Continual Deep Learning via Progressive Learning

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

Haytham M. Fayek

B.Eng. (Hons), Petronas University of Technology
M.Sc., Petronas University of Technology

School of Engineering
College of Science, Engineering and Health
RMIT University
February 2019

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Haytham M. Fayek
Melbourne, Victoria
February 4, 2019

Abstract

Machine learning is one of several approaches to artificial intelligence. It allows us to build machines that can learn from experience as opposed to being explicitly programmed. Current machine learning formulations are mostly designed for learning and performing a particular task from a tabula rasa using data available for that task. For machine learning to converge to artificial intelligence, in addition to other desiderata, it must be in a state of continual learning, i.e., have the ability to be in a continuous learning process, such that when a new task is presented, the system can leverage prior knowledge from prior tasks, in learning and performing this new task, and augment the prior knowledge with the newly acquired knowledge without having a significant adverse effect on the prior knowledge. Continual learning is key to advancing machine learning and artificial intelligence.

Deep learning is a powerful general-purpose approach to machine learning that is able to solve numerous and various tasks with minimal modification. Deep learning extends machine learning, and specially neural networks, to learn multiple levels of distributed representations together with the required mapping function into a single composite function. The emergence of deep learning and neural networks as a generic approach to machine learning, coupled with their ability to learn versatile hierarchical representations, has paved the way for continual learning. The main aim of this thesis is the study and development of a structured approach to continual learning, leveraging the success of deep learning and neural networks.

This thesis studies the application of deep learning to a number of supervised learning tasks, and in particular, classification tasks in machine perception, e.g., image recognition, automatic speech recognition, and speech emotion recognition. The relation between the systems developed for these tasks is investigated to illuminate the layer-wise relevance of features in deep networks trained for these tasks via transfer learning, and these independent systems are unified into continual learning systems.

The main contribution of this thesis is the construction and formulation of a deep learning framework, denoted progressive learning, that allows a holistic and systematic approach to continual learning. Progressive learning comprises a number of procedures that address the continual learning desiderata. It is shown that, when tasks are related, progressive learning leads to faster learning that converges to better generalization performance using less amounts of data and a smaller number of dedicated parameters, for the tasks studied in this thesis, by accumulating and leveraging knowledge learned across tasks in a continuous manner. It is envisioned that progressive learning is a step towards a fully general continual learning framework.

Acknowledgements

I am eternally grateful to all my teachers, lecturers, supervisors, and mentors. I am particularly grateful to my PhD supervisors: Professor Lawrence Cavedon and Professor Hong Ren Wu, for they have provided me the freedom to pursue my interests, were always available to provide feedback and support when needed, and were tremendously generous with their advice and mentorship.

I am thankful to Dr. Ravish Mehra and Dr. Laurens van der Maaten for hosting me at Facebook Research. My internship at Facebook was one of the highlights of this journey. I also acknowledge Associate Professor Margaret Lech for her guidance during the first year of my candidature. I am also thankful to members of the Evolutionary Computing and Machine Learning (ECML) group, collaborators, and fellow students for all the discussions and musings.

I am indebted to the Royal Melbourne Institute of Technology (RMIT) for the generous Vice-Chancellor's PhD Scholarship (VCPS). I also acknowledge the Australian National Computing Infrastructure (NCI) and NVIDIA for the computational resources. I am thankful to the School of Engineering's administrative staff, and particularly Ms. Bethany McKinnon, for helping me navigate the university's processes.

No words can express my gratitude to my dear parents Mohamed Fayek and Manal for their selfless devotion to my brother Hesham and myself; it is no coincidence that we both chose to pursue scholarly careers. To my late beloved grandparents, grandfather Ibrahim Elsayed and grandmother Aida Khatab, I will always cherish the values you instilled in me. To my lovely wife Agata, for her love and support.

List of Publications

Fayek, H. M., Cavedon, L., and Wu, H. R. (2018). On the transferability of representations in neural networks between datasets and tasks. In *Continual Learning Workshop, Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, Canada.

Fayek, H. M. (2017). MatDL: A lightweight deep learning library in MATLAB. *The Journal of Open Source Software*, 2(19):413.

Fayek, H. M., Lech, M., and Cavedon, L. (2017). Evaluating deep learning architectures for speech emotion recognition. *Neural Networks*, 92:60–68. Advances in Cognitive Engineering Using Neural Networks.

Fayek, H. M., Lech, M., and Cavedon, L. (2016b). On the correlation and transferability of features between automatic speech recognition and speech emotion recognition. In *Interspeech*, pages 3618–3622.

Fayek, H. M., Lech, M., and Cavedon, L. (2016a). Modeling subjectiveness in emotion recognition with deep neural networks: Ensembles vs soft labels. In *International Joint Conference on Neural Networks (IJCNN)*, pages 566–570.

Fayek, H. M. (2016). A deep learning framework for hybrid linguistic-paralinguistic speech systems. In *2nd Doctoral Consortium at Interspeech 2016*, pages 1–2, Berkeley, United States.

Fayek, H. M., Lech, M., and Cavedon, L. (2015). Towards real-time speech emotion recognition using deep neural networks. In *International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–5.

Contents

Abstract ii

Acknowledgements iv

List of Publications v

Contents vi

List of Figures ix

List of Tables xv

List of Algorithms xviii

List of Abbreviations xix

List of Symbols xxi

1 Introduction 1
- 1.1 Artificial Intelligence and Machine Learning . . . 2
- 1.2 Inductive Bias and Catastrophic Forgetting . . . 4
- 1.3 Scope . . . 5
- 1.4 Contributions . . . 6
- 1.5 Thesis Outline . . . 7
- 1.6 Notation . . . 8

2 Machine Learning and Deep Learning 9
- 2.1 Machine Learning . . . 10
- 2.2 Regularization . . . 13
- 2.3 Optimization . . . 16
- 2.4 Deep Learning . . . 21
- 2.5 Neural Networks . . . 23

2.6 Convolutional Neural Networks 28
2.7 Recurrent Neural Networks 30
2.8 Learning Multiple Tasks 32
2.9 Continual Learning 36
2.10 Summary 37

3 Tabula Rasa Learning **38**
3.1 Machine Perception 39
3.2 Image Recognition 40
3.2.1 Background 40
3.2.2 Experimental Setup 40
3.2.3 Results 45
3.3 Automatic Speech Recognition 45
3.3.1 Background 46
3.3.2 Automatic Speech Recognition System 47
3.3.3 Experimental Setup 49
3.3.4 Results 53
3.4 Speech Emotion Recognition 54
3.4.1 Background 54
3.4.2 Related Work 55
3.4.3 Speech Emotion Recognition System 56
3.4.4 Experimental Setup 58
3.4.5 Results 60
3.5 Discussion 69
3.6 Summary 69

4 Relevance of Features and Task Relatedness **71**
4.1 Background 72
4.2 Gradual Transfer Learning 73
4.3 Experiments in Image Recognition 75
4.3.1 Experimental Setup 75
4.3.2 Results 78
4.4 Experiments in Speech Recognition 80
4.4.1 Experimental Setup 80
4.4.2 Results 85
4.5 Discussion 89
4.6 Summary 90

5 Progressive Learning **92**
5.1 Background 93
5.2 Related Work 95

5.3 Progressive Learning . . . 97
5.3.1 Curriculum . . . 98
5.3.2 Progression . . . 99
5.3.3 Pruning . . . 101
5.4 Experiments in Image Recognition . . . 103
5.4.1 Experimental Setup . . . 103
5.4.2 Evaluation Criteria . . . 106
5.4.3 Results . . . 107
5.5 Experiments in Speech Recognition . . . 113
5.5.1 Experimental Setup . . . 115
5.5.2 Evaluation Criteria . . . 118
5.5.3 Results . . . 118
5.6 Discussion . . . 121
5.7 Summary . . . 122

6 Conclusions and Future Work 124
6.1 Conclusions . . . 125
6.2 Future Work . . . 126

Bibliography 128

A Datasets 146
A.1 CIFAR . . . 146
A.2 eNTERFACE . . . 149
A.3 IEMOCAP . . . 149
A.4 ImageNet . . . 149
A.5 SVHN . . . 150
A.6 TIMIT . . . 150

List of Figures

1.1 Conceptual taxonomy of artificial intelligence, continual learning, classical machine learning, representation learning, and deep learning. Note that this taxonomy does not imply that all classical machine learning, representation learning, and deep learning methods are continual learning methods. 3

2.1 Dropout. **Left.** A complete fully connected neural network with one hidden layer. **Right.** The same fully connected neural network with a number of omitted units. 16
2.2 Feed-forward fully connected neural network with two hidden layers. 23
2.3 Non-linear activation functions. **Left.** Logistic sigmoid function. **Centre.** Hyperbolic tangent function. **Right.** Rectified linear unit. 25
2.4 Convolutional neural network with three Convolutional (Conv) and Pooling (Pool) layers followed by a Fully Connected (FC) layer. . . 29
2.5 Recurrent neural network with two recurrent hidden layers characterized by self-connections (blue). 30
2.6 Transfer learning via a neural network with three hidden layers. **Top.** The parameters of the model were randomly initialized, and trained for Task A. **Bottom.** The parameters of the model were initialized using the parameters trained for Task A, and fine-tuned for Task B. 34
2.7 Multi-task learning with three tasks via a neural network that has three hidden layers. All layers are shared between the three tasks except the output layer which is unique to each task. 35

3.1 Acoustic speech signal. **Top.** Raw acoustic signal in the time domain. **Centre.** Normalized log Mel Frequency Spectral Coefficients (MFSCs) computed from the raw acoustic signal (top). **Bottom.** Normalized Mel Scale Cepstral Coefficients (MFCCs) computed from the raw acoustic signal (top). 48

