Springboard Data Science

CIFAR-10 Image Classification - Capstone II

Jan 2019

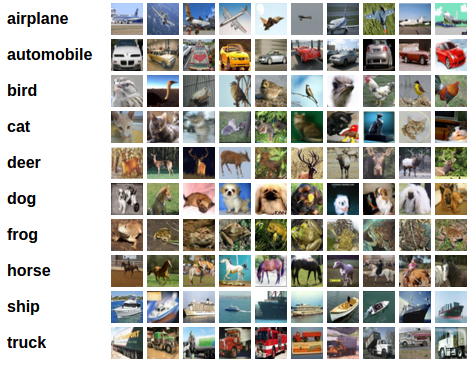
Bhargavsinh Ravalji

### Contents:

1. [Overview \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1](#_1._Overview:)
2. [Data Exploration & Preprocessing \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3](#_2._Data_Exploration_1)
   1. [Date Description \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3](#_2-1._Data_Description)
   2. [Data Exploration\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3](#_2-2._Data_Exploration)
3. [Convolutional Neural Networks (CNN)](#_3._Convolutional_Neural) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 5
   1. [Model \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5](#_3-1._Model)
4. [Training Model \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9](#_4._Training_the)
5. [Testing Model \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 11](#_5._Testing_the)
6. [Conclusion \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 12](#_6._Conclusion)
7. [References \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 13](#_7._References)

# **1. Overview:**

Image classification is one of the great essential problems in Machine Learning. It is the basic element for complex problems such as Computer Vision, Face Recognition System, or Self-driving car. There are many classification models that can be used for this task; however, it is important to fully understand the concepts of each model, and how they perform on different dataset.



**Figure 1: CIFAR-10 Dataset view**

The CIFAR-10 Dataset is an important image classification dataset. It consists of 60000 32x32 color images in 10 classes (airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks), with 6000 images per class. There are 50000 training images and 10000 test images.

The goals of this project are to:

* Learn how to preprocess the image data
* Implement Convolutional Neural Networks (CNN) classifiers using GPU-enabled

**Tools:**

* GPU
* PyTorch
* Colab

# **2. Data Exploration & Preprocessing**

### 2-1. Data Description

The version I used is CIFAR-10 python3 version. The CIFAR 10 training dataset has five batches of file, and each batch contains 10,000 images. The test dataset has one file that contains 10,000 images.

The training set we get is Numpy ndarray with shape (50,000, 3072) and test set is Numpy ndarray with shape (10,000, 3072). Each row of the array stores a 32 x 32 color image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.

The labels for training and test dataset are Numpy array with shape (50,000, 1) and (10,000, 1). They are not one-hot-encoding yet.

The output of torch vision datasets is PILImage images of range [0,1]. Therefore, we need to transform images to tensors of normalized range [-1,1]. So, I normalized the data using transforms, Normalize method. After transformation, I loaded train and test data into train loader and test loader accordingly.

### 2-2. Data Exploration

We reshape each row into a (32,32,3) Numpy array, with one inner array as one pixel with three channels: red, green and blue. The reshaped training data is of shape (50,000, 32, 32, 3). The reshaped test data is of shape (10,000, 32, 32, 3).

Then we plot the first 32 images in training set with true class labels. This is for better understanding of the dataset. The images are plotted to tier thought the train data called train loader



**Figure 2: The first 32 images in training set**

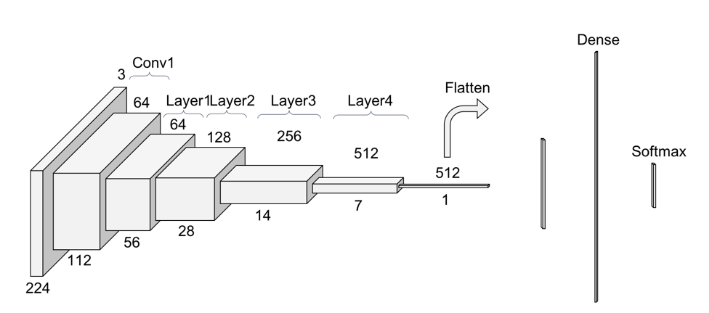
# **3. Convolutional Neural Networks (CNN)**

To start with, I applied pre trained ResNet34 model to build a CNN classifier and explore details to understand how convolution works.

