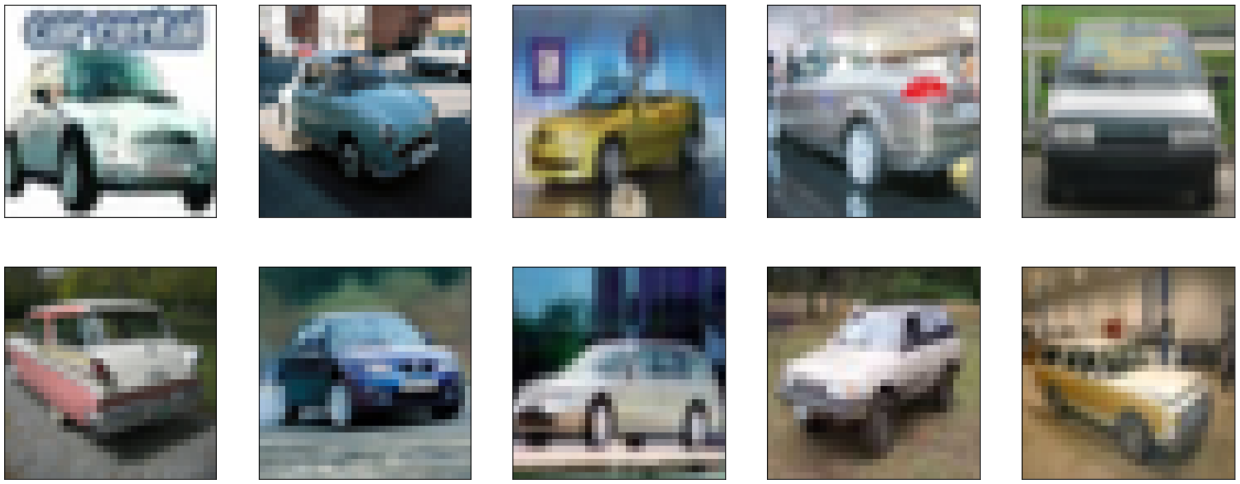
1.1 Visualize 10 random images from each class of the dataset

Class No.	0	1	2	3	4	5	6	7	8	9
Class Name	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck

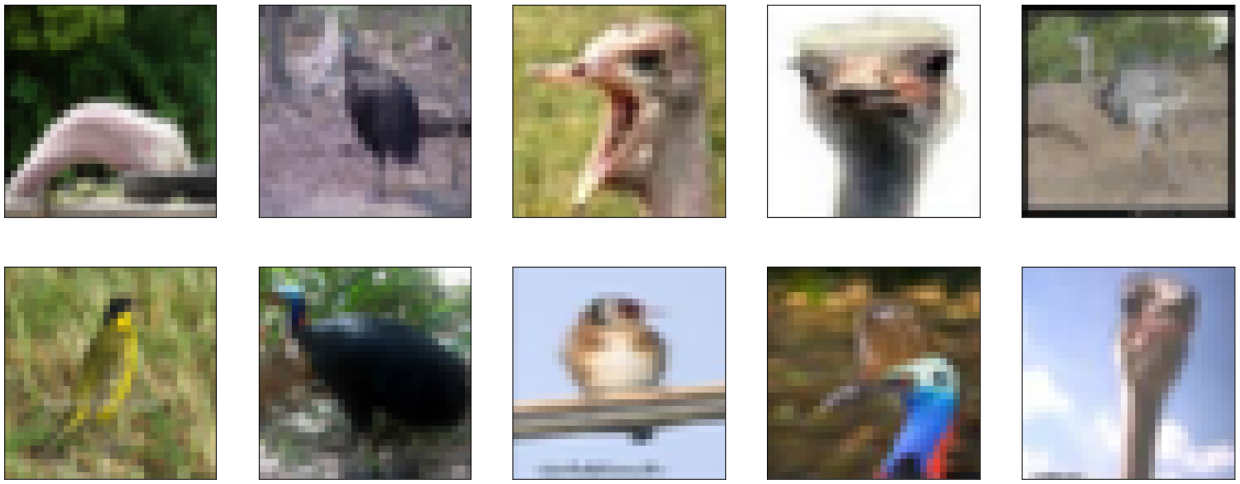
Class 0:



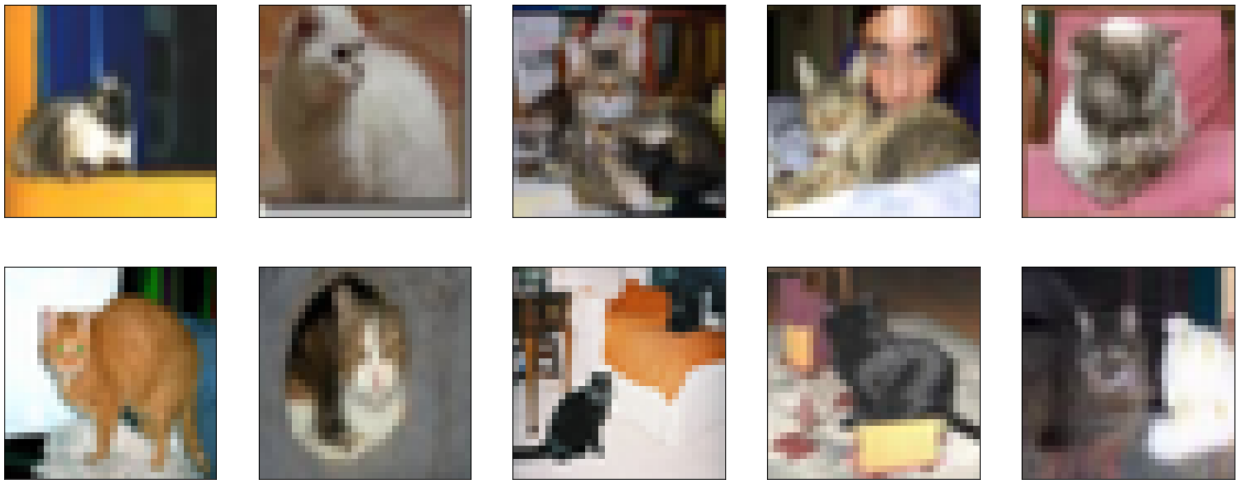
Class 1:



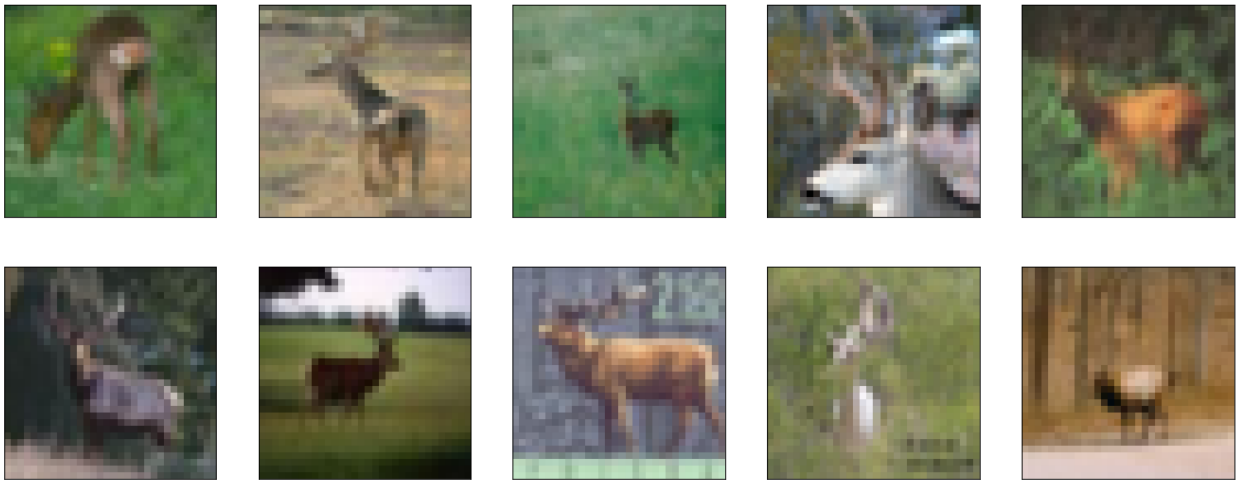
Class 2:



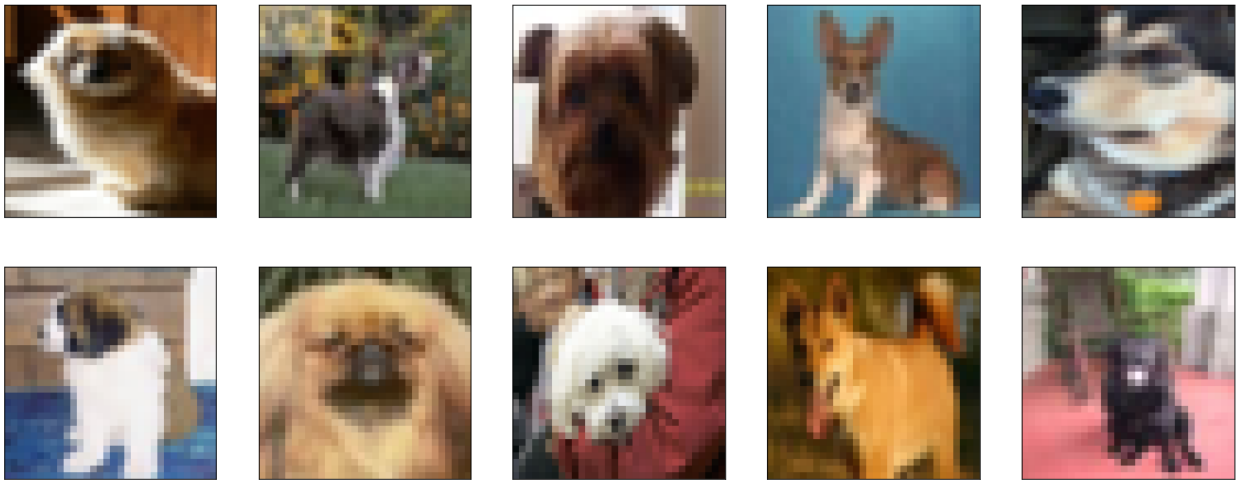
Class 3:



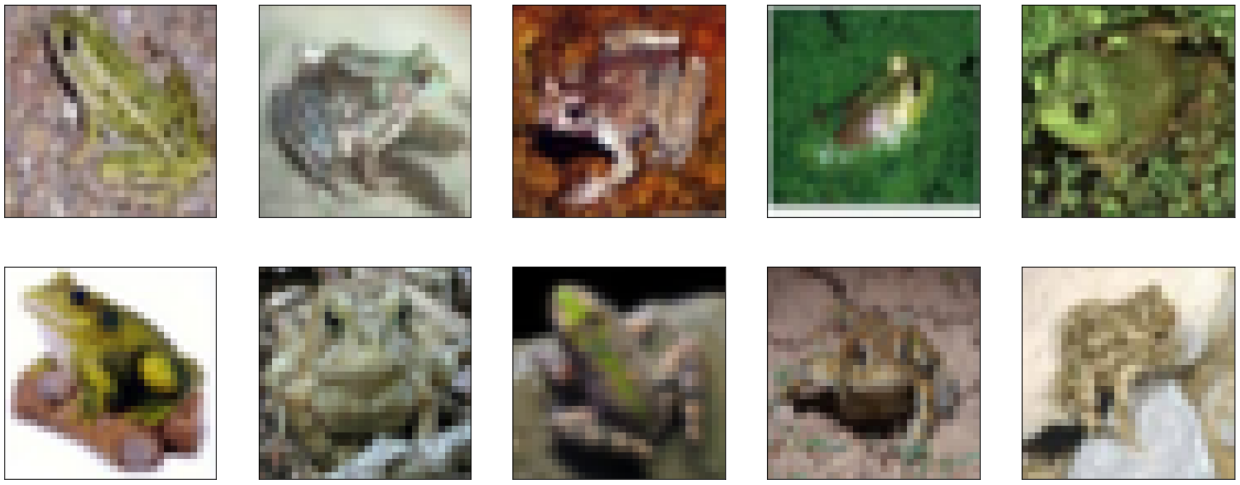
Class 4:



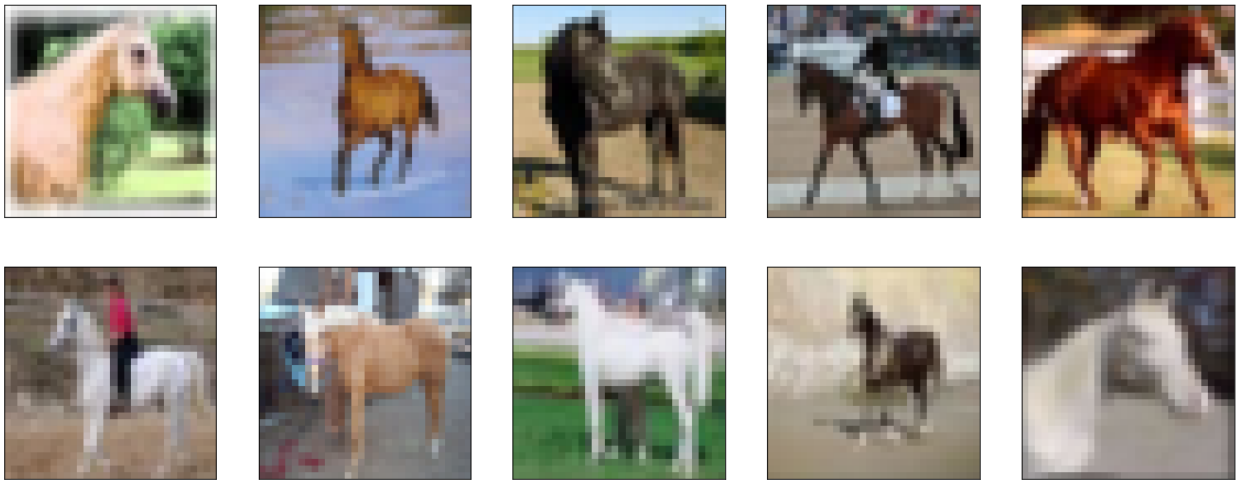
Class 5:



