Name Surname: Duygu Serbes
Student Number: 2020700171

CMPE 597 Sp. Tp. Deep Learning
Spring 2021 Project II
Due: May 26 by 11.59pm

Answers

1. (15 pts) **Implement data augmentation techniques, such as random flip and random crop. Do not forget to standardize or normalize your dataset before training.**

   CIFAR10 consists of 60000 images in 10 classes. In this project, the training datasets splits randomly into two parts as traning which contains 40000 images and validation which contains 10000 images. Also, test dataset contains 10000 images [1]. Validation dataset is used to performance while training without applying any data augmentation methods such as cropping, flipping. According to validation loss value, the best model is saved to calculate test accuracy.

   As a base model, the model take images in the shape of 32x32x3. The first CNN layer has 16 feature maps, the second one has 32 feature maps, the last one has 64 feature maps. 2 fully connected layer which are 512 and 10 dimensional layer are used respectively and ReLu activation function is chosen to be used to all layer except the last fully connected layer, where softmax activation function is used. I have used kernel size of 3 with padding 1 and stride 1 and 2x2 pooling is used. In the base model there is no batch normalization and dropout. Learning rate is chosen as 0.01 and batch size equals to 32. SGD optimizer is used and cross-entrpy loss function is selected for base model[2].

   For the normalization, mean values are $[0.4914, 0.4822, 0.4465]$ and standard deviations are $[0.2023, 0.1994, 0.2010]$ for each channel respectively[3].

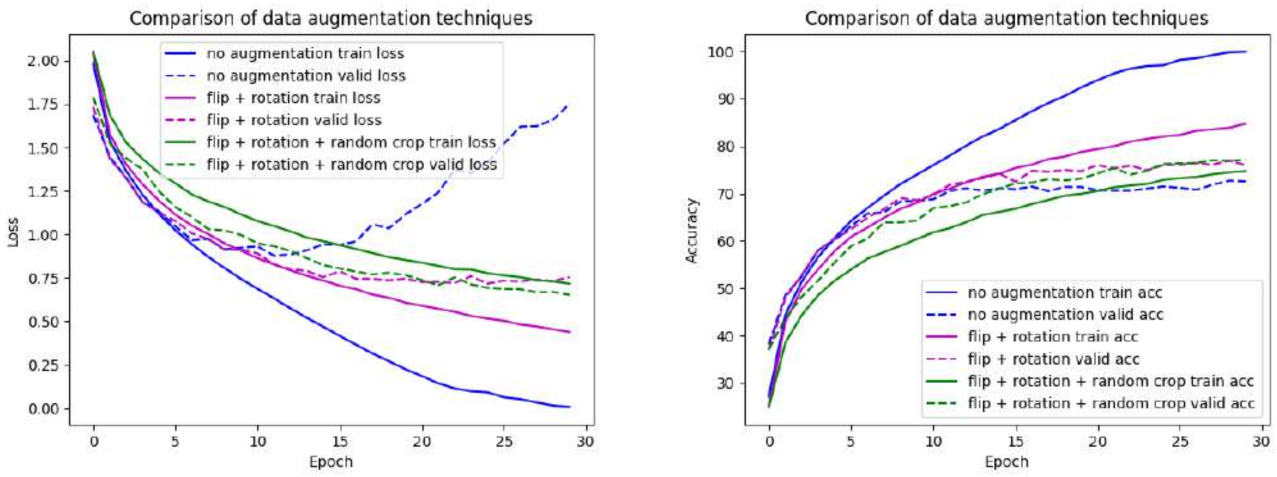
   Different data augmentation techniques are compared the model with no data augmentation. The comparison experiment models iterated up to 30th epoch. Each data augmentation technique does not handle one by one. Some of combinations of them are listed here. After normalization, the first model does not include any data augmentation method. The second model includes horizontal flip and rotation in the range of 10 degree. The third model put random cropping transformation with padding equals 4, which is added to the second model transformation by selecting 'padding_mode' parameter as 'edge', which fills the crop region according to last remaining pixel in that region [4].

   As seen in Table1, the best performing data augmentation technique is the combination of horizontal flip, 10 degree rotation and random crop. The rest of the experiment, this augmentation combination is used.

Table 1: Data Augmentation Experiments

Model	Test Accuracy
No Data Augmentation	0.70
Horizontal flip + 10 Degree Rotation	0.75
Horizontal flip + 10 Degree Rotation + Random Crop	0.77

In the Figure 1, the test and training loss values belongs to these 3 model can be seen. It is obvious to see that when there is no data augmentation, the model is overfitting.



