Report: Three-Layer Convolutional Neural Network

In this project, we try to create a convolutional neural network model with three layers for CIFAR10 dataset. CIFAR10 dataset consists of 50000 train images and 10000 test images. When we unpickle the given data, the data is in the form (length of data, 3072). We know that CIFAR10 dataset has 32x32 RGB images; that is (32x32x3) images. Therefore, we can show one of the data by plotting it with matplotlib library. CIFAR10 dataset has 10 groups as labels are 'airplane', 'automobile','bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', and 'truck'. We can see a truck as an example data in Figure 1 and note that it is augmented image.

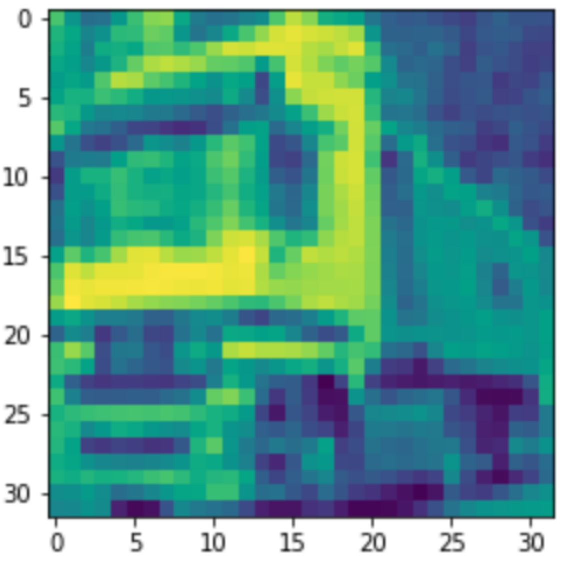

Figure 1: Truck - One of the data in CIFAR10

Perez and Wang (2017) says that data augmentation increases the model performance and thus model effectiveness. For **data augmentation**, random horizontal and vertical flips with the probability 0.1 for both, normalizing data with 0.30, 0.45, 0.50 means and 0.25, 0.36, 0.49 standard deviations and random rotation with 20 degrees are used. These numbers are selected randomly as small numbers between 0 and 1 and the degree for rotation is also random. Here, there is exactly no rule as the means should be 0.30. However, normalizing data enables us to get values of the images between 0 and 1 and therefore we use smaller numbers in model training. Sun et al (2020) talk about normalizing data which prevents too much increase in weights. Therefore, normalizing data is very important before model training.

In this project, Pytorch library is used to create a model. Pytorch wants inputs to be in the form (channels x height x width). Then, we have to reshape our data as (3, 32, 32) images. By using the information from one of our online classes, we can say that one of these three layers can include one convolution operation, activation function and pooling. For this purpose, we create three different models. Ritchie Ng and Jie Fu (2019) has Deep Learning Wizard webpage and we can follow this resource while creating our networks.

First Model use only convolution operation and ReLu as activation function in all three layers. The reason behind using ReLu activation function is that Agarap, A. F. (2018) says ReLu gives better solution rather than softmax function as an activation function. First layer has 8 (5x5x3) filters with stride 1 and padding 2. We can apply the formula to find output shape

$$O = \frac{W - K + 2P}{S} + 1$$

where O is output height/width, W is input height/width, K is kernel(or filter) size, P is padding when we assume same padding and S is stride. Therefore, we get (32x32x8) output of first layer. Second layer has 16 (5x5x8) filters and thus, we have (32x32x16) as an output of second layer. Similarly, third convolution layer has 32 (5x5x16) filters and its output shape will be (32x32x32). Therefore, we can apply fully connected layer. First model has only one fully connected layer. When we apply linear function to this fully connected layer, we want to 10 different outputs since we have 10 groups for this image classification task. Therefore, our fully connected layer has (32 * 32 * 32, 10) linear filter.

In deep learning models, the depth of the model increases as we go deeper because we use many filters to learn the features in the images. Here, we can discuss why we select 8 filters in first layer or why we select 16 filters in second layer. However, there is no definite answers for these questions because we can use another number of the filters. For example, talking about the first filter, there may not be a big difference for model performance between using 8 filters and using 9 filters. However, each filter enables us to information about features of the images. Although we deduce from here that the more filters we use, the more our model learns, but using too many filters can also cause overfitting problem. We have to be careful about this too. In this model, we want to get same convolution; that is the height and width of the input image is the same for output image.