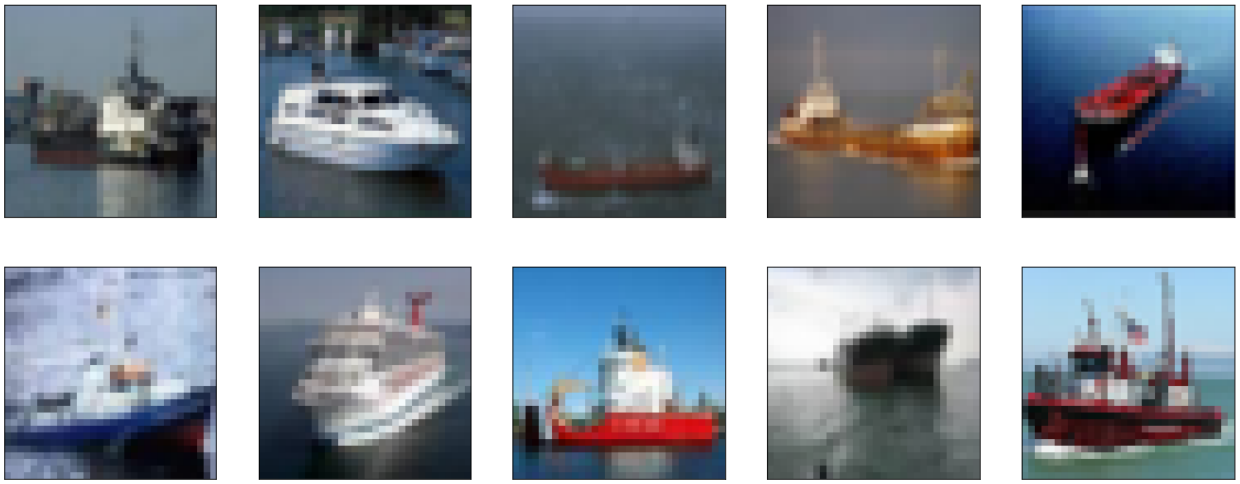
Class 6:



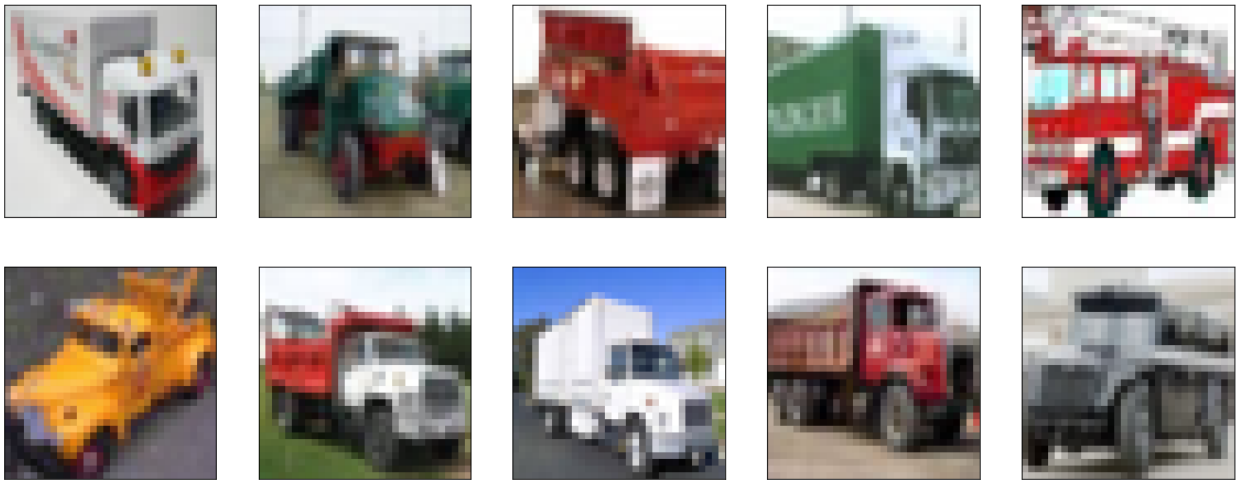
Class 7:



Class 8:



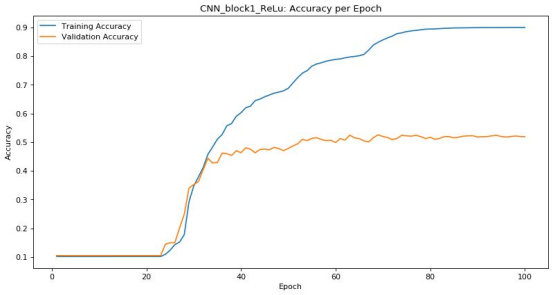
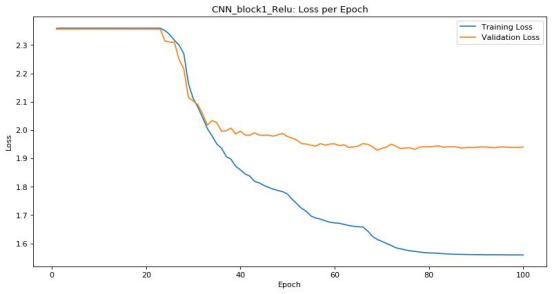
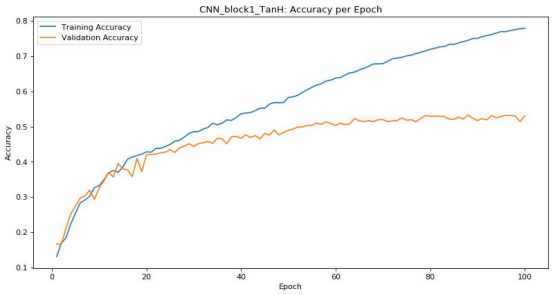
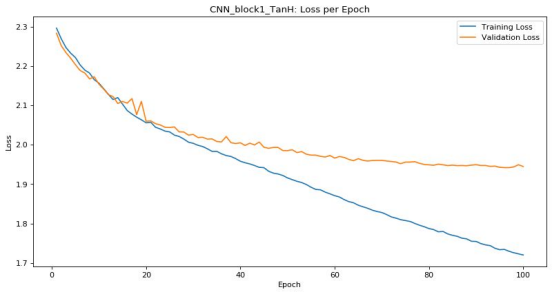
Class 9:

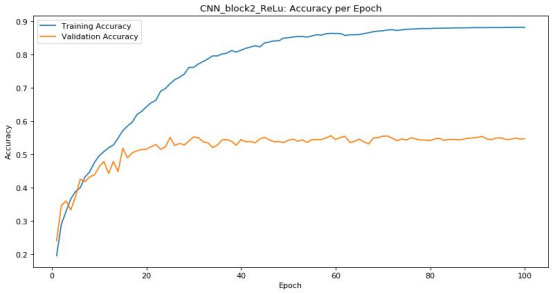
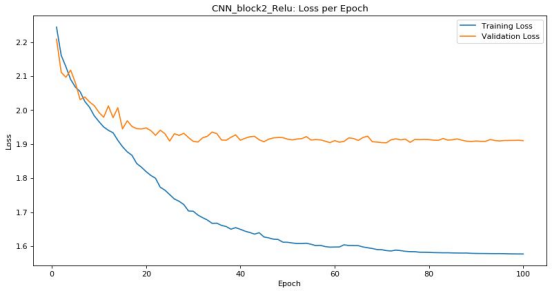
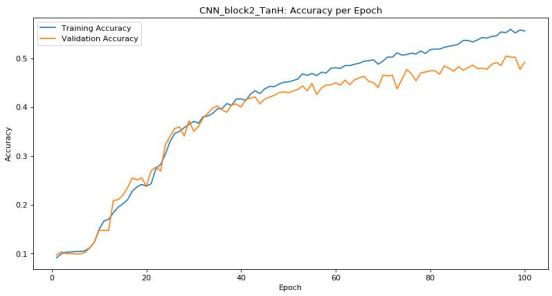
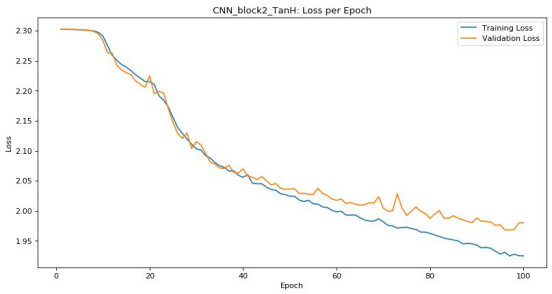


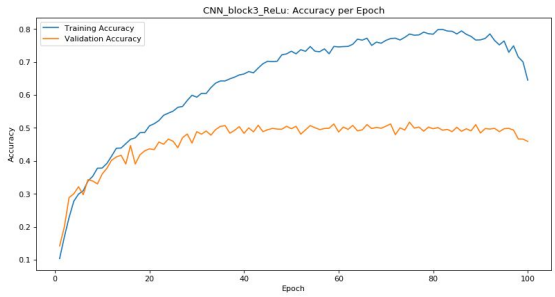
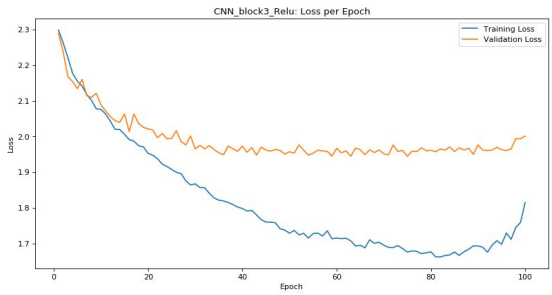
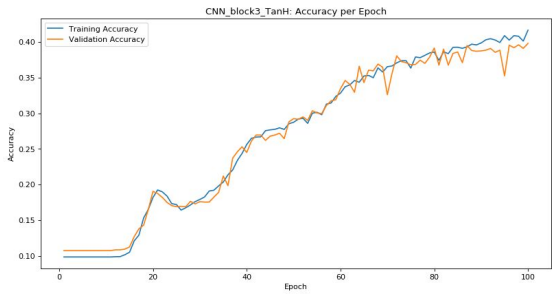
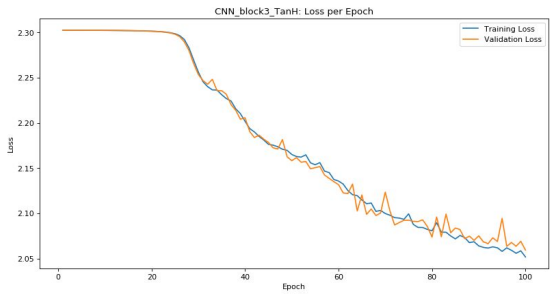
1.2 Analyze the accuracy and loss while adding block 1, block 2 and block 3 (with mentioned non-linearities)

Accuracy after 100 epochs:

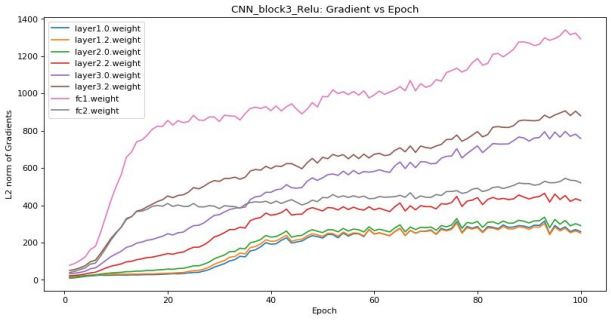
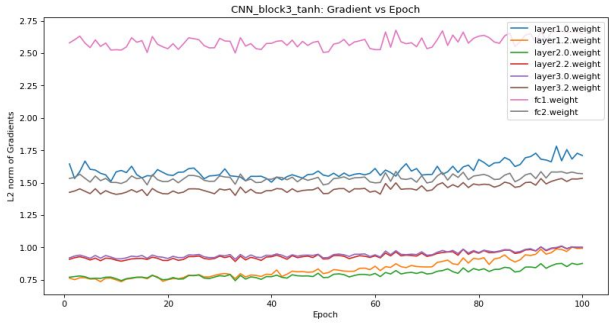
Activation	**ReLu**		**Tanh**	
Blocks	**Training Acc**	**Validation Acc**	**Training Acc**	**Validation Acc**
Block 1	89.98	51.9	77.9	53.05
Block 1 & 2	88.2125	54.75	55.6	49.25
Block 1, 2 & 3	64.48	46	41.66	39.8

Block 1	Accuracy per Epoch	Loss per Epoch
ReLu		
TanH		

Block 2	Accuracy per Epoch	Loss per Epoch
ReLu		
TanH		

Block 3	Accuracy per Epoch	Loss per Epoch
ReLu		
TanH		

Gradient per epoch graph for 3 blocks and 2 FC layers:

ReLu	TanH
	

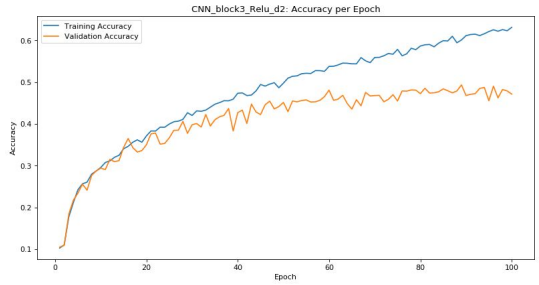
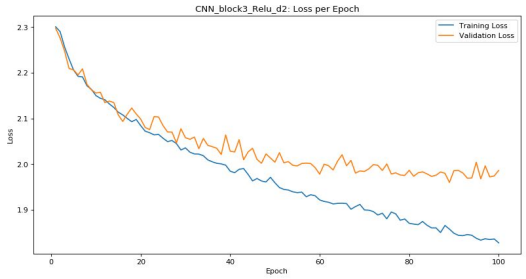
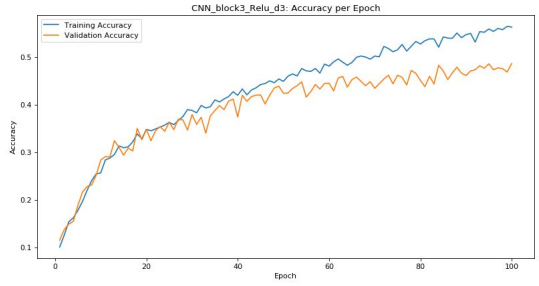
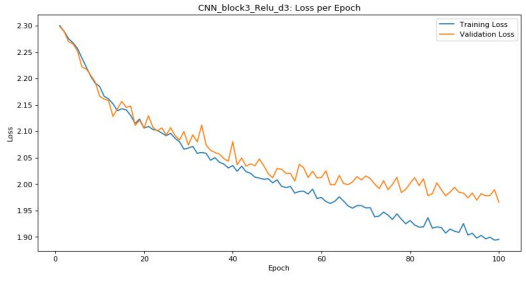
No Pre-processing, Data augmentation or any regularization technique such as dropout or batch norm has been used in any of the above cases which justifies why there is a large gap between training and validation accuracy indicating that the model is overfitting. Accuracies are higher when the number of layers are less as there are less parameters to be trained so model is more simple but also overfitting reduces to some extent and model shows some realistic performance when number of layers are increased. Also ReLU performs better than TanH so further all the analysis is done using ReLU as the activation function.

