Alesya Trubchik
Computer Vision
Northeastern University
11/26/17

Option 2 – CIFAR-100 classification with Convolutional Neural Networks

---

Dependencies

python 2.7
tensorflow (1.4)
tflearn
scipy
numpy
sklearn
matplotlib

Overview of files

`predict-test-labels.pkl`
a numpy array of shape (10000, 100) containing the predicted test labels for the test image set. The rows represent test images and the columns contain their class labels, where column 0 represents the label with the highest probability and column 100 represents the label with the lowest probability for the test images.

`true-test-labels.pkl`
a numpy array of shape (10000,) containing the true labels for the test image set, where index i contains the true label (0-99) for test image i .

`train-val_split.py`
Splits Cifar100 train set into train and validation sets, by random choice
`-- train-ind.pkl`
Indices of images to be used for training, selected by random choice. 90% of the Cifar100 train set.
`-- val_ind.pkl`
Indices of images to be used for validation, selected by random choice. 10% of the Cifar 100 train set.

`model1.py – model6.py`
Different model architectures used to train on the Cifar100 dataset

`plot_confusion_matrix.py`
Plots the confusion matrix for the winning architecture (model6)

The following are in located in `/home/alesya` on Maria's computer:

`tmp/tflearn_logs/`
Contains the tensorflow logs from training models 1-6. They can be viewed with command: tensorboard --logdir tmp/tflearn_logs/ by navigating to localhost:6006, also displayed in report below.

`cifar_model6.tflearn.data-00000-of-00001`
`cifar_model6.tflear.index`
`cifar_model6.tflearn.meta`
Saved model6, the winning architecture. Can be loaded from model6.py after the model initialization with model.load('./cifar_model6.tflearn').

Details

I worked with the TensorFlow and tflearn tensorflow libraries.

I used tflearn's example convnet designed for Cifar 10 as a starting point:
https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_cifar10.py
This is the architecture trained in model 1, which achieve accuracy just above 40%

Model 2 experiments with same architecture as model 1, except using sgd optimizer instead of adam. This failed to train – the accuracy didn't increase at all.

Model 4 – added additional dropout to model 1. Performance about the same

Model 5 – increased dimensions of the conv layers, outperformed previous models

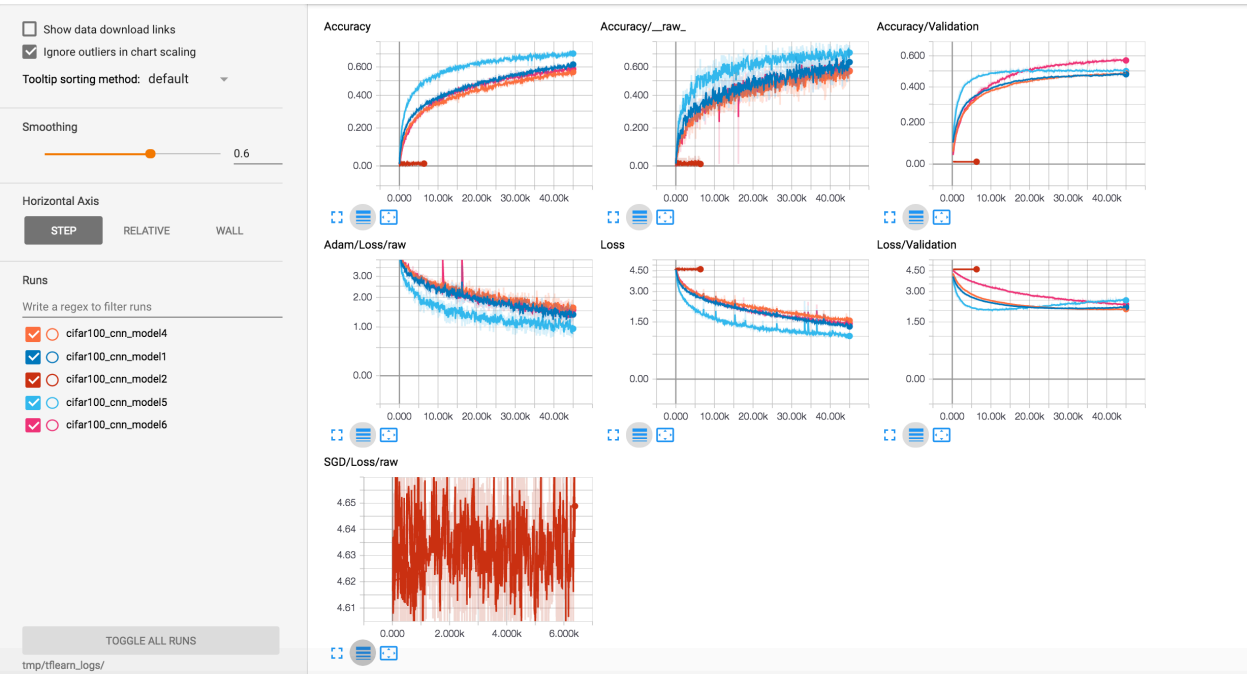
Model 6 - **winning architecture**, inspired by
"Let's keep it simple, Using simple architectures to outperform deeper and more complex architectures," Hossein Hasanpour et al. (https://arxiv.org/pdf/1608.06037.pdf)

