1 2 9 0

UNIVERSIDADE Đ
COIMBRA

Nguyen Huu Phong

Pattern Recognition: Contributions and Applications to Image Classification and Video Recognition

PhD Thesis in Informatics Engineering, Intelligent Systems, supervised by Professor Bernardete Ribeiro, and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

April 2023

Abstract

Convolutional Neural Networks (CNNs) have demonstrated remarkable performance in tasks such as image classification and action recognition, outperforming other methods. However, Capsule Networks (CNs) offer promising new architectures for the community. While the effectiveness of CNs has been demonstrated on datasets such as MNIST and smallNORB, they face challenges when dealing with images containing distinct contexts.

In this thesis, we propose an improved design for CNs (Vector version) by incorporating additional Pooling layers to filter out image backgrounds and increasing the number of Reconstruction layers to enhance image restoration. Moreover, we perform experiments to compare the accuracy and speed of CNs with Deep Learning (DL) models such as Inception V3, DenseNet V201, NASNet, MobileNet V1, and MobileNet V2, tailored for powerful computers and small/embedded devices. To evaluate our models, we utilize a fingerspelling alphabet dataset from American Sign Language (ASL). Our results demonstrate that CNs perform comparably to DL models while significantly reducing the training time. Additionally, we provide a demonstration and a link to illustrate our approach.

Capsule Networks are exciting and promising in computer vision, but they face a critical challenge when dealing with image backgrounds, which can significantly impact their performance. To address this issue, we propose an improved Capsule Networks architecture that replaces the Standard Convolution with a Depthwise Separable Convolution. This new design significantly reduces the total number of parameters in the model while enhancing its stability and providing competitive accuracy. Furthermore, our proposed model performs exceptionally well on 64×64 pixel images and outperforms standard models on both 32×32 and 64×64 pixel images. To evaluate our models, we conduct empirical experiments using state-of-the-art Transfer Learning networks such as Inception V3 and MobileNet V1. The results show that Capsule Networks can perform comparably against Deep Learning models. Notably, our work is the first to integrate Depthwise Separable Convolution into Capsule Networks, and we believe that this approach holds significant potential for improving the performance of Capsule Networks.

In addition to exploring Capsule Networks, in this thesis we investigate the potential of recurrent neural networks (RNNs) for image recognition tasks. Although RNNs are commonly used for sequential data and time series with 1-D information, we propose integrating an RNN as an additional layer in the design of image recognition models. We also develop end-to-end multi-model ensembles that combine several models to produce expert predictions. To achieve state-of-the-art performance, we extend the training strategy of our model, resulting in comparable or even superior results compared to leading models on challenging datasets such as SVHN (0.99), Cifar-100 (0.9027), and Cifar-10 (0.9852). Our model also sets a new record on the Surrey dataset with an accuracy of 0.949. This research highlights the potential of integrating RNNs into image recognition models and the effectiveness of multi-model ensembles for improving performance. Our proposed approach achieves highly competitive results and contributes to advancing the state-of-the-art in image recognition tasks.

While Convolutional Neural Networks have been widely used in image classification, action recognition, and other fields, the challenges and dynamics of training these neural networks are still not fully understood, and training them can be computationally expensive. To address this challenge, numerous architectures and training strategies have been proposed for improving the performance of CNNs in image processing tasks, such as speech, image and action recognition, and object detection.

In this work, we propose a novel approach for training CNNs using Particle Swarm Optimization (PSO). Our PSO-based training framework allows for the collaborative dynamics of the PSO algorithm to interplay with Stochastic Gradient Descent (SGD) to improve training performance and generalization. Specifically, we first train each CNN independently via SGD in a regular phase. In a collaborative phase, the CNNs share their current vector of weights (or particle-position) along with their gradient estimates of the loss function. We use distinct step sizes for each CNN and blend CNNs with large (possibly random) step-sizes along with more conservative ones. Our approach achieves competitive performance compared to other PSO-based approaches on the challenging Cifar-10 and Cifar-100 datasets, achieving an accuracy of 98.31% and 87.48%, respectively, using only four collaborative CNNs. We expect these results to scale with the number of collaborative CNNs. By using PSO-based training for CNNs, we provide a novel and effective approach to improve their training performance, especially in complex image processing tasks. This work contributes to a deeper understanding of the dynamics of training ConvNets and demonstrates the potential of PSO-based training for improving their performance in challenging image recognition tasks.

The work presented introduces the concept of collaborative learning and we further propose to apply it to the challenging task of recognizing human actions in videos, known as Human Action Recognition (HAR). While CNNs have shown remarkable success in image recognition, they are not always directly applicable to HAR, as temporal features are critical for accurate classification. In this thesis, we propose a novel dynamic PSO-ConvNet model that leverages the collaborative learning framework where the weight vector of each neural network represents the position of a particle in phase space. Through this framework, particles share their current weight vectors and gradient estimates of the Loss function. To extend our approach to video, we integrate CNNs with state-of-the-art temporal methods such as Transformer and Recurrent Neural Networks. Our experimental results on the challenging UCF-101 dataset demonstrate substantial improvements of up to 9% in accuracy, which confirms the effectiveness of our proposed method. Overall, our dynamic PSO-ConvNet model provides a promising direction for improving HAR by better capturing the spatio-temporal dynamics of human actions in videos.

Keywords: image classification, action recognition, pattern recognition, deep learning, computer vision, convolution neural networks, recurrent neural networks, capsule networks, transformer, particle swarm optimization, collaborative learning

Resumo

As Redes Neuronais Convolucionais (CNNs) demonstraram um desempenho notável em tarefas como classificação de imagem e reconhecimento de ações, superando outros métodos. No entanto, as Redes de Cápsulas (CNs) oferecem novas arquiteturas promissoras para a comunidade científica. Embora a eficiência das CNs tenha sido demonstrada em conjuntos de dados como MNIST e smallNORB, enfrentam ainda assim desafios com imagens contendo contextos distintos.

Nesta tese, propomos uma versão aperfeiçoada das CNs (versão Vetorial) incorporando camadas adicionais de Pooling para filtrar os planos de fundo das imagens e aumentar o número de camadas de Reconstrução para melhorar a restauração de imagens. Além disso, realizamos experiências para comparar a precisão e velocidade das CNs com modelos de Aprendizagem Profunda (DL) como Inception V3, DenseNet V201, NASNet, MobileNet V1 e MobileNet V2, adaptados para computadores poderosos e dispositivos pequenos/embutidos. Para avaliar os modelos, utilizamos um conjunto de dados de alfabeto manual do American Sign Language (ASL). Os resultados obtidos demonstram que as CNs têm um desempenho comparável aos modelos DL, enquanto reduzem significativamente o tempo de treino. Para efeitos de demonstração, disponibilizamos um link para ilustrar esta abordagem.

As Redes de Cápsulas são muito promissoras na visão computacional, mas enfrentam um desafio crítico ao lidar com fundos de imagens, o que pode afetar significativamente o seu desempenho. Para lidar com esta questão, propomos uma arquitetura de Redes de Cápsulas aperfeiçoada que substitui a Convolução Padrão por uma Convolução Separável em Profundidade. Este novo design reduz significativamente o número total de parâmetros no modelo, enquanto permite melhorar não só a sua estabilidade, mas também apresentar uma precisão competitiva. Além disso, o modelo proposto funciona excepcionalmente bem em imagens de 64×64 pixels e supera os modelos padrão em imagens de 32×32 e 64×64 pixels. Para avaliar estes modelos, realizámos experiências usando redes de Transfer Learning de ponta, como Inception V3 e MobileNet V1. Os resultados mostram que as Redes de Cápsulas podem ter desempenho comparável aos modelos de Aprendizagem Profunda.

Destaca-se que o trabalho desenvolvido nesta tese é o primeiro a integrar a Convolução Separável em Profundidade nas Redes de Cápsulas, e tudo leva a crer que esta abordagem tenha um potencial significativo para melhorar o desempenho das Redes de Cápsulas.

Para além de explorar as Redes de Cápsulas, investigamos o potencial das redes neuronais recorrentes (RNNs) para tarefas de reconhecimento de imagem. Embora as RNNs sejam comumente usadas para dados sequenciais e séries temporais com informação 1-D, propomos integrar uma RNN como camada adicional no design de modelos de reconhecimento de imagem. Também desenvolvemos conjuntos de modelos multimodelos de ponta a ponta que combinam vários modelos para produzir previsões especializadas. Para alcançar um desempenho de última geração, estendemos a estratégia de treino do nosso modelo, resultando em resultados comparáveis ou até superiores em comparação com os principais modelos em conjuntos de dados mais complexos, como SVHN (0,99), Cifar-100 (0,9027) e Cifar-10 (0,9852). O nosso modelo também atinge um novo recorde no conjunto de dados Surrey, com uma precisão de 0,949. Este estudo destaca o potencial de integração de RNNs em modelos de reconhecimento de imagem e a eficiência de conjuntos de modelos multimodelos para melhorar o desempenho. A abordagem proposta alcança resultados altamente competitivos e contribui para avançar o estado-da-arte em tarefas de reconhecimento de imagem.

Embora as Redes Neuronais Convolucionais tenham sido amplamente utilizadas na classificação de imagens, reconhecimento de ações e outras áreas, os desafios e dinâmicas de treino dessas redes neuronais ainda não são completamente compreendidos, e o treino pode ser computacionalmente elevado. Para responder a este desafio, inúmeras arquiteturas e estratégias de treino têm sido propostas para melhorar o desempenho de CNNs em tarefas de reconhecimento de fala, imagem e ação, e detecção de objetos.

Nesta tese, propomos uma abordagem inovadora para treinar CNNs usando a Otimização por Enxame de Partículas (PSO). A nossa framework de treino baseada em PSO permite que a dinâmica colaborativa do algoritmo PSO interaja com o Gradiente Estocástico Descendente (SGD) para melhorar o desempenho e a generalização do treino. Especificamente, primeiro treinamos cada CNN independentemente via SGD numa fase dita regular. Numa fase colaborativa, as CNNs compartilham seu vetor atual de pesos (ou posição de partícula) juntamente com as estimativas do gradiente da função de perda. Usamos tamanhos de passo distintos para cada CNN e combinamos CNNs com tamanhos de passo grandes (possivelmente aleatórios) juntamente com tamanhos de passo mais conservadores. Esta abordagem alcança um desempenho competitivo em comparação com outras abordagens baseadas em PSO nos conjuntos de dados Cifar-10 e Cifar-100, alcançando uma precisão de 98,31% e 87,48%, respectivamente, usando apenas quatro CNNs colaborativos. Estamos em crer, que estes resultados possam vir a melhorar substancialmente com o aumento do número de CNNs colaborativos.

Ao adotarmos o treino baseado em PSO para as CNNs, propomos uma abordagem inovadora e eficaz que contribui para melhorar significativamente o desempenho destas redes, especialmente em tarefas complexas de processamento de imagem. Além disso, este trabalho também contribui para uma compreensão mais profunda da dinâmica de treino das CNNs e evidencia o potencial do treino baseado em PSO para aprimorar o desempenho dessas redes em tarefas desafiadoras de reconhecimento de imagem.

O trabalho realizado apresenta o conceito de aprendizagem colaborativa e aplica-o à desafiadora tarefa de reconhecimento de ações humanas em vídeos, conhecida como Reconhecimento de Ações Humanas (HAR). Embora as CNNs tenham demonstrado um sucesso notável no reconhecimento de imagens, nem sempre são diretamente aplicáveis no reconhecimento de ações humanas uma vez que as características temporais são críticas para uma classificação precisa. Nesta tese, propomos um novo modelo dinâmico PSO-ConvNet que utiliza a framework de aprendizagem colaborativa onde o vetor de pesos de cada rede neuronal representa a posição de uma partícula no espaço de fases. Através desta framework, as partículas partilham os seus vetores de pesos correntes bem como as estimativas de gradientes da função de perda. Para estender a nossa abordagem a vídeos, integramos as CNNs com métodos temporais de ponta, como o Transformer e as Redes Neuronais Recorrentes. Os resultados experimentais em conjunto de dados complexos e desafiante como o UCF-101 demonstram melhorias substanciais de até 9% em precisão, o que confirma a eficácia do nosso método proposto. Em geral, o nosso modelo dinâmico PSO-ConvNet aponta uma direção promissora para contribuir na área de HAR, capturando melhor as dinâmicas espaço-temporais das ações humanas em vídeos.

Palavras Chave: classificação de imagem, reconhecimento de ações, reconhecimento de padrão, aprendizagem profunda, visão computacional, redes neuronais convolucionais, redes neuronais recorrentes, redes de cápsulas, transformer, otimização por enxame de partículas, aprendizagem colaborativa

Acknowledgments

In my reflection, undertaking this PhD study has truly been a life-changing journey for me and it would never have been possible without the strong support and supervision I received from many people.

Initially, I would like to take this opportunity to express my deep gratitude to my supervisor Bernardete Ribeiro for her devoted works, diligent and wise support as well as invaluable guidance, which ensured this uncertain research arrived on the right track. She provided me with freedom to explore my own ideas that at times led to exciting findings and also helped me to stay within the scope. I have learned a lot about academic research from her outstanding experience and broad knowledge of science.

I would also like to thank all my friends, specially at LARN Lab for their friendship and support to make my PhD life in Portugal enjoyable and achieve remarkable milestones. My special appreciation is dedicated to my wife and two wonderful kids for their unconditional love, in addition to my parents for their endless care and support. They contributed a huge part of what I am today, I owe to all of them.

Finally, the research that led to this thesis would not have been possible without the funding and support provided by the FEDER funds through the programs COMPETE – "Programa Operacional Factores de Competitividade" and Centro2020 – "Centro Portugal Regional Operational Programme", and by national funds through FCT – "Fundação para a Ciência e a Tecnologia", under the project UIDB/00326/2020, and by the Centre for Informatics and Systems of University of Coimbra (CISUC).

Contents

List of Tables

List of Figures

List of Acronyms

ABCO	Artificial Bee Colony Optimization. 62
ACO	Ant Colony Optimization. 62
ASL	American Sign Language. 3, 14–19, 24, 26, 28–30, 120–124
BiLSTM	Bidirectional LSTM. 59
BiRNN	Bidirectional RNN. 38, 39
BP	Back-propagation. 11, 13
CLR	Cyclical Learning Rate. 12
CN	Capsule Network. 1–4, 6, 14, 15, 17, 19–26, 29, 30, 32, 33, 107
CNN	Convolutional Neural Network. 113–118
ConvNet	Convolution Neural Network. 1–4, 6–8, 11–13, 15, 17, 18, 20–23, 28–30, 32, 36–39, 41–43, 45, 50, 54, 59–61, 66–70, 72, 77, 79, 80, 91, 98, 106, 107, 111, 116, 119
DC	Depthwise Convolution. 25, 28
DL	Deep Learning. 1–4, 6, 12–15, 18–26, 28–30, 32, 33, 40, 42, 110, 112, 116, 118–120
DW	Depthwise Separable Convolution. 3, 4, 6, 20, 22, 25, 26, 28–30, 32, 33
E2E	End-To-End. 4, 36, 42, 59
EC	Evolutionary Computation. 2, 11
EL	Ensemble learning. 40, 50
FC	Fully Connected. 3, 8, 9, 15, 19, 22, 23, 28, 42, 44, 77, 107, 117, 118

FGVC Fine-grained Visual Categorization. 43

FLOPS Floating-point Operations per Second. 45

GA Genetic Algorithm. 12, 13

GAP Global Average Pooling. 42, 77

GPU Graphics Processing Unit. 1, 7, 11, 21, 22, 36, 39, 42, 44, 47, 51, 69, 77, 96, 110

GRU Gated Recurrent Unit. 4, 38, 39, 46, 59, 60, 99

HAR Human Action Recognition. 2, 7, 9, 10, 12, 13, 91, 106

ILSVRC ImageNet Large-Scale Visual Recognition Challenge. 7, 8, 18

KNN K-Nearest Neighbors. 4, 73–75, 82

LRCN Long-term Recurrent Convolutional Networks. 10

LSTM Long Short Term Memory. 3, 4, 10, 15, 19, 20, 28–30, 32, 33, 38, 39, 92, 98, 99, 122, 123

ML Machine Learning. 15, 40

MLP Multilayer Perceptron. 3, 15, 19, 20, 28–30, 32, 33, 111, 113–115, 122

MNIST Modified National Institute of Standards and Technology. 14, 16, 25, 47

ModPSO Modified Particle Swarm Optimization. 11, 13

MP Max Pooling. 14, 30, 32, 42

NAS Neural Architecture Search. 12

NLP Natural Language Processing. 110

NLU Natural Language Understanding. 110

PC Pointwise Convolution. 25, 28

PSO Particle Swarm Optimization. 2, 4, 6, 7, 9, 11, 13, 62, 63, 67, 68, 70, 73–80, 82, 92

ReLU Rectified Linear. 8, 113, 117

RNN Recurrent Neural Network. 2–4, 6, 7, 10, 36–40, 42, 45–47, 50, 54, 59, 60, 91, 92, 96, 98, 99, 106

SC Standard Convolution. 3, 6, 26, 27, 29, 30, 32, 33

SGD Stochastic Gradient Descent. 4, 6, 61, 63, 73, 78, 93, 117

SoC System on a Chip. 37, 60

SVHN Street View House Numbers. 2, 4, 14, 42, 56–58, 68

SVM Support Vector Machine. 110

TL Transfer Learning. 6, 7, 14, 15, 18, 19, 28–30, 43, 70, 77, 78

TTA Test Time Augmentation. 7, 37, 49

List of Symbols

The notation utilized in this thesis is intended to be consistent and intuitive, while being as coherent as possible with the state of the art. In order to achieve that, symbols can sometimes change meaning during one or more chapters. However, such changes will be explicitly mentioned in the text, so that the meaning of a given symbol is always clear from the context.

Lowercase letters, such as x, represent variables where the n_{th} element is referred as x_n. All vectors are assumed to be column vectors. Uppercase Roman letters, such as M, denote constants. A superscript $\intercal$ denotes the transpose of a matrix or vector, so transpose of vector $u^\intercal$ will be a row vector.

General notations

∇	Vector of first derivatives	
e	Euler's number	$2.71828\ldots$
$x \sim p$	x is distributed according to distribution p	
$\exp(x)$	Exponential function, $\exp(x) = e^x$	
i.i.d.	Independent and Identically Distributed	
$\mathrm{ReLU}(x)$	Rectified linear unit activation function, $\mathrm{ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$	
$\mathrm{Sigmoid}(x)$	Sigmoid (logistic) activation function, $\mathrm{Sigmoid}(x) = 1/(1+e^{-x})$	
$\mathrm{Tanh}(x)$	Tanh (hyperbolic) activation function, $\mathrm{Tanh}(x) = (e^x - e^{-x})/(e^x + e^{-x})$	

Deep Learning notations

$\mathcal{N}(n,t)$	Set of k nearest neighbor particles of particle n at time t
$\phi^{(n)}(t)$, $\psi^{(n)}(t)$	Intermediate position and intermediate velocity of particle n at time t.
$\theta(t)$	Vector collects all the weights of the Neural Network at the iteration t

$\widehat{L}$	Estimate of loss function.
$\{x_i\}_{i=1}^T$	Set of T samples $x_1, x_2, \ldots, x_T$
c, c_1, c_2	Accelerator coefficients.
D	Number of weights (and dimension of the phase space)
f_t, i_t, o_t, c_t, $c_t^{'}$	Forget gate, input gate, output gate and cell states at time step t.
M, β	Constants
S	Subset of particles(Chapter 6) Set of videos (Chapter 7)
W	Weight matrix of the layer
w	Inertia weight of PSO
$x^n(t)$, $v^n(t)$	Position and velocity vector of particle n at time t.
y_t, $\hat{y}_t$	Output of a cell at time step t.
σ	Sigmoid function.
b	Routing coefficient (Chapter 3) Bias constant (Chapter 5).
f_i	Weight operation for convolution, pooling or fully connected layers at layer i_{th}.
g_i	Activation function at layer i_{th}.
h_t	Hidden cell state at time step t.
L	Loss function.
O_i	Output for layer i_{th}.
$P^n(t)$	Best position visited up until time t by particle n.
$P_g^n(t)$	Best position across all previous positions of the particle n jointly with its nearest-neighbors up until time t.
$r(t)$	Random uniform within the interval [0,1].
X	Input image (Chapter 2) Input frame in video sequences for ConvNet (Chapter 7).
x_t	Input sequence of RNN at time step t.

F, G, $\hat{\mathrm{G}}$	Feature maps
Softmax(x,W)	Softmax function, Softmax$(x,W)_c = \frac{\exp(W_c^\intercal x)}{\sum_l \exp(W_l^\intercal x)}$, for $c \in \{0, 1, 2, \ldots, C\}$

Chapter 1

Introduction

1.1 Motivation

Hinton, who is the godfather of Neural Networks, explained the idea in a video lecture that "If we want to tell a computer to do a thing, we program steps exactly the way we would do by ourselves. But if we like a computer to describe an image, how could we program? Can the computer learn from just seeing the images?". It was the reason for the foundation of Neural Networks.

Though, after decades of active development, the field was slowed down to the point where articles from top researchers could get rejected as the topics were not in the interests of journals. However, recent years have seen a re-appearance of Neural Networks (as well as Deep Learning, generally) thanks to the availability of Graphics Processing Unit (GPU) and BigData [101].

In academy, the technique has achieved significant higher classification accuracy on competitions such as image recognition [79] and speech recognition [124]. In business, Google self-driving cars have been tested in large cities and accumulated hundred years of human driving experience [51]. Uber also made a breakthrough in public service as the first company to offer self-driving cars [16].

Before the arrival of Deep Learning (DL) in image classification, the field has evolved through several stages from Linear Classifier to Support Vector Machine and Neural Networks. These methods commonly require selection of features that eventually needs involvement of experts in particular fields. DL on the other hand can choose the best feature automatically [101].

1.2 Problem Statement

Inspired by the above motivation, we formulate these five research questions:

1. Can Capsule Networks perform better than the current best method in image recognition? Convolution Neural Networks (ConvNets) have shown superiority in the field of image recognition. Capsule Network (CN) is introduced and expected to refresh ideas in the area. However, the networks can show improvements only for 1-D images in MNIST and smallnorb datasets, there are still rooms for improvement in 3-D images, e.g., Cifar-10, Cifar-100 and Street View House Numbers (SVHN) datasets, etc.

2. How can Capsule Network be improved? Capsule Networks face a critical problem in computer vision in the sense that the image background can challenge its performance, although they learn very well on training data. To solve this problem, we argue that primary filters to eliminate such backgrounds should be as important as other parts of CNs' architecture. Just, improvements on speed and accuracy of these convolution layers can be bonus points for the network.

3. Can we apply Recurrent Neural Networks for image recognition and how about other techniques? Over the long history of machine learning, which dates back several decades, Recurrent Neural Networks (RNNs) have been used mainly for sequential data and time series and generally with 1-D information. Even in some rare studies on 2-D images, these networks are used merely to learn and generate data sequentially rather than for image recognition tasks. In this study, we try to integrate an RNN as an additional layer for image recognition models and other relevant techniques to leverage the accuracy.

4. How does collaborative learning enhance image classification? ConvNets have been candidly deployed in the scope of computer vision and related fields. Nevertheless, the dynamics of training of these Neural Networks lie still elusive: it is hard and computationally expensive to train them. A myriad of architectures and training strategies have been proposed to overcome this challenge and address several problems in image processing such as speech, image and action recognition as well as object detection. In this sense, Particle Swarm Optimization (PSO) is a better choice as the algorithm requires much less computation. In contrast to Evolutionary Computation (EC) methods which evolve via competition, in PSO, particles cooperate to share information, e.g., best position, current location and direction.

5. Can collaborative learning apply to other fields? Recognizing human actions in video sequences, known as Human Action Recognition (HAR), is a challenging task in pattern recognition. While ConvNets have shown remarkable success in image recognition, they are not always directly applicable to HAR, as temporal features are critical for accurate classification. Therefore, we propose a novel dynamic PSO-ConvNet model for learning actions in videos, building on our recent work in image recognition. Our approach leverages a framework where the weight vector of each neural network represents the position of a particle in phase space, and particles share their current weight vectors and gradient estimates of the Loss function.

1.3 Contributions

The aim of this thesis is to classify images and recognize actions with the latest emerging Deep Learning techniques. The first two parts of our work investigated the use of Capsule Networks, which was first proposed for 1-D images, for 3-D images instead. A comparison study was performed to analyze the performance of the CNs versus ConvNets on image recognition. An extension of CNs with integration of Depthwise Separable Convolution (DW) was also performed.

Most studies utilize RNNs for sequential data and time series. Except for rare cases, RNNs are used largely to generate sequences of image pixels. In the third part, we propose integrating RNNs as an essential layer of ConvNets. In addition, we present our core idea for designing a ConvNet in which the model is able to learn multiple decisions from expert models. Another main contribution involves a training strategy that allows our models to perform competitively, matching previous approaches on several datasets and outperforming state-of-the-art models.

Extensive experiments and an extended study were carried in the fourth part of this work introduces a new dynamic collaborative learning using hybrid of PSO-ConvNet and we utilized this proposed model for image classification. We investigated the effect of distinct parameters (M and β), number of nearest neighbors, additional strategies for improvement.

The fifth part of this thesis attempts to extend the framework proposed in the fourth part to recognize human actions in video. Then, we proposed the integration of dynamics in the hybrid of ConvNet with recent advanced methods for temporal data, e.g., Transformer and Recurrent Neural Network. The results demonstrate a round improvement of $2\% - 9\%$ in accuracy over baseline methods, such as an 8.72% increase in accuracy for the DenseNet-201 Transformer using Dynamics 2 and a 7.26% increase in accuracy for the ResNet-152 Transformer using Dynamics 1. Our approach outperforms the current state-of-the-art methods, offering significant improvements in video action recognition. To sum up, the following contributions can be identified from this thesis:

- We improve the design of the Vector Capsule Networks and perform empirical comparisons versus Deep Learning models on accuracy and speed using an American Sign Language (ASL) finger spelling alphabet. First, we propose to modify the Vector CN's architecture to find the most efficient designs. Namely, we extended convolution layers to better filter input images and varied Fully Connecteds (FCs) layers in the Reconstruction to leverage image restoration. Second, we explore distinct DL models for demanding devices (Inception V3 and DenseNet V201) and small devices (NasNet and MobileNets) when integrated with Multilayer Perceptron (MLP) and Long Short Term Memory (LSTM). We use the former setting as a baseline to compare with the latter. Third, Vector CNs are analyzed against DL models. We find out that CNs perform comparatively on both accuracy and speed. In addition, CNs have an advantage as pre-training is not required. Capsules can be trained within an hour compare to days or more for pre-trained TL models. Finally, we make a demonstration to compare CNs and DLs on videos for teaching ASL alphabets. The results are very promising as our models can recognize almost of all signs without previously seen.

- We propose to replace Standard Convolution (SC) with Depthwise Separable Convolution in Capsule Networks' architecture and performed empirical evaluations of the proposed CNs against DLs models. Regarding the design of CNs, we found that the integration of DW can significantly reduce the model size, increase stability and yield higher accuracy than its counterpart. With respect to experimental evaluations of CNs versus DL models, the Capsule models perform competitively both on model size and accuracy.

- We attempt to integrate RNNs into ConvNets even though RNNs are mainly optimized for 1-D sequential data rather than 2-D images. Our results on the Fashion-MNIST dataset show that ConvNets with RNN, GRU and BiLSTM modules can outperform standard ConvNets. Second, we designed E2E-3M ConvNets that learn predictions from several models. Our E2E-3M model outperforms a standard single model by a large margin. The advantage of using an End-To-End (E2E) design is that the model can run immediately in real time and it is suitable for system-on-a-chip platforms. Third, we propose a training strategy and pruning for the softmax layer that yields comparable accuracies on the Cifar-10 and Fashion-MNIST datasets. By using the ensemble technique, our models perform competitively, matching previous state-of-the-art methods (accuracies of 0.99, 0.9027 and 0.9852 on SVHN, Cifar-100 and Cifar-10, respectively) even with limited resources. Moreover, our method outperforms other approaches on the Surrey dataset, achieving an accuracy of 0.949.

- Novel formulations (Dynamics 1 and Dynamics 2) have been successfully created by incorporating distilled Cucker-Smale elements into the PSO algorithm using K-Nearest Neighbors (KNN) and intertwining the training with Stochastic Gradient Descent (SGD). In addition, a new type of particle, i.e., wilder PSO with random learning rate is introduced which has capability of attracting conservative PSOs to stronger minima; Moreover, a distributed environment is developed for parallel collaboration that significantly accelerates the training. The proposed algorithms are evaluated on Cifar-10 and Cifar-100 benchmark datasets and compared to state-of-the-art algorithms to verify the superior effectiveness.

- We incorporate Dynamics 1 and Dynamics 2 into a hybrid model that combines ConvNet with two popular sequence modeling techniques - RNN and Transformer. Next, we extend the distributed collaborative learning framework to address the task of human action recognition. Extensive experiments on the challenging UCF-101 dataset were performed and compared against state-of-the-art methods to validate its effectiveness.

- We make our software available as open sources according to GNU General Public License v3.0 at GitHub.com for each respective paper as follows: "Rethinking recurrent neural networks and other improvements for image classification" `https://github.com/leonlha/e2e-3m`, "Pso convolutional neural networks with heterogeneous learning rate" `https://github.com/leonlha/PSO-ConvNet-Dynamics` and "Video action recognition collaborative learning with dynamics via pso-convnet transformer" `https://github.com/leonlha/Video-Action-Recognition-Collaborative-Learning-with-Dynamics-via-PSO-ConvNet-Transformer`

The contributions mentioned above are resulted in the following list of publications including three international journal (two of them are peer-reviewed Q1 journals) and four international and national conferences. The blanket indicate the chapter(s) or section(s) where the content are referred.

- Nguyen Huu Phong, Augusto Santos, and Bernardete Ribeiro. Pso-convolutional neural networks with heterogeneous learning rate. IEEE Access, 10:89970–89988, 2022. (Chapter 6, Section 2.1.1, Section 2.1.2 Section 2.2.1, and Section 2.2.2)
- Nguyen Huu Phong and Bernardete Ribeiro. Video action recognition collaborative learning with dynamics via pso-convnet transformer. arXiv preprint arXiv:2302.09187, 2023. (Chapter 7, Section 2.1.3 and Section 2.2.3) **(Under review at Scientific Reports Journal)**
- Nguyen Huu Phong and Bernardete Ribeiro. Rethinking recurrent neural networks and other improvements for image classification. arXiv preprint arXiv:2007.15161, 2020. (Chapter 5)
- Nguyen Huu Phong and Bernardete Ribeiro. Advanced capsule networks via context awareness. In Artificial Neural Networks and Machine Learning–ICANN 2019: Theoretical Neural Computation: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part I 28, pages 166–177. Springer, 2019. (Chapter 3)
- Nguyen Huu Phong and Bernardete Ribeiro. An improvement for capsule networks using depthwise separable convolution. In Pattern Recognition and Image Analysis: 9th Iberian Conference, IbPRIA 2019, Madrid, Spain, July 1–4, 2019, Proceedings, Part I, pages 521–530. Springer, 2019. (Chapter 4)
- Nguyen Huu Phong and Bernardete Ribeiro. Offline and online deep learning for image recognition. In 2017 4th Experiment@ International Conference (exp. at'17), pages 171–175. IEEE, 2017. (Appendix A)
- Nguyen Huu Phong and Bernardete Ribeiro. Action recognition for american sign language. In 24th Portuguese Conference on Pattern Recognition (RECPAD). RECPAD, 2018. (Appendix B)

1.4 Outline of the dissertation

The other chapters in this dissertation are structured as follows:

- Chapter 2 provides comprehensive background concepts for this thesis and introduces a review for the current state of the art of image classification, particle swarm optimization and human action recognition fields. In addition, the development of the researches in the past decades is presented to provide a general introduction to traditional and modern approaches for the fields, including various ConvNets models, PSO techniques and action recognition techniques that have been proposed by other researchers.

- Chapter 3 we propose to improve the design of Capsule Network by expanding more pooling layers to filter image backgrounds and increase Reconstruction layers to make better image restoration. Additionally, we perform comparative studies to compare accuracy and speed of CNs versus DLs models.
- Chapter 4 follows the proposed idea in Chapter 3, and presents a solution to enhance the performance of Capsule Networks by utilizing Depthwise Separable Convolution instead of a Standard Convolution. The new design significantly reduces the model's total parameters while increases stability and offers competitive accuracy. Moreover, we empirically evaluate these models with Deep Learning architectures using state-of-the-art TL networks such as Inception V3 and MobileNet V1. The results show that CNs can perform comparably against DL models.
- Chapter 5 focuses on various RNN formulations, including a typical RNN and more advanced RNNs. In addition, our core idea for designing ConvNet models that gain experience from expert models is highlighted. We also discuss our learning rate strategy and the softmax pruning technology. Furthermore, we extend our experiments to include more challenging datasets (i.e., SVHN, Cifar-100, and Cifar-10). We show that our approach outperforms the state-of-the-art methods on Surrey dataset by a large margin.
- Chapter 6 presents a novel model for the image classification, which uses collaborative learning to optimize training process. In this framework, the vector of weights of each ConvNet presents the position of a particle in phase space whereby PSO collaborative dynamics intertwines with SGD in order to boost training performance and generalization. By properly blending ConvNets with large step-sizes along with more conservative ones, we propose an algorithm with competitive performance with respect to other PSO-based approaches on Cifar-10 and Cifar-100 (accuracy of 98.31% and 87.48%).
- Chapter 7 aims to explore the potential of the proposed collaborative learning with dynamics further. The dynamics were extended from image classification to video action recognition by addressing temporal feature property. In order to determine the most efficient features for the hybrid ConvNet with RNN and Transformer models, we explored the performance of different combinations of hyper-parameters extensively on the UCF-101 dataset.
- Chapter 8 summarizes and concludes this thesis, emphasizes the contributions, points out the limitations and highlights directions for the future research.
- Appendices A and B discuss initial works on image classification and action recognition experiments.

Chapter 2

Literature Review

The aim of this thesis is to address image classification and video action recognition problems by using powerful Convolution Neural Network, Recurrent Neural Network, Particle Swarm Optimization and related techniques. In this Chapter, we will review and discuss essential background of ConvNet, PSO and HAR (Section 2.1.1, 2.1.2 and 2.1.3, respectively). We also deal with related works of hybrid PSO-ConvNet, Learning Rate Tuning and Video Action Recognition in Section 2.2.1, 2.2.2 and 2.2.3. For in-depth techniques, e.g., Transfer Learning, Re-training, Test Time Augmentation (TTA), Ensemble Learning, Pruning, etc., we will explain in more detail later in this thesis (see Chapter 5).

2.1 Background

2.1.1 Convolutional Neural Networks

Convolution Neural Networks have demonstrated superior performance when compared to other methods in image classification and related tasks. The technique was originally proposed by LeCun et al. [103] in 1989, though, only after 2012 when AlexNet [97] outperformed contemporary state-of-the-art, ConvNets became the most representative Neural Networks in the area. This breakthrough in popularity has been motivated largely by: (i) availability of high-performance computing hardware—particularly, modern Graphics Processing Units and (ii) promotions of large-scale datasets such as the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). The early designs of ConvNets were shallow and included only a few layers; however, the field has been evolving to cope with the rampant scaling in computer power and image/data resolution. For example, GoogLeNet [170] – which was a winner of ILSVRC 2014, introduced Inception model which helps to reduce the computational cost as it reduces significantly the number of parameters involved in a network. VGGNet [160] in 2014 showed that deeper layers improve the performance of ConvNets. ResNet [65] which was the best of ILSVRC in 2015, introduced the idea of residual learning. The later developments involved a delicate trade-off among network depth, width and image resolution as EfficientNet [172] or adaptation for small size devices (MobileNet [72]). In addition, SqueezeNet [77], SENet [73], DenseNet [76], ResNeXt [196], Xception [22] and other families of Neural Networks thereof, have been proposed and demonstrated to efficiently perform in many applications.

As discussed above, ConvNets differ from one to another in their architectures, though, a typical design is illustrated in Figure 2.1 which stacks multiple trainable layers. The feature extraction module is comprised of several convolution operations for filtering and pooling operations for subsampling (reducing the network size). Within this module, the input and output of each layer are given by arrays and referred to as feature maps. The classification module contains Fully Connected layers.

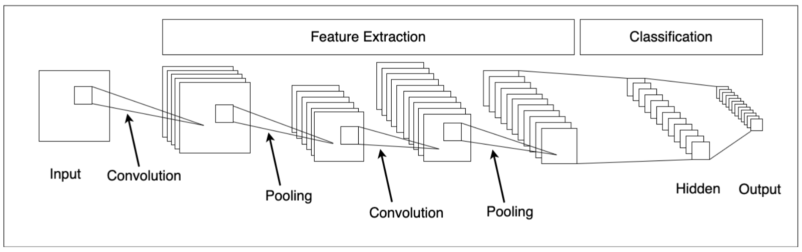


Figure 2.1: Illustration of a traditional ConvNet design [105].

For the sake of clarity regarding the operations comprised by a ConvNet, the mathematical formulations can be summarized as follows:

$$\begin{cases} O_i = X & \text{if } i = 1 \\ Y_i = f_i(O_{i-1}, W_i) & \text{if } i > 1 \\ O_i = g_i(Y_i) \end{cases} \tag{2.1}$$

$$\begin{cases} Y_i = W_i \circledast O_{i-1} & i^{th} \text{ layer is a convolution} \\ Y_i = \boxplus_{n,m} O_{i-1} & i^{th} \text{ layer is a pool} \\ Y_i = W_i * O_{i-1} & i^{th} \text{ layer is a FC} \end{cases} \tag{2.2}$$

where X represents the input image; O_i is the output for layer i^{th}; W_i indicates the weights of the layer; $f_i(\cdot)$ denotes weight operation for convolution, pooling or FC layers; $g_i(\cdot)$ is an activation function, for example, sigmoid, tanh and Rectified Linear (ReLU) or more recently Leaky ReLU [121]; The symbol ($\circledast$) acts as a convolution operation which uses *shared* weights to reduce expensive matrix computation [105]; Window ($\boxplus_{n,m}$) shows an average or a max pooling operation which compute average or max values over neighbor region of size $n \times m$ in each feature map. Matrix multiplication of weights between layer i^{th} and the layer $(i-1)^{th}$ in FC is represented as $(*)$.

2.1.2 Particle Swarm Optimization

Particle Swarm Optimization is a population based stochastic optimization algorithm originally introduced by Kennedy and Eberhart in 1995 [87] and modified in [157]. The attractive feature of PSO is attributed to the ability of particles to learn from others (social behavior) and from their individual experience (cognitive behavior). In PSO, the position of each particle represents a potential solution to a problem and is obtained via random search. At first, each particle (among N independent particles in a D-dimensional phase space) is randomly pre-assigned to a position x in a search space Ω^D, and during the evolution process, continues to update its position according to the following dynamics:

$$\begin{aligned} v_{id}(t+1) &= wv_{id}(t) + c_1 r_1 (P_{id}(t) - x_{id}(t)) \\ &\quad + c_2 r_2 (P_{gd}(t) - x_{id}(t)), \\ x_{id}(t+1) &= x_{id}(t) + v_{id}(t+1). \end{aligned} \tag{2.3}$$

where, $v_{id}(t)$ and $x_{id}(t)$ represent the d component of the velocity and position of particle i at the iteration (or time) t; r_1 and r_2 are independent random variables uniformly distributed over the interval $[0, 1]$; P_{id} and P_{gd} serve as the particle's own best position and swarm's best position; The t denotes the iteration; The parameter w is called "inertia weight" and it was introduced in the modified version besides c_1 (social coefficient accelerator) and c_2 (cognitive coefficient accelerator) for controlling the behavior of the particles and balancing the interplay between exploration and exploitation.

PSO is drawn from a simple nature inspired concept with affordable implementation and computational efficiency, even though, the algorithm tends to be trapped into local minima after few iterations. Thus, a great amount of research has been attempting to tackle the problem. For example, authors proposed a PSO algorithm with an adaptive learning strategy (PSO-ALS) that groups a swarm into several sub-swarms to maintain population diversity [218]. Other authors attempted to tune the inertia-related parameters of the PSO dynamics in order to hinder early convergence to local minima [60]. The inertia related weights were also the primary focus of the works [108, 9, 131, 113].

2.1.3 Human Action Recognition

Human Action Recognition plays a vital role for distinguishing a particular behavior of interest in the video. It has critical applications including visual surveillance for detection of suspicious human activities to prevent the fatal accidents [167, 107], automation-based driving to sense and predict human behavior for safe navigation [144, 201]. In addition, there are large amount of non-trivial applications such as human-machine interaction [141, 140], video retrieval [222], crowd scene analysis [32] and identity recognition [134].

In the early days, the majority of research in HAR conducted using hand-crafted methods [189, 190, 52]. However, as deep learning technology evolved and gained increasing recognition in the research community, a multitude of new techniques have been proposed, achieving remarkable results.

Action recognition preserves a similar property of image recognition since both of the fields handle visual contents. In addition, action recognition classifies not only still images but also dynamics temporal information from the sequence of images. Built on these intrinsic characteristics, action recognition's methods can be grouped into two main approaches namely recurrent neural networks (RNN) based approach and 3-D ConvNet based approach. Besides of the main ones, there are other methods that utilize the content from both spatial and temporal and coined the name two-stream 2-D ConvNet based approach [159].

Initially, action recognition was viewed as a natural extension of image recognition, and spatial features from still frames could be extracted using ConvNet, which is one of the most efficient techniques in the image recognition field. However, traditional ConvNets are only capable of processing a single 2-D image at a time. To expand to multiple 2-D images, the neural network architecture needs to be re-designed, including adding an extra dimension to operations such as convolution and pooling to accommodate 3-D images. Examples of such techniques include C3D [177], I3D [15], R3D [62], S3D [197], T3D [37], LTC [182], among others

Similarly, since a video primarily consists of a temporal sequence, techniques for sequential data, such as Recurrent Neural Networks and specifically Long Short Term Memory, can be utilized to analyze the temporal information. Despite the larger size of images, feature extraction is often employed. Long-term Recurrent Convolutional Networks (LRCN)[38] and Beyond-Short-Snippets[208] were among the first attempts to extract feature maps from 2-D ConvNets and integrate them with LSTMs to make video predictions. Other works have adopted bi-directional LSTMs [179, 64], which are composed of two separate LSTMs, to explore both forward and backward temporal information.

To further improve performance, other researchers argue that videos usually contain repetitive frames or even hard-to-classify ones which makes the computation expensive. By selecting relevant frames, it can help to improve action recognition performance both in terms of efficiency and accuracy [54]. A similar concept based on attention mechanisms is the main focus in recent researches to boost overall performance of the ConvNet-LSTM frameworks [47, 194].

While RNNs are superior in the field, they process data sequentially, meaning that information flows from one state to the next, hindering the ability to speed up training in parallel and causing the architectures to become larger in size. These issues limit the application of RNNs to longer sequences. In light of these challenges, a new approach, the Transformer, emerged [184, 176, 41, 7, 118].

2.2 Related Works

2.2.1 Hybrid PSO-ConvNets

At present, several research studies have been applied to optimizing hyper-parameters of ConvNets. Some works aim at a narrower sense where the hyper-parameters are manually set based on trial-and-error experiments. Others follow a broader sense in which the learning rate and the structure of the layers can be automatically generated. For example, in [146], starting from a simple neural network of one layer, the model evolves into full architectures that are competitive with the state-of-the-art counterparts. Further examples are: i) Evolving Deep CNN where the layers are optimized via a genetic algorithm [168]; ii) genetic programming to optimize the architecture of the ConvNet for image recognition [166]. Therefore, instead of an arbitrary manual design, EC algorithms have shown their potential in drawing optimal architectures with global optimization search capability.

However, training a ConvNet is computationally expensive, for instance, models in [146, 145] critically resort to a cluster of hundreds or even thousands of GPUs which are hardly affordable for most of research centres. In this sense, PSO is a better choice as the algorithm requires much less computation. In contrast to EC methods which evolve via competition, in PSO, particles cooperate to share information, e.g., best position, current location and direction. As demonstrated in [178], the fusion of Modified Particle Swarm Optimization (ModPSO) together with Back-propagation (BP) and ConvNet was proposed. The method offers strategies to adjust inertia weight, accelerator parameters, update velocity. The involvement of ModPSO and BP improves training ConvNet by avoiding early convergence and local minima. In addition, the dynamic and adaptive parameters strike a trade-off between the global and local search ability, and the diversity of the swarm.