3.2 Mel scale filter banks. The amplitudes of 40 filter banks on a Mel scale in the frequency range 0–4 kHz. 49
3.3 Overview of the proposed speech emotion recognition system. A deep multi-layered neural network, composed of several fully connected, convolutional, or recurrent layers, ingests a target frame (solid), concatenated with a number of context frames (dashed), to predict the probabilities over emotion classes corresponding to the target frame. 57
3.4 Speech emotion recognition test accuracy and test Unweighted Average Recall (UAR) on the IEMOCAP dataset with a deep neural network as a function of the number of context frames. 61
3.5 Speech emotion recognition test accuracy on the IEMOCAP dataset of a Long Short-Term Memory (LSTM)-Recurrent Neural Network (RNN) with various number of context frames. LSTM-RNN-c denotes the sequence length which the model was trained on. The number of frames denotes the sequence length which the model was evaluated on. 64
3.6 Speech emotion recognition test Unweighted Average Recall (UAR) on the IEMOCAP dataset of a Long Short-Term Memory (LSTM)-Recurrent Neural Network (RNN) with various number of context frames. LSTM-RNN-c denotes the sequence length which the model was trained on. The number of frames denotes the sequence length which the model was evaluated on. 65
3.7 Input speech utterances (top) and corresponding aligned output (below) of the speech emotion recognition system for a number of selected utterances from the test subset of the IEMOCAP dataset. The output is the probabilities over classes denoting the confidence of the model. Transcripts: **(a)**: Oh, laugh at me all you like but why does this happen every night she comes back? She goes to sleep in his room and his memorial breaks in pieces. Look at it, Joe look. (Angry); **(b)**: I will never forgive you. All I'd done was sit around wondering if I was crazy waiting so long, wondering if you were thinking about me. (Happy); **(c)**: OKay. So I am putting out the pets, getting the car our the garage. (Neutral); **(d)**: They didn't die. They killed themselves for each other. I mean that, exactly. Just a little more selfish and they would all be here today. (Sad); **(e)**: Oh yeah, that would be. Well, depends on what type of car you had, though too. I guess it would be worth it. helicopter. Yeah, helicopter. There is a helipad there, right? Yeah, exactly. (Happy). 67

4.1 Gradual transfer learning between two tasks. The parameters of the first and second models are initialized randomly (grey) and trained for Task A (blue) and Task B (green) respectively. The third and fourth models are examples of gradual transfer learning. The parameters of the third model are initialized using the trained Task A model (blue) and the final three layers are fine-tuned for Task B (green). The parameters of the fourth model are initialized using the trained Task B model (green) and all layers except the first layer are fine-tuned for Task A (blue). 74
4.2 Gradual transfer learning between the CIFAR-10 and CIFAR-100 datasets. **Left.** The validation and test accuracies of independent CIFAR-10 (dashed) and CIFAR-10 fine-tuned from CIFAR-100 (solid) as a function of the number of constant layers. **Right.** The validation and test accuracies of independent CIFAR-100 (dashed) and CIFAR-100 fine-tuned from CIFAR-10 (solid) as a function of the number of constant layers. 79
4.3 Gradual transfer learning between the CIFAR-10 and SVHN datasets. **Left.** The validation and test accuracies of independent CIFAR-10 (dashed) and CIFAR-10 fine-tuned from SVHN (solid) as a function of the number of constant layers. **Right.** The validation and test accuracies of independent SVHN (dashed) and SVHN fine-tuned from CIFAR-10 (solid) as a function of the number of constant layers. 79
4.4 Gradual transfer learning between the CIFAR-100 and SVHN datasets. **Left.** The validation and test accuracies of independent CIFAR-100 (dashed) and CIFAR-100 fine-tuned from SVHN (solid) as a function of the number of constant layers. **Right.** The validation and test accuracies of independent SVHN (dashed) and SVHN fine-tuned from CIFAR-100 (solid) as a function of the number of constant layers. 80
4.5 Learned features in the first layer of the convolutional neural network Model **A** detailed in Table 4.3 for the automatic speech recognition task (left) and the speech emotion recognition task (right). 86

4.6 Gradual transfer learning between the automatic speech recognition task and the speech emotion recognition task with convolutional neural network Model **A**. **Left.** The Phone Error Rate (PER) of independent automatic speech recognition (TIMIT) (dashed) and automatic speech recognition (TIMIT) fine-tuned from speech emotion recognition (IEMOCAP) (solid) as a function of the number of constant layers. **Right.** The Unweighted Error (UE) of independent speech emotion recognition (IEMOCAP) (dashed) and speech emotion recognition (IEMOCAP) fine-tuned from automatic speech recognition (TIMIT) (solid) as a function of the number of constant layers. 87
4.7 Gradual transfer learning between the automatic speech recognition task and the speech emotion recognition task with convolutional neural network Model **B**. **Left.** The Phone Error Rate (PER) of independent automatic speech recognition (TIMIT) (dashed) and automatic speech recognition (TIMIT) fine-tuned from speech emotion recognition (IEMOCAP) (solid) as a function of the number of constant layers. **Right.** The Unweighted Error (UE) of independent speech emotion recognition (IEMOCAP) (dashed) and speech emotion recognition (IEMOCAP) fine-tuned from automatic speech recognition (TIMIT) (solid) as a function of the number of constant layers. 88

5.1 Overview of progressive learning for three tasks. Initially, a curriculum strategy is used to select a task (blue) from the pool of candidate tasks. Second, a model is trained to perform the selected task (blue), and the learned parameters are constant thereafter. Third, the curriculum strategy is employed to select the subsequent task (purple). Fourth, new model parameters, denoted progressive block, which draw connections from the preceding layer in the block as well as the preceding layer in prior progressive block(s), are added and trained to perform the selected task (purple). Fifth, after training the newly added progressive block to convergence, a pruning procedure is used to remove weights without compromising performance. Finally, the curriculum, progression, and pruning procedures are repeated for the third task (green), and for all remaining task(s) subsequently. . 94

5.2 Procedural illustration of progressive learning for three tasks. In the first iteration, a curriculum strategy is used to select a task (blue) from the pool of candidate tasks. Then, a model is trained to perform the selected task (blue), and the learned parameters are constant thereafter. In the second iteration, the curriculum strategy is employed to select the subsequent task (purple). Then, new model parameters, denoted progressive block, which draw connections from the preceding layer in the block as well as the preceding layer in prior progressive block(s), are added and trained to perform the selected task (purple). Subsequently, after training the newly added progressive block to convergence, a pruning procedure is used to remove weights without compromising performance. In the third iteration, the curriculum, progression, and pruning procedures are repeated for the third task (green). Further iterations of the three procedures continue for all following task(s). 97
5.3 Illustration of the concatenation operation for three blocks. 101
5.4 Accuracy of progressive learning vs. independent learning for all 11 image recognition tasks. The tasks are ordered according to the outcome of the curriculum procedure. C-10 denotes the CIFAR-10 task, C-100-s indicates the sth CIFAR-100 task. 110
5.5 Learning curves, validation error as a function of training iterations, for progressive and independent learning for the 11 CIFAR-10 and CIFAR-100 tasks. Progressive Learning demonstrates faster learning compared with independent learning. Note that tasks are ordered according to the outcome of the curriculum strategy. Models are reset to random initialization at the beginning of each task in the case of independent learning. 112
5.6 Total number of parameters in the model as a function of tasks. **Left.** The total number of parameters in the model as a function of tasks before and after pruning. **Centre.** The number of fixed parameters (parameters in preceding progressive blocks) as a function of tasks before and after pruning. **Right.** The number of adaptive parameters (parameters in the progressive block being trained) as a function of tasks before and after pruning. 113
5.7 Percentage of pruned weights as a function of layers. It can be noted that initial layers are more prone to pruning as progressive blocks can rely on features learned in initial layers in prior progressive blocks. 114
5.8 Progressive learning vs independent learning for all four speech recognition tasks. The tasks are ordered according to the outcome of the curriculum procedure. 119

5.9 Learning curves, validation error as a function of training iterations, for progressive and independent learning for the four speech recognition tasks. Note that tasks are ordered according to the outcome of the curriculum strategy. Models are reset to random initialization at the beginning of each task in the case of independent learning. . 120

A.1 Sample images randomly drawn from the training set of the CIFAR-10 dataset. Each row was drawn from a single class in the following order: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. 148
A.2 Sample images randomly drawn from the training set of the SVHN dataset. Each row represents a class or a digit 0–9 respectively. . . . 151

List of Tables

3.1 Convolutional neural network Model **A** architecture for the image recognition task. K denotes the number of output classes. 43
3.2 Densely connected convolutional network Model **B** architecture for the image recognition task. The outputs of the convolutional layers in Blocks 2, 4, and 6, are concatenated with the inputs to the layer and fed to the subsequent layer in the same block. K denotes the number of output classes. 44
3.3 Image recognition Validation Classification Accuracy (Val ACC) and Test Classification Accuracy (Test ACC) on the CIFAR-10, CIFAR-100, and SVHN datasets. 46
3.4 Convolutional neural network Model **A** architecture for the automatic speech recognition task. 51
3.5 Convolutional neural network Model **B** architecture for the automatic speech recognition task. 52
3.6 Automatic speech recognition validation and test Frame Error Rate (FER) and Phone Error Rate (PER) on the TIMIT dataset. 54
3.7 Speech emotion recognition test Accuracy (ACC) and test Unweighted Average Recall (UAR) on the IEMOCAP dataset with various convolutional neural network architectures. Conv($c \times j \times k$) and Conv1D($c \times j \times k$) denote a spatial convolutional layer and a temporal convolutional layer respectively of c filters, each of size $j \times k$, with stride 2, followed by Batch Normalization (BatchNorm) and Rectified Linear Units (ReLUs). FC(n_l) denotes a fully connected layer of n_l units followed by BatchNorm, ReLUs, and dropout. All architectures have a fully connected layer with a softmax function as the output layer. 62

3.8 Speech emotion recognition test Accuracy (ACC) and Unweighted Average Recall (UAR) on the IEMOCAP dataset with various neural network architectures. Conv($c \times j \times k$) denote a spatial convolutional layer of c filters, each of size $j \times k$, with stride 2, followed by Batch Normalization (BatchNorm) and Rectified Linear Units (ReLUs). FC(n_l) denotes a fully connected layer of n_l units followed by BatchNorm, ReLUs, and dropout. LSTM-RNN(n_l) denotes a Long Short-Term Memory (LSTM)-Recurrent Neural Network (RNN) of n_l units. All architectures have a fully connected layer with a softmax function as the output layer. 66
3.9 Speech Emotion Recognition (SER) results reported in prior work on the IEMOCAP dataset. Note that differences in data subsets and other experiment conditions should be taken into consideration when comparing the following results against each other, see references for more details. 68

