HLCV EXERCISE 3
CONVOLUTIONAL NEURAL NETWORKS

Devikalyan Das
MSc. in Visual Computing
Universität des Saarlandes
Matriculation Number : 7007352
deda00002@stud.uni-saarland.de

Nobel Jacob Varghese
MSc. in Data Science and Artificial Intelligence
Universität des Saarlandes
Matriculation Number : 7002401
noja00001@stud.uni-saarland.de

Shashank Agarwal
MSc. in Embedded Systems
Universität des Saarlandes
Matriculation Number : 7009562
shag00001@stud.uni-saarland.de

1 Question 1

Part A

Train accuracy : **92.7%** Validation Accuracy : **80.1%**

Part B

Number of trainable parameters (weights + bias) : **7,678,474**.
It has also been cross-checked using the PyTorch library function "summary()" from "torchsummary library".

Part C

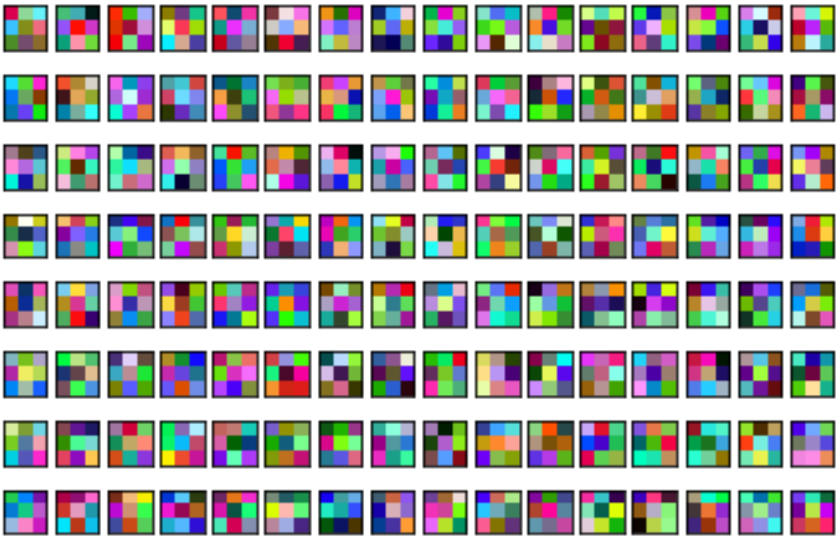

Figure 1: "Before training" weight visualization of first convolution layer

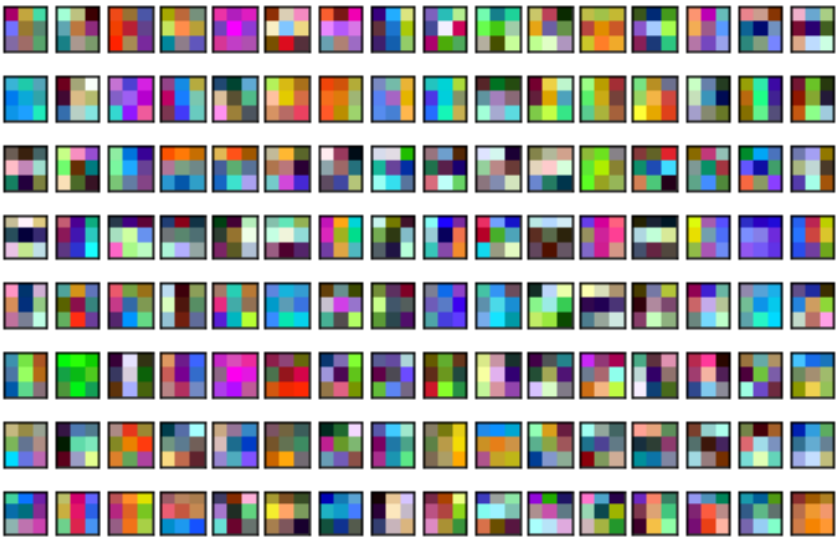

Figure 2: "After training" weight visualization of first convolution layer

Weight visualization analysis :
From Fig. 1, its evident that before training the model, all the weights for the 3x3 kernel had random values. In Fig. 2, after training the model for 20 epochs, although we cannot see much of a difference (since its the first layer), but some patterns are visible. In some of the 3x3 kernels we can see that some rows/columns have become darker and so they are trying to detect edges along that rows/columns but it is still not that easily evident. If we would have trained the model for some more epochs, we would have visualised the features more easily which are being detected.

