INCREASING CNN REPRESENTATIONAL POWER USING ABSOLUTE COSINE VALUE REGULARIZATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

William S. Singleton

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2020

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Mohamed A. El-Sharkawy

Department of Electrical and Computer Engineering

Dr. Brian S. King

Department of Electrical and Computer Engineering

Dr. Dongsoo S. Kim

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian S. King

Head of Graduate Program

Dedicated to

My Parents: Daryl and Tammy Singleton

My Brother: Alexander

ACKNOWLEDGMENTS

I would like to begin by honoring my adviser, Dr. Mohamed El-Sharkawy, whose wisdom, and guidance, has been indispensable throughout my journey. I would also like to thank Dr. Brian King, and Dr. Dongsoo Kim, for serving on my graduation committee, and lending their expertise to my research. Additionally, I would like to thank Sherrie Tucker, who patiently assisted me throughout my Graduate studies.

I am grateful to have been a member of the IOT lab, and to have had the opportunity to learn from so many talented people. I would like to thank my friends, and lab members: Debjyoti Sinha, Niranjan Ravi, and Juan Chong, for their many insights. I would also like to have a special thanks for Dewant Katare, whose guidance led me to my Thesis topic. Finally, I would like to thank Dr. Zina Ben Miled, who inspired my research in the field of Artificial Intelligence.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

ABBREVIATIONS xi

ABSTRACT xii

1 BACKGROUND 1

1.1 What is The Convolutional Neural Network? 1

1.2 Why is The Convolutional Neural Network Important? 6

1.3 What is a Filter Weight Matrix Regularizer? 6

2 INTRODUCTION 9

2.1 Contribution 11

2.2 Related Work 11

2.3 Proposed Mathematical Description of ACVR 12

2.3.1 Calculating Regularizer's Contribution to Loss Function 13

2.3.2 Expansion of Partial Derivative for Back-Propagation 15

2.3.3 Discussion of Analytic Solution 16

3 PROPOSED ACVR ALGORITHM PROOF OF CONCEPT IN LOW-DIMENSIONAL CNN 17

3.1 Network Architecture and Training Conditions 17

3.2 Visualizing Effects of Regularization 18

3.2.1 Discussion of Experiment One for Proposed Algorithm 21

3.3 Long-Term Behavior of Regularization 21

3.3.1 Discussion of Experiment Two for Proposed Algorithm 25

3.4 Summary of Proposed ACVR Algorithm Proof of Concept in Low-Dimensional CNN 26

Page

4 HYPER-PARAMETER SEARCH AND MODEL CONFIGURATION CONSIDERATION 27

5 PROPOSED ACVR ALGORITHM EXPERIMENTS AND RESULTS FOR VGG-19 28

5.1 Search Process and Benchmarks 28

5.2 ACVR Layer Configuration Search 30

5.2.1 Refined ACVR Layer Configuration Search 31

5.2.2 ACVR and Benchmark Graphical Data 31

5.3 Discussion of ACVR Results for VGG-19 35

5.4 Proposed Dynamic-ACVR Algorithm 36

5.5 D-ACVR Layer Configuration Search 37

5.5.1 Refined D-ACVR Layer Configuration Search 38

5.5.2 D-ACVR Optimal Gamma Search 39

5.5.3 Comparison of Best D-ACVR and Benchmark Models 40

5.5.4 D-ACVR Graphical Data 40

5.6 Discussion of Proposed D-ACVR Algorithm Results for VGG-19 42

5.7 Summary of Proposed ACVR Algorithm Experiments and Results for VGG-19 43

6 PROPOSED ACVR AND D-ACVR ALGORITHM EXPERIMENTS AND RESULTS FOR MOBILENETV1 44

6.1 Search Process and Benchmarks 46

6.2 ACVR Layer Configuration Search 47

6.3 D-ACVR Layer Configuration Search 48

6.3.1 D-ACVR Optimal Gamma Search 49

6.4 D-ACVR and Benchmark Graphical Data 50

6.5 Discussion of Proposed ACVR and D-ACVR Algorithm Experiments for MobileNetv1 54

6.6 Summary of Proposed ACVR and D-ACVR Algorithm Experiments for MobileNetv1 55

7 CONCLUSION 57

Page

7.0.1 Future Directions 59
REFERENCES 60
VITA 62

LIST OF TABLES

Table		Page
3.1	Cosine Comparison of Layer One Filters before Training	18
3.2	Cosine Comparison of Layer Two Filters before Training	18
3.3	Cosine Comparison of Layer One Filters after Training without ACVR	19
3.4	Cosine Comparison of Layer Two Filters after Training without ACVR	19
3.5	Cosine Comparison of Layer One Filters after Training with ACVR	19
3.6	Cosine Comparison of Layer Two Filters after Training with ACVR	19
5.1	VGG-19 Benchmarks without ACVR	29
5.2	ACVR Layer Configuration Search with Fixed Batch Size and Gamma	30
5.3	Refined ACVR Layer Configuration Search with Fixed Batch Size and Gamma	31
5.4	D-ACVR Layer Configuration Search with Fixed Batch Size and Gamma	37
5.5	Refined D-ACVR Layer Configuration Search with Fixed Batch Size and Gamma	38
5.6	Gamma Value Search on Best D-ACVR Model	39
5.7	Accuracy Comparison on Benchmark Model and Best D-ACVR Model	40
6.1	Accuracy Comparison on Base Model	46
6.2	ACVR Layer Configuration Search with Fixed Batch Size and Gamma	47
6.3	D-ACVR Layer Configuration Search with Fixed Batch Size and Gamma	48
6.4	Gamma Value Search of Best D-ACVR Model	49

LIST OF FIGURES

Figure Page

1.1 Convolutional Layer with 2-D Filter 2

1.2 Different Types of Non-Linear Layers 3

1.3 Max-Pooling Layer 4

3.1 Convolutional Layer Two Filters before Training 20

3.2 Convolutional Layer Two Filters after Training with ACVR 20

3.3 Convolutional Layer One without ACVR 22

3.4 Convolutional Layer Two without ACVR 22

3.5 Loss and Accuracy Plots without ACVR 23

3.6 Convolutional Layer One with ACVR 23

3.7 Convolutional Layer Two with ACVR 24

3.8 Loss and Accuracy Plots with ACVR 24

5.1 List of VGG Architectures, Model E Is Chosen for Experiments 29

5.2 ACVR Raw Accuracy and Loss at Batch Size 32 and Configuration: Before Maxpool and 9-16 31

5.3 ACVR Convolutional Layers 2 and 4 at Batch Size 32 and Configuration: Before Maxpool and 9-16 32

5.4 ACVR Convolutional Layers 8 and 9 at Batch Size 32 and Configuration: Before Maxpool and 9-16 32

5.5 Raw Accuracy and Loss of Benchmark Model, Batch Size 64 33

5.6 Raw Accuracy and Loss of Benchmark Model, Batch Size 256 33

5.7 Benchmark Model Convolutional Layers 2 and 4, Batch Size 256 34

5.8 Benchmark Model Convolutional Layers 8 and 9, Batch Size 256 34

5.9 D-ACVR Raw Accuracy and Loss at Batch Size 32 and Configuration: Before Maxpool and 9-16 40

Figure Page

5.10 D-ACVR Convolutional Layers 2 and 4 at Batch Size 32 and Configuration: Before Maxpool and 9-16 41

5.11 D-ACVR Convolutional Layers 8 and 9 at Batch Size 32 and Configuration: Before Maxpool and 9-16 41

6.1 MobileNetv1 Architecture 45

6.2 Smooth Accuracy and Loss of Benchmark Model at Batch Size 32 50

6.3 ACV of Benchmark Pointwise Convolutional Layers 6 and 7 at Batch Size 32. 50

6.4 ACV of Benchmark Pointwise Convolutional Layers 8 and 9 at Batch Size 32. 51

6.5 ACV of Benchmark Pointwise Convolutional Layers 10 and 11 at Batch Size 32 51

6.6 Smooth Accuracy and Loss of D-ACVR Model with Pointwise Layers 6-11 Regularized at Batch Size 32 52

6.7 ACV of D-ACVR Model with Pointwise Layers 6-11 Regularized. Pointwise Convolutional Layers 6 and 7 are shown at Batch Size 32 52

6.8 ACV of D-ACVR Model with Pointwise Layers 6-11 Regularized. Pointwise Convolutional Layers 8 and 9 are shown at Batch Size 32 53

6.9 ACV of D-ACVR Model with Pointwise Layers 6-11 Regularized. Pointwise Convolutional Pointwise Layers 10 and 11 are shown at Batch Size 32 53

ABBREVIATIONS

CNN	Convolutional Neural Network
ACVR	Absolute Cosine Value Regularization
D-ACVR	Dynamic-Absolute Cosine Value Regularization
ACV	Absolute Cosine Value

ABSTRACT

Singleton, William S. M.S.E.C.E., Purdue University, May 2020. Increasing CNN Representational Power Using Absolute Cosine Value Regularization. Major Professor: Mohamed El-Sharkawy.

The Convolutional Neural Network (CNN) is a mathematical model designed to distill input information into a more useful representation. This distillation process removes information over time through a series of dimensionality reductions, which ultimately, grant the model the ability to resist noise, and generalize effectively. However, CNNs often contain elements that are ineffective at contributing towards useful representations. This Thesis aims at providing a remedy for this problem by introducing Absolute Cosine Value Regularization (ACVR). This is a regularization technique hypothesized to increase the representational power of CNNs by using a Gradient Descent Orthogonalization algorithm to force the vectors that constitute their filters at any given convolutional layer to occupy unique positions in R^n. This method should in theory, lead to a more effective balance between information loss and representational power, ultimately, increasing network performance. The following Thesis proposes and examines the mathematics and intuition behind ACVR, and goes on to propose Dynamic-ACVR (D-ACVR). This Thesis also proposes and examines the effects of ACVR on the filters of a low-dimensional CNN, as well as the effects of ACVR and D-ACVR on traditional Convolutional filters in VGG-19. Finally, this Thesis proposes and examines regularization of the Pointwise filters in MobileNetv1.

1. BACKGROUND

Before beginning this Thesis, a brief discussion of the fundamentals of Convolutional Neural Networks, and Filter Weight Matrix Regularization is in order. This discussion, while an inadequate treatment of a topic, which to date, still resides in a realm beyond the horizon of human knowledge, will provide the background, and continuity necessary, to confidently propound the Thesis statement offered by this work. This discussion will reveal itself in the form of a catechism, explicating the prerequisites, and providing a firm foundation, upon which, the merits of this work shall be made manifest.

1.1 What is The Convolutional Neural Network?

The Convolutional Neural Network is a mathematical model designed to take information as an input, process this information through a series of operations, and finally, provide an output. In this manner, the CNN can be seen simply as a function, taking an input, and returning an output. While this definition appears very abstract, this is intentional, as the CNN can take many different forms, and be useful for many different tasks. However, there are several features that unite all CNN architectures.

Firstly, the CNN must take an input. This can be simply a number, or something more complex, like an image. This input data is fed to the CNN, in the hope that the CNN will return a useful representation of the data. This can be as simple as a yes/no decision corresponding to a 1 or 0, or perhaps, in the case of the image input, a vector containing the locations and classifications of all interesting objects in the image. It is immediately apparent that if the CNN would be capable of performing these tasks,

it would be a tremendously powerful artifact. This artifact however, would only be useful if there was some way of easily constructing it, and more importantly, instructing it how to generate these outputs from input data.

Fortunately, both of these questions have solutions. To address the first, The CNN itself, is canonically comprised of a series of layers which perform specific operations. From our initial definition of the CNN, this can be understood as a series of nested functions: an input is passed to the first layer, which then processes it before passing it to the next. The final layer being an output of the original data distilled into a more useful representation. This leads us to ask exactly what the layers are, how they work, and how we decide how many, and in what order to place them.

Traditionally, the CNN will consist of the following layers: Input, Convolutional, Non-Linear, Pooling, Fully-Connected, and Output [1]. The Input and Output layers are simply the initial input data, and the output representation, respectively. The other layers form the core of most CNN architectures. These layers are the functions that receive, process, and pass forward data.

The first notable layer is the Convolutional Layer, whose operation is seen in Figure 1.1:

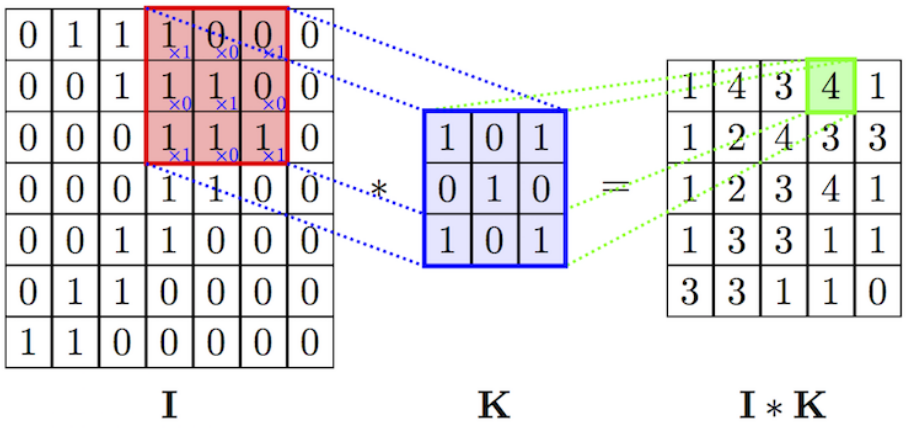


Fig. 1.1. Convolutional Layer with 2-D Filter

This layer contains a series of filters, which are arrays of numbers that will perform a Frobenius Inner Product with data received from the proceeding layer, and pass these values forward. Each filter in the Convolutional Layer will act on all of the data passed to it, producing an output that is a feature map. The collective filters, will each produce one feature map, and send all of these values to the next layer. in the case of Figure 1.1, we can see the Frobenius Inner Product of Filter K, with the Data I, of the same size. This is a singular example of the convolution process, which is repeated for every filter, on all Inner Product Locations in I.