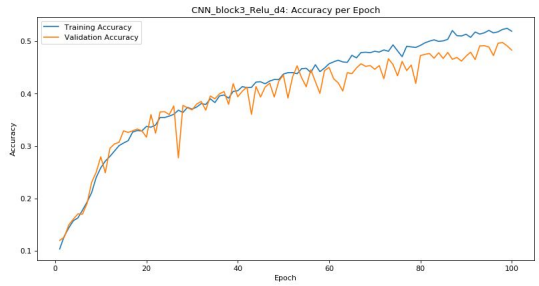
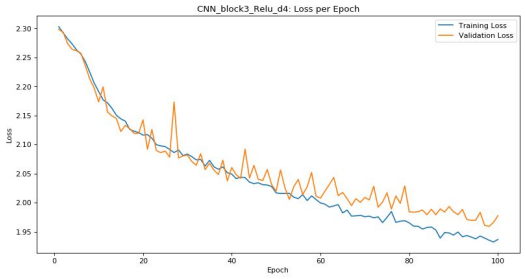
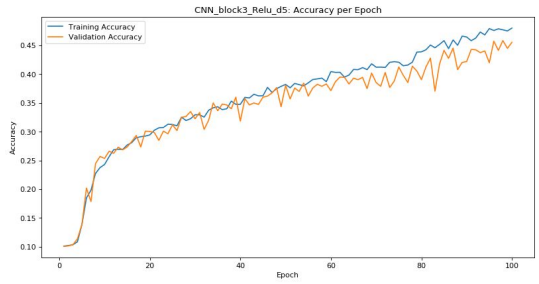
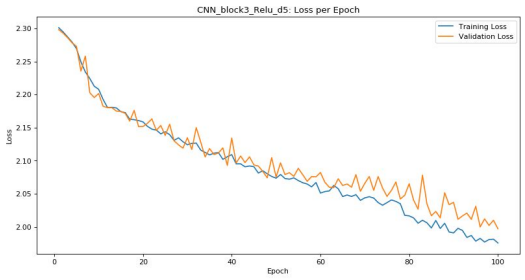
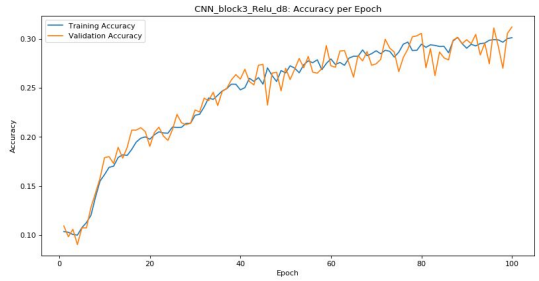
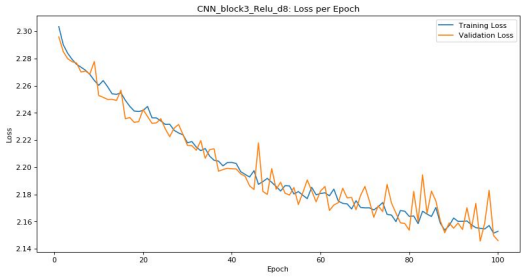
1.3 Analyze the accuracy and loss while changing the dropout probability. (Try at least three Dropout probabilities e.g. [.2, .5, .8])

Dropout applied after Convolution layers in each of the 3 blocks

Accuracy after 100 epochs:

Activation	**ReLu**	
Dropout	**Training Acc**	**Validation Acc**
No	64.48	46
0.2	63.15	47.15
0.3	**56.38**	**48.7**
0.4	**51.9**	**48.3**
0.5	48.025	45.55
0.8	30.11	31.2

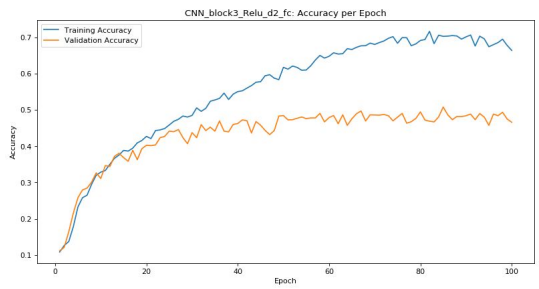
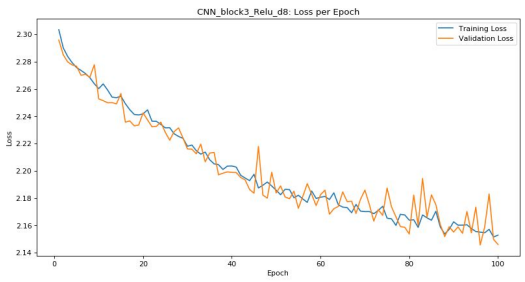
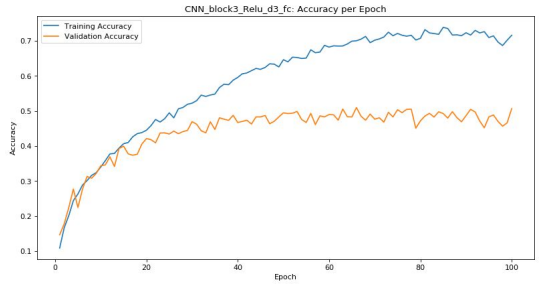
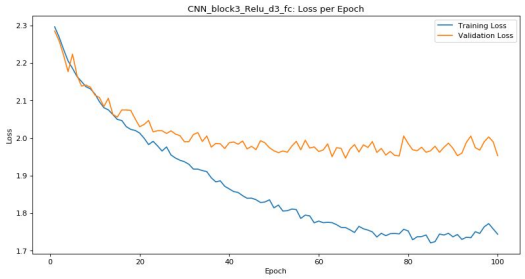
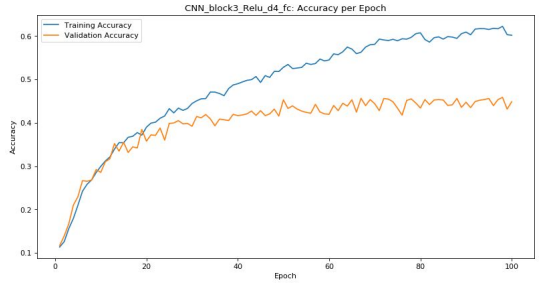
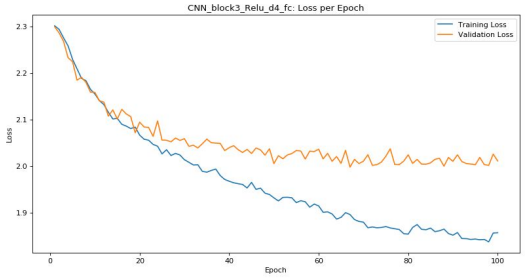
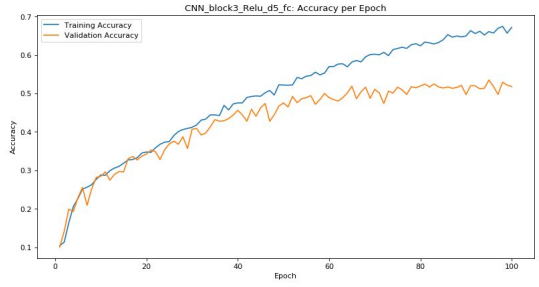
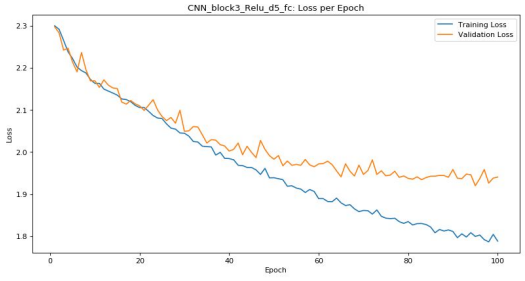
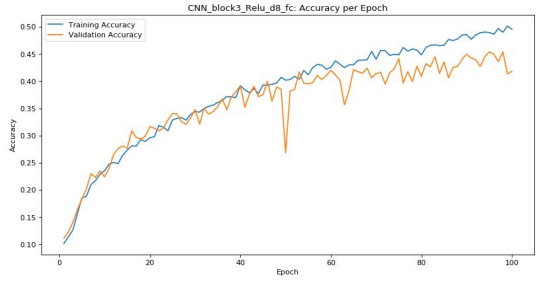
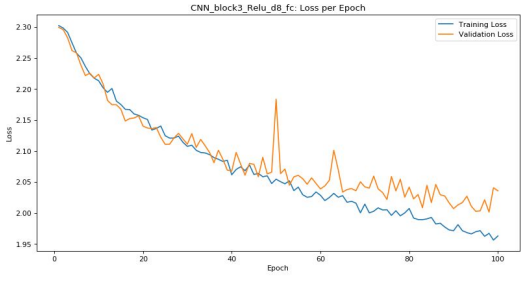
Dropout	**Accuracy per Epoch**	**Loss per Epoch**
0.2		
0.3		

0.4		
0.5		
0.8		

Dropout applied after FC layers

Accuracy after 100 epochs:

Activation	**ReLu**	
Dropout	**Training Acc**	**Validation Acc**
No	64.48	46
0.2	66.41	46.6
0.3	**71.58**	**50.65**
0.4	**60.16**	**44.85**
0.5	67.2	51.8
0.8	49.55	41.8

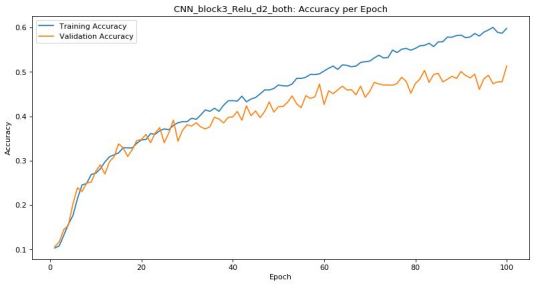
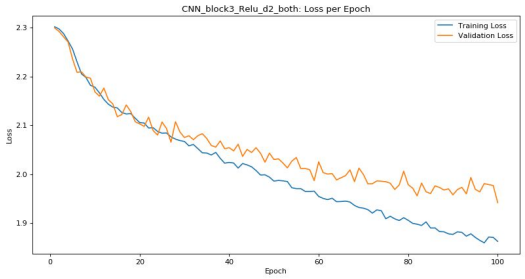
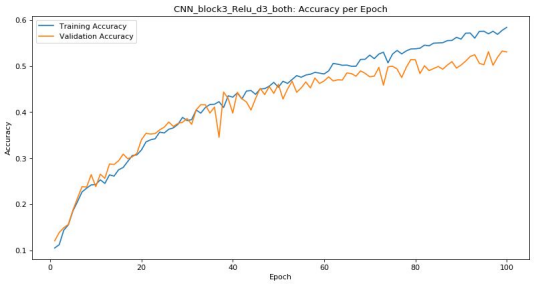
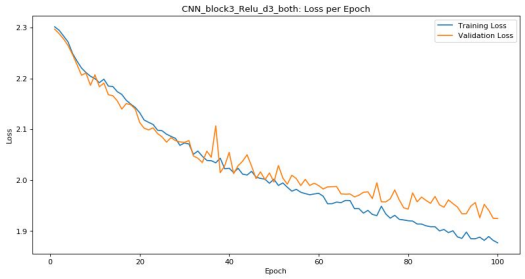
Dropout	Accuracy per Epoch	Loss per Epoch
0.2		
0.3		
0.4		
0.5		
0.8		

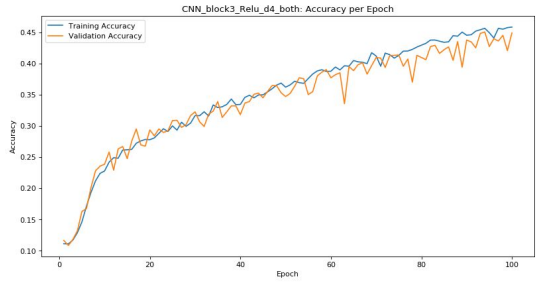
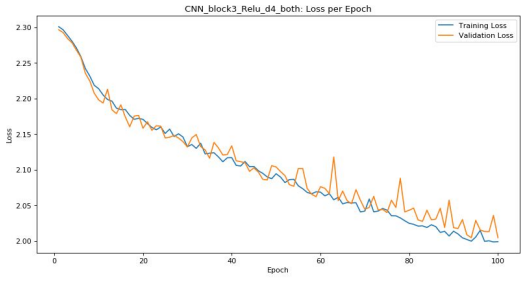
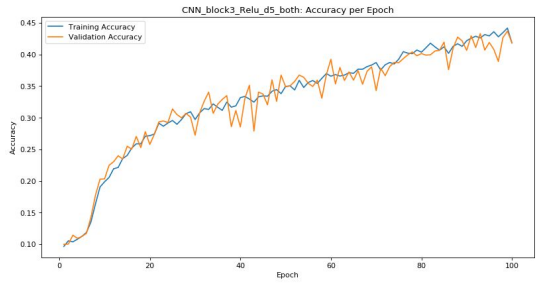
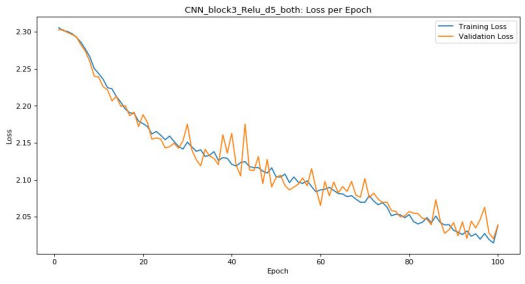
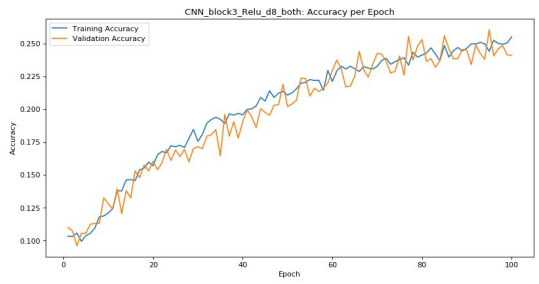
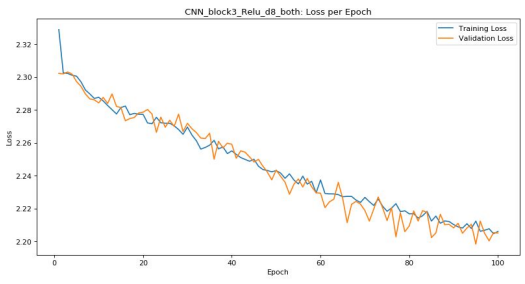
Dropout applied after all layers

Accuracy after 100 epochs:

Activation	ReLu	
Dropout	**Training Acc**	**Validation Acc**
No	64.48	46
0.2	59.825	51.35
0.3	**58.375**	**53.05**
0.4	45.82	44.9
0.5	41.81	41.85
0.8	25.51	24.1

All the above three scenario shows that on increasing the number of probability of dropping the nodes the model initially regularizes well i.e. reduces overfitting but after an extent both training and validation accuracies drop because higher probability like 0.8 means there are very less number of nodes for training and the model is not able to learn well. It can be seen with probability such as 0.3 or 0.4 the model is able to regularize better. Also applying dropout after each and every layer i.e. convolution as well as FC with 0.3 probability gives the best result.

