Experiments on CIFAR-10 - Effect of variation of parameters of Convolutional Neural Networks on their performance

Coursework 4 - Machine Learning Practical

s1655337
University of Edinburgh

1 Introduction

The goal of the experiments is to find performance of Convolutional Neural Networks (CNNs) on CIFAR-10 [6] using a similiar network structure to AlexNet [7] but simpler in terms of number of layers and improve upon the results generated previously for different activations functions using feedforward networks [10].

CIFAR-10 consists of a set of 60000 colored images divided into 10 classes with 6000 images in each classes. They are handled with dimensions of width, height, depth of 32 x 32 x 3 for the experiments for the classification task and the 'CIFAR10 Data Provider' provided in the Machine Learning Practical, University of Edinburgh (infr11132) is used to get the image sets for training and validation to be operated upon in Tensorflow. The operations such as Convolution and Fractional Max Pooling are tend to change the shape of the inputs provided to them, so it is important to understand the changes that would be made to the shape of the image to accomodate for these changes.
The motivation to explore CNNs is that they have already been successfully applied to CIFAR-10 with high accuracy [2] over the years. The current state of the art for CIFAR-10 is Fractional Max-Pooling [5] [2] and CIFAR-100 is Exponential Linear Units (ELUs) [3] [2]. Previous results achieved in [10] for RELUs over CIFAR-10 with variation in the structure and operations of simple feed forward network brought in the scope for testing the dataset over more complex architectures such as CNNs and also the good results of RELUs could be pushed further with implementation of Leaky RELUs and ELUs. With ELUs currently giving state of art performance for the CIFAR-100 dataset, and also it's ability to handle vanishing gradient problem like RELUs makes it an interesting choice to start with. Later, the network is modified to see how these changes effect the performance of the network.
Hypothesis: CNN with ELUs is going to outperform the RELUs and Leaky RELUs over big layer networks but overfitting may become an issue over smaller

networks and effect of operations such as max pooling, fractional max pooling and dropout is going to effect the networks heavily. Baseline tests for this model are run over both ELUs and RELUs, but some tests for which reasons for degradation seem obvious from previous tests are done only over RELUs to show the difference in learning curves of error and accuracy over training and validation sets.

We test the network over various parameters in the following sections with configurations and environment mentioned within each section with reasons for choice of that particular configuration. The baseline architecture can be seen in Figure 1.

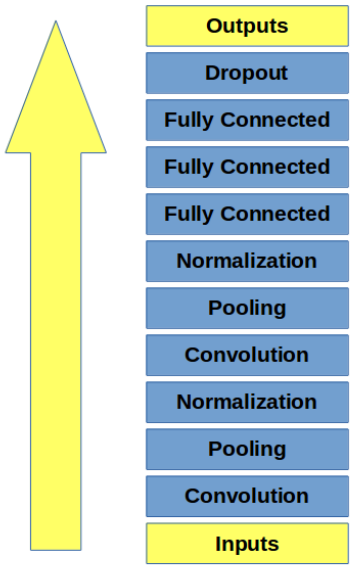


Figure 1: Baseline Architecture Used

2 Baseline Architecture

This sections describes the structural details of the model used for initial formation of a standard. The model used is similar to AlexNet implementation given in Tensforflow Documentation. The methods used for implementing the network are discussed in the next section, but it is important to understand the transition of the network and the changes in shape as 'reshape' function is used to make the network compatible.
The initial input is tensor of images in batches of **50** (this can be set accordingly

depending on the number of inputs one wants to test). To perform convolution, we reshape the image into **32x32x3** where, height and width is **32** and **3** is defined for the RGB channels to be provided to convolution layer. Then the convolution is one on first layer for **64** features (**NOTE:** this will half the image size), then max pooling and normalization are applied.
The output is provided to second convolution layer performing **64** feature operations followed by pooling and normalization. The image is now, **8x8** after two convolutions, hence the first fully connected layer is formed according to be compatible to this size and outputs **384** units, reduced to **192** units in the second fully connected layer and **10** (for the number of classes) in the last layer and dropout is performed with a probability of **0.5**. The output is tested for error and accuracy during training and validation phases.

