Machine Learning Practical - Assignment 4

s1247438

March 21, 2017

Introduction

This coursework continues to explore more advanced topics in Neural Networks following the work from the last assignment [1]. The experiments are focused on the CIFAR-10 and CIFAR-100 datasets with a 40,000 training , 10,000 validation and 10,000 test split. These contain sets of 32x32 labeled images falling into 10 and 100 categories respectively. The investigated methods will try to improve upon the baseline model found in the previous coursework, taking advantage of what has been learned during the model development. The baseline model is listed in Table 1, the main takeaways for further experiments are the number of layers, number of hidden units and data augmentation (horizontal reflection, random brightness, random contrast).

model	3 Relu
error	cross-entropy softmax
hidden units per layer	200
learning rule	AdaGrad
data augmentation	horizontal reflection
batch size	50
result accuracy C-10	55%

Table 1: Baseline Model parameters

This coursework focuses on exploring topics regarding Convolutional Neural Networks(CNN) expecting a dramatic increase in performance that CNNs are reported to bring on image classification tasks[2][4][5]. More specifically the coursework will explore the following topics:

- What is a well performing network architecture in terms of the number of convolution, pooling and fully connected layers? This topic will use the previously learned baseline model as a starting point expecting to reach a conclusion in terms of number of layers similar to what papers exploring CIFAR suggest[2]. A good model architecture is needed for further topics to be explored.
- Can further data augmentation prevent over-fitting? Despite the data aumentations explored, the previous model was still prone to overfitting. One approach that has not been explored in the previous model but reported as benefical is random cropping[3][7]

- What is an appropriate convolution filter size and stride? This question will expect to arrive to the same conclusions as many papers suggesting only few reasonable values for these [2][4][5].
- What is an appropriate pooling type, kernel size and stride? As with the question above the expectation is to default to standard values. For this reason the question will further explore the possible benefits of fractional pooling over max pooling as suggested by a recent paper[5].
- How does the performance of an all convolutional network[10] compare? The all convolutional network replaces pooling layers by convolution layers with stride 2 and reports similar results as network with pooling layers.

Implementation

The code for exploring Convolutional Neural Networks is an adaptation from the official tensorflow tutorial[6]. The tutorial code was chosen as a basis over the one developed for the previous model and data provider for performance and structure reasons. The tutorial code takes advantages of multi-threaded way of creating queues using *tf.train.batch()* and more efficient methods for data augmentation from *tf.image* library. The code also uses *tf.train.MonitoredTrainingSession* and hooks to output data during training which seems as a better practice, as well as better integration with tensorboard and storing checkpoints. Taking advantage of *tf.app.flags* the model can be instructed from the command line to what to parameters to test, improving scalability. This code was then adapted to work on a basis of epochs rather than step and tracking accuracy and loss on a per epoch basis. The LoggerHook attached to the MonitoredTrainingSession keeps track of undertaken steps and epochs. On each epoch it stores the training accuracy and loss per epoch, as well as runs a validation script that loads the last stored model and stores the validation accuracy and loss. These can be later easily compared in tensorboard, the only limitation is that tensorboard does not label x axis, which in this case are the number of epochs. This code was then deployed on Amazon AWS p2.xlarge GPU instance taking 60-100 seconds per epoch.

Methods

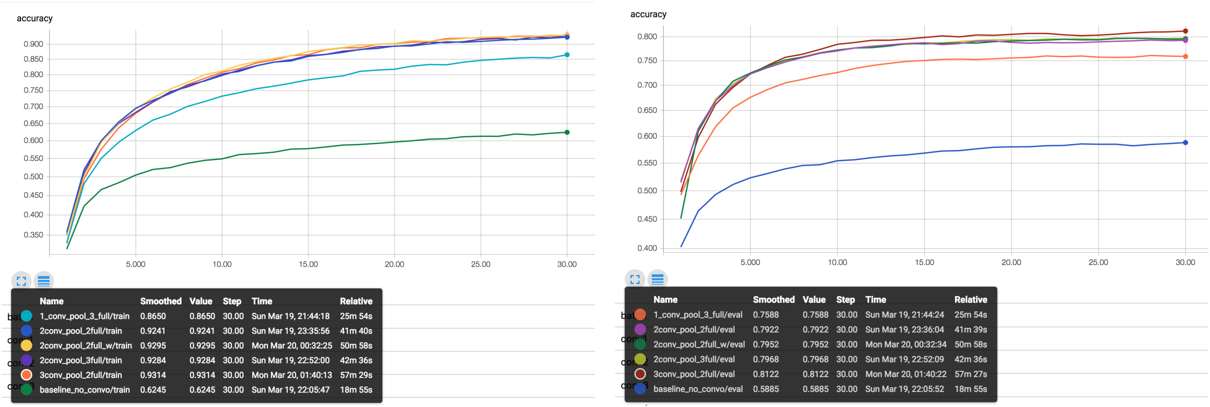
Network Architecture

Various network architectures were explored mainly the number of convolution and pooling layers. The model found in previous coursework depicted on Table 1 was used as both a baseline and the fully connected layers. Three fully connected layers of 200 hidden units each were used at the start. Explored models were trained on an augmented dataset (flip,contrast,brightness). The experiments consisted of iteratively adding convolution and pooling layers before the fully connected ones taking advantage of the tensorflow functions *tf.nn.conv2d(input, filter, strides, padding, usecudnnongpu=None, dataformat=None, name=None)* and *tf.nn.maxpool(value, ksize, strides, padding, dataformat='NHWC', name=None)*. However these functions require parameters for filter, stride and kernel size values. The effect of these and an appropriate choice will be explored in the latter sections. The initial values for these were taken from a paper which had a well performing model for CIFAR [2] displayed on Table 2. The maximum number of convolution and pooling layers were three as after 3 pooling layers the resulting data size was too small for the chosen kernel size.

