Module 6: Critical Thinking

Implementation of CIFAR10 with CNNs Using TensorFlow

Nolan Byrnes

CSC580 – Applying Machine Learning and Neural Networks - Capstone

Colorado State University – Global Campus

Dr. Isaac Gang

October 21st, 2023

Module 6: Critical Thinking

I developed a Convolutional Neural Network (CNN) to classify images from the CIFAR10 dataset. The CIFAR10 dataset “consists of 60000 32x32 colour images in 10 classes, with 6000 images per class” (Krizhevsky et. al, 2009, para. 3). This dataset includes ten different classes of images, which include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. This paper goes over how I trained the model and evaluates the results of training.

The training dataset had 50,000 images, whereas the test dataset contained 10,000 images. I split the test dataset in half, so that I could have 5,000 images as the validation dataset. After splitting the data, I normalized the pixel values of each of the images, so that the pixel values were between 0 and 1. Chng explains that images can be standardized by calculating “the mean and the standard deviation on the set of data and transform each sample by subtracting the mean and dividing by the standard deviation” (Chng, 2023, para. 6). Chng explains that normalization “helps to stabilize the gradient descent step” and can “help models converge faster for a given learning rate” (Chng, 2023, para. 6).

During training, I had the model go through 100 epochs, however, I provided an early stopping callback on the validation loss. This was so that training could persist until the validation loss was no longer decreasing between epochs. By monitoring the validation loss, it could prevent the model from becoming overfit. The training ended up stopping at epoch 14. You can view the loss, accuracy, validation loss, and validation accuracy of the final epoch in figure 1 below.

Figure 1.

Training Epochs

A screen shot of a computer

Description automatically generated

Note. This figure displays the epochs the model went through during training, displaying the loss and accuracy of the training and validation data sets at the end of each epoch.

After training ended, I mapped out the history of the training, including the accuracy of the training data as well as the validation data. Brownlee defined Cross-entropy as “a measure of the difference between two probability distributions for a given random variable or set of events” and is “widely used as a loss function when optimizing classification models” (Brownlee, 2020, para. 8, 64). Since the predictions of the model would give us the probabilities of how confident it is that an image belongs to the different categories, we want its predictions to become more confident.

By using accuracy, it only tells us if the model got the answer correct or not, but by using entropy, it tells us how correct the predictions were. For example, if the model predicted an image to have an airplane with a 50% confidence, or another image with an 80% confidence, the accuracy metric would tell us both predictions are correct, however, loss would give us a score that is an indicator on how correct the predictions are. The goal is to minimize the loss.

I had the model have an early stopping callback on the validation loss because I wanted to have it so that the model would quit before it becomes overfit. The validation loss was calculated at the end of each epoch with a validation data set, which was not seen by the model when training through the epoch. This would represent data that the model would encounter in real-world scenarios. In figure 2 below, you will see how both the loss and the validation loss kept on decreasing, but the validation loss started to level out. If we continued to have the model train past the 14th epoch, the model could become overfit, as the loss would continue to increase, where the validation loss could have continued to move in a horizontal fashion, or even begin to increase. We stopped training before the model became overfit with the training data, by paying close attention to the validation loss, and stopping when the model was no longer decreasing loss on the validation data set, which is a representative of what it would see in the real world.

Figure 2.

Best performing hyperparameter combinations

A graph with blue and orange lines

Description automatically generated

Note. This figure displays the top three combinations of hyperparameter values with the highest average accuracy rate of the validation dataset.

At the end of the 14th epoch, there was a 75% accuracy rate on the test data set, and an 79.8% accuracy rate on the training data set. Observing the training history on the accuracy and validation accuracy in figure 3 below, you can see that the model’s validation accuracy and training accuracy was still close together, not overfit, and with a high accuracy rate, indicating that the model is accurate, without being overfit.

Figure 3.

TensorBoard Time Series Graph of Best Hyperparameter Combination

A graph with a line

Description automatically generated

Note. This figure represents the loss throughout training for the three trials of training with the best performing hyperparameter combination.

Next, I needed to evaluate how the model would perform with unseen data, so I evaluated the test data set. Since the model has not seen the test data set during training at all, it is a good representation of how the model would perform in the real world. The loss of the test data at the 14th epoch came out to be .778, which was similar to the validation loss, which was .7575. This suggests that the model could be well generalized in that it should perform similarly on unseen data as it did with the validation data. You can view the loss and accuracy of the evaluated test data set in figure 4 below.

Figure 4.

Loss and Accuracy of the Test Dataset



Note. This figure displays the accuracy and loss that the model had when evaluating the Test Dataset.

Since all the image data was normalized before training the dataset, the data that is being sent through the model when evaluating and predicting must also be normalized in the same fashion. To display the images after having the model predict the values, we must renormalize the pixel values back to their original state. Since we normalized the pixel data by subtracting the mean and dividing it by the standard deviation, the inverse is applied so that the images can be viewed in their original formatting. You can view a sample of 25 images of the test dataset, along with the predicted and actual values in figure 5 below.

Figure 5.

Sample images of the Test dataset with models’ predictions

A collage of images of animals and cars

Description automatically generated

Note. This figure displays a sample of 25 images from the test dataset, along with the predicted and actual categories.

Conclusion

When we are training a model to solve a multi-classification problem, loss is a great metric to pay attention to, as it tells us how right the model is in its predictions, rather than just if it made the prediction correct. I had an early stopping mechanism in place on the validation loss, so that the training would end when validation loss was no longer decreasing. As a result, we saw that the model was not overfit, and it achieved a similar loss on the test data set, which is a sign that the model would be consistent with its predictions with unseen data, while having an accuracy rate of 74.4% on the test data. Considering that there are a total of ten categories each image could represent, the model performed well with unseen data.

Normalizing the data as part of preprocessing helped the model train easier, since each of the pixel values were translated to be between the values of 0 and 1, which helps the model during gradient decent, and helps the model converge faster. Since we normalized the pixel values of the images as part of preprocessing, we had to change the pixel values back to their original values for us to view the images.

**REFERENCES**

Brownlee, J. (2020, December 20). A Gentle Introduction to Cross-Entropy for Machine Learning. Machine Learning Mastery. <https://machinelearningmastery.com/cross-entropy-for-machine-learning/>

Chng, Z. M. (2022, June 20). Using Normalization Layers to Improve Deep Learning Models. Machine Learning Mastery. <https://machinelearningmastery.com/using-normalization-layers-to-improve-deep-learning-models/>

Krizhevsky, A., Nair, V., & Hinton, G. (2009). CIFAR-10 and CIFAR-100 datasets. Department of Computer Science, University of Toronto. <https://www.cs.toronto.edu/~kriz/cifar.html>