,

Learning-Rate-Free Learning for Object Detection and Convolutional Image Classification By D-Adaptation

Project Report submitted by

Kadiyam Aswin Tirumala Someswar (421154)
Kadarla Suhas Krishna (421153)
B Karamthot Rakshitha (421124)

Under the supervision of

Dr. KARTHICK S
Assistant Professor

Department of Computer Science and Engineering,
National Institute of Technology, Andhra Pradesh

04-05-24

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY, ANDHRA PRADESH

Certificate

This is to certify the thesis entitled "Learning-Rate-Free Learning for Object Detection and Convolutional Image Classification By D-Adaptation" submitted by Kadiyam Aswin Tirumala Someswar (Roll No. 421154), Kadarla Suhas Krishna (Roll No. 421153), B Karamthot Rakshitha (Roll No. 421124) to National Institute of Technology, Tadepalligudam in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a record of bonafide research work carried out by them under my supervision and guidance. This work has not been submitted elsewhere for the award of any degree

Dr. KARTHICK S
Assistant Professor
Dept. of CSE

Place: Tadepalligudem
Date:

May 8, 2024

Table of Contents

List of Figures

List of Tables

Chapter 1

Abstract

In the two applications we worked on, namely Object Detection and Image Classification, the D Adaptation Technique showed increased metric results compared to scenarios without D adaptation. We utilized the Detectron2[4] model for object detection and obtained results based on the model with the datasets provided. Image classification, on the other hand, is a simple machine learning task that assigns a label to an image. In this case, we applied the Adam optimizer with D adaptation and compared the results with a normal image classifier lacking D adaptation. We used the Adam variant with D adaptation algorithm to obtain metric results. This variant of Adam adapts quicker than the SGD variant, resulting in better outcomes compared to SGD.

Chapter 2

Introduction

Detectron2 [4] is a Computer Vision Model which helps to detect objects using domain Adaptation. We used Detectron2[4] for a masked R-CNN object detection algorithm. Detectron2[4] is developed by Facebook AI Research (FAIR). Detectron2[4] is built on top of the PyTorch[6] deep learning framework and offers a wide range of pre-trained models. It gains More significance in Computer Vision due to its performance and ease of use. Masked R-CNN looks at a picture, it not only figures out where objects are located but also outlines it, such as drawing a box around it.

ADAM is a Adaptive Moment Estimation that combines aspects of both Momentum based Optimization and RMSprop. ADAM is an Optimization Algorithm that can be used in place of Stochastic Gradient Descent[7] to update network weights. In SGD, norms are unweighted, But in ADAM the norms are weighted. Norm is nothing but of Root mean square of previous gradients or squared gradients. The norms are weighted to control the influence of past gradients on the current update step. Adam adjusts how much it learns from past experiences by considering the history of gradients it has encountered during training. In the original Adam algorithm, at the start of training, there is a chance for the values to lean towards zero because of how the averages are set up initially. To address this issue, the weighted norms are corrected using a bias correction term.

Chapter 3

Literature Review

3.1 Learning-Rate-Free Learning by D-adaptation

The paper (Learning-Rate-Free Learning by D-adaptation) by Aaron DeFazio[1] and Konstantin Mishchenko[1], presents a new approach , known as D-Adaptation, which offers an automated method for setting learning rate, achieving optimal convergence rates for minimizing convex Lipschitz functions. We say a function is Lipschitz when the difference between the function values of any two points in a dimensional space is bounded by a constant multiplied by the distance between the points. A convex Lipschitz function is a function that satisfies both the convexity and Lipschitz function properties. In many cases, the loss functions of machine learning models are designed to be convex and Lipschitz continuous. By leveraging the convexity and Lipschitz continuity of the loss function, D-Adaptation ensures stable and optimal convergence.

There are algorithms or strategies in the context of machine learning and optimization, known as schedulers, that dynamically adjust the learning rate or the other hyperparameters during the training process. These adjustments are made based on some predefined criteria to improve the convergence process thereby enhancing the performance. The Schedulers solely focus on adjusting the parameters but not on the dynamic distributions of training data. So, The integration of the D-Adaptation technique with optimizer algorithms enables dynamic data handling and optimal convergence simultaneously, making it a valuable addition to modern machine learning workflows. Many real-world problems involve uncertainty or randomness, making stochastic optimization techniques essential for effectively addressing these challenges. Stochastic optimization involves optimizing an objective function where the objective function is stochastic. Adapting the D-Adaptation technique to stochastic optimization allows for flexible modification of hyperparameters in response to the stochastic nature of the objective function, thereby enhancing the adaptability of machine learning models across diverse tasks and datasets.

3.2 Less regret via online conditioning

Less regret via online conditioning is a concept aimed at minimizing the difference between the performance of an algorithm and the performance of the best possible action, in online learning

scenarios.Adagrad(Adaptive Gradient Algorithm) which belongs to the family of adaptive gradient algorithms exemplifies this principle. unlike the traditional gradient descent methods, Adagard adjusts the learning rate for each parameter individually based on the gradients of that parameter. The d-Adaptation technique is based on the ideas behind Adagrad but takes them further. By continuous adaptation of learning rates based on gradients and parameter updates, D-Adaptation aims to achieve optimal and rapid convergence. some fundamental aspects of Adagrad include Adaptive learning rate, Sparse data handling, and hyperparameter tuning. Adaptive learning rate involves adjustment of learning rates for each parameter based on the gradients of the parameter. Sparse data handling involves handling the datasets where the most of feature values are absent or zero. Hyperparameters are the parameters set before the training process and are not learned from the data, Hyperparameter tuning involves a systematic search for the optimal values of these parameters for the best performance of the model. All these aspects are inherited by the D-Adaptation technique to achieve optimal performance of the model.