Filter size	5
Filter stride	1
Number of filters	64
Kernel size	3
Kernel stride	2
Batch size	128

Table 2: Initial parameters for Convolution and Pooling layers

The results of different architectures are displayed on Figure 1. All models were run only for 30 epochs due to computational complexity, relying that the number of epochs will be enough to compare architectures and see trends occurring. These show a significant improvement on validation set accuracy over the non-covolutional baseline model which achieved 58% and the 3 convolution and pooling layers coupled with two fully connected layers of 200 hidden units achieved 81% accuracy in just 30 epochs. This architecture will be used across the next sections.



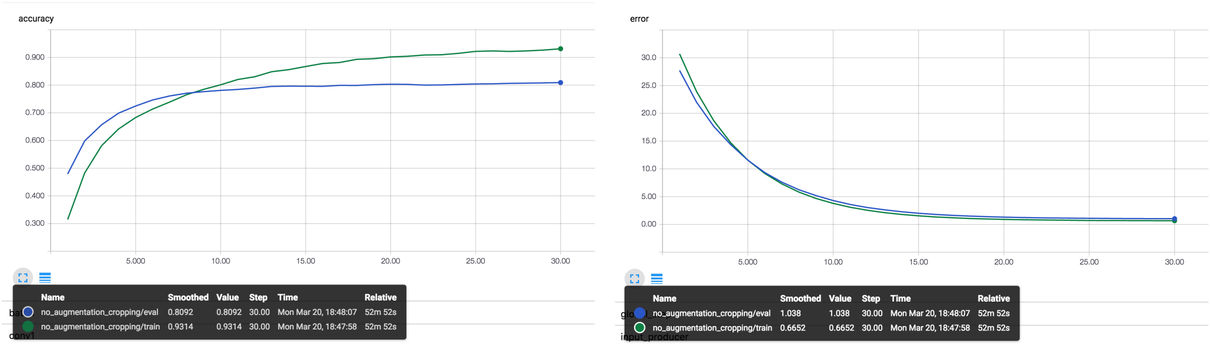
	Name	Smoothed	Value	Step	Time	Relative
●	1_conv_pool_3_full/train	0.8650	0.8650	30.00	Sun Mar 19, 21:44:18	25m 54s
●	2conv_pool_2full/train	0.9241	0.9241	30.00	Sun Mar 19, 23:35:56	41m 40s
●	2conv_pool_2full_w/train	0.9295	0.9295	30.00	Mon Mar 20, 00:32:25	50m 58s
●	2conv_pool_3full/train	0.9284	0.9284	30.00	Sun Mar 19, 22:52:00	42m 36s
●	3conv_pool_2full/train	0.9314	0.9314	30.00	Mon Mar 20, 01:40:13	57m 29s
●	baseline_no_convo/train	0.6245	0.6245	30.00	Sun Mar 19, 22:05:47	18m 55s

	Name	Smoothed	Value	Step	Time	Relative
●	1_conv_pool_3_full/eval	0.7588	0.7588	30.00	Sun Mar 19, 21:44:24	25m 54s
●	2conv_pool_2full/eval	0.7922	0.7922	30.00	Sun Mar 19, 23:36:04	41m 39s
●	2conv_pool_2full_w/eval	0.7952	0.7952	30.00	Mon Mar 20, 00:32:34	50m 58s
●	2conv_pool_3full/eval	0.7968	0.7968	30.00	Sun Mar 19, 22:52:09	42m 36s
●	3conv_pool_2full/eval	0.8122	0.8122	30.00	Mon Mar 20, 01:40:22	57m 27s
●	baseline_no_convo/eval	0.5885	0.5885	30.00	Sun Mar 19, 22:05:52	18m 55s

Figure 1: Comparison of different architecture model accuracies for 30 epochs, training set on the left and validation on the right. Highlighting the best performing one

Data Augmentation

Previous coursework[1] explored various data augmentation methods and found flipping, brightness and contrast to be effective in decreasing the level of over-fitting and improving accuracy. Exploring the graphs on Figure 2 one can see the model is still over-fitting to the training data. One popular data augmentation method mentioned in a number of papers [3][7] is random cropping. This section explored performing a random 24x24 crop of the original 32x32 images using *tf.random_ crop()*, achieving a significant increase of the training data.

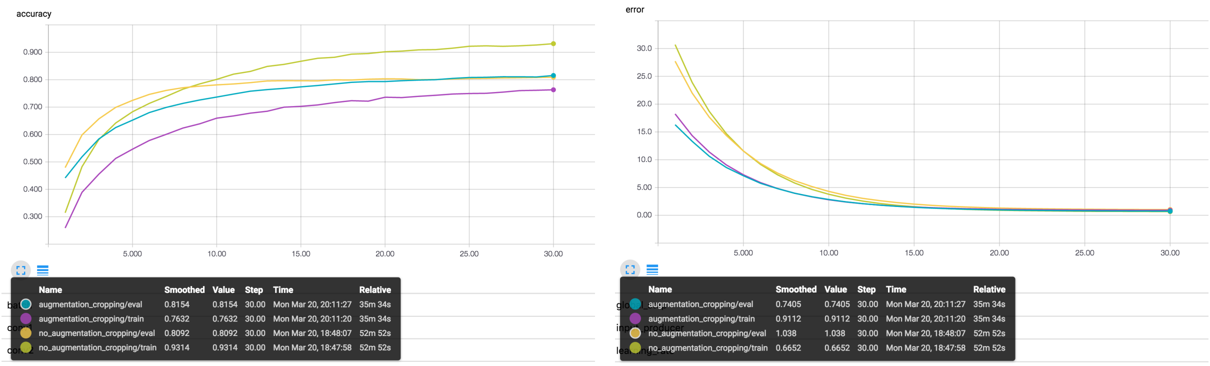


	Name	Smoothed	Value	Step	Time	Relative
●	no_augmentation_cropping/eval	0.8092	0.8092	30.00	Mon Mar 20, 18:48:07	52m 52s
●	no_augmentation_cropping/train	0.9314	0.9314	30.00	Mon Mar 20, 18:47:58	52m 52s

	Name	Smoothed	Value	Step	Time	Relative
●	no_augmentation_cropping/eval	1.038	1.038	30.00	Mon Mar 20, 18:48:07	52m 52s
●	no_augmentation_cropping/train	0.6652	0.6652	30.00	Mon Mar 20, 18:47:58	52m 52s

Figure 2: Model performance without random cropping data augmentation showing overfitting.

