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 machine learning model that could classify color images from 10 categories with an accuracy of 80% or better.

Project Plan

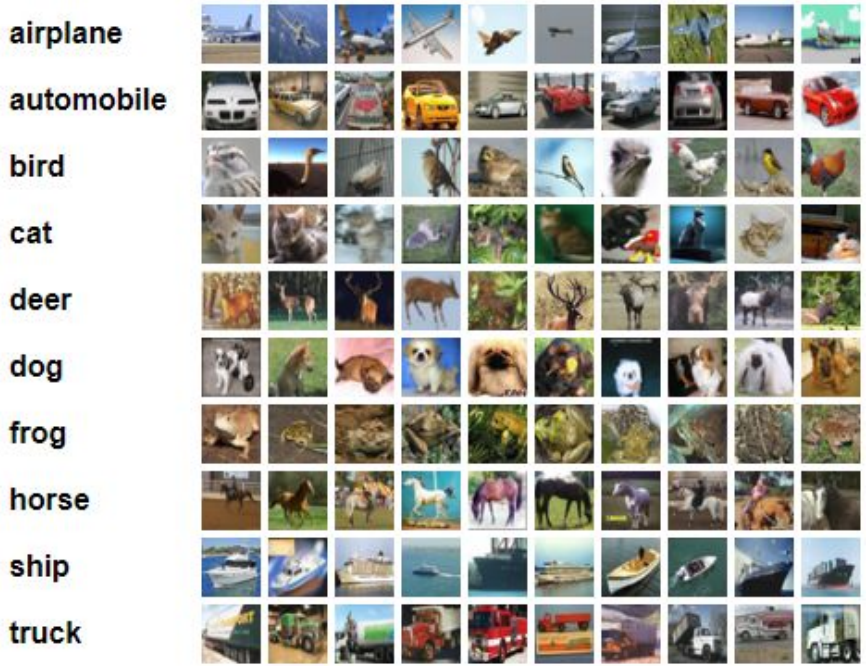
1. Learn how to create a baseline neural network in Keras with Tensorflow.
2. Add convolutional layers to improve the accuracy.
3. Create an Amazon Web Services (AWS) instance with a GPU.

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

The dataset

- The CIFAR-10 data is a labeled subset of the 80 million tiny images dataset, collected by a team from the University of Toronto.
- It consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- The training set contains exactly 5000 images from each class. The test batch contains exactly 1000 randomly-selected images from each class.
- The classes are completely mutually exclusive. There is no overlap between automobiles and trucks.



Data import

Dataset → Training data → Testing data

Import training data from Keras datasets

Data used was imported from the Keras datasets library. Preprocessing was required for some steps and is described on the next slide.

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 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 the numpy reshape command.

- 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 kept freezing 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.

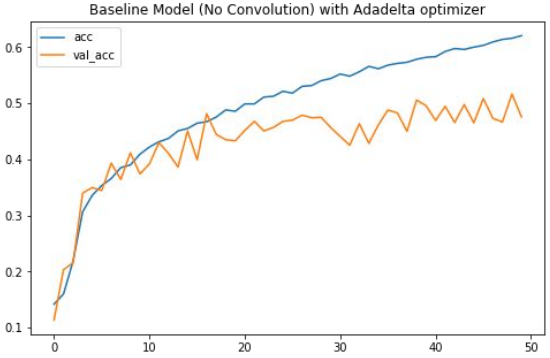
Deep CNN (Convolutional Neural Network)

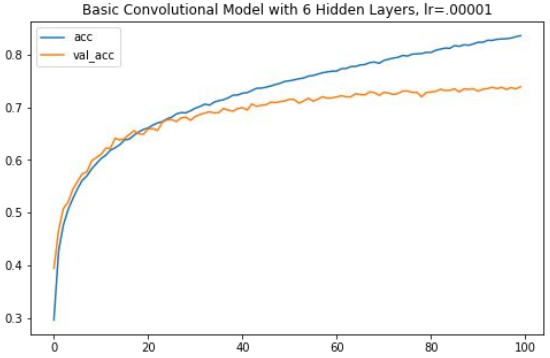
1. A deep CNN with 16 hidden layers was used to compare three learning rates for a single optimizer to the baseline model.
2. Each of the three learning rates was paired with a set of layers that started with 32 (a pair of layers with 32 filters, a pair with 64 filters, and a pair with 128 filters).
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. All activations were ReLU (Rectified Linear Units).