3.3 The power of factorial powers

The Primal Averaging Technique, introduced by Defazio and extended by Defazio and Gower, keeps track of the average of the primal variables during the optimization process. The parameters that directly manipulate the objective function are subjected to certain constraints. these parameters are called prime variables. The hyperparameter beta, known as the momentum, improves the optimization process by relating the information of the past gradients to the current update step of the solution. the parameter beta is commonly depicted by a value ranging between 0 and 1, with a higher value indicating a strong momentum effect. During the optimization process, the algorithm keeps track of the average of primal variables and is updated iteratively along the process. The momentum term(beta) decides the amount of influence the past gradients have on the current update in the optimization process. it navigates the process along the direction in which the gradients consistently point, effectively helping to escape the saddle points. several optimizers utilize momentum to improve optimization performance, but the usage scenarios vary according to the optimizer. As per the research, Adam and RMSProp are known to perform well with the incorporation of momentum-like features over the other optimizers. Despite the performances, the choice of the optimizer always depends upon the problem at hand. Primal Averaging Technique effectively handles dynamic data distributions where the distribution of the data changes over time or across distinct segments within the datasets. As Dynamism is the core part of the D-Adaptation strategies, the integration Primal Averaging Technique improves the convergence properties and stability, especially in scenarios with noisy or non-smooth objective functions.

Chapter 4

Datasets

Datasets are valuable resources in many machine learning and computer vision tasks. It plays a crucial role in solving problems. We have utilized datasets such as the COCO Dataset, CIFAR-10, CIFAR-100 data sets. These datasets are widely used in the fields of object detection and image classification.

4.1 COCO Dataset

COCO dataset, short for Common Objects in COntext, is a large-scale dataset used for evaluating algorithms and models for computer vision tasks. It is tailored for vision tasks such as object detection, segmentation, and captioning. The dataset comprises tens of thousands of images, with labels provided for each image, including object instances, segmentation masks, and captions. labels or annotations for object instances details such as coordinates and dimensions of bounding boxes outlining the object. These annotations enable accurate recognition of objects within images. Image segmentation is a fundamental step in computer vision tasks. The segmentation task can be categorized into several types based on the required criteria, two of the important categorizations are semantic and object segmentation. object segmentation involves precisely outlining and separating individual objects or entities within an image, effectively partitioning the image into distinct regions corresponding to each object or entity present in the scene. semantic segmentation involves pixel-wise labeling of the entire image, allowing the understanding of the semantic meaning of each pixel. The instance segmentation involves assigning each pixel a label and a unique identifier. Several segmentation techniques are applied to COCO images to ensure better understanding of images and real-world scenes. The diversity of the dataset ensures that models trained on the COCO dataset can generalize well to real-world situations.

4.2 ClFAR-10

CIFAR-10, a popular dataset used for training image classification models. it consists of sixty thousand 32 cross 32 color images, it is a labeled subset of 80 million small images collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The number 10 in CIFAR-10 stands for the

10 output image classifications which the model is to be trained upon. Helps in the evaluation and training of machine learning models, and algorithms. Unlike the other popular datasets like ImageNet, CIFAR-10 is less complex and manageable. along with the other postulates, its availability and accessibility make it preferable. The classification outputs include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck as the classes in the order. The dataset is divided into 6 equal sets, 5 sets for training and one for testing. Each set consists of exactly 10000 random images of available 10 classes. In the test set, the images of different classes are in equal proportionate. Unlike the test set, any one of the training sets has no obligation, but the overall training set does have exactly 5000 images from each class.

4.3 CIFAR-100

CIFAR-100 is similar to CIFAR-10 but contains 100 classes, Each containing 600 images. CIFAR-100 consists of 60,000 32 x 32 color images, These images are organized into 100 classes. The dataset is split into training and testing sets, with 50,000 images for training and 10,000 images for testing. The 100 classes are divided into 20 superclasses, each containing 5 fine-grained classes. CIFAR-100 is mostly suitable for fine-grained classification tasks. Fine-grained Classification involves dividing images into subclasses within a superclass. Our goal is to distinguish between these subclasses based on the differences in the features of the images. The 20 superclasses in the CIFAR-100 are Aquatic mammals, Fish, Flowers, Food containers, Fruit and vegetables, Household electrical devices, Household furniture, Insects, Large carnivores, Large man-made outdoor things, Large natural outdoor scenes, Large omnivores and herbivores, Medium-sized mammals, Non-insect invertebrates, People, Reptiles, Small mammals, Trees, Vehicles 1 (land), Vehicles 2 (water and air). For Example, For aquatic mammals, we have beaver, dolphin, otter, seal, and whale as the subclasses.

Chapter 5

Proposed Method

5.1 Why Only Masked R-CNN in Object Detection?

The choice of object detection model depends on a variety of factors based on specific needs. We choose Mask Region-based Convolutional Neural Network(Mask R-CNN).

Instance Segmentation : Mask R-CNN is well suited for tasks requiring both object detection and segmentation as well. Segmentation refers to obtaining precise boundaries of objects in the images in a box form or in the outline manner. Instance segmentation provides pixel-wise labeling for each object instance of an image. this property allows for fine-grained analysis and understanding of complex images. Instance segmentation algorithms are adept at managing complex scenarios where multiple objects of the same or different classes overlap in an image. The complexity of the task lies in not only objectifying the objects in an image but also differentiating between separate occurrences of objects belonging to the same category. Despite a few challenges including occlusion, and complex object shapes, the diverse applications and remarkable performance in instance segmentations contribute to the reasons why Mask R-CNN is considered preferable.