Other works [193, 33] utilize PSOs to search for hyper-parameters of ConvNets. For example, a novel variant of the PSO algorithm (cPSO-CNN) was proposed in [193] to optimize the kernel size, filter number, stride and padding. The method improved the PSO's exploration ability by applying confidence function defined by a normal compound distribution and reformulated PSO's scalar acceleration coefficients as vectors for better adapting to the variant ranges of the hyper-parameters. In addition, a linear prediction model predicted the ranking of the PSO particles for fast fitness evaluation.

2.2.2 Learning Rate Tuning

When training ConvNets, learning rate might be the most essential hyperparameter as emphasized by Yoshua Bengio in the practical book "Neural networks: Tricks of the trade" [11]. Take into account the kernel size, for example, a subtle image may be convoluted better with a small kernel size, but a larger kernel size would scan just a little worse than its counterpart [175]. The larger the number of filters, the better the emergent patterns, though, fewer filters should not cause a significant lost in ConvNet's performance. Setting a learning rate is totally different as a small/large learning rate could compromise convergence in training.

The main objective of learning rate is tuning to find global minima, local minima, or generally an area where loss function obtains adequately low values (ideally the cost reaches zero $L_{(z,\theta)} \to 0$). Tremendous efforts have been made to reduce execution time to yield better performance.

The goal of learning rate schedules is to regulate the learning rate following a prefixed schedule, e.g., time-based decay, step decay and exponential decay. Likewise, adaptive learning rate methods ease this burden by providing automated tuning. AdaGrad [43], for example, is one of the pioneer adaptive schemes which performs learning rate estimation from the gradients. Other methods are derived from AdaGrad such as AdaDelta [210], AdaSecant [59], RMSprop [174] and Adam optimizers [91].

Cyclical Learning Rate (CLR) addresses an issue in training neural networks, i.e., the need to search for the optimal initial rate and subsequent scheduling. The method allows the learning rate to repeatedly swing between boundary limits according to triangle policy that offers more choices in selection of the learning rate. In addition, CLR enhances classification accuracy in a shorter training [161].

Warmup technique was proposed in early works like [184], where training utilizes a scheme of starting with a small learning rate and gradually ramping up to a larger value during the number of iterations is much less than the whole length of training ($warmup\_iterations \ll epochs$). The method is built on a theory where the ratio of the learning rate and the batch size affects the dynamics of training. Remark that when training large datasets, we can increase the batch size in order to reduce the training time. However, increasing batch size causes more loss and the benefit of shorter training gained is not proportional to the drawback of increasing loss. For this reason, warmup techniques are relevant in training ConvNets [184, 55, 53, 114].

2.2.3 Video Action Recognition

In recent years, Deep Learning has greatly succeed in computer vision fields, e.g., object detection, image classification and action recognition [78, 54, 7]. One consequence of this success has been a sharp increase in the number of investments in searching for good neural network architectures. An emerging promising approach is changing from the manual design to automatic Neural Architecture Search (NAS). As an essential part of automated machine learning, NAS automatically generates neural networks which have led to state-of-the-art results [146, 127, 132]. Among various approaches for NAS already present in the literature, evolutionary search stands out as one of the most remarkable methods. For example, beginning with just one layer of neural network, the model develops into a competitive architecture that outperforms contemporary counterparts [146]. As a result, the efficacy of the their proposed classification system for HAR on UCF-50 dataset was demonstrated [78] by initializing the weights of a ConvNet classifier based on solutions generated from Genetic Algorithms (GAs).

In addition to GAs, PSO - a population-based stochastic search method influenced by the social behavior of flocking birds and schooling fish - has proven to be an efficient technique for feature selection [87, 157]. A novel approach that combines a ModPSO with BP was put forth for image recognition, by adjusting the inertia weight, acceleration parameters, and velocity [178]. This fusion allows for dynamic and adaptive tuning of the parameters between global and local search capability, and promotes diversity within the swarm. In catfish particle swarm optimization, the particle with the worst fitness is introduced into the search space when the fitness of the global best particle has not improved after a number of consecutive iterations [24]. Moreover, a PSO based multi-objective for discriminative feature selection was introduced to enhance classification problems [198].

There have been several efforts to apply swarm intelligence to action recognition from video. One such approach employs a combination of binary histogram, Harris corner points, and wavelet coefficients as features extracted from the spatiotemporal volume of the video sequence [216]. To minimize computational complexity, the feature space is reduced through the use of PSO with a multi-objective fitness function.

Furthermore, another approach combining DL and swarm intelligence-based metaheuristics for HAR was proposed [10]. Here, four different types of features extracted from skeletal data - Distance, Distance Velocity, Angle, and Angle Velocity - are optimized using the nature-inspired Ant Lion Optimizer metaheuristic to eliminate non-informative or misleading features and decrease the size of the feature set. Despite these advances, the field remains largely uncharted, especially with respect to recent and emerging techniques.

2.3 Conclusions

General concepts and a review of the current state of image classification and action recognition were presented in this chapter. We grouped essential theories of Convolution Neural Network, Particle Swarm Optimization and Human Action Recognition in the background Sections since ConvNet is the most efficient and effective method for image classification as the technique outperforms other shadow methods recently and PSO will be used to leverage training of ConvNet. As ConvNet and PSO will be trained in an intertwined manner, we discussed relevant works regarding hybrid PSO-ConvNet. In addition, we also examined methods for Learning Rate Turning because it is one of essential parts of our method. Moreover, recent works in Video Action Recognition are critically analyzed.

Chapter 3

Advanced Capsule Networks

3.1 Introduction

Capsule Networks arrive in the field of Deep Learning at the time when many issues are considered solved e.g. in image and object recognition, very deep networks with hundreds of layers are able to outperform human. One reason behind the success of DL is the Max Pooling (MP) layer which not only reduces dimension of images but also selects most critical pixels for routing from one layer to another. Though MP works very well, Hinton argues that MP causes loss of useful information. As a consequence, this layer is replaced with a routing algorithm and a new architecture namely Capsule Networks is designed [150]. The CN can also be referred as Vector CN since the approach is based on agreements between vectors. The design achieved 0.25% test error on Modified National Institute of Standards and Technology (MNIST) dataset in comparison with the state of the art using DropConnect 0.39% without data augmentation [187]. The other CN applying Expectation Maximization (Matrix CN) reduces the best error rate by 45% on SmallNORB dataset [68]. However, the networks still face challenges on other datasets e.g. Cifar-10, SVHN and ImageNet.

In this research, we choose to build our CNs based on the Vector CN architecture since there are few implementations in literature for Matrix CNs and our preliminary results show that the latter takes longer convergence time in the ASL alphabet dataset.

For DL architecture, we choose several prestige Transfer Learning models including Inception V3 [171], DenseNet V201 [74], NASNet [224], MobileNet V1 [72] and MobileNet V2 [153] to generate feature maps and compare these models to explore which model is the best in our setting. As TL is a very fast growing field, the models are trained on a wide variety of platforms such as Tensorflow, Caffe, Torch and Theano. In addition, just a year ago, Keras which was one of the biggest independent platforms had been integrated into Tensorflow. As Keras includes pre-trained models for most of leading TLs, we prefer to use this platform to set a unified environment for comparison of all TL models. We also classify models in two groups one that is mainly used for demanding computers (Inception V3, DenseNet V201) and the other for smaller devices (NASNet, MobileNet V1 and V2) since mobiles have become a crucial tool in our daily life.

We perform experiments on static signs of American Sign Language dataset. In ASL, there are two distinct signs namely dynamic signs and static signs. Our aim is to build an action recognition framework to recognize signs in continuous frames (e.g. transcription generators for ASL songs or conversations). In this work, we focus on ASL alphabet signs. For this problem, models are usually based on ConvNets [4, 12], other Machine Learning (ML) techniques like Multilayer Random Forest [98] or TL models such as GoogLeNet and AlexNet [84, 46].

The rest of this article is structured as follows. We highlight our main contributions in Section 3.2. Next, we describe an ASL dataset for this research in Section 3.3.1. Then we discuss about Vector CNs and TL models' architectures in Section 3.3.2 and Section 3.3.3, accordingly. Experiments and respective results are analyzed and discussed in Section 3.4. We conclude this work in Section 3.5.

3.2 Contributions

In our research, we improve the design of the Vector CNs and perform empirical comparisons versus DL models on accuracy and speed using an ASL fingerspelling alphabet.

First, we propose to modify the Vector CN's architecture to find the most efficient designs. Namely, we extended convolution layers to better filter Input images and varied FC layers in the Reconstruction to leverage image restoration.

Second, we explore distinct DL models for demanding devices (Inception V3 and DenseNet V201) and small devices (NasNet and MobileNets) when integrated with MLP and LSTM. We use the former setting as a baseline to compare with the latter.

Third, Vector CNs are analyzed against DL models. We find out that CNs perform comparatively on both accuracy and speed. In addition, CNs have an advantage as pre-training is not required. Capsules can be trained within an hour compare to days or more for pre-trained DL models.

Finally, we make a demonstration to compare CNs and DLs on videos for teaching ASL alphabets. The results are very promising as our models can recognize almost of all signs without previously seen.

3.3 Proposed Methods

In this section, we first discuss about an ASL dataset that will be used in our experiments. Then we deal with architectures of Vector CNs and DL models.

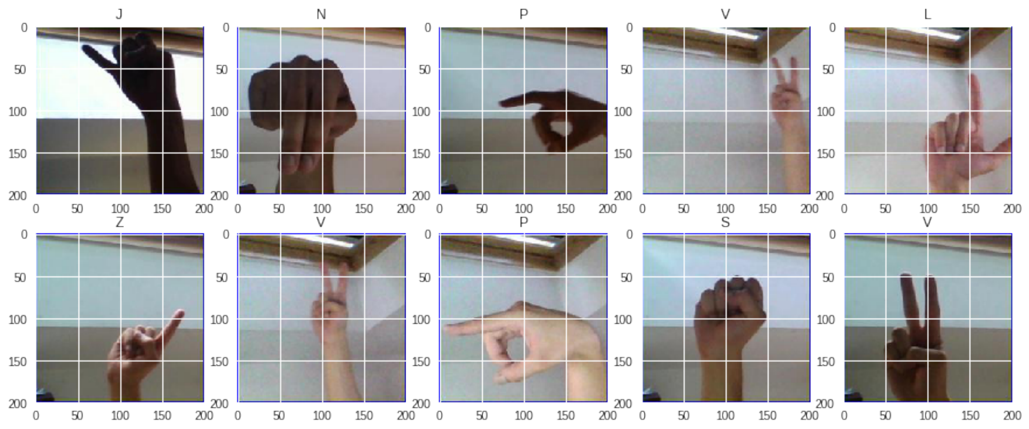


Figure 3.1: Random Samples from ASL Dataset

3.3.1 ASL Dataset

One of our ultimate goals is to build an ASL translator that is capable of classifying alphabet signs from language training videos. In these videos, professional trainers illustrate hand shapes for signs from A to Z. As they perform demonstrations, the hand is moving around the screen from left to right, up to down and vice versa. To train our models, we search for a dataset that captures similar movements with a large sample size. We expect that each sign should have thousands of instances since a standard MNIST has 60000 samples for 10 classes. With these constraints and since ASL datasets are relatively fewer than for English alphabet, we found only one dataset from Kaggle website[1] that meets our requirements. This set of data includes all 26 signs including dynamic signs “J” and “Z” with 3 additional signs. Each sign includes 3000 samples (200×200 pixels), totally 87000 for all signs. In these images, hands are placed in distinct positions on the screen, distance and lightning are varied.

Figure 3.1 shows 10 random samples from the dataset. We notice that signs “N” and “P” have different shapes compared to the target video. In stead of replacing them, these signs are retained for the reason mentioned above.

The data are selected randomly and split into training and testing sets with the ratio of 70/30. In addition, the value range is re-scaled from $[0, 255]$ to $[0, 1]$. For data augmentation, the rotation, shear, width shift and height shift are set in the ranges of 20, 0.2, 0.2 and 0.2 respectively. Moreover, images’ brightness and contrast are spanned using random uniform within distances of 0.6 and 1.5.

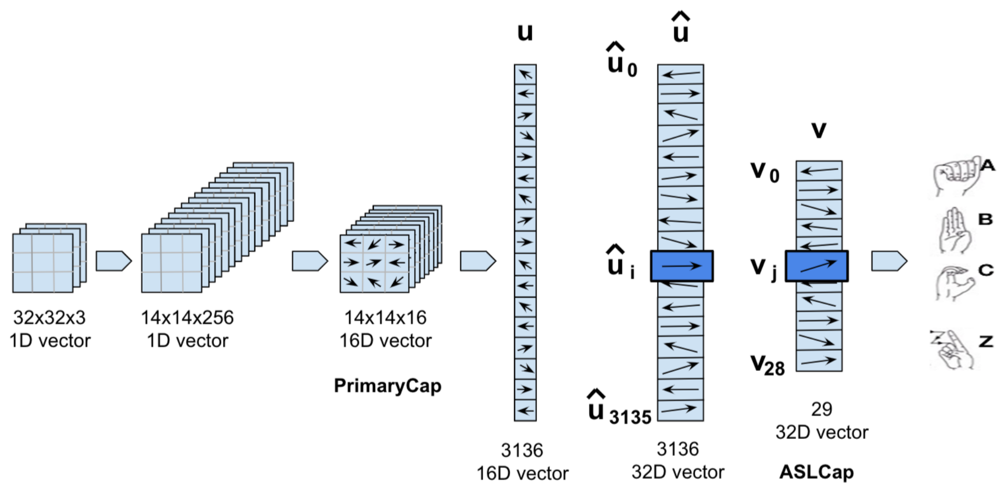


Figure 3.2: Vector Capsule Networks Architecture for ASL

3.3.2 Capsule Networks

Hinton describes a CNs as a group of neurons that represent distinct properties of the same entity. In Vector CN, a capsule in one layer sends its activity to the capsule in above layer and the method checks which one agrees the most. The essential structure of CNs is shown in the Figure 3.2.

The initial step of Vector CNs is similar to ConvNets where Input images are filtered to detect features such as edge and curve. Then, in PrimaryCaps, the generated features are grouped to create multi-dimension vectors. Illustrated in the Figure, 256 feature maps of size 14×14 are transformed into 16 capsules each contains 14×14 vectors of 16 dimensions. Routing from this layer to the ASL Capsule (ASLCap) layer is computed as follows.

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||}, s_j = \sum_i c_{ij} \hat{u}_{j|i} \tag{3.1}$$

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})}, \hat{u}_{j|i} = W_{ij} u_i \tag{3.2}$$

$$b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} v_j \tag{3.3}$$

where v_j represents the vector output of capsule j in ASLCap and s_j is its total input produced by a weighted sum of all predictions from layer below. Next, the signal is passed through a squash non-linearity so that the value is in the range of $[0, 1]$. The length of this vector suggests the probability an entity (represented by the capsule) being detected. For example, if the vector v_j is set to represent

[1]https://www.kaggle.com/grassknoted/asl-alphabet

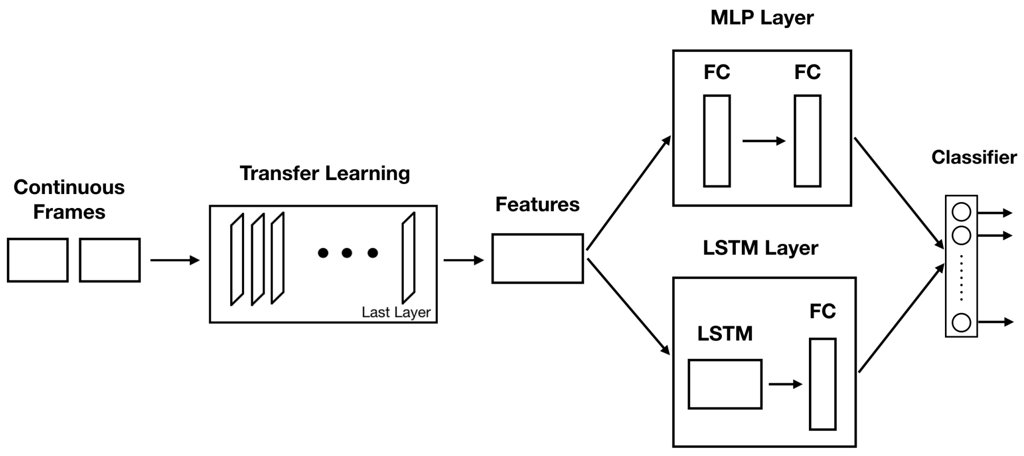


Figure 3.3: Transfer Learning Architecture for ASL

the sign “L” then the length of this vector indicates if the sign is actually presented. The output of vector u_i is transformed to the vector $\hat{u}_{j|i}$ by multiplying with a weight matrix W_{ij}. The routing coefficient b_{ij} will be increased or decreased based on whether the output v_j has a similar direction with its prediction $\hat{u}_{j|i}$.

3.3.3 Deep Learning Models

This section deals with the integration of TL models in DL architecture for ASL recognition to classify alphabet signs in continuous frames. ConvNets were introduced 20 years ago with the notable architecture from LeNet [104]. Throughout the time, ConvNets tended to go deeper e.g. VGG (19 layers). With recent advances in computer hardware, very deep architectures such as Highway Networks [165] and Residual Networks [65] have exceeded 100 layers. Training these models can take days or even months; this gives rise to TL where models are pre-trained on a dataset and re-used on others. In Computer Vision, earlier convolution layers are considered to behave similarly to edge and curve filters. Thus, these frozen weights can be used on our ASL dataset. A modification in the last layer is necessary as the number of signs is 29 compared to 1000 categories in ImageNet.

In our research, we include Inception V3, DenseNet V201, NASNet, MobileNet V1 and MobileNet V2 models in DL Architectures. Inception V1 (or often called GoogLeNet to honor LeCun’s Networks) was a winner for image classification challenge in ILSVRC 2014 and achieved top-5 error of 6.67% [170]. The results for Inception V3 [171] and DenseNet V201 [74] were 5.6% and 6.34%/5.54% (on single-crop/10-crop) respectively.

NASNet was recently introduced and obtained state-of-the-art results on several datasets including Cifar-10, ImageNet and COCO [224]. NASNet comes with two versions one for computers (NASNetLarge) and the other for small devices (NASNetMobile). Since the computation for NASNetLarge takes twice the required time for the slowest models DenseNet V201, we select only NASNetMobile in our experiments. Despite of being made for small devices, the model accomplishes 8.4% top-5 error on ImageNet. Additionally, we choose MobileNets[72, 153] for comparisons within mobile platform.

As shown in Figure 3.3, ASL's signs are illustrated in a video and extracted as a sequence of frames (Please see our demonstration for more information). At the runtime, an active model with trained weights is loaded. Based on each TLs model, the extracted features have length variations i.e. 2048, 1920, 1056, 1024 and 1280 for Inception V3, DenseNet V201, NASNetMobile, MobileNet V1 and V2.

After this step, the flow goes through either MLP Layer or LSTM Layers. The former comprises two FCs whereas the latter contains one LSTM and one FC. The LSTM has 2048 units with history's lookback of one for classifying one frame per time. Besides, all FCs are composed by 512 neurons. We use MLP in our DL models as a baseline for comparison with Vector CNs. This is also interesting to see performance of DL built on other techniques like LSTM.

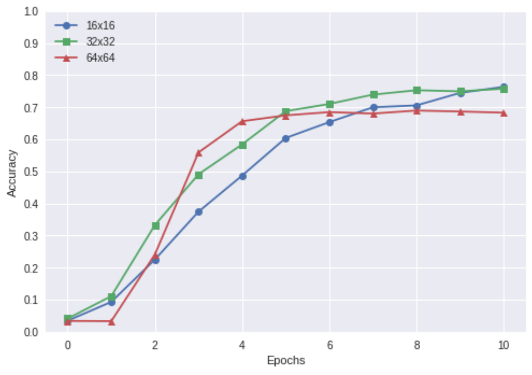
3.4 Experiments and Results

In this section, we first discuss the performances of Vector CNs and DL models on variations of Input image size and number of samples. Later, we take two models in DL and compare with two models in Capsule.

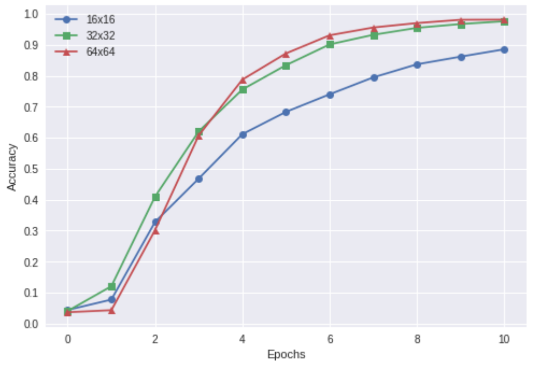
3.4.1 Experiment 1: Effective of Dataset Size and Image Size on Capsule Accuracy

To perform this experiment, we scale the Input images to 64×64, 32×32 and 16×16 pixels and use datasets of $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, and a full size. We vary the size of the dataset so that we can observe the effects when the number of samples is small. We exclude dataset of $\frac{1}{32}$ since this yields fewer than 100 samples per class which may not be enough for recognition. The dataset is split into train and test with the ratio of 70/30. We also use two FCs for Reconstruction, one FC has a half of Input image size and the other has an equal size e.g. when the Input image is 64×64, the two FCs have sizes of 1024 (32×32) and 4096 (64×64), accordingly.

Figure 3.4a shows a result of this experiment using $\frac{1}{16}$ dataset. We notice that the accuracy for 64×64 resolution images are lower than for 32×32 and 16×16. This contradicts our expectation since with a larger resolution, the image is clearer and should be recognized better. From Figure



(a) $\frac{1}{16}$ Dataset



(b) All Dataset

Figure 3.4: Comparisons of Capsule Networks on Image Sizes

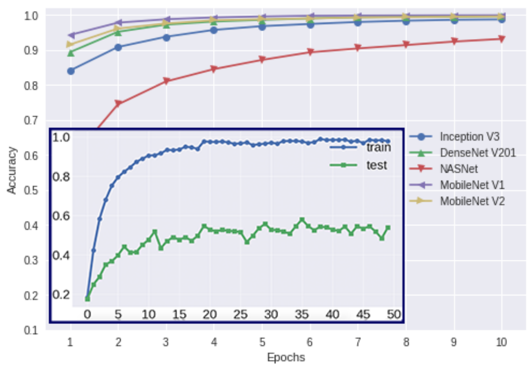
3.4b, we see a different approach where 64×64 resolution images perform better than the others. We exclude results for remaining sets to save space.

In summary, these experiments show that a larger image's resolution does not always yield a better accuracy because a small number of samples may affect this performance. With a larger dataset, a higher resolution generally results in a better accuracy. In addition, we can observe that both 64×64 and 32×32 image resolutions achieve an accuracy of approximately 0.99 after 10 epochs using full dataset.

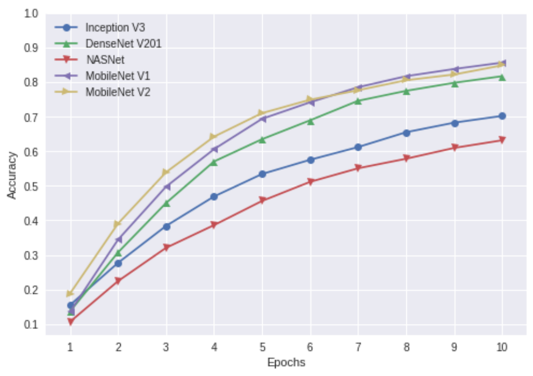
3.4.2 Experiment 2: Comparisons of Deep Learning Models

In this experiment, we use the same variations of dataset size as in Vector CN models and compare all DL models on MLP and LSTM. Figure 3.5a shows accuracy of all models with MLP and results from a typical ConvNet using full dataset. We can observe that, despite of being simple, the accuracy approaches 100% on training set. However, this yields only near 60% on test.

Much to our surprise, two versions of MobileNets both outperform Inception V3 and DenseNet 201. This maybe caused by the efficiency of Depthwise Separable Convolution layers. It also can be seen that DenseNet V201 performs better than Inception V3. Perhaps, the model's blocks in which one layer is connected to all other layers helps to reserve more important information. Additionally, a similar trend can be seen using $\frac{1}{16}$ dataset with accuracy be offset approximately $10\% - 20\%$. Moreover, we can observe analogous results using LSTM in Figure 3.6. Overall, DL models classified with LSTM show faster convergences.



(a) All Dataset



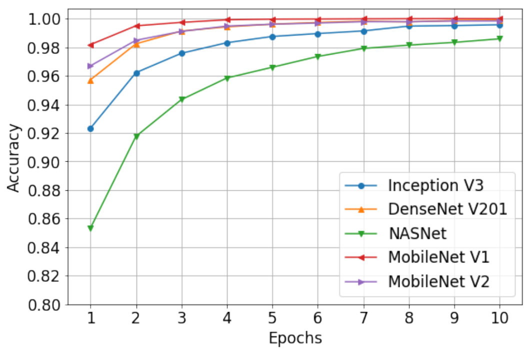
(b) $\frac{1}{16}$ Dataset

Figure 3.5: Comparisons of Deep Learning Models on MLP. The Figure on the left hand side also includes a result of training and testing on a typical ConvNet (two Convolution layers each with 32 filters, one FC of 128 neurons, Adam optimizer and 50 epochs)

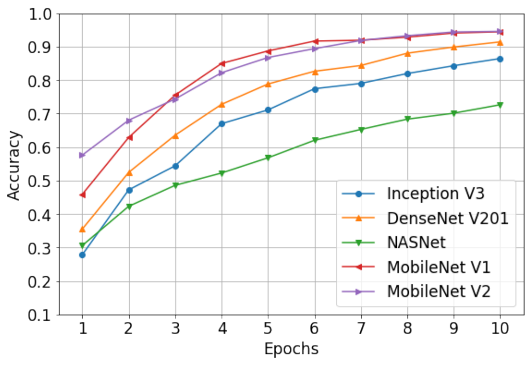
3.4.3 Experiment 3: Comparisons of Vector Capsule Networks and Deep Learning Models on Speed

In this experiment, we perform speed tests of Vector CNs and DL models on running times for Feature Extraction and Prediction. Regarding the first metric, all models are executed on GPUs of Tesla K80 and P100. Capsules are excluded since extraction of features is not essential.

We observed that with only one ConvNet, the model performed very well on images 64×64 but poorly on the target video. This maybe caused by the model learns very well on the training data



(a) All Dataset



(b) 1/16 Dataset

Figure 3.6: Comparisons of Deep Learning Models on LSTM

but distinct backgrounds between the two hamper its performance. For this reason, we add one more convolution layer without pooling after Input image to filter better the backgrounds. We did not add more ConvNets since we experienced the vanishing gradient problem with accuracy becoming zeros even after a long training.

The structure of Vector CNs are changed as follows. For Capsule with Input image size 32×32 (aka Capsule 32 V1), we use one convolution layer after the input layer and four FCs in Reconstruction with sizes of 4×4, 8×8, 16×16 and 32×32. For Capsule with Input image sizes of 64×64, we add two convolution layers where the first layer is followed by a pooling with size 2×2 and the second convolution layer is without a pooling. We call this is Capsule 32 V2. In Reconstruction, we design four FCs with sizes 8×8, 16×16, 32×32 and 64×64.

Table 3.1: Comparisons of Vector CNs and DL Models on Speed

		Metrics		
	Models	**Feature Extraction (Tesla K80)**	**Feature Extraction (Tesla P100)**	**Prediction (CPU 2.7GHz)**
Time (s)	Inception V3	3453	1910	0.248
	DenseNet V201	6250	3199	-
	NASNet	3018	2592	-
	MobileNet V1	870	507	0.070
	MobileNet V2	1175	758	-
	Capsule 32 V1	-	-	0.069
	Capsule 32 V2	-	-	0.086

It can be gleaned from Table 3.1 that MobileNet V1 runs faster than any other models on both GPUs whereas DenseNet V201 is the slowest. This clearly shows the advantage of Depthwise Separable Convolution in reducing computation. As we expected, all models for mobiles perform faster than models for computers with an exception that NASNet is slower than Inception V3 on Tesla P100.

On prediction speed comparison, we pickup two models from DL. We select MobileNet V1 because of its best accuracy and fastest speed. Besides, Inception V3 is chosen instead of DenseNet V201 based on a faster speed. We perform this experiment on a computer with 4-CPUs of 2.7GHz. It is unexpected that Capsule 32 V1 is only slightly faster than MobileNet V1 since the model has only 2 CN layers (and 3 ConvNets) compared to 28 ConvNets of its counterpart. It is also noticed that Capsule V2 runs three times faster than Inception 32 V3 but quite slower than MobileNet V1.

3.4.4 Experiment 4: Comparison of Vector Capsule Networks and Deep Learning Models on Accuracy

This experiment is designed to compare accuracy of Capsules and DL architectures. We select MobileNet V1, Inception V3, Capsule 32 V1 and Capsule 32 V2 as discussed in the previous section.

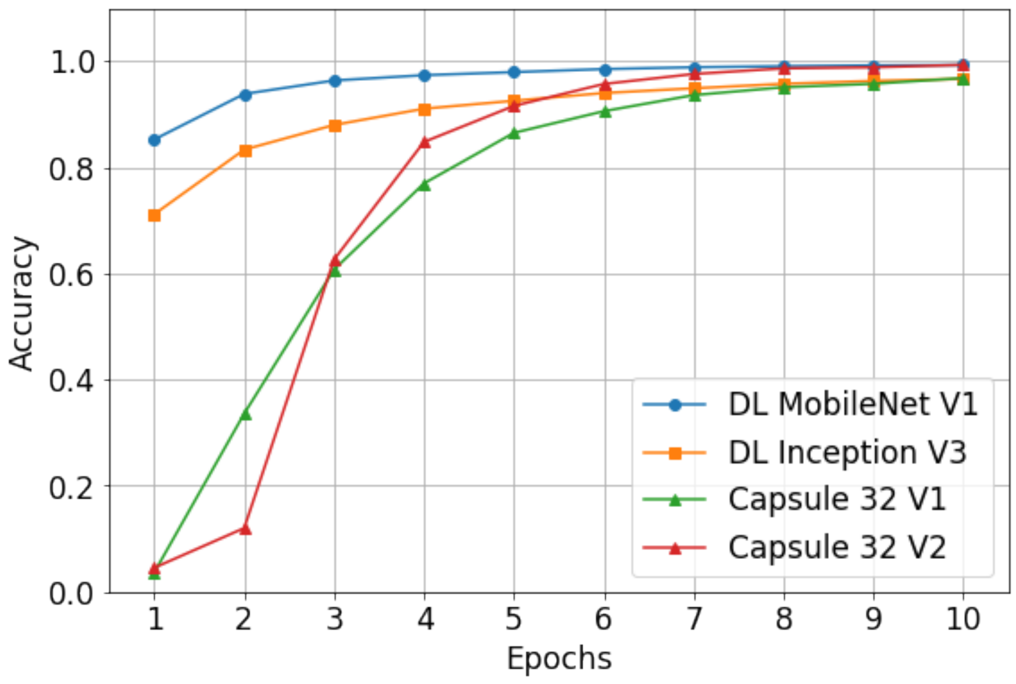


Figure 3.7: Capsule Networks vs Deep Learning Models on Accuracy

As we can see from Figure 3.7, MobileNet V1 and Inception V3 perform better than Capsules in first few epochs. However, as number of epochs increases, Capsule 32 V1 achieves a similar accuracy of Inception V3 and Capsule 32 V2 approaches the accuracy of MobileNet V1.

3.5 Conclusions

In this research, we propose to improve the design of Vector CN by extending ConvNet layers after Input image and vary the number of FCs in Reconstruction to accomplish better accuracy. Having a larger image size is critical in our approach since certain signs are similar. A small change in the position of fingers can yield distinct signs. In addition, variation of background contexts may hamper Capsules' performance. Using our approach we attain the goals as follows. First, our results show that Capsule 32 V2 performs comparatively to DL MobileNet V1 on accuracy and Capsule 32 V1 runs a slightly faster than its counterpart. Second, Vector CNs are greater than DL models in the sense that the latter requires pre-trained weights which can take days or even months for training whereas the former can be trained on-the-fly within an hour. Third, as a result of our exploration, we found that MobileNet V1 even though was mainly built for small devices but is superior to all other DL models both on accuracy and speed in this dataset.

Furthermore, we made a demonstration to illustrate our approach where we compare Vector CN V1 with DL MobileNet V1 in an ASL video. The recorded file can be accessed via the link[2]. Although, Vector CN can recognize most of all signs (excluding signs that we mentioned earlier), the model is more sensitive to changes than the DL model. This suggests pooling in the DL may better than just rescaling as in the Vector CN. In addition, the DL performs better although the accuracy (when testing on the ASL dataset) is similar to that of Vector CN. This can indicate that the DL model is more generative. For this reason, additional samples are needed to improve the overall performance.

Although we demonstrate this approach in the context of ASL alphabet signs, the approach has broader applications to any video recognition tasks where each individual frame's information are crucial.

In the future, we plan to redesign CN' structure to perform on larger images. We also plan to build Vector CNs and DL MobileNet V1 on mobile devices as the accuracy and speed allow these networks to run in realtime.

[2]http://bit.ly/2O4sJSU

Chapter 4

Depthwise Separable Convolution Capsule Networks

4.1 Introduction

In our previous research [130], we performed experiments to compare accuracy and speed of Capsule Networks versus Deep Learning models. We found that even though CNs have a fewer number of layers than the best DL model using MobileNet V1 [72], the network performs just slightly faster than its counterpart.

We first explored details of MobileNet V1's architecture and observed that the model utilizes Depthwise Separable Convolution for the speed improvement. The layer comprises of a Depthwise Convolution (DC) and Pointwise Convolution (PC) which sufficiently reduces the model size and computation. Since CNs also integrate a convolution layer in its architecture, we propose to replace this layer with the faster convolution.

At the time this article is being written, there are 439 articles citing the original CNs paper [150]. Among these articles, a couple of works attempt to improve speed and accuracy of CNs. For example, the authors [8] propose Spectral Capsule Networks which is composed by a voting mechanism based on the alignment of extracted features into a one-dimensional vector space. In addition, other authors claim that by using a Convolutional Decoder in the Reconstruction layer could decrease the restoration error and increase the classification accuracy [125].

CNs illustrated its effectiveness on MNIST dataset, though much variations of background to models e.g. in CIFAR-10 probably causes the poorer performance [150]. To solve this problem, we argue that primary filters to eliminate such backgrounds should be as important as other parts of CNs' architecture. Just, improvements on speed and accuracy of these Convolution layers can be bonus points for the network.

After a thorough search of relevant literature, we believe that this is the first work on the integration of DW into CNs.

The rest of this article is organized as follows. In Section 4.2, we highlight our main contributions. Next, we analyse the proposed design in Section 4.3.1. Following, the design of DL models for the purpose of comparison is illustrated in Section 4.3.2. Experiments and results are discussed in Section 4.4 accordingly. Finally, we conclude this work in Section 4.5.

4.2 Contributions

Our main contributions are twofold: first, on the replacement of Standard Convolution with Depthwise Separable Convolution in CNs' architecture and, second, on the empirical evaluations of the proposed Capsule Network against Deep Learning models.

Regarding the design of CNs, we found that the integration of DW can significantly reduce the model size, increase stability and yield higher accuracy than its counterpart.

With respect to experimental evaluations of CNs versus DL models, the Capsule models perform competitively both on model size and accuracy.

4.3 Proposed Methods

4.3.1 Integration of Depthwise Separable Convolution and Capsule Networks

As mentioned in the previous Section, we propose to substitute a SC of CNs with a DW. Figure 4.1 illustrates the architecture of the proposed model. Additionally, we apply this architecture for an ASL dataset with 29 signs. We discuss about this dataset more details in Section 4.4.1.

We divide this architecture into three main layers including Initial Filter, Depthwise Separable Convolution Layer and Capsules Layer. In principle, we can change the first SC with a DW. However, the Input images have only three depth channels which require fewer computations than in the second layer so that we keep the first Convolution intact. The reduction of computation cost using DW is formulated as follows.

An SC takes an input $D_F \times D_F \times M$ feature map F and generates $D_G \times D_G \times N$ feature map G where D_F is the width and height of the input, M is the number of input channels, D_G is the width and height of the output, and N is the number of output channels.

The SC is equipped with a kernel of size $D_K \times D_K \times M \times N$ where D_K is assumed to be a spacial square. M and N are the number of input and output channels as mentioned above.

The output of the feature map G using an SC with kernel stride is 1 and same padding (or

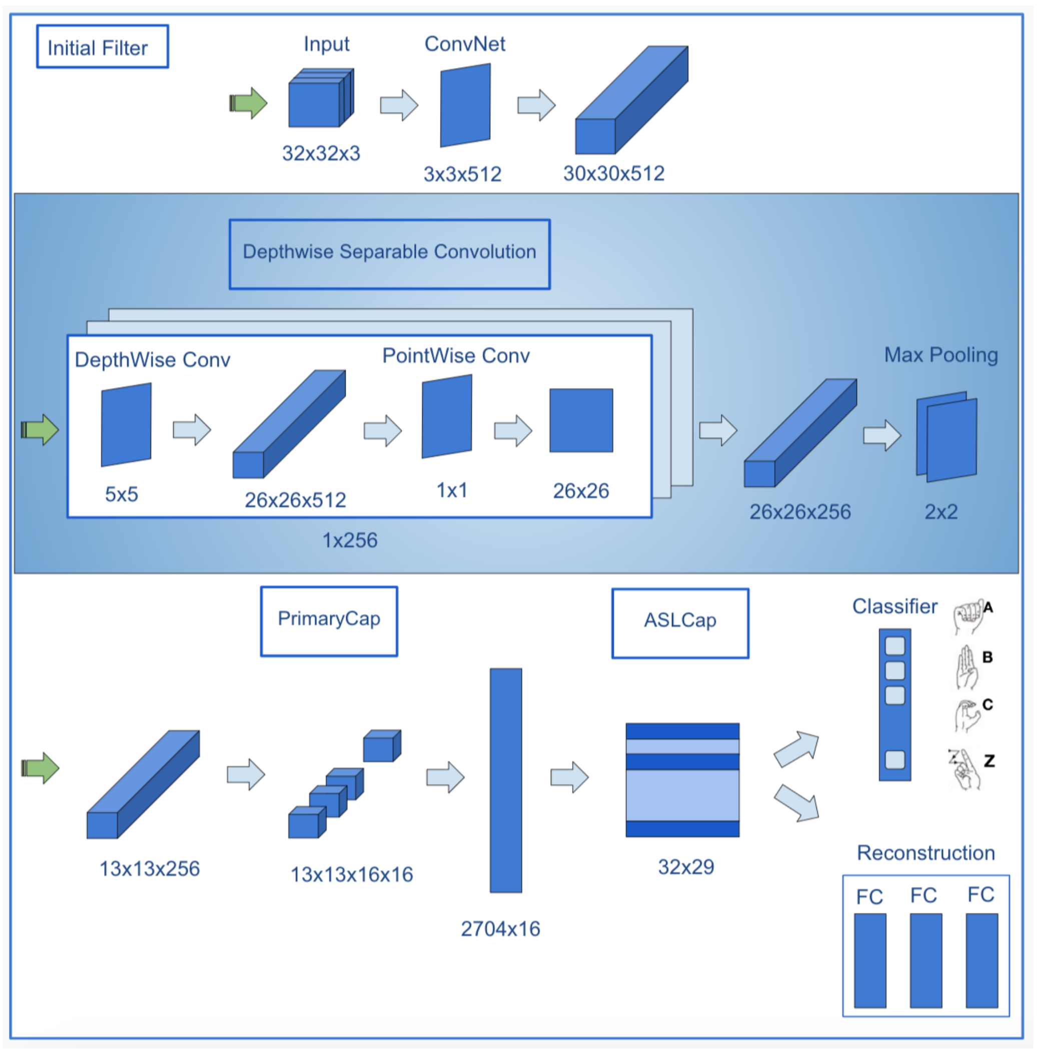


Figure 4.1: Depthwise Separable Convolution Capsule Architecture.

padding in short):

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} \cdot F_{k+i-1,l+j-1,m} \tag{4.1}$$

The computation cost of the SC can be written as:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{4.2}$$

For one channel, the output of feature map Ĝ after Depthwise Convolution:

$$\hat{G}_{k,l,m} = \sum_{i,j,m} \hat{K}_{i,j,m} \cdot F_{k+i-1,l+j-1,m} \tag{4.3}$$

where K̂ is the denotation of a Depthwise Convolution with kernel size $D_K \times D_K \times M$ and this Convolution applies m_{th} filter to m_{th} channel of F yields the m_{th} channel in the output.

The computation cost of the Depthwise Convolution is written as:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{4.4}$$

Pointwise Convolution uses 1×1 Convolution, therefore, the total cost after Pointwise Convolution:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{4.5}$$

The computation cost of Depthwise Separable Convolution after two convolutions is reduced as:

$$\begin{aligned} &\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} \\ &= \frac{1}{N} + \frac{1}{D_K^2} \end{aligned} \tag{4.6}$$

For more details of these computations, please refer to the MobileNet V1 article [150].

In Figure 4.1, we show a sample where colour images have a size of $32 \times 32 \times 3$ and filtered by a ConvNet which is equipped with a 3×3 kernel and 512 filters. If the DW is applied in this layer, then the cost of computation can be reduced between 8 and 9 times. Though the input image has a depth of only 3, the reduction of computation cost in this layer is unmatched with hundreds of channels in the second layer.

4.3.2 Deep Learning Models

For the purpose of comparison, we also briefly discuss the DL's architecture as shown in Figure 4.2. As we can see from the Figure, this architecture comprises of several crucial layers including TL, MLP and LSTM layers. In TL layer, we utilize one of the models including Inception V3 [171], DenseNet V201 [74], NASNetMobile [224], MobileNet V1 [72] and MobileNet V2 [153] to extract features from input ASL signs. We then use MLP Layer as a baseline to compare with LSTM Layer. The MLP Layer includes two FCs each with 512 neurons whereas the LSTM Layer contains one LSTM with 2048 units and one FC. More detail of the architecture can be referred to our earlier work.

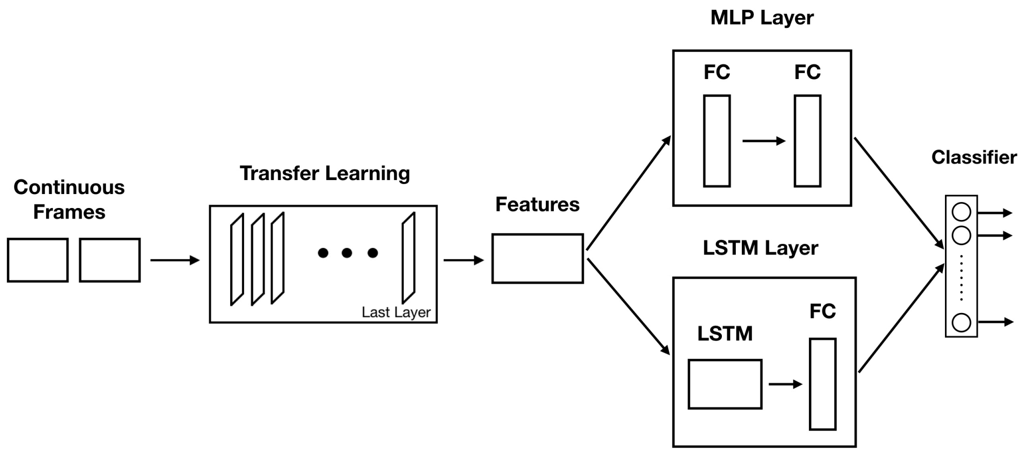


Figure 4.2: Deep Learning Architecture

4.4 Experiments and Results

In this section, we first discuss an ASL Dataset that will be used as our testbed. Then we setup experiments to compare our proposed Capsule's architecture using DW against typical ConvNets. Next, we analyse performances of TL models including Inception V3, DenseNet V201, NASNet, MobileNet V1 and MobileNet V2 using MLP and LSTM. Finally, we pickup the best of CNs and challenge the best of DL models.

