# Convolutional Neural Network

**Donggyu Kim**

## Introduction

This report details the creation and evaluation of a Convolutional Neural Network (CNN) designed to classify images from the CIFAR-10 dataset, an established benchmark in machine learning. The CIFAR-10 dataset comprises 60,000 32x32 color images in 10 distinct classes, representing a diverse collection of objects and animals. The objective of this assignment was to develop a CNN that can accurately recognize and categorize these images, a task that simulates the challenges faced in real-world image recognition applications.

The programming assignment involved several stages: data preprocessing, model architecture design, training the model, and finally, testing its performance. Through this process, the aim was to not only achieve high accuracy in classification but also to gain deeper insights into how CNNs function and why they are suited for image recognition tasks. The report encapsulates the methodology adopted, the challenges encountered, and the solutions implemented, thus serving as a comprehensive account of the journey from conception to realization of the CNN model for CIFAR-10.

## Experimental setup

I used the spyder IED to run the pytorch and python libraries. The version of pytorch was 2.1.2 with python 3.10.8 and the version of cuda was 11.8 for this programming assignment. To run the simulation of the learning, I used the laptop with CPU i7-8750 and GPU RTX 2070 max_Q(for laptop).

## Hyper Parameters and Model description

There are several types of hyper parameters in this model.

1. Learning rate: For this model, I adjusted the learning rate by observing and analyzing the graph of inaccuracy over epochs. to prevent the overfitting, Learning rate was setted to 0.01 starting from 0.05

2. Number of epoch: I let the model train for 100 epochs. As the epoch value was adjusted from 250, it was confirmed that overfitting occurred as the difference between training and testing results increased when it exceeded 100.

3. Weight and Bias size, drop out, pooling window and Norm(regularization) initialization:
   Fully connected layer weight and bias initialization
   a. The weight of pytorch in linear initialized with the learnable weights of the module of shape(out_features,in_features). The values are initialized from -root(k) to root(k) where k = 1/in_feature

b. The bias of of pytorch in linear initialized with the learnable weights of the module of shape(out_features).The values are initialized from -root(k) to root(k) where k = 1/in_feature

c. Max pooling is applied between each convolution layers with 2 × 2 window and stride of 2

d. Drop out is applied after and before first hidden layer of fully connected layer using nn.Dropout(0.1). Starting at 0.5, it was adjusted to 0.1 to prevent overfitting without losing the characteristics of the data

Convolution layer kernel, bias, drop out and norm initialization

a. The learnable weights of the module of shape (out_channels, in_channels, Kernel_size, Kernel_size) The value of these weights are sampled form -root(k) to root(k) where k = 1/channel*Kernel_size^2

b. The learnable bias of the module of shape (out_channels). The value of these weights are sampled form -root(k) to root(k) where k = 1/channel*Kernel_size^2

c. The Batch norm is applied in convolution layer using pytorch.BatchNorm2d(). The number of feature is same as out channel of convolution layer. And this regulization is applied for each layer in convolution layer to prevent overfitting.

