Image Classification with Deep Neural Networks

Ahmad KHAZAIE [1]

Abstract

This document provides the report of image classification project using deep neural networks on CIFAR10 dataset. In this project I developed simple ResNet and I run different tests to show the effect of each parameter on the accuracy and loss of the model.[1]

1. Introduction

Deep neural networks are powerful and popular algorithms and a lot of their success lays in the careful design of the neural network architecture. In recent years, many works have been done through convolutional neural networks. Convolutional neural networks are fantastic for visual recognition tasks and they have very high performance in image classification.(He et al., 2016) Deep learning has absolutely dominated computer vision over the last few years, achieving top scores on many tasks and their related competitions. Also improvements in computer hardware and network structure makes training of truly deep CNNs possible only recently.(Huang et al., 2017). During last recent years, different novel methods have been presented and several refinements have been proposed to tackle some problems in the aim of achieving better results. Some of them tried to get higher accuracy and lower error in the cost of increasing complexity while some others tried to keep simplicity and achieved good enough results. In this project, I am going to use one of well-known methods, ResNet, for making an image classifier in order to be used on CIFAR10 dataset. Since there is a lot of different algorithms, I decided to choose a method which can give very good results in a reasonable training time while it is simple enough to be implemented. I implemented a simple ResNet, which won the ILSVRC 2015, because of its simplicity and high accuracy. As the training time is important, I didn't implement a very deep network and I used ResNet-20 architecture, which gives very good results in compare with other methods.

[1] https://github.com/amdkhz/cifar10.git

2. Preliminaries

The data provided for this task is in kaggle with the name of CIFAR-10 [2]. The CIFAR-10 data consists of 60,000 of 32x32 color images in 10 exactly balanced and mutually exclusive classes, (6000 images per class). There are 50,000 training images which are generated by randomly selected 5000 images from each class and 10,000 remained images as test set in the official data. The data is split in two train and test sections from the original dataset. The label classes in the dataset are:
{'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'}

The two main goals this project is first to learn a classifier on a pair of classes of the CIFAR10 dataset. The second one is to learn a multi-class classifier on all of 10 classes that can classify a given image into a unique class. Moreover, in this project I focus on utilizing deep learning approaches, a state-of-the-art category of machine learning algorithms in visual object recognition, to solve this problem. In all experiments, each image considered as a vector of 3x32x32. The 3 is considered because of color of images(RGB features). The training is done solely in the train set. Though the learning algorithms do not use test set for optimizing model parameters directly, **early stopping** and **learning rate decay** is based on the test performance.

2.1. Deep Learning

The main characteristic of Deep learning which is different from the traditional machine learning algorithms is that deep learning learns hierarchical features (representations) from data instead of manually designing representative features. It is especially effective in the field of computer vision where the algorithms have to deal with highly correlated (redundant) and highly-variant (noise) information in images(Chen et al., 2016). In the following, we're going to explain about feed-forward and convolutional neural network.

2.1.1. Deep feed-forward neural network

Deep feed-forward neural networks, also often called feed-forward neural networks, Deep feed-forward networks or multilayer perceptrons(MLPs), are the quintessential deep

[2] https://www.kaggle.com/c/cifar-10/data

learning models. These models are called feed-forward because information flows through the function being evaluated from input through the intermediate computations and finally to the output. There are no feedback connections in which outputs of the model are fed back into itself. When feed-forward neural networks are extended to include feedback connections, they are called recurrent neural networks.

2.1.2. Deep convolutional neural networks

Convolutional neural networks (CNNs) have become the dominant machine learning approach for visual object recognition. They were originally introduced over 20 years ago. However, improvements in computer hardware and network structure have enabled the training of truly deep CNNs only recently(Huang et al., 2017). A Convolutional neural networks works likes a feed-forward networks. There are input layer, output layer and more than one hidden layers. However, the hidden layers in CNN make use of spatially or temporally correlated information in input. In feed-forward neural network, each neuron in a hidden layer collects the output from all neurons in the previous layer and simply computes a weighted sum. Similarly, a neuron in a CNN hidden layer collects the output from all neurons in the previous layer. However, it computes and outputs weighted sums over many parts of the data instead of a single value.

