CUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

PhD THESIS

Logarithmic Learning Differential Convolutional Neural Network

Yasin MAGOMBE

Department of Computer Engıneerıng

February, 2024

CUKUROVA UNIVERSITY

INSTITUTE OF NATURAL AND APPLIED SCIENCES

PhD THESIS APPROVAL

Logarithmic Learning Differential Convolutional Neural Network

Yasin MAGOMBE

DEPARTMENT OF COMPUTER ENGINEERING

This Doctorate Thesis was evaluated by the following Jury Members on …/.../…… and was approved by unanimity / majority of votes.

Jury : Prof. Dr. Mutlu AVCI (Advisor) ……

: Prof. Dr. Selma Ayşe ÖZEL ……

: Prof. Dr. Serdar YILDIRIM ……

: Assoc. Prof. Dr. Ali İNAN ……

: Asst. Prof. Dr. Mehmet SARIGÜL ……

This Thesis was written in the Department of Computer Engineering, Institute of Natural and Applied Sciences.

Thesis Number: 2019913153

Prof. Dr. Sadık DİNÇER
Director
Institute of Natural and Applied Sciences

This work was supported by the XXXXX.
Project ID: XXX-XXXX-XXXX

CONTENTS

ABSTRACT I
ÖZ II
GENİŞLETİLMİŞ ÖZET III
ACKNOWLEDGEMENTS V
LIST OF TABLES VI
LIST OF FIGURES VIII
LIST OF EQUATIONS XIII
LIST OF ALGORITHM XIV
SYMBOLS AND ABBREVIATIONS XV
1. INTRODUCTION 1
1.1. The Aims and Objectives 2
1.2. Contribution to Existing Knowledge 2
1.3. Main Contributions of the Research 3
1.4. Outline of the Thesis 3
2. PRELIMINARY WORK 5
3. MATERIAL AND METHOD 11
3.1. Introduction 11
3.2. Convolutional Neural Network 11
3.2.1. Convolution Layer 11
3.2.2. Pooling Layer 11
3.2.3. Fully Connected Layer 12
3.2.4. Data Augmentation 12
3.2.5. Dropout 12
3.3. Pretrained Deep-Learning Structures 12
3.3.1. AlexNet 13
3.3.2. VGG 13
3.3.3. GoogLeNet (Inception v1) 13
3.3.4. ResNet (Residual Network) 13
3.3.5. DenseNet 14
3.3.6. Xception 14
3.3.7. SE-ResNet (Squeeze-and-Excitation ResNet) 14
3.3.8. NASNet (Neural Architecture Search Network) 14
3.3.9. MobileNet-v1 15
3.3.10. ShuffleNet 15
3.3.11. EfficientNet 15

3.3.12. MobileNet-v3 15
3.3.13. FBNet 15
3.3.14. MnasNet 16
3.3.15. ProxylessNAS 16
3.4. Differential Convolutional Neural Network (DiffCNN) 16
3.5. Logarithmic Differential Convolutional Neural Network (LDiffCNN) 19
3.5.1. AND /OR Implementation 23
3.5.2. LogRelu Activation Function 27
3.5.3. Implemented CNN Structures 28
3.6. Datasets 31
3.6.1. Cifar-10 Dataset 32
3.6.2. Animal Faces Dataset 32
3.6.3. Vegetable Image Dataset 33
3.6.4. 10- Monkey Species 33
3.6.5. Flower Classification 33
3.6.6. Intel Image Scene Classification 34
3.6.7. Tomato Leaf Dataset. 34
3.6.8. MNIST Dataset 35
3.6.9. Caltech-UCSD Birds-200-2011 (CUB-200-2011) 35
3.6.10. Stanford Cars 35
3.7. Hardware and Libraries 35
3.7.1. Hardware 35
3.7.2. Libraries 36
3.8. Experiment Details 36
3.9. Evaluation Metrics 36
3.9.1. Accuracy 37
3.9.2. Precision 37
3.9.3. Recall 37
3.9.4. F1-Score 37
4. RESULTS AND DISCUSSIONS 39
4.1. Experimental Results 39
4.2. The Classification Performance of MNIST 59
4.2.1. Analysis of Various Learning for the Adam Optimizer 59
4.2.2. Analysis of SGD Optimizer Different Learning Rate 59
4.2.3. Analysis of RMSprop Optimizer with Different Learning Rate 60
4.2.4. The DiffCNN Classification Performance on the MINST Dataset 86
4.3. The Classification Performance of CIFAR-10 Dataset 95

4.3.1. The Classification DiffCNN Performance 95
4.3.2. Pretrained Models Performance on Cifar10 Results 104
5. CONCLUSIONS 107
REFERENCES 109
CURRICULUM VITAE 121

CUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

PhD THESIS

Logarithmic Learning Differential Convolutional Neural Network

Yasin MAGOMBE

Advisor: Prof. Dr. Mutlu AVCI

Department of Computer Engineering

ABSTRACT

Convolutional Neural Networks (CNNs) have been instrumental in transforming the field of computer vision by their innovative design and training methodologies for image classification. The differential convolutional neural network with simultaneous multidimensional filter realization has been successful in improving the performance of the convolutional neural network, although it suffers from calculation cost drawbacks. This Thesis proposes a solution to the drawback by introducing logarithmic learning integration into the differential Convolutional neural network. The proposed approach employs LogRelu activation, a logarithmic cost function, and a unique logarithmic learning method for faster error minimization and convergence. The study evaluates the effectiveness of these proposed methods across multiple datasets and optimization algorithms, including SGD and Adam. The experimental findings show that integrating LogRelu leads to performance improvements ranging from 1.61% to 5.44% across convolutional neural networks, while the same integration on ResNet-18, ResNet-34, and ResNet-50 enhances top-1 accuracy in the range of 3.07% and 9.96%. Moreover, the Logarithmic Differential CNN consistently outperforms standard CNNs with an accuracy increase of up to 3.02% with the adaptation of the Logarithmic Cost Function. The study also evaluates various activation functions on Differential CNN models and pre-trained models using MNIST and Cifar10 datasets. Among the activations, LeakyReLU, ELU, SELU, and LogRelu consistently outperform ReLU, with LogRelu being particularly effective at lower learning rates. Although the differential CNN faces compatibility issues with certain optimizers, it displays adaptability and excels with an SGD rate of 0.01 and lower rates for Adam and RMSprop.The experimental results proved the efficiency of the proposed approach.

Keywords: Logarithmic Learning, Convolutional Neural Networks, Differential Convolution

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DOKTORA TEZİ

Logaritmik Öğrenme Diferansiyel Evrişimsel Sinir Ağı

Yasin MAGOMBE

Danışman: Prof. Dr. Mutlu AVCI

Anabilim Dalı :Bilgisayar Mühendisliği Bölümü

ÖZ

Evrişimsel Sinir Ağları (CNN'ler), görüntü sınıflandırmaya yönelik yenilikçi tasarım ve eğitim metodolojileriyle bilgisayarlı görme alanını dönüştürmede etkili olmuştur. Eşzamanlı çok boyutlu filtre gerçekleştirmeli diferansiyel evrişimli sinir ağı, hesaplama maliyeti dezavantajlarına sahip olmasına rağmen, evrişimli sinir ağının performansını artırmada başarılı olmuştur. Bu Tez, Diferansiyel Evrişimli sinir ağına logaritmik öğrenme entegrasyonu sağlayarak bu dezavantaja bir çözüm önermektedir. Önerilen yaklaşım, daha hızlı hata minimizasyonu ve yakınsama için LogRelu aktivasyonunu, logaritmik maliyet fonksiyonunu ve benzersiz bir logaritmik öğrenme yöntemini kullanır. Çalışma, önerilen bu yöntemlerin etkinliğini SGD ve Adam da dahil olmak üzere birden fazla veri kümesi ve optimizasyon algoritması genelinde değerlendiriyor. Deneysel bulgular, LogRelu'nun entegrasyonunun, konvolüsyonel sinir ağlarında %1.61 ila %5.44 arasında performans iyileştirmelerine neden olduğunu göstermektedir. Aynı entegrasyonun ResNet-18, ResNet-34 ve ResNet-50 üzerinde uygulanması, top-1 doğruluğunu %3.07 ila %9.96 aralığında artırmaktadır. Üstelik Logaritmik Diferansiyel CNN, Logaritmik Maliyet Fonksiyonunun uyarlanmasıyla %3.02'ye varan doğruluk artışıyla standart CNN'lerden daha iyi performans gösteriyor. Çalışma aynı zamanda Diferansiyel CNN modelleri ve MNIST ve Cifar10 veri kümelerini kullanan önceden eğitilmiş modeller üzerindeki çeşitli aktivasyon fonksiyonlarını da değerlendirmektedir. Aktivasyonlar arasında LeakyReLU, ELU, SELU ve LogRelu sürekli olarak ReLU'dan daha iyi performans gösteriyor ve LogRelu özellikle düşük öğrenme oranlarında etkili oluyor. Diferansiyel CNN, belirli optimize edicilerle uyumluluk sorunlarıyla karşı karşıya kalsa da, uyarlanabilirlik sergiliyor ve 0,01 SGD oranıyla ve Adam ve RMSprop için daha düşük oranlarla öne çıkıyor.Deneysel sonuçlar önerilen yaklaşımın etkinliğini kanıtladı.

Anahtar Kelimeler: Logaritmik Öğrenme, Konvolüsyonel Sinir Ağları, Diferansiyel Konvolüsyon

GENİŞLETİLMİŞ ÖZET

Bu tezde, Logaritmik öğrenmenin Diferansiyel Evrişimli Sinir Ağı (DiffCNN) üzerinde uygulanması, LogRelu adı verilen yeni bir logaritmik aktivasyon fonksiyonunun, bir Logaritmik Maliyet Fonksiyonu ve Logaritmik öğrenme metodolojisinin geliştirilmesini içermektedir. Önerilen yaklaşım, yedi farklı veri seti kullanılarak CNN, LCNN1, LCNN2, DiffCNN, LDiffCNN1 ve LDiffCNN2 etiketli altı farklı CNN modeli üzerinde değerlendirilmektedir: : CIFAR-10 (Krizhevsky & Hinton, 2009), Animal Faces(Choi et al., 2020), Vegetable Image(Ahmed et al., 2021), 10-Monkey Species(Utkarsh Saxena, 2022a), Tomato Leaves(Motwani, 2021), Flower Classification(Utkarsh Saxena, 2022b), and Intel Scenes(Bansal, n.d.). Modellerin etkililiği doğruluk, kesinlik, hatırlama ve F1 puanı olmak üzere dört ölçüt kullanılarak değerlendirilir. Ek olarak, SGD için le-2 ile stokastik gradyan iniş (SGD) ve Adam optimizasyonu kullanıldı ve Adam, her türlü CNN uygulaması sırasında 5e-4 ve 3e-4 öğrenme oranlarını kullandı. Önerilen yöntemlerin etkinliğini kanıtlamak için iki farklı deney düzenlendi. İlk deneyde LogRelu aktivasyon fonksiyonunun DiffCNN üzerinde uygulaması yapılmış ve sonuçlar diğer modellerle karşılaştırılmıştır. Sonuçlar %1,61 ile %5,44 arasında değişen performans iyileşmesini kanıtladı. En büyük iyileştirmeler 10 Maymun Türü veri setinde gözlemlendi; burada doğruluk, F1 Puanı, Geri Çağırma ve Hassasiyet sırasıyla %5,28, %4,60, %5,44 ve %5,16 oranında iyileştirildi. Ancak Domates Yaprakları veri setinde standart CNN, %0,51'lik bir farkla marjinal olarak daha iyi performans gösteriyor. İkinci deneyde, Logaritmik Maliyet Fonksiyonunun DCNN ve logaritmik Diferansiyel Evrişimli Sinir Ağı'na (LDiffCNN) uygulanması, genellikle tüm veri kümelerinin çoğunda standart CNN modelinden daha iyi performans gösterir. Modeller için daha yüksek doğruluk, F1 puanı, geri çağırma ve hassasiyet ile performans farklılıkları yaklaşık %2,33 ile %3,02 arasında değişen nispeten orta düzeydedir. Ayrıca, Sebze ve Domates Yaprakları veri kümeleri için, üç modelin tümü (CNN, DiffCNN ve LDiffCNN) dar aralıklarda (sırasıyla %0,30 ve %1,07) tutarlı bir şekilde benzer sonuçlar üretir. Bunun aksine, CIFAR-10, Hayvan Yüzleri ve Intel Sahne veri kümeleri orta derecede çeşitlilik gösterirken, Çiçek Sınıflandırması veri kümesi en geniş çeşitliliği (%5,28) gösterirken, 10 Maymun Türü veri kümesi orta ila yüksek değişkenlik göstermektedir. Son olarak, geliştirilmiş performansın yanı sıra, yineleme sayısının da farklı veri kümeleri arasındaki dönemler arasında %1'den %38'e kadar olan farkı azalttığını belirtmekte fayda var. Test performans sonuçları önerilen yöntemlerin etkinliğini kanıtlamaktadır. Aynı entegrasyonun ResNet-18, ResNet-34 ve ResNet-50 üzerinde uygulanması, top-1 doğruluğunu %3.07 ila %9.96 aralığında artırmaktadır. Üstelik Logaritmik Diferansiyel CNN, Logaritmik Maliyet Fonksiyonunun uyarlanmasıyla %3.02'ye varan doğruluk artışıyla standart CNN'lerden daha iyi performans gösteriyor

Çalışma ayrıca MNIST ve Cifar10 veri kümelerini kullanarak MobileNet, LeNet, DenseNet ve ResNet dahil olmak üzere çeşitli derin öğrenme modellerinde LogRelu ve diğer aktivasyon fonksiyonlarının (ReLU, LeakyReLU, ELU, SELU, Mish) kapsamlı bir değerlendirmesini

gerçekleştirdi. Etkinleştirme işlevi seçimi, LeakyReLU, ELU, SELU ve LogRelu'nun sürekli olarak ReLU'dan daha iyi performans göstermesiyle model performansını etkileyen kritik bir faktör olarak ortaya çıktı. Özellikle, 0,001 öğrenme oranıyla LeakyReLU ve ELU, birden fazla modelde %98,8 ile %99,1 arasında değişen etkileyici doğruluk seviyeleri elde ederken ReLU, %9,8 ile %98,8 gibi düşük bir doğrulukla 0,01'de zorlandı. Ek olarak LogRelu, özellikle düşük öğrenme oranlarında genellikle diğer aktivasyonlarla eşleşen veya onları geride bırakan güçlü bir performans sergiledi.

Ancak DiffCNN'u uygularken, 0,01 öğrenme oranında hem Adam hem de RMSprop iyileştiricileriyle uyumluluk sorunları ortaya çıktı. Öte yandan, daha düşük öğrenme oranlarında, özellikle 0,001, 0,0005 ve 0,0003'te SGD optimize ediciyle uyumluluk sorunları vardı; SGD ile 0,01 gibi daha yüksek bir öğrenme oranı kullanıldığında özellikle daha iyi sonuçlar gözlemlendi. Bu bulgular, model yapılandırma sürecinde hem optimize edici seçiminin hem de öğrenme oranının dikkatle değerlendirilmesinin kritik öneminin altını çizmektedir. Ayrıca, değerlendirme için kullanılan modellerden DenseNet'in, sonuçlarda da ortaya çıktığı gibi, bu çalışmada kullanılan aktivasyon fonksiyonları ve öğrenme oranları genelinde eğitimi tamamlaması uzun bir çalışma süresi almaktadır.

Genel olarak çalışmamız, yalnızca performansı artırmakla kalmayıp aynı zamanda sinir ağı eğitiminin yakınsamasını da hızlandıran önerilen LDiffCNN metodolojisinin verimliliğini göstermektedir.

ACKNOWLEDGEMENTS

I extend my sincere gratitude to my supervisor, Prof. Dr. Mutlu AVCI, for his unwavering support and guidance throughout my PhD journey. His invaluable feedback, patience, and encouragement have played a pivotal role in shaping the trajectory of my research work.

I am equally thankful to the esteemed members of my Thesis Jury committee, Prof. Dr. Selma Ayşe ÖZEL, Asst. Prof. Dr. Mehmet SARIGÜL, Prof. Dr. Serdar YILDIRIM, and Assoc. Prof. Dr. Ali İNAN. Their critical insights and constructive feedback have significantly contributed to refining and improving the quality of my research.

My heartfelt appreciation also goes to my colleagues and friends who have been a constant source of support, motivation, and encouragement throughout my PhD journey. Their kind words, thoughtful suggestions, and shared laughter have made this academic pursuit more enjoyable and memorable.

I express my gratitude to the Council of Higher Education (YÖK SCHOLARSHIP) and the Islamic University in Uganda (IUIU) for providing me with the opportunity to pursue my Ph.D. Their support, along with the research facilities and resources made available to me, has been instrumental in the successful completion of my doctoral studies.

Finally, I want to convey my deepest thanks to my family for their unconditional love, unwavering support, and continuous encouragement throughout my academic career. Their faith in me has been a constant source of strength and inspiration.

To everyone who has contributed to my Ph.D. journey, thank you for your unwavering support and encouragement. Your collective efforts have transformed this academic experience into a remarkable and enriching chapter of my life.

LIST OF TABLES

Table 3.1: Truth table of AND Logical Operation 24
Table 3.2: Truth table of OR Logical Operation 24
Table 3.3: Numerical Example for AND Operation....25
Table 3.4: Numerical Example for OR Operation....26
Table 3.5: Network Structures....28
Table 3.6: Class Distribution of Datasets 32
Table 3.7: CIFAR-10 Dataset Contents....32
Table 3.8: Animal Faces Dataset Details....32
Table 3.9: Vegetable Image Dataset Details....33
Table 3.10: 10-Monkey Species Dataset Details 33
Table 3.11: Flower Classification Dataset Details....34
Table 3.12: Intel Image Scene Classification Dataset Details....34
Table 3.13: Tomato Leaf Dataset Details 35
Table 3.14: Overview of Model Configurations. 36
Table 4.1: Results Summary for SGD with Learning Rate of 1e-2....40
Table 4.2: Summary of Results for Adam Optimizer with Learning Rate of 5e-4....40
Table 4.3: Results Summary for Adam optimizer with Learning Rate=3e-4....40
Table 4.4: Results Summary for SGD with Learning Rate of 1e-2....42
Table 4.5: Results Summary for Adam (optimizer) Learning Rate=5e-4 42
Table 4.6: Results Summary for Adam (optimizer) Learning Rate=3e-4 42
Table 4.7: Results Summary for 10-Monkey Species SGD with Learning Rate of 1e-2 44
Table 4.8: Results Summary for 10-Monkey Species with Adam and Learning Rate of 5e-4....44
Table 4.9: Results Summary for 10-Monkey Species Dataset with Adam and Learning Rate of 3e-4 44
Table 4.10: Results Summary for SGD with Learning Rate of 1e-2 on Cifar-10 Dataset....46
Table 4.11: Results Summary for Adam with a Learning Rate of 5e-4 on Cifar-10 Dataset 46
Table 4.12: Results Summary for Adam Learning Rate of 3e-4 on Cifar-10 Dataset 46
Table 4.13: Results Summary for Intel Scene Dataset with SGD and Learning Rate of 1e-2....48
Table 4.14: Results Summary for Adam with a Learning Rate of 5e-4 on Intel Scene Dataset....48
Table 4.15: Results Summary for Adam with a Learning Rate of 3e-4 on Intel Scene Dataset....48
Table 4.16: Results Summary for Flower Classification with SGD and Learning Rate of 1e-2 50
Table 4.17: Results Summary for Adam with Learning Rate 5e-4 on Flower Classification Dataset....50
Table 4.18: Results Summary for Adam with Learning Rate 3e-4 on Flower Classification Dataset....50

Table 4.19: Results Summary for SGD with Learning Rate of 1e-2 on Tomato Leaves Dataset.....52
Table 4.20: Results Summary for Adam with Learning Rate of 5e-4 on Tomato Leaves Dataset..52
Table 4.21: Results Summary for Adam with 3e-4 Learning Rate on Tomato Leaves Dataset 52
Table 4.22: Model Accuracy on Cub-200-2011 and Stanford-Cars Datasets.....54
Table 4.23: Results Summary for Adam with 5e-4 Learning Rate overall best.....55
Table 4.24: Results Summary for Adam with 3e-4 Learning Rate overall best.....56
Table 4.25: Performance of Adam Optimizer with LR: 0.001.....61
Table 4.26: Performance of Adam Optimizer with LR: 0.01.....61
Table 4.27: Performance of Adam Optimizer with LR: 5e-4 61
Table 4.28: Performance of SGD Optimizer with LR: 0.001 69
Table 4.29: Performance of SGD Optimizer with LR: 0.01 69
Table 4.30: Performance of SGD Optimizer with LR: 0.0005 69
Table 4.31: Performance of RMSprop Optimizer with LR: 0.001.....78
Table 4.32: Performance of RMSprop Optimizer with LR: 0.01.....78
Table 4.33: Performance of RMSprop Optimizer with LR: 5e-4.....79
Table 4.34: DiffCNN with Adam Optimizer 88
Table 4.35: DiffCNN with SGD Optimizer 88
Table 4.36: DiffCNN with RMSprop Optimizer 88
Table 4.37: DiffCNN with Adam Optimizer 96
Table 4.38: DiffCNN with SGD Optimizer 96
Table 4.39: DiffCNN with RMSprop Optimizer 96
Table 4.40: Shows Results Utilizing Adam Optimizer with a Learning Rate of 0.001 105
Table 4.41: Shows Results Utilizing Adam Optimizer with a Learning Rate of 0.01 105
Table 4.42: Shows Results Utilizing Adam Optimizer with a Learning Rate of 0.0005 105
Table 4.43: Shows Utilizing SGD Optimizer with a Learning Rate of 0.001.....105
Table 4.44: Shows Utilizing SGD Optimizer with a Learning Rate of 0.01.....105
Table 4.45: Shows Results Utilizing SGD Optimizer with a Learning Rate of 0.0005.....106
Table 4.46: Shows Results Utilizing RMSprop Optimizer with a Learning Rate of 0.001 106
Table 4.47: Shows Results Utilizing RMSprop Optimizer with a Learning Rate of 0.01 106
Table 4.48: Shows Results Utilizing RMSprop Optimizer with a Learning Rate of 0.005 106

LIST OF FIGURES

Figure 3.1: Fixed Filters 17
Figure 3.2. Differential Convolution Operation(Sarıgül et al., 2019) 18
Figure 3.3: Log Transformations 21
Figure 3.4: Log Transformation on cubic equation for global and local minima 21
Figure 3.5: Log Transformation on Quadratic Equation for Local and global minima 22
Figure 3.6: Log Transformation on Quartic Equation Global and local minima 22
Figure 37: Single Neuron 23
Figure 3.8: MSE and MSEgd 26
Figure 3.9: CNN Structure 30
Figure 3.10: LCNN1 Structure 30
Figure 3.11: LCNN2 Structure 30
Figure 3.12:DiffCNN Structure 31
Figure 3.13:LDiffCNN1 Structure 31
Figure 3.14:LDiffCNN2 Structure 31
Figure 4.1: Training Loss for SGD 41
Figure 4.2: Test Accuracy for SGD 41
Figure 4.3: Training Loss for Adam (5e-4) 41
Figure 4.4: Test Accuracy for Adam (3e-4) 41
Figure 4.5: Training Loss for Adam (3e-4) 41
Figure 4.6: Test Accuracy for Adam (3e-4) 41
Figure 4.7: Training Loss for SGD 43
Figure 4.8: Test Accuracy for SGD 43
Figure 4.9: Training Loss for Adam (5e-4) 43
Figure 4.10: Test Accuracy for Adam (5e-4) 43
Figure 4.11: Training Loss for Adam (3e-4) 43
Figure 4.12: Test Accuracy for Adam (3e-4) 43
Figure 4.13: Training Loss for SGD 45
Figure 4.14: Test Accuracy for SGD 45
Figure 4.15: Training Loss for Adam (5e-4) 45
Figure 4.16: Test Accuracy for Adam (5e-4) 45
Figure 4.17:Training Loss for Adam (3e-4) 45
Figure 4.18: Test Accuracy for Adam (3e-4) 45
Figure 4.19: Training Loss for SGD 47
Figure 4.20: Test Accuracy for SGD 47
Figure 4.21: Training Loss for Adam (5e-4) 47

Figure 4.22: Test Accuracy for Adam (5e-4) 47
Figure 4.23: Training Loss for Adam (3e-4) 47
Figure 4.24: Test Accuracy for Adam (3e-4) 47
Figure 4.25: Training Loss for SGD 49
Figure 4.26 Test Accuracy for SGD 49
Figure 4.27: Training Loss for Adam (5e-4) 49
Figure 4.28: Test Accuracy for Adam (5e-4) 49
Figure 4.29: Training Loss for Adam (3e-4) 49
Figure 4.30: Test Accuracy for Adam (3e-4) 49
Figure 4.31: Training Loss for SGD 51
Figure 4.32: Test Accuracy for SGD 51
Figure 4.33: Training Loss for Adam (5e-4) 51
Figure 4.34: Test Accuracy for Adam (5e-4) 51
Figure 4.35: Training Loss for Adam (3e-4) 51
Figure 4.36: Test Accuracy for Adam (3e-4) 51
Figure 4.37: Training Loss for SGD 53
Figure 4.38: Test Accuracy for SGD 53
Figure 4.39: Training Loss for Adam (5e-4) 53
Figure 4.40: Test Accuracy for Adam (5e-4) 53
Figure 4.41: Training Loss for Adam (3e-4) 53
Figure 4.42: Test Accuracy for Adam (3e-4) 53
Figure 4.43:Recall Performance 55
Figure 4.44:F1-Score Performance 55
Figure 4.45:Precision Performance 55
Figure 4.46:Accuracy Performance 55
Figure 4.47: Accuracy Performance 56
Figure 4.48:F1-Score Performance 56
Figure 4.49:Recall Performance 56
Figure 4.50:Precision Performance 56
Figure 4.51:Training Loss 57
Figure 4.52:Test Accuracy 57
Figure 4.53:Training Loss 57
Figure 4.54:Test Accuracy 57
Figure 4.55:Trainining Loss 57
Figure 4.56::Test Accuracy 57
Figure 4.57:Training Loss 57
Figure 4.58::Test Accuracy 57

Figure 4.59:Training Loss 58
Figure 4.60:Test Accuracy 58
Figure 4.61:Training Loss 58
Figure 4.62:Test Accuracy 58
Figure 4.63:Training Loss 58
Figure 4.4:Test Accuracy 58
Figure 4.65 MobileNet with Adam(0.001) 62
Figure 4.66:LeNet with Adam(0.001) 62
Figure 4.67:DenseNet with Adam(0.001) 63
Figure 4.68:ResNet with Adam(0.001) 63
Figure 4.69:Run Time for Adam(0.001) 64
Figure 4.70:MobileNet with Adam(0.01) 64
Figure 4.71:LeNet with Adam(0.01) 65
Figure 4.72:DenseNet with Adam(0.01) 65
Figure 4.73:ResNet with Adam(0.01) 66
Figure 4.74::Run Time for Adam (0.01) 66
Figure 4.75:MobileNet with Adam(0.0005 67
Figure 4.76:LeNet with Adam(0.0005) 67
Figure 4.77:DenseNet with Adam(0.0005) 68
Figure 4.78:ResNet with Adam (0.0005) 68
Figure 4.79:Run Time for Adam(0.0005) 69
Figure 4.80:MobileNet with SGD (0.001) 70
Figure 4.81:LeNet with SGD(0.001) 70
Figure 4.82:DenseNet with SGD (0.001) 71
Figure 4.83:ResNet with SGD(0.001) 71
Figure 4.84:Run Time for SGD (0.001) 72
Figure 4.85:MobileNet with SGD(0.01) 73
Figure 4.86: LeNet with SGD (0.01) 73
Figure 4.87:DenseNet with SGD(0.01) 74
Figure 4.88:ResNet with SGD(0.01) 74
Figure 4.89:Run Time with SGD (0.01) 75
Figure 4.90:MobileNet with SGD(0.0005) 76
Figure 4.91:LeNet with SGD(0.0005) 76
Figure 4.92:DenseNet with SGD (0.0005) 77
Figure 4.93:ResNet with SGD (0.0005) 77
Figure 4.94:Run Time with SGD (0.0005) 78
Figure 4.95:MobileNet with RMSprop (0.001) 79

Figure 4.96:LeNet with RMSprop (0.001)....79
Figure 4.97: DenseNet with RMSprop(0.001)....80
Figure 4.98:ResNet with RMSprop (0.001)....80
Figure 4.99:Run Time for RMSprop (0.001)....81
Figure 4.100:MobileNet with RMSprop(0.01)....82
Figure 4.101:LeNet with RMSprop (0.01)....82
Figure 4.102:DenseNet with RMSprop (0.01)....83
Figure 4.103:ResNet with RMSprop (0.01)....83
Figure 4.104:Run Time with RMSprop (0.01)....84
Figure 4.105::MobileNet for RMSprop(0.0005)....84
Figure 4.106:LeNet with RMSprop (0.0005)....85
Figure 4.107:DenseNet with RMSprop (0.0005)....85
Figure 4.108:ResNet with RMSprop (0.0005)....86
Figure 4.109:Run Time with RMSprop (0.0005)....86
Figure 4.110:DiffCNN with Adam (0.001)....89
Figure 4.111:DiffCNN with Adam (0.01)....89
Figure 4.112:DiffCNN with Adam (0.0005)....90
Figure 4.113:Run Time with DiffCNN with Adam Optimizer....90
Figure 4.114:DiffCNN with SDG (0.001)....91
Figure 4.115:DiffCNN with SDG (0.01)....91
Figure 4.116: DiffCNN with SDG (0.0005)....92
Figure 4.117: Run Time with DiffCNN with SGD Optimizer....92
Figure 4.118: DiffCNN with RMSprop (0.001)....93
Figure 4.119: DiffCNN with RMSprop (0.01)....93
Figure 4.120: Run Time with DiffCNN with RMSprop Optimizer....94
Figure 4.121: DiffCNN with RMSprop (0.0005)....94
Figure 4.122: DiffConv with Adam (0.001)....97
Figure 4.123: DiffConv with Adam (0.01)....97
Figure 4.124: DiffConv with Adam (0.0005)....98
Figure 4.125: DiffConv with Adam (0.0003)....98
Figure 4.126:Run Time for DiffCNN with Adam....99
Figure 4.127: DiffCNNwith SGD (0.001)....99
Figure 4.128: DiffCNNwith SGD (0.01)....100
Figure 4.129: DiffCNN with SGD (0.0005)....100
Figure 4.130: DiffCNN with SGD (0.0005)....101
Figure 4.131: Run Time for DiffCNN with SGD....101
Figure 4.132: DiffCNN with RMSprop (0.001)....102

Figure 4.133: DiffCNN with RMSprop (0.01).....102
Figure 4.134: DiffCNN with RMSprop (0.0005).....103
Figure 4.135: DiffCNN with RMSprop (0.0003).....103
Figure 4.136: Run Time for DiffCNN with RMSprop.....104

LIST OF EQUATIONS

Eq.(3. 1).....11
Eq. (3. 2).....12
Eq.(3. 3).....17
Eq.(3. 4).....17
Eq.(3. 5).....17
Eq.(3. 6).....17
Eq.(3.7).....18
Eq.(3. 8).....19
Eq.(3. 9).....19
Eq.(3. 10).....20
Eq.(3. 11).....20
Eq.(3. 12).....20
Eq.(3. 13).....23
Eq.(3. 14).....23
Eq(3. 15).....23
Eq.(3. 16).....27
Eq.(3.17).....37
Eq.(3. 18).....37
Eq.(3. 19).....37
Eq.(3. 20).....37

LIST OF ALGORITHM

Algorithm 1: Proposed Logarithmic Cost Function.....24
Algorithm 2: Pseudocode for Gradient Descent 25

SYMBOLS AND ABBREVIATIONS

ADAM : Adaptive Moment Estimation
CNN : Convolutional Neural Network
DiffCNN : Differential Convolutional Neural Network
ELU : Exponential Linear Unit
FC : Fully Connected
LDiffCNN : Logarithmic Differential Convolutional Neural Network
LR : Learning Rate
MLE : Maximum Likelihood Estimation
MNIST : Modified National Institute of Standards and Technology
NASNet : Neural Architecture Search Network
RELU : Rectified Linear Unit
RMSprop : Root Mean Squared Propagation
SGD : Stochastic Gradient Descent
VGG : Visual Geometry Group

1. INTRODUCTION

Over the past decade, deep learning has appeared in widespread applications in various data science fields (Bao et al., 2019; Caicedo et al., 2019), such as object detection (Apostolopoulos & Tzani, 2023; Gupta et al., 2021; Sharma & Mir, 2020; Singh & Desai, 2023; Zaidi et al., 2022; Zou et al., 2023), and feature extraction(Dhiman et al., 2023; Kaur & Sharma, 2023; Mutlag et al., 2020; Sarigul et al., 2020; Xiao et al., 2020) included in Artificial Intelligence(AI) (Ali et al., 2023; Almotairi et al., 2023; Kobrinskii, 2023; Ngombu et al., 2023, 2023; H. Wang et al., 2023). One of the most successful deep learning architectures for computer vision tasks is the Convolutional Neural Network (CNN) (Bharadiya, 2023; Deepa & Rasi, 2023; Moutik et al., 2023; Taye, 2023). CNNs integrate feature extraction and classification parts, this eliminates the explicit segmentation and feature extraction requirements. In the convolutional layers, multiple filters also known as kernels, are used for generating feature maps by sliding over input images. The feature maps utilize the receptive field of neurons to link them across layers within a specific area determined by the size of the matching filters. A significant advantage of CNNs is the weight-sharing mechanism, where the same filter is applied to the entire image through a convolution task. This approach enables the detection of equivalent features across the entire image. The convolution operation aids in the detection of visual patterns in the input data. This contributes to the success of CNNs in vision tasks with fast and accurate feature extraction(Lei et al., 2019; Sarıgül et al., 2019).

In addition to classical convolution methods of CNNs, some improved convolution techniques are also proposed for better image segmentation and recognition. One of the most up-to-date techniques is the differential convolution method(Dişken, 2023; Kader et al., 2021; Qu et al., 2020a; Sarıgül et al., 2019). Differential convolution preserves directional change information among pixel activations to realize extraction of pattern orientations. It increases the number of feature maps without increasing the number of trainable filters. It facilitates error propagation through neighborhood activations for obtaining improved performance and accelerated error minimization. Another area of improvement takes place in learning methods and computational elements for classifiers. Logarithmic Learning Generalized Classifier Neural network(L-GCNN), which uses a logarithmic function to represent the cost instead of squared error, resulting in total error minimization is one of these approaches. L-GCNN achieves faster convergence to optimal solutions and reduces the number of iterations required during training. This ability makes it a promising development approach in neural network literature(Melis & Avci, 2014). In this work, a Logarithmic Differential Convolutional Neural Network (LDiffCNN) is introduced by adapting a novel logarithmic learning approach to the differential convolution technique.

1.1. The Aims and Objectives

Deep convolutional neural networks are comprised of two key components: the feature extractor and the classifier. The feature extractor is composed of convolutional and pooling layers, while the classifier primarily utilizes fully connected layers. Constructing an efficient fully connected classifier involves determining the optimal number of layers and neurons for each layer. Equally important is the selection of appropriate activation functions and loss functions for these layers. This study aims to capitalize on activation and loss functions by presenting an implementation approach for logarithmic learning within the context of the Differential Convolutional Neural Network (DCNN). The proposed method involves the creation of a novel logarithmic activation function named LogRelu, a Logarithmic Cost Function, and a logarithmic learning methodology to adapt them to the Differential Convolutional Neural Network.

1.2. Contribution to Existing Knowledge

This thesis introduces logarithmic learning into the Differential Convolutional Neural Network (DiffCNN), featuring the creation of a novel logarithmic activation function named LogRelu, a Logarithmic Cost Function, and a Logarithmic learning approach. The study includes the development and evaluation of six CNN models (CNN, LCNN1, LCNN2, DiffCNN, LDiffCNN1, and LDiffCNN2) using seven diverse datasets. The models are assessed based on accuracy, precision, recall, and F1-score metrics, employing stochastic gradient descent (SGD) and the Adam optimizer in all implementations. Three experiments validate the effectiveness of the proposed methods.

In the initial experiment, the LogRelu activation function is implemented on CNN and DiffCNN, showcasing a performance improvement of up to 5.44%, with significant enhancements in the 10-Monkey Species dataset. The second experiment extends logarithmic learning differential convolution to ResNet-18, ResNet-34, and ResNet-50, revealing notable accuracy gains on the Cub-200-2011 and Stanford-Cars datasets. The third experiment highlights LDiffCNN's dominance, with moderate performance increments and a reduction in the number of iterations by up to 38%.

Furthermore, the study evaluates various activation functions on Differential CNN models and pre-trained models using MNIST and the CIFAR-10 datasets. LeakyReLU, ELU, SELU, and LogRelu consistently outperform ReLU, with LogRelu proving particularly effective at lower learning rates. LogRelu exhibits outstanding accuracy on the MNIST dataset and competitive performance on CIFAR-10, demonstrating stability across different learning rates and optimizers for DiffCNN.

Conclusively, LogRelu emerges as the most reliable activation function for achieving high accuracy levels. The study emphasizes the importance of considering these insights when selecting activation functions and optimizers for models. The efficiency of the proposed methods is affirmed by the test performance results.

1.3. Main Contributions of the Research

The main contributions of this study are summarized as follows:

- The proposal of the LogRelu activation function which exploits the use of the log1p function for image classification problems.
- The adaptation of the proposed activation function in differential CNN.
- A novel logarithmic cost function with logarithmic learning that makes use of maximum likelihood is proposed and implemented.
- A comprehensive evaluation of various activation functions in deep learning models, highlighting their significant impact on model performance on pre-trained models and DiffCNN
- Integration of Lograthmic learning into ResNet models

1.4. Outline of the Thesis

This thesis is divided into five main sections.

- The Literature Review (Section 2) provides an overview of related studies in the field of deep learning.
- In Section 3, the Methodology section, the materials used in the study, including structures, algorithms, and training methodologies, are elaborated in detail.
- Section 4, the Experimental Analysis, provides an exhaustive account of all experiments conducted, including a comparison of the proposed neural network structures and classical structures, as well as an in-depth performance analysis of the models.
- The classification results obtained are discussed in detail, and potential future works are highlighted.
- Finally, Section 5 presents a comprehensive summary of all the studies and findings presented within the Conclusion section.

2. PRELIMINARY WORK

This section delivers a comprehensive overview of the literature in this field. It commences by delving into the historical evolution and advancements of convolutional neural networks, tracing the growth of the Differential Convolutional Neural Network (DiffCNN) and the specific structure under consideration. The discussion extends to an in-depth exploration of analogous methodologies and adaptation. Following this, the focus shifts to an elaborate examination of techniques similar to the proposed novel convolution technique, accompanied by an analysis of their respective merits and limitations.

Recent literature indicates a dynamic research landscape in CNNs for image classification. Researchers have explored various architectures, optimization techniques, and training methodologies to enhance the performance of CNNs in this domain. Advancements in transfer learning(Alzubaidi et al., 2020; Iman et al., 2023; Niu et al., 2020; Pathak et al., 2022; Zhuang et al., 2020), architectural innovations(Bello et al., 2021; Bhatt et al., 2021; Eminaga et al., 2023; Sarigul et al., 2020; Zhong et al., 2021), attention mechanisms(Lei et al., 2023; X. Li et al., 2023; Roy et al., 2023; Teodoro et al., 2023; Y. Wang et al., 2023), data augmentation(Alomar et al., 2023; Liang et al., 2023; Naveed et al., 2024; Z. Yang et al., 2023; Yoo & Kang, 2023), normalization techniques(L. Huang et al., 2023; W. Ma et al., 2023; Yeo et al., 2023), ensemble methods(Hafiz et al., 2023; Han et al., 2023; Iqball & Wani, 2023), neural architecture search(Kang et al., 2023; Lee et al., 2023; White et al., 2023), adversarial training(Dong et al., 2020; L. Ma & Liang, 2023; Ryu & Choi, 2023; Woods et al., 2019), explainability(Kamakshi & Krishnan, 2023; Paiss et al., 2022; Patrício et al., 2023; Preechakul et al., 2022; Schorr et al., 2021; Vermeire et al., 2022), interpretability(Marchand–Maillet & Müller, n.d.; Radhakrishnan et al., 2017; Raglin & Basak, 2023; Y. Yang et al., 2023), hardware acceleration(Bolhasani et al., 2023; Örnhag et al., 2023; Rokh et al., 2023; Silvano et al., 2023; Yanamala & Pullakandam, 2023), and ethical considerations (Bird & Lotfi, 2023; Boopathi & Kanike, 2023) have collectively propelled the field forward. The continual exploration of these areas not only enhances the performance and efficiency of CNNs.

Convolutional Neural Networks (CNNs) for image classification have undergone significant progress and refinement. Transfer learning remains a fundamental strategy, employing pre-trained models like VGG, ResNet, and Inception, originally trained on ImageNet, to boost performance in various image classification tasks (He et al., 2016; Ioffe & Szegedy, 2015). Architectural innovations, such as dense connections in DenseNet and

wider networks in architectures like Wide ResNet, have highlighted the importance of feature reuse and network width in enhancing generalization capabilities (Bhatt et al., 2021; Elhani et al., 2023; Eminaga et al., 2023; Khan et al., 2020a; Sun et al., 2020; Tripathi, 2021). Attention mechanisms, inspired by the Transformer model, have been explored to enable CNNs to focus on relevant image regions, particularly beneficial in scenarios with complex scenes (Cheng et al., 2023; X. Li et al., 2023)

Data augmentation techniques continue to be pivotal for regularizing CNNs, including rotation, scaling, and flipping augmentations that enhance the model's adaptability to diverse datasets (Shorten & Khoshgoftaar, 2022). Normalization techniques like batch normalization, group normalization, and layer normalization remain crucial for stabilizing training and improving convergence (Zhou et al., 2020). Ensemble methods, involving the combination of predictions from multiple CNN models, have proven effective in boosting overall performance and increasing model robustness (Hafiz et al., 2023; Iqball & Wani, 2023)

Neural Architecture Search (NAS) methods, leading to discoveries like NASNet and EfficientNet, automate the architectural design process, exploring extensive search spaces to find optimized models (Kang et al., 2023; S. Li et al., 2023; Termritthikun et al., 2023; W. Wang et al., 2023). Adversarial training, which introduces adversarial examples during the training process, has been explored as a means to enhance CNN's robustness against adversarial attacks (Bai et al., 2020; Hashemi & Mozaffari, 2021; Tramèr et al., 2017; Y. Wang et al., 2019). The growing emphasis on explainability and interpretability in CNNs has become particularly evident, especially in critical applications such as healthcare. Additionally, hardware acceleration techniques, including the utilization of GPUs and TPUs, along with model quantization, continue to be explored to meet real-time computational demands (Corral et al., 2024; Kljucaric & George, 2023; Ravikumar & Sriraman, 2023).

The widespread adoption of deep learning architectures is driven by their exceptional representation capabilities, largely attributed to the foundational convolution operation. This operation employs trainable filters to slide over images, computing activation values that extract feature maps crucial for pattern recognition. The surge in deep learning popularity has spurred the development of novel convolutional models and techniques, including improved methods for image segmentation and recognition. One prominent technique is the differential convolution method introduced by (Sarıgül et al., 2019). This method preserves directional change information among pixel activations, enabling the extraction of pattern orientations, while also increasing feature maps without escalating filter numbers.

Additionally, it supports error propagation through neighborhood activations, resulting in enhanced performance and accelerated error reduction. The effectiveness of the differential convolution technique was evaluated through four experiments: integrating it with a traditional CNN structure yielded an accuracy boost of up to 55.29% across 10 datasets; on an adapted AlexNet, it improved Top1 and Top5 accuracies by 5.3% and 4.75% respectively; outperforming traditional CNNs and other convolution techniques like Tiled CNN and Dilated CNN, the Differential CNN showed superiority with 4.33% and 7.72% higher top1 performance values for CIFAR-10 and CIFAR-100; applying the technique to VGGNet and NIN models resulted in notable top-1 accuracy values of 93.58% and 75.06% for CIFAR10 and CIFAR100 with VGGNet, and 92.44% and 72.65% with NIN. These findings underscore the adaptability, efficacy, and benefits of the proposed differential convolution technique across various CNN architectures. Moreover, the advantages of differential convolution have sparked further research, exemplified by the study conducted by Kader et al. in 2021. Their work is built upon the merits of the differential deep-CNN model, which involves analyzing pixel-directional patterns in images using contrast calculations. The model exhibits a strong capability to accurately classify a substantial image database, devoid of technical complications. Consequently, this approach showcases exceptional overall performance. To evaluate and train the model's proficiency, a dataset comprising 25,000 brain magnetic resonance imaging (MRI) images, encompassing both abnormal and normal cases, was utilized. The experimental findings revealed that the proposed model achieved an impressive accuracy of 99.25%. This investigation underscores the potential of the proposed differential deep-CNN model as a valuable tool for automating the classification of brain tumors.

Similarly, another study by (Qu et al., 2020a) utilized a differential convolutional technique in identifying Standard Plane Identification in Fetal Brain Ultrasound Scans. The experimental results showed that this method achieved an accuracy of 92.93% when the algorithms, were evaluated on 30,000 2D ultrasound images from 155 fetal subjects ranging from 16 to 34 weeks. The study revealed that the differential CNN can be used to facilitate the implementation of the automated identification of FBSPs.In summary, the Differential convolution-adapted CNN structures showed impressive performance results in all experiments and outperformed the alternative structures as per the literature review.

Moreover, in a study by Dişken (2023), an improvement in spoof detection system resilience against speech signal additive noise is emphasized. This enhancement, achieved through noise mask integration, utilizes differential convolutional neural networks. The

study's numerical evidence shows an average equal error rate (EER) of 2.67% for known noise types and 3.10% for unknown types. Under pristine conditions, the ASVspoof 2015 data achieves a low EER of 0.83%. For the ASVspoof 2019 logical access condition, a commendable EER of 2.6% is reported. These findings robustly affirm the efficacy of the proposed methodology, particularly in diverse additive noise conditions(Dişken, 2023).

In addition, another area which seen advancements is learning methods and computational elements for classifiers. These advancements have sparked various studies, including the Logarithmic Learning Generalized Classifier Neural Network (L-GCNN) introduced by Melis and Avci in 2014. Utilizing a logarithmic function for cost representation, L-GCNN achieves total error minimization, leveraging the benefits of logarithmic fast convergence. This method significantly reduces training time by up to 99.2%, concurrently enhancing classification performance by 60%. The test results not only address the time requirement issue in generalized classifier neural networks but also indicate potential improvements in classification accuracy.