After the Convolutional Layer Comes the Non-Linear Layer. This layer is designed to produce a non-linear mapping between its inputs and outputs. The reason for this, is that the process of convolution is a linear operation. By stacking only Convolutional Layers, there would be no benefit in terms of the models representational power. This process is remedied by Non-Linear Layers such as those seen below in Figure 1.2:

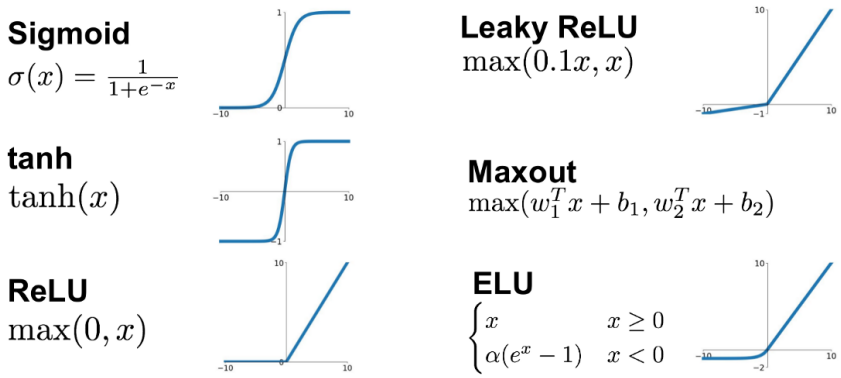


Fig. 1.2. Different Types of Non-Linear Layers

Each of these layers will receive data from the previous function and perform one of the operations shown in Figure 1.2. This process of introducing non-linearities to increase representational power is the primary agent in the success of the CNN.

The classical CNNs also consist of a Pooling Layer. This layer will reduce the dimensionality of the data it receives. The main purpose of this, is that the CNN will be able to distill the input information into the desired representation. The principal

theory of operation held by the CNN, is that by performing necessary operations on our data and procedurally reducing it over time, we are sacrificing data fidelity, for increased representational power. This is a very worthy and effective trade off. Shown below in Figure 1.3 is the most common type of Pooling Layer, known as Max-Pooling.

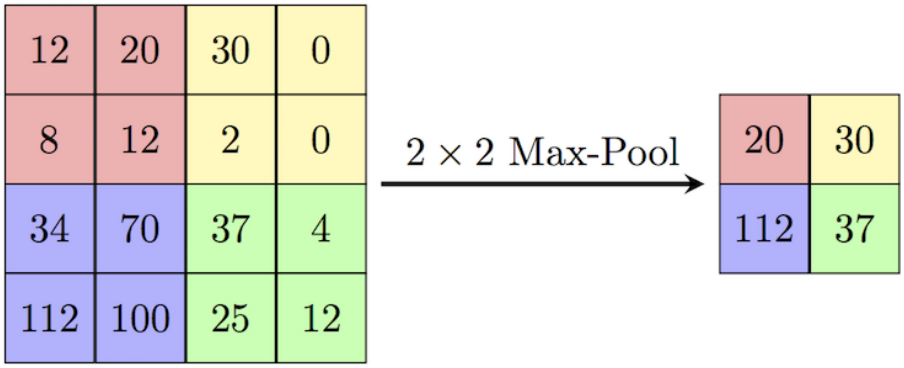


Fig. 1.3. Max-Pooling Layer

This layer operates by taking only the highest value that exists within a certain window, and passing that data forward to the following layers. This process is performed several times in a CNN architecture, to reduce data complexity, and yield increased generalization power.

The Final Layer is the Fully-Connected Layer. This layer takes all of the input data, flattens it into one vector, and performs a Dot Product with a Weight matrix. While there are many graphical representations of this process, they are not as adequate as the knowledge that this layer performs a simple dot product, and passes the resulting data forward.

All the previously mentioned layers are present in the classical CNN architectures. Unfortunately, the choice of how many of these layers to use is not straightforward. A popular strategy is to use many of these layers at first, and then slowly scale-back

until the network accuracy lowers. However, the choice of the number of layers, and the sizes of each layer, is generally an iterative, and heavily implementation dependent process.

The ordering of the layers is generally more straightforward. Classically, there is the input layer, followed by a series of Convolutional Layers interspersed with Pooling Layers. Non-Linear Layers always follow Convolutional layers, but never Pooling Layers. Finally, after the last Pooling Layer, there is a series of Fully-Connected Layers, each of which is followed by a Non-Linear Layer. And finally, there is the output layer. As previously mentioned, the exact number of each layer is implementation dependent, but the order is largely conserved.

Knowing the core elements of CNNs and their order, we are left with the question of instructing the Network to take input data and provide useful representations. This process is surprisingly simple given a well-constructed model, and training data that properly represents that which is attempting to be represented.

The Output Layer provides us with the final model representation. This representation can be compared against its target representation, and we can determine exactly how correct or incorrect the model is. The method of determining this, is the Loss Function. This function will return higher values for incorrect model representations, and lower values for correct representations. The Loss Functions are implementation dependent, but a basic Loss Function can be seen below in 1.1:

$$L(\hat{y}) = \sum_{(\hat{y},y)} loss(f(\hat{y}, W), y) \tag{1.1}$$

In this case, $\hat{y}$ is representative of the input data, W is representative of the network weights, and y is representative of the target representations. While the Loss Functions are implementation dependent, their use in the training and testing process is not. The Loss Function must has a derivative, so that Back-propagation can occur. Back-propagation is the process of recursively applying the chain rule to the

Loss Function, in order to determine what the rate of change of the Loss Function is with respect to different model layer elements. If we know exactly how much changing a given model element changes the Loss Function, we can update the weights of the model in order to reduce the error, and get more accurate predictions. This loss determination and updating process will occur until we are sufficiently satisfied with the models performance. After the model is satisfactory, we use the Output Layer representation by itself, without the addition of the Loss Function.

Given an overview of the CNN, its contents, and operation, we can now move forward to more important questions.

1.2 Why is The Convolutional Neural Network Important?

Simply, the Convolutional Neural Network is important because it is a powerful mathematical model that is capable of processing data and making decisions without the assistance of human beings. This makes it a prime candidate to automate work that is unfit, or impractical for human beings. These models potentially save money, time, and lives. The model is excellent at resisting aberrations in training data, and has excellent generalization power when faced with new data related to its training data. It is also capable of matching input data to target representations with little need for hand engineered features, as the model is capable of training itself given proper data and training conditions.

1.3 What is a Filter Weight Matrix Regularizer?

A Filter Weight Matrix Regularizer is a function that we add onto the Loss Function as a means of fine-tuning the training process. We realize that the weights of the Filters in the Convolutional Layers of a CNN are significant, as ultimately, they are the values that are used to produce the representations we desire. We have the intuition, that should the weights do something like grow to immense or minuscule values during the process of minimizing the Loss Function, this may in the long term,

cause the network to fail to reach a satisfactory level of accuracy and stability. With this knowledge, we can perform a balancing act between the loss generated from the training data, and that which is generated by the Filter Weight Matrix Regularizer. In other words, this is a balancing act between knowledge generated from data, and our preferences about how the weights should be updated over time. The Loss Function with the addition of a regularizer can be seen below in 1.2:

$$L(\hat{y}) = \sum_{(\hat{y},y)} loss(f(\hat{y}, W), y) + \gamma * \sum_{(l \in L)} R(W^l) \tag{1.2}$$

The left hand side of this equation is maintained from 1.1, with the addition of the regularizer term on the right hand side. In this term, R represents the Regularizer function, which is parameterized by the Weight Matrices present at that layer. This term is summed over the layers in which the Regularizer is added, and multiplied by the hyper-parameter gamma. This gamma value is the scalar that scales the addition of the Regularizer to the Loss Function. When attempting to minimize this Loss Function using Back-Propagation, the manner in which the network weights are updated will be reflective of both portions of the Loss Function.

Understanding what Filter Weight Matrix Regularization means to CNNs, we can discuss two common techniques for Filter Weight Matrix Regularization, being L_1 and L_2 Regularization [1]. The operation of both of these methods is nearly identical. In L_1 Regularization, the function R is a simple function that takes the weight matrices, and sums the absolute value of each of the weight elements. In L_2 Regularization, the function R takes the weight matrices, and sums the square of the individual weight elements. Both of these techniques add to the Loss Function a value that is reflective of the magnitudes of the weight matrices, meaning that Back-Propagation will be attempting to cause unnecessary weights to approach zero magnitude. This results in a network with fewer large weights, corresponding to a network that is intuitively less complex.

While L_1 and L_2 Regularization are common and effective in practice, there are many more ways to regularize filter weights, and this Thesis will go on to discuss Absolute Cosine Value Regularization.

2. INTRODUCTION

The principle of operation of the Convolutional Neural Network (CNN) is that it distills information through a series of geometric transformations to provide a more manageable representation of the input data. To the network architect, the primary concern for this final representation is whether or not it matches the target representation. In other words, the concern is how accurate the network's output is in reference to the desired output. This desire for greater accuracy has driven the search for CNN architectures that provide the greatest possible representation of the input data. However, given an effective CNN architecture, it is often the case that the architecture contains representational power that is misused or underutilized. This can be understood through the processes of pruning [2] and network quantization [3].Pruning is a method through which a Neural Network is reduced in size by selectively removing individual weights, filters, or even entire layers. Quantization on the other hand, is a process through which weights present in a network are given a reduction in precision, corresponding to a lower bit representation. In both cases, elements are entirely removed from the CNN. Rather unexpectedly however, the networks can retain their effectiveness, and in some cases, improve their ability to provide accurate outputs [4]. This process provides evidence that any CNN may have elements that are underutilized, or in extreme cases, entirely useless. While methods such as pruning and quantization appear to be effective in practice, they lead in the direction of a conclusion that is challenging and counter-intuitive. By removing elements from a network, representational power is lost. Given this understanding, there must surely be some method through which the network architect can wield this underutilized representational power without excising entire elements.

Attempting to understand this problem begins with the convolutional filters themselves. The hope for every filter at any given layer, is that it will stochastically learn weights that will propagate useful information throughout the remainder of the network. Since each filter acts on all of the input data from the preceding layer, the quality of propagated information is determined at each convolutional layer. With this in mind, the search for greater CNN representational power begins with one simple idea: if each filter in a layer learns a very similar combination of weights, the information passed to the following layers will be similar as well. However, if each filter learns a weight pattern that is significantly different, the following layers will receive a richer representation, and in turn, be able to produce their own. The goal then, is to learn filter weights that will preserve the greatest amount of information throughout the network, and allow the distillation process the greatest probability of obtaining useful information. This idea can not be taken too literally however, as learning only unique filter values removes any ability of the network to learn from its training data. The search for greater representational power is therefore, a search for an effective balance between filter diversity, and data generated knowledge.

The diversity of any two filters in this case, is given by their respective Absolute Cosine Value given in 2.1:

$$\left|Cos(\theta)\right| = \left|\frac{x \cdot y}{\|x\|\|y\|}\right| \tag{2.1}$$

This number is mathematically convenient to represent as an addition to the Loss Function, as well as intuitive in the fact that it is a representation of the similarity of the vector spaces of each filter. Understanding the problem of underutilized network elements, and the belief that novel filters provide richer representations, this Thesis aims at providing evidence that forcing CNN filters to learn weights to position themselves into unique spaces in R^n, may improve their ability to generalize well to new data, and increase their representational power.

This Thesis will be presented in four discrete stages. Principally, Absolute Cosine Value Regularization (ACVR) will be given a mathematical definition and implementation description. Secondly, ACVR will be tested on a low-dimensional CNN whose filters will be treated as vectors and visualized in R^3. Next, the traditional convolutional layers of the VGG-19 [5] architecture will be regularized, and the effects of AVCR and Dynamic-ACVR (D-ACVR) on the network will be examined. Finally, the Pointwise layers of MobileNetv1 [6] will be regularized, and the effects ACVR and D-ACVR on the network will be examined.

2.1 Contribution

This Thesis provides several unique contributions:

- Proposes implementation and mathematics of ACVR algorithm
- Proposes demonstration of proof of concept for ACVR algorithm in low-dimensional CNN
- Examines effects of ACVR algorithm on traditional convolutional filters in VGG-19
- Proposes and examines Dynamic-ACVR algorithm
- Proposes and examines use of ACVR and D-ACVR algorithms to regularize Pointwise filters in MobileNetv1

2.2 Related Work

A well-known method of retaining information throughout a CNN that inspires this work is ResNet [7]. Furthermore, using the Cosine Formula to improve Neural Network accuracies is not a recent paradigm. Reference [8] uses the cosine definition of the Dot Product as a means of propagating information throughout a network in a bounded manner. The concept of increasing representational power by enforcing

diversities in Support Vector Machines using the Cosine Formula was discussed in [9]. In addition, regularizing filter diversities for the purpose of increasing network accuracy in CNNs is discussed by [10]. However, this publication is chiefly concerned with constructing smaller architectures through pruning methods, and combined the activity of diversity regularization in an ensemble with other regularization terms. A rigorous explanation for the existence of the Regularizer, its proof of concept, operation on individual layers, and implementation, is not given.

2.3 Proposed Mathematical Description of ACVR

In order to understand the Absolute Cosine Value Regularization algorithm, it is necessary to define its implementation. Equation 2.2 represents the total network Loss Function with the addition of the regularization term ACV (Absolute Cosine Value), which is a function of the weights W, and summed over the layers in which it is added. This term is returning a value which corresponds to the similarity of the filter vector spaces at each layer, by minimizing it, the system will be attempting to generate high-diversity filter vectors. The scalar gamma, represents the hyper-parameter that adjusts the contribution of the Regularizer, a factor which scales the ratio of filter diversity to data generated knowledge.

$$L(\hat{y}) = \sum_{(\hat{y},y)} loss(f(\hat{y}, W), y) + \gamma * \sum_{(l \in L)} ACV(W^l) \tag{2.2}$$

This formula appears simple to implement, however care must be given to its implementation, as the Regularizer must compare all filters against one-another at each layer. This results in a N^2 matrix for every regularized layer. However, since this comparison matrix is symmetric, and the filters need not be compared against

themselves, the portion of the ACV matrix containing relevant information is upper-triangular. This leaves the number of ACV calculations done at each layer to be given by 2.3:

$$comparisons_l = \frac{N_l^2 - N_l}{2} \tag{2.3}$$

This formula states that this algorithm will become increasingly time consuming with large values of N, necessitating an efficient vectorized computation method.