Figure 3 shows the effects of random cropping data augmentation. The model learns a little slower but even in the small number of epoch matches and starts overtaking the classification performance of the one without random cropping. The model results in a higher validation than training accuracy clearly benefiting with the essentially larger dataset. Random cropping augmentation was applied for the next sections.



	Name	Smoothed	Value	Step	Time	Relative
●	augmentation_cropping/eval	0.8154	0.8154	30.00	Mon Mar 20, 20:11:27	35m 34s
●	augmentation_cropping/train	0.7632	0.7632	30.00	Mon Mar 20, 20:11:20	35m 34s
●	no_augmentation_cropping/eval	0.8092	0.8092	30.00	Mon Mar 20, 18:48:07	52m 52s
●	no_augmentation_cropping/train	0.9314	0.9314	30.00	Mon Mar 20, 18:47:58	52m 52s

	Name	Smoothed	Value	Step	Time	Relative
●	augmentation_cropping/eval	0.7405	0.7405	30.00	Mon Mar 20, 20:11:27	35m 34s
●	augmentation_cropping/train	0.9112	0.9112	30.00	Mon Mar 20, 20:11:20	35m 34s
●	no_augmentation_cropping/eval	1.038	1.038	30.00	Mon Mar 20, 18:48:07	52m 52s
●	no_augmentation_cropping/train	0.6652	0.6652	30.00	Mon Mar 20, 18:47:58	52m 52s

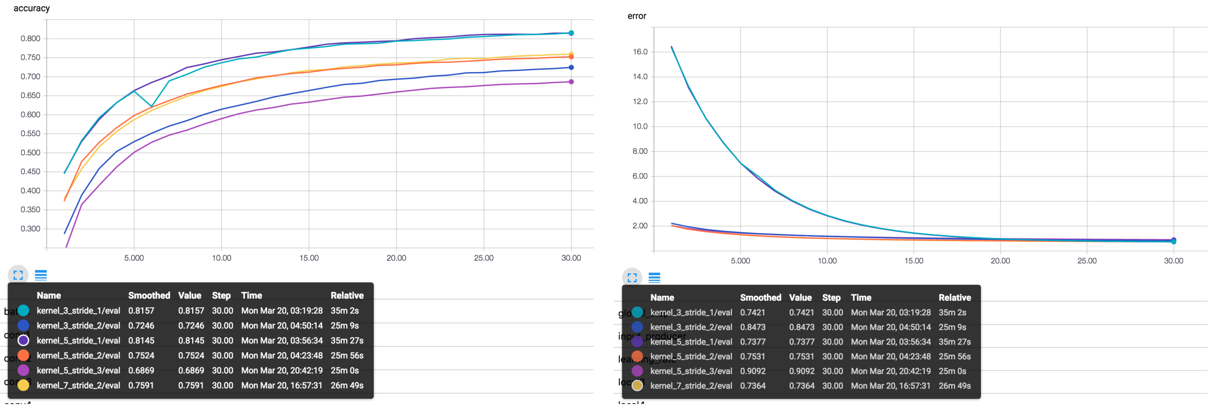
Figure 3: Model performance with random cropping data augmentation.

Convolution filter size and stride length

Papers[3][2][4] mentioning models on CIFAR and others often suggest a 5x5 filter size. The Stanford class on Convolutional Neural networks[8] similarly suggests using a 3x3, 5x5 or 7x7 filters with a stride of 1. A larger filter sizes are described as uncommon, AlexNet[9] used a filter size of 11x11 on the first convolutional layer, but the size of their input for ImageNet was much larger 224x224. This section experiments with the smaller filter sizes and strides. The results are shown on Figure 4 confirming the initial hypothesis and suggesting a 3x3 or 5x5 filter with stride 1 is a reasonable choice.

Pooling kernel size, stride length and fractional pooling

Similarly as with the section above the common kernel sizes and stride lengths suggested[3][2][8] and used are size 2x2 or 3x3 and stride 2 for max pooling. Max pooling was implemented

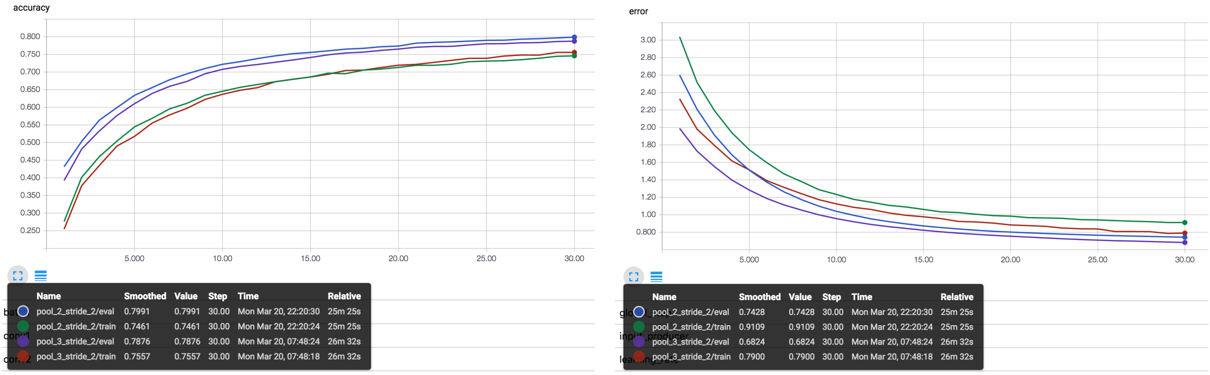


	Name	Smoothed	Value	Step	Time	Relative
●	kernel_3_stride_1/eval	0.8157	0.8157	30.00	Mon Mar 20, 03:19:28	35m 2s
●	kernel_3_stride_2/eval	0.7246	0.7246	30.00	Mon Mar 20, 04:50:14	25m 9s
●	kernel_5_stride_1/eval	0.8145	0.8145	30.00	Mon Mar 20, 03:56:34	35m 27s
●	kernel_5_stride_2/eval	0.7524	0.7524	30.00	Mon Mar 20, 04:23:48	25m 56s
●	kernel_5_stride_3/eval	0.6869	0.6869	30.00	Mon Mar 20, 20:42:19	25m 0s
●	kernel_7_stride_2/eval	0.7591	0.7591	30.00	Mon Mar 20, 16:57:31	26m 49s