In the study conducted by Yasin et al. (2024), the benefits of Differential CNN were strategically harnessed through the integration of logarithmic learning. The empirical findings consistently highlighted a discernible enhancement in accuracy, with LDiffCNN achieving a significant 3.02% improvement compared to its conventional counterpart. Of particular significance is the observation that the Logarithmic Differential Convolutional Neural Network exhibited a substantial reduction in training iterations, indicative of an accelerated convergence rate and a promising avenue for improving the efficiency of neural network training. These findings have significant implications for the development of more efficient and accurate neural network configurations and emphasize the importance of exploring innovative approaches to improve performance(Yasin et al., 2024).

In addition, the integration of logarithmic learning capitalizes on activation and loss functions by presenting an implementation approach for logarithmic learning within the context of the Differential Convolutional Neural Network (DiffCNN). Equally important is to give a brief on some of the existing activation functions

Regarding activation functions, various choices offer unique advantages and considerations. Sigmoid(Rumelhart et al., 1986) and Tanh(LeCun et al., 1998) functions provide bounded outputs, suitable for tasks like binary classification, but they suffer from vanishing gradient problems that can hinder deep network training. Rectified Linear Unit (ReLU)(Nair & Hinton, 2010), a widely adopted choice, introduces non-linearity and addresses vanishing gradients for positive inputs; however, its "dying ReLU" issue can lead

to neuron inactivity during training. Leaky ReLU(Maas et al., 2013), Parametric ReLU (PReLU) (He et al., 2015), and Exponential Linear Unit (ELU)(Clevert et al., 2015), emerge as remedies, enabling gradients for negative inputs and enhancing deeper network performance. Swish(Ramachandran, 2017) introduces balance, blending linear and sigmoid transformations, while Mish (Misra, 2019)combines the strengths of ReLU and Tanh. Each activation function caters to different scenarios, and their pros and cons guide their selection to optimize neural network performance. These activation functions offer a spectrum of solutions to the challenges posed by different types of data and network architectures. Swish and Mish, as newer contenders, have shown promise for improved convergence and performance, bridging the gap between simplicity and efficacy. The choice of activation function is a crucial decision in designing effective neural networks, as it affects not only convergence speed but also the network's ability to capture complex patterns in the data (Elfwing et al., 2018; Liu et al., 2019; Xu et al., 2015; Zhu et al., 2021). Furthermore, a study by (Zhu et al., 2021) also introduced a novel nonlinear nonmonotonic activation function for a convolutional neural network named Logish. The findings in this study revealed that Logish achieved 94.8% top-1 accuracy with ResNet–50 networks on CIFAR 10 dataset, and can reach 99.24% top-1 accuracy with DenseNet on the MNIST dataset and 88.52% top-1 accuracy with SE-Inception-v4 network on SVHN dataset respectively. It is higher than the Sigmoid, Tanh, ReLU, Swish, and Mish activation functions in the corresponding dataset. However, the other activations performed relatively better.

In a study conducted by Liu et al. (2019), the NLReLU activation function was introduced as an improvement upon ReLU. NLReLU maintains ReLU's sparsity while addressing issues like "dying ReLU" and vanishing gradients. It also mitigates bias variation and heteroscedasticity in the distribution of neuron data, thereby accelerating the learning process. Evaluated across ten convolutional neural networks with varying depths and diverse datasets, NLReLU demonstrated superior accuracy compared to ReLU and proved comparable to other established activation functions. In shallow convolutional networks using MNIST and CIFAR-10, NLReLU achieved 0.16% and 2.04% higher classification accuracy than ReLU. For deep networks on CIFAR-10, the findings indicated an average accuracy improvement of 1.35% due to NLReLU (Liu et al., 2019).

Recent studies by Gao & Zhang (2023) and Nag et al. (2023) continue to explore the significance of activation, aligning with the literature's emphasis (Yasin et al., 2024). The LogRelu activation function proposed in this thesis addresses the limitations of the ReLU function. While ReLU returns zero for negative inputs and the input value for positive inputs,

LogRelu ensures a non-zero output for inputs close to zero through a logarithmic transformation using the LOG1P function. This approach enhances numerical stability, particularly in image processing scenarios involving data normalization and probability computations. LogRelu effectively manages small probabilities, converting them into log probabilities and avoiding underflow issues. Its smooth, continuous, and differentiable nature makes it suitable for training deep neural networks, mitigating the problem of dead neurons associated with ReLU. The effectiveness of LogRelu is demonstrated in a study by Yasin et al. (2024).

In addition to the advancements discussed, the integration of transfer learning and pre-trained models, specifically ResNet18, ResNet34, and ResNet50, has become integral in recent neural network design. These models, pre-trained on extensive datasets like ImageNet, serve as powerful feature extractors for various computer vision tasks. Leveraging transfer learning with ResNet18, ResNet34, and ResNet50 involves fine-tuning specific tasks, accelerating convergence, and providing robust representations. Recent studies, including those by (Stephen et al., 2023; Ter-Sarkisov, 2020; Yadav et al., 2021), affirm the efficacy of using transfer learning with these pre-trained models across image-related tasks, highlighting their versatility and effectiveness in contemporary deep learning methodologies.

In this thesis, ResNet architectures 18, 34, and 50 undergo an integration of a convolution technique specifically, differential convolution and logarithmic learning, facilitated through the LogRelu activation. This alternative approach incorporates logarithmic learning differential convolution into the residual block, resulting in LDiffResNet-18, LDiffResNet-34, and LDiffResNet-50. Remarkably, the outcomes reveal enhanced top-1 accuracy on the Cub-200-2011 dataset, with improvements of 3.07%, 9.96%, and 7.59% for LDiffResNet-18, LDiffResNet-34, and LDiffResNet-50, respectively. Similarly, positive trends are observed in the Stanford-Cars dataset, showcasing top-1 accuracy gains of 5.64%, 3.99%, and 4.59% for LDiffResNet-18, LDiffResNet-34, and LDiffResNet-50(Yasin et al., 2024).

3. MATERIAL AND METHOD

3.1. Introduction

This section provides comprehensive details on the methodology of Deep Convolutional Neural Networks. It elaborates on the architecture of these networks and thoroughly outlines the methodologies employed during their training. Subsequently, the section introduces a collection of widely utilized datasets, serving as a benchmark for assessing the effectiveness of the proposed structures. Following this, various well-known convolutional neural network architectures are presented. Lastly, the section delves into the explanation of evaluation metrics and methods utilized to assess and compare the classification results' quality in the conducted experiments.

3.2. Convolutional Neural Network

CNNs are widely used for image recognition and have shown phenomenal performance on complex visual tasks (Lindsay, 2021; Melis & Avci, 2014). They utilize convolution, pooling, fully connected, and output layers (Khan et al., 2020b; Z. Li et al., 2021; Sarıgül et al., 2019). CNNs extract visual features using locally trained filters on input images, followed by downsizing through pooling. Deep features are extracted through successive convolutional layers. Finally, a classifier makes decisions based on these features.

3.2.1. Convolution Layer

In a convolutional layer, n input feature maps are convolved with shifting windows of size m×n and stride s. The convolutions are summed, and a bias value is added to each pixel. Non-linear activation functions (e.g., tanh, sigmoid, ReLU) are applied to obtain m output feature maps. Mathematically, the output feature maps C_i can be expressed as

$$\mathrm{C_j} = \mathrm{f}\left(\sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{L_i} * \mathrm{K_{i,j}} + \mathrm{B_j}\right) \qquad \text{Eq.(3. 1)}$$

Where L_j represents the *jth* input feature map, $K_{i,j}$ denotes the convolution kernel between the jth input feature map and the ith output feature map, B_j is the bias value, and f is the activation function.

3.2.2. Pooling Layer

The pooling layer applies a kernel and pooling operation (maximum, average, or L2) to produce a new feature map. It reduces neuron count in subsequent layers and generates location-

independent features, enhancing the computational efficiency and generalization capabilities of the network.(Ozyildirim & Avci, 2015; Sarigul et al., 2020; Sarıgül et al., 2019).

3.2.3. Fully Connected Layer

After convolution and pooling, the data becomes a 1D vector for the fully connected network. Hidden layers multiply weights, add bias, and apply an activation function to pass the calculated value to the next layer. These calculations within a neuron in this layer can be observed in the equation(3. 2)

$$fc_1 = f\left(b + \sum_{j=1}^{N} w_{1,j} * O_j\right) \quad \text{Eq. (3. 2)}$$

Where f is the activation function, w is the weight vector, o is the input vector of the jth neuron and b is the bias value.

3.2.4. Data Augmentation

The major reason for data augmentation is to prevent the overfitting situation in neural network training. The operations for data augmentation include resizing the image, changing its orientation, and the lighting condition. These artificially generated data are also included in the training dataset. This leads to a more general feature extraction ability (Hinton et al., 2012).

3.2.5. Dropout

The DropOut method serves as a preventive measure against overfitting in neural networks. It involves randomly deactivating certain neurons within hidden layers based on a dropout probability. Throughout a specific training phase, the activities of chosen neurons are set to zero during both the forward and backward steps. This dynamic approach promotes more robust and reliable learning by ensuring diverse neuron activation in each iteration. While it extends the training duration, this technique contributes to enhanced model generalization(Krizhevsky et al., 2012)

3.3. Pretrained Deep-Learning Structures

The field of convolutional neural networks (CNNs) has witnessed remarkable progress over the years, resulting in a plethora of state-of-the-art architectures. Each of these architectures has contributed to the advancement of computer vision tasks, revolutionizing image recognition, object detection, and other visual processing applications. In this overview, we will delve into some of the

most prominent CNN structures, highlighting their distinctive features, strengths, and contributions to the field.

3.3.1. AlexNet

AlexNet (Krizhevsky et al., 2012) is a pioneering CNN structure that gained substantial attention after its triumphant performance in the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Comprising five convolutional layers followed by max-pooling layers and three fully connected layers, AlexNet introduced several groundbreaking elements to CNN architecture. The use of ReLU (Rectified Linear Unit) activation functions, dropout regularization, and local response normalization were instrumental in enhancing the model's learning capacity. Despite its early successes, AlexNet's design now appears relatively straightforward compared to more recent and sophisticated architectures, limiting its performance on contemporary benchmarks.

3.3.2. VGG

The VGG architecture (Simonyan & Zisserman, 2014) is renowned for its simplicity and uniformity. Employing mostly 3x3 convolutional layers and 2x2 max-pooling layers, VGG stands out for its depth and ability to learn intricate features, thanks to its deeper layers compared to AlexNet. The availability of various configurations, such as VGG-11, VGG-13, VGG-16, and VGG-19, each with a different number of layers, provides flexibility to tailor the model to specific tasks. However, VGG's large number of parameters comes at the cost of computational expenses, making it less feasible for resource-constrained devices.

3.3.3. GoogLeNet (Inception v1)

GoogLeNet (Szegedy et al., 2015), also known as Inception v1, introduced the concept of "Inception" modules, utilizing multiple filter sizes in parallel to capture scale-specific features effectively. Balancing model depth and computation, GoogLeNet emerged as the winner of the ILSVRC 2014 competition. The integration of 1x1 convolutions significantly reduces computational complexity, making the training of deeper networks more feasible.

3.3.4. ResNet (Residual Network)

ResNet (He et al., 2016) is a groundbreaking architecture that introduced the concept of residual connections. By utilizing skip connections to shortcut certain layers, ResNet addressed the vanishing gradient problem, enabling effective training of very deep networks. ResNet's variants, such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, with varying depths, have consistently achieved state-of-the-art results in many computer vision tasks.

3.3.5. DenseNet

DenseNet (G. Huang et al., 2017) is celebrated for its densely connected layers, where each layer receives feature maps from all preceding layers, promoting feature reuse and reducing the number of parameters. Available in variants like DenseNet-121, DenseNet-169, DenseNet-201, and DenseNet-161, each denoting the total number of layers, DenseNet exhibits robust performance, particularly in scenarios with limited training data. The architecture's efficiency in parameter usage has contributed to its popularity in various image recognition tasks.

3.3.6. Xception

Xception (Chollet, 2017) is an extension of the Inception architecture, introducing depthwise separable convolutions that factorize a standard convolution into a depthwise convolution and a 1x1 pointwise convolution. This approach reduces computational complexity while maintaining expressive power, making Xception computationally efficient and well-suited for resource-constrained environments.

3.3.7. SE-ResNet (Squeeze-and-Excitation ResNet)

SE-ResNet (Hu et al., 2018) incorporates "Squeeze-and-Excitation" blocks that adaptively recalibrate the feature maps by learning channel-wise attention weights. This mechanism enhances informative features while suppressing less relevant ones, leading to improved performance. The various variants of SE-ResNet, along with SE-ResNeXt, have demonstrated state-of-the-art results on multiple benchmarks.

3.3.8. NASNet (Neural Architecture Search Network)

NASNet (Zoph et al., 2018) represents a family of neural architectures designed through an automated architecture search process. Notable variants include NASNet-A-Large and NASNet-A-Mobile, optimized for accuracy and efficiency, respectively. NASNet-A-Large achieved impressive performance across various tasks, while NASNet-A-Mobile demonstrated high efficiency without compromising accuracy.

In addition to these state-of-the-art CNN architectures, it is essential to acknowledge the trailblazing contribution of LeNet (LeCun et al., 1998) which marked the inception of CNNs and played a pivotal role in the advancement of computer vision. Furthermore, OverFeat (Sermanet et al., 2013)secured acclaim as the winner of the localization task in the ILSVRC-2013 competition, showcasing its robustness in image recognition tasks and making it a valuable asset for transfer learning applications.

The continuous evolution of CNN architectures exemplifies the ongoing pursuit of advancing deep learning's capabilities. From pioneering designs to the latest automated architecture

search methods, each innovation brings unique insights and contributes to the ever-growing success of CNNs in addressing diverse computer vision challenges.

Efficiency-oriented models, also known as lightweight or low-complexity models, are specialized convolutional neural network (CNN) architectures designed to achieve high performance while minimizing computational and memory requirements. These models are particularly useful for resource-constrained environments, such as mobile devices, embedded systems, and edge computing, where computational efficiency is crucial. Here are some prominent efficiency-oriented models:

3.3.9. MobileNet-v1

MobileNet-v1 (A. G. Howard et al., 2017) introduced depthwise separable convolutions, which split standard convolutions into depthwise and pointwise convolutions. This factorization significantly reduces the number of computations and model sizes, making it ideal for mobile and embedded devices.

MobileNet-v2: MobileNet-v2 (Sandler et al., 2018) is an evolution of MobileNet-v1, further optimizing for efficiency. It introduces inverted residual blocks and linear bottlenecks to improve performance and training stability while maintaining low computational complexity.

3.3.10. ShuffleNet

ShuffleNet (Zhang et al., 2018) employs channel shuffling to reduce computation and memory overhead. This operation promotes feature reuse across channels, making the model efficient for edge devices and resource-limited scenarios.

3.3.11. EfficientNet

EfficientNet (Tan & Le, 2019) is based on a compound scaling method that uniformly scales depth, width, and resolution to optimize model efficiency. It achieves state-of-the-art performance across different resource constraints.

3.3.12. MobileNet-v3

MobileNet-v3 (A. Howard et al., 2019) focuses on improving efficiency by using a combination of strategies, including attentive lightweight depthwise separable convolutions and activation functions like Swish.

3.3.13. FBNet

FBNet (Wu et al., 2019) introduces a novel search space for efficient model design, using a differentiable architecture search method. The model architecture can be optimized for specific efficiency requirements.

3.3.14. MnasNet

MnasNet (Tan et al., 2019) employs a specialized automated architecture search method to design efficient CNN architectures. It achieves competitive accuracy with fewer parameters and reduced computation.

3.3.15. ProxylessNAS

ProxylessNAS (Cai et al., 2018) is another automated architecture search method that explores the architecture space using continuous relaxation. It allows direct training of efficient models without proxy tasks.

These efficiency-oriented models have revolutionized the deployment of deep learning models in real-world applications. By striking a balance between model performance and computational requirements, they have expanded the reach of AI technologies to devices with limited resources. As the demand for edge AI and on-device processing continues to grow, efficiency-oriented models remain at the forefront of research, driving innovations that enable practical and scalable AI solutions for a wide range of applications.

3.4. Differential Convolutional Neural Network (DiffCNN)

In traditional convolutional neural networks (CNNs), Convolution is the key factor of a deep learning structure(Kader et al., 2021; Qu et al., 2020b). This operation is realized by sliding a set of filters over the input image to extract visual patterns. The more feature maps generated by the CNN, the more features the classifier obtains(Qu et al., 2020b). In traditional convolutional neural networks, feature maps are generated via random initialization or transferred knowledge(Qu et al., 2020b). However, the traditional convolutional operation does not take into account the directional changes among a pixel and its neighbors. Which is the main idea behind the Differential Convolution technique(Kader et al., 2021; Qu et al., 2020b; Sarıgül et al., 2019). Unlike standard CNNs, differential CNNs build feature maps utilizing traditional convolution feature maps and a differential operator. After the initial feature map is formed using classic convolution, the Differential Convolution method executes additional convolution operations using predetermined constant filters. These filters are utilized to calculate the differences in activation levels in different directions, resulting in additional feature maps containing signal differences for each direction(Qu et al., 2020b; Sarıgül et al., 2019). These newly created feature maps are then extended with zeros until they reach the size of the initial feature map. This process also expands the filter's receptive field by one unit along two axes. This permits the Differential CNN to obtain more features while keeping the number of trainable parameters constant. This study maintained the same parameters as in (Sarıgül et al., 2019). Starting with the fixed filters as seen in Figure 3.1

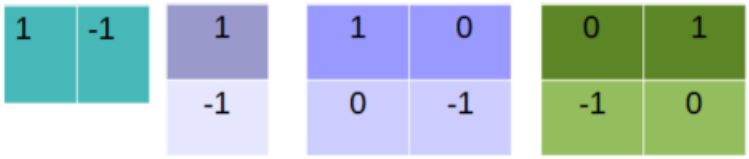

Figure 3.1: Fixed Filters

In addition to fixed filters, the calculation of differential operation took the same assumption as well. Where $g1$ denotes the feature map generated with traditional convolution and $g2$, $g3$, $g4$, and $g5$ signify the extra four feature maps. The values of the neurons of these feature maps are generated as in equations Eq.(3. 3) through Eq.(3. 6) . If the size of $g1$ is assumed as MxN, sizes of the additional maps $g2$, $g3$, $g4$ and $g5$ will be $(M-1)xN, Mx(N-1), (M-1)x(N-1) and\ (M-1)x(N-1)$. Since the size of the feature maps must be the same, these produce additional feature maps are padded with zeros and brought to the size of the first one. Additionally, one unit along two axes with this operation enlarges the receptive field of the filter. Examples of the differential feature maps are given in Figure 3.2

$$g2,i,j = g1,i,j - g1,i+1,j, 1 \le i < M, 1 \le j \le N \quad \text{Eq.(3. 3)}$$

$$g3,i,j = g1,i,j - g1,i,j+1, 1 \le i \le M, 1 \le j < N \quad \text{Eq.(3. 4)}$$

$$g4,i,j = g1,i,j - g1,i+1,j+1, 1 \le i < M, 1 \le j < N \quad \text{Eq.(3. 5)}$$

$$g5,i,j = g1,i+1,j - g1,i,j+1, 1 \le i < M, 1 \le j < N \quad \text{Eq.(3. 6)}$$

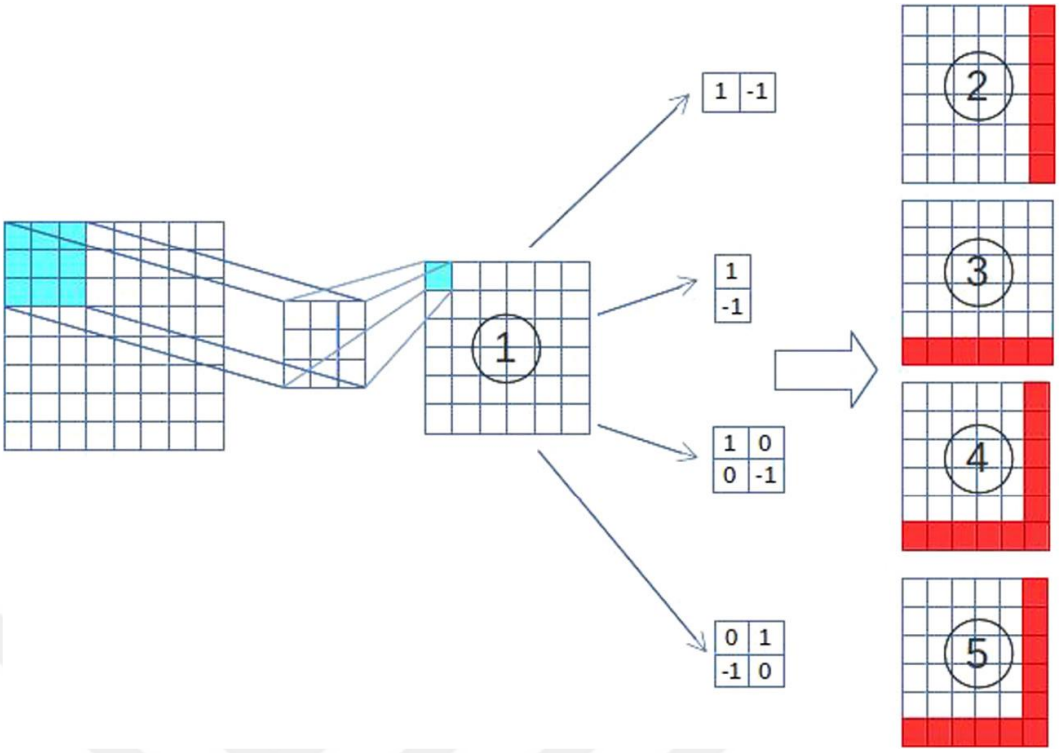

Figure 3.2. Differential Convolution Operation(Sarıgül et al., 2019)

Furthermore, differential convolution generates new feature maps. The generated feature maps are not compatible with the traditional error back-propagation method. This led to an error backpropagation method known as accumulative error backpropagation, which is compatible with the operation of the technique. In the accumulative method, each neuron has received an error fed over the neighbor activations during the backpropagation. Error calculations for the relevant filter are given in equations Eq.(3.7)-Eq.(3. 9).

$$Ei,j = h1,i,j - h2,i,j-1 + h2,i,j - h3,i-1,j + h3,i,j - h4,i-1,j-1 + h4,i,j - h5,i-1,j + h5,i,j-1 \quad \text{Eq.(3.7)}$$

Where i > 1, j > 1, i < M, and j < N. As is seen in equation Eq.(3.7) neurons neither in the corner nor in the edges take error feeds from all neighborhood neurons. Corner neurons of the map take error feeds from 3 neighborhood neurons as in Equation Eq.(3. 8)

$$Ei,j = \begin{cases} \mathrm{h1,i,j} + \mathrm{h2,i,j} + \mathrm{h3,i,j} + \mathrm{h4,i,j} & \mathrm{i} = 1, \mathrm{j} = 1 \\ \mathrm{h1,i,j} - \mathrm{h2,i,j-1} + \mathrm{h3,i,j} + \mathrm{h5,i,j-1} & \mathrm{i} = 1, \mathrm{j} = \mathrm{N} \\ \mathrm{h1,i,j} + \mathrm{h2,i,j} - \mathrm{h3,i-1,j} - \mathrm{h5,i-1,j} & \mathrm{i} = \mathrm{M}, \mathrm{j} = 1 \\ \mathrm{h1,i,j} - \mathrm{h2,i,j-1} - \mathrm{h3,i-1,j} - \mathrm{h4,i-1,j-1} & \mathrm{i} = \mathrm{M}, \mathrm{j} = \mathrm{N} \end{cases} \quad \text{Eq.(3. 8)}$$

Errors to be propagated to the edge neurons of the map are calculated as in Eq.(3. 9). Edge neurons receive error feeds from 5 neighborhood neurons.

$$Ei,j = \begin{cases} \mathrm{h1,i,j} - \mathrm{h2,i,j-1} + \mathrm{h2,i,j} + \mathrm{h3,i,j} + \mathrm{h4,i,j} + \mathrm{h5,i,j-1} \\ \quad \mathrm{i} = 1, 1 < \mathrm{j} < \mathrm{N} \\ \mathrm{h1,i,j} - \mathrm{h2,i,j-1} + \mathrm{h2,i,j} - \mathrm{h3,i-1,j} - \mathrm{h4,i-1,j-1} - \mathrm{h5,i-1,j} \\ \quad \mathrm{i} = \mathrm{M}, 1 < \mathrm{j} < \mathrm{N} \\ \mathrm{h1,i,j} + \mathrm{h2,i,j} + \mathrm{h3,i,j} - \mathrm{h3,i-1,j} + \mathrm{h4,i,j} - \mathrm{h5,i-1,j} \\ \quad 1 < \mathrm{i} < \mathrm{M}, \mathrm{j} = 1 \\ \mathrm{h1,i,j} - \mathrm{h2,i,j-1} + \mathrm{h3,i,j} - \mathrm{h3,i-1,j} - \mathrm{h4,i-1,j-1} + \mathrm{h5,i,j-1} \\ 1 < \mathrm{i} < \mathrm{M}, \mathrm{j} = \mathrm{N} \end{cases} \quad \text{Eq.(3. 9)}$$

3.5. Logarithmic Differential Convolutional Neural Network (LDiffCNN)

The idea behind the Logarithmic Learning Differential Convolutional Neural Network is directly related to the drawbacks of differential convolution. Simultaneous multi-directional execution of additional constant valued filters for neighbor pixels leads to additional time consumption which dominates especially during training. Training a convolutional neural network with a large amount of data requires faster learning methods to be useful. The differential convolutional neural network with logarithmic learning overcomes this time consumption by realizing faster convergence. The logarithm function with exponential term-based representation of the data makes faster convergence for minimal error. The applied logarithmic approaches in the literature utilize facts such as maximum likelihood estimation(MLE)(Abdelli et al., 2023; Shalileh, 2023; S. Wang & Gui, 2020). Since MLE can be adapted to classical neural network classifiers, the classifier part of CNN can utilize MLE to realize the logarithmic cost function.

In image classification, the objective is to classify images into different predefined categories or classes. Assume a dataset with *N* training images, where each image i is represented by a feature vector x_i and its corresponding class label y_i.

It can be assumed that the class labels y_i drawn from a probability distribution parameterized by θ, and we denote this distribution as $P(y_i|\boldsymbol{x_i};\theta)$. The likelihood function, denoted as $L(\theta)$, measures the probability of observing the given set of training images, given the model parameters θ. Computed as the product of the individual probabilities of each training image:

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} P(y_i|\boldsymbol{x_i};\theta) \quad \text{Eq.(3. 10)}$$

To estimate the model parameters θ, we seek to maximize the likelihood function. In practice, it is more convenient to maximize the log-likelihood, denoted as $L(\theta)$, and obtained by taking the logarithm of the likelihood function:

$$\mathcal{L}(\theta) = \log L(\theta) \quad \text{Eq.(3. 11)}$$

By maximizing the log-likelihood, we find the optimal values of the model parameters that make the observed training data most likely.

During the training phase, optimization algorithms such as stochastic gradient descent (SGD) are used to iteratively update the model parameters θ by taking steps in the direction of the gradient of the log-likelihood function with respect to θ.

$$\theta_j = \theta_j + lr\frac{\partial l(\theta)}{\partial \theta_j} \quad \text{Eq.(3. 12)}$$

Here, θ_j represents the jth parameter, lr denotes the learning rate, and $\frac{\partial l(\theta)}{\partial \theta_j}$ signifies the partial derivative of the log-likelihood function with respect to θ_j .

Once the image classifier is trained, it can be utilized to classify new images by computing the probabilities of each class for the input image and selecting the class with the highest probability. Furthermore, Logarithms enhance machine learning convergence through feature scaling, simplifying loss functions, stabilizing probabilities, compressing information, and aiding regularization. Scaling features with logarithms compresses their range, addressing convergence issues caused by magnitude differences. Applying logarithms to loss functions results in smoother gradients and faster convergence. Taking logarithms of probabilities stabilizes calculations and aids multi-class classification convergence. Logarithms compress information, focusing on relevant data, reducing outliers' impact, and noise for efficient convergence. In regularization, logarithmic transformations facilitate smoother penalty terms, promoting parameter sparsity and improved convergence while preventing overfitting. These benefits make logarithms invaluable for optimizing machine learning models. As depicted in Figure 3.3 through Figure 3.6.

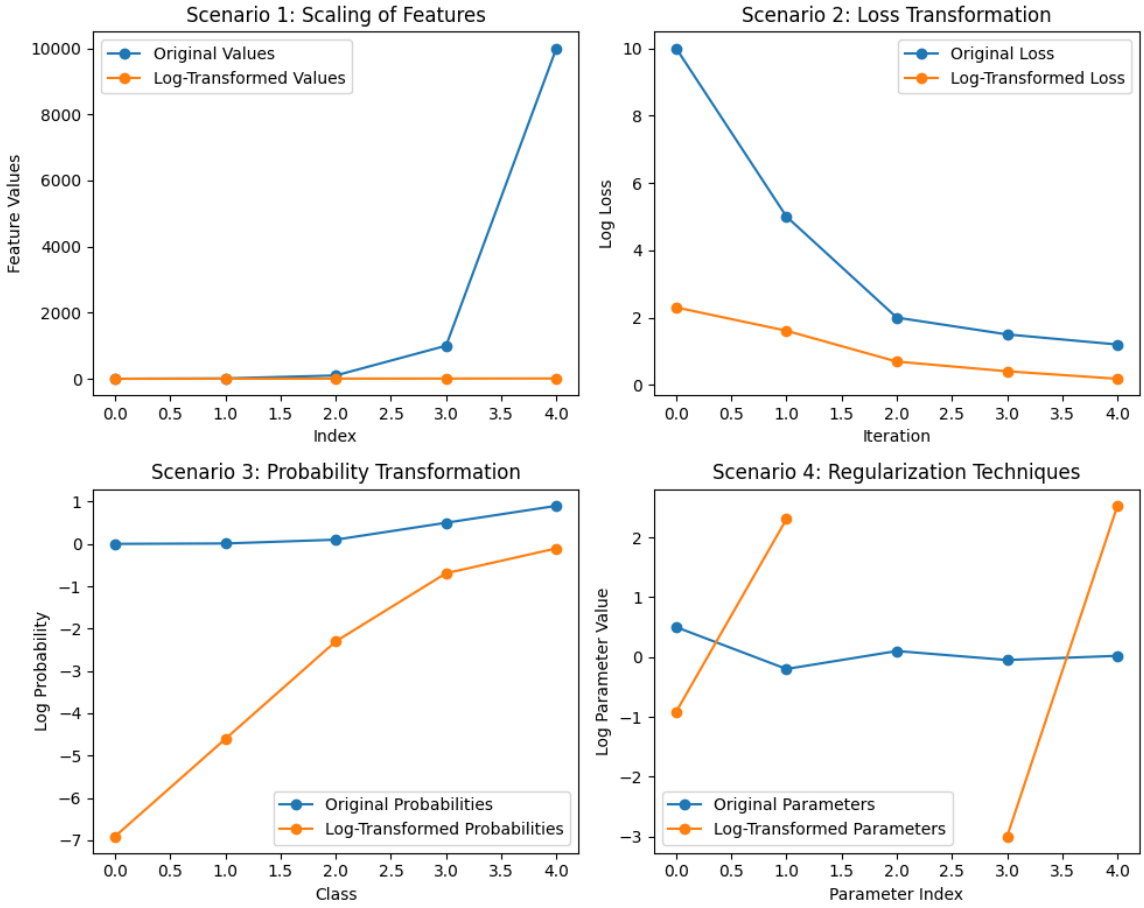

Figure 3.3:Log Transformations

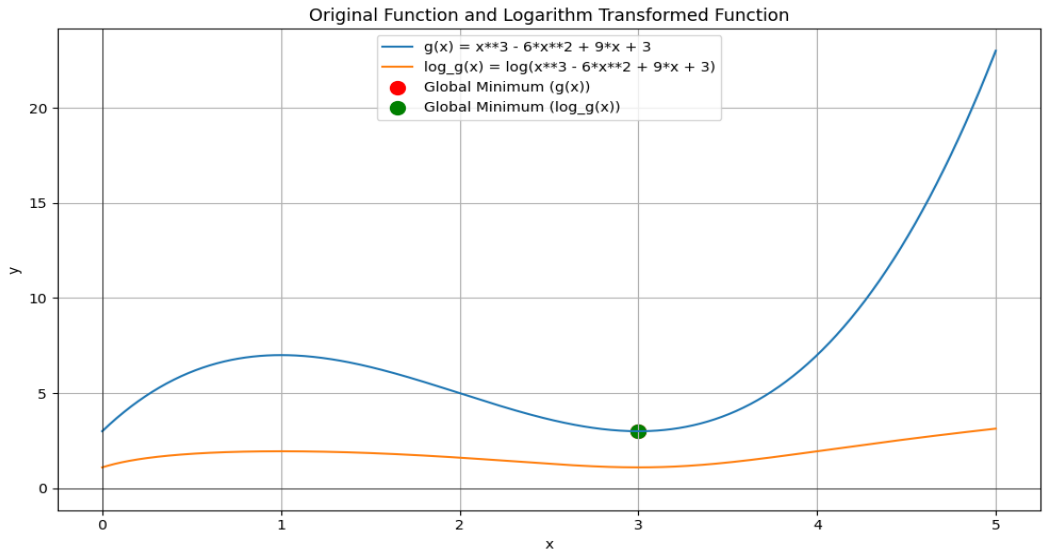

Figure 3.4: Log Transformation on cubic equation for global and local minima

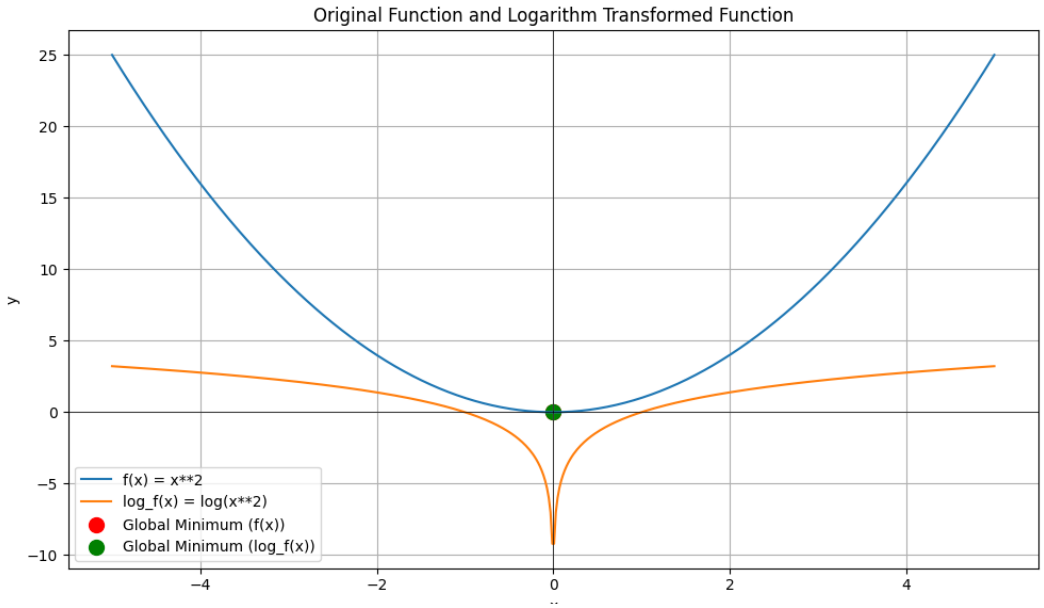

Figure 3.5:Log Transformation on Quadratic Equation for Local and global minima

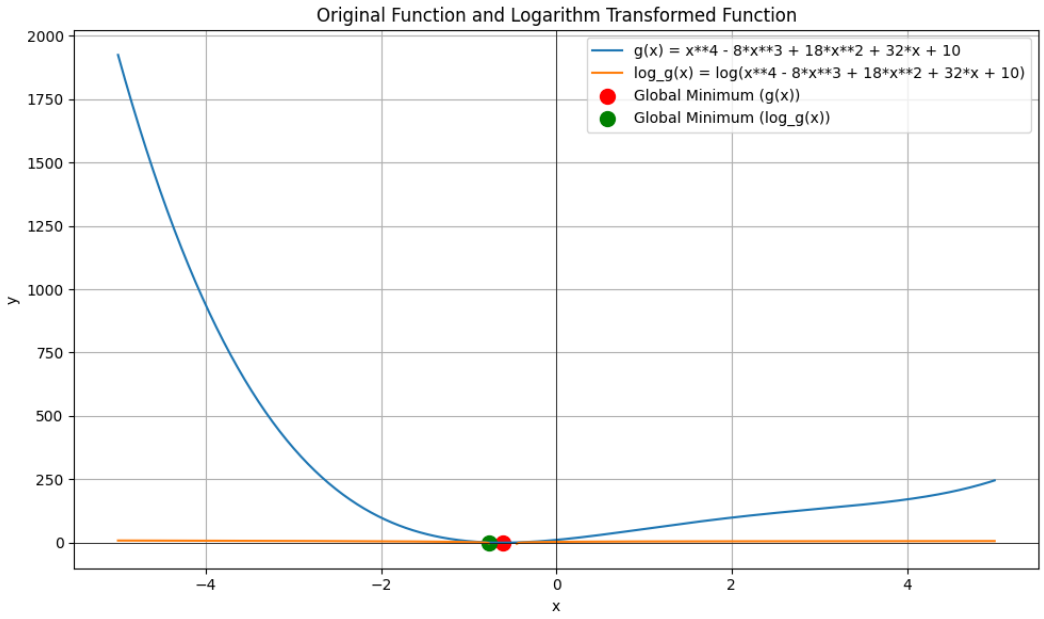

Figure 3.6:Log Transformation on Quartic Equation Global and local minima

In addition, during the optimization process, a loss function is typically minimized, often employing the cross-entropy loss. This loss quantifies the difference between the predicted class Once the image classifier is trained, it can be utilized to classify new images by computing the probabilities of each class for the input image and selecting the class with the highest probability. Moreover, a loss function is typically minimized during the optimization process, often employing the cross-entropy loss. This loss quantifies the difference between the predicted class probabilities and the true class labels. However, in our case, the error minimization is based on logarithmic learning, which is defined by the following equation:

$$e = \mathrm{d}.\ln(\mathrm{y}) + (1 - \mathrm{d}).\ln(1 - \mathrm{y}) \qquad \text{Eq.(3. 13)}$$

Here, e defines the cost function, d denotes the target values of the training input data, and y denotes the output of the network. To find weight value changes, the derivative of the MSE function is obtained as given in

Eq.(3. 14)

$$f'(net) = -(d/y - (\frac{1-d}{1-y})(y - y^2) \qquad \text{Eq.(3. 14)}$$

3.5.1. AND /OR Implementation

Since one of the main contributions of this work is on the logarithmic cost, it is applied to a single neuron to solve logical AND functions. This numerical example is given to demonstrate faster convergence efficiency of the proposed approach, with respect to a standard single gradient descent-based learning. This example also gives practical implementation ideas for the utilization of the proposed logarithmic cost function. The proposed and gradient descent-based error minimization implementations are done on the single neuron shown in Figure 3. 7

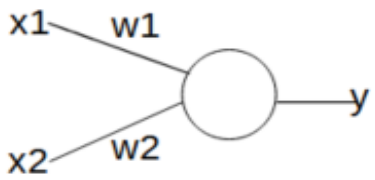

Figure 3. 7:Single Neuron

Where x_1 and x_2 are inputs, w_1 and w_2 are weights, and y is the output. The operation done by the neuron is given in Eq(3. 15)

$$y = f\,(w_1 \cdot x_1 + w_2 \cdot x_2 + b) \qquad \text{Eq(3. 15)}$$

where f is the activation function and b is the bias term. In this application, the logarithmic sigmoid (logsig) activation function is used. The target data for logical AND and OR functions are given in Table 3.1 and Table 3.2

Table 3.1:Truth table of AND Logical Operation

X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.2: Truth table of OR Logical Operation

X_1	X_2	X_1 OR X_2
0	0	0
0	1	1
1	0	1
1	1	1

All the implementations for both approaches are done in the MATLAB environment with the same initial weights and bias values. The performance evaluation is on the epoch versus MSE minimization. Weight value updates are done in batch mode. The following pseudocodes given

Step 1: Randomly initialize w and b
Step 2: $Initialize\ learning\ rate\ , epoch\ , MSE$
Step 3:$While(MS\ E\ >\ error\ goal)$**do**
Step 4: Calculate network output:

- $net\ =\ w\ \cdot\ x$
- $y\ =\ logsig(net)$

Step 5: Compute error:

- $e\ =\ d\ \cdot\ ln(y)\ +\ (1-\ d)\ \cdot\ ln(1-\ y)$

Step 6: Compute the derivative of the activation function:

- $f\ '(net)\ =\ -(d/y\ -\ (1\ -\ d)/(1\ -\ y))(y\ -y^2\ )$

Step 7: Compute Mean Squared Error (MSE):

- $MSE\ =\frac{1}{k}\sum(e^2)$

Step 8: Update the weights and bias:

- $w_{new}\ =\ w_{old}\ +\ lr\ \cdot\ e\ \cdot\ f\ '(net)\ \cdot\ x$
- $epoch\ =\ epoch\ +\ 1$

Step 9: Loop back to Step 3

Algorithm 1. Proposed Logarithmic Cost Function

1: **Step 1**: $Randomly\ initialize\ w\ and\ b$
2: **Step 2:** $Initialize\ Set\ learning\ rate\ , epoch\ , MSE$
3: **Step 3:**$While(MS\ E\ >\ error\ goal)\boldsymbol{do}$
4: **Step 4**: $Calculate\ network\ output$:
- $net\ =\ w\ \cdot\ x$
- $y\ =\ logsig(net)$

5: **Step 5**: $Compute\ error$:
- $e\ =\ d\ -\ y$

6: **Step 6**: $Compute\ derivative\ of\ the\ activation\ function$:
- $f\ '(net)\ =\ y\ -\ y2$

7: **Step 7**: $Compute\ Mean\ Squared\ Error\ (MSE)$:
- $MSE\ =\ \frac{1}{k}\sum(e^2)$

8: **Step 8**: $Update\ the\ weights\ and\ bias$:
- $w_{new}\ =\ w_{\ old}+\ lr\ \cdot\ e\ \cdot\ f\ '(net)\ \cdot\ x$
- $epoch\ =\ epoch\ +\ 1$

9: **Step 9**: Loop back to Step 3

Algorithm 2. Pseudocode for Gradient Descent

Table 3.3:Numerical Example for AND Operation

Epoch	MSE Proposed	MSE Gradient Descent
1	2.076101	0.477024
2	3.682022	0.402537
3	6.445563	0.316165
4	10.817582	0.239828
5	15.121984	0.193743
6	19.098013	0.171881
7	19.382854	0.160543
8	14.141750	0.153114
9	9 .456446	0.147279
10	0.00182	0.142223

Initial values for the neuron are:
b = 0.824376
w = [0.982663 0.730249]
The learning rate (lr) is 0.9, and the target MSE is 0.1.
After 10 epochs, the proposed cost function reaches the target MSE goal.

After 10 epochs, the weight values of the proposed approach are:
b = −11.999987
w = [7.22332 7.223319]

The gradient descent weight update results:
b = −1.234552
w = [0.69043 0.51228]

The outputs are:
$y_{proposed}$ = [0 0.98678 0.68464 0.06078]
$y_{gradient\ descent}$ = [0.2376 0.3312 0.3748 0.4879]

Although monotonic MSE decrease observed for the gradient descent which keeps it around a local minima, the proposed method has reached a much better minimum error point. This very effective decrease of MSE is always observed through the datasets as in this AND example

Table 3.4: Numerical Example for OR Operation

Epoch	MSE Proposed	MSE Gradient Descent
1	0.46151	0.16143
2	0.28836	0.15771
3	0.24654	0.15401
4	0.21611	0.15032
5	0.19281	0.14663
6	0.17435	0.14295
7	0.15930	0.13929
8	0.14676	0.13566
9	0.13613	0.13206
10	0.12697	0.12852
11	0.11900	0.12504
12	0.11198	0.12162
13	0.10574	0.11829
14	0.10017	0.11504
15	0.09514	0.11188

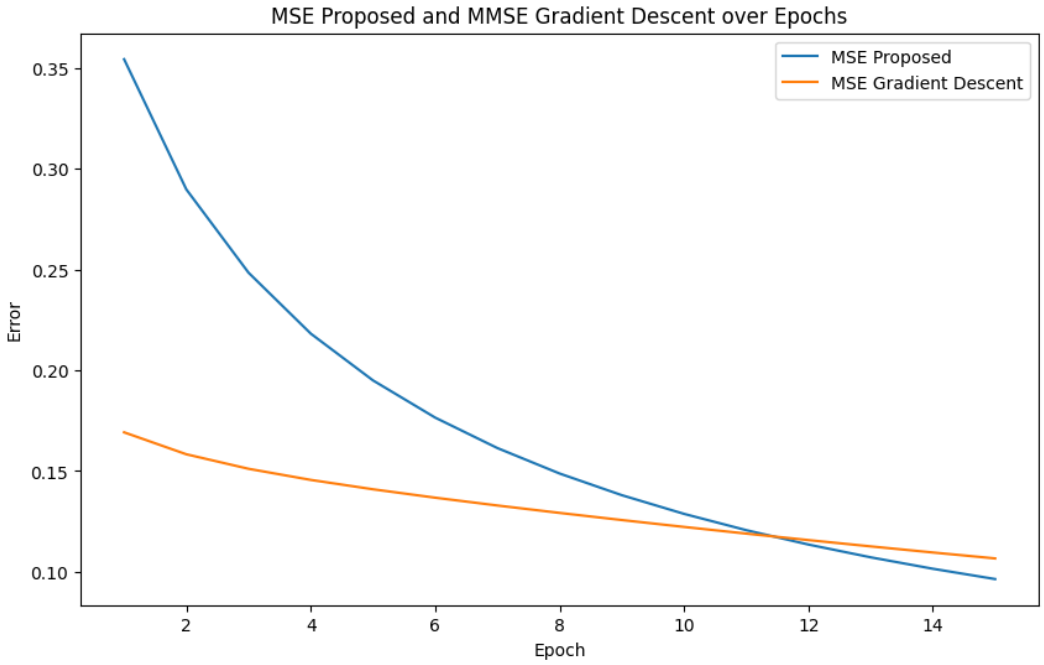

Figure 3.8: MSE and MSEgd

3.5.2. LogRelu Activation Function

The LogRelu activation function is a modification of the ReLU function, addressing its limitations. While ReLU returns 0 for negative inputs and the input value for positive inputs, LogRelu ensures a non-zero output for inputs close to zero. It achieves this through a logarithmic transformation using the LOG1P function. This function improves numerical stability, particularly in image and text processing scenarios involving data normalization and probability computations. LogRelu handles small probabilities by converting them into log probabilities, avoiding underflow issues.