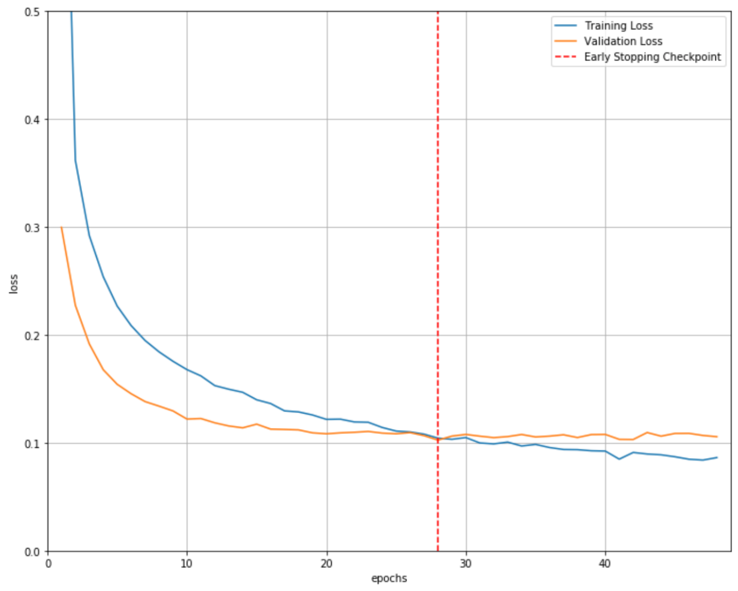

Figure 2: Early Stopping Criteria

In Figure 3, we can see the change of test and train loss for each epoch. Before analyzing our result for the first model, we should discuss our **early stopping** rule. In Figure 2, we can see that early stopping criteria (Bjarte Mehus Sunde, 2019). Zhang et al. (2016) also says we can stop the training when validation loss is greater than train loss. Here, we can use Bjarte's GitHub repository (2019) for early stopping. We can copy for early stopping file with pytorch, and then we can use it for model training. Increase in validation loss can say overfitting problem in the model. He says the model should stop the training if the validation loss starts to increase. In batch training or minibatch training, the losses are very changable, that is, increasing and decreasing continuously. However, we want to find the point that the validation loss always continue to increase. Then, we can also decide this point by using patience. For example, if patience is selected as 5, the training will stop after the validation loss increases 5 times.

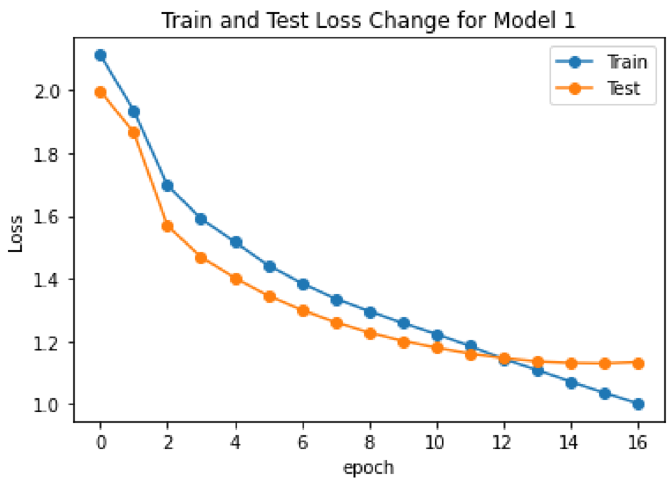

Figure 3: Three Layer CNN using convolution operation, activation function and 1 fully connected layer with SGD optimizer

Now, we can go back to talk about model 1 results. As we see in Figure 3, both test loss and validation loss values decrease for each epoch. We train the model with batch size 100 and we have 50000 training images. We can determine batch size by trying smaller dataset choosing from CIFAR10; as an example, we can use batch 1 images to determine batch size or other hyperparameters like learning rate. Loss values should decrease for each epoch since the model learns more after epochs. The model stops the training in 16th epoch since early stopping criteria. After the training, the model reach 0.6693800091743469 as training accuracy and 0.608299970626831 as test accuracy. According to Zhang et al. (2016), we may stop at 13th epoch. If we select this method for early stopping, we will get 0.6330599784851074 train accuracy and 0.597000002861023 test accuracy. After 16 epoch training, the results are good for this model. Although this model has only convolution operation, the result is not bad.