	Name	Smoothed	Value	Step	Time	Relative
●	kernel_3_stride_1/eval	0.7421	0.7421	30.00	Mon Mar 20, 03:19:28	35m 2s
●	kernel_3_stride_2/eval	0.8473	0.8473	30.00	Mon Mar 20, 04:50:14	25m 9s
●	kernel_5_stride_1/eval	0.7377	0.7377	30.00	Mon Mar 20, 03:56:34	35m 27s
●	kernel_5_stride_2/eval	0.7531	0.7531	30.00	Mon Mar 20, 04:23:48	25m 56s
●	kernel_5_stride_3/eval	0.9092	0.9092	30.00	Mon Mar 20, 20:42:19	25m 0s
●	kernel_7_stride_2/eval	0.7364	0.7364	30.00	Mon Mar 20, 16:57:31	26m 49s

Figure 4: Model performance on validation set with various filter sizes and stride lengths.

using *tf.nn.max_ pool()* and the expected results are that overlapping pooling 3x3 with stride 2 will perform better than non-overlapping 2x2 kernel with stride 2[5]. The results are shown in Figure 5.



	Name	Smoothed	Value	Step	Time	Relative
●	pool_2_stride_2/eval	0.7991	0.7991	30.00	Mon Mar 20, 22:20:30	25m 25s
●	pool_2_stride_2/train	0.7461	0.7461	30.00	Mon Mar 20, 22:20:24	25m 25s
●	pool_3_stride_2/eval	0.7876	0.7876	30.00	Mon Mar 20, 07:48:24	26m 32s
●	pool_3_stride_2/train	0.7557	0.7557	30.00	Mon Mar 20, 07:48:18	26m 32s

	Name	Smoothed	Value	Step	Time	Relative
●	pool_2_stride_2/eval	0.7428	0.7428	30.00	Mon Mar 20, 22:20:30	25m 25s
●	pool_2_stride_2/train	0.9109	0.9109	30.00	Mon Mar 20, 22:20:24	25m 25s
●	pool_3_stride_2/eval	0.6824	0.6824	30.00	Mon Mar 20, 07:48:24	26m 32s
●	pool_3_stride_2/train	0.7900	0.7900	30.00	Mon Mar 20, 07:48:18	26m 32s

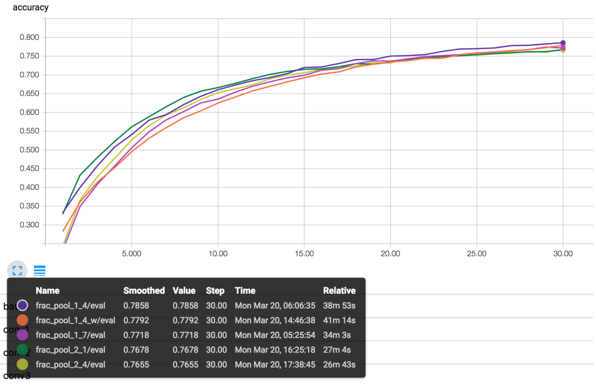
Figure 5: Comparison of two standard pooling parameters 2x2 and 3x3 with stride 2.

Max pooling has been shown to be very advantageous however the standard approaches usually reduce the input size by a factor of 2 which might be too rapid. One paper[5] suggests replacing max pooling with fractional pooling that allows to decrease the input size by an non integer factor for example $\sqrt{2}$. This would allow to have essential twice as many convolutional and pooling layers each viewing the input image at a different scale. Fractional pooling was implemented using *tf.nn.fractional_ max_ pool()* and tested with different parameters allowing for a different number of pooling layers based on how quickly they decrease the input size as shown in Table 2. And the pooling region choice was set to pseudo-random as the paper suggests for models which use data augmentation.

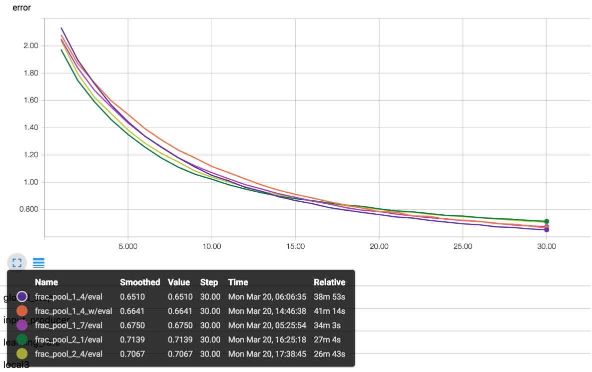
Pooling parameter	Pooling layers
1.4	6
1.7	5
2.1	3
2.4	2

Table 3: Fractional pooling parameters

The fractional pooling results are displayed on Figure 6.