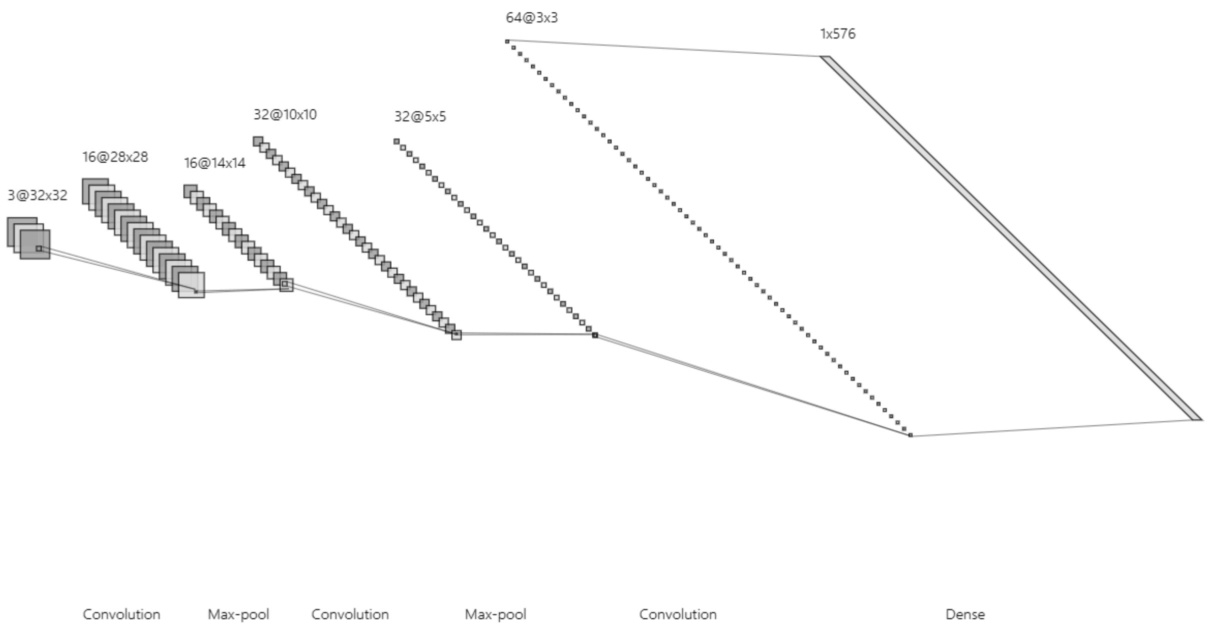
d. Drop out is applied after second layer of convolution layer using nn.Dropout2d(0.05). Starting at 0.5, it was adjusted to 0.05 to prevent overfitting without losing the characteristics of the data

4. Mini Batch Size:

   a. The mini Batch size of this model is 200 which implies there are 250 iteration for each epoch. I tried various sizes such as 50,100,500, but 250 showed the most accurate result

## Model description

The model of this assignment is consist of 2 main parts which are the convolution layer and Fully connected layer. For Convolution layer part, There are 3 convolutional layers with 2 max pooling operation layer. For Fully connected layer, There are one hidden layer and output layer



**Structure of Convolution layer**

## Specific details of each layer

First convolutional layer: This layer has 16 5 × 5 kernels with no zero padding and stride of 1. The activation function of this layer is Relu. And Batch norm with size of 16 is applied for regulization to prevent overfitting

Max-pooling layer 1: This layer has 2 × 2 window with stride of 2

Second convolutional layer: This layer has 32 5 × 5 kernels with no zero padding and stride of 1. The activation function of this layer is Relu. Drop out is applied after this layer with 0.05 to prevent overfitting of this model. And Batch norm with size of 32 is applied for regulization to prevent overfitting

Max-pooling layer 2: This layer has 2 × 2 window with stride of 2

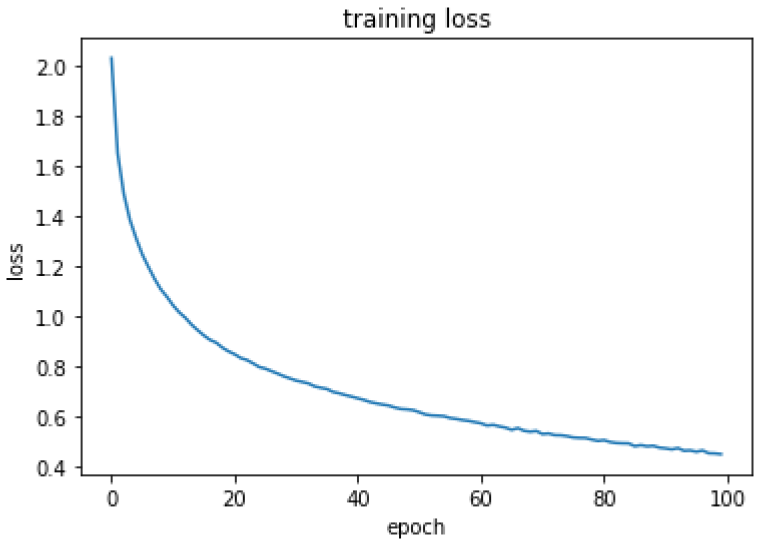
Third convolutional layer: This layer has 64 3 × 3 kernels with no zero padding and stride of 1. The activation function of this layer is Relu. And Batch norm with size of 64 is applied for regulization to prevent overfitting