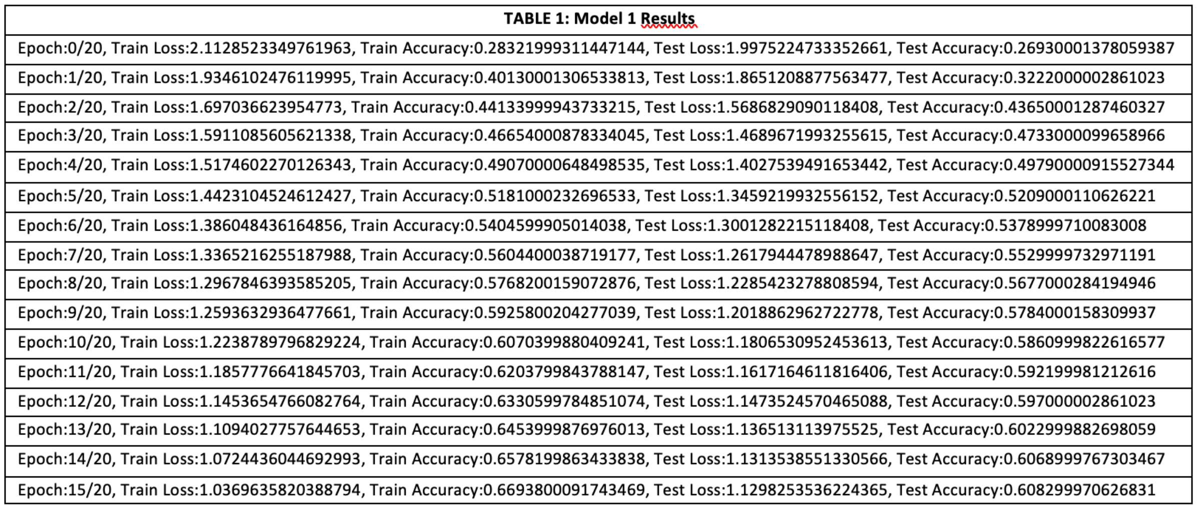

TABLE 1: Model 1 Results
Epoch:0/20, Train Loss:2.1128523349761963, Train Accuracy:0.28321999311447144, Test Loss:1.9975224733352661, Test Accuracy:0.26930001378059387
Epoch:1/20, Train Loss:1.9346102476119995, Train Accuracy:0.40130001306533813, Test Loss:1.8651208877563477, Test Accuracy:0.3222000002861023
Epoch:2/20, Train Loss:1.697036623954773, Train Accuracy:0.44133999943733215, Test Loss:1.5686829090118408, Test Accuracy:0.43650001287460327
Epoch:3/20, Train Loss:1.5911085605621338, Train Accuracy:0.46654000878334045, Test Loss:1.4689671993255615, Test Accuracy:0.4733000099658966
Epoch:4/20, Train Loss:1.5174602270126343, Train Accuracy:0.49070000648498535, Test Loss:1.4027539491653442, Test Accuracy:0.49790000915527344
Epoch:5/20, Train Loss:1.4423104524612427, Train Accuracy:0.5181000232696533, Test Loss:1.3459219932556152, Test Accuracy:0.5209000110626221
Epoch:6/20, Train Loss:1.386048436164856, Train Accuracy:0.5404599905014038, Test Loss:1.3001282215118408, Test Accuracy:0.5378999710083008
Epoch:7/20, Train Loss:1.3365216255187988, Train Accuracy:0.5604400038719177, Test Loss:1.2617944478988647, Test Accuracy:0.5529999732971191
Epoch:8/20, Train Loss:1.2967846393585205, Train Accuracy:0.5768200159072876, Test Loss:1.2285423278808594, Test Accuracy:0.5677000284194946
Epoch:9/20, Train Loss:1.2593632936477661, Train Accuracy:0.5925800204277039, Test Loss:1.2018862962722778, Test Accuracy:0.5784000158309937
Epoch:10/20, Train Loss:1.2238789796829224, Train Accuracy:0.6070399880409241, Test Loss:1.1806530952453613, Test Accuracy:0.5860999822616577
Epoch:11/20, Train Loss:1.1857776641845703, Train Accuracy:0.6203799843788147, Test Loss:1.1617164611816406, Test Accuracy:0.592199981212616
Epoch:12/20, Train Loss:1.1453654766082764, Train Accuracy:0.6330599784851074, Test Loss:1.1473524570465088, Test Accuracy:0.597000002861023
Epoch:13/20, Train Loss:1.1094027757644653, Train Accuracy:0.6453999876976013, Test Loss:1.136513113975525, Test Accuracy:0.6022999882698059
Epoch:14/20, Train Loss:1.0724436044692993, Train Accuracy:0.6578199863433838, Test Loss:1.1313538551330566, Test Accuracy:0.6068999767303467
Epoch:15/20, Train Loss:1.0369635820388794, Train Accuracy:0.6693800091743469, Test Loss:1.1298253536224365, Test Accuracy:0.608299970626831

Li et al (2019) enables us to better model for CIFAR10 when we dropout in between layers. Dropout is to ignore some parts in layers. For example, to classify image, we do not have to all data because some parts in the image may not be very important like information in edges. Also, the study of Thakkar et al (2018) shows batch normalization increase model performance for CIFAR10 dataset. Batch normalization enables us to get a method for normalizing inputs between layers. It also decreases the time for model training.

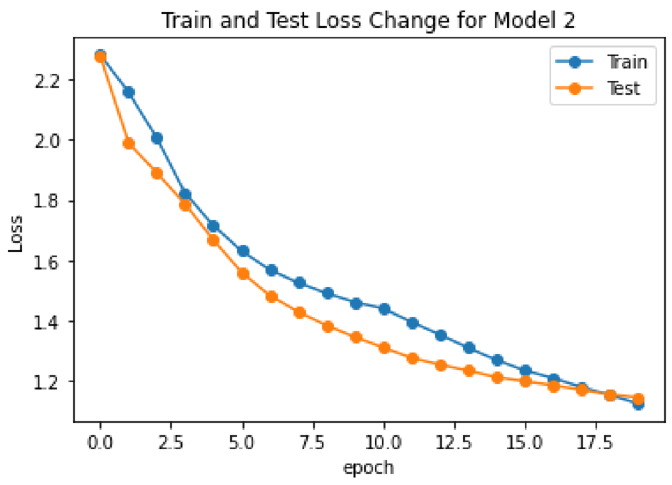

Figure 4: Three Layer CNN using convolution operation, activation function and 3 fully connected layers with SGD optimizer

Therefore, we will create 3 more models. Second model will show effect of number of fully connected layers, third model is to see effect of dropout and last model is for effect of batch normalization. To compare models, some parts are the same for all of them. For example, kernel size, padding, stride, number of filters and activation functions. After talking about these models, we will discuss whether or not we get the same solution in literature. We take first model as a base model and then we add other parts like dropout, batch normalization and number of fully connected layers. Also learning rates and optimizers are the same for the models. Learning rate is 0.01 and optimizer is SGD optimizer. After training all models, we will try it for different optimizers like Adam.