2.2. Tenserflow

There are many tools that reduce the eort to experiment with deep learning algorithms, such as Caffe[3], Tensorflow[4] and pytorch[5]. Here, I used tensorflow. Tensorflow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows developer to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

3. Related Work

Different model of CNNs have introduced through recent years: the original LeNet5 consisted of 5 layers, VGG featured 19, and only last year Highway Networks and Residual Networks (ResNets) have surpassed the 100-layer barrier(Huang et al., 2017). By becoming CNNs increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and wash out by the time it reaches the end (or beginning) of the network. There are different approaches: ResNets and Highway Networks bypass signal from one layer to the next through identity connections. Stochastic depth shortens ResNets by randomly dropping layers during training to allow better information and gradient flow. FractalNets repeatedly combine several parallel layer sequences with different number of convolutional blocks to obtain a large nominal depth, while maintaining many short paths in the network.These different approaches are varied in network topology and training procedure but they all share a key characteristic: they create short paths from early layers to later layers(Huang et al., 2017). In the following I am going to explain more about some CNNs models such as Deep Residual Networks(Resnet), Alexnet, VGG, etc.

AlexNet

AlexNet, proposed by Alex Krizhevsky, uses ReLu(Rectified Linear Unit) for the non-linear part, instead of a Tanh or Sigmoid function which was the earlier standard for traditional neural networks. (Krizhevsky et al., 2012). They used a relatively simple layout, compared to modern architectures. The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers. The network they designed was used for classification with 1000 possible categories.

This was the first time a model performed so well on a historically difficult ImageNet dataset. Utilizing techniques that are still used today, such as data augmentation and dropout. The advantage of the ReLu over sigmoid is that it trains much faster than the latter because the derivative of sigmoid becomes very small in the saturating region and therefore the updates to the weights almost vanish. This is called vanishing gradient problem. Also this model solved reducing the over-fitting by using a Dropout layer after every FC layer.

The idea behind the dropout is similar to the model ensembles. Due to the dropout layer, different sets of neurons which are switched off, represent a different architecture and all these different architectures are trained in parallel with weight given to each subset and the summation of weights being one.

ZF Net

This architecture was more of a fine tuning to the previous AlexNet structure, but still developed some very keys ideas about improving performance. The authors spent a good amount of time explaining a lot of the intuition behind Con-

[3] http://caffe.berkeleyvision.org/
[4] https://www.tensorflow.org/
[5] http://http://pytorch.org/

vNets and showing how to visualize the filters and weights correctly. They discussed the idea that this renewed interest in CNNs is due to the accessibility of large training sets and increased computational power with the usage of GPUs. They also talk about the limited knowledge that researchers had on inner mechanisms of these models, saying that without this insight, the development of better models is reduced to trial and error.(Zeiler & Fergus, 2014).

VGG

This architecture is from VGG group, Oxford. Simonyan and Andrew Zisserman of the University of Oxford created a 19 layer CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 maxpooling layers with stride 2. It makes the improvement over AlexNet by replacing large kernel-sized filters(11 and 5 in the first and second convolutional layer, respectively) with multiple 3X3 kernel-sized filters one after another. With a given receptive field (the effective area size of input image on which output depends), multiple stacked smaller size kernel is better than the one with a larger size kernel because multiple non-linear layers increases the depth of the network which enables it to learn more complex features, and that too at a lower cost. The VGG convolutional layers are followed by 3 fully connected layers. The width of the network starts at a small value of 64 and increases by a factor of 2 after every subsampling/pooling layer. It achieves the top-5 accuracy of 92.3% on ImageNet.(Simonyan & Zisserman, 2014)

GoogLeNet

While VGG achieves a phenomenal accuracy on ImageNet dataset, its deployment on even the most modest sized GPUs is a problem because of huge computational requirements, both in terms of memory and time. It becomes inefficient due to large width of convolutional layers. In a convolutional operation at one location, every output channel (512 in the example above), is connected to every input channel, and so I call it a dense connection architecture. The GoogLeNet builds on the idea that most of the activations in deep network are either unnecessary(value of zero) or redundant because of correlations between them. Therefore most efficient architecture of a deep network will have a sparse connection between the activations, which implies that all 512 output channels will not have a connection with all the 512 input channels. There are techniques to prune out such connections which would result in a sparse weight/connection. But kernels for sparse matrix multiplication are not optimized in BLAS or CuBlas(CUDA for GPU) packages which render them to be even slower than their dense counterparts. So GoogLeNet devised a module called inception module that approximates a sparse CNN with a normal dense construction. Since only a small number of neurons are effective as mentioned earlier, width/number of the convolutional filters of a particular kernel size is kept small. Also, it uses convolutions of different sizes to capture details at varied scales(5X5, 3X3, 1X1).

Another salient point about the module is that it has a so-called bottleneck layer(1X1 convolutions in the figure). It helps in massive reduction of the computation requirement. Another change that GoogLeNet made, was to replace the fully-connected layers at the end with a simple global average pooling which averages out the channel values across the 2D feature map, after the last convolutional layer. This drastically reduces the total number of parameters. This can be understood from AlexNet, where FC layers contain approx. 90% of parameters. Use of a large network width and depth allows GoogLeNet to remove the FC layers without affecting the accuracy. It achieves 93.3% top-5 accuracy on ImageNet and is much faster than VGG.