(a) Loss values of models

(b) Accuracy values of models

Figure 1: Data Augmentation Methods Comparison

2. (30 pts) **Design a 3-layer CNN architecture for object classification task. A 3-layer CNN may not give you the state-of-the-art performance on CIFAR10 data. You will explore ways to obtain as high performance as possible with the 3-layer CNN.**

   - Discuss the reasoning behind your choices on the kernel size, activation functions,number of feature maps, number of fully connected layers, and the size of the fully connected layers
   - Investigate techniques to improve the performance, such as adding dropout, batch normalization, residual blocks. Discuss your observations on the effect of these techniques on the performance.

At this stage, I changed learning rate 0.001 with momentum 0.9. It will increase the test accuracy 1.5 percent at the epoch 50. The rest of model remains same as the defined model in question 1. But the tuning of learning rate will be analyzed in detail during the optimizer selection.

The number of parameter increases with the kernel size, which computationally inefficient to use large kernel size as learnt in the class. When we take a look to the literature, most famous architecture tends to use small kernel size like 3x3 with deeper networks instead of using large kernel size in recent years [5]. It is more computationally efficient and it increase accuracy by taking benefits of deeper network. For instance, using 2 successive convolution layer of kernel size 3 gives the same shape of output with the convolution layer with kernel size of 5. But the first one requires 18k parameters while the second architecture requires 25k parameters. In order to conduct more experiments for the other hyper-parameters, it is better to keep number of parameter low to decrease the training time. Therefore, kernel size is selected as 3x3 matrix.

In this project we are restricted to use 3 convolution layer, therefore, conducting some experiments by using different number of feature maps. When we increase the number of maps in the convolution layer, the dimension of vector passes the first fully connected layer is increased. Therefore, in that case adding one more fully connected layer will be helpful. The model, which has 48, 96 and 192 feature maps respectively for the three defined CNN layers, has output 3072x4x4 to be flatten, it is quite large number for the fully connected layer. In order to not lose any information, one additional fully connected layer is added.

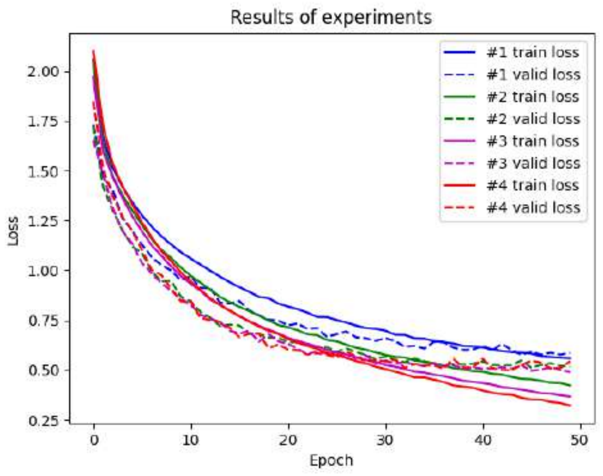
Some experiments are carried out and architecture parameters is given at Table 2. The first three column shows the number of feature maps in the CNN layers. Next 3 column shows the output vector size of fully connected layer. As seen in the table, test accuracy is the best for the forth model which has 3 fully connected layer; but the difference quite small from the third model. Adding one more fully connected layer is increasing the computational requirement and chance of over fitting. When considering advantage and disadvantage of additional fully connected layer, 2 layer fully connected layer in model seems more logical.

According to one research and its experiments, the shallow CNN models needs to more nodes in fully connected layers as well as more number of fully connected layers when compared to deep CNN models [6]. Therefore using only one fully connected layer is not suitable in that case.

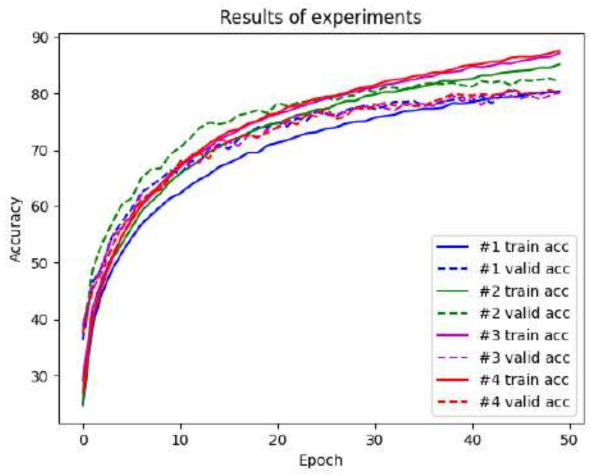
Table 2: Experiment for fearure maps and fully connected layers

#	CNN1	CNN2	CNN3	FC1	FC2	FC3	Train Acc	Valid Acc	Test Acc	Test Loss
1	16	32	64	512	10	-	0.7995	0.8017	0.7956	0.5933
2	32	64	128	512	10	-	0.8405	0.8243	0.8229	0.5196
3	48	96	192	512	10	-	0.8781	0.8425	0.8398	0.4960
4	48	96	192	1024	256	10	0.8665	0.8370	0.8385	0.4937

Also, the Figure 2 shows the training and validation loss and accuracy through the epochs up to 50. As seen in this figure, there is no significant difference between model 3 and model 4. Also better loss value is taken from third model with 0.4898. Therefore, as a result of this experiment, continuing with 2 fully connected layer will be better in terms of computation time and loss.