	Name	Smoothed	Value	Step	Time	Relative
●	frac_pool_1_4/eval	0.7858	0.7858	30.00	Mon Mar 20, 06:06:35	38m 53s
●	frac_pool_1_4_w/eval	0.7792	0.7792	30.00	Mon Mar 20, 14:46:38	41m 14s
●	frac_pool_1_7/eval	0.7718	0.7718	30.00	Mon Mar 20, 05:25:54	34m 3s
●	frac_pool_2_1/eval	0.7678	0.7678	30.00	Mon Mar 20, 16:25:18	27m 4s
●	frac_pool_2_4/eval	0.7655	0.7655	30.00	Mon Mar 20, 17:38:45	26m 43s

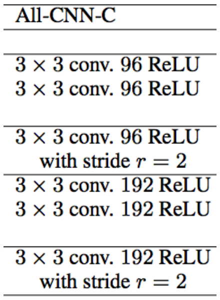


	Name	Smoothed	Value	Step	Time	Relative
●	frac_pool_1_4/eval	0.6510	0.6510	30.00	Mon Mar 20, 06:06:35	38m 53s
●	frac_pool_1_4_w/eval	0.6641	0.6641	30.00	Mon Mar 20, 14:46:38	41m 14s
●	frac_pool_1_7/eval	0.6750	0.6750	30.00	Mon Mar 20, 05:25:54	34m 3s
●	frac_pool_2_1/eval	0.7139	0.7139	30.00	Mon Mar 20, 16:25:18	27m 4s
●	frac_pool_2_4/eval	0.7067	0.7067	30.00	Mon Mar 20, 17:38:45	26m 43s

Figure 6: Fractional pooling performance with different parameters and number of layers.

The all conovolutional network

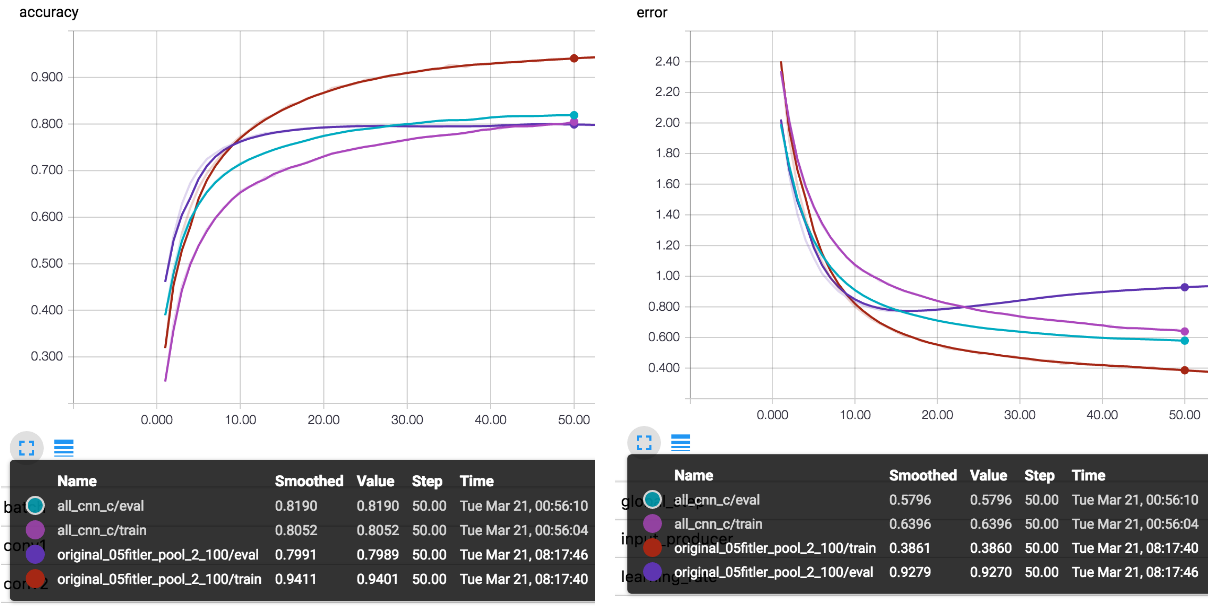
This section is inspired by the starving for simplicity: The all convolutional network paper [10]. The paper challenges the standard widely used pipeline of convolutional, pooling and fully connected layers. Suggesting that max pool layers can be replaced by a convolutional layer with a stride 2 to achieve the same result. An advantage is that since the input size is not being reduced (a problem fractional pooling was trying to address), allowing to build deeper networks. Implementation in tensorflow meant replacing the existing maxpool layers with conv2d layers with a stride 2. The architecture is the paper proposed All-CNN-C as described in Table 7.



All-CNN-C
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
3×3 conv. 192 ReLU with stride $r = 2$

Figure 7: The All-CNN-C architecture

The comparison of performances between our standard pipeline model and and the all convolutional network can be seen on Figure 8. Showing a 2% increase in validation accuracy of the All-CNN-C model over the standard pipeline model explored in this coursework.



	Name	Smoothed	Value	Step	Time
●	all_cnn_c/eval	0.8190	0.8190	50.00	Tue Mar 21, 00:56:10
●	all_cnn_c/train	0.8052	0.8052	50.00	Tue Mar 21, 00:56:04
●	original_05fitler_pool_2_100/eval	0.7991	0.7989	50.00	Tue Mar 21, 08:17:46
●	original_05fitler_pool_2_100/train	0.9411	0.9401	50.00	Tue Mar 21, 08:17:40

	Name	Smoothed	Value	Step	Time
●	all_cnn_c/eval	0.5796	0.5796	50.00	Tue Mar 21, 00:56:10
●	all_cnn_c/train	0.6396	0.6396	50.00	Tue Mar 21, 00:56:04
●	original_05fitler_pool_2_100/train	0.3861	0.3860	50.00	Tue Mar 21, 08:17:40
●	original_05fitler_pool_2_100/eval	0.9279	0.9270	50.00	Tue Mar 21, 08:17:46