2 Question 2

Part A : BATCH NORMALIZATION

Table 1: Model performance comparison between default model and the model with batch normalization

Model	Train accuracy	Validation Accuracy	Model Size	Test Accuracy
Without Batch Normalization (Q1.a)	92.7%	80.1%	7678474	79.9%
With Batch Normalization	98.1%	82.9%	7682826	81.9%

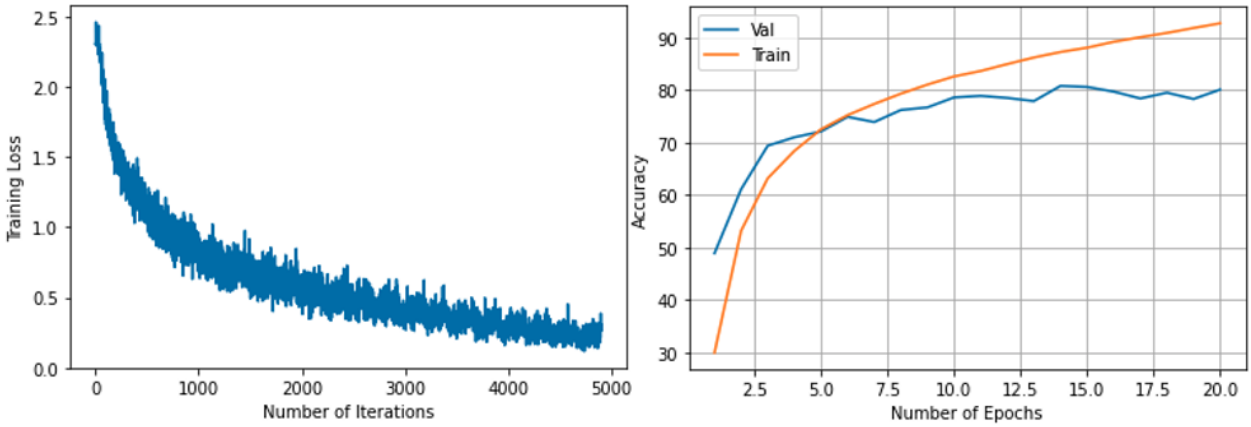

Figure 3: Learning curves of Model **without batch normalization** (Q1.a)