2.3.1 Calculating Regularizer's Contribution to Loss Function

Each convolutional layer contains N convolutional filters parameterized by their height, width, and number of channels. This leaves a 4-Dimensional Tensor that can be unrolled for each filter into a 2-Dimensional tensor for efficient computation as follows in 2.4:

Unrolled 4-Dimensional Tensor $\equiv$ X, $each filter \equiv \mathrm{f}_i$

$$\frac{X \cdot X^T}{\|X\| \cdot \|X\|^T} \equiv \frac{\begin{bmatrix} - & f_1 & - \\ - & f_2 & - \\ - & \vdots & - \\ - & f_N & - \end{bmatrix} \begin{bmatrix} | & | & | & | \\ f_1 & f_2 & \cdots & f_N \\ | & | & | & | \end{bmatrix}}{\begin{bmatrix} \|f_1\| \\ \|f_2\| \\ \vdots \\ \|f_N\| \end{bmatrix} \begin{bmatrix} \|f_1\| & \|f_2\| & \cdots & \|f_N\| \end{bmatrix}} \tag{2.4}$$

$$\frac{\begin{bmatrix} f_1 f_1 & \cdots & f_1 f_N \\ \vdots & \ddots & \vdots \\ f_1 f_N & \cdots & f_N f_N \end{bmatrix}}{\begin{bmatrix} \|f_1\|\|f_1\| & \cdots & \|f_1\|\|f_N\| \\ \vdots & \ddots & \vdots \\ \|f_1\|\|f_N\| & \cdots & \|f_N\|\|f_N\| \end{bmatrix}} \tag{2.5}$$

$$\begin{bmatrix} \frac{f_1 f_1}{\|f_1\|\|f_1\|} & \cdots & \frac{f_1 f_1}{\|f_1\|\|f_N\|} \\ \vdots & \ddots & \vdots \\ \frac{f_1 f_N}{\|f_1\|\|f_N\|} & \cdots & \frac{f_N f_N}{\|f_N\|\|f_N\|} \end{bmatrix} \tag{2.6}$$

$$Cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|} \tag{2.7}$$

After element-wise division in 2.5 each cell in 2.6 is identical to the Cosine Formula 2.7, for respective vectors. It is important to note that the Cosine Formula for two identical vectors does not provide any information, as it always yields a value of one. This means all values along the main diagonal are unimportant. Values below the main diagonal are also unimportant, as they provide the same information as the values above the main diagonal. The only useful information in matrix 2.6 is contained above the main diagonal, as such, all values below are set to zero in 2.8:

$$\begin{bmatrix} 0 & \cdots & \frac{f_1 f_1}{\|f_1\|\|f_N\|} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \tag{2.8}$$

This leaves a final Tensor containing useful information which is summed and multiplied by the hyper parameter gamma as shown in 2.9:

$$ACV(W^l) = \gamma * \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} \left| \frac{f_j f_i}{\|f_j\| \|f_i\|} \right| \tag{2.9}$$

Equation 2.9 represents the loss returned for each regularized layer, and is a singular instance of the ACV function from 2.2.

2.3.2 Expansion of Partial Derivative for Back-Propagation

The following steps demonstrate the Calculus involved in determining a closed form solution to the partial derivative of the Loss Function with respect to a given weight in f_1, from the perspective of the ACV Regularizer:

$$ACV(W^l) = \gamma * \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} \sqrt{\left(\frac{f_j f_i}{\|f_j\| \|f_i\|} \right)^2} \tag{2.10}$$

Definition of Absolute Value

$$\frac{\partial ACV(W^l)}{\partial f_{1k}} = \gamma * \frac{\partial}{\partial f_{1k}} \sum_{i=2}^{N} \sqrt{\left(\frac{f_1 f_i}{\|f_1\| \|f_i\|} \right)^2} \tag{2.11}$$

Partial Derivative of Loss Function With Respect to k_{th} Element of Vector f_1

$$\frac{\partial ACV(W^l)}{\partial f_{1k}} = \gamma * \frac{\partial}{\partial f_{1k}} \sum_{i=2}^{N} \sqrt{\left(\frac{\sum f_1 f_i \sum f_1 f_i}{\sum f_1^2 \sum f_i^2} \right)} \tag{2.12}$$

Expanding Definition of ACV

$$\frac{\partial ACV(W^l)}{\partial f_{1k}} = \gamma * \sum_{i=2}^{N} \left[\frac{f_{ik} \sum f_1 f_i}{\sqrt{\left(\sum f_1 f_i \sum f_1 f_i \right)} \sqrt{\left(\sum f_1^2 \sum f_i^2 \right)}} - \frac{f_{1k} \sum f_i^2 \sqrt{\left(\sum f_1 f_i \sum f_1 f_i \right)}}{\sqrt{\left(\sum f_1^2 \sum f_i^2 \right)^3}} \right] \tag{2.13}$$

Final Analytic Expression

2.3.3 Discussion of Analytic Solution

Equation 2.13 is representative of the rate of change of the Loss Function from the perspective of the ACV Regularizer, with respect to the vector element f_{1k}. In Equation 2.13 any index of f_1 can be substituted for, and the complete contribution of the Regularizer to the Loss Function for f_1 can be found by solving 2.13, for all elements k. Equation 2.13 by itself is a large multi-variable expression, however its form yields insight into potential operation of the Regularizer. All values under radicals can not be negative, and neither can the square of f_i on the right hand side of 2.13. Only f_{1k}, f_{ik}, and $\sum f_1 f_i$ can have negative values. We have no insight into the sign of $f_{ik} \sum f_1 f_i$, however, the sign of f_{1k} is extremely significant. The sign of the right hand side of the equation is given solely by the sign of f_{1k}, and in particular, it is opposite to the sign of f_{1k}. This signifies that the right hand side of the equation is actively producing a value that is attempting to minimize the value of f_{1k}, and will make this value $N-1$ times for each instance in 2.13. The significance of this is that in addition to attempting to explicitly push all vectors into unique positions, the ACV Regularizer may be implicitly working to suppress their L_1 Magnitudes, further stabilizing the network.

3. PROPOSED ACVR ALGORITHM PROOF OF CONCEPT IN LOW-DIMENSIONAL CNN

The ACV Regularizer aims at forcing filter vectors into unique positions in R^n. The success of this objective can be easily visualized in R^2 or R^3. Therefore, in order to verify the effectiveness of this Regularizer, a trial CNN is constructed that excluding the fully connected base, consists of two convolutional layers. The purpose of this CNN is not to achieve competitive accuracies, but to demonstrate that by applying ACVR at high gamma values, the filter vectors within the network will spread apart from each other in a manner that is near maximal, a result of minimizing the Loss Function 2.2. In this case, the differences between vectors is given by 2.7. Two Experiments are conducted to examine the proposed algorithm: one with the purpose of visualizing the movements of the CNN's filter vectors with, and without regularization. The second, to determine the Regularizer's effect on the network when trained to validation set convergence. The filters of the network consist of only three elements and therefore exist in R^3, rendering visualization of the Regularizer's activity trivial.

3.1 Network Architecture and Training Conditions

The architecture of the network devised to test the ACV Regularizer consists of two convolutional layers, consisting of three and five filters respectively, and one fully-connected layer. The data set used is the CIFAR-10 small images data set [11], with a batch size of 32, and data set augmentation consisting of horizontal transfer and random width and height shifting. The model is constructed using the Keras API [12], and the Regularizer is developed from the TensorFlow API [13]. Post processing is done using SciPy [14] and MatLab [15].

3.2 Visualizing Effects of Regularization

To determine the effectiveness of the ACV Regularizer, the network is run for 8 epochs with, and without its presence. The gamma value for the Regularizer is found empirically, and is determined to be $5.0 * 10^{-1}$. From the network, the initial filter vectors are saved, along with the vectors from the network after training in both training instances. To determine vector similarity, the Cosine Formula 2.7 is employed, and six sets of similarity matrices are produced. The following six tables display the data provided by both sets of convolutional filters. In addition, two images are displayed to provide a simple graphical representation of the filter vectors of Convolutional Layer Two before, and after training with ACVR.

Table 3.1.
Cosine Comparison of Layer One Filters before Training

Filters	K2	K3
K1	0.136855	0.582584
K2	-	0.364344

Table 3.2.
Cosine Comparison of Layer Two Filters before Training

Filters	K2	K3	K4	K5
K1	0.749633	-0.524069	0.501112	-0.985155
K2	-	-0.951118	-0.191307	-0.8513
K3	-	-	0.445442	0.657606
K4	-	-	-	-0.345152

Table 3.3.
Cosine Comparison of Layer One Filters after Training without ACVR

Filters	K2	K3
K1	-0.866503	-0.635898
K2	-	0.724771

Table 3.4.
Cosine Comparison of Layer Two Filters after Training without ACVR

Filters	K2	K3	K4	K5
K1	-0.780891	-0.989761	-0.121355	0.789708
K2	-	0.743862	0.70981	-0.989937
K3	-	-	0.0573636	-0.772605
K4	-	-	-	-0.666673

Table 3.5.
Cosine Comparison of Layer One Filters after Training with ACVR

Filters	K2	K3
K1	-0.000126887	-3.77776e-05
K2	-	-0.00032148

Table 3.6.
Cosine Comparison of Layer Two Filters after Training with ACVR

Filters	K2	K3	K4	K5
K1	-0.000174307	0.000104767	-1.35988e-05	-1
K2	-	-1	-0.000178602	0.000214833
K3	-	-	0.000311102	-0.000145298
K4	-	-	-	-0.000135528

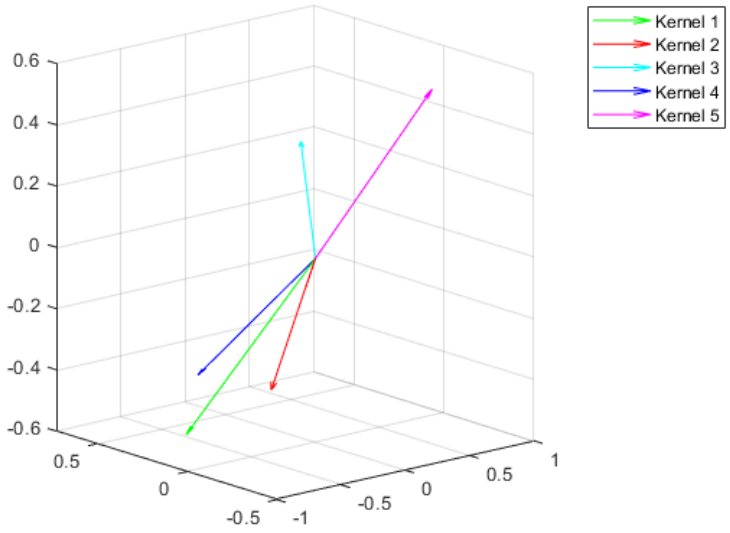


Fig. 3.1. Convolutional Layer Two Filters before Training

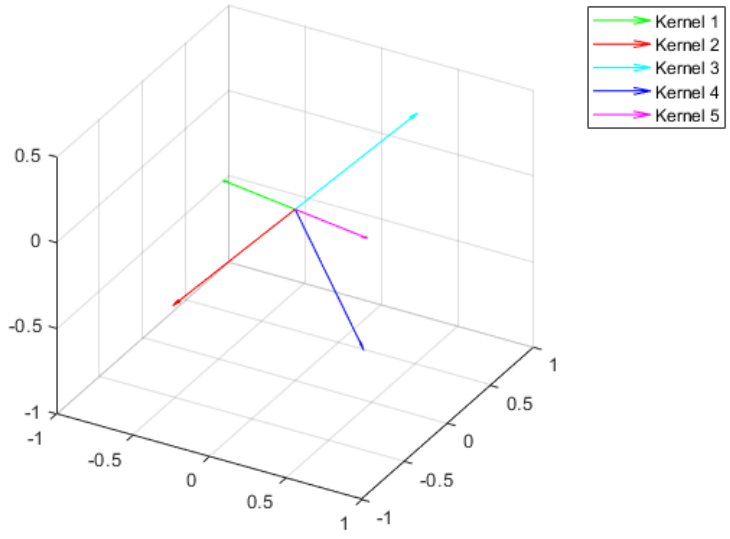


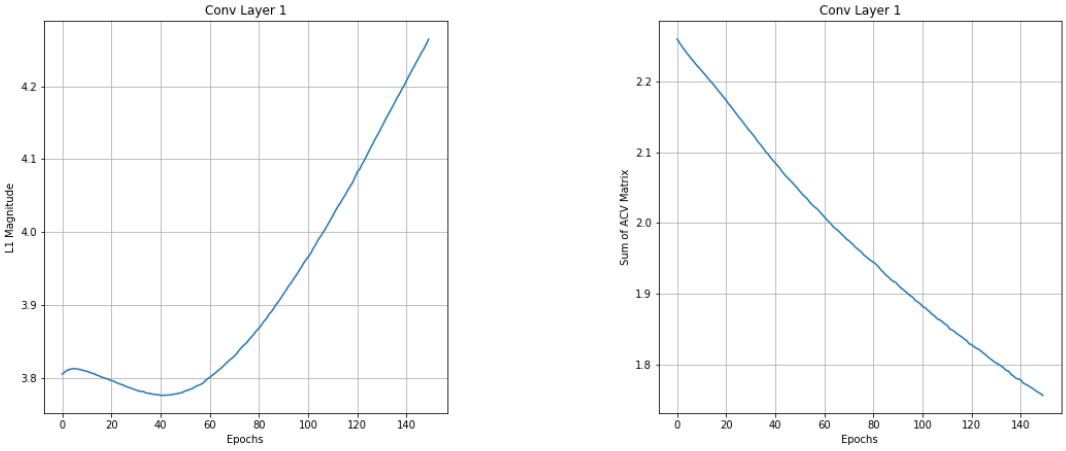
Fig. 3.2. Convolutional Layer Two Filters after Training with ACVR

3.2.1 Discussion of Experiment One for Proposed Algorithm

The goal of the first experiment is to demonstrate that when the activity of the Regularizer dominates the Loss Function, it is capable of spreading the filter vectors out in a manner which is near maximal as given by Equation 2.2. The matrix of cosine values is displayed to provide a numerical representation of the diversity among filter vectors. The filter vectors as they are initialized in Convolutional Layer Two can be seen in Figure 3.1, along with the cosine matrices of Convolutional Layers One and Two in Tables 3.1 and 3.2. In the case of the non-regularized model, the filter vectors numerically maintain a large degree of similarity after training as seen in Tables 3.3 and 3.4. In the case of the regularized model whose Layer Two Convolutional vectors after training can be seen in Figure 3.2, the filters of the convolutional layer visually spread apart in a near maximal manner, which is confirmed by the cosine value matrices in Tables 3.5 and 3.6. It is clear from the experiments that the ACV Regularizer is capable of providing immense vector diversity, whereas the non-regularized model does not. The behavior exhibited by the Regularizer is exactly that which was hoped, lending substantial evidence to its correct theory and implementation.