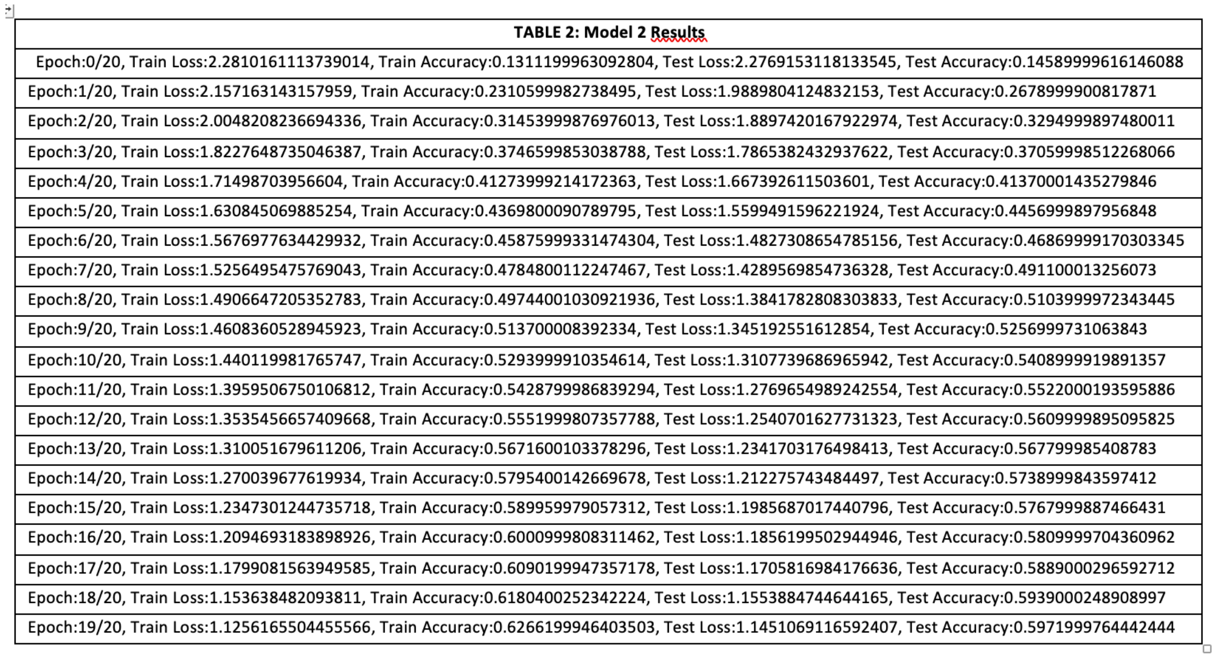

TABLE 2: Model 2 Results
Epoch:0/20, Train Loss:2.2810161113739014, Train Accuracy:0.1311199963092804, Test Loss:2.2769153118133545, Test Accuracy:0.14589999616146088
Epoch:1/20, Train Loss:2.157163143157959, Train Accuracy:0.2310599982738495, Test Loss:1.9889804124832153, Test Accuracy:0.2678999900817871
Epoch:2/20, Train Loss:2.0048208236694336, Train Accuracy:0.31453999876976013, Test Loss:1.8897420167922974, Test Accuracy:0.3294999897480011
Epoch:3/20, Train Loss:1.8227648735046387, Train Accuracy:0.3746599853038788, Test Loss:1.7865382432937622, Test Accuracy:0.37059998512268066
Epoch:4/20, Train Loss:1.71498703956604, Train Accuracy:0.41273999214172363, Test Loss:1.667392611503601, Test Accuracy:0.41370001435279846
Epoch:5/20, Train Loss:1.630845069885254, Train Accuracy:0.4369800090789795, Test Loss:1.5599491596221924, Test Accuracy:0.4456999897956848
Epoch:6/20, Train Loss:1.5676977634429932, Train Accuracy:0.45875999331474304, Test Loss:1.4827308654785156, Test Accuracy:0.46869999170303345
Epoch:7/20, Train Loss:1.5256495475769043, Train Accuracy:0.4784800112247467, Test Loss:1.4289569854736328, Test Accuracy:0.491100013256073
Epoch:8/20, Train Loss:1.4906647205352783, Train Accuracy:0.49744001030921936, Test Loss:1.3841782808303833, Test Accuracy:0.5103999972343445
Epoch:9/20, Train Loss:1.4608360528945923, Train Accuracy:0.513700008392334, Test Loss:1.345192551612854, Test Accuracy:0.5256999731063843
Epoch:10/20, Train Loss:1.440119981765747, Train Accuracy:0.5293999910354614, Test Loss:1.3107739686965942, Test Accuracy:0.5408999919891357
Epoch:11/20, Train Loss:1.3959506750106812, Train Accuracy:0.5428799986839294, Test Loss:1.2769654989242554, Test Accuracy:0.5522000193595886
Epoch:12/20, Train Loss:1.3535456657409668, Train Accuracy:0.5551999807357788, Test Loss:1.2540701627731323, Test Accuracy:0.5609999895095825
Epoch:13/20, Train Loss:1.310051679611206, Train Accuracy:0.5671600103378296, Test Loss:1.2341703176498413, Test Accuracy:0.567799985408783
Epoch:14/20, Train Loss:1.270039677619934, Train Accuracy:0.5795400142669678, Test Loss:1.212275743484497, Test Accuracy:0.5738999843597412
Epoch:15/20, Train Loss:1.2347301244735718, Train Accuracy:0.589959979057312, Test Loss:1.1985687017440796, Test Accuracy:0.5767999887466431
Epoch:16/20, Train Loss:1.2094693183898926, Train Accuracy:0.6000999808311462, Test Loss:1.1856199502944946, Test Accuracy:0.5809999704360962
Epoch:17/20, Train Loss:1.1799081563949585, Train Accuracy:0.6090199947357178, Test Loss:1.1705816984176636, Test Accuracy:0.5889000296592712
Epoch:18/20, Train Loss:1.153638482093811, Train Accuracy:0.6180400252342224, Test Loss:1.1553884744644165, Test Accuracy:0.5939000248908997
Epoch:19/20, Train Loss:1.1256165504455566, Train Accuracy:0.6266199946403503, Test Loss:1.1451069116592407, Test Accuracy:0.5971999764442444

In Figure 4, we can see the loss change for both test and train data for second model. Let's talk about **Second Model**. The second model has the same convolution layers with the first model but it has 3 fully connected layers. In model 1, there is only one fully connected layer and its size is (32*32*32, 10). In this model, first fully connected layer size is (32*32*32, 1000), second fully connected layer size is (1000, 100) and finally, third fully connected layer size is (100, 10). To decide sizes and number of fully connected layers, we can try another numbers but here, we get this number by trying different numbers on given batch 1 training data. As we see in Figure 4, the model trains during 20 epochs and we get 0.6266199946403503 train accuracy and 0.5971999764442444 test accuracy.