It is smooth, continuous, and differentiable, making it suitable for training deep neural networks and mitigating the problem of dead neurons associated with ReLU as seen in **Eq.(3. 16)**.

$$y = \log1p(x), \text{LogRelu}(x) = \begin{cases} y, & x < 0 \\ x, & \text{otherwise} \end{cases} \quad \text{Eq.(3. 16)}$$

3.5.3. Implemented CNN Structures

Table 3.5: Network Structures

Layer No	CNN	LCNN1	LCNN3	DIFFCNN	LDIFFCNN1	LDIFFCNN2
Input Layer	3x32x32 (RGB)/ Input size	3x32x32 (RGB)/Input size	3x32x32 (RGB)/Input size	3x32x32(RGB)/Input size	3x32x32(RGB) /Input size	3x32x32(RGB) /Input size
2	5x5 sized 32 filters , ,padding=2, stride=1	5x5 sized 32 filters , ,padding=2, stride=1	5x5 sized 32 filters , ,padding=2, stride=1	DiffCNN	DiffCNN	DiffCNN
3	2x2 max-pooling	2x2 max-pooling	2x2 max-pooling	5x5 sized 15 filters , ,padding=2, stride=1	5x5 sized 15 filters , ,padding=2, stride=1	5x5 sized 15 filters , ,padding=2, stride=1
4	5x5 sized 64 filters , ,padding=2, stride=1	5x5 sized 64 filters , ,padding=2, stride=1	5x5 sized 64 filters , ,padding=2, stride=1	5x5 sized 64 filters , ,padding=2, stride=1	5x5 sized 64 filters , ,padding=2, stride=1	5x5 sized 64 filters , ,padding=2, stride=1
5	2x2 max-pooling	2x2 max-pooling	2x2 max-pooling	2x2 max-pooling	2x2 max-pooling	2x2 max-pooling
6	3x3 sized 128 filters with no stride	3x3 sized 128 filters with no stride	3x3 sized 128 filters with no stride	3x3 sized 128 filters with no stride	3x3 sized 128 filters with no stride	3x3 sized 128 filters with no stride
7	Flatten	Flatten	Flatten	2x2 max-pooling	2x2 max-pooling	2x2 max-pooling
8	FC(flattened value, 128)	FC(flattened value, 128)	FC(flattened value, 128)	3x3 sized 256 filters with no stride	3x3 sized 256 filters with no stride	3x3 sized 256 filters with no stride
9	Dropout	Dropout	Dropout	Flatten	Flatten	Flatten
10	FC(128, 256)	FC(128, 256)	FC(128, 256)	FC(flattened value, 128)	FC(flattened value, 128)	FC(flattened value, 128)
11	Dropout	Dropout	Dropout	Dropout	Dropout	Dropout
12	(256, No-classes)	(256, No-classes)	(256, No-classes)	FC(128,256)	FC(128, 256)	FC(128, 256)
13	Output	Output	Output	Dropout	Dropout	Dropout
14				(256,No-classes)	(256,No-classes)	(256,No-classes)
15				Output	Output	Output

The CNN, LCNN1, and LCNN2 architectures all feature an input size of 32x32 pixels. They employ kernel sizes of 5x5 for the initial two layers and 3x3 for the final layer of feature extraction. Filter sizes of 32, 64, and 128 are used during the feature extraction phase, while sizes of 128, 256, and the total number of input classes are employed for classification. Additionally, a default dropout rate of 0.5 is applied between classification layers. To maintain consistent dimensions across all layers, a padding of two is applied to the first two layers, and a stride of one is used. For feature extraction, max pooling of 2x2 is utilized in layers 3 and 4.

The distinguishing feature among these three structures lies in their activation functions. The CNN employs the Relu activation function for both feature extraction and classification. However, LCNN1 uses the LogRelu activation in its classification phase instead of Relu. LCNN2, on the other hand, utilizes LogRelu for both feature extraction and classification.

Regarding the DiffCNN, LDiffCNN1, and LDiffCNN2 architectures, the initial layer employs Differential convolution followed by a standard convolutional layer. Notably, neither of these layers employs any activation functions. This Differential convolution introduces an additional layer and alters the filter sizes to 15, 64, 128, and 256 for the feature extraction layers. In terms of activation functions, DiffCNN utilizes the Relu activation for both feature extraction and classification. LDiffCNN1 employs LogRelu in its classification phase instead of Relu. In contrast, LDiffCNN2 uses LogRelu for both feature extraction and classification. However, the classification phase retains the same structure as the preceding three architectures. This configuration is detailed in Table 3.5 and visually depicted in Figure 3.9 through Figure 3.14.

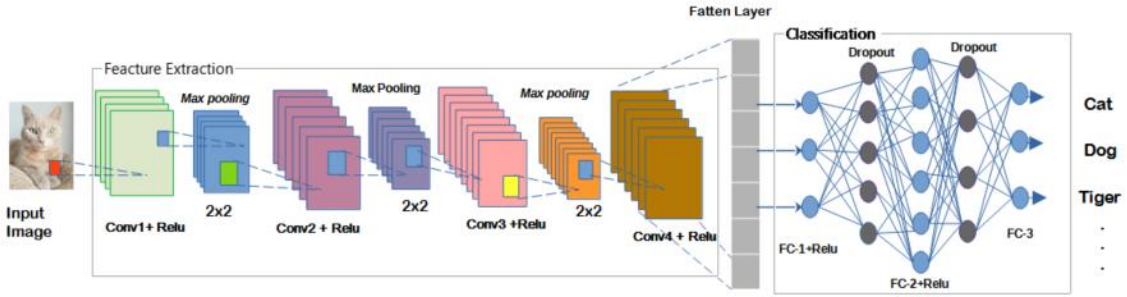

Figure 3.9: CNN Structure

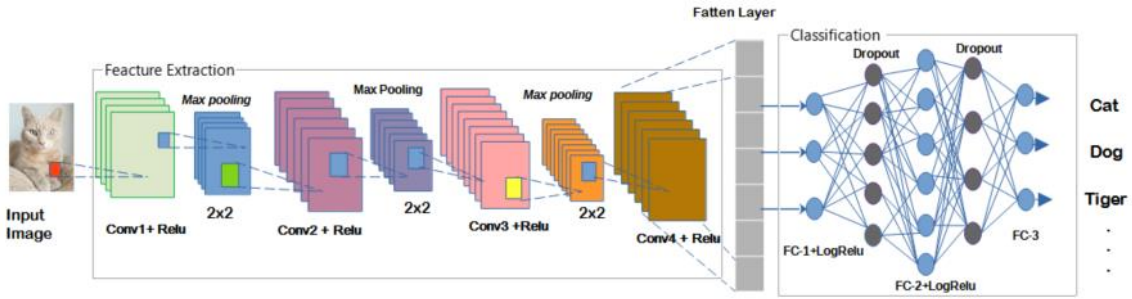

Figure 3.10: LCNN1 Structure

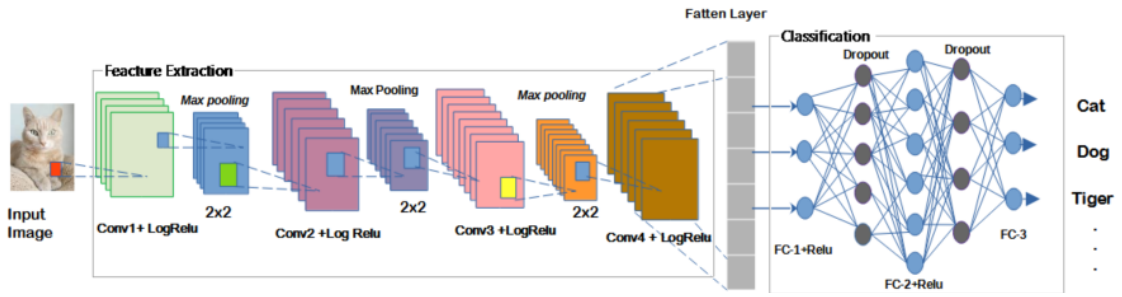

Figure 3.11: LCNN2 Structure

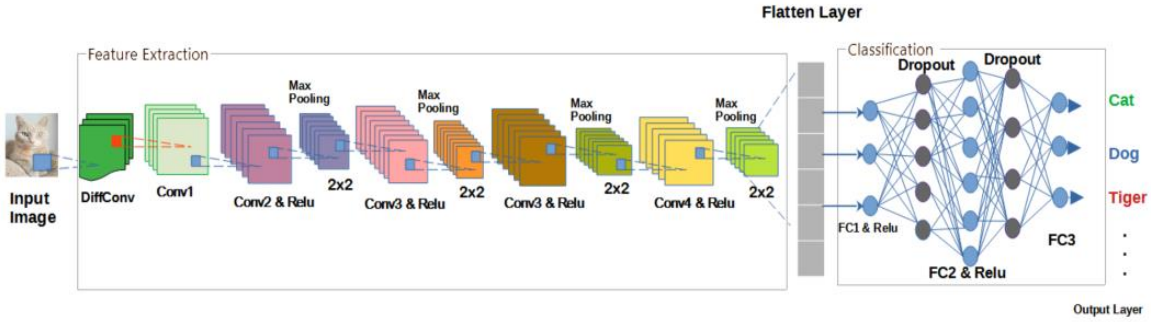

Figure 3.12:DiffCNN Structure

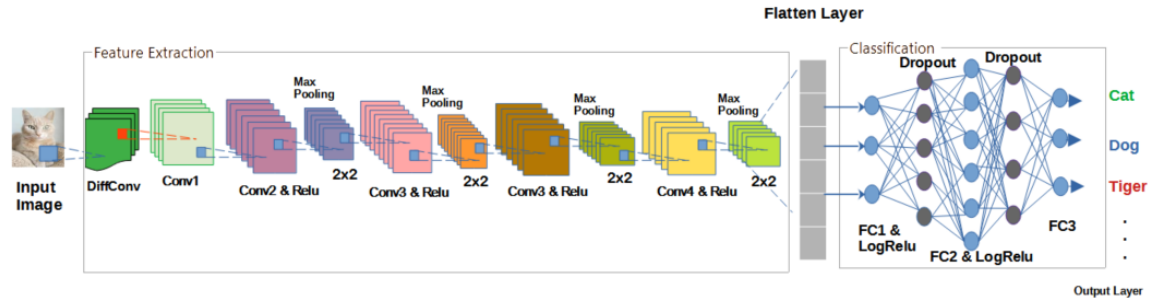

Figure 3.13:LDiffCNN1 Structure

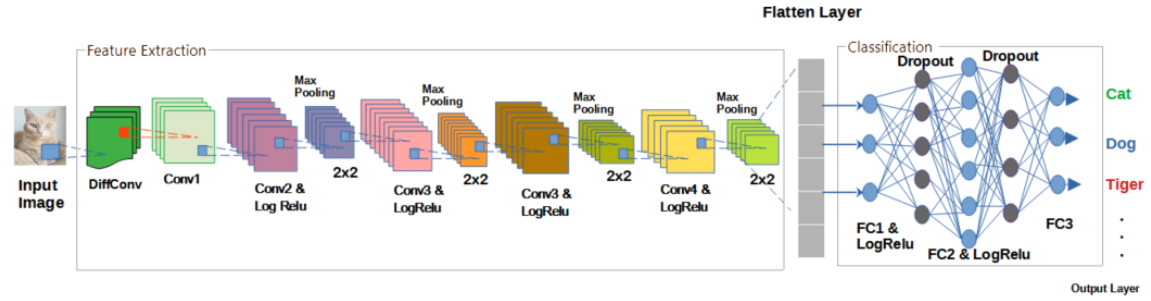

Figure 3.14:LDiffCNN2 Structure

3.6. Datasets

The models were tested on seven distinct datasets, each with specific characteristics. CIFAR-10(Krizhevsky & Hinton, 2009) contained 60,000 32x32 images in 10 classes. Animal Faces(Choi et al., 2020) had 16,130 images across three domains. Vegetable Image (Ahmed et al., 2021)had 21,000 images in 15 classes. 10-Monkey Species ((Utkarsh Saxena, 2022a) had nearly 1,400 images. The dataset of Tomato Leaves(Motwani, 2021) had over 20,000 labeled images in 11 classes. Flower Classification (Cengıl & Çinar, 2019) had 10 classes, Intel scenes had around 25,000 images, and the MNIST dataset(LeCun et al., 2010). CUB-200-2011 (Wah et al., 2011) had around 11,788 images, and Stanford Cars(Krause et al., 2013) with around 6,185 images having 196 categories.

These diverse datasets facilitated comprehensive testing across various image classes and domains. The class distribution of the datasets is given in Table 3.6.

Table 3.6: Class Distribution of Datasets

Dataset Name	Number of Images	Number of Classes
CIFAR-10	60000	10
Vegetable Image	21000	15
Animal Faces	16130	03
10 Monkey Species	1400	10
Flower Classification	2000	10
Intel scenes dataset	25000	6
Tomato Leaves	20000	11
MNIST Dataset	60000	10
CUB-200-2011	11,788	200
Stanford Cars	16,185	196

3.6.1. Cifar-10 Dataset

The CIFAR-10 dataset (Krizhevsky & Hinton, 2009), is a very popular image dataset containing 60000 32x32 images divided into 10 different classes.

Table 3.7: CIFAR-10 Dataset Contents

Class Name	Number of Images
Airplane	6000
Automobile	6000
Bird	6000
Cat	6000
Deer	6000
Dog	6000
Frog	6000
Horse	6000
Ship	6000
Truck	6000

3.6.2. Animal Faces Dataset

The Animal Faces dataset (Choi et al., 2020). Also known as Animal Faces-HQ (AFHQ), consists of 16,130 high-quality images at 512×512 resolution divided into three (3) domains of classes each providing about 5000 images.

Table 3.8: Animal Faces Dataset Details

Class Name	Number of Images
Cat	5000
Dog	5000
Wildlife	5000

3.6.3. Vegetable Image Dataset

The Vegetable Image dataset (Ahmed et al., 2021) has 15 types of common vegetables found worldwide. A total of 21000 images from 15 classes are used where each class contains 1400 images of size 224×224 and in jpg format.

Table 3.9: Vegetable Image Dataset Details

Class Name	Number of Images
Bean,	1400
Bitter gourd	1400
Bottle gourd	1400
Brinjal	1400
Broccoli	1400
Cabbage	1400
Capsicum	1400
Carrot	1400
Cauliflower	1400
Cucumber	1400
Papaya	1400
Potato	1400
Pumpkin	1400
Radish	1400
Tomato	1400

3.6.4. 10- Monkey Species.

10-Monkey Species (Utkarsh Saxena, 2022a). These images are of dimensions 400x300 or larger, and they are saved in JPEG format, totaling approximately 1400 images.

Table 3.10: 10-Monkey Species Dataset Details

Class Name	Number of Images
Bald Uakari	157
Emperor Tamarin	167
Golden Monkey	1000
Gray Langur	182
Hamadryas Baboon	157
Mandril	169
Proboscis Monkey	158
Red Howler	170
Vervet Monkey	160
White Faced Saki	158

3.6.5. Flower Classification

The Flower Classification Dataset(Utkarsh Saxena, 2022b), has 10 Classes. All classes are updated to have 1,500 Training Images, 500 Validation Images, and a random Number of Testing Images.

Table 3.11:Flower Classification Dataset Details

Class Name	Number of Images
Aster	20146
Daisy	20260
Iris	20317
Lavender	20217
Lily	20350
Marigold	20108
Orchid	20202
Poppy	20158
Rose	20269
Sunflower	20328

3.6.6. Intel Image Scene Classification

This dataset comprises approximately 25,000 images, each sized 150x150 pixels, classified into six distinct categories: buildings, forest, glacier, mountain, sea, and street. The data is divided into three separate sets stored in individual zip files. The Train set contains around 14,000 images, the Test set comprises approximately 3,000 images, and the Prediction set includes about 7,000 images (Bansal, n.d.).

Table 3.12: Intel Image Scene Classification Dataset Details

Class Name	Number of Images
Buildings	2628
Forest	2745
Glacier	2957
Mountain	3037
Sea	2784
Street	2883

3.6.7. Tomato Leaf Dataset.

This dataset contains more than 20,000 images of tomato leaves, encompassing 10 different diseases as well as a healthy class categorized into 11 classes (Motwani, 2021).

Table 3.13:Tomato Leaf Dataset Details

Class Name	Number of Images
Bacterial spot	3558
Early blight	3098
Healthy	3857
Late blight	3905
Leaf Mold	3493
Septoria leaf spot	3628
Target Spot	2284
Tomato mosaic virus	2737
Tomato yellow leaf curl virus	2537
Two-spotted spider mite	2182
Powdery Mildew	1256

3.6.8. MNIST Dataset

The MNIST dataset (Modified National Institute of Standards and Technology database) is one of the most popular datasets in machine learning. It has a training set of 60,000 examples and a test set of 10,000 examples with the size of square 28×28 pixel images of handwritten single digits between 0 and 9. The images are in grayscale format(LeCun et al., 2010).

3.6.9. Caltech-UCSD Birds-200-2011 (CUB-200-2011)

The Caltech-UCSD Birds-200-2011 (CUB-200-2011) dataset is the most widely used dataset for fine-grained visual categorization tasks. It contains 11,788 images of 200 subcategories belonging to birds, 5,994 for training and 5,794 for testing. (Wah et al., 2011).

3.6.10. Stanford Cars

The Stanford Cars dataset encompasses 196 car classes, comprising a collective of 16,185 rear-view images. The dataset is nearly evenly split between training and testing sets, with 8,144 images designated for training and 8,041 for testing. These categories generally categorize cars by Make, Model, and Year. Each image has dimensions of 360×240 pixels(Krause et al., 2013).

3.7. Hardware and Libraries

3.7.1. Hardware

The proposed deep learning implementations were developed using a remote machine, and Laptop computers that have the following features:

- Laptop computer 1: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz, Windows 10 Operating System.

- Remote Machine (Google's free cloud service for AI developers): Google Colab with both paid and free T4 GPU

3.7.2. Libraries

Libraries play a pivotal role in Python's ecosystem, each offering unique capabilities to empower developers and data scientists in their respective domains. In this study libraries utilized the following

- Matplotlib(Hunter, 2007) stands out as a versatile visualization library, simplifying the creation of static, animated, and interactive visualizations.
- NumPy(Harris et al., 2020) serves as the backbone of scientific computing, providing multidimensional arrays and a wide range of operations crucial for numerical computation.
- PyTorch(Paszke et al., 2019), an open-source gem, excels in deep learning with its tensor computations and GPU acceleration.
- Scikit-learn(Pedregosa et al., 2011), often referred to as Sklearn, is the go-to library for machine learning tasks, offering a consistent interface for classification, regression, clustering, and dimensionality reduction.
- Finally, Pandas(McKinney, 2010) is an indispensable tool for data science and analysis, building on NumPy's foundation to provide powerful data manipulation capabilities.

3.8. Experiment Details

Table 3.14:Overview of Model Configurations.

Model/Hyper-parameter	Value/Range
Epochs	05-250
Batch size	32,64,224
Learning rate	3e-4, 5e-4 , 1e-3 and 1e-2
Optimizer	Stochastic Gradient Descent (SGD), RMSprop and Adam
Loss function/ Criterion	Cross-Entropy and Logarithmic Cost Function
Input shape	32x32,64x64 and 224x224
Pooling	Max pooling
Activation	ReLU, LeakyReLU, ELU, SELU, Mish and LogRelu

3.9. Evaluation Metrics

When evaluating a model's classification capability, various standard error metrics come into play. In this particular research, we employ four specific metrics for assessment: accuracy, precision, recall, and the F1-score. Accuracy, being the most commonly used metric, measures the ratio of

correctly predicted instances to the total instances in a dataset. Relying on a single metric can be misleading. For instance, a high accuracy combined with low precision implies a significant number of false positives, indicating inaccurate positive predictions. Therefore, using precision and recall in conjunction provides a more holistic view of the model's efficacy. This multi-metric approach underscores the importance of employing these four metrics. Additionally, both Top 1 and Top 5 accuracy were incorporated into the evaluation process.

3.9.1. Accuracy

The accuracy value can be seen in Equation **Hata! Başvuru kaynağı bulunamadı.**, where t is the number of correctly classified instances and m is the number of instances in a dataset.

$$Accuracy = \frac{t}{m} \quad \text{Eq.(3.17)}$$

3.9.2. Precision

Precision is the ratio of correctly classified instances among all instances in a class. The precision value of class c, Pc, can be seen in Equation **Hata! Başvuru kaynağı bulunamadı.**, where mc is the total number of instances in class c, and tc is the number of correctly classified instances in class c.

$$Pc = \frac{t_c}{m_c} \quad \text{Eq.(3. 18)}$$

3.9.3. Recall

The recall is the ratio of correctly classified instances among all instances classified in the corresponding class. The recall value of class c, Rc, is shown in EquationEq.(3. 19, where lc is the total number of instances classified as belonging to class c.

$$Rc = \frac{t_c}{l_c} \quad \text{Eq.(3. 19)}$$

3.9.4. F1-Score

The F1-score (F1c) is a combination of precision and recall measures. It is computed using EquationEq.(3. 20), where Pc is the precision of class c and Rc is the recall of class c.

$$F1_c = \frac{2 * (p_c * R_c}{P_c + R_c} \quad \text{Eq.(3. 20)}$$

4. RESULTS AND DISCUSSIONS

In this section, we present a detailed discussion of all experiments conducted, along with their respective results

4.1. Experimental Results

The results of the first experiment sets indicate that LDiffCNN1 achieved the highest accuracy scores for the Animal Faces dataset, with results of 98.20%, 98.5%, and 98.73% for different learning rates as presented in Table 4.1, Table 4.2, and Table 4.3, supported by visual representations in Figure 4.1 through Figure 4.6.

For the vegetable dataset, LDiffCNN1 consistently demonstrated the best overall performance across various metrics, achieving accuracy scores ranging from 98.80% to 99.00%. DiffCNN also performed well, closely following LDiffCNN1 in accuracy and F1-score. CNN and LCNN1 showed moderate performance with slightly lower accuracies, while LCNN2 consistently had the lowest accuracy. LDiffCNN2 fell between the top-performing models and LCNN2 in terms of performance, as observed in Table 4. 4, Table 4. 5, and Table 4. 6, visually presented in Figure 4.7 through Figure 4.12. When evaluating the models on the 10-Monkey Species dataset (Table 4. 7, Table 4. 8, and Table 4. 9), DiffCNN achieved the highest accuracy, F1-score, and recall, while LDiffCNN1 excelled in precision. LDiffCNN1 performed consistently well across the metrics, with respectable scores overall. In Table 4. 9, LDiffCNN1 displayed the best accuracy and recall, while DiffCNN attained the highest F1-score and precision. CNN and LCNN1 exhibited moderate performance, and LCNN2 consistently had the lowest accuracy. LDiffCNN2 performed better than LCNN2 but lower than the top-performing models, presented visually in Figure 4.13 through Figure 4.18.

In the evaluation of the Cifar-10 dataset (Table 4.10, Table 4.11, and Table 4.12), DiffCNN highlighted the highest overall performance in terms of accuracy, while LDiffCNN1 excelled in precision. CNN and LCNN1 achieved slightly lower accuracy but still obtained respectable scores. LCNN2 consistently had the lowest accuracy, and LDiffCNN2 performed slightly better than LCNN2. (Table 4.13, Table 4.14, and Table 4.15), LDiffCNN1 consistently demonstrated the highest overall performance in accuracy, F1-score, recall, and precision. DiffCNN also performed well across the metrics, while CNN and LCNN1 achieved slightly lower accuracy. LCNN2 had the lowest accuracy, and LDiffCNN2 performed slightly better. As presented visually in Figure 4.19 through Figure 4.24

Across the multiple evaluation metrics presented in Table 4.16, Table 4.17, and Table 4.18, DiffCNN steadily demonstrated better performance on the Intel scene dataset on metrics of accuracy, F1-score, and Recall. However, LDiffCNN1 also performed well, particularly in precision. CNN1 and CNN2 achieved slightly lower accuracy but still obtained respectable scores. LCNN2

consistently had the lowest accuracy, while LDiffCNN2 performed slightly better than LCNN2. These results are visually presented in Figure 4.25 through Figure 4.30.

Table 4.19, shows CNN achieving the highest overall performance in accuracy, F1-score, and recall. DiffCNN also performed well, excelling in precision. CNN and LCNN1 achieved slightly lower accuracy but still obtained respectable scores, while LCNN2 performed slightly better than LDiffCNN2, which had the lowest accuracy. In addition, these results are visually presented in Figure 4. 31 through Figure 4. 36.

Lastly, the subsequent tables (Table 4. 20, and Table 4. 21) continued to highlight the consistent performance of LDiffCNN1 and DiffCNN. LDiffCNN1 maintained its dominance as the top-performing model with high accuracy, F1-Score, recall, and precision, while DiffCNN displayed excellent precision and recall scores. CNN1 and LCNN1 achieved slightly lower accuracy compared to LDiffCNN1 but remained competitive. LCNN2 and LDiffCNN2 showed the lowest accuracy among the models, with LDiffCNN2 performing slightly better than LCNN2. The visual presentation is shown in Figure 4.37 through Figure 4.42

Table 4.1:Results Summary for SGD with Learning Rate of 1e-2

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	85	97.47	97.80	97.80	97.83
LCNN1	69	97.93	97.93	97.93	97.95
LCNN2	78	96.60	96.80	96.80	96.80
DiffCNN	84	98.13	98.53	98.53	98.53
LDiffCNN1	65	98.20	98.74	98.73	98.72
LDiffCNN2	98	97.30	97.47	97.47	97.73

Table 4.2:Summary of Results for Adam Optimizer with Learning Rate of 5e-4

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	99	97.70	97.40	97.40	97.50
LCNN1	86	97.80	97.93	97.93	97.95
LCNN2	75	97.00	97.40	97.40	97.41
DiffCNN	97	98.40	98.47	98.47	98.48
LDiffCNN1	86	98.50	98.57	98.57	98.57
LDiffCNN2	82	97.40	96.82	96.80	96.80

Table 4.3:Results Summary for Adam optimizer with Learning Rate=3e-4

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	80	97.80	97.53	97.53	97.55
LCNN1	88	97.70	97.53	97.53	97.54
LCNN2	92	97.00	96.27	96.27	96.41
DiffCNN	80	98.60	98.67	98.67	98.68
LDiffCNN1	42	98.73	98.76	98.76	98.76
LDiffCNN2	85	97.20	96.43	96.43	96.41

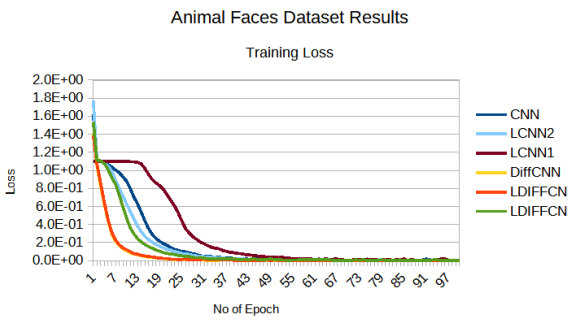

Figure 4.1: Training Loss for SGD

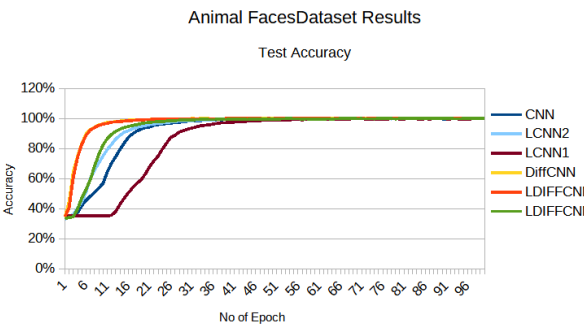

Figure 4.2: Test Accuracy for SGD

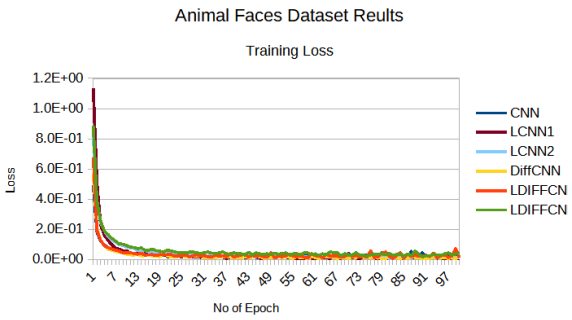

Figure 4.3:Training Loss for Adam (5e-4)

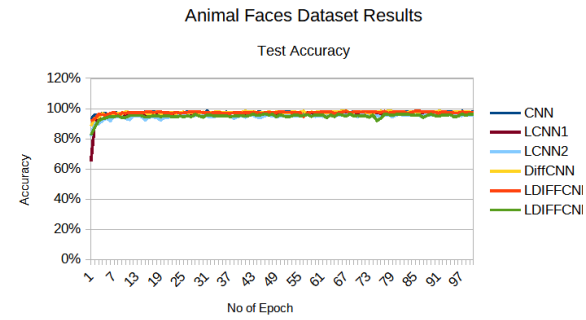

Figure 4.4:Test Accuracy for Adam (3e-4)

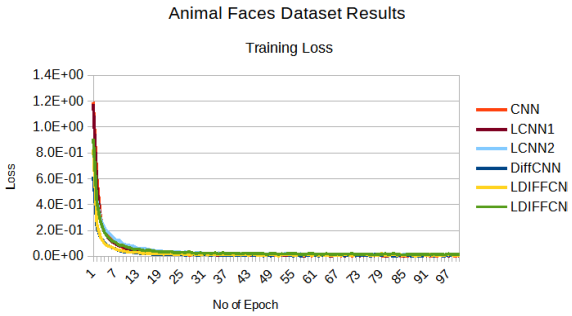

Figure 4.5: Training Loss for Adam (3e-4)

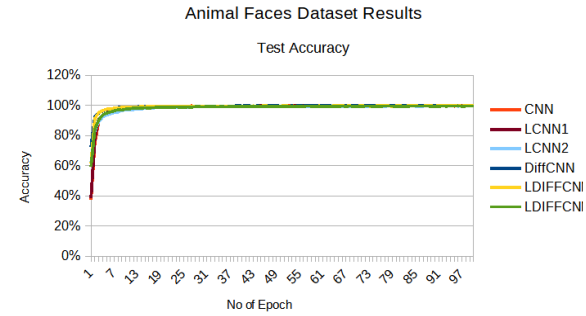

Figure 4.6: Test Accuracy for Adam (3e-4)

Table 4. 4:Results Summary for SGD with Learning Rate of 1e-2

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	97	98.10	97.20	97.20	97.82
LCNN1	97	98.20	97.73	9.773	97.80
LCNN2	95	96.90	96.77	96.77	96.77
DiffCNN	83	98.30	98.03	98.05	98.03
LDiffCNN1	96	99.00	98.58	98.58	98.60
LDiffCNN2	93	97.20	96.11	96.11	96.73

Table 4. 5:Results Summary for Adam (optimizer) Learning Rate=5e-4

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	82	98.20	96.97	96.97	97.01
LCNN1	42	98.40	97.73	97.73	97.78
LCNN2	84	97.40	96.10	96.10	96.11
DiffCNN	84	98.40	98.33	98.33	98.34
LDiffCNN1	87	98.80	98.75	98.75	98.58
LDiffCNN2	62	97.40	96.40	96.40	96.46

Table 4. 6:Results Summary for Adam (optimizer) Learning Rate=3e-4

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	64	98.40	97.43	96.97	97.43
LCNN1	64	98.60	97.73	97.73	97.76
LCNN2	90	97.50	96.57	96.57	96.59
DiffCNN	98	98.80	98.42	98.42	98.43
LDiffCNN1	80	98.70	98.64	98.64	98.64
LDiffCNN2	91	97.80	97.63	97.63	97.64

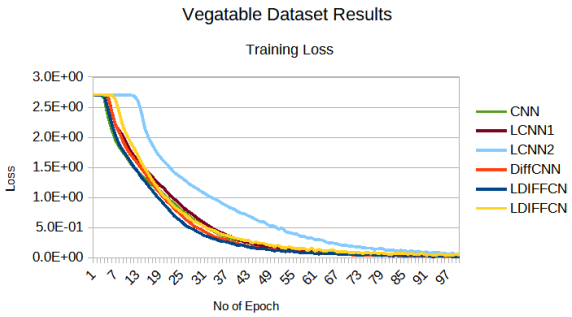

Figure 4.7: Training Loss for SGD

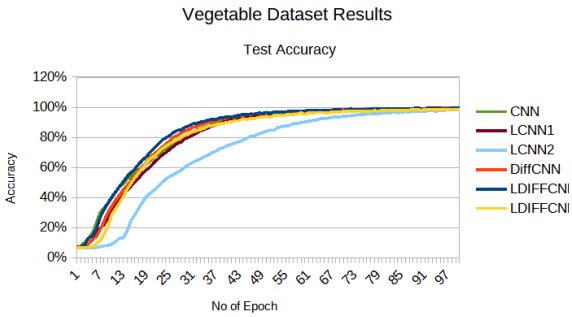

Figure 4.8: Test Accuracy for SGD

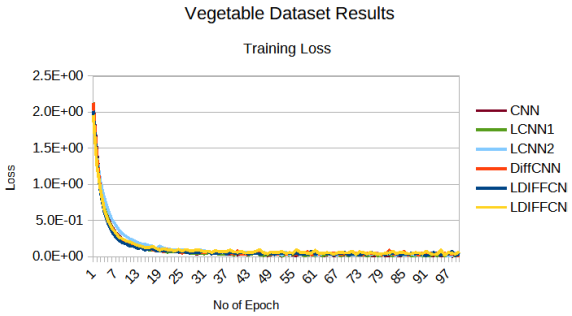

Figure 4.9: Training Loss for Adam (5e-4)

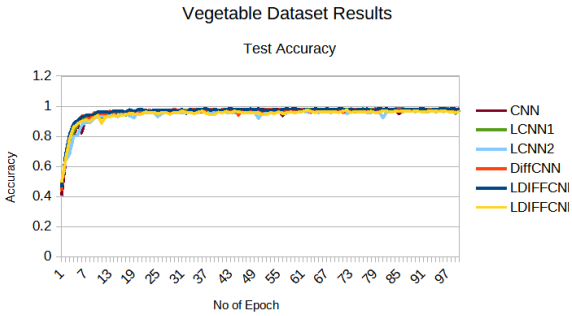

Figure 4.10: Test Accuracy for Adam (5e-4)

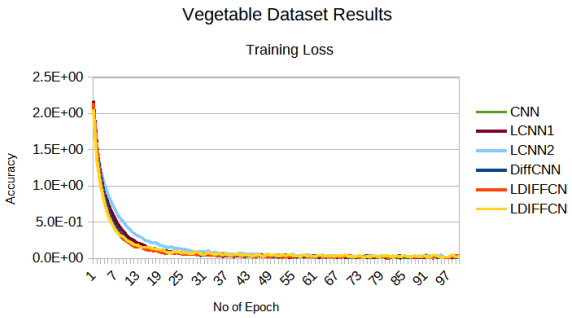

Figure 4.11: Training Loss for Adam (3e-4)

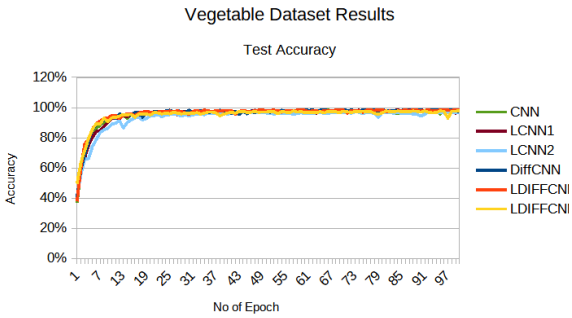

Figure 4.12: Test Accuracy for Adam (3e-4)

Table 4. 7:Results Summary for 10-Monkey Species SGD with Learning Rate of 1e-2

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	93	84.50	84.49	84.79	83.93
LCNN1	97	80.40	79.25	79.26	77.81
LCNN2	93	72.40	72.58	72.85	68.86
DiffCNN	93	89.50	89.47	88.86	86.99
LDiffCNN1	97	88.00	89.09	88.23	88.09
LDiffCNN2	90	73.97	73.35	73.37	71.71

Table 4. 8:Results Summary for 10-Monkey Species with Adam and Learning Rate of 5e-4

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	83	88.70	86.80	87.67	86.64
LCNN1	49	78.56	78.65	77.64	75.45
LCNN2	58	72.82	72.32	72.84	68.68
DiffCNN	42	91.60	90.47	91.14	89.83
LDiffCNN1	96	92.20	91.01	92.44	90.87
LDiffCNN2	73	73.89	73.21	73.00	71.75

Table 4. 9:Results Summary for 10-Monkey Species with Adam and Learning Rate of 3e-4

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	92	87.40	86.66	87.18	86.17
LCNN1	53	78.70	78.64	78.80	77.61
LCNN2	61	74.12	72.43	73.94	71.44
DiffCNN	71	92.20	92.54	91.93	91.48
LDiffCNN1	89	92.50	92.01	92.46	90.93
LDiffCNN2	43	74.73	71.14	72.97	70.90

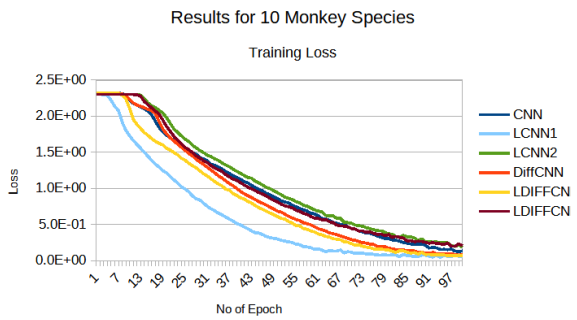

Figure 4.13: Training Loss for SGD

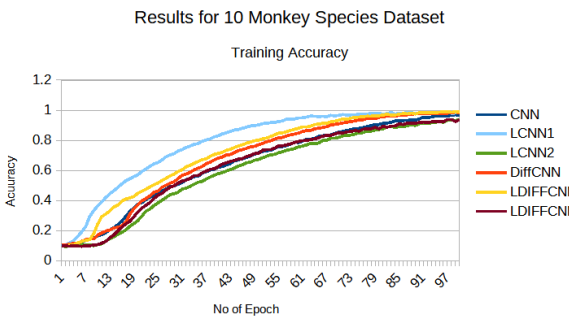

Figure 4.14: Test Accuracy for SGD

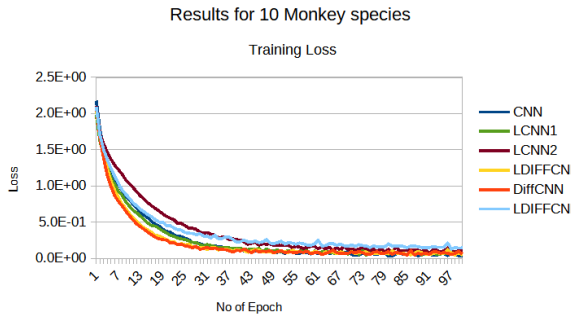

Figure 4.15: Training Loss for Adam (5e-4)

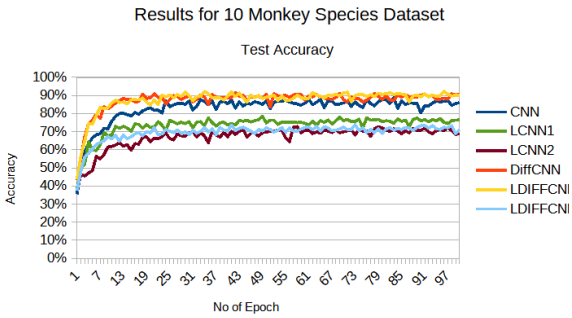

Figure 4.16: Test Accuracy for Adam (5e-4)

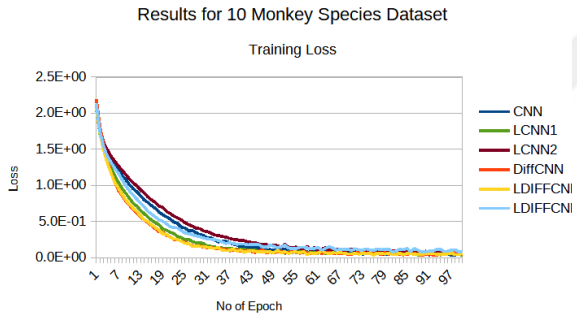

Figure 4.17:Training Loss for Adam (3e-4)

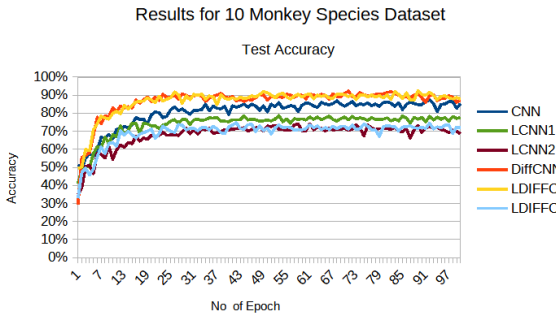

Figure 4.18: Test Accuracy for Adam (3e-4)

Table 4.10:Results Summary for SGD with Learning Rate of 1e-2 on Cifar-10 Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	99	80.30	79.74	79.74	80.03
LCNN1	97	80.00	79.82	79.82	79.72
LCNN2	100	70.50	70.59	70.59	71.11
DiffCNN	96	83.90	83.40	83.40	83.71
LDiffCNN1	99	83.80	83.36	83.36	84.06
LDiffCNN2	99	74.70	74.20	74.20	74.18

Table 4.11:Results Summary for Adam with a Learning Rate of 5e-4 on Cifar-10 Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	100	84.24	84.36	84.36	84.34
LCNN1	89	83.16	83.06	83.06	83.22
LCNN2	100	78.27	78.23	78.23	78.52
DiffCNN	80	84.52	84.41	84.41	84.56
LDiffCNN1	99	85.14	85.31	85.31	85.40
LDiffCNN2	93	78.11	77.89	77.89	78.26

Table 4.12:Results Summary for Adam Learning Rate of 3e-4 on Cifar-10 Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	99	83.80	83.35	83.35	83.71
LCNN1	88	83.60	83.12	83.12	83.09
LCNN2	100	78.30	78.35	78.35	78.44
DiffCNN	97	85.70	85.61	85.61	85.82
LDiffCNN1	95	85.60	85.48	85.48	85.53
LDiffCNN2	97	79.30	78.84	79.05	78.84

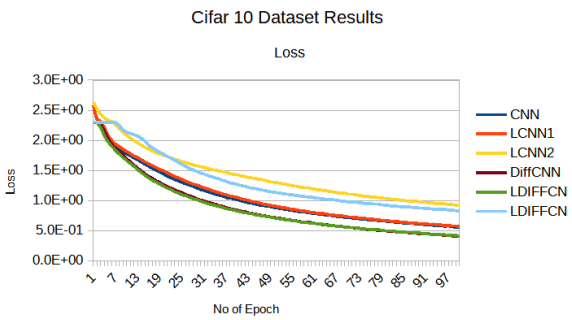

Figure 4.19: Training Loss for SGD

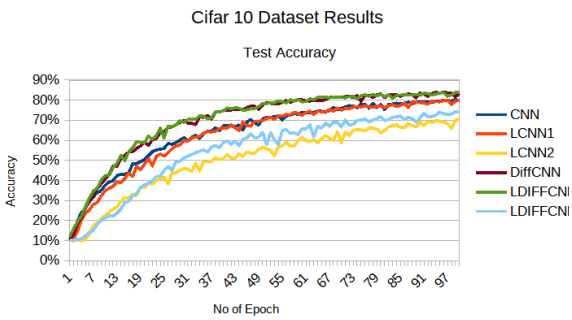

Figure 4.20: Test Accuracy for SGD

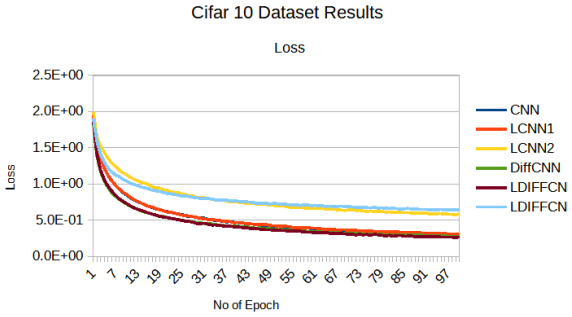

Figure 4.21: Training Loss for Adam (5e-4)

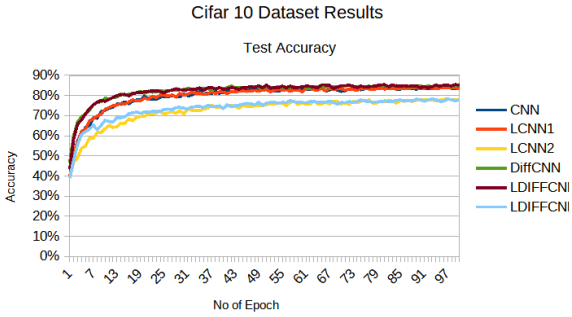

Figure 4.22: Test Accuracy for Adam (5e-4)

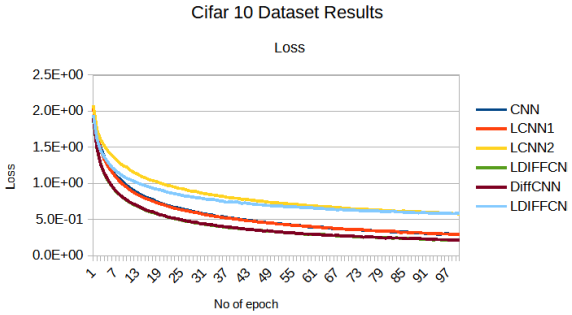

Figure 4.23: Training Loss for Adam (3e-4)

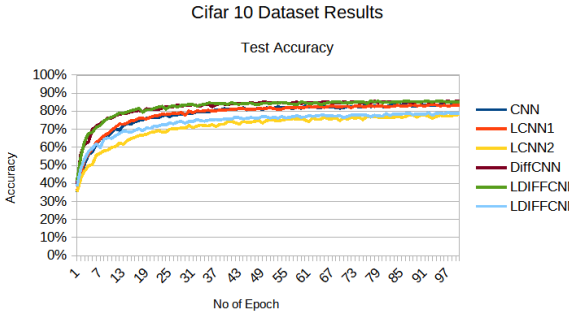

Figure 4.24: Test Accuracy for Adam (3e-4)

Table 4.13:Results Summary for Intel Scene Dataset with SGD and Learning Rate of 1e-2

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	56	80.00	79.07	79.38	79.95
LCNN1	88	80.20	80.10	80.27	80.16
LCNN2	62	79.00	77.23	77.19	77.47
DiffCNN	47	81.67	80.53	80.56	80.85
LDiffCNN1	47	82.83	81.03	81.12	81.15
LDiffCNN2	62	79.30	79.90	79.82	79.48

Table 4.14:Results Summary for Adam with a Learning Rate of 5e-4 on Intel Scene Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	13	80.27	79.70	79.74	80.03
LCNN1	54	80.17	80.33	80.32	80.85
LCNN2	67	78.80	78.90	78.95	78.94
DiffCNN	25	82.93	82.90	82.02	82.14
LDiffCNN1	20	83.00	83.07	83.18	83.09
LDiffCNN2	12	79.70	79.63	79.66	79.87

