CSCI 315 - CNN Model

Professor Cody Watson
Fady Gouda and Abhi Jha
Project 5

Introduction

In this assignment, we built a convolutional neural network (CNN) model to classify images from the CIFAR dataset. In order for the model to execute and produce robust results, we loaded the dataset and performed simple preprocessing of data. We then built a model, using keras' Sequential method, with multiple layers so that the images could be understood, or 'seen', accurately by the model. We also changed the features of the images from the CIFAR dataset. Those images were then fed into the model for it to train. Finally, we evaluated the results of the model, specifically, we looked into the accuracy and loss over time for the training data. All of these steps are described in detail below.

We had to utilize libraries such as numpy, pandas, keras, and tensorflow in order to implement our strategy. We also accessed the documentation of each of these modules on the internet to understand their functionality.

Elements of Learning

Input
The input in this case are the images from the CIFAR10 dataset. Specifically, the dataset contains 60,000 images of dimension 32x32. There are 10 classes as the output, and there are 6000 images for each class.

Output
For each of the images from the input, it can be classified into ten distinct classes. These ten distinct classes are the output. The ten classes of the output are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Target Function
The target in this case is f: X -> Y. We know that X are the images and Y are the classes that are associated with the images. So, we want a function f such that it can correctly identify the class object of the image.

Data
The CIFAR10 dataset has 60,000 images, which are the x values, and its associated y values are the object classes that the images fall into.

Hypothesis

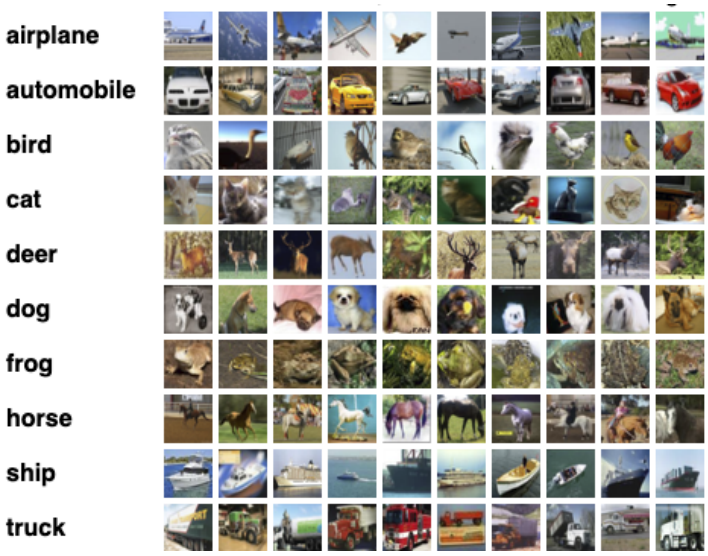
The hypothesis set is the set of all possible functions. The hypothesis set comprises functions that can be represented by the weights the model chooses for each epoch. Our goal is to select the function from this set that best approximates the target function in order to predict the most accurate values of Y for our test set.

Code Functionality

Our codebase can be divided into several parts: retrieving the data, building the model, image data generation, and lastly, evaluation. We will delve into each part in detail.

Retrieving the data
For this assignment, we used the CIFAR10 dataset. The CIFAR10 dataset contains 60,000 32x32 images with 6000 images per class. Furthermore, the ratio of training and testing images is 5:1. According to the CIFAR website, the ten classes of images are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. We used keras to load the cifar dataset by simply using the load_data method. This method split the dataset into training and testing sets for us. Since the images contain RGB pixel values that range from 0 - 255, which we know the model has a harder time processing, we normalized the pixel values to be between 0 and 1. We also one-hot encoded the y training and testing sets by using the to_categorical API from keras with num_classes parameter set to 10.

Here is an example of images provided by the official authors of the CIFAR10 dataset:



Building the model
We first initialized the keras Sequential model, after which we began adding layers. We will get into the details of each layer below:

Conv2D:
We use the Conv2D layer to allow the deep learning model to use the kernel (or filter / windows) to ‘see’ the images. The reason why we don’t reshape and flatten the pixels is because the Conv2D layer is able to analyze the spatial relationship of the images as well, unlike other dense layers. We start with 32 filters with the kernel size of (3, 3). What this means is that the model is going to use each filter to understand a feature of the image and each filter is of the size three by three pixels. So the filter is going to look at 9 pixels of the image to understand and distinguish between the different features in the image. Furthermore, we use the relu activation function and he_uniform kernel initializer. The relu activation function adds non-linearity to the values whereas the he_uniform makes sure that during initialization of the kernel, the values are drawn from a specified range that depends on input units on the weight tensor.

Batch Normalization
As mentioned above, we use the relu activation function in the Conv2D layer. The activated values are then batch normalized in order to maintain the standard deviation. The distribution of the data going through each layer is important for effectively reaching the minimum gradient point. Even when the data is normalized, throughout the iterations, when the weights are updated, the updated weights could change the distribution of the activated data. This phenomenon has been termed ‘internal covariate shift.’ Batch normalizing achieves to maintain the standard distribution of data by normalizing a batch of activated values at each layer. This makes learning faster and more efficient.

