ResNet with Adaptive Moment Estimation

Ramandeep Singh
Department of Computer Science
University of Surrey
Guildford, England
rs01790@surrey.ac.uk

***Abstract*— In this coursework report, we will discuss image classification with the help of a Residual Neural Network which comes from the family of convolutional neural networks. Our aim will be to optimize the chosen model with the best optimization technique to get the best possible results i.e., higher accuracy for predictions. In this coursework, we will be implementing the ResNet i.e., Residual neural network with the help of Keras Library & ADAM (Adaptive Moment Estimation) optimizer and we will compare the accuracy of the model with SGD i.e., Stochastic gradient descent optimizer for same parameters. The Dataset used for the coursework is Cifar-10 and we will be implementing Data augmentation & regularization to prevent overfitting.**

I. Introduction

Convolutional neural networks are widely used for image classification. These neural networks are made up of two layers which are the pooling layers & convolutional layers. There are multiple ways to use these layers to solve a given problem but there are already many defined architectures that can be used. One can use these defined architectures and leverage these by adding some new layers and parameters. Although the most difficult part is to determine how to build topologies with which we can use these pre-defined architectures. Now, we will begin by discussing some of the previous work i.e., already-defined models. LeNet-5, ResNet, and AlexNet are a few examples of Convolutional Neural Networks which are earlier presented by researchers. Let's talk about these further. (He, Zhang, Ren, & Sun)

A. LeNet-5

This was one of the first convolutional neural networks which were accepted by people across the industry. It was designed for recognizing hand writings at first but afterward, it was referred to as an exceptional approach to Graph transformer networks. It is described thoroughly by Yann LeCun in their paper. A few notable things are that this neural network consists of 7 layers, it consists of a convolutional layer followed by a pooling layer which is said to be a sub-sampling layer. This is repeated at least twice before output feature maps are flattened & input is shared into fully connected layers for interpreting and predicting. In the diagram mentioned below, the architecture of LeNet-5 can be seen.

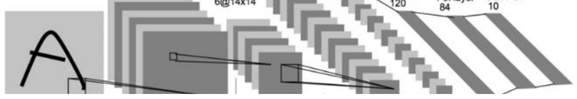


Figure 1: The architecture of the LeNet-5 Convolutional Neural Network

B. AlexNet

This neural network was created and published by Alex Krizhevsky, it was defined for classifying images from a dataset with a thousand categories. It resulted in the creation of many other innovations and betterments in convolutional neural networks, it suggested that deep and successful models can be defined without using unsupervised approaches. If we talk about its architecture, after convolutional layers it persists of a rectified linear activation function as the nonlinearly instead of using old functions, a softMax activation function was also, which is currently used by the industry. Instead of an old pooling method, a new max pooling method was defined in this. To rectify overfitting, a new dropout strategy was used in the middle of fully connected layers. Surely the model presented is a complex one and it is built on patterns defined by LeNet-5, the picture below sums up the model architecture, which in the current given case was broken down into 2 pipelines to train on the present GPU technology.

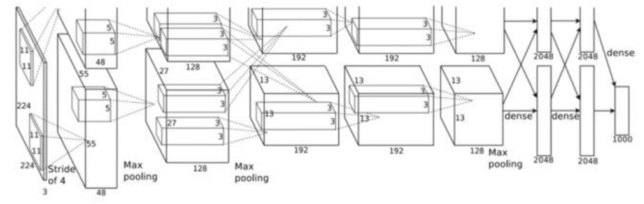


Figure 2:The architecture of the AlexNet Convolutional Neural Network for Object Photo Classification (taken from the 2012 paper).

C. ResNet

ResNet is one of the most important innovations in convolutional neural networks which was also created for image recognition at first. When created it consisted of 152 layers, it also referred to the concept of a leftover block that uses shortcut connections, which is central to the model's build.

These are network design connections in which the input is maintained unweighted and directed to a higher layer, for example by bypassing the layer underneath. A residual block is a sequence comprising two 2 convolutional layers with rectified linear activation function, such as the shortcut connection, where the residual block's result i.e., its output is connected to the residual block's input. When the design of the residual block's input differs from the design of the residual block's output, 1x1 is used to use a projected version of the input.