Dropout	Accuracy per Epoch	Loss per Epoch
0.2		
0.3		

0.4		
0.5		
0.8		

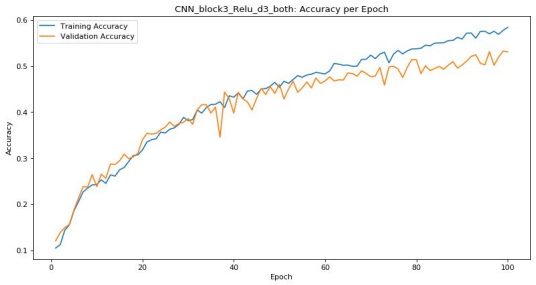
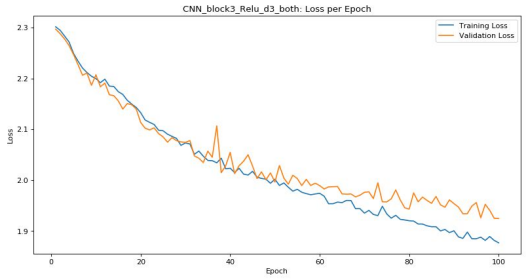
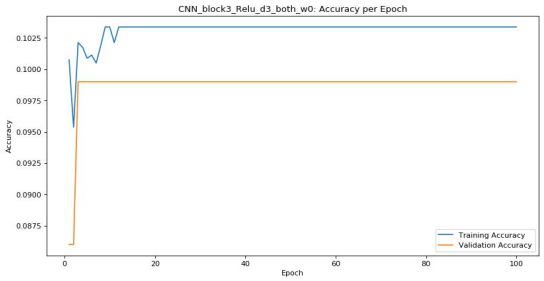
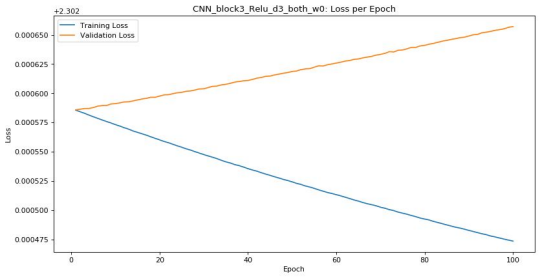
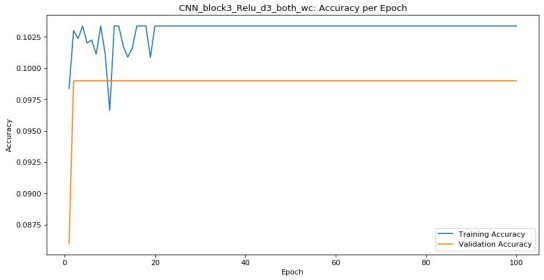
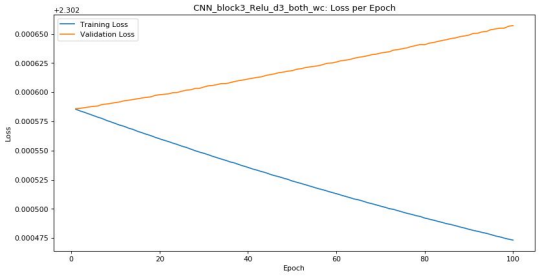
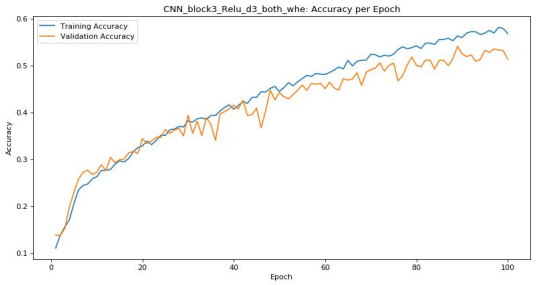
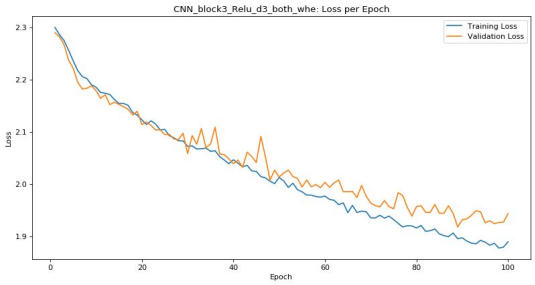
1.4 Initialize the neural network weights by following: Zero initialization, Random Initialization and He initialization. Which initialization approach is best and why?

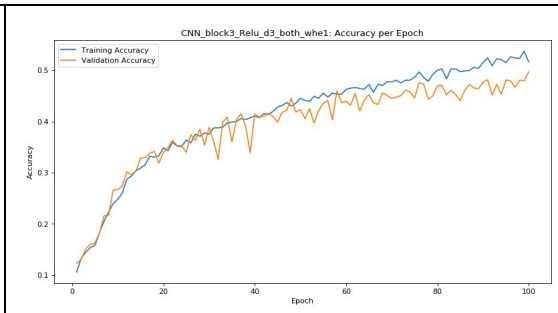
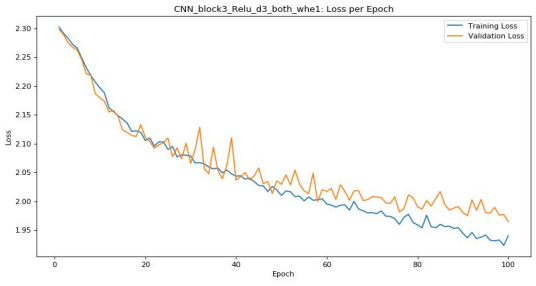
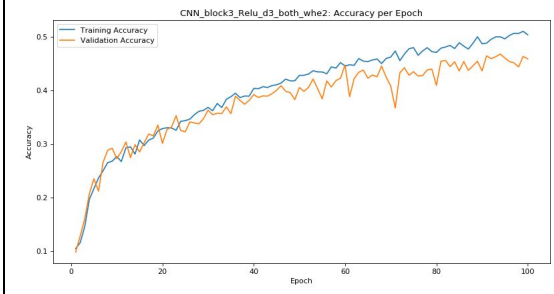
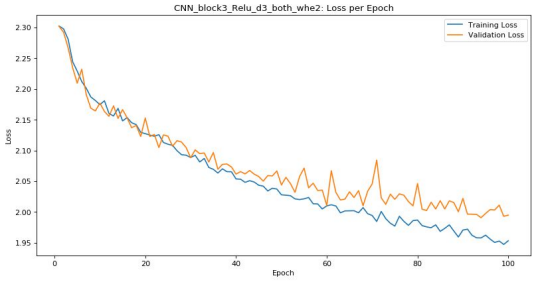
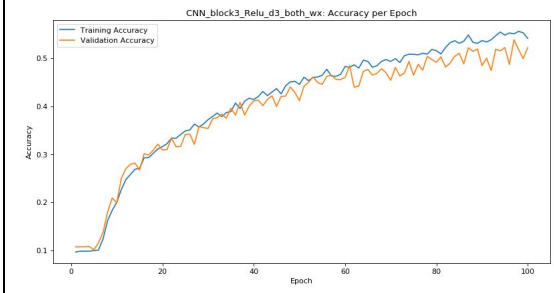
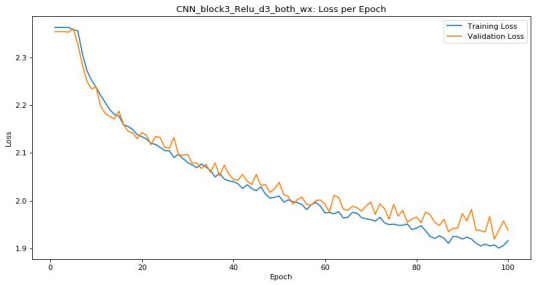
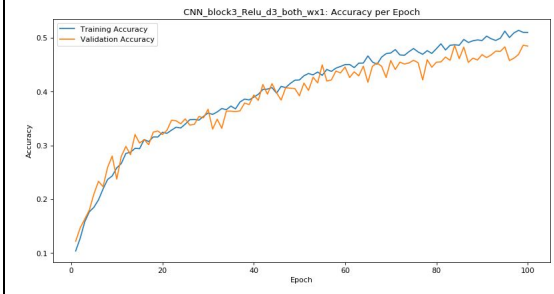
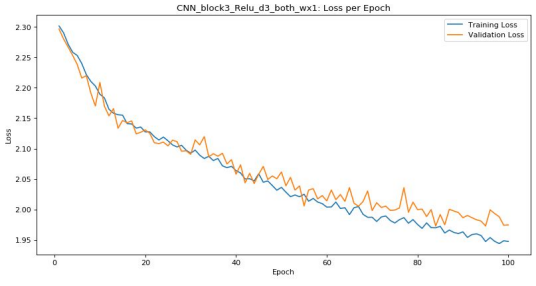
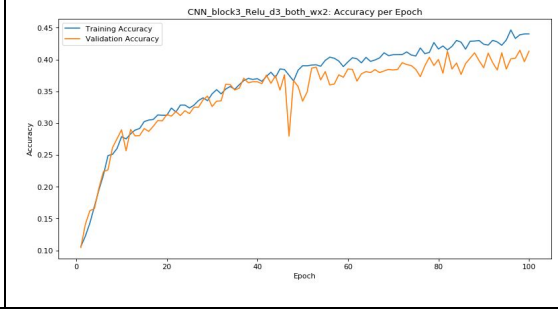
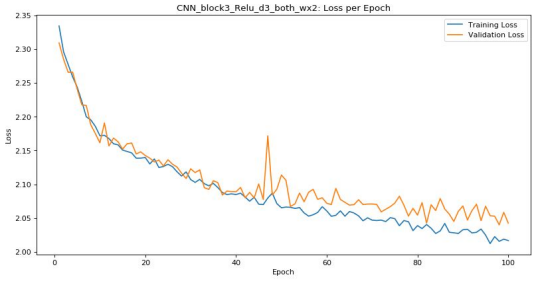
All the different weight initialization techniques are performed while using dropout with probability 0.3 after each & every layer as it's working better as compared to the rest.

Accuracy after 100 epochs:

Initialization	Training Acc	Validation Acc
Random	**58.375**	**53.05**
Zero	10.337	9.9
Constant	10.3375	9.9
He (Conv)	**56.83**	**51.4**
He (FC)	**51.66**	**49.7**
He (Both)	**50.35**	**45.85**
Xavier (Uniform) (Conv)	**54.1**	**52.15**

Xavier (Uniform) (FC)	**51.0125**	**48.5**
Xavier (Uniform) (Both)	44.02	41.3

Initialization	Accuracy per Epoch	Loss per Epoch
Random		
Zero		
Constant		
He (Conv)		

He (FC)		
He (Both)		
Xavier (Conv)		
Xavier (FC)		
Xavier (Both)		

The model is not able to learn incase of zero or constant initialization of weights because the gradients are not back propagated properly which harms the update of weight. By default the

initialization of weights is random from a uniform distribution which shows better performance as compared to the zero and constant initialization. He, Xavier and random initialization perform close enough and He and random outstand as compared to others.

1.5 In the end, report the best accuracy with model architecture and detail analysis of choosing specific hyperparameters and any augmentation or preprocessing if done.

HyperParameters

Variations on learning rate was tried also by changing momentum, stable model was seen when learning rate was set to 0.001 and momentum to 0.9

Optimizers such as SGD and Adam were used.

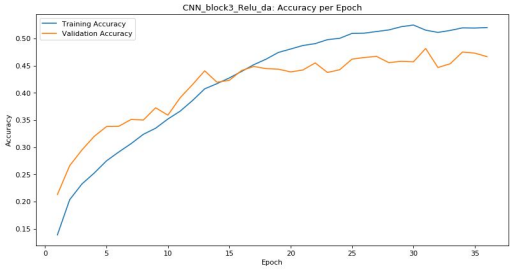
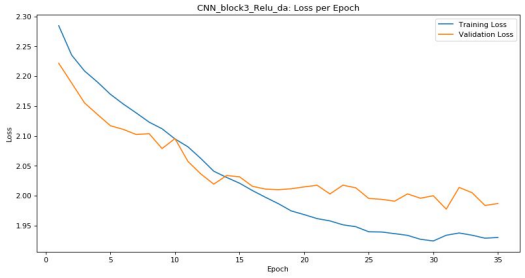
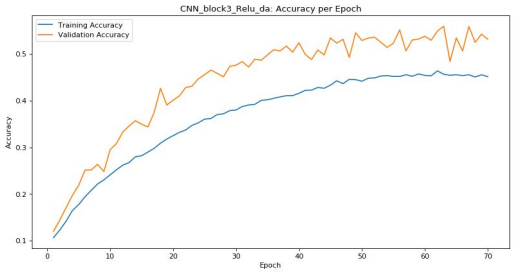
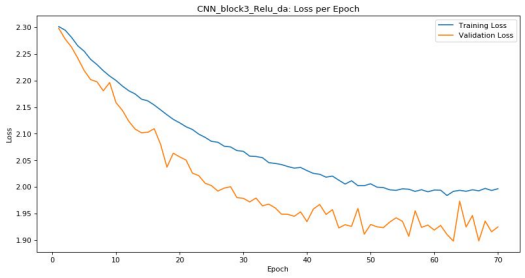
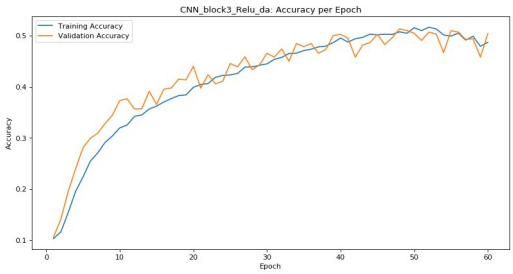
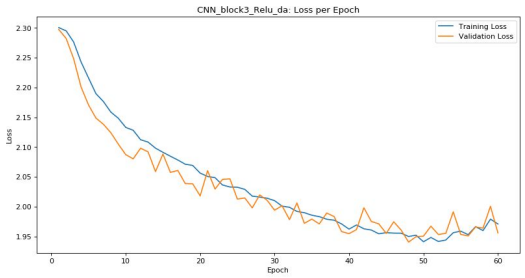
2 FC layers were used one converting mapping the input to that layer into an output of 100 and the second one into output of number of classes i.e. 10 in case of CIFAR10 dataset.