Hidden layer: This layer has 576 × 500 weight matrix. 576 is flattened data size of last convolution layer. The size of bias vector is 500. The activation function of this layer is Relu. Drop out is applied after and before this layer with 0.1 to prevent overfitting of this model.

Output layer: This layer has 500 × 10 weight matrix and 10 × 1 bias vector. The Output function of this layer is softmax(torch.softmax is not applied in this model's forward function because nn.CrossEntropyLoss() function applies softmax automatically).

## Training Error, Test Error and Training loss

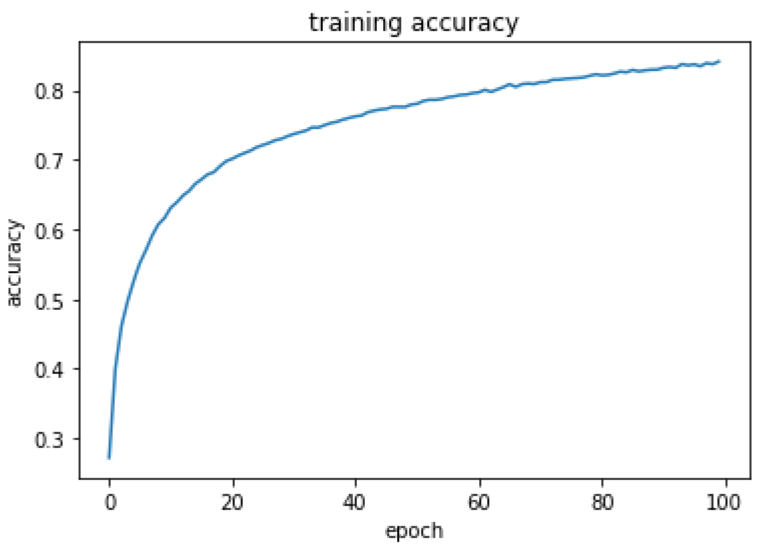
When the model was first created and learning was conducted, there was a big difference in the accuracy of the learning data and the accuracy of the test data, showing overfitting symptoms. To compensate for this, drop out, regularization, and batch size adjustment were conducted, and the following results were obtained



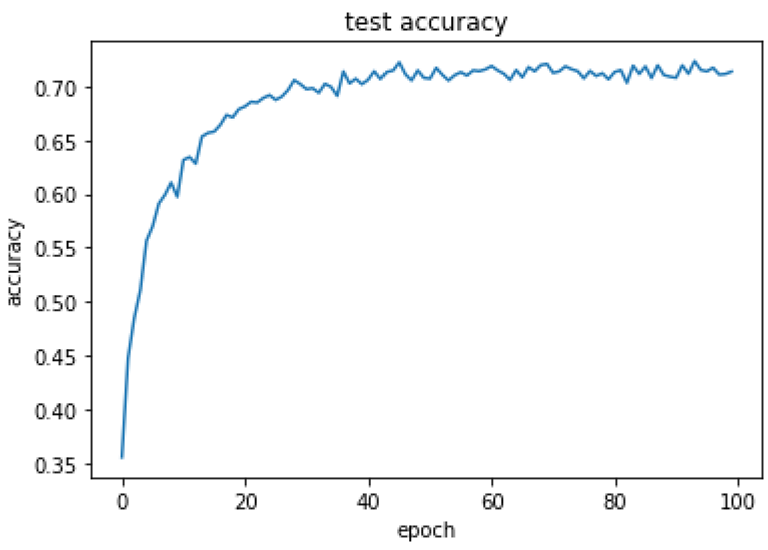
**Training loss versus epoch**

The loss starts at a high value and decreases sharply in the initial epochs, indicating that the CNN is learning quickly from the data in the early stages of training.As the epochs increase, the rate of decrease in loss slows down, with the curve beginning to flatten out. This suggests that the model is starting to converge, finding a more optimal set of weights, and extracting relevant features from the CIFAR-10 dataset.Towards the end of the training (around epoch 90 and beyond), the loss levels off, showing signs of stabilization.