In contrast to identity shortcut connections, these are termed as projected shortcut connections. The researchers start with a simple network, with the deep neural network with small filters (3x3), clustered convolution layers, with no pooling in the middle of layers, and a pooling layer at the end of the feature detector section before the completely integrated output layer having softmax activation function. By introducing shortcut links to create residual blocks, the plain network can be upgraded into a residual network.

A shortcut connection's input is generally the same size as that of the leftover block's output. The image below, from left to right, illustrates the architecture of a plain convolutional model, and a refinement of the plain convolutional with residual modules termed a residual network. (Brownlee, n.d.)

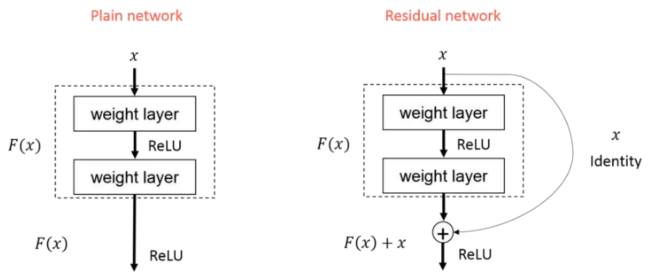


Figure 3:3 The architecture of the Residual Network for Object Photo Classification (taken from the 2016 paper).

A residual network with multiple layers that addresses the degradation problem will be used in the coursework. It expressly permits these layers to befit a residual mapping instead of assuming every few stacked layers will be the same as a required underlying mapping. Fundamentally, we will let these layers match another mapping of F(x) = I(x)-x, indicating the required underlying mapping as I(x). The initial mapping gets transformed into F(x)+x. As a result, enhancing the residual mapping should be easier than enhancing the initial one. If identity mapping was ideal, reducing the residual to zero would be faster than fitting an identity mapping with a few nonlinear layers. F(x) +x can be produced using forward-feed networks with shortcut connections. These are connections that bypass one or more layers. In this instance, identity mapping is handled by the shortcut connections, and its result is appended to the stacked layers' result (Fig. 3).
Identity shortcut connections introduce no new parameters or computational complexity.
Adaptive Moment Estimation will be used to optimize the complete network end-to-end, and it is simple to implement using standard libraries such as PyTorch without altering the solvers. In the course work, comprehensive experiments were performed to resolve the degradation problem and evaluate & optimize the used model.

II. Model & Architecture

A. ResNet-9

Residual Neural Networks mainly use shortcut connections which are also referred to as identity connections. These typically function by hopping across one or perhaps more layers and establishing shortcuts within those. The main reason for using these connections is to solve the problem of descending gradient which can be seen in deep networks. These connections help eliminate the degradation problem by reusing the activations from the preceding layer. These connections typically accomplish more than just bypassing the connections, it uses previous layer activations as well. Skipping the connection reduces the network, enabling it to train faster than it used to. Following connection reduction, the layers are stretched to ensure that the residual network can train and investigate a broader feature set. (Rajagopalan)

There are many versions of the residual network. ResNet-9 which is being used has 9 deep layers. This was designed to enhance the efficiency of convolutional layers by productive operations. Its architecture is structured in a way that it accepts the efficient operation of multiple convolutional layers. It is said that using multiple deep layers in a network can result in disturbed output, the phenomenon is referred to as the vanishing gradient problem. It usually takes place when networks are trained with backpropagation and if it minimizes weight using gradient descent. Having multiple layers will eventually cause continual multiplication which finally results in making the gradient smaller and eventually, it will vanish, this process will be responsible for the performance degradation.

B. Architecture

RESNET9 consists of 9 deep layers, 3x3 kernel, it contains 4 layers each having convolutional nets, batch normalization & rectified linear unit function, which is similar to each other. Among these layers, 2 fully connected layers consist of 2 residual blocks each and each such block contains 2 weight layers with an identity connection which is attached to the output of the second weight layer with a rectified linear activation unit. If it gives the same output as the input passed to the convolutional layer then the identity function will be used, else convolutional pooling will be done on the skip connection. While pre-processing, augmentation takes place which defines the input size given to the model which is (32, 32, 4) in the case of ResNet-9, where 32 shows the height and width whereas 4 shows the pixels of padding done. Output is being carried out as a fully connected layer to the subsequent sequential layer. At the end of the last layer in ResNet-9, it consists of an extra sequential layer. The rectified linear unit function will be fed to the classifier layer with pooling and flattening. Afterward, a linear layer (515,10) is implemented. RestNet-9 uses PyTorch framework.