Deep Residual Networks

Deep residual networks (ResNets) have emerged as a family of extremely deep architectures showing compelling accuracy and nice convergence behaviors. ResNet is a new 152 layer network architecture that set new records in classification, detection, and localization through one incredible architecture. the problem with increased depth is that the signal required to change the weights, which arises from the end of the network by comparing ground-truth and prediction becomes very small at the earlier layers, because of increased depth. It essentially means that earlier layers are almost negligible learned. This is called vanishing gradient. The second problem with training the deeper networks is, performing the optimization on huge parameter space and therefore naively adding the layers leading to higher training error. Residual networks allow training of such deep networks by constructing the network through modules called residual models. Also, similar to GoogLeNet, it uses a global average pooling followed by the classification layer.Through the changes mentioned, ResNets were learned with network depth of as large as 152. It achieves better accuracy than VGGNet and GoogLeNet while being computationally more efficient than VGGNet. ResNet-152 achieves 95.51 top-5 accuracies.(He et al., 2016) ResNet is very powerful and many tried to tackle its problems and improve the performance. Xie et al.(Xie et al., 2017) proposed a variant of ResNet that is codenamed ResNeXt which is very similar to Inception model of (Szegedy et al., 2015). Huang et al.(Huang et al., 2017) proposed a novel architecture called DenseNet that further exploits the effects of shortcut connections. it connects all layers directly with each other. In this novel architecture, the input of each layer consists of the feature maps of all earlier layer, and its output is passed to each subsequent layer. Although ResNet has proven powerful in many applications, one major drawback is that deeper network usually requires weeks

for training, making it practically infeasible in real-world applications. To tackle this issue, Huang et al.(Huang et al., 2016) introduced a counter-intuitive method of randomly dropping layers during training, and using the full network in testing. Huang et al.(Huang et al., 2016) proposed a counter-intuitive way of training a very deep network by randomly dropping its layers during training and using the full network in testing time. Veit et al.(Veit et al., 2016) had an even more counter-intuitive finding for dropping some of the layers of a trained ResNet and still have comparable performance. Zagoruyko et al. (Zagoruyko & Komodakis, 2016) showed in Resnet training very deep residual networks has a problem of diminishing feature reuse, which makes these networks very slow to train. So they tackled this problem by conducting a detailed experimental study on the architecture of ResNet blocks, based on which they proposed a novel architecture where they decreased depth and increase width of residual networks. Gastaldi(Gastaldi, 2017) proposed another method to deal with an overfit problem. Devries et al.(DeVries & Taylor, 2017) showed how the simple regularization technique of randomly masking out square regions of input during training, which they called cutout, can be used to improve the robustness and overall performance of convolutional neural networks. As I implemented this architecture in our project, I will explain the details of this model later.

Region Based CNNs

R-CNN

The purpose of R-CNNs is to solve the problem of object detection. The goal is to be able to draw bounding boxes over all of the objects. The process can be split into two general components, the region proposal step and the classification step. Selective Search is used in particular for RCNN which performs the function of generating 2000 different regions that have the highest probability of containing an object. These proposals are then warped into an image size that can be fed into a trained CNN (AlexNet in this case) that extracts a feature vector for each region. This vector is then used as the input to a set of linear SVMs that are trained for each class and output a classification. The vector also gets fed into a bounding box regressor to obtain the most accurate coordinates.(Girshick, 2015)

Fast R-CNN

Improvements were made to the original model because of 3 main problems. Training took multiple stages, was computationally expensive, and was extremely slow. Fast R-CNN was able to solve the problem of speed by basically sharing computation of the conv layers between different proposals and swapping the order of generating region proposals and running the CNN. In this model, the image is first fed through a ConvNet, features of the region proposals are obtained from the last feature map of the ConvNet and lastly I have my fully connected layers as well as our regression and classification heads.(Girshick, 2015).

Faster R-CNN

Faster R-CNN works to combat the somewhat complex training pipeline that both R-CNN and Fast R-CNN exhibited. The authors insert a region proposal network (RPN) after the last convolutional layer. This network is able to just look at the last convolutional feature map and produce region proposals from that. From that stage, the same pipeline as R-CNN is used.(Ren et al., 2015)

CIFAR10

There are so many papers which used the CIFAR10 dataset for image classification. To be able to compare our results with other works I showed some of them in Table.1 and Table.2. For example, McDonnell et al.(McDonnell & Vladusich, 2015) designed a rapid learning method without data augmentation which is very simple to implement but the result is only 75.86%. Coates et al.(Coates et al., 2011) applied several learning algorithms and used a single layer network to show that large number of hidden nodes and dense feature extraction are as critical to achieving high performance as the choice of algorithm itself and they achieved 79.6% accuracy on the dataset. Graham(Graham, 2014a) formulated a fractional version of max-pooling where alpha is allowed to take non-integer values which is stochastic and achieved the best result, 96.53%. Springenberg et al.(Springenberg et al., 2014) used their model with different type of data augmentation and best result achieved after using aggressive data augmentation and large network.