4.1 Densely connected convolutional network architecture for image recognition. The outputs of the convolutional layers in Blocks 2, 4, and 6, are concatenated with the inputs to the layer and fed to the subsequent layer in the same block. K denotes the number of output classes. 76
4.2 Speech recognition convolutional neural network Model **A** architecture. K denotes the number of output classes. 83
4.3 Speech recognition convolutional neural network Model **B** architecture. K denotes the number of output classes. 84
4.4 Validation and test Frame Error Rate (FER) and Phone Error Rate (PER) of gradual transfer learning from the speech emotion recognition task (IEMOCAP) to the automatic speech recognition task (TIMIT) with convolutional neural network Model **A**. 86
4.5 Validation and test error and unweighted error of gradual transfer learning from the automatic speech recognition task (TIMIT) to the speech emotion recognition task (IEMOCAP) with convolutional neural network Model **A**. 87
4.6 Validation and test Frame Error Rate (FER) and Phone Error Rate (PER) of gradual transfer learning from the speech emotion recognition task (IEMOCAP) to the automatic speech recognition task (TIMIT) with convolutional neural network Model **B**. 88
4.7 Validation and test error and unweighted error of gradual transfer learning from the automatic speech recognition task (TIMIT) to the speech emotion recognition task (IEMOCAP) with convolutional neural network Model **B**. 89

5.1 Convolutional neural network architecture for the image recognition tasks. The concatenation operation indicates the layers at which the output of the previous layer in all prior blocks are concatenated. The concatenation operation can be ignored in independent learning. 105
5.2 Progressive learning Validation Average Accuracy (Val AA) and Test Average Accuracy (Test AA) over all 11 tasks using the CIFAR-10 and CIFAR-100 datasets unless otherwise indicated. 108
5.3 Progressive learning Validation Progressive Knowledge Transfer (Val PKT) and Test Progressive Knowledge Transfer (Test PKT) over all 11 tasks using the CIFAR-10 and CIFAR-100 datasets unless otherwise indicated. 109
5.4 Convolutional neural network architecture for the speech recognition tasks. The concatenation operation indicates the layers at which the output of the previous layer in all prior blocks are concatenated. The concatenation operation can be ignored in independent learning. K denotes the number of output classes. 117

A.1 Classes in the CIFAR-100 dataset. 147

List of Algorithms

2.1 Early stopping algorithm for determining when to terminate an iterative training algorithm. 15

2.2 Stochastic gradient descent algorithm. Note that supervised learning is assumed herein, but the algorithm is valid for other types of learning that can provide means to compute the gradients. 18

2.3 Adam algorithm. ϵ is a small constant for numerical stability (e.g., $\epsilon = 10^{-8}$). The division and square root in Step 10 are applied element-wise. 20

2.4 Forward propagation through a fully connected (deep) neural network. ϕ denotes a non-linear operation applied element-wise. Note that for notational convenience, the output of the output layer in the network $\hat{\mathbf{y}}^{(L)}$ is abbreviated to $\hat{\mathbf{y}}$. The algorithm assumes a single exemplar m but can be extended to the case with a mini-batch of exemplars or the entire dataset. 27

2.5 Backward computation through a fully connected (deep) neural network. ϕ' denotes the derivative of the non-linear operation applied element-wise. $\odot$ denotes the Hadamard product. $\top$ denotes the transpose operation. λ is the regularization weight and the entire term can be ignored if the loss function does not constitute a regularization penalty Ω. Note that for notational convenience, the output of the output layer in the network $\hat{\mathbf{y}}^{(L)}$ is abbreviated to $\hat{\mathbf{y}}$. The algorithm assumes a single exemplar m but can be extended to the case with a mini-batch of exemplars or the entire dataset. . . . 28

5.6 Functional decomposition of progressive learning in the supervised learning case. 99

5.7 Greedy layer-wise pruning procedure in progressive learning. 103

List of Abbreviations

AI	Artificial Intelligence.
ANN	Artificial Neural Network.
ASR	Automatic Speech Recognition.
BatchNorm	Batch Normalization.
BPTT	BackPropagation Through Time.
ConvNet	Convolutional Neural Network.
CPU	Central Processing Unit.
DCT	Discrete Cosine Transform.
DenseNet	Densely Connected Convolutional Network.
DFT	Discrete Fourier Transform.
DNN	Deep Neural Network.
ELM	Extreme Learning Machine.
ERM	Empirical Risk Minimization.
FER	Frame Error Rate.
GMM	Gaussian Mixture Model.
GPU	Graphics Processing Unit.
GR	Gender Recognition.
HMM	Hidden Markov Model.
HOG	Histograms of Oriented Gradient.
IID	Independent and Identically Distributed.

LOSO	Leave-One-Speaker-Out.
LSTM	Long Short-Term Memory.
MAP	Maximum A Priori.
MFCC	Mel Frequency Cepstral Coefficient.
MFSC	Mel Frequency Spectral Coefficient.
MLE	Maximum Likelihood Estimation.
MLP	Multi-Layer Perceptron.
MSE	Mean Squared Error.
NN	Neural Network.
PCA	Principal Component Analysis.
PER	Phone Error Rate.
RBM	Restricted Boltzmann Machine.
ReLU	Rectified Linear Unit.
ResNet	Residual Network.
RNN	Recurrent Neural Network.
SER	Speech Emotion Recognition.
SGD	Stochastic Gradient Descent.
SIFT	Scale Invariant Feature Transform.
SR	Speaker Recognition.
SVM	Support Vector Machine.
UAR	Unweighted Average Recall.
UE	Unweighted Error.
VC	Vapnik-Chervonenkis.

List of Symbols

x	Scalar.
X	Constant.
$\mathbf{x}$	Vector.
$\mathbf{X}$	Matrix or Tensor.
$\mathcal{X}$	Set or Special Notation.
$\mathbb{X}$	Set or Special Notation.
$\mathbf{x}_i$	Element i of Vector $\mathbf{x}$.
$\mathbb{X}_i$	Element i of Set $\mathbb{X}$.
$\mathbf{X}_{ij}$	Element in Row i and Column j of Matrix $\mathbf{X}$.
$\mathbf{X}_{i:}$	Element(s) in Row i of Matrix $\mathbf{X}$.
$p(i)$	Probability Distribution Over i.
$p(i\|j)$	Conditional Probability Distribution Over i Given j.
$p(i\|j;\boldsymbol{\theta})$	$p(i\|j)$ Parametrized by Parameters $\boldsymbol{\theta}$.
$\|\|$	Concatenation Operation.
$*$	Convolution Operation.
$\mathbb{E}$	Expectation.
$\nabla\mathbf{i}$	Gradient of $\mathbf{i}$.
$\odot$	Hadamard Product.
ϵ	Small Constant.
$\top$	Transpose Operation.
$\|\|\mathbf{i}\|\|_j$	L^j Norm of $\mathbf{i}$.
$sigm$	Logistic Sigmoid Function.
$\tanh$	Hyperbolic Tangent Function.

$\mathbb{R}$	Set of Real Numbers.
$\mathbb{D}$	Dataset.
Θ	Parameter Space.
$\boldsymbol{\theta}$	Vector of All Adaptive Parameters in a Model.
$\mathbf{W}$	Matrix of Weights.
$\mathbf{W}^{(i)}$	Matrix of Weights of layer i.
$\mathbf{b}$	Vector of Biases.
L	Number of Layers in a Deep Network.
L_H	Number of Hidden Layers in a Deep Network.

Chapter 1

Introduction

ARTIFICIAL intelligence aims to understand and build intelligent entities. Machine learning is one of several approaches to artificial intelligence. It allows us to build machines that can learn from experience as opposed to being explicitly programmed. Current machine learning algorithms and methodologies are mostly designed for learning and performing a particular task from a tabula rasa using data available for the task at hand. For machine learning to converge to artificial intelligence, in addition to other desiderata, it must be in a state of continual learning, i.e., have the ability to be in a continuous learning process, such that when a new task is presented, it can leverage prior knowledge from prior tasks, in learning and performing this new task, and augment the prior knowledge with the newly acquired knowledge without having a significant adverse effect on the prior knowledge. Continual learning is key to advancing machine learning and artificial intelligence.

Outline. This chapter is structured as follows. Section 1.1 provides a brief introduction to artificial intelligence, machine learning, and continual learning. Section 1.2 describes the problem statement which this work is set to address. Section 1.3 outlines the scope of this work. Section 1.4 lists the contributions of this thesis. Section 1.5 details the thesis outline. Finally, Section 1.6 summarizes the notation used throughout the thesis.

1.1 Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) aims to understand and build intelligent entities*. Contemporary AI has been sought after ever since the inception of the first programmable digital computer [Turing, 1950]. Nonetheless, the seeds of AI were planted long before that, and can be traced to antiquity, when humans devised and solicited statues and automatons depicting gods for wisdom and emotion [McCorduck, 2004]. Later, classical philosophers delineated formal processes for logical reasoning, e.g., Aristotelian syllogism, which was built on the notion that human thought could be mechanized. This notion is one of the main assumptions in many approaches to AI [Russell and Norvig, 2003], and can be conceived using mathematical logic [Boole, 1854] implemented using programmable digital computers. Progress in the field of AI has been swift relative to its young age [McCorduck et al., 1977]. Today, there are many approaches to AI, such as symbolic reasoning and computational intelligence, that may one day lead to thinking machines that have intellectual capabilities superior to those of humans. Theoretically, thinking machines are only limited by the limits of computation and physics [Schmidhuber, 2002].

Machine learning is one of several approaches to AI. Notably, it is able to deal with tasks that are easy for humans to perform but hard for them to formalize how it was performed [Goodfellow et al., 2016], such as recognizing and converting speech into text, assessing visual aesthetics, or driving a car. Each of these tasks cannot be defined by a complete set of formal rules, and therefore programming machines to directly perform such tasks would result in error-prone, fragile, or incomplete programs. Machine learning allows machines to learn from experience as opposed to being explicitly programmed [Samuel, 1959]. Machine learning can be formally defined as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [Mitchell, 1997].

The machine learning approach relies on data to learn a model that can be used to accomplish the required task [Bishop, 1995]. This encompasses engineering a data representation, choosing an architecture for the model, deriving an appropriate *loss function*†, and selecting a suitable training algorithm. Such decisions require domain knowledge and are naturally heuristic. Engineering a representation of the data, which is known as *feature engineering*, is especially challenging, as it is difficult to determine which features should be used in advance for a particular problem;

*See [Russell and Norvig, 2003] for formal definitions of AI.

†The loss function is also known as cost function, error function, fitness function, or objective function.