Table 4.15:Results Summary for Adam with a Learning Rate of 3e-4 on Intel Scene Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	17	80.27	80.50	80.63	80.73
LCNN1	17	80.03	80.17	80.26	80.41
LCNN2	27	75.93	75.97	76.08	76.52
DiffCNN	15	81.53	82.97	82.86	82.69
LDiffCNN1	14	83.83	83.23	83.37	83.48
LDiffCNN2	16	76.7	76.40	76.71	76.62

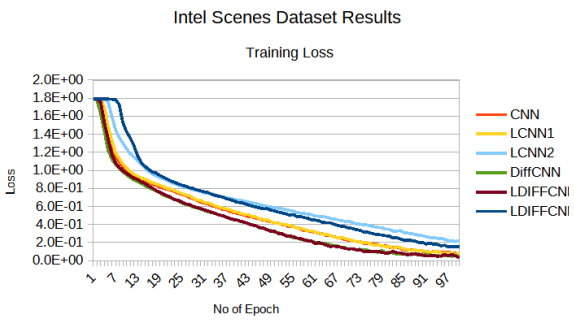

Figure 4.25: Training Loss for SGD

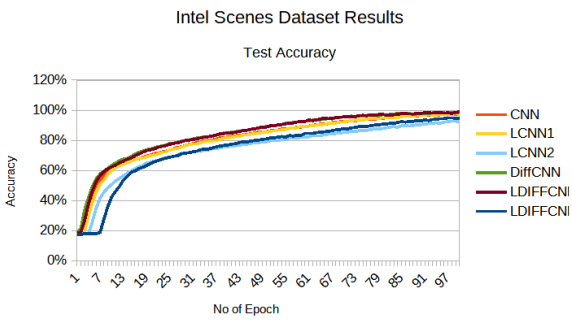

Figure 4.26 Test Accuracy for SGD

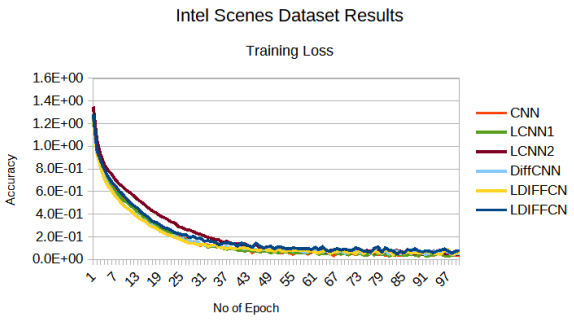

Figure 4.27: Training Loss for Adam (5e-4)

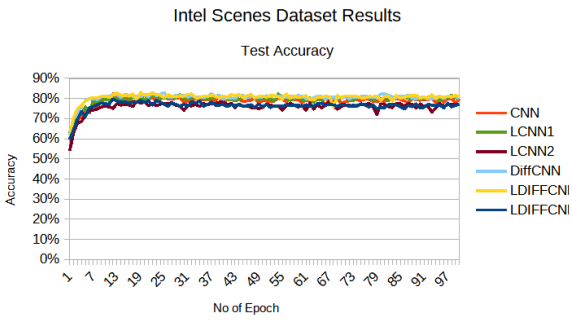

Figure 4.28: Test Accuracy for Adam (5e-4)

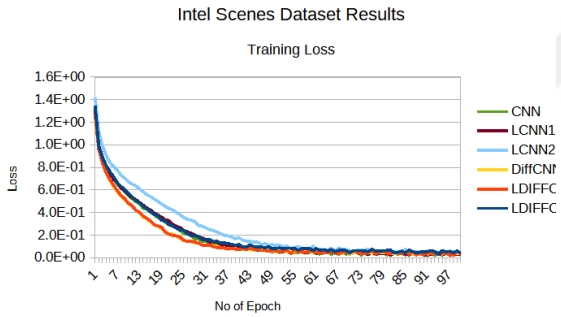

Figure 4.29: Training Loss for Adam (3e-4)

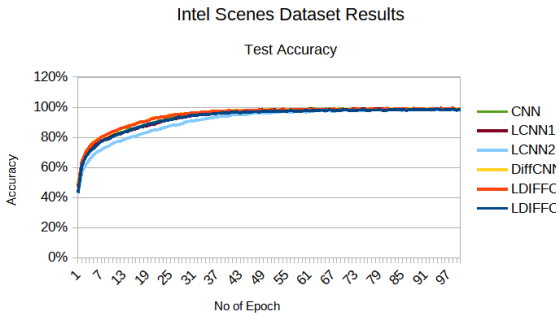

Figure 4.30: Test Accuracy for Adam (3e-4)

Table 4.16:Results Summary for Flower Classification with SGD and Learning Rate of 1e-2

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	84	73.40	72.54	72.45	72.16
LCNN1	90	70.36	69.34	69.31	67.84
LCNN2	93	63.31	63.81	63.19	63.43
DiffCNN	99	75.38	75.96	75.17	74.47
LDiffCNN1	97	75.07	75.61	75.99	75.45
LDiffCNN2	93	66.31	65.69	65.56	64.39

Table 4.17:Results Summary for Adam with Learning Rate 5e-4 on Flower Classification Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	47	72.70	71.03	72.95	71.28
LCNN1	67	69.85	68.83	68.96	66.81
LCNN2	92	62.72	60.47	61.16	60.16
DiffCNN	58	76.70	75.47	75.15	76.94
LDiffCNN1	87	73.80	73.21	73.98	73.33
LDiffCNN2	36	67.64	64.84	65.22	62.72

Table 4.18:Results Summary for Adam with Learning Rate 3e-4 on Flower Classification Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	85	73.75	73.36	73.97	72.71
LCNN1	95	70.06	70.43	70.30	70.54
LCNN2	78	63.44	61.23	62.16	62.59
DiffCNN	53	76.80	75.91	76.57	75.53
LDiffCNN1	91	76.90	76.74	77.09	76.51
LDiffCNN2	49	68.62	68.65	68.85	67.82

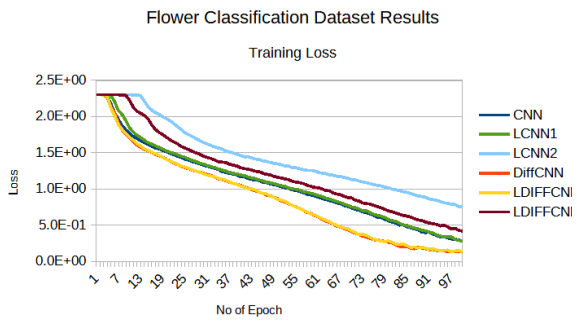

Figure 4. 31: Training Loss for SGD

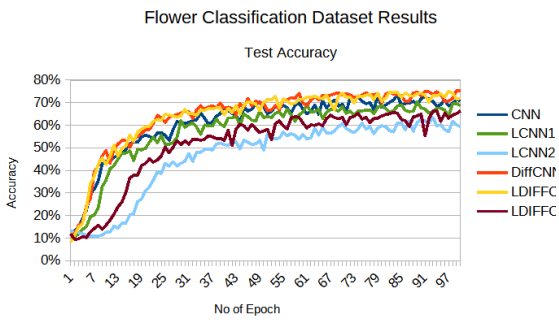

Figure 4. 32: Test Accuracy for SGD

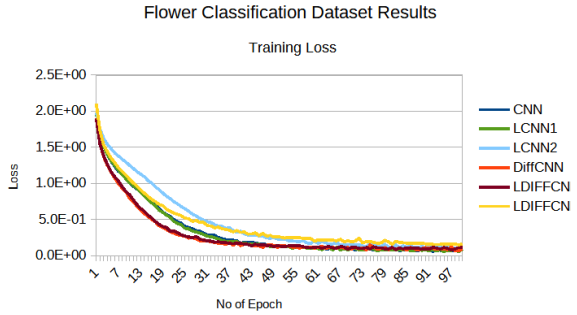

Figure 4. 33: Training Loss for Adam (5e-4)

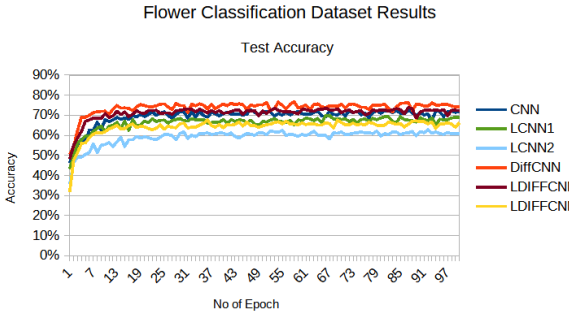

Figure 4. 34: Test Accuracy for Adam (5e-4)

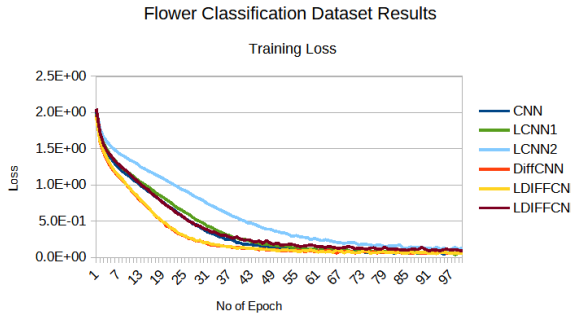

Figure 4. 35: Training Loss for Adam (3e-4)

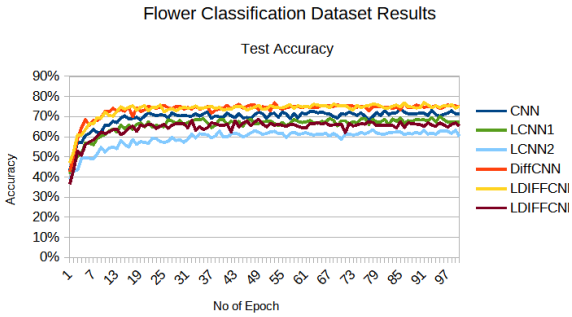

Figure 4. 36: Test Accuracy for Adam (3e-4)

Table 4.19:Results Summary for SGD with Learning Rate of 1e-2 on Tomato Leaves Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	99	94.13	93.89	93.93	93.51
LCNN1	100	93.90	93.88	93.74	93.31
LCNN2	100	91.13	91.81	91.19	91.43
DiffCNN	95	93.92	93.82	93.65	93.61
LDiffCNN1	92	93.97	93.47	93.42	93.11
LDiffCNN2	97	90.02	90.64	90.18	89.64

Table 4. 20:Results Summary for Adam with Learning Rate of 5e-4 on Tomato Leaves Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	62	93.60	92.91	93.10	92.33
LCNN1	74	94.01	93.10	92.97	92.86
LCNN2	81	92.05	91.59	91.24	91.24
DiffCNN	70	94.30	94.18	93.81	94.31
LDiffCNN1	85	94.33	94.18	94.31	94.24
LDiffCNN2	94	91.43	90.84	90.22	90.72

Table 4. 21:Results Summary for Adam with 3e-4 Learning Rate on Tomato Leaves Dataset

Structure/Model	Epochs	Accuracy	F1-Score	Recall	Precision
CNN	80	94.18	93.78	93.87	93.41
LCNN1	62	94.03	93.57	93.52	93.44
LCNN2	97	92.70	92.46	92.36	91.94
DiffCNN	90	94.90	94.12	93.98	94.03
LDiffCNN1	58	95.02	95.21	94.26	94.02
LDiffCNN2	97	92.08	91.49	91.34	91.12

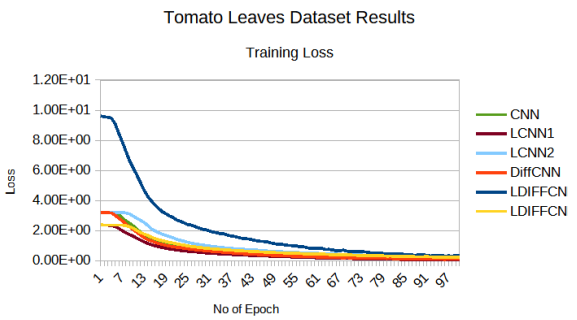

Figure 4.37: Training Loss for SGD

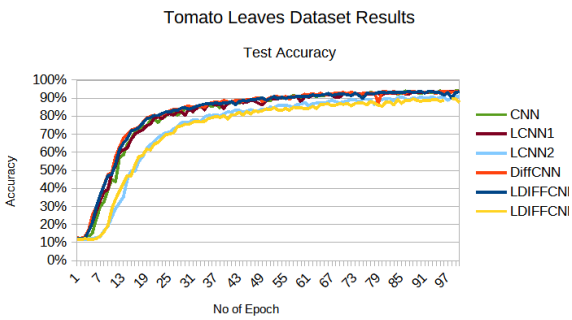

Figure 4.38: Test Accuracy for SGD

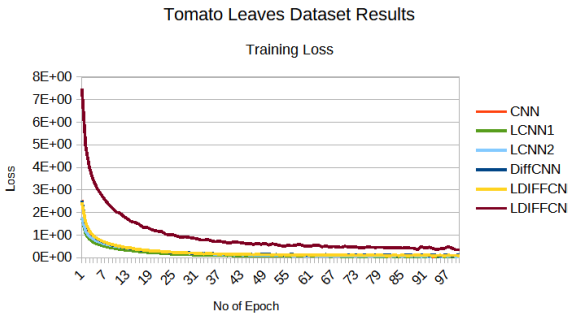

Figure 4.39: Training Loss for Adam (5e-4)

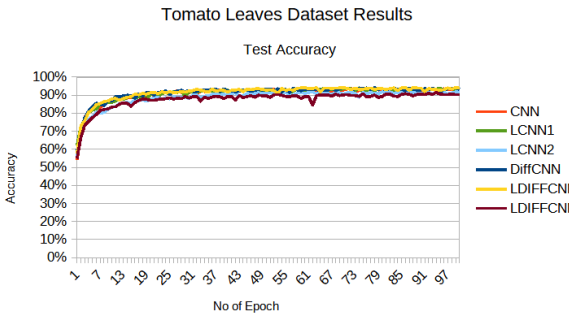

Figure 4 40: Test Accuracy for Adam (5e-4)

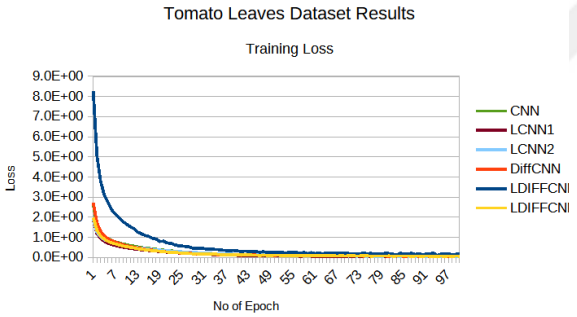

Figure 4.41: Training Loss for Adam (3e-4)

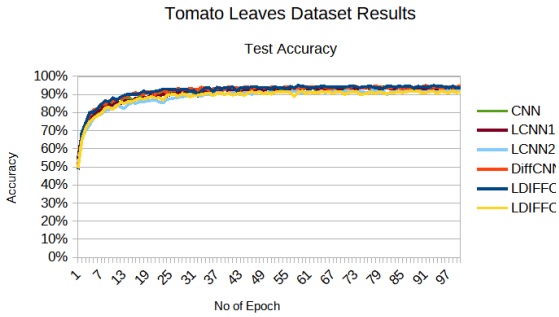

Figure 4.42: Test Accuracy for Adam (3e-4)

Table 4.22: Model Accuracy on Cub-200-2011 and Stanford-Cars Datasets

Model	**Cub-200-2011**		**Stanford Cars**	
	Top 1	**Top 5**	**Top 1**	**Top 5**
ResNet 18	66.43	85.37	78.46	91.84
ResNet 34	65.72	85.40	77.91	92.36
ResNet 50	65.72	85.09	77.89	92.69
LDiff ResNet1-8	70.50	**91.59**	**84.10**	94.34
LDiff ResNet-34	**75.68**	90.16	81.90	94.91
LDiff ResNet-50	74.31	89.66	82.48	**95.06**
LogResNet-18	65.92	85.47	77.44	92.17
LogResNet-34	64.08	84.47	76.31	92.41
LogResNet-50	62.28	84.10	76.31	92.25

In the second experiment, differential convolution was added to ResNet-18, ResNet-34, and ResNet-50, well-known deep models with diverse training policies. Six models, including Lograthmic differentially convolution-adapted ResNets and the original ResNet, were trained under identical policies. Performance metrics assessed the impact of differential convolution on ResNet structures against the baseline. Notably, on the Cub-200-2011 dataset, top-1 accuracy increased by 3.07%, 9.96%, and 7.59% for LDiffResNet-18, LDiffResNet34, and LDiffResNet-50, respectively. Similar improvements were observed on the Stanford-cars dataset, with top-1 accuracy gains of 6.02%, 5.58%, and 5.11% for LDiffResNet-18, LDiffResNet-34, and LDiffResNet-50 and with a top-5 accuracy gain of 2.17%,2.70% and,2.22% respectively. The results of the second experiment can be seen in Table 4.22.

In addition to activation functions, our study also explored cost functions/loss functions. The third set experiment involved three models: CNN, DiffCNN, and LDiffCNN. CNN used input sizes of 32x32 pixels, with kernel sizes of 5x5 and 3x3 for feature extraction. Filter sizes of 32, 64, and 128 were used. DiffCNN and LDiffCNN1 employed differential convolution followed by a normal convolutional layer. The models used cross-entropy loss for CNN and DiffCNN, while LDiffCNN1 utilized an acustomized cost function. The findings reveal that outperforms the standard CNN model across most of all datasets. The performance differences are relatively moderate, ranging from approximately 2.33% to 3.02%, with higher accuracy, F1-score, recall, and precision for the models. Moreover, for Vegetable and Tomato Leaves datasets, all three models(CNN, DiffCNN, and LDiffCNN) consistently produce similar results within narrow ranges (0.30% and 1.07%, respectively).In contrast, the CIFAR-10, Animal Faces, and Intel Scenes datasets exhibit moderate variation, while the Flower Classification dataset shows the widest variation (5.28%), and the 10-Monkey Species dataset demonstrates moderate to high variability. The experiment results can be found in Table 4. 23 and Table 4.25 and also visually present in Figure 4. 43 through Figure 4.64.

Moreover, this investigation delved into the efficacy of our activation function, LogRelu, within pre-trained models, along with DiffCNN. A comprehensive comparison of its performance with alternative activation functions is detailed in section 4.2.

Table 4. 23:Results Summary for Adam with 5e-4 Learning Rate overall best

	CNN				DiffCNN				LDiffCNN			
Dataset Name	**Acc**	**F1**	**R**	**P**	**Acc**	**F1**	**R**	**P**	**Acc**	**F1**	**R**	**P**
Cifar-10	83.20	83.11	83.20	83.39	85.32	85.27	85.32	85.55	85.55	85.53	85.55	85.68
Vegetable	98.37	98.37	98.37	98.38	98.67	98.67	98.67	98.68	98.63	98.63	98.63	98.64
Animal Faces	97.53	97.53	97.53	97.54	98.93	98.93	98.93	98.94	98.53	98.53	98.53	98.54
10-Monkey Species	83.54	83.98	84.07	85.86	84.07	84.11	82.7	85.28	83.38	83.86	83.38	86.31
Intel Scene	81.43	81.49	81.47	81.59	82.7	82.83	82.7	83.21	83.1	83.16	83.1	83.37
Flower classification	70.49	70.37	70.49	71.46	74.27	74.03	74.27	74.69	74.39	74.33	74.39	75.77
Tomato Leaves	93.89	93.85	93.84	93.89	94.66	94.66	94.66	94.69	94.87	94.86	94.87	94.91

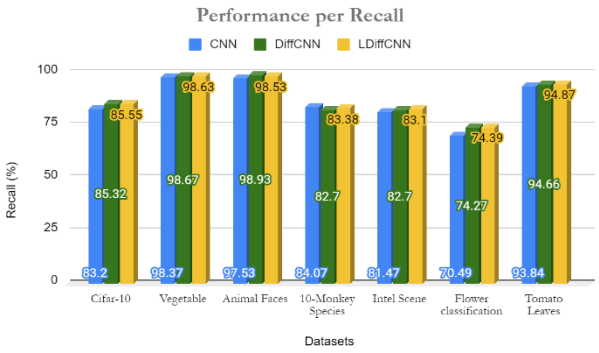

Figure 4. 43:Recall Performance

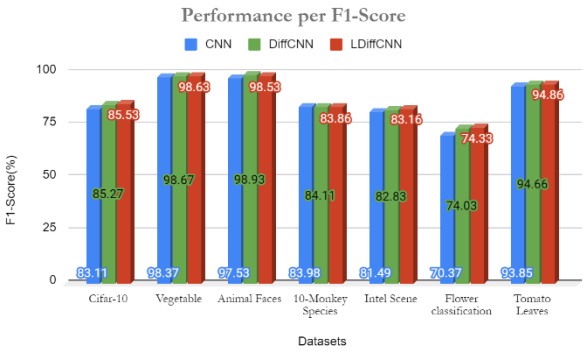

Figure 4. 44:F1-Score Performance

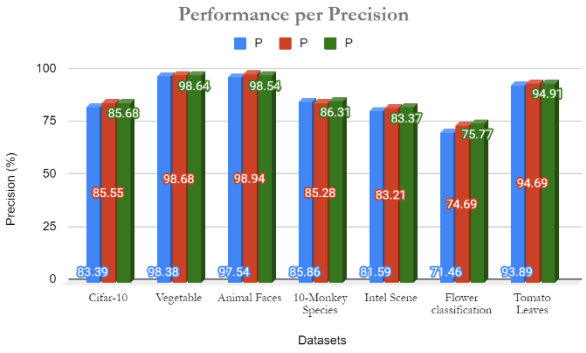

Figure 4.45:Precision Performance

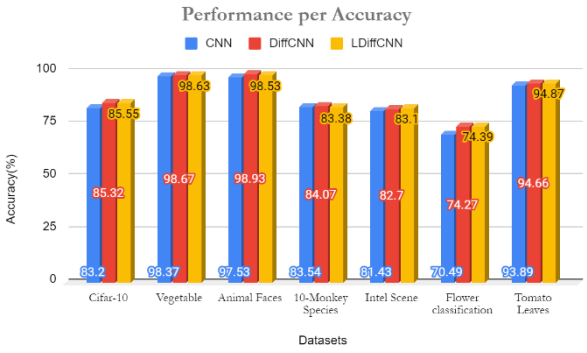

Figure 4.46:Accuracy Performance

Table 4. 24:Results Summary for Adam with 3e-4 Learning Rate overall best

	CNN				DiffCNN				LDiffCNN			
Dataset Name	**Acc**	**F1**	**R**	**P**	**Acc**	**F1**	**R**	**P**	**Acc**	**F1**	**R**	**P**
Cifar-10	83.91	83.85	83.91	83.84	86.23	86.25	86.23	86.37	85.87	85.87	85.94	85.88
Vegetable	98.3	98.30	98.30	98.31	98.87	98.87	98.87	98.87	98.8	98.80	98.80	98.81
Animal Faces	97.60	97.60	97.60	97.60	98.6	98.60	98.60	98.60	98.47	98.47	98.47	98.47
10-Monkey Species	78.41	78.78	78.41	81.15	83.38	83.71	83.38	85.68	83.54	83.98	83.54	85.86
Intel Scene	82.13	82.08	82.13	82.27	82.9	82.97	82.90	83.27	83.9	83.90	83.90	84.15
Flower classification	70.49	70.37	70.49	71.46	74.1	74.07	74.10	75.26	74.31	74.33	74.31	75.44
Tomato Leaves	94.14	94.14	94.12	94.22	94.58	94.58	94.58	94.64	94.55	94.55	94.55	94.58

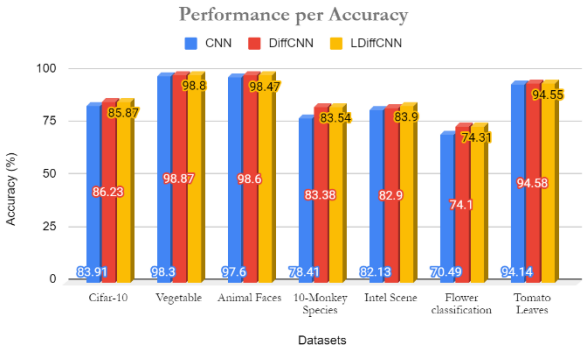

Figure 4. 47: Accuracy Performance

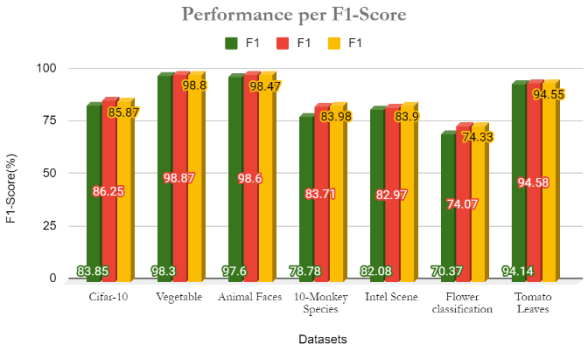

Figure 4. 48:F1-Score Performance

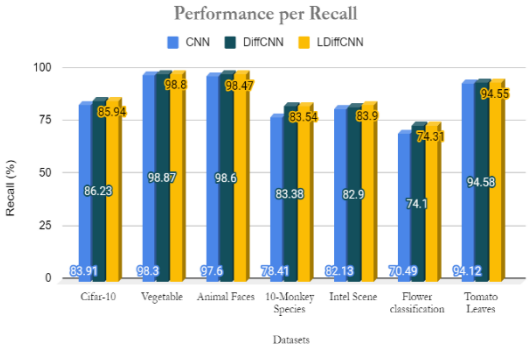

Figure 4.49:Recall Performance

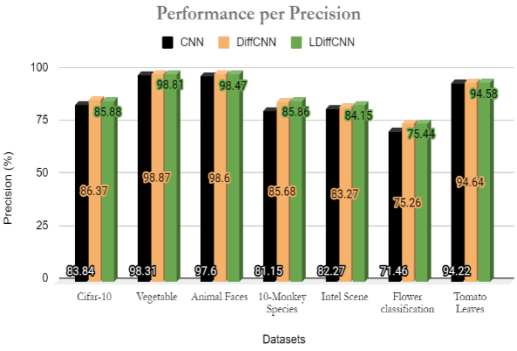

Figure 4.50:Precision Performance

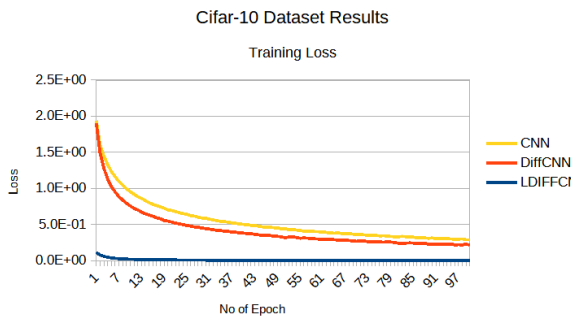

Figure 4.51:Training Loss

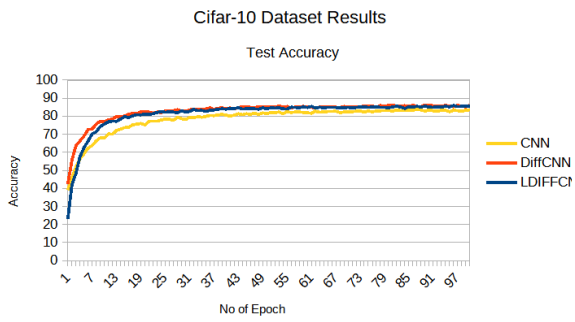

Figure 4.52:Test Accuracy

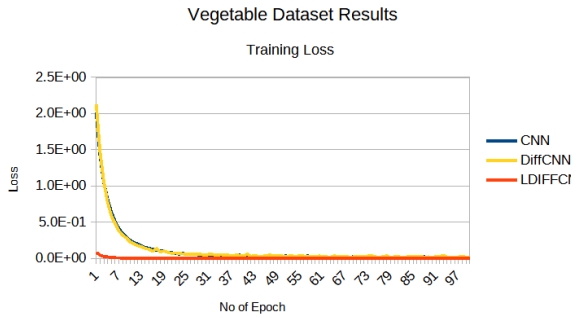

Figure 4.53:Training Loss

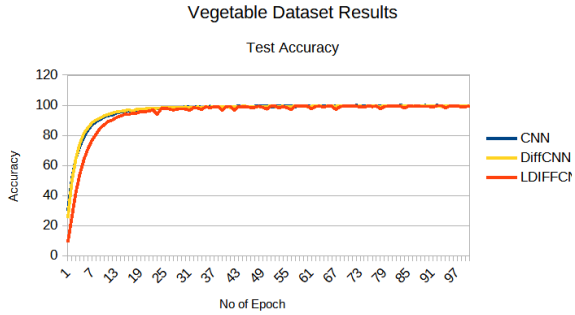

Figure 4.54:Test Accuracy

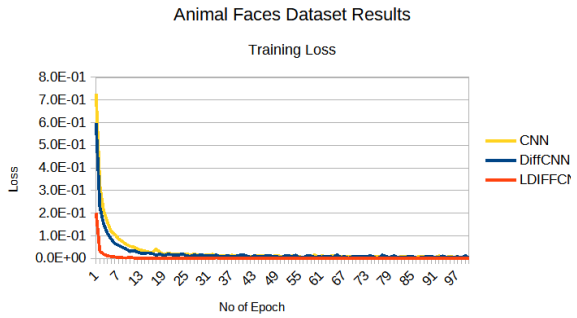

Figure 4.55:Trainining Loss

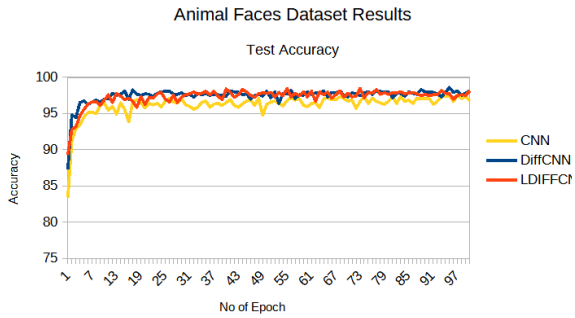

Figure 4.56::Test Accuracy

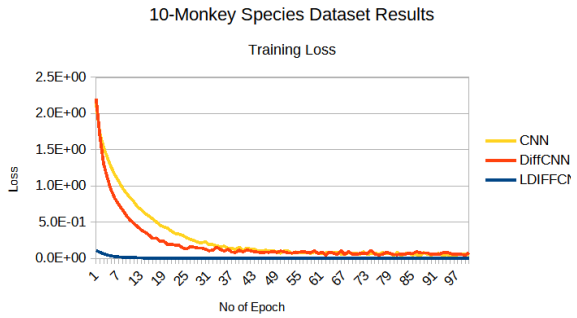

Figure 4.57:Training Loss

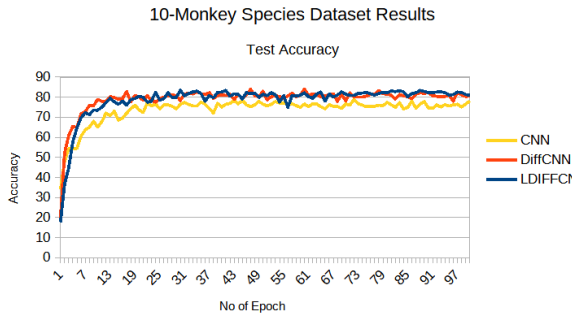

Figure 4.58::Test Accuracy

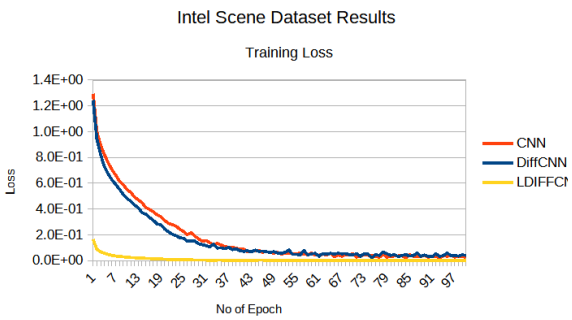

Figure 4.59:Training Loss

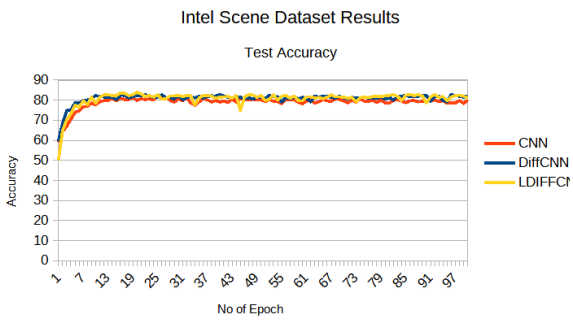

Figure 4.60:Test Accuracy

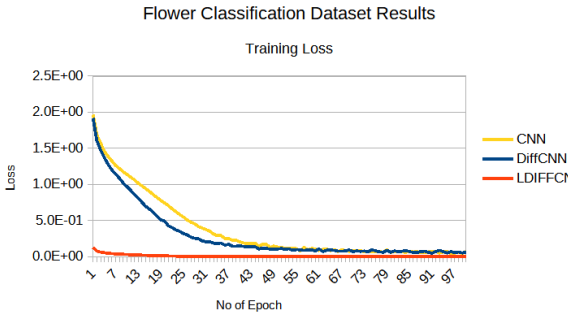

Figure 4.61:Training Loss

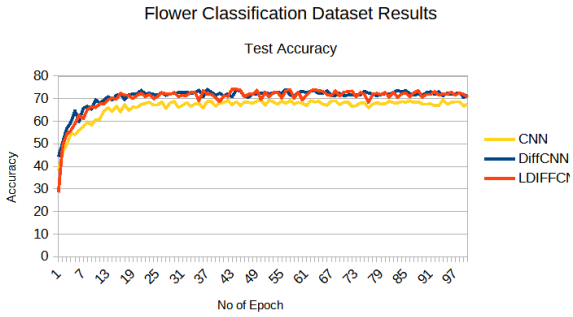

Figure 4.62:Test Accuracy

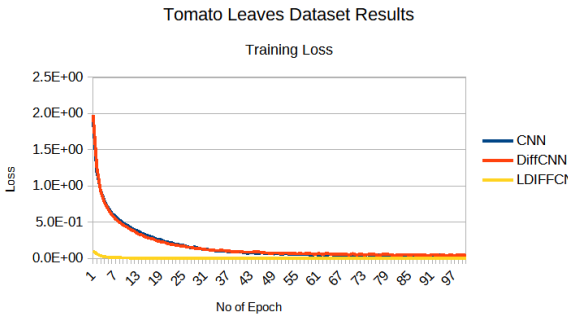

Figure 4.63:Training Loss

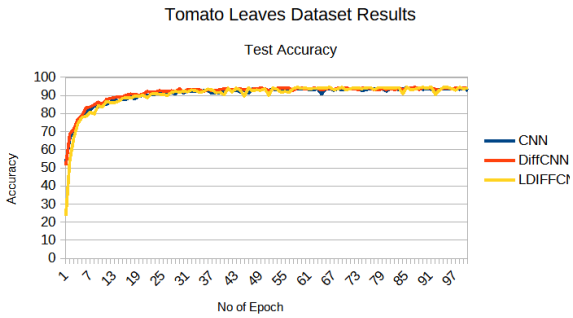

Figure 4.64:Test Accuracy

4.2. The Classification Performance of MNIST

In this section, we present results for LogRelu, ReLU, LeakyReLU, ELU, SELU, and Mish activation functions applied to pre-trained MobileNet, LeNet, DenseNet, and ResNet models and DiffCNN. These assessments cover various optimizers (Adam, SGD, and RMSprop) with learning rates (0.001, 0.1, and 0.0005) on the MNIST dataset.

4.2.1. Analysis of Various Learning for the Adam Optimizer

Table 4.25 presents LogRelu's performance across various models, highlighting its competitive accuracy. On MobileNet, it achieves a top-1 accuracy of 98.96% and consistently high top-5 accuracy at 99.97%. In LeNet, LogRelu maintains competitive top-1 accuracy (98.80%) and an impressive 99.98% for top-5 accuracy. DenseNet highlights LogRelu's excellence with a top-1 accuracy of 99.11% and a perfect 100% top-5 accuracy, surpassing other functions. In ResNet, LogRelu sustains a strong top-1 accuracy of 99.06% and a perfect 100% top-5 accuracy, all while maintaining run times in line with other activations.

Table 4.26 reveals the incompatibility of ReLU and LogRelu with a learning rate of 0.1 on MobileNet and ResNet, as well as a similar issue with Mish, SELU, and ReLU on DenseNet, resulting in notable declines in performance, with top-1 accuracy dropping by 9.80% to 20%, and top-5 accuracy decreasing by 51.39% to 79.77%. Table 4.27 reaffirms LogRelu's robust performance as observed across models, emphasizing its effectiveness with top1 accuracy ranging from (98.47-99.14%) and with top5 accuracy (99.97 -100 %) across all the models. Further, this is depicted in Figure 4.80 through Figure 4.94.

4.2.2. Analysis of SGD Optimizer Different Learning Rate

Table 4. 28 reveals that LogRelu on MobileNet achieves 97.85% top-1 accuracy and consistently high 99.98% top-5 accuracy. DenseNet highlights LogRelu's excellence with 98.27% top-1 accuracy and a perfect 99.95% top-5 accuracy, surpassing other functions. In ResNet, LogRelu maintains a strong 97.45% top-1 accuracy and a perfect 99.90% top-5 accuracy, with run times in line with other activations. Notably, LeNet, ReLU, LeakyReLU, Mish, and LogRelu are incompatible with SGD and have a learning rate of 0.001, resulting in significant top-1 accuracy drops (11.47% to 35.92%) and top-5 accuracy declines (68.90% to 84.84%). Table 4. 29 confirms LogRelu's consistency, with top-1 accuracy ranging from 96.94% to 98.85% and top-5 accuracy from 99.95% to 99.99%. Table 4. 30 highlights similar incompatibility issues in LeNet, causing top-1 accuracy drops of 10.28% to 25.59% and top-5 accuracy decreases of 56.57% to 66.12%. This is also highlighted in Figure 4.80 through Figure 4.94.

4.2.3. Analysis of RMSprop Optimizer with Different Learning Rate

Table 4.31 reveals that LogRelu on MobileNet achieves 83.18% top-1 accuracy and consistently high 99.06% top-5 accuracy. For LeNet a 99.01 and 99.99 for Top-1 and Top-5 accuracy scores. DenseNet highlights LogRelu's excellence with 99.24% top-1 accuracy and a perfect 99.99% top-5 accuracy. Table 4.32 confirms LogRelu's consistency, with top-1 accuracy ranging from 96.94% to 98.85% and top-5 accuracy from 99.95% to 99.99% for LeNet and DenseNet. Similarly,

Table 4.33 highlights a top-1 accuracy ranging from 99.05% to 99.41% and a top-5 accuracy from 99.99% to 100% for MobileNet, LeNet, and DenseNet. It is worth noting that incompatibility issues in MobileNet for ReLU, SELU for DenseNet, LogRelu, and ReLU for ResNet with RMSprop optimizer with 0.001 learning rate. Furthermore, with RMSprop Optimizer with LR: 0.01, LogRelu, ReLU, ELU, and Mish are not compatible with MobileNet Similarly For DenseNet ReLU, SELU, Mish, and ResNet, RELU, and LogRelu are not compatible this is observed by the drop of top-1 accuracy drops of 9.80% to 24.53% and top-5 accuracy decreases up to 51.39%. In addition, notably, for MobileNet, ReLU is incompatible with RMSprop and has a learning rate of 0.0005, SELU and Mish for DenseNet and LogRelu, ReLU for ResNet resulting in significant top-1 accuracy drops (30.57% to 41.80%), and top-5 accuracy declines (51.39% to 70.95%). This is also visually shown in 4.95 through 4.109.

Table 4.25: Performance of Adam Optimizer with LR: 0.001

Model	**MobileNet**			**LeNet**			**DenseNet**			**ResNet**		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	89.36	98.90	182.4	98.61	99.98	83.86	71.41	81.11	403.32	88.50	99.91	158.78
LeakyReLU	98.98	99.99	172.46	98.90	100	82.48	99.05	100	406.34	99.14	99.97	149.57
ELU	98.42	99.89	170.37	98.92	100	80.96	99.19	99.99	399.98	98.26	99.89	146.94
SELU	98.54	99.95	174.96	98.94	100	80.00	99.41	100	402.57	99.05	99.97	145.06
Mish	98.88	99.99	171.26	99.13	99.99	81.95	61.46	71.51	415.84	98.89	99.98	147.06
LogRelu	98.96	99.97	171.65	98.80	99.98	91.61	99.11	100	395.41	99.06	100	147.77

Table 4.26: Performance of Adam Optimizer with LR: 0.01

	MobileNet			**LeNet**			**DenseNet**			**ResNet**		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	9.80	51.39	174.10	98.01	99.97	82.84	19.96	61.54	397.73	9.80	51.39	149.56
LeakyReLU	97.63	99.80	171.71	96.82	99.86	83.61	98.87	100	401.21	96.96	99.90	148.00
ELU	98.00	99.92	171.04	97.22	99.27	79.76	98.58	99.97	396.42	98.30	99.96	146.57
SELU	97.47	99.93	174.84	91.82	97.70	80.35	9.80	51.39	401.90	98.54	99.97	146.03
Mish	51.98	95.98	170.96	98.25	99.83	82.29	9.80	51.39	402.79	87.01	99.17	147.67
LogRelu	9.80	51.39	173.04	97.77	99.72	82.45	98.91	100	398.00	20.95	79.77	148.51

Table 4.27: Performance of Adam Optimizer with LR: 5e-4

	MobileNet			**LeNet**			**DenseNet**			**ResNet**		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	98.29	99.98	176.93	98.87	100	83.39	99.34	100	397.92	98.68	99.98	150.27
LeakyReLU	98.70	99.99	172.82	98.82	100	83.61	99.38	100	401.80	99.05	99.98	146.55
ELU	98.74	100	173.65	98.75	100	80.14	99.32	100	395.51	98.99	99.97	146.99
SELU	98.49	99.99	174.64	89.92	99.40	78.81	99.35	100	406.37	99.08	99.99	145.36
Mish	98.46	99.99	171.64	98.96	100	81.24	99.05	99.99	395.61	98.92	100	146.76
LogRelu	99.03	99.97	172.89	98.47	100	82.90	99.14	100	406.00	99.14	99.99	149.85

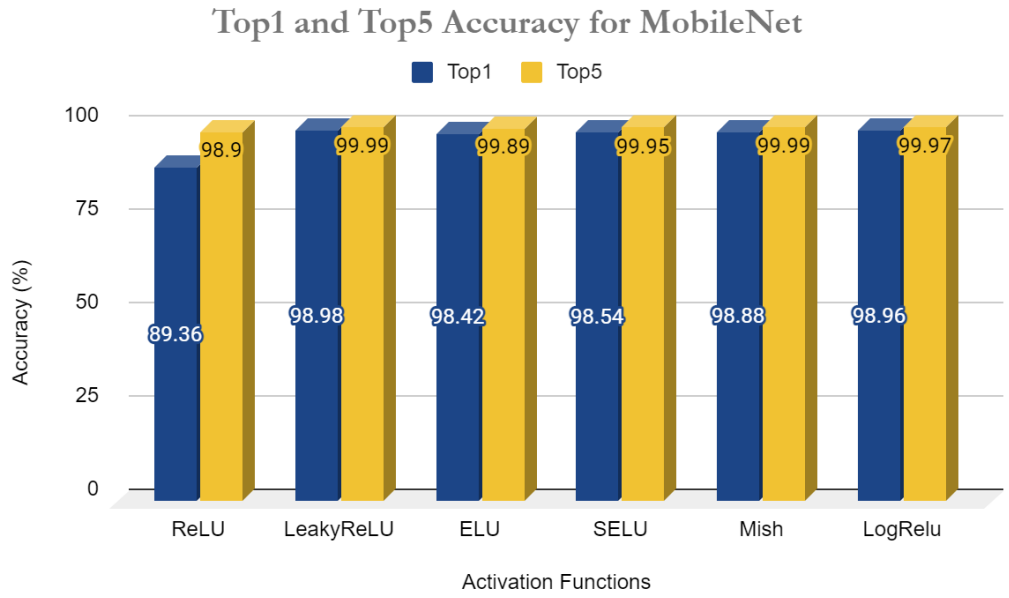

Figure 4.65 MobileNet with Adam(0.001)

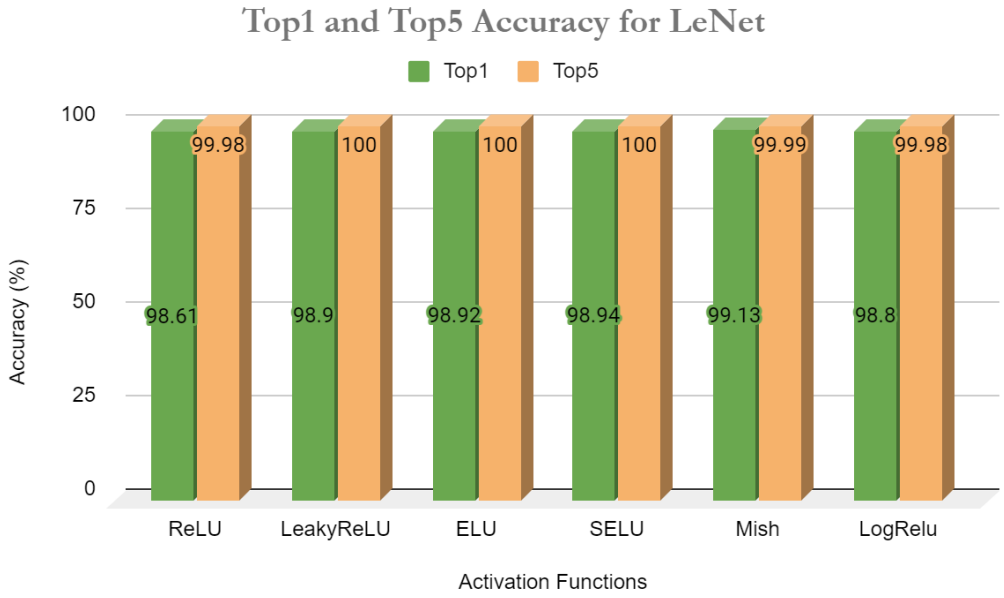

Figure 4.66:LeNet with Adam(0.001)