4. Approach

To achieve the goals of this project, I decided to follow a ResNet-20 architecture as it gives very good accuracy and low error and the training is not very time consuming.(Krizhevsky et al., 2012). Besides, the implementation is not very complicated. First I chose 'deer' and 'dog' as the pair of classes for first task and I changed the size of train and test set. In second part, I used all 10 classes and changed different parameter for number of epochs, batch size and size of training and validation set to observe the effect of each parameter on the results. As ResNet is very powerful and successful in recent years, many different refinements have been made in the architecture. The list of these methods contains but not limited to: ResNeXt, DenseNet, Wide ResNet, Stochastic ResNet, Improved Regularization, Shake-Shake, ShakeDrop Regularization, etc. Among all different variations of ResNet, I chose the simple method to implement which is proposed by (He et al., 2016).

Table 1. Classification accuracies for different papers which used CIFAR10 dataset

METHOD	RESULT
FRACTIONAL MAX-POOLING(GRAHAM, 2014A)	96.53%
STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET(SPRINGENBERG ET AL., 2014)	95.59%
ALL YOU NEED IS A GOOD INIT(MISHKIN & MATAS, 2015)	94.16%
LESSONS LEARNED FROM MANUALLY CLASSIFYING CIFAR-10(KARPATHY, 2011)	94%
GENERALIZING POOLING FUNCTIONS IN CONVOLUTIONAL NEURAL NETWORKS: MIXED, GATED, AND TREE(KARPATHY, 2011)	93.95%
SPATIALLY-SPARSE CONVOLUTIONAL NEURAL NETWORKS(GRAHAM, 2014B)	93.72%
SCALABLE BAYESIAN OPTIMIZATION USING DEEP NEURAL NETWORKS(SNOEK ET AL., 2015)	93.63%
DEEP RESIDUAL LEARNING FOR IMAGE RECOGNITION(HE ET AL., 2016)	93.57%
FAST AND ACCURATE DEEP NETWORK LEARNING BY EXPONENTIAL LINEAR UNITS(CLEVERT ET AL., 2015)	93.45%
UNIVERSUM PRESCRIPTION: REGULARIZATION USING UNLABELED DATA(ZHANG & LECUN, 2017)	93.34%
BATCH-NORMALIZED MAXOUT NETWORK IN NETWORK(CHANG & CHEN, 2015)	93.25%
COMPETITIVE MULTI-SCALE CONVOLUTION (LIAO & CARNEIRO, 2015)	93.13%
RECURRENT CONVOLUTIONAL NEURAL NETWORK FOR OBJECT RECOGNITION(LIANG & HU, 2015)	92.91%
LEARNING ACTIVATION FUNCTIONS TO IMPROVE DEEP NEURAL NETWORKS(AGOSTINELLI ET AL., 2014)	92.49%

Table 2. Classification accuracies for different papers which used CIFAR10 dataset

METHOD	RESULT
TRAINING VERY DEEP NETWORKS(SRIVASTAVA ET AL., 2015)	92.40%
STACKED WHAT-WHERE AUTO-ENCODERS(ZHAO ET AL.)	92.23%
MULTI-LOSS REGULARIZED DEEP NEURAL NETWORK(XU ET AL., 2016)	91.88%
DEEPLY-SUPERVISED NETS(LEE ET AL., 2015)	91.78%
BINARYCONNECT: TRAINING DEEP NEURAL NETWORKS WITH BINARY WEIGHTS DURING PROPAGATIONS(COURBARIAUX ET AL., 2015)	91.73%
ON THE IMPORTANCE OF NORMALISATION LAYERS IN DEEP LEARNING WITH PIECEWISE LINEAR ACTIVATION UNITS(COURBARIAUX ET AL., 2015)	91.48%
SPECTRAL REPRESENTATIONS FOR CONVOLUTIONAL NEURAL NETWORKS(RIPPEL ET AL., 2015)	91.40%
NETWORK IN NETWORK(LIN ET AL., 2013)	91.20%
SPEEDING UP AUTOMATIC HYPERPARAMETER OPTIMIZATION OF DEEP NEURAL NETWORKS BY EXTRAPOLATION OF LEARNING CURVES(DOMHAN ET AL., 2015)	91.19%
DEEP NETWORKS WITH INTERNAL SELECTIVE ATTENTION THROUGH FEEDBACK CONNECTIONS(STOLLENGA ET AL., 2014)	90.78%
REGULARIZATION OF NEURAL NETWORKS USING DROPCONNECT(WAN ET AL., 2013)	90.68%
MAXOUT NETWORKS(GOODFELLOW ET AL., 2013)	90.65%
IMPROVING DEEP NEURAL NETWORKS WITH PROBABILISTIC MAXOUT UNITS(SPRINGENBERG & RIEDMILLER, 2013)	90.61%
PRACTICAL BAYESIAN OPTIMIZATION OF MACHINE LEARNING ALGORITHMS(SNOEK ET AL., 2012)	90.5%