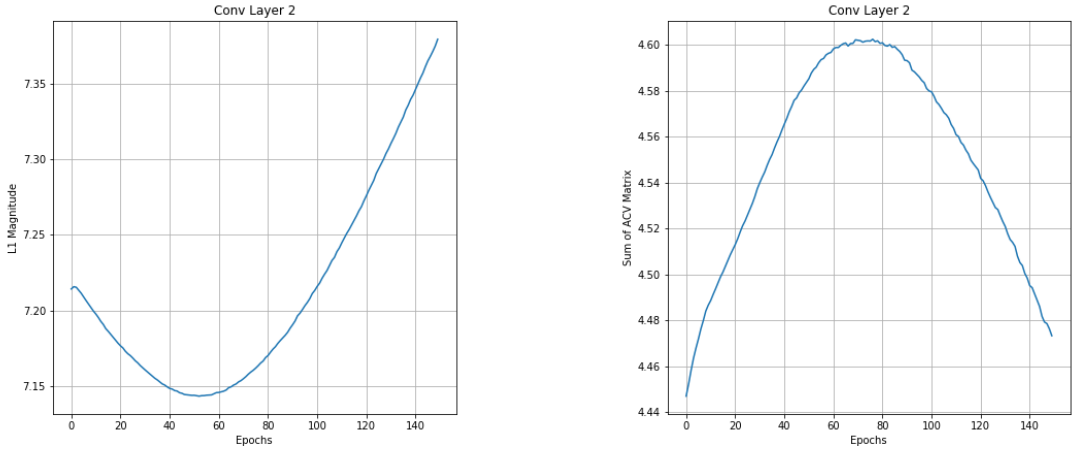
3.3 Long-Term Behavior of Regularization

The second experiment is run under the same conditions as the first, for 150 epochs, with the purpose of determining the long term behavior of the ACV Regularizer on the model. From these trials, the training and validation set accuracies and losses of both models are obtained, as well as the L_1 Magnitudes and net sum of the ACV values from each of the two convolutional layers.



(a) L_1 Magnitude (b) Sum of ACV Matrix

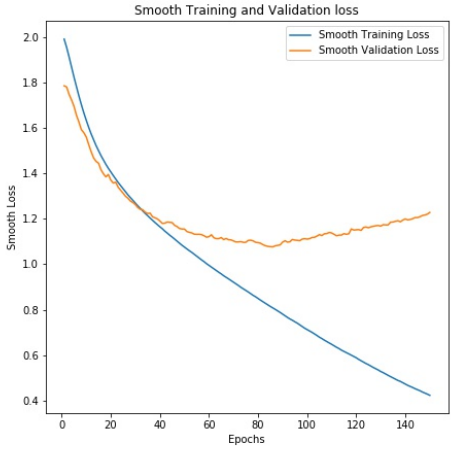
Fig. 3.3. Convolutional Layer One without ACVR



(a) L_1 Magnitude (b) Sum of ACV Matrix

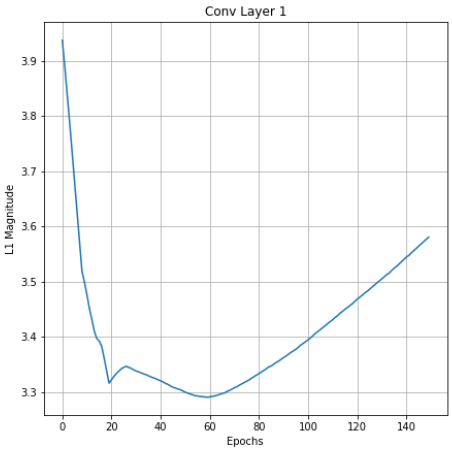
Fig. 3.4. Convolutional Layer Two without ACVR

(a) Validation Set Weighted Accuracy: 0.63%, Standard Deviation: 0.0087

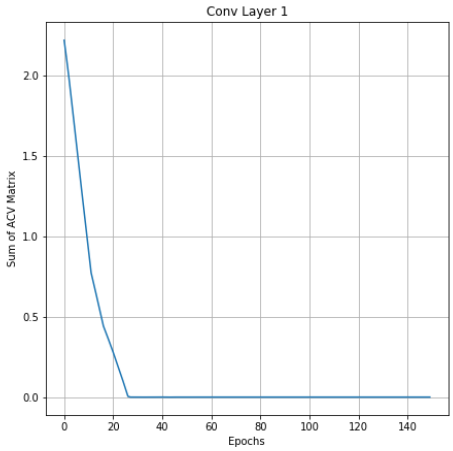


(b) Loss

Fig. 3.5. Loss and Accuracy Plots without ACVR

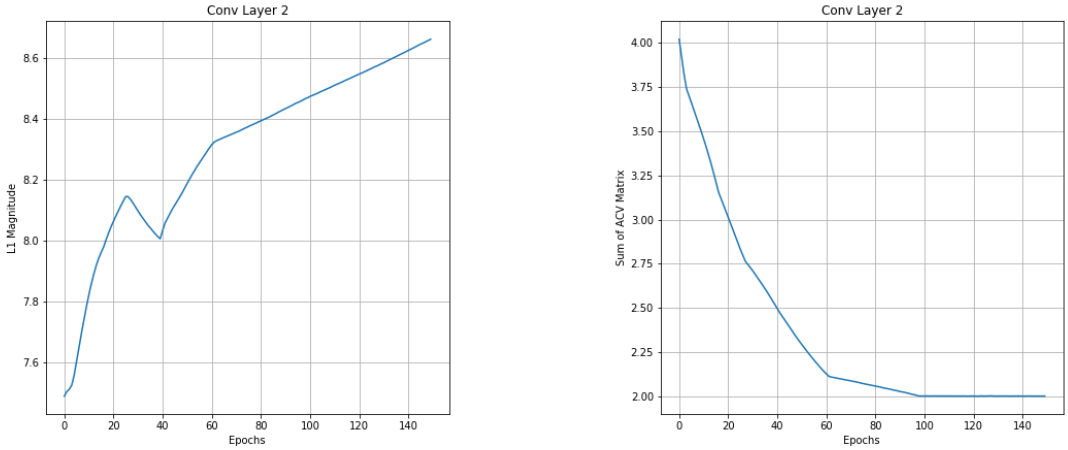


(a) L_1 Magnitude



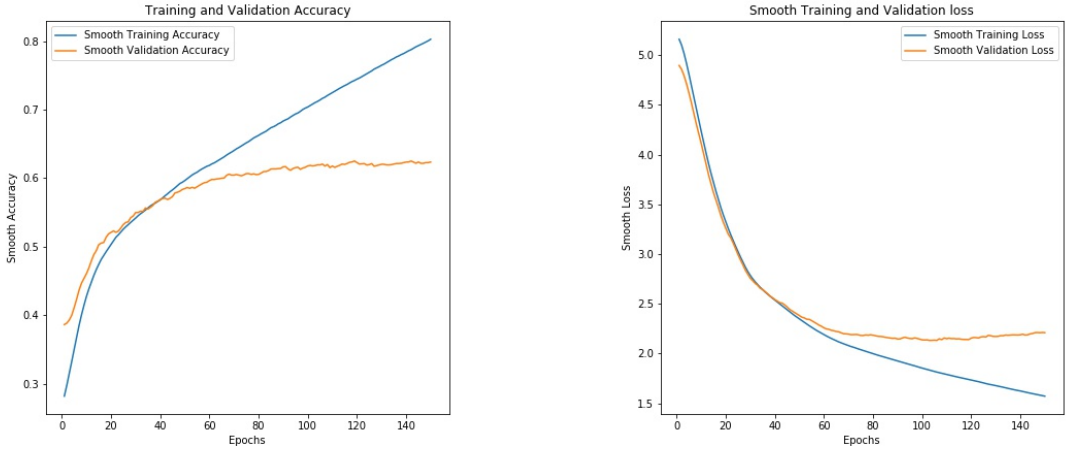
(b) Sum of ACV Matrix

Fig. 3.6. Convolutional Layer One with ACVR



(a) L_1 Magnitude (b) Sum of ACV Matrix

Fig. 3.7. Convolutional Layer Two with ACVR



(a) Validation Set Weighted Accuracy: 0.629%, Standard Deviation: 0.0093 (b) Loss

Fig. 3.8. Loss and Accuracy Plots with ACVR

3.3.1 Discussion of Experiment Two for Proposed Algorithm

The second experiment is designed to evaluate the behavior of the model when trained to validation set convergence. While the ACV Regularizer was hypothesized to provide an L_1 Regularization effect, the evidence provided from Figures 3.3(a), 3.4(a), 3.6(a), and 3.7(a), is inconclusive. This can be clarified from Equation 2.13: it is understood that the right ratio is attempting to provide an L_1 Regularization effect, but little can be predicted regarding the left ratio. Given this mathematical definition, and the inconclusive nature of the evidence provided, it can be said that ACVR can not be guaranteed to provide any L_1 Regularization effect. The graphs in Figures 3.3(b), 3.4(b), 3.6(b), and 3.7(b), demonstrating the sum of ACV values, align exactly with that which was determined from Experiment One, as the regularized model approaches the minimum possible ACV values given the number of filters in each layer. Whereas, the non-regularized model maintains a significantly higher degree of vector similarity. Both models display signs of diverging loss, as seen in Figure 3.5(b) and Figure 3.8(b). Finally, both models in Figure 3.5(a) and Figure 3.8(a) suffer from a high degree of training set overfitting, but the non-regularized model maintains a slightly higher validation set accuracy. This does not indicate that the Regularizer is ineffective, as its vectors have spread out in a near maximal manner, a configuration which is not guaranteed to be optimal. It is important to remember that the goal of these experiments is to prove the practicality of the concept of the Regularizer, and not determine the most efficient ratio between filter diversity and data generated knowledge.

3.4 Summary of Proposed ACVR Algorithm Proof of Concept in Low-Dimensional CNN

Absolute Cosine Value Regularization is a method designed to employ the under-utilized elements in a CNN, and promote optimal information distillation. So far in Chapter 2.3, this Thesis has discussed the mathematics behind this regularization technique, and described its practical implementation. Furthermore, in Chapter 3, the first experiment conducted on a low-dimensional CNN has provided evidence that this Regularizer is capable of generating high diversity filter vectors. The second experiment confirmed this result, but also produced evidence that the ability of ACV Regularization to act as an L_1 Regularizer is not guaranteed. The evidence gathered concerning filter diversity is promising, as it provides visual and numerical data demonstrating the effectiveness of the implementation, and reinforces the veracity of the theory. It is hypothesized that at optimum values of gamma, the inclusion of this Regularizer in a full-scale CNN with many layers, and filters per-layer, such as VGG-19, will increase its representational power. In addition, this Regularizer is hypothesized to be of even greater benefit to CNNs that are targeted towards mobile applications, such as MobileNet. Overall, the evidence gathered is an excellent proof of concept, and the effect of the Regularizer on a full-scale CNN will be examined in the following chapters.

4. HYPER-PARAMETER SEARCH AND MODEL CONFIGURATION CONSIDERATION

The Absolute Cosine Value Regularizer is a weight matrix regularizer. This means that it may be placed on any convolutional layer, or any combination of layers. This is significant in that it means the search for an optimum configuration is factorial in the number of convolutional layers in a network. For example, the VGG-19 architecture has 2.092279e+13 potential regularizer combinations, considering only the convolutional layers. This problem is compounded by the fact that the search for the optimum value of gamma exists in a continuous space. This signifies that finding the optimal regularizer configuration and gamma value is extremely implausible. This realization does not however, signify that finding a regularizer combination and gamma value that improves network performance is implausible. Therefore, the proposition of a search for an optimal ratio between filter diversity and data generated knowledge, is a proposition that is ill-posed given the constraints imposed by nature of the task. The correct proposition, is to examine the effects of ACV Regularization, and to demonstrate there exists a regularization combination and gamma value, that can be of benefit to a specific CNN architecture.

5. PROPOSED ACVR ALGORITHM EXPERIMENTS AND RESULTS FOR VGG-19

The VGG-19 architecture as seen in category E of Figure 5.1 [5], is a large, well-established model, that has been extensively referenced in literature. It is for these reasons that it is chosen as a means of testing ACVR for Image Classification. The data set used for testing is the CIFAR-10 small images data set, with data set augmentation consisting of horizontal transfer, and random width and height shifting. The optimizer used is RMSprop with a learning rate of 1e-4 and decay of 1e-6. The model is constructed using the Keras API, and the Regularizer is developed from the TensorFlow API. Post processing is done using SciPy and MatLab.

5.1 Search Process and Benchmarks

Given the considerations from Chapter 4, the process of finding effective regularizer configurations and gamma values is largely empirical, and one of many possibilities. To reduce the complexity of the search space, the process of finding an effective configuration begins with two strategies: test configurations that are inherently simple, such as those which are highly symmetric, and, regularize layers that have the largest number of filters in order to receive the greatest benefit of filter diversity. Upon discovering an effective regularizer configuration, a hyper-parameter search will be done to fine-tune the effectiveness of that configuration. This process while attempting to pursue only a subset of the regularizer configurations and gamma values, produces a substantial quantity of data, only some of which is necessary for reproduction here. Table 5.1 displays the benchmarked accuracies at 250 epochs of the VGG-19 architecture at different batch sizes, and the previously given training conditions:

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig. 5.1. List of VGG Architectures, Model E Is Chosen for Experiments

Table 5.1.
VGG-19 Benchmarks without ACVR

Batch Size	BASE
32	0.7188 ± 0.0141
64	0.8118 ± 0.0064
128	0.8439 ± 0.0061
256	0.8478 ± 0.0052

These benchmarks reveal that the accuracy value of 0.8478 is the target, and a regularizer configuration and gamma value must be found to surpass this metric.

5.2 ACVR Layer Configuration Search

Table 5.2 represents the configurations which are generated to test ACVR, and their associated accuracies at fixed hyper-parameter values. The batch size, gamma value, and number of epochs are fixed at 128, $3.0 * 10^{-1}$, and 100, respectively.