4.4.1 ASL Dataset

For the purpose of comparison with our previous work, we use the same ASL dataset for finger-spelling. The dataset was obtained from Kaggle website and includes 26 signs for letters A to Z with 3 additional signs using in other cases. Each sign contains 3000 samples (200×200 pixels), totally 87000 samples for all signs. Figure 4.3 shows 10 random samples from this dataset. In these experiments, we use half of the number of samples since the accuracy are similar to that of using all dataset and this also reduces the training time by half. We divide the data into a train set and a test set with the ratio 70 and 30.

4.4.2 Experiment 1: DW Capsules vs SC Capsules

In this experiment, we perform experimental evaluations of DW Capsules against SC Capsules. First, we vary the size of Convolution's kernel including 9×9, 7×7, 5×5 and 3×3 using the Input image's size of 32×32. Then we add one more ConvNet in the first layer in the CNs's architecture as

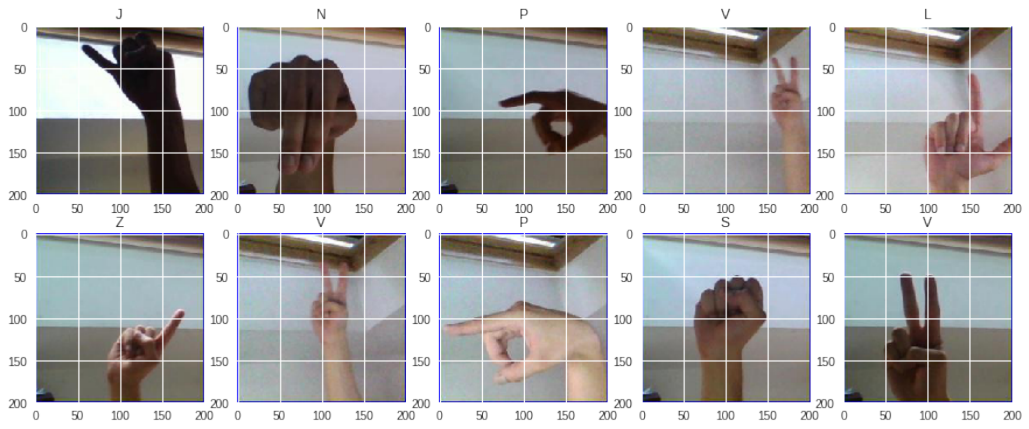


Figure 4.3: Random Samples from the ASL Dataset

shown in Figure 4.1. We also provide two variations of CNs, one for scaling down the total number of model parameters by adding a Max Pooling after the second ConvNet (Mini version) and, another, which increases Input image sizes to 64×64 (Max version).

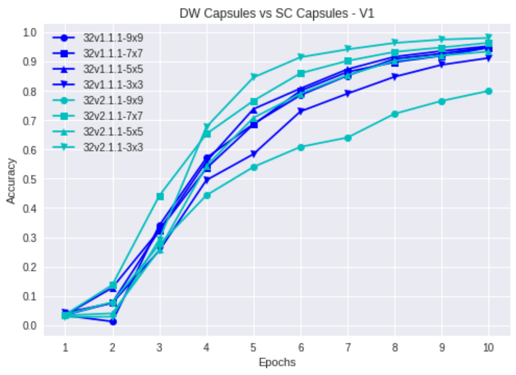
Figure 4.4b and 4.4d show that DW Capsules perform equivalently or even better on accuracy than SC Capsules. Additionally, SC Capsules seems to be unstable and fluctuating based on kernel's size. Figure 4.4a shows a similar trend when SC Capsules are more likely to volatile when a half of SC Capsules are outperformed by DW Capsules. Only in Figure 4.4c, all SC Capsules achieve higher accuracy than that of DW Capsules. Though this can be a trade of between training epoch and training speed.

4.4.3 Experiment 2: Deep Learning Models using MLP and LSTM

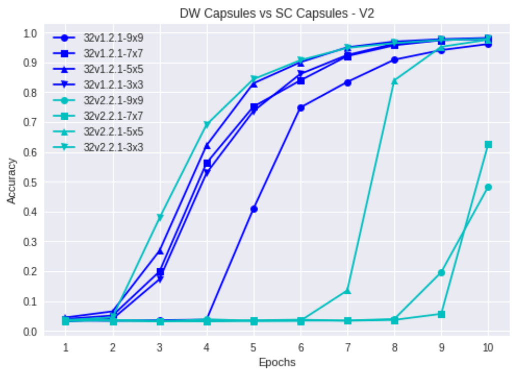
In this section, we perform experiments of DL models using MLP and LSTM on variations of the ASL dataset including $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$ and the whole dataset.

Due to the limited number of pages allowed, we excerpt results from most of all DL models and preserve only one model for mobile devices and one model for powerful computers i.e. MobileNet V1 and Inception V3. We select MobileNet V1 because of its best accuracy and Inception V3 since the model is faster than DenseNet V201.

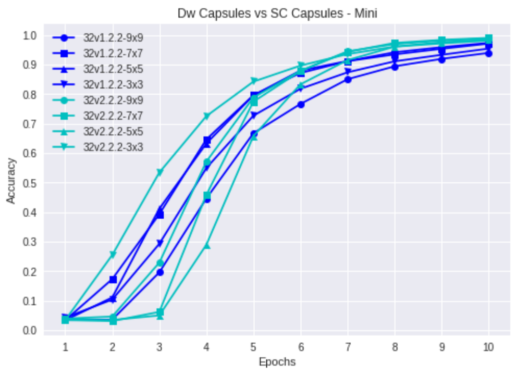
The Figure 4.5a shows that Inception V3 TL model when integrated with LSTM outperforms its version on MLP in all sets of data. Similarly, MobileNet V1 LSTM achieves a higher accuracy than MobileNet V1 MLP. Remarkably, MobileNet V1 performs better than Inception V3 on both MLP and LSTM versions even though the model is mainly built for much smaller devices.



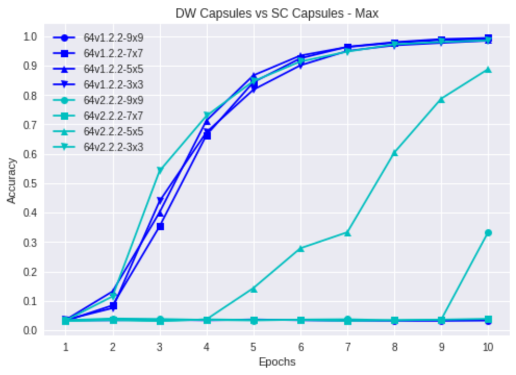
(a) DW vs SC V1



(b) DW vs SC V2



(c) DW vs SC Mini



(d) DW vs SC Max

Figure 4.4: DW Capsules (blue) vs SC Capsules (cyan). The numbers 32 and 64 denote Input image sizes 32×32 and 64×64, respectively; v1 stands for DW Capsule whereas v2 stands for SC Capsule. The next number expresses the amount of ConvNets. The last number denotes whether the ConvNet is followed by a Max Pooling (1 if not, 2 if followed). All ConvNets in the primary layer have the kernel sizes vary from 9×9, 7×7, 5×5 to 3×3.

4.4.4 Capsule Networks vs Deep Learning Models on Model Size and Accuracy

In this Section, we analyse CNs and DL models with respect to the total size of these models as well as accuracy. We select the best DW Capsules and SC Capsules including: 1. Input image size 32×32 pixel followed by two ConvNets and a MP (Figure 4.4c) 2. Input image size 32×32 pixel one for the best accuracy SC (Figure 4.4a) and one for the best accuracy DW (Figure 4.4b) 3. Input image size 64×64 pixel integrated with two ConvNets and one MP (Figure 4.4d). These CNs are denoted as DW Mini, SC Mini, DW, SC, DW Max and SC Max, respectively.

Generally, we can see from Figure 4.6 that DW Capsules drastically decrease the models' size. More specifically, Capsule 32 DW Mini reduces the model size by 21% while Capsule 64 DW Max shrinks 25% of the total parameters. It can be noted that Capsule 32 DW reduces the number of parameters by 40% which is more than Capsule 32 DW Mini and Max due to one more convolution.

In comparison between CNs and DL models, we can observe that MobileNet V1 MLP has a smallest model size followed by Inception V3 MLP. Additionally, Capsule 32 DW and Capsule 64 DW Max have smaller sizes than MobileNet LSTM and Inception V3 LSTM.

In terms of accuracy, MobileNet V1 LSTM outperforms all other models. In spite of that, Capsule 64 DW Max reaches the second position and performs better than MobileNet V1 MLP after 10 epochs. In addition, Capsule 64 DW Max outperforms both versions of Inception V3 LSTM and MLP by a large extent. Though Capsule 64 DW has 40 times larger model size than MobileNet V1 MLP and 20% more than MobileNet V1 LSTM, its total parameter is less than that of Inception V3 LSTM.

It can be noticed that Capsule 32 DW Mini though having the smallest size among all CNs but is outperformed by MobileNet V1 MLP with regards to accuracy. However, when we perform these experiments with more epochs, this gap can be eliminated as Capsule 32 DW Mini yields the accuracy of MobileNet V1 MLP (0.9928) after 17 epochs. A similar trend also occurs to Capsule 32 SC Mini as it reaches the accuracy of MobileNet V1 LSTM (0.9986) after 18 epochs (approximately 1h training on Tesla K80).

4.5 Conclusions

In this research, we first propose to replace Standard Convolution in Capsule Networks' architecture with Depthwise Separable Convolution. Then we perform empirical comparisons of the best CNs with the best DL models.

The results show that our proposed DW Capsules remarkably decrease the size of models. Among the chosen CNs, the total parameters had shrunk roughly by an amount between 21% – 25%.

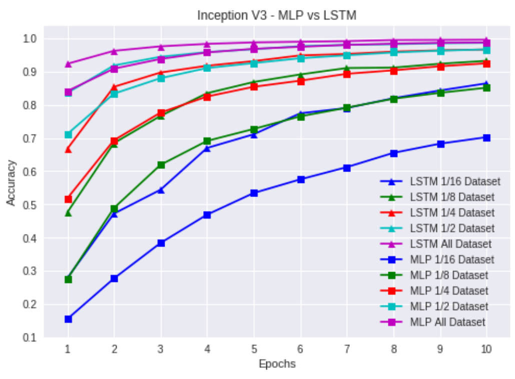
In terms of accuracy, Capsule 64 DW Max performs better than other Capsule models. Though Capsule 32 SC Mini can be a trade-off between the accuracy and the number of parameters.

In comparison with DL models, Capsule 32 DW Mini has a larger number of parameters, yet it achieves the accuracy of MobileNet V1 after a few more epochs. Meanwhile, Capsule 32 SC Mini can attain that of MobileNet V1 LSTM's accuracy with 3 to 4 times smaller in the number of parameters.

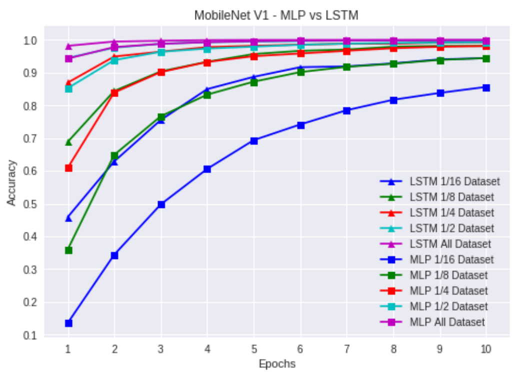
Capsule 32 DW Mini and 64 DW Max outperform Inception V3 MLP and LSTM on accuracy. Moreover, Capsule 64 DW Max occupies 5% less the number of parameters than Inception V3 LSTM though Capsule 32 DW Mini has 4 times larger size than Inception V3 MLP.

After a thorough literature search, we believe that this is the first work that proposes the integration of Depthwise Separable Convolution into Capsule Networks. Additionally, we provide empirical evaluations of the proposed CNs versus the best DL models.

In future work, we will apply the proposed CNs on different datasets and will develop these networks for mobile platforms.



(a) MLP vs LSTM on Inception V3



(b) MLP vs LSTM on MobileNet V1

Figure 4.5: Comparisons of MLP and LSTM on Deep Learning Models.

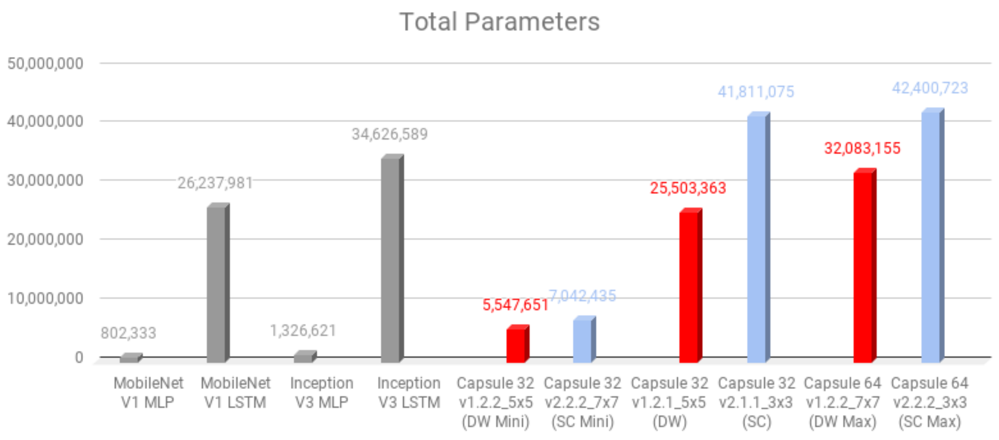


Figure 4.6: Total Parameters of Capsule Networks and Deep Learning Models. The numbers 32 and 64 denote Input image sizes 32×32 and 64×64, respectively; v1 stands for DW Capsule whereas v2 stands for SC Capsule. The next number expresses the amount of ConvNets. The last number denotes whether the ConvNet is followed by a Max Pooling (1 if not, 2 if followed). All ConvNets in the primary layer have the kernel sizes vary from 9×9, 7×7, 5×5 to 3×3.

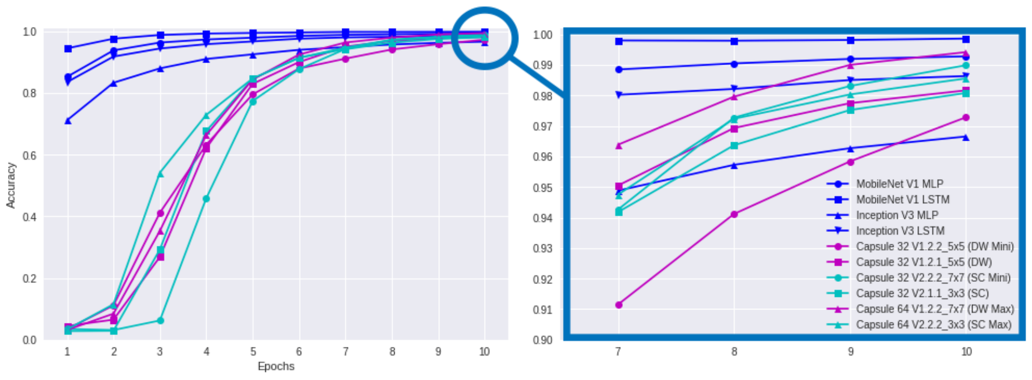


Figure 4.7: Capsule Networks vs Deep Learning Models

Chapter 5

Rethinking Recurrent Neural Networks and others

5.1 Introduction

Recently, the image recognition task has been transformed by the availability of high-performance computing hardware—particularly modern GPUs and large-scale datasets. The early designs of ConvNets in the 1990s were shallow and included only a few layers; however, as the ever-increasing volume of image data with higher resolutions required a concomitant increase in computing power, the field has evolved; modern ConvNets have deeper and wider layers with improved efficiency and accuracy [160, 171, 169, 65, 196, 22, 73]. The later developments involved a balancing act among network depth, width and image resolution [172] and determining appropriate augmentation policies [29].

During this same period, RNNs have proven successful at various applications, including natural language processing [116, 205], machine translation [23], speech recognition [20, 17], weather forecasting [143], human action recognition [215, 162, 115], drug discovery [155], and so on. However, in the image recognition field, RNNs are largely used merely to generate image pixel sequences [58, 133] rather than being applied for whole-image recognition purposes.

As the architecture of RNNs has evolved through several forms and become optimized, it could be interesting to study whether these spectacular advances have a direct effect on image classification. In this study, we take a distinct approach by integrating an RNN and considering it as an essential layer when designing an image recognition model.

In addition, we propose End-To-End multiple-model ensembles that learn expertise through the various models. This approach was the result of a critical observation: when training models for specific datasets, we often select the most accurate models or group some models into an ensemble.

We argue that merged predictions can provide a better solution than can a single model. Moreover, the ensembling process essentially breaks the complete operation (from obtaining input data to final prediction) into separate stages and each step may be performed on different platforms. This approach can cause serious issues that may even make it impossible to integrate the operation into a single location (e.g., on real-time systems) [192, 156] or to a future platform such as a System on a Chip (SoC) [50, 223].

Additionally, we explore other key techniques, such as the learning rate strategy and Test Time Augmentation, to obtain overall improvements. The remainder of this article is organized as follows. Our main contributions are discussed in Section 5.2. Various RNN formulations, including a typical RNN and more advanced RNNs, are the focus of subsection 5.3.1. In addition, our core idea for designing ConvNet models that gain experience from expert models is highlighted in subsection 5.3.2. In subsection 5.4.1, we evaluate our design on the iNaturalist dataset, and in subsection 5.4.2, the performances of various image recognition models integrated with RNNs are thoroughly analyzed. In subsection 5.4.3, we evaluate our design on the iCassava dataset. In subsections 5.4.4 and 5.4.5, we discuss our learning rate strategy and the softmax pruning technology. Additionally, we extend our experiments in subsection 5.4.6 to include more challenging datasets (i.e., SVHN, Cifar-100, and Cifar-10) and show that our model's performance can match the state-of-the-art models even under limited resources. In subsection 5.4.7, we show that our approach outperforms the state-of-the-art methods by a large margin. Finally, we conclude our research in Section 5.5.

5.2 Contributions

Our research differs from previous works in several ways. First, most studies utilize RNNs for sequential data and time series. Except for rare cases, RNNs are used largely to generate sequences of image pixels. Instead, we propose integrating RNNs as an essential layer of ConvNets.

Second, we present our core idea for designing a ConvNet in which the model is able to learn decisions from expert models. Typically, we choose predictions from a single model or an ensemble of models.

Another main contribution of this work involves a training strategy that allows our models to perform competitively, matching previous approaches on several datasets and outperforming some state-of-the-art models.

We make our source code so other researchers can replicate this work. The program is written in the Jupyter Notebook environment using a web-based interface with a few extra libraries.

5.3 Proposed Methods

Our key idea is to integrate an RNN layer into ConvNet models. We propose several RNNs and present the computational formulas. The concept of training a model to learn predictions from multiple individual models and the design of such a model is also discussed.

5.3.1 Recurrent Neural Networks

For the purpose of performance analysis and comparison, we adopt both a typical RNN and some more advanced RNNs i.e., LSTM and GRU as well as a Bidirectional RNN (BiRNN). The formulations of the selected RNNs are presented as follows.

Considering a standard RNN with a given input sequence $x_1, x_2, ..., x_T$, the hidden cell state is updated at a time step t as follows:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b), \tag{5.1}$$

where W_h and W_x denote weight matrices, b represents the bias, and σ is a sigmoid function that outputs values between 0 and 1.

The output of a cell, for ease of notation, is defined as

$$y_t = h_t, \tag{5.2}$$

but can also be shown using the $softmax$ function, in which $\hat{y_t}$ is the output and y_t is the target:

$$\hat{y_t} = softmax(W_y h_t + b_y). \tag{5.3}$$

A more sophisticated RNN or LSTM that includes the concept of a forget gate can be expressed as shown in the following equations:

$$f_t = \sigma(W_{fh} h_{t-1} + W_{fx} x_t + b_f), \tag{5.4}$$
$$i_t = \sigma(W_{ih} h_{t-1} + W_{ix} x_t + b_i), \tag{5.5}$$
$$c'_t = tanh(W_{c'h} h_{t-1} + W_{c'x} x_t + b'_c), \tag{5.6}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t, \tag{5.7}$$
$$o_t = \sigma(W_{oh} h_{t-1} + W_{ox} x_t + b_o), \tag{5.8}$$
$$h_t = o_t \odot tanh(c_t), \tag{5.9}$$

where the $\odot$ operation represents an elementwise vector product, and f, i, o and c are the forget gate, input gate, output gate and cell state, respectively. Information is retained when the forget gate f_t becomes 1 and eliminated when f_t is set to 0.

Because LSTMs require powerful computing resources, we use a variation, (i.e., a GRU) for optimization purposes. The GRU combines the input gate and forget gate into a single gate—namely, the update gate. The mathematical formulas are expressed as follows:

$$r_t = \sigma(W_{rh}h_{t-1} + W_{rx}x_t + b_r), \tag{5.10}$$

$$z_t = \sigma(W_{zh}h_{t-1} + W_{zx}x_t + b_z), \tag{5.11}$$

$$h'_t = tanh(W_{h'h}(r_t \odot h_{t-1}) + W_{h'x}x_t + b_z), \tag{5.12}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h'_t. \tag{5.13}$$

Finally, while a typical RNN essentially takes only previous information, BiRNNs integrate both past and future information:

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h), \tag{5.14}$$

$$z_t = \sigma(W_{ZX}x_t + W_{HX}h_{t+1} + b_z), \tag{5.15}$$

$$\hat{y_t} = softmax(W_{yh}h_t + W_{yz}z_t + b_y), \tag{5.16}$$

where h_{t-1} and h_{t+1} indicate hidden cell states at the previous time step $(t-1)$ and the future time step $(t+1)$.

For more details on the RNN, LSTM, GRU and BiRNN models, please refer to the following articles [82, 152, 207], [57, 48], [21, 82] and [112, 154] respectively.

5.3.2 End-to-end Ensembles of Multiple Models

Our main idea for this design is that when several models are trained on a certain dataset, we would typically choose the model that yields the best accuracy. However, we could also construct a model that combines expertise from all the individual models. We illustrate this idea in Figure 5.1.

As shown in the upper part, each actor represents a trained single model. For each presented sample, each actor predicts the probability that the sample belongs to each category. These probabilities are then combined and utilized to train our model.

The bottom part of the figure represents our ConvNet design for this idea. We essentially select three models to make predictions instead of two. Using only two models may result in a situation where one model dominates the other. In other words, we would use the predictions primarily from only one model. Adding one additional model provides a balance that offsets this weakness of a two-model architecture. Note that we limit our designs to just three models because of resource limitations (i.e., GPU memory). We name this model E2E-3M, where “E2E“ is an abbreviation of the “end-to-end“ learning process [66, 14, 13], in which a model performs all phases from training until final prediction. The “ 3M“ simply documents the combination of three models.

Each individual model (named Net 1, Net 2, and Net 3) is a fine-tuned model in which the last

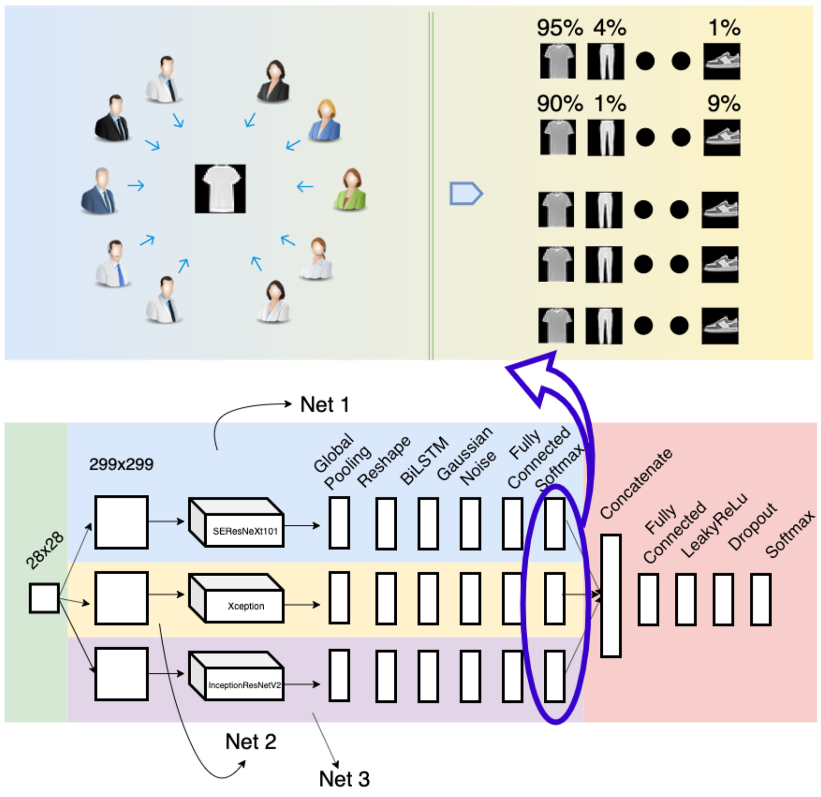


Figure 5.1: End-to-end Ensembles of Multiple Models: Concept and Design. The upper part of this image illustrates our key idea, in which several actors make predictions for a sample and output the probability of the sample belonging to each category. The lower part presents a ConvNet design in which three distinct and recent models are aggregated and trained using advanced neural networks. This design also illustrates our proposal of integrating an RNN into an image recognition model. (Best viewed in color)

layer is removed and replaced by more additional layers, e.g., global pooling to reduce the network size and an RNN module (including a reshape layer and an RNN layer). The models also employ Gaussian noise to prevent overfitting, include a fully connected layer and each model has its own softmax layer. The outputs from these three models are concatenated and utilized to train the subsequent neural network module, which consists of a fully connected layer, a LeakyReLU[121] layer, a dropout[164] layer and a softmax layer for classification.

Ensemble learning (EL) is a key aspect of this design. An ensemble refers to an aggregation of weaker models (base learners) combined to construct a model that performs more efficiently (a strong learner) [151]. The ensembling technique is more prevalent in ML than in DL, especially in the image recognition field, because convolution requires considerable computational power. Most of the recent related studies focus on a simple averaging or voting mechanism [136, 6, 42, 183, 191], and few have investigated integrating trainable neural networks [5, 45]. Our research differs from prior studies, because we study the design on a much larger scale using a larger number of up-to-date ConvNets.

Suppose that our ConvNet design has n fine-tuned models or classifiers and that the dataset contains c classes. The output of each classifier can be represented as a distribution vector:

$$\Delta_j = [\delta_{1j} \quad \delta_{2j} \quad \dots \quad \delta_{cj}], \tag{5.17}$$

where

$$\begin{gathered} 1 \leq j \leq n, \\ 0 \leq \delta_{ij} \leq 1 \quad \forall 1 \leq i \leq c, \\ \sum_{i=1}^{c} \delta_{ij} = 1. \end{gathered}$$

After concatenating the outputs of the n classifiers, the distribution vector becomes

$$\Delta = [\Delta_1 \quad \Delta_2 \quad \dots \quad \Delta_n]. \tag{5.18}$$

For formulaic convenience, we assume that the neural network has only one layer and that the number of neurons is equal to the number of classes. As usual, the networks' weights, Θ, are initialized randomly. The distribution vector is computed as follows:

$$\Delta^{'} = \Theta \cdot \Delta = [\delta_1^{'} \quad \delta_2^{'} \quad \dots \quad \delta_c^{'}], \tag{5.19}$$

where

$$\begin{gathered} 1 \leq j \leq c, \\ \delta_j^{'} = \sum_{i=1}^{nc} \delta_i w_{ij}. \end{gathered}$$

Finally, after the softmax activation, the output is

$$\eta_j^{'} = \frac{e^{\delta_j^{'}}}{\sum\limits_{i=1}^{c} e^{\delta_j^{'}}}. \tag{5.20}$$

5.4 Experiments and Results

In this section, we will present the experiments used to evaluate our initial design (without an RNN) and then analyze the performance when RNNs are integrated. We also describe our E2E ensemble multiple models, the developed training strategy, and the extension of the softmax layer. These experiments were performed on the iNaturalist'19 [181] and iCassava'19 Challenges [126], and on the Cifar-10 [95] and Fashion-MNIST [195] datasets. We also extend our experiments on Cifar-100 [95], SVHN [128] and Surrey [142].

5.4.1 Experiment 1: iNaturalist'19

DL [102] and ConvNet have achieved notable successes in the image recognition field. From the early LeNet model [103], first proposed several decades ago, to the recent AlexNet [97], Inception [170, 171, 169], ResNet [65], SENet [73] and EfficientNet [172] models, these ConvNets have leveraged automated classification to exceed human performance in several applications. This efficiency is due to the high availability of powerful computer hardware, specifically GPUs and big data.

In this subsection, we examine the significance of our design, which utilizes several ConvNets constructed based on leading architectures such as InceptionV3, ResNet50, Inception-ResNetV2, Xception, MobileNetV1 and SEResNeXt101. We adopt InceptionV3 as a baseline model since the popularity of ConvNets among deep learning researchers means they can often be used as standard testbeds. In addition, InceptionV3 is known for employing sliding kernels, e.g., 1×1, 3×3 or 5×5, in parallel, which essentially reduces the computation and increases the accuracy. We also use the simplest version of residual networks, i.e., . ResNet50 (ResNet has several versions, including ResNet50, ResNet101 and ResNet152, named according to the number of depth layers), which introduces short circuits through each network layer that greatly reduce the training time. In addition, we explore other ConvNets to facilitate the comparisons and evaluations.

Our initial design is depicted in Figure 5.2. The core of this process implements one of the abovementioned ConvNet models. While the architectures of these ConvNets differ, usually their top layers function as classifiers and can be replaced to adapt the models to different datasets. For example, Xception and ResNet50 use a Global Pooling and a FC layer as the top layers, while VGG19 [211] uses a flatten layer and two FCs layers (the original article used MP, but for some reason the Keras implementation uses a flatten layer).

Our design adds Global Pooling or Global Average Pooling (GAP) to decrease the output size of the networks (this is in line with most ConvNets but contrasts with VGGs, in which flatten layers are utilized extensively). Notably, we also insert an RNN module to evaluate of our proposed approach. The module comprises a reshape layer and an RNN layer as described in subsection 5.3.1. Moreover, we add a Gaussian noise layer to increase sample variation and prevent overfitting. In the FC layer, the number of neurons varies , (e.g., 256, 512, 1024 or 2048); the actual value is based on the specific

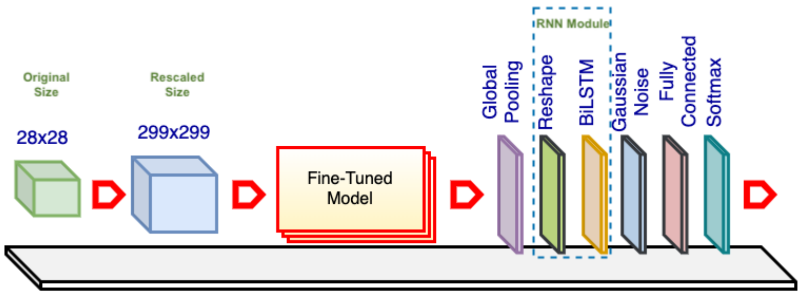


Figure 5.2: Single E2E-3M Model. The fine-tuned model is a pretrained ConvNet (e.g., InceptionV3) with the top layers excluded and the weights retrained. The base model comes preloaded with ImageNet weights. The original images are rescaled as needed to match the required input size of the fine-tuned model or for image resolution analysis. The global pooling layer reduces the networks' size, and the reshaping layer converts the data to a standard input form for the RNN layer. The Gaussian noise layer improves the variation among samples to prevent overfitting. The fully connected layer aims to improve classification. The softmax layer is another fully connected layer that has the same number of neurons as the number of dataset categories, and it utilizes softmax activation.

experiment. The softmax layer uses the same number of outputs as the number of iNaturalist'19 categories.

All the networks' layers from the ConvNets are unfrozen; we reuse only the models' architectures and pretrained weights (these ConvNets are pretrained on the ImageNet dataset [34, 149]). Reusing the trained weights offers several advantages because retraining from scratch takes days, weeks or even months on a large dataset such as ImageNet. Typically, TL can be used for most applications based on the concept that the early layers act like edge and curve filters; once trained, ConvNets can be reused on other, similar datasets [211]. However, when the target dataset differs substantially from the pretrained dataset, retraining or fine-tuning can increase the accuracy. To distinguish these ConvNets from the original ones, we refer to each model as a fine-tuned model.

We conduct our experiments on the iNaturalist'19 dataset, which was originally compiled for the iNaturalist Challenge, conducted at the 6th Fine-grained Visual Categorization (FGVC) workshop at CVPR 2019. In the computer vision area, FGVC has been a focus of researchers since approximately 2011 [185, 203, 90], although research on similar topics appeared long before [28, 49]. FGVC or subordinate categorization aims to classify visual objects at a more subtle level of detail than basic level categories [67], for example, bird species rather than just birds [185], dog types [90], car brands [94], and aircraft models [122]. The iNaturalist dataset was created in line with the development of FGVC [181], and the dataset is comparable with ImageNet regarding size and category variation. The specific dataset used in this research (iNaturalist'19) focuses on more similar categories than previous versions and is composed of 1,010 species collected from approximately two hundred thousand real plants and animals. Figure 5.3 shows some random images from this dataset and indicates the species shown as well as their respective classes and subcategories.

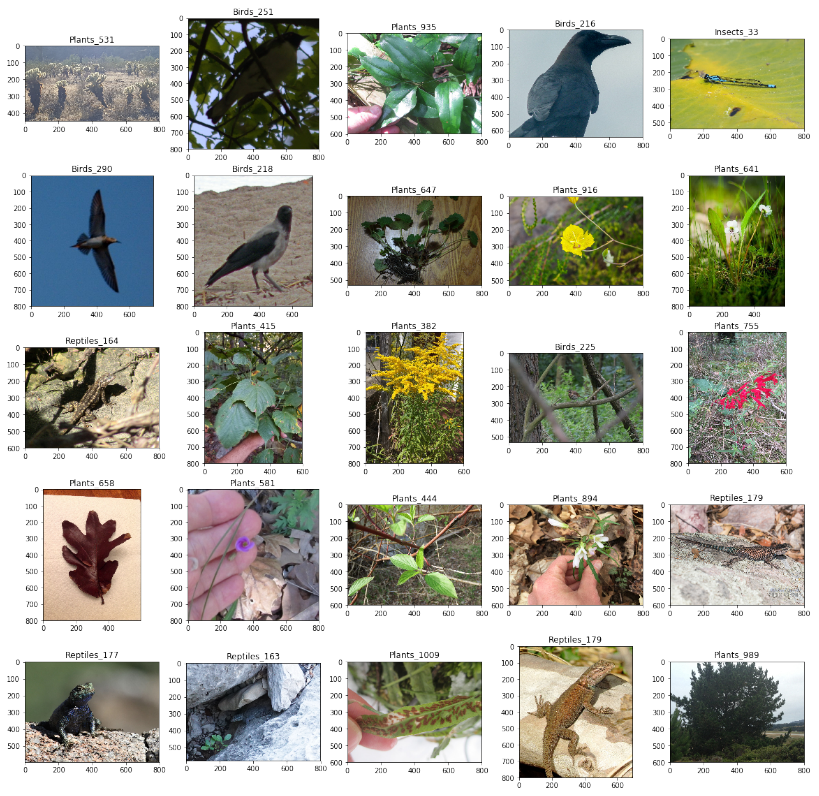


Figure 5.3: Random Samples from iNaturalist'19 dataset. Each image is labeled with the species name and its subcategory

The dataset is randomly split into training and test sets at a ratio of 80 : 20. In addition, the images are resized to appropriate resolutions, e.g., for the InceptionV3 standard, the rescaled images have a size of 299×299. The resolution is also increased to 401×401 or even 421×421. In the Gaussian noise layer, we set the amount of noise to 0.1, and in the FC layer, we chose 1,024 neurons based on experience, because it is impractical to evaluate all the layers with every setting.

We configured a Jupyter Notebook server running on a Linux operating system using 4 GPUs (GeForce® GTX 1080 Ti) each with 12 GB of RAM. For coding, we used Keras with a TensorFlow backend [2] as our platform. Keras is written in the Python programming language and has been developed as an independent wrapper that runs on top of several backend platforms, including TensorFlow. The project was recently acquired by Google Inc. and has become a part of TensorFlow.

Figure 5.4 shows the results of this experiment in which the top-1 accuracy is plotted against the Floating-point Operations per Second (FLOPS). The size of each model or the total number of parameters is also displayed. The top-1 accuracy was obtained by submitting predictions to the challenge website, obtaining the top-1 error level from the private leaderboard and subtracting the result from 1. During a tournament, the public leaderboard is computed based on 51% of the official test data. After the tournament, the private leaderboard contains a summary of all the data.

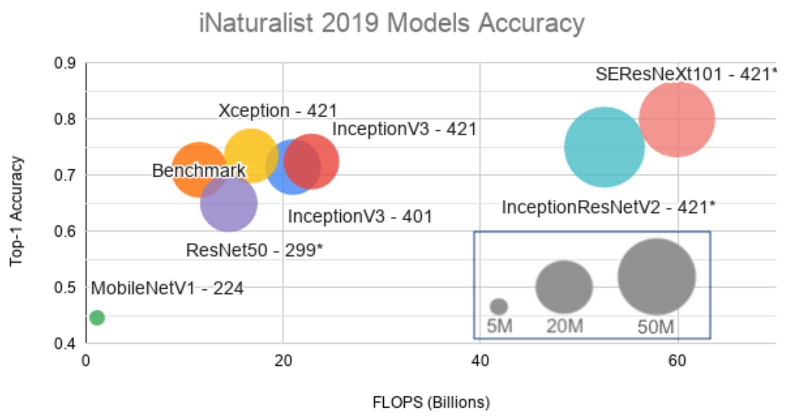


Figure 5.4: iNaturalist'19 Model Accuracy. Here, "Benchmark" denotes the InceptionV3 result using the default input setting (an image resolution of 299×299). The sizes of the input images for the other models are indicated along with their respective names. For example, Xception-421 indicates that the input images for that model have been rescaled to 421×421.

As expected, a higher image resolution yields greater accuracy but also uses more computing power for the same model (InceptionV3). As a side note, our benchmark achieved an accuracy of 0.7097, which is a marginal gap from the benchmark model on the Challenge website (0.7139). Because we have no knowledge of the organizers' ConvNet designs, settings and working environments, we cannot delve further into the reasons for this difference. Later, we increased the image resolutions from 299×299 to 401×401 and 421×421 and switched the fine-tuned models. Using Xception-421, our model obtained an accuracy of approximately 0.7347.

Because our server is shared, training takes approximately one week each time. This is the reason why we could increase the image size to only 421×421. In addition, we did not obtain the results for SEResNeXt101-421, Inception-ResNetV2-421 and ResNet50-299in this experiment, but projected their approximated accuracies; we will use these models in later experiments. Additionally, because adding an RNN module would substantially increase the training time, the RNN module is not analyzed on the iNaturalist'19 dataset.

5.4.2 Experiment 2: Fashion-MNIST

As mentioned in the previous section, most of the research regarding RNNs has focused on sequential data or time series. Despite the little attention paid to using RNNs with images, the main goal is to generate sequences of pixels rather than direct image recognition. Our approach differs significantly from typical image recognition models, in which all the image pixels are presented simultaneously rather than in several time steps.

We systematically evaluated the proposed design that utilizes distinct recurrent neural networks. These models include a typical RNN, an advanced GRU and a bidirectional RNN–BiLSTM—and we compared them against a standard (STD) model without an RNN module. In addition, we selected representative fine-tuned models, namely, InceptionV3, Xception, ResNet50, Inception-ResNetV2, MobileNetV1, VGG19 and SEResNeXt101, for comparison and analysis.

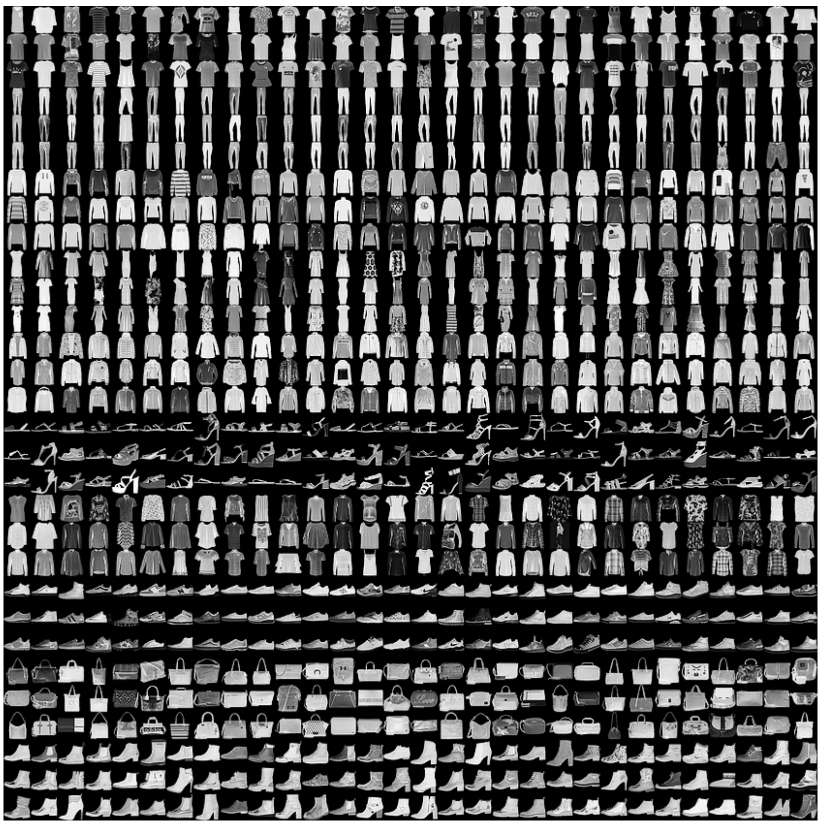


Figure 5.5: Some Samples from the Fashion-MNIST dataset

In this experiment, we employ the Fashion-MNIST dataset, which was recently created by Zalando SE and intended to serve as a direct replacement for the MNIST dataset as a machine learning benchmark because some models have achieved an almost perfect result of 100% accuracy on MNIST. The Fashion-MNIST dataset contains the same amount of data as MNIST–$50,000$ training and $10,000$ testing samples–and includes 10 categories. Figure 5.5 visualizes how this dataset looks; each sample is a 28×28 grayscale image.

Our initial design (as discussed previously) is reused with a highlighted note that the RNN module has been incorporated. The number of units for each RNN is set to 2,048 (in the BiLSTM, the number of units is 1,024); the time step number is simply one, which shows the entire image each time. In addition, because the image size in the Fashion-MNIST dataset is smaller than the desired resolutions (e.g., 244×244 or 299×299 for MobileNetV1 and InceptionV3), all the images are upsampled.

We performed these experiments on Google Colab[1], even though our server runs faster. The main reason is that Jupyter Notebook occupies all the GPUs for the first login section. In other words, only one program executes with full capability. In contrast, Colab can run multiple environments in parallel. A secondary reason is that the virtual environment allows rapid development (i.e., it supports the installation of additional libraries and can run programs instantly). All the experiments are set to run for 12 hours; some take less time before overfitting, but others take more than the maximum time allotted. We repeated each experiment 3 times. Moreover, in this study, we often submit the results to challenge websites and record only the highest accuracy rather than employing other metrics. The accuracy metric is defined as follows:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total numbers of predictions made}}. \tag{5.21}$$

Table 5.1 shows comparisons of the various models using different RNN modules built on InceptionV3 and MobileNetV1. The former is often chosen as a baseline model, whereas the latter is the lightest model used here in terms of parameters and computation. The results in Figure 5.6 show that the models with the additional RNN modules achieve substantially higher accuracy than do the STD models.