Flexibility : Mask R-CNN gives a flexible framework that can be adapted for various object detection tasks. Suppose you are trying to find cells in a microscopic image, we can use Mask R-CNN. It's not just about finding cells, we can also teach it to identify different types of cells by Mask R-CNN. Mask R-CNN provides the flexibility needed to accommodate different scenarios and allows for experimenting on different fine-tuned parameters, and configurations leading to high and improved performance of computer vision tasks.

Top Performance : Mask R-CNN gives the highest performance in the world of Computer Vision. When it comes to spotting and outlining objects in images, Mask R-CNN is the best choice than other models.Its ability to effectively extract features from images using convolutional neural networks (CNNs), accurately detect object bounding boxes using region proposal networks (RPNs), and precisely segment object instances using fully convolutional networks (FCNs). Additionally, its multi-task learning approach optimizes both object detection and segmentation objectives simultaneously, enhancing overall performance and efficiency. Because of these

technical abilities, Mask R-CNN outperforms other models in computer vision tasks. It can find objects more accurately, handle complex situations, and work well even with large amounts of data.

All-in-One Training : We do not need to do training separately for object detection and segmentation. We can train all at once, It learns both object detection and segmentation at once. This makes it easier and faster to train.

Easy to Use : We can find Mask R-CNN in popular tools like TensorFlow and PyTorch[6]. We can directly make use of it by importing it into our environment. The available modules in TensorFlow and PyTorch[6] make it easy for users to use.

5.2 Why Only ADAM?

ADAM(Adaptive Moment Estimation) is an optimization Algorithm commonly used in Training image classifiers.It blends the postulates of AdaGrad and RMSprop optimization techniques, integrating their strengths while addressing some of their limitations. The intuition of maintaining a per-parameter learning-rate that is based on the average of recent magnitudes of the gradients is inherited from the AdaGrad and RMSProp algorithms. The simple Mathematical intuitions used in the algorithms make it easy to implement and use. Adam optimizers can handle noisy or sparse gradients effectively. The several attributes of the Adam optimizer make it an optimal choice for advanced optimization scenarios.

Efficient Optimization : Adam is known for its efficiency by combining or adding weights to norms such as momentum to navigate the higher dimensional parameter space effectively. Adam allows for faster convergence compared to other optimization algorithms like stochastic gradient descent(SGD)[7], Adam combines both adaptive learning rate methods by adjusting the learning rate dynamically and momentum based optimization methods through momentum estimations using the primal averaging[5] technique, following the approach of Defazio (2020) and Defazio Gower (2021)[3].

Little Memory Requirements : Unlike the other optimization techniques like SGD, the memory requirements are relatively lesser. as its ability to handle different learning rates for different parameters and to adjust the learning rates dynamically makes it much faster and makes the usage of memory relatively less. momentum estimations which keep track of gradients information reduce the need for buffering during training. Adam Optimizer's streamlined procedures and computational algorithms enhance its overall memory efficiency. While its computational complexity aligns with other adaptive optimization techniques like RMSprop and AdaGrad, Adam typically imposes lower memory overhead. This advantage stems from its compact handling of adaptive learning rates, requiring less storage space for additional parameters.

Robustness to Sparse Gradients : With dynamic adjustment of learning rate and momentum based behaviour, the sparse and noisy gradients are handled well. Adam optimizer can handle situations where only a small fraction of the parameters have significant gradients during training, while the majority have gradients of approximately zero. In other words, only a few parameters contribute significantly to loss function which can lead to instability in optimization as certain parameters do not receive the optimal updates. In such cases, traditional optimization algorithms like stochastic gradient descent (SGD) can not effectively handle updating the parameters, leading to a slower convergence rate and suboptimal performance of models. Dynamic adjustment of learning rates for each parameter prevents stagnation in parameter updates. parameters with sparse gradients receive smaller updates and the parameters with larger gradients receive larger updates. this adaptability prevents overfitting and ensures better generalization to unseen data, faster convergence rates, and optimal performances.

Chapter 6

Methodology Techniques

6.1 Masked R-CNN

Masked R-CNN is purely based on Image Segmentation. It is of types : Semantic Segmentation, Instance Segmentation. Semantic Segmentation classifies sets of the same categories as single without differentiating them. While Instance Segmentation classifies all the objects(each object as a single category).

