**Assignment 4: Deep Learning**

Emily Strong

CSYE 7245 Spring 2018

**Introduction**

For this assignment, I am developing a face detection convolutional neural network (CNN). This is a binary classifier in which an image is categorized as either containing a face or not. Because my project data set is only of faces, I am combining it with the CIFAR-10 [1], a large data set of 32 x 32 pixel images commonly used in image classification tutorials. I chose the CIFAR-10 because six of the ten categories included are of animals, which should help to ensure that my face detector is specific to human faces and does not detect a single facial feature that might be similar among species (e.g. eyes).

**Methods and Results**

**Data Set**

I am using the FaceScrub [2] data set combined with the CIFAR-10. To standardize the FaceScrub images I ran them through the OpenFace align step to generate square images cropped to the most prominent face in the image. While this means that the images are all aligned and thus the face detector could only detect faces that are positioned in this way, the image augmentation I use in my network should help to prevent this as it randomly transforms the images in various ways. After generating the aligned images I then resized them using the PIL library to be 32 x 32 to match the CIFAR-10 images, and selected 50,000 of the images at random so that the face and no-face sample sizes would match. To use the images with a neural network I then converted the images to numpy arrays with three color channels (RGB), giving an array shape of (32, 32, 3).

**Part A: Deep Learning Model**

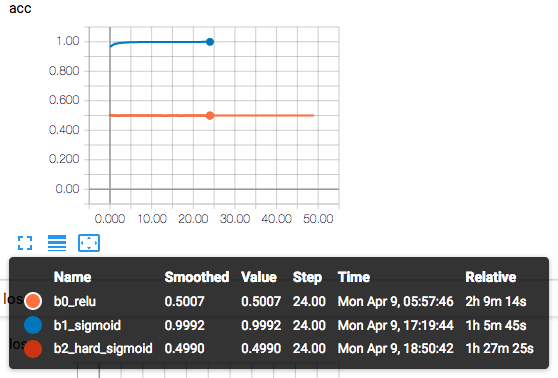
I modeled my network on the Keras CIFAR-10 example [3] and a binary image classification tutorial from Medium [4], with adaptations to the final model based on the results of Parts B – G. Notably, both of these tutorials used rectified linear unit (relu) as the activation function but my model consistently performed at chance using that, and it was only during Part B when I tested sigmoid that my model finally started improving its accuracy. The network layers are:

* 2D convolutional, size 32
* 2D max pooling
* 2D convolutional, size 64
* 2D convolutional, size 64
* 2D max pooling
* dense, size 512
* dropout to regularize
* dense, single node to output the final binary classification

I use ImageDataGenerator to increase the number of images trained on in each epoch and diversify the images since the faces are all aligned, the Adam optimizer, and random normal initialization.

**Part B: Activation Function**

Since from Part A relu was performing at chance, I decided to test sigmoid as it is similar to a logistic regression which would be appropriate for a binary classification problem. I also tested hard\_sigmoid, a variation of sigmoid that uses linear approximation. Sigmoid achieved 99.9% accuracy on the training and test sets. Hard sigmoid, like rely, performed at chance (50% since this is a binary classification problem).

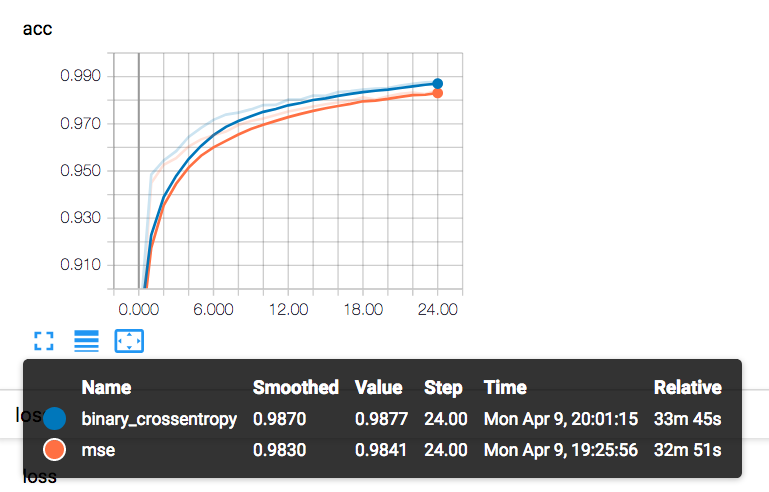


*Figure 1: Accuracies for different activation functions*

Sigmoid also plateaued within a few epochs, so for the remaining parts I chose to make the problem harder by increasing the image transformations and decreasing the number of images in each epoch so that it would be easier to differentiate results.