Table 5.1: Accuracy comparison of models using distinct recurrent neural networks built on InceptionV3 and MobileNetV1

	Recurrent Neural Networks			
Fine-tuned Models	STD	RNN	GRU	BiLSTM
InceptionV3	0.9434	0.9453	0.9430	0.9445
MobileNetV1	0.9404	0.9438	0.9439	0.9410

[1]https://colab.research.google.com/, which began as an internal project built based on Jupyter Notebook but was opened for public use in 2018. At the time of this writing, the virtual environment supports only a single 12 GB NVIDIA Tesla K80 GPU.

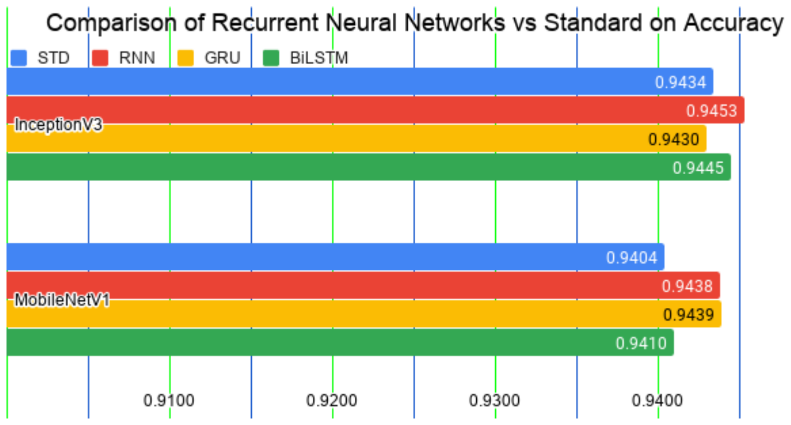


Figure 5.6: Accuracy comparison of models integrated with recurrent neural networks vs. the standard model (STD). The integrated models significantly outperform the STD models.

Table 5.2: Comparison of BiLSTM and STD on different fine-tuned models, including InceptionV3, Xception, ResNet50, Inception-ResNetV2, MobileNetV1, VGG19 and SEResNeXt101.

	Fine-tuned Models			
	Inception V3	**Xception**	**ResNet 50**	**InceptionResNet V2**
STD	0.9434	0.9374	0.9375	0.9364
BiLSTM	0.9445	0.9383	0.9377	0.9360
	MobileNet V1	**VGG19**	**SEResNeXt 101**	
STD	0.9404	0.9075	N/A	
BiLSTM	0.9410	N/A	N/A	

We also extended this experiment to include more models, including Xception, ResNet50, Inception-ResNetV2, VGG19 and SEResNeXt101. Table 5.2 shows the comparisons of models integrated with the BiLSTM module versus the standard models. Most of the models integrated with BiLSTMs significantly outperform the STDs, except for Inception-ResNetV2. Please note that training the VGG19 model required excessive training time; the model did not reach the same accuracy level as the other models even after 12 hours of training. Similarly, we were unable to obtain results for the VGG19-BiLSTM or SEResNeXt101 models. Therefore, Figure 5.7 shows only the results for InceptionV3, Xception, ResNet50, Inception-ResNetV2 and MobileNetV1.

5.4.3 Experiment 3: iCassava

Often, when training a model, we randomly split a dataset into training and testing sets with a desired ratio. Then, we repeat our evaluation several times, and finally, obtain the results from one

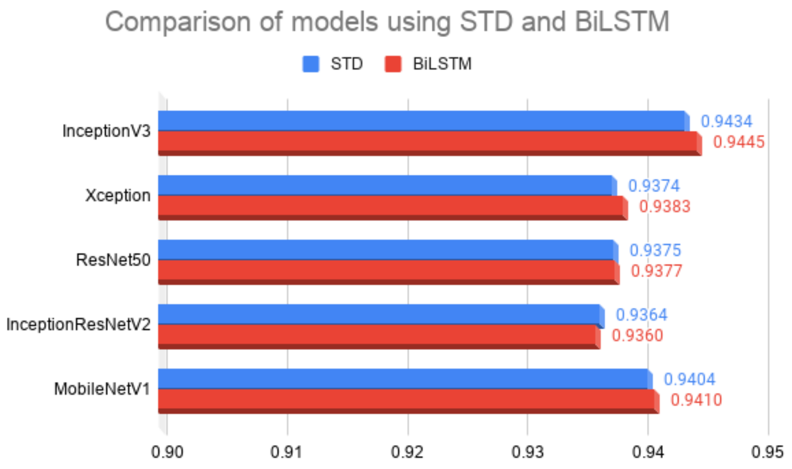


Figure 5.7: Comparison of the accuracy of BiLSTM and STD models using InceptionV3, Xception, ResNet50, Inception-ResNetV2 and MobileNetV1. The BiLSTM models substantially outperform the STD models in several instances.

of the measurement methods, e.g., accuracy mean and standard deviation. However, in competitions such as those on Kaggle, a test set is completely separated from a training set. If we naively divide the training set into a training set and a validation set, we face a dilemma in which all the samples of the original training set cannot be used for training because a portion of the dataset is always needed for validation. To solve this problem, we apply k-fold validation to the training set by dividing the dataset into k subsets, of which one is reserved for validation. We expand this process and make predictions on the official test set. This method allows our models (one model for each set) to learn from all the images in the official training set.

We also attempted to increase model robustness via data augmentation techniques. The goal of such techniques is to transform an original training dataset into an expanded dataset whose true labels are known[80, 221, 119]. Importantly, this teaches the model to be invariant to and uninfluenced by input variations [100]. For example, flipping an image of a car horizontally does not change its corresponding category. We applied the augmentation approach from [97] on the test set. In this approach a sample is cropped multiple times and the model makes predictions for each instance. The procedure has recently become standard practice in image recognition tasks and is referred to as “Test Time Augmentation (TTA)“. In this study, we cropped the images at random locations instead of at only the four corners and the center. In addition, we utilized the most prevalent augmentation techniques for geometric and texture transformations, such as rotation, width/height shifts, shear and zoom, horizontal and vertical flips and channel shift.

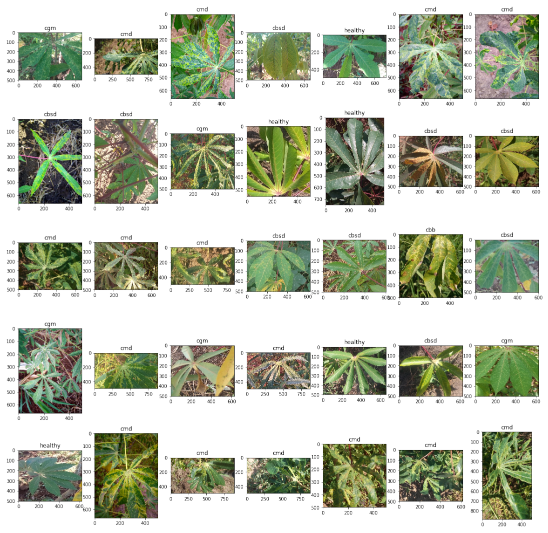


Figure 5.8: Random Samples from the iCassava dataset. CMD, CGM, CBB and CBSD denote cassava mosaic disease, cassava green mite disease, cassava bacterial blight, and cassava brown streak disease, respectively.

We also applied state-of-the-art EL because the technique is generally more accurate than is prediction from a single model. We use a simple averaging approach to aggregate all the models (a.k.a AVG-3M). Additionally, it is important to ensure a variety of the fine-tuned models to increase the classifier diversity because combining multiple redundant classifiers would be meaningless. We finally chose the SERestNeXt101, Xception and Inception-ResNetV2 fine-tuned models, as these ConvNets yield higher results than do others. Please note that the RNN module (including the reshape and BiLSTM layers) is excluded.

One last crucial technique is our training strategy, which helps to reduce loss and increase accuracy by searching for a better global minimum (more details will be presented in the next section).

We evaluated our approach using the iCassava Challenge dataset, which was also compiled for the FGVC6 workshop at CVPR19. In the iCassava dataset, the leaf images of cassava plants are divided into 4 disease categories: cassava mosaic disease, cassava green mite disease, cassava bacterial blight, and cassava brown streak disease, and 1 category of healthy plants, comprising 9,436 labeled images. The challenge was organized on the Kaggle website [2], ran from the 26th of April to the 2nd of June 2019, and attracted nearly 100 teams from around the world. The proposed models were evaluated on 3,774 official test samples, and the results were submitted to the website. The public leaderboard is computed from 40% of the test data, whereas the private leaderboard is computed from all the test data. Figure 5.8 shows some random samples from this dataset.

[2]https://www.kaggle.com/c/cassava-disease

Table 5.3: iCassava Result. The dataset is divided into 5 subsets. Four of the subsets are combined to form a training set and the other subset is used as for validation. Each of these combinations (including training and validation sets) composes a dataset, named Sets 1–5. A "Valid" label denotes results obtained using valid data, while "Private" and "Public" represent results obtained from corresponding categories on the challenge website. Test–0 Crop indicates test results obtained without cropping, whereas Test–3 Crop indicates test results obtained using the 3 cropping methods. The best results for each subset are summarized in the table at the bottom left, including both the private and public results. The table on the bottom right represents the results obtained using AVG-3M for each individual set and when each result is combined with all the others

Models		**Set 1**		**Set 2**		**Set 3**		**Set 4**		**Set 5**	
SEResNeXt 101	**Valid**	0.9043		0.9105		0.8963		0.9025		0.9061	
		Private	**Public**	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**
	Test–0 Crop	0.9019	0.8874	0.9129	0.8947	0.9041	0.8854	0.905	0.8814	0.9041	0.8874
	Test–3 Crop	0.9023	0.8887	0.9143	0.9026	0.9015	0.888	0.9032	0.89	0.9121	0.896
Xception	**Valid**	-		-		0.9025		0.914		0.9158	
		Private	**Public**	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**
	Test–0 Crop	-	-	-	-	0.901	0.89	0.9059	0.8986	0.9081	0.9
	Test–3 Crop	-	-	-	-	0.9045	0.8907	0.909	0.8966	0.9045	0.8986
InceptionResNet V2	**Valid**	-		-		0.9043			0.9078	0.9114	
		Private	**Public**	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**
	Test–0 Crop	-	-	-	-	0.9081	0.8993	0.9098	0.894	0.9156	0.9026
	Test–3 Crop	-	-	-	-	0.8869	0.8854	0.9094	0.8962	0.9151	0.9026

	Private	**Public**
Set 5	0.9121	0.896
Set 4	0.905	0.8814
Set 3	0.9041	0.8854
Set 2	0.9143	0.9026
Set 1	0.9023	0.8887
All	0.928	0.9139

	Private	**Public**
AVG-3M SET 5	0.9240	0.9125
All	0.9324	0.9165
AVG-3M SET 4	0.9235	0.9099
All	**0.9368**	0.9198
AVG-3M SET 3	0.9121	0.9079
All	0.9341	0.9258

The iCassava official training set is split into 5 subsets for k-fold cross validation, in which one subset is reserved for testing and the others are used for training in turn. We performed these experiments on a server using a GeForce® GTX 1080 Ti graphics card with 4 GPUs, each with 12 GB of RAM.

When evaluating the test set, we upsampled the images to a higher resolution (540×540) and then randomly cropped them to the input size (501×501). We varied the numbers of crops among 1, 3, 5, 7 and 9, and finally selected just 3 crops, as this option yielded higher accuracy than did the other cropping choices in most subsets. We collected the results for two methods, one without cropping and the other with 3 crops, and selected the one with the higher accuracy for each set. Then, we averaged the results for all the sets to provide a result for the entire dataset. The overall result achieved an accuracy of 0.928.

Furthermore, we applied the AVG-3M design to sets number 3, 4 and 5 and averaged each of these results with all other sets. Table 5.3 shows the results for this experiment. As the table shows, employing our AVG-3M model for these sets improves the accuracy. The combination of AVG-3M on Set 4 with the remaining sets yields the highest result (0.9368). It is critical to note that even though the public leaderboard result for all sets with the AVG-3M of Set 3 yields a higher result than that of Set 4, eventually, the latter achieved a better result. Therefore, choosing the submission

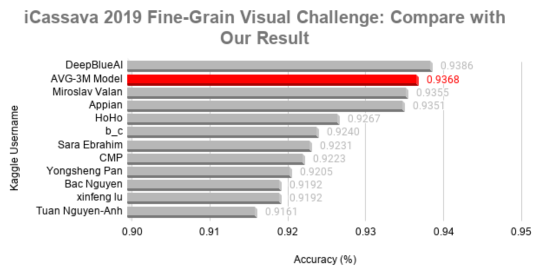


Figure 5.9: Comparison of our AVG-3M result on the iCassava challenge dataset with those of the top-10 teams

for the final evaluation is efficient and essential. Figure 5.9 shows our results in comparison with those of other top-10 teams.

5.4.4 Experiment 4: Training Strategy

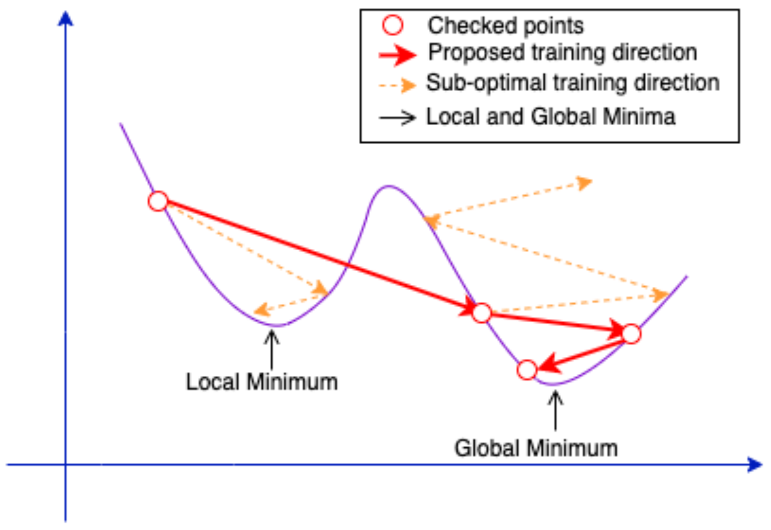


Figure 5.10: Training Strategy. We start with a moderate learning rate so that training does not become stuck in a local minimum. Then, we reduce the learning rate for the 2nd and 3rd iterations. If the learning rate is too large, the global minimum might be ignored; however, if it is too small, the training time becomes excessive.

In this subsection, we address the overall improvement of our model using a learning rate strategy.

We apply the Adam optimizer [91], which is an advanced optimizer in the deep learning area. The computations are as follows:

$$w_t = w_{t-1} - \eta_t \cdot \frac{m_t}{(\sqrt{v_t} + \hat{\epsilon})}, \tag{5.22}$$

$$\eta_t = \eta \cdot \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}, \tag{5.23}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t, \tag{5.24}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2, \tag{5.25}$$

where w and η are the weight and the learning rate of the neural networks, respectively; m, v and g are the moving averages and gradient of the current mini-batch; and the betas (β_1, β_2) and epsilon ϵ are set to 0.9, 0.999 and 10^{-8}, respectively.

We used Keras to implement the models; on that platform, the formula for computing the learning rate with decay is

$$\eta = \eta \cdot \frac{1}{1 + decay \cdot iterations}. \tag{5.26}$$

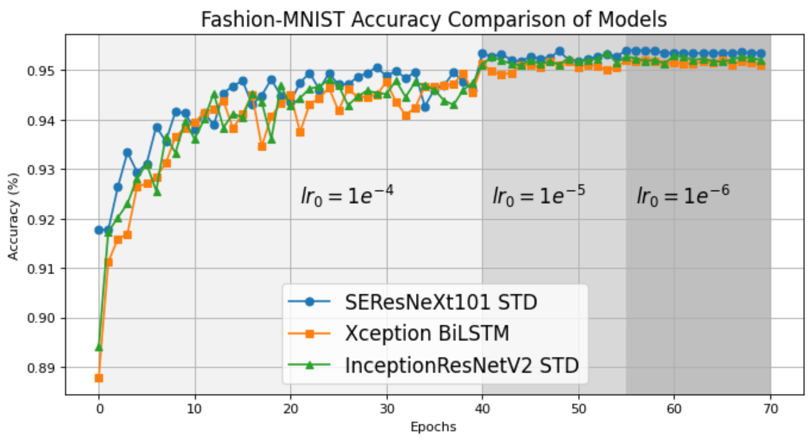


Figure 5.11: Model Accuracy on the Fashion-MNIST dataset Using Our Training Strategy. The models are trained with a learning rate starting at $1e^{-4}$ for 40 epochs. The RNN modules (STD, RNN, LSTM and GRU) are compared. The best module is selected for each model, e.g., the BiLSTM for Xception. The models are reloaded with the highest check points and trained two more times for 15 epochs each at learning rates of $1e^{-5}$ and $1e^{-6}$, respectively.

Choosing an appropriate learning rate is both essential and critical, because the training time is often dramatically reduced with an appropriate learning rate. However, selecting an appropriate learning rate is difficult. When the step size is too large, the global minimum might be ignored, whereas a too-small step size can result in an excessive training time. In these experiments, we started with a moderate learning rate and trained the model until the accuracy stopped improving. Then, we reduced the learning rate, reloaded the ConvNet weights that achieved the highest accuracy and repeated this process for two more times. Figure 5.10 illustrates our proposed training procedure.

We evaluated these experiments on the Cifar-10 and Fashion-MNIST datasets. Initially, the learning rate was set to $1e^{-4}$ and then sequentially changed to $1e^{-5}$ and $1e^{-6}$. The decay was derived by dividing the learning rate by the number of epochs.

Figure 5.11 shows the performances of the top 3 models on Fashion-MNIST using SEResNeXt101 STD, Xception LSTM and Inception-ResNetV2 STD and the figure excludes the results of other ConvNets discussed in the previous sections. The three RNN variations were also analyzed against the STD models, and only the model with the highest accuracy is shown in the figure. The accuracies are effectively increased after the transition. SEResNeXt101 achieved a result of 0.9541 under the STD setting.

This result was further improved to 0.9585 when using the E2E-3M model The design is shown in Figure 5.1, and the steps are detailed in Algorithm 1. The settings were as follows: the fully connected layer (after the concatenation layer) had 4,096 neurons; the LeakyReLU had a slope of 0.2 and dropout was set at 0.5. Please note that when reloading weights, we sometimes need to convert the weights to Pickle format rather than Keras' standard HDF5 format because the networks' weights are too large.

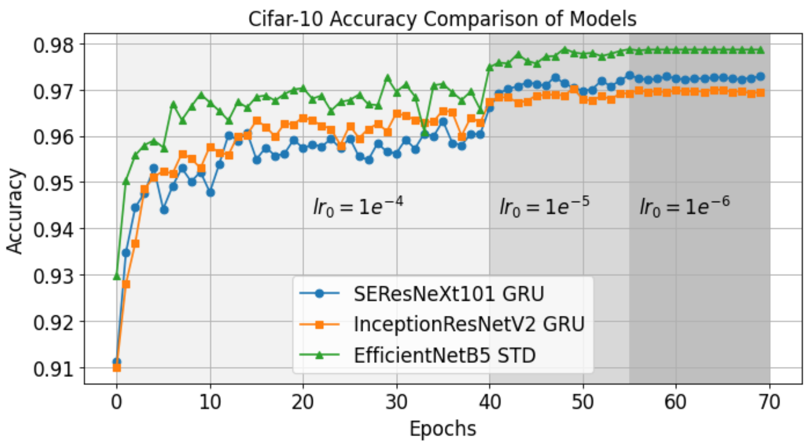


Figure 5.12: Model Accuracies on Cifar-10 Using Our Training Strategy. The models are trained with a learning rate starting at $1e^{-4}$ for 40 epochs. Then the RNN modules (including STD, RNN, LSTM and GRU) are compared. The best module is selected for each model, (e.g., STD in the case of EfficientNetB5). The models are then reloaded with the highest checkpoints and trained twice more for 15 epochs each at learning rates of $1e^{-5}$ and $1e^{-6}$, respectively.

We conducted experiments in the same manner on the Cifar-10 dataset. The results are shown

Algorithm 1 E2E-3M

```
Input: A training set with c categories $D := (a_1, b_1), (a_2, b_2) \dots (a_n, b_n)$
Output: Class Predictions
 1: Step 1: Train the Level-1 classifiers
        Number of L1 learners = m
 2: Step 2: Train m fine-tuned models
 3: Step 3: Select top-3 models
 4: Step 4: Reload the weights of the three models
 5: Step 5: Construct a new dataset of predictions
 6: for i = 1 to $n$ do
        $M_i = (a_i^{'}, b_i)$
        where:
        $a_i^{'} = [\Delta_1(a_i)\ \Delta_2(a_i)\ \Delta_3(a_i)]^T$
        $\Delta_j = [\delta_1\ \delta_2 \dots \delta_c]$
 7: end for
 8: Step 6: Train $\lambda$ neurons in the fully connected neural networks ($M$)
        $\mu_j = \sum_{i=0}^{3c-1} \delta_i w_{ij},\ 1 \leq j \leq \lambda$
 9: Step 7: Apply LeakyReLU activation (slope = $k$)
        $\eta_j = \begin{cases} \mu_j, & \text{if } \mu_j \geq 0 \\ k\mu_j, & \text{otherwise} \end{cases}$
10: Step 8: Regularize using dropout (rate = $p$)
        $\upsilon_j = \gamma \eta_j$
        where $\gamma$ is a gating variable 0-1 that follows a Bernoulli distribution with $P(\gamma = 1) = p$
11: Step 9: Train the Final Fully Connected Neural Networks
12: Step 10: Apply Softmax
```

in Figure 5.12. Please note that we also added EfficientNetB5, as this ConvNet is one of the latest models in the field. Using EfficientNetB5, we achieved an accuracy of 0.9788 under the STD setting.

In summary, our learning rate strategy differs from conventional methods, where the learning rates are predefined based on experiments. Applying an adaptive learning rates offers more flexibility to adjust the learning rates automatically during training. Our approach applies the Adam optimizer but in an intuitive way. After training for some epochs with the optimizer, the models begin overfitting. Thus, in our method, we reduce the learning rate and reload the best model, which leverages the accuracy to a higher point.

5.4.5 Experiment 5: Pruning

In this subsection, our setup includes a variation of the softmax layer in which only the outputs of the most active neurons are used for prediction. We observe that in multicategory prediction, often, only a few or even just one confidence value on a category is large enough to be meaningful, while others are very small, i.e., a confidence level of nearly zero percent. For this reason, we propose eliminating meaningless confidence levels by zeroing them before use in ensembling. We compared this approach (namely, EXT-Softmax) with a typical method where multiple predictions are averaged to obtain a final prediction (AVG-Softmax). The essential steps are illustrated in Algorithm 2.

Algorithm 2 Pruning

Input: Prediction Confidence A
Output: Pruned A

```
1: function PRUNE($A$)
2:     $N \leftarrow length(A)$
3:     $M \leftarrow length(A[0])$
4:     for $i \leftarrow 0$ to $N - 1$ do
5:         $indexMax \leftarrow findIndexOfMaxValue(A)$
6:         for $j \leftarrow 0$ to $M - 1$ do
7:             if $A[i] \neq indexMax$ then
8:                 $A[i] \leftarrow 0$
9:             end if
10:        end for
11:    end for
12:    return $A$
13: end function
```

The results on Fashion-MNIST are 0.9592 and 0.9591 without improvement using the proposed approach. However, on Cifar-10, the accuracy is higher, ranging from 0.9833 to 0.9836. Tables 5.4 and 5.5 show the latest achievements on these datasets. Please note that the Fashion-MNIST results were voluntarily submitted and were not officially verified. However, we analyzed each profile and selected only results that are supported by publications. Importantly, the dataset was changed recently to eliminate duplicates ; our results might have been even higher if the previous version had been used.

5.4.6 Experiment 6: Ensemble learning on Cifars and SVHN

In this subsection, we extend our experiments from an ensemble with only the three most accurate ConvNet models to a broader number of ConvNet models because a greater variety of predictions may yield better results. We also evaluated our models to the following more challenge datasets, i.e., SVHN [128] and Cifar-100 [95] datasets, in addition to Cifar-10. SVHN offers two versions: an original version with roughly similar number of samples as Cifar-10 and an extra version that

Table 5.4: List of latest achievements on Fashion-MNIST. The results were submitted to the official website for the Fashion-MNIST dataset. “Classifier“ indicates the main method that was used to achieve the result.

Classifier	**Accuracy**	**Submitter**
WRN-28-10 + Random Erasing	0.963	@zhunzhong07
WRN-28-10	0.959	@zhunzhong07
Dual path network with WRN-28-10	0.957	@Queequeg
DENSER	0.953	@fillassuncao
MobileNet	0.950	@Bojone
CNN with optional shortcuts	0.947	@kennivich
Google AutoML	0.939	@Sebastian Heinz
Capsule Network	0.936	@XifengGuo
VGG16	0.935	@QuantumLiu
LeNet	0.934	@cmasch
AVG-Softmax	**0.9592**	N/A
EXT-Softmax	**0.9591**	N/A
E2E-3M	**0.9585**	N/A
SeResNeXt101-STD	**0.9541**	N/A

Table 5.5: List of recent achievements on Cifar-10 along with the results of our models. The proposed approach performs comparably to the top models.

Authors	**Accuracy (%)**
Huang et al. [75]	99.00
Cubuk et al. [29]	98.52
Nayman et al. [127]	98.40
EXT-Softmax	**98.36**
AVG-Softmax	**98.33**
EfficientNetB5-STD	**97.88**
Yamada et al. [199]	97.69
DeVries et al. [35]	97.44
SEResNeXt101-GRU	**97.31**
Inception-ResNetV2-GRU	**97.02**
Zhong et al. [220]	96.92
Liang et al. [110]	96.55
Huang et al. [74]	96.54
Graham [56]	96.53
Zhang et al. [214]	96.23

increases the number of samples by an order of magnitude. Cifar-100, as the name suggests, expands the number of categories from ten to one hundred.

We use the same settings for Cifar-10 and Cifar-100 and slightly adjusted these configurations for SVHN. In addition to eliminating the flipping augmentations (since the numbers are changed even become another number e.g. number 2 looks similar to number 5 after flipping), we started the learning rate at $1e^{-3}$ instead of $1e^{4}$. Moreover, because the required training time is 10 times longer, we repeated the process only once rather than twice. For the specific settings, please refer to the provided source code.

Table 5.6 compares the performances of our models with those of other approaches on Cifar-10, Cifar-100 and SVHN. For Cifar-10, our ensemble outperforms the previous state-of-the-art architecture [127] by a large margin. Across all the datasets, the neural architecture search seems to work well on small datasets but appears to struggle on larger datasets as search space grows exponentially. For Cifar-100, our method is much better than Autoaugment [29], which requires approximately 5000 hours of training for image augmentation (we trained each model for few days to one week that depends on the model size). In addition, we attempted to evaluate our model on SVHN. To match recent works, we utilized the extra dataset. The results of our first attempt were as good as the population-based augmentation method[69]. For our second attempt, we upgraded our server to the latest libraries, which improved the accuracy to match that of Randaugment [30]. Additionally, our method outperforms the abovementioned methods on Cifar-10.

5.4.7 Experiment 7: Surrey

In this subsection, we compared our method with state-of-the-art approaches on the Surrey dataset [142] using leave-one-out cross validation. The dataset contains more than $120,000$ color and depth images, each allocated into five subsets. The highest result of 0.9353 accuracy was achieved by the recent approach in [202]. Our model outperforms the state-of-the-art model by 1.4% when using only color images. Figure 5.13 shows our results on the InceptionV3 and MobileNet fine-tuned ConvNets. In Table 5.7, we compare the performance of our ensemble with those of other models.

Generally, these results demonstrate that our approach outperforms the state-of-the-art by large margins and is suitable for use in a variety of circumstances.

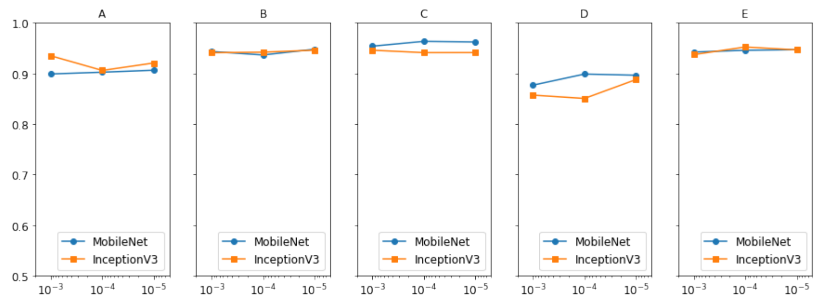


Figure 5.13: Performances of MobileNet and InceptionV3 on leave-one-out cross validation using the five sets from Surrey

Table 5.6: Results on Cifar-10, Cifar-100 and SVHN

Datasets						
Cifar-10		Cifar-100		SVHN		
Method	Accuracy	Method	Accuracy	Method	Accuracy	
Randaugment [30]	0.9850	[30]	-	[30]	0.9900	
Ho et al. [69]	-	[69]	-	[69]	0.9899	
Autoaugment [29]	0.9852	[29]	0.8933	[29]	0.9898	
GPipe [75]	0.9900	[75]	0.913	[75]	-	
Big Transfer (BiT) [92]	-	[92]	0.9351	[92]	-	
EfficientNet [172]	-	[172]	0.917	[172]	-	
DeVries et al. [36]	0.9744	[36]	0.844	[36]	0.9870	
Zagoruyko et al. [209]	-	[209]	0.817	[209]	0.9846	
Zhang et al. [214]	-	[214]	-	[214]	0.9841	
Huang et al. [74]	0.9654	[74]	0.8282	[74]	0.9841	
Lee et al. [106]	-	[106]	-	[106]	0.9831	
XNAS [127]	0.9840	[127]	0.8640	[127]	0.9828	
Liao et al. [111]	-	[111]	-	[111]	0.9828	
Yamada et al. [200]	0.9769	[200]	0.8781	[200]	-	
Zhong et al. [221]	0.9692	[221]	0.8235	[221]	-	
SEResNeXt101	0.9731	EfficientNetB0	0.8487	EfficientNetB0	0.9867	0.9873*
InceptionResNetV2	0.9702	EfficientNetB1	0.8657	EfficientNetB1	0.9872*	0.9871
EfficientNetB5-345	0.9788	EfiicientNetB2	0.8664	EfiicientNetB2	0.9871	0.9875
EfficientNetB7-299	0.9779	EfficientNetB3	0.8700	EfficientNetB3	0.9866*	0.9867
EfficientNetB4-299	0.9785	EfficientNetB4	0.8663	EfficientNetB4	0.9871	0.9878*
Ensemble	**0.9852**	EfficientNetB5	0.8721	EfficientNetB5	0.9870	0.9874
		EfficientNetB6	0.8804	EfficientNetB6	0.9877*	0.988
		EfficientNetB7	0.8721	EfficientNetB7	0.9860*	0.9863
		SEResNeXt101	0.8541	InceptionResNetV2	0.9851	0.986*
		InceptionResNetV2	0.8609	SEResNeXt101	0.9858	0.9861
		Xception	0.8430	Xception	0.9843	0.9853
		InceptionV3	0.8365	**Ensemble***	**0.9899**	
		Ensemble (Exclude InceptionV3)	**0.9027**	EfficientNetB0	0.9864	0.9874
				EfficientNetB1	0.9873	0.9874*
				EfiicientNetB2	0.9865	0.9875
				EfficientNetB3	0.9867	0.9869
				EfficientNetB4	0.9870	0.9877
				EfficientNetB5	0.9875*	0.9873
				EfficientNetB6	0.9869	0.9864
				EfficientNetB7	0.9876	0.9872
				InceptionResNetV2	0.9849*	0.9862
				Ensemble*	**0.9900**	

5.5 Conclusions

In this work, we presented our core ideas for improving ConvNets by integrating RNNs as essential layers in ConvNets when designing E2E multiple-model ensembles that gain expertise from each individual E2E, an improved training strategy, and a softmax layer extension.

First, we propose integrating RNNs into ConvNets even though RNNs are mainly optimized for 1D sequential data rather than 2-D images. Our results on the Fashion-MNIST dataset show that ConvNets with RNN, GRU and BiLSTM modules can outperform standard ConvNets. We employed a variety of fine-tuned models in a virtual environment, including InceptionV3, Xception, ResNet50, Inception-ResNetV2 and MobileNetV1. Similar results can be obtained using a dedicated server for the SEResNeXt101 and EfficientNetB5 models on the Cifar-10 and Fashion-MNIST datasets. Although adding RNN modules requires more computing power, there is a potential trade-off between

Table 5.7: Comparison of our method with previous state of the art approaches on Surrey dataset.

Method	F1	Recall	Precision	Accuracy
[142]	-	-	0.4900	-
[88]	-	-	-	0.8430
[98]	-	-	-	0.5700
[39]	-	-	-	0.7330
[213]	-	-	-	0.5600
[135]	-	0.7700	0.7900	-
[188]	-	-	-	0.7000
[4]	-	-	-	0.7580
[147]	-	-	-	0.8380
[173]	0.792	-	-	0.8034
[212]	-	0.9240	0.9350	0.9270
[202]	0.9326	0.9348	0.941	0.9353
Our	-	-	-	**0.949**

accuracy and running time.

Second, we designed E2E-3M ConvNets that learn predictions from several models. We initially tested this model on the iNaturalist'19 dataset using only a single E2E-3M model. Then, we evaluated various models and image resolutions and compared them with the Inception benchmark. Then, we added RNNs to the model and analyzed the performances on the Fashion-MNIST dataset. Our E2E-3M model outperforms a standard single model by a large margin. The advantage of using an end-to-end design is that the model can run immediately in real time and it is suitable for SoC platforms.

Third, we proposed a training strategy and pruning for the softmax layer that yields comparable accuracies on the Cifar-10 and Fashion-MNIST datasets.

Fourth, using the ensemble technique, our models perform competitively, matching previous state-of-the-art methods (accuracies of 0.99, 0.9027 and 0.9852 on SVHN, Cifar-100 and Cifar-10, respectively) even with limited resources.

Finally, our method outperforms other approaches on the Surrey dataset, achieving an accuracy of 0.949.

In the future, we plan to extend our models to include a greater variety of settings. For example, we will evaluate bidirectional RNN and bidirectional GRU modules. In addition, we are eager to test our models on other datasets e.g. ImageNet.

Chapter 6

PSO-Convolutional Neural Networks

6.1 Introduction

Image classification lies at the core of many computer vision related tasks: extracting relevant information from telescope images in astronomy, navigation in robotics, cancer classification in medical image, security, to cite a few examples. It is currently almost an ontological commitment that Convolution Neural Networks are particularly tailored to Image Processing. However, training these (large-scale) neural networks for generalization is still a big part of the puzzle since the performance is sensitive to the architecture, the training set, the sample size (a.k.a. sample complexity) among other attributes.

More concretely, in the classification problem, ConvNets – or, more broadly, Deep Neural Networks – are trained to learn a target function $f^{\star} : \mathbb{R}^N \longrightarrow \{0, 1, 2, \ldots, M\}$ traditionally via SGD

$$\theta(t+1) = \theta(t) - \frac{\eta}{K} \sum_{i=1}^{K} \nabla_{\theta} \widehat{L}\left(x_i, f^{\star}(x_i), \theta(t)\right), \tag{6.1}$$

where $\widehat{L}$ is an estimate of the loss function; $\theta(t)$ is the vector collecting all the weights of the neural network at the iterate t; η is the step size; $K < T$ is the batch size and $\{x_i, f^{\star}(x_i)\}_{i=1}^{T}$ is the training set. The landscape being high-dimensional, nonconvex and with a *geometry*[1] that is sensitive to the architecture render the problem quite unstable to tuning. Generalization and consistency may be technically studied in certain ideal cases, e.g., under certain thermodynamic limit regimes on the number of neurons [123]. But for practical purposes, ascertaining a *proper* structure for the network

[1]The term geometry hereby subsumes a collection of attributes of the Loss function, e.g., regularity (how smooth it is), how deep and wide are the minima, etc., that impacts quite critically the SGD dynamics (6.1).

– i.e., a parsimonious structure yielding a Loss function that is amenable to generalization – is still quite challenging.

Recently, distributed approaches have been leveraged to overcome the *geometry* sensitivity of the landscape and yield a more robust approach to generalization. At the limelight lies nature inspired approaches: PSO [71, 87, 157], Ant Colony Optimization (ACO) [148, 40], Artificial Bee Colony Optimization (ABCO) [81, 85], etc. The core idea is that each *particle* treads the landscape exchanging information with neighboring particles about its current estimate of the geometry (e.g., the gradient of the Loss function) and its position. The overall goal in this framework is to devise a distributed collaborative algorithm that boosts the optimization performance by leading (at least some of the) particles up to the *best* minimum.

In this work, we propose a modified PSO-ConvNet training by incorporating some elements of Cucker-Smale dynamics [31] into the PSO algorithm. In addition, a novel alternative dynamics is introduced based on our observation of positions of particles. Further, we set one of the particles with a large random step-size. The idea in its core is simple: i) this *wilder* particle can scan the land in a faster time-scale; ii) it can only be trapped by *deeper* minima; iii) by properly tuning the weights, we enable this particle to attract the more conservative ones to the stronger minimum. A more detailed description of the algorithm will be provided in Section 6.2.

The main contributions of this work are:

1. It is the first work to integrate a modified Cucker-Smale dynamics into PSO algorithm on image classification to enhance particle's exploration capability.

2. A novel dynamics is proposed that achieves state-of-the-art results.

3. Special characteristics of hybrid ConvNet-PSO is taken into consideration by introducing wilder particles incorporated by a restricted random learning rate.

4. Extensive experiments are performed on large-scale Cifar-10 and Cifar-100 benchmark datasets to verify the effectiveness of the proposed methods.

The rest of the chapter is organized as follows: In Section 6.2, we describe our approach in detail from its dynamics formulas to proposed methods for improvement. Experiments and result discussions are presented in Section 6.3 and 6.4, respectively. In Section 6.5, we compare our approach with recent state-of-the-art to claim its superiority. Ultimately, we conclude our work in Section 6.6.

6.2 Proposed Methods

6.2.1 Collaborative Neural Networks

Define $\mathcal{N}(n,t)$ as the set of k nearest neighbor particles of particle n at time t, where $k \in \mathbb{N}$ is some predefined number. In particular, $\mathcal{N}(n,t) = \{x^{(n)}(t), x^{(i_1)}(t), x^{(i_2)}(t), \dots, x^{(i_k)}(t)\}$, where i_1, i_2,... i_k are the k closest particles to n and $x^{(i_k)}(t) \in \mathbb{R}^D$ represents the position of particle i_k at time t. Figure 6.1 depicts this idea. Given a (continuous) function $L : \mathbb{R}^D \longrightarrow \mathbb{R}$ and a (compact) subset

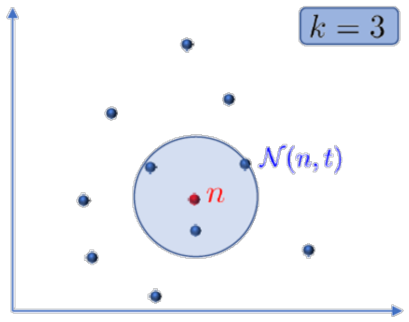


Figure 6.1: Illustration of the three closest particles to particle n. The neighborhood $\mathcal{N}(n,t)$ comprises the positions of these particles plus the position of particle n itself.

$S \subset \mathbb{R}^D$, define

$$\mathcal{Y} = \mathsf{argmin}\,\{L(y) \,:\, y \in S\} \tag{6.2}$$

as the subset of points that minimize L in S, i.e., $L(z) \leq L(w)$ for any $z \in \mathcal{Y} \subset S$ and $w \in S$.

We consider a collection of neural networks collaborating in a distributed manner to minimize a Loss function L. The neural networks are trained in two phases: i) [**regular phase**] each neural network is trained via (stochastic) gradient descent; ii)[**PSO phase**] the algorithm executes an intermediate step of SGD followed by a step of PSO-based cooperation: the vector of weights of each neural network can be cast as the position of a particle in $\mathbb{R}^D$, where D is the number of weights (and dimension of the phase space), and the dynamics of the particles (or neural networks) follow equation (6.4). Figure 6.2 illustrates the general idea. More concretely, the update rule is given by

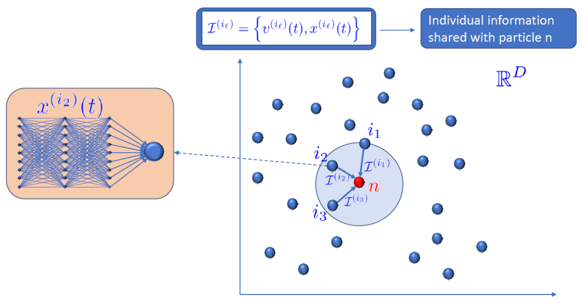

$\mathcal{I}^{(i_\ell)} = \left\{v^{(i_\ell)}(t), x^{(i_\ell)}(t)\right\}$
Individual information shared with particle n
$\mathbb{R}^D$
$x^{(i_2)}(t)$
i_1
i_2
i_3
$\mathcal{I}^{(i_1)}$
$\mathcal{I}^{(i_2)}$
$\mathcal{I}^{(i_3)}$
n

Figure 6.2: Illustration of the PSO phase. At each time instant t, particles share information with their current nearest neighbors. In particular, each particle knows at time t, the positions and velocities of its neighbors. The position of each particle at time t represents the weights of the underlying neural network.

the following dynamics

$$\begin{aligned}
\psi^{(n)}(t+1) &= -\eta \nabla L\left(x^{(n)}(t)\right) \\
\phi^{(n)}(t+1) &= x^{(n)}(t) + \psi^{(n)}(t+1) \\
v^{(n)}(t+1) &= \sum_{\ell \in \mathcal{N}(n,t)} w_{n\ell}\, \psi^{(\ell)}(t+1) \\
&\quad + c_1 r(t) \left(P^{(n)}(t) - \phi^{(n)}(t+1)\right) \\
&\quad + c_2 r(t) \left(P_g^{(n)}(t) - \phi^{(n)}(t+1)\right) \\
x^{(n)}(t+1) &= x^{(n)}(t) + v^{(n)}(t)
\end{aligned} \tag{6.3}$$

where $v^{(n)}(t) \in \mathbb{R}^D$ is the velocity vector of particle n at time t; $\psi^{(n)}(t)$ is an intermediate velocity computed from the gradient of the Loss function at $x^{(n)}(t)$; $\phi^{(n)}(t)$ is the intermediate position computed from the intermediate velocity $\psi^{(n)}(t)$; $r(t) \overset{i.i.d.}{\sim} \mathsf{Uniform}\left([0,1]\right)$ is randomly drawn from the interval $[0,1]$ and we assume that the sequence $r(0)$, $r(1)$, $r(2)$, $\ldots$ is i.i.d.; $P^{(n)}(t) \in \mathbb{R}^D$ represents the *best* position visited up until time t by particle n, i.e., the position with the minimum value of the Loss function over all previous positions $x^{(n)}(0)$, $x^{(n)}(1)$, $\ldots$, $x^{(n)}(t)$; $P_g^{(n)}(t)$ represents its nearest-neighbors' counterpart, i.e., the best position across all previous positions of the particle

n jointly with its corresponding nearest-neighbors $\bigcup_{s\leq t}\mathcal{N}(n,s)$ up until time t:

$$\begin{aligned} P^{(n)}(t+1) &\in \mathsf{argmin}\left\{L(y)\,:\,y=P^{(n)}(t),x^{(n)}(t)\right\} \\ P_g^{(n)}(t+1) &\in \mathsf{argmin}\Big\{L(y)\,:\,y=P_g^{(n)}(t),x^{(k)}(t);\ \ . \\ & \quad k\in\mathcal{N}(n,t)\} \end{aligned} \tag{6.4}$$

The weights $w_{n\ell}$ are defined as

$$w_{n\ell}=f\left(\left|\left|x^{(n)}(t)-x^{(\ell)}(t)\right|\right|\right), \tag{6.5}$$

with $||\cdot||$ being the Euclidean norm and $f:\mathbb{R}\to\mathbb{R}$ being a decreasing (or at least non-increasing) function. We start by assuming that

$$f(z)=\frac{M}{(1+z)^{\beta}}, \tag{6.6}$$

for some constants $M,\beta>0$.