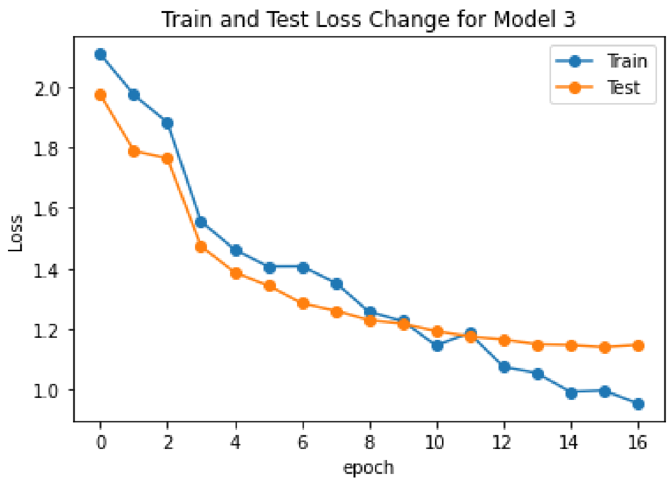

Figure 5: Three Layer CNN using convolution operation, activation function, 1 fully connected layer and dropout for each layer with SGD optimizer

Third Model is the same with first model except dropout in each layer. Dropouts has the probability 0.01. From the knowledge in literature, we expect that the dropout in the layers enables us to better model performance. The model trains until 16th epoch and after this epoch, test loss starts to increase and early stopping precedure works. We get 0.6669600009918213 train accuracy and 0.607200026512146 test accuracy. The accuracies of this model are too close to first model's. Here, couldn't see the contribution of dropouts. It may be stem from our learning rate value or optimizer. Also, this may be due to the value of the probability for dropout. We select to delete 1 percent after activation function for each layer. Maybe we can increase this possibility to see the affect of dropout clearly.

TABLE 3: Model 3 Results
Epoch:0/20, Train Loss:2.1100962162017822, Train Accuracy:0.20734000205993652, Test Loss:1.9756795167922974, Test Accuracy:0.301800012588501
Epoch:1/20, Train Loss:1.9747323989868164, Train Accuracy:0.3413800001144409, Test Loss:1.7895599603652954, Test Accuracy:0.3626999855041504
Epoch:2/20, Train Loss:1.8829058408737183, Train Accuracy:0.4172399938106537, Test Loss:1.7647377252578735, Test Accuracy:0.3772999942302704
Epoch:3/20, Train Loss:1.5558559894561768, Train Accuracy:0.4700999855995178, Test Loss:1.4739551544189453, Test Accuracy:0.47119998931884766
Epoch:4/20, Train Loss:1.462608814239502, Train Accuracy:0.5015599727630615, Test Loss:1.3871097564697266, Test Accuracy:0.4999000132083893
Epoch:5/20, Train Loss:1.4066340923309326, Train Accuracy:0.5234799981117249, Test Loss:1.3436051607131958, Test Accuracy:0.5189999938011169
Epoch:6/20, Train Loss:1.4070138931274414, Train Accuracy:0.5425800085067749, Test Loss:1.2851042747497559, Test Accuracy:0.5432999730110168
Epoch:7/20, Train Loss:1.3536032438278198, Train Accuracy:0.5610399842262268, Test Loss:1.2607226371765137, Test Accuracy:0.5490000247955322
Epoch:8/20, Train Loss:1.2572872638702393, Train Accuracy:0.5748400092124939, Test Loss:1.2296266555786133, Test Accuracy:0.5673999786376953
Epoch:9/20, Train Loss:1.2262240648269653, Train Accuracy:0.592199981212616, Test Loss:1.2175521850585938, Test Accuracy:0.5692999958992004
Epoch:10/20, Train Loss:1.1458054780960083, Train Accuracy:0.6057000160217285, Test Loss:1.1925276517868042, Test Accuracy:0.5796999931335449
Epoch:11/20, Train Loss:1.1861652135849, Train Accuracy:0.6185600161552429, Test Loss:1.1751407384872437, Test Accuracy:0.5871999859809875
Epoch:12/20, Train Loss:1.0752471685409546, Train Accuracy:0.6337800025939941, Test Loss:1.1654484272003174, Test Accuracy:0.5945000052452087
Epoch:13/20, Train Loss:1.0542963743209839, Train Accuracy:0.6460999846458435, Test Loss:1.1497102975845337, Test Accuracy:0.5990999937057495
Epoch:14/20, Train Loss:0.991759181022644, Train Accuracy:0.6573399901390076, Test Loss:1.147274374961853, Test Accuracy:0.6031000018119812
Epoch:15/20, Train Loss:0.9959551095962524, Train Accuracy:0.6669600009918213, Test Loss:1.1396251916885376, Test Accuracy:0.607200026512146

Also, we can see the results for **Forth Model** in Figure 6. This model has the same convolution layers with the first model except batch normalization. The model has batch normalization after convolution operation and ReLu activation function for each layer. It arrive very quickly early stopping point. The model stops at 3rd epoch. As a result, we get 0.7610200047492981 training accuracy and 0.604200005531311 test accuracy. Also, the losses are smaller than other models. Although it stops very quickly, it gives us to better performance and we get more effective model with batch normalization. In the Figure 7, we can see effect of batch normalization clearly. Batch normalization enables us to work with higher learning rates. If we try to train the model with higher learning rate, we may get better result. We will try to train this model with Adam and Adagrad optimizers. Therefore, we will compare optimizer effects. We will use the best of them to plot tsne function to get classes.

