Machine Learning Practical - Assignment 3

s1247438

February 16, 2017

Introduction

This coursework aims to find an effective baseline Neural Network model for the CIFAR-10 and CIFAR-100 datasets. These contain sets of 32x32 labeled images falling into 10 and 100 categories respectively. The resulting model will be used in the following coursework to experiment with more complex network architectures. Finding such baseline model will be achieved by exploring the following topics.

- What are the effects of transforming the RGB dataset into different colour spaces on time and accuracy?
- What is a good baseline network architecture in terms of number of layers and hidden units?
- What are the effect of different optimizers?
- Regularization
- Data augmentation

Methodology

The resulting best-performing model found consisted of 4 fully connected layers described in table 1. Various parameters were tested for each of the decision made.

model	3 Relu
error	cross-entropy softmax
hidden units per layer	200
learning rule	AdaGrad
data augmentation	horizontal reflection
batch size	50
result accuracy	55%

Table 1: Model parameters

Questions

Colour Spaces

A number of papers mention preprocessing the CIFAR dataset by converting each image into grey-scale[1][2]. These papers suggest the worth of slightly drop-off in performance for the time saving caused by having 3 times smaller input size. Three colour spaces were explored : the original RGB, HSV and greyscale. Within the greyscale spectrum exist numerous algorithms to transform a RGB image into greyscale[3]. Despite the differences between the conversion algorithms their effect on classification performance was minimal. All of the algorithms were implemented as scripts ran on the original CIFAR dataset, creating a new pre-processed one. The comparison of the colour space impact on classification performance is displayed on Figure 1 and with respect to time on Table 2.

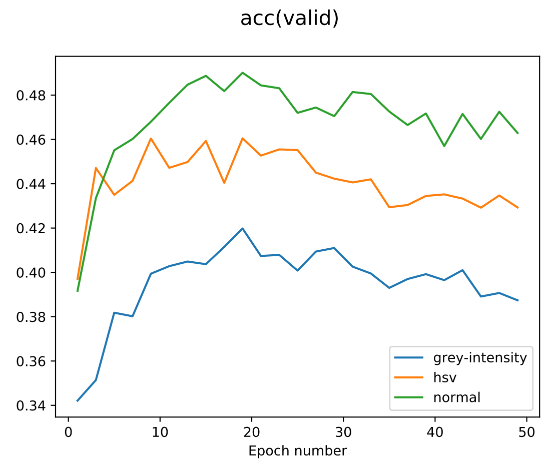

Figure 1: Effect of pre-porcessing the CIFAR dataset by transforming into different colour spaces on classification performance. Trained using AdamOptimizer with 3 layers of 200 units each.

RGB - original	243083ms
HSV	270901ms
greyscale	138732ms

Table 2: Effect of pre-porcessing the CIFAR dataset by transforming into different colour spaces on training time.

Greyscale pre-processing shows a significant time saving, however a larger decrease in classification performance than initially expected. For this reason the greyscale dataset will be used to find an appropriate architecture which will be then later trained on the original RGB dataset.

Architectures

Different model architectures were explored in terms of number of layers and number of hidden units. The exploration was initially done on the grayscale pre-processed dataset after showing that the explored architectures respond similarly to grayscale and RGB input data. A greater number of architectures were explored by leveraging the time saving of having a smaller input dimension of the grayscale datasest. The found best performing greyscale based architectures were compared to the RGB space ones to ensure this correlation was not broken and that an optimal architecture would be found for the RGB space. The explored experiment results are displayed on Figure 2, showing that too wide or deep networks are too complex for our dataset as they decrease the classification performance.

