Mini-Project 3: Classification of Image Data

Hong Kun Tian
hong.tian@mail.mcgill.ca

Dima Romanov
dmytro.romanov@mail.mcgill.ca

Jia R. Shao
jia.r.shao@mail.mcgill.ca

April 19, 2020

Abstract

In this report we aim to thoroughly explain two image classification models we have developed: the multilayer perceptron and the convolutional neural network. We use each of these models with the CIFAR-10 dataset and its default test and train partitions [2]. Below we document the train and test performances of the models based on the number of training epochs, the total number of parameters, and using different pre-processing techniques such as grayscaling. Our results indicate that with transforms, CNN achieved 88% accuracy, which corresponds to a 7% improvement from the tests run without transforms. The same preprocessing that made our CNN model successful did not translate to our MLP model, achieving an accuracy of 50%.
Keywords: *Artificial neural networks, multilayer perceptron, image classification*

1 Introduction

The multilayer perceptron was implemented from scratch. It includes a backpropagation and a mini-batch gradient descent algorithm. To tune our model, we did as encouraged and experimented with our choice of activation function, the number of layers considered, the number of units per layer, as well as many data pre-processing techniques (which we will explain in detail further in our report).

The convolutional neural network was built upon the foundation provided by the CNN tutorial for CIFAR-10 data [3]. We largely personalized this code by adding our own data pre-processing steps, thus dramatically improving results by approximately 10% across 1 to 20 epochs. As required by the project instructions, we have commented all functions to justify the reasoning behind these steps and to demonstrate our understanding of this model's implementation.

The CIFAR-10 dataset we study using each of these two models is loaded via PyTorch's built-in load functions. CIFAR-10 is comprised of 60,000 colour images, each being of dimensions 32x32 pixels. Each of these 60,000 images belongs to one single class among 10 (i.e. the classes are mutually exclusive), with an even distribution of 6000 images per class.

2 Related Work

In Zhu and Bain's 2017 article where they study the CIFAR-10 dataset, they use a variant of the Convolution Neural Network named B-CNN (Branch Convolutional Neural Network) [5]. Although B-CNN was intended for image classification tasks where classes are hierarchichal (contrary to the CIFAR-10 dataset, where all classes are mutually exclusive), we believed this paper offers excellent insight regarding strategies for improving the performance of traditional CNN models.

B-CNN considers the hierarchy of target classes as prior knowledge. The model outputs multiple ordered predictions then uses a Branch Training strategy to manipulate the weights attributed to the priors and to the output layers. The traditional CNN assumes it is equally difficult to distinguish all target classes from one another and outputs only one prediction. This is a major flaw of CNN, as can be seen from Appendix A where the prediction accuracy for the *cat* target class is much lower than for all other classes. With B-CNN for examle, the *cat* and *dog* classes would be grouped under a *pet* category, so the classifier would first find a coarse prediction of *pet* then make a finer prediction of *cat*. This article inspired us to experiment with pre-processing so our model will have more data to learn from, and hopefully obtain better results from a horizontal class structure.

3 Data Sets and Set-up

To maximize the accuracy we strongly focused on the preprocessing of our image data. By increasing the variation of our model, we hoped to make our models more versatile and capable of differentiating images with higher accuracy.

For both the multilayer perceptron (MLP) and the convolutional neural network (CNN), we split train and test as the *torchvision.datasets* import defines. Then, we used *SubsetRandomSampler()* to separate 20% of the training set into our validation set. On this newly defined training set, we used a few data augmentation transforms, namely *RandomCrop(), RandomHorizontalFlip()*, and *RandomRotation()* (between -20 and 20 degrees). We limited the rotations by twenty degrees to maintain the natural orientation of the images. The motivation behind this choice is that these randomized alterations will increase the amount of data we can use to train our CNN model, which in turn should decrease model variance.

The last transform we decided to use was *Grayscale*. This transform removed the color information, but at the same time, it decreased the size of data allowing us to run more tests in the same amount of time. Finally, *ToTensor()* is called, and *Normalize()* is used to normalize the images' pixel values in the range $[-1, 1]$. Normalization reduces skewness, thus making training more stable and fast [4].

4 Proposed Approach

4.1 CNN

For the CNN model we followed the PyTorch tutorial and enhanced our model's architecture by using a similar architecture to those of well known breakthrough models that are known to perform well for image classification, such as LeNet, AlexNet, and VGG-16. To better understand CNN, we experimented with early stopping and altered the code provided in the tutorial to fit our needs. This decision was motivated by us noticing that the function for validation loss monotonically increases after reaching some minimum. The early stopping mechanism allows our model to avoid making excessive, unnecessary calculations past this minimum, thus reducing training time. Our function *train_net()*, which implements said mechanism, dictates that training stops as soon as there is no improvement after ten consecutive epochs.