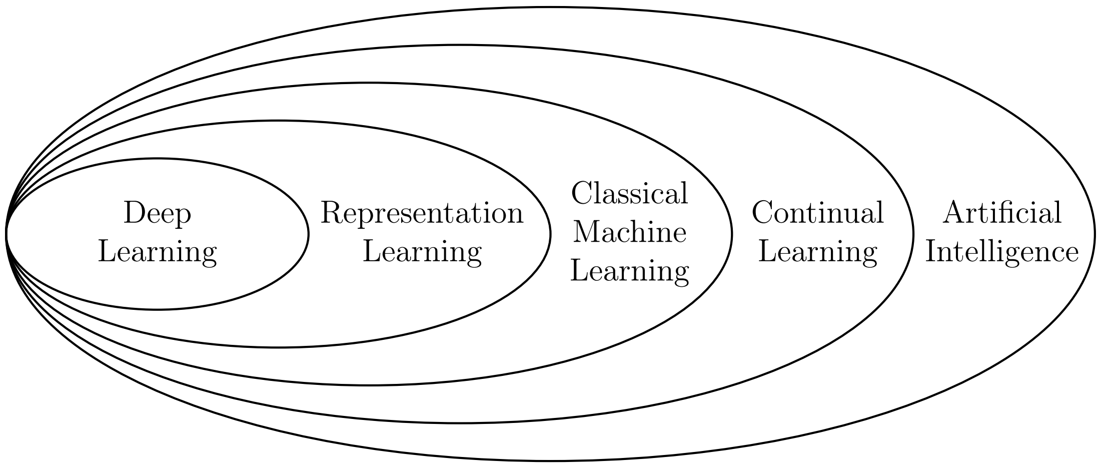

Figure 1.1: Conceptual taxonomy of artificial intelligence, continual learning, classical machine learning, representation learning, and deep learning. Note that this taxonomy does not imply that all classical machine learning, representation learning, and deep learning methods are continual learning methods.

with a poor choice of features, all subsequent effort can be futile [Mitchell, 1997]. *Representation learning* alleviates the dependence of machine learning on feature engineering by learning appropriate representations for the required task [Bengio et al., 2013]. *Deep learning* extends representation learning to learn multiple levels of representations together with the mapping function, be it classification or regression or otherwise, into a single composite function.

Deep learning is, therefore, an approach to machine learning (see Figure 1.1). It allows learning representations and concepts in a hierarchical way. This can be achieved by decomposing computational models or graphs into multiple layers of processing, with the aim of learning representations of the data with multiple levels of abstraction [LeCun et al., 2015]. In doing so, the model can adaptively learn low-level features from raw data and higher-level features from the low-level features in a hierarchical manner [Hinton et al., 2006]. Notably, deep learning presents itself as a general-purpose approach to machine learning that is able to solve numerous and various tasks with minimal modification. At present, deep learning is the state-of-the-art approach to machine learning, and is prominent in numerous fields, such as *computer vision*, *speech recognition*, and *natural language processing*.

Neural networks are a class of machine learning systems[‡] [Rosenblatt, 1958, Rumelhart and McClelland, 1986]. A neural network is a collection of nodes or units that are inter-connected via adaptive weights to form a directed weighted

[‡]Artificial Neural Networks (ANNs) are also referred to as Multi-Layer Perceptrons (MLPs).

graph, which can learn distributed representations and ultimately the task at hand. A Deep Neural Network (DNN), a neural network with many hierarchical layers, is the most prevalent example of deep learning, where one layer feeds the subsequent layer, such that initial layers learn simple features from the data and subsequent layers learn more complex features using the simple features in initial layers.

Learning from a *tabula rasa* is the most common machine learning paradigm [Mikolov et al., 2018, Lake et al., 2017]. For an arbitrary task, a model is initialized and trained using a dataset or environment to achieve a certain objective. Contrary to human learning, the model does not typically take into account knowledge learned in prior related tasks that used prior datasets or environments, which could lead to a slower learning process that requires more data and possibly suboptimal performance [Lake et al., 2017].

For machine learning to converge to AI, in addition to other desiderata, it must be in a state of *continual learning* [Thrun and Mitchell, 1995, Mitchell et al., 2018], i.e., have the ability to be in a continuous learning process, such that when a new task is presented, it can leverage prior knowledge from prior tasks, in learning and performing this new task, and augment the prior knowledge with the newly acquired knowledge without having a significant adverse effect on the prior knowledge. Quintessentially, humans are always in a state of continual learning [Harlow, 1949, Smith et al., 2002, Dewar and Xu, 2010]. For instance, when presented with the task of learning a new language, one will automatically draw from one's own past experiences with languages they are familiar with, to facilitate the learning of this new language, e.g., English speakers would naturally use their knowledge of the English syntax when learning French.

Within machine learning, deep learning and neural networks are particularly suited for continual learning, due to their unprecedented success in numerous and various tasks with minimal modification and their innate ability to learn multiple hierarchical levels of versatile distributed representations. For example, a unit in a neural network that has learned the concept of a wheel in a bicycle detection task can be used in learning to detect cars in a self-driving car task, which avoids the need to re-learn the concept of the wheel.

Continual learning is key to advancing machine learning and AI, where knowledge can be accumulated, repurposed, and reused over tasks.

1.2 Inductive Bias and Catastrophic Forgetting

Every machine learning algorithm with the ability to generalize beyond the data which it has encountered during training has, by definition, some form of prior, known as *inductive bias* [Mitchell, 1980]. Inductive bias is the set of assumptions that the model relies on to generalize beyond the data it has not encountered during training. For example, *linear regression* assumes that the relation between the independent variables (input) and dependent variables (output) is linear.

Ideally, one aspires to minimize the inductive bias in a machine learning algorithm. There is, however, a trade-off between the inductive bias of an algorithm and the amount of data required to ensure reliable *generalization* to unseen data [Mitchell, 1980, Baxter, 2000], in that the search space for a solution needs to be large enough to contain a solution to the task, and small enough to not require a large dataset to navigate the search space during training. Revisiting the previous example regarding linear regression, by assuming that the relation between the independent variables and dependent variables is linear, all non-linear solutions are effectively eliminated from the search space; the search space is simpler yet may not contain the solution to the task if a non-linear solution is necessary.

Deep learning and neural networks require large datasets to learn a given task due to the large search space inherent in their design. Inductive bias can play an important role in navigating this large search space during training, and thus, alleviate the requirement of large datasets which typically leads to a slower learning process.

Continual learning is in itself a form of inductive bias. However, it can be considered a soft form of inductive bias, whereby the model can rely on existing solutions to related tasks in its search for a solution in the search space, or use data available for the task at hand to navigate the search space, or both. In doing so, one can expect a faster learning process and better generalization, using less amounts of data.

It can be seen how continual learning can be used as a form of inductive bias to improve the learning and execution of new tasks given prior related tasks; however, this should not be at the expense of corrupting the solutions to prior tasks.

Catastrophic forgetting in machine learning is the tendency to forget previously learned information when learning new information [McCloskey and Cohen, 1989, Ratcliff, 1990]. Catastrophic forgetting is evident in parametric machine learning algorithms, where parameters learned for old task(s) are repurposed for the new task at hand, and the model forgets the old task(s) as the parameters move in the search space during the training phase for the new task.

Inducing the correct inductive bias and mitigating catastrophic forgetting in an efficient manner are two of the main challenges in continual learning.

1.3 Scope

Continual learning applies to all branches of machine learning, including *supervised learning*, *unsupervised learning*, and *reinforcement learning*[§]. The focus of this

[§]See Section 2.1 for the definitions of the classifications of machine learning.

work lies at the intersection between continual learning and deep learning. The class of tasks studied in this work are all cast as supervised learning problems, and in particular, classification tasks in the image recognition and speech recognition domains, both of which are sub-domains of *machine perception*. Nevertheless, the propositions and conclusions put forward herein are not confined to such learning tasks, and are envisaged to be applicable to other branches of machine learning, particularly those that utilize deep learning.

1.4 Contributions

The contributions of the thesis are as follows.

- The application of deep learning requires a plethora of design decisions and extensive tuning of hyperparameters to ensure good performance, e.g., choosing the architecture of the neural network, including the number of layers and size of each layer. These design decisions and hyperparameters are cumbersome and are usually task-specific or domain-specific. It is therefore advantageous to understand some of these design decisions and hyperparameters across a variety of tasks. To this end, various tasks, namely, image recognition, Automatic Speech Recognition (ASR), and Speech Emotion Recognition (SER), were formulated and studied in a systematic manner. The SER task was comprehensively considered and used as a test bed to explore various neural network architectures as the task itself can be formulated in multiple ways. This is the first empirical exploration of various deep learning formulations and architectures applied to SER. Empirical analysis provided intuition and insights on the effectiveness of some of these architectures and their suitability to particular tasks. As a consequent result of the systematic exploration, state-of-the-art results were reported on the Interactive Emotional Dyadic Motion Capture (IEMOCAP) dataset [Busso et al., 2008] for speaker-independent SER[¶].

- In order to move from single-task systems to systems that aim to address multiple tasks, one ought to understand the relation between these tasks. A methodology for understanding the relation between two tasks using the features learned for each task in a deep network is proposed in this work, to illuminate the relevance of each layer of features in the multi-layered neural network trained for one task to the other task via transfer learning. Understanding the layer-wise transferability of features and task relatedness is envisaged to be valuable in designing and implementing systems that

[¶]Compared to prior literature when the work was published in [Fayek et al., 2017].

aim to address multiple tasks, by taking into consideration the overlap between closely related tasks as well as the interference between non-related or adversary tasks.

- Continual learning is a machine learning paradigm, whereby tasks are learned in sequence with the ability to use prior knowledge from previously learned tasks to facilitate the learning and execution of new ones. The main contribution of this work is the construction and formulation of a novel deep learning framework, named *progressive learning*, that allows a holistic and systematic approach to continual learning. Progressive learning comprises a number of procedures that address the continual learning desiderata. It is shown that, when tasks are related, progressive learning leads to faster learning that converges to better generalization performance using less amounts of data and a smaller number of dedicated parameters, for the tasks studied in this thesis, by accumulating and leveraging knowledge learned across tasks in a continuous manner.

The above three contributions are put forward in Chapters 3, 4 and 5 respectively[‖].

1.5 Thesis Outline

The thesis is structured as follows.

Chapter 1 presents a brief introduction to AI, machine learning, and continual learning, and an outlook on the state-of-the-art; provides the problem statement which this thesis is set to address; defines the scope and highlights contributions of this work, as well as the thesis outline and notation used.

Chapter 2 reviews machine learning, deep learning, and continual learning. The concepts, theory, and mathematical framework that constitute the foundations of this work are detailed therein.

Chapter 3 builds understanding, insights, and baselines of the various tasks used throughout this work, as well as datasets, deep learning architectures, and best practices. The SER task is comprehensively considered and used as a test bed

[‖]An additional contribution is the development and release of an open-source lightweight deep learning library, called *MatDL* [Fayek, 2017]. MatDL was developed natively in Matlab and implements some commonly used deep learning building blocks and algorithms. MatDL is convenient in cases where Matlab is preferred, or if it is required to be closely linked with other libraries written in Matlab or Octave. MatDL is ideal for rapid machine learning research and experimentation, specially with small or medium-sized datasets, as it was designed with an emphasis on modularity, flexibility, and extensibility.

to explore various neural network architectures. The work presented in this chapter is based on [Fayek et al., 2015, Fayek et al., 2016a, Fayek et al., 2017, Fayek, 2017].