### 3-1. Model

I applied pertained model of ResNet34 for 10 epochs. In each epoch, I used the learning rate 0.001 and 0.0001.

**Architecture of ResNet:**



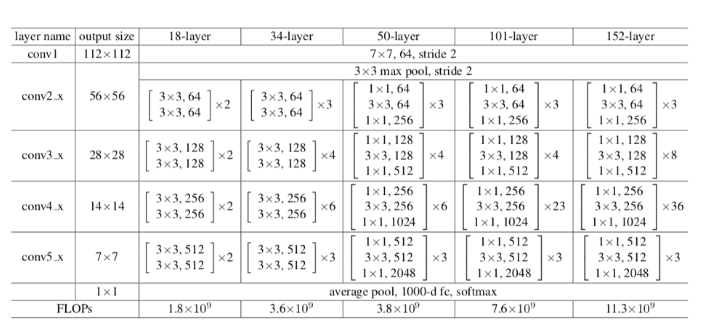
**Figure 3: Image of RestNet34**

**Convolution 1:**

Before entering into the common layer behavior, the first step on the ResNet is a 3x3 convolution with a batch normalization operation. The stride is 1 and there is a padding of 1 to match the output size with the input size. The big difference with ResNet for ImageNet we have not include the max pooling operation in this first block. Because each convolution filter (of the 64) is providing one channel in the output volume, we end up with a (112x112x64) output volume.

**Layers:**

Every layer of a ResNet is composed of few blocks. This is because when ResNet go deeper, it increases the number of operations within a block, but the number of total layers remains the same which is 4. An operation indicates to a convolution a batch normalization and a ReLU activation function to an input, except the last operation of a block, that does not have the ReLU activation function. So, in PyTorch implementation they distinguish between the blocks that includes 2 operations **Basic Block** and **Bottleneck Block**. Each of these operations is called layer, but we are using layer already for a group of blocks. The input volume is the last output volume from Conv1.



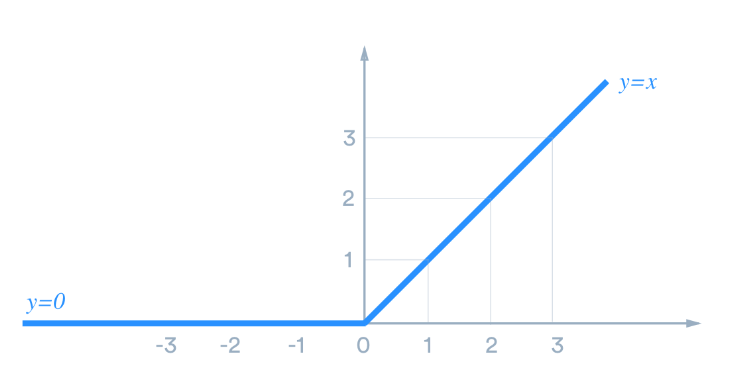
**Figure 4: ResNet34 Architectures**

**BatchNorm2d Layer:**

**BatchNorm2d** normalizes all inputs to have zero mean and unit variance. It greatly boosts the accuracy of CNN models.  It reports the problem of internal covariate shift. It also performances as a regularized, in some cases removing the need for Dropout. Batch Normalization achieves the same accuracy with fewer training steps thus speeding up the training process.

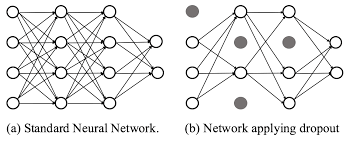
**ReLU Activation Function:**

[ReLU](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))activation function takes an input value and outputs a new value ranging from 0 to infinity. It basically thresholds all incoming features to be 0 or greater. When you apply ReLU to the incoming features, any number less than 0 is changed to zero, while others are kept the same.