4.2 Multilayer Perceptron

For our multilayer perceptron, our final model consists of 3 hidden layers and an output layer. The first layer takes as input 3072 entries (number of pixels multiplied by three, channels corresponding to the RGB colors) and has an output size of 1024; next layers

consisted of 512 cells, 512 cells, and 10 cells (output). Due to the large capacity of the network, we first experimented with a smaller model that had 2 hidden layers: 1024, 128 in size. However, we had to change our layers to 300, 100 when we used grayscale pre-processing. We had to make this change since grayscale limits us to only a luminance value instead of RBG values. Therefore, the input data was three times smaller at 1024.

We knew that the multilayer perceptron would take a significant time to run. Consequently, we implemented code to process our data in mini-batches. Specifically, we ran 256 data points at the same, in every mini-batch. (128 for our experimentation model)

4.3 Evaluation

Finally, we used a validation set to test the accuracy of our model. Considering the time complexity of training the model, it was efficient to choose this method instead of cross-validation.

5 Results

5.1 CNN

In the end, we achieved 88% accuracy after running the CNN model. Using our stopping approach it took 31 epochs to reach this accuracy. Furthermore, we observed that our transforms resulted in a seven percent improvement of accuracy; the model without transforms gave us an accuracy of 81%. Additionally, we saw that early stopping mechanism was quite efficient since running one hundred epochs did not result in any improvement in accuracy of the training set and validation set.

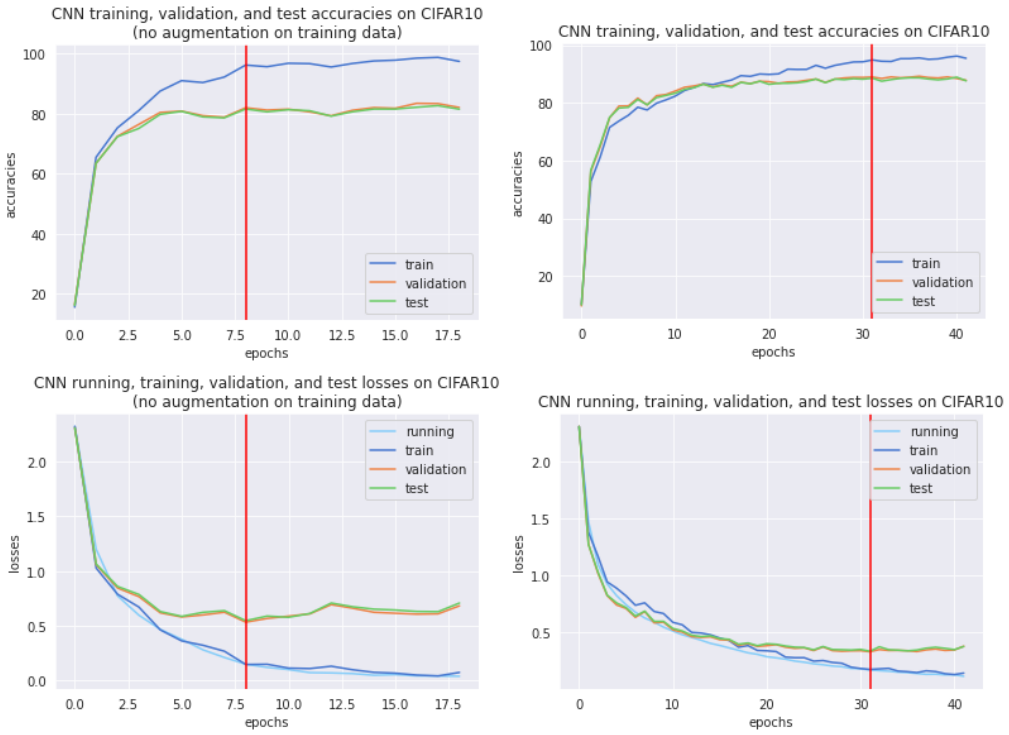

Figure 1: Visualizing and Comparing CNN Accuracies and Losses. The vertical red line segment indicates the early stopping point.

5.2 Multilayer Perceptron

With our multilayer perceptron model, we achieved a 50% accuracy on our final model. We first experimented with a simpler model with only 2 hidden layers. We found through our first model that pre-processing had negative effect on the accuracy of our model. When we only used the orientation transformation or the grayscale transformation, we saw a ten percent decline in accuracy. Moreover, when we used all of our transforms, the accuracy dropped by more than twenty percent (see Figure 2).