Data Augmentation

Training data is augmented by applying techniques like rotating the image by 45 degree, flipping image upside down (vertical) and right to left (horizontal) also adding random gaussian noise to the image. Hence the training dataset now contains 40000 images instead of 8000 images.

Accuracy after 100 epochs:

Activation	**ReLU**	
Block 1, 2 & 3	**Training Acc**	**Validation Acc**
All Data Augmentation but no Dropout	51.9	47.3
Dropout 0.3 at every layer	45.13	53.15
Data Augmentation without Random Noise	48.68	50.35

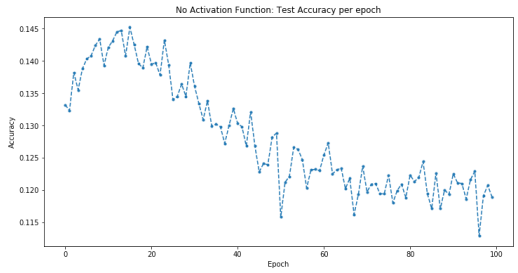
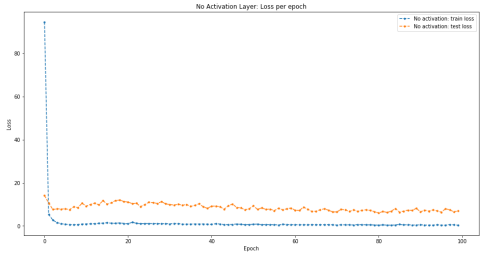
Activation	ReLU	
Block 1, 2 & 3	**Accuracy per Epoch**	**Loss per Epoch**
All Data Augmentation but no Dropout		
Dropout 0.3 at every layer		
Data Augmentation without Random Noise		

1.7 Analyze the results of the best model when all the activation functions are removed. Justify the performance drop.

No Activation Function

Accuracy after 100 epochs:

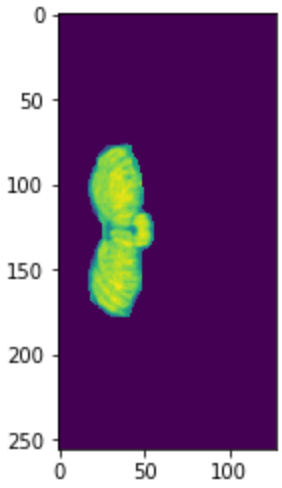
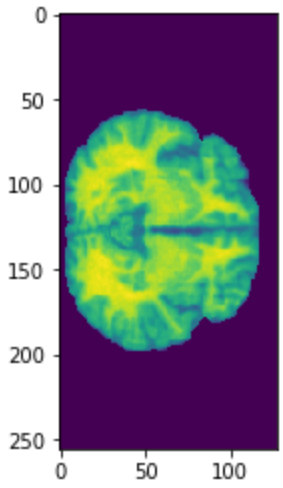
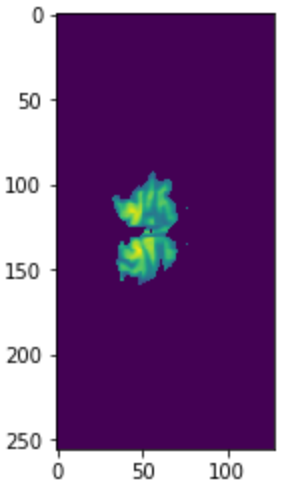
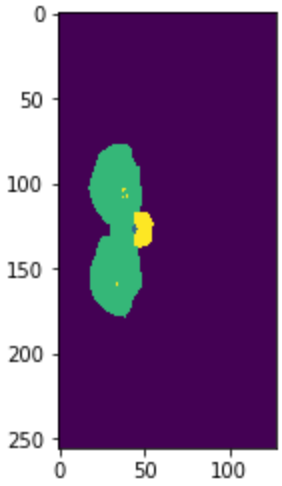
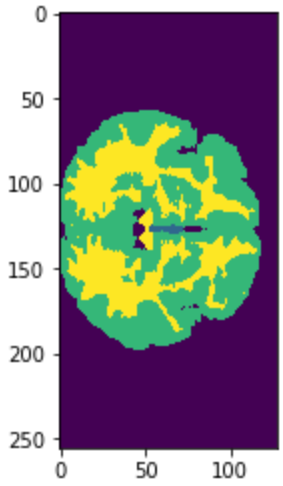
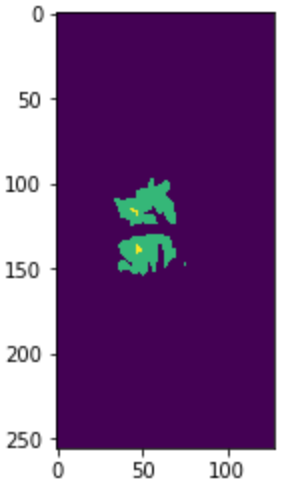
Activation	None	
Block 1, 2 & 3	**Training Acc**	**Validation Acc**
Dropout 0.3 at every layer with Random weight initialization	10.15	10.1

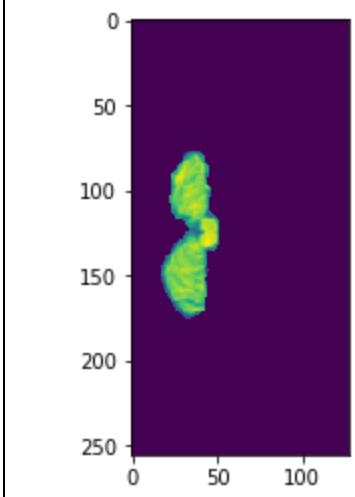
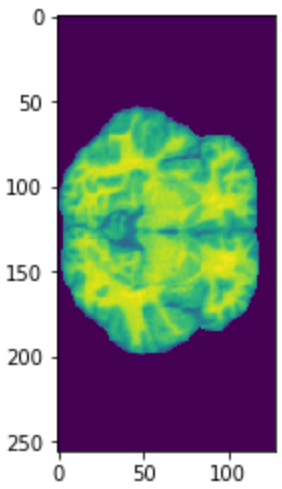
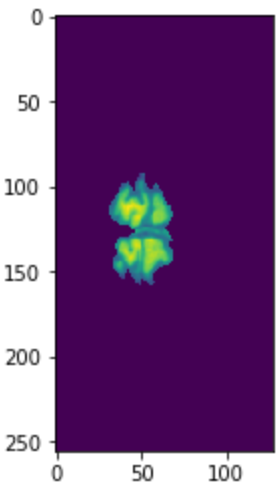
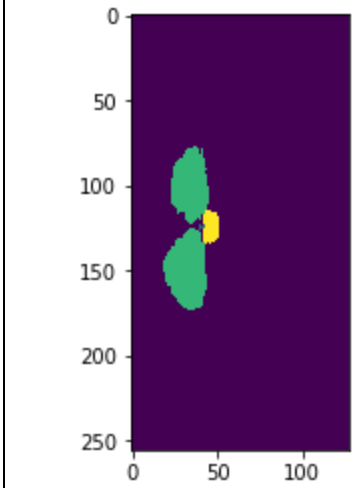
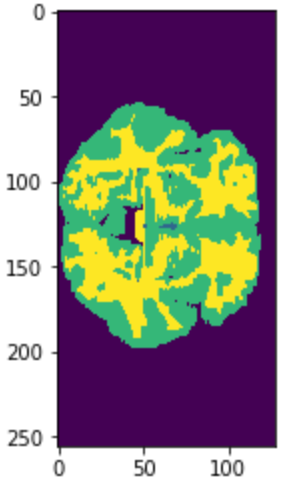
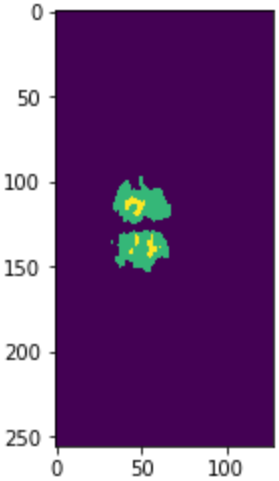
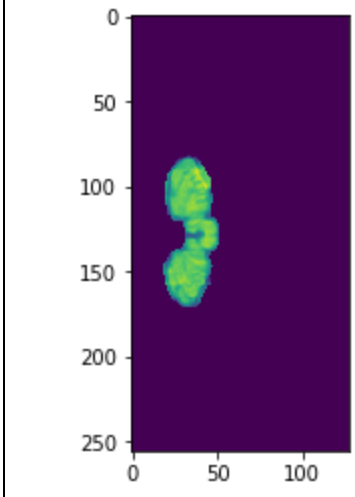
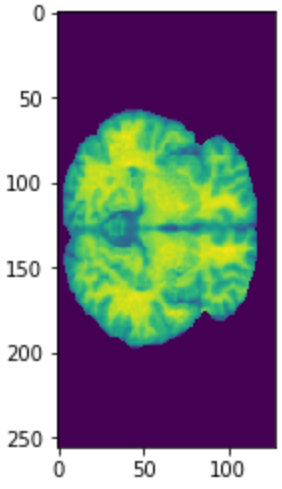
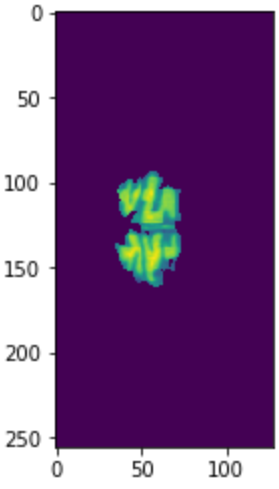
Activation	Accuracy per Epoch	Loss per Epoch
None		

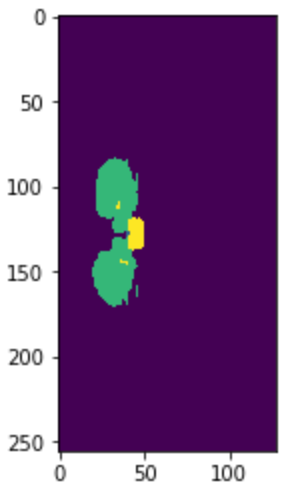
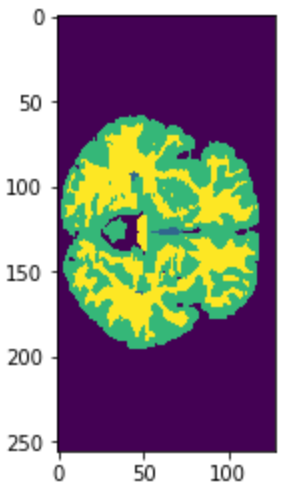
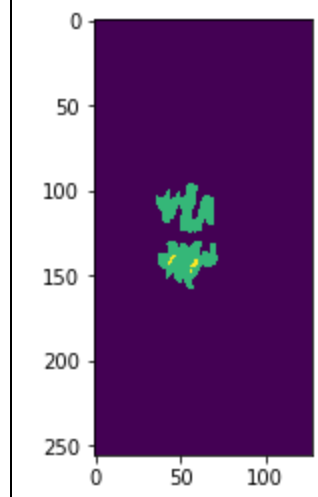
The purpose of the activation function is to introduce **non-linearity** and allows to model a response variable which varies non-linearly with its explanatory variables. Non-linear means that the output cannot be reproduced from a linear combination of the inputs. Without a non-linear activation function in the network, a NN, no matter how many layers it had, would behave just like a single-layer perceptron, because summing these layers would give you just another linear function. Similarly, all layers in either convolutional network or plain multi-layer feed forward network, plays a simple linear transformation over input data. Without nonlinear activation function after each layer, the whole network acts as a simple linear transformation, which does not have so much power for complicated tasks such as digit recognition and image-net classification. Which activation like Sigmoid, Tanh, ReLU, multilayer feed forward neural network or CNN can act as a **universal function approximator**, which can in principle approximate any function over input values. Hence it all comes down to this, we need to apply an Activation function so as to make the network more powerful and add ability to it to learn something complex and complicated form data and represent non-linear complex arbitrary functional mappings between inputs and outputs.

Q2) Automated brain tissue segmentation into white matter (WM), gray matter (GM) & cerebro-spinal fluid (CSF) from magnetic resonance images (MRI) using Fully Convolutional Neural Network (FCN).

2.1 Input and Ground Truth Visualisation

		Top Area	Middle Area	Bottom Area
Volume 1	**Input**			
	Ground Truth			

Volume 2	**Input**			
	Ground Truth			
Volume 3	**Input**			

	Ground Truth			

2.2 Explanation of architectures, preprocessing steps, loss functions and training strategy

2.2.1 Dataset Description and Pre-processing :-

- Training Data - 3 brain volumes (3D data), each of size 256x128x256.
- Validation Data - 1 brain volume of size 256x128x256

To convert it to 2D data (image slices), data is sliced along the third dimension, giving 256 patches each of size 256 x 128 in every volume. Thus, there are 768 slices in training data and 256 slices in validation data.

Ground truth contains 4 different pixel values - 0, 1, 2 and 3, where 0 signifies background and the other 3 classes signify CSF, GM and WM respectively.

Class distribution across all 2D slices in all 3 volumes in training data (**taking all pixels in all slices**) is :-

	Class 0	**Class 1**	**Class 2**	**Class 3**
Volume 1	7294795	14555	725132	354126
Volume 2	7273521	13701	675954	425432
Volume 3	7438497	13784	596162	340165
Total	22006813	42050	1997248	1119723

Not considering background as a separate class, semantic segmentation needs to be performed for 3 classes - labelled 1, 2 and 3.