## .Training Error, Test Error



**Training accuracy versus epoch**

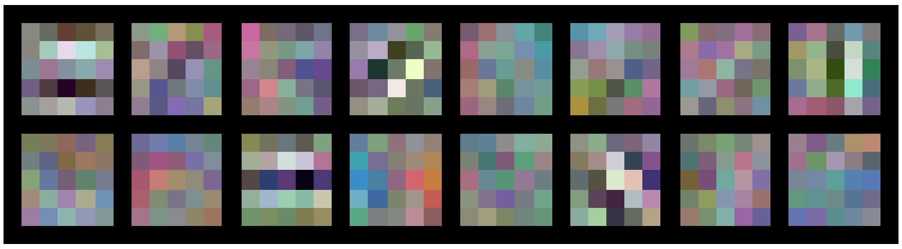


**Testing accuracy versus epoch**

The training accuracy starts at a relatively low value but rapidly increases within the first 20 epochs. This indicates that the model is initially learning and improving quickly. As the epochs continue, the accuracy incrementally improves at a slower rate. Towards the end of the 100 epochs, the training accuracy is showing a plateau, which suggests the model may be nearing its peak performance on the training data.

The test accuracy follows a similar trend to the training accuracy, with a sharp increase initially and a gradual slow down in improvement as the epochs progress.Notably, the test accuracy curve begins to plateau much earlier than the training accuracy and does so at a lower level. This is because models usually perform slightly worse on unseen data (test data) compared to data they have been trained on.There is a small gap between the training and test accuracy, indicating that while the model is generalizing well, there is a slight overfit to the training data. The model is performing better on the training data than on the test data.

## Visualization of the filters in first convolutional layer



**Visualization of the filters in first convolutional layer**

### Analysis

These filters are the first point of contact between raw pixel data and the neural network's learning process. They are typically sensitive to low-level features such as edges, textures, and color gradients. The presence of multiple colors within the filters suggests that the network has learned to detect a range of colors, which is crucial for distinguishing between classes like 'airplane' (which might have a lot of blue for the sky) and 'automobile' (which could have varied colors). However, overemphasis on color could be detrimental if the network becomes too reliant on color cues that are not consistently indicative of the class. Some filters appear to have a certain structure to them that suggests they might be good at detecting spatial arrangements in the input data, like the alignment of wheels on a vehicle or the specific shape of a wing. However, there could be a risk that the network is not capturing enough variation to deal with different orientations or scales, which could be why 'truck' and 'automobile' might be confused. There does not seem to be obvious symmetry or repetition in the filters, suggesting the network is not overfitting to specific, repeated features in the training data.

## Accuracy of testing data for each class

| Class | Accuracy |
|---|---|
| 1(Airplane) | 0.7172131147540983 |
| 2(Automobile) | 0.8871287128712871 |
| 3(Bird) | 0.587890625 |
| 4(Cat) | 0.5875251509054326 |
| 5(Deer) | 0.7061143984220908 |
| 6(Dog) | 0.6168032786885246 |
| 7(Frog) | 0.8024439918533605 |
| 8(Horse) | 0.7858585858585858 |
| 9(Ship) | 0.8353174603174603 |
| 10(Truck) | 0.7816764132553606 |
| AVG | 0.73079717319262 |

**Accuracy of testing data for each class**

The average accuracy across all classes is approximately 73.07%. This suggests that the model is fairly accurate, but there might still be room for improvement, especially for certain classes.