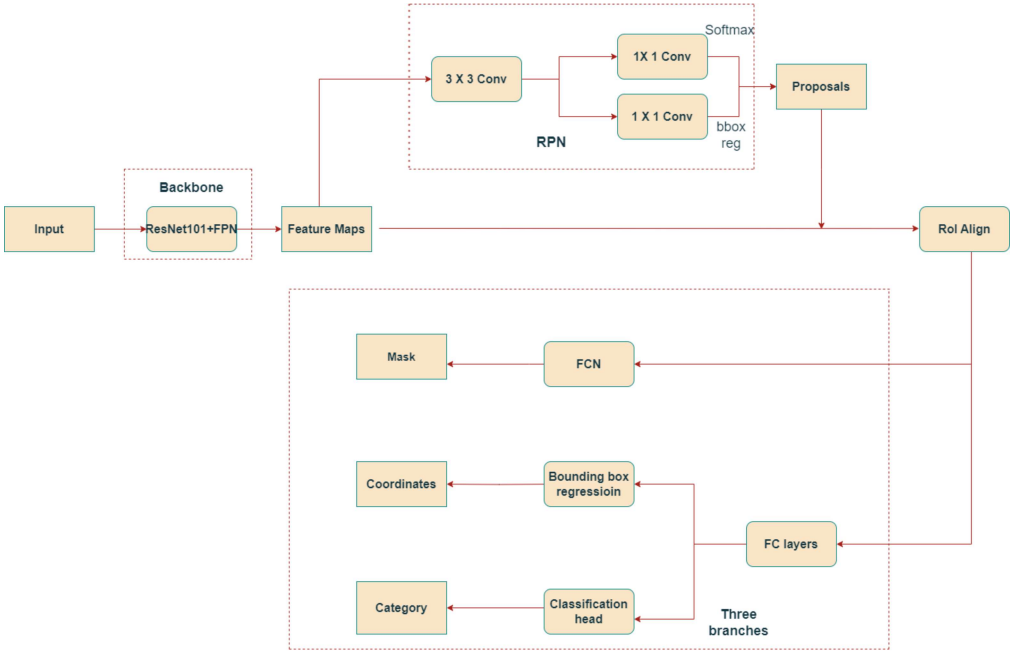

Figure 6.1: The Structure diagram of Mask-RCNN algorithm[4].

6.1.1 ResetNet101 + FPN

A deep CNN architecture is used for computer vision tasks. ResNet stands for Residual Network and the number 101 stands for the number of layers it consists of. There are other variations too like ResNet-50 which are relatively less powerful than ResNet-101. The residual connections carry the information bypassing the subsequent layers or network, encouraging feature reuse and preventing the loss of important information during the training phase. Skip connections, also known as shortcut connections, which carry the information directly without any mathematical manipulation, help in avoiding the vanishing gradients. these connections facilitate training deep neural networks, which are embodied in ReseNet101. FPN stands for Feature Pyramid Network which addresses the challenge of Scale invariance, the property of a system for which the system remains unaltered when the size of the input is changed. In the context of Computer vision it means the size of the image. The resulting architecture is often used in Mask R-CNN for feature extraction of images and mapping them to 2D representations of learned features, typically called feature maps.

6.1.2 Convolution Layers

The fundamental blocks of convolution neural networks. CNN layers are involved in the task of extraction of meaningful features and representation of the image through the feature extraction process. The extraction process involves convolution operations. The process of encoding the images occurs through convolution operations. The operations include feature extraction, stride, padding, introducing non-linearity, and others. The features of the image are detected through multiple filters applied to the input image. The filters are the matrices that slide over the matrix representation of the input image. the submatrices that the kernels point out are the regions of the image. As the feature matrices do have representations of the features, finding a region that is similar to the feature matrix means that we have detected that particular feature in the input image. The similarity between the kernels and feature matrices can be detected by passing them as inputs to similarity detection functions. All the detected feature representations of the image are concatenated to produce feature maps. These feature maps also represent the features of the input image. In other words, feature maps represent a collection of features that indeed become a feature of the input image. The stride determines the spatial distance by which the filter should move across the data. adding extra border pixels to the input data to prevent information loss is known as padding. the process produces a representation of the input image in the form of a flattened tensor or one-dimensional array..

6.1.3 SoftMax layers

Typically positioned as the final layer of any neural network architecture or any sub-neural networks mapping their outputs to possible predictions. it computes element-wise raw scores. the scores are then weighed to produce probabilities that represent the classes. Each probability score says about the possibility of respective input belonging to a particular class or category. The output result is represented as a 1-dimensional tensor. The values of the output tensor always sum to one. SoftMax

layers are often paired with cross-entropy loss functions during training to compute the prevailing loss. Minimizing cross-entropy loss produces a probability distribution that closely aligns with ground truth.

6.1.4 ROI Align

ROI stands for Region of Interest, typically representing a region of the input image that does have objects of interest. Regional Proposal Network[RPN], a small neural network that operates on feature maps, takes responsibility for the generation of object-specific bounding boxes[regional proposals], from the input image. These RPN outputs are forwarded to subsequent Convolution neural network layers pipeline as inputs. The primary responsibility of RPN is to generate region proposals, which are the regions in the input data of the image in which the model is interested. These regions of interest are produced through the feature extraction process. RPN predicts whether an object that is of interest is present. The prediction can be done through binary classifications. this is achieved through logistic regression, which outputs the possibility of the presence of a particular object. The outputs of RPN are the ROIs. ROI Align is a technique that addresses misalignment issues of features that are extracted from input images. the general tasks involved are pooling, interpolation, and enabling sub-pixel accuracy. pooling involves dividing the ROIs into grids and performing max-pooling in those fixed grids, producing dimensionally reduced feature maps. There are issues like misalignment with the pooling operations which can be addressed through interpolation. Sub-pixel accuracy means the ability to make predictions precisely about the objects in an image at very low resolutions. the segmented outputs are then passed to the subsequent layers for classification tasks.

6.1.5 BBOX Reg

BBOX stands for Bounding box regressions, a task that involves refining the coordinates of bounding boxes. it localises the data by predicting outline or boundary coordinates. it refines by considering the ground-truth object boundaries, in the training phase.

6.1.6 FCN

FCN stands for Fully Convolutional network, a neural architecture designed for assigning each pixel of an input image a label. Typically it is an encoder-decoder architecture, the encoder part extracts the feature information, capturing both low-level and high-level information. The decoder part is responsible for pixel-wise predictions, known as upsampling, to match the input resolution. The final prediction layer, typically a SoftMax layer, produces class probabilities. The loss is calculated through gradients via backpropagation. In the training phase, the loss is calculated through cross-entropy loss that computes pixel-wise loss function score which is compared to a ground-truth segmentation map. The aggregated loss is used to update the model parameters via backpropagation.