The idea behind a residual block, Fig.1 is that you have your input x go through conv-relu-conv series. This will give you some F(x). That result is then added to the original input x. Lets call that H(x) = F(x) + x. In traditional CNNs, your H(x) would just be equal to F(x) right? So, instead of just computing that transformation (straight from x to F(x)), were computing the term that you have to add, F(x), to your input, x. Basically, the mini module shown below is computing a delta or a slight change to the original input x to get a slightly altered representation (When I think of traditional CNNs, I go from x to F(x) which is a completely new representation that doesnt keep any information about the original x). The authors believe that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. Another reason for why this residual block might be effective is that during the backward pass of back-propagation, the gradient will flow easily through the graph because I have addition operations, which distributes the gradient. Fig.2 shows the block architecture proposed by (He et al., 2016)[6].

5. Experiments and Evaluation

5.1. Implementation

In this section, I describe all technologies and technical aspects which I used in our project. All the code is developed in python 3.6 and I used Tensorflow 1.5 for implementing deep neural networks. However there are some other libraries, e.g. pyTorch, for using in deep learning methods. Since I followed the suggestions of the reference paper of ResNet, I used all their weights and parameter and I didn't put more time for doing more experiment to find out the best parameters. I downloaded python version of the CIFAR10 data and extracted the labels and values from the files. Learning rate is going to be reduced after 80, 120, 160

[6]http://torch.ch

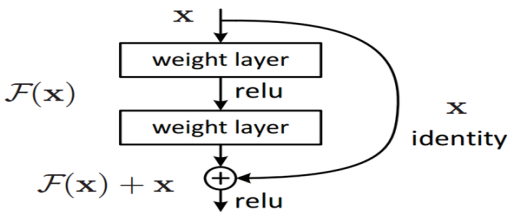

Figure 1. A Residual Block

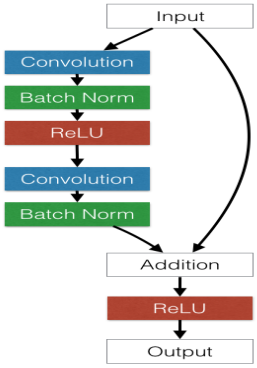

Figure 2. ResNet Block Architecture

and 180 respectively. I have 16 filters and minimum learning rate is $0.5e^{-6}$. After each improvement in validation accuracy I save the model and for storage efficiency I only keep the best one. Also I implemented data augmentation, which can effect the results significantly(Zhang et al., 2016)(Wang & Perez, 2017)(Krizhevsky et al., 2012), and I used horizontal random shift, vertical random shift and horizontal flip to generate more images and increase the accuracy of the model while trying to avoid over-fitting problem in early stages.

5.2. Experiments

Training the deep neural networks is very time consuming and it needs a lot of resources specially when the depth increased. As one of us is currently working in Nokia Bell Labs for the thesis, using deep learning for malware detection, I have access to a very powerful server for deep learning. Thanks to the company, I could train our model very fast using a server which has a Intel Xeon CPU E5-2623 with 16 cores and 128GB RAM and 8 Tesla K80 GPU. So all the experiments run so fast and I could get very good results by changing different parameters. For the first goal, as I mentioned, I chose 'deer' and 'dog' and I run the experiments for different size of train and test data set. The size of train set decreased from 10000 samples to 6000 samples and the test set increased from 2000 to 6000 and for each experiment I stored the best accuracy and lowest loss to be able to compare the effect of the size of train and test set. In this part I did all experiments for 200 epochs and 128 images in each batch. For second part, I was more interested to reach the highest possible accuracy and I did more experiments by changing more parameters. So I changed batch size, number of epochs and size of train and test sets and I compare the results of experiments. I changed the batch size from 32 to 512. Number of epochs increased from 100 to 500 and separately, size of training data decreased from 50000 to 30000 while the test set size increased from 10000 to 30000.