Chapter 4 is an investigation into the specificity and transferability of learned features in deep networks across multiple tasks, and how this can be used to understand task relatedness. The aim of this chapter is to gain intuition into how information propagates in deep networks that can be used to build systems with multiple tasks. The work presented in this chapter is based on [Fayek, 2016, Fayek et al., 2016b, Fayek et al., 2018].

Chapter 5 builds on work carried out in the previous two chapters and proposes progressive learning, a novel deep learning framework that formulates continual learning into three procedures: *curriculum*, *progression*, and *pruning*. Progressive learning is evaluated on a number of tasks in the image recognition and speech recognition domains that were studied in the previous chapters to demonstrate its advantages compared with baseline methods.

Chapter 6 concludes the thesis and highlights avenues for future work.

1.6 Notation

The following notation is used throughout this thesis unless otherwise specified. Standard weight lower-case letters (e.g., x) are used to denote scalars. Standard weight upper-case letters (e.g., X) are used to denote constant scalars. Boldface lower-case letters (e.g., $\mathbf{x}$) are used to denote vectors. Boldface upper-case letters (e.g., $\mathbf{X}$) are used to denote matrices or tensors. Standard weight upper-case calligraphic or open-face letters (e.g., $\mathcal{X}$ or $\mathbb{X}$) are used to denote sets and special notation.

Single subscripts are used to denote an element in a vector or set (e.g., $\mathbf{x}_i$ or $\mathbb{X}_i$). Double (or triple or more, depending on the order) subscripts are used to denote an element or elements in a matrix or tensor (e.g., $\mathbf{X}_{ij}$ or $\mathbf{X}_{ijk}$), where the symbol ":" may be used to denote the entire row, or column, or dimension (e.g., $\mathbf{X}_{i:}$ for row i in matrix $\mathbf{X}$). Single superscripts or subscripts in parentheses are used to differentiate variables (e.g., $\mathbf{W}^{(i)}$ can refer to the matrix of weights of layer i).

Chapter 2

Machine Learning and Deep Learning

MACHINE learning can be formally defined as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [Mitchell, 1997]. Following this definition, the concepts, theory, and mathematical framework that constitute the foundations of this work are detailed in this chapter, including statistical machine learning, regularization, optimization, deep learning, neural networks, learning multiple tasks, and continual learning.

Outline. This chapter is structured as follows. Section 2.1 provides the assumptions, theory, formulation, and mathematical framework of statistical machine learning. Section 2.2 presents popular regularization techniques for improving the generalization of machine learning models. Section 2.3 describes gradient-based optimization algorithms relevant to training neural networks. Section 2.4 presents the motivation, background, and fundamentals of deep learning. Section 2.5 is an exposition of feed-forward fully connected neural networks. Section 2.6 presents convolutional neural networks. Section 2.7 describes recurrent and long short-term memory neural networks. Section 2.8 outlines machine learning paradigms that incorporate learning multiple tasks. Section 2.9 introduces continual learning and reviews selected prior literature. Finally, Section 2.10 summarizes the chapter.

2.1 Machine Learning

Machine learning can be broadly classified into supervised learning, reinforcement learning, and unsupervised learning [Bishop, 2006, Russell and Norvig, 2003, Barber, 2012]. Supervised learning relies on labels or targets in labelled datasets for the training signal during the training phase. Reinforcement learning uses a reward function, that can provide positive or negative values to indicate desired or undesired outcomes, often associated with actions, to obtain the training signal during the training phase. Unsupervised learning does not require extra information, such as labels, targets, or a reward function, for the training signal. Other classifications may include other branches of machine learning, e.g., *semi-supervised learning*, which can be interpreted as a variation of, or a combination of two or more of, the previously mentioned branches. The remainder of this section focuses on supervised learning.

In supervised learning, the objective is to learn a mapping function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$. In *statistical learning theory* [Vapnik, 2000], the relation between the input space $\mathcal{X}$ and the output space $\mathcal{Y}$ is assumed to be governed by a data-generating probability distribution p_{data}. The true data-generating probability distribution p_{data} is usually unknown; an empirical data-generating probability distribution $\hat{p}_{data}$ can be used instead, following the Empirical Risk Minimization (ERM) principle [Vapnik, 2000].

The objective can thus be decomposed into two sub-objectives: an explicit objective and an implicit objective. The explicit objective is to learn the function $\mathbf{y} = f(\mathbf{x})$ using a training set $\mathbb{D}^{(train)} \sim \hat{p}_{data}$ drawn from the empirical data-generating probability distribution $\hat{p}_{data}$, composed of M exemplars $\mathbf{X} \in \mathbb{R}^{M \times N}$ and corresponding labels or targets $\mathbf{Y} \in \mathbb{R}^{M \times K}$, such that N and K are the dimensionalities of the input and output respectively[*], $\mathbf{x} \in \mathbf{X}$, and $\mathbf{y} \in \mathbf{Y}$. The implicit objective, however, is generalization; that is to learn the function f that generalizes to unseen data $\mathbb{D}^{(test)}$, i.e., perform the same mapping task well on data not observed during training. Note that $\mathbb{D}^{(train)}$ and $\mathbb{D}^{(test)}$ are mutually exclusive.

Statistical learning theory relies on two assumptions to generalize beyond the exemplars observed in training. The first assumption is that the training set $\mathbb{D}^{(train)}$ and the test set $\mathbb{D}^{(test)}$ are from the same data-generating probability distribution that characterizes the task t. The second assumption is that each exemplar $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})$ for $1 \leq m \leq M$ in the dataset $\mathbb{D}^{(train)}$ is an Independent and Identically Distributed (IID) exemplar drawn from the probability distribution $\hat{p}_{data}$.

In the probabilistic perspective of machine learning, the function f is set to estimate the probability $p(\mathbf{y} \mid \mathbf{x})$, or more specifically $p(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta})$ in the parametric

[*]Note that N and K are loosely defined such that $\boldsymbol{X}$ and $\boldsymbol{Y}$ may be of lower or higher rank.

case, where f is parametrized by parameters[†] $\boldsymbol{\theta}$. To achieve the explicit objective, the function f is trained to minimize a loss function ℓ, where the loss function ℓ is related to, but not necessarily exactly equal to, the true objective o of the task t. The loss function ℓ, used in training the function f parametrized by parameters $\boldsymbol{\theta}$, can follow the conditional Maximum Likelihood Estimation (MLE) principle to estimate $\boldsymbol{\theta}$, i.e.,

$$\boldsymbol{\theta}_{ML} = \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} p(\mathbf{Y} \mid \mathbf{X}; \boldsymbol{\theta}), \tag{2.1}$$

where $p(\mathbf{Y} \mid \mathbf{X}; \boldsymbol{\theta})$ is the conditional probability, i.e., predict $\mathbf{Y}$ given $\mathbf{X}$, the subscripts in $\boldsymbol{\theta}_{ML}$ denote maximum likelihood, and Θ is the parameter space. Equation (2.1) can be decomposed into Equation (2.2) assuming that all exemplars $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})$ in the training set $\mathbb{D}^{(train)}$ are IID as follows:

$$\begin{aligned}\boldsymbol{\theta}_{ML} &= \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \prod_{m=1}^{M} p(\mathbf{y}^{(m)} \mid \mathbf{x}^{(m)}; \boldsymbol{\theta}), &(2.2)\\ &= \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \sum_{m=1}^{M} \log p(\mathbf{y}^{(m)} \mid \mathbf{x}^{(m)}; \boldsymbol{\theta}). &(2.3)\end{aligned}$$

Of-course, maximizing the log-likelihood in Equation (2.3) is equivalent to minimizing the negative log-likelihood, which is more commonly used as the loss function ℓ that is minimized during training:

$$\begin{aligned}\boldsymbol{\theta}_{ML} &= \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \frac{1}{M} \sum_{m=1}^{M} -\log p(\mathbf{y}^{(m)} \mid \mathbf{x}^{(m)}; \boldsymbol{\theta}), &(2.4)\\ &= \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} -\mathbb{E}_{(\mathbf{x}^{(m)},\mathbf{y}^{(m)})\sim\mathbb{D}^{(train)}} \log p(\mathbf{y}^{(m)} \mid \mathbf{x}^{(m)}; \boldsymbol{\theta}), &(2.5)\end{aligned}$$

where $\mathbb{E}_{(\mathbf{x}^{(m)},\mathbf{y}^{(m)})\sim\mathbb{D}^{(train)}}$ denotes the expectation over the training set $\mathbb{D}^{(train)}$. Additional terms can be added to Equations (2.4) and (2.5) to improve generalization as discussed in Section 2.2. Optimization algorithms used to minimize ℓ are discussed in Section 2.3. Equation (2.4) will be revisited in Section 2.5 in the context of neural networks.

Using the principles of MLE, it can be shown that as the number of exemplars in the training set $\mathbb{D}^{(train)}$ approaches infinity, $M \to \infty$, the maximum likelihood estimate of the parameters converges to the true value of the parameters with arbitrary precision, according to the property of *consistency*, under mild conditions; the implicit objective may be achieved, in expectation, relying on the assumption

[†]Parameters $\boldsymbol{\theta}$ are all the adaptive variables in a machine learning model that can be trained during the training phase, e.g., weights and biases in a neural network, where $\boldsymbol{\theta} \in \Theta$, and Θ is the parameter space.

that both the training set $\mathbb{D}^{(train)}$ and test set $\mathbb{D}^{(test)}$ were drawn from the same data-generating probability distribution. This forms the basis of many supervised learning algorithms [Goodfellow et al., 2016].

The explicit objective error, i.e., the error computed on the training set $\mathbb{D}^{(train)}$, is referred to as the *training error*, whereas the implicit objective error, i.e., the error computed on the test set $\mathbb{D}^{(test)}$, is referred to as the *test error* or *generalization error*. *Under-fitting* refers to the case when the training error is large. *Over-fitting* refers to the case when the training error is small but the test error is large. The trade-off between under-fitting and over-fitting can be handled by altering the *capacity* of the function f. The capacity of a function is loosely a measure of its complexity, which pertains to its ability to fit a wide variety of solutions. For example, in classification, a popular measure of the capacity of a classification function is the Vapnik-Chervonenkis (VC) dimension. The VC dimension for a binary classifier is the largest number of exemplars M in a training set $\mathbb{D}^{(train)}$ that the classifier can perfectly model without any misclassification error. A function f with low capacity will tend to under-fit, i.e., not be able to model the training set, while a function f with high capacity will tend to over-fit, i.e., memorize the training set including noise in a way that harms generalization.