Performed best, reaching validation accuracy of 56-58% while training. This model has several conv layers of dimension 128 x 3, with maxpool of size 2 and dropout of 50% in between most of these layers.

Note: Model 6 with sgd validation accuracy didn't get above 2% after 40 epochs. I also experimented by adding batch norm layers in between the conv 128x3 layers, but didn't achieve better performance.

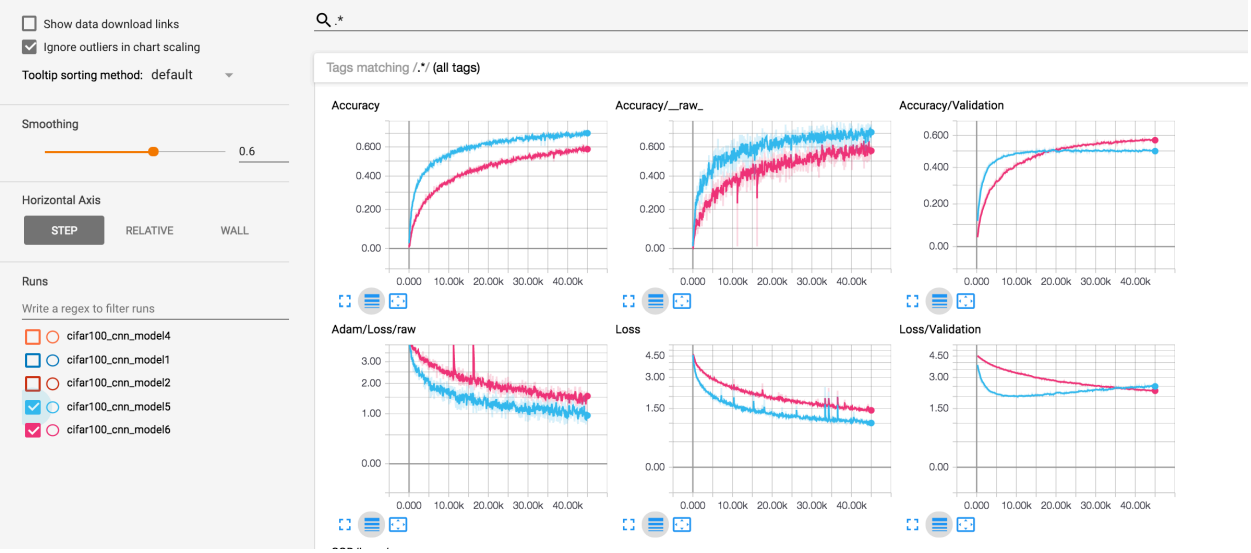
Below are training/validation accuracy and loss graphs.

Graph 1: All models. Even though model 6 did not achieve lowest loss or training accuracy, it achieved best accuracy on the validation set.