6.2 Adam Technique

$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (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)

Figure 6.2: Adam Algorithm With out D-Adaptation

$$
\begin{aligned}
&g_k \in \partial f(x_k, \xi_k) \\
&m_{k+1} = \beta_1 m_k + (1 - \beta_1) d_k \gamma_k g_k \\
&v_{k+1} = \beta_2 v_k + (1 - \beta_2) g_k^2 \\
&A_{k+1} = \operatorname{diag}(\sqrt{v_{k+1}} + \epsilon) \\
&x_{k+1} = x_k - A_{k+1}^{-1} m_{k+1} \\
&\textit{Learning rate update} \\
&s_{k+1} = \sqrt{\beta_2} s_k + (1 - \sqrt{\beta_2}) d_k \gamma_k g_k \\
&r_{k+1} = \sqrt{\beta_2} r_k + (1 - \sqrt{\beta_2}) d_k \gamma_k \left\langle g_k, s_k \right\rangle_{A_{k+1}^{-1}} \\
&\hat{d}_{k+1} = \frac{r_{k+1}}{(1 - \sqrt{\beta_2}) \left\| s_{k+1} \right\|_1} \\
&d_{k+1} = \max\left(d_k, \hat{d}_{k+1}\right)
\end{aligned}
$$

Figure 6.3: Adam Algorithm With D-Adaptation

6.2.1 Parameters

6.2.1.1 First Moment Decay Rate (beta1)

The decay rate for the first-moment estimation(m), is a hyperparameter that controls the exponential decay of gradients. the default value used is 0.9. the beta1 prioritizes the recent gradients while holding on to the past gradients. As per theoretical aspects, the beta1 does not change over time, it is fixed before it is used.

6.2.1.2 Second Moment Decay Rate(beta2)

Decay rate for second moment estimate(v), a hyper parameter that controls the exponential decay of squared gradients. the default value used is 0.999. it is similar to the first moment decay rate(beta1) but has different purpose of usage. it also assigns more weightage to the recent squared gradients while holding on to past squared gradients.

6.2.1.3 First Moment Estimation (m)

The hyperparameter ”m” provides the information about the mean of gradients. typically it says about the direction of gradients, whether they are increasing or decreasing. as per the formulae, it controls the weightage given to recent gradients and the past gradients. the value is updated for every iteration by combining the current gradient with previous estimate.it plays a part in deciding direction in which the learning should be adjusted. it influences the update of parameters, helping in navigation to optimization landscape. it is biased towards zero, it says that initial values start with lower magnitudes and gradually increase along the successive iterations. the biases say about initial behaviors of estimate parameters.

6.2.1.4 Second Moment Estimation (v)

Represents the exponential weighed past gradients, also called as Exponentially weighted moving average [EWMA]. it is used in tracking the uncentered variance of the gradients. this is used in dynamic learning rate setting leading to efficient and stable convergence. like first momentum estimate (m)it is biased towards zero, saying that initial values start with lower magnitudes and gradually increase along the successive iterations.

6.2.1.5 Epsilon (e)

A small constant added to the denominator in the parameter update rule. it is used to prevent the division with zero, which eventually leads to run time error. it’s value depicted to be 1e-8. it has no influence on update of parameters but prevents the case of division with zero.

6.2.1.6 Diagonal Matrix (A)

In d-adaptation technique, along with the second moment estimate, it is used to update the parameters. typically used for scaling of each parameter update direction based on the second moment estimate(v).

6.2.1.7 Learning Rate

It is used in the update rule of parameters. it influences the amount of change in the parameters. generally, an optimal value is considered, but when the value is either too small or too large causes a problem in the optimization process. when it is too large, over the process the parameters oscillate quickly causing an increase in computational time and memory. when it is too low, the same problem of computational time and memory increment arises. generally, in optimizers like Adam and SGD, the default value of learning-rate is set to 0.001.

In d-adaptation, the learning rate is set to the default value which is one. As it includes setting the learning rate dynamically, the value is given in terms of first and second momentum estimates. the values of ”m” and ”v” and their mathematical combination act as the learning rate. the learning rate update includes two parameters, scaling factor(s) and learning rate parameter (r), and adaptive update direction parameter(d). the values ”s” and ”v” influence the value of the update direction parameter. the update direction parameter is used in the calculation of the first momentum estimate (m).

6.2.2 Layers

6.2.2.1 Convolutional layer 1 (self.conv1)

Input : This layer takes an input image, with 3 color channels(RGB).

Operation : It applies 32 filters to the input image. Each filter is looking at each pattern or feature. The filters slides across the image to detect features like edges, colors, etc.,

Output : After applying the filters, it produces 32 feature maps, each one highlights the specific feature in the input image.

6.2.2.2 Convolutional layer 2 (self.conv2)

Input : The 32 feature maps from the previous layer.

Operation : Similarly to the first layer, It applies 64 filters to the input feature maps. This time it extracts more complex patterns and features.

Output : After applying the filters, this layer produces 64 feature maps, each one highlights the specific feature in the input image.

6.2.2.3 Convolutional layer 3 (self.conv3)

Input : The 64 feature maps from the previous layer.

Operation : Similarly, It applies 128 filters to the input feature maps. This time it extracts even more complex patterns and features.

Output : After applying the filters, this layer produces 128 feature maps, each one highlights the specific feature in the input image.

6.2.2.4 Max Pooling Layer(self. Pool)

Operation : This layer decreases the dimensions of the feature maps by taking the maximum value with each pooling region. It helps in reducing the computational complexity and focuses on more relevant information.

Output : After pooling, The size of the feature maps is halved. It will retain the most important features.

6.2.2.5 Fully Connected Layer 1(self. fc1)

Operation : This layer flattens the pooled feature maps into a one-dimensional vector. It passes it through a fully connected neural network with 512 neurons. Each neuron is connected to every element in the input vector and applies a weighted sum followed by a non-linear activation function.

Output : It produces a feature vector with 512 dimensions. It gives high level representation of input image features.