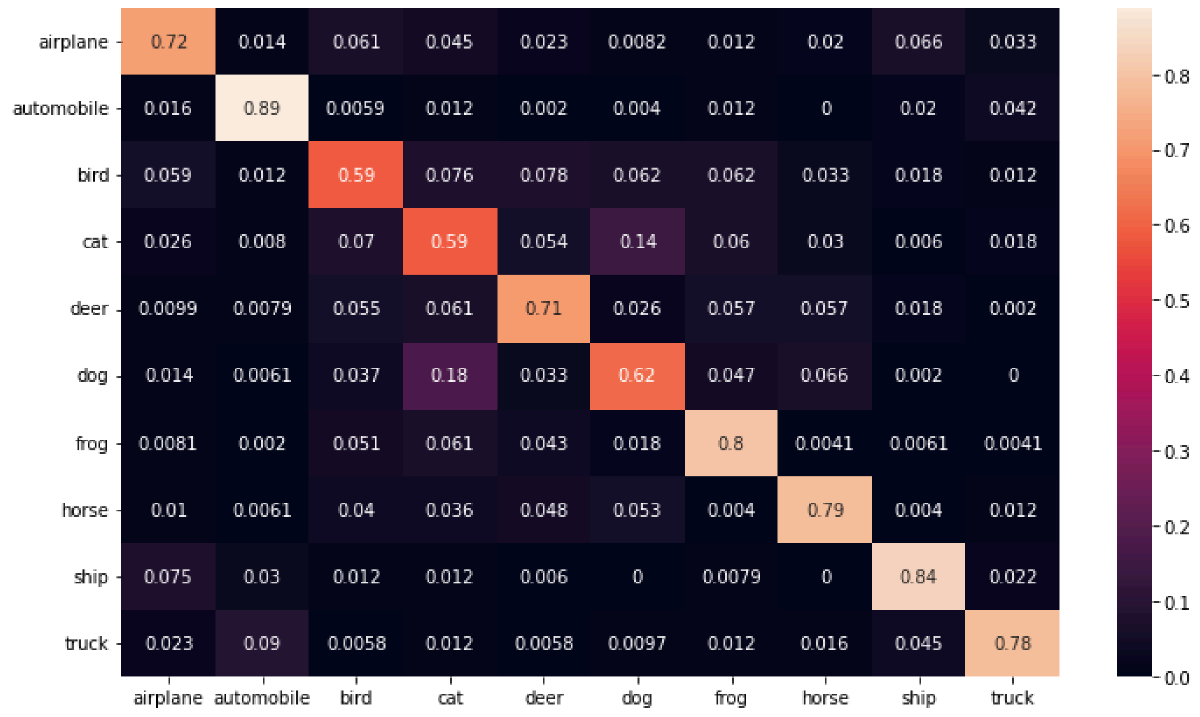
The class with the highest accuracy is the 'automobile' class with an accuracy of about 88.7%. Interestingly, this is the class with the best performance, which might indicate that the features of automobiles are distinct enough for the model to recognize them more easily compared to other classes.

The class with the lowest accuracy is the 'cat’ class with an accuracy of about 58.75%. This might suggest that the model has more difficulty distinguishing cats, possibly due to their similarity in features to other classes or less representative training data.

There is a notable variability in accuracy across different classes. For instance, 'cat' and 'bird' classes have relatively low accuracies compared to 'automobile' and 'ship'. This could be due to several reasons, such as imbalanced dataset, where some classes have more training samples than others, or it could be that some classes are inherently more difficult for the model to learn due to inter-class similarities or intra-class variability.

It is also worth considering that certain inaccuracies might be due to the model confusing similar classes, like 'truck' and 'automobile', which often look alike, or 'cat' and 'dog', which can share similar features. A confusion matrix could provide additional insight into these kinds of errors.