Table 5.2.
ACVR Layer Configuration Search with Fixed Batch Size and Gamma

Layer Configuration	Accuracy
1	0.8122 ± 0.0024
16	0.8283 ± 0.0088
Even Layers	0.8149 ± 0.0032
Odd Layers	0.8094 ± 0.0036
After Maxpool and Input	0.8046 ± 0.0056
Before Maxpool	0.8327 ± 0.0019
Before Maxpool And 9-12	0.8191 ± 0.0033
Before Maxpool And 9-16	0.8174 ± 0.0023
Before Maxpool And 13-16	0.8337 ± 0.0031
1-2	0.7532 ± 0.0209
3-4	0.7643 ± 0.0135
5-8	0.7302 ± 0.0335
9-12	0.8465 ± 0.0029
3-16	0.7544 ± 0.0025
5-16	0.7952 ± 0.0022
9-16	0.8428 ± 0.0043
13-16	0.8516 ± 0.0039
All	0.7188 ± 0.0020

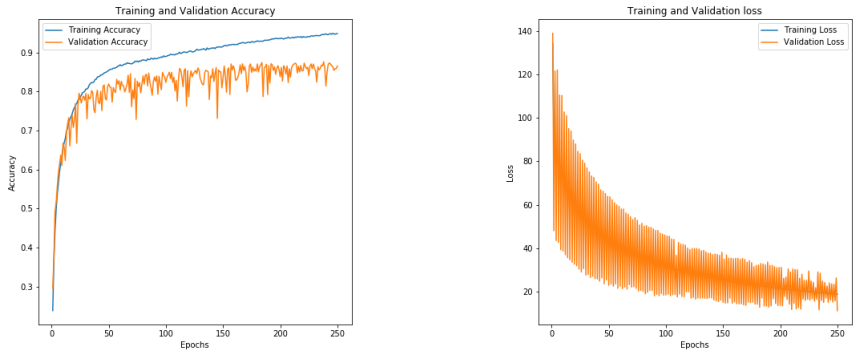
5.2.1 Refined ACVR Layer Configuration Search

Table 5.3 represents the configurations which are generated from knowledge acquired by the initial search to test ACVR, and their associated accuracies at fixed hyper-parameter values. The batch size, gamma value, and number of epochs are fixed at 32, $3.0 * 10^{-1}$, and 250, respectively.

Table 5.3.
Refined ACVR Layer Configuration Search with Fixed Batch Size and Gamma

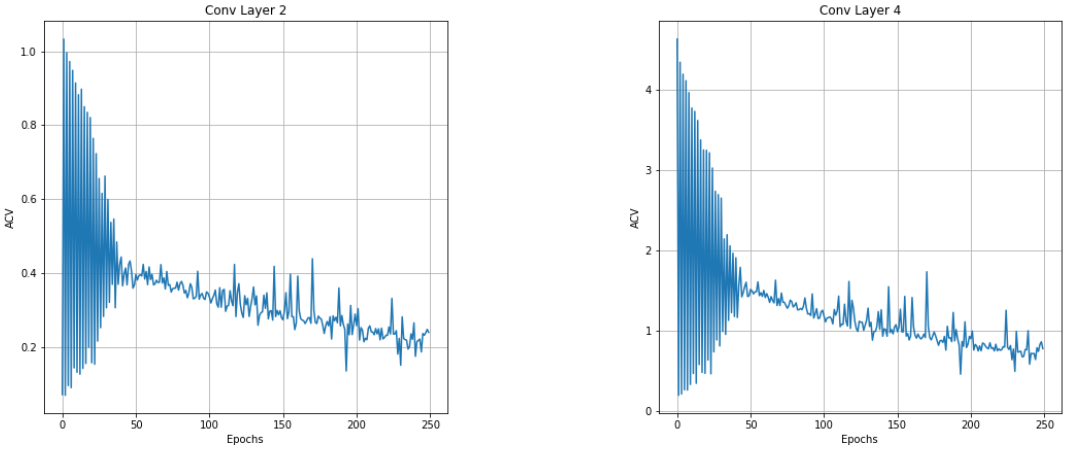
Layer Configuration	Accuracy
Before Maxpool	0.7594 ± 0.0152
Before Maxpool And 9-16	0.8554 ± 0.0055
Before Maxpool And 13-16	0.8275 ± 0.0052
9-12	0.6863 ± 0.0158
9-16	0.8504 ± 0.0084
13-16	0.7552 ± 0.0092

5.2.2 ACVR and Benchmark Graphical Data



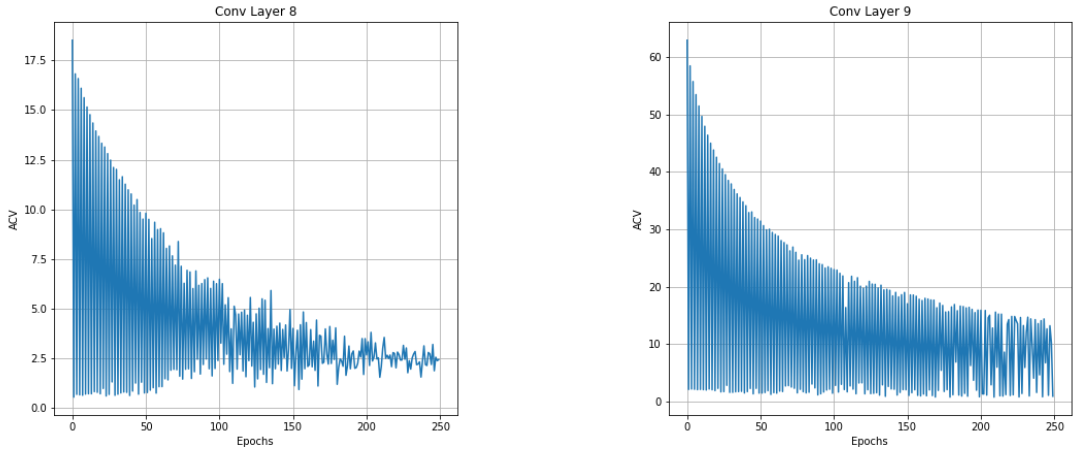
(a) Raw Accuracy (b) Raw Loss

Fig. 5.2. ACVR Raw Accuracy and Loss at Batch Size 32 and Configuration: Before Maxpool and 9-16



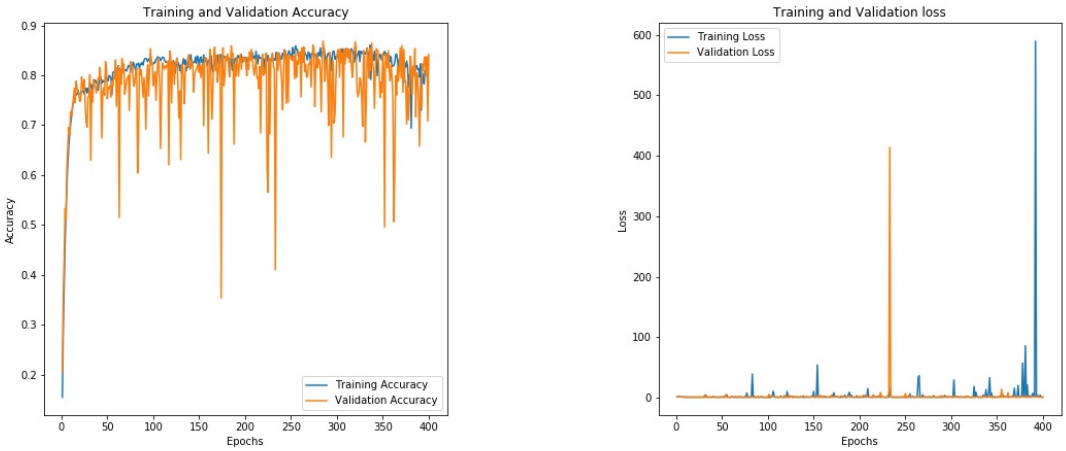
(a) Convolutional Layer 2 (b) Convolutional Layer 4

Fig. 5.3. ACVR Convolutional Layers 2 and 4 at Batch Size 32 and Configuration: Before Maxpool and 9-16



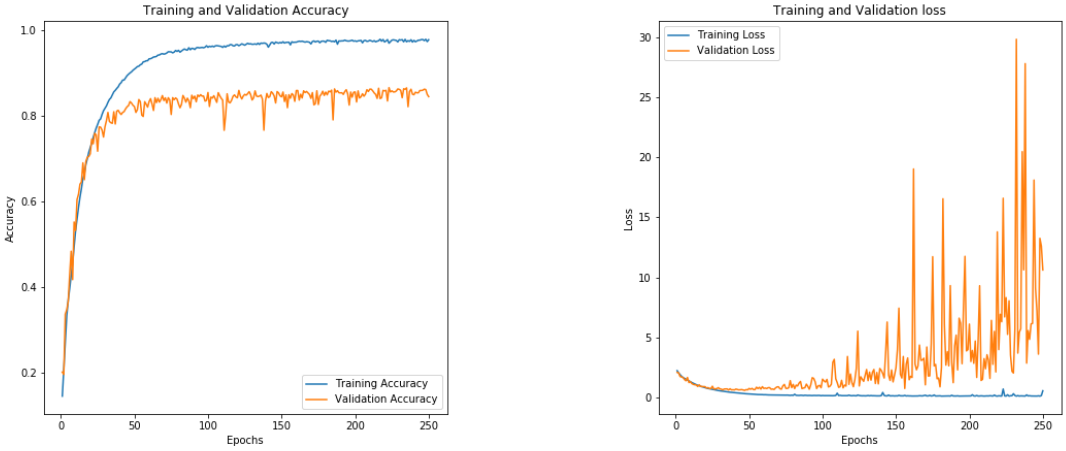
(a) Convolutional Layer 8 (b) Convolutional Layer 9

Fig. 5.4. ACVR Convolutional Layers 8 and 9 at Batch Size 32 and Configuration: Before Maxpool and 9-16



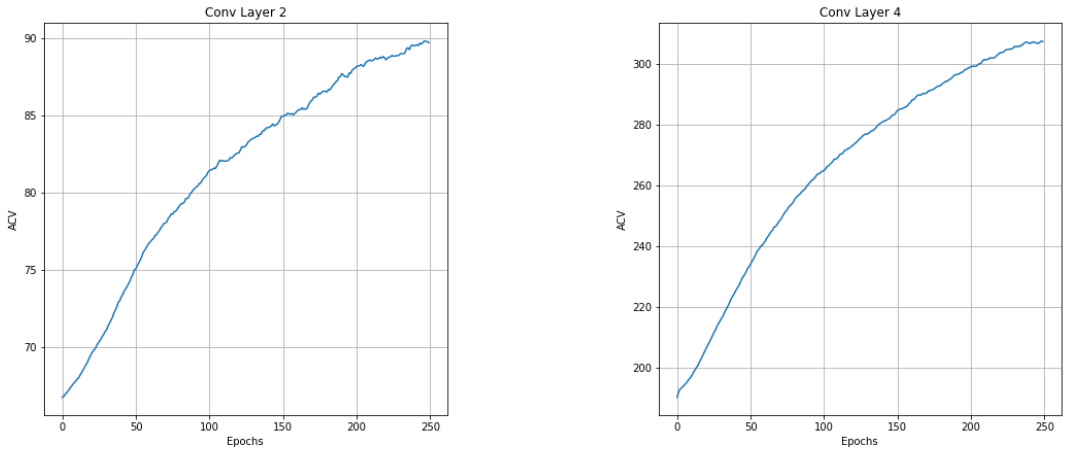
(a) Raw Accuracy (b) Raw Loss

Fig. 5.5. Raw Accuracy and Loss of Benchmark Model, Batch Size 64



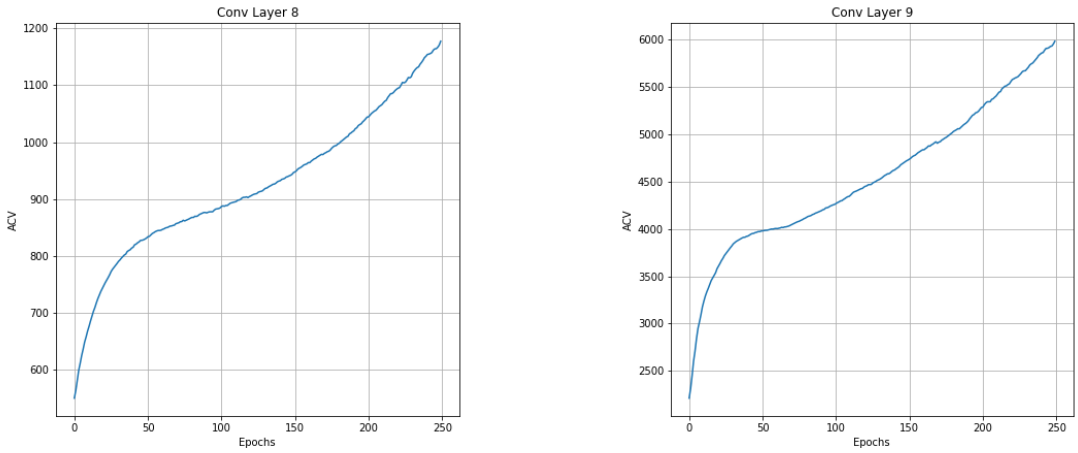
(a) Raw Accuracy (b) Raw Loss

Fig. 5.6. Raw Accuracy and Loss of Benchmark Model, Batch Size 256



(a) Convolutional Layer 2 (b) Convolutional Layer 4

Fig. 5.7. Benchmark Model Convolutional Layers 2 and 4, Batch Size 256



(a) Convolutional Layer 8 (b) Convolutional Layer 9

Fig. 5.8. Benchmark Model Convolutional Layers 8 and 9, Batch Size 256

5.3 Discussion of ACVR Results for VGG-19

The purpose of ACVR is to produce diverse filters at each convolutional layer in a CNN, a process which is hypothesized to increase the representational power of the network, ultimately increasing its accuracy. From the initial experiments conducted in Tables 5.2 and 5.3, this increase in representational power is achieved in three instances. Table 5.3 contains the highest accuracy achieved at 0.8554. While this does qualify as successful in that the baseline accuracy has been exceeded by 0.0076 (0.76%), this is a very limited demonstration of the benefits of ACVR. The raw accuracies and losses of the top scoring ACVR model can be seen in Figure 5.2. In addition, the raw accuracies and losses of the benchmark models at batch sizes of 64 and 256 can be seen from Figures 5.5 and 5.6 respectively. It is immediately apparent that the ACVR model produces values that contain significantly less variance than the benchmark model at batch size of 64, and is competitive with the benchmark model at batch size 256. This reduction in aberrations despite small batch size is a positive quality in a regularizer, and gives us more faith in the abilities of the model to properly classify images. Figures 5.3 and 5.4 display the ACV values for convolutional layers 2 and 4, and 8 and 9 respectively, for the top scoring ACVR model. Initially, these figures reinforce the knowledge that as training continues, filter diversity will continue to increase, ultimately producing high diversity filters. This can be contrasted with Figures 5.7 and 5.8, which demonstrate that the top scoring benchmark model has filters of lower diversity, and on each epoch the ACV values increase.

The fact that a competitive accuracy has been achieved, and high diversity filters are being generated is good news for the Regularizer. However, the advantage is only minor, and more consideration must be made to produce an even greater benefit. In order to receive this benefit, Figures 5.3 and 5.4 must be examined carefully. It is true that the purpose of ACVR is to produce high diversity filters, but this process is the mechanism of our intended success, and not the object. The initial intuition of the Regularizer is to create a means of optimal information distillation in a network.

