CIFAR-10 Image Classification

Training Convolutional Neural Networks using AWS

Springboard Career Track - Capstone 2

Ken Wallace - January 2018

The problem

Objective

The goal is to create a Convolutional Neural Network to perform a computer vision image classification on the CIFAR-10 dataset that achieves better than 80% accuracy on the test set.

Project Plan

1. Learn how to create a baseline neural network in Keras with Tensorflow.
2. Add convolutional layers to improve the accuracy.
3. Create an Amazon Web Services (AWS) instance with a GPU.

Use Cases

Applications ranging from self-driving cars to satellite image analysis to medical image (e.g. X-ray, MRI, CT) analysis to smart kitchen appliances

Data import

Dataset

Import training data from Keras datasets

After some trial and error, this data seemed to yield better results than the data downloaded and unpickled from the original source

Training data

Training data consisted of 50,000 color (RGB) images, each 32x32 pixels, with labels identifying its content. Each image was used as either a (32x32x3) array or as a (1, 3072) array. 32*32*3=3072

Testing data

Testing data consisted of 10,000 images in the same format as the training data. This set remained untrained as was used for checking the accuracy of each model.

Data preprocessing

- For baseline models, the 4D arrays, (50000,32,32,3) and (10000,32,32,3) required reshaping to (50000,3072) and (10000,3072). This was accomplished using

- For convolutional models, the arrays were used as they were imported, (50000,32,32,3) and (10000,32,32,3).

- Data required normalization so all the RGB values ranged from 0 to 1 rather than 0 to 255. This was done by dividing all values by 255.

- Label arrays (1D arrays that tell the model what category each image belongs to for supervised learning) required conversion to categorical variables.

Baseline modeling

1. The non-optimized baseline model was originally attempted with the Adam optimizer using the default learning rate (0.001) and all models only achieved a 10% accuracy.
2. It turned out that either Adam was not a good choice for this dataset, or the default learning rate was the issue.
3. Using a baseline model, three subsequent models were fit, varying the optimizer and learning rate:
    a. SGD, or Stochastic Gradient Descent, was used with a learning rate of .0001
    b. Adadelta was used with the default learning rate of 1.0
    c. Adam was used with a learning rate of .0001.
4. Then a GridSearchCV was run to check different batch sizes and # of epochs using the best baseline optimizer from above.
    a. The jupyter notebook for this keeps hanging during GSCV, so code was modified to eliminate the larger batch sizes.

Simple CNN (Convolutional Neural Network)

1. A basic CNN with 6 hidden layers was used to compare three learning rates for a single optimizer to the baseline model.
   a. The model contained only 2 convolutional layers with filter sizes of 32 and 64, with a MaxPooling layer after each, and a 20% Dropout layer, a flattening layer, and a Dense Layer prior to the output layer. All activations were ReLU (Rectified Linear Units).
   b. Learning rates for this model were .001, .0001, and .00001.

Deep CNN (Convolutional Neural Network)

1. A deep CNN with 16 hidden layers was used to compare three learning rates for a single optimizer to the baseline model.
2. Each of the three learning rates was paired with a set of layers that started with 32 (a pair of layers with 32 filters, a pair with 64 filters, and a pair with 128 filters).
3. Subsequent models had layers with filters of 48, 96, 192, and 64, 128, 256.
4. Additionally, there were MaxPooling and Dropout layers after each pair of Conv2D layers, and Flatten, Dense, and more Dropout layers as it moved toward the output layer.The model contained only 2 convolutional layers with filter sizes of 32 and 64, with a MaxPooling layer after each, and a 20% Dropout layer, a flattening layer, and a Dense Layer prior to the output layer. All activations were ReLU (Rectified Linear Units).
    a. Learning rates for this model were .001, .0001, and .00001.

Initial Findings

1. Baseline models
   a. SGD accuracy (lr = .0001) = 47.41%
   b. Adadelta accuracy (lr = 1.0) = 47.59%
   c. Adam accuracy (lr = .0001) = 47.39%
2. GridSearchCV using Baseline model to find batch_size*
   a. Best batch size was 64 at 100 epochs (52.23% vs 50.16% using batch size 128)
3. Test of Adam optimizer at different learning rates using basic CNN
   a. Accuracy (lr = .001) = 73.78%
   b. Accuracy (lr = .0001) = 72.23%
   c. Accuracy (lr = .00001) = 73.94%

Secondary Findings

For the Deep CNNs, the first pass through had all the batch sizes set to 128*.

In addition, they were set to stop early if the loss stopped decreasing (threshold .0005 max over 10 epochs).

The best results from the the first pass through were tested again with a batch size of 64 and left to run for a full 200 epochs.

*This was an error due to misreading the results of the GridSearchCV, which indicated that, at least for the Baseline model, batch_size=64 was better than batch_size=128.

Secondary Findings

Top 4 results for validation accuracy:

1. Filter 1 = 48, batch_size = 64, epochs = 300: **Accuracy = 84.84%**
    a. Difference from Training accuracy: 12.72%
2. Filter 1 = 64, batch_size = 128, epochs = 500: **Accuracy = 84.73%**
    a. Difference from Training accuracy: 14.54%
3. Filter 1 = 48, batch_size = 64, epochs = 200: **Accuracy = 83.87%**
    a. Difference from Training accuracy: 12.55%
4. Filter 1 = 64, batch_size = 128, epochs = 200, with decay: **Accuracy = 83.77%**
    a. Difference from Training accuracy: 14.32%

Conclusions

1. The final models (with basically equivalent results) achieved 84.84% and 84.73% accuracy to the test set.
2. Based on testing a number of different combinations of hyperparameters, this appears to be about the best that can be achieved without a bump in computing power (e.g. multiple GPUs, more memory, etc.) and an even deeper network.
3. There are nearly infinite combinations of hidden Conv2D layers (count and size), number of filters (at each layer), kernel sizes, number and size of Dense layers, number and size of Dropout layers, optimizers, loss functions, and activation types.
4. The next step should be to implement image augmentation.
5. Other methods that have achieved better results include SVM as final layer activation and DropConnect, neither of which have Keras implementations at this time.