Figure 4:Architecture of ResNet-9

ResNet-9 has been chosen for the problem as it is easy to optimize and implement and it can be trained faster when compared to plain neural networks, plain neural networks exhibit more training errors as dept grows as discussed earlier and ResNets can easily benefit from much-increased depth, producing results that are far superior to earlier networks.

III. Training and optimization

First, we will start loading data in the datasets created with PyTorch. Channel-wise data normalization & randomized data augmentation is carried out before moving to data loaders with the help of which we will define the size of the datasets and no. of workers. Afterward, the Residual block is defined, by which the output from convolutional layers is summed up with the original input passed. This is one of the things which is an addition to our model and makes it different from plain convolutional neural networks, another is batch normalization which is used after every convolutional layer, and it normalizes the output from layers and improves the performance of our model.

Batch Normalization discussed above is a technique that removes the mean and normalizes the standard deviation of a channel of activations. Statistics μ(x) and σ(x) are computed over pixels in an image and examples in a batch. They are frozen at test time.

$$x \leftarrow \frac{x-\mu(x)}{\sigma(x)}$$

A learnable output mean β and standard deviation ϒ are usually applied, potentially undoing everything:

$$x \leftarrow \frac{x-\mu(x)}{\sigma(x)}\gamma + \beta$$

Before training the model, we will define a function called fit to make some improvements. Some of these are Learning rate scheduling with which the learning rate can be changed for every batch, Weight decay which stops the weights to be very high, Gradient clipping which controls the value of gradients and keep it low, resulting in controlling changes in parameters due to gradient values being high, etc. Fit function has been implemented as its results add up insight with every epoch while training the model.

For training and optimizing the model, we will be using the Adaptive Moment Estimation technique as it contains parameters such as momentum & adaptive learning rates for speeding up the learning. It can be seen in the results of the experiment in the next section as well.

Adaptive Moment Estimation technique i.e., Adam, which calculates adaptive learning rates for each parameter is being used for the model optimization. The algorithm can be seen in fig. below.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Parameters required for implementing the above presented optimizer-

β1 — This is used for delaying the running average of the gradient

β2 — This is used for delaying the running average of the square of the gradient

α — Step size parameter

ε- It is to prevent Division from zero error. (Hong)

The algorithm has been defined uniquely with the help of passing parameters to it, which are stated below-

a) Maximum Learning rate: 0.01
b) Momentum=0.1
c) Epochs: 40
d) Weight Decay: 1e-04
e) Batch size =400
f) gradient clip = 0.1

For minimizing the losses, a loss parameter has been included which uses categorical cross entropy which determines the quantity that a model should try to minimize throughout training.

IV. Results

We have tested our model with and without batch normalization, below table shows the accuracy of the model with respect to epochs without batch normalization being implemented and the batch-normalized version of our model. It is clear that the batch-normalized version helps in improving model performance as it removes the mean and normalizes the standard deviation of a channel of activations.

Table 1:Accuracy& Loss with & without Batch Normalization

Model Training	**Accuracy %**	**Loss**
Without Batch Normalization	67.91	0.91
With Batch Normalization	**93.18**	**0.23**

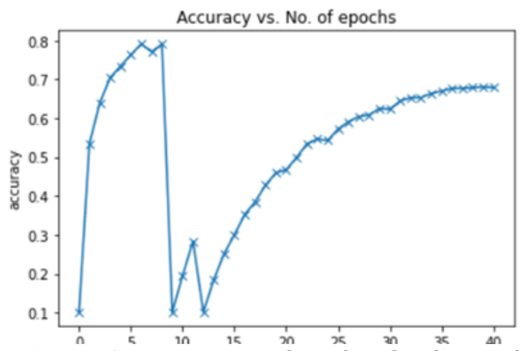
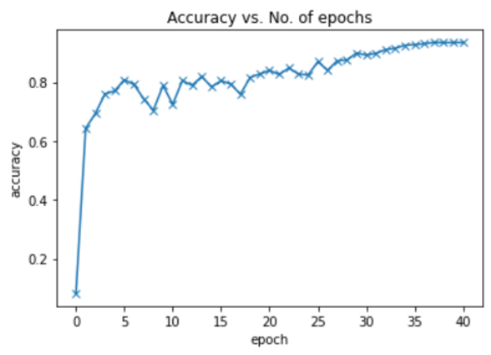