Not all supervised learning algorithms and techniques conform to the above views and formalisms. For example, Support Vector Machines (SVMs) are non-probabilistic linear models that belong to a wider family of models known as kernel methods [Schölkopf and Smola, 2001]. Kernel methods and SVMs employ the *kernel trick* [Cortes and Vapnik, 1995], which enables them to operate in an implicit high-dimensional feature space without ever computing that space, but rather by simply carrying out the computation in some other space, e.g., computing the dot products between data exemplars. The kernel trick allows such methods to learn non-linear functions with respect to the input using convex optimization techniques by appropriately choosing a suitable kernel function. Another example is *k-nearest neighbours*, a family of non-parametric techniques that can be used for classification or regression. There is no training phase in k-nearest neighbours; instead, at test time, for a given input $\mathbf{x}$, the k-nearest neighbours to $\mathbf{x}$ in the training set $\mathbb{D}^{(train)}$ are located and the average of the corresponding $\mathbf{y}$ values in the training set is returned, where the k-nearest neighbours can be determined according to a distance metric, e.g., Euclidean distance. *Decision trees* are another family of learning algorithms that break the input space into regions and assign separate parameters for each region [Breiman et al., 1984]. Decision trees can be used for classification or regression, where branches can represent conjunctions of features that lead to leaves, and leaves can represent class labels in classification problems or take on continuous values in regression problems.

The learning algorithms above have their own advantages and limitations, and are suitable for certain types of problems. The *no free lunch theorem* for machine learning states that, averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved exemplars [Wolpert, 1996]. At first glance, this may seem disappointing, but fortunately, if assumptions about the kinds of probability distributions encountered in real-world applications are made, then learning algorithms that perform well on these distributions can be designed [Goodfellow et al., 2016].

2.2 Regularization

Regularization refers to any modification to the learning algorithm that intends to reduce the test error but not the training error [Bishop, 1995], i.e., strategies that aim to combat over-fitting. There are many regularization strategies, of which many abide by the following principle[‡]: among competing hypotheses that explain observations well, one should choose the simplest hypothesis [Vapnik and Chervonenkis, 2015]. The following strategies are some of the most popular regularization strategies in the literature that were used in this work.

Data Augmentation. The simplest way to obtain better generalization error of a machine learning model is to increase the size of the training set. This may not be practical as the process of obtaining additional data can be cumbersome. Nevertheless, additional artificial data could be generated by augmenting the training set with transformations applied to the original data in the training set. For example, in classification, and particularly, image recognition, simple transformations applied to the images in the training set, such as flipping, rotation, cropping, and scaling, can easily generate additional images given that the resultant image has the same label as the original image [Bishop, 2006, Krizhevsky et al., 2012]. Similarly, in speech recognition, label-persevering operations on speech utterances, such as resampling or vocal tract length perturbation, were found to be effective in improving generalization error [Jaitly and Hinton, 2013, Fayek et al., 2015]. Many data augmentation methods can be integrated into the training pipeline, and artificial exemplars can be generated online during training, making these methods popular regularization methods.

Parametric Norm Penalties. Penalties that are a function of some or all of the model parameters, e.g., norm penalties, can be added to the loss function to

[‡]This is commonly referred to as *Occam's razor*, which appears to have been adapted from: "Pluralitas non est ponenda sine neccesitate": Plurality should not be posited without necessity, or "Frustra fit per plura quod potest fieri per pauciora": It is futile to do with more things that which can be done with fewer [Thorburn, 1918].

penalize certain solutions over others. A norm penalty can be added to a loss function ℓ as follows:

$$\mathcal{L}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y}) = \ell(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y}) + \lambda\Omega(\boldsymbol{\theta}), \tag{2.6}$$

where $\mathcal{L}$ denotes the regularized loss function, λ is a hyperparameter[§] to weigh both terms, and Ω is a norm penalty, e.g., the L^1 norm or the L^2 norm.

The L^1 norm penalty can be computed as follows:

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_i |\boldsymbol{\theta}_i|. \tag{2.7}$$

Note that Equation (2.7) implicitly regularizes the parameters $\boldsymbol{\theta}$ towards zero; other values $\tilde{\boldsymbol{\theta}}$ could be chosen by using the full expression of $\|\boldsymbol{\theta}\|_1$, that is $\sum_i |\boldsymbol{\theta}_i - \tilde{\boldsymbol{\theta}}_i\ |$. The L^1 norm penalty encourages the parameters $\boldsymbol{\theta}$ to be sparse.

The L^2 norm penalty, which is also known as *weight decay*, can be computed as follows:

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_2^2 = \frac{1}{2}\left(\sqrt{\sum_i \boldsymbol{\theta}_i^2}\right)^2 = \frac{1}{2}\sum_i \boldsymbol{\theta}_i^2. \tag{2.8}$$

where, the multiplication by $1/2$ and exponentiation by 2 are used to simplify the computation of the derivative of the function. Similar to Equation (2.7), Equation (2.8) can be generalized to regularize the parameters $\boldsymbol{\theta}$ towards values other than zero. The L^2 norm penalty encourages the parameters $\boldsymbol{\theta}$ to have a small magnitude but not necessarily exactly zero. It is interesting to note that L^2 regularization can be interpreted as having a Gaussian prior on the parameters $\boldsymbol{\theta}$, which links MLE to the Maximum A Priori (MAP) approximation (see [Graves, 2008] for more details).

Noise Injection. Injecting noise to the inputs during training as well as to the parameters $\boldsymbol{\theta}$ is a well-established method to improve generalization. Adding noise to the inputs is a form of data augmentation and can aid the model in dealing with noisy exemplars. Adding noise to some or all of the parameters $\boldsymbol{\theta}$ during training may not only aid in escaping local minima during optimization but can also force the model to learn parameters that are resilient to small variations [Graves, 2013].

Early Stopping. In the initial stages of training, the training error typically decreases rapidly, and the speed at which the error decreases declines as training progresses; given sufficient time, it is likely to almost plateau. Should the test

[§]Hyperparameters are the set of typically non-adaptive variables that control the behaviour of a learning algorithm.

Algorithm 2.1 Early stopping algorithm for determining when to terminate an iterative training algorithm.

```
Require: Initial parameters $\boldsymbol{\theta}$
Require: Number of steps between validation evaluations $i$
Require: Patience $q$
 1: Initialize:
    $j \leftarrow 0$
    $e \leftarrow \infty$
    $\boldsymbol{\theta}^\star \leftarrow \boldsymbol{\theta}$
 2: while $j < q$ do
 3:   Update $\boldsymbol{\theta}$ for $i$ steps        // see Section 2.3 for training algorithms
 4:   $e' \leftarrow \text{ValidationError}(\boldsymbol{\theta})$
 5:   if $e' < e$ then
 6:     $j \leftarrow 0$
 7:     $\boldsymbol{\theta}^\star \leftarrow \boldsymbol{\theta}$
 8:     $e \leftarrow e'$
 9:   else
10:     $j \leftarrow j + 1$
11:   end if
12: end while
13: return Parameters $\boldsymbol{\theta}^\star$
```

error be measured throughout training, and if the model has enough capacity to over-fit the training set, the test error will also decrease rapidly during the initial stages of training, and the speed at which the test error decreases will decline as training progresses; however, the test error can subsequently increase as the model begins to over-fit the training data, and the gap between the training error and test error can continue to increase as training progresses. Thus, a better test error can be obtained at the point before the test error begins to increase. This can be achieved by using a *validation set*, $\mathbb{D}^{(val)}$, which is a mutually exclusive subset from the training set, to monitor the generalization error, in this case the *validation error*, and return the model that has the lowest validation error during training, as opposed to the model obtained when training ceases[¶]. This is known as *early stopping* as training can be halted as soon as over-fitting appears to occur. An algorithmic view of early stopping is listed in Algorithm 2.1. Early stopping is one of the most commonly used regularization methods.

[¶]A validation set is also useful in other scenarios, e.g., exploring and validating hyperparameters.

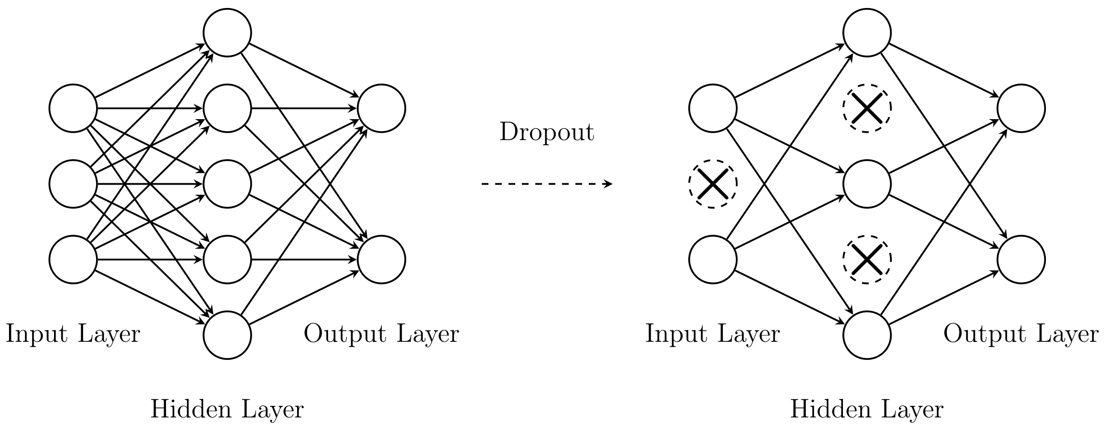

Figure 2.1: Dropout. **Left.** A complete fully connected neural network with one hidden layer. **Right.** The same fully connected neural network with a number of omitted units.

Dropout. Dropout is a powerful yet simple regularization method [Srivastava et al., 2014]. Unlike the methods above, it is specifically designed for neural networks. The key idea in dropout is to stochastically omit units along with their connections with probability r, where r is a hyperparameter, from the neural network at each iteration during training, as illustrated in Figure 2.1. This prevents co-adaptation of units during training, as units cannot rely on other units as they may be omitted at various iterations during training. The full neural network is used at test time with the magnitude of the connections in the model adjusted proportionally to r to compensate for the fan-in and fan-out of the units. Dropout can be loosely interpreted as training an ensemble of narrower models sampled from the original model during training and using the original model at test time.