Max Pooling 2D
The Conv2D layer essentially creates a feature map describing the features of the image. The problem with that representation is that the values are precise on the feature map and any deviation from what the feature maps present could lead to misclassification. So, we need to find a way to make sure that deviations as long as they aren’t drastic don’t lead to misclassification. One of the ways of doing that is by adding the Max Pooling layer. While there are different ways of pooling, we use the max pooling to highlight the most prevalent features in the image as opposed to the average of the features.

Dropout
The dropout layer is implemented in order to prevent overfitting. The dropout layer sets some of the input values to zero with the frequency of rate. For the values that are not set to zero, they are scaled by 1/1-rate, such that the sum of all input values remains the same. The input parameter in our case is 0.2.

We repeat the use of the layers mentioned above in that sequence three times, where each time we are doubling the number of filters used in the Conv2D layer. So for each iteration, the analysis of image features becomes much more detailed. Finally, we flatten the data and feed it through a dense layer so that all the pixel values are connected and form one coherent image from all the feature maps generated by the Conv2D layers.

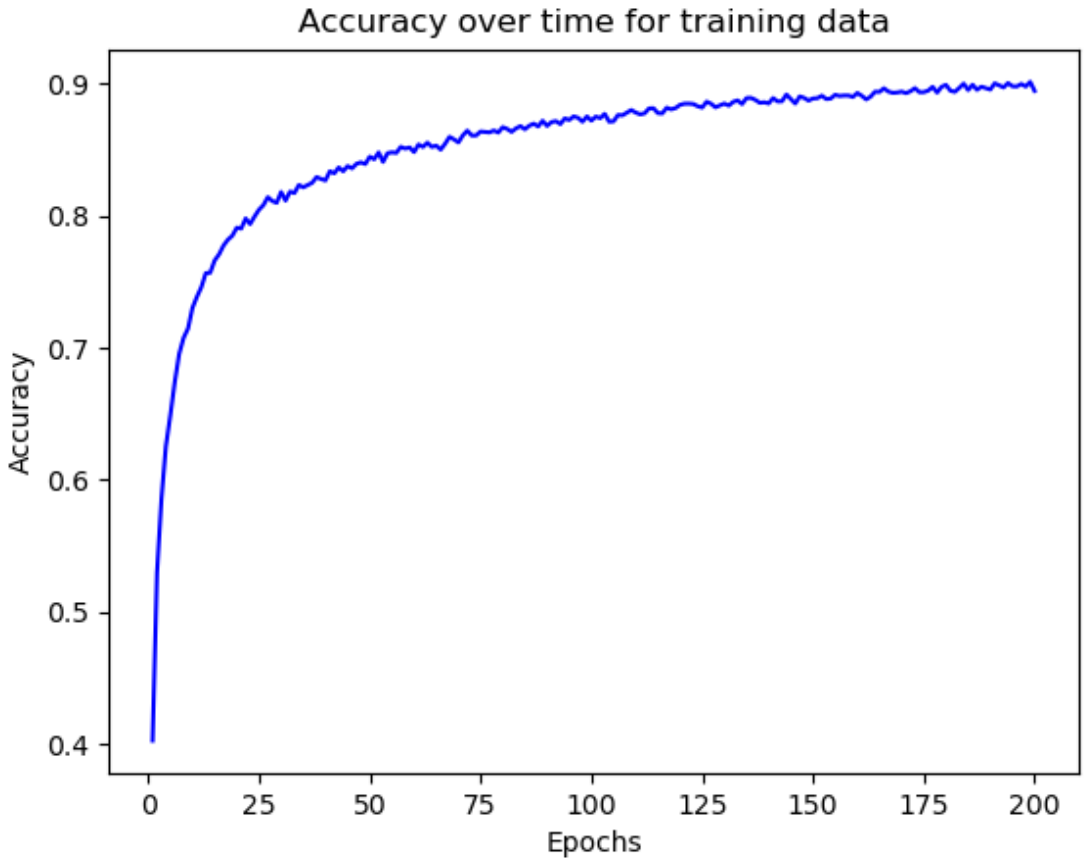
Image Data Generation
We use the image data generator object from keras to modify our images. For example, we changed the width and height, rotated the image by 20 degrees and flipped it horizontally. We then fit the model using these generated images.

For the compilation of the model, we set the epochs to be 200, and steps per epoch to be the length of training data divided by 64.

Evaluation
On the validation set, we achieved an accuracy of 88.24% and a loss 0.3780

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3780 - accuracy: 0.8824
Accuracy : 0.8823999762535095
```

The accuracy over time during training is presented by the graph below:



The loss over time during training is presented by the graph below: