CSC 732 Final Project

Part 1 Problem 2 Analysis

Dominic Klusek Jonathan Rozen

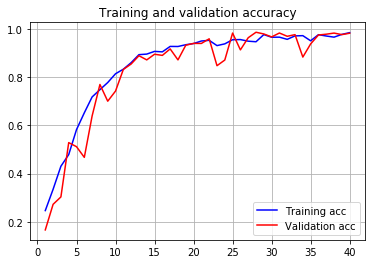
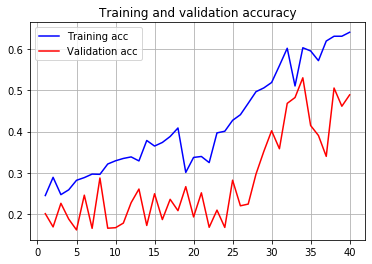
Convolutional Neural Networks (CNNs) are a Deep Learning algorithm which learns filters to extract certain information from images with the end goal to solve a problem such as classification. The architecture of CNNs is inspired by the organization of the Visual Cortex wherein individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field; with a collection of these neurons they collectively cover the whole visual area. The main advantage of CNN over other architectures is the learning of filters which can capture both spatial and temporal dependencies rather than learning certain numerical patterns for pixel data. Not only that, but CNNs are very versatile and their output can easily be fed into a classifier such as Dense layers that are utilized in architectures such as Multilayered Perceptron. For these reasons, we utilize a CNN connected to 2 Denser layers to effectively learn features to classify different polyhedral dice which are in various poses.

The dataset we used is a subset of Dice: d4, d6, d8, d10, d12, d20 Images which contains 16,000 images of various colored sets of polyhedral dice. Different dice have a different number of sides ranging from the d4 with 4 to the d20 with 20 sides. The images vary based on rotation of dice, the surface they are on, and the color of the dice. We split the data randomly into three subsets: training which contains 6,600 images(1,100 for each class), validation which contains 1,800 images (300 for each class), and testing which contains 1,200 images (200 for each class). We did this to also remove some of the class imbalance that was present as some classes such as d6 had >4,000 images while other like d4 had <2,000 images total; in addition many of the images were similar to one another as the authors of the dataset took images where the only difference was a slight rotation of the die. Thus, taking random subsets of the dataset reduces bias and redundancy.

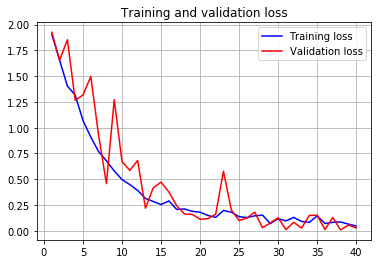
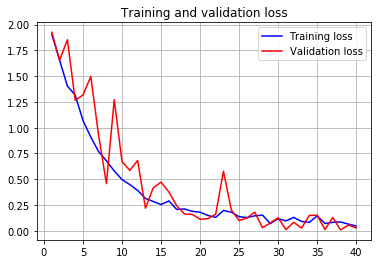
To gauge performance between training our CNN with Stochastic Gradient Descent (SGD) with Nesterov Momentum and with Adam; we trained the model for 40 epochs with a batch size of 32 Looking at Figures 1-4 we can see that when training our model with SGD the training is more erratic, and it isn’t until epoch 20 that the model begins to actively start learning and improving in performance; while with Adam training is rather smooth with validation being a bit erratic due to the differences in images, but overall Adam consistently learned and improved over the entire training. When looking at SGD we see a significant amount of overfitting roughly 10% difference between the training and validation accuracy ;while Adam manages to maintain little to no overfitting, except for some epochs where the validation accuracy dipped. It is easy to see that Adam outperformed SGD by a significant amount, which makes sense as Adam was built utilizing improvements that other researchers made to SGD with Nesterov Momentum.

The reasons for this difference in performance and training are most likely: the variable learning rate the Adam possesses. The variable learning rate is an idea that is employed by many more modern optimizers that build on top of SGD such as Adagrad, RMSProp, and Adam; it allows for larger learning rates when moving in a consistent direction, and slowing down when moving in opposing directions. To do this Adam keeps an exponential moving average of the gradient and the squared gradient, and the quotient of the average of the gradient and squared gradient controls the learning rate. Utilizing an optimizer like Adam, but instead a learning rate scheduler could be used to manually control the learning rate during a range of epochs.

From the results the CNN can effectively learn to classify different polyhedral dice due to it effectively learning filters that capture relevant data that can differentiate between the different classes of dice. Not only that, but it can also be seen that the improvements made in the optimizers have benefitted the training of NNs in a large way; with the introduction of variable learning rates allows for more consistent learning and allows models to train faster when moving towards a minimum. In addition Dropout layers provide and excellent way to mitigate overfitting on properly training models while BatchNormalization ensure more consistent training with fewer sporadic updates. In conclusion, our model was a success when utilizing Adam, and the inclusion of an ImageGenerator, Dropout layers, and a BatchNormalization layer helped the model to learn general features and regularize training to reduce sporadic training; ultimately achieving a high accuracy of 97.77% on our testing set.



Figures 1-2: Training and validation accuracy plots of training with SGD(left) and Adam(right)



Figures 3-4: Training and validation loss plots of training with SGD(left) and Adam(right)