2.3 Optimization

Within the formalism adopted in Section 2.1, training machine learning models is formulated — in most but not all cases — as an optimization problem that aims to minimize a loss function $\mathcal{L}$ using a training set $\mathbb{D}^{(train)}$ with the objective of generalization to unseen data. The remainder of this section focuses on iterative gradient-based training algorithms that are suited to neural networks with loss functions that are differentiable with respect to all parameters $\boldsymbol{\theta}$ in the model to obtain the gradients (see [Williams, 1992, Sutton et al., 2000] for algorithms that deal with the non-differentiable case).

Gradient Descent. *Gradient descent*, also known as *steepest descent*, is a first-order iterative optimization algorithm. Assuming that the loss function $\mathcal{L}$ is differentiable with respect to all parameters $\boldsymbol{\theta}$ in the model, the gradient descent algorithm iteratively updates parameters $\boldsymbol{\theta}$ proportional to the negative gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\mathbf{X};\boldsymbol{\theta}),\mathbf{Y})$, as in Equation (2.9), until a stopping criterion is met:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}}\mathcal{L}\left(f\left(\mathbf{X};\boldsymbol{\theta}\right),\mathbf{Y}\right), \tag{2.9}$$

where α is the learning rate, and the update is applied independently to each parameter in $\boldsymbol{\theta}$.

It is important to note that gradient descent does not guarantee convergence to the global optimum, or even a local optimum, in non-convex problems characterized by the existence of many local minima. Moreover, the quality of the results can be dependent on the quality of the initial values of the model parameters $\boldsymbol{\theta}$, which makes the initialization of parameters an important exercise [Mishkin and Matas, 2015], as discussed below. Furthermore, in problems that contain plateaus, saddle points, or steep curvatures, gradient descent may require an excessive amount of time to converge to a solution, and careful tuning of the learning rate α can be crucial to its success [Dauphin et al., 2014]. This forms the motivation for algorithms such as RMSProp [Tieleman and Hinton, 2012, Dauphin et al., 2015] and Adam [Kingma and Ba, 2014] that aim to adapt the learning rate for each parameter individually during training, as discussed later in this section.

Initialization. Initialization is an important step for any iterative local-search optimization algorithm, e.g., gradient descent, dealing with non-convex problems. Initialization is particularly important in multi-layered computational differentiable graphs, such as neural networks, as initializing the parameters with values that are too small or too large may lead to unstable gradients, and consequently the optimization process may never converge [Hochreiter, 1991, Hochreiter, 1998]. Initialization strategies have been devised for various computational architectures that will be discussed in Section 2.5.

Stochastic Gradient Descent. Considering Equation (2.4) and Equation (2.9), each update requires computing the loss function $\mathcal{L}$ over the entire training set $\mathbb{D}^{(train)}$. If the training set $\mathbb{D}^{(train)}$ is considerably large, each update would require a large amount of computation. Therefore, it would be advantageous to compute the gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ using only a small subset, called mini-batch, of M_b exemplars sampled from the training set. This will only provide an estimate of the true gradients at each update [LeCun et al., 1998], hence, the algorithm is called Stochastic Gradient Descent (SGD). The stochasticity in SGD is also beneficial for escaping local minima during training, since the landscape of the loss function might

Algorithm 2.2 Stochastic gradient descent algorithm. Note that supervised learning is assumed herein, but the algorithm is valid for other types of learning that can provide means to compute the gradients.

```
Require: Training set $\mathbb{D}^{(train)}$ of exemplars $\left(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\right)$
Require: Initial parameters $\boldsymbol{\theta}$
Require: Learning rate $\alpha$
Require: Mini-batch size $M_b$
1: while stopping criteria not met do
2:     Sample mini-batch $\{\left(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\right) \mid 1 \leq m \leq M_b\} \sim \mathbb{D}^{(train)}$
3:     Update parameters $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \left((1/M_b) \sum_{m=1}^{M_b} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\mathbf{x}^{(m)}; \boldsymbol{\theta}\right), \mathbf{y}^{(m)}\right)\right)$
4: end while
5: return Trained parameters $\boldsymbol{\theta}$
```

vary for each mini-batch [Bottou, 2004]. Algorithm 2.2 lists the steps involved in the SGD algorithm.

Momentum. The addition of a momentum term is a popular modification to the original SGD algorithm. The momentum term can accelerate the learning process, especially in the face of noisy gradients or high curvature in the landscape of the loss function. The momentum term accumulates an exponentially decaying moving average of the gradients and influences the update in their direction as follows:

$$\bar{\boldsymbol{\theta}} \leftarrow \eta\bar{\boldsymbol{\theta}} - \alpha\left(\frac{1}{M_b}\sum_{m=1}^{M_b} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\mathbf{x}^{(m)}; \boldsymbol{\theta}\right), \mathbf{y}^{(m)}\right)\right), \tag{2.10}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \bar{\boldsymbol{\theta}}, \tag{2.11}$$

where $\bar{\boldsymbol{\theta}}$ is a decaying moving average of the gradients initialized to zero, η is the decay rate, and α is the learning rate.

Nesterov Momentum. A popular variant of the momentum algorithm is the Nesterov momentum algorithm [Sutskever et al., 2013]. The difference between the standard momentum algorithm and the Nesterov momentum algorithm is where the gradients are evaluated. In the Nesterov momentum algorithm, the gradients are evaluated after the momentum term is applied as follows:

$$\bar{\boldsymbol{\theta}} \leftarrow \eta\bar{\boldsymbol{\theta}} - \alpha\left(\frac{1}{M_b}\sum_{m=1}^{M_b} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\mathbf{x}^{(m)}; \boldsymbol{\theta} + \eta\bar{\boldsymbol{\theta}}\right), \mathbf{y}^{(m)}\right)\right), \tag{2.12}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \bar{\boldsymbol{\theta}}. \tag{2.13}$$

This can be interpreted as adding a correction factor to the standard momentum algorithm.

RMSProp. The RMSProp algorithm is a first-order adaptive variant of SGD [Tieleman and Hinton, 2012, Dauphin et al., 2015], which aims to adjust the learning rate per-parameter using an exponentially decaying moving average of the squared gradients, as in Equation (2.14) and Equation (2.15).

$$\bar{\boldsymbol{\theta}} \leftarrow \eta\bar{\boldsymbol{\theta}} + (1-\eta)\left(\frac{1}{M_b}\sum_{m=1}^{M_b}\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(f\left(\mathbf{x}^{(m)};\boldsymbol{\theta}\right),\mathbf{y}^{(m)}\right)\right)^2, \tag{2.14}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha}{\sqrt{\bar{\boldsymbol{\theta}}+\epsilon}} \odot \left(\frac{1}{M_b}\sum_{m=1}^{M_b}\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(f\left(\mathbf{x}^{(m)};\boldsymbol{\theta}\right),\mathbf{y}^{(m)}\right)\right), \tag{2.15}$$

where $\bar{\boldsymbol{\theta}}$ is a decaying moving average of the squared gradients initialized to zero, η is the decay rate, α is the learning rate, ϵ is a small constant for numerical stability (e.g., $\epsilon = 10^{-8}$), $\odot$ denotes the Hadamard product, and the division and square root in Equation (2.15) are applied element-wise.

Adam. The Adam algorithm, listed in Algorithm 2.3, is also a first-order adaptive variant of SGD [Kingma and Ba, 2014]. Adam uses adaptive estimates of the first and second moments of the gradients to adjust the learning rate for each parameter individually.

Batch Normalization. Computational differentiable graphs with multiple layers, such as neural networks, can be difficult to train due to what is commonly known as *the vanishing or exploding gradient problem* [Hochreiter, 1991, Hochreiter, 1998]. The problem arises when gradients flowing through the graph are too small or too large, to a point that can hinder learning. This also makes choosing hyperparameters, particularly the learning rate α, a tedious process, as the chosen hyperparameters must deal with the entire spectrum of gradients. Batch Normalization (BatchNorm) is an adaptive re-parametrization method that aims to alleviate the difficulty of training computational differentiable graphs with multiple layers [Ioffe and Szegedy, 2015]. The key idea in BatchNorm is to normalize the means and standard deviations, over a mini-batch of exemplars, of the output $\mathbf{H}^{(l)} \in \mathbb{R}^{M_b \times n_l}$ of units in layer l as follows, where M_b is the number of exemplars in the mini-batch, and n_l

Algorithm 2.3 Adam algorithm. ϵ is a small constant for numerical stability (e.g., $\epsilon = 10^{-8}$). The division and square root in Step 10 are applied element-wise.

```
Require: Training set $\mathbb{D}^{(train)}$ of exemplars $\left(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\right)$
Require: Initial parameters $\boldsymbol{\theta}$
Require: Learning rate $\alpha$
Require: Mini-batch size $M_b$
Require: Decay rates for moment estimates $\beta_1$ and $\beta_2$
 1: Initialize first and second moment estimates $\boldsymbol{\theta}^{(s)} \leftarrow \mathbf{0}$, $\boldsymbol{\theta}^{(v)} \leftarrow \mathbf{0}$
 2: Initialize iteration counter $i \leftarrow 0$
 3: while stopping criteria not met do
 4:     $i \leftarrow i + 1$
 5:     Sample mini-batch $\{\left(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\right) \mid 1 \leq m \leq M_b\} \sim \mathbb{D}^{(train)}$
 6:     $\boldsymbol{\theta}^{(s)} \leftarrow \beta_1 \boldsymbol{\theta}^{(s)} + (1 - \beta_1)\left((1/M_b)\sum_{m=1}^{M_b} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\mathbf{x}^{(m)}; \boldsymbol{\theta}\right), \mathbf{y}^{(m)}\right)\right)$
 7:     $\boldsymbol{\theta}^{(v)} \leftarrow \beta_2 \boldsymbol{\theta}^{(v)} + (1 - \beta_2)\left((1/M_b)\sum_{m=1}^{M_b} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\mathbf{x}^{(m)}; \boldsymbol{\theta}\right), \mathbf{y}^{(m)}\right)\right)^2$
 8:     $\hat{\boldsymbol{\theta}}^{(s)} \leftarrow \boldsymbol{\theta}^{(s)} / (1 - \beta_1^i)$          // correct bias of first moment
 9:     $\hat{\boldsymbol{\theta}}^{(v)} \leftarrow \boldsymbol{\theta}^{(v)} / (1 - \beta_2^i)$          // correct bias of second moment
10:     Apply updates $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\left(\hat{\boldsymbol{\theta}}^{(s)} / \left(\sqrt{\hat{\boldsymbol{\theta}}^{(v)}} + \epsilon\right)\right)$
11: end while
12: return Trained parameters $\boldsymbol{\theta}$
```