Graph 1

Graph 2: Model 5 (blue) vs Model 6 (pink) comparison. Model 6 has two more conv layers of dimension (input_size x 128 x 3). In addition, it includes 50% dropout in between most conv layers. Interesting observation of the modification from model 5 to model 6: model 6 training accuracy remained below validation accuracy during most of the training time, until they both reached around 50% accuracy. This is probably due to the dropout layers, which are good at keeping the model from overfitting to the training set, resulting in better generalization to the validation and test sets.



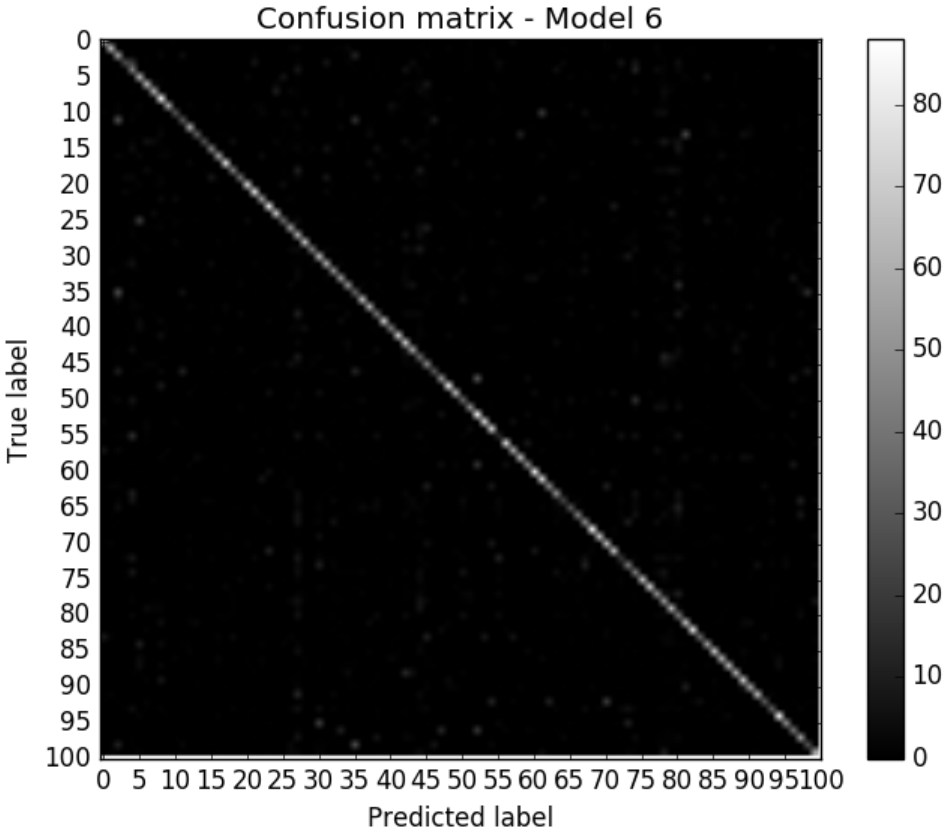
Graph 2
Success in using dropout and a deeper model – Model 6 in pink.

Confusion matrix

Model 6
Based off the confusion matrix, the 10 classes that performed the poorest, i.e., had the smallest values along the diagonal, are: 55 72 64 3 14 98 10 46 11 65, class label names below.

The top 10 most “confused” classes are on the right. Looking at these classes, we can see that it makes sense that some of them would have lowest prediction accuracy. For instance, “seal” and “otter” are very similar mammals, that humans could often confuse as well. We also see that “woman”, “boy”, and “man” are all in the poorest 10 performing classes, which suggests the model may have learned to recognize humans in general, but has not learned the characteristics that set apart genders and/or the age of a human. So it confuses boy, woman, and man.

55: 'otter'
72: 'seal'
64: 'possum'
3: 'bear'
14: 'butterfly'
98: 'woman'
10: 'bowl'
46: 'man'
11: 'boy'
65: 'rabbit



Finally, Model 6 obtained **56.4%** accuracy on the evaluation of the Cifar100 test set.
(To run evaluation: python model6.py)