5.3. Evaluation

In this part, I explain the achieved results by the model and the reasons for improvements affected by different parameters. For the first task I compare the results in Fig.3. As you can see I can get the best accuracy and the minimum loss when the training size is increased. Since I need to keep part of the data set for test and validation, I cannot train the model with whole data set, which is 12000 images. As I already mentioned, I run all the test in the same environment with 200 epochs and batch size 128. In Fig. 4 I showed the changes of training accuracy, training loss, validation accuracy and validation loss for 200 epochs of batch size 128 and training set size equals to 10000 and validation set equals to 2000. As you can see after approximately 80 epochs validation loss increased while the validation accuracy is almost is the same with slow slope of changing and training loss is decreasing continue sly. So at this point I can observe a kind of over-fitting in this problem.

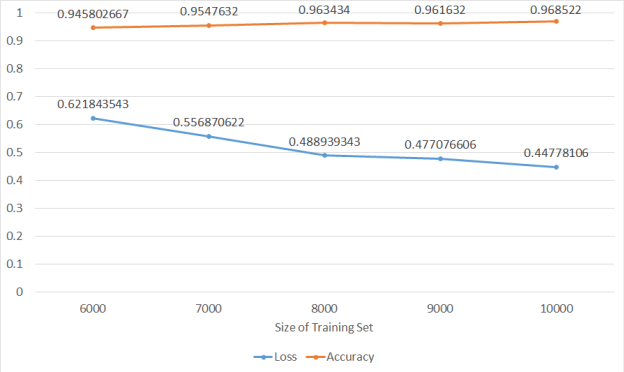

Figure 3. Accuracy and Loss for different size of training set for pair of classes 'deer' and 'dog'

In the second part of the project, I am going to do the classification on all 10 classes. In this part, I was more motivated to do more experiments and try to compete with the results of previous works and get the values as close as the ones which the different authors claimed in their papers. So first

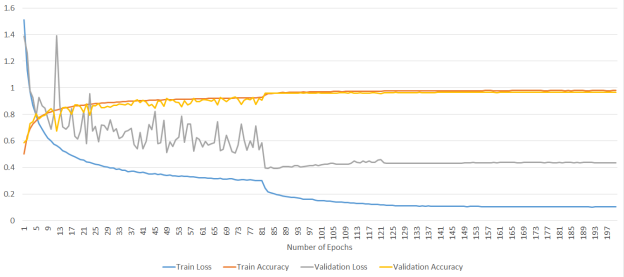

Figure 4. Accuracy and Loss for train and validation set for different epochs for 10K training set for pair of classes 'deer' and 'dog'

experiment has been done on size of training set. Since I am using the whole CIFAR10 dataset, the total number of available images is 60000 and I run the experiment by increasing the size of training set from 30000 to 50000 by step of 5000. The results of this experiment has been shown in Fig.5. Again I observed that the the accuracy will increase and the loss will decrease when I train our model based on a larger training set. The best achieved value for the accuracy is 91.37 while the authors claimed 93.57 which is very close to their implementation. In Fig.6 the growth of accuracy and decreasing loss is shown. I can observe kind of over-fitting after almost 80 epochs. Although the training accuracy is going up, close to 1, but the validation accuracy does't change that much.

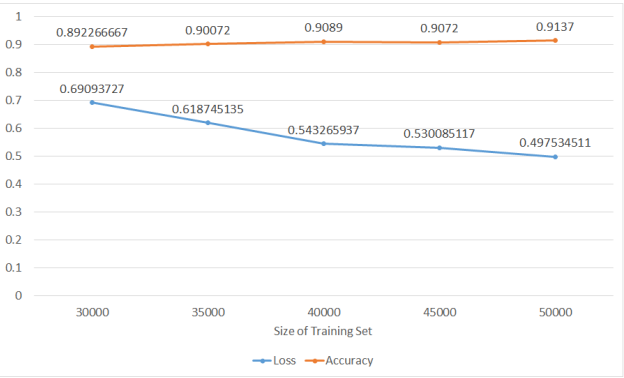

Figure 5. Accuracy and Loss for different size of training set for all classes

For next experiment, I kept 200 epochs and I changed the batch size for 50K training dataset and 10K test set. The results of the accuracy and loss is shown in Fig.7. As it shows the smaller batch size will give the better results but since the training time will increase significantly, it is not a good idea to use very small batch size. Fig.8 shows the changes of training accuracy, validation accuracy and their errors for batch size 32 in different epochs. Approximately after 80 epochs the training accuracy will keep increasing