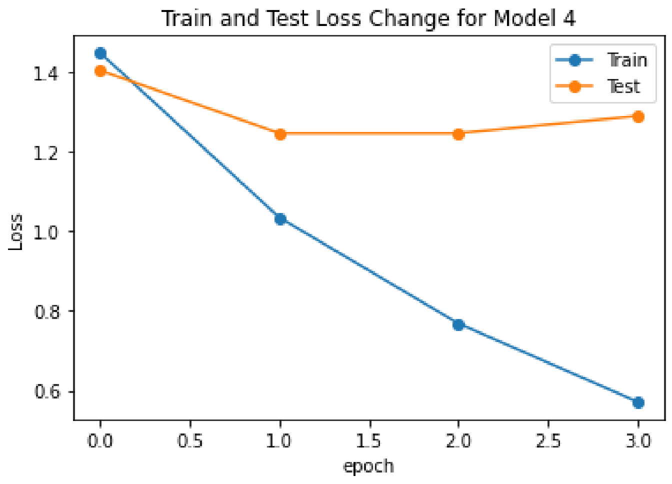

Figure 6: Three Layer CNN using convolution operation, activation function, 1 fully connected layer and batch normalization for each layer with SGD optimizer

TABLE 4: Model 4 Results with SGD Optimizer
Epoch:0/20, Train Loss:1.4344650506973267, Train Accuracy:0.47082000970840454, Test Loss:1.3455121517181396, Test Accuracy:0.572700023651123
Epoch:1/20, Train Loss:0.9825313091278076, Train Accuracy:0.6595199704170227, Test Loss:1.2123373746871948, Test Accuracy:0.6119999885559082
Epoch:2/20, Train Loss:0.7300711870193481, Train Accuracy:0.764959990978241, Test Loss:1.2116796970367432, Test Accuracy:0.6179999709129333

When we train the model 4 with Adam optimizer, we get 0.4628799855709076 training accuracy and 0.47510001063346863 test accuracy. When we compare the results with SGD optimizer, we can say that SGD optimizer enables us to better model performance. We can also see loss change for model 4 with Adam optimizer in Figure 7. Similarly, in Figure 8, we can see the results for Adagrad optimizer. The best of them is SGD optimizer with 0.01 learning rate.

TABLE 5: Model 4 with Adam Optimizer
Epoch:0/20, Train Loss:1.7250549793243408, Train Accuracy:0.33254000544548035, Test Loss:1.6637271642684937, Test Accuracy:0.4043999910354614
Epoch:1/20, Train Loss:1.5318048000335693, Train Accuracy:0.43022000789642334, Test Loss:1.549291968345642, Test Accuracy:0.4465000033378601
Epoch:2/20, Train Loss:1.4064948558807373, Train Accuracy:0.4628799855709076, Test Loss:1.4795496463775635, Test Accuracy:0.47510001063346863

TABLE 6: Model 4 with Adagrad Optimizer
Epoch:0/20, Train Loss:2.4717681407928467, Train Accuracy:0.4072999954223633, Test Loss:2.431269645690918, Test Accuracy:0.4860999882221222
Epoch:1/20, Train Loss:1.6963026523590088, Train Accuracy:0.5259600281715393, Test Loss:1.8417644500732422, Test Accuracy:0.529699981212616
Epoch:2/20, Train Loss:1.6952242851257324, Train Accuracy:0.5768600106239319, Test Loss:1.6705304384231567, Test Accuracy:0.551800012588501
Epoch:3/20, Train Loss:1.355155348777771, Train Accuracy:0.6117799878120422, Test Loss:1.6106351613998413, Test Accuracy:0.5667999982833862
Epoch:4/20, Train Loss:1.2301480770111084, Train Accuracy:0.6495199799537659, Test Loss:1.5635632276535034, Test Accuracy:0.5781999826431274
Epoch:5/20, Train Loss:1.0748863220214844, Train Accuracy:0.6776800155639648, Test Loss:1.4420300722122192, Test Accuracy:0.582099974155426

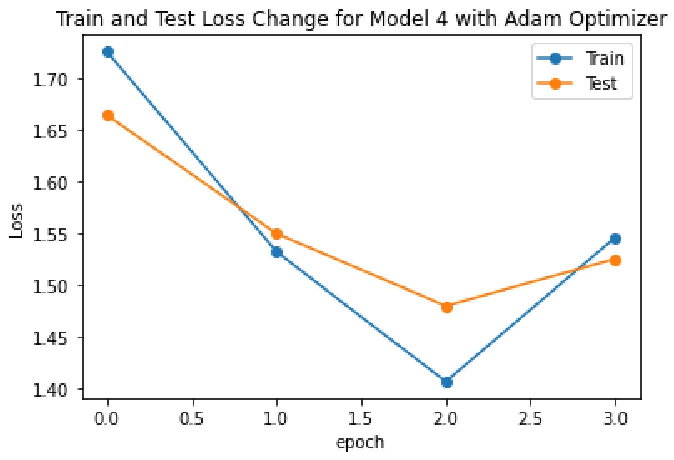

Figure 7: Three Layer CNN using convolution operation, activation function, 1 fully connected layer and batch normalization for each layer with Adam optimizer