3 Methods and Experiments

In the following sections we focus on setting structure and operation of the baseline model and then vary it over different paramters. The parameters are not tested radomly over huge ranges but tested over values with proven prior results to draw a comparison line of understanding the effect of variation of these factors on the performance.
The experiments are divided into sections and their results discussed within with an overall comparison at the end. The parameters over which the network is tested over has some similarity to previous done work and some sections of it is are development over it, whereas factors such as convolutional are experiment newly in this test.

3.1 Activation Functions

This section deals with testing over different activation functions. With RELUs being a popular choice for CNNs, It's variations such as Noisy RELUs, Parametric RELUs, Leaky RELUs and shifted RELUs are compared for performance [12] give a view of how they are expected to perform over the CIFAR-10 dataset in a CNN architecture.Latest development of ELUs give an interesting match to see the networks efficiency.
The activation functions are applied at Convolutional and Fully Connected Layers before outputting the intermediate results from these layers respectively. The Tensorflow's built in activation functions are used for ELUs and Regular ELUs with Glorot initialization [4].

3.1.1 Rectified Linear Units (RELUs)

RELUs [9] are ramp functions for units help eliminating negative values and the vanishing gradient problem, they have been consistely used over different

models for CNN for image recognition. The activation is used on units after performing convolutional or full connections before passing on the next layer using Tensorflow's **'tf.nn.relu'**
It is basically performing the operation:
$x = max(0, x)$

3.1.2 Leaky Rectified Linear Units (Leaky RELUs)

Leaky RELUs are variation of RELUs where instead of setting the negative inputs to 0, we take into account with a very small factor.
It is performing the operation with α mostly used with value 0.01 [8] :
$x = max(x, \alpha * x)$
Tensorflow doesn't support the built in Leaky RELUs so simple function is defined in python tensorflow implementing the above equation and replaces RELUs used in the previous section.

3.1.3 Exponential Linear Units (ELUs)

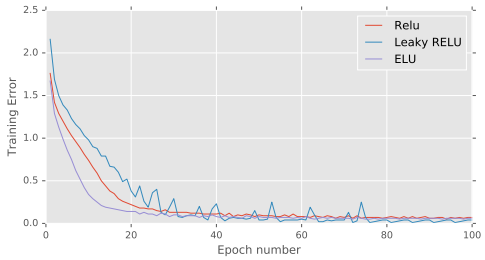
The current state of art for CIFAR-100 are ELUs, they are able to push mean activation to zero like batch normalization [3]. They use the operation:
$f(x) = x, if, x > 0$
$\alpha * (exp(x) - 1), if, x <= 0$
where α is the hyperparameter controling the degree of saturation for the units.
It is implemented using Tensorflow's built-in activation function **'tf.nn.elu'**

3.1.4 Results

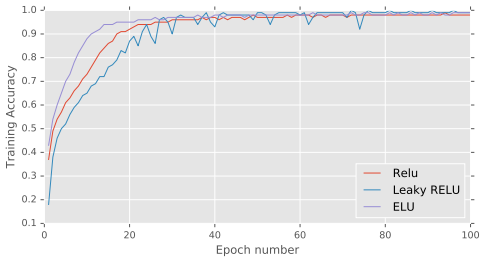
The plot is provided in the figure 2.
The results suggest Leaky RELUs perform better for our model than ELU, so this contradicts our hypothesis and the possible causes because of the structure of ELU seem to be less number of classes in CIFAR-10 for an exponential model, this could be expected for a smaller dataset than CIFAR-100 where there are 100 classes. The RELUs also work better than the results obtained in our previous work 5

Activations	**RELU**	**Leaky RELU**	**ELU**
error(train)	0.07	0.04	0.05
acc(error)	0.98	0.99	0.99
error(valid)	5.13	2.97	3.54
acc(valid)	0.56	0.63	0.60

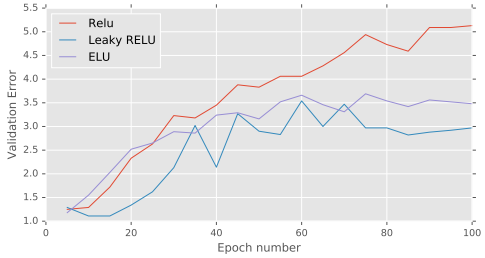
Table 1: **Comparison of Activation Functions**