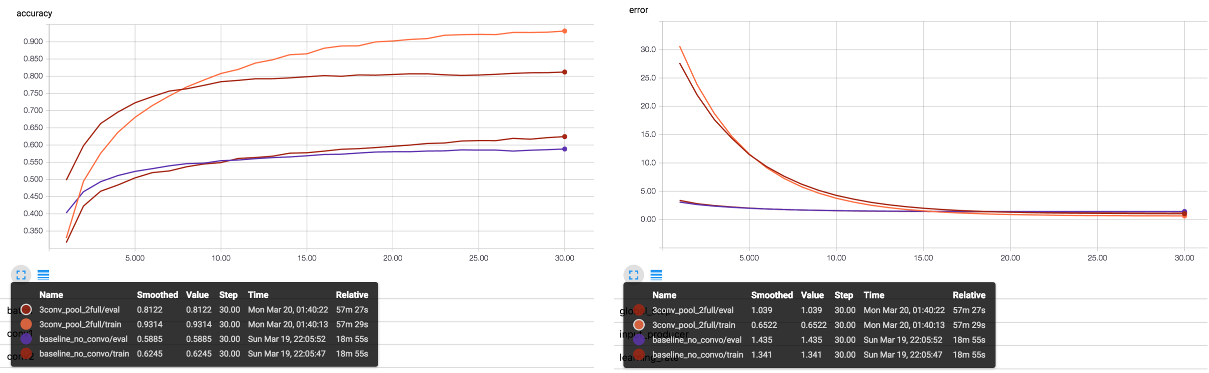
Figure 8: Comparison of the All-CNN-C and the standard pipeline model explored in this coursework.

Discussion

The experiments were performed on CIFAR-10 as CIFAR-100 not only required a larger number of epochs to learn, but also due to an expanded number of classes can be viewed as a different dataset with it's own range of problems to be addressed. An acknowledged limitation of some of the experiments performed was the rather small number of 30 epochs. Due to computational and time complexity it was not possible to perform all of these with a larger number of epochs, with the hopes that the trends would be apparent even with this smaller number. Running some models for longer periods of time as will be shown in discussion confirm this intuition.

Network architecture

The network architecture section shows a clear motivation for using conovlutional neural networks compared to just networks with fully connected layers explored in the previous assignment. Multiple iterations of convolution and pooling layers followed by fully connected layers were explored. More convolutional and pooling layers allowed the network to learn features and feature of features of the images, allowing to learn features like corners or edges. The larger number of of these convolutional layers proved to be beneficial to a cut off point when the input size became too small for convolutional filters. Increasing the number of fully connected layers following these beyond 2 did not improve the results likely due to the small input entering them. The comparison of our previously found model[1] and the 3 convolution/pooling layer followed by 2 fully connected layers is shown on Figure 9. The advantages of conolutional networks were expected as per a large number of papers reporting positive results[2][9][8].



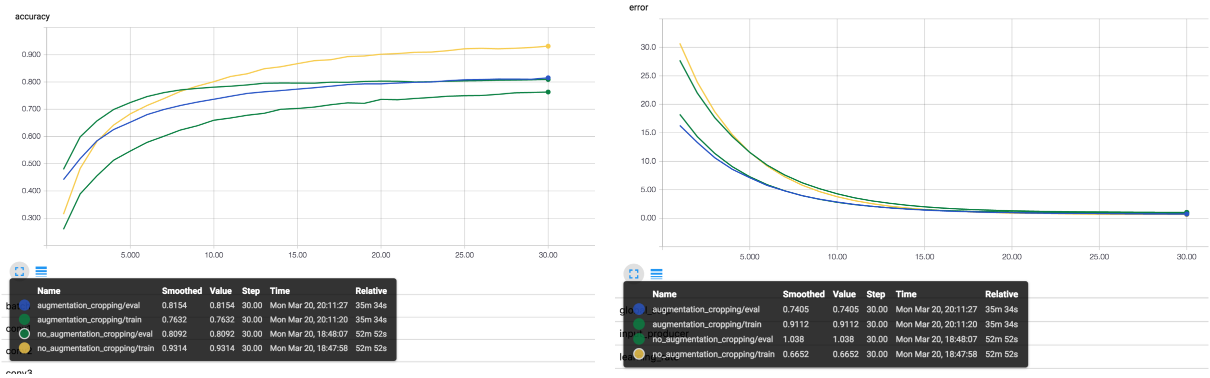
	Name	Smoothed	Value	Step	Time	Relative
●	3conv_pool_2full/eval	0.8122	0.8122	30.00	Mon Mar 20, 01:40:22	57m 27s
●	3conv_pool_2full/train	0.9314	0.9314	30.00	Mon Mar 20, 01:40:13	57m 29s
●	baseline_no_convo/eval	0.5885	0.5885	30.00	Sun Mar 19, 22:05:52	18m 55s
●	baseline_no_convo/train	0.6245	0.6245	30.00	Sun Mar 19, 22:05:47	18m 55s

	Name	Smoothed	Value	Step	Time	Relative
●	3conv_pool_2full/eval	1.039	1.039	30.00	Mon Mar 20, 01:40:22	57m 27s
●	3conv_pool_2full/train	0.6522	0.6522	30.00	Mon Mar 20, 01:40:13	57m 29s
●	baseline_no_convo/eval	1.435	1.435	30.00	Sun Mar 19, 22:05:52	18m 55s
●	baseline_no_convo/train	1.341	1.341	30.00	Sun Mar 19, 22:05:47	18m 55s

Figure 9: Comparison of baseline and conovlutional network

Data Augmentation

As can be observed from Figure 9 the conolutional model was still prone to overfitting despite the number of data augmentation methods already applied. One popular method that was not applied was random cropping used by many papers[3][7]. This method is able to mimic a significantly larger dataset and the results are shown on Figure 10. These show a small increase in performance over the small number of epochs but more importantly by an essentially larger dataset the model was no longer overfitting, which should be even more apparent with a larger number of epochs ran. The reported results confirm the initial hypothesis of this being an effective data augmentation method as was suggested by the papers.