Therefore, it is possible that by forcing the filters in layer 2, 4, and 8, to into positions of extreme diversity as shown, is reducing the models ability to learn from its training data, and information is being lost too early in the model. It also appears that all configurations that perform well do so by regularizing at least layers 13-16. Based on this data and intuition, the process of optimizing ACVR leads us in the direction of some system in which later layers receive a higher gamma coefficient, but earlier layers receive a lower gamma coefficient, creating a better ratio of diversity to data generated knowledge, and allowing for optimal information distillation.

5.4 Proposed Dynamic-ACVR Algorithm

The layer configuration and hyper-parameter search space is already immense. Optimizing ACVR by altering gamma values for each layer adds yet another layer of complexity to this issue, making an effective search nearly impractical. In order to work around this problem, a method is defined that is both simple, and retains all of the benefits previously known to ACVR. This method is Dynamic-ACVR (D-ACVR). This is an alteration to the previously defined regularizer architecture that is a simple function of the number of filters present in a given layer. Equation 2.3 yields the number of elements in the upper triangular portion of a matrix as a function of the diagonal, which in this case is the number of filters in a given layer. Knowing that Equation 2.1 can have a maximum value of one, the largest magnitude that can be returned by the ACV function from Equation 2.9 is equal to Equation 2.3 evaluated at the number of filters for that layer multiplied by gamma. Knowing that a gamma value of $3.0 * 10^{-1}$ has been shown to be effective in layers 13-16 which have 512 filters each, the gamma value at any regularized layer in the network is scaled by the ratio between Equation 2.3 evaluated using the current layer's number of filters, divided by 2.3 evaluated using 512 as the number of filters. This Equation can be seen as 5.1:

$$\gamma_l(N_l) = \gamma * \frac{N_l^2 - N_l}{N_{max}^2 - N_{max}} \tag{5.1}$$

Armed with this new Dynamic Regularization strategy, the previous experiments are run again to determine if a greater network accuracy can be found.

5.5 D-ACVR Layer Configuration Search

Table 5.4 represents the configurations which are generated to test D-ACVR, and their associated accuracies at fixed hyper-parameter values. The batch size, gamma value, and number of epochs are fixed at 128, $3.0 * 10^{-1}$, and 100, respectively.

Table 5.4.
D-ACVR Layer Configuration Search with Fixed Batch Size and Gamma

Layer Configuration	Accuracy
1	0.8329 ± 0.0081
Even Layers	0.8415 ± 0.0030
Odd Layers	0.8325 ± 0.0027
After Maxpool and Input	0.8323 ± 0.0055
Before Maxpool	0.8432 ± 0.0041
Before Maxpool And 9-12	0.8448 ± 0.0043
Before Maxpool And 9-16	0.8395 ± 0.0042
Before Maxpool And 13-16	0.8498 ± 0.0027
1-2	0.8292 ± 0.0087
3-4	0.8229 ± 0.0076
5-8	0.7952 ± 0.0216
3-16	0.8126 ± 0.0048
5-16	0.8189 ± 0.0024
All	0.8032 ± 0.0031

5.5.1 Refined D-ACVR Layer Configuration Search

Table 5.5 represents the configurations which are generated from knowledge acquired by the initial search to test D-ACVR, and their associated accuracies at fixed hyper-parameter values. The batch size, gamma value, and number of epochs are fixed at 32, $3.0 * 10^{-1}$, and 250, respectively.

Table 5.5.
Refined D-ACVR Layer Configuration Search with Fixed Batch Size and Gamma

Layer Configuration	Accuracy
Even Layers	0.8502 ± 0.0040
Before Maxpool	0.8201 ± 0.0149
Before Maxpool And 9-12	0.8488 ± 0.0040
Before Maxpool And 9-16	0.8790 ± 0.0024
Before Maxpool And 13-16	0.8524 ± 0.0050

5.5.2 D-ACVR Optimal Gamma Search

Table 5.6 represents search process for an optimal gamma value for Configuration: Before Maxpool and 9-16. The batch size, and number of epochs are fixed at 32, and 250, respectively.

Table 5.6.
Gamma Value Search on Best D-ACVR Model

Gamma	Accuracy
0.02	0.8577 ± 0.0060
0.04	0.8664 ± 0.0065
0.06	0.8753 ± 0.0044
0.08	0.8725 ± 0.0034
0.10	0.8653 ± 0.0053
0.12	0.8760 ± 0.0037
0.14	0.8711 ± 0.0050
0.16	0.8732 ± 0.0036
0.18	0.8768 ± 0.0020
0.20	0.8742 ± 0.0034
0.22	0.8724 ± 0.0032
0.24	0.8712 ± 0.0042
0.26	0.8681 ± 0.0053
0.28	0.8697 ± 0.0032
0.30	0.8790 ± 0.0024
0.32	0.8702 ± 0.0041
0.34	0.8730 ± 0.0041
0.36	0.8687 ± 0.0050
0.38	0.8667 ± 0.0053
0.4	0.8655 ± 0.0058

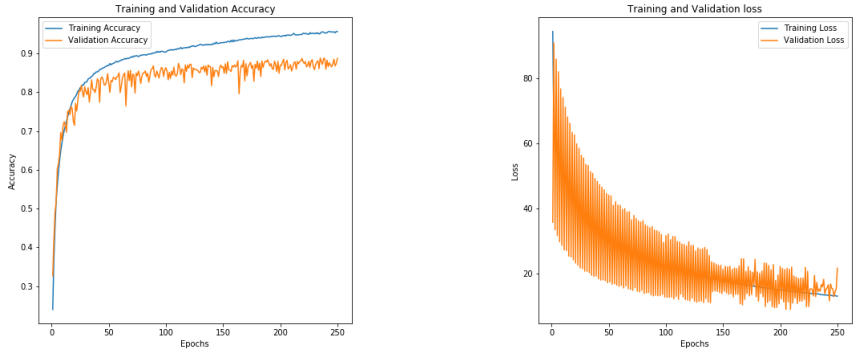
5.5.3 Comparison of Best D-ACVR and Benchmark Models

Table 5.7 represents the best D-ACVR configuration found through experimentation, contrasted with the benchmark models. The optimal value of gamma is found to be $3.0 * 10^{-1}$, and the most effective convolutional layers to regularize are layers 9 through 16, as well as 8, 4, and 2. Models have been run for 250 epochs.

Table 5.7.
Accuracy Comparison on Benchmark Model and Best D-ACVR Model

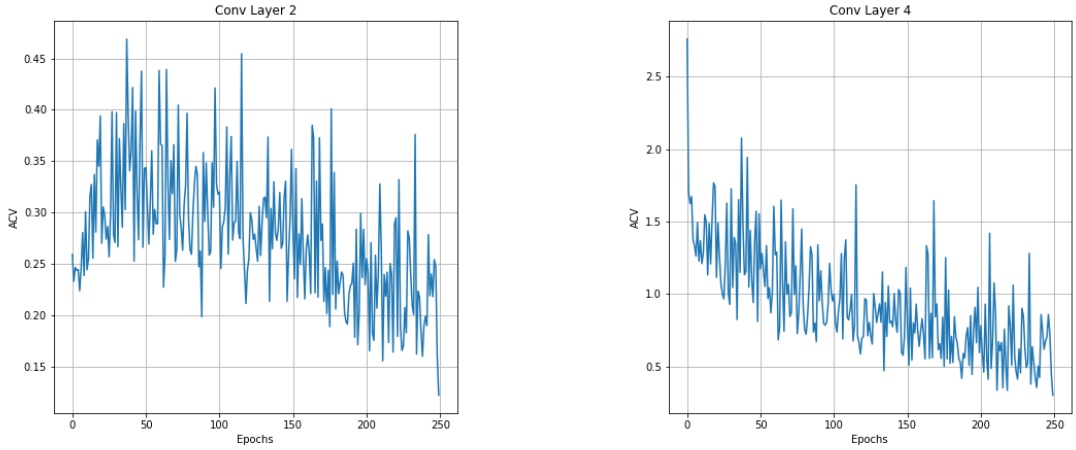
Batch Size	BASE	D-ACVR
32	0.7188 ± 0.0141	0.8790 ± 0.0024
64	0.8118 ± 0.0064	0.8642 ± 0.0022
128	0.8439 ± 0.0061	0.8679 ± 0.0037
256	0.8478 ± 0.0052	0.8605 ± 0.0026

5.5.4 D-ACVR Graphical Data



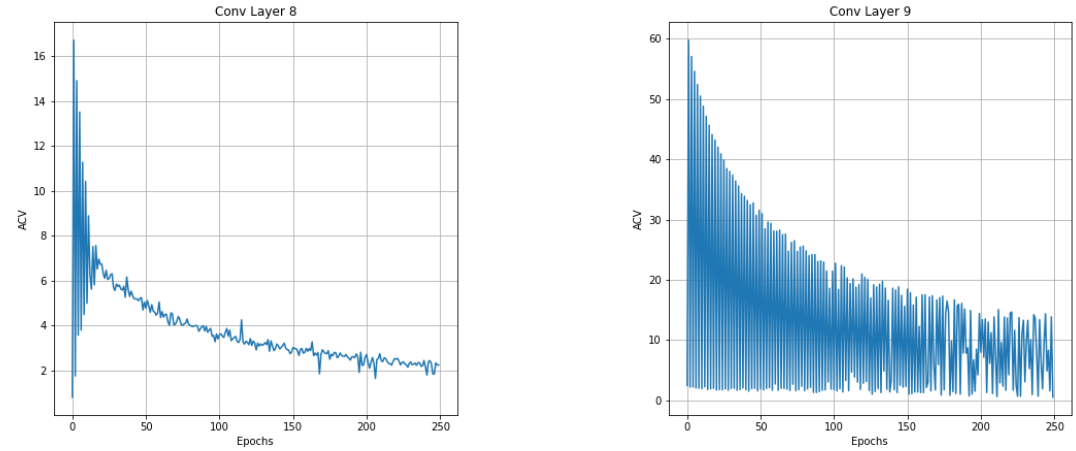
(a) Raw Accuracy (b) Raw Loss

Fig. 5.9. D-ACVR Raw Accuracy and Loss at Batch Size 32 and Configuration: Before Maxpool and 9-16



(a) Convolutional Layer 2 (b) Convolutional Layer 4

Fig. 5.10. D-ACVR Convolutional Layers 2 and 4 at Batch Size 32 and Configuration: Before Maxpool and 9-16



(a) Convolutional Layer 8 (b) Convolutional Layer 9

Fig. 5.11. D-ACVR Convolutional Layers 8 and 9 at Batch Size 32 and Configuration: Before Maxpool and 9-16

5.6 Discussion of Proposed D-ACVR Algorithm Results for VGG-19

It was hoped that D-ACVR would further increase the accuracies of the regularized models and outperform the baseline models by a margin which is less negligible. As visible in Tables 5.2 and 5.4, it is immediately clear that D-ACVR outperforms ACVR on all accounts on the initial layer configuration search. It is also clear that D-ACVR outperforms ACVR on all accounts on the Refined search process as visible in Tables 5.3 and 5.5. Ultimately, as visible in Table 5.7, the optimal configuration of D-ACVR found by regularizing all layers before Maxpooling in addition to layers 9-16, outperforms the most accurate benchmark model by a margin of 3.12%. This is not an aberrant increase in accuracy, as the gamma hyper-parameter search shown in Table 5.6 shows that this configuration at batch size of 32 produces values that are highly stable and nearly agnostic to the gamma value between a wide range. This value is therefore representative of extremely stable long term trends in the acquired data, and is significantly more trustworthy and indicative of the Regularizer's value and success than the 0.76% increase in accuracy due to ACVR. Figure 5.9 shows that D-ACVR also provides a more stable accuracy evaluation than ACVR, in addition to a higher convergent accuracy. Finally, one of the main inspirations for D-ACVR is to reduce the effects of ACVR on the earlier layers, and maximize it on the later layers. Figures 5.10 and 5.11 display the ACV values for the Best D-ACVR model found. It is clear that convolutional layers 2, 4, and 8, have ACV values that oscillate significantly less than their ACVR equivalents in Figures 5.3 and 5.4. Convolutional Layer 9 in both models features nearly identical behavior, which is expected as these layers have the same level of regularization. Despite this lack of oscillations in layers 2, 4, and 8, in the earlier epochs, the final ACV values of both models is largely identical. This realization defies the intended purpose of D-ACVR, but the increase in regularity of the ACV values may be the factor which is of benefit to the network.

Ultimately, D-ACVR is successful in that it has outperformed ACVR on all accounts, provided a more stable evaluation of the held out data-set, and demonstrated long term superiority over the benchmark models.

5.7 Summary of Proposed ACVR Algorithm Experiments and Results for VGG-19

Chapter 4 clarified that the purpose of the experiments in Chapter 5 is to examine the effects of ACVR, and to demonstrate that there exists a regularization combination and gamma value, that can be of benefit to the VGG-19 architecture. This process began by performing a layer configuration search with ACVR, a process which held the batch size and gamma value of the network constant and searched for the most effective combination of layers to regularize. After discovering an effective combination, a refined search was done to determine the configuration which was indeed, the best. The data generated by this search led to the need for a less brittle way to regularize the network and provide a means of optimal information distillation. This lead to the genesis of D-ACVR which dynamically regularized different layers based on the ratio of their ACV upper triangular matrix size to that of the largest ACV upper triangular matrix size present in the network. By performing the same layer configuration search with D-ACVR as done with ACVR, a layer configuration and gamma value was found that provided a 3.12% improvement over the highest scoring benchmark model. This data demonstrates that D-ACVR is capable of providing a stabilizing effect on the testing accuracies in a CNN, as well as a consistent improvement in overall network accuracy for image classification, lending credibility to its utility as a filter weight matrix regularizer.

6. PROPOSED ACVR AND D-ACVR ALGORITHM EXPERIMENTS AND RESULTS FOR MOBILENETV1

Hitherto, the effects of ACVR and D-ACVR have been examined on the traditional convolutional filters of a low-dimensional CNN, and the VGG-19 network. This is in contrast with the current chapter, which is focused on regularizing Pointwise filters. The intuition behind this idea is simple, Pointwise filters interact with the entire feature map tensor present in a given convolutional layer, just as traditional convolutional filters do. In addition, Pointwise filters operate exactly as traditional convolutional filters, except they have a width and height of one. Given this understanding of Pointwise convolution and the data generated from previous experiments, determining the effects of ACVR and D-ACVR on these Pointwise convolutional layers is a natural extension of the previous work.