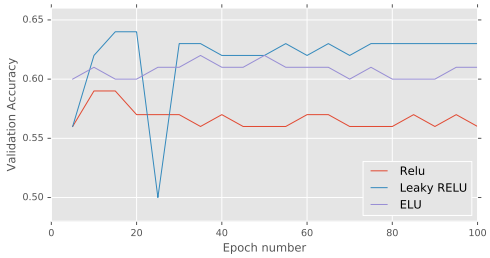
(a) Training Error



(b) Training Accuracy



(c) Validation Error



(d) Validation Accuracy

Figure 2: Results for Activation Functions

3.2 Droput

Dropout is performed to reduce overfitting in the neural networks [11]. The intuition behind is dropping some connections hoping to remove bad and unreliable connections to impove the performance. As we dropout units, the chance for overfitting is very low as the features are updated based on a probability now. We use Tensorflow's **tf.nn.dropout** for dropouts, and they are implented at the last layer

3.2.1 No Dropout

If no dropout is applied, it has a high chance of overfitting as the network grows bigger, that's one of the first motivations to actually include a dropout.

3.2.2 Dropout with 0.5 and 0.75

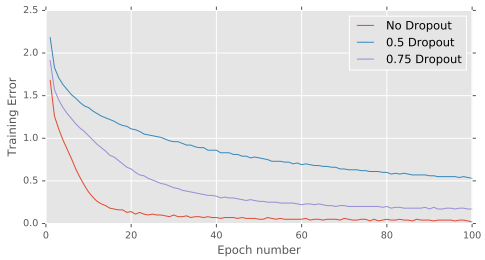
Dropout is done with generally used optimal value 0.5 [11] and 0.75 to test out how it effects the performance of the CNN model.

3.2.3 Results

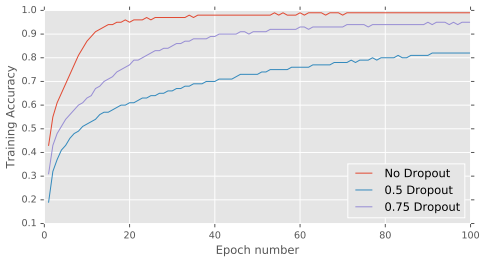
The plot is provided in the figure 3 As expected, the network without dropout

Dropouts	No Dropout	0.5	0.75
error(train)	0.07	0.53	0.17
acc(error)	0.98	0.82	0.95
error(valid)	5.13	2.00	2.74
acc(valid)	0.56	0.56	0.57

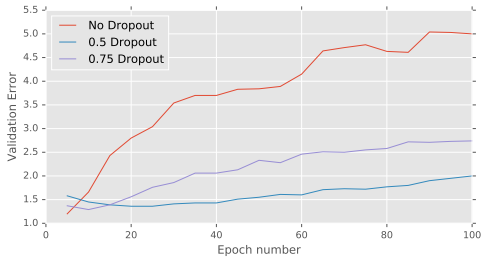
Table 2: **Dropouts**



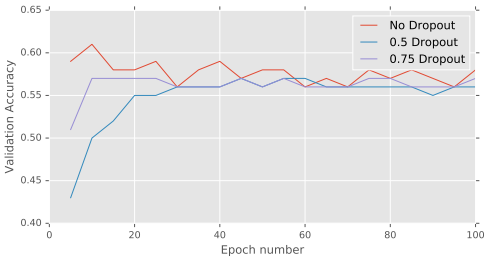
(a) Training Error



(b) Training Accuracy



(c) Validation Error



(d) Validation Accuracy

Figure 3: Results for Dropout

is overfitting on the dataset, and also giving a high validation error. It is also notable that a network with 0.75 dropout is having higher validation error than one with 0.5, this verifies the claim made in the literature [11] already.

3.3 Number of Convolutional Layers

We are using the convolution function of Tensorflow **tf.nn.conv2d** to implement the model. As mentioned earlier, the first layer is used to produce 64 features and same for the second one, leading to 8*8 image from 32*32, where in case of 1 layer network, the size is reduced to 16*16 only. Reshaping of the images have to be done accordingly to be able to pass the output to fully connected layers. **tf.reshape** has to be used for shaping the inputs before passing to the layer with values [-1, 32, 32, 3] where 32 is the image height and width and 3 is

the depth introduced for RGB.