6.2.2 Alternative Dynamics

One alternative to the equation (6.3) is to *pull back* a particle rather than push the particle in the same direction of gradient.

Figure 6.3 illustrates the conceptions of the dynamics. In the previous section, particles are assumed to locate in the same side of a valley of the loss function. However, when at least one of the particles lies in an “antipode” side of the valley relative to the cluster of remaining particles, the particle is actually pulled away from the minimum by using the first dynamics (Dynamics 1). This is the reason why we introduce the second dynamics (Dynamics 2) to pull the particle back instead. Thus, the formula is as follows:

$$\begin{aligned} x_{(i)}(t+1) &= x_{(i)}(t) \\ & \quad +\textstyle\sum_{j=1}^{N}\frac{M_{ij}}{(1+||x_i(t)-x_j(t)||^2)^{\beta}}(x_j(t) \\ & \quad -\nabla L(x_j(t))) \\ & \quad +cr\left(P_{nbest(i)}(t)-x_i(t)\right) \end{aligned} \tag{6.7}$$

where $x_{(i)}(t)\in\mathbb{R}^D$ is the position of particle i at time t; M, β and c are constants decided by experiments with $||\cdot||$ being the Euclidean norm; $r(t)\overset{i.i.d.}{\sim}\mathsf{Uniform}([0,1])$ is randomly drawn from the interval $[0,1]$ and we assume that the sequence $r(0)$, $r(1)$, $r(2)$, ... is i.i.d.; $P_{nbest(i)}(t)\in\mathbb{R}^D$

represents nearest-neighbors' best , i.e., the best position across all previous positions of the particle n jointly with its corresponding nearest-neighbors $\bigcup_{s\le t}\mathcal{N}(n,s)$ up until time t.

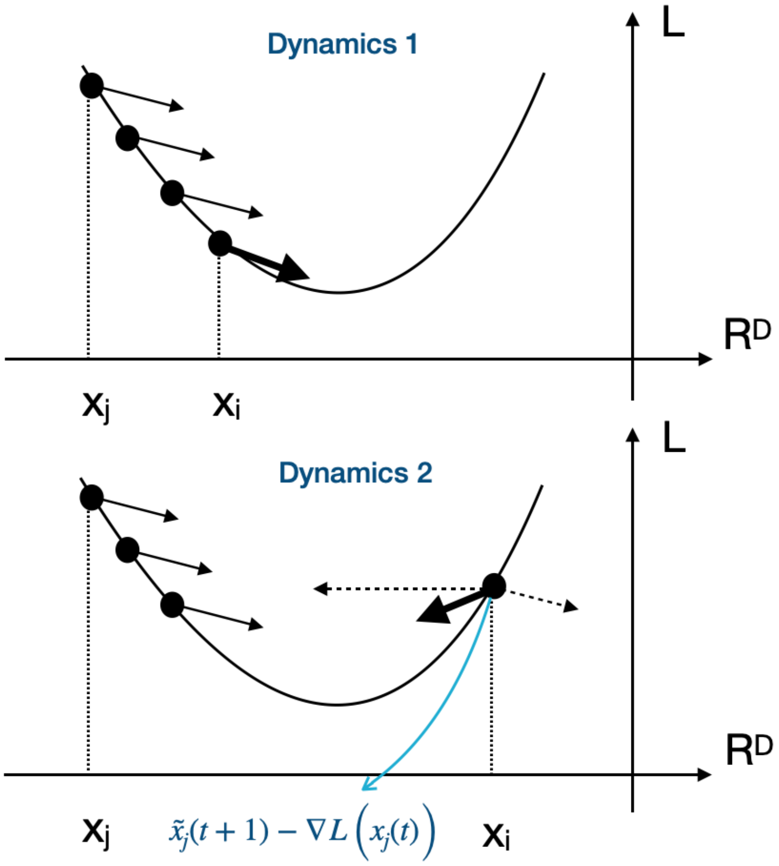


Figure 6.3: Illustration of the Dynamics. The top part of the figure demonstrates the concept of equation (6.3) whereas the bottom part of the figure shows the concept of equation (6.7). The solid line arrows indicate the directions of the particles. The dash line arrows are vector components.

6.2.3 Further improvements

As referred before, the vector of weights of each ConvNet plays the role of the position of a particle in phase space. In our framework, we have a *swarm* of ConvNets designed to collaborate in order to attain the best minimum. We discuss further improvements including wider PSO-ConvNet strategy, warmup training and cluster warmup learning rate as follows.

Wilder PSO-ConvNet Strategy

In this section, we propose to improve the dynamics by distinguishing particles based on an essential inner characteristic. Previously, particles are influenced by only distances from their neighbors. However, we argue that some particles might have more effect than the other particles because ConvNet are distinct with intrinsic hyper-parameters (e.g. learning rate, kernel size and number of filters, just to name a few).

Specially, since training ConvNet is an essential phase (intertwining between ConvNet and PSO phases) but has a tendency of getting stuck in local optima, we enable ConvNets to escape the local-minima i.e. a particle accompanied by a learning rate that can freely change (in regular phase). We choose a *random* learning rate over other algorithms as the movement is wildest and much stronger than adaptive learning rates and simpler than the cyclic learning rate. As a result, two types of particles are introduced: i) [**fast particle**] endowed with a large learning rate; ii) [**slow particle**] abiding by a small learning rate. A larger learning rate offers a faster training (easier to move out of local minima), but compromises the generalization performance as it incurs greater error. On the other hand, a small learning rate fosters generalization (can be trapped by deeper minima) while requiring a large training time.

Thus, in PSO phase (collaborative phase), a mechanism for conservative particles to be attracted by the wilder particle is proposed. The main idea is simple, by tuning proper weight, the wilder particle would guide other conservative particles to the better minimum.

We restrict the learning rate hyper-parameter to a fixed interval of admissible values. This is motivated by: (i) with a *small* learning rate, the training would take significant longer time; (ii) with a *large* learning rate, the training dynamics becomes unstable compromising convergence to the minima [61, 70]. Therefore, in our formula (equation (6.8)), we set the minimum and the maximum values of the learning rate range. Because the learning rate is in an exponential range, e.g., from 10^{-6} to 10^{-1}, we re-scale this range into a linear range and generate learning rates using uniform distribution function so that the small learning rates get an equally likely number of samples as large learning rates. The min and max boundary can be determined via running a learning rate scan between low to high values [161].

$$f(lr_{min}, lr_{max}) = 10^{U(lr_{min}, lr_{max})} \tag{6.8}$$

where U denotes uniform distribution function; $lr_{min} = log_{10}(min)$ and $lr_{max} = log_{10}(max)$.

Warmup Training

Inspired by warmup concept that recently emerged in training deep neural networks [184, 55, 53, 114], we propose *warump training.* Our idea differs in terms of proportion of warmup time and full training time. In the former, the time required is typically very short, whereas in our technique, warump time often takes a large proportion for ConvNets to train until a desired performance is achieved.

The main reason why we propose warmup training lies in the principle of ConvNets – training ConvNets takes time. Training on big datasets, e.g., Cifar-10, Cifar-100, SVHN or ImageNet requires from several hours to weeks or months. In addition, we have a few number of ConvNets, so collaboration at an earlier time may reduce opportunity for individual ConvNets to explore more freely. In other words, early collaboration maybe premature. Therefore, we propose to split the warmup training into two steps including (1) a training without intertwining between ConvNet and PSO; (2) and a training with the intertwining. In the second step, training a ConvNets is still needed since the size of ConvNet is huge with millions of weights, thus, a few PSO-ConvNets is certainly not enough for optimization.

Cluster Warmup Learning Rate

We attempt to improve the performance of our hybrid PSO-ConvNet by proposing cluster warmup learning rate. Motivated by a conventional training method where we first train a neural network with a large learning rate and gradually reduce the learning rate, in a similar manner, we train all neural networks at a high speed learning rate and then decrease to a slower learning rate. Though, in our approach, the learning rates are obtained from ranges which are generated randomly rather than deterministic.

6.3 Experiments

In this section, we will discuss in detail our experiments including chosen benchmark dataset, implementation, how we estimate the learning rate range and evaluation metric.

6.3.1 Benchmark Datasets

The Cifar-10 and Cifar-100 [95] are popular and challenging datasets for training deep learning. The two datasets contain 50000 training images and 10000 testing images with the size of 32×32 pixels. As the name suggests, Cifar-10 has exactly 10 categories, i.e., airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck whereas Cifar-100 extends the number of categories from 10 to 100. Having the same total number of training images while increasing the number of categories causes Cifar-100 harder to train since there is less images to see in each category than in Cifar-10. Figure 6.4 depicts a snapshot of random images from the Cifar-10 dataset.

6.3.2 Implementation

In this work, we propose a collaborative training of ConvNets where they share relevant geometric information such as their current gradient and position. In the following subsections, we discuss our

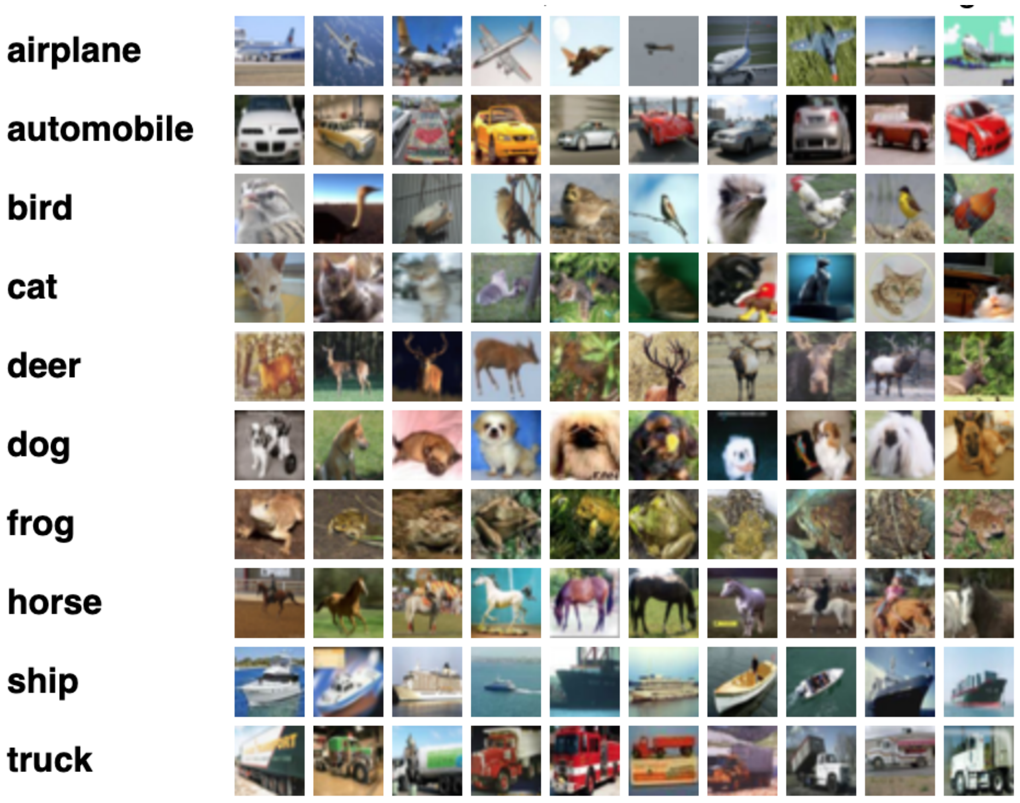


Figure 6.4: A snapshot of samples from Cifar-10 dataset [96].

proposed Parallel PSO ConvNet approach.

Parallel PSO ConvNets

A crucial aspect of the implementation is to create a distributed environment where particles (ConvNets) cooperate with each other. Figure 6.5 illustrates the design. Typically, ConvNets are often trained using just one local computer or using a remote server. With multiple GPUs, several ConvNets can be trained simultaneously, though, training is performed independently.

We build our PSO-ConvNet design around a web client-server architecture a collaborative training. At the center of the design lies dedicated server which hosts entirely the ecosystem including software stack, modern GPUs (GeForce® GTX 1080 Ti), network and application layers. Each client connects to one virtual machine in the server via a specific port, and runs a set of procedures so to yield collaborative training of the ConvNets which rests primarily on sharing their information: current location, previous location, estimate of the gradient of the loss function, among other elements. As illustrated in the Figure 6.5, information from each particle are stored in a database, namely "Score Board". After each epoch, the latest data will be inserted into the database for each particle.

More concretely, as entailed in the dynamics of equation (6.3), particles will share their current personal best position, the best positions of neighbors as well as their estimate of the gradient. We will further compare our approach with a baseline classical PSO (Section 6.4.6) where particles only share their current positions. This will highlight more transparently the importance of sharing the geometric attributes entailed in the dynamics.

ConvNets

We consider topologically distinct ConvNets, namely, shallow and deep ConvNets in the mix (Figure 6.6 shows architecture of Inception-v3 with 42 layers for illustration purposes). When training ConvNets, TL is often utilized in typical approaches. Though, usually faster, the disadvantage of using TL is that the training stage most likely appears to become stuck in the local minima of the loss function and over-fit the models quickly. Therefore, the optimal procedure here is re-training rather than TL.

Accordingly, all layers of the ConvNets are unfrozen so that the models' architecture and the pre-trained weights (from ImageNet dataset [97, 149]) can be reused. Following practices from ResNet, Xception and many others, the top layers are replaced and a global pooling is added to reduce the network size. In addition, for enhancing sample variation and also avoiding over-fitting, a small amount of noise is introduced into the networks.

The re-trained ConvNets architecture is illustrated in Figure 6.7. The re-trained model is a pre-trained ConvNet (e.g.: Inception-v3, VGG and ResNet) with the top layers removed and the weights re-trained. The base model is a pre-loaded one with ImageNet weights. The original images are re-scaled to match the required input size of the re-trained model. The global pooling layer reduces the network size while the Gaussian noise layer prevents over-fitting. The fully connected layer and softmax layer will improve the classification.

6.3.3 Estimate learning rates

For training deep neural networks, the learning rate is one of the most important hyper-parameter as the learning rate decides the amount of gradient to be back propagated or how fast the network moves towards minima. A smaller learning rate requires more training time since only a small change of weights is updated after each epoch whereas a larger learning rate makes training more rapid but also causes more instability. Usually, a shallow method would try out some learning rates and check which one yields a better performance. In contrast, an advanced method would scan from a low to a high value stopping once the Loss increases above a certain threshold (a.k.a learning rate range test) [161].

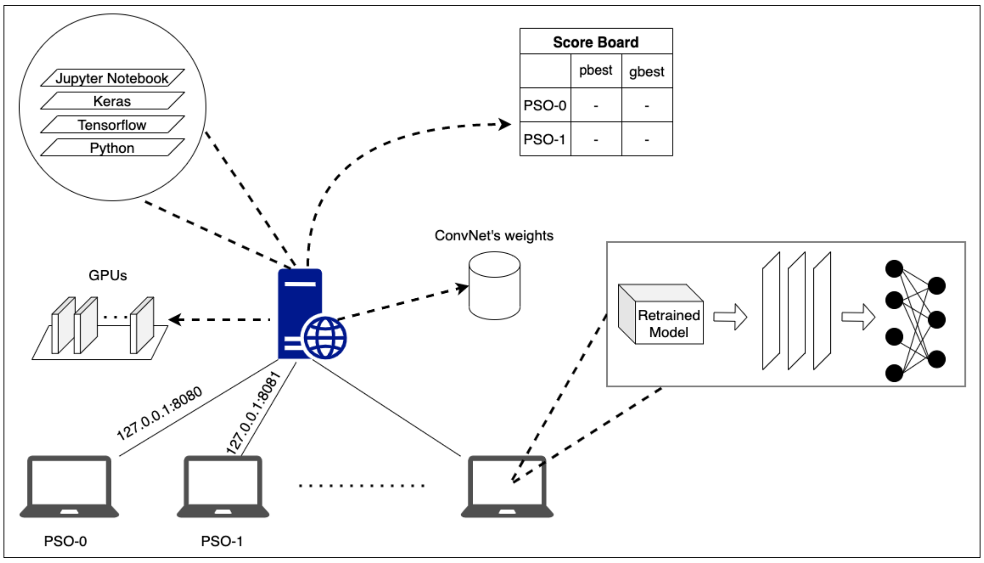


Figure 6.5: Proposed PSO-ConvNets system. PSOs share information with other particles via a file sharing. The information include current location, previous location among the others.

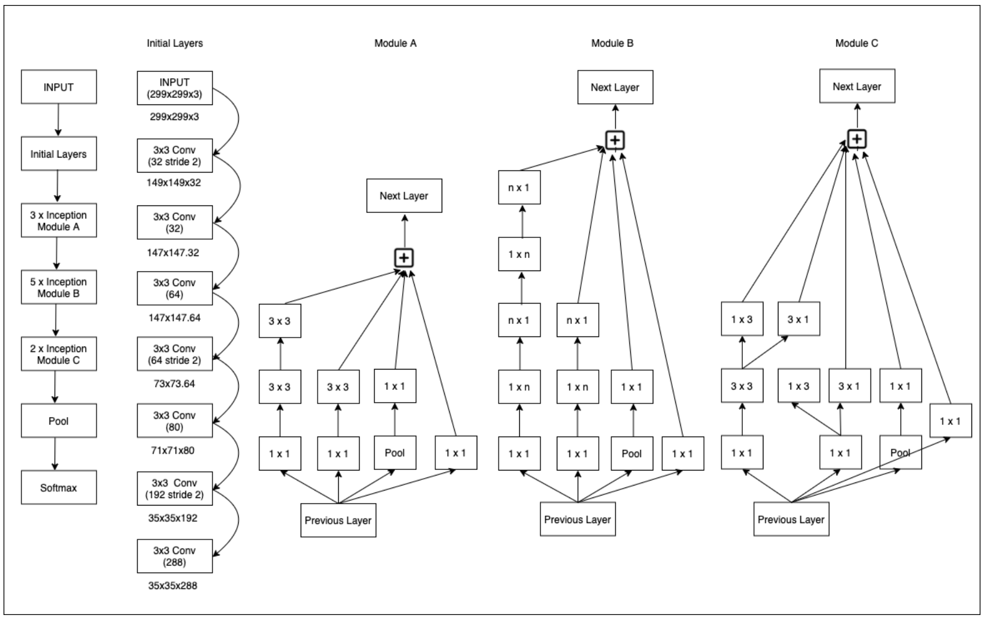


Figure 6.6: Detail of Inception-v3 architecture [169] which contains initial layers and several modules A, B and C. Each module comprises factorized convolutions to reduce the computational cost as it decreases the number of parameters.

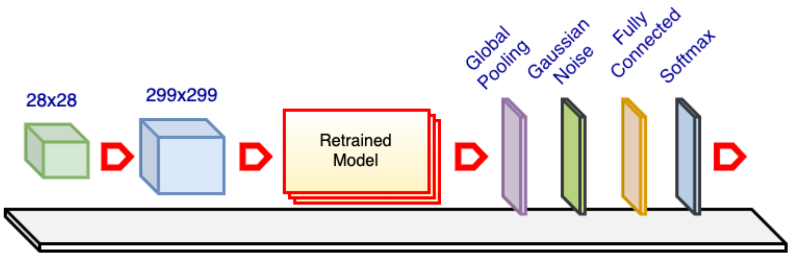


Figure 6.7: Design of re-trained ConvNets architecture. The re-trained model is a pre-trained ConvNet (e.g.: Inception-v3, VGG and ResNet) with the top layers excluded and the weights re-trained. The base model comes pre-loaded with ImageNet weights. The original images are re-scaled as needed to match the required input size of the re-trained model. The global pooling layer reduces the networks size. The Gaussian noise layer improves the variation among samples to prevent over-fitting. The fully connected layer aims to improve classification. The softmax layer is another fully connected layer that has the same number of neurons as the number of dataset categories, and it utilizes softmax activation function.

Figure 6.8 shows results of this learning range test using three different ConvNets namely Inception-v3, EfficientNet-B0 and MobileNet-v1 on Cifar-10 dataset. Though slightly different, we

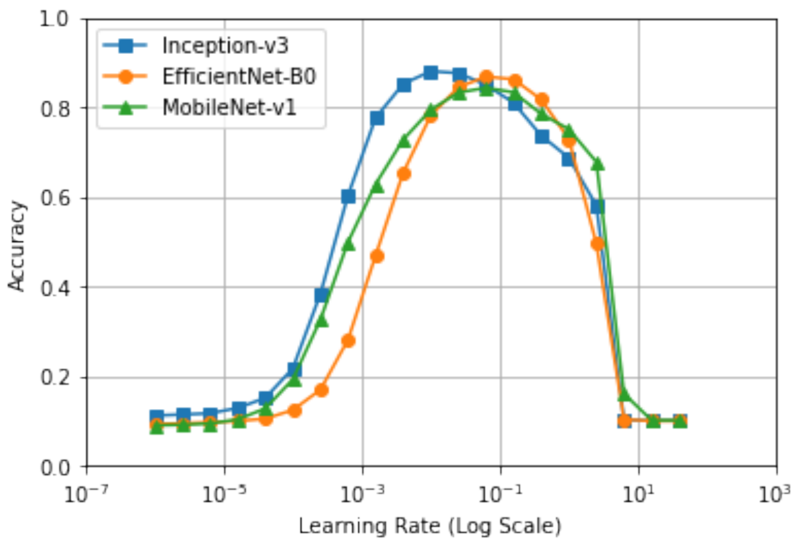


Figure 6.8: Learning rate range scan for Inception-v3, EfficientNet-B0 and MobileNet-v1 on Cifar-10 dataset.

can glean that accuracy drastically increases and decreases at around $[10^{-5}, 10^{-4}]$ and $[10^{-2}, 10^{-1}]$, respectively. Therefore, we set 10^{-5} and 10^{-1} as the minimum and the maximum boundaries for

PSOs with random learning rates.

6.3.4 Evaluation Metric

In this work, we evaluate the visual classification performance of different algorithms using overall accuracy. The metric is popular for the comparison and analysis of results in computer vision tasks. The metric is defined as follows:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total numbers of predictions made}}. \tag{6.9}$$

6.4 Classification Results

In this section, we discuss the results of our proposed hybrid PSO-ConvNet according to equation (6.3) which we call Dynamics 1 and also the approach with the interplay of SGD and PSO according to equation (6.7) which we call Dynamics 2. We refer to as the terms "Dynamics" to emphasize an essential property of the formulas where a particle adjusts its direction towards the average of its neighbors' directions. It means that the particle weighs more nearer particles than those farther away.

6.4.1 Dynamics 1

In this subsection, we discuss the results of our proposed hybrid PSO-ConvNet according to equation (6.3). For easiness of reference to each element in that equation, we identify the first element as **Gradient** since the element mainly relates to vectors of particles; the second element as **Personal best (pBest)** and the last element as **Nearest Neighbor best (KNN)**, just because the elements describe the best locations obtained by individual particles and group of nearby neighbor particles.

We first see the results with the influence from the nearest neighbors, then from the gradients and at last the combination of the two. Further, we will enable the pBest term to see the effects. In the experiments with KNN, we keep the accelerator c_2 as a constant 0.5 and vary the number of nearest neighbors from 1 to 3. In case we have only one nearest neighbor it means that a particle shares information with 1 neighbor ($k = 1$). The same also applies for other numbers of neighbors. In the experiments with Gradient, we select M as 0.1, 1 and 10 while β is set to 1. The results with the above settings are summarized in Table 6.1. Interestingly, it is easy to observe that a combination of KNN and Gradient generates a higher accuracy (approximately 0.9800) than using either KNN or Gradient alone (0.9780).

In Figure 6.9, we graphically show the proposed PSO-ConvNets with Dynamics 1. Similarly, as referred above, the results show higher accuracy with the combination of KNN and Gradient. However, with the inclusion of pBest, it is observed that the accuracy overall decreased. Since adding more elements would cause our experiments to grow exponentially, we decided to discard

Table 6.1: Accuracy on Cifar-10 for the Dynamics 1. Effects of KNN and Gradient are evaluated separately besides combination of the two. Results with pBest are also included. In all experiments, c_2 is kept at 0.5

Method	Accuracy				Index
	PSO-1	PSO-2	PSO-3	PSO-4	
KNN ($k = 1$)	0.9783	0.9717	0.9717	**0.9785**	Ex17_01
KNN ($k = 2$)	0.9715	0.9759	0.9760	0.9769	Ex17_02
KNN ($k = 3$)	0.9713	0.9773	0.9769	0.9765	Ex17_03
Gradient ($M = 10$)	0.9750	**0.9777**	0.9708	0.9752	Ex18_01
Gradient ($M = 1$)	0.9762	0.9730	0.9419	0.9699	Ex18_02
Gradient ($M = 0.1$)	0.9744	0.9738	0.9343	0.9747	Ex18_03
KNN ($k = 1$), Gradient ($M = 10$)	0.9720	0.9755	0.9729	0.9754	Ex19_01
KNN ($k = 1$), Gradient ($M = 1$)	0.9796	0.9707	0.9712	**0.9797**	Ex19_02
KNN ($k = 3$), Gradient ($M = 1$)	0.9738	0.9790	0.9790	0.9787	Ex19_03
KNN ($k = 3$), Gradient ($M = 2$), pBest ($c_1 = 0.5$)	0.9734	0.9783	0.9781	0.9774	Ex13_01
KNN ($k = 3$), Gradient ($M = 2$), pBest ($c_1 = 0.2$)	0.9733	**0.9785**	0.9784	0.9777	Ex13_02

the pBest element. Thus, from here onwards, the Dynamics 1 will include only Gradient and KNN elements and exclude pBest unless stated otherwise.

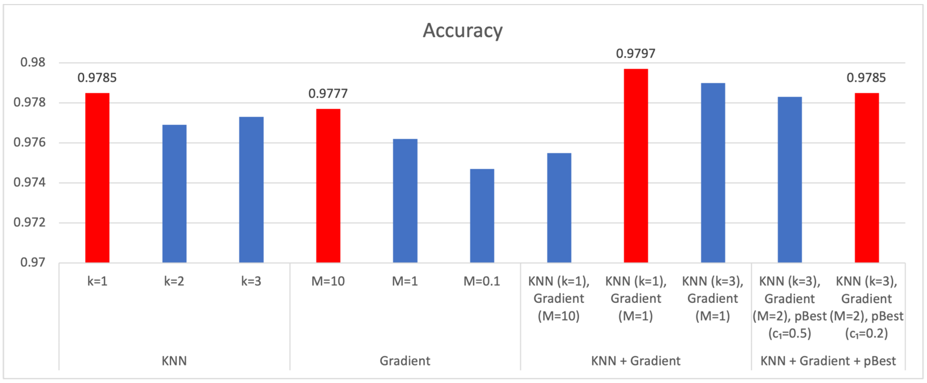


Figure 6.9: Proposed PSO-ConvNets Dynamics 1 (equation (6.3)). Comparison when KNN and Gradient are tested separated and combined, and also with the inclusion of pBest. Each column summaries the best accuracy of all PSOs (PSO-1, PSO-2, etc.). The red column highlights the best result for each group (KNN, Gradient, etc.).

In Figure 6.10, we demonstrates the advantage of random learning rate as at several times the PSO achieves higher accuracy than the nearest neighbor (see the indicated arrows' location). For example, in case of PSO-1 and PSO-4 couple in the upper part of the figure, the PSO-1 appears to be stuck at a local minimum between epoch 5th and 10th, then PSO-4 finds a better accuracy.This occurrence repeats several times during the training even though PSO-4 goes to worse accuracy areas.

Table 6.2: Accuracy of PSOs on variety of M, c_2 and *warmup* parameters

Parameters															Accuracy			
KNN		*warmup*	Gradient															
k	c_2		PSO-1			PSO-2			PSO-3			PSO-4			PSO-1	PSO-2	PSO-3	PSO-4
			M12	M13	M14	M21	M23	M24	M31	M32	M34	M41	M42	M43				
3	1.7	0	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9740	0.9797	0.9798	0.9801
3	1.7	10	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9756	0.9803	0.9798	0.9802
3	1.7	20	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9743	0.9798	0.9801	0.9796
3	1.7	30	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9741	0.9778	0.9772	0.9773
3	1.2	0	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9779	0.9713	0.9783	0.9782
3	1.2	10	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9758	0.9789	0.9789	0.9792
3	1.2	20	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9719	0.9781	0.9778	0.9780
3	1.2	30	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9759	0.9791	0.9793	0.9781
3	0.2	0	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9764	0.9798	0.9798	0.9788
3	0.2	10	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9746	0.9777	0.9781	0.9788
3	0.2	20	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9761	0.9786	0.9795	0.9774
3	0.2	30	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9768	0.9770	0.9771	0.9760
3	0.5	0	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9773	0.9802	0.9800	0.9797
3	0.5	10	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9744	0.9789	0.9792	0.9788
3	0.5	20	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9716	0.9786	0.9787	0.9782
3	0.5	30	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.9758	0.9783	0.9788	0.9777
3	0.5	0	0.2	0.2	10	1.7	0.2	10	1.7	1.7	10	1.7	0.2	0.2	0.9774	0.9814	0.9814	0.9816
3	0.5	10	0.2	0.2	10	1.7	0.2	10	1.7	1.7	10	1.7	0.2	0.2	0.9764	0.9806	0.9811	0.9805
3	0.5	20	0.2	0.2	10	1.7	0.2	10	1.7	1.7	10	1.7	0.2	0.2	0.9770	0.9782	0.9782	0.9775

6.4.2 Optimization of parameter M

We evaluate the performance of the hybrid PSO-ConNets based on the variation of parameter M. In our experiment, rather than fixing a value M for every particle as in [31], the exchanged gradients are properly weighted. Thus, we further sharpen the capability of the algorithm which is not only distinguishing particles based on distance but also taking into account other intrinsic attributes of the particles. In this sense, a particle can be incorporated with a large random learning rate or sometimes called *wilder* particle since the particle can scan the landscape faster and also goes deeper in local minima. For the capability, this particle should be attracted more by other *conservative* particles. Or in other words, particles would be pulled faster toward the gradient directions of the wilder PSOs than to other particles.

Initially, we set the weight M between conservative particles being small numbers (e.g.: 0 and 0.2); in the case of conservative and wilder (with more liberty) particles or among wilder particles we set larger values for M (e.g.: 1.2 and 1.7).

Figure 6.11 shows the results when c_2 and *warmup* are fixed at 0.5 and 0. We recall that accelerator c_2 controls the effect of KNN element in the equation (6.3), i.e., the faster c_2 the more significant the element is. In the meantime, *warmup* indicates at which epoch all particles begin to collaborate. For each experiment, we summary the highest accuracy among PSOs as "Best PSO". Further, we notice that on average the accuracy settles at approximately 0.9790 and the best accuracy rises up to 0.9800.

We then increase the weight towards wilder particles to a much higher value ($M = 10$). The results for variety of c_2 and *warmup* are shown in Table 6.2 and also plotted on Figure 6.12 to

Table 6.3: Settings weight M

	Gradient Weights										
Settings	M12	M13	M14	M21	M23	M24	M31	M32	M34	M42	M43
S1	0.2	0.2	10	1.7	0.2	10	1.7	1.7	10	1.7	0.2
S2	0.2	0.2	1.7	1.7	0.2	1.7	1.7	1.7	1.7	1.7	0.2

facilitate the analysis. On the left-hand side, among variations, c_2 equals to 1.7 and 0.5 accomplish better accuracy than that of 1.2 and 0.2. In addition, pulling particles with distinct weights favors those warmups that start at the beginning of training (at $c_2 = 0.2$ and $c_2 = 0.5$). Remarkably, on the right-hand side where M is greatly increased, we achieve a milestone result with accuracy of 0.9816.

Next, we look at the dynamics of change in distances between PSOs for the best result above as shown in Figure 6.13. We can see that PSO-2, PSO-3 and PSO-4 gradually approach each other (the distances to PSO-1 have similar values). In addition, PSO-1, though also approaches the group, keeps a further distance. It indicates that possessing a larger learning rate (10^{-2} vs. 10^{-3} and 10^{-4}) prevents the PSO from going into steeper minima. Additionally, PSO-4 (random PSO) scans the solution space more broadly as the distances fluctuate wilder, even sometimes move out of the group.

6.4.3 Optimization of parameter β

In this experiment, we attempt to improve the performance of our proposed PSO-ConvNets by changing the parameter β in Dynamics 1. The parameter regulates the rate of decay (in the gradient) which is a distinct feature in comparison with kinetics gradients. In addition, β amplifies the effect of the distance, i.e., setting a smaller value will have a similar effect as when the particles stick together whereas a bigger value will separate the particles.

Figure 6.14 shows results using different β values (0.5, 0.9, 1.1 and 2). As an aside remark, two experimental settings, S1 and S2, are described in Table 6.3. Though, the accuracy is not better than the one found in the previous section, we notice an important qualitative behavior when we look in more detail which is illustrated by inspecting both graphs in Figure 6.15. In fact, we select particle PSO-3 for illustration since its dynamics changes more than the others. In the left graph ($\beta = 0.5$), it appears to have more disturbances than on the right one when $\beta = 2$. At points where PSO-3's loss increases, the particle moves away from the group as the distances from the particle to all the others increase. The behavior does not appear on the right graph. It seems that when particles move closer, a particle is pushed away in the opposite direction. This is the reason why we propose the second dynamics (equation (6.7)) whose results will be discussed in Section 6.4.8.

6.4.4 Number of nearest neighbors

To study the performance of PSO-ConvNets with respect to the number k of nearest neighbors, $k = 1$ and $k = 3$ are applied (we exclude $k = 2$ because in this case, the random PSO will be kept far from the group of the other three and we need the PSO to attract other PSOs). When $k = 1$, a particle obtains information from only one nearest neighbor while when $k = 3$, all the particles cooperate with each other (we recall that we assume a total of four particles in this work). The classification accuracy is illustrated in Figure 6.16. We see that the accuracy is overall higher when $k = 3$, i.e., when all particles exchange information among all the other particles. However, we assume that if the number of PSOs could be scaled up to a much larger number, obtaining data from all neighbors would be more expensive than from just some nearest neighbors. Therefore, in the large-scale setting, meeting a trade off between the number of neighbors and efficiency would be a better strategy.

6.4.5 Neural Networks' baseline

Our proposed PSO-ConvNet approach is comprised of two phases in which the first phase refers to the training of each ConvNet independently and the second phase refers to the collective training of the ConvNets wherein each ConvNet integrates information exchanged by neighboring ConvNets following a PSO inspired approach. In this experiment, we focus on analysis of training ConvNets and leave out PSO phase. As we described in Section 6.3.2, a ConvNet is built on a re-trained model instead of TL model because the training is faced with over-fitting after a short time. In addition, we unfrozen the layers of ConvNets so that we can re-use the model's architecture and weights. Retraining a model from scratch would take more time than re-using weights. In this sense, we compare the performance of a re-trained model (model with unfrozen layers) to a TL model (model with frozen layers). We also call the TL model as a baseline. Furthermore, to enable augmentation for image input, we train the baseline in only one step. This means that an input after feature extraction is continuously becoming the input for the next layers (usually comprises of GAP and FC layers). In TL, training usually is separated into two separated steps which includes feature extraction and fine-tuning. The parameters for this experiment are described in Table 6.4 and can be categorized into two groups, namely, ConvNets and Augmentation. The first one concerns internal settings for ConvNets including the lengths of training, Gaussian noise level and size of FC layer as well as batch size. We set the number of iterations at 40 since the training appears to start over-fitting by that iteration. We also add Gaussian noise as well as carefully choosing the number of neurons in FC layer to reduce over-fitting. The batch size is set at 32 to take full capability of GPUs' memory. The second group controls the augmentation for ConvNets including standard techniques such as rotation range, width shift range, height shift range, shear range and zoom range. Besides, channel shift range, fill mode and preprocessing also makes ConvNets more robust.

Since a standard for Cifar-10 architecture and hyper-parameter settings are not available in Keras

Table 6.4: ConvNets Hyper-parameters

ConvNets	
# of iterations full training	40
Gaussian noise standard deviation	0.1
Fully connected number of neurons	1024
Batch size	32
Augmentation	
Rotation range	30
Width shift range	0.3
Height shift range	0.3
Shear range	0.3
Zoom range	0.3
Channel shift range	10
Fill mode	nearest
Preprocessing	[-1 1]

ecosystem (a famous API to develop deep learning applications) to the best of our knowledge, we decide to build our models based on popular Inception-v3 structure and compare our results with recent state-of-the-art using the same architecture.

First, as we can glean from Figure 6.17, the results of TL models using frozen layers are approximately 0.89. In comparison with recent works, the authors in [3] and [83] reported accuracy of 0.86 and 0.92, respectively. Thus, the differences are in a decent range and we will use the model as a baseline to improve our work. Second, we see that results of unfrozen neural networks (re-trained model) outperform those with frozen layers by a large margin increasing by roughly 8% in settings with and without augmentations.

6.4.6 PSO gBest

We next test a PSOs using global best approach (namely, PSO gBest) similar to Dynamics 1. While the training is still intertwined between PSO and SGD but the differences are that (i) the gradient element is excluded and (ii) instead of updating particle's location toward a nearest best neighbor's location, particles in gBest will be updated toward the global best as in the following formula.

$$v^{(n)}(t+1) = v^{(n)}(t) + cr(t)\left(P^{(n)}_{gBest}(t) - \phi^{(n)}(t+1)\right) \tag{6.10}$$

where $v^{(n)}(t) \in \mathbb{R}^D$ is the velocity vector of particle n at time t; $r(t) \overset{i.i.d.}{\sim} \mathsf{Uniform}([0,1])$ is randomly drawn from the interval $[0,1]$ and we assume that the sequence $r(0)$, $r(1)$, $r(2)$, $\ldots$ is i.i.d.; $P^{(n)}_{gBest}(t)$ represents its global best, i.e., the best position across all previous positions of all particles up until time t.

In these experiments, we set c as a constant with a value of 0.5. Besides, ConvNets cooperate from the beginning and at iterations 10th, 20th and 30th. We also utilize two distinct ConvNets' architectures, i.e., Inception-v3 and EfficientNet-B0 for comparison. The results are shown in Figure 6.18. Overall, Inception-v3 performs slightly better than its counterpart, e.g., particle PSO-4 in Inception-v3 structure achieves an accuracy of approximately 0.9799 when $warmup = 30$ while the latter obtains a smaller value (0.9789). It is also interesting to notice that delaying cooperation at a later time mostly brings accuracy to a higher value. This differs from the results in Section 6.4.2 where collaborating earlier is generally better. One explanation is that, in Dynamics 1, a particle is attracted due to more causal effects, e.g., direction to the best location, directions of other particles so the particle spans more landscape in finding solutions, thus, training earlier is essential.

6.4.7 Accelerator coefficient

We try to find a connection between accelerators c among PSOs and report accuracy of PSO gBest method in Figure 6.19. Due to exponential growth in number of experiments, we evaluate only two PSOs rather than using all four PSOs. We choose PSO-1 and PSO-3 since these PSOs are set at fastest and slowest learning rates (PSO-4 is excluded because of instability). We also select the ConvNets' architecture EfficientNet-B0 because its network size is smaller, i.e., each neural network has about five million total number of parameters in comparison with twenty four millions that of Inception-v3. According to the learning rate range scan in Figure 6.8, the speed needs to be faster. Thus, we change PSO-1, PSO-2 and PSO-3's learning rates to 10^{-1}, 10^{-2} and 10^{-3}, respectively. In the same manner, the range for PSO-4 is also moved to between 10^{-1} and 10^{-5}. We can observe that, for certain settings, e.g., when PSO-1's accelerator equals to 1.7, the overall accuracy seems to be reduced and when the value is 0.5, the accuracy appears to be increased.

6.4.8 Additional strategies for improvement

Multi-wilder particles

Now, we consider particles with larger random learning rate (*wilder* particles) and particles with fixed learning rates (*conservative* particles). In this experiment, the number of wilder particles expands to more than just one. Thus, we expect that more wilder particles would scan more the solution space, consequently find better minima and to improve overall performance.

As we can see from Figure 6.20, in the case of two wilder particles, a combination of two conservative PSO-1 and PSO-2 with larger learning rates (10^{-2}, 10^{-3}) obtains a higher accuracy than the other options.

Regarding three wilder particles and one conservative particle, when PSO-2 is conservative (blue column), the group accomplishes a better performance than all the others in both settings. The rationale is that the conservative one provides an *essential direction* for all the other three. In case the learning rate is small (10^{-4}), the particle slowly explores the solution space that affects the performance of the whole group. On the other hand, when the learning rate is higher (10^{-2}), the particle explores faster but may skip deeper minima, just discarding opportunities to improve accuracy. In other words, a conservative needs neither training slow nor fast to provide safer solutions at the times when other particles scan the landscape wildly. Therefore, conservative particle also plays an important role to the swarm even wilder particles are the key for improvement.

In general, experiments with three random learning rate outperform the experiments with two random learning rate (0.9816 versus 0.9811).

Cluster Warmup Learning Rate

The results for these experiments are shown on Figure 6.21. Analogous to the typical training where the learning rate for a ConvNet is gradually reduced, we also setup a cluster of particles where learning rate range is reduced from the range $[10^{-3}, 10^{-1}]$ to the range $[10^{-5}, 10^{-1}]$ at distinct iterations, i.e., three PSOs have the random learning rate range changing and PSO-2 is fixed at 10^{-3} (the best setting obtained in the previous Section). Among choices, reducing learning rate after 30 iterations of a total of 40 yields a better accuracy (0.9815) despite slightly lower than the best result obtained before (see Section 6.4.2).

Dynamics 2

In Section 6.4.3, we have learnt that for some instances when a particle is coming closer to others, the particle tends to be pulled away. Theoretically, we would expect that, because of the effect of PSO algorithm, after a while particles will eventually stick together. Thus, these results seem to contradict our initial assumption (see equation (6.3), and therefore, we propose a modification to the algorithm so that the particle will be pulled back instead. As such the new formulation is stated in equation (6.7) which we call Dynamics 2.

We test the proposed algorithm in conjunction with strategies from previous sections and compare the performance of Dynamics 2 versus Dynamics 1. Figure 6.22 shows results of this comparison. We can see that, in most cases, Dynamics 2 not only outperforms the former, but, remarkably, achieved state-of-the-art result when the accuracy is improved to 0.9831. This proves that the notion of pulling back particle is a critical mechanism for improvement.

6.4.9 Performance on Cifar-100 benchmark dataset

For a better evaluation of our proposed methods, apart from Cifar-10, we also performed experiments on the Cifar-100 benchmark dataset. In fact, as an example, we illustrated in Table 6.5 the accuracy is increased by 2.78% with Cifar-100 for Dynamics 2 with MobileNet confirming our work claim.

These experiments were performed by running learning rate scans on Inception-v3 and MobileNet and the results indicate similar parameter ranges as in Cifar-10. For other settings, we reuse the best hyper-parameters which were found in the previous experiments. Even though, different datasets might require distinct settings but this strategy saves time using a rational guess. The results are illustrated in Figure 6.23 using the retrained architectures. The experiments confirm the effectiveness of our methods.

Table 6.5: Performance on Cifar-100. The methods ConvNet TL (Transfer Learning), ConvNet+Aug (Augmentation), PSO gBest, Dynamics 1 and Dynamics 2 are evaluated on retrained architectures Inception-v3 and MobileNet. The ConvNet+Aug method is utilized as a baseline. The most accuracy among PSOs is selected to compute the percentage increase.