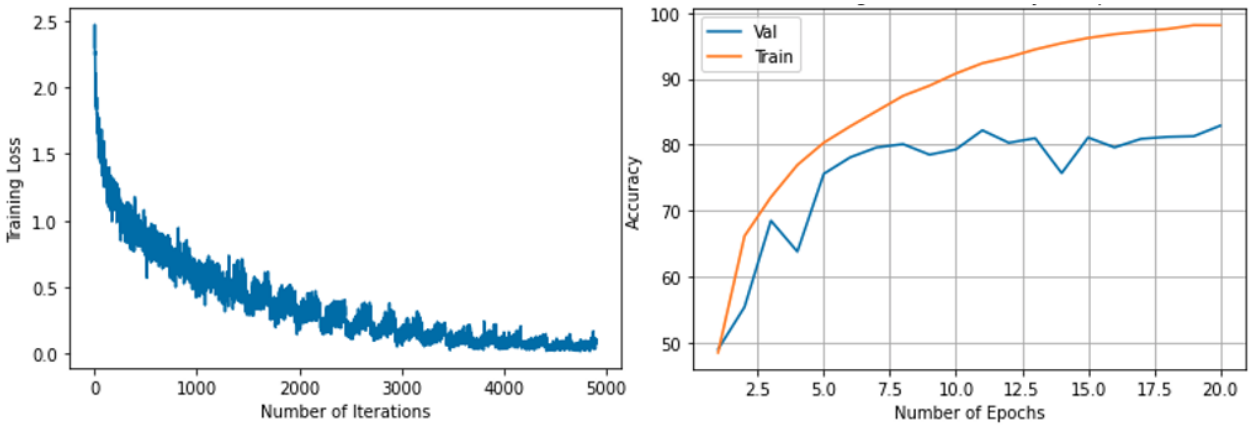

Figure 4: Learning curves of Model **with batch normalization**

Batch Normalization Analysis :
Since batch normalization, normalizes the layer's inputs int he architecture, the model's performance has increased as we can see from the Table 1 and Fig. 3.
Also, since we were asked to keep the other hyper-parameters the same, we did not change the learning rate. However, there is a possibility that the model with batch normalization would have performed better with a higher learning rate as well because the input distribution for each layer in the model is now normalized and we do not have to worry about the parameter initialization anymore.
From Table 1, we can also see that on adding batch normalisation at the inputs of every hidden layer, the model size increased. This is due to the fact that batch normalisation requires two extra learnable parameters (γ and β) for each neuron in the hidden layer.
Note : A flag (norm_flag) has been created in ex3_convnet.py (line 43) to enable/disable batch normalization.

Part B : EARLY STOPPING

Since, early stopping runs the model till the model starts to over-fit, we tried with different patience level and finally chose 10 as the hyper-parameter. Note : Since, we wont be running the model for all the epochs in case of early stopping, the best model (as asked in the question) is the model which gives the best validation accuracy and the latest model (as per the question) is the model trained after the 10th patience level.

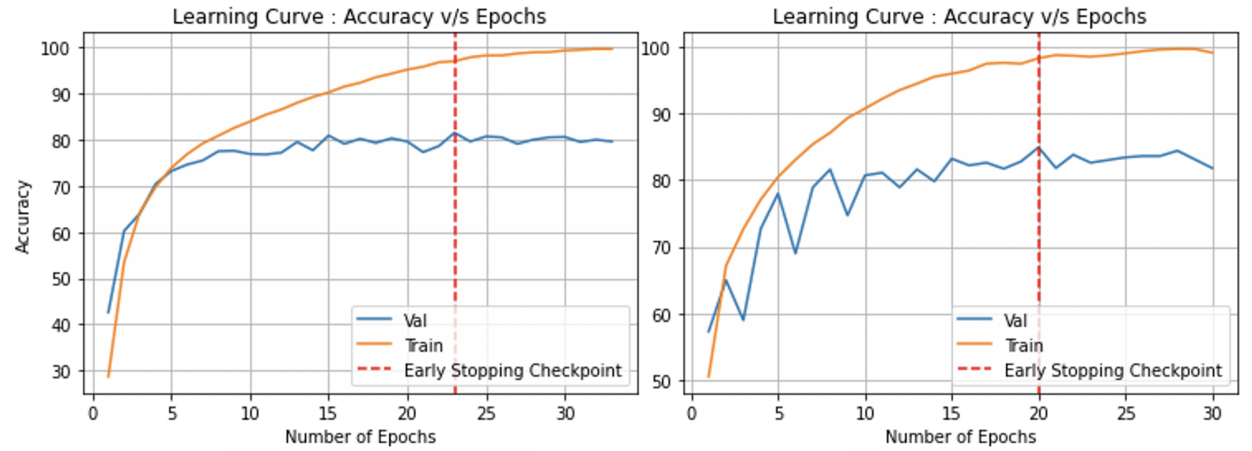

Figure 5: Accuracy curves for models having early stopping

Table 2: Model performance using Early Stopping with and without batch normalization with a patience level of 10