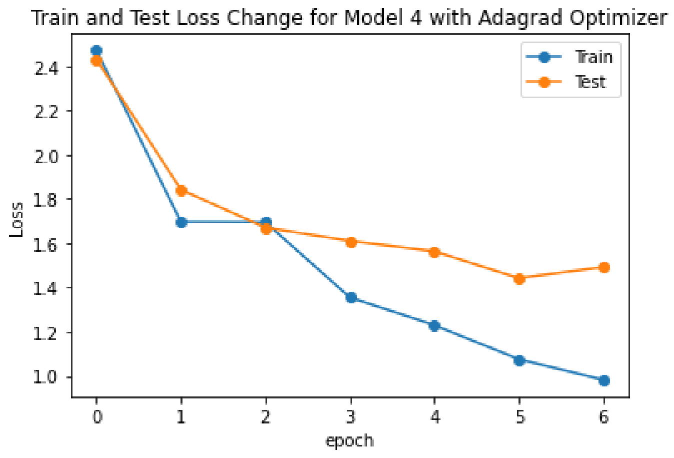

Figure 8: Three Layer CNN using convolution operation, activation function, 1 fully connected layer and batch normalization for each layer with Adagrad optimizer

Let's summarize the results until this time. We use four different models. First model uses convolution operation and activation function for each layer. The second model is to compare what kind of result we will get when we increase the number of fully connected layers. It is basically the same as the first model, only the 2nd model has 3 fully connected layers. Also, third model use first model as base model. The third model additionally has dropouts between layers. Similarly, forth model has batch normalization between layers. At the end of the models training, model 4 gives us to better result. It achieves 76 percent training accuracy after 3 layers. Therefore, we also use model 4 to compare different optimizer effects. Therefore, we get the better model with SGD optimizer. However, the forth model stops after 3 epochs since early stopping procedure. To solve this problem, we can use higher learning rate and then we can train the model again.

In Figure 9, we can see tsne plot in the beginning of the training. It is difficult to classify the groups clearly. This plot is after only 1 epochs. Here, we use model 4 to get tsne plot and the model hyperparameters are the same. Because we know the forth model stops at 4th epochs, we train it with 4 epochs. Therefore, we get tsne plot for each epoch.

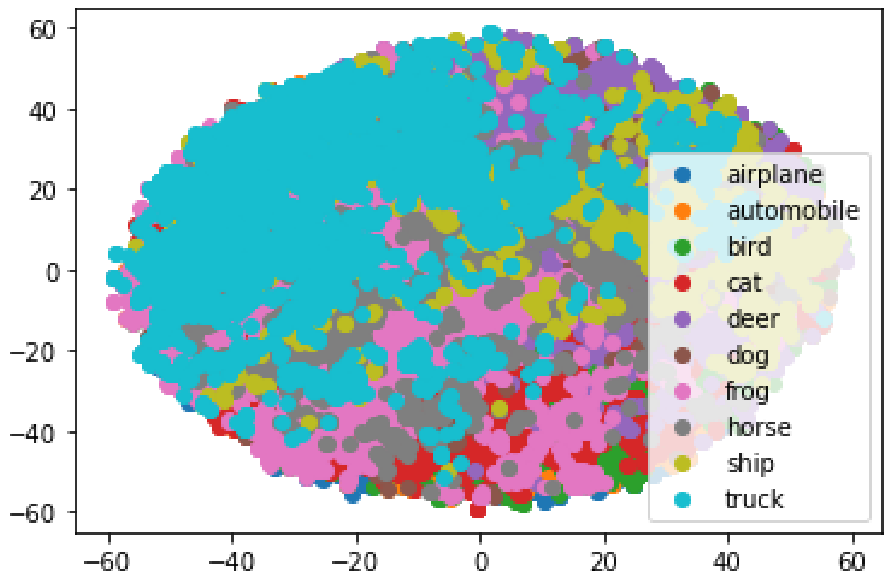

Figure 9: Tsne Plot in the Beginning of the Training (after 1 epoch)

Figure 10, Figure 11 and Figure 12 are tsne plots after 2, 3 and 4 epochs respectively. We can say that we see the classes clearly in Figure 12. For why, the model learns a lot after 4 epochs.

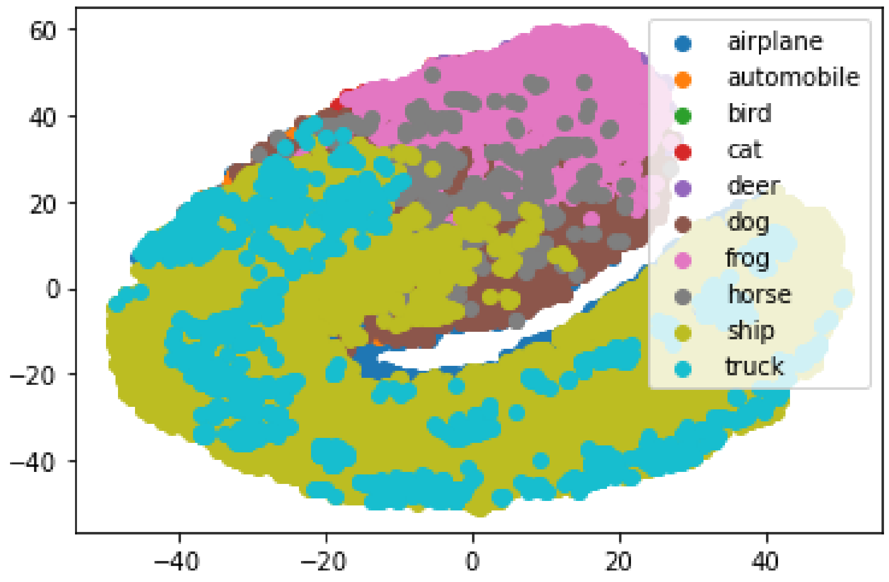

Figure 10: Tsne Plot in the Middle of the Training (after 2 epoch)