Architecture	Method	Accuracy				Increase (%)
		PSO-1	PSO-2	PSO-3	PSO-4	
Inception-v3	ConvNet TL	0.7014	0.6714	0.5309	0.7016	-14.54
	ConvNet+Aug	0.8454	0.847	0.6725	0.8428	0
	PSO gBest	0.8471	0.8719	0.865	0.8676	+2.49
	Dynamics 1	0.8729	0.8736	0.8744	0.8748	+2.78
	Dynamics 2	0.8742	0.8709	0.8724	0.8734	+2.72
MobileNet	ConvNet TL	0.7076	0.6951	0.6786	0.6988	-8.93
	ConvNet+Aug	0.7791	0.7969	0.7693	0.7951	0
	PSO gBest	0.7699	0.8119	0.8106	0.8123	+1.54
	Dynamics 1	0.8216	0.813	0.8222	0.8215	+2.53
	Dynamics 2	0.8243	0.8158	0.8246	0.8247	**+2.78**

6.5 Comparison with state-of-the-art methods

The comparison results between the proposed method and other state-of-the-art counterpart algorithms are displayed in Table 6.6. A set of distinct methods were collected for a fair comparison. The results clearly show our algorithm outperforms other benchmark methods. We can see that ConvNets+Aug (fourth row counting from the bottom) performs 2.5% and 0.75% better than CLR and AmoebaNet in terms of accuracy on Cifar-10 dataset. In addition, this baseline method is much better than K-means and GA-CNN. Moreover, Tiled CNN method was achieved a lowest classification accuracy (73.1%). Though, PyramidNet AMP outperforms our PSO gBest (97.99%), the architecture is outweighed by Dynamics 1. Similarly, ResNet52-SAM performs better than Dynamics 1 but is surpassed by Dynamics 2 which yields 98.31% accuracy. In the same manner, our proposed methods achieve the highest accuracy on Cifar-100 benchmark dataset, thus, confirming its superiority.

Table 6.6: Classification accuracy (%) on the Cifar-10 and Cifar-100 benchmark datasets.

Method	Accuracy (%)	
	Cifar-10	Cifar-100
Tiled CNN [129]	73.10	-
K-means [27]	79.64	-
GA-CNN [206]	74.59	-
cPSO-CNN [193]	91.35	-
CLR [161]	94.90	75.90
AmoebaNet-A (N=6, F=36) [145]	96.66	-
ENAS [137]	97.11	-
SENet [73]	97.88	84.59
PyramidNet AMP [219]	98.02	86.64
ViT-B/16 [41]	98.13	87.13
ResNet152-SAM [19]	98.20	87.08
ConvNet+Aug	97.41	84.7
PSO gBest	97.99	87.19
Dynamics 1	98.16	**87.48**
Dynamics 2	**98.31**	87.42

The bold letter indicates that the proposed algorithm achieved best efficiency amongst the compared algorithms

6.6 Conclusions and Contributions

Image recognition has been a gold standard approach for many computer vision related tasks: extracting relevant information from telescope images in astronomy, navigation in robotics, cancer classification in medical image etc. However, training these large-scale neural networks for generalization is still a non-trivial task since the performance is sensitive to the architecture, the training set, the sample size among other attributes that renders the problem quite unstable to tuning.

In this article, we have proposed a novel distributed collaborative PSO-ConvNet algorithm for the optimization performance which is capable of leading particles up to a better minimum. The key contributions of this article are: (1) novel formulations (Dynamics 1 and Dynamics 2) have been successfully created by incorporating distilled Cucker-Smale elements into the PSO algorithm using KNN and intertwining the training with SGD; (2) a new type of particle, i.e., wilder PSO with random learning rate is introduced which has capability of attracting conservative PSOs to stronger minima; (3) a distributed environment is developed for parallel collaboration that significantly accelerates the training; (4) the proposed algorithms are evaluated on Cifar-10 and Cifar-100 benchmark datasets and compared to state-of-the-art algorithms to verify the superior effectiveness. In the future, we will explore our algorithms on more datasets, e.g., ImageNet. In addition, we intend to incorporate our approach for action recognition.

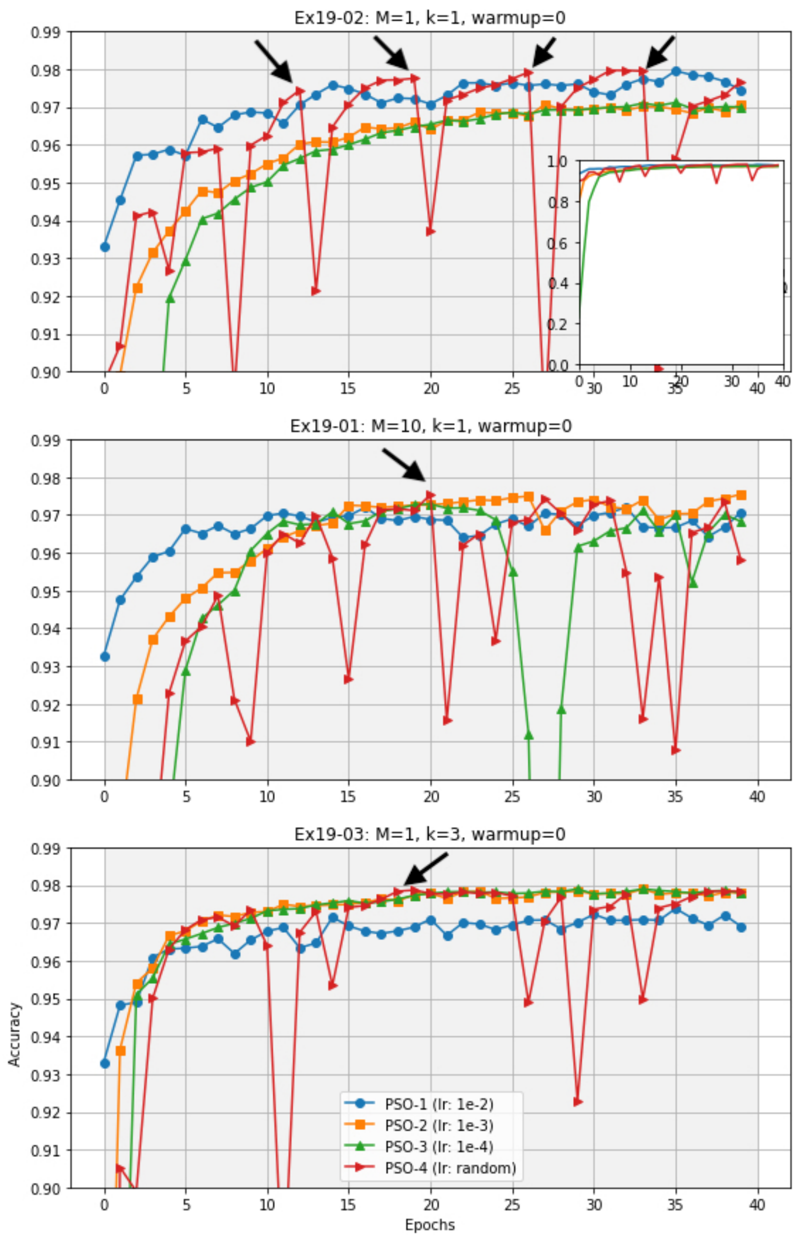


Figure 6.10: Accuracy during training of Hybrid PSO-ConvNets. Examples when $M = 1$ and $M = 10$, warmup $= 0$ and on variation of k ($k = 1$ and $k = 3$). Learning rates of PSO-1, PSO-2 and PSO-3 are set at 10^{-2}, 10^{-3} and 10^{-4} while PSO-4's learning rate is random in the range $[10^{-5}, 10^{-1}]$. The arrows indicate where PSO-4 finds better locations. The zoom in figure displays the accuracy in full scale whereas the others show a narrower scale for better view in detail.

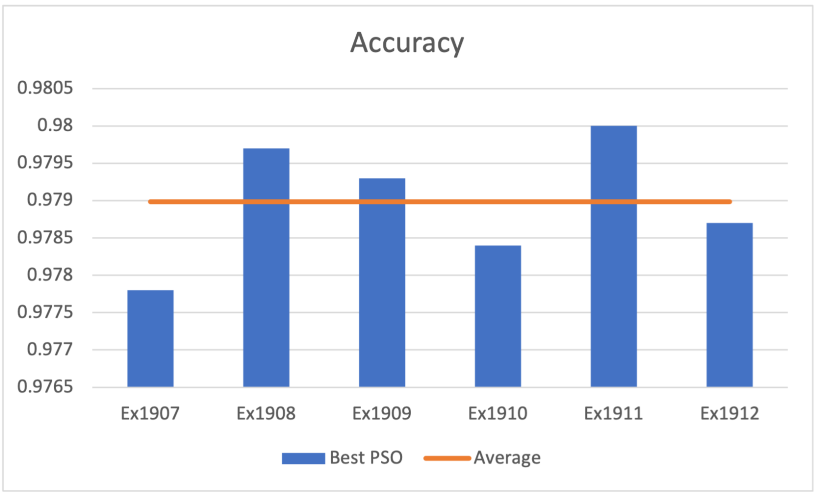


Figure 6.11: Results when c_2 and *warmup* are fixed at 0.5 and 0, respectively.

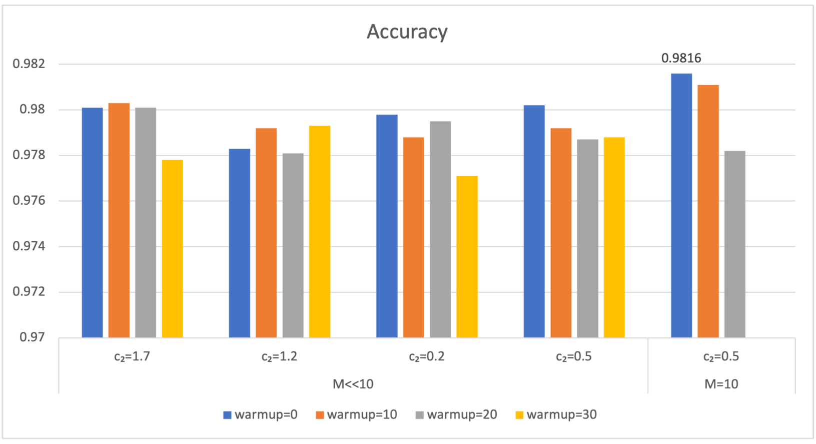


Figure 6.12: Results on variation of c_2 and *warmup*. The weight M from all particles toward wilder particles is set at a higher value. The best accuracy is recorded for all PSOs in each experiment.

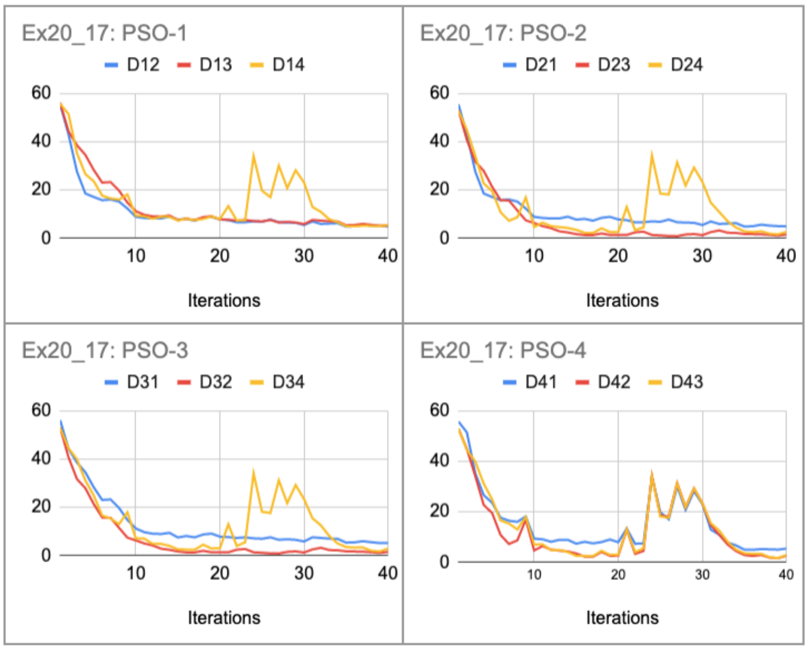


Figure 6.13: Distances between PSOs. D_{ij} denotes the distance between particle i and particle j. Learning rates of PSO-1, PSO-2 and PSO-3 are set at 10^{-2}, 10^{-3} and 10^{-4} while PSO-4's learning rate is random in the range $[10^{-5}, 10^{-1}]$.

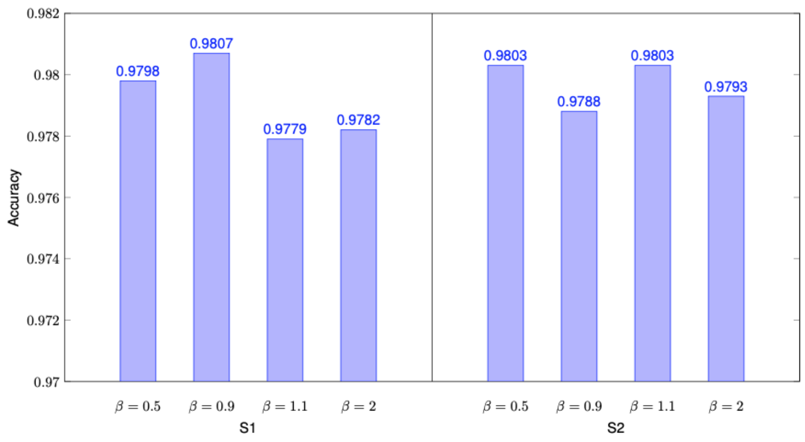


Figure 6.14: Analysis of β. The best accuracy for all PSOs are recorded for each experiment. Besides, k, c_2 and *warmup* are set at 3, 0.5 and 0, respectively.

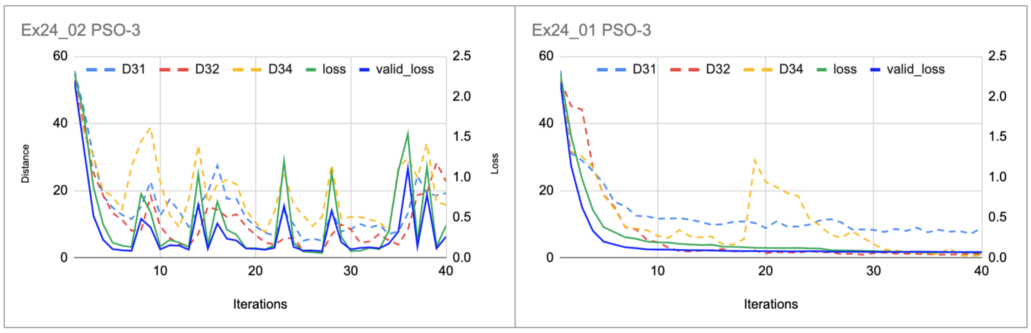


Figure 6.15: Effects of β to distances between PSOs for $\beta = 0.5$ and $\beta = 2$.

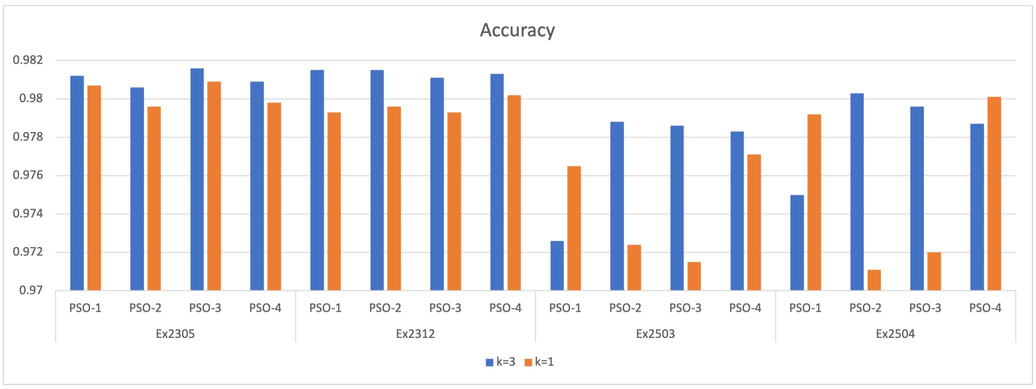


Figure 6.16: Comparison accuracy performance when number of nearest neighbors $k = 3$ and $k = 1$.

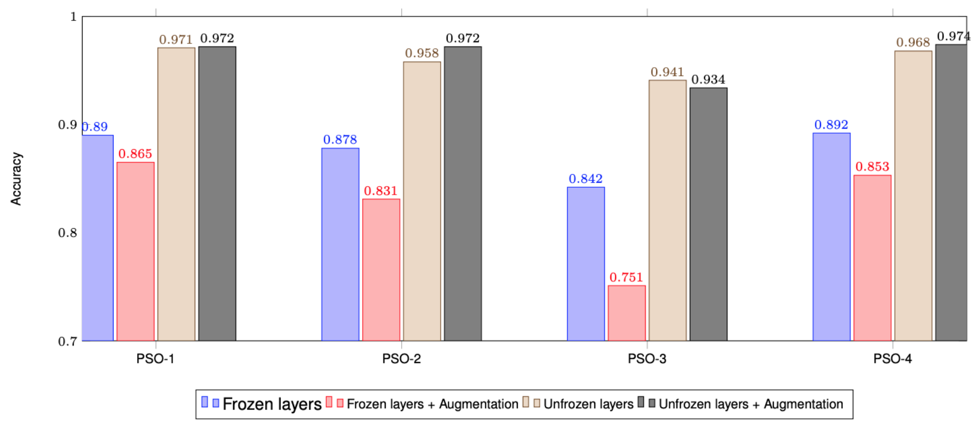


Figure 6.17: Results of ConvNets when the main layers of networks are frozen and unfrozen along with augmentation. Learning rates of PSO-1, PSO-2 and PSO-3 are set at 10^{-2}, 10^{-3} and 10^{-4} while PSO-4's learning rate is random in the range $[10^{-5}, 10^{-1}]$. ConvNets are not collaborated in this experiment.

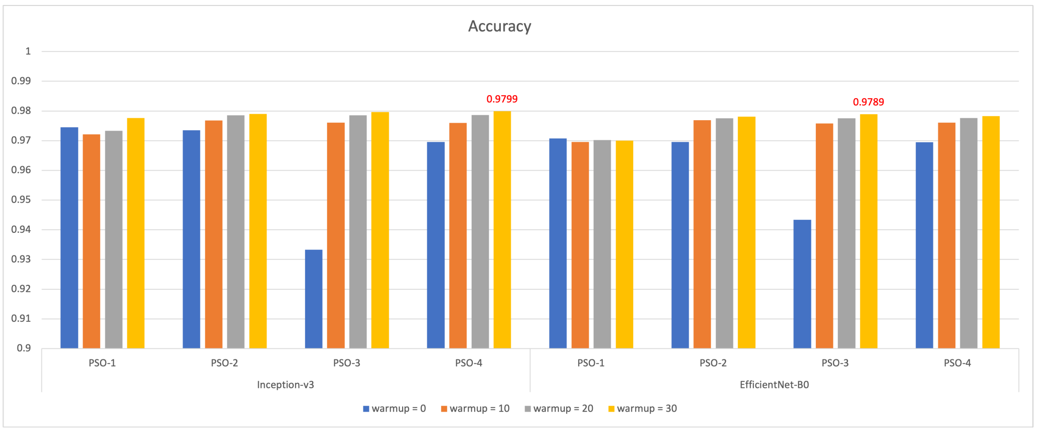


Figure 6.18: PSO gBest. Performance of Inception-v3 and EfficientNet-B0 on several values of *warmup*.

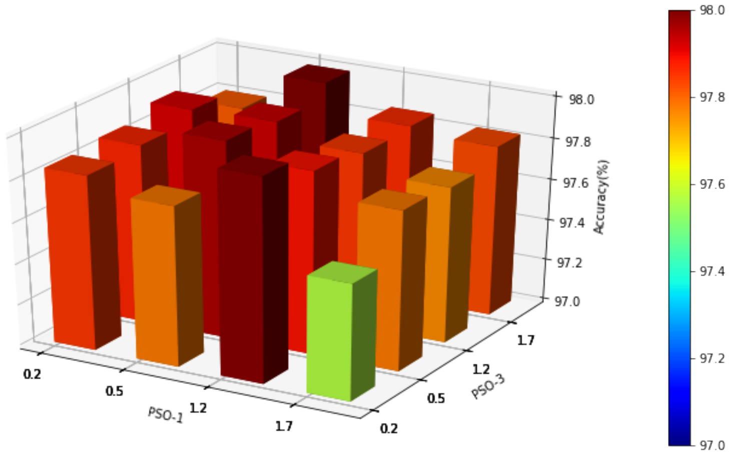


Figure 6.19: Evaluation of PSO's accelerator. The results are obtained from a re-trained EfficientNet-B0 model using gBest approach. The accelerators of PSO-2 and PSO-4 are fixed at 0.5 while PSO-1 and PSO-3 vary.

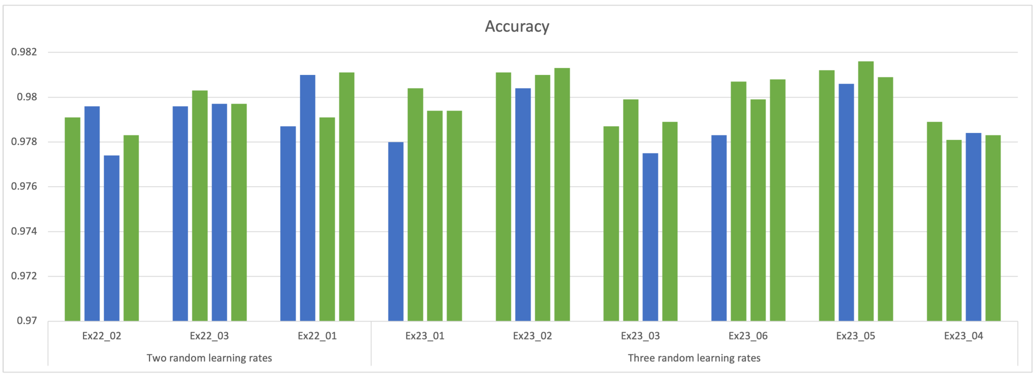


Figure 6.20: Multi-random learning rates. The accuracy for experiments with two and three random particles. The latter is tested in two different settings. In each experiment, the columns from left to right hand side indicate PSO-1, PSO-2, PSO-3 and PSO-4 in order. When PSO-1, PSO-2 and PSO-3 are conservative the learning rates are set at 10^{-2}, 10^{-3} and 10^{-4}, respectively. The PSO-4's learning rate is always in a random range. Green columns denote random PSOs whereas blue columns represent conservative ones.

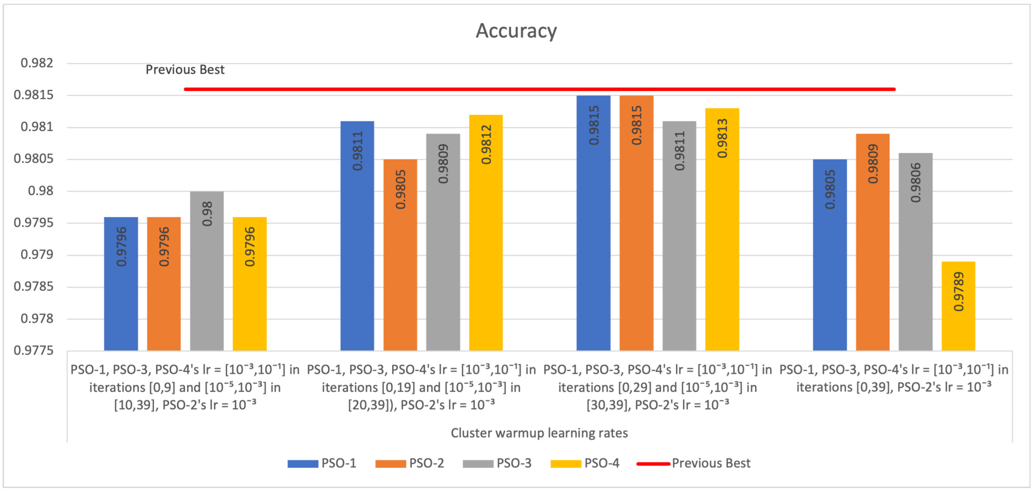


Figure 6.21: Results for cluster warmup learning rates.

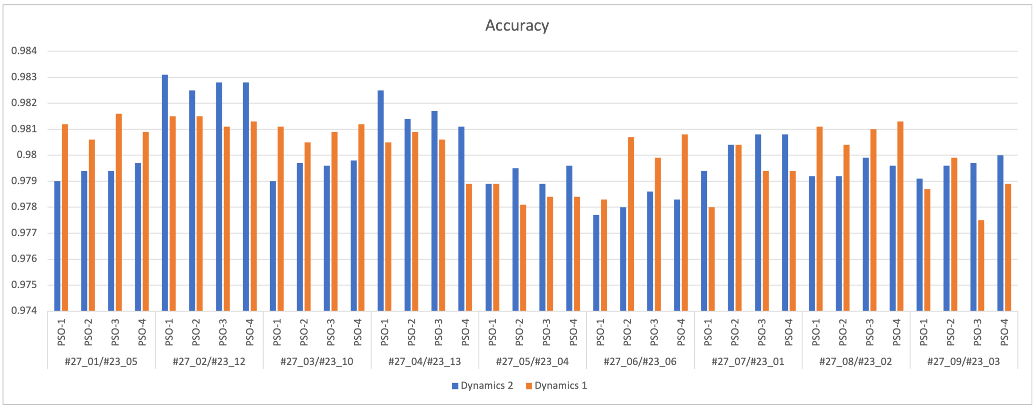


Figure 6.22: Comparison of accuracy performance between Dynamics 1 and Dynamics 2. The latter outperforms the former and the best accuracy is further improved.

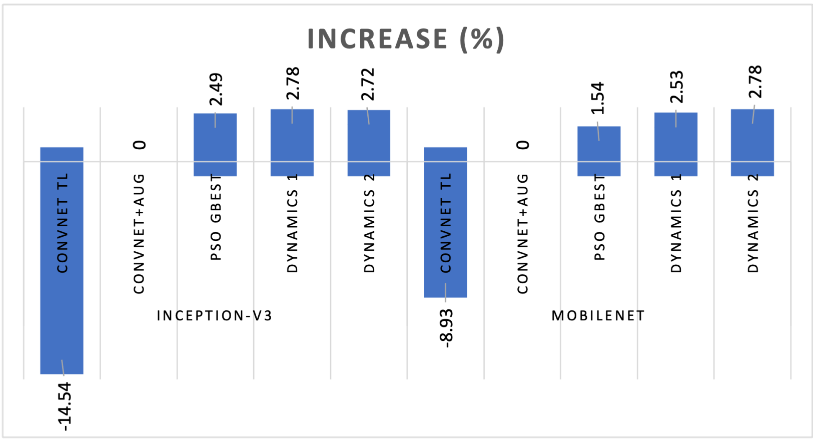


Figure 6.23: Percentage increase of the proposed methods on Cifar-100.

Chapter 7

PSO-ConvNet Transformer

7.1 Introduction

Human Action Recognition plays a vital role for distinguishing a particular behavior of interest in the video. It has critical applications including visual surveillance for detection of suspicious human activities to prevent the fatal accidents [167, 107], automation-based driving to sense and predict human behavior for safe navigation [144, 201]. In addition, there are large amount of non-trivial applications such as human-machine interaction [141, 140], video retrieval [222], crowd scene analysis [32] and identity recognition [134].

In the early days, the majority of research in Human Activity Recognition was conducted using hand-crafted methods [189, 190, 52]. However, as deep learning technology evolved and gained increasing recognition in the research community, a multitude of new techniques have been proposed, achieving remarkable results.

Action recognition preserves a similar property of image recognition since both of the fields handle visual contents. In addition, action recognition classifies not only still images but also dynamics temporal information from the sequence of images. Built on these intrinsic characteristics, action recognition's methods can be grouped into two main approaches namely RNN based approach and 3-D ConvNet based approach. Besides of the main ones, there are other methods that utilize the content from both spatial and temporal and coined the name two-stream 2-D ConvNet based approach [159].

Initially, action recognition was viewed as a natural extension of image recognition, and spatial features from still frames could be extracted using ConvNet, which is one of the most efficient techniques in the image recognition field. However, traditional ConvNets are only capable of processing a single 2-D image at a time. To expand to multiple 2-D images, the neural network architecture needs to be re-designed, including adding an extra dimension to operations such as convolution and pooling to accommodate 3-D images. Examples of such techniques include C3D [177], I3D [15], R3D [62], S3D [197], T3D [37], LTC [182], among others

Similarly, since a video primarily consists of a temporal sequence, techniques for sequential data, such as RNNs and specifically LSTM, can be utilized to analyze the temporal information. Despite the larger size of images, feature extraction is often employed. Long-term Recurrent Convolutional Networks (LRCN)[38] and Beyond-Short-Snippets[208] were among the first attempts to extract feature maps from 2-D ConvNets and integrate them with LSTMs to make video predictions. Other works have adopted bi-directional LSTMs [179, 64], which are composed of two separate LSTMs, to explore both forward and backward temporal information.

To further improve performance, other researchers argue that videos usually contain repetitive frames or even hard-to-classify ones which makes the computation expensive. By selecting relevant frames, it can help to improve action recognition performance both in terms of efficiency and accuracy [54]. A similar concept based on attention mechanisms is the main focus in recent researches to boost overall performance of the ConvNet-LSTM frameworks [47, 194].

While RNNs are superior in the field, they process data sequentially, meaning that information flows from one state to the next, hindering the ability to speed up training in parallel and causing the architectures to become larger in size. These issues limit the application of RNNs to longer sequences. In light of these challenges, a new approach, the Transformer, emerged [184, 176, 41, 7, 118].

There has been a rapid advancement in action recognition in recent years, from 3-D ConvNets to 2-D ConvNets-LSTM, two-stream ConvNets, and more recently, Transformers. While these advancements have brought many benefits, they have also created a critical issue as previous techniques are unable to keep up with the rapidly changing pace. Although techniques such as evolutionary computation offer a crucial mechanism for architecture search in image recognition, and swarm intelligence provides a straightforward method to improve performance, they remain largely unexplored in the realm of action recognition.

In our recent research [139], we developed a dynamic PSO framework for image classification. In this framework, each particle navigates the landscape, exchanging information with neighboring particles about its current estimate of the geometry (such as the gradient of the Loss function) and its position. The overall goal of this framework is to create a distributed, collaborative algorithm that improves the optimization performance by guiding some of the particles up to the best minimum of the loss function. We extend this framework to action recognition by incorporating state-of-the-art methods for temporal data (Transformer and RNN) with the ConvNet module in an end-to-end training setup.

7.2 Proposed Methods

7.2.1 Collaborative Dynamic Neural Networks

Define $\mathcal{N}(n,t)$ as the set of k nearest neighbor particles of particle n at time t, where $k \in \mathbb{N}$ is some predefined number. In particular,

$$\mathcal{N}(n,t) = \{(x^{(n)}(t), v^{(n)}(t)), (x^{(i_1)}(t), v^{(i_1)}(t)), (x^{(i_2)}, v^{(i_2)})(t), \ldots, (x^{(i_k)}(t), v^{(i_k)}(t))\} \tag{7.1}$$

where i_1, i_2,... i_k are the k closest particles to n and $x^{(i_k)}(t)$ and $v^{(i_k)}(t) \in \mathbb{R}^D$ represent the position and velocity of particle i_k at time t. Figure 7.1 illustrates this concept for $k = 4$ particles.

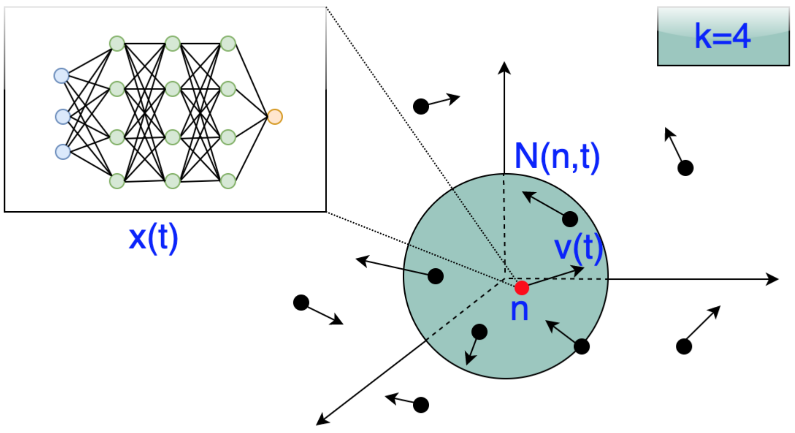


Figure 7.1: A demonstration of the $\mathcal{N}(n,t)$ neighborhood, consisting of the positions of four closest particles and particle n itself, is shown. The velocities of the particles are depicted by arrows.

Given a (continuous) function $L : \mathbb{R}^D \longrightarrow \mathbb{R}$ and a (compact) subset $S \subset \mathbb{R}^D$, define

$$\mathcal{Y} = \mathsf{argmin}\,\{L(y) \,:\, y \in S\} \tag{7.2}$$

as the subset of points that minimize L in S, i.e., $L(z) \leq L(w)$ for any $z \in \mathcal{Y} \subset S$ and $w \in S$. **Dynamics 1:** We investigate a set of neural networks that work together in a decentralized manner to minimize a Loss function L. The training process is comprised of two phases: i) individual training of each neural network using (stochastic) gradient descent, and ii) a combined phase of SGD and PSO-based cooperation. The weight vector of each neural network is represented as the position of a particle in a D-dimensional phase space, where D is the number of weights. The evolution of the particles (or neural networks) is governed by equation (7.3), with the update rule specified by the following dynamics:

$$\begin{aligned}
\psi^{(n)}(t+1) &= -\eta \nabla L\left(x^{(n)}(t)\right) \\
\phi^{(n)}(t+1) &= x^{(n)}(t) + \psi^{(n)}(t+1) \\
v^{(n)}(t+1) &= \sum_{\ell \in \mathcal{N}(n,t)} w_{n\ell}\psi^{(\ell)}(t+1) + c_1 r(t)\left(P^{(n)}(t) - \phi^{(n)}(t+1)\right) + c_2 r(t)\left(P_g^{(n)}(t) - \phi^{(n)}(t+1)\right) \\
x^{(n)}(t+1) &= x^{(n)}(t) + v^{(n)}(t)
\end{aligned} \tag{7.3}$$

where $v^{(n)}(t) \in \mathbb{R}^D$ is the velocity vector of particle n at time t; $\psi^{(n)}(t)$ is an intermediate velocity computed from the gradient of the Loss function at $x^{(n)}(t)$; $\phi^{(n)}(t)$ is the intermediate position computed from the intermediate velocity $\psi^{(n)}(t)$; $r(t) \overset{i.i.d.}{\sim} \mathsf{Uniform}\left([0,1]\right)$ is randomly drawn from the interval $[0,1]$ and we assume that the sequence $r(0)$, $r(1)$, $r(2)$, $\ldots$ is i.i.d.; $P^{(n)}(t) \in \mathbb{R}^D$ represents the *best* position visited up until time t by particle n, i.e., the position with the minimum value of the Loss function over all previous positions $x^{(n)}(0)$, $x^{(n)}(1)$, $\ldots$, $x^{(n)}(t)$; $P_g^{(n)}(t)$ represents its nearest-neighbors' counterpart, i.e., the best position across all previous positions of the particle n jointly with its corresponding nearest-neighbors $\bigcup_{s \le t} \mathcal{N}(n,s)$ up until time t:

$$\begin{aligned}
P^{(n)}(t+1) &\in \mathsf{argmin}\left\{L(y) \,:\, y = P^{(n)}(t), x^{(n)}(t)\right\} \\
P_g^{(n)}(t+1) &\in \mathsf{argmin}\Big\{L(y) \,:\, y = P_g^{(n)}(t), x^{(k)}(t); \; . \\
&\quad k \in \mathcal{N}(n,t)\}
\end{aligned} \tag{7.4}$$

The weights $w_{n\ell}$ are defined as

$$w_{n\ell} = f\left(\left|\left|x^{(n)}(t) - x^{(\ell)}(t)\right|\right|\right), \tag{7.5}$$

with $||\cdot||$ being the Euclidean norm and $f : \mathbb{R} \to \mathbb{R}$ being a decreasing (or at least non-increasing) function. In Dynamic 1, we assume that

$$f(z) = \frac{M}{(1+z)^{\beta}}, \tag{7.6}$$

for some constants $M, \beta > 0$. This *strengthens* the collaboration learning between any of two particles by pushing each particle against each other.

Dynamics 2: An alternative to equation (7.3) is to pull back a particle instead of pushing it in the direction of the gradient. In the previous section, the assumption was that all particles were located on the same side of a valley in the loss function. However, if one particle is on the opposite side of the valley relative to the rest of the particles, it will be pulled further away from the minimum using the first dynamic. To address this issue, we introduce a second dynamic (Dynamics 2) that pulls the particle back. The formula for this dynamics is as follows:

$$x_{(i)}(t+1) \quad = \quad x_{(i)}(t) + \sum_{j=1}^{N} \frac{M_{ij}}{(1+||x_i(t)-x_j(t)||^2)^{\beta}} (x_j(t) - \nabla L(x_j(t))) + cr\left(P_{nbest(i)}(t) - x_i(t)\right) \tag{7.7}$$

where $x_{(i)}(t) \in \mathbb{R}^D$ is the position of particle i at time t; M, β and c are constants set up by experiments with $||\cdot||$ being the Euclidean norm; $r(t) \overset{i.i.d.}{\sim} \mathsf{Uniform}\left([0,1]\right)$ randomly drawn from the interval $[0,1]$ and we assume that the sequence $r(0)$, $r(1)$, $r(2)$, $\ldots$ is i.i.d.; $P_{nbest(i)}(t) \in \mathbb{R}^D$ represents nearest-neighbors' best , i.e., the best position across all previous positions of the particle n jointly with its corresponding nearest-neighbors $\bigcup_{s\leq t} \mathcal{N}\left(n,s\right)$ up until time t.

7.2.2 ConvNet Transformer Architecture for Action Recognition

In this section, we discuss a hybrid ConvNet-Transformer architecture that replaces the traditional ConvNet-RNN block for temporal input to classify human action in videos.

The architecture is composed of several components, including a feature extraction module using ConvNet, a position embedding layer, multiple transformer encoder blocks, and classification and aggregation modules. The overall diagram of the architecture can be seen in Figure 7.2. The goal of the architecture is to effectively capture the temporal information present in the video sequences, in order to perform accurate human action recognition. The hybrid ConvNet-Transformer design leverages the strengths of both ConvNets and Transformers, offering a powerful solution for this challenging task.

Features Extraction via ConvNet and Position Embedding

In the early days of using Transformer for visual classification, especially for images, the frames were typically divided into smaller patches and used as the primary input [118, 117, 217]. However, these features were often quite large, leading to high computational requirements for the Transformer. To balance efficiency and accuracy, ConvNet can be utilized to extract crucial features from images, reducing the size of the input without sacrificing performance.

We assume that, for each frame, the extracted features from ConNet have a size of (w, h, c) where w and h are the width and height of a 2D feature and c is the number of filters. To further reduce the size of the features, global average pooling is applied, reducing the size from $w \times h \times c$ to c.

The position encoding mechanism in Transformer is used to encode the position of each frame in the sequence. The position encoding vector, which has the same size as the feature, is summed with the feature and its values are computed using the following formulas. This differs from the sequential processing of data in the RNN block, allowing for parallel handling of all entities in the sequence.

$$
\begin{aligned}
PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\
PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}})
\end{aligned}
\tag{7.8}
$$

where pos, i and PE are the time step index of the input vector, the dimension and the positional encoding matrix; d_{model} refers to the length of the position encoding vector.

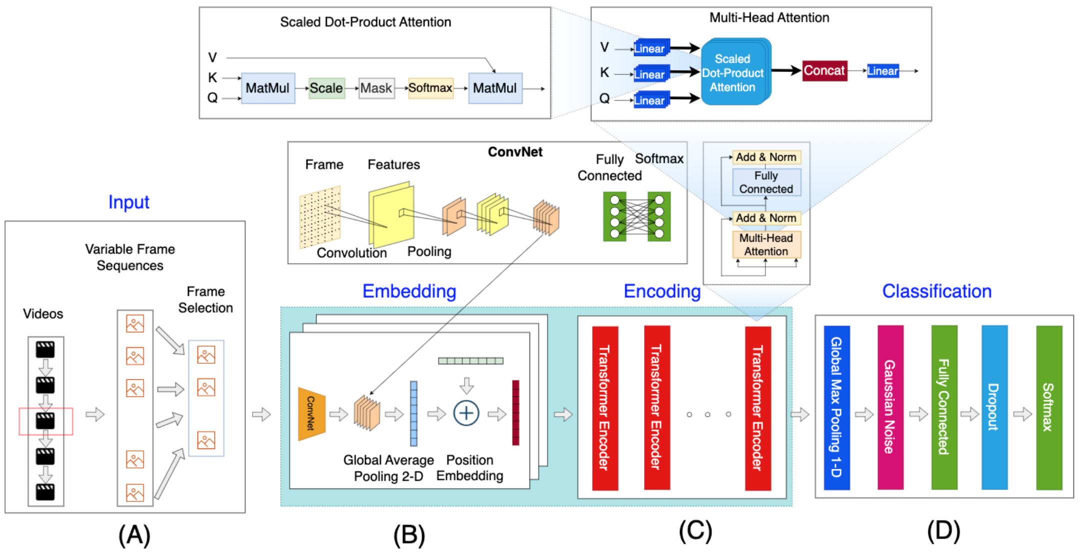


Figure 7.2: Rendering End-to-end ConvNet-Transformer Architecture.

Transformer Encoder

The Transformer Encoder is a key component of the hybrid ConvNet-Transformer architecture. It consists of a stack of N identical layers, each comprising multi-head self-attention and position-wise fully connected feed-forward network sub-layers. To ensure the retention of important input information, residual connections are employed before each operation, followed by layer normalization.

The core of the module is the multi-head self-attention mechanism, which is composed of several self-attention blocks. This mechanism is similar to RNN, as it encodes sequential data by determining the relevance between each element in the sequence. It leverages the inherent relationships between frames in a video to provide a more accurate representation. Furthermore, the self-attention operates on the entire sequence at once, resulting in significant improvements in runtime, as the computation can be parallelized using modern GPUs.

Our architecture employs only the encoder component of a full transformer, as the goal is to obtain a classification label for the video action rather than a sequence. The full transformer consists of both encoder and decoder modules, however, in our case, the use of only the encoder module suffices to achieve the desired result.

Assuming the input sequence ($X = x_1, x_2, ..., x_n$) is first projected onto these weight matrices $Q = XW_Q$, $K = XW_K$, $V = XW_V$ with W_Q,W_K and W_V are three trainable weights, the query ($Q = q_1, q_2, ..., q_n$), key ($K = k_1, k_2, ..., k_n$) of dimension d_k, and value ($V = v_1, v_2, ..., v_n$) of dimension d_v, the output of self-attention is computed as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \quad (7.9)$$

As the name suggested, multi-head attention is composed of several heads and all are concatenated and fed into another linear projection to produce the final outputs as follows:

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O. \quad (7.10)$$

$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad (7.11)$$

where parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$, $i = 1, 2, ..., h$ with h denotes the number of heads.

Frame Selection and Data Pre-processing

Input videos with varying number of frames can pose a challenge for the model which requires a fixed number of inputs. Put simply, to process a video sequence, we incorporated a time distributed layer that requires a predetermined number of frames. To address this issue, we employ several strategies for selecting a smaller subset of frames.

One approach is the "shadow method," where a maximum sequence length is established for each video. While this method is straightforward, it can result in the cutting of longer videos and the loss of information, particularly when the desired length is not reached. In the second method, we utilize a step size to skip some frames, allowing us to achieve the full length of the video while reducing the number of frames used. Additionally, the images are center-cropped to create square images. The efficacy of each method will be evaluated in our experiments.

Layers for Classification

Assuming, we have a set of videos $S(S_1, S_2, ..., S_m)$ with corresponding labels $y(y_1, y_2, ..., y_m)$ where m is the number of samples. We select l frames from the videos and obtain g features from the global average pooling 2-D layer. Each transformer encoder generates a set of representations by consuming the output from the previous block. After N transformer encoder blocks, we can obtain the multi-level representation $H^N(h_1^N, h_2^N, ..., h_l^N)$ where each representation is 1-D vector with the length of g (see Figure 7.2 block (A) $\rightarrow$ (D)).

The classification module incorporates traditional layers, such as fully connected and softmax, and also employs global max pooling to reduce network size. To prevent overfitting, we include Gaussian noise and dropout layers in the design. The ConvNet-Transformer model is trained using stochastic gradient descent and the categorical cross entropy loss is used as the optimization criterion.

7.2.3 ConvNet-RNN

Recent studies have explored the combination of ConvNets and RNNs, particularly LSTMs, to take into account temporal data of frame features for action recognition in videos [38, 208, 179, 64, 180, 18].