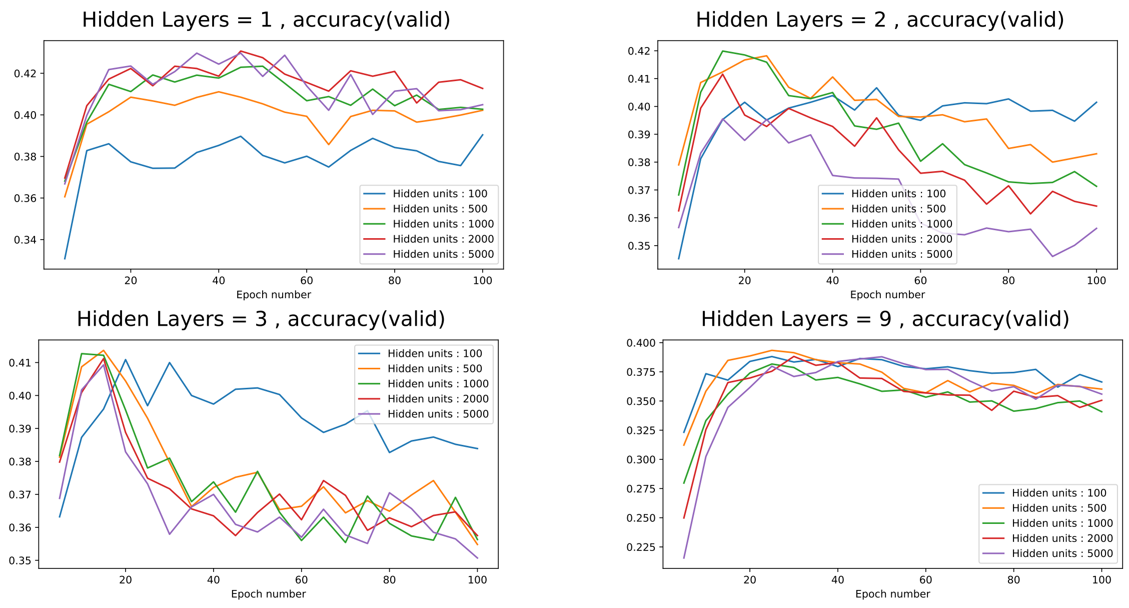

Figure 2: Classification accuracy comparison on architectures with different numbers of layers and hidden units.

The original RGB dataset was then tested on the well performing networks. The results pointing towards several similar good candidates 2 layer architecture of 500 units, or 3 architecture of 200. As these produce good results whilst keeping the number of hidden unit to a minimum. A number of 3 layer architectures were experimented with to confirm this on the RGB dataset as can be seen on Figure 3.

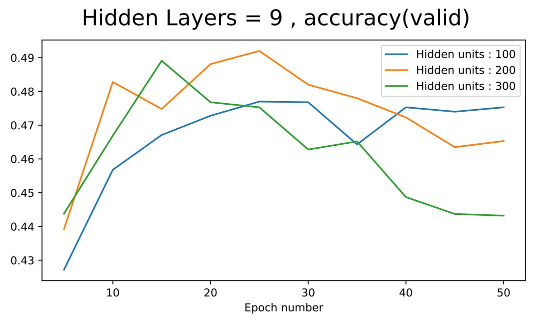

Figure 3: Classification accuracy for RGB dataset on a 3 layer architecture.

Based on the experiments the chosen structure was 3 layers of 200 hidden units each for the original dataset allowing for quite high training speed due to a small overall number of hidden units.

Learning rate update rules

A number of different learning rate update rules or optimizers were explored to see their effect on this dataset. The tested optimizers were : Gradient Descent, Momentum, Adam[5], AdaGrad[4], AdaDelta[6]. The classification accuracy using each of these with their default parameters can be seen on Figure 4. The results showing signifcant improvements over Adam using gradient descent or AdaGrad optimizers. For all further experiments AdaGrad was chosed due to it's performance and robustness with a learning rate of 0.01. These expriments have shown that the right choice of a learning rate update rule can have a significant effect on the model's accuracy.

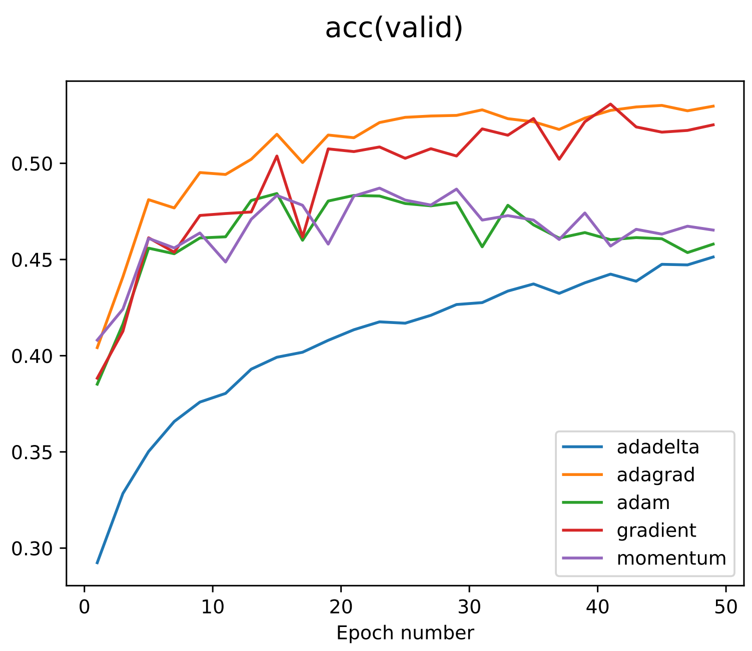

Figure 4: Classification accuracy for different Learning rate update rules trained on a 3 layer network of 200 hidden units per layer.

Data Augmentation

Data Augmentation can be seen as a form of regularization and fights sparse training sets. Both of these features are useful here, regularization will prevent our model for CIFAR-10 over-fitting to the training set, while creating new data will be especially useful for the sparse CIFAR-100. In the previous coursework ,various common data augmentations were explored with regards to the MNIST dataset : rotation, shifting, random noise. Rotation for example performed exceptionally well due to people often writing digits under an angle. While random noise had little effect since the dataset has already been cleaned. These examples suggest that when thinking about data augmentations one needs to think what makes sense with regards to the dataset. In the case of CIFAR, rotations would not be expected to help as they would correspond to the picture just being taken under a slight angle. However random noise[2] (salt and pepper) could be of use here as the dataset has not been cleaned. Papers[7] refer to horizontal reflection as to a helpful data augmentation, which is very logical if one images for example a photo of a horse, flipping this image would transform it as if it were standing the opposite way which is a common occurrence on images. Figure 5 shows a couple of data augmentation functions with their best performing parameters all applied randomly to 25% of each batch. The specific parameters for contrast being multiplying by a factor of 0.1 and for noise adding/subtracting a small factor less than 0.1.