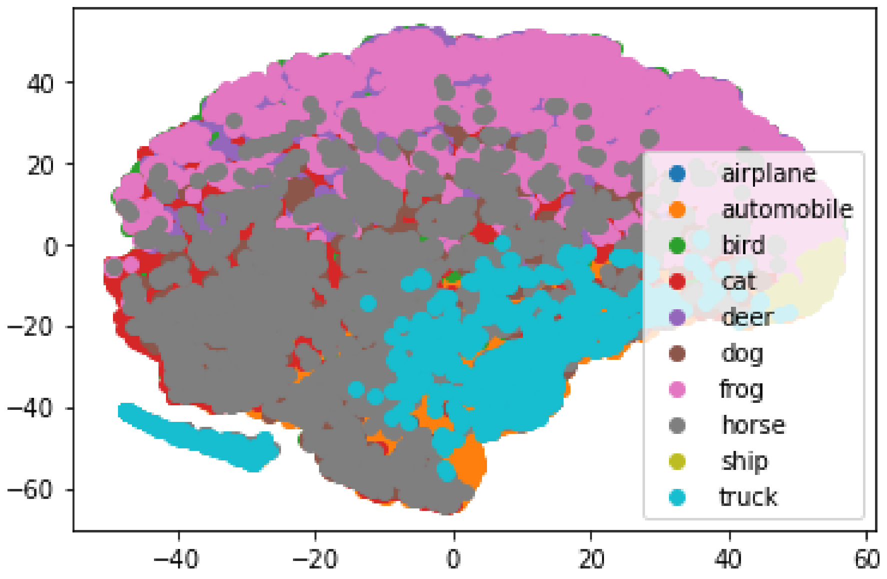

Figure 11: Tsne Plot in the Middle of the Training (after 3 epoch)

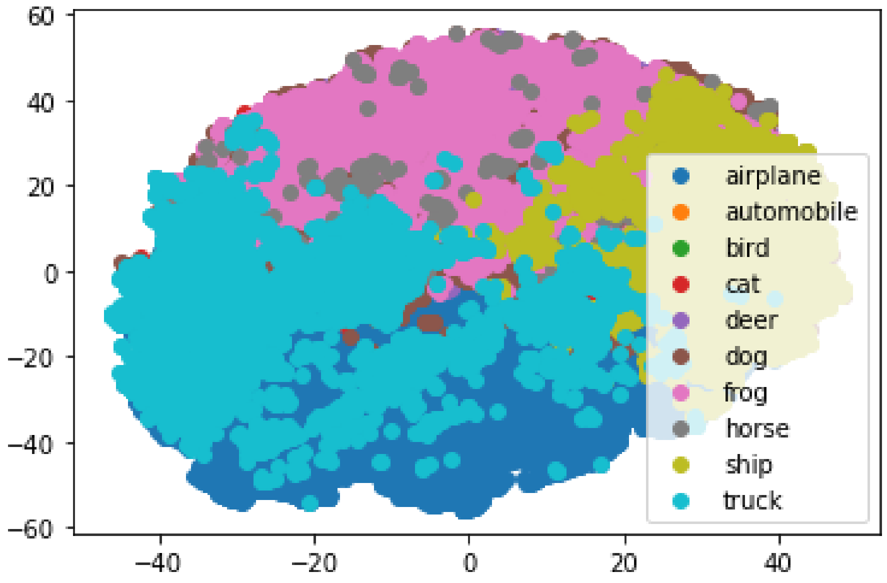

Figure 12: Tsne Plot in the End of the Training (after 4 epoch)

REFERENCES

- Agarap, A. F. (2018). Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375.
- Cataltepe, Z., Abu-Mostafa, Y. S., & Magdon-Ismail, M. (1999). No free lunch for early stopping. Neural computation, 11(4), 995-1009.
- Fu, J. & Ng, R. (2019). Deep Learning Wizard. Doi: 10.5281/zenodo.1480879. https://zenodo.org/record/2644957#.YLAJ3i0Rrq0
- Li, X., Chen, S., Hu, X., & Yang, J. (2019). Understanding the disharmony between dropout and batch normalization by variance shift. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2682-2690.
- Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., & Li, Z. (2019). Darts+: Improved differentiable architecture search with early stopping. arXiv preprint arXiv:1909.06035.
- Perez, L., & Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.
- Sun, J., Cao, X., Liang, H., Huang, W., Chen, Z., & Li, Z. (2020). New interpretations of normalization methods in deep learning. In Proceedings of the AAAI Conference on Artificial Intelligence 34(04), 5875-5882.
- Thakkar, V., Tewary, S., & Chakraborty, C. (2018). Batch Normalization in Convolutional Neural Networks—A comparative study with CIFAR-10 data. In 2018 fifth international conference on emerging applications of information technology (EAIT), IEEE, 1-5.
- Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530.
- https://gradientscience.org/batchnorm/
- https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
- https://zhenye-na.github.io/2018/09/28/pytorch-cnn-cifar10.html
- https://blog.paperspace.com/pytorch-101-building-neural-networks/
- https://heartbeat.fritz.ai/basics-of-image-classification-with-pytorch-2f8973c51864
- Bjarte Mehus Sunde (2019) https://github.com/Bjarten/early-stopping-pytorch
- https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_convolutional_neuralnetwork