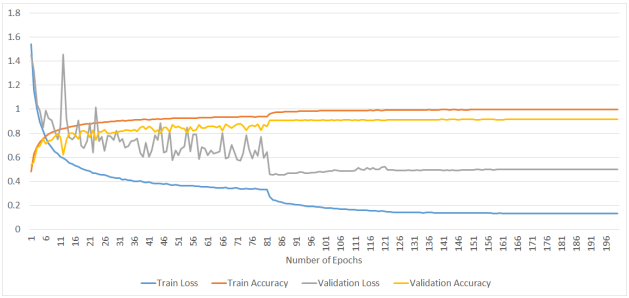

Figure 6. Accuracy and Loss for train and validation set for different epochs for 50K training set for all classes

but the validation accuracy remains the same and validation error starts to increase a bit.

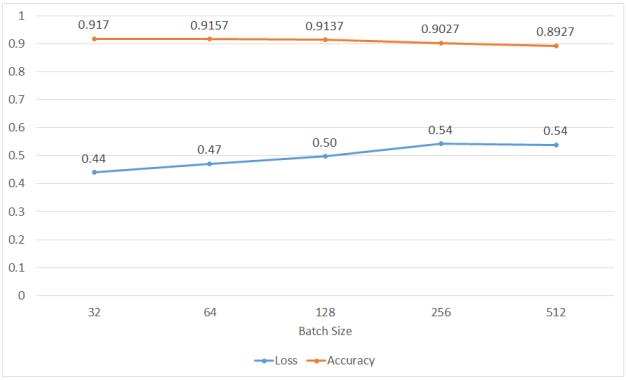

Figure 7. Accuracy and Loss for different batch size for all classes

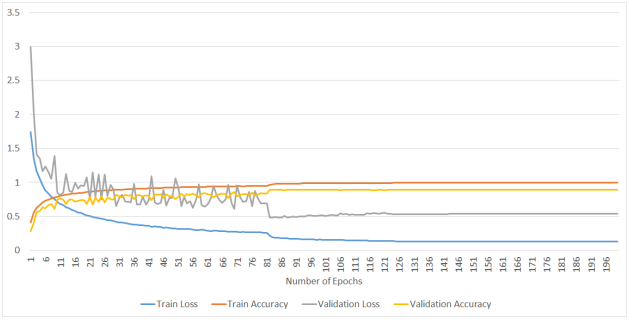

Figure 8. Accuracy and Loss for train and validation set for different epochs for batch size 32 for all classes

For investigating the effect of number of epochs on accuracy and loss, I run the experiment for batch size 128 and 50K training set and then I changed the number of epochs from 100 to 500 by step of 100. The results is shown in Fig.9. As it shows, the best accuracy achieved by 200 epochs and after that it decreased by increasing number of epochs. Also the minimum loss achieved at 200 epochs and after that it increased by increasing number of epochs.

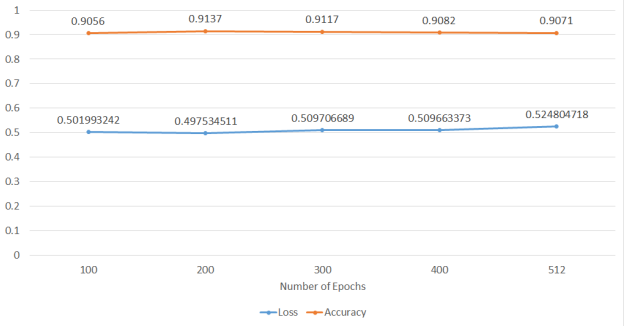

Figure 9. Accuracy and Loss for train and validation set for different number of epochs for all classes

6. Conclusion and Perspective

In this project, I aimed to build a classifier for CIFAR10 dataset using deep neural networks. After reviewing related works, I decided to implement a simple Deep Residual Network for image classification. However there is many different refinements for ResNet, which tackled different problems of simple ResNet and improved the performance by modifying the base method. I seek two different goals, first I selected a pair of classes of the dataset and classify the images using our model. In this phase, I changed the size of training and validation set to observe the effect of this parameter on the accuracy and loss value. The best accuracy that I could reach after 200 epochs and batch size 128, is 96.85% and the minimum loss at this step is 0.4477. Secondly, I trained the model for all 10 classes and I experimented the effect of batch size, training set size and number of epochs on the accuracy and loss. In this part the best achieved value for accuracy is 91.37%, when I used 50K training data with 200 epochs and 128 batch size, and 91.7%, when I used 32 batch size and 50K training size and 200 epochs. By comparing these results with previous works, I can say that our results are closed enough to them and I am able to improve it by using the new ideas, which gives better results than simple method. After that I can do more research about the parts which limit the performance and try to solve these problems. Working on some ideas like data augmentation or making the neural network deeper can effect the performance significantly. So, in near future I expect to see better results in this field as the number of the works which are done in this field are increasing very fast. Any new idea should be considered quickly and implemented as soon as possible. Many suggested to use different types of filter, weights and depth of the network and try to improve the performance.