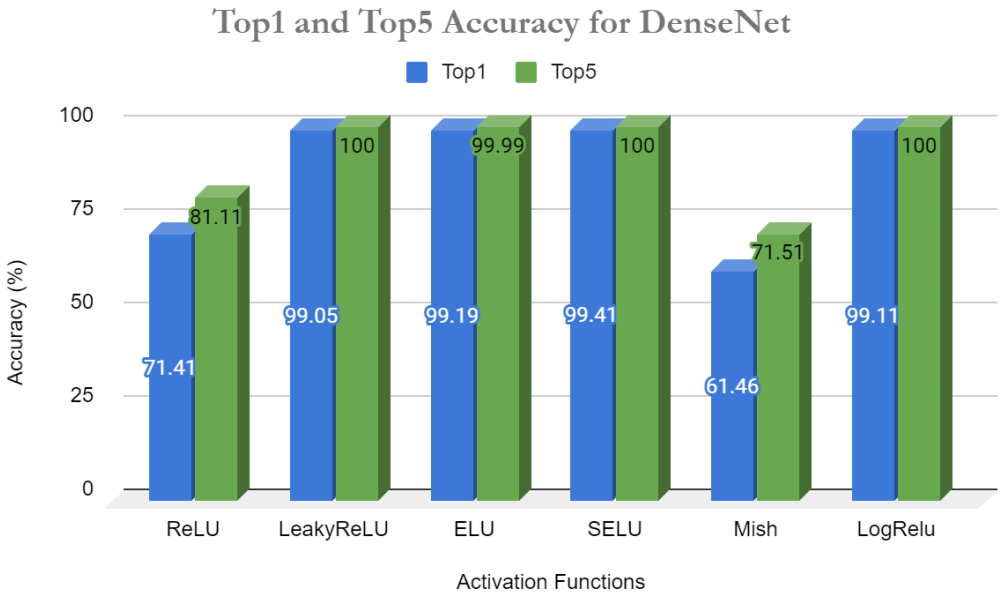

Figure 4.67:DenseNet with Adam(0.001)

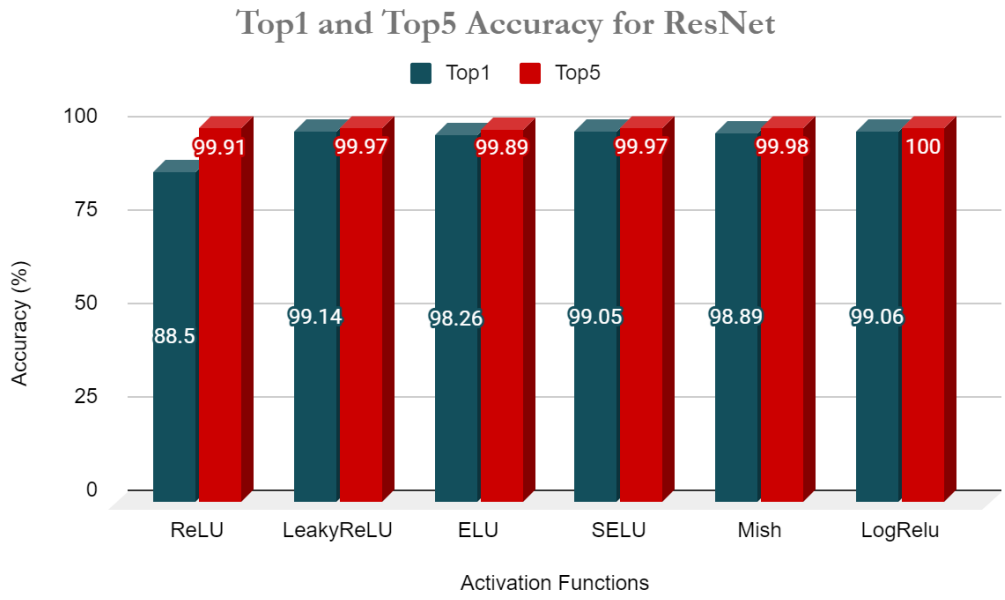

Figure 4.68:ResNet with Adam(0.001)

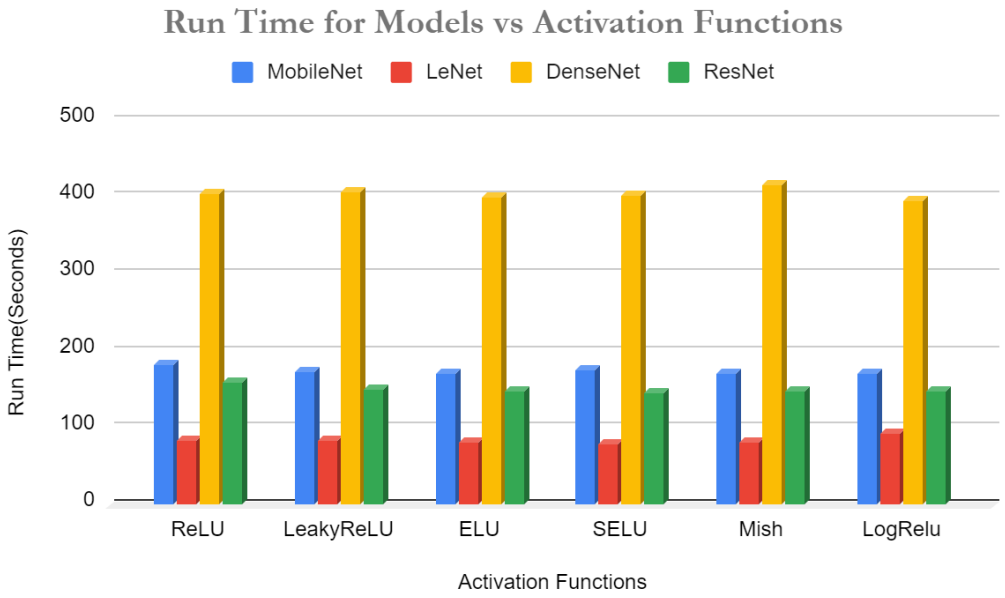

Figure 4.69:Run Time for Adam(0.001)

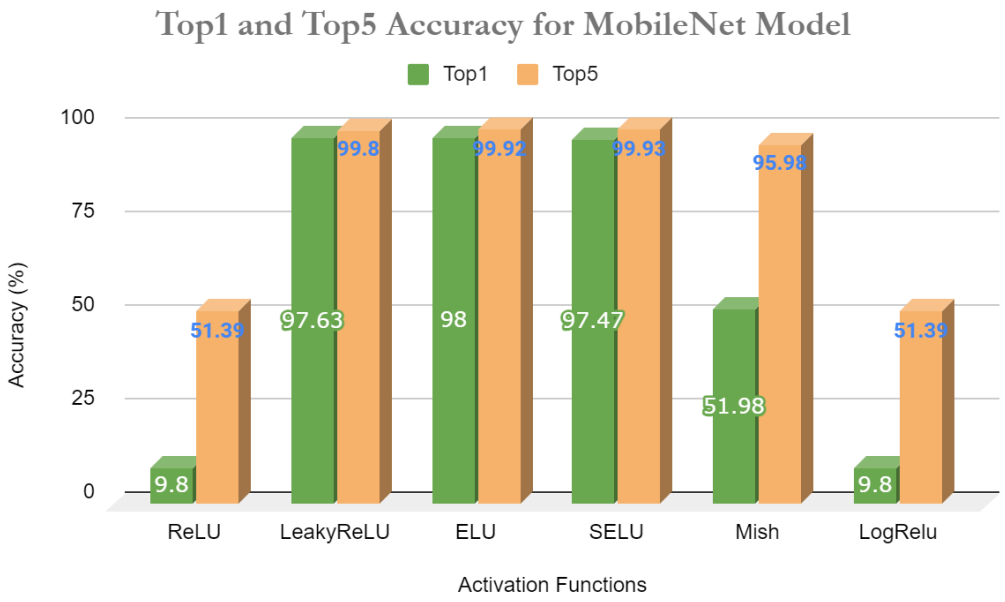

Figure 4.70:MobileNet with Adam(0.01)

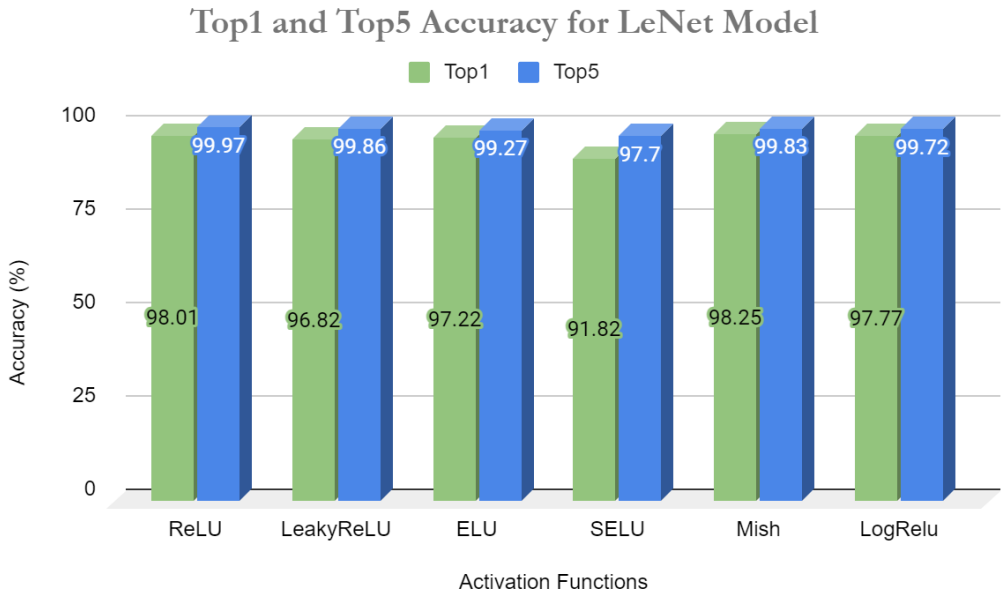

Figure 4.71:LeNet with Adam(0.01)

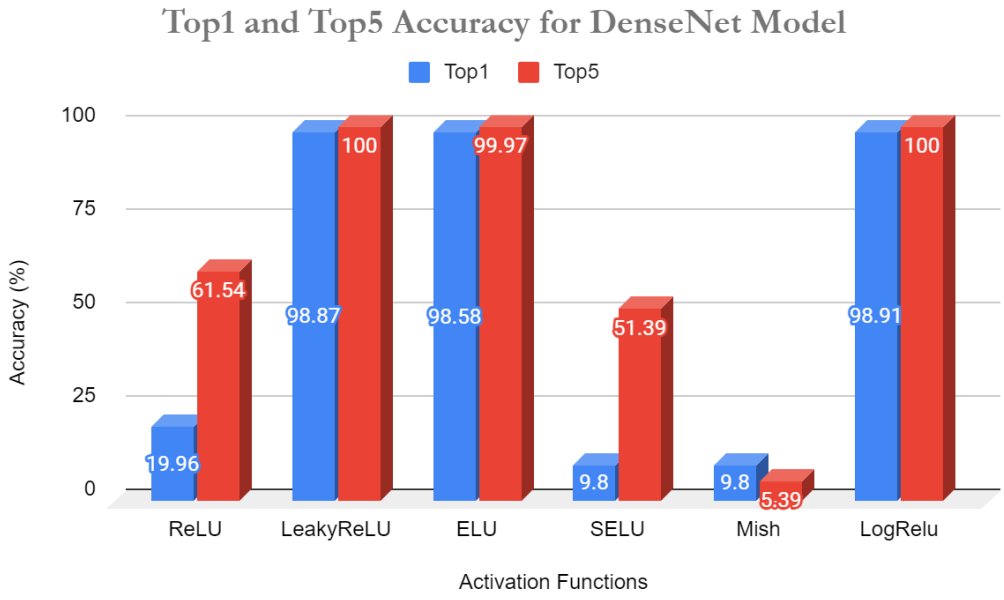

Figure 4.72:DenseNet with Adam(0.01)

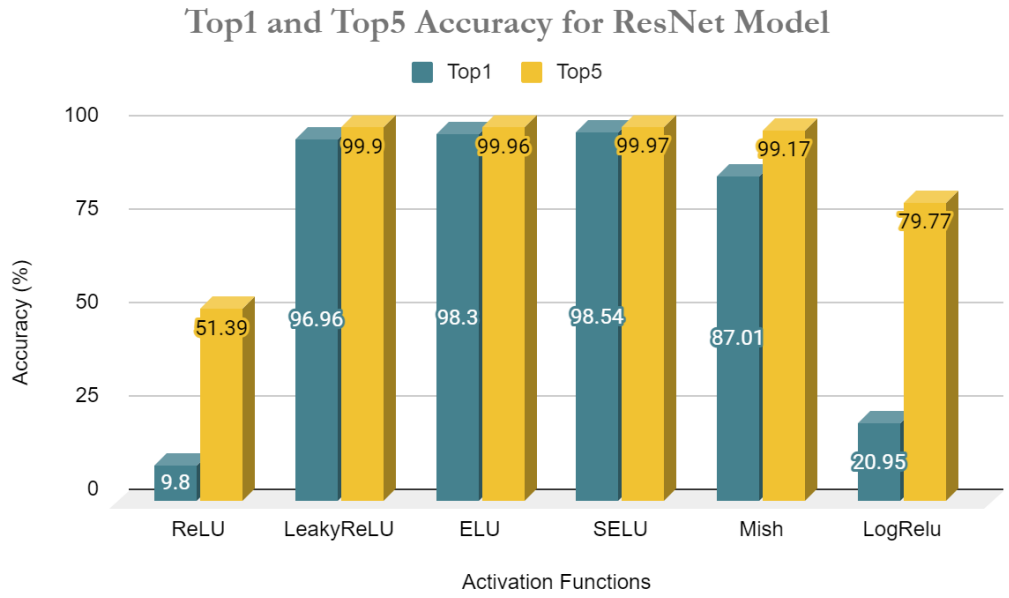

Figure 4.73:ResNet with Adam(0.01)

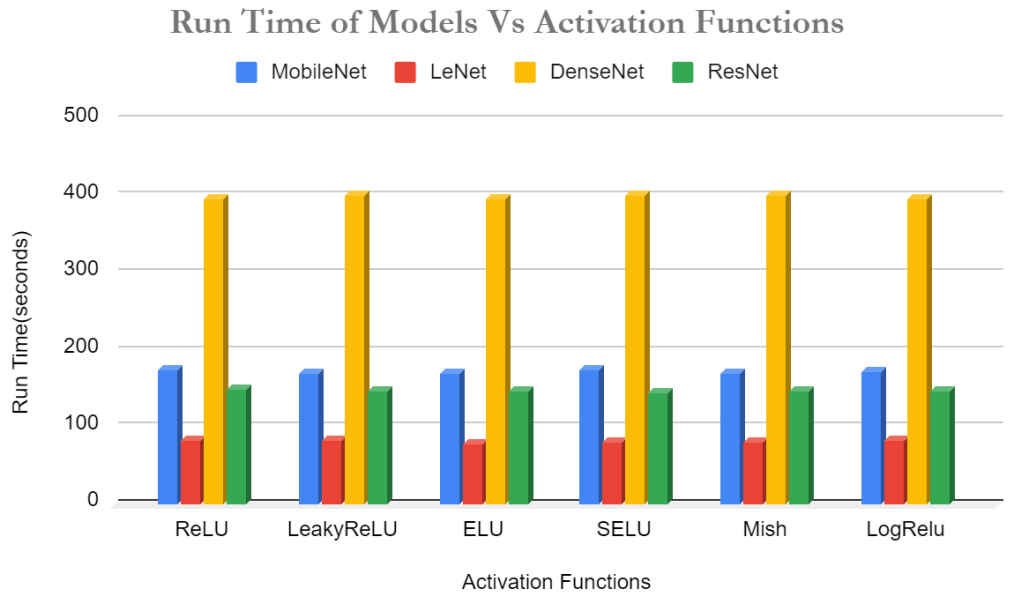

Figure 4.74::Run Time for Adam (0.01)

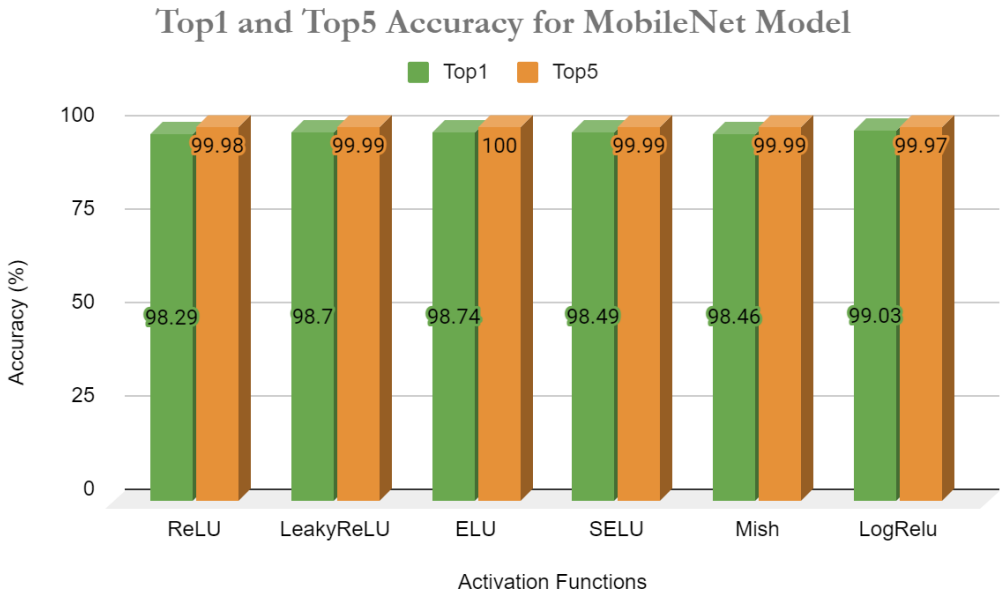

Figure 4.75:MobileNet with Adam(0.0005

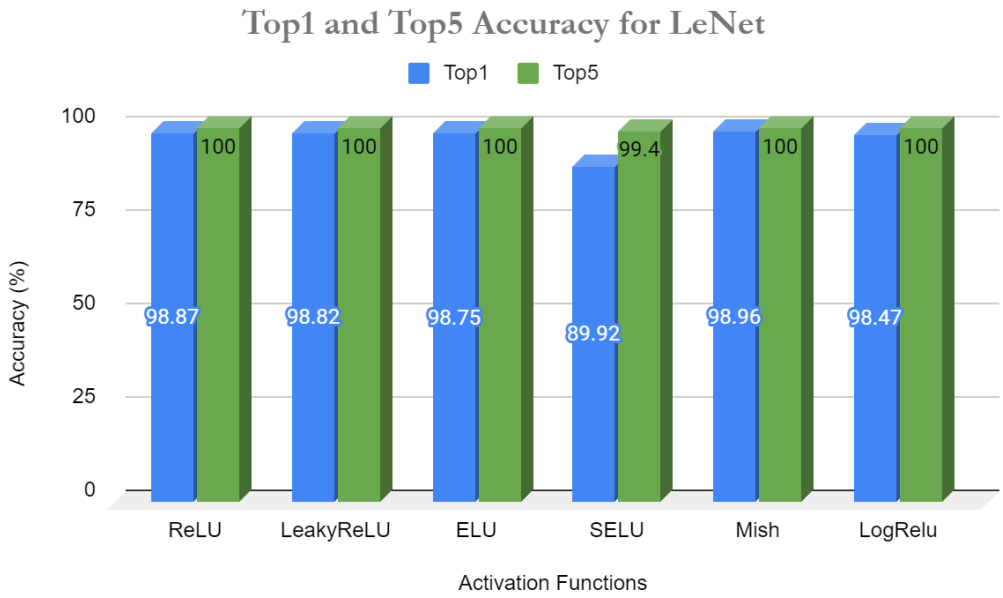

Figure 4.76:LeNet with Adam(0.0005)

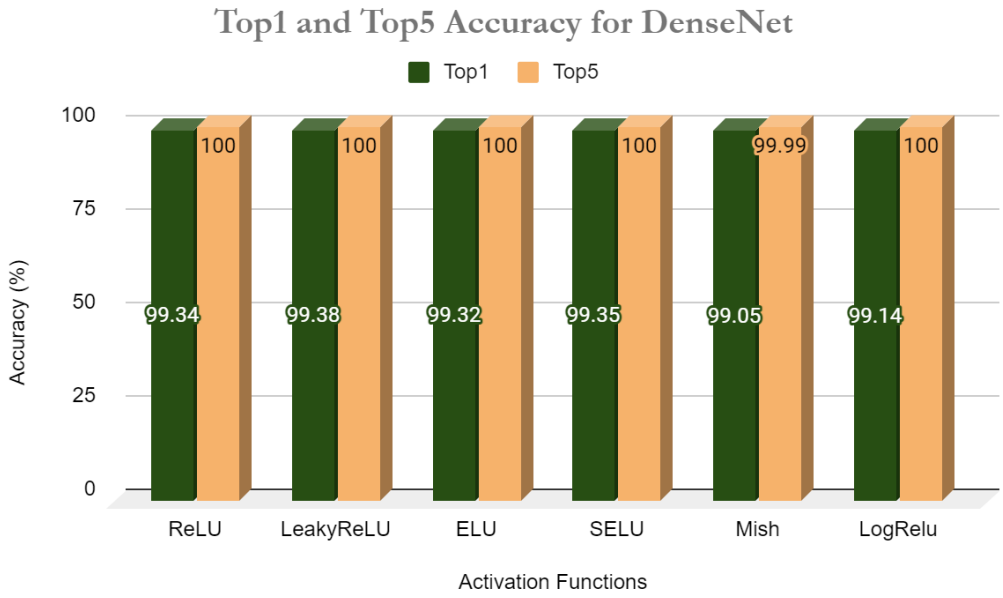

Figure 4.77:DenseNet with Adam(0.0005)

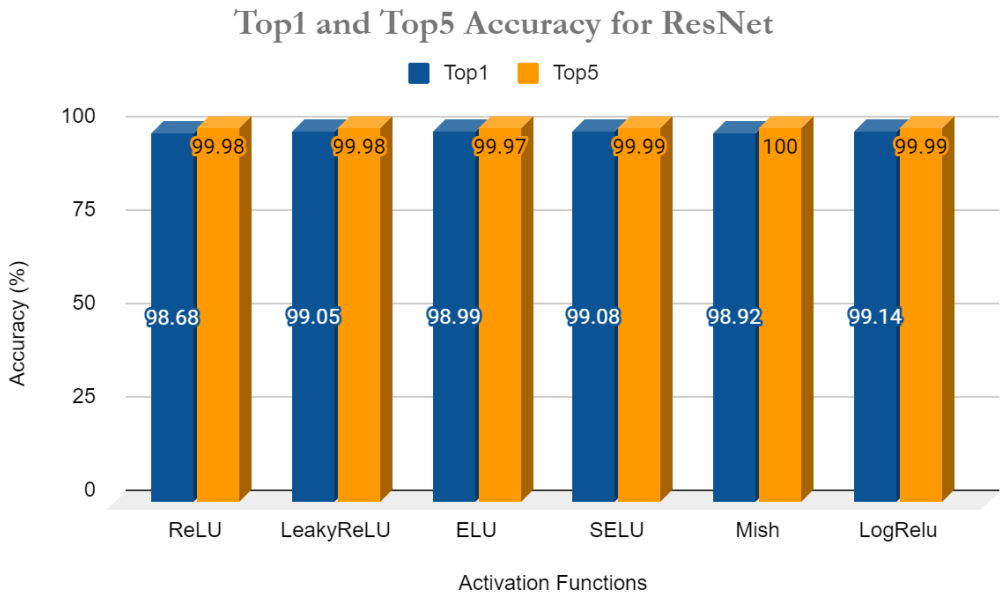

Figure 4.78:ResNet with Adam (0.0005)

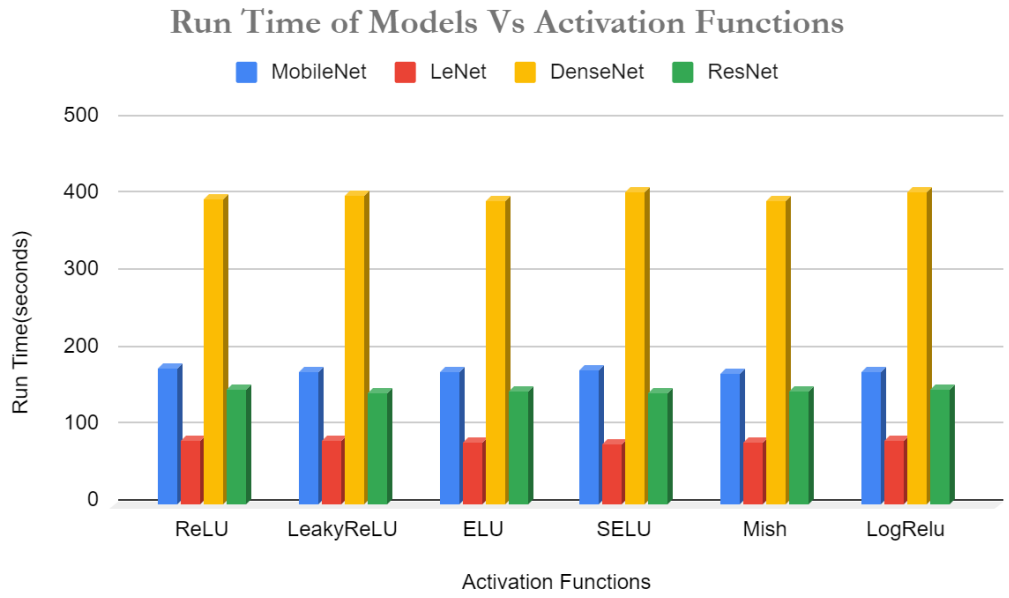

Figure 4.79:Run Time for Adam(0.0005)

Table 4. 28:Performance of SGD Optimizer with LR: 0.001

	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	98.00	99.94	171.83	34.28	68.90	81.32	98.47	99.993	382.48	88.03	90.30	134.14
LeakyReLU	97.77	99.99	167.54	17.79	84.34	80.88	98.39	99.93	389.01	97.50	99.89	129.30
ELU	98.00	99.99	169.89	85.98	98.88	79.50	98.63	99.95	382.34	97.22	99.94	130.43
SELU	97.95	99.96	169.73	89.92	99.40	78.81	98.32	99.96	386.52	97.60	99.92	129.71
Mish	98.02	99.97	166.66	11.47	75.54	80.85	98.42	99.97	382.34	97.58	99.87	130.68
LogRelu	97.85	99.98	169.54	35.92	84.27	81.65	98.27	99.95	380.51	97.45	99.90	156.05

Table 4. 29: Performance of SGD Optimizer with LR: 0.01

	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	99.09	100	172.44	96.85	99.96	81.18	99.04	99.98	383.13	99.08	100	133.29
LeakyReLU	99.13	99.99	167.20	97.29	99.98	81.32	99.11	100	389.22	98.87	99.99	130.73
ELU	98.86	99.99	169.63	97.43	99.98	78.90	99.28	100	380.82	98.95	99.99	131.65
SELU	98.95	99.98	168.17	97.55	99.99	79.55	99.16	99.99	389.52	98.83	99.99	132.22
Mish	98.47	99.95	167.65	97.17	99.95	80.50	99.24	100	387.22	98.89	100	129.57
LogRelu	98.85	99.99	168.39	96.94	99.99	82.06	98.27	99.95	380.51	98.78	99.97	153.00

Table 4. 30: Performance of SGD Optimizer with LR: 0.0005

	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	96.65	99.90	169.12	11.35	58.04	81.12	97.66	99.92	385.21	96.23	99.75	133.61
LeakyReLU	96.76	99.90	166.74	17.10	66.12	81.03	97.69	99.98	387.25	96.14	99.84	130.57
ELU	97.16	99.97	167.49	56.18	95.45	79.36	97.57	99.89	382.02	96.44	99.84	130.87
SELU	96.95	99.95	166.10	86.81	99.09	78.65	97.61	99.93	393.94	96.63	99.86	131.91
Mish	96.54	99.92	166.06	10.28	56.57	81.13	97.22	99.94	381.32	96.29	99.88	130.06
LogRelu	96.34	99.91	168.32	25.59	61.92	82.24	97.78	99.88	382.26	96.56	99.87	152.18

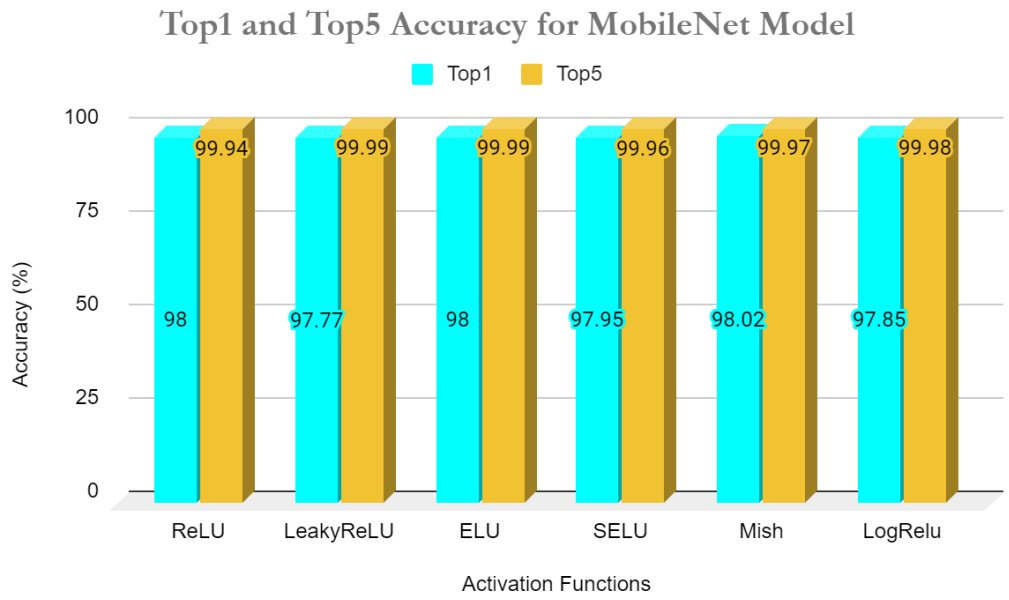

Figure 4.80:MobileNet with SGD (0.001)

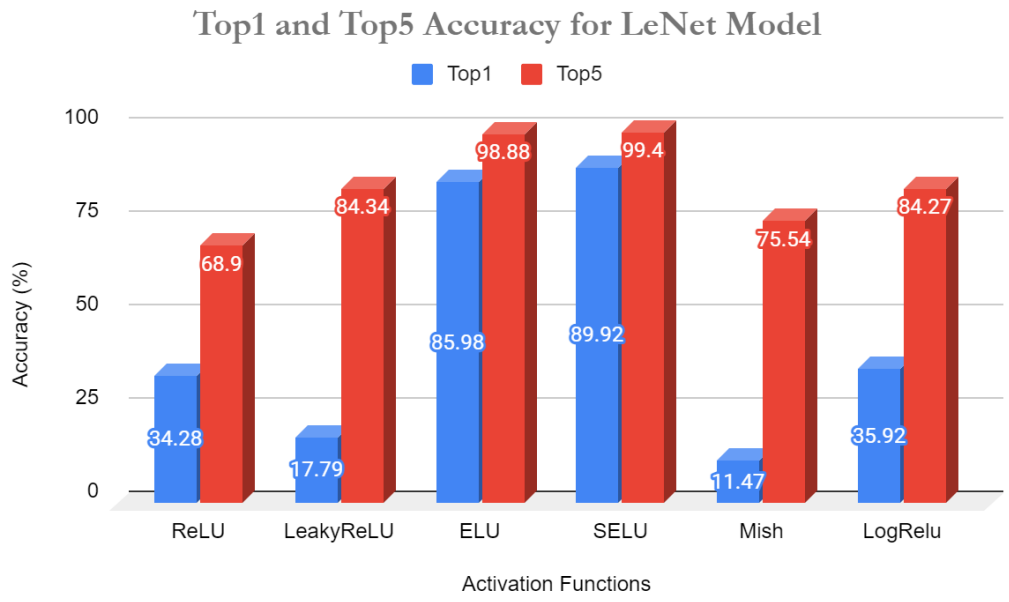

Figure 4.81:LeNet with SGD(0.001)

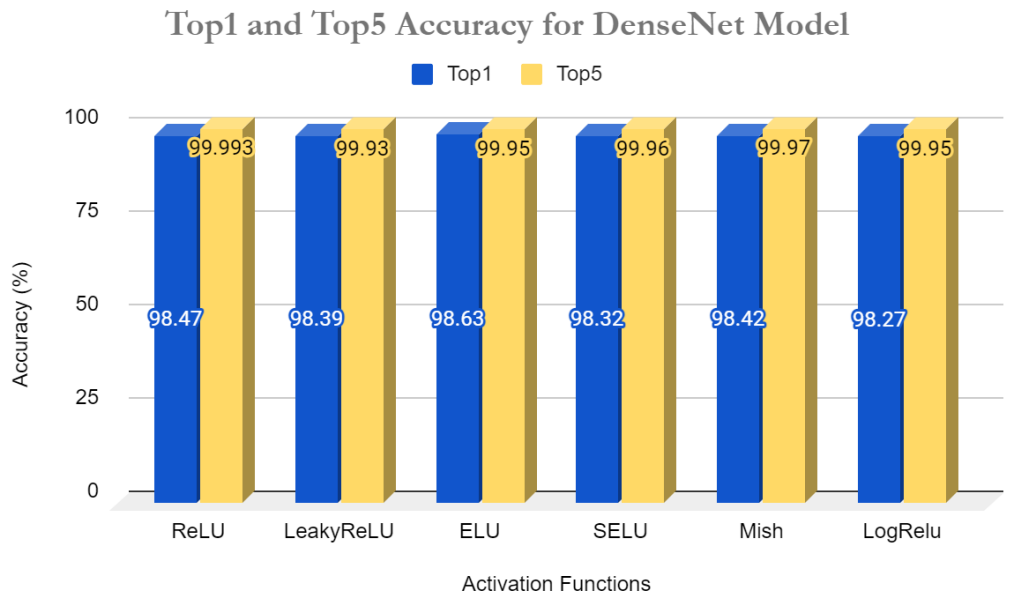

Figure 4.82:DenseNet with SGD (0.001)

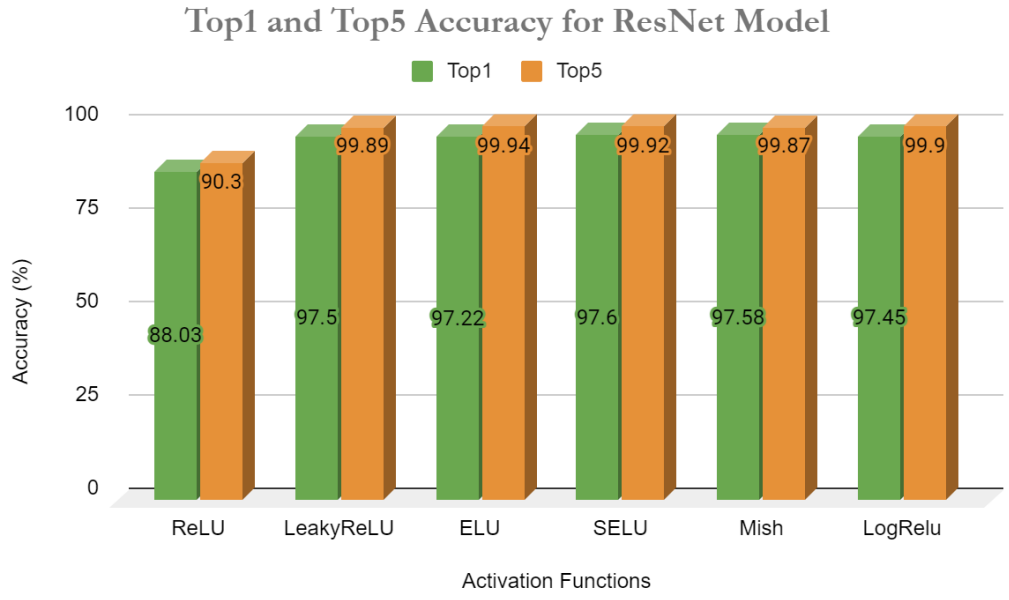

Figure 4.83:ResNet with SGD(0.001)

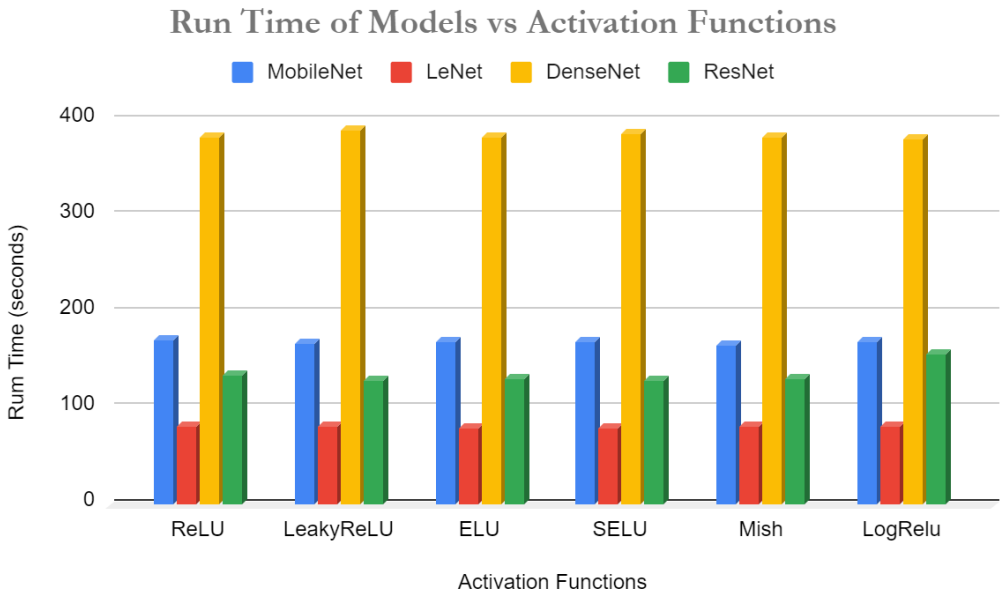

Figure 4.84:Run Time for SGD (0.001)

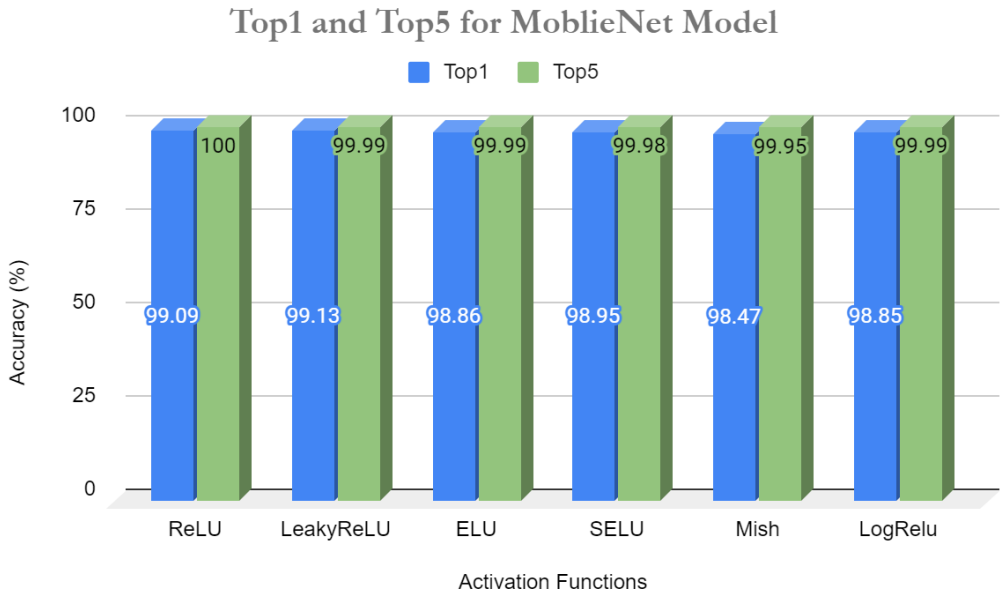

Figure 4.85:MobileNet with SGD(0.01)

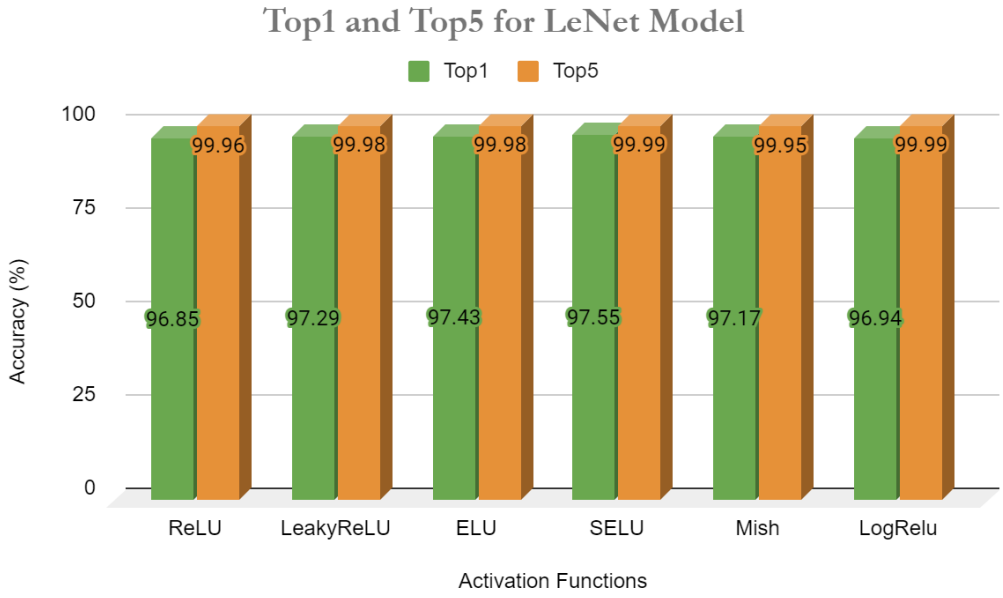

Figure 4.86: LeNet with SGD (0.01)

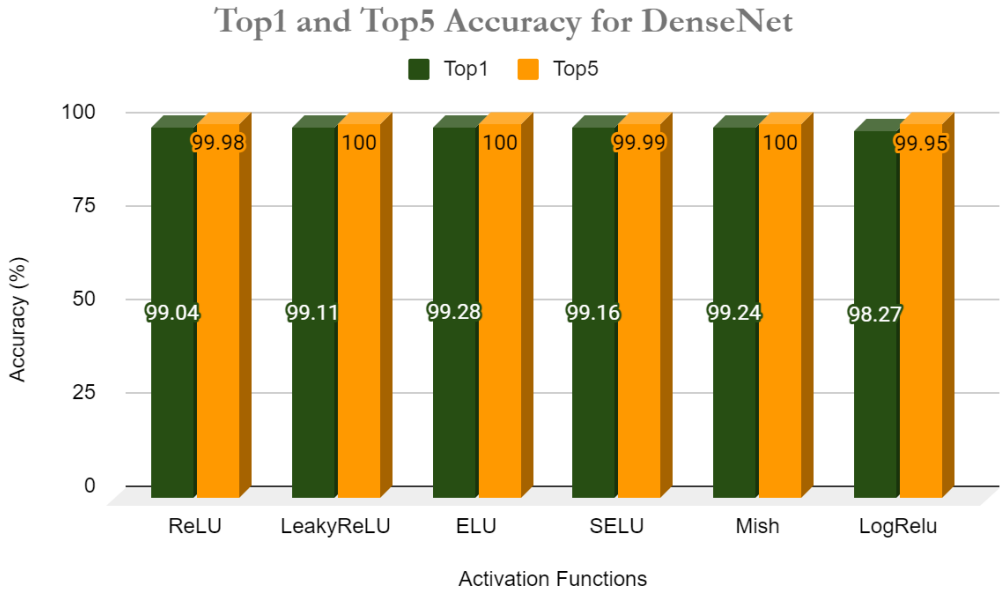

Figure 4.87:DenseNet with SGD(0.01)

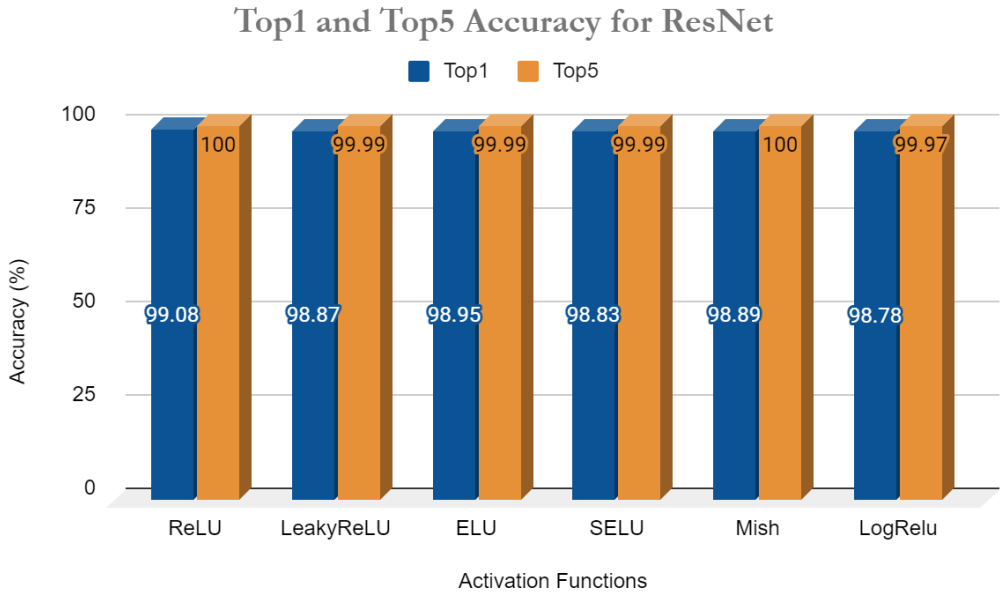

Figure 4.88:ResNet with SGD(0.01)

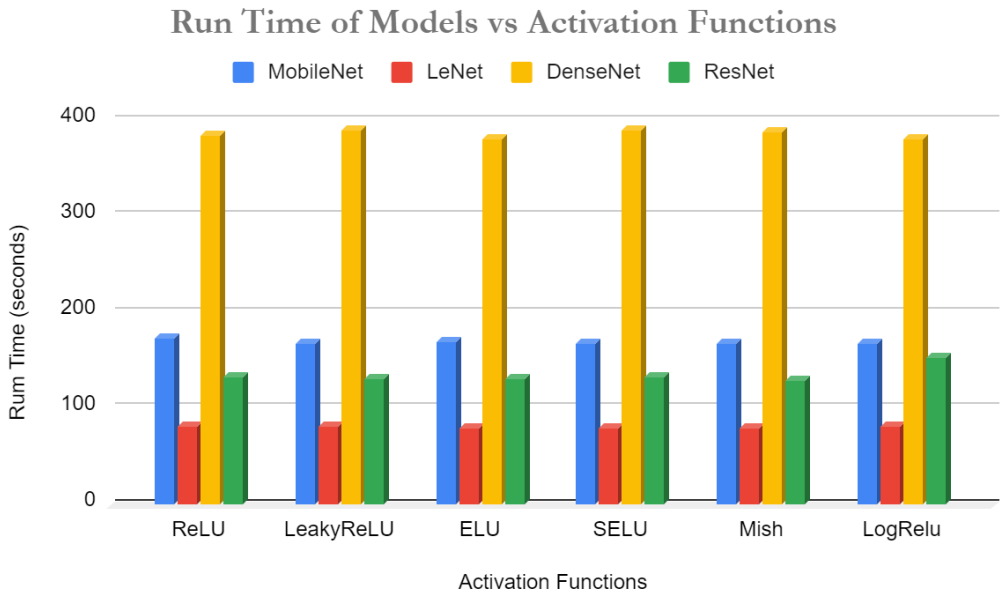

Figure 4.89:Run Time with SGD (0.01)