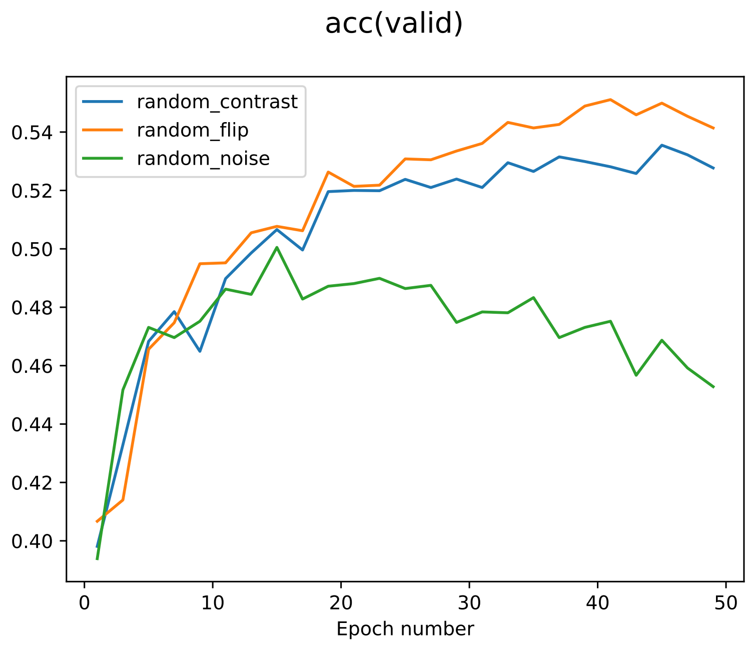

Figure 5: Classification accuracy when applying different functions for data augmentation.

Other explored augmentations were random brightness and a mix of well performing augmentation, randomly applying a flip or change of contrast. However, neither of these produced a better result than just flipping on it's own which is what was added to the final model.

Regularization

The topic of regularization was also explored. Data augmentation can be seen as a form of regularization on it's own. L2 and dropout were introduced by themselves and in unison to see whether they can prevent over-fitting and increase the classification accuracy. Figure 6 compares the model so far and the model with applied regularization.

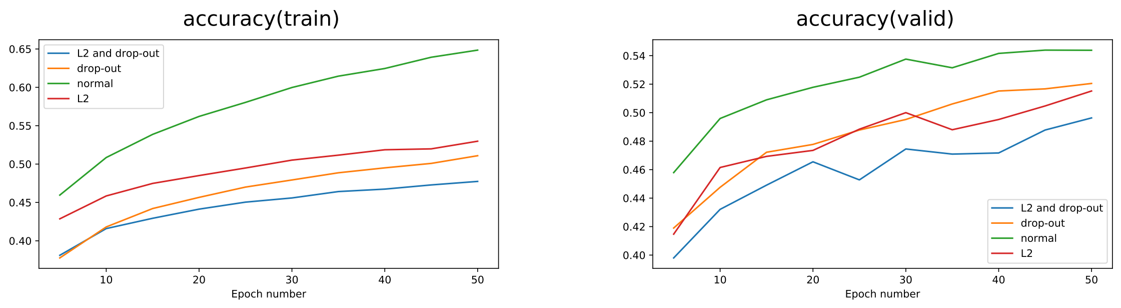

Figure 6: Effects of L2 and dropout regularization

The results showed decrease in performance when using both L2 or dropout alone and together in contrast to a network without these. These methods made the model too rigid, it was not over-fitting but it was also unable to properly capture the data. Therefor, regularization was not introduced to our final model.

Implementation

The network in use has been built using Tensorflow, this section touches on what has been added compared to the vanilla code provided. The features mentioned will go in line with the order of sections above. Data pre-processing consisted of a script which read the original npz file to transform in to a different colour space. The dataset has been pre-processed as doing this operation on-the-fly proved to be too slow. The model consisted of a number of fully connected layers as were provided in the vanilla code. Tensorflow's train library was leveraged for all optimizers used, needing to change only one line in order to switch to a new learning rate schedule. Similarly for regularization, Tensorflow's nn library was used for the L2 and dropout functions.

References

[1] Shawn McCann, Jim Reesman *Object Detection using Convolutional Neural Networks*

[2] Dan Ciresan, Ueli Meier, Jurgen Schmidhuber *Multi-column Deep Neural Networks for Image Classification*, 2012

[3] Tanner Helland *Seven grayscale conversion algorithms (with pseudocode and VB6 source code)*, http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/

[4] Duchi et al, http://jmlr.org/papers/v12/duchi11a.html

[5] Kingma, Ba, https://arxiv.org/abs/1412.6980

[6] Matthew D. Zeiler *ADADELTA An Adaptive Learning Rate Method*

[7] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton *ImageNet Classification with Deep Convolutional Neural Networks*