Considering only these pixels in our dataset, it can be seen that the distribution is highly skewed:-
Class 1: 1%
Class 2: 63.22%
Class 3: 35.4%

To solve this problem of class imbalance:-
Image slices are not fed as it is to the network as it includes the background class and the data as such is highly imbalanced. The image slice is divided into small patches, such that all the patches are somewhat balanced. **This is done by finding overlapping patches throughout the image which does not consist of any 0 valued pixels and the count of pixels of other 3 classes satisfies a minimum threshold.**

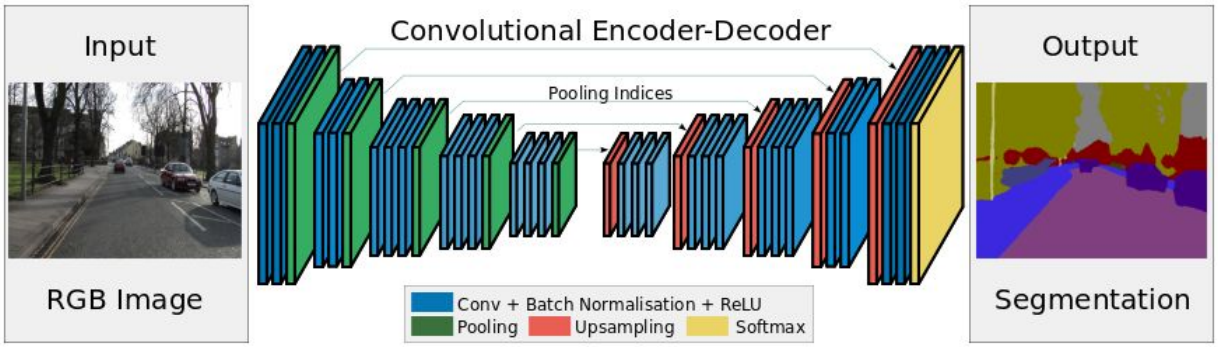
After experimenting with some patch sizes - 16x16, 32x32 and 40x40, the patch size of 16x16 was seen to be giving better performance.

This resulted in a total of **9009** patches, each of 16x16 from the training set, which had the following class distribution:-
Class 1: 787058 pixels = **34.12 %**
Class 2: 672458 pixels = **29.15%**
Class 3: 846788 pixels = **36.71 %**
This ensures that the network is getting a balanced dataset.

Since the network has to take as input small patches, the validation and test set also need to be divided into patches. This is done by simply splitting the entire image into 128 non-overlapping patches using stride 16, giving approximately 6000 patches.

2.2.2 Architectures

(1) Baseline - SegNet
(https://meetshah1995.github.io/semantic-segmentation/deep-learning/pytorch/visdom/2017/06/01/semantic-segmentation-over-the-years.html)



The architecture above was used as the baseline, with few modifications. Pooling was done only after 1st, 3rd and 5th convolution blocks and **unpooling** operation was used

for upsampling. In the end, a softmax layer was added which gave 3 probabilities for each pixel.

(2) Transposed Convolution

Instead of unpooling in the above architecture, convTranspose2D was used to achieve interpolation.

(3) Strided Convolution

In place of max-pooling for downsampling, convolution layers with a stride of 2 were used.

(4) Skip Connections

2 skip connections were introduced between the encoder and decoder part of the network, thus concatenating the feature maps.'

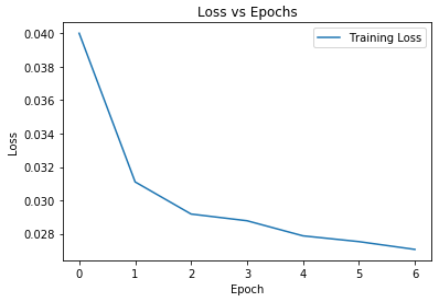
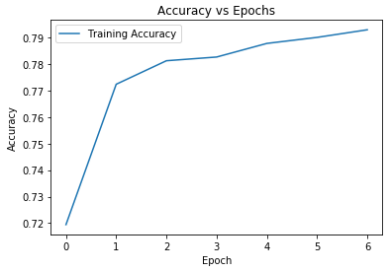
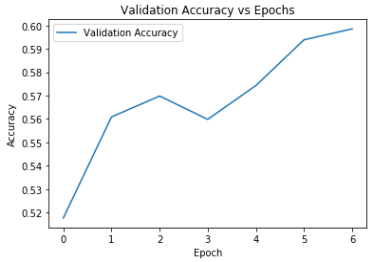
2.2.3 Training Strategy and Hyperparameters

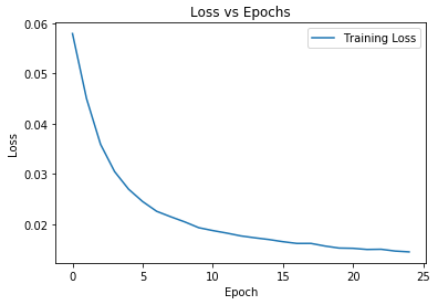
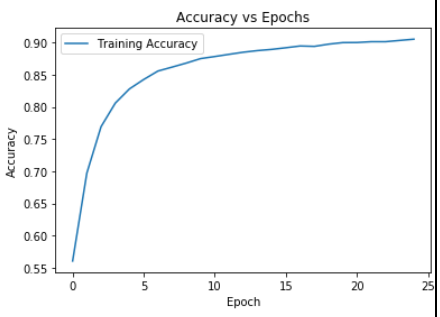
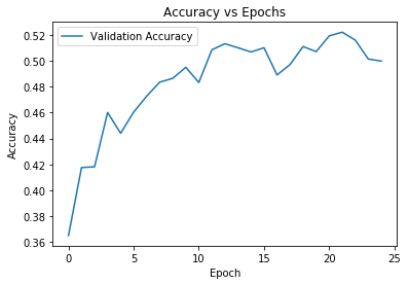
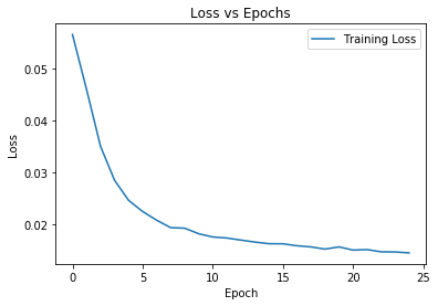
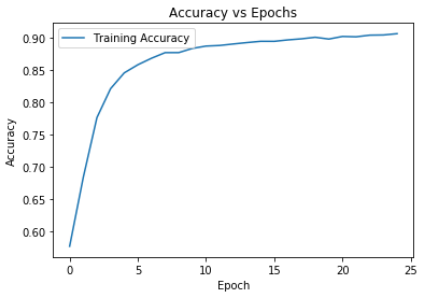
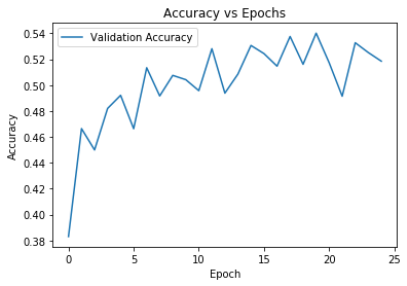
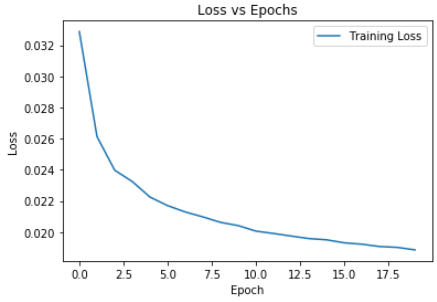
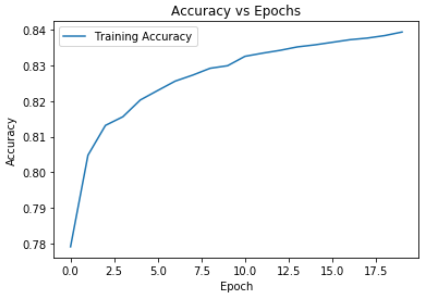
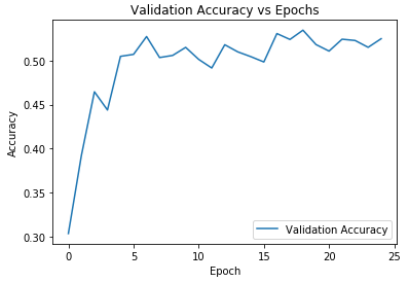
- Input Patch shape - (1, 16, 16)
- Output Shape - (3, 16, 16)
- Batch size - 16
- Loss function - Cross Entropy Loss
- Optimiser - Adam optimiser
- Kernel size across all conv layers - 3x3
- Learning Rate = 0.001

Training Loss, Training Accuracy and validation accuracy were computed in each epoch and the results for the same are shown below.

2.3 Analysis of Performance on different models

Convergence Plots (During Training) :-

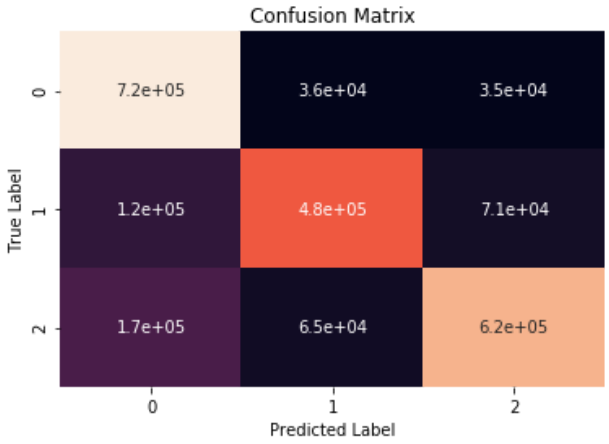
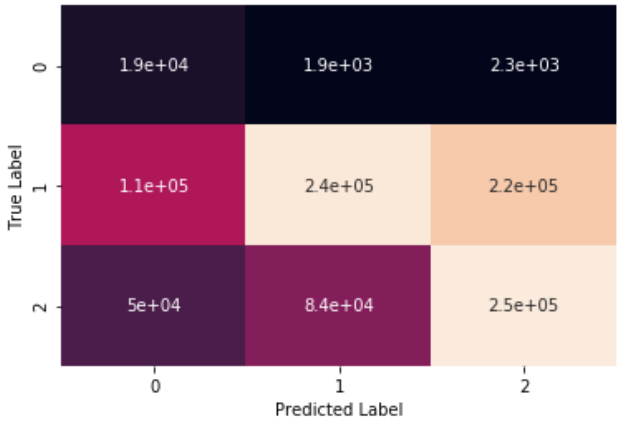
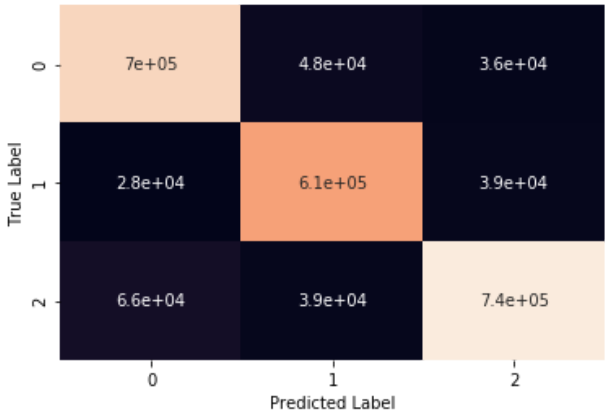
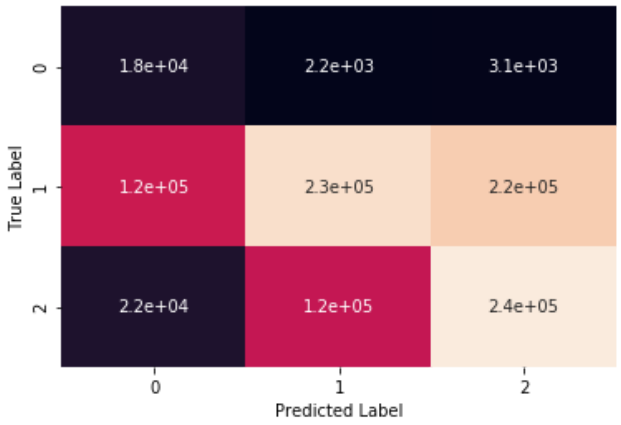
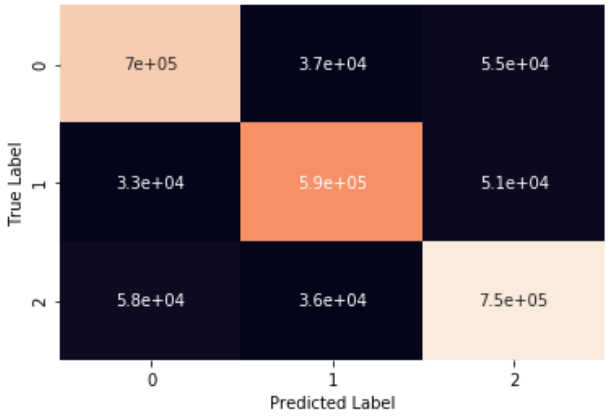
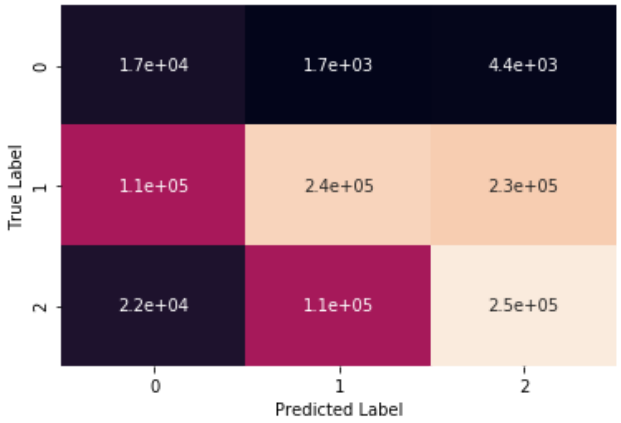
	Training Loss	Training Accuracy	Validation Accuracy
Baseline (Model 1)			

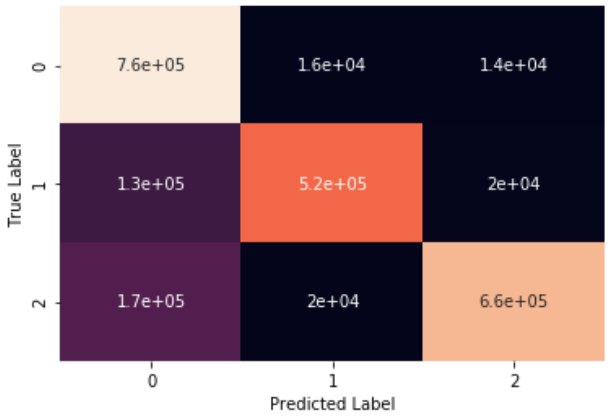
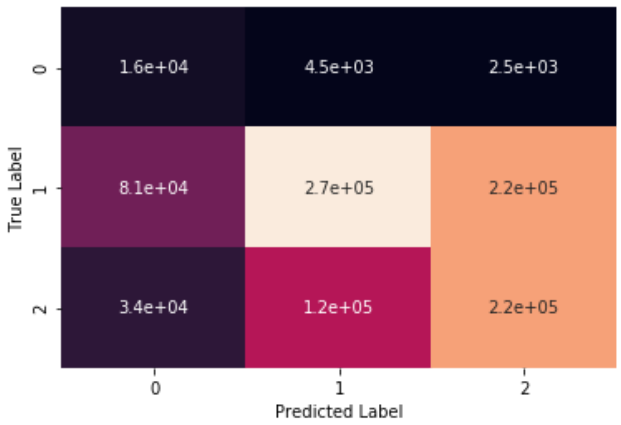
Transposed Convolution (Model 2)			
Strided Convolution (Model 3)			
Skip Connection (Model 4)			