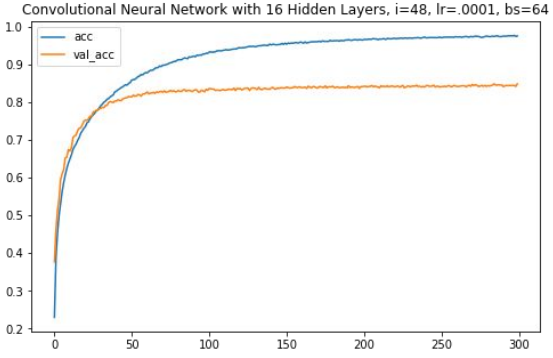
Results

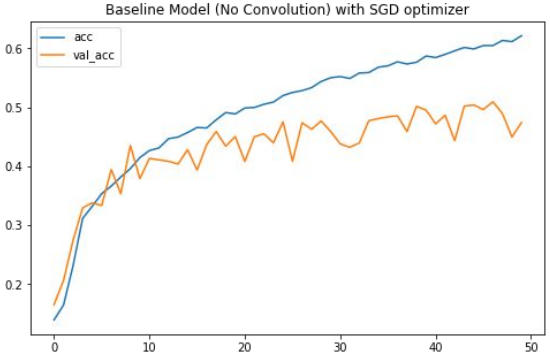
Baseline Model		**Simple CNN (Adam Optimizer)**		**Deep CNN**	
Optimizer	*Accuracy*	*Learning Rate*	*Accuracy*	*# params / batch sizes / epochs / learning rate*	*Accuracy*
SGD (lr=.0001)	47.41%	.001	73.78%	1,451,914 / 64 / 300 / .0001	84.84%
Adadelta (lr=1.0)	47.59%	.0001	72.23%	2,578,954 / 128 / 500 / .0001	84.73%
Adam (lr=.0001)	47.39%	.00001	73.94%	2,578,954 / 128 / 382 / .00001	80.60%

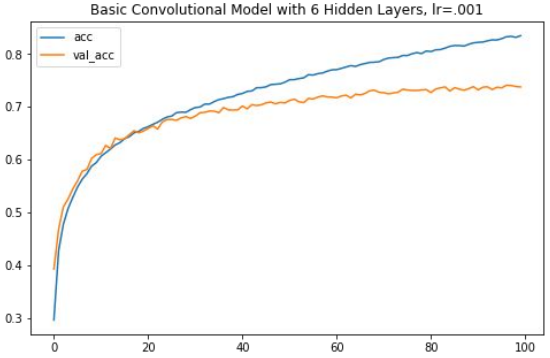
Accuracy Results Plots

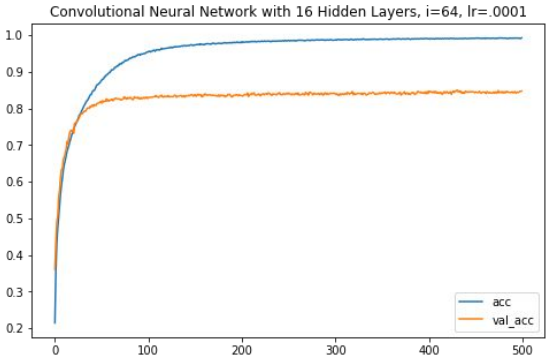












The top result is best because the test accuracy is highest and it is less overfitted than the next best model.

Explanation of Results

1. Baseline Models
    a. GridSearchCV using Baseline model to find batch_size
        i. Best batch size was 64 at 100 epochs (52.23% vs 50.16% using batch size 128)

    b. Test of three different optimizers resulted in very similar accuracy results.

2. Simple CNN
    a. Test of Adam optimizer at different learning rates resulted in very similar results as well, which says that the learning rate for the Adam optimizer is not that sensitive for this dataset.

Explanation of Results

3. Deep CNN
    a. 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).
    b. 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.
    c. It was possible to achieve similar accuracy numbers with fewer parameters (neurons) in the network and with fewer epochs as one with more parameters and epochs.
    d. Despite that, with more computing power, higher accuracies can be achieved with even deeper networks (i.e. at least an order of magnitude more parameters).

Challenges

Challenges to solving a problem of this nature using a convolutional neural network include:

1. Computational limitations of my laptop led to me to use AWS (Amazon Web Services) in order to use a GPU. I used a p2.xlarge instance that had CUDA GPU support and also had Tensorflow preinstalled.
2. Even with a GPU instance with 8 GiB of RAM, I was hitting the limit of the virtual machine during what was ultimately a minimal GridSearchCV for batch_size and # of epochs.
3. In addition, my computational graph topped out at 2.5 million parameters, and with that many parameters and 500 epochs it took over 4 hours to train. Ideally the graph would have 10X as many parameters in order to better train this model.
4. Lack of Keras support for methods that might have achieved better results, including SVM and DropConnect.

Conclusions

1. **The final models (with basically equivalent results) achieved 84.84% and 84.73% accuracy to the test set.**
2. Given the AWS instance that I was using and the limited time available to try more combinations of hyperparameters, I would call these results successful.
3. The next step, which should be able to run on this AWS instance, would be to implement image augmentation to increase the size of the training dataset.
4. 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.
5. This was a very interesting project that could have broad implications in the world of computer vision, deep learning, and AI.