Statistics	Without Batch Norm (Q1.a)	With Batch Norm (Q2.a)
Early Stop Checkpoint	23	20
Best Model Validation Accuracy	81.5%	84.9%
Latest Model Validation Accuracy	79.6%	81.8%
Best Model Train Accuracy	97.0%	98.2%
Latest Model Train Accuracy	99.6%	99.0%

Early stopping analysis :
Epochs were increased to 50 to get a better analysis on how early stopping overcomes the problem of over-fitting. Two experiments were performed using early-stopping - a) without batch normalization , b) with batch normalization, with a patience level of 10 (i.e. how long to wait after last time validation loss improved).

From the table and plots given above, its evident that if we try to train the model even after the best model, the train accuracy improves but the validation accuracy does not. This proves that this technique avoids the model to over-fit and saves the computation as well.
We chose this value for the hyper-parameter (patience = 10) because a value lower than this was stopping the model to get trained at a lower level of validation accuracy, and for a value higher than this was not improving the validation accuracy much.

Note : Checkpoint for the best and latest models (with/without batch norm) will be saved in the same folder with their relevant names once the model is trained. We could not save and send the checkpoints due to submission file size limit fo 10MB.
A flag (earlystop) has been created in ex3_convnet.py (line 57) to enable/disable early stopping.

3 Question 3

Part A

Data augmentation analysis : We see that data augmentation helps in regularizing the model and improves the performance for the validation set, thus preventing the model to over-fit. In order to implement this, we used different techniques to vary the data like geometric and color transformations. From the table given below and the learning curves, it is evident that Random Horizontal Flip (with probability = 0.5) performs the best out of all the other techniques used, giving a validation accuracy of 84.9%. Moreover, we believe that the model performance can also improve further if we randomly select these techniques on the data set. **Note :** In the default code, these techniques have been commented in ex3_convnet.py (line 79-87).

Table 3: Model performance with different data augmentation techniques

Type of transformation	Data Augmentation technique	Validation accuracy
Geometric Transform	Random Vertical Flip (p=0.5)	79.3%
	Random Horizontal Flip (p=0.5)	**84.9%**
	Random Rotation (degree=45)	75.9%
	Resize with random crop	78.9%
	Translation (translate=(0.5,0.5))	77.1%
Color Transform	Color Jitter	78.8%
	Greyscale	80.0%
	Scaling (scale = (0.7,0.7))	76.7%