(a) Loss values of models



(b) Accuracy values of models

Figure 2: Results of experiments

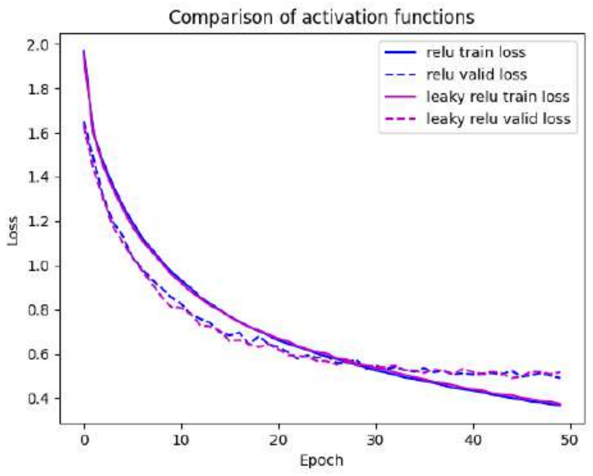
After that point, the architecture designed as 3 CNN layer and first CNN layer has 48, the second CNN layer has 96,and the third CNN layer has 192 feature maps. After each layer pooling is used. After the flattening the input vector of first fully connected layer is

ReLu or Leaky ReLu activation functions are most recommended activation functions for CNNs. The reason behind the ReLu popularity is that it is less prone to vanishing gradient problem and more computationally efficient by converting all negative inputs to zero. But it causes "dying ReLu" problem by learning nothing in negative region. In that case, Leaky Relu can be used [7][8]. Experiments are conducted by using these two activation function except last layer.

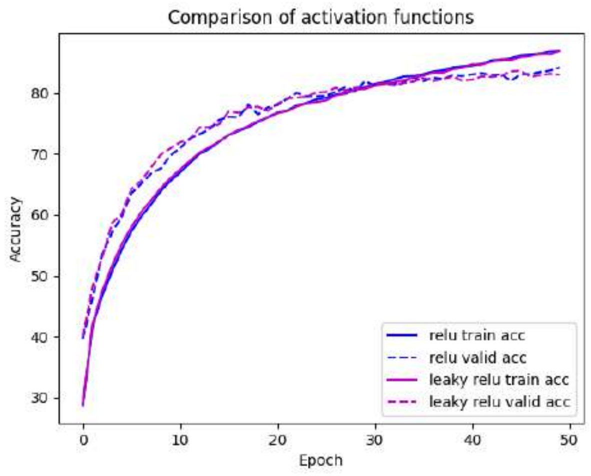
Table 3 demonstrates best training accuracy, minimum training loss, best validation accuracy, minimum validation loss, test accuracy and test loss. In figure 3, the behaviour of loss and accuracy can be observed up to epoch 50. As seen the table and figure, there is no significant difference between ReLu and Leaky ReLu. Therefore, ReLu is preferred for the rest of the project as used as famous CNN models.

Table 3: Activation function experiments

Activation Function	**Train acc**	**Train loss**	**Valid acc**	**Valid loss**	**Test acc**	**Test loss**
ReLu	0.8758	0.3668	0.8425	0.4898	0.8390	0.4960
Leaky ReLu	0.8548	0.4132	0.8360	0.4899	0.8293	0.5068



(a) Loss values of models



(b) Accuracy values of models

Figure 3: Comparison of activation functions

To increase the accuracy of model, dropout technique is added to the model because it prevents overfitting by deactivating some neurons randomly. The dropout technique implemented fully connected layers as first in literature but some studies suggest to use dropout in CNNs with low drop probability to helps filters to learn features[9][10]. The paper suggest to not use dropout for each CNN layer due to the risk of low training process because when dropout is used successively, activation signals are dropped significantly[10].

In the first iteration, one dropout added after the first fully connected layer with probability 0.3. In the second iteration another dropout added after the second CNN layer with the probability 0.1. The second iteration is repeated in the third iteration by changing dropout value from 0.1 to 0.2. At the forth iteration new dropout is added after flatting of CNN3 with probability rate is selected as 0.3 at the top of the second model. The result can be seen on Table 4.