	Name	Smoothed	Value	Step	Time	Relative
●	augmentation_cropping/eval	0.8154	0.8154	30.00	Mon Mar 20, 20:11:27	35m 34s
●	augmentation_cropping/train	0.7632	0.7632	30.00	Mon Mar 20, 20:11:20	35m 34s
●	no_augmentation_cropping/eval	0.8092	0.8092	30.00	Mon Mar 20, 18:48:07	52m 52s
●	no_augmentation_cropping/train	0.9314	0.9314	30.00	Mon Mar 20, 18:47:58	52m 52s

	Name	Smoothed	Value	Step	Time	Relative
●	augmentation_cropping/eval	0.7405	0.7405	30.00	Mon Mar 20, 20:11:27	35m 34s
●	augmentation_cropping/train	0.9112	0.9112	30.00	Mon Mar 20, 20:11:20	35m 34s
●	no_augmentation_cropping/eval	1.038	1.038	30.00	Mon Mar 20, 18:48:07	52m 52s
●	no_augmentation_cropping/train	0.6652	0.6652	30.00	Mon Mar 20, 18:47:58	52m 52s

Figure 10: Effects of using random cropping.

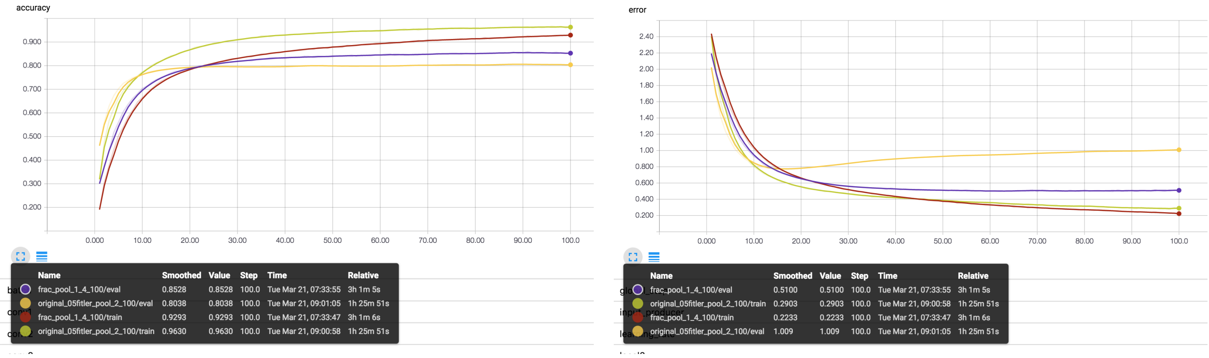
Convolution filter size and stride length

This section aimed to confirm the generally accepted values[3][2][4] for the filter sizes of 3x3 and 5x5. As was shown in Figure 4 the results confirm this notion with respect to this dataset that these two values perform both really well, whilst increasing the filter size does not. Larger filter sizes can still be beneficial[9] mainly in cases with larger input sizes. Increasing stride length did not improve the results as at that point the convolutional filters starts behaving more like a polling layer. Further experiments could have explored varying filter sizes, having a larger filter size in the first layer and a small one at the last one. The experiments could have also explored filter depths which were here kept to 64[2].

Pooling kernel size, stride length and fractional pooling

As with the section above the aim was to first explore the two generally used values for pooling 2x2 and 3x3 with stride 1 and 2 respectively. The expected results were that the 3x3 overlapping pooling will outperform the 2x2 as mentioned in some papers[2][5]. The results on Figure 5 reported similar performances. The larger pooling kernel size was expected to improve generalization, however due to a limited number of epochs and effective data augmentation this might have not became apparent.

Fractional pooling[5] was then explored which allowed to reduce the size of input by a factor smaller than 2 and hence introduce more conovlutional layers. Comparison of fractional and max pooling is shown on Figure 11. The results showed a significant improvement over standard pooling approach. The best performing was pooling with factor of 1.4 and 6 conovlutional/pooling layers. These results confirm the expectation from the mentioned paper which currently reports one of the highest accuracies on CIFAR-10.



	Name	Smoothed	Value	Step	Time	Relative
●	frac_pool_1_4_100/eval	0.8528	0.8528	100.0	Tue Mar 21, 07:33:55	3h 1m 5s
●	original_05fitler_pool_2_100/eval	0.8038	0.8038	100.0	Tue Mar 21, 09:01:05	1h 25m 51s
●	frac_pool_1_4_100/train	0.9293	0.9293	100.0	Tue Mar 21, 07:33:47	3h 1m 6s
●	original_05fitler_pool_2_100/train	0.9630	0.9630	100.0	Tue Mar 21, 09:00:58	1h 25m 51s

	Name	Smoothed	Value	Step	Time	Relative
●	frac_pool_1_4_100/eval	0.5100	0.5100	100.0	Tue Mar 21, 07:33:55	3h 1m 5s
●	original_05fitler_pool_2_100/train	0.2903	0.2903	100.0	Tue Mar 21, 09:00:58	1h 25m 51s
●	frac_pool_1_4_100/train	0.2233	0.2233	100.0	Tue Mar 21, 07:33:47	3h 1m 6s
●	original_05fitler_pool_2_100/eval	1.009	1.009	100.0	Tue Mar 21, 09:01:05	1h 25m 51s