6.2.2.6 Fully Connected Layer 2(self. fc2)

Operation : Similar to FC 1. It passes it through a fully connected neural network with 256 neurons. Each neuron is connected to every element in the input vector and applies a weighted sum followed by a non-linear activation function.

Output : It produces a feature vector with 256 dimensions. It gives even more high level representation of input image features.

6.2.2.7 Output Layer (self.fc3)

Operation : This layer is the final Output Layer of the Network. It takes feature vector from previous layer and maps it to 100 output classes(one for each category in CIFAR-100) if we take CIFAR-100 Dataset or to 10 output class(one for each category in CIFAR-10) if we take CIFAR-10 Dataset.

Output : If we take the CIFAR-100 Dataset, The Output of this layer is a vector with 100 elements, representing the predicted probabilities for each class in CIFAR-100.

6.2.2.8 Dropout Layer(self. Dropout)

Operation : Dropout is a regularization technique used to prevent Overfitting. It helps the network to generalise better for unseen data.

Output : The random layer randomly drops out 50 percent of the input units, it helps the network to learn more representations of the data.

6.3 Forward Pass

6.3.1 Input('x')

The input x is an image tensor with shape (batch size=64),3,height,weight) representing a batch of RGB images.

6.3.2 Convolutional Layer 1 (self.conv1), Activation (ReLU), and Max Pooling(self. Pool)

x=self.conv1(x) : Apply the first Convolutional layer to the input x to extract features.
x=torch.nn.functional.relu(x) : Apply the ReLU activation function to introduce non-linearity.
x=self. Pool(x) : Apply max pooling to down sample the feature maps.

6.3.3 Convolutional Layer 2 (self.conv2), Activation (ReLU), and Max Pooling(self. Pool)

x=self.conv2(x) : Apply the second Convolutional layer to the feature maps from the previous layer.
x=torch.nn.functional.relu(x) : Apply the ReLU activation function to introduce non-linearity.
x=self. Pool(x) : Apply max pooling to down sample the feature maps.

6.3.4 Convolutional Layer 3 (self.conv3), Activation (ReLU), and Max Pooling(self. Pool)

x=self.conv3(x) : Apply the third Convolutional layer to the feature maps from the previous layer.
x=torch.nn.functional.relu(x) : Apply the ReLU activation function to introduce non-linearity.
x=self. Pool(x) : Apply max pooling to down sample the feature maps.

6.3.5 Flattening

x.view(-1,128*4*4): Reshape the feature maps into a one-dimensional vector.

6.3.6 Fully Connected Layer 1 (self.fc1) and Activation(ReLU)

x=self.fc1(x) : Apply the first fully connected layer to the flattened feature vector (x.view(-1,128*4*4)).
x= torch.nn.functional.relu(x) : Apply Relu Activation.

6.3.7 Dropout

x= self.dropout(x) Apply dropout regularization to the output of the first fully connected layer.

6.3.8 Fully Connected Layer 2 (self.fc2) and Activation(ReLU)

x=self.fc2(x) : Apply the second fully connected layer to the output of the dropout layer.
x= torch.nn.functional.relu(x) : Apply Relu Activation.

6.3.9 Dropout

x= self.dropout(x) Apply dropout regularization to the output of the second fully connected layer.

6.3.10 Output Layer

x= self.fc3(x) AApply the final fully connected layer to obtain the output logits representing the predicted scores for each class.

6.3.11 Return Output

Return the final output logits, which will be useful for finding the loss and making predicting during training and inference.

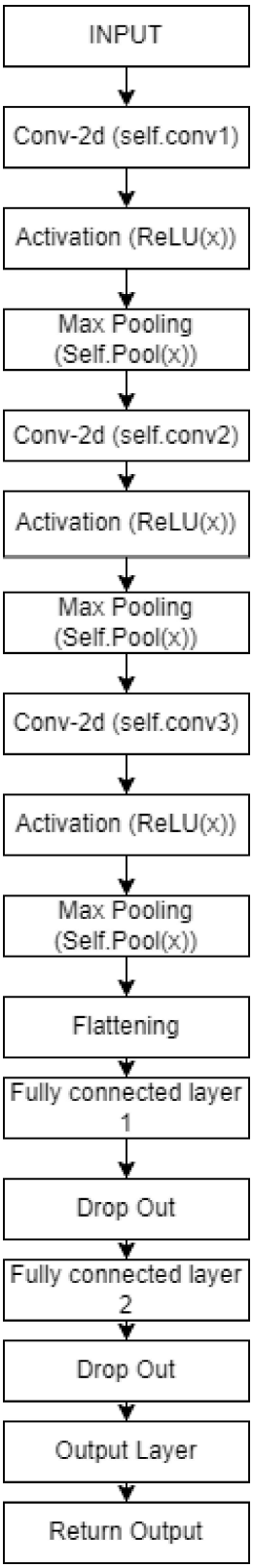

Figure 6.4: Layers Of Adam.

Chapter 7

Experiments and Results

7.1 Comparing Accuracy

We are comparing the accuracy of the Image classifier on the Test Set with the ADAM Optimization technique with and without D-Adaptation for both the CIFAR-10 and CIFAR-100 Datasets.