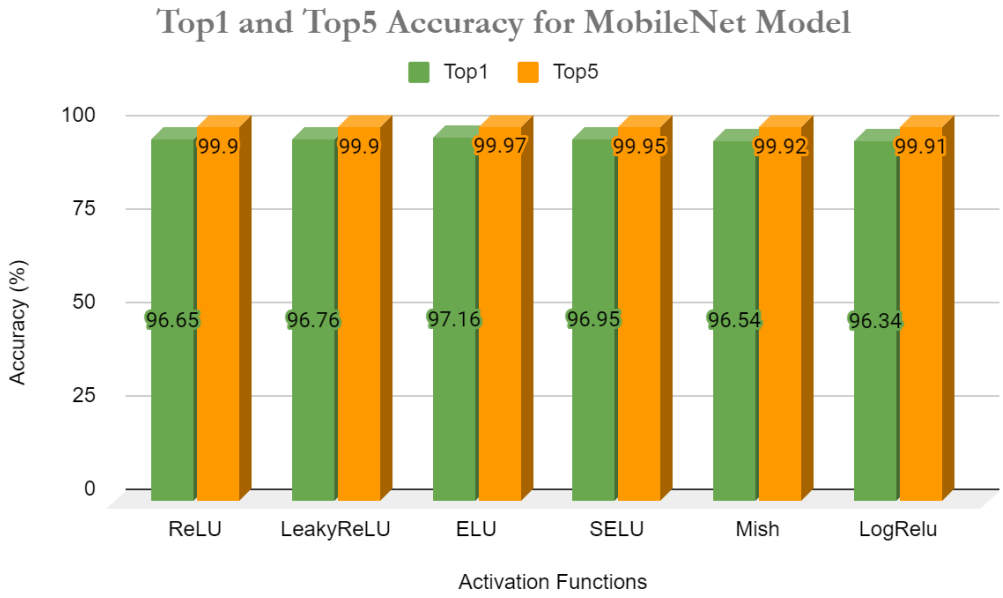

Figure 4.90:MobileNet with SGD(0.0005)

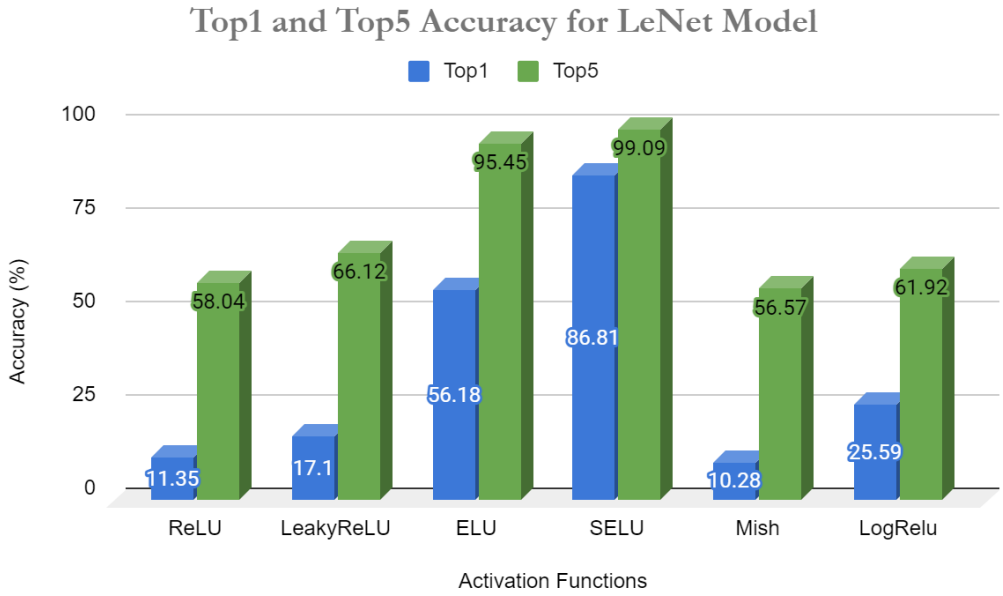

Figure 4.91:LeNet with SGD(0.0005)

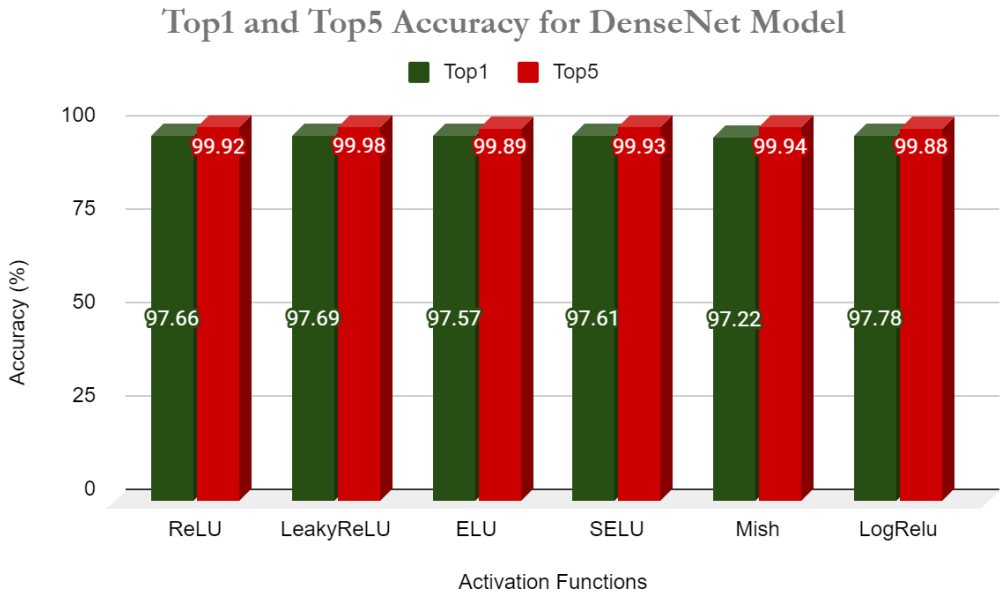

Figure 4.92:DenseNet with SGD (0.0005)

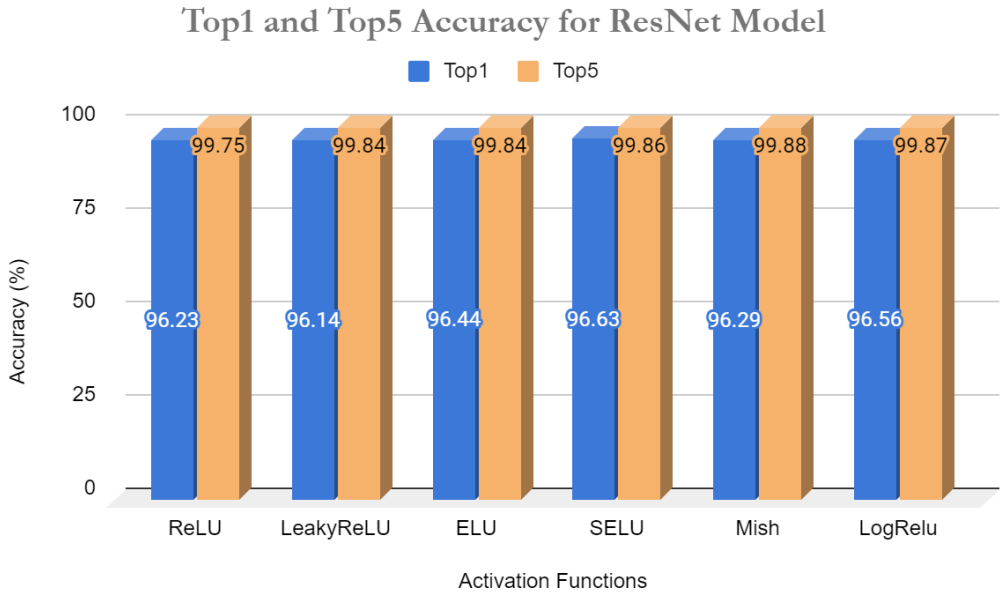

Figure 4.93:ResNet with SGD (0.0005)

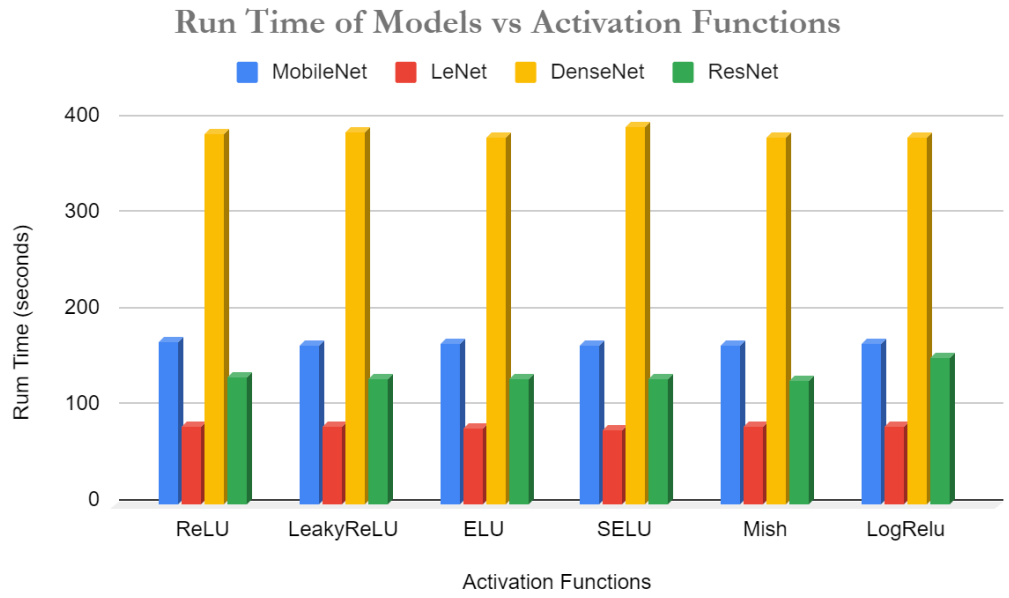

Figure 4.94:Run Time with SGD (0.0005)

Table 4.31: Performance of RMSprop Optimizer with LR: 0.001

	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	9.80	51.39	171.38	98.77	99.99	80.75	21.10	51.39	390.59	29.25	61.02	142.64
LeakyReLU	98.02	99.93	166.55	98.50	99.99	82.38	99.25	100	392.85	98.75	99.94	140.29
ELU	98.91	99.96	167.87	98.88	100	78.56	99.18	100	385.08	97.54	99.99	140.65
SELU	97.56	98.77	168.31	98.71	99.98	79.31	9.80	51.39	397.35	98.90	100	140.19
Mish	75.86	97.80	169.11	98.95	99.99	83.07	19.27	60.78	392.01	67.98	99.90	139.29
LogRelu	83.18	99.06	170.51	99.01	99.99	82.62	99.24	99.99	392.99	9.80	51.39	162.27

Table 4.32: Performance of RMSprop Optimizer with LR: 0.01

	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	9.80	51.39	173.14	97.15	99.72	81.55	9.80	51.39	400.45	9.80	51.39	141.82
LeakyReLU	97.11	99.92	168.13	97.36	99.84	81.70	98.56	99.99	390.79	98.79	99.99	140.98
ELU	9.81	51.39	168.49	78.28	97.48	79.02	96.22	97.60	392.59	96.27	99.02	139.88
SELU	87.45	99.59	168.19	96.50	99.26	79.61	9.80	51.39	387.44	98.81	100	140.81
Mish	24.53	88.19	168.359	97.22	99.89	82.54	9.80	51.39	392.97	92.90	98.64	139.30
LogRelu	9.80	51.39	169.42	97.39	99.76	82.35	99.22	100	391.21	9.80	51.39	162.27

Table 4.33: Performance of RMSprop Optimizer with LR: 5e-4

	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	41.80	62.55	169.88	98.53	99.99	82.13	50.42	60.91	396.14	30.57	60.80	142.81
LeakyReLU	98.78	100	167.92	98.71	100	80.96	99.32	99.99	391.16	99.07	99.97	140.67
ELU	98.95	99.97	170.42	98.69	100	79.58	99.46	100	393.80	99.28	99.98	138.62
SELU	98.88	99.99	171.26	98.76	100	79.01	31.40	51.39	389.41	99.13	100	138.83
Mish	98.11	99.99	169.18	97.90	99.97	86.32	39.28	80.91	393.38	97.57	99.64	139.15
LogRelu	99.19	99.99	168.90	99.05	100	82.18	99.41	100	384.64	39.61	70.95	162.24

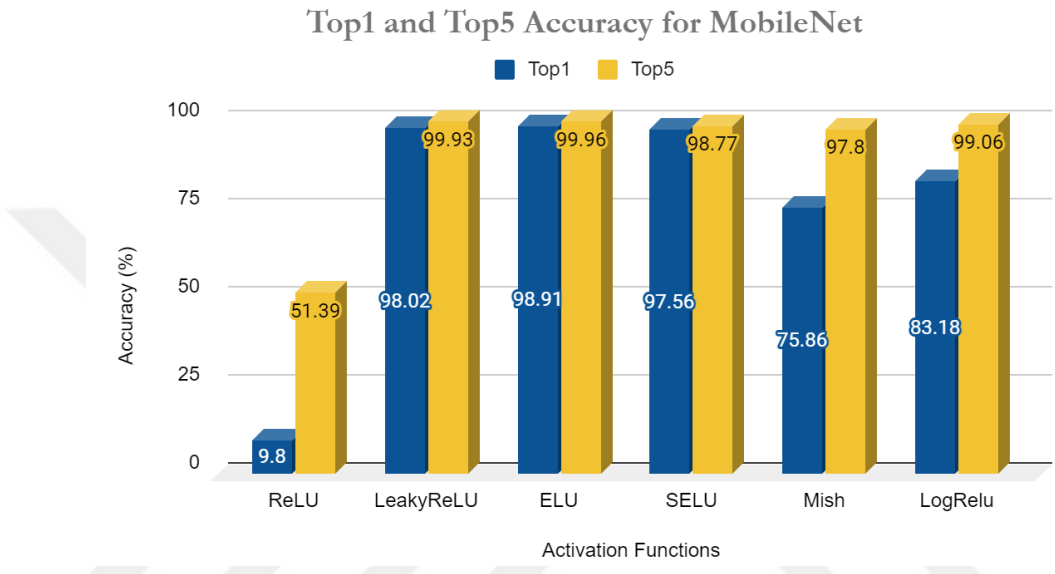

Figure 4.96:MobileNet with RMSprop (0.001)

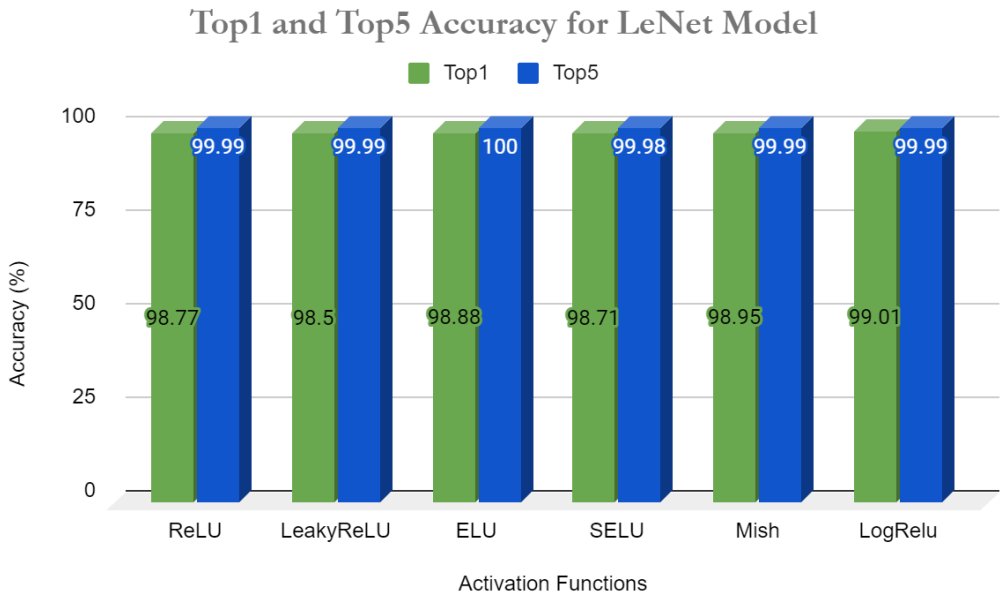

Figure 4.95:LeNet with RMSprop (0.001)

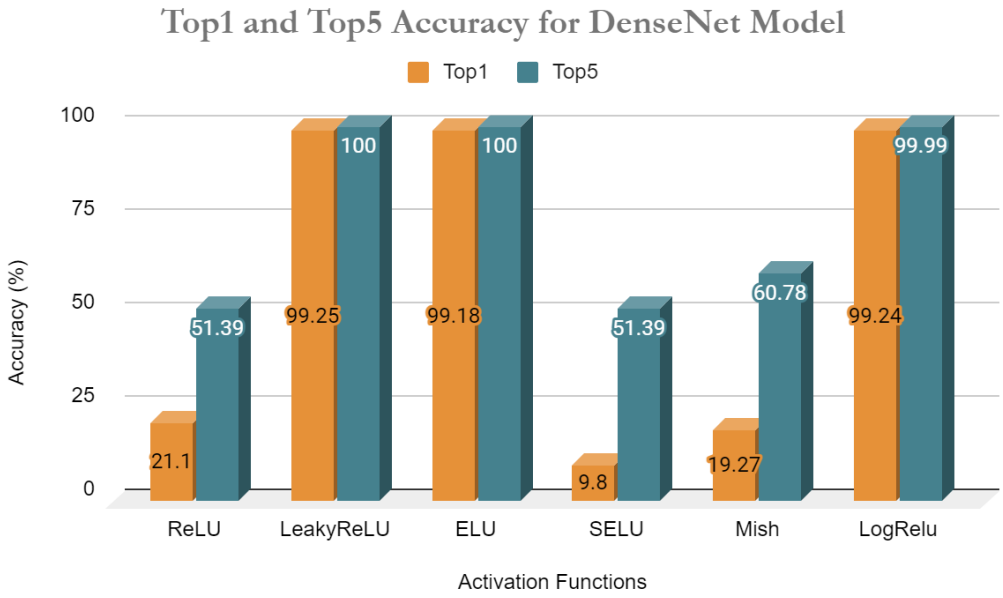

Figure 4.97: DenseNet with RMSprop(0.001)

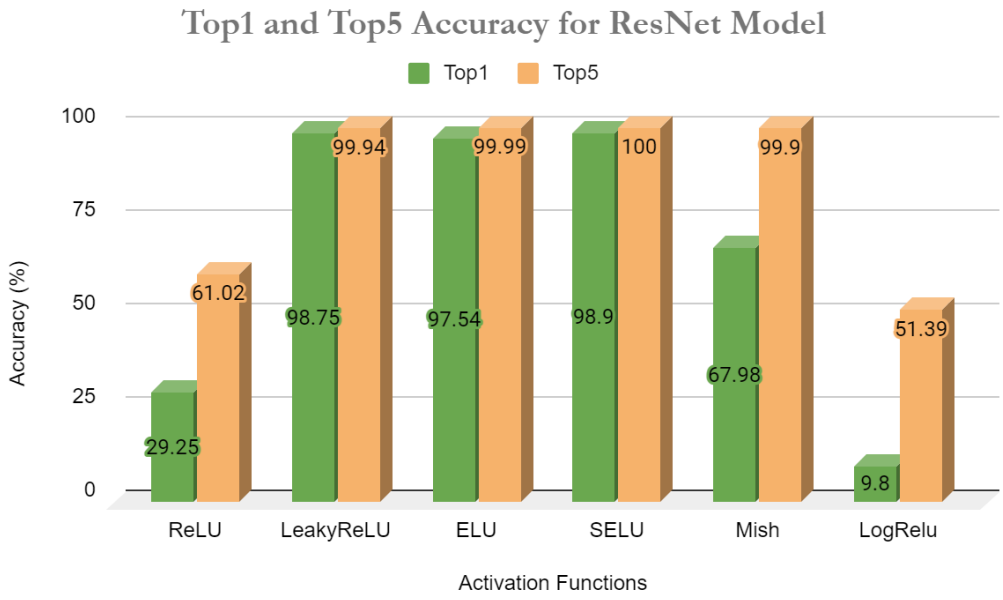

Figure 4.98:ResNet with RMSprop (0.001)

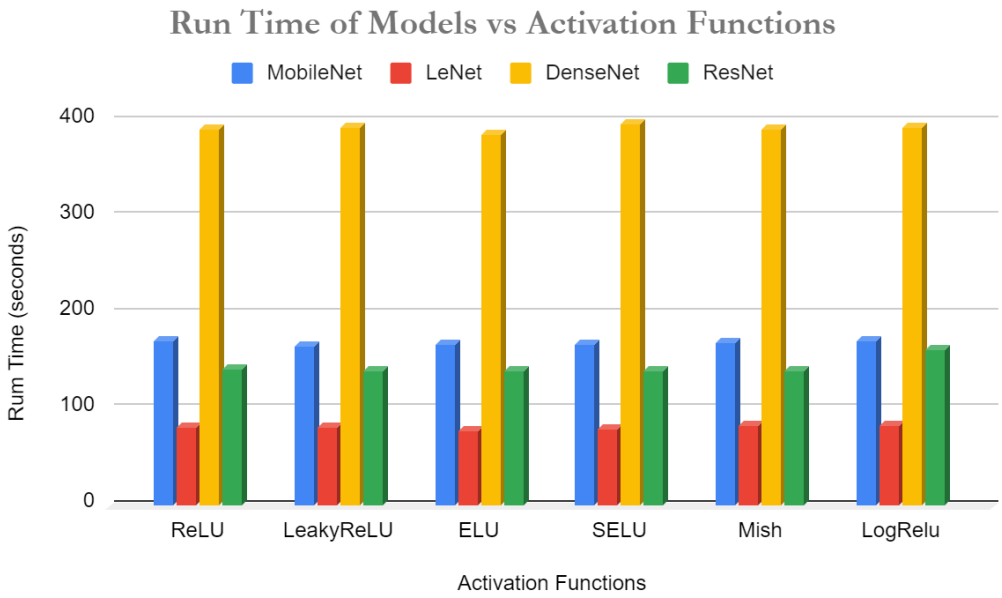

Figure 4.99:Run Time for RMSprop (0.001)

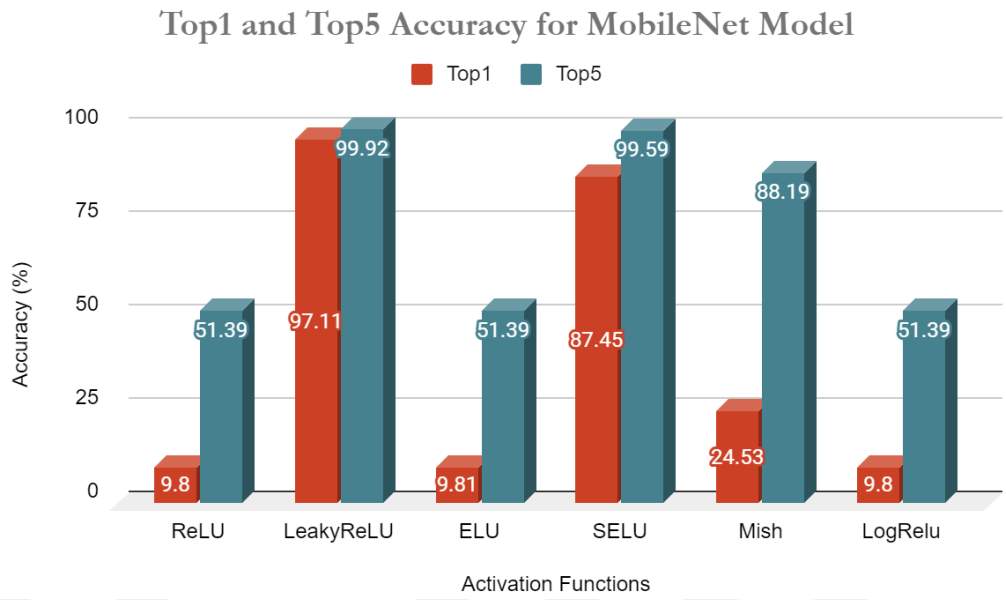

Figure 4.100:MobileNet with RMSprop(0.01)

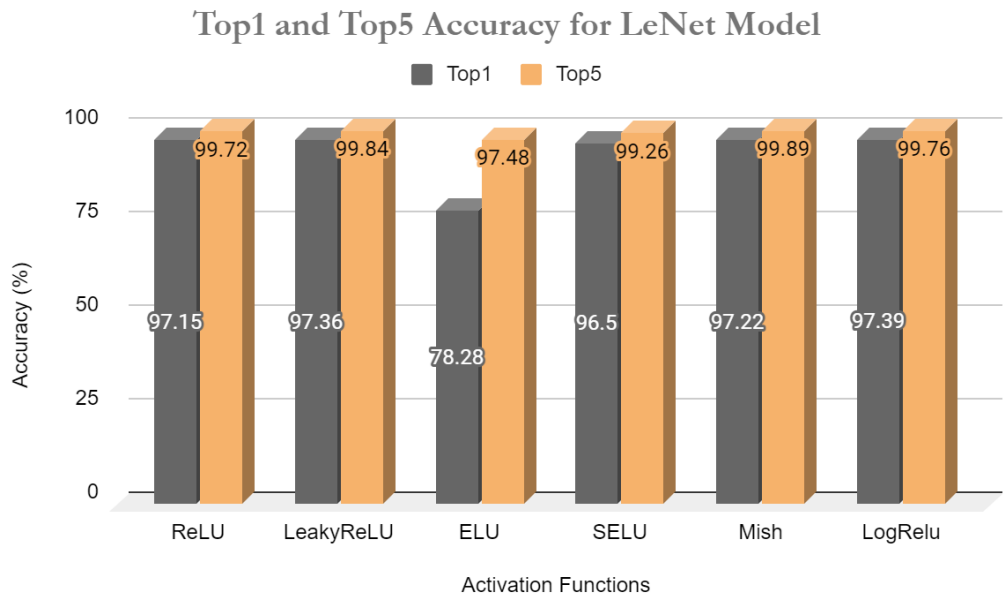

Figure 4.101:LeNet with RMSprop (0.01)

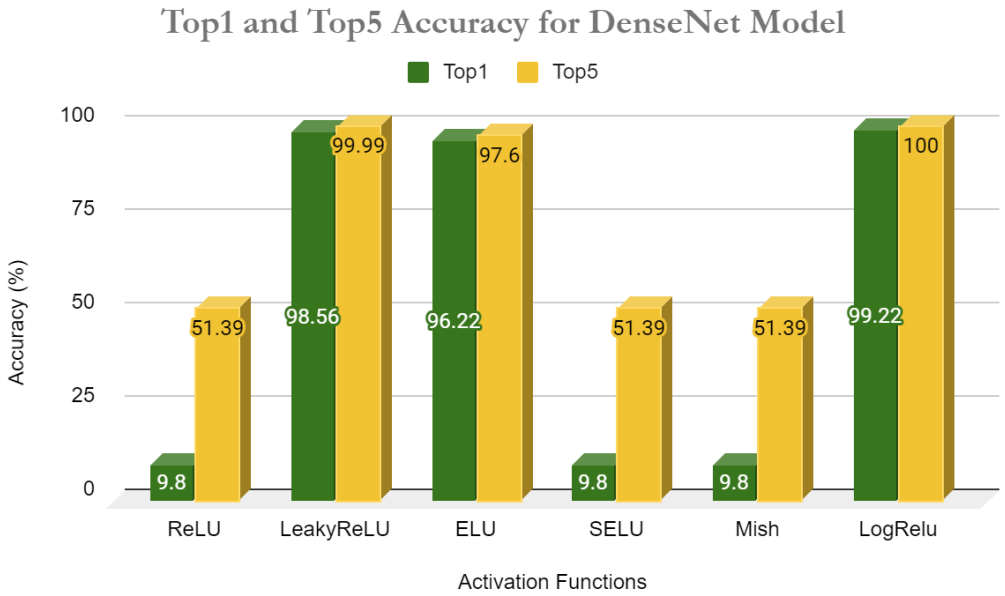

Figure 4.102:DenseNet with RMSprop (0.01)

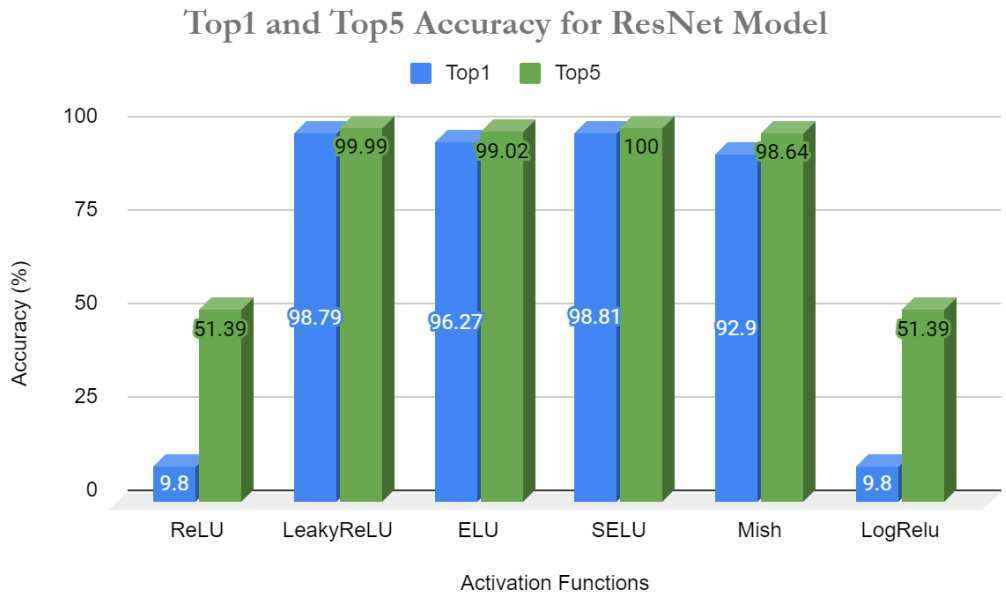

Figure 4.103:ResNet with RMSprop (0.01)

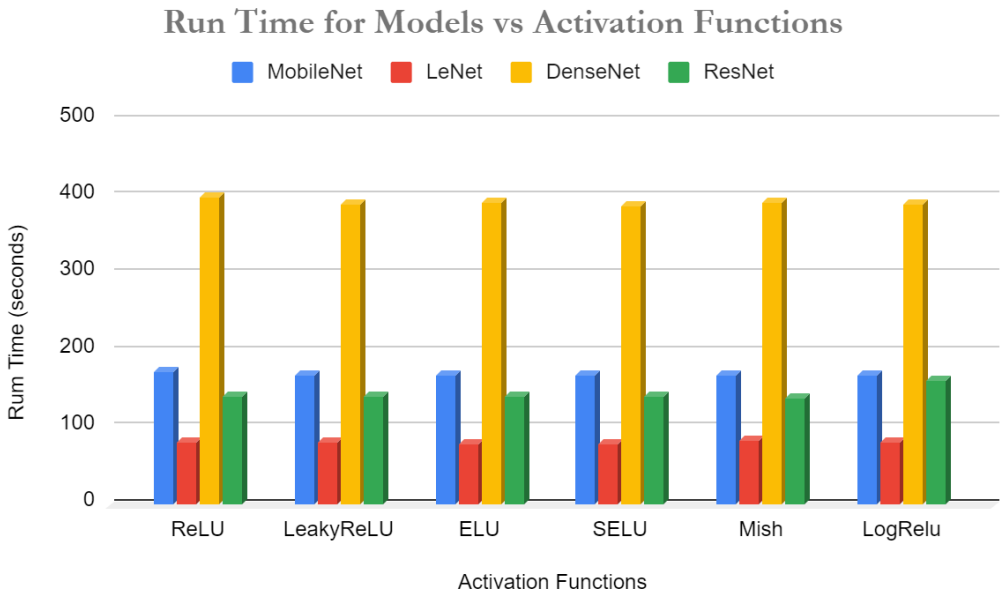

Figure 4.104:Run Time with RMSprop (0.01)

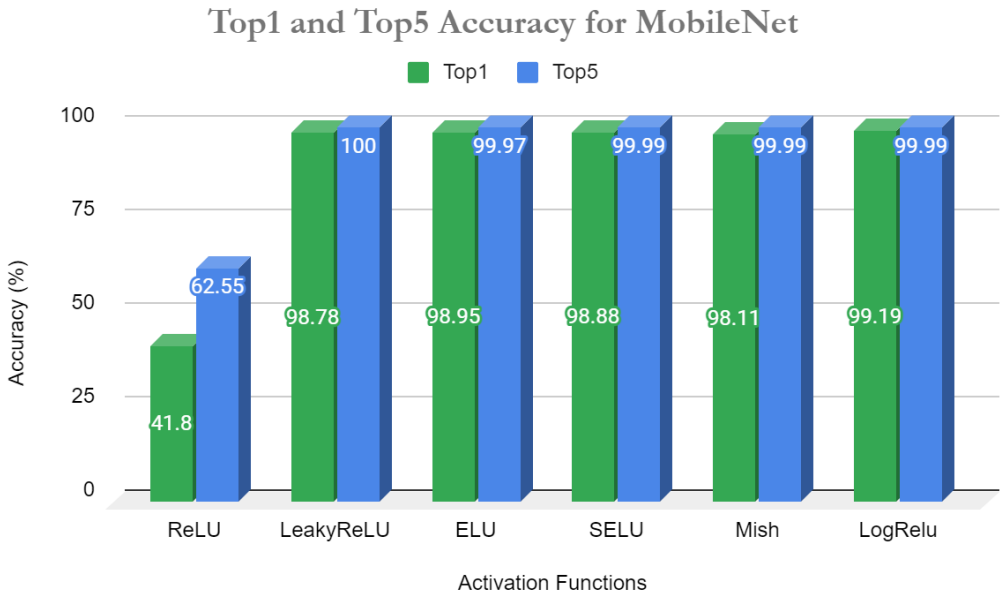

Figure 4.105::MobileNet for RMSprop(0.0005)

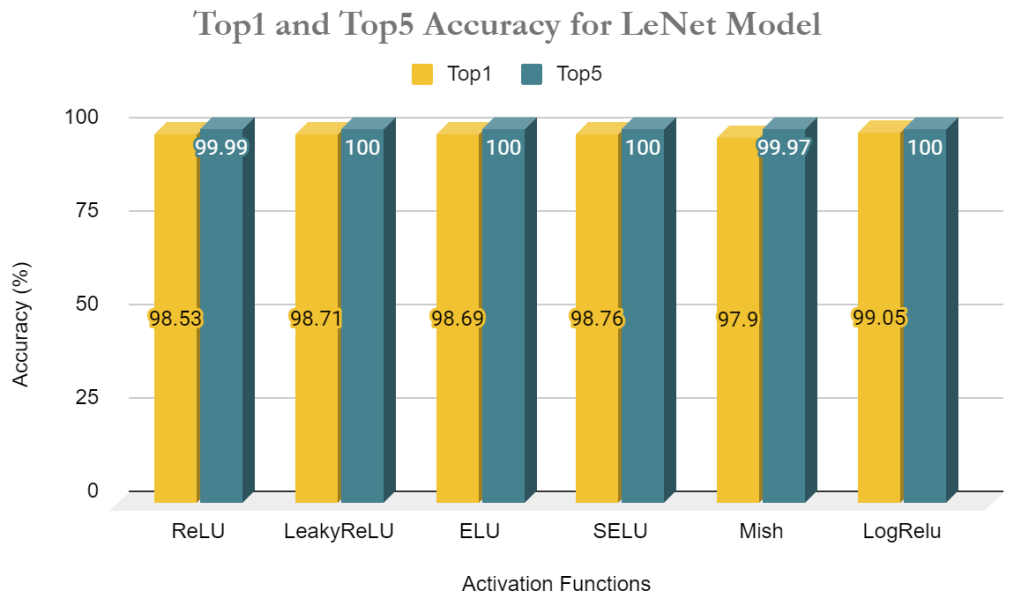

Figure 4.106:LeNet with RMSprop (0.0005)

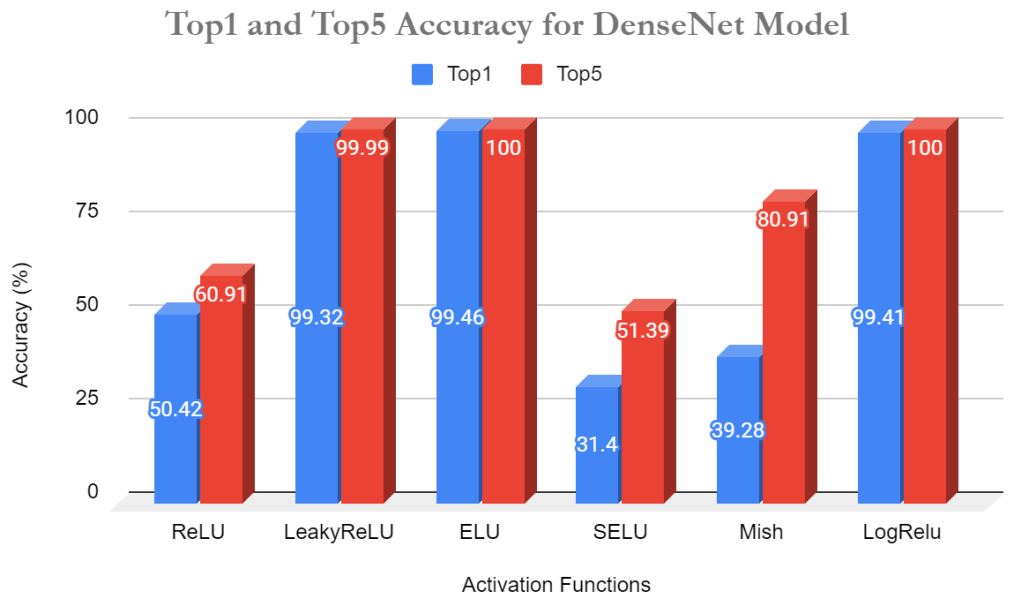

Figure 4.107:DenseNet with RMSprop (0.0005)

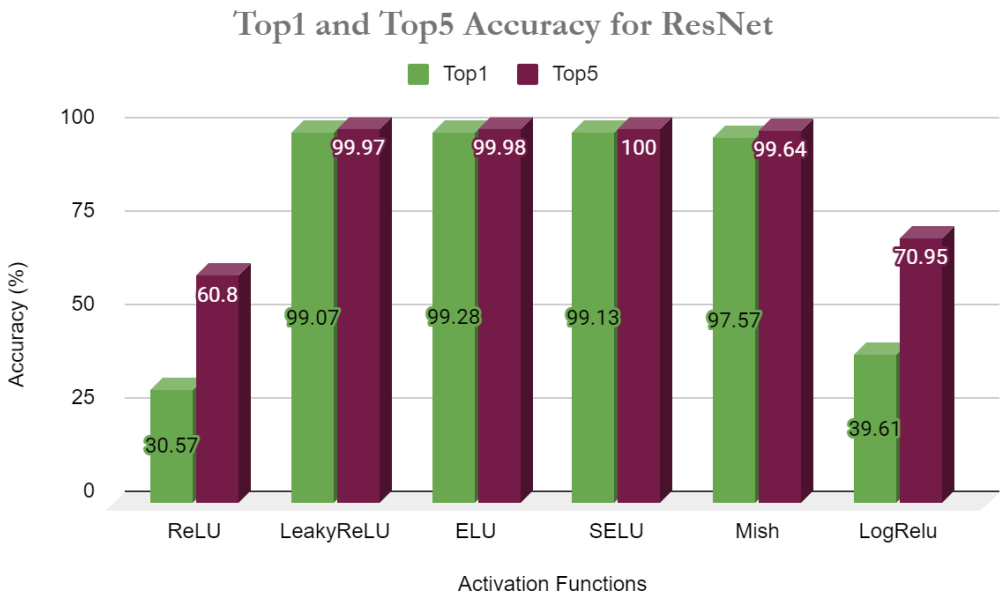

Figure 4.108:ResNet with RMSprop (0.0005)

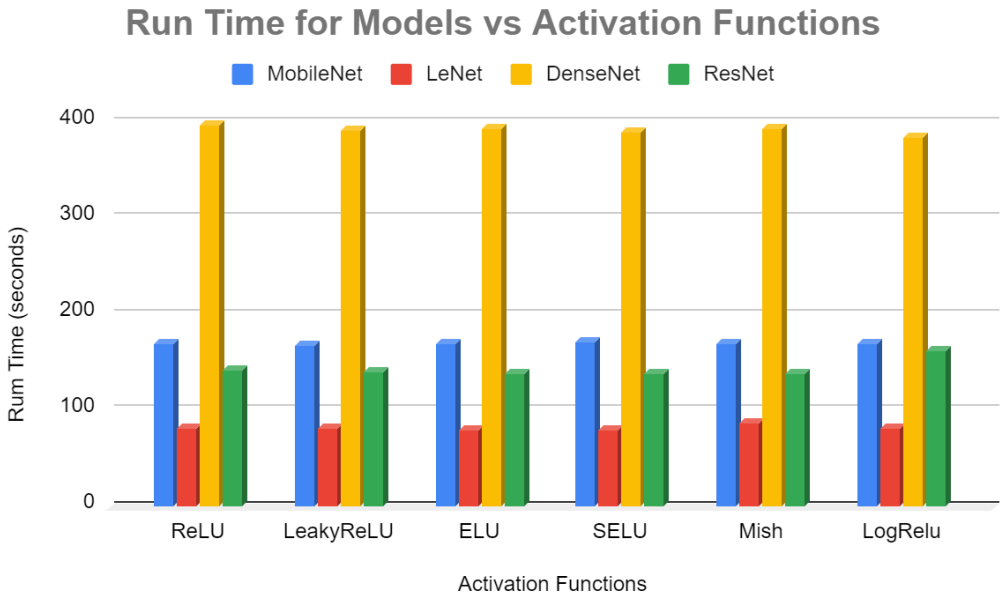

Figure 4.109:Run Time with RMSprop (0.0005)

4.2.4. The DiffCNN Classification Performance on the MINST Dataset

We present results for LogRelu, ReLU, LeakyReLU, ELU, SELU, and Mish activation functions applied to the DiffCNN model. These assessments cover various optimizers (Adam, SGD, and RMSprop) with learning rates (0.001, 0.01, and 0.0005) on the MINST dataset.

Across different learning rates, LogRelu demonstrates its effectiveness in maintaining competitive Top1 Accuracy with slight advantages over other activation functions, ranging from 0.01% to 0.48% improvement, particularly noticeable at a learning rate of 0.001. However, it is worth

noting that the differences in Top5 Accuracy are generally minimal, hovering around 0.00%. At a learning rate of 0.01, all activation functions perform equally poorly, indicating limited learning due compatibility of these parameters (Adam with 0.01). Similarly, at a learning rate of 0.0005, LogRelu's performance remains on par with others, with differences in accuracy close to zero as shown in Table 4.34 as well as visual representation in Figure 4.110 through Figure 4.113. In addition, Table 4.35 shows that LogRelu and the other activation functions generally exhibit slightly lower performance in terms of both Top1 and Top5 Accuracy with learning rates of 0.001 and 0.0005. However, with a learning rate of 0.01, Top1 and Top5 Accuracy are better ranging from (98.00 -98.87) and (99.99 - 100%) respectively as seen in Figure 4.114 through Figure 4.117. Table 4.36 and Figure 4.118 through Figure 4.120 highlights that at a learning rate of 0.001, LogRelu displays substantial enhancements, with Top1 Accuracy improvements ranging from 0.03% to a remarkable 87.86%. Additionally, with 0.0005, LogRelu competently competes with other activations, with minor differences in both accuracy metrics. Notably, there are incompatibility issues observed at a learning rate of 0.01 with RMSprop Optimizer as observed with the drop in top1 and top5 accuracy.