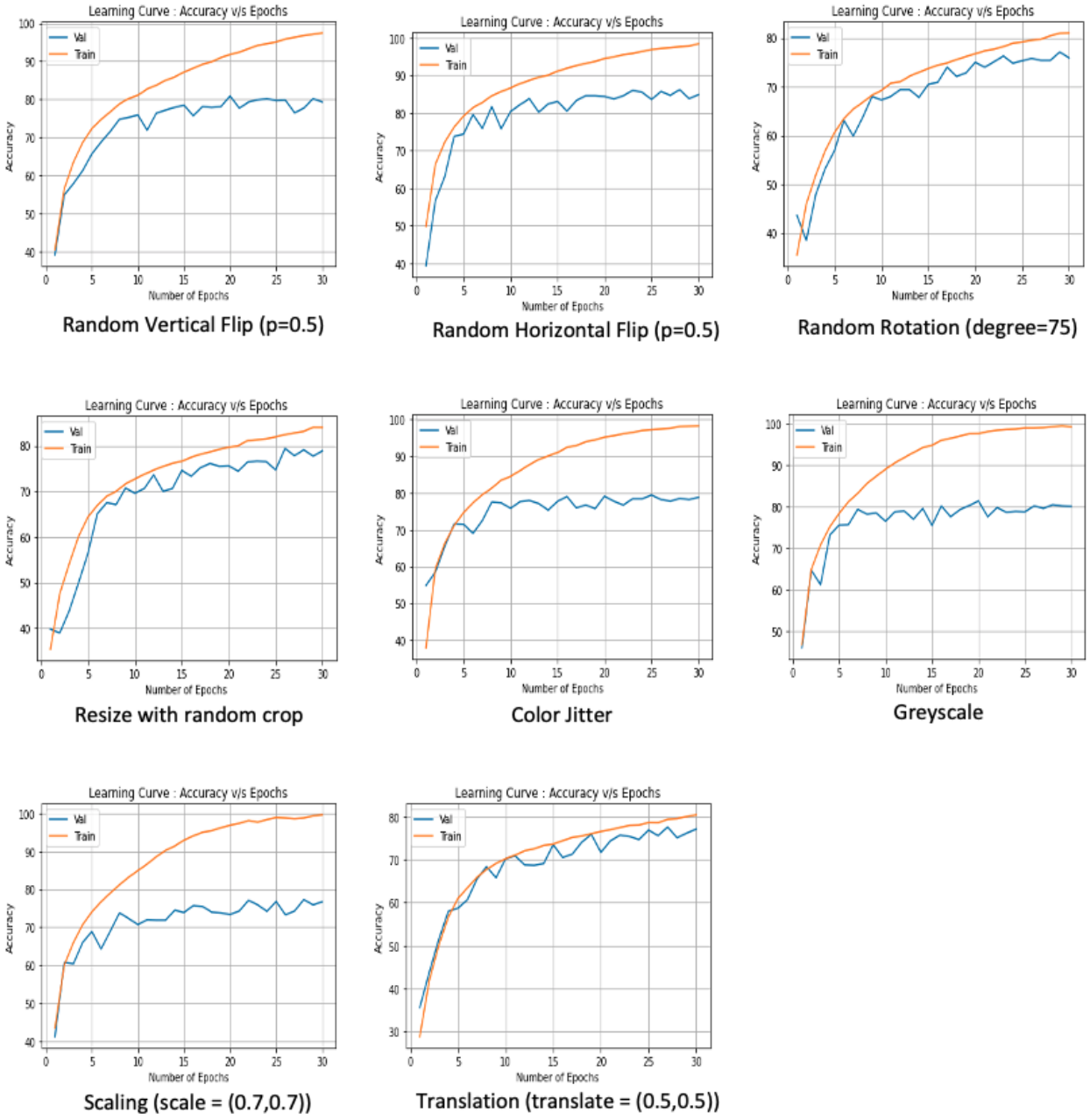

Figure 6: Learning curves with different data augmentation techniques

Part B

Dropout analysis :
From the given plots and table, it can be clearly seen that as we increase the value of dropout, the model starts to regularize and avoid over-fitting the model. At an optimal value of p = 0.3, it performs the best and on further increasing this value, the model capacity decreases significantly and performs badly even for the training set. Therefore, the best hyper-parameter for dropout value (p) is **0.3** providing the highest validation accuracy of **85.9%**.
Note : A flag (dropout) has been created in ex3_convnet.py (line 50) to enable/disable dropout. The same flag is also used to pass the dropout value.
The same value of dropout has been used in all the layers of the model.

Table 4: Model performance for different values of dropout (0.1-0.9)

Dropout Value	Train accuracy	Validation Accuracy
0.1	93.1%	83.8%
0.2	89.4%	85.8%
0.3	**85.7**%	**85.9**%
0.4	82.3%	82.1%
0.5	79.1%	82.1%
0.6	74.9%	74.5%
0.7	69.1%	66.0%
0.8	58.7%	42.3%
0.9	37.0%	10.3%