As a result of these iteration second iteration is selected with the lowest validation loss, 0.4554, but in order to select the best I have to iterate until the convergences it can last up to 200 epoch, but in order to decide in quick manner, the iteration conducted up to epoch 50. The rest of the experiments will be done using second iteration combination as a base model. It means 2 dropout, one of them is after second CNN layer with probability 0.1, the other one is after first fully connected layer with probability 0.3.

Table 4: Dropout effect

Models	Train acc	Train loss	Valid acc	Valid loss	Test acc	Test loss
Base	0.8758	0.3668	0.8425	0.4898	0.8390	0.4960
One dropout after FC1	0.8399	0.4526	0.8385	0.4818	0.8359	0.4843
One dropout after CNN2 (p=0.1) + One dropout after FC1	0.8256	0.5004	0.8456	0.4554	0.8381	0.4727
One dropout after CNN2 (p=0.2) + One dropout after FC1	0.8157	0.5152	0.8398	0.4653	0.8326	0.4873
One dropout after CNN2 (p=0.1) + One dropout after CNN3 flattening (p=0.3) +One dropout after FC1	0.7952	0.5795	0.8375	0.4767	0.8315	0.4844

As learnt in the DL class, batch normalization prevents internal covariate shift and provide fast convergence. Therefore, I have added batch normalization all CNN layers and first fully connected layer. It means I use batch normalization for all layer except the last layer. It is important to note that while using bath normalization, "model.eval()" operation provides training mean and standard deviation on validation and testing time.

Table 5 summarizes the experiment result, batch normalization significantly increase validation loss and test accuracy with increase 2 percentage. Therefore, for the rest of the experiments batch normalization usage will be kept.

Table 5: Batch normalization effect

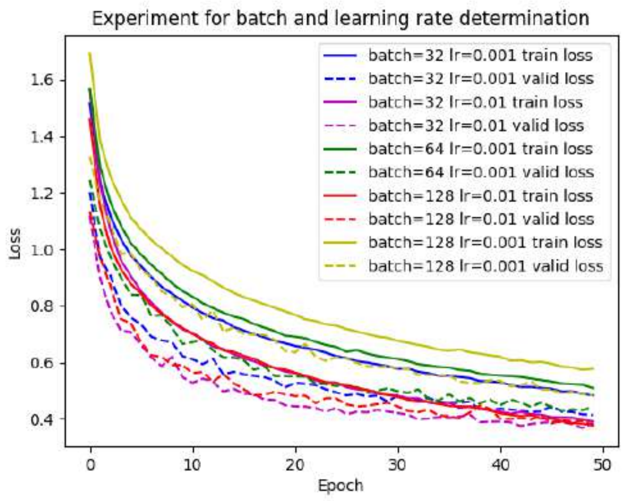
Models	Train acc	Train loss	Valid acc	Valid loss	Test acc	Test loss
Base	0.8256	0.5004	0.8456	0.4554	0.8381	0.4727
Base with batch normalization	0.8314	0.4839	0.8597	0.4144	0.8548	0.4230

3. (15 pts) **Choose one optimizer and train your network. Plot the value of the loss function with respect to number of epochs. Determine the batch size. Report training and test classification accuracy. Discuss your early stopping procedure.**

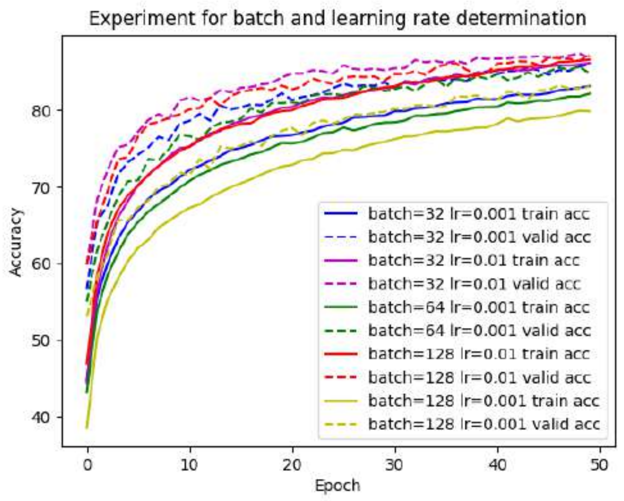
   Up to that point, all iterations conducted with SGD while learning rate equals to 0.001 and momentum equals to 0.9. To see the overall performance of this combination, I will keep the SGD optimizer for this question and iterate the model up to 120 epoch after determining the batch size and optimum learning rate. In the forth question, Adam optimizer will be tried as a more state-of-art optimizer.