Table 4.34: DiffCNN with Adam Optimizer

LR	0.001			0.01			0.0005		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	98.97	100	279.77	11.35	50.96	270.08	99.43	100	270.95
LeakyReLU	99.20	99.98	294.0	11.35	51.12	284.99	99.30	99.99	287.84
ELU	98.99	99.96	258.91	11.35	51.10	285.40	99.23	99.99	259.19
SELU	98.88	99.96	258.98	11.35	51.63	260.09	99.15	100	259.10
Mish	98.73	99.98	288.68	11.35	51.87	279.05	99.19	100	281.21
LogRelu	99.21	99.99	111.99	11.35	51.62	281.00	99.16	100	279.42

Table 4.35: DiffCNN with SGD Optimizer

LR	0.001			0.01			0.0005		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	11.35	51.58	292.52	98.72	100	277.91	18.13	51.17	282.72
LeakyReLU	20.62	52.43	278.77	98.57	99.99	270.01	15.44	51.84	270.05
ELU	25.28	69.54	257.57	98.87	99.99	258.94	26.84	51.99	258.09
SELU	56.10	89.93	258.79	98.83	100	258.57	40.11	80.33	258.68
Mish	22.85	54.27	277.80	98.49	99.99	277.55	9.58	49.87	272.62
LogRelu	10.28	51.12	276.89	98.00	100	274.41	11.51	51.60	275.23

Table 4.36:DiffCNN with RMSprop Optimizer

LR	0.001			0.01			0.0005		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	11.35	52.14	277.68	11.35	52.14	277.68	99.31	99.99	281.43
LeakyReLU	98.98	99.97	258.89	9.58	51.62	259.11	99.22	99.97	257.75
ELU	98.20	99.96	259.52	11.35	51.62	259.62	99.19	100	258.14
SELU	98.74	99.93	260.72	11.35	51.40	261.93	98.76	99.94	259.50
Mish	99.18	99.97	275.14	11.35	52.14	281.46	99.23	99.99	278.73
LogRelu	99.21	100	166.07	11.35	51.79	279.73	99.31	99.99	167.20

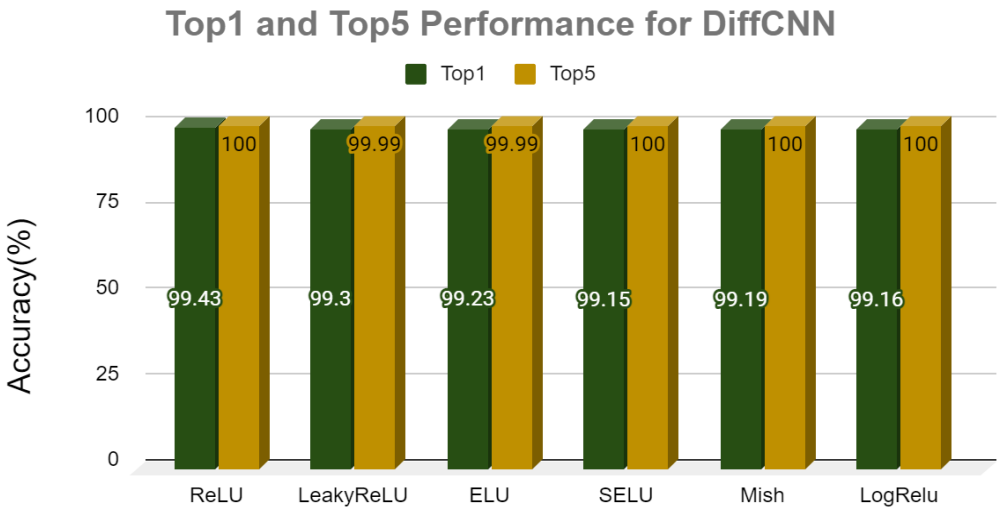

Figure 4.110:DiffCNN with Adam (0.001)

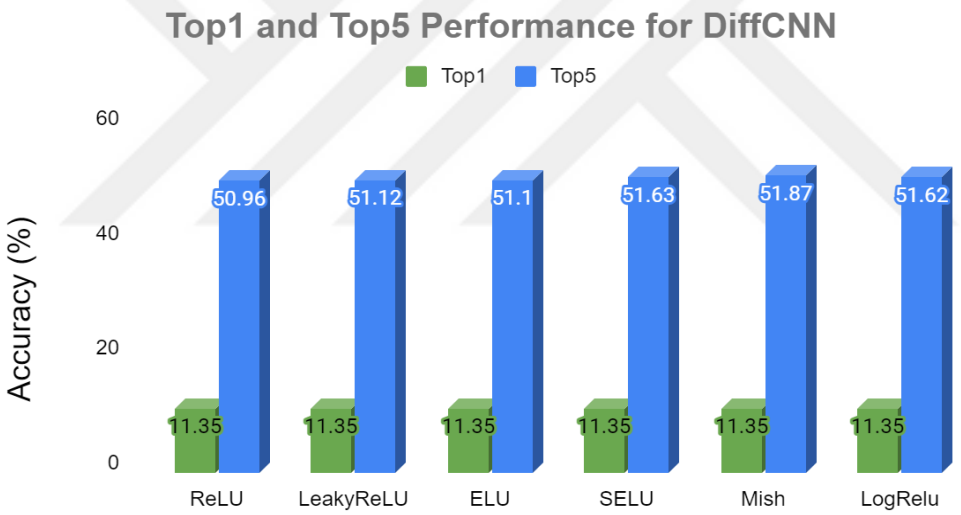

Figure 4.111:DiffCNN with Adam (0.01)

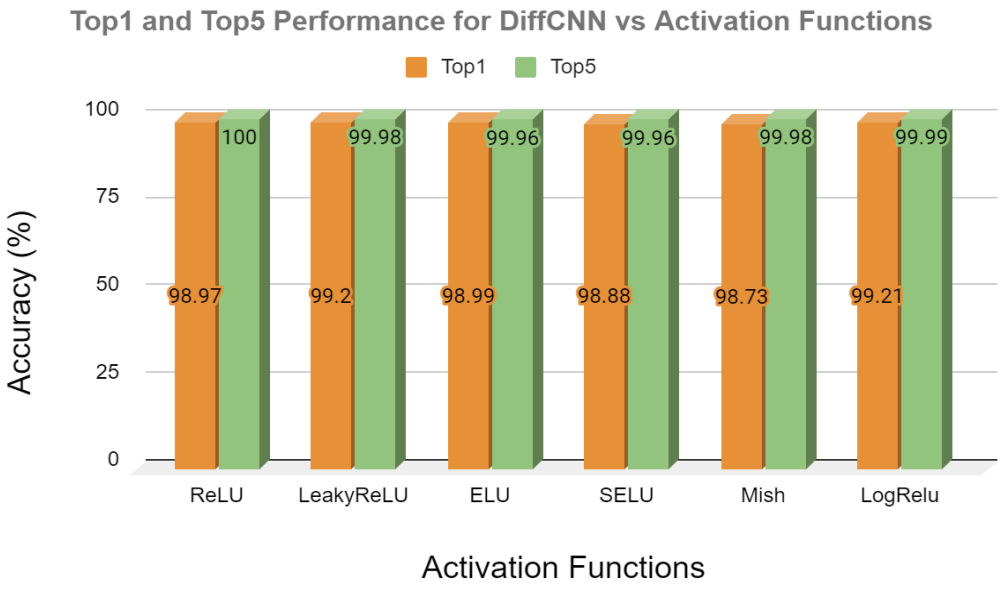

Figure 4. 112:DiffCNN with Adam (0.0005)

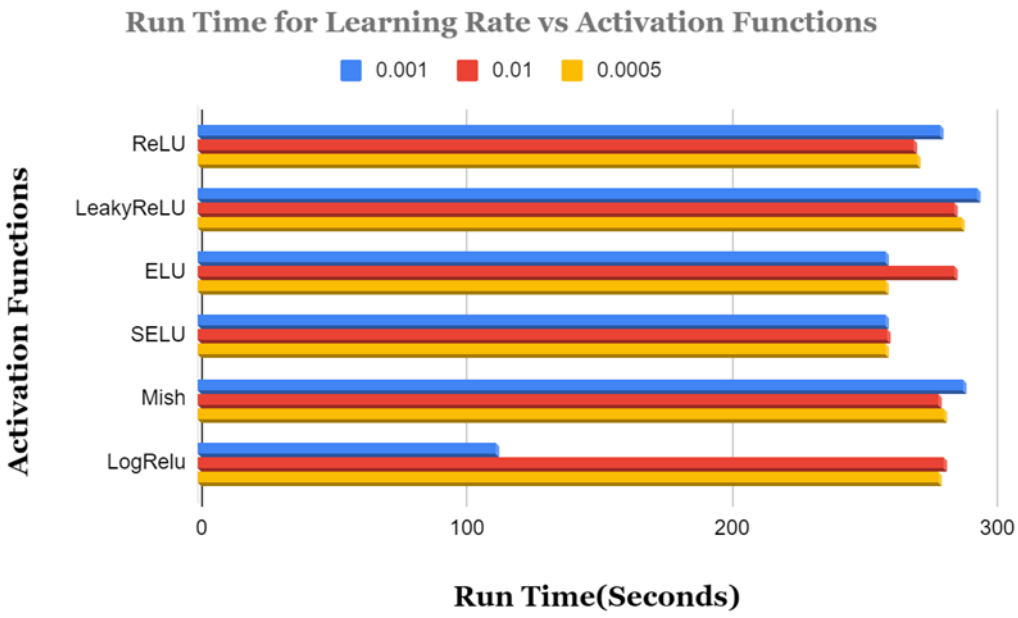

Figure 4.113:Run Time with DiffCNN with Adam Optimizer

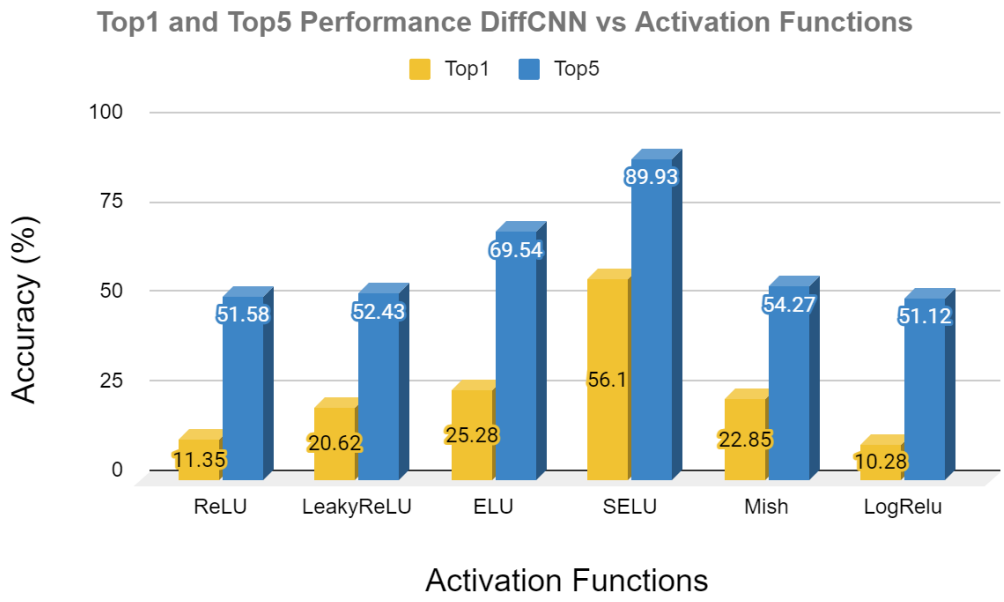

Figure 4.114:DiffCNN with SDG (0.001)

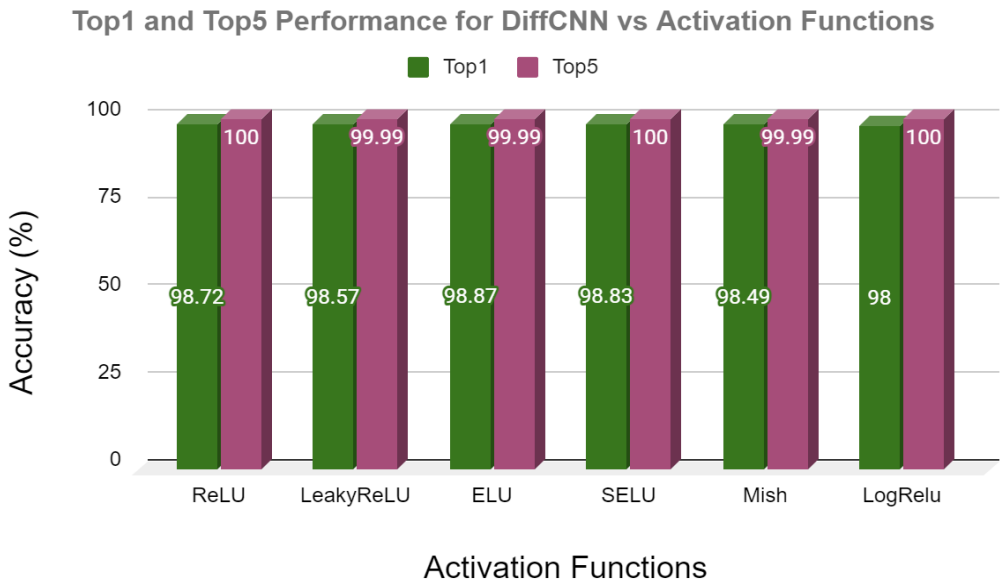

Figure 4.115:DiffCNN with SDG (0.01)

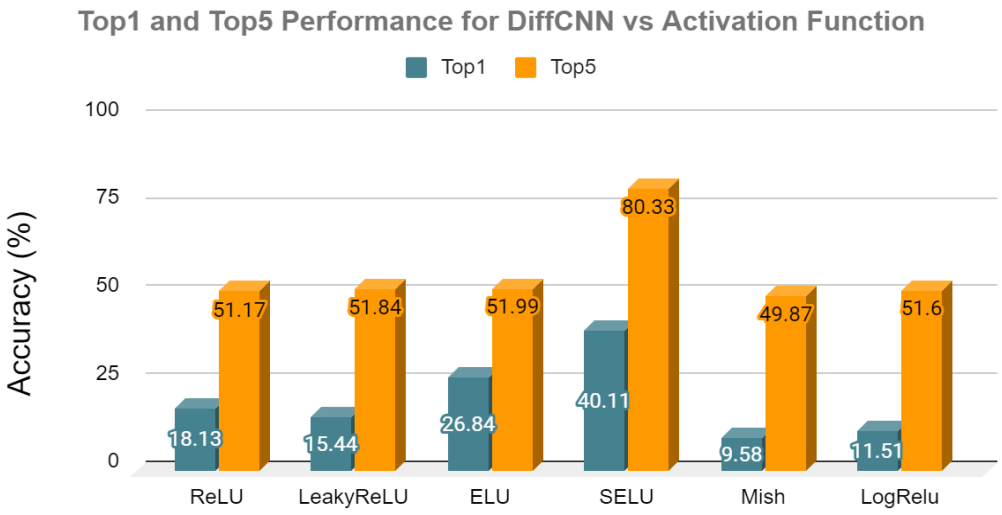

Figure 4.116: DiffCNN with SDG (0.0005)

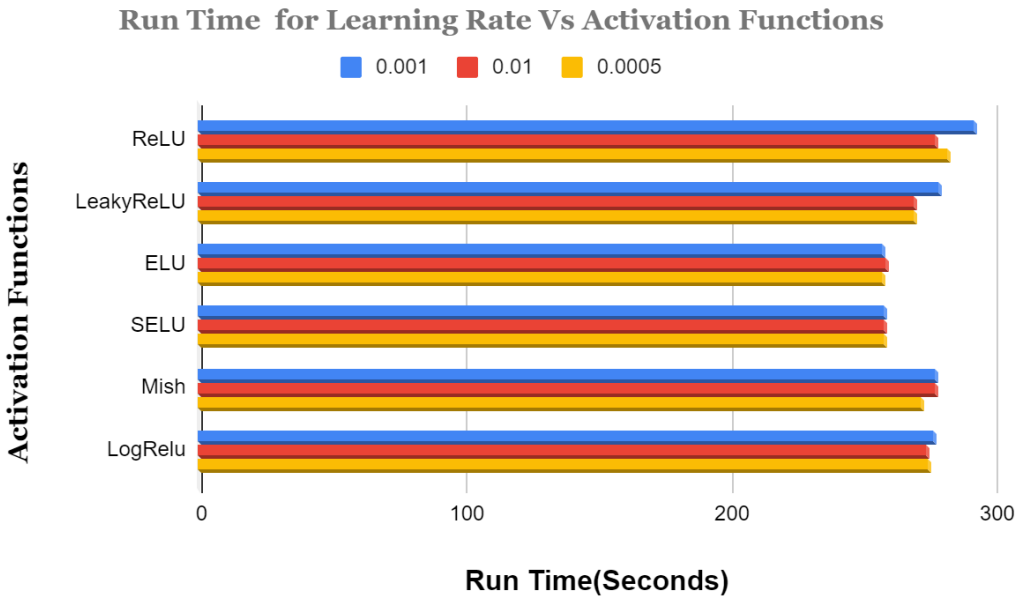

Figure 4.117: Run Time with DiffCNN with SGD Optimizer

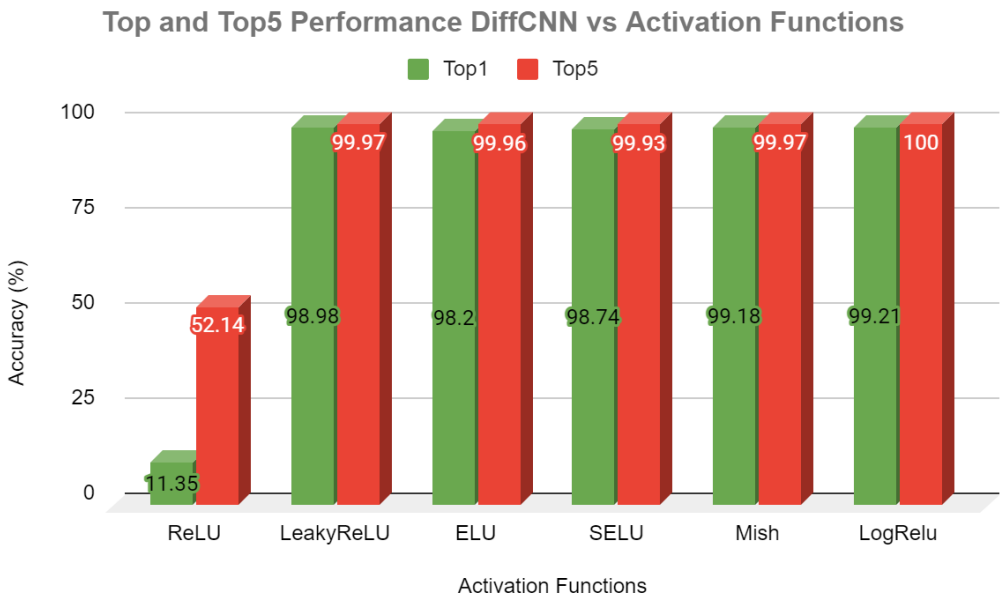

Figure 4.118: DiffCNN with RMSprop (0.001)

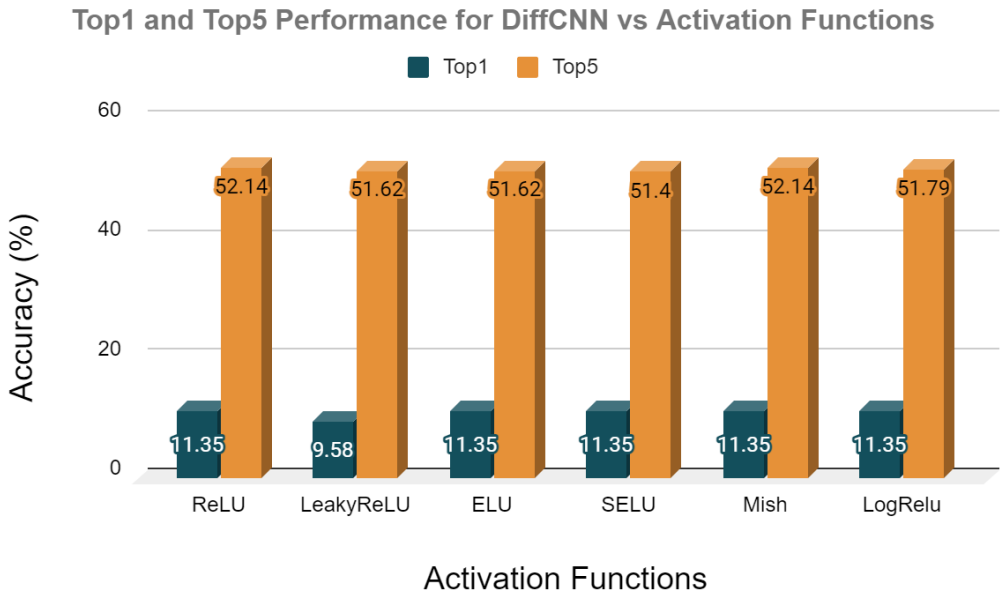

Figure 4.119: DiffCNN with RMSprop (0.01)

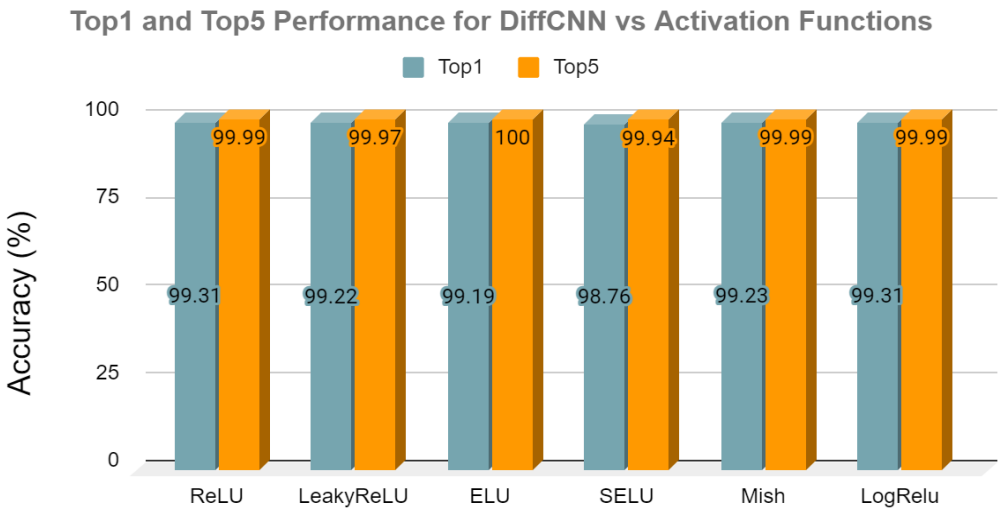

Figure 4.121: DiffCNN with RMSprop (0.0005)

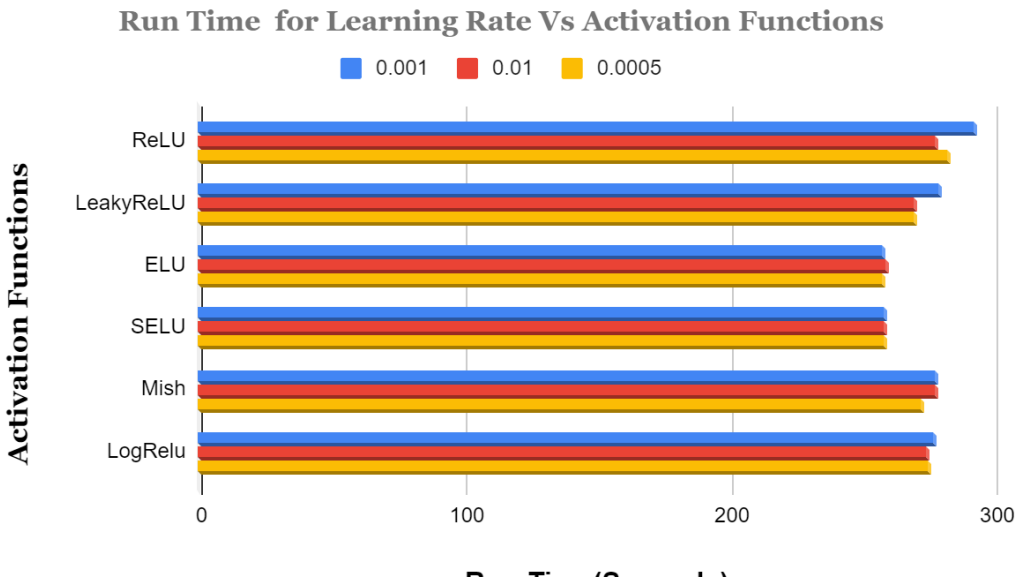

Figure 4.120: Run Time with DiffCNN with RMSprop Optimizer

4.3. The Classification Performance of CIFAR-10 Dataset

In this section, we present results for LogRelu, ReLU, LeakyReLU, ELU, SELU, and Mish activation functions applied to pre-trained MobileNet, LeNet, DenseNet, and ResNet models. These assessments cover various optimizers (Adam, SGD, and RMSprop) with learning rates (0.001, 0.1, and 0.0005) on the CIFAR-10 Dataset

4.3.1. The Classification DiffCNN Performance

In this section, we present results for LogRelu, ReLU, LeakyReLU, ELU, SELU, and Mish activation functions applied to the DiffCNN model. These assessments cover various optimizers (Adam, SGD, and RMSprop) with learning rates (0.001, 0.1, 0.0005, and 0.0003) on the Cifar10 dataset.

Table 4.37, analyzes activation functions at diverse learning rates. ReLU at 0.001 delivers Top1 and Top5 Accuracies of 74.38% and 96.89%, with a runtime of 1380.12 seconds. Remarkably, at 0.01, all reach 10.00% Top1 Accuracy, with similar runtimes. LeakyReLU maintains strong performance (Top1: 76.56% to 77.57%, Top5: 96.90%) and runtime like ReLU (around 1370 seconds). ELU remains stable (Top1: 76.01% to 76.98%, Top5: 96.60% to 97.44%) with slightly lower runtime, especially at 0.0003 (1319.92 seconds). SELU sustains consistent accuracy (Top1: 75.63% to 77.47%, Top5: 96.82% to 97.57%) and shorter runtimes (1292.78 to 1307.77 seconds). Mish is versatile (Top1: 74.97% to 78.32%, Top5: 96.91% to 97.70%) but runtime varies (lowest at 0.0003: 1174.60 seconds). LogRelu is competitive (Top1: 75.16% to 78.35%, Top5: 96.59% to 97.67%), with reduced runtime at 0.0003 (843.74 seconds) and 0.0005 (1108.26 seconds). At 0.01, all converge to 10.00% Top1 Accuracy, indicating the incomparability of Adam optimizer to activation choice with higher learning rates.

Table 4.38, shows that a learning rate of 0.01 consistently stands out as the most effective, with ReLU achieving excellent Top1 (69.58%) and Top5 (97.36%) Accuracy in 1329.30 seconds. LeakyReLU closely follows suit with strong results at 0.01 (Top1: 72.16%, Top5: 97.34%) and runtime similar to ReLU. ELU performs well at 0.01 (Top1: 70.44%, Top5: 97.25%) with varying runtimes. SELU excels at 0.01 (Top1: 71.20%, Top5: 97.17%) and maintains shorter runtimes. Mish displays competitive accuracy across rates (Top1: 17.18% to 66.86%, Top5: 96.47% to 97.70%) with runtime variations. LogRelu competes well (Top1: 16.34% to 69.11%, Top5: 96.93% to 97.67%) and notably shortens runtime at 0.01 (1226.90 seconds). In summary, a learning rate of 0.01 consistently delivers better accuracy, with LeakyReLU and SELU standing out as top performers among the activation functions.

Table 4. 39, summarizes activation functions' performance at different learning rates: At 0.001, ReLU achieves 76.08% Top1 Accuracy, LeakyReLU 75.76%, ELU 75.65%, SELU 73.63%, Mish 73.55%, and LogRelu 76.40%. At 0.01, all converge to 10.00% Top1 Accuracy. At 0.0005, ReLU excels with 77.67% Top1 and 97.82% Top5 Accuracy, while LogRelu achieves 77.33% Top1.

At 0.0003, SELU shines with 78.11% Top1 and 97.39% Top5 Accuracy, and LogRelu achieves 77.68% Top1 with reduced runtime. Overall, at 0.01, all activations reach 10.00% accuracy, indicating reduced activation influence at higher learning rates.

Table 4.37: DiffCNN with Adam Optimizer

LR	**0.001**			**0.01**			**0.0005**			**0.0003**		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top1**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	74.38	96.89	1380.12	10.00	50.00	1380.12	77.51	97.44	1375.16	77.09	97.56	1374.19
LeakyReLU	76.56	96.90	1369.30	10.00	50.00	1372.49	77.57	97.64	1370.39	76.56	97.47	1365.24
ELU	76.01	96.60	1319.92	10.00	50.00	1344.86	76.73	97.44	1342.53	76.98	97.41	1319.92
SELU	75.63	96.82	1292.78	10.00	50.00	1296.39	76.13	97.43	1307.77	77.47	97.57	1292.78
Mish	74.97	96.91	1308.96	10.00	50.00	1319.33	78.32	97.52	1174.60	77.02	97.70	1316.97
LogRelu	75.16	96.59	1376.60	10.00	50.00	1395.35	78.35	97.43	843.74	78.18	97.67	1108.26

Table 4.38:DiffCNN with SGD Optimizer

LR	**0.001**			**0.01**			**0.0005**			**0.0003**		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top1**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	16.95	65.90	1316.26	69.58	97.36	1329.30	12.19	52.77	1336.00	10.50	50.91	1321.63
LeakyReLU	16.90	66.24	1366.17	72.16	97.34	1366.07	12.79	54.59	1364.07	10.37	50.79	1368.73
ELU	14.51	58.47	1339.50	70.44	97.25	1417.13	12.15	54.83	1341.84	10.27	49.89	1353.43
SELU	17.00	68.26	1289.00	71.20	97.17	1295.04	12.17	53.48	1287.82	10.34	51.34	1287.56
Mish	17.18	66.77	1340.38	66.86	96.47	1326.02	13.64	53.90	1316.81	10.67	51.31	1353.69
LogRelu	16.34	64.47	1384.24	69.11	96.93	1226.90	11.60	52.70	1385.36	10.68	51.56	1375.94

Table 4. 39: DiffCNN with RMSprop Optimizer

LR	**0.001**			**0.01**			**0.0005**			**0.0003**		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top1**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	76.08	96.48	1317.68	10.00	50.00	1380.12	77.67	97.82	1325.43	77.03	97.43	1308.32
LeakyReLU	75.76	96.74	1324.85	10.00	50.00	1349.54	76.76	97.65	1380.73	73.68	97.17	1348.53
ELU	75.65	96.40	1388.23	10.00	50.00	1386.85	77.01	97.68	1400.86	76.82	97.33	1390.07
SELU	73.63	96.38	1355.33	10.00	50.00	1342.49	76.21	97.59	1355.80	78.11	97.39	1343.76
Mish	73.55	96.42	1339.87	10.00	50.00	1336.51	77.19	97.41	1350.01	77.48	97.60	1342.31
LogRelu	76.40	96.90	627.11	10.00	50.00	1305.41	77.33	97.21	1283.82	77.68	97.44	684.11

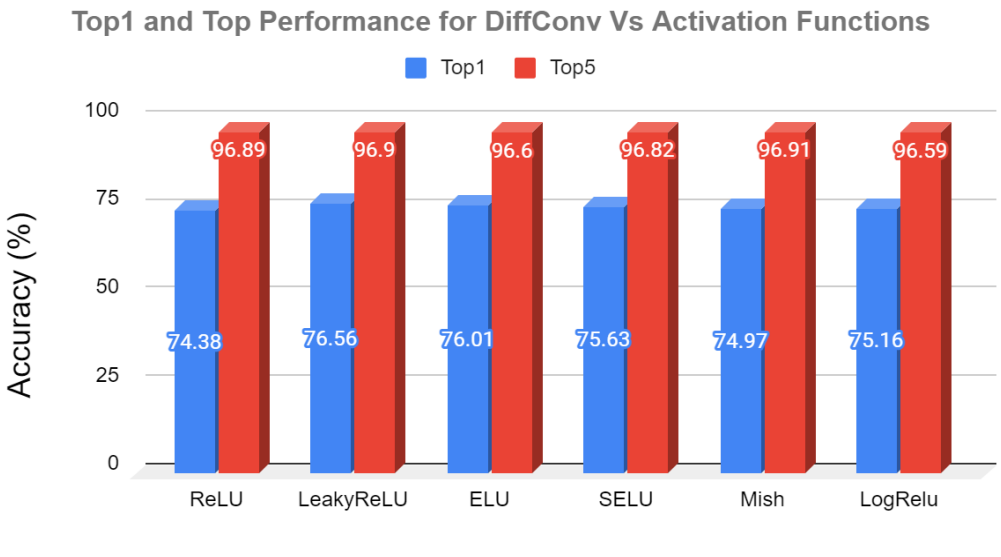

Figure 4.122: DiffConv with Adam (0.001)

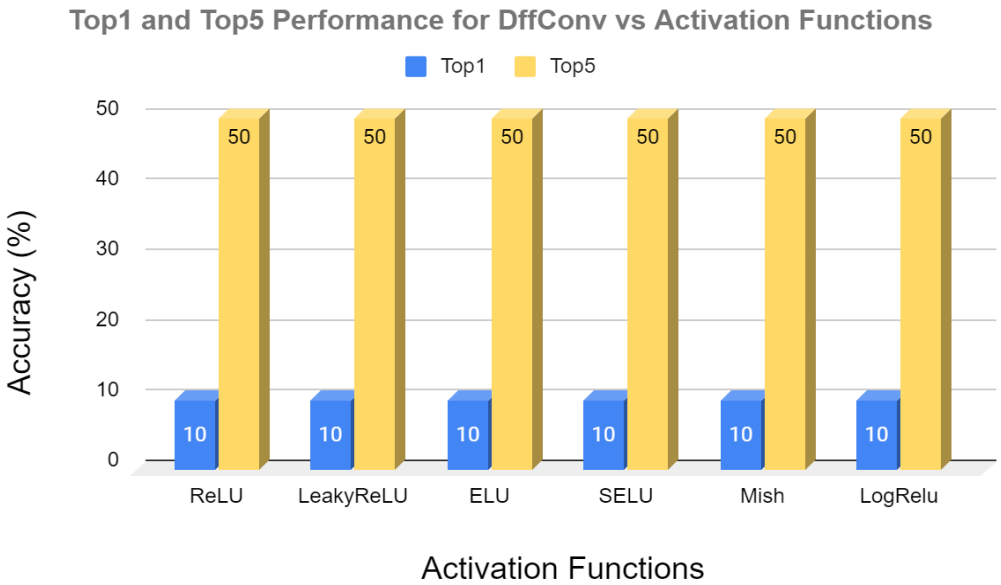

Figure 4.123: DiffConv with Adam (0.01)

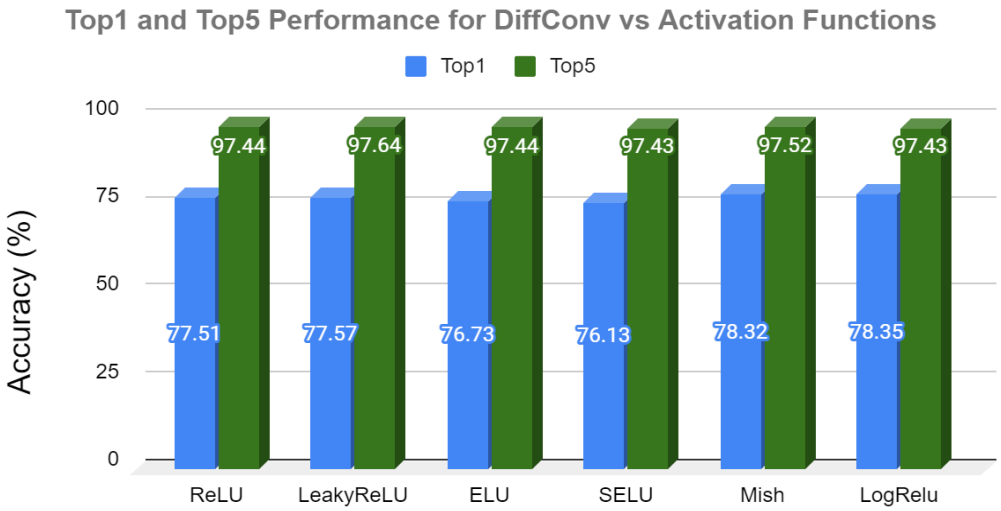

Figure 4.124: DiffConv with Adam (0.0005)

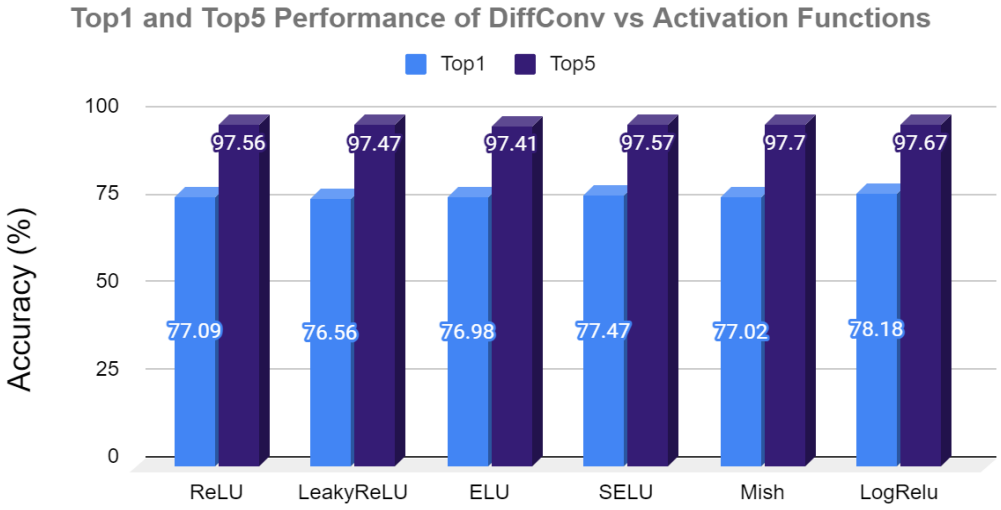

Figure 4.125: DiffConv with Adam (0.0003)

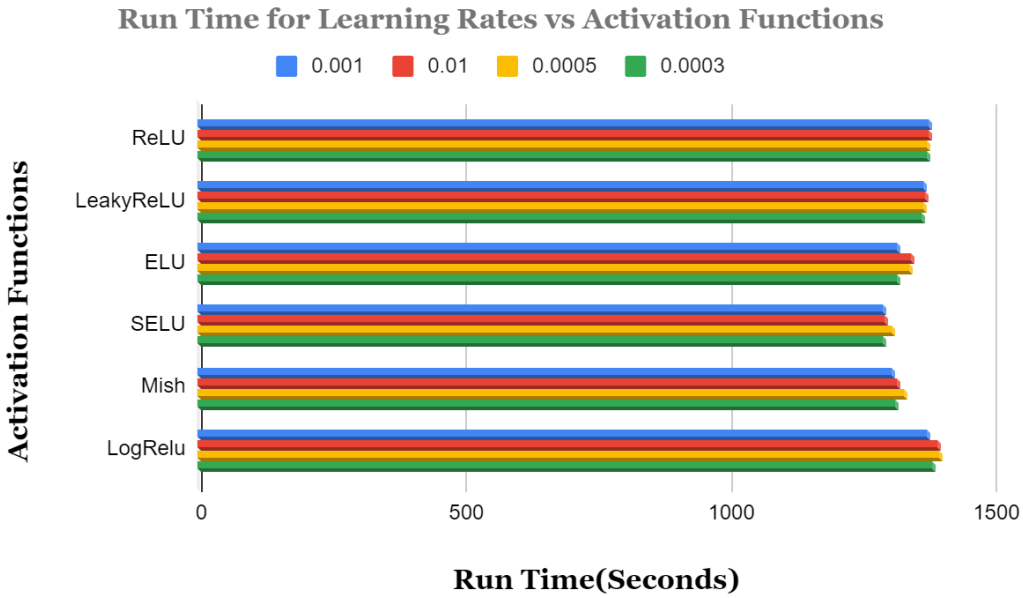

Figure 4.126:Run Time for DiffCNN with Adam

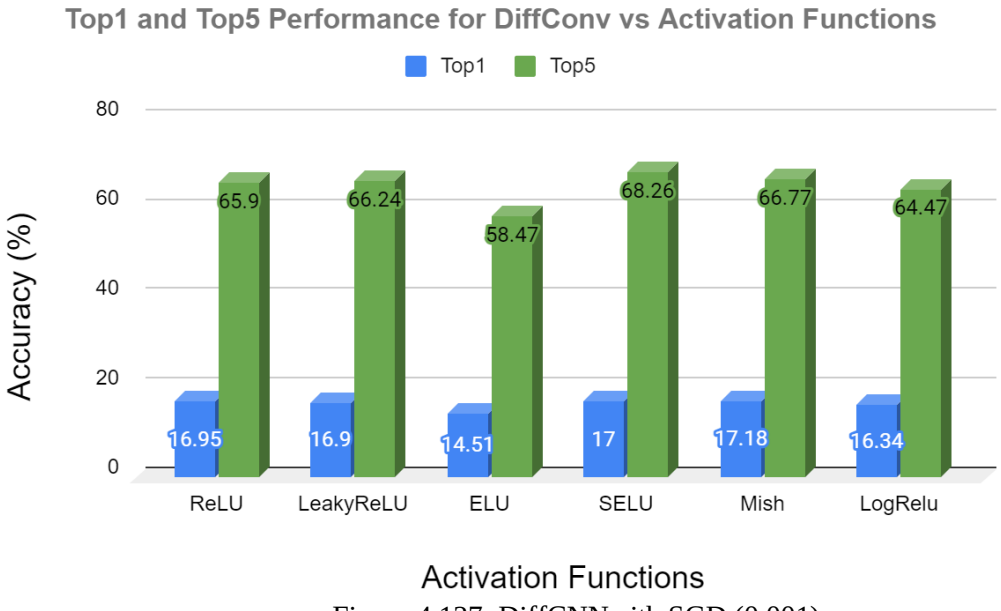

Figure 4.127: DiffCNNwith SGD (0.001)

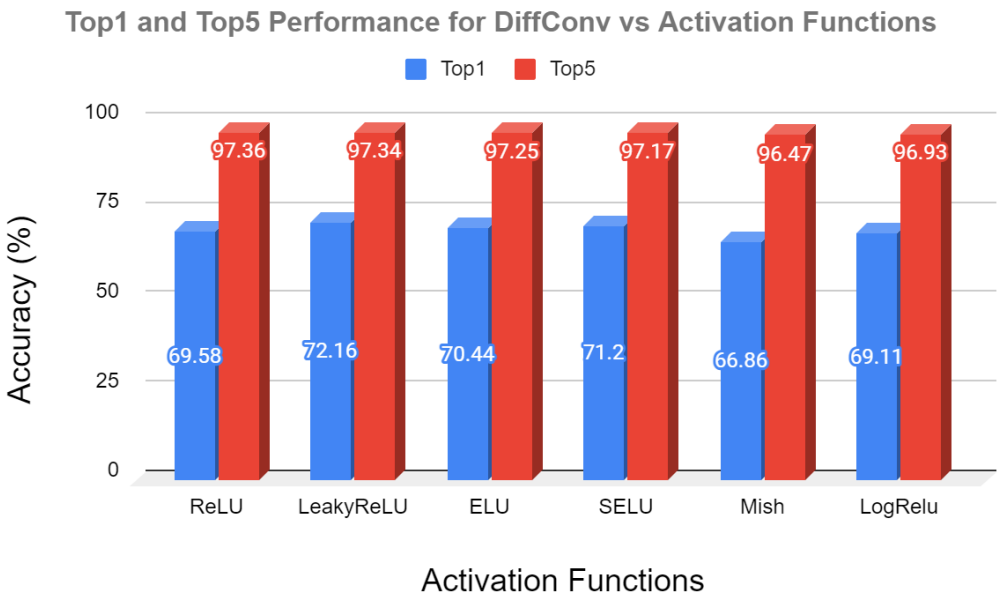

Figure 4.128: DiffCNNwith SGD (0.01)

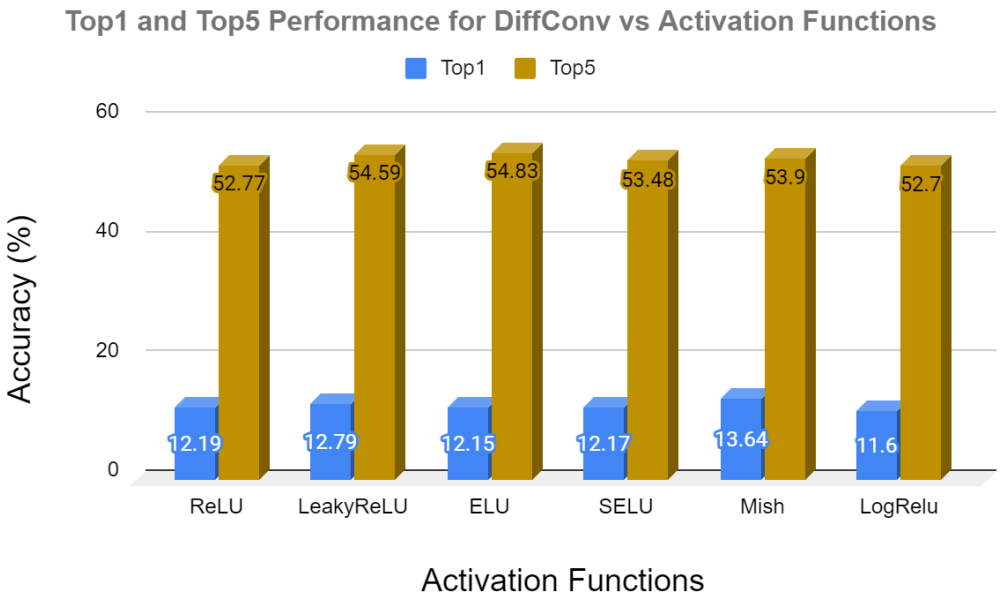

Figure 4.129: DiffCNN with SGD (0.0005)

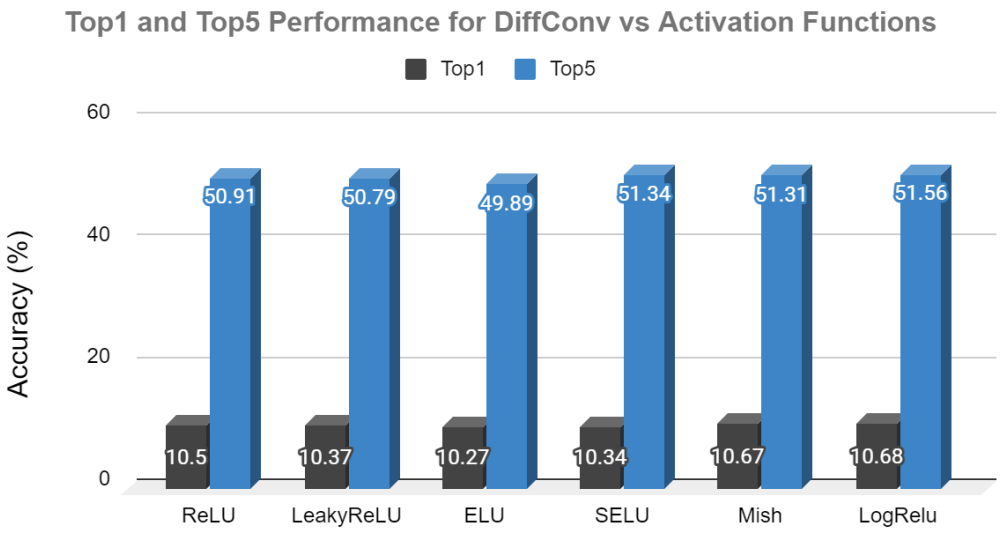

Figure 4.130: DiffCNN with SGD (0.0005)