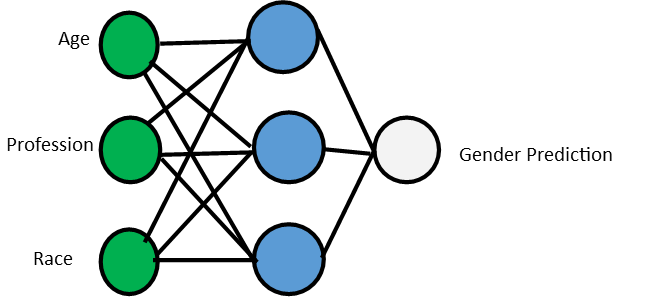
**Figure 5: ReLU Activation Function Dropout**

# **Dropout** is a technique for addressing overfitting problem. The main idea is randomly drop units from the neural network during training. The decrease in number of parameters in each step of training has effect of regularization. Dropout has shown improvements in the performance of neural networks on supervised learning tasks in computer vision.



**Linear Layer:**

The final layer of our network is the linear layer. It’s a standard, fully connected layer that calculates the scores for each of our ten classes. In our linear layer, we have to specify the number of input features, and the number of output features should match to the number of classes we need.



**Figure 6: Example of Linear Layer**

# **4. Training the Model**

First, import the [Adam optimizer](https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/) as: from torch.optim import Adam

Step 1: Instantiate the Model, create the optimizer and [Loss function](https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0)

* Here Adam optimizer added parameters of learning rate of 0.001 and 0.0001
* Loss function applied on variable criterion.

Step 2: Write training function

* First we loop over the loader for the training set
* If GPU support is available, we move both the images and labels to the GPU
  + images = images.to(device)
  + labels = labels.to(device)
* Next, clear all currently stored gradients.
  + optimizer.zero\_grad()
* This is significant because weights in a neural network are adjusted based on gradients stored for each batch, therefore for each new batch, gradients must be reset to zero, so images in a previous batch would not spread gradients to a new batch.
* Next, we pass our images into the model. It returns the predictions, and then we pass both the predictions and actual labels into the loss function.
  + output = net(images)
  + loss = criterion(output, labels)
* Next, we call **loss.backward()**to spread the gradients, and then we call **optimizer.step()**to modify our model parameters in accordance with the spread gradients.
* Then, compute all the metrics:
* cum\_loss += loss.item()
* out = torch.argmax(output.detach(),dim=1)
* assert out.shape==labels.shape
* running\_acc += (labels==out).sum().item()

Here we retrieve the actual loss and then obtain the maximum predicted class. Finally, we sum up the number of correct predictions in the batch and add it to the total train\_acc.

After each epoch, we call the learning rate adjustment function, compute the average of the training loss and training accuracy, find the test accuracy, and log the results.

More importantly, we keep track of the best accuracy, and if the current test accuracy is greater than our current best, we’d call the save models function.

# **5. Testing the model**

After training the network over the training dataset, we need to check if the network has learnt anything at all. After testing the model on test loader, model achieved accuracy around 82% by applying ResNet34 pertained CNN.

# **6. Conclusion**

Deep learning CNN has revealed great potential in the area of Image classification. CIFAR-10 dataset is well studied dataset which partial dataset of ImageNet. Therefore, we can observe the conclusion from CIFAR-10 and apply to ImageNet dataset as well. Using PyTorch library and Resnet34 pertained CNN, we achieved around 82% accuracy after 25 iterations.

My next approach will be to apply deep learning model from scratch and improve accuracy on test data accuracy by tuning hyper-parameters. There are so many well tune models on large dataset available which can be used for solving a similar problem.

# **7. References**

Chansung, P. (2018, May 17). 302 Found. Retrieved February 3, 2020, from <https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c>

Jay, P. (2018, June 20). Understanding and Implementing Architectures of ResNet and ResNeXt for state-of-the-art Image… Retrieved February 3, 2020, from <https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624>

Kathuria, A. (2019, July 10). Building a CIFAR classifier neural network with PyTorch. Retrieved February 3, 2020, from <https://blog.paperspace.com/pytorch-101-building-neural-networks/>

Kumar, A. (2019, July 1). Achieving 90% accuracy in Object Recognition Task on CIFAR-10 Dataset with Keras: Convolutional Neural Networks. Retrieved February 3, 2020, from https://appliedmachinelearning.blog/2018/03/24/achieving-90-accuracy-in-object-recognition-task-on-cifar-10-dataset-with-keras-convolutional-neural-networks/

Medium. (2020). Basics of Image Classification with PyTorch. [online] Available at: https://heartbeat.fritz.ai/basics-of-image-classification-with-pytorch-2f8973c51864 [Accessed 3 Feb. 2020].