In order to determine batch size and learning rate below experiment shown in Table 6 are conducted. The batch size 32, 64 and 128 will be iterated with learning rate 0.01 and 0.001 while keeping momentum 0.9. As known, large batch size can cause poor generalization. According to table 6, the best validation loss value and best test accuracy can occur with batch size 32 and learning rate 0.01. Also in Figure 5, there can be seen the loss and accuracy values of training and validation datasets. Starting with learning rate 0.01 works well for all batch size. while batch size increases while keeping learning rate constant, the loss and accuracy values do not improve effectively. Therefore, rest of the experiment, batch size equals to 32 and learning rate equals to 0.01 at the beginning of training for SGD.

Batch size	**Learning rate**	**Train acc**	**Train loss**	**Valid acc**	**Valid loss**	**Test acc**	**Test loss**
32	0.001	0.8314	0.4839	0.8597	0.4144	0.8548	0.423
32	0.01	0.8598	0.3983	0.874	0.3685	0.8732	0.3807
64	0.001	0.8151	0.5253	0.8559	0.4284	0.8432	0.4488
64	0.01	0.8727	0.3605	0.8714	0.3863	0.8727	0.3915
128	0.001	0.7985	0.5778	0.8309	0.4836	0.8276	0.4923
128	0.01	0.8661	0.3785	0.8714	0.3789	0.8676	0.3937



(a) Loss values of models



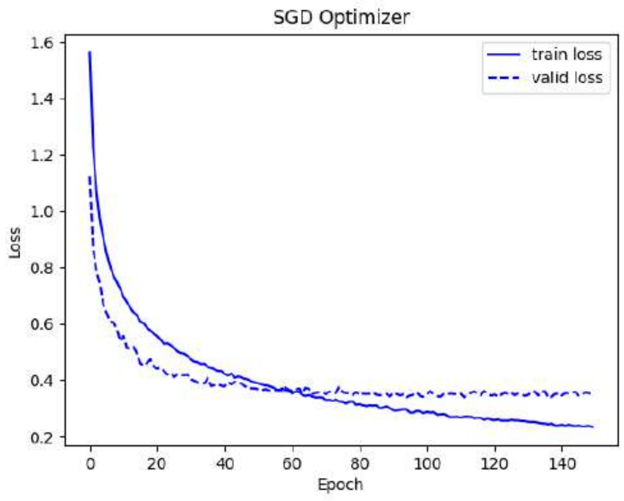
(b) Accuracy values of models

Figure 4: Experiments for batchsize and learning rate determination

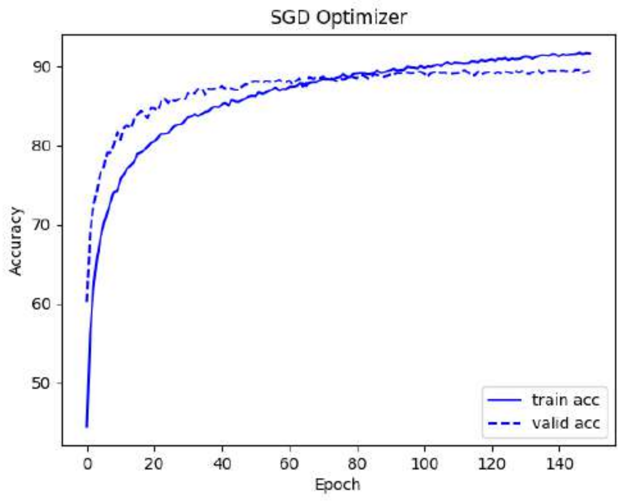
But in further epoch, learning rate can be small to get better result. In order to do achieve get better learning rate in further epoch decay can be used or directly divide learning rate by 10 after certain epoch. I chose to divide learning rate by 10 after epoch 100.

For this project and also in general manner, best model is determined by looking to minimum validation loss. After each epoch, algorithm checks if it find lower validation loss and save the model according to that. Thanks to early stopping, we prevents overfitting by stopping it if stopping metric(validation loss) is not improved. As a early stopping procedure applied in this project, if the algorithm cannot find the better result through 20 epoch, it decides to stop and get the test result according to last saved algorithm. 20 epoch no improvement can be high, it can be reduced, but here I want to observe the all iterations.

The final model gives 91.38% train accuracy, 89.46% validation accuracy at epoch 133 as best model with 0.338 validation loss. The saved model provides 88.40% test accuracy. The loss and accuracy values can be seen on Figure 5 the algorithm iterated through 150 epochs. As seen in this figure, the algorithm cannot be improved after epoch 60, therefore the number of epoch the early stopping have to take criteria can be minimized to 10 to be an example. If 10 was selected, the algorithm stopped at 78th epoch with 88.34% validation accuracy and 0.354 validation loss, which is quite effective results for this kind of model.