To provide a clear understanding of the mathematical operations performed by ConvNets, the following is a summary of the relevant formulations:

$$\begin{cases} O_i = X & \text{if } i = 1 \\ Y_i = f_i(O_{i-1}, W_i) & \text{if } i > 1 \\ O_i = g_i(Y_i) \end{cases} \tag{7.12}$$

$$\begin{cases} Y_i = W_i \circledast O_{i-1} & i^{th} \text{ layer is a convolution} \\ Y_i = \boxplus_{n,m} O_{i-1} & i^{th} \text{ layer is a pool} \\ Y_i = W_i * O_{i-1} & i^{th} \text{ layer is a FC} \end{cases} \tag{7.13}$$

where X represents the input image; O_i is the output for layer i^{th}; W_i indicates the weights of the layer; $f_i(\cdot)$ denotes weight operation for convolution, pooling or FC layers; $g_i(\cdot)$ is an activation function, for example, sigmoid, tanh and rectified linear (ReLU) or more recently Leaky ReLU [121]; The symbol ($\circledast$) acts as a convolution operation which uses *shared* weights to reduce expensive matrix computation [105]; Window ($\boxplus_{n,m}$) shows an average or a max pooling operation which computes average or max values over neighbor region of size $n \times m$ in each feature map. Matrix multiplication of weights between layer i^{th} and the layer $(i-1)^{th}$ in FC is represented as $(*)$.

The last layer in the ConvNet (FC layer) acts as a classifier and is usually discarded for the purpose of using transfer learning. Thereafter, the outputs of the ConvNet from frames in the video sequences are fed as inputs to the RNN layer.

Considering a standard RNN with a given input sequence $x_1, x_2, ..., x_T$, the hidden cell state is updated at a time step t as follows:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b), \tag{7.14}$$

where W_h and W_x denote weight matrices, b represents the bias, and σ is a sigmoid function that outputs values between 0 and 1.

The output of a cell, for ease of notation, is defined as

$$y_t = h_t, \tag{7.15}$$

but can also be shown using the *softmax* function, in which $\hat{y_t}$ is the output and y_t is the target:

$$\hat{y_t} = softmax(W_y h_t + b_y). \tag{7.16}$$

A more sophisticated RNN or LSTM that includes the concept of a forget gate can be expressed as shown in the following equations:

$$f_t = \sigma(W_{fh} h_{t-1} + W_{fx} x_t + b_f), \tag{7.17}$$
$$i_t = \sigma(W_{ih} h_{t-1} + W_{ix} x_t + b_i), \tag{7.18}$$
$$c'_t = tanh(W_{c'h} h_{t-1} + W_{c'x} x_t + b'_c), \tag{7.19}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t, \tag{7.20}$$
$$o_t = \sigma(W_{oh} h_{t-1} + W_{ox} x_t + b_o), \tag{7.21}$$
$$h_t = o_t \odot tanh(c_t), \tag{7.22}$$

where the $\odot$ operation represents an elementwise vector product, and f, i, o and c are the forget gate, input gate, output gate and cell state, respectively. Information is retained when the forget gate f_t becomes 1 and eliminated when f_t is set to 0.

For optimization purposes, an alternative to LSTMs, the GRU, can be utilized due to its lower computational demands. The GRU merges the input gate and forget gate into a single update gate, and the mathematical representation is given by the following equations:

$$r_t = \sigma(W_{rh} h_{t-1} + W_{rx} x_t + b_r), \tag{7.23}$$
$$z_t = \sigma(W_{zh} h_{t-1} + W_{zx} x_t + b_z), \tag{7.24}$$
$$h'_t = tanh(W_{h'h}(r_t \odot h_{t-1}) + W_{h'x} x_t + b_z), \tag{7.25}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h'_t. \tag{7.26}$$

Finally, it's worth noting that while traditional RNNs only consider previous information, bidirectional RNNs incorporate both past and future information in their computations:

$$h_t = \sigma(W_{hx} x_t + W_{hh} h_{t-1} + b_h), \tag{7.27}$$
$$z_t = \sigma(W_{ZX} x_t + W_{HX} h_{t+1} + b_z), \tag{7.28}$$
$$\hat{y_t} = softmax(W_{yh} h_t + W_{yz} z_t + b_y), \tag{7.29}$$

where h_{t-1} and h_{t+1} indicate hidden cell states at the previous time step $(t-1)$ and the future time step $(t+1)$.

7.3 Classification Results

7.3.1 Benchmark Datasets

The UCF-101 dataset, introduced in 2012, is one of the largest annotated video datasets available, and an expansion of the UCF-50 dataset. It comprises 13320 realistic video clips collected from YouTube and covers 101 categories of human actions, such as punching, boxing, and walking. The dataset has three distinct official splits (rather than a pre-divided training set and testing set), and the final accuracy in our experiments is calculated as the arithmetic average of the results across all three splits. Our experiments were conducted using Tensorflow-2.8.2 [1], Keras-2.6.0, and a powerful 4-GPU system (GeForce® GTX 1080 Ti). We used Hiplot [63] for data visualization. Figure 7.3 provides snapshot of samples from each of the action categories.

7.3.2 Evaluation Metric

For evaluating our results, we employ the standard classification accuracy metric, which is defined as follows:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total numbers of predictions made}}. \tag{7.30}$$

7.3.3 Implementation

Training our collaborative models for action recognition involves building a new, dedicated system, as these models require real-time information exchange. To the best of our knowledge, this is the first such system ever built for this purpose. To accommodate the large hardware resources required, each model is trained in a separate environment. After one training epoch, each model updates its current location, previous location, estimate of the gradient of the loss function, and other relevant information, which is then broadcast to neighboring models. To clarify the concept, we provide a diagram of the collaborative system and provide a brief description in this subsection.

Our system for distributed PSO-ConvNets is designed based on a web client-server architecture, as depicted in Figure 7.4. The system consists of two main components: the client side, which is any computer with a web browser interface, and the server side, which comprises three essential services: cloud services, app services, and data services.

The cloud services host the models in virtual machines, while the app services run the ConvNet RNN or ConvNet Transformer models. The information generated by each model is managed by the data services and stored in a data storage. In order to calculate the next positions of particles, each

Figure 7.3: A snapshot of samples of all actions from UCF-101 dataset [163].

particle must wait for all other particles to finish the training cycle in order to obtain the current information.

The system is designed to be operated through a web-based interface, which facilitates the advanced development process and allows for easy interactions between users and the system.

7.3.4 Effectiveness of The Proposed Method

Table 7.1 presents the results of Dynamics 1 and Dynamics 2 on action recognition models. The experiment settings are consistent with our previous research for a fair comparison. As shown in Figure 7.2, we consider two different ConvNet architectures, namely DenseNet-201 and ResNet-152, and select eight models from the Inception [171], EfficientNet [172], DenseNet [74], and ResNet [65] families. In the baseline action recognition methods (DenseNet-201 RNN, ResNet-152 RNN, DenseNet-201 Transformer, and ResNet-152 Transformer), features are first extracted from ConvNets using transfer learning and then fine-tuned. However, in our proposed method, the models are retrained in

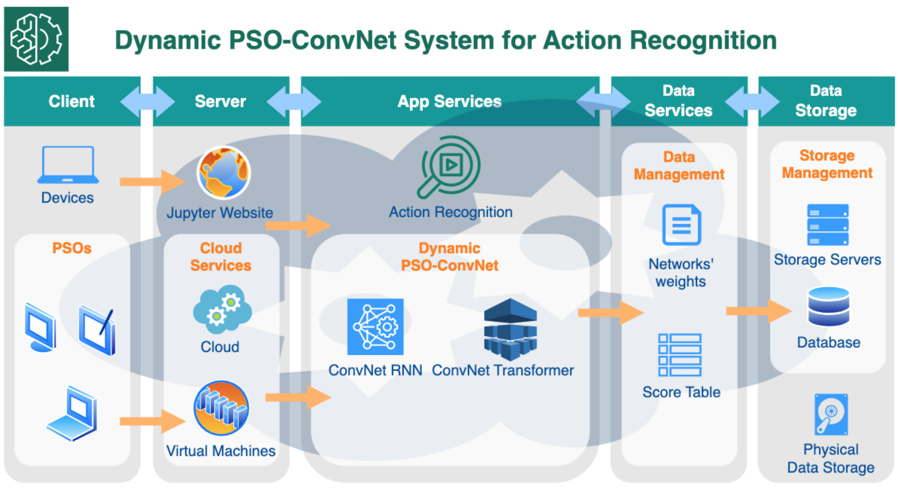


Figure 7.4: Dynamic PSO-ConvNets System Design. The system is divided into two main components, client and server. The client side is accessed through web browser interface while the server side comprises of cloud, app, and data services. The cloud stores virtual machine environments where the models reside. The app service is where the ConvNet-RNN or ConvNet-Transformer runs, and the information generated by each model is managed and saved by the data service. The particles in the system update their positions based on shared information, including current and previous locations, after completing a training cycle.

an end-to-end fashion. Pretrained weights from the ImageNet dataset [97] are utilized to enhance the training speed. Our results show an improvement in accuracy between 1.58% and 8.72%. Notably, the Dynamics 2 for DenseNet-201 Transformer achieves the best result. We also report the time taken to run each method. Fine-tuning takes less time, but the technique can lead to overfitting after a few epochs.

7.3.5 Comparison with state-of-the-art methods

The comparison between our method (Dynamics 2 for ConvNet Transformer) and state-of-the-art (SOTA) approaches is shown in Table 7.2. The second method (which is similar to ours) trains the models on a Sports-1M Youtube dataset and uses the features for UCF-101 recognition. However, the transfer learning procedure is slightly different as their ConvNet architectures were designed specifically for action recognition. While it may have been better to use a pretrained weight for action recognition datasets, such weights are not readily available as the models differ. Also, training the video dataset with millions of samples within a reasonable time is a real challenge for most research centers. Despite these limitations, the use of Transformer and RNN seem to provide a better understanding of temporal characteristics compared to fusion methods. The third method,

Table 7.1: Three-fold classification accuracy (%) on the UCF-101 benchmark dataset. The results of Dynamics 1 and Dynamics 2 using DenseNet-201 Transformer and Resnet-152 Transformer models and compared to baseline models, e.g., Dynamics 1 for DenseNet-201 Transformer versus DenseNet-201 Transformer. The N/A is the abbreviation for the phrase not applicable, for instance, the transfer learning is not applicable in the Dynamics 2 for DenseNet-201 Transformer as the method uses the end-to-end training instead.

				Time per fold (h)	
Dynamics Method	Model	Accuracy	Improve (%)	Transfer Learning	Fine-tune/Retrain
-	DenseNet-201 Transformer	0.7741	N/A	26	0.5
	ResNet-152 Transformer	0.7679	N/A	24	
-	DenseNet-201 RNN	0.8195	N/A	26	0.1
	ResNet-152 RNN	0.8118	N/A	24	
Dynamics 1	DenseNet-201 Transformer	0.8579	8.38	N/A	5.5
	ResNet-152 Transformer	0.8405	7.26	N/A	
	DenseNet-201 RNN	0.8462	2.67	N/A	
	ResNet-152 RNN	0.8276	1.58	N/A	
Dynamics 2	DenseNet-201 Transformer	**0.8613**	**8.72**	N/A	
	ResNet-152 Transformer	0.8399	7.20	N/A	

Table 7.2: Comparisons of the proposed method and the state-of-the-art methods on the UCF-101 benchmark dataset

Method	Accuracy (%)
Bag of words [163]	44.50
Transfer Learning and Fusions [86]	65.40
P-ODN [158]	78.64
Our model (Dynamics 2 for ConvNet Transformer)	**86.13**

Prototype Open Deep Network (P-ODN), introduces prototype learning into open set recognition. The method learns the unknown classes by applying a multi-class triplet thresholding method based on the distance metric between features and prototypes. Our Dynamics 2 method outperformed P-ODN by roughly 9%. The bag of words method was originally used as a baseline for the dataset and achieved the lowest recognition accuracy (44.5%).

7.3.6 Hyperparameter Optimization

In these experiments, we aimed to find the optimal settings for each model. Table 7.3 presents the results of the DenseNet-201 Transformer and ResNet-152 Transformer using transfer learning, where we varied the maximum sequence length, number of frames, number of attention heads, and dense size. The number of frames represents the amount of frames extracted from the sequence, calculated by $step$ = (maximum sequence length)/(number of frames). The results indicate that longer sequences of frames lead to better accuracy, but having a large number of frames is not necessarily the best strategy; a balanced approach yields higher accuracy. Furthermore, we discovered that models performed best with 6 attention heads and a dense size of either 32 or 64 neurons.

Figures 7.5 and 7.6 show the results for ConvNet RNN models using transfer learning. In the experiments, we first evaluated the performance of eight ConvNets (Inception-v3, ResNet-101, ResNet-152, DenseNet-121, DenseNet-201, EfficientNet-B0, EfficientNet-B4, and EfficientNet-B7). The two best performers, DenseNet-121 and ResNet-152 ConvNet architectures, were selected for further experimentation. The results of varying the number of frames showed a preference for longer maximum sequence lengths.

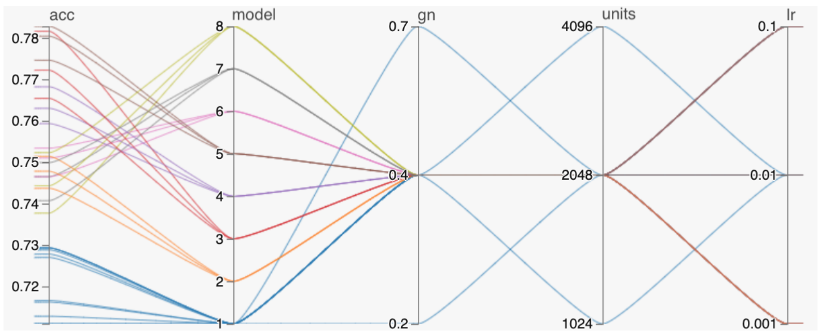


Figure 7.5: Hyperparameter Optimization Results for ConvNet RNN Models with Transfer Learning. The models are numbered as follows: 1.Inception-v3, 2.ResNet-101, 3.ResNet-152, 4.DenseNet-121, 5.DenseNet-201, 6.EfficientNet-B0, 7.EfficientNet-B4, 8.EfficientNet-B7. The abbreviations *acc*, *gn*, and *lr* stand for accuracy, Gaussian noise, and learning rate, respectively.

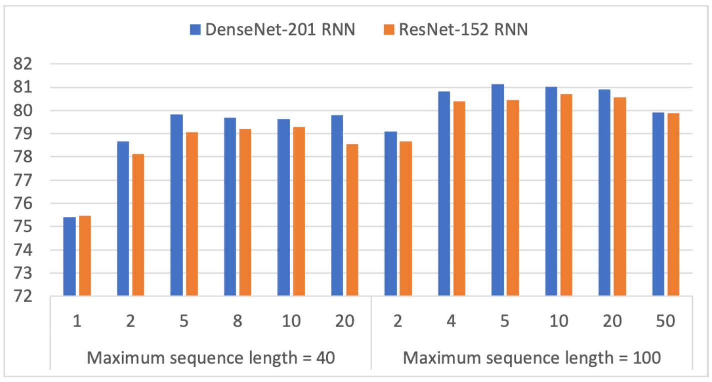


Figure 7.6: Impact of Varying the Number of Frames on the Three-Fold Accuracy of DenseNet-201 RNN and ResNet-152 RNN Using Transfer Learning on the UCF-101 Benchmark Dataset.

7.4 Discussion

The performance of action recognition methods such as ConvNet Transformer and ConvNet RNN is largely dependent on various factors, including the number of attention heads, the number of

Table 7.3: Three-Fold Classification Accuracy Results (%) on the UCF-101 Benchmark Dataset for DenseNet-201 Transformer and ResNet-152 Transformer with Transfer Learning Training.

					Accuracy (%)			
Model	Maximum sequence length	Number of frames	Number of Attention Heads	Dense Size	Set 1	Set 2	Set 3	Avg
DenseNet-201 Transformer	100	2	1	4	69.94	69.42	69.21	69.52
				128	69.57	69.68	68.51	69.25
			4	64	74.36	75.39	75.43	75.06
				128	76	75.15	74.81	75.32
			8	128	74.57	74.75	74.65	74.66
				256	74.94	74.75	75.05	74.91
		4	1	4	70.92	69.47	71.67	70.69
				8	70.02	69.63	70.32	69.99
				16	69.87	68.53	68.29	68.90
				32	70.37	71	69.1	70.16
				64	70.08	69.52	67.99	69.20
				128	69.26	69.42	70.54	69.74
		10	4	64	76.71	76	75.3	76.00
				128	77.43	75.33	76.46	76.41
			6	8	77.08	76.57	76.95	76.87
				16	77.64	76.62	76.06	76.77
				32	77.4	76.73	77.08	77.07
				64	76.87	77.16	78.19	**77.41**
				1024	74.94	73.57	72.48	73.66
		20	6	8	76	77.34	77.25	76.86
				32	76.71	75.99	75.3	76.00
				64	77.16	76	76.46	76.54
				128	77.4	75.87	77.03	76.77
	40	2	4	32	74.65	74.53	74.3	74.49
				64	74.94	74.26	73.54	74.25
				128	74.46	74.29	74.46	74.40
				1024	71.03	70.73	70.45	70.74
			6	16	73.38	74.48	73.97	73.94
				32	74.39	73.43	73.97	73.93
				64	75.34	74.37	73.57	74.43
				128	73.96	74.21	74.03	74.07
		10	6	32	76.37	74.64	74.35	75.12
				64	75.73	75.55	74.81	75.36
				128	76.08	76	75.08	75.72
ResNet-154 Transformer	100	2	1	4	71.08	70.65	68.59	70.11
				128	70.05	71.51	68.89	70.15
			4	128	75.02	76.03	74.73	75.26
				256	74.76	75.66	75.19	75.20
			6	64	75.39	75.52	74.65	75.19
				128	75.57	75.6	75.78	75.65
		10	6	4	76.55	76.78	75.65	76.33
				8	77.08	76.35	76.14	76.52
				16	76.55	76.25	76.33	76.38
				32	77.24	76.49	76.65	**76.79**
				64	75.84	77.53	76.54	76.64

dense neurons, the number of units in RNN, and the learning rate, among others. Collaborative learning is an effective approach to improve the training of neural networks, where multiple models are trained simultaneously and both their positions and directions, as determined by the gradients of the loss function, are shared. In our previous research, we applied dynamics to ConvNets for image classification and in this study, we extend the concept to hybrid ConvNet Transformer and ConvNet RNN models for human action recognition in sequences of images. We first aim to identify the optimal settings that lead to the highest accuracy for the baseline models. As seen in Table 7.1, the ConvNet Transformer models did not perform as well as the ConvNet RNN models with transfer learning, which could be due to the limited data available for training, as transformers typically require more data than RNN-based models. However, our proposed method, incorporating dynamics and end-to-end training, not only outperforms the baseline models, but also results in the ConvNet Transformer models outperforming their ConvNet RNN counterparts. This can be attributed to the additional data provided to the transformer models through data augmentation and additional noise.

7.5 Conclusions and Contributions

Recognizing human actions in videos is a fascinating problem in the art of recognition, and while ConvNets provide a powerful method for image classification, their application to HAR can be complex, as temporal features play a critical role.

In this study, we present a novel video action recognition framework that leverages collaborative learning with dynamics. Our approach explores the hybridization of ConvNet RNN and the recent advanced method Transformer, which has been adapted from Natural Language Processing for video sequences. The experiments include the exploration of two dynamics models, Dynamics 1 and Dynamics 2. The results demonstrate a round improvement of $2\% - 9\%$ in accuracy over baseline methods, such as an 8.72% increase in accuracy for the DenseNet-201 Transformer using Dynamics 2 and a 7.26% increase in accuracy for the ResNet-152 Transformer using Dynamics 1. Our approach outperforms the current state-of-the-art methods, offering significant improvements in video action recognition.

In summary, our work makes three key contributions: (1) We incorporate Dynamics 1 and Dynamics 2 into a hybrid model that combines ConvNet with two popular sequence modeling techniques - RNN and Transformer. (2) We extend the distributed collaborative learning framework to address the task of human action recognition. (3) We conduct extensive experiments on the challenging UCF-101 dataset, over a period of 60 days, and compare our approach against state-of-the-art methods to validate its effectiveness.

Chapter 8

Conclusions and Future Work

The goal of this thesis is to classify images with the latest emerging deep learning techniques and extend the proposed methods to recognize human actions in videos. We identified the following research statements:

RS1. Can capsule networks perform better than the current best method in image recognition (Convolution Neural Networks)?

RS2. How can capsule network be improved?

RS3. Can we apply Recurrent Neural Networks for image recognition and how about other techniques?

RS4. How does collaborative learning enhance image classification?

RS5. Can collaborative learning apply to other fields?

To answer RS1, we propose to improve the design of Vector CN by extending ConvNet layers after Input image and vary the number of FCs in Reconstruction to accomplish better accuracy. Having a larger image size is critical in our approach since certain signs are similar. A small change in the position of fingers can yield distinct signs. In addition, variation of background contexts may hamper Capsules' performance. Using our approach we attain the goals as follows. First, our results show that Capsule 32 V2 performs comparatively to DL MobileNet V1 on accuracy and Capsule 32 V1 runs a slightly faster than its counterpart. Second, Vector CNs are greater than DL models in the sense that the latter requires pre-trained weights which can take days or even months for training whereas the former can be trained on-the-fly within an hour. Third, as a result of our exploration, we found that MobileNet V1 even though was mainly built for small devices but is superior to all other DL models both on accuracy and speed in this dataset. Furthermore, we made a demonstration to illustrate our approach where we compare Vector CN V1 with DL MobileNet V1 in an ASL video. Although, Vector CN can recognize most of all signs (excluding signs that we mentioned earlier), the model is more sensitive to changes than the DL model. This suggests pooling in the DL may better than just rescaling as in the Vector CN. In addition, the DL performs better although the accuracy (when testing on the ASL dataset) is similar to that of Vector CN. This can indicate that the DL model is more generative. For this reason, additional samples are needed to improve the overall performance. Although we demonstrate this approach in the context of ASL alphabet signs, the approach has broader applications to any video recognition tasks where each individual frame's information are crucial.

For RS2, we first propose to replace Standard Convolution in Capsule Networks' Architecture with Depthwise Separable Convolution. Then we perform empirical comparisons of the best Capsule Networks with the best Deep Learning models. The results show that our proposed DW Capsules remarkably decrease the size of models. Among the chosen Capsule Networks, the total parameters had shrunk roughly by an amount between 21% – 25%. In terms of accuracy, Capsule 64 DW Max performs better than other Capsule models. Though Capsule 32 SC Mini can be a trade-off between the accuracy and the number of parameters. In comparison with Deep Learning models, Capsule 32 DW Mini has a larger number of parameters, yet it achieves the accuracy of MobileNet V1 after a few more epochs. Meanwhile, Capsule 32 SC Mini can attain that of MobileNet V1 LSTM's accuracy with 3 to 4 times smaller in the number of parameters. Capsule 32 DW Mini and 64 DW Max outperform Inception V3 MLP and LSTM on accuracy. Moreover, Capsule 64 DW Max occupies 5% less the number of parameters than Inception V3 LSTM though Capsule 32 DW Mini has 4 times larger size than Inception V3 MLP. After a thorough literature search, we believe that this is the first work that proposes the integration of Depthwise Separable Convolution into Capsule Networks. Additionally, we provide empirical evaluations of the proposed Capsule Networks versus the best Deep Learning models.

Regarding RS3, we presented our core ideas for improving ConvNets by integrating RNNs as essential layers in ConvNets when designing end-to-end multiple-model ensembles that gain expertise from each individual ConvNet, an improved training strategy, and a softmax layer extension. First, we propose integrating RNNs into ConvNets even though RNNs are mainly optimized for 1D sequential data rather than 2D images. Our results on the Fashion-MNIST dataset show that ConvNets with RNN, GRU and BiLSTM modules can outperform standard ConvNets. We employed a variety of fine-tuned models in a virtual environment, including InceptionV3, Xception, ResNet50, Inception-ResNetV2 and MobileNetV1. Similar results can be obtained using a dedicated server for the SEResNeXt101 and EfficientNetB5 models on the Cifar-10 and Fashion-MNIST datasets. Although adding RNN modules requires more computing power, there is a potential trade-off between accuracy and running time. Second, we designed E2E-3M ConvNets that learn predictions from several models. We initially tested this model on the iNaturalist'19 dataset using only a single E2E-3M model. Then, we evaluated various models and image resolutions and compared them with the Inception benchmark. Then, we added RNNs to the model and analyzed the performances on the Fashion-MNIST dataset. Our E2E-3M model outperforms a standard single model by a large margin. The advantage of using an end-to-end design is that the model can run immediately in real time and it is suitable for system-on-a-chip platforms. Third, we proposed a training strategy and pruning for the softmax layer that yields comparable accuracies on the Cifar-10 and Fashion-MNIST datasets. Fourth, using the ensemble technique, our models perform competitively, matching previous state-of-the-art methods (accuracies of 0.99, 0.9027 and 0.9852 on SVHN, Cifar-100 and Cifar-10, respectively) even with limited resources. Finally, our method outperforms other approaches on the Surrey dataset, achieving an accuracy of 0.949.

About RS4, we have proposed a novel distributed collaborative PSO-ConvNet algorithm for the optimization performance which is capable of leading particles up to a better minimum. The key contributions of this article are: (1) novel formulations (Dynamics 1 and Dynamics 2) have been successfully created by incorporating distilled Cucker-Smale elements into the PSO algorithm using KNN and intertwining the training with SGD; (2) a new type of particle, i.e., wilder PSO with random learning rate is introduced which has capability of attracting conservative PSOs to stronger minima; (3) a distributed environment is developed for parallel collaboration that significantly accelerates the training; (4) the proposed algorithms are evaluated on Cifar-10 and Cifar-100 benchmark datasets and compared to state-of-the-art algorithms to verify the superior effectiveness.

For RS5, we present a novel video action recognition framework that leverages collaborative learning with dynamics. Our approach explores the hybridization of ConvNet RNN and the recent advanced method Transformer, which has been adapted from Natural Language Processing for video sequences. The experiments include the exploration of two dynamics models, Dynamics 1 and Dynamics 2. The results demonstrate a round improvement of $2\% - 9\%$ in accuracy over baseline methods, such as an 8.72% increase in accuracy for the DenseNet-201 Transformer using Dynamics 2 and a 7.26% increase in accuracy for the ResNet-152 Transformer using Dynamics 1. Our approach outperforms the current state-of-the-art methods, offering significant improvements in video action recognition.

During the course of this study, some of our initial plans were full-filled in later works. For example, we extended our collaborative learning from image recognition to video action recognition. However, there are still ideas to develop. In the future, we plan to redesign CNs' structure to perform on larger images. We also plan to build Vector CNs and DL MobileNet V1 on mobile devices as the accuracy and speed allow these networks to run in realtime. We will try to extend our e2e-3m models to include a greater variety of settings. Finally, we will explore our proposed collaborative learning on more and larger datasets.

Appendix A

Offline and Online Deep Learning for Image Recognition

A.1 Introduction

Recent years have seen a re-appearance of Deep Learning from academy to business area. In academy, the technique has achieved significant higher classification accuracy on competitions such as image recognition [79] and speech recognition [124]. These results inspired by the previous works of LeCun, Bengio and Hilton on DL were catapulted with the availability of GPU and BigData [101]. In business, Google self-driving cars have been tested in large cities and accumulated hundred years of human driving experience [51]. Uber also made a breakthrough in public service as the first company to offer self-driving cars [16]. DL is being seen in Natural Language Processing (NLP) e.g. Apple Siri, Google Now and Amazon Alexa which offer voice recognition services to assist customers in searching information. We believe that one of the next steps is Natural Language Understanding (NLU) for which much achievement is expected soon.

Before the arrival of DL in image classification, the field has evolved through several stages from Linear Classifier to Support Vector Machine and Neural Networks. These methods commonly require selection of features that eventually needs involvement of experts in particular fields. DL on the other hand can choose the best feature automatically [101].

In this article, we employ MNIST and Cifar 10 – standards for digit recognition and image recognition as testbeds for our classifiers. Table A.1 shows a summary of performances of MNIST based on several classifiers. The earliest one made by LeCun (also a main contributor of MNIST) with one layer neural network produced 12% in terms of error rate [99]. Since then, several researches have been done to improve the performance. For example, authors in [44] applied a concept of Support Vector Machine (SVM) that reduces error rate to 2.75%. In addition, authors in [89] decreased the error rate by 5 times. With the recent popularization of the ConvNet, researchers were able to archive lower error rates with deeper layers of convolution as seen in references [109, 26, 25, 186]. At the time of this writing, the best error is 0.21. For Cifar 10, the best error rate is 3.74 [56].

The rest of this work is organized as follows. In Section A.2 we highlight our main contribution. In Section A.3, we deal with offline implementation and setups for comparisons of the shallow and deep classifiers in MNIST dataset. We discuss the results in Section A.4. We also setup other online deep learning experiments with Cifar 10 in Section A.5. Finally, we summarise our findings and approaches toward future work in Section A.6.

Table A.1: Classifiers' Performance on MNIST.

# Ref	Classifier	Error Rate (%)
[99]	1 Layer NN	12
[44]	SVM	2.75
[89]	KNN with IDM	0.54
[109]	Deep CNN	0.47 ± 0.05
[26]	7 CNN	0.27 ± 0.02
[25]	35 CNN	0.23
[186]	DropConnect NN	0.21

A.2 Contribution

In this research, we present our findings for a better performance in image recognition in offline and online settings. We have setup a development framework for performing offline image recognition. We also looked for the best setting in MLP and compare with ConvNet. Moreover, in online setting, we also tried to find the most efficient architecture. Even though still preliminary, these results are very promising and provide approaches toward future exploration.

A.3 Offline Implementation

In this section, we discuss the benchmark dataset, the development framework as well as the setups of Multilayer Perceptron and Convolution Neural Networks.

A.3.1 MNIST

We obtain dataset from MNIST which is a Modified version of United States' National Institute of Standards and Technology. The data has a training set of 60000 samples and a testing set of 10000 samples. The training and testing sets each include scanned handwritten images and desired outputs. These images were rescaled into 20×20 pixel box and then centered in 28×28 pixel field. Each digit in an image represents gray level ranging from 0 to 255 where 0 means white color and

255 means black color [99]. The visual figure of the first ten digits in the training set is shown in Fig. A.1.

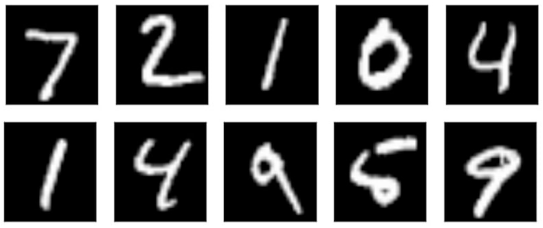


Figure A.1: Visualization of first ten digits MNIST.

A.3.2 Development Framework

In order to perform image recognition with DL, we setup a development framework which includes three layers, namely, OS Layer, Programming Layer, and Toolkit Layer as depicted in Fig. A.2. In the first layer, we perform our experiments on a Macbook Pro (Intel 2.7 GHz). In the programming layer, we choose Python since this language is one of the most popular programming languages in scientific community and the other reason is that Python operates very fast in runtime. In the toolkit layer, we build our image recognition surrounding Tensorflow library which is a DL library and has been supported by Google Inc since 2015. Besides of Tensorflow, there are also several DL libraries e.g. Theano, Torch and Deeplearning4j. For convenience of dealing with different libraries, we decided to use Keras on top of Tensorflow.

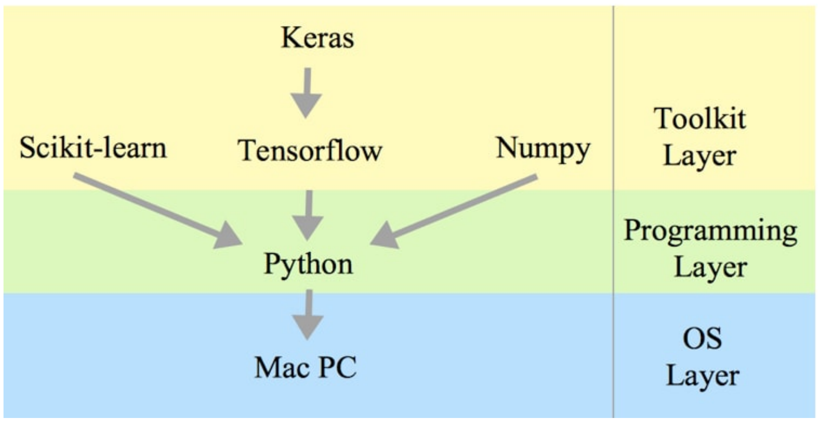


Figure A.2: Development Framework

A.3.3 Setup for MLP

This section deals with how we perform training and testing of the datasets on MLP. Fig. A.3 shows the process of these steps. First of all, as mentioned in the previous section, each image encoded in an array of 28 rows and 28 columns is permuted into a vector of 784 columns. Then the vector will be used as input of our Neural Networks. After that, the data is processed in a fully connected MLP. In the output layer, we set 10 neurons for decoding 10 digits from 0 to 9. For example, the binary 0000000001 would represent digit 0. The binary 0000000010 would represent digit 1 so on and so forth.

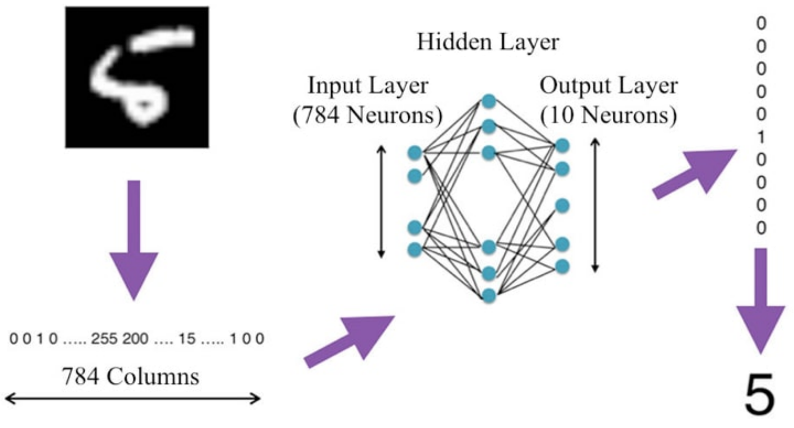


Figure A.3: Process of classifying a sample digit 5 using an MLP

The essential of an MLP is computing weights via Equation A.1 where w_i is the weight of the neuron i, η is the learning rate, t,o and x are target, output and input respectively.

$$w_i \leftarrow w_i + \eta(t - o)x \tag{A.1}$$

A.3.4 Setup for CNN

The structure of the Convolutional Neural Network (CNN) is shown in Fig. A.4. As we can see, this structure includes a convolutional layer in addition to MLP layers. The convolutional layer may include one or more combinations of convolution, pooling and ReLU stages. A unit employing the rectifier activation function is called a rectified linear unit (ReLU). While convolution performs as feature extraction, pooling and ReLU reduce the dimension of the convolution map but still keep essential information [1].

Fig. A.5 depicts an illustration regarding digit 5 when using different feature extractions (filter effects), namely, Origin (where there is no filter applied), Pencil, Scribble and Escher [120]. As we

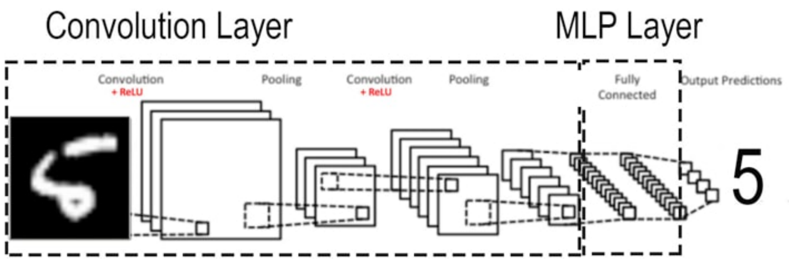


Figure A.4: Process of classifying a sample digit 5 using an CNN (Adapted from [101])

can see, these filters remove background noise in the image and make the digit more or less easier to recognize.

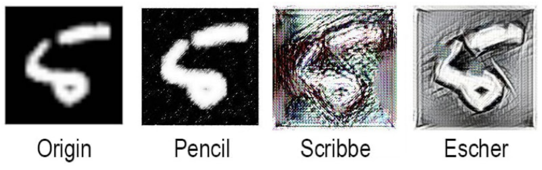


Figure A.5: Feature extraction with different filters for digit 5

A.4 Results

In this section, we discuss results of three experiments including finding the best learning rates for MLP, comparison of MLPs and CNN and performance of detection on each digit.

A.4.1 Experiment 1: Finding the Best Learning Rates

Since Keras sets default learning rate at 0.5, we modified this value to find the best learning rate for our dataset. We varied the values as 1, 0.5, 0.1 and 0.01 and performed iterations from 1 to 10 with 1 increment. From Table A.2, we found that when the learning rate is 0.1, the error rate is lowest (0.05 versus 5.1, 1.03, 2.67 for learning rates of 1, 0.5 and 0.01).

A.4.2 Experiment 2: CNN vs MLPs comparison

In this experiment, we compared MLPs (with default learning rate and our best learning rate) and CNN. We varied the number of neurons in the input layer from 196 ($\frac{1}{4}$ the size of the input vector) to 12544 (16 times the size of the input vector). Table A.3 shows the results regarding the error rates. We also plot these results in Fig. A.6 for convenient comparison.

Table A.2: Error Rates for Different Learning Rates of MLP.

#Iteration	Error Rates (%)			
	lr=1	lr=0.5	lr=0.1	lr=0.01
1	10.74	3.46	3.31	8.38
2	7.45	2.16	2.08	6.73
3	7.66	2.21	1.31	5.71
4	6.84	2.29	1.19	4.86
5	8.80	1.69	0.64	4.44
6	6.61	0.97	0.42	3.91
7	4.82	0.80	0.37	3.56
8	4.82	0.73	0.14	3.16
9	6.36	0.67	0.08	2.88
10	5.10	1.03	0.05	2.67

As it can be observed from Fig. A.6, CNN outperforms both MLPs with the default learning rate and the best learning rate. We also see that when MLPs start getting overfit (error rates begin to increase), CNN is still stable.

Table A.3: Percentage of error rates for different numbers of input neurons for MLP and CNN.

#Number of input neurons	Error (%)		
	MLP (lr=0.5)	MLP (lr=0)	CNN
196	6.36	4.25	2.42
392	5	4.24	2.14
784	4.09	3.91	2.04
1568	3.52	3.52	1.94
3136	3.32	3.19	1.91
6272	3.3	2.99	2.11
9408	5.13	4.14	1.99
12544	13.53	10.33	2.02

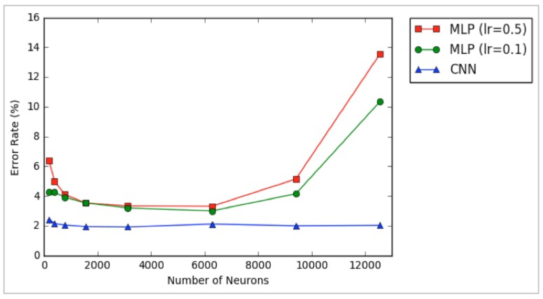


Figure A.6: Comparison of CNN and MLPs

A.4.3 Experiment 3: Performance of Recognition on individual Digits

This experiment is designed to determine if our CNN can recognize a certain digit better than others. Table A.4 shows Precision, Recall and F1-Score of these digits. From this table, digit 0 can be recognized better than the others in terms of Precision. However, in terms of Recall and F1-Score, digit 1 performs better than other digits.

Notice that, these results are varied each time we change the weights of CNN (we initiate the CNN with different seeds). For example, if the seed is 7, the best F1-Score is on digit 1, but if seed is 17, the best F1-Score is on digit 4. So we may conclude that the performance of CNN on each digit depends on the initial weights of CNN.

Table A.4: Precision, Recall and F1-Score on Digits.

Digit	Precision	Recall	F1-Score
0	99.78	98.88	98.83
1	98.86	99.30	99.08
2	98.34	97.67	98.01
3	97.54	98.22	97.88
4	97.87	98.37	98.12
5	98.20	97.87	98.03
6	98.13	98.75	98.44
7	98.53	97.67	98.09
8	97.95	98.25	98.10
9	98.00	97.22	97.61

A.5 Online Implementation

Offline implementation has shown a promising approach in DL. However, it suffers from a drawback in terms of usability because the training process occurs inside local computers that makes accessing from outsiders difficult. Thus, the offline approach limits the number of users cooperating in DL projects. Recently, there is several attempts to bring DL for online production via web for example ConvNetJS and NeuroJS. ConvNetJS is written entirely in Javascript and supports ConvNets and other Neural Networks. In the following, we present our experiments based on this ConvNetJS library.

A.5.1 Cifar-10 Dataset

We performed these experiments based on Cifar-10 which offers 60000 colour images with the size of 32×32 pixels. There are 10 categories including airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck with 6000 images in each category. All images are bundled into a training of 50000 images and testing set of 10000 images. These training images are grouped into 5 batches of

1000 images beside of 1 batch for testing images. The testing batch contains exactly 1000 images of each class where as a training batch may contains more or less than 1000 images of each class [95]. Fig. A.7 shows 10 random images because of limitation in the number of pages.

Figure A.7: Visualization of random images Cifar-10

A.5.2 Experiment 4: Optimizers

This experiment is designed to compare performances of a CNN with regard to several optimizers. The network mainly consists of two convolutional layers and a FC network. The first convolutional layer involves 16 filters have the size of 5×5 and followed by a pool with the size of 2×2. The second convolutional layer comprises of 20 filters and a pool with the same settings as the first layer.

ReLU activation is used in both layers. The output layer is set to classify the 10 different categories. We perform comparisons of two SGD optimizers with the default learning rate and two variations of momentum 0.0 and 0.9. We named these optimizers as SGD and SGD+, respectively. Results of the loss function, training and testing accuracies on number of samples are shown in Fig. A.8. As it can be depicted from the figure, the performances of default SGD are sometimes greater than the performances of SGD+, but at other times are less than those of SGD+. When the number of samples is small (approximately 400000 samples), SGD+ performs better across Loss, Training accuracy and Testing accuracy domains. However, SGD performs generally better with more samples. In addition, accuracies in training and testing are increasing steadily despite slow rates (accuracy achieves roundly 0.5 after 2000k samples).

A.5.3 Experiment 5: Further Improvement

In this experiment, we tried to improve the accuracy of the CNN by varying different architectures. We added a dropout in second convolutional layer and also an Adadelta optimizer. Results of this experiment are plotted in the Fig. A.9. We can observe that performances are improved significantly. In training set, SGD+ can accomplish the accuracy of 0.6 after about 70k samples (versus 2000k in previous setting). In the same time, Adadelta performs a slightly better than SGD+. We also see similar trends for testing set.