Table 7.1: Accuracy on the test set with CIFAR 10 Dataset with 5 epochs

Method	**Accuracy**
Adam Without D-Adaptation	63.23%
Adam With D-Adaptation	68.00%

Table 7.2: Accuracy on the test set with CIFAR 100 Dataset with 5 epochs

Method	**Accuracy**
Adam Without D-Adaptation	25.62%
Adam With D-Adaptation	27.19%

7.2 Implementation Details

We used Convolutional Image Classifier model to find the accuracy between the two datasets by adam technique with and with out d-adaptation.

```
[04/13 12:12:11 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[04/13 12:12:11 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.01 seconds.
[04/13 12:12:11 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[04/13 12:12:11 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.00 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.736
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.837
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.808
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.545
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.907
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.242
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.758
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.758
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.588
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.930
[04/13 12:12:11 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 73.594 | 83.663 | 80.843 | 0.000 | 54.524 | 90.664 |
Loading and preparing results...
DONE (t=0.00s)
creating index...
index created!
[04/13 12:12:11 d2.evaluation.fast_eval_api]: Evaluate annotation type *segm*
[04/13 12:12:11 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.01 seconds.
[04/13 12:12:11 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[04/13 12:12:11 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.00 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.761
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.808
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.808
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.529
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.963
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.256
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.778
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.778
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.571
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.973
[04/13 12:12:11 d2.evaluation.coco_evaluation]: Evaluation results for segm:
```

Figure 7.1: Metric Results for Detectron Model

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1, Loss: 1.2904853525064182
Epoch 2, Loss: 0.9198102056980133
Epoch 3, Loss: 0.7625911615007673
Epoch 4, Loss: 0.6454524386416921
Epoch 5, Loss: 0.5434595562155594
Accuracy of the network on the 10000 test images: 68%
```

Figure 7.2: Output for Cifar 10 dataset with D-adaptation.

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1, Loss: 4.2089831076009805
Epoch 2, Loss: 3.6109540386273125
Epoch 3, Loss: 3.3119604727801155
Epoch 4, Loss: 3.095112749072902
Epoch 5, Loss: 2.9331651968724284
Accuracy of the network on the test images: 27.19%
```

Figure 7.3: Output for Cifar 100 dataset with D-adaptation.

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1, Loss: 1.686, Accuracy: 38.53%
Accuracy on test set: 48.54%
Epoch 2, Loss: 1.425, Accuracy: 48.66%
Accuracy on test set: 57.52%
Epoch 3, Loss: 1.295, Accuracy: 53.80%
Accuracy on test set: 59.64%
Epoch 4, Loss: 1.206, Accuracy: 56.88%
Accuracy on test set: 62.45%
Epoch 5, Loss: 1.140, Accuracy: 59.52%
Accuracy on test set: 63.23%
Finished Training
```

Figure 7.4: Output for Cifar 10 dataset with out D-adaptation.

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1, Loss: 4.255, Accuracy: 4.31%
Accuracy on test set: 9.82%
Epoch 2, Loss: 3.858, Accuracy: 9.24%
Accuracy on test set: 15.28%
Epoch 3, Loss: 3.638, Accuracy: 12.82%
Accuracy on test set: 20.16%
Accuracy on test set: 24.03%
Epoch 5, Loss: 3.339, Accuracy: 18.21%
Accuracy on test set: 25.62%
Finished Training
```

Figure 7.5: Output for Cifar 100 dataset with out D-adaptation.