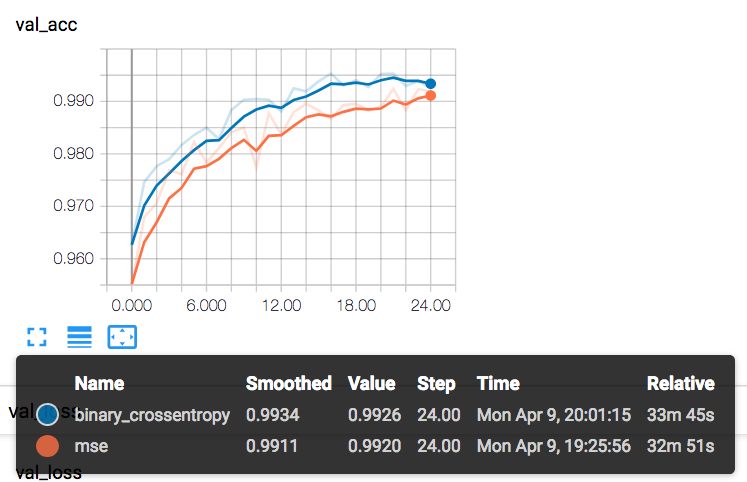
**Part C: Cost Function**

In the previous parts I used binary cross entropy as my loss function. For Part C, I compared it to the mean squared error. I made the changes previously mentioned and reran binary cross entropy to make it directly comparable to MSE.



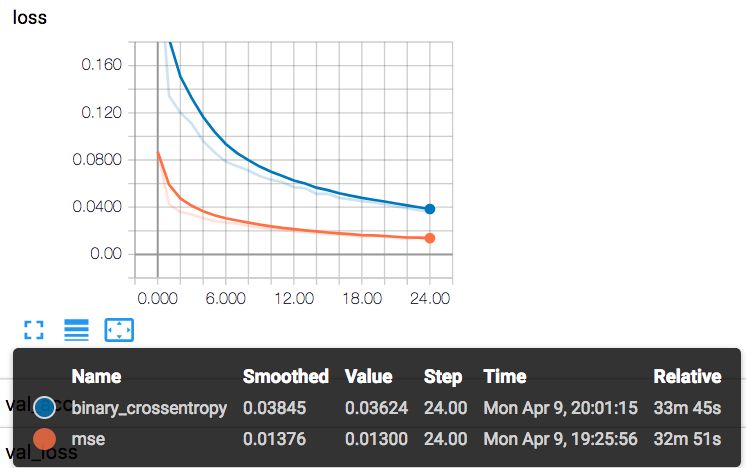
*Figure 2: Accuracies for different cost functions*

For training accuracy, binary cross entropy performed marginally better and both plateaued at around 10 epochs.



*Figure 3: Validation accuracies for different cost functions*

For the validation accuracies, binary cross entropy again performed better though MSE was within 0.3%. Binary cross entropy plateaued somewhere between 15 and 20 epochs whereas MSE was still increasing at 25. However, scale needs to be considered. These accuracies were in the 90s from the first epoch, so though MSE may have grown more and eventually surpassed binary cross entropy, the model is at risk of overfitting and it is unlikely that the final model will have the number of epochs necessary to reach that point.



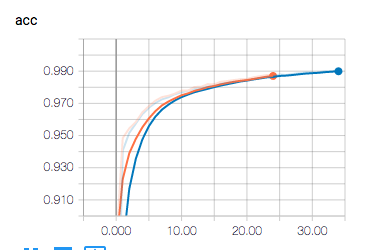
*Figure 4: Loss for different cost functions*

The loss with MSE rapidly plateaued and was overall much lower.