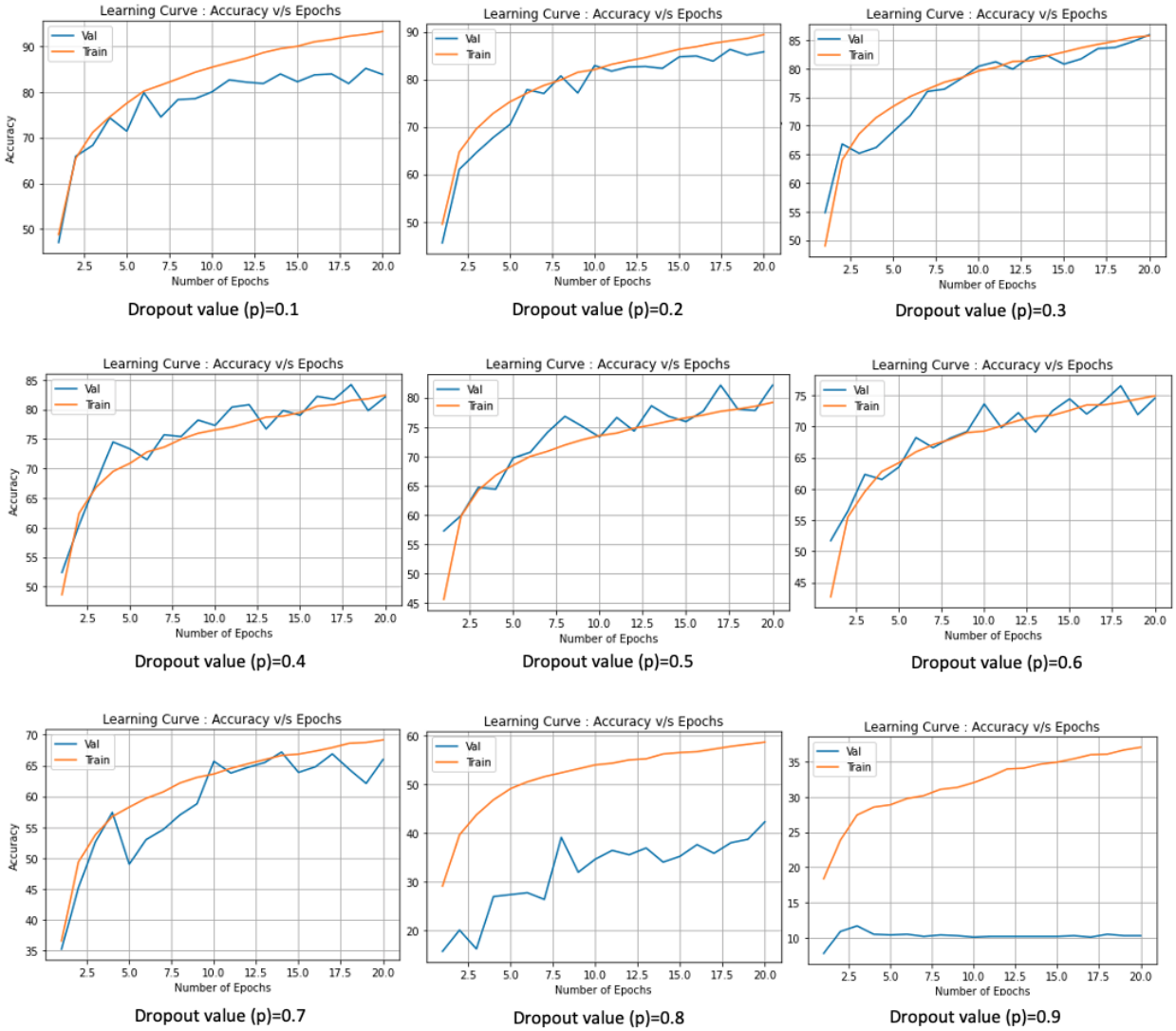

Figure 7: Plots of training and validation accuracy for different values of dropout (0.1-0.9)

4 Question 4

Part A

Analysis : It can be seen from below figures and table, that even though the model is able to learn decently but still the model performance is very poor and the accuracy only lies between 60-66%. This is due to the fact that the weights of pre-trained model have been directly used without letting them trained, and the only learning which is taking place is for the weights belonging to the newly added layers. The pre-trained weights were good for ImageNet dataset which have images with different dimensions, however they are not suitable for CIFAR-10. Therefore, we definitely need to fine tune them in order to get better performance on CIFAR-10 dataset.
We have used the same Early Stopping mechanism as we used in the previous question (Q2.b) with a patience level of 10.
(Best Model's Checkpoint saved as : best_model_part_4a.ckpt)