(a) Loss values of model



(b) Accuracy values of model

Figure 5: Model Results (Optimizer=SGD)

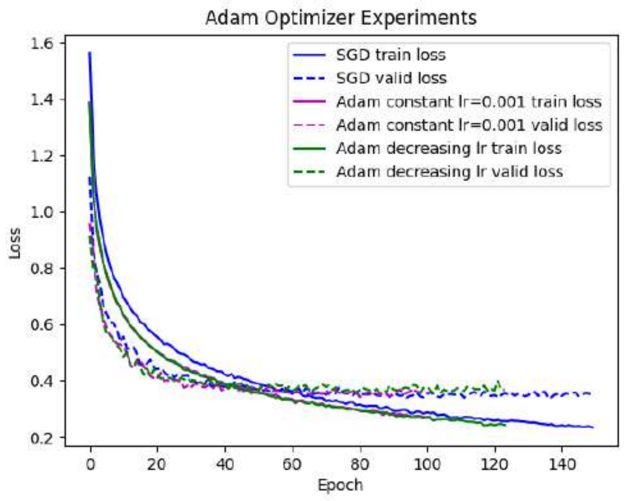
4. (15 pts) **After determining the final hyperparameters, try different optimizers and compare their performance and convergence properties.**

   Adam optimizer is very popular optimizer in recent years due it its advantages such as easy implementation, low memory requirements, little tuning requirement for hyperparameter, high performance on sparce or noisy gradients etc [11]. There, selecting Adam optimizer can be good choice. Some experiment conducted to see what happens while using Adam optimizer and compare it with the pre-trained model using SGD optimizer. In the first model, Adam optimizer and learning rate 0.001 are selected. In that case, the model is stopped at epoch 101th with 90.38% training accuracy, 0.2719 training loss, 88.31% validation accuracy and 0.3595 validation loss. The test accuracy is 88.06%.

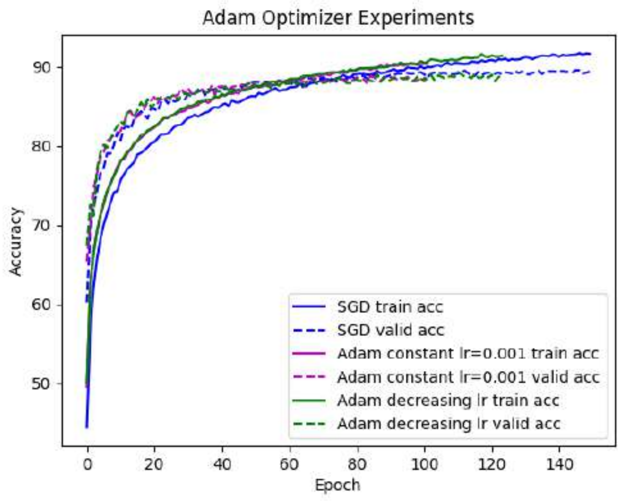
As a second iteration conducted with Adam optimizer has changing learning rate. The algorithm starts with learning rate 0.001 and at epoch 50, it is reduced to 0.0001 and at epoch 100 it is reduced to 0.00001. This iteration stops at epoch 123 with 88.38& test accuracy. Validation accuracy is 89.12% and validation loss is 0.3574.

In Figure 6, blue line is putted for comparison, represents the model trained with SGD. The purple line demonstrate first iteration conducted with Adam. The green line shows the second iteration which has decreasing learning rate according to epoch. As a result, the model which using Adam converge faster. Due to it, they are already stopped before 150th epoch.

When we look at the testing accuracies, the best model is SGD with 88.40% test accuracy but the second iteration has small difference(0.02) with 88.38% test accuracy. So using Adam optimizer can be advantageous while considering fast convergence. By this way, we can stop the model at lower epoch and save time.



(a) Loss values of model



(b) Accuracy values of model

Figure 6: Adam optimizer experiments

5. (20 pts) **Visualize the latent space of the model for the training samples (output of the feature extraction layers) in the beginning of the training, in the middle of the training, and in the end of the training. To visualize, map the latent representations into 2-D space using t-SNE. Use different colors to illustrate the categories. Compare the plots. Can you observe 10 groupings in the end of the training?**

   As a result of analysis carried out with the flattening layer outputs of training data, which is 3072 dimensional vector for each sample in my best model, below figures are generated belongs to different epoch using t-SNE [12]. In Figure 7 there is no significant clustering as expected at epoch 1. But since the training size includes 40000 images, the model is trained a little bit due to large number of mini-batch size.

It has 49.37% training accuracy which causes some small clustering in some regions. When we look at the t-SNE results shown in Figure 8, 9, 10, and 11, though the epochs the clustering belongs to 10 classes become clearer. To be more specific, at epoch 5 training accuracy equals to 70.91%, its t-SNE result is shown on Figure 8 and at epoch 60, training accuracy is 88.61% and clusters are more apparent as seen on Figure 11 when compared to previous plots. Here each class represented with different color and at high epochs, each class is clustered at certain regions even if there is no strict distinction between classes region.