Due to its consistently superior accuracy on both the train and validation data, I will continue to use binary cross entropy for the remaining parts, though the difference between the two is so minor MSE would be an equally good choice.

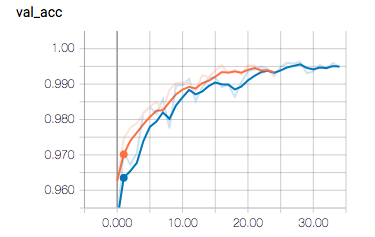
**Part D: Epochs**

I compared my 25 epoch run from Part C with a 35 epoch run.

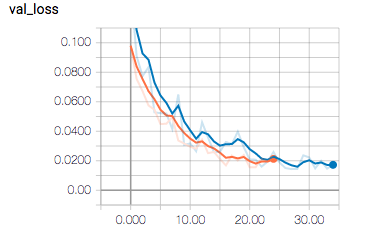


*Figure 5: Accuracies for 25 and 35 epochs*

In looking at the training accuracies, the plateau around 10 is consistent between the two epoch counts. It is thus the validation graphs that are needed to differnetiat the two.



*Figure 6: Validation accuracies for 25 and 35 epochs*

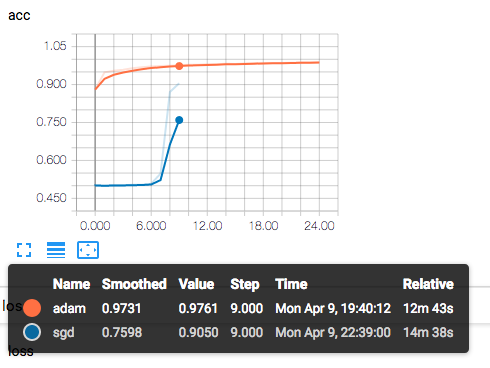


*Figure 6: Validation losses for 25 and 35 epochs*

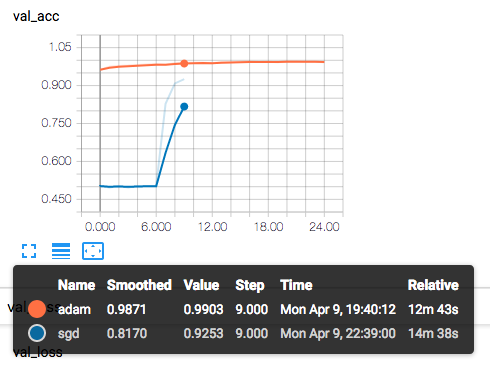
As figures 5 and 6 illustrate, at 25 epochs the validation accuracy and loss begin to plateau. This is thus an appropriate number to use for the final model as a larger number would lead to overfitting.

**Part E: Gradient Estimation**

I chose SGD to compare to the Adam optimizer I have been using. As figures 7 and 8 show, it was initially performing at chance so I truncated it to 10 epochs to illustrate that behavior without wasting time on failed training. Interestingly, after 8 epochs it suddenly started to improve. However, the train accuracy with the Adam optimizer has already plateaued at the point at which the runs terminated for SGD and as the relative times illustrate, SGD consistently ran slower than the Adam optimizer. This comparison does highlight that while in other parts the accuracy with the Adam optimizer has appeared to plateau at 10 epochs, when the Y axis has a larger scale it actually plateaus around 4 epochs.