3.3.1 1 layer

A Single Convolutional layer with 3 chanels, 5 kernel size is used to produce 64 features, and this layer is going to be half the image once. The performace of this layer is expected not to vary too much from the baseline model, as many architures of CNNs use a single layer.The output from this layer is then passed to the fully connected layer, after pooling and doing Local Response Normalization [7] using Tensorflow **tf.nn.lrn**.

3.3.2 2 layer

This is our baseline architecture we are using to establish the model, it performs good for the given inputs as is also motivated by Tensorflow's implementation for CIFAR-10. The orignal ALexNet [7] had more convolution layers, but for testing of our hypothesis we have considered a much simpler model.
The same convolution operation of layer 1 is applied here as we keep 64 features both the layers, but this is cause the output to be halved twice and will be 8*8, so the next layer has to reshape according to these outputs.

3.3.3 Results

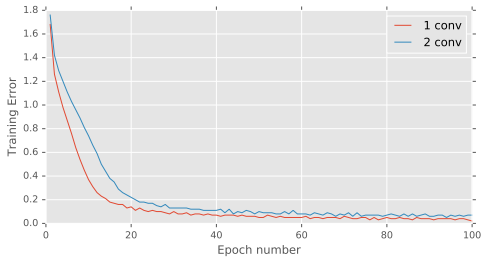
The plot is provided in the figure 4. As expected, the removal of 1 convolutional layer did not cause any major changes away from our baseline architecutre, althoguh the minor changes were surely expected to be observed. There is difference of 0.02 accuracy between the two models in our implementation.

Convolution	**1 Layer**	**2 Layers**
error(train)	0.07	0.02
acc(error)	0.98	0.99
error(valid)	5.13	5.00
acc(valid)	0.56	0.58

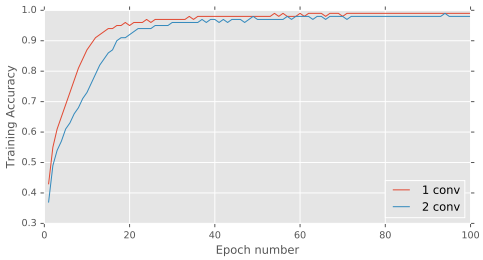
Table 3: **Convolution Layers**

3.4 Pooling

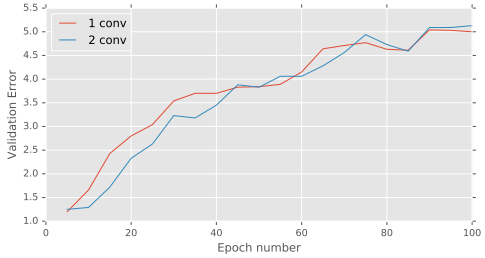
The motivation for pooling of selection specifically is to partition the data so as to reduce the size of the set and also provide removal of overfittng with these strides. We compare the max pooling [7] and fractional max pooling [5] which is currently the state of art for CIFAR-10 for RELUs and ELUs over the model to see how both of them shape over the changes in the pooling.



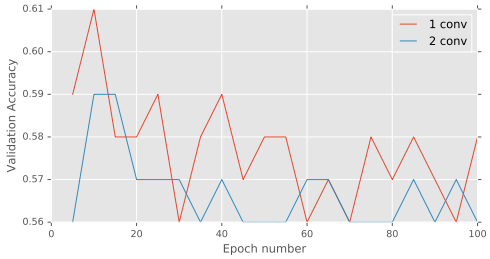
(a) Training Error



(b) Training Accuracy



(c) Validation Error



(d) Validation Accuracy

Figure 4: Results for Convolution

3.4.1 Max Pooling

We use tensorflow's nn.max pool with ksize = [1,3,3,1] and strides = [1,2,2,1] with 'paddig = SAME' for the this part. This is the pooling generally implemented in many established models for CNN. This is expected to perform good, but we assume it would perform inferior to Fractional Max Pooling in our model.