Figure 6:Accuracy wrt epochs without batch normalization

Figure 5: Accuracy wrt epochs with batch normalization



The above figures show the accuracy of the model optimized with the Adaptive Moment Estimation optimizer. It can be seen that the accuracy of the model increases with more epochs, but it does take more computational power and time.

Table 2:Accuracy & Loss for SGD and ADAM

Optimization	**Accuracy %**	**Loss**
SGD	90.96	0.31
ADAM	93.18	0.23

The above table shows the accuracy for SGD i.e., Stochastic gradient descent, and ADAM i.e., Adaptive Moment Estimation. In comparison, it is visible that ADAM performs better than SGD when given the same circumstances i.e. when the same or similar parameters are passed. Therefore, we will proceed with the ADAM optimization algorithm for optimizing our model.

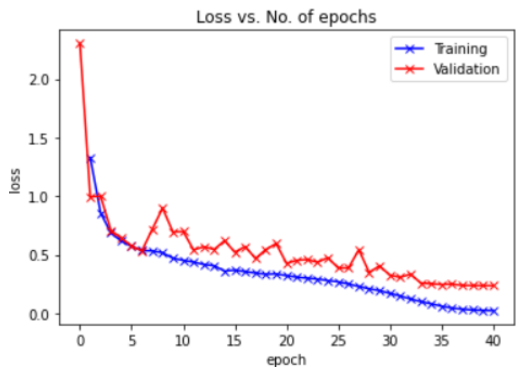


Figure 7:Loss wrt epochs for training & validation

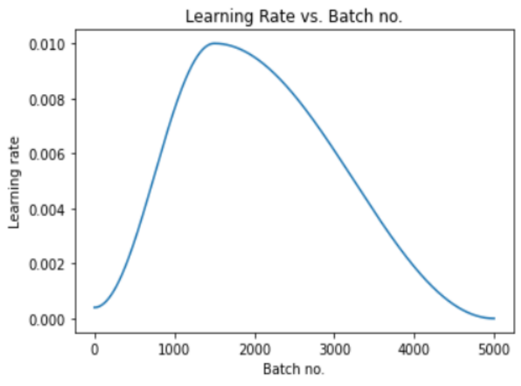


Figure 8:Learning Rate wrt batches

The above figure shows the Loss of the model optimized with the Adaptive Moment Estimation optimizer while testing & training. It can be seen that the Loss of the model decreases with more epochs, and at around 35 epochs, Loss almost became constant for both testing and training.

The learning rate starts with a low value and slowly increases for a few iterations to a maximum value, which is 0.01, and then gradually comes down to a very small value.

V. CONCLUSION

To sum everything up, at first, the result turns out to be that when batch normalization is absent, networks with standard initializations do not produce good output, and the output tends to be constant distribution in which input has been side-lined. With the introduction of batch normalization, this was improved. While working on training the model, during baseline training it can be seen that the config. near the parameters configurations are not performing well. By calculating gradients of the means of the per channel output distributions, we specifically determined such adjacent setups. We later extended this by computing gradients of variance and skew of per-channel output distributions, claiming that modifying these higher-order statistics would likewise result in a significant increase in loss. We demonstrated how batch norms lower gradients in these directions by inhibiting the propagation of changes to the statistics of internal layer distributions. In the end, when we compared two different methods of optimization for our dataset while training the model with the same parameters, SGD shows the almost same rate of learning as the ADAM, but the accuracy obtained at the end of 40 epochs in ADAM was higher when compared to SGD. Therefore, for our dataset and model, we will be implementing batch normalization within our model and ADAM with gradient clipping, weight decay, and controlled learning rate with a maximum learning rate defined.

REFERENCES

[1] (He, Zhang, Ren, & Sun)
[2] (Rajagopalan)
[3] (Brownlee, n.d.)
[4] (Hong)