	Training Accuracy(%)	Validation Accuracy(%)
No transforms	38.8	36.6
Orientation transforms	29.3	26.8
Graysale transform	25.7	25.3
Grayscale and Orientation transform	18.0	16.8

Figure 2: Multilayer Perceptron results

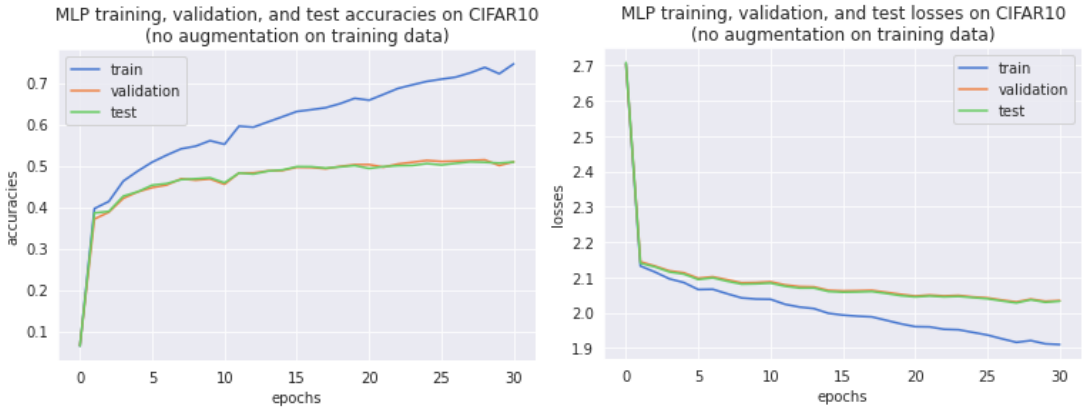

Figure 3: Visualizing and Comparing MLP Accuracies and Losses.

6 Discussion and Conclusion

In the end we observe, that CNN performed much better than multilayer perceptron. Furthermore, the transforms had a positive effect on the accuracy of CNN model, but they had the opposite effect on the MLP model.

The difference in performance between these models could be explained by the fact that CNN is much better suited for this task. The image classification implies that objects can be located at different positions, and CNN handles it much better since it is spatially invariant. On the other hand, MLP takes the pixel position into account. Consequently, it can easily overfit the data, and furthermore, it would be very sensitive to the orientation of the object.

In the future projects, it could be beneficial to obtain images of a higher resolution. This could allow our models to pick up on more subtle traits of the objects; additionally, it could make grayscale images more well-defined making it more useful in our task. Another direction which could be taken in further research could be the application of a Gabor filter [1]. It could let us mimic how human brain works, and allow us to take an advantage of pre-existing biological mechanisms.

7 Statement of Contributions

Hong Kun Tian and Dima Romanov wrote code for CNN and MLP, and conducted testing. Jia Shao wrote the report with Dima.

References

[1] KINNIKAR, A., HUSAIN, M., AND S M, M. Face recognition using gabor filter and convolutional neural network. pp. 1–4.

[2] KRIZHEVSKY, A. Learning multiple layers of features from tiny images. *University of Toronto* (05 2012).

[3] PYTORCH.ORG. Training a classifier, 2017.

[4] WANG, S. Normalization in mnist example, 2017.

[5] ZHU, X., AND BAIN, M. B-cnn: Branch convolutional neural network for hierarchical classification.

A Performance of CNN (No Augmentation) for each Target Class

```
Accuracy of the network on the 10000 test images: 81.330 %

Accuracy of plane : 82 %
Accuracy of   car : 98 %
Accuracy of  bird : 87 %
Accuracy of   cat : 41 %
Accuracy of  deer : 76 %
Accuracy of   dog : 81 %
Accuracy of  frog : 89 %
Accuracy of horse : 85 %
Accuracy of  ship : 87 %
Accuracy of truck : 87 %
```

B Performance of CNN (With Augmentation) for each Target Class

```
Accuracy of the network on the 10000 test images: 88.540 %

Accuracy of plane : 94 %
Accuracy of   car : 90 %
Accuracy of  bird : 84 %
Accuracy of   cat : 72 %
Accuracy of  deer : 85 %
Accuracy of   dog : 86 %
Accuracy of  frog : 94 %
Accuracy of horse : 87 %
Accuracy of  ship : 96 %
Accuracy of truck : 92 %
```