References

Agostinelli, Forest, Hoffman, Matthew, Sadowski, Peter, and Baldi, Pierre. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

Chang, Jia-Ren and Chen, Yong-Sheng. Batch-normalized maxout network in network. *arXiv preprint arXiv:1511.02583*, 2015.

Chen, Rui, Jia, Huizhu, Xie, Xiaodong, and Gao, Wen. Correlation preserving sparse coding over multi-level dictionaries for image denoising. *arXiv preprint arXiv:1612.08049*, 2016.

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

Coates, Adam, Ng, Andrew, and Lee, Honglak. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011.

Courbariaux, Matthieu, Bengio, Yoshua, and David, Jean-Pierre. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

DeVries, Terrance and Taylor, Graham W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Domhan, Tobias, Springenberg, Jost Tobias, and Hutter, Frank. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, volume 15, pp. 3460–8, 2015.

Gastaldi, Xavier. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

Girshick, Ross. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Graham, Benjamin. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014a.

Graham, Benjamin. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014b.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Huang, Gao, Sun, Yu, Liu, Zhuang, Sedra, Daniel, and Weinberger, Kilian Q. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pp. 646–661. Springer, 2016.

Huang, Gao, Liu, Zhuang, Weinberger, Kilian Q, and van der Maaten, Laurens. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, pp. 3, 2017.

Karpathy, Andrej. Lessons learned from manually classifying cifar-10. *Published online at http://karpathy. github. io/2011/04/27/manually-classifying-cifar10*, 2011.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. In *Artificial Intelligence and Statistics*, pp. 562–570, 2015.

Liang, Ming and Hu, Xiaolin. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3367–3375, 2015.

Liao, Zhibin and Carneiro, Gustavo. Competitive multi-scale convolution. *arXiv preprint arXiv:1511.05635*, 2015.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

McDonnell, Mark D and Vladusich, Tony. Enhanced image classification with a fast-learning shallow convolutional neural network. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pp. 1–7. IEEE, 2015.

Mishkin, Dmytro and Matas, Jiri. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

Ren, Shaoqing, He, Kaiming, Girshick, Ross, and Sun, Jian. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Rippel, Oren, Snoek, Jasper, and Adams, Ryan P. Spectral representations for convolutional neural networks. In *Advances in neural information processing systems*, pp. 2449–2457, 2015.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Mostofa, Prabhat, Mr, and Adams, Ryan. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pp. 2171–2180, 2015.

Springenberg, Jost Tobias and Riedmiller, Martin. Improving deep neural networks with probabilistic maxout units. *arXiv preprint arXiv:1312.6116*, 2013.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Srivastava, Rupesh K, Greff, Klaus, and Schmidhuber, Jürgen. Training very deep networks. In *Advances in neural information processing systems*, pp. 2377–2385, 2015.

Stollenga, Marijn F, Masci, Jonathan, Gomez, Faustino, and Schmidhuber, Jürgen. Deep networks with internal selective attention through feedback connections. In *Advances in neural information processing systems*, pp. 3545–3553, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, Rabinovich, Andrew, et al. Going deeper with convolutions. Cvpr, 2015.

Veit, Andreas, Wilber, Michael J, and Belongie, Serge. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pp. 550–558, 2016.

Wan, Li, Zeiler, Matthew, Zhang, Sixin, Le Cun, Yann, and Fergus, Rob. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pp. 1058–1066, 2013.

Wang, Jason and Perez, Luis. The effectiveness of data augmentation in image classification using deep learning. Technical report, Technical report, 2017.

Xie, Saining, Girshick, Ross, Dollár, Piotr, Tu, Zhuowen, and He, Kaiming. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 5987–5995. IEEE, 2017.

Xu, Chunyan, Lu, Canyi, Liang, Xiaodan, Gao, Junbin, Zheng, Wei, Wang, Tianjiang, and Yan, Shuicheng. Multi-loss regularized deep neural network. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(12): 2273–2283, 2016.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zeiler, Matthew D and Fergus, Rob. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, and Vinyals, Oriol. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Zhang, Xiang and LeCun, Yann. Universum prescription: Regularization using unlabeled data. In *AAAI*, pp. 2907–2913, 2017.

Zhao, J, Mathieu, M, Goroshin, R, and Lecun, Y. Stacked what-where auto-encoders. arxiv 2015. *arXiv preprint arXiv:1506.02351*.