Quantitative analysis - dice ratio, confusion matrix and accuracy (per pixel accuracy) :-

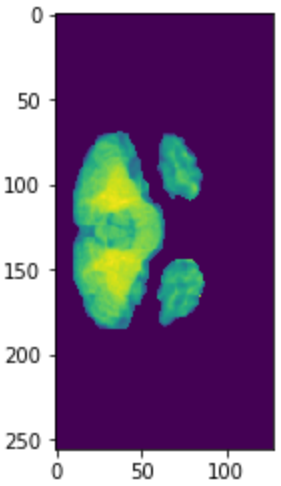
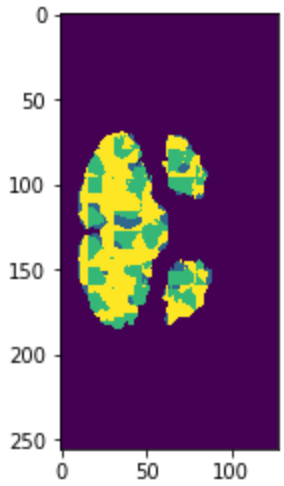
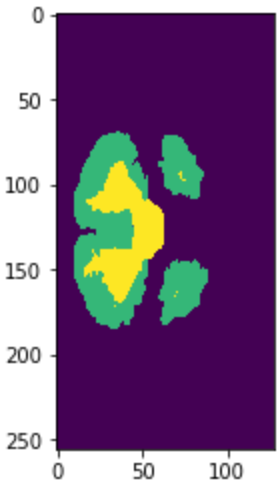
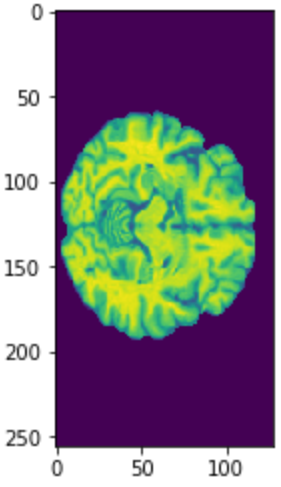
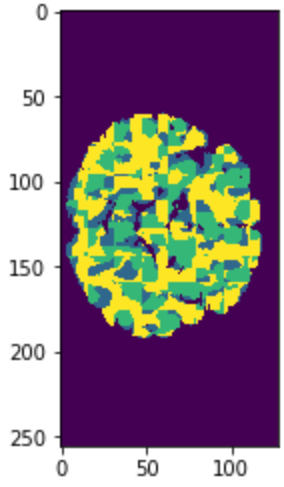
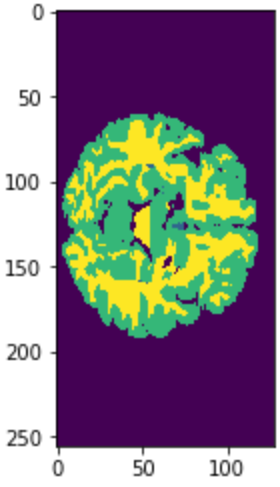
Metric →		**DiceRatio**				**Accuracy**			
Models ↓		**Class 1**	**Class 2**	**Class 3**	**Avg.**	**Class 1**	**Class 2**	**Class 3**	**Avg.**
Baseline (Model 1)	Training	0.80	0.765	0.785	0.783	0.91	0.71	0.727	0.78
	Validation	0.21	0.538	0.576	0.435	0.821	0.42	0.64	0.51
Transposed Convolution (Model 2)	Training	0.887	0.887	0.892	0.892	0.89	0.90	0.87	0.889
	Validation	0.20	0.50	0.57	0.422	0.77	0.40	0.63	0.50
Strided Convolution (Model 3)	Training	0.883	0.881	0.882	0.882	0.88	0.87	0.88	0.88
	Validation	0.202	0.51	0.58	0.43	0.735	0.41	0.66	0.518

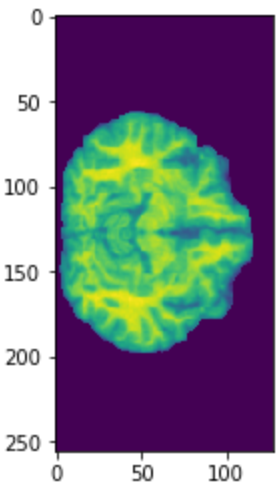
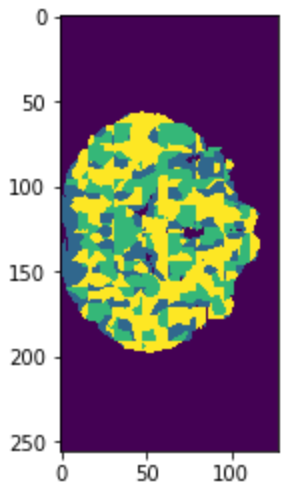
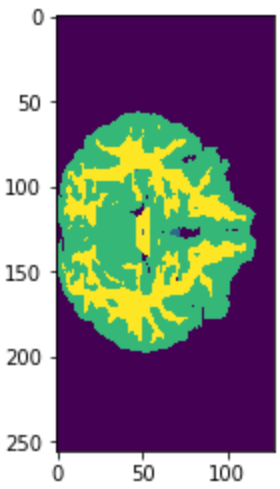
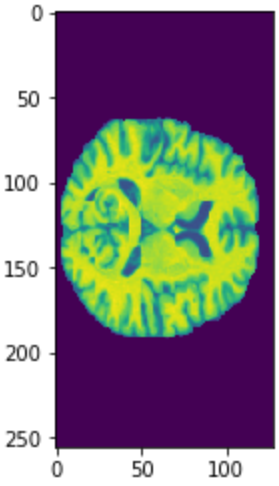
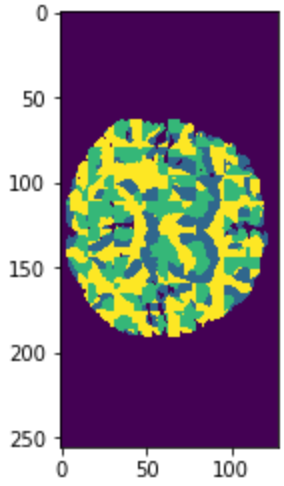
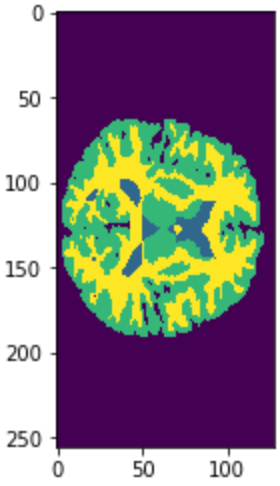
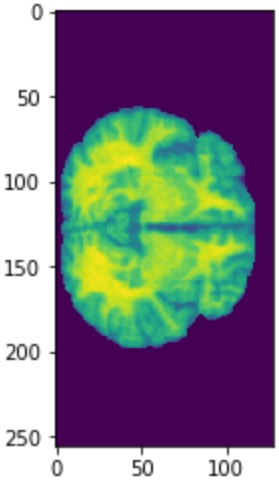
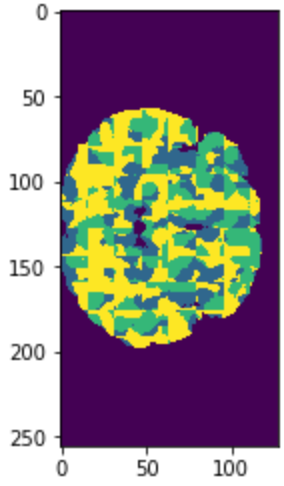
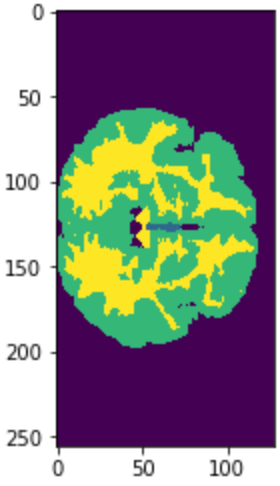
Skip Connections (Model 4)	Training	0.91	0.93	**0.92**	0.92	0.90	0.92	0.89	**0.912**
	Validation	0.4	0.58	**0.66**	0.55	0.805	0.49	0.85	**0.61**

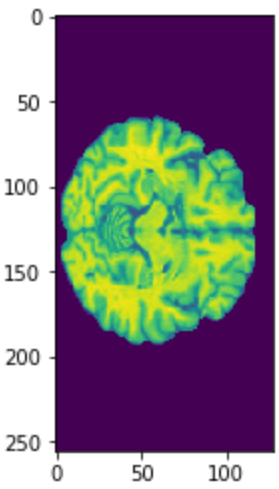
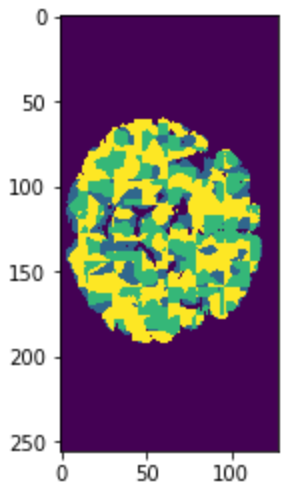
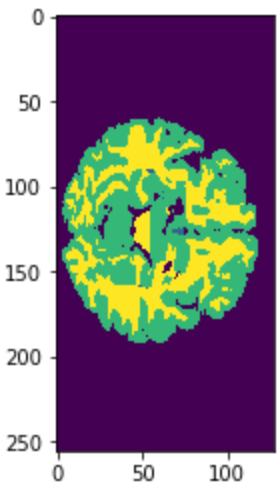
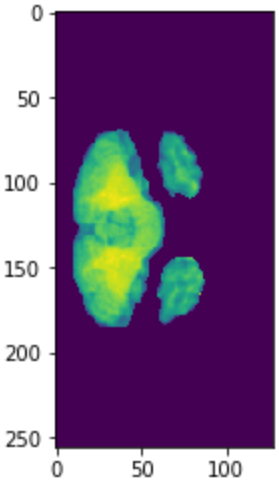
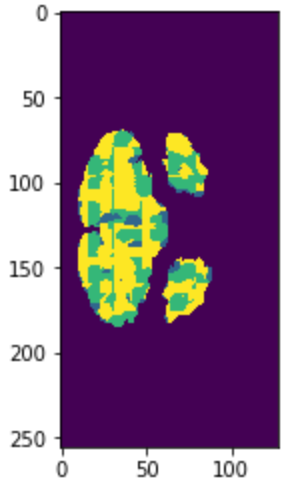
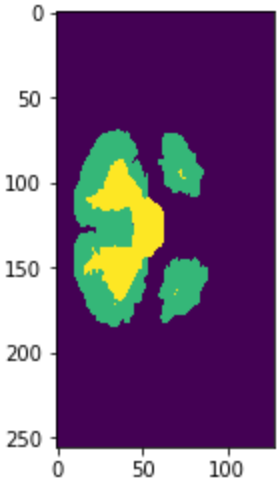
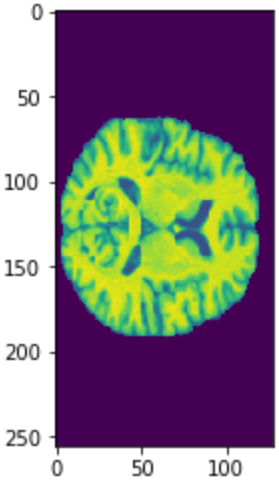
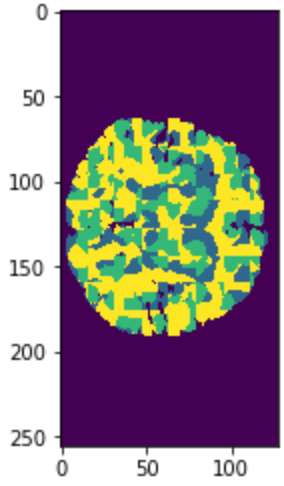
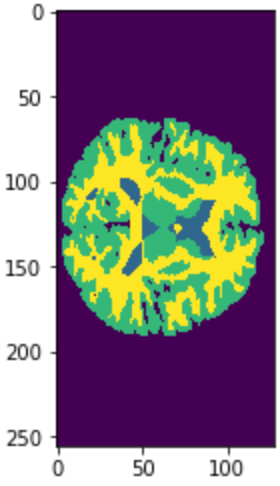
	Training Confusion Matrix	Validation Confusion Matrix
Baseline (Model 1)		
Transposed Convolution (Model 2)		
Strided Convolution (Model 3)		

Skip Connections (Model 4)		