The t-SNE plots includes 40000 points. Therefore in some regions, it is hard to see the details. But when I plot each class dots one by one, I can see the clustering improvement. On the other hand, when I investigate the accuracy level belongs to each class, I observed that highest accuracy belongs to 'automotive' class, it is represented with green in the plots and green color points are clustered apparently.

Also, according to my assumption some class share some features detected by some CNN layers. For instance in Figure 11, the closest class to the green points are the purple ones and they are belongs to 'truck' class and other vehicle types 'ship'(yellow) and 'airplanes'(red) are close to truck and automobile. To sum up, all vehicle types located the left hand side of the Figure 11 and the animal species are located to right hand side of the figure generally.

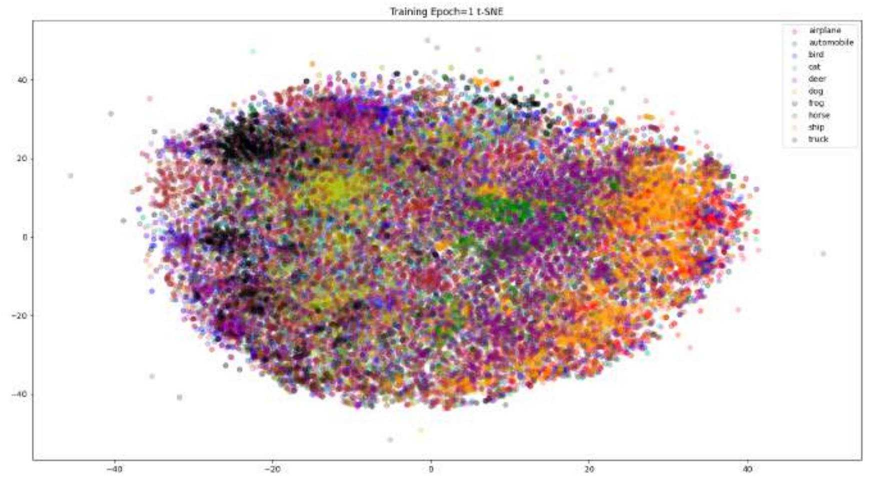


Figure 7: t-SNE Result Epoch=1

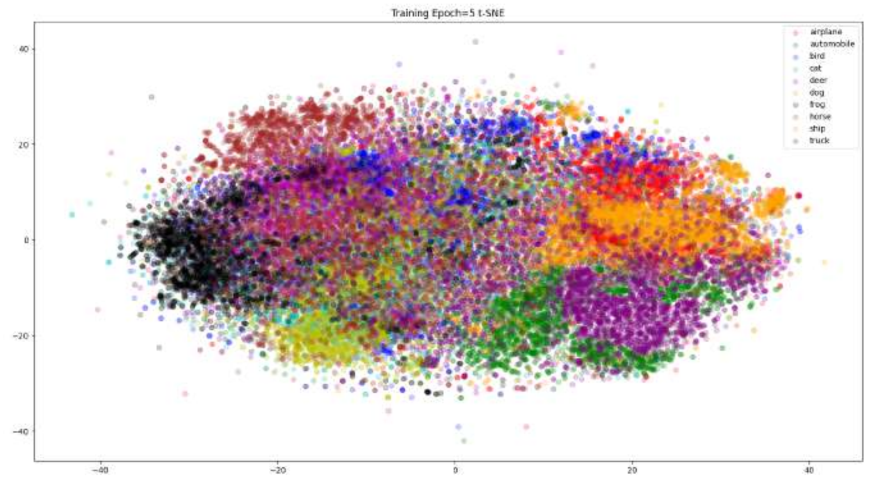


Figure 8: t-SNE Result Epoch=5

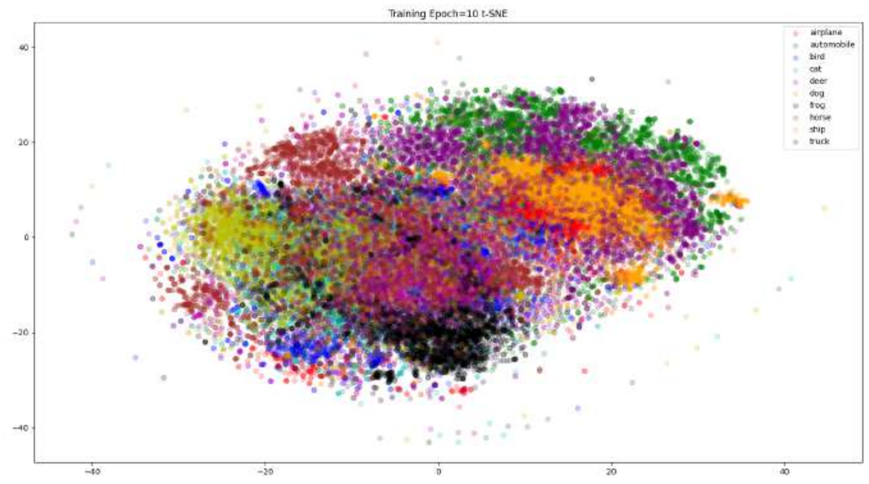


Figure 9: t-SNE Result Epoch=10

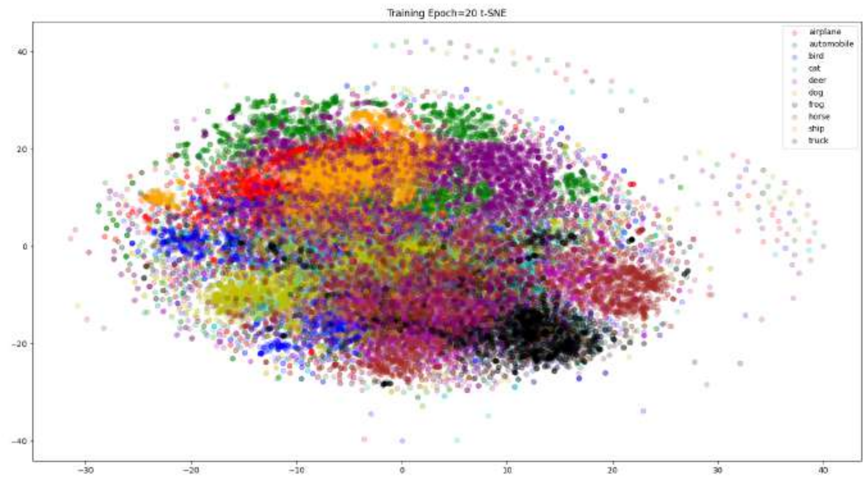


Figure 10: t-SNE Result Epoch=20

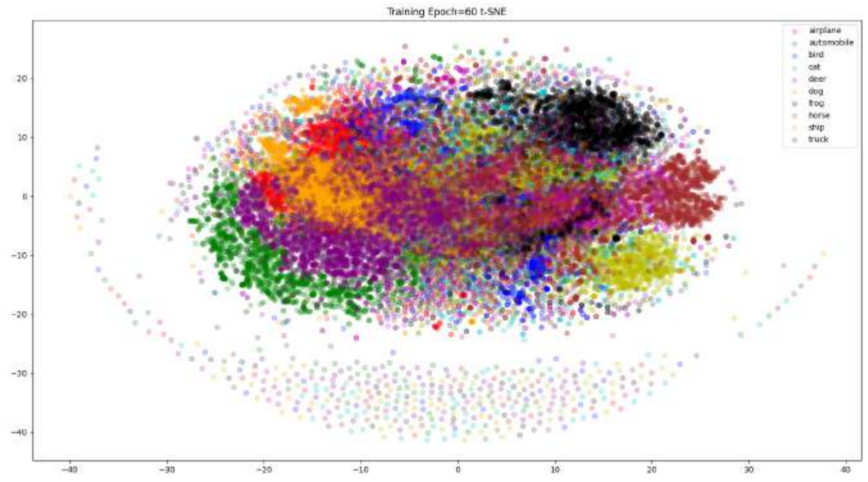


Figure 11: t-SNE Result Epoch=60

References

[1] A. Krizhevsky. The cifar-10 dataset. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[2] Training a classifier. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

[3] C. Lehmen. (2019) Learning to segment cifar10. [Online]. Available: https://charlielehman.github.io/post/weak-segmentation-cifar10/

[4] Pytorch torchvision.tranform. [Online]. Available: https://pytorch.org/vision/stable/transforms.html

[5] A. Chazareix. (2020) About convolutional layer and convolution kernel. [Online]. Available: https://www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel

[6] S. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "Impact of fully connected layers on performance of convolutional neural networks for image classification," *Neurocomputing*, vol. 378, p. 112–119, Feb 2020. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2019.10.008

[7] V. Jain. (2019) Everything you need to know about "activation functions" in deep learning models. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

[8] U. Udofia. (2018) Basic overview of convolutional neural network (cnn). [Online]. Available: https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012.

[10] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," in *ACCV*, 2016.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[12] tsne to visualize digits. [Online]. Available: http://scipy-lectures.org/packages/scikit-learn/auto_examples/plot_tsne.html

[13] Udacity Deep Learning Nanodegree Program self-notes.

[14] Bogazici University, CMPE597 Deep Learning Class self-notes.