Table 5: Performance of model with disabled gradients for pretrained parameters

Validation accuracy	Test Accuracy
65.1%	61.6%

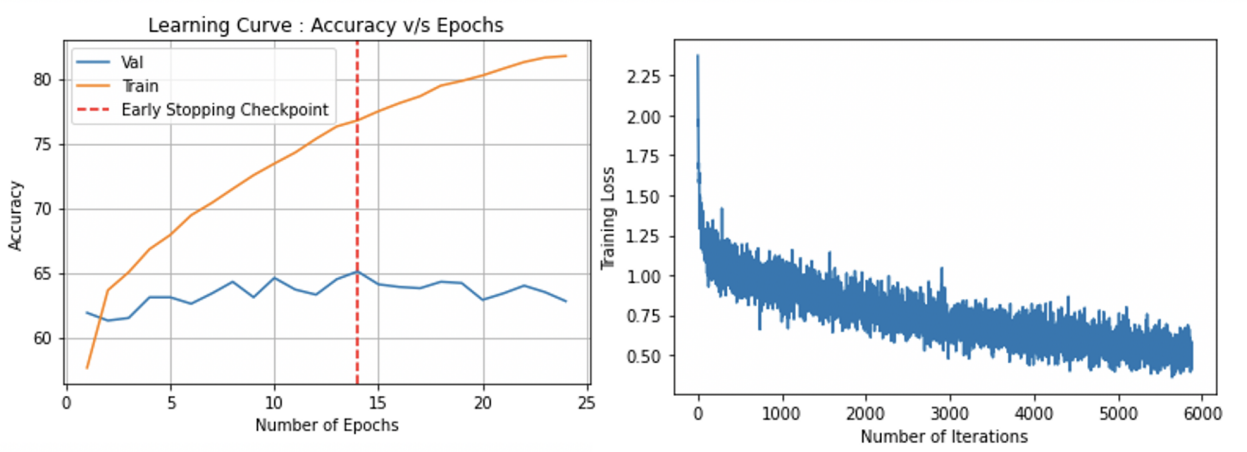

Figure 8: Learning curves of 4(a)

Part B

Model 1 : Fine tuned model in which model is not trained from scratch but gradients have been computed for all parameters. *(Best Model's Checkpoint will be saved as : best_model_part_4b_1.ckpt)*
Model 2 : Baseline model which has been trained from scratch.*(Best Model's Checkpoint will be saved as : best_model_part_4b_2.ckpt)*

Table 6: Performance of fine-tuned models for the best model using Early Stopping

Models	Early Stop Checkpoint	Validation accuracy	Train accuracy	Test Accuracy
Model 1	11	89.1%	97.8%	88.6%
Model 2	19	88.7%	98.0%	84.4%

Analysis : On performing the fine-tuning for the whole network, it can be seen that the model performance increases rapidly from 60-65% (as we saw in Q4.a) to 88-89% (Model 1 and Model 2 in Q4.b).
Moreover, it can also be concluded that the model performance does not vary a lot between Model 1 and Model 2. Model 1 is only slightly better than Model 2 when it comes to validation accuracy with a margin of hardly 1%. In term of test accuracy, Model 1 clearly outperforms Model 2 with a margin of 4%. Even if, we run these models again the overall trend would almost be the same with only slight differences.

Note : By default, ex3_pretrained.py will run Model 1 which gives the highest validation accuracy.