Qualitative Slice Visualisation :-

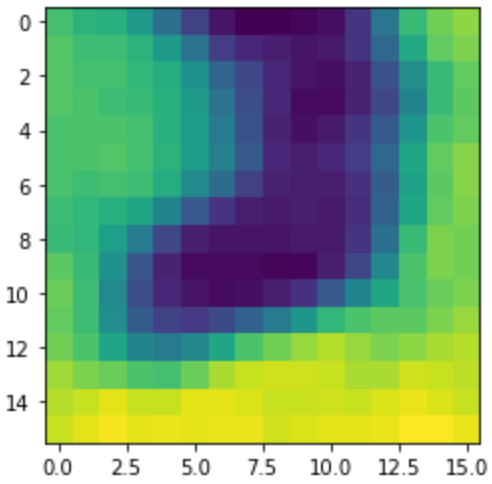
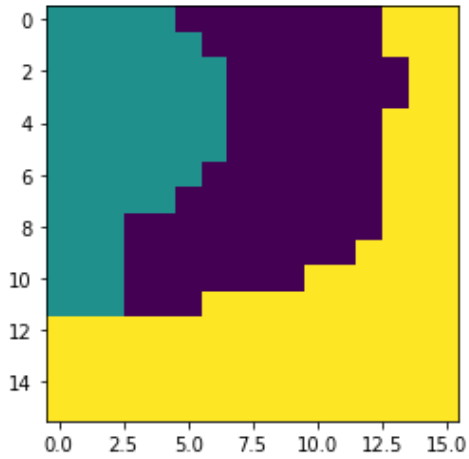
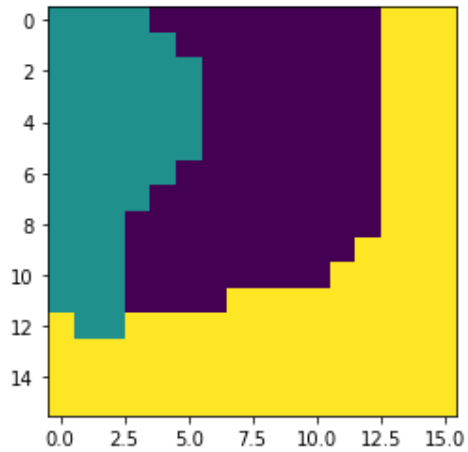
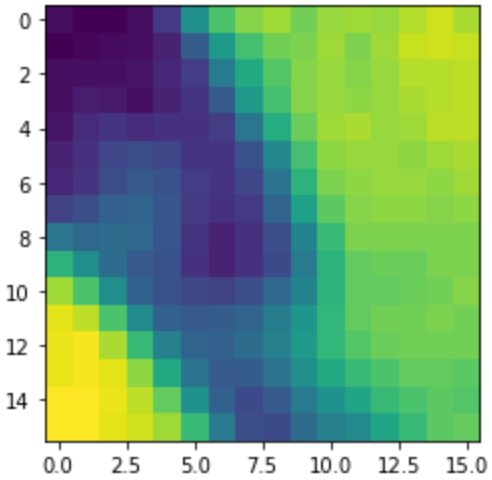
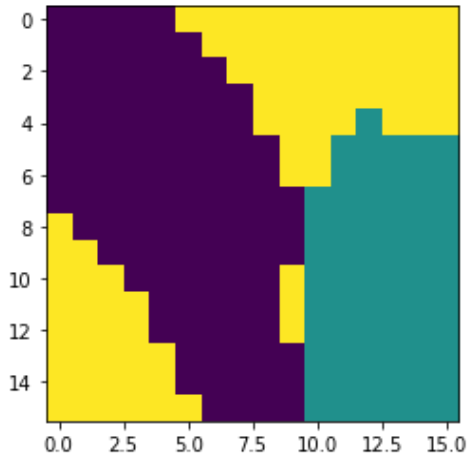
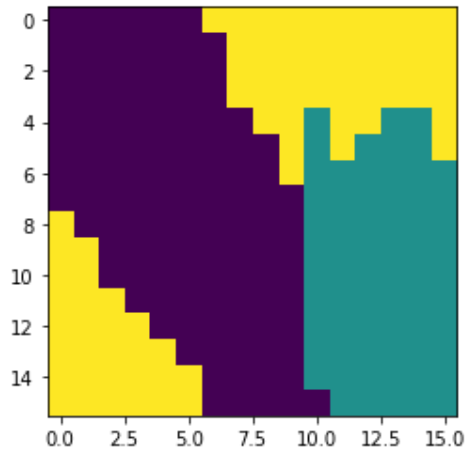
		Input Image Slice	Predicted Image Slice	Actual Image Slice
Baseline (Model 1)	Training			
	Validation			

Transposed Convolution (Model 2)	Training			
	Validation			
Strided Convolution (Model 3)	Training			

	Validation			
Skip Connection (Model 4)	Training			
	Validation			

Qualitative Patch Visualisation (From Training) :-

For the best performing model, which uses skip connections.

Patch from Input Image	Predicted Patch	Actual Patch
		
		

From the above qualitative and quantitative results of the various models, it can be observed:-

- The best model was obtained by incorporating skip connections in the architecture. Skip connections ensure that information that is lost while downsampling, is captured in the upsampling layers, and thus the entire information together helps in creating back image.
- The model with strided convolution is performing slightly better than that which uses max-pooling. This might be because strided convolution is a learnable operation whereas max-pooling is not and thus the former will learn the feature maps better.
- The dice ratio does not change much across the first 3 architectures as the region of impact is small and thus during backpropagation, the changes in loss might be small, causing little variation in dice ratio.
- The training accuracy is always quite higher than validation accuracy because the kind of patches in both training and validation is quite different, they are balanced on training while not at all so in validation.
- Convergence can be seen approximately after 20-25 epochs.

Q3) Pre-trained retinanet model for the MS Coco Dataset

- **Dataset Details:**

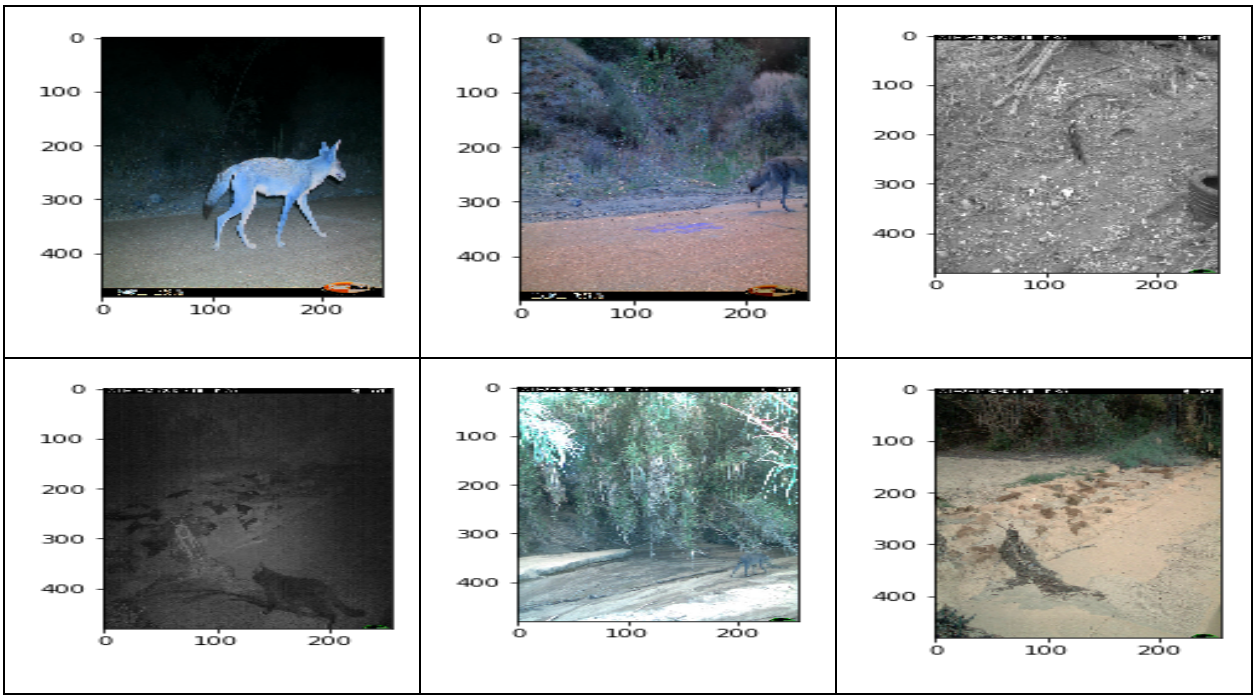
  Dataset name : MS Coco Dataset
  Number of classes : 8
  Number of data points : 1969
  Link:https://drive.google.com/drive/folders/1PlpmUGCYJvvVHQh7CL1JjQXinOp8EUo-?usp=sharing

  Preprocessing Steps:

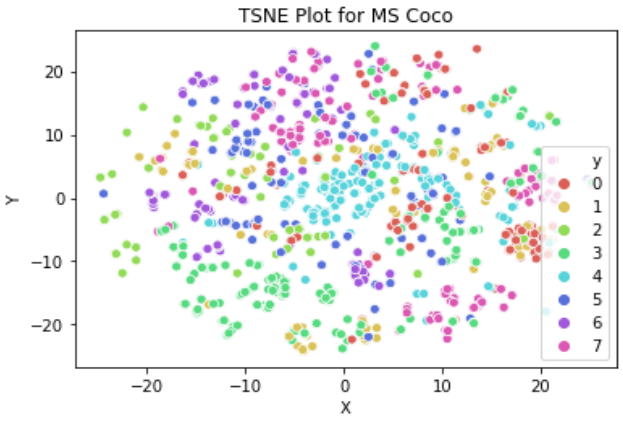
  1. We made the csv file for train and validation dataset.
  2. Then, we used the retinanet model by resizing the images.
  3. We make split of valid : train :: 469:1500

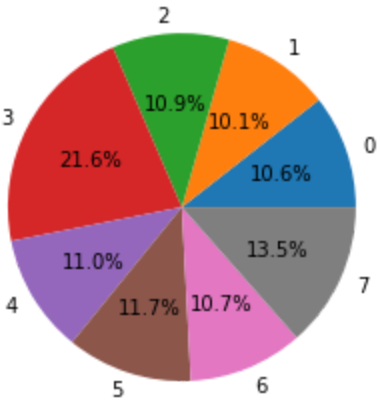
- **Analysis of dataset:**

  Some data samples



  Dataset Visualization:

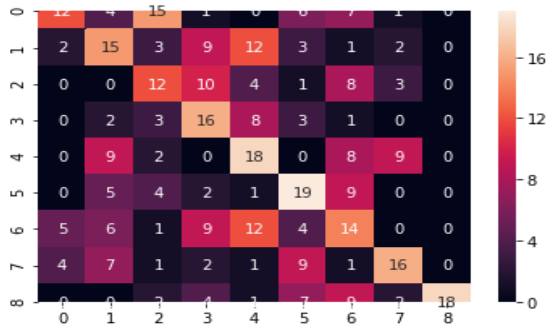
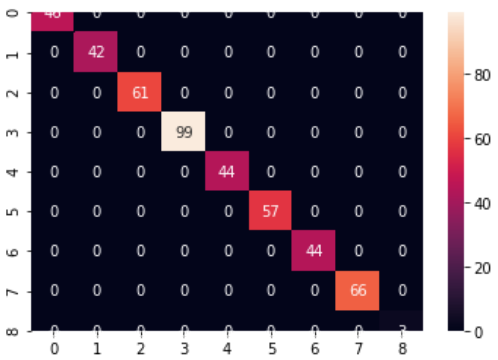




Observations:

1. The data is not distributed uniformly among the classes.
2. The data is not separated linearly as shown in TSNE plot

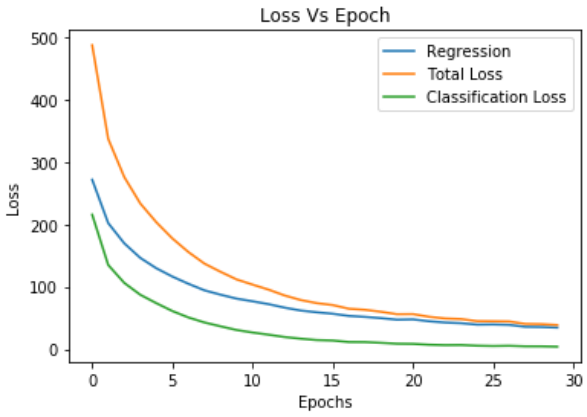
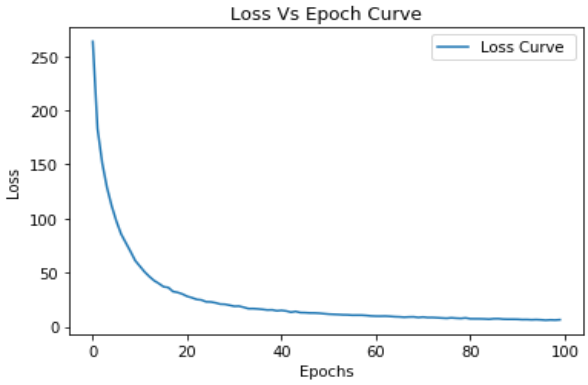
3.1 Without Fine tuning :

Class wise Performance MAP without finetune	Class wise performance MAP with fine tune
	
Coyote: 0.231 Bobcat: 0.22 Bird: 0.24 Raccoon: 0.34 Car: 0.35 Rabbit: 0.35 Squirrel: 0.421 Cat: 0.347	Coyote : 1.0 Bobcat : 0.951 Bird : 0.968 Raccoon : 0.99 Car : 1.0 Rabbit : 1.0 Squirrel : 1.0 Cat : 0.971

Observation:

1. MAP is used for class wise performance.
2. After fine tuning, model is performing better than before.

3.3 Loss Observation Table:

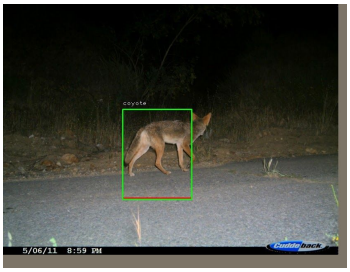
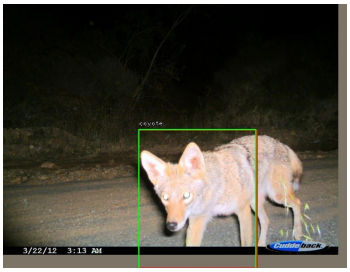
Parameters & Configurations: **Epoch : 100** **Batch size :** 4 **learning_rate :** 0.001 **Momentum :** 0.9	
With fine tuning	
	

Observations:

1. Loss decreases with increase with training epochs.
2. So, model is learning after every epoch as shown in curve.
3. Since regression curve is always higher than classification loss since regression is more magnitude wise task.

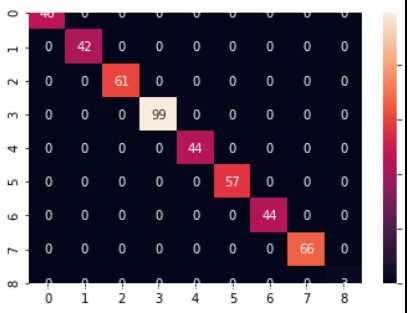
3.4 Observation for Random samples table:

Class	Figure 1	Figure 2	Figure 3
Bird			

Bobcat			
Car			
Cat			
Coyote			
Rabbit			

Raccoon			
Squirrel			

Observation Table:

Observation for Situations	Confusion Matrix	IOU	Class wise Analysis (MAP)
After fine tuning with pre-trained model		**93.1**	Coyote : 1.0 Bobcat : 0.951 Bird : 0.968 Raccoon : 0.99 Car : 1.0 Rabbit : 1.0 Squirrel : 1.0 Cat : 0.971

Observations:

1. After fine tuning, the model performed better than before.
2. Regression boxes are very close to predicted one.
3. IOU is very near to 1 implies model is better.

References:

1. https://github.com/yhenon/pytorch-retinanet/blob/master/train.py
2. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c
3. https://github.com/meetshah1995/pytorch-semseg

Links to Models:-

https://drive.google.com/open?id=1atKtv6RRE5ZVwuXCAG66ugbi3be0rHyx
https://drive.google.com/open?id=1P_d_B66DZHo_L6cq4PATya_4vtkFE3HF