*Figure 7: Train accuracies for Adam vs SGD optimizers*

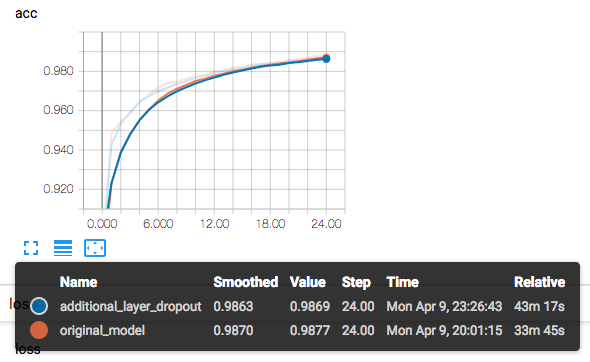


*Figure 8: Validation accuracies for Adam vs SGD optimizers*

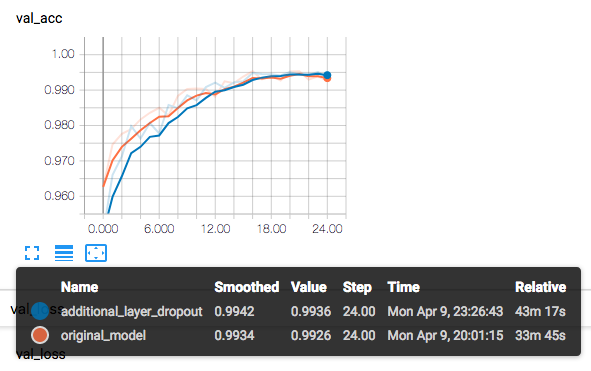
Overall, the Adam optimizer is a better choice for this network since is faster and reaches a plateau more quickly.

**Part F: Network Architecture**

In Part A, I initially used a network architecture from the Keras CIFAR-10 example but removed portions of it while troubleshooting why the network wasn't successfully training. For Part F, I have added back those component. In particular, the first convolutional layer now uses 5x5 convolutions rather than 3x3 and there is a second convolutional layer of size 32, and I have added dropouts after the two poolings to regularize the network.



*Figure 9: Train accuracies of original vs new CNN*

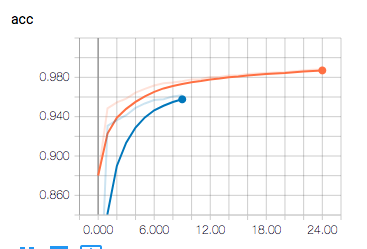


*Figure 10: Validation accuracies of original vs new CNN*

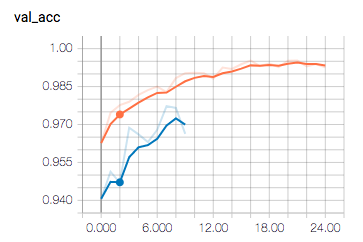
As figures 9 and 10 show, the two networks are identical in training accuracy, and plateau at the same point despite the additional convolutional layer and the two additional dropouts. The new network however does take much longer to run. In contrast, the network I have been using in Parts B-E performs better initially on the validation data, though the new network does catch up. Because of the equivalent performance and the shorter run time, the network I have been using is best.

**Part G: Network Initialization**

I previously experimented with network initialization in Part A. I have been using random normal initialization for both kernel and bias, so I decided to try the other configuration I had tested in A with the network that now works. Specifically, I am using random uniform for kernel and ones for bias. Since I previously compared these in attempts 6 and 7 of Part A and I suspect random normally distributed numbers are better for my model since I have been getting such strong results, I trained the model for only 10 epochs to test this hypothesis.



*Figure 11: Training accuracies for random normal and uniform random/ones initialization*



*Figure 12: Validation accuracies for random normal and uniform random/ones initialization*

As figures 11 and 12 show, my hypothesis was correct. Although the initialization does not have as significant an effect on accuracy as other parameters such as the optimizer, the uniform random/ones initialization is slower to train.

**Discussion**

As the problems I encountered in Part A that were resolved by changing the activation function in Part B, selecting the right activation function for the task is critical for a deep learning network. Since I created a binary classification model, using an activation function that is similarly binary is essential. My results in the successive parts of the assignment demonstrate how the gradient estimation algorithm can dominate the efficiency of the model. With the Adam optimizer, my models were getting greater than 90% accuracy on the first epoch, even after I modified the training to be less efficient so it would be easier to compare results. Other hyper-parameters had much more marginal effects on the training speed and accuracy of the network.

**References**

[1] <http://www.cs.toronto.edu/~kriz/cifar.html>

[2] <http://vintage.winklerbros.net/facescrub.html>

[3] <https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py>

[4] https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721