Pointwise filters, which together with Depthwise filters, form the basis of the Depthwise-Seperable filters present in the well-established model MobileNetv1 [6] seen in Figure 6.1. Much like VGG-19, MobileNetv1 has been extensively referenced in literature, and more importantly, is mainly composed of Depthwise-Seperable convolutional layers. It is for these reasons that it is chosen as a means of testing ACVR and D-ACVR on Pointwise filters for Image Classification. The data set used for testing is the CIFAR-10 small images data set, with data set augmentation consisting of horizontal transfer, and random width and height shifting. The optimizer used is Adam, with a learning rate of 1e-3, and beta 1 and beta 2 value of 0.9, and 0.999, respectively. The MobileNetv1 architecture is tested at full width, with an alpha multiplier of one. The model is constructed using the Keras API, and the Regularizer is developed from the TensorFlow API. Post processing is done using SciPy and MatLab.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fig. 6.1. MobileNetv1 Architecture

6.1 Search Process and Benchmarks

Given the considerations from Chapter 4, the process of finding effective regularizer configurations and gamma values is largely empirical, and one of many possibilities. To reduce the complexity of the search space, the process of finding an effective configuration begins with two strategies: test configurations that are inherently simple, such as those which are highly symmetric, and, regularize layers that have the largest number of filters in order to receive the greatest benefit of filter diversity. Upon discovering an effective regularizer configuration, a hyper-parameter search will be done to fine-tune the effectiveness of that configuration. This process while attempting to pursue only a subset of the regularizer configurations and gamma values, produces a substantial quantity of data, only some of which is necessary for reproduction here. Table 6.1 displays the benchmarked accuracies at 220 epochs of the MobileNetV1 architecture at different batch sizes, and the previously given training conditions:

Table 6.1.
Accuracy Comparison on Base Model

Batch Size	BASE
32	0.8455 ± 0.0031
64	0.8370 ± 0.0026
128	0.8274 ± 0.0029
256	0.8091 ± 0.0044

These benchmarks reveal that the accuracy value of 0.8455 is the target, and a regularizer configuration and gamma value must be found to surpass this metric.

6.2 ACVR Layer Configuration Search

Table 6.2 represents the configurations which are generated to test ACVR, and their associated accuracies at fixed hyper-parameter values. The batch size, gamma value, and number of epochs are fixed at 32, $0.1 * 10^{-1}$, and 220, respectively.

Table 6.2.
ACVR Layer Configuration Search with Fixed Batch Size and Gamma

Layer Configuration	Accuracy
All	0.7877 ± 0.0027
Conv2D	0.8420 ± 0.0026
Pointwise 1	0.8373 ± 0.0035
Pointwise 13	0.8266 ± 0.0030
Pointwise 2	0.8359 ± 0.0027
Pointwise 2-3	0.8234 ± 0.0032
Pointwise 4-5	0.8358 ± 0.0035
Pointwise 6-11	0.8482 ± 0.0048
Pointwise 12-13	0.8396 ± 0.0029
Pointwise 2-13	0.8071 ± 0.0029
Pointwise 4-13	0.8330 ± 0.0035
Pointwise 6-13	0.8497 ± 0.0022
Even Pointwise Layers	0.8274 ± 0.0028
Odd Pointwise Layers	0.8273 ± 0.0027
Before Stride 2	0.8190 ± 0.0031
After Stride 2	0.8274 ± 0.0032
Before Stride 2 and 6-13	0.8216 ± 0.0026
After Stride 2 and 6-13	0.8328 ± 0.0031
Before Stride 2 and 12-13	0.8191 ± 0.0032
After Stride 2 and 12-13	0.8228 ± 0.0031

6.3 D-ACVR Layer Configuration Search

Table 6.3 represents the configurations which are generated to test D-ACVR, and their associated accuracies at fixed hyper-parameter values. The batch size, gamma value, and number of epochs are fixed at 32, $0.1 * 10^{-1}$, and 220, respectively.

Table 6.3.
D-ACVR Layer Configuration Search with Fixed Batch Size and Gamma

Layer Configuration	Accuracy
All	0.8124 ± 0.0036
Conv2D	0.8446 ± 0.0028
Pointwise 1	0.8361 ± 0.0026
Pointwise 13	0.8361 ± 0.0029
Pointwise 2	0.8427 ± 0.0026
Pointwise 2-3	0.8298 ± 0.0024
Pointwise 4-5	0.8371 ± 0.0030
Pointwise 6-11	0.8554 ± 0.0024
Pointwise 12-13	0.8313 ± 0.0035
Pointwise 2-13	0.8191 ± 0.0024
Pointwise 4-13	0.8400 ± 0.0029
Pointwise 6-13	0.8460 ± 0.0027
Even Pointwise Layers	0.8312 ± 0.0025
Odd Pointwise Layers	0.8347 ± 0.0035
Before Stride 2	0.8320 ± 0.0030
After Stride 2	0.8391 ± 0.0027
Before Stride 2 and 6-13	0.8312 ± 0.0026
After Stride 2 and 6-13	0.8371 ± 0.0024
Before Stride 2 and 12-13	0.8294 ± 0.0035
After Stride 2 and 12-13	0.8238 ± 0.0040

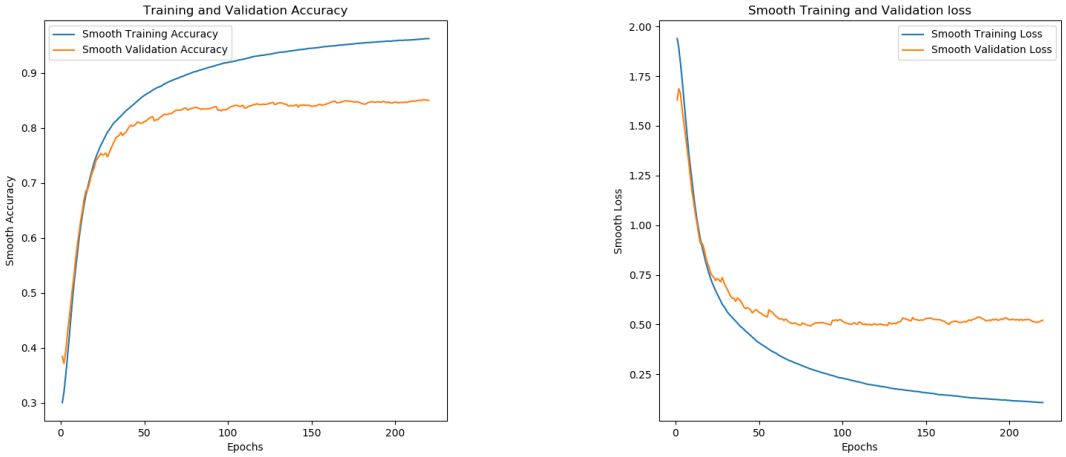
6.3.1 D-ACVR Optimal Gamma Search

Table 6.4 represents the configuration which regularizes Pointwise layers 6-11 at different gamma values. This configuration achieved the maximum accuracy during the layer configuration search. The batch size, and number of epochs are fixed at 32, and 220, respectively.

Table 6.4.
Gamma Value Search of Best D-ACVR Model

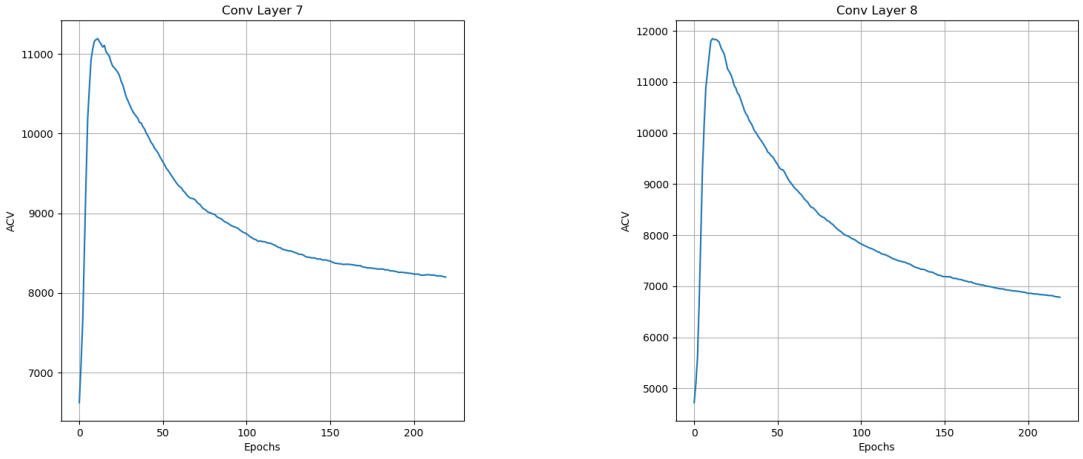
Gamma	Accuracy
0.02	0.8488 ± 0.0023
0.04	0.8499 ± 0.0026
0.06	0.8495 ± 0.0020
0.08	0.8471 ± 0.0061
0.10	0.8554 ± 0.0024
0.12	0.8523 ± 0.0032
0.14	0.8504 ± 0.0031
0.16	0.8491 ± 0.0036
0.18	0.8480 ± 0.0037
0.20	0.8462 ± 0.0025

6.4 D-ACVR and Benchmark Graphical Data



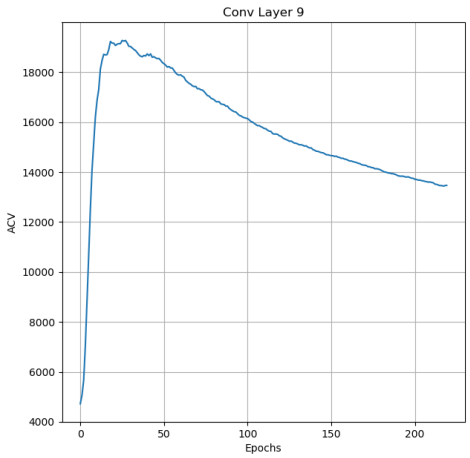
(a) Accuracies (b) Loss

Fig. 6.2. Smooth Accuracy and Loss of Benchmark Model at Batch Size 32

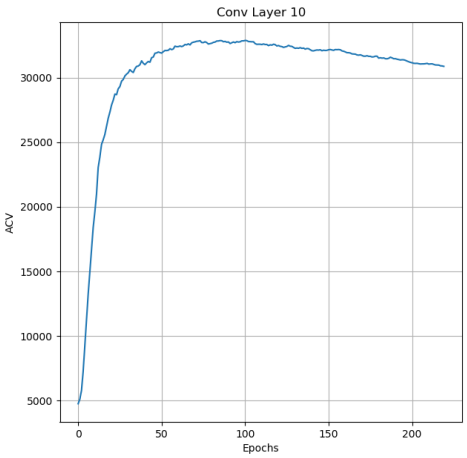


(a) Pointwise Convolutional Layer 6 (b) Pointwise Convolutional Layer 7

Fig. 6.3. ACV of Benchmark Pointwise Convolutional Layers 6 and 7 at Batch Size 32.

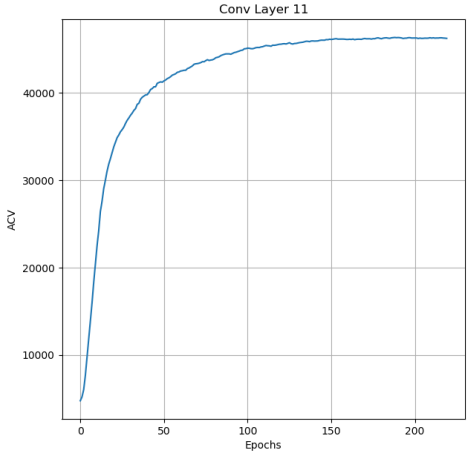


(a) Pointwise Convolutional Layer 8

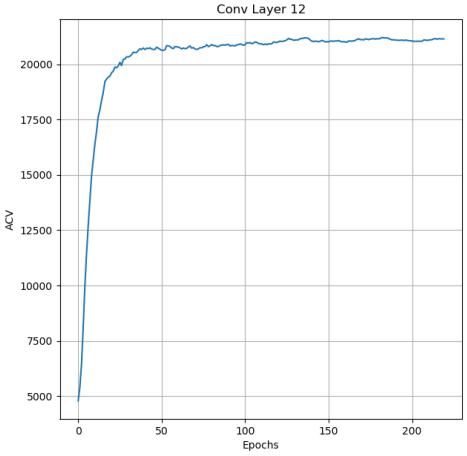


(b) Pointwise Convolutional Layer 9

Fig. 6.4. ACV of Benchmark Pointwise Convolutional Layers 8 and 9 at Batch Size 32.