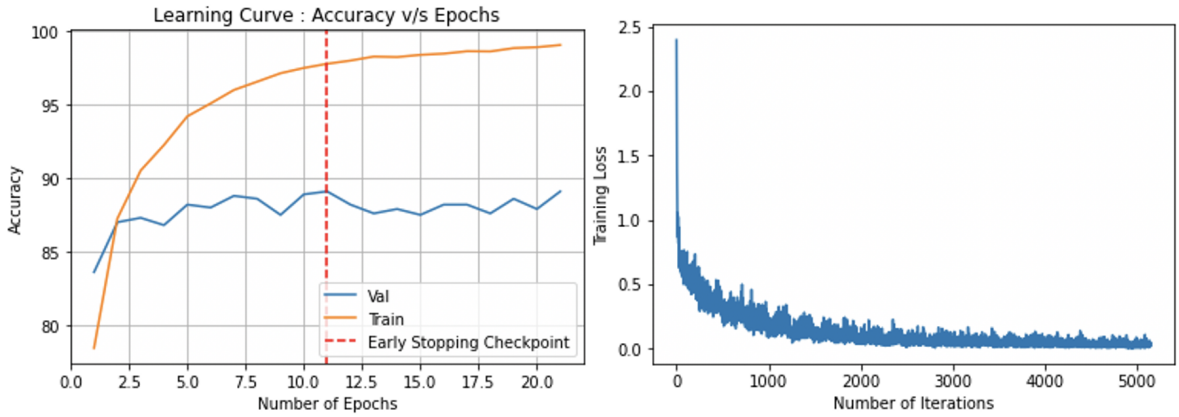

Figure 9: Learning curves of Model 1

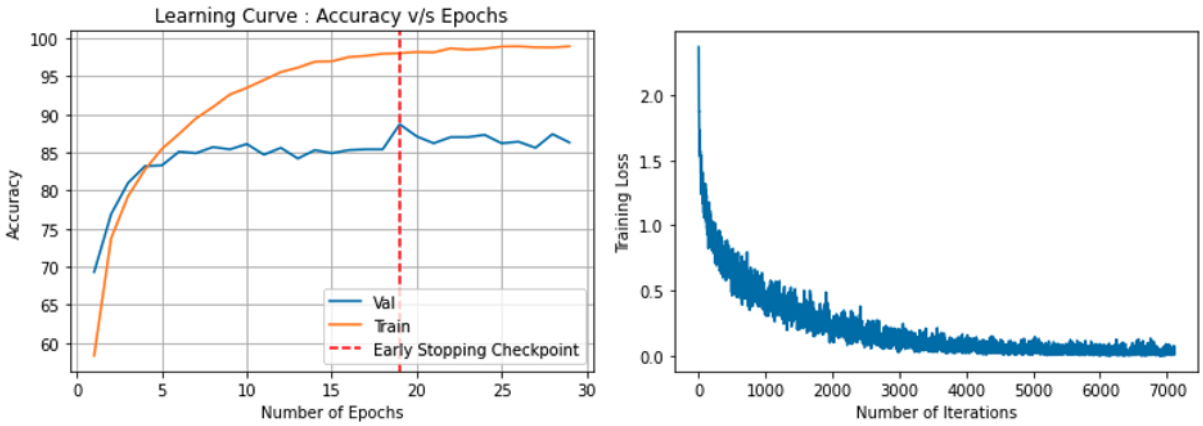

Figure 10: Learning curves of Model 2

Overall Analysis : In this assignment we used different approaches to improve the generalization of the CNN model and we saw that techniques like Early Stopping and Dropout not only helped in avoiding over-fitting of the model but also saved a lot of computational effort in terms of time as well as memory. Similarily, batch normalization and data augmentation helped in better generalization of the model.

Furthermore, in Q4 we saw that transfer learning with fine tuning approach significantly increased the model's performance with a great margin when compared to the techniques which we used in Q2,3.

Conclusion : We tried and tested various popular techniques to better generalise the model however, that there is no fixed rule for using a particular technique. This is because the model performance mostly depends upon the kind of available data set and the type of model being used.

NOTE : Due to submission file size limitation of 10MB, we were not able to send any checkpoints for the best models.