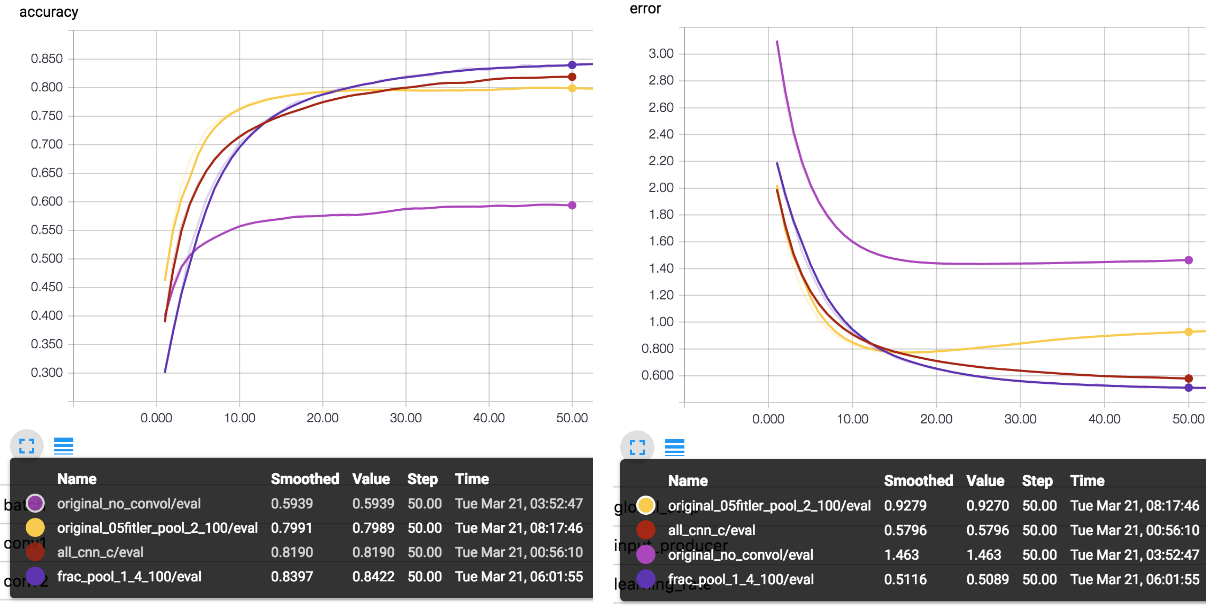
Figure 11: Comparison of fractional and max pooling.

The all conovolutional network

The all convolutional network[10] is an interesting approach which replaces pooling layers with conovlutional layers with stride of 2. Suggesting that it might replace and simplify the standard convolutional neural network pipeline[8]. The paper reported very positive results on CIFAR and hence was explored in this setting and compared to the previously explored models. The architecture was directly adapted from the paper, but the data with augmentations is the same as with previously explored models for comparison purposes. The results confirm the paper's suggestions improved results over our best found model so far, trained on 50 epochs for little more reliable results as shown on Figure 8. Further all conovlutional architectures in terms of number of layers, filter sizes and depths could be explored.

Overall

The comparison of the three found best performing models with the baseline are shown on Figure 12. These models are 3(C-MP2) 2FC(200), 6(C-FP1.4)-2FC(200) and All-CNN-C. All three display similar performance on the rather small 50 number of epochs but a significant improvement over the baseline model. The best performing model was one using fractional pooling which corresponds with highest recorded accuracy in literature[5]. The resulting model is displayed on Table 4. Second best performing was the All-CNN-C which shows a lot of potential and again reporting the high expected results[10].



	Name	Smoothed	Value	Step	Time
●	original_no_convol/eval	0.5939	0.5939	50.00	Tue Mar 21, 03:52:47
●	original_05fitler_pool_2_100/eval	0.7991	0.7989	50.00	Tue Mar 21, 08:17:46
●	all_cnn_c/eval	0.8190	0.8190	50.00	Tue Mar 21, 00:56:10
●	frac_pool_1_4_100/eval	0.8397	0.8422	50.00	Tue Mar 21, 06:01:55

	Name	Smoothed	Value	Step	Time
●	original_05fitler_pool_2_100/eval	0.9279	0.9270	50.00	Tue Mar 21, 08:17:46
●	all_cnn_c/eval	0.5796	0.5796	50.00	Tue Mar 21, 00:56:10
●	original_no_convol/eval	1.463	1.463	50.00	Tue Mar 21, 03:52:47
●	frac_pool_1_4_100/eval	0.5116	0.5089	50.00	Tue Mar 21, 06:01:55

Figure 12: Comparison of the best performing models explored with baseline.

Conv/Pool layers	6
FC layer	1 of 200 hidden units
Activation function	Relu
Data Augmentation	Brightness,Contrast,Whiten,Crop,Flip
Filter size	5
Filter stride	1
Number of filters	64
Pooling ratio	1.4
Batch size	128
Number of Epochs	100
Accuracy	85.3%

Table 4: Initial parameters for Convolution and Pooling layers

Conclusion

This coursework showed a clear motivation for using conolutional neural networks as they achieved over 25% higher accuracy than previously explored netoworks using only fully connected layers, which coresponds to what was expected from literature[2][9][8]. The network architecture section showed that more convolutional/pooling layers tend to increase performance as they allow to learn features and features of features. This trend can be also extracted from later sections where both fractional pooling and all convolutional network benefited with a larger number of convolutional layers. The filter and pooling section confirmed the notion of using generally accepted values for filter and pooling sizes[3][2][4]. These work well for general cases, it can however be beneficial in some cases to modify these : When the input size is larger[9]; When using non integer values to decrease the factor of how quickly the size degrades[5] or when using filter stride 2 to replace pooling layers[10]. Two of these cases of using non-standard parameters for filters and pooling layers were explored. Both fractional pooling and all convolutional network showed improved results over the previous approches as expected from their paper[5][10]. The all convolutional network seems as a very interesting approach greatly simplifying the network pipeline and should be explored further. Overall the best performing model with 85.3% accuracy on 100 epochs being the one using fractional pooling corresponding to what was expected from the literature[5].

References

[1] s1247438 *Machine Learning Practical - Assignment 3*, 2017

[2] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, 2012

[3] Matthew D. Zeiler, Rob Fergus *Stochastic Pooling for Regularization of Deep Convolutional Neural Networks*,2013

[4] Patrice Y. Simard, Dave Steinkraus, John C. Platt *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis*, 2003

[5] Benjamin Graham *Fractional Max-Pooling*, 2015

[6] Convolutional Neural Networks, https://www.tensorflow.org/tutorials/deepcnn

[7] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, *Return of the Devil in the Details: Delving Deep into Convolutional Nets*, 2014

[8] http://cs231n.github.io/convolutional-networks

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

[10] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller *STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET*, 2015