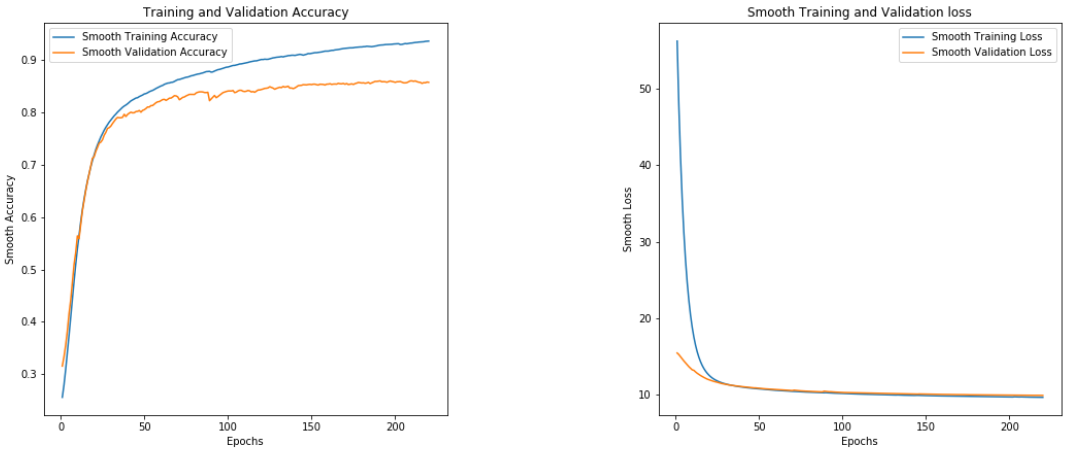


(a) Pointwise Convolutional Layer 10



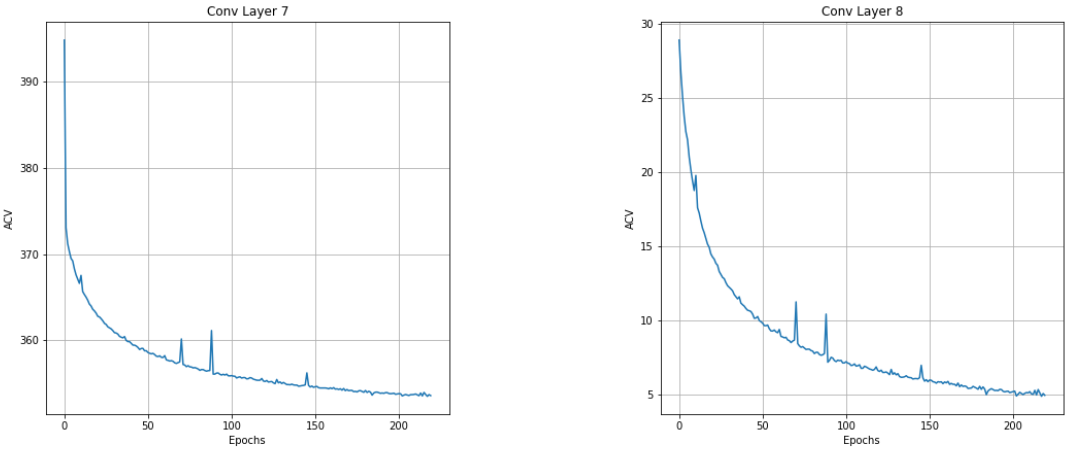
(b) Pointwise Convolutional Layer 11

Fig. 6.5. ACV of Benchmark Pointwise Convolutional Layers 10 and 11 at Batch Size 32



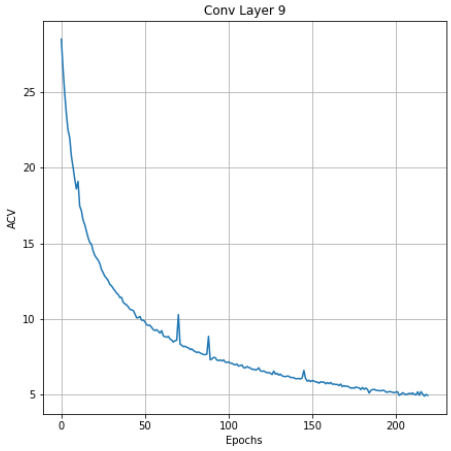
(a) Accuracies (b) Loss

Fig. 6.6. Smooth Accuracy and Loss of D-ACVR Model with Pointwise Layers 6-11 Regularized at Batch Size 32

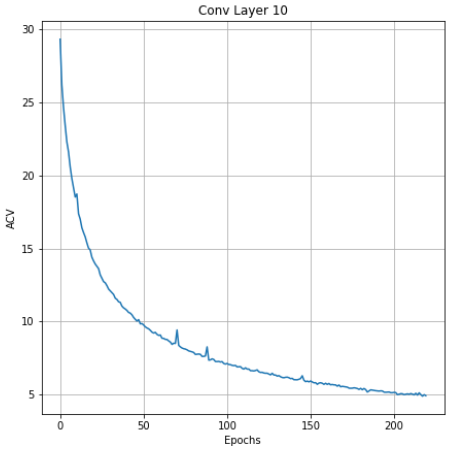


(a) Pointwise Convolutional Layer 6 (b) Pointwise Convolutional Layer 7

Fig. 6.7. ACV of D-ACVR Model with Pointwise Layers 6-11 Regularized. Pointwise Convolutional Layers 6 and 7 are shown at Batch Size 32

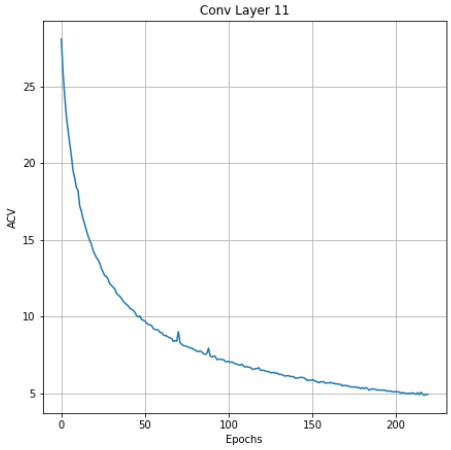


(a) Pointwise Convolutional Layer 8

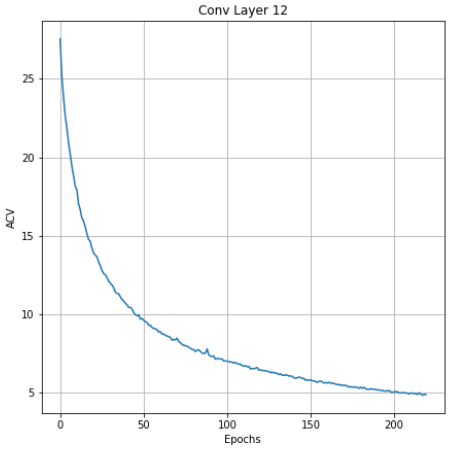


(b) Pointwise Convolutional Layer 9

Fig. 6.8. ACV of D-ACVR Model with Pointwise Layers 6-11 Regularized. Pointwise Convolutional Layers 8 and 9 are shown at Batch Size 32



(a) Pointwise Convolutional Layer 10



(b) Pointwise Convolutional Layer 11

Fig. 6.9. ACV of D-ACVR Model with Pointwise Layers 6-11 Regularized. Pointwise Convolutional Pointwise Layers 10 and 11 are shown at Batch Size 32

6.5 Discussion of Proposed ACVR and D-ACVR Algorithm Experiments for MobileNetv1

This chapter is designed to test the effects of ACVR and D-ACVR on the Pointwise convolutional layers of the MobileNetv1 architecture. Originally, it was determined that the benchmark accuracy to beat from Table 6.1 is 0.8455. Ultimately, the highest accuracy value found from the D-ACVR and ACVR networks from Tables 6.2 and 6.3 is 0.8554. This corresponds to a 0.99% increase in convergent accuracy. This maximum convergent accuracy is reconfirmed by Table 6.4, which demonstrates that the D-ACVR regularizer configuration 6-11 is capable of surpassing the Benchmark model at ten separate gamma values. Given the definition of the problem from Chapter 4, this constitutes as a success, as the Regularizer provides the network with a very limited, but still consistent and detectable increase in accuracy.

The Figures 6.2 and 6.6 display the exponentially averaged accuracies and losses of the highest scoring benchmark model and the highest scoring D-ACVR model. The D-ACVR model has a higher loss value, as well as convergent accuracy, as expected. In addition, looking at Figures 6.3 - 6.5, and 6.7 - 6.9, the ACV graphs of the benchmark models contain values that are significantly higher than those of the D-ACVR model. The Benchmark model has ACV values that either strictly increase or converge to large values, while the D-ACVR model features a smooth decrease in ACV over time, which once again is as expected.

Given this success, there are still several points that need to be discussed. Initially, much like with the VGG-19 model, D-ACVR outperformed ACVR on MobileNetv1. However, D-ACVR does not outperform ACVR in all instances. More importantly, both ACVR and D-ACVR reach their maximum accuracies, exceeding that of the baseline model, by regularizing Pointwise layers 6-11 and 6-13. This is troubling for D-ACVR, as it achieves its maximum accuracy by regularizing Pointwise layers 6-11, a series which all contain the same number of Pointwise filters. In this configuration, D-ACVR is simply ACVR with a reduced value of gamma, ultimately

defying its intended purpose. Despite this realization, D-ACVR has still managed to outperform the baseline architecture. Combining the success of Regularization on the MobileNetv1 architecture, with that of the VGG-19, D-ACVR once again, has demonstrated its potential value as a weight matrix regularizer.

6.6 Summary of Proposed ACVR and D-ACVR Algorithm Experiments for MobileNetv1

Chapter 4 clarified that the purpose of the experiments in Chapter 6 is to examine the effects of ACVR, and to demonstrate that there exists a regularization combination and gamma value, that can be of benefit to the MobileNetv1 architecture. This process began by performing a layer configuration search with ACVR, a process which held the batch size and gamma value of the network constant and searched for the most effective combination of layers to regularize. This process was repeated for D-ACVR. Ultimately, several layer configurations were found to surpass the benchmark metric of 0.8455. However, the D-ACVR configuration of 6-11 at batch size 32, and gamma value of $0.1 * 10^{-1}$, was found to be the best configuration discovered. A gamma value search was then done at this configuration by holding the batch size and and number of epochs constant and constantly changing the value of gamma within a small window. This search ultimately demonstrated that at gamma value $0.1 * 10^{-1}$, the D-ACVR configuration was able to outperform the benchmark accuracy by 0.99% , with a total convergent accuracy of 0.8554. By viewing the graphical data produced by this automated search process, the results are exactly those which were desired, as the D-ACVR model converged to a higher accuracy, as well as continuously decreased the ACV of all regularized layers. This is in contrast with the benchmark model, which converged to a lower accuracy, and had significantly larger ACV throughout training. Overall, while the increase in accuracy is somewhat limited, this data demonstrates that D-ACVR is capable of providing a

consistent and measurable improvement in MobileNetv1 network accuracy for image classification, once again lending credibility to its utility as a Filter Weight Matrix Regularizer.

7. CONCLUSION

This Thesis principally discussed the relevant background necessary for the development of this work in Chapter 1. Next, we discussed the intuition for developing ACVR in Chapter 2 as a means of employing underutilized network elements, and achieving the full representational power of a given CNN architecture. The method of this intended success is to use a Gradient Descent Orthogonalization algorithm parameterized by gamma, to vary the ratio of filter diversity, to data generated knowledge. Following this introduction, we briefly discussed the major contributions of this work in Section 2.1, and its inspirations in Section 2.2. This led to the discussion of the implementation and mathematics of ACVR in Section 2.3. This chapter provided both an efficient vectorized computation method, as well as an analytic solution to the Back-propagation Calculus from the perspective of the ACV Regularizer using a specific vector element. Knowing both the reason for our efforts, and our intended goal, we tested the effects of ACVR on the filters of a low-dimensional CNN in Chapter 3. These tests provided both graphical and numerical evidence that ACVR is capable of providing immense vector diversity, whereas regular training implementations do not. This gave credibility to the notion that ACVR has been both properly implemented, and is theoretically sound. This led to the problem description of regularization in Chapter 4: stating that our goal is to discover a regularization configuration, and gamma value, that is capable of consistently improving the network accuracy. Next, the effects of ACVR was tested on the traditional convolutional filters of VGG-19 in Chapter 5, which led to the creation of D-ACVR. From these experiments, we determined that there exists a regularization configuration and gamma value of D-ACVR that is capable of improving the network accuracy by up to 3.12%. Finally, in Chap-

ter 6, the effects of ACVR and D-ACVR was tested on the Pointwise convolutional filters in MobileNetv1. These experiments yielded a smaller, but still consistent and detectable, increase in convergent accuracy of 0.99%, with D-ACVR.

Overall, the process of training and testing models for the purpose of determining the effects of ACVR and D-ACVR has been a long and arduous endeavor. In order to facilitate the effective acquisition of the data necessary to make this work feasible, an automated training and testing suite was developed, and the work was distributed across multiple GPUs. Simply put, this hyper-parameter and layer configuration search takes a great deal of time and effort. Given that Deep Learning has since its inception been a balancing act between practicality, and optimal performance, it is important to discuss whether the accuracy increases due to ACVR and D-ACVR are relevant given these prerequisites. Unfortunately, the answer to this question is implementation dependent. In low-risk models, an accuracy increase of 3.12% can be highly negligible, and the increased accuracy may not be worth the time and resources required to obtain it. However, in CNN models targeted towards medical or flight control industries, an accuracy increase of 3.12% can have extreme significance. Luckily, there is one thing that can be claimed without conjecture or implementation dependencies: this Thesis has gathered evidence that D-ACVR has been able to provide tangible and consistent improvements to model performance, being tested in both traditional convolutional filters, and Pointwise convolutional filters. And finally, this evidence lends substantial credibility to D-ACVR's utility as a Filter Weight Matrix Regularizer. However, given the nature of the Regularization problem, it is likely that D-ACVR may have yet to reach its true potential, and merits further investigation.

7.0.1 Future Directions

Looking forward, this investigation can take the form of several open questions. Firstly, ACVR and D-ACVR have been examined on only three CNN architectures. There are many CNN architectures that may benefit from Regularization of this kind, both novel, and classical. Next, since a vector has two qualities, direction and magnitude, ACVR and D-ACVR may be enhanced by adding an additional component that accounts not only for vector similarity, but also, vector magnitude. In addition, ACVR and D-ACVR have yet to be fully examined alongside other regularization techniques. Combination with techniques such as L_1 and L_2 Regularization, or Dropout [1], may substantially increase model performance. Finally, It is hypothesized that ACVR and D-ACVR may be useful in creating model ensembles, by enforcing different architectures of the same type to form entirely separate methods of arriving at the same conclusion. This parallel training, culminating in an ensemble, may increase generalization power, and provide a more stable, and accurate model.

REFERENCES

REFERENCES

[1] F. Chollet, *Deep learning with Python*. Shelter Island, NY: Manning Publications Co., 2018.

[2] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," arXiv.org, 2016, last accessed on 3/21/2020.

[3] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," arXiv.org, 2018, last accessed on 3/21/2020.

[4] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," vol. 89. NIPS, 1989.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv.org, 2014, last accessed on 3/21/2020.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv.org, 2017, last accessed on 3/21/2020.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," arXiv.org, 2015, last accessed on 3/21/2020.

[8] C. Luo, J. Zhan, L. Wang, and Q. Yang, "Cosine normalization: Using cosine similarity instead of dot product in neural networks," arXiv.org, 2017, last accessed on 3/21/2020.

[9] Y. Yu, Y. Li, and Z. Zhou, "Diversity regularized machine." IJCAI, 2011, pp. "1603–1608".

[10] T. Wang, L. Fan, and H. Wang, "Simultaneously learning architectures and features of deep neural networks," arXiv.org, 2019, last accessed on 3/21/2020.

[11] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[12] F. Chollet *et al.*, "Keras," https://keras.io, 2015, last accessed on 3/21/2020.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org, last accessed on 3/21/2020. [Online]. Available: https://www.tensorflow.org/

[14] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, 2020.

[15] MATLAB, *9.6.0.1072779 (R2019a)*. Natick, Massachusetts: The MathWorks Inc., 2019.

VITA

VITA

William Singleton received a BS in Bio-Medical Engineering from the Purdue School of Engineering and Technology in 2018. Currently, he is pursuing a MS in Electrical and Computer Engineering at the Purdue School of Engineering and Technology in the field of artificial intelligence. His research interests include: Applied Mathematics, Embedded Systems, and Machine Learning Algorithms.