3.4.2 Fractional Max Pooling

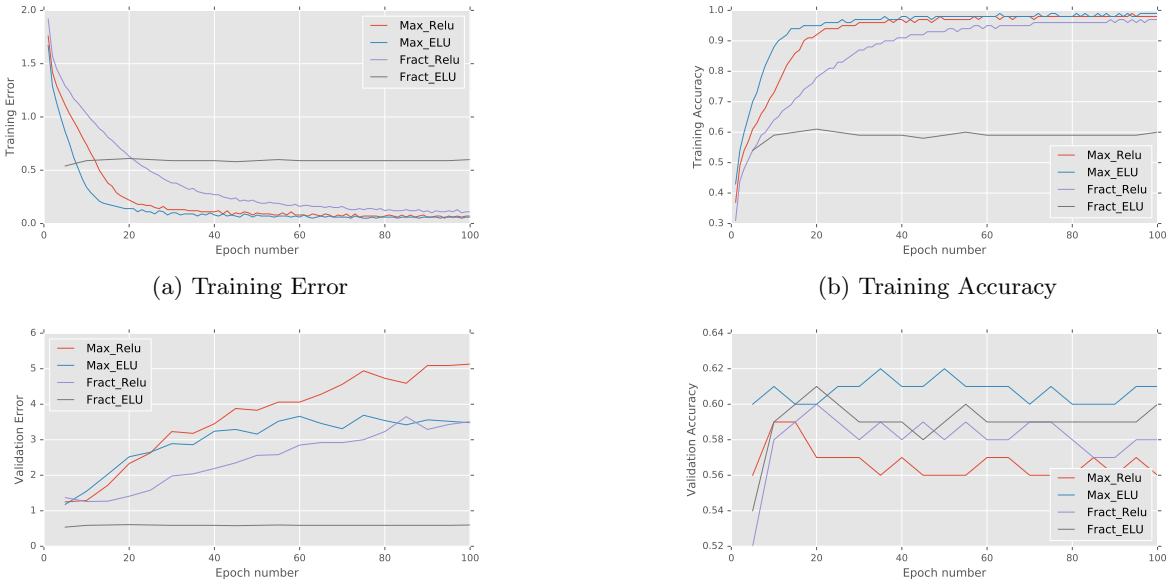
Current state of art method for CIFAR-10 [5] it is implemented after convolutional layers, and we use tensorflow's **nn.fractional max pool** with values for pooling as [1.0, 1.44 , 1.73, 1.0] and reshape the next layer according as they ared changing the image dimensions. They are expected to improve upon the max pool.

3.4.3 Results

The plot is provided in the figure 5. Due to the architecture of our baseline model, the difference between the results of max pool and fractional max pool is not very significantly high, but is still visible and improves upon the previous results.

Pooling	Relu-max	ELU-max	Relu-fracmax	ELU-fracmax
error(train)	0.07	0.05	0.11	0.13
acc(error)	0.98	0.99	0.97	0.96
error(valid)	5.13	3.54	3.51	3.40
acc(valid)	0.56	0.60	0.58	0.60

Table 4: **Pooling**



(a) Training Error

(b) Training Accuracy

(c) Validation Error

(d) Validation Accuracy

Figure 5: Results for Pooling

3.5 Overall Results

Our hypothecsis proved to be correct over parameters of performance, but Leaky RELUs did perform better than ELUs on CIFAR-10 on some instances, so ELUs although being the state of art for CIFAR-100, still have scope for improvement over the CIFAR-10 dataset. We were also able to improve our previously achieved accuracy in the previous work significantly.
The Leaky ELUs did perform well but the best accuracy we achived was with fractional max polling with ELUs and it is not heavily overfitting as well. We hope to improve these results as there is scope of improvement regarding the model architecture we used for these tests.

4 Conclusions

Previous Results	Relu-Glorot Initialization
error(train)	0.50
acc(error)	0.82
error(valid)	2.17
acc(valid)	0.48

Table 5: **Results from Previous Work** [10]

4.1 Further Work

Find a table 1

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Rodrigo Benenson. Classification datasets results. Online, `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`, 2016. Accessed at March 21, 2017.

[3] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[5] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[6] Alex Krizhevsky. Learning multiple layers of features from tiny images. Online,`https://www.cs.toronto.edu/~kriz/cifar.html`, 2009.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[9] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[10] s1655337. Study of behavior of rectified linear units (cousework 3 - machine learning practical)(university of edinburgh). Submitted February 16, 2017.

[11] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[12] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.