## Confusion matrix of classification



| | airplane | automobile | bird | cat | deer | dog | frog | horse | ship | truck |
|---|---|---|---|---|---|---|---|---|---|---|
| airplane | 0.72 | 0.014 | 0.061 | 0.045 | 0.023 | 0.0082 | 0.012 | 0.02 | 0.066 | 0.033 |
| automobile | 0.016 | 0.89 | 0.0059 | 0.012 | 0.002 | 0.004 | 0.012 | 0 | 0.02 | 0.042 |
| bird | 0.059 | 0.012 | 0.59 | 0.076 | 0.078 | 0.062 | 0.062 | 0.033 | 0.018 | 0.012 |
| cat | 0.026 | 0.008 | 0.07 | 0.59 | 0.054 | 0.14 | 0.06 | 0.03 | 0.006 | 0.018 |
| deer | 0.0099 | 0.0079 | 0.055 | 0.061 | 0.71 | 0.026 | 0.057 | 0.057 | 0.018 | 0.002 |
| dog | 0.014 | 0.0061 | 0.037 | 0.18 | 0.033 | 0.62 | 0.047 | 0.066 | 0.002 | 0 |
| frog | 0.0081 | 0.002 | 0.051 | 0.061 | 0.043 | 0.018 | 0.8 | 0.0041 | 0.0061 | 0.0041 |
| horse | 0.01 | 0.0061 | 0.04 | 0.036 | 0.048 | 0.053 | 0.004 | 0.79 | 0.004 | 0.012 |
| ship | 0.075 | 0.03 | 0.012 | 0.012 | 0.006 | 0 | 0.0079 | 0 | 0.84 | 0.022 |
| truck | 0.023 | 0.09 | 0.0058 | 0.012 | 0.0058 | 0.0097 | 0.012 | 0.016 | 0.045 | 0.78 |

**confusion matrix of classification**

The values along the diagonal from the top left to the bottom right represent the percentage of correct predictions for each class. For instance, the model predicted 'airplane' correctly 72% of the time, 'automobile' 89%, 'bird' 59%, and so on. These values are expected to be high as they indicate accurate predictions.

Values off the diagonal indicate confusion between classes. For example, 'cats' are often confused with 'dogs' (14%) and 'bird' (7%). Similarly, 'dogs' are also confused with 'cats' (18%). These confusions are understandable given that these classes can share similar features.

Some classes have low confusion with each other, such as 'frog' and 'horse' with most other classes. This suggests that the features of these classes are distinct enough for the model to identify them accurately.

In conclusion, while the model performs well for certain classes like 'automobile' and 'ship', it struggles with others like 'cat' and 'bird', ‘dog’, likely due to the visual similarities between those classes. Enhancing the training set or post-processing predictions could mitigate these confusions.

## Conclusion

In conclusion, the development and evaluation of a Convolutional Neural Network (CNN) on the CIFAR-10 dataset have provided valuable insights into the model's learning behavior and generalization capabilities. The training process displayed a healthy learning curve, with the training loss decreasing and training accuracy increasing in a consistent manner over 100 epochs. This indicates that the model was effectively learning from the training data and improving its predictive capabilities.

The test accuracy closely mirrored the training accuracy trends, though it plateaued at a lower level, which is common in machine learning models due to the inherent differences between seen (training) and unseen (testing) data. The model generalized well to the test data, suggesting that the features learned from the CIFAR-10 dataset were robust and transferable to new data samples.

Despite these successes, there remains a gap between training and test performance, which opens up opportunities for further model optimization. Techniques such as data augmentation, hyperparameter tuning, regularization could potentially bridge this gap, leading to even higher accuracy and better model performance.

Moreover, the analysis of the confusion matrix and the examination of individual class performance could guide targeted improvements in the training process, possibly through class-specific data enhancements or cost-sensitive learning strategies.

This project has reinforced the effectiveness of CNNs in handling complex image classification tasks and demonstrated the importance of a methodical approach to training and evaluating deep learning models. The skills and knowledge gained from this assignment lay a foundation for future projects in computer vision and beyond, highlighting the potential of CNNs in advancing the field of AI.