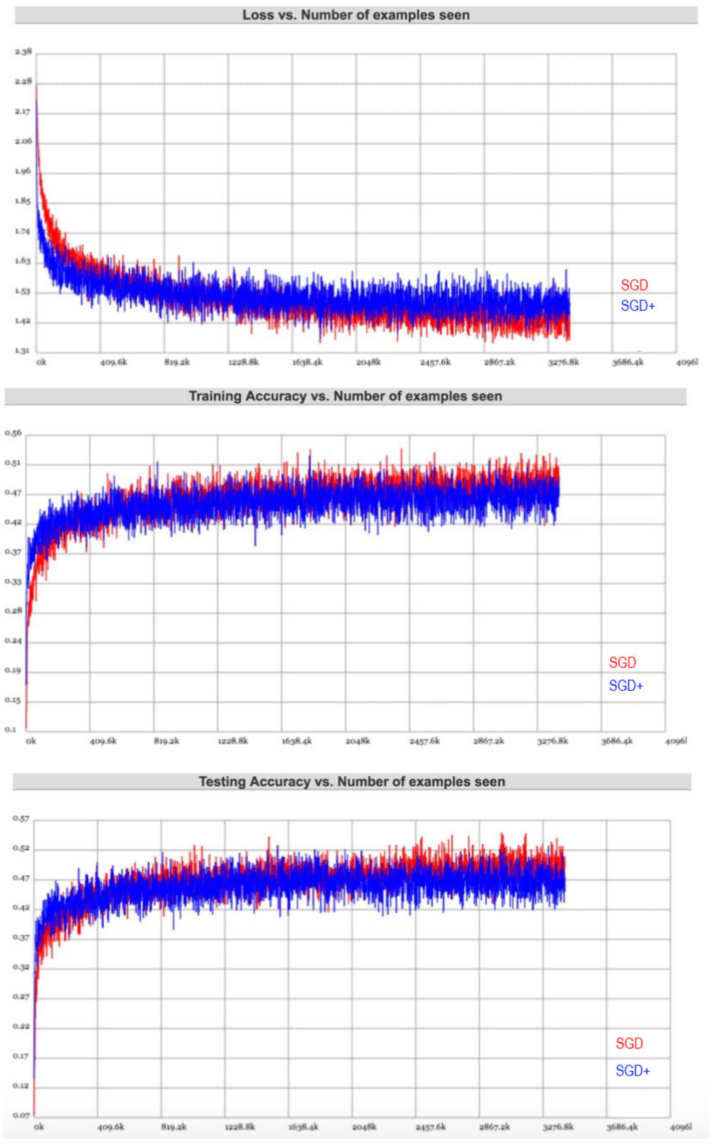
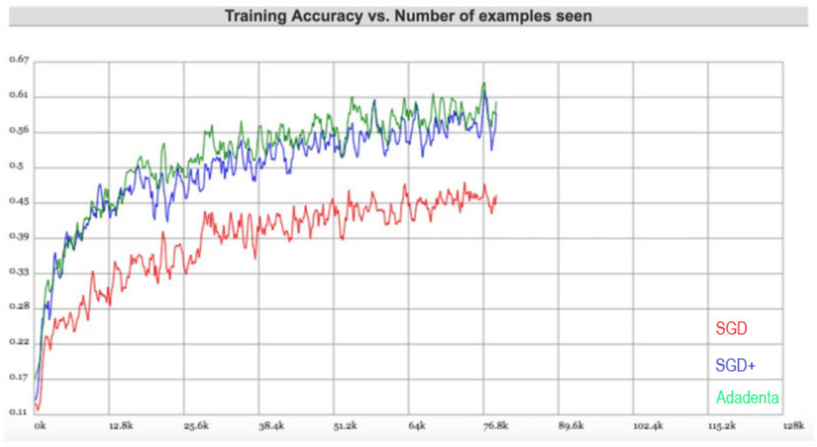


Figure A.8: Comparisons of SGDs in Cifar-10

A.6 Conclusion and Future Work

In this research, we present our findings for offline and online DL in image recognition. In offline recognition, we setup a DL development environment built around TensorFlow and Keras. We also performed comparisons of different optimizers. Results showed that CNN achieves more accuracy and is more stable than typical FC networks. In addition, performances are varied across digits. In



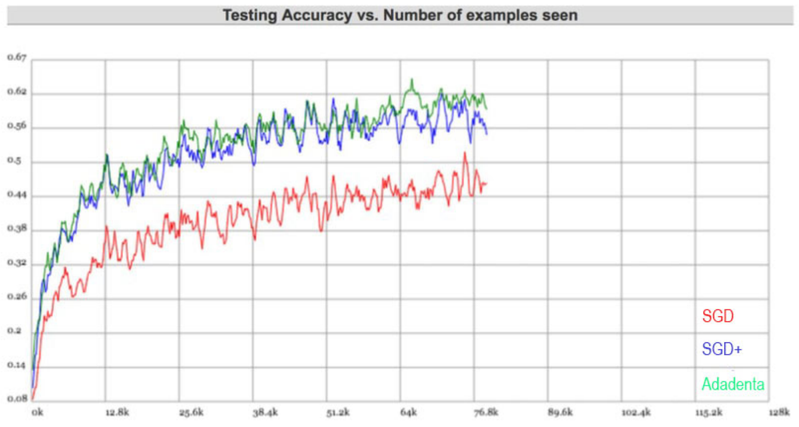


Figure A.9: Comparison of different optimizers in Cifar-10

online image recognition, we setup a web-based approach surrounding a Javascript library for DL. Several optimizers were tested and Adadelta slightly outperforms the best SGD in our setting.

We argue that though using ConvNet does not require expert's knowledge, handcraft filters may result in a better performance since classifying certain objects may indeed require a more concrete understanding of a typical field. Besides of handcraft filters, we also plan to further improve the performance of the optimizers.

Appendix B

Action Recognition for American Sign Language

B.1 Introduction

The use of Deep Learning for action recognition in video has been an active research in recent years. However, they often fall into recognition of actions as the whole picture e.g. an actor is running, jogging or walking or activities of a group playing football, tennis so on and so forth. Recognizing a more subtle action can be seen in emotion or a more relevant to our research as hand gesture. Though hand gesture is often be limited by number of actions e.g. moving left and right, pointing or twisting etc. On the contrary, sign language offers a more standard and abundance of vocabularies for actions. Just only in America, there are approximately a half of million people using American Sign Language.

Research in sign languages mostly focus on static images e.g. numbers or alphabets. For example in ASL alphabets, letters J and Z which are dynamic signs are excluded [98, 4]. Some works explore continuous signs – shown as continuous frames – but vocabularies are just static signs [93]. In our research, we analyze dynamic ASL signs where a sign requires at least two or more shapes.

The structure of this work is as follows. In the next section, we discuss about datasets gathered for this research and present our framework in Section 3. In Section 4, experiments are demonstrated and results analyzed. Conclusions and future works are addressed in Section 5.

B.2 Dataset

We collected our dataset via ASL dictionaries and resources on Internet for 10 different signs, particularly, referring to animals. These include bear, bird, cat, elephant, fish, giraffe, horse, lion, monkey and mouse. However, we exclude signs having more than one expression e.g. dog (this sign requires tapping on one's hip and optionally twisting thump and index fingers). Overall, we obtain 15 videos for each sign: 10 for training and 5 for testing. Our dataset presents a diversity of signers including 23 females, 10 males and 7 children. Each of them performs maximum 10 signs, minimum 1 sign (18 signers) and 3.75 signs on average. In each video, we cut frames from starting of an action until it ends, within 1-2 seconds long approximately. Figure B.1 shows 2 signs sequences for lion and cat. We later extend the dataset to include 15 signs in Section 4.

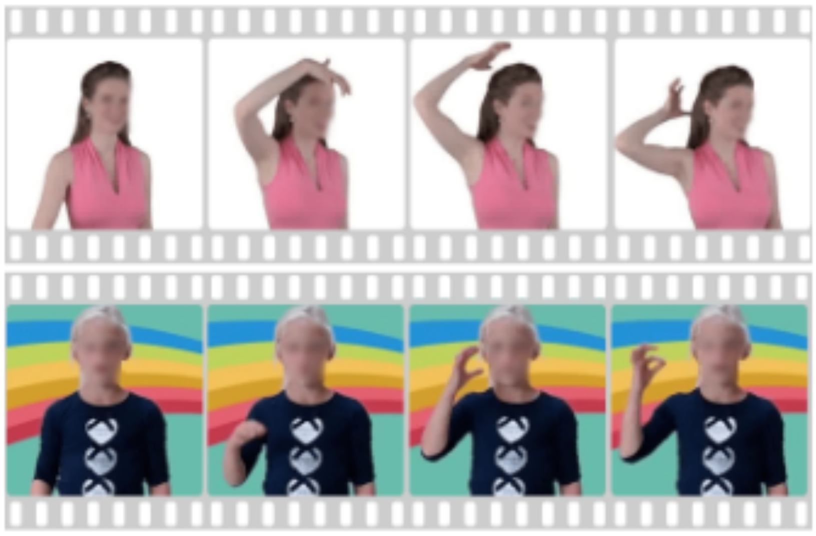


Figure B.1: ASL sequences of lion and cat signs

B.3 Proposed Methods

We show our architecture for ASL Action Recognition in Figure B.2. In this architecture, videos are extracted into frames at different rates e.g. 2 frames/s, 3 frames/s etc. Since visual contents usually share similar elements and training a descent deep net can take several weeks to months, for these reasons, we reuse trained deep networks to filter frames in our architecture and retrain the last layer for our dataset. At this step, we employ different transfer learning models naming InceptionV3, InceptionRestNetV2, Resnet50, DenseNet201 and VGG16/VGG19 to explore which model is the most suitable. Then these frames are extracted to a fixed set of features accordingly.

In classifying features from ASL signs, our approach is innovative compared to others in a similar problem endowing classification of only one-shape signs even when signs are in continuous frames. Please note that an one-shape sign can be recognized using only one frame but a dynamic sign requires at least 2 or more frames. We apply MLP as a baseline on each set of features since the neural network could perform comparable with other more advanced models [138]. Then we perform comparisons of MLP with LSTM. Performances of these structures are analyzed in next section.

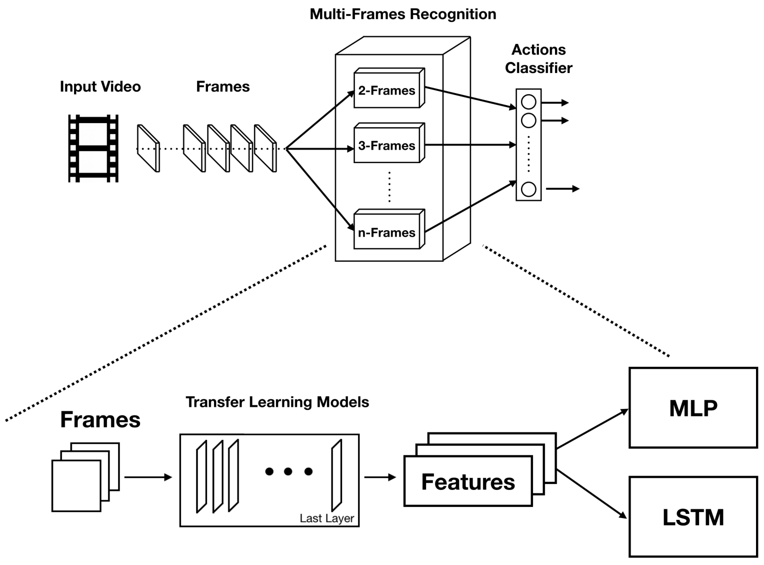


Figure B.2: Framework for Action Recognition

B.4 Experiments and Results

In this section, we fist perform our experiments on ASL raw data for training phase and a pre-processing stage for the testing phase. We vary the length of sequence including 2, 4, 12 and 24 frames per video.

Table B.1 shows results for only InceptionResNetV2, InceptionV3 and DenseNet201. The results of ResNes50 and VGG16/VGG19 are excluded from this table since the accuracy is not much better than randomly guess. Performances of MLP and LSTM are also compared. We can observe that MLP usually outperforms LSTM despite of being a simpler structure. This is also interesting to notice that the accuracy of DenseNet201 is better than other models while it is not the finest model on ImageNet dataset. Regarding the choice of sequence length, a sign can be recognized with the highest accuracy using a sequence of just 12 frames. On the other hand, with only the first frame and the middle frame of an ASL sign, an accuracy of approximated 0.8 can be observed.

In Experiment 2, we compare performances of MLP and LSTM on testing data using transfer learning model DenseNet201. From results for raw ASL data, we can observe that the architecture is not performing well and accuracy are just around 0.3. For this reason, we pre-proceed background subtraction for videos and we also cut the first frame to remove backgrounds left by the subtraction process as we set history window to 1. In addition, the threshold is optimized to 50 since it gives the best accuracy. Beside, we use a median blur filter to remove noise in the frames. We can see from the result that the accuracy is much better with an improved accuracy of 0.58 using LSTM on 12 frames per gesture.

Table B.1: ASL data classification on training DL models

	InceptionResNetV2		InceptionV3		DenseNet201	
#Seq Length	MLP	LSTM	MLP	LSTM	MLP	LSTM
2	-	-	0.86	0.80	0.86	0.88
4	-	-	0.97	0.89	0.98	0.89
12	-	-	0.96	0.92	1.00	0.91
24	0.86	0.75	0.93	0.92	0.98	0.94

We find out that the accuracy of 0.58 may not be helpful for many applications and this could not getting better with the current setting. Further investigations point out that actors perform signs differently e.g. actors may repeat a gesture more often than others. For this reason, we strictly reinforce rules for these signs. Likewise, initial signs were replaced for those we could not find enough videos according to the defined rules. After testing several times, we found that the combination of DenseNet201 and LSTM performs the best. Figure B.3 and Figure B.4 show the accuracy and normalized confusion matrix for the training and testing. We observe the highest accuracy of 0.86 yielded in test. However, in the confusion matrix, we notice that the model misclassifies between a mouse and a cat since their hand positions and shapes are somewhat similar.

In Experiment 4, we extend our dataset from 10 signs to 15 signs including more general conversation terms as finish, hello, love, please and thank you. Confusion matrix is shown in Figure B.5. We can see that while several classes e.g. elephant, lion, monkey and tiger are still being recognized correctly, other classes e.g. bear and love are mixed because of their similarity.

B.5 Conclusions and Future works

In our main contribution, we explore the use of several transfer learning models in combination with deep neural networks to classify dynamic ASL signs. We found that an integration of DenseNet201 and LSTM performs the best. In addition, we collected our own database of 150 videos represent 10 signs and an extension of 225 videos that represent 15 signs for verification. We also vary the number of frames per sign to find the best setting.

Our results show that when we vary number of frames per sign, the use of 12 frames gives the highest accuracy. With two frames for one sign, the sign can also be recognized with accuracy of

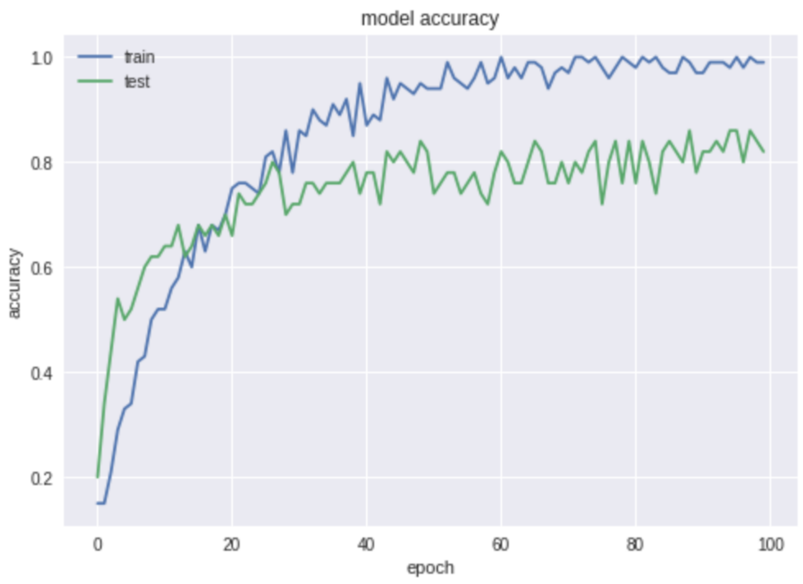


Figure B.3: Model Accuracy on DenseNet201 after Preprocessing

around 0.8. On testing data, to obtain a higher accuracy, we subtract video backgrounds and strictly follow rules for signs. As a consequence, a highest accuracy of 0.86 can be obtained. Our approach different from most of other researches which focus only on static ASL signs and another recent research [204] needs more than one input channel to obtain an accuracy of 0.69.

Future work will address extending the dataset and improving our framework to better recognize signs and perform in realtime.

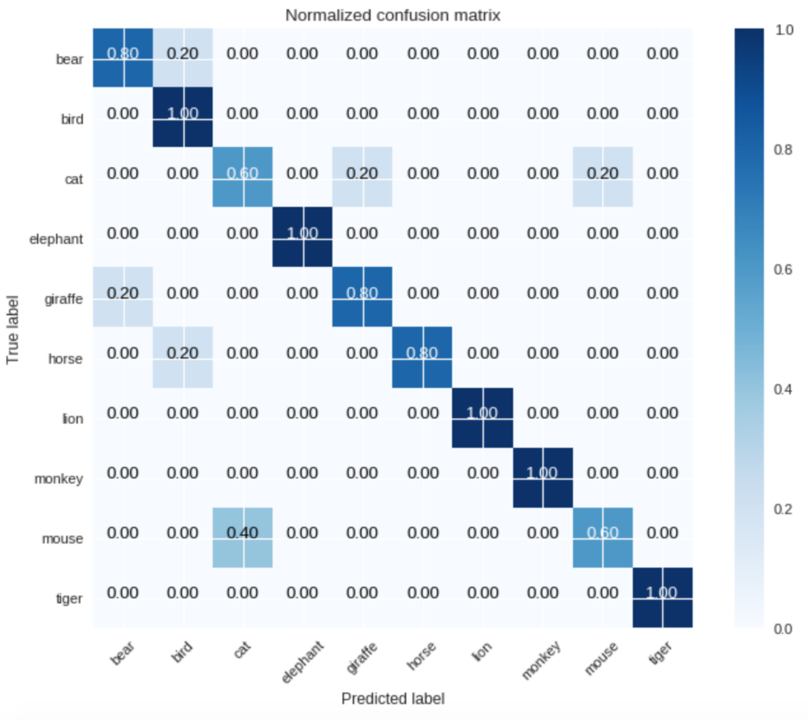
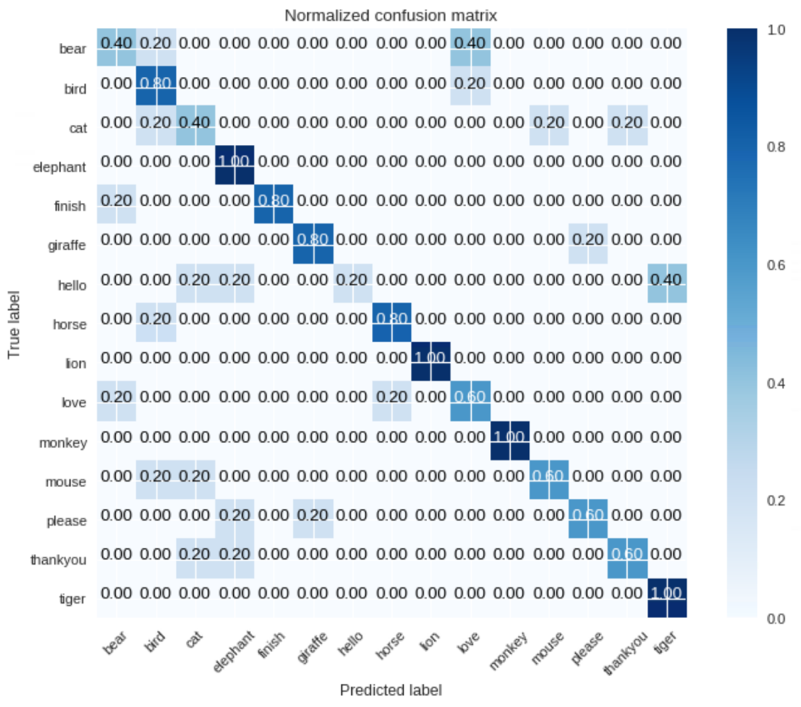


Figure B.4: Normalized Confusion Matrix with Accuracy of 0.86 for 10 Signs



	bear	bird	cat	elephant	finish	giraffe	hello	horse	lion	love	monkey	mouse	please	thankyou	tiger
bear	0.40	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	0.00	0.00	0.00	0.00	0.00
bird	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.00	0.00	0.00	0.00	0.00
cat	0.00	0.20	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.00	0.20	0.00
elephant	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
finish	0.20	0.00	0.00	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
giraffe	0.00	0.00	0.00	0.00	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.00	0.00
hello	0.00	0.00	0.20	0.20	0.00	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40
horse	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
lion	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
love	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.00	0.60	0.00	0.00	0.00	0.00	0.00
monkey	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
mouse	0.00	0.20	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00	0.00	0.00
please	0.00	0.00	0.00	0.20	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00	0.00
thankyou	0.00	0.00	0.20	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00
tiger	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Figure B.5: Normalized Confusion Matrix for Extended Signs

Bibliography

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, software available from tensorflow. org (2015). *URL https://www.tensorflow.org*, 2015.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[3] Asmaa A Alkhouly, Ammar Mohammed, and Hesham A Hefny. Improving the performance of deep neural networks using two proposed activation functions. *IEEE Access*, 9:82249–82271, 2021.

[4] Salem Ameen and Sunil Vadera. A convolutional neural network to classify american sign language fingerspelling from depth and colour images. *Expert Systems*, 34(3):e12197, 2017.

[5] Mostafa Amin-Naji, Ali Aghagolzadeh, and Mehdi Ezoji. Ensemble of cnn for multi-focus image fusion. *Information fusion*, 51:201–214, 2019.

[6] Grigory Antipov, Sid-Ahmed Berrani, and Jean-Luc Dugelay. Minimalistic cnn-based ensemble model for gender prediction from face images. *Pattern recognition letters*, 70:59–65, 2016.

[7] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6836–6846, 2021.

[8] Mohammad Taha Bahadori. Spectral capsule networks. *6th International Conference on Learning Representations*, 2018.

[9] J. C. Bansal, P. K. Singh, Mukesh Saraswat, Abhishek Verma, Shimpi Singh Jadon, and Ajith Abraham. Inertia weight strategies in particle swarm optimization. In *2011 Third World Congress on Nature and Biologically Inspired Computing*, pages 633–640, 2011.

[10] Hritam Basak, Rohit Kundu, Pawan Kumar Singh, Muhammad Fazal Ijaz, Marcin Woźniak, and Ram Sarkar. A union of deep learning and swarm-based optimization for 3d human action recognition. *Scientific Reports*, 12(1):1–17, 2022.

[11] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[12] Vivek Bheda and Dianna Radpour. Using deep convolutional networks for gesture recognition in american sign language. *arXiv preprint arXiv:1710.06836*, 2017.

[13] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[14] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[15] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[16] Kang Cecilia. No driver? bring it on. how pittsburgh became uber's testing ground. `https://www.nytimes.com/2016/09/11/technology/no-driver-bring-it-on-how-pittsburgh-became-ubers-testing-ground.html`. Accessed: 2017-01-01.

[17] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964. IEEE, 2016.

[18] Jun Chen, R. Dinesh Jackson Samuel, and Parthasarathy Poovendran. Lstm with bio inspired algorithm for action recognition in sports videos. *Image and Vision Computing*, 112:104214, 2021.

[19] Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. When vision transformers outperform resnets without pre-training or strong data augmentations. *arXiv preprint arXiv:2106.01548*, 2021.

[20] Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Ekaterina Gonina, et al. State-of-the-art speech recognition with sequence-to-sequence models. In *2018 IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4774–4778. IEEE, 2018.

[21] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[22] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[23] Sumit Chopra, Michael Auli, and Alexander M Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, 2016.

[24] Li-Yeh Chuang, Sheng-Wei Tsai, and Cheng-Hong Yang. Improved binary particle swarm optimization using catfish effect for feature selection. *Expert Systems with Applications*, 38(10):12699–12707, 2011.

[25] Dan CireşAn, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[26] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber. Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1135–1139. IEEE, 2011.

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

[28] Charles A Collin and Patricia A Mcmullen. Subordinate-level categorization relies on high spatial frequencies to a greater degree than basic-level categorization. *Perception & psychophysics*, 67(2):354–364, 2005.

[29] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 113–123, 2019.

[30] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[31] Felipe Cucker and Steve Smale. Emergent behavior in flocks. *IEEE Transactions on automatic control*, 52(5):852–862, 2007.

[32] Sean Curtis, Basim Zafar, Adnan Gutub, and Dinesh Manocha. Right of way. *The Visual Computer*, 29(12):1277–1292, 2013.

[33] Giovanni Lucca França da Silva, Thales Levi Azevedo Valente, Aristófanes Corrêa Silva, Anselmo Cardoso de Paiva, and Marcelo Gattass. Convolutional neural network-based pso for lung nodule false positive reduction on ct images. *Computer methods and programs in biomedicine*, 162:109–118, 2018.

[34] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

[36] R. Dey and F. M. Salemt. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1597–1600, Aug 2017.

[37] Ali Diba, Mohsen Fayyaz, Vivek Sharma, Amir Hossein Karami, Mohammad Mahdi Arzani, Rahman Yousefzadeh, and Luc Van Gool. Temporal 3d convnets: New architecture and transfer learning for video classification. *arXiv preprint arXiv:1711.08200*, 2017.

[38] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[39] Cao Dong, Ming C Leu, and Zhaozheng Yin. American sign language alphabet recognition using microsoft kinect. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 44–52, 2015.

[40] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[41] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[42] Mingxing Duan, Kenli Li, and Keqin Li. An ensemble cnn2elm for age estimation. *IEEE Transactions on Information Forensics and Security*, 13(3):758–772, 2017.

[43] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[44] Reza Ebrahimzadeh and Mahdi Jampour. Efficient handwritten digit recognition based on histogram of oriented gradients and svm. *International Journal of Computer Applications*, 104(9), 2014.

[45] Yingruo Fan, Jacqueline CK Lam, and Victor OK Li. Multi-region ensemble convolutional neural network for facial expression recognition. In *International Conference on Artificial Neural Networks*, pages 84–94. Springer, 2018.

[46] Brandon Garcia and Sigberto Alarcon Viesca. Real-time american sign language recognition with convolutional neural networks. *Convolutional Neural Networks for Visual Recognition*, 2, 2016.

[47] Hongwei Ge, Zehang Yan, Wenhao Yu, and Liang Sun. An attention mechanism based convolutional lstm network for video action recognition. *Multimedia Tools and Applications*, 78(14):20533–20556, 2019.

[48] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[49] Céline R Gillebert, Hans P Op de Beeck, Sven Panis, and Johan Wagemans. Subordinate categorization enhances the neural selectivity in human object-selective cortex for fine shape differences. *Journal of Cognitive Neuroscience*, 21(6):1054–1064, 2009.

[50] Davide Giri, Kuan-Lin Chiu, Giuseppe Di Guglielmo, Paolo Mantovani, and Luca P Carloni. Esp4ml: Platform-based design of systems-on-chip for embedded machine learning. *arXiv preprint arXiv:2004.03640*, 2020.

[51] google.com. Google self-driving cars project. `https://waymo.com/`. Accessed: 2017-01-01.

[52] Lena Gorelick, Moshe Blank, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as space-time shapes. *IEEE transactions on pattern analysis and machine intelligence*, 29(12):2247–2253, 2007.

[53] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*, 2018.

[54] Shreyank N Gowda, Marcus Rohrbach, and Laura Sevilla-Lara. Smart frame selection for action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1451–1459, 2021.

[55] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[56] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[57] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[58] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[59] Caglar Gulcehre, Marcin Moczulski, and Yoshua Bengio. Adasecant: robust adaptive secant method for stochastic gradient. *arXiv preprint arXiv:1412.7419*, 2014.

[60] Hong-Gui Han, Wei Lu, Ying Hou, and Jun-Fei Qiao. An adaptive-pso-based self-organizing rbf neural network. *IEEE Transactions on Neural Networks and Learning Systems*, 29(1):104–117, 2018.

[61] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? *Advances in neural information processing systems*, 31, 2018.

[62] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018.

[63] D. Haziza, J. Rapin, and G. Synnaeve. Hiplot, interactive high-dimensionality plots. `https://github.com/facebookresearch/hiplot`, 2020.

[64] Jun-Yan He, Xiao Wu, Zhi-Qi Cheng, Zhaoquan Yuan, and Yu-Gang Jiang. Db-lstm: Densely-connected bi-directional lstm for human action recognition. *Neurocomputing*, 444:319–331, 2021.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. doi:10.1109/CVPR.2016.90.

[66] Yanzhang He, Tara N Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, et al. Streaming end-to-end speech recognition for mobile devices. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6381–6385. IEEE, 2019.

[67] Aharon B Hillel and Daphna Weinshall. Subordinate class recognition using relational object models. In *Advances in Neural Information Processing Systems*, pages 73–80, 2007.

[68] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *6th International Conference on Learning Representations, ICLR*, 2018.

[69] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In *International Conference on Machine Learning*, pages 2731–2741. PMLR, 2019.

[70] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[71] Essam H Houssein, Ahmed G Gad, Kashif Hussain, and Ponnuthurai Nagaratnam Suganthan. Major advances in particle swarm optimization: theory, analysis, and application. *Swarm and Evolutionary Computation*, 63:100868, 2021.

[72] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[73] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[74] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. doi:10.1109/CVPR.2017.243.

[75] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[76] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[77] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[78] Earnest Paul Ijjina and Krishna Mohan Chalavadi. Human action recognition using genetic algorithms and convolutional neural networks. *Pattern recognition*, 59:199–212, 2016.

[79] image net.org. Imagenet contest result 2012. `http://image-net.org/challenges/LSVRC/2012/results.html`. Accessed: 2017-01-01.

[80] Philip TG Jackson, Amir Atapour Abarghouei, Stephen Bonner, Toby P Breckon, and Boguslaw Obara. Style augmentation: data augmentation via style randomization. In *CVPR Workshops*, pages 83–92, 2019.

[81] I Jeena Jacob and P Ebby Darney. Artificial bee colony optimization algorithm for enhancing routing in wireless networks. *Journal of Artificial Intelligence*, 3(01):62–71, 2021.

[82] Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljacic, and Yoshua Bengio. Gated orthogonal recurrent units: On learning to forget. *Neural computation*, 31(4):765–783, 2019.

[83] Mahdi M Kalayeh and Mubarak Shah. Training faster by separating modes of variation in batch-normalized models. *IEEE transactions on pattern analysis and machine intelligence*, 42(6):1483–1500, 2019.

[84] Byeongkeun Kang, Subarna Tripathi, and Truong Q Nguyen. Real-time sign language fingerspelling recognition using convolutional neural networks from depth map. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 136–140. IEEE, 2015. doi:10.1109/ACPR.2015.7486481.

[85] Dervis Karaboga et al. An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer . . . , 2005.

[86] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[87] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[88] Cem Keskin, Furkan Kıraç, Yunus Emre Kara, and Lale Akarun. Hand pose estimation and hand shape classification using multi-layered randomized decision forests. In *European Conference on Computer Vision*, pages 852–863. Springer, 2012.

[89] Daniel Keysers, Thomas Deselaers, Christian Gollan, and Hermann Ney. Deformation models for image recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8), 2007.

[90] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, Providence, RI, USA, 2011.

[91] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[92] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 6(2):8, 2019.

[93] Oscar Koller, Hermann Ney, and Richard Bowden. Deep hand: How to train a cnn on 1 million hand images when your data is continuous and weakly labelled. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3793–3802, 2016.

[94] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[95] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

[96] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55(5), 2014.

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

[98] Alina Kuznetsova, Laura Leal-Taixé, and Bodo Rosenhahn. Real-time sign language recognition using a consumer depth camera. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 83–90, 2013. doi:10.1109/ICCVW.2013.18.

[99] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[100] Yann LeCun. Learning invariant feature hierarchies. In *European conference on computer vision*, pages 496–505. Springer, 2012.

[101] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[102] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[103] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[104] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi:10.1109/5.726791.

[105] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*, pages 253–256. IEEE, 2010.

[106] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial intelligence and statistics*, pages 464–472, 2016.

[107] Ang Li, Zhenjiang Miao, Yigang Cen, Xiao-Ping Zhang, Linna Zhang, and Shiming Chen. Abnormal event detection in surveillance videos based on low-rank and compact coefficient dictionary learning. *Pattern Recognition*, 108:107355, 2020.

[108] Mi Li, Huan Chen, Xin Shi, Sa Liu, Ming Zhang, and Shengfu Lu. A multi-information fusion "triple variables with iteration" inertia weight pso algorithm and its application. *Applied Soft Computing*, 84:105677, 2019.

[109] Yang Li, Hang Li, Yulong Xu, Jiabao Wang, and Yafei Zhang. Very deep neural network for handwritten digit recognition. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 174–182. Springer, 2016.

[110] Senwei Liang, Yuehaw Khoo, and Haizhao Yang. Drop-activation: Implicit parameter reduction and harmonic regularization. *arXiv preprint arXiv:1811.05850*, 2018.

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

[112] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[113] Jianhua Liu, Yi Mei, and Xiaodong Li. An analysis of the inertia weight parameter for binary particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 20(5):666–681, 2016.

[114] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[115] Mengyuan Liu, Hong Liu, and Chen Chen. Enhanced skeleton visualization for view invariant human action recognition. *Pattern Recognition*, 68:346–362, 2017.

[116] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2016.

[117] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[118] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3202–3211, 2022.

[119] Raphael Gontijo Lopes, Dong Yin, Ben Poole, Justin Gilmer, and Ekin D Cubuk. Improving robustness without sacrificing accuracy with patch gaussian augmentation. *arXiv preprint arXiv:1906.02611*, 2019.

[120] LunaPic.com. Online image filters. `http://www198.lunapic.com/editor/`. Accessed: 2017-04-19.

[121] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, page 3. Citeseer, 2013.

[122] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[123] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

[124] Microsoft.com. Microsoft researchers achieve speech recognition milestones. `https://blogs.microsoft.com/next/2016/09/13/microsoft-researchers-achieve-speech-recognition-milestone/`. Accessed: 2017-01-01.

[125] Aryan Mobiny and Hien Van Nguyen. Fast capsnet for lung cancer screening. *arXiv preprint arXiv:1806.07416*, 2018.

[126] Ernest Mwebaze, Timnit Gebru, Andrea Frome, Solomon Nsumba, and Jeremy Tusubira. icassava 2019fine-grained visual categorization challenge. *arXiv preprint arXiv:1908.02900*, 2019.

[127] Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik. Xnas: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems*, pages 1975–1985, 2019.

[128] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[129] Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang Koh, Quoc Le, and Andrew Ng. Tiled convolutional neural networks. *Advances in neural information processing systems*, 23, 2010.

[130] Huu Phong Nguyen and Bernardete Ribeiro. Advanced capsule networks via context awareness. In *Artificial Neural Networks and Machine Learning–ICANN 2019: Theoretical Neural Computation: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part I 28*, pages 166–177. Springer, 2019.

[131] Ahmad Nickabadi, Mohammad Mehdi Ebadzadeh, and Reza Safabakhsh. A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied Soft Computing*, 11(4):3658–3670, 2011.

[132] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. Asap: Architecture search, anneal and prune. In *International Conference on Artificial Intelligence and Statistics*, pages 493–503. PMLR, 2020.

[133] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[134] S Nissi Paul and Y Jayanta Singh. Survey on video analysis of human walking motion. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 7(3):99–122, 2014.

[135] Fabrizio Pedersoli, Sergio Benini, Nicola Adami, and Riccardo Leonardi. Xkin: an open source framework for hand pose and gesture recognition using kinect. *The Visual Computer*, 30(10):1107–1122, 2014.

[136] Fábio Perez, Sandra Avila, and Eduardo Valle. Solo or ensemble? choosing a cnn architecture for melanoma classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[137] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[138] Nguyen Huu Phong and Bernardete Ribeiro. Offline and online deep learning for image recognition. In *2017 4th Experiment@ International Conference (exp. at'17)*, pages 171–175. IEEE, 2017.

[139] Nguyen Huu Phong, Augusto Santos, and Bernardete Ribeiro. Pso-convolutional neural networks with heterogeneous learning rate. *IEEE Access*, 10:89970–89988, 2022.

[140] Ronald Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.

[141] Liliana Lo Presti and Marco La Cascia. 3d skeleton-based human action classification: A survey. *Pattern Recognition*, 53:130–147, 2016.

[142] Nicolas Pugeault and Richard Bowden. Spelling it out: Real-time asl fingerspelling recognition. In *2011 IEEE International conference on computer vision workshops (ICCV workshops)*, pages 1114–1119. IEEE, 2011.

[143] Xiangyun Qing and Yugang Niu. Hourly day-ahead solar irradiance prediction using weather forecasts by lstm. *Energy*, 148:461–468, 2018.

[144] Haziq Razali, Taylor Mordan, and Alexandre Alahi. Pedestrian intention prediction: A convolutional bottom-up multi-task approach. *Transportation research part C: emerging technologies*, 130:103259, 2021.

[145] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, pages 4780–4789, 2019.

[146] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

[147] Lucas Rioux-Maldague and Philippe Giguere. Sign language fingerspelling classification from depth and color images using a deep belief network. In *2014 Canadian Conference on Computer and Robot Vision*, pages 92–97. IEEE, 2014.

[148] Nizar Rokbani, Raghvendra Kumar, Ajith Abraham, Adel M Alimi, Hoang Viet Long, Ishaani Priyadarshini, and Le Hoang Son. Bi-heuristic ant colony optimization-based approaches for traveling salesman problem. *Soft Computing*, 25(5):3775–3794, 2021.

[149] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[150] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[151] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[152] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

[153] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. doi:10.1109/CVPR.2018.00474.

[154] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[155] Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1):120–131, 2018.

[156] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[157] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE, 1998.

[158] Yu Shu, Yemin Shi, Yaowei Wang, Tiejun Huang, and Yonghong Tian. P-odn: Prototype-based open deep network for open set recognition. *Scientific reports*, 10(1):1–13, 2020.

[159] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 27, 2014.

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

[161] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[162] Sijie Song, Cuiling Lan, Junliang Xing, Wenjun Zeng, and Jiaying Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[163] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[164] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

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

[166] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504, 2017.

[167] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6479–6488, 2018.

[168] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2):394–407, 2019.

[169] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[170] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[171] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the*

IEEE conference on computer vision and pattern recognition, pages 2818–2826, 2016. doi:10.1109/CVPR.2016.308.

[172] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[173] Wenjin Tao, Ming C Leu, and Zhaozheng Yin. American sign language alphabet recognition using convolutional neural networks with multiview augmentation and inference fusion. *Engineering Applications of Artificial Intelligence*, 76:202–213, 2018.

[174] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[175] Nergis Tomen and Jan C van Gemert. Spectral leakage and rethinking the kernel size in cnns. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5138–5147, 2021.

[176] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 32–42, 2021.

[177] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[178] Shanshan Tu, Sadaqat ur Rehman, Muhammad Waqas, Obaid ur Rehman, Zubair Shah, Zhongliang Yang, and Anis Koubaa. Modpso-cnn: an evolutionary convolution neural network with application to visual recognition. *Soft Computing*, 25(3):2165–2176, 2021.

[179] Amin Ullah, Jamil Ahmad, Khan Muhammad, Muhammad Sajjad, and Sung Wook Baik. Action recognition in video sequences using deep bi-directional lstm with cnn features. *IEEE access*, 6:1155–1166, 2017.

[180] Amin Ullah, Jamil Ahmad, Khan Muhammad, Muhammad Sajjad, and Sung Wook Baik. Action recognition in video sequences using deep bi-directional lstm with cnn features. *IEEE Access*, 6:1155–1166, 2018.

[181] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8769–8778, 2018.

[182] Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1510–1517, 2017.

[183] Danish Vasan, Mamoun Alazab, Sobia Wassan, Babak Safaei, and Qin Zheng. Image-based malware classification using ensemble of cnn architectures (imcec). *Computers & Security*, page 101748, 2020.

[184] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[185] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. *Technical Report*, 2011.

[186] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[187] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.

[188] Chong Wang, Zhong Liu, and Shing-Chow Chan. Superpixel-based hand gesture recognition with kinect depth camera. *IEEE transactions on multimedia*, 17(1):29–39, 2014.

[189] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *CVPR 2011*, pages 3169–3176, 2011.

[190] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision*, pages 3551–3558, 2013.

[191] Rui Wang, Wei Li, and Liang Zhang. Blur image identification with ensemble convolution neural networks. *Signal Processing*, 155:73–82, 2019.

[192] Xinchen Wang, Weiwei Zhang, Xuncheng Wu, Lingyun Xiao, Yubin Qian, and Zhi Fang. Real-time vehicle type classification with deep convolutional neural networks. *Journal of Real-Time Image Processing*, 16(1):5–14, 2019.

[193] Yulong Wang, Haoxin Zhang, and Guangwei Zhang. cpso-cnn: An efficient pso-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. *Swarm and Evolutionary Computation*, 49:114–123, 2019.

[194] Zuxuan Wu, Caiming Xiong, Chih-Yao Ma, Richard Socher, and Larry S Davis. Adaframe: Adaptive frame selection for fast video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1278–1287, 2019.

[195] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[196] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[197] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European conference on computer vision (ECCV)*, pages 305–321, 2018.

[198] Bing Xue, Mengjie Zhang, and Will N Browne. Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE transactions on cybernetics*, 43(6):1656–1671, 2012.

[199] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise. Shakedrop regularization for deep residual learning. *arXiv preprint arXiv:1802.02375*, 2018.

[200] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise. Shakedrop regularization for deep residual learning. *IEEE Access*, 7:186126–186136, 2019.

[201] Hao Yang, Li Liu, Weidong Min, Xiaosong Yang, and Xin Xiong. Driver yawning detection based on subtle facial action recognition. *IEEE Transactions on Multimedia*, 23:572–583, 2020.

[202] Shih-Hung Yang, Wei-Ren Chen, Wun-Jhu Huang, and Yon-Ping Chen. Ddanet: Dual-path depth-aware attention network for fingerspelling recognition using rgb-d images. *IEEE Access*, 2020.

[203] Bangpeng Yao, Aditya Khosla, and Li Fei-Fei. Combining randomization and discrimination for fine-grained image categorization. In *CVPR 2011*, pages 1577–1584. IEEE, 2011.

[204] Yuancheng Ye, Yingli Tian, Matt Huenerfauth, Jingya Liu, Nataniel Ruiz, Eunji Chong, James M Rehg, Sveinn Palsson, Eirikur Agustsson, Radu Timofte, et al. Recognizing american sign language gestures from within continuous videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2064–2073, 2018.

[205] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

[206] Steven R Young, Derek C Rose, Thomas P Karnowski, Seung-Hwan Lim, and Robert M Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the workshop on machine learning in high-performance computing environments*, pages 1–5, 2015.

[207] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[208] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

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

[210] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

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

[212] Chenyang Zhang and Yingli Tian. Histogram of 3d facets: A depth descriptor for human action and hand gesture recognition. *Computer Vision and Image Understanding*, 139:29–39, 2015.

[213] Chenyang Zhang, Xiaodong Yang, and YingLi Tian. Histogram of 3d facets: A characteristic descriptor for hand gesture recognition. In *2013 10th IEEE international conference and workshops on automatic face and gesture recognition (FG)*, pages 1–8. IEEE, 2013.

[214] Ke Zhang, Miao Sun, Tony X Han, Xingfang Yuan, Liru Guo, and Tao Liu. Residual networks of residual networks: Multilevel residual networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(6):1303–1314, 2017.

[215] Pengfei Zhang, Cuiling Lan, Junliang Xing, Wenjun Zeng, Jianru Xue, and Nanning Zheng. View adaptive recurrent neural networks for high performance human action recognition from skeleton data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2117–2126, 2017.

[216] Rui Zhang. Sports action recognition based on particle swarm optimization neural networks. *Wireless Communications and Mobile Computing*, 2022, 2022.

[217] Yanyi Zhang, Xinyu Li, Chunhui Liu, Bing Shuai, Yi Zhu, Biagio Brattoli, Hao Chen, Ivan Marsic, and Joseph Tighe. Vidtr: Video transformer without convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 13577–13587, 2021.

[218] Yunfeng Zhang, Xinxin Liu, Fangxun Bao, Jing Chi, Caiming Zhang, and Peide Liu. Particle swarm optimization with adaptive learning strategy. *Knowledge-Based Systems*, 196:105789, 2020.

[219] Yaowei Zheng, Richong Zhang, and Yongyi Mao. Regularizing neural networks via adversarial model perturbation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8156–8165, 2021.

[220] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

[221] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, pages 13001–13008, 2020.

[222] Hongyuan Zhu, Romain Vial, and Shijian Lu. Tornado: A spatio-temporal convolutional regression network for video action proposal. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5813–5821, 2017.

[223] Yuhao Zhu, Matthew Mattina, and Paul Whatmough. Mobile machine learning hardware at arm: a systems-on-chip (soc) perspective. *arXiv preprint arXiv:1801.06274*, 2018.

[224] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. doi:10.1109/CVPR.2018.00907.