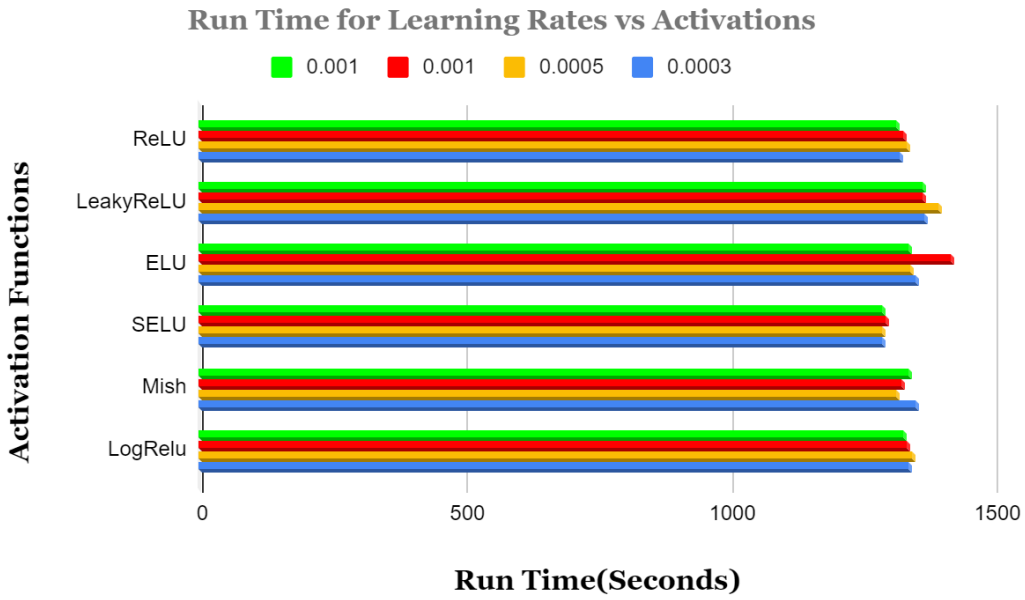

Figure 4.131: Run Time for DiffCNN with SGD

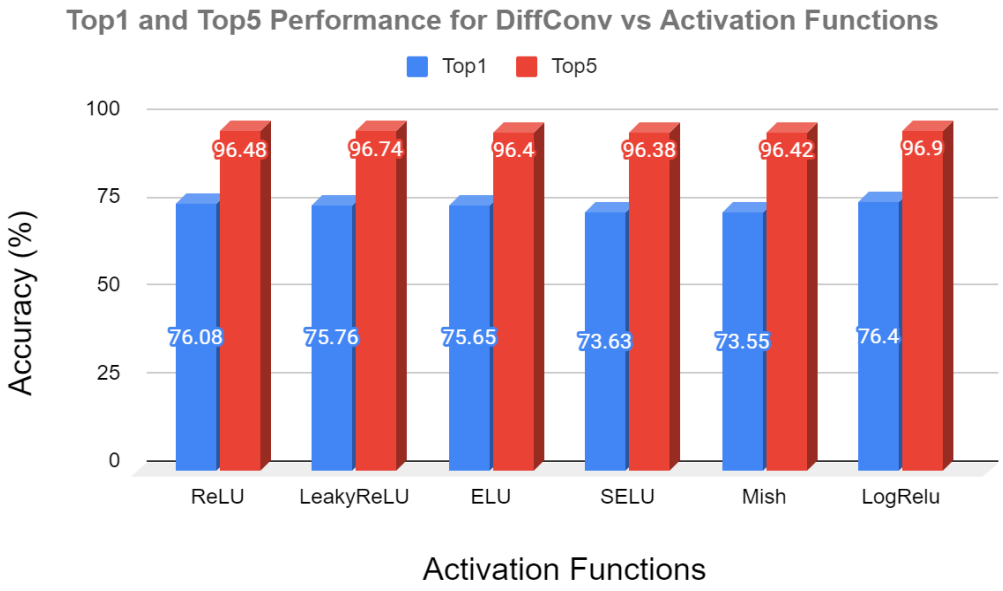

Figure 4.132: DiffCNN with RMSprop (0.001)

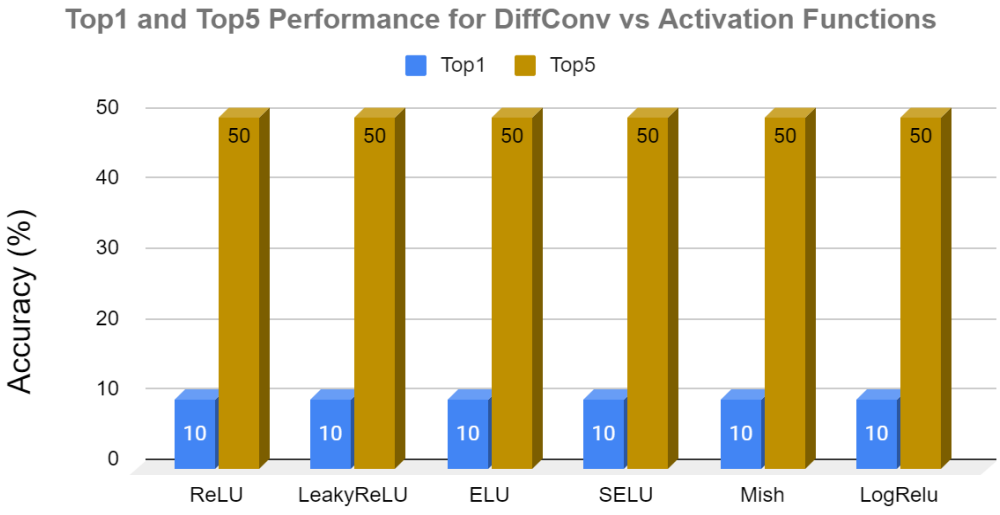

Figure 4.133: DiffCNN with RMSprop (0.01)

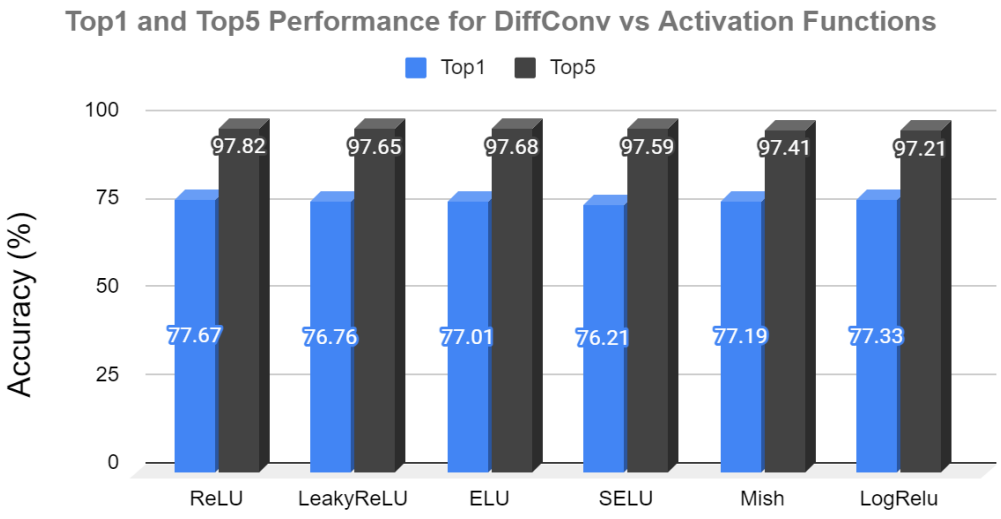

Figure 4.134: DiffCNN with RMSprop (0.0005)

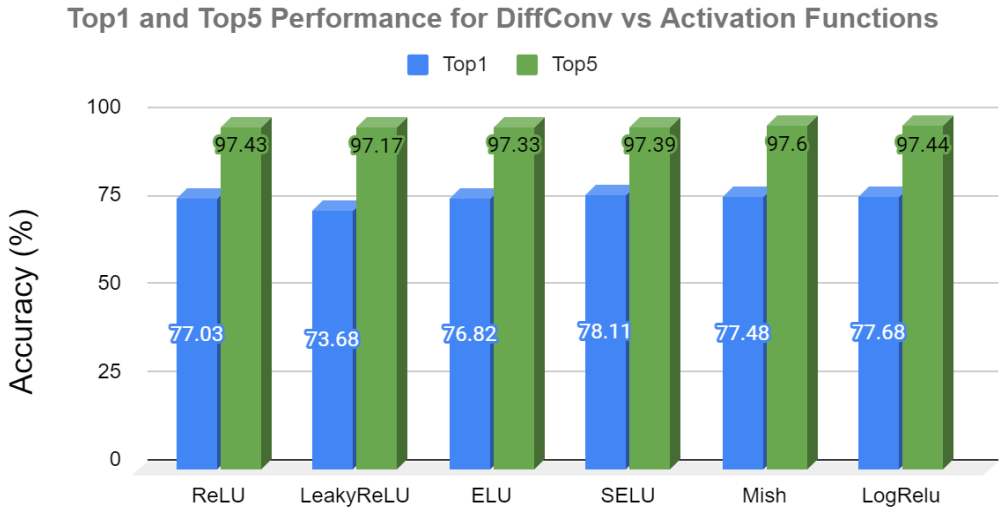

Figure 4.135: DiffCNN with RMSprop (0.0003)

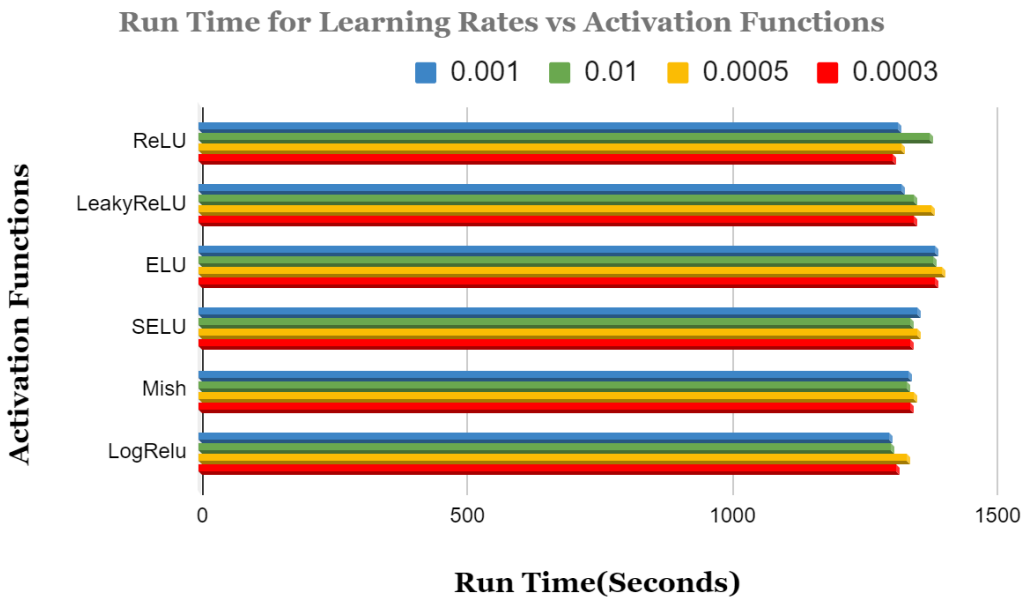

Figure 4.136: Run Time for DiffCNN with RMSprop

4.3.2. Pretrained Models Performance on Cifar10 Results

In this section, we present results for LogRelu, ReLU, LeakyReLU, ELU, SELU, and Mish activation functions applied to pre-trained MobileNet, LeNet, DenseNet, and ResNet models. These assessments cover various optimizers (Adam, SGD, and RMSprop) with learning rates (0.001, 0.1, and 0.0005) on the Cifar10 dataset.

The CIFAR-10 classification performance evaluation, detailed in Table 4.40, highlights ELU as a standout activation function, achieving impressive Top-1 and Top-5 Accuracy of 76.05% and 98.25%, respectively. LogRelu proves robust with a Top-1 Accuracy of 73.15% and an efficient runtime of 550.0 units, while LeakyReLU demonstrates competitive performance with a Top-1 Accuracy of 75.73% and a high Top-5 Accuracy of 97.90%. The consistent excellence of ELU and the balanced performance of LogRelu underscore their efficacy across various models.

Table 4.41 underscores LogRelu's excellence in MobileNet, LeNet, and ResNet, emphasizing its compatibility and efficiency. However, issues arise in DenseNet, revealing the importance of activation function selection tailored to specific architectural requirements. Notably, LogRelu maintains efficiency in ResNet, showcasing its versatility across different models.

Further analysis in Table 4.42 through Table 4.48 reveals activation function performance across diverse neural network models. LeakyReLU, ELU, and SELU consistently excel in MobileNet, with LogRelu showcasing efficiency. Noteworthy variations in performance are observed in LeNet, DenseNet, and ResNet, emphasizing the impact of activation function choice on model outcomes. Runtime differences across functions further underscore their varying computational efficiencies. Overall, these findings provide valuable insights for practitioners in selecting the most suitable activation functions for their specific model architectures and tasks.

Table 4.40:Shows Results Utilizing Adam Optimizer with a Learning Rate of 0.001

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	52.26	77.69	831.40	63.07	95.99	418.21	69.85	88.43	1907.03	52.45	75.29	702.79
LeakyReLU	75.73	97.90	780.16	62.79	95.87	398.41	45.80	76.85	1891.05	58.19	93.71	680.72
ELU	76.05	98.25	771.56	63.91	95.46	391.79	50.85	77.38	1912.79	73.30	97.49	687.44
SELU	52.69	77.94	768.66	65.41	96.04	205.91	51.40	77.70	1933.21	61.40	85.09	679.53
Mish	74.33	96.87	665.65	64.43	95.28	206.70	59.74	77.40	1898.18	74.53	97.36	675.46
LogRelu	73.15	98.10	550.55	64.78	96.02	182.53	43.32	76.24	1872.27	75.83	97.22	529.42

Table 4.41: Shows Results Utilizing Adam Optimizer with a Learning Rate of 0.01

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	10.00	50.00	841.86	49.35	90.84	410.44	10.00	50.00	1914.18	10.00	50.00	682.74
LeakyReLU	75.75	97.55	860.21	50.84	92.76	395.39	10.00	50.00	1867.65	74.15	97.40	682.05
ELU	75.61	97.74	728.73	10.00	50.00	392.11	10.00	50.00	1909.26	74.84	97.96	694.79
SELU	10.00	50.00	758.42	57.49	79.10	126.83	10.00	50.00	1910.26	10.00	50.00	670.24
Mish	76.19	98.02	712.64	53.23	91.58	371.62	10.00	50.00	1880.65	75.79	97.67	701.83
LogRelu	76.20	97.89	706.29	48.07	91.45	391.49	10.00	50.00	1871.41	71.97	97.39	612.28

Table 4.42: Shows Results Utilizing Adam Optimizer with a Learning Rate of 0.0005

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	69.03	94.34	723.24	63.90	96.01	406.67	78.38	99.01	1897.59	76.47	97.64	687.78
LeakyReLU	80.64	98.66	820.60	63.81	96.35	395.27	78.82	88.81	1874.77	76.70	97.75	679.50
ELU	79.78	98.41	735.30	65.73	96.59	219.86	87.78	99.18	1923.88	76.54	97.71	686.91
SELU	80.45	98.79	752.95	66.36	96.92	296.26	78.32	89.12	1913.75	77.95	97.82	682.75
Mish	78.03	98.39	711.88	64.71	96.36	429.19	62.36	86.48	1889.87	77.16	97.93	690.85
LogRelu	80.10	98.44	528.57	63.68	95.87	363.80	78.77	89.08	1896.82	76.14	97.99	568.04

Table 4.43: Shows Utilizing SGD Optimizer with a Learning Rate of 0.001

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	68.41	97.42	753.56	36.11	87.04	402.92	69.47	97.03	1871.90	60.67	95.39	619.47
LeakyReLU	72.77	98.06	851.34	31.82	82.78	389.82	69.86	97.29	1808.93	59.43	95.11	614.48
ELU	71.19	97.51	755.67	41.31	89.76	390.18	70.10	97.33	1869.71	61.83	95.39	636.12
SELU	71.17	97.70	746.43	46.58	91.86	389.86	70.31	97.28	1864.70	61.73	95.66	615.88
Mish	73.31	97.97	747.23	60.70	95.04	389.44	72.42	97.91	2065.56	59.58	95.21	625.54
LogRelu	73.00	98.24	884.32	25.64	79.62	398.37	70.66	97.28	1837.01	60.47	95.59	616.24

Table 4.44: Shows Utilizing SGD Optimizer with a Learning Rate of 0.01

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	78.04	98.28	809.13	59.32	95.23	400.46	81.15	98.85	1864.86	7097	97.10	629.90
LeakyReLU	78.52	98.47	775.34	60.22	95.55	392.86	83.89	98.83	1814.26	73.99	97.64	612.74
ELU	77.12	98.17	709.92	63.26	95.72	371.40	80.74	98.32	1849.36	71.15	97.12	657.82
SELU	78.34	98.42	739.61	64.56	96.62	353.72	82.56	98.85	1861.32	72.45	96.96	637.80
Mish	78.01	98.07	580.29	59.20	95.04	405.69	79.69	98.58	2073.61	61.11	93.94	620.47
LogRelu	77.56	98.28	919.52	60.90	96.00	347.14	83.42	98.49	1710.65	72.41	96.10	559.42

Table 4.45: Shows Results Utilizing SGD Optimizer with a Learning Rate of 0.0005

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	66.24	97.25	813.13	14.07	56.85	396.20	66.40	96.90	1863.00	57.25	94.57	621.76
LeakyReLU	66.38	97.40	798.13	22.02	74.15	392.36	66.13	96.90	1864.23	57.34	94.86	602.44
ELU	64.20	96.57	734.67	31.82	81.98	387.44	67.37	97.07	1860.69	58.70	95.35	657.93
SELU	66.64	97.11	732.17	41.14	87.83	386.05	67.94	96.97	1861.94	57.50	94.85	626.10
Mish	68.03	97.74	788.41	23.43	72.73	420.35	67.72	96.84	2079.32	56.94	94.63	629.20
LogRelu	65.38	97.06	915.44	19.47	67.75	389.60	10.00	50.00	1884.82	59.19	95.05	609.81

Table 4.46: Shows Results Utilizing RMSprop Optimizer with a Learning Rate of 0.001

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	10.00	50.00	834.86	60.60	95.34	398.49	10.00	50.00	1858.40	10.00	50.00	654.69
LeakyReLU	68.86	97.09	805.35	60.48	95.18	391.83	10.00	50.00	1818.21	10.00	50.00	670.88
ELU	79.11	98.07	692.69	65.11	95.72	280.39	10.00	50.00	1818.21	76.22	97.63	690.45
SELU	10.00	50.00	761.35	65.26	95.58	225.48	10.00	50.00	1882.45	10.00	50.00	661.14
Mish	10.00	50.00	787.59	63.37	95.52	237.14	10.00	50.00	1863.74	68.91	95.69	681.71
LogRelu	10.00	50.00	959.89	63.06	94.29	144.01	10.00	50.00	1884.82	10.00	50.00	675.81

Table 4.47: Shows Results Utilizing RMSprop Optimizer with a Learning Rate of 0.01

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	10.00	50.00	836.02	44.49	87.71	382.24	10.00	50.00	1883.32	10.00	50.00	653.72
LeakyReLU	69.24	97.00	838.20	52.43	93.98	390.15	10.00	50.00	1820.63	10.00	50.00	675.08
ELU	72.61	96.63	651.83	61.95	95.72	387.31	10.00	50.00	1820.63	73.30	97.66	681.28
SELU	10.00	50.00	745.11	10.00	50.00	384.62	10.00	50.00	1872.51	10.00	50.00	650.82
Mish	10.00	50.00	781.89	49.24	89.45	303.20	10.00	50.00	1873.66	73.80	97.54	666.87
LogRelu	10.00	50.00	921.70	53.02	92.46	366.01	10.00	50.00	1886.58	10.00	50.00	669.30

Table 4.48: Shows Results Utilizing RMSprop Optimizer with a Learning Rate of 0.005

Model	MobileNet			LeNet			DenseNet			ResNet		
Activations	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**	**Top1**	**Top5**	**Time**
ReLU	18.67	59.29	832.90	62.71	95.48	394.67	26.68	66.25	1874.73	42.39	67.67	661.59
LeakyReLU	78.91	98.53	632.20	64.54	96.12	249.35	10.00	50.00	1826.95	17.89	58.71	660.23
ELU	78.25	98.41	516.25	65.80	96.41	297.07	10.00	50.00	1826.95	74.90	97.69	675.50
SELU	74.33	96.87	665.65	64.89	96.15	362.05	35.77	58.83	1895.19	26.30	58.31	641.03
Mish	10.00	50.00	793.37	64.31	96.01	367.86	25.68	58.52	1900.80	73.34	97.07	668.96
LogRelu	10.00	50.00	933.79	66.47	96.05	318.59	10.00	50.00	1877.28	16.37	50.00	649.01

5. CONCLUSIONS

In this thesis, a novel approach called Logarithmic Learning Differential Convolutional Neural Network(LDiffCNN) is proposed to address the time-intensive training challenges inherent in the Differential Convolutional Neural Network (DiffCNN). LDiffCNN mitigates these issues by incorporating the LogRelu activation function, a Logarithmic Cost Function, and innovative logarithmic learning techniques. Through experiments, it is demonstrated that integrating LogRelu and Logarithmic cost function into CNN and DiffCNN leads to significant accuracy improvements on different datasets. Additionally, LDiffCNN integrating the Logarithmic Cost Function with differential convolution, consistently outperforms the standard CNN model across the majority of the datasets. Notably, LDiffCNN also shows a reduction in training iterations and achieves rapid convergence.

The study also conducted a comprehensive evaluation of LogRelu and other activation functions (ReLU, LeakyReLU, ELU, SELU, Mish) across various deep learning models, including MobileNet, LeNet, DenseNet, and ResNet, using the MNIST and Cifar-10 datasets. Activation function selection emerged as a critical factor influencing model performance, with LeakyReLU, ELU, SELU, and LogRelu consistently outperforming ReLU. Notably, at a learning rate of 0.001, LeakyReLU and ELU achieved impressive accuracy levels ranging from 98.8% to 99.1% across multiple models, while ReLU struggled at 0.01 with accuracy as low as 9.8%-98.8%. Additionally, LogRelu demonstrated robust performance, often matching or surpassing other activations, particularly at lower learning rates.

However, when implementing LDiffCNN, compatibility issues arose with both the Adam and RMSprop optimizers at a learning rate of 0.01. On the other hand, there were compatibility issues with the SGD optimizer at lower learning rates, specifically at 0.001, 0.0005, and 0.0003, while notably better results were observed when using a higher learning rate, such as 0.01, with SGD. These findings underscore the critical importance of carefully considering both the choice of optimizer and the learning rate in the model configuration process. In addition, Among the models used for evaluation, DenseNet takes a long runtime to complete the training across the activation functions and the learning rates used in this study as revealed in the results.

These results emphasize the effectiveness of the proposed logarithmic learning approach in convolutional neural networks. In future studies, the transferability of LogRelu and Logarithmic Cost Function to other neural network architectures will be investigated along with their applicability to real-world scenarios.

REFERENCES

Abdelli, A., Yachir, A., Amamra, A., & Khaldi, B. (2023). Maximum likelihood estimate sharing for collective perception in static environments for swarm robotics. *Robotica*, 1–20.

Ahmed, M. I., Mamun, S. M., & Asif, A. U. Z. (2021). DCNN-based vegetable image classification using transfer learning: A comparative study. *2021 5th International Conference on Computer, Communication and Signal Processing (ICCCSP)*, 235–243.

Ali, S., Abuhmed, T., El-Sappagh, S., Muhammad, K., Alonso-Moral, J. M., Confalonieri, R., Guidotti, R., Del Ser, J., Díaz-Rodríguez, N., & Herrera, F. (2023). Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence. *Information Fusion*, *99*, 101805.

Almotairi, K. H., Hussein, A. M., Abualigah, L., Abujayyab, S. K. M., Mahmoud, E. H., Ghanem, B. O., & Gandomi, A. H. (2023). Impact of artificial intelligence on COVID-19 pandemic: a survey of image processing, tracking of disease, prediction of outcomes, and computational medicine. *Big Data and Cognitive Computing*, *7*(1), 11.

Alomar, K., Aysel, H. I., & Cai, X. (2023). Data augmentation in classification and segmentation: A survey and new strategies. *Journal of Imaging*, *9*(2), 46.

Alzubaidi, L., Fadhel, M. A., Al-Shamma, O., Zhang, J., Santamaría, J., Duan, Y., & R. Oleiwi, S. (2020). Towards a better understanding of transfer learning for medical imaging: a case study. *Applied Sciences*, *10*(13), 4523.

Apostolopoulos, I. D., & Tzani, M. A. (2023). Industrial object and defect recognition utilizing multilevel feature extraction from industrial scenes with Deep Learning approach. *Journal of Ambient Intelligence and Humanized Computing*, *14*(8), 10263–10276.

Bai, T., Luo, J., & Zhao, J. (2020). Recent advances in understanding adversarial robustness of deep neural networks. *ArXiv Preprint ArXiv:2011.01539*.

Bansal, P. (n.d.). *Intel Scene Image Classification*. Retrieved April 10, 2023, from https://www.kaggle.com/datasets/puneet6060/intel-image-classification?select=seg_test

Bao, Y., Chen, Z., Wei, S., Xu, Y., Tang, Z., & Li, H. (2019). The state of the art of data science and engineering in structural health monitoring. *Engineering*, *5*(2), 234–242.

Bello, I., Fedus, W., Du, X., Cubuk, E. D., Srinivas, A., Lin, T.-Y., Shlens, J., & Zoph, B. (2021). Revisiting resnets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems*, *34*, 22614–22627.

Bharadiya, J. (2023). Convolutional Neural Networks for Image Classification. *International Journal of Innovative Science and Research Technology*, *8*(5), 673–677.

Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., Modi, K., & Ghayvat, H. (2021). CNN variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, *10*(20), 2470.

Bird, J. J., & Lotfi, A. (2023). CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. *ArXiv Preprint ArXiv:2303.14126*.

Bolhasani, H., Jassbi, S. J., & Sharifi, A. (2023). DLA-H: A Deep Learning Accelerator for Histopathologic Image Classification. *Journal of Digital Imaging*, *36*(2), 433–440.

Boopathi, S., & Kanike, U. K. (2023). Applications of Artificial Intelligent and Machine Learning Techniques in Image Processing. In *Handbook of Research on Thrust Technologies' Effect on Image Processing* (pp. 151–173). IGI Global.

Cai, H., Zhu, L., & Han, S. (2018). Proxylessnas: Direct neural architecture search on target task and hardware. *ArXiv Preprint ArXiv:1812.00332*.

Caicedo, J. C., Goodman, A., Karhohs, K. W., Cimini, B. A., Ackerman, J., Haghighi, M., Heng, C., Becker, T., Doan, M., & McQuin, C. (2019). Nucleus segmentation across imaging experiments: the 2018 Data Science Bowl. *Nature Methods*, *16*(12), 1247–1253.

Cengıl, E., & Çinar, A. (2019). Multiple classification of flower images using transfer learning. *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, 1–6.

Cheng, G., Lai, P., Gao, D., & Han, J. (2023). Class attention network for image recognition. *Science China Information Sciences*, *66*(3), 132105.

Choi, Y., Uh, Y., Yoo, J., & Ha, J.-W. (2020). Stargan v2: Diverse image synthesis for multiple domains. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8188–8197.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1251–1258.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *ArXiv Preprint ArXiv:1511.07289*.

Corral, J. M. R., Civit-Masot, J., Luna-Perejón, F., Díaz-Cano, I., Morgado-Estévez, A., & Domínguez-Morales, M. (2024). Energy efficiency in edge TPU vs. embedded GPU for computer-aided medical imaging segmentation and classification. *Engineering Applications of Artificial Intelligence*, *127*, 107298.

Deepa, S. N., & Rasi, D. (2023). FHGSO: Flower Henry gas solubility optimization integrated deep convolutional neural network for image classification. *Applied Intelligence*, *53*(6), 7278–7297.

Dhiman, G., Kumar, A. V., Nirmalan, R., Sujitha, S., Srihari, K., Yuvaraj, N., Arulprakash, P., & Raja, R. A. (2023). Multi-modal active learning with deep reinforcement learning for target feature extraction in multi-media image processing applications. *Multimedia Tools and Applications*, *82*(4), 5343–5367.

Dişken, G. (2023). Differential convolutional network for noise mask estimation. *Applied Acoustics*, *211*, 109568.

Dong, Y., Fu, Q.-A., Yang, X., Pang, T., Su, H., Xiao, Z., & Zhu, J. (2020). Benchmarking

adversarial robustness on image classification. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 321–331.

Elfwing, S., Uchibe, E., & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, *107*, 3–11.

Elhani, D., Megherbi, A. C., Zitouni, A., Dornaika, F., Sbaa, S., & Taleb-Ahmed, A. (2023). Optimizing convolutional neural networks architecture using a modified particle swarm optimization for image classification. *Expert Systems with Applications*, *229*, 120411.

Eminaga, O., Abbas, M., Shen, J., Laurie, M., Brooks, J. D., Liao, J. C., & Rubin, D. L. (2023). PlexusNet: A neural network architectural concept for medical image classification. *Computers in Biology and Medicine*, *154*, 106594.

Gupta, A., Anpalagan, A., Guan, L., & Khwaja, A. S. (2021). Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array*, *10*, 100057.

Hafiz, A. M., Bhat, R. A., & Hassaballah, M. (2023). Image classification using convolutional neural network tree ensembles. *Multimedia Tools and Applications*, *82*(5), 6867–6884.

Han, Y., Lei, Y., Shkolnikov, V., Xin, D., Auduong, A., Barcelo, S., Allebach, J., & Delp, E. J. (2023). An Ensemble Method With Edge Awareness for Abnormally Shaped Nuclei Segmentation. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4314–4324.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., & Smith, N. J. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

Hashemi, A. S., & Mozaffari, S. (2021). CNN adversarial attack mitigation using perturbed samples training. *Multimedia Tools and Applications*, *80*, 22077–22095.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision*, 1026–1034.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv Preprint ArXiv:1207.0580*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv Preprint ArXiv:1704.04861*.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., & Vasudevan, V. (2019). Searching for mobilenetv3. *Proceedings of the IEEE/CVF*

International Conference on Computer Vision, 1314–1324.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4700–4708.

Huang, L., Qin, J., Zhou, Y., Zhu, F., Liu, L., & Shao, L. (2023). Normalization techniques in training dnns: Methodology, analysis and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(03), 90–95.

Iman, M., Arabnia, H. R., & Rasheed, K. (2023). A review of deep transfer learning and recent advancements. *Technologies*, *11*(2), 40.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, 448–456.

Iqball, T., & Wani, M. A. (2023). Weighted ensemble model for image classification. *International Journal of Information Technology*, *15*(2), 557–564.

Kader, I. A. El, Xu, G., Shuai, Z., Saminu, S., Javaid, I., & Ahmad, I. S. (2021). Differential deep convolutional neural network model for brain tumor classification. *Brain Sciences*, *11*(3). https://doi.org/10.3390/brainsci11030352

Kamakshi, V., & Krishnan, N. C. (2023). Explainable image classification: The journey so far and the road ahead. *AI*, *4*(3), 620–651.

Kang, J.-S., Kang, J., Kim, J.-J., Jeon, K.-W., Chung, H.-J., & Park, B.-H. (2023). Neural Architecture Search Survey: A Computer Vision Perspective. *Sensors*, *23*(3), 1713.

Kaur, G., & Sharma, A. (2023). A deep learning-based model using hybrid feature extraction approach for consumer sentiment analysis. *Journal of Big Data*, *10*(1), 5.

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020a). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, *53*, 5455–5516.

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020b). A survey of the recent architectures of deep convolutional neural networks. In *Artificial Intelligence Review* (Vol. 53, Issue 8). Springer Netherlands. https://doi.org/10.1007/s10462-020-09825-6

Kljucaric, L., & George, A. D. (2023). Deep Learning Inferencing with High-performance Hardware Accelerators. *ACM Transactions on Intelligent Systems and Technology*, *14*(4), 1–25.

Kobrinskii, B. A. (2023). Artificial Intelligence: Problems, Solutions, and Prospects. *Pattern Recognition and Image Analysis*, *33*(3), 217–220.

Krause, J., Stark, M., Deng, J., & Fei-Fei, L. (2013). 3d object representations for fine-grained categorization. *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 554–561.

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., Cortes, C., & Burges, C. (2010). *MNIST handwritten digit database*. Florham Park, NJ, USA.

Lee, B., Ko, K., Hong, J., & Ko, H. (2023). Domain-agnostic single-image super-resolution via a meta-transfer neural architecture search. *Neurocomputing*, *524*, 59–68.

Lei, X., Pan, H., & Huang, X. (2019). A dilated cnn model for image classification. *IEEE Access*, *7*, 124087–124095. https://doi.org/10.1109/ACCESS.2019.2927169

Lei, X., Xia, Y., Wang, A., Jian, X., Zhong, H., & Sun, L. (2023). Mutual information based anomaly detection of monitoring data with attention mechanism and residual learning. *Mechanical Systems and Signal Processing*, *182*, 109607.

Li, S., Mao, Y., Zhang, F., Wang, D., & Zhong, G. (2023). Dlw-nas: Differentiable light-weight neural architecture search. *Cognitive Computation*, *15*(2), 429–439.

Li, X., Li, M., Yan, P., Li, G., Jiang, Y., Luo, H., & Yin, S. (2023). Deep learning attention mechanism in medical image analysis: Basics and beyonds. *International Journal of Network Dynamics and Intelligence*, 93–116.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*.

Liang, W., Liang, Y., & Jia, J. (2023). MiAMix: Enhancing Image Classification through a Multi-Stage Augmented Mixed Sample Data Augmentation Method. *Processes*, *11*(12), 3284.

Lindsay, G. W. (2021). Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of Cognitive Neuroscience*, *33*(10), 2017–2031.

Liu, Y., Zhang, J., Gao, C., Qu, J., & Ji, L. (2019). Natural-logarithm-rectified activation function in convolutional neural networks. *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, 2000–2008.

Ma, L., & Liang, L. (2023). Increasing-margin adversarial (IMA) training to improve adversarial robustness of neural networks. *Computer Methods and Programs in Biomedicine*, 107687.

Ma, W., Zhou, T., Qin, J., Xiang, X., Tan, Y., & Cai, Z. (2023). Adaptive multi-feature fusion via cross-entropy normalization for effective image retrieval. *Information Processing & Management*, *60*(1), 103119.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *Proc. Icml*, *30*(1), 3.

Marchand–Maillet, S., & Müller, H. (n.d.). *Interpretability of Deep Learning for Medical Image Classification: Improved Understandability and Generalization*.

McKinney, W. (2010). Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, *445*(1), 51–56.

Melis, B., & Avci, M. (2014). Logarithmic learning for generalized classifier neural network. *Neural Networks*, *60*, 133–140. https://doi.org/10.1016/j.neunet.2014.08.004

Misra, D. (2019). Mish: A self regularized non-monotonic activation function. *ArXiv Preprint ArXiv:1908.08681*.

Motwani, A. (2021). *Tomato Leaves Dataset*. https://www.kaggle.com/datasets/ashishmotwani/tomato

Moutik, O., Sekkat, H., Tigani, S., Chehri, A., Saadane, R., Tchakoucht, T. A., & Paul, A. (2023). Convolutional neural networks or vision transformers: Who will win the race for action recognitions in visual data? *Sensors*, *23*(2), 734.

Mutlag, W. K., Ali, S. K., Aydam, Z. M., & Taher, B. H. (2020). Feature extraction methods: a review. *Journal of Physics: Conference Series*, *1591*(1), 12028.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807–814.

Naveed, H., Anwar, S., Hayat, M., Javed, K., & Mian, A. (2024). Survey: Image mixing and deleting for data augmentation. *Engineering Applications of Artificial Intelligence*, *131*, 107791.

Ngombu, S., Binol, H., Gurcan, M. N., & Moberly, A. C. (2023). Advances in artificial intelligence to diagnose otitis media: state of the art review. *Otolaryngology–Head and Neck Surgery*, *168*(4), 635–642.

Niu, S., Liu, Y., Wang, J., & Song, H. (2020). A decade survey of transfer learning (2010–2020). *IEEE Transactions on Artificial Intelligence*, *1*(2), 151–166.

Örnhag, M. V., Güler, P., Knyaginin, D., & Borg, M. (2023). Accelerating AI using next-generation hardware: Possibilities and challenges with analog in-memory computing. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 488–496.

Ozyildirim, B. M., & Avci, M. (2015). One Pass Learning For Generalized Classi er Neural Network. *Neural Networks*. https://doi.org/10.1016/j.neunet.2015.10.008

Paiss, R., Chefer, H., & Wolf, L. (2022). No token left behind: Explainability-aided image classification and generation. *European Conference on Computer Vision*, 334–350.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., & Antiga, L. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*.

Pathak, Y., Shukla, P. K., Tiwari, A., Stalin, S., & Singh, S. (2022). Deep transfer learning based classification model for COVID-19 disease. *Irbm*, *43*(2), 87–92.

Patrício, C., Neves, J. C., & Teixeira, L. F. (2023). Explainable Deep Learning Methods in Medical Image Classification: A Survey. *ACM Computing Surveys*, *56*(4), 1–41.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., & Dubourg, V. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Preechakul, K., Sriswasdi, S., Kijsirikul, B., & Chuangsuwanich, E. (2022). Improved image classification explainability with high-accuracy heatmaps. *Iscience*, *25*(3).

Qu, R., Xu, G., Ding, C., Jia, W., & Sun, M. (2020a). Standard plane identification in fetal brain ultrasound scans using a differential convolutional neural network. *IEEE Access*, *8*, 83821–83830. https://doi.org/10.1109/ACCESS.2020.2991845

Qu, R., Xu, G., Ding, C., Jia, W., & Sun, M. (2020b). Standard plane identification in fetal brain ultrasound scans using a differential convolutional neural network. *IEEE Access*, *8*, 83821–83830. https://doi.org/10.1109/ACCESS.2020.2991845

Radhakrishnan, A., Durham, C., Soylemezoglu, A., & Uhler, C. (2017). Patchnet: interpretable neural networks for image classification. *ArXiv Preprint ArXiv:1705.08078*.

Raglin, A. J., & Basak, A. (2023). Exploring stable diffusion and interpretability of image classification using neural features. *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications V*, *12538*, 414–418.

Ramachandran, P. (2017). Le Q. Searching for activation functions. ArXiv. *Preprint.*

Ravikumar, A., & Sriraman, H. (2023). Acceleration of Image Processing and Computer Vision Algorithms. In *Handbook of Research on Computer Vision and Image Processing in the Deep Learning Era* (pp. 1–18). IGI Global.

Rokh, B., Azarpeyvand, A., & Khanteymoori, A. (2023). A comprehensive survey on model quantization for deep neural networks in image classification. *ACM Transactions on Intelligent Systems and Technology*, *14*(6), 1–50.

Roy, S. K., Deria, A., Shah, C., Haut, J. M., Du, Q., & Plaza, A. (2023). Spectral–Spatial Morphological Attention Transformer for Hyperspectral Image Classification. *IEEE Transactions on Geoscience and Remote Sensing*, *61*, 1–15.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Ryu, G., & Choi, D. (2023). A hybrid adversarial training for deep learning model and denoising network resistant to adversarial examples. *Applied Intelligence*, *53*(8), 9174–9187.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Sarigul, M., Ozyildirim, B. M., & Avci, M. (2020). Deep Convolutional Generalized Classifier Neural Network. *Neural Processing Letters*, *51*(3), 2839–2854. https://doi.org/10.1007/s11063-020-10233-8

Sarıgül, M., Ozyildirim, B. M., & Avci, M. (2019). Differential convolutional neural network. *Neural*

Networks, *116*, 279–287. https://doi.org/10.1016/j.neunet.2019.04.025

Schorr, C., Goodarzi, P., Chen, F., & Dahmen, T. (2021). Neuroscope: An explainable ai toolbox for semantic segmentation and image classification of convolutional neural nets. *Applied Sciences*, *11*(5), 2199.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *ArXiv Preprint ArXiv:1312.6229*.

Shalileh, S. (2023). Classification Using Marginalized Maximum Likelihood Estimation and Black-Box Variational Inference. In *Data Analysis and Optimization: In Honor of Boris Mirkin's 80th Birthday* (pp. 349–361). Springer.

Sharma, V., & Mir, R. N. (2020). A comprehensive and systematic look up into deep learning based object detection techniques: A review. *Computer Science Review*, *38*, 100301.

Shorten, C., & Khoshgoftaar, T. M. (2022). An Exploration of Consistency Learning with Data Augmentation. *The International FLAIRS Conference Proceedings*, *35*.

Silvano, C., Ielmini, D., Ferrandi, F., Fiorin, L., Curzel, S., Benini, L., Conti, F., Garofalo, A., Zambelli, C., & Calore, E. (2023). A survey on deep learning hardware accelerators for heterogeneous hpc platforms. *ArXiv Preprint ArXiv:2306.15552*.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *ArXiv Preprint ArXiv:1409.1556*.

Singh, S. A., & Desai, K. A. (2023). Automated surface defect detection framework using machine vision and convolutional neural networks. *Journal of Intelligent Manufacturing*, *34*(4), 1995–2011.

Stephen, A., Punitha, A., & Chandrasekar, A. (2023). Designing self attention-based ResNet architecture for rice leaf disease classification. *Neural Computing and Applications*, *35*(9), 6737–6751.

Sun, Y., Xue, B., Zhang, M., Yen, G. G., & Lv, J. (2020). Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, *50*(9), 3840–3854.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1–9.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2820–2828.

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning*, 6105–6114.

Taye, M. M. (2023). Theoretical understanding of convolutional neural network: concepts,

architectures, applications, future directions. *Computation*, *11*(3), 52.

Teodoro, A. A. M., Silva, D. H., Rosa, R. L., Saadi, M., Wuttisittikulkij, L., Mumtaz, R. A., & Rodriguez, D. Z. (2023). A skin cancer classification approach using gan and roi-based attention mechanism. *Journal of Signal Processing Systems*, *95*(2–3), 211–224.

Ter-Sarkisov, A. (2020). Lightweight model for the prediction of COVID-19 through the Detection and Segmentation of lesions in chest ct scans. *MedRxiv*, 2010–2020.

Termritthikun, C., Jamtsho, Y., Muneesawang, P., Zhao, J., & Lee, I. (2023). Evolutionary neural architecture search based on efficient CNN models population for image classification. *Multimedia Tools and Applications*, *82*(16), 23917–23943.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., & McDaniel, P. (2017). Ensemble adversarial training: Attacks and defenses. *ArXiv Preprint ArXiv:1705.07204*.

Tripathi, M. (2021). Analysis of convolutional neural network based image classification techniques. *Journal of Innovative Image Processing (JIIP)*, *3*(02), 100–117.

Utkarsh Saxena. (2022a). *10 Species of Monkey - Multiclass Classification*. https://www.kaggle.com/datasets/utkarshsaxenadn/10-species-of-monkey-multiclass-classification

Utkarsh Saxena. (2022b). *Flower Classification*. https://www.kaggle.com/datasets/utkarshsaxenadn/flower-classification-5-classes-roselilyetc

Vermeire, T., Brughmans, D., Goethals, S., de Oliveira, R. M. B., & Martens, D. (2022). Explainable image classification with evidence counterfactual. *Pattern Analysis and Applications*, *25*(2), 315–335.

Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2011). *The caltech-ucsd birds-200-2011 dataset*.

Wang, H., Fu, T., Du, Y., Gao, W., Huang, K., Liu, Z., Chandak, P., Liu, S., Van Katwyk, P., & Deac, A. (2023). Scientific discovery in the age of artificial intelligence. *Nature*, *620*(7972), 47–60.

Wang, S., & Gui, W. (2020). Corrected maximum likelihood estimations of the lognormal distribution parameters. *Symmetry*, *12*(6), 968.

Wang, W., Zhang, X., Cui, H., Yin, H., & Zhang, Y. (2023). FP-DARTS: Fast parallel differentiable neural architecture search for image classification. *Pattern Recognition*, *136*, 109193.

Wang, Y., Yang, G., Li, S., Li, Y., He, L., & Liu, D. (2023). Arrhythmia classification algorithm based on multi-head self-attention mechanism. *Biomedical Signal Processing and Control*, *79*, 104206.

Wang, Y., Zou, D., Yi, J., Bailey, J., Ma, X., & Gu, Q. (2019). Improving adversarial robustness requires revisiting misclassified examples. *International Conference on Learning Representations*.

White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., & Hutter, F. (2023). Neural architecture search: Insights from 1000 papers. *ArXiv Preprint ArXiv:2301.08727*.

Woods, W., Chen, J., & Teuscher, C. (2019). Adversarial explanations for understanding image classification decisions and improved neural network robustness. *Nature Machine Intelligence*, *1*(11), 508–516.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., & Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10734–10742.

Xiao, X., Chen, Y., Gong, Y.-J., & Zhou, Y. (2020). Low-rank preserving t-linear projection for robust image feature extraction. *IEEE Transactions on Image Processing*, *30*, 108–120.

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *ArXiv Preprint ArXiv:1505.00853*.

Yadav, A., Verma, V. K., Pal, V., & Singh, S. (2021). Automatic detection of COVID 19 infection using deep learning models from X-ray images. *IOP Conference Series: Materials Science and Engineering*, *1099*(1), 12050.

Yanamala, R. M. R., & Pullakandam, M. (2023). A high-speed reusable quantized hardware accelerator design for CNN on constrained edge device. *Design Automation for Embedded Systems*, 1–25.

Yang, Y., Panagopoulou, A., Zhou, S., Jin, D., Callison-Burch, C., & Yatskar, M. (2023). Language in a bottle: Language model guided concept bottlenecks for interpretable image classification. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 19187–19197.

Yang, Z., Sinnott, R. O., Bailey, J., & Ke, Q. (2023). A survey of automated data augmentation algorithms for deep learning-based image classification tasks. *Knowledge and Information Systems*, *65*(7), 2805–2861.

Yasin, M., Sarıgül, M., & Avci, M. (2024). Logarithmic learning differential convolutional neural network. *Neural Networks*, 106114.

Yeo, Y.-J., Sagong, M.-C., Park, S., Ko, S.-J., & Shin, Y.-G. (2023). Image generation with self pixel-wise normalization. *Applied Intelligence*, *53*(8), 9409–9423.

Yoo, J., & Kang, S. (2023). Class-Adaptive Data Augmentation for Image Classification. *IEEE Access*, *11*, 26393–26402.

Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., & Lee, B. (2022). A survey of modern deep learning based object detection models. *Digital Signal Processing*, 103514.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6848–6856.

Zhong, Z., Li, Y., Ma, L., Li, J., & Zheng, W.-S. (2021). Spectral–spatial transformer network for hyperspectral image classification: A factorized architecture search framework. *IEEE Transactions on Geoscience and Remote Sensing*, *60*, 1–15.

Zhou, X.-Y., Sun, J., Ye, N., Lan, X., Luo, Q., Lai, B.-L., Esperanca, P., Yang, G.-Z., & Li, Z. (2020). Batch group normalization. *ArXiv Preprint ArXiv:2012.02782*.

Zhu, H., Zeng, H., Liu, J., & Zhang, X. (2021). Logish: A new nonlinear nonmonotonic activation function for convolutional neural network. *Neurocomputing*, *458*, 490–499. https://doi.org/10.1016/j.neucom.2021.06.067

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, *109*(1), 43–76.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8697–8710.

Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2023). Object detection in 20 years: A survey. *Proceedings of the IEEE*.

CURRICULUM VITAE

Magombe Yasin, a dedicated professional from Uganda, demonstrated commitment in Information Technology and Computer Science. From Nov 2018 to Sep 2019, he was Acting Head of ICT at Islamic University in Uganda. Since 2013, Magombe served as a part-time Lecturer in Computer Science. From Oct 2011, he held roles as User Support Technician, Systems Administrator, and ICT Officer, enhancing ICT infrastructure. Magombe's academic journey includes a BIT degree from the Islamic University in Uganda (2012) and an M.Sc.T.E. in Computer Science and Engineering from the Islamic University of Technology, Dhaka, Bangladesh.