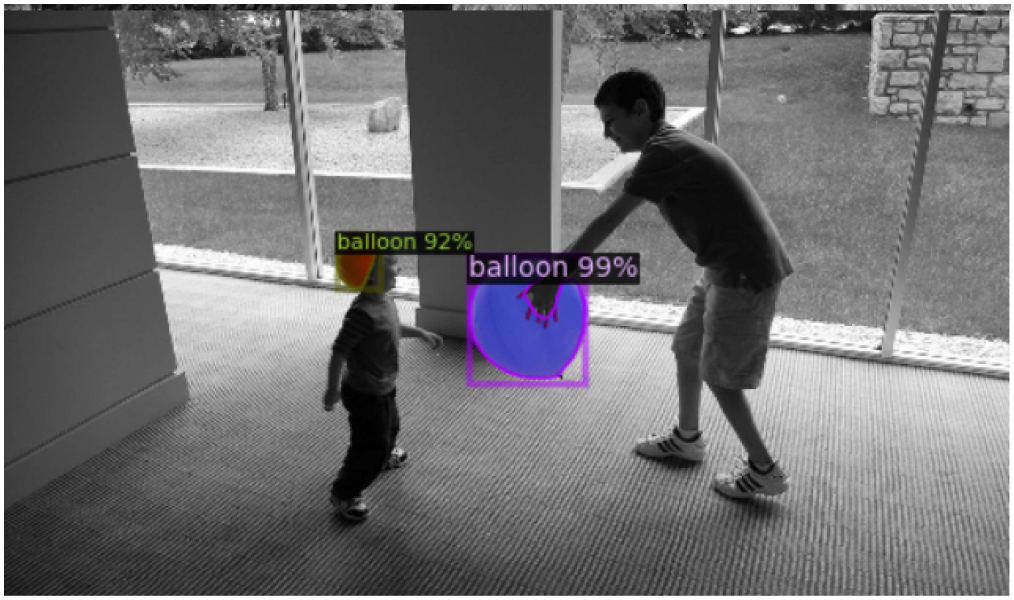

Figure 7.6: Object Detection Result

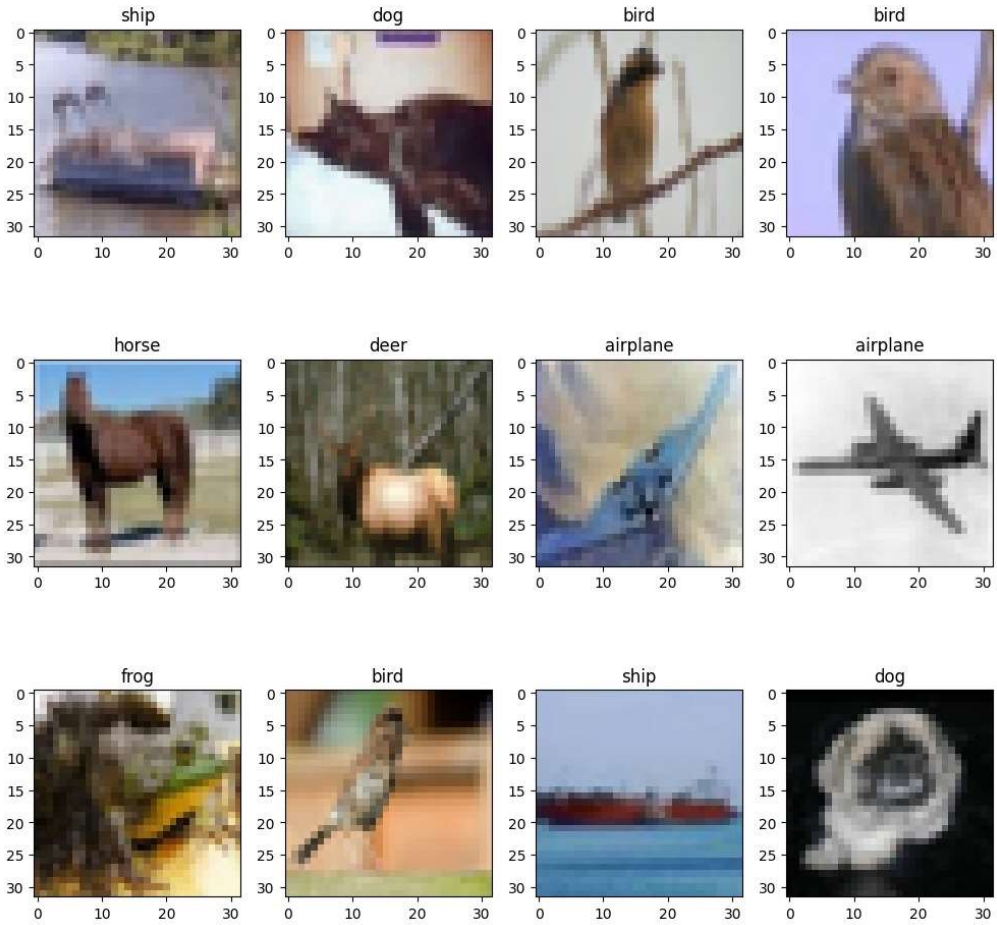

Figure 7.7: Image classification result

Chapter 8

Conclusion

The results obtained from our image classification task utilizing the Adam optimizer with d-adaptation technique demonstrate promising outcomes. By incorporating the d-adaptation technique into the optimization process, we have successfully enhanced the efficiency of the Adam optimizer for image classification tasks.

Our findings suggest that the integration of the d-adaptation technique allows the Adam optimizer to dynamically adjust its behavior based on the characteristics of the dataset and the optimization landscape. The integration of the d-adaptation technique leads to improved convergence and generalization performance, resulting in more accurate and reliable image classification results.

The accuracy of the optimizer with integration of d-adaptation technique has been relatively higher than a general Adam optimizer. As we generalise it, we can say that the integration of d-adaptation techniques improves the performance and the accuracy of the optimizers or models.

Overall, the utilization of the Adam optimizer with d-adaptation technique offers a promising approach for addressing image classification tasks. Further research and experimentation in this direction hold the potential to unlock even greater performance gains and advancements in the field of deep learning-based image classification.

Acknowledgement

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we now have the opportunity to express our gratitude for all of them. We owe our sincere gratitude to our project guide DR. KARTHICK S, Department of Computer Science and Engineering, National Institute of Technology Andhra Pradesh, who took keen interest and guided us all along, till the completion of our project work by providing all the necessary information. We avail ourselves of this proud privilege to express our gratitude to all the faculty of the department of Computer Science and Engineering at NIT Andhra Pradesh for emphasizing and providing us with all the necessary facilities throughout the work. We offer our sincere thanks to all our friends, fellow mates and other persons who knowingly or unknowingly helped us to complete this project.

References

[1] DeFazio, A., Mishchenko, K. (2023). *Learning-Rate-Free Learning by D-Adaptation*. *Machine Learning*; *Artificial Intelligence*; *Optimization and Control*; *Machine Learning*, revised 7 Jul 2023[v5]. Retrieved from https://arxiv.org/abs/2301.07733.

[2] Streeter, M., & McMahan, H. B. (2010). Less regret via online conditioning.

[3] DeFazio, A., & Gower, R. M. (2021). *The power of factorial powers: New parameter settings for* (stochastic) optimization. In *Proceedings of The 13th Asian Conference on Machine Learning*, volume 157 of *Proceedings of Machine Learning Research* (PMLR).

[4] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., & Girshick, R. (2019). Detectron2. Retrieved from https://github.com/facebookresearch/detectron2.

[5] DeFazio, A. (2020). Momentum via primal averaging: *Theoretical insights and learning rate schedules for non-convex optimization*.

[6] Wightman, R. (2019). Pytorch image models. Retrieved from https://github.com/rwightman/pytorch-image-models.

[7] Orabona, F., & Pál, D. (2021). Parameter-free stochastic optimization of variationally coherent functions.

[8] Detection and Classification of Multi-Magnetic Targets Using Mask-RCNN - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-Structure-diagram-of-Mask-RCNN-algorithm$_{f}ig1_{3}47267552[accessed4May,2024]$.

[9] Adam Algorithm Without D-Adaptation. Available from: https://github.com/theroyakash/Adam.

[10] Adam Algorithm With D-Adaptation. Available from: https://arxiv.org/abs/2301.07733.