is the number of units in layer l:

$$\boldsymbol{\mu}_b = \frac{1}{M_b}\sum_{m=1}^{M_b} \mathbf{H}^{(l)}_{m:}, \tag{2.16}$$

$$\boldsymbol{\sigma}_b = \sqrt{\left(\frac{1}{M_b}\sum_{m=1}^{M_b}\left(\mathbf{H}^{(l)}_{m:} - \boldsymbol{\mu}_b\right)^2\right) + \epsilon}, \tag{2.17}$$

$$\text{BatchNorm}(\mathbf{H}^{(l)}_{m:}; \boldsymbol{\gamma}, \boldsymbol{\beta}) = \boldsymbol{\beta} + \boldsymbol{\gamma} \odot \left(\frac{\mathbf{H}^{(l)}_{m:} - \boldsymbol{\mu}_b}{\boldsymbol{\sigma}_b}\right), \tag{2.18}$$

where $\boldsymbol{\gamma} \in \mathbb{R}^{n_l}$ and $\boldsymbol{\beta} \in \mathbb{R}^{n_l}$ are adaptive parameters that determine the mean and standard deviation of the outputs respectively, and the square root in Equation (2.17) and the division in Equation (2.18) are applied element-wise. The normalized outputs are then used instead of the unnormalized outputs $\mathbf{H}^{(l)}$ in the forward propagation, and dealt with accordingly in the backward propagation when computing the gradients [Ioffe and Szegedy, 2015] (see Section 2.5 for more details on the forward and backward propagations). At test time, $\boldsymbol{\mu}_b$ and $\boldsymbol{\sigma}_b$ can be replaced by running averages that were computed during training to deal with single exemplar inferences. BatchNorm leads to more stable gradients and consequently faster training.

2.4 Deep Learning

Shallow machine learning algorithms, e.g., SVMs [Cortes and Vapnik, 1995], which are composed of a single processing layer, typically struggle to learn complex functions, especially directly from raw data. Shallow machine learning algorithms rely instead on features, engineered from raw data, that are representative of the task at hand. Devising representative features requires considerable engineering and domain expertise, and is inherently empirical. This leads to fragile features that are not robust enough as these features are fixed as opposed to learned. No machine learning algorithm can overcome a poor choice of features.

The foundations of deep learning date back to the introduction of Multi-Layer Perceptrons (MLPs) and the *backpropagation algorithm* [Rumelhart et al., 1986, Schmidhuber, 2015], with the advent of MLPs formulated as a series of affine transformations weighted by a set of adaptive parameters and non-linear operations. These techniques worked well for shallow neural networks that contain a few — typically one — hidden layers but did not work as well for Deep Neural Networks (DNNs) with many hidden layers. Subsequently, this problem was studied, and the reason was identified [Hochreiter, 1991, Hochreiter, 1998], the so-called vanishing or exploding gradient problem. It was found that as gradients flow through many layers, the magnitudes of these gradients either shrink indefinitely and vanish or grow exponentially and explode, hindering the training process. *Greedy layer-wise unsupervised pre-training* [Hinton et al., 2006] was introduced to overcome this problem, which utilized unsupervised learning to pre-train the model, layer-by-layer, by extracting prominent features from the previous layer using a Restricted Boltzmann Machine (RBM) [Hinton, 2012], followed by supervised learning using the backpropagation algorithm to fine-tune all layers in the model. Such unsupervised pre-training was later rendered unnecessary with subsequent advances, such as Rectified Linear Units (ReLUs) and BatchNorm.

Deep learning is an approach to machine learning. While shallow learning aims to learn a mapping from the input space $\mathcal{X}$ — typically engineered features — to the output space $\mathcal{Y}$, deep learning aims to learn multiple layers of hierarchical representations from raw data $\mathcal{X}$, together with the mapping from the representation space to the output space $\mathcal{Y}$, in a single composite function. Learning representations with a fairly general learning algorithm as opposed to using engineered features is much more advantageous as it requires less human intervention and domain-specific knowledge, performs well across various tasks, and ultimately leads to better performance.

Deep learning allows learning representations and concepts in a hierarchical manner, i.e., representations that are expressed in terms of other representations. This can be achieved by decomposing computational models or graphs into multiple layers of processing, with the aim of learning representations of the data with multiple levels of abstraction [LeCun et al., 2015]. In doing so, the model can adaptively learn low-level features from raw data and higher-level features from the low-level features in a hierarchical way [Hinton et al., 2006, Zeiler and Fergus, 2014].

For example, let $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ be a deep learning model parametrized by parameters $\boldsymbol{\theta}$ that comprises four layers, $f^{(1)}$, $f^{(2)}$, $f^{(3)}$, and $f^{(4)}$, where $\mathbf{x}$ is the input and $\mathbf{y}$ is the output. This expression can be rewritten more verbosely as in Equation (2.19).

$$\mathbf{y} = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}; \boldsymbol{\theta}^{\mathbf{(1)}}); \boldsymbol{\theta}^{\mathbf{(2)}}); \boldsymbol{\theta}^{\mathbf{(3)}}); \boldsymbol{\theta}^{\mathbf{(4)}}), \tag{2.19}$$

where $\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \boldsymbol{\theta}^{(3)}, \boldsymbol{\theta}^{(4)}\} \in \boldsymbol{\theta}$ are the parameters of the first, second, third, and output layers, $f^{(1)}, f^{(2)}, f^{(3)}, f^{(4)}$, respectively. Herein, it would be expected that $\boldsymbol{\theta}^{(1)}$ would learn simple low-level features, $\boldsymbol{\theta}^{(2)}$ would utilize $\boldsymbol{\theta}^{(1)}$ to learn slightly more abstract mid-level features, $\boldsymbol{\theta}^{(3)}$ would utilize $\boldsymbol{\theta}^{(2)}$ to learn abstract high-level features, while $\boldsymbol{\theta}^{(4)}$ would utilize $\boldsymbol{\theta}^{(3)}$ to learning the mapping to $\mathbf{y}$. Concretely, in an image recognition task, $\boldsymbol{\theta}^{(1)}$ can resemble Gabor filters, $\boldsymbol{\theta}^{(2)}$ can resemble contours composed of multiple Gabor filters, $\boldsymbol{\theta}^{(3)}$ can resemble motifs or objects composed of multiple contours, and $\boldsymbol{\theta}^{(4)}$ can represent the classes in the recognition task at hand. With the composition of enough functions $f^{(l)}$, one can more easily learn complex functions.

Typically, the entire deep learning model is trained *end-to-end*, i.e., all layers are trained simultaneously, using methods outlined in Sections 2.1, 2.2 and 2.3; a loss function $\mathcal{L}$ is minimized using a training set $\mathbb{D}_k^{(train)}$ typically via an iterative gradient-based optimization algorithm that utilizes gradients computed throughout the model with the objective of generalization to unseen data.

At present, deep learning is the state-of-the-art approach to many problems in machine learning, and is prominent in numerous fields, such as computer vision [Krizhevsky et al., 2012, He et al., 2015, Ren et al., 2015, Johnson et al., 2017], speech recognition [Hinton et al., 2012, Graves et al., 2013, Abdel-Hamid et al., 2014, Chorowski et al., 2015, Fayek et al., 2017], and natural language processing [Sutskever et al., 2014, Bordes et al., 2015, Hermann et al., 2015, See et al., 2017].

Despite its success, deep learning suffers a number of drawbacks, some of which are major research areas in the field. For example, deep learning typically

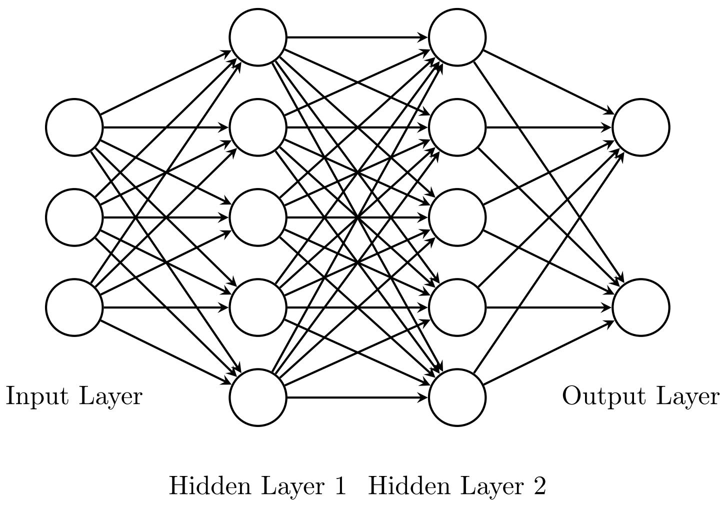

Figure 2.2: Feed-forward fully connected neural network with two hidden layers.

requires more data and computational resources compared with competing methods, c.f., [Redmon et al., 2016]. Deep learning lacks interpretability and deep models are often perceived as black-boxes, which can make the reliance on such models unpredictable, c.f., [Goodfellow et al., 2015]. Deep learning also presents its own engineering challenges due to the difficulty associated with tuning hyperparameters, c.f., [Snoek et al., 2012].

2.5 Neural Networks

A neural network is a collection of nodes or units that are inter-connected via adaptive weights to form a directed weighted graph that can learn distributed representations and ultimately the task at hand. Neural networks were initially conceived as a simplified model of biological neural networks of the human brain; though it is now clear that neural networks and biological neural networks bear little resemblance to each other. Nevertheless, neural networks remain one of the most powerful learning systems.

A neural network with at least a single non-linear hidden layer composed of an arbitrary number of units and a linear output layer can approximate any Borel measurable function with some arbitrary accuracy, according to the *universal approximation theorem* [Hornik et al., 1989, Cybenko, 1989], assuming the neural network has enough units in its hidden layer [Goodfellow et al., 2016].

The most widely used type of neural networks is the feed-forward fully connected neural network, as illustrated in Figure 2.2. Feed-forward fully connected neural networks are unidirectional acyclic graphs that comprise one or more layers of affine transformations and non-linear operations. Each layer is composed of one or more nodes or units that are connected via adaptive weights to units in the previous layer and the subsequent layer.

Affine Transformation. The output of each unit in the previous layer is the input to each unit in the following layer via an adaptive weight as in Equation (2.20):

$$\mathbf{h}_i^{(l)} = \sum_{j=1}^{n_{(l-1)}} \mathbf{W}_{ij}^{(l)} \hat{\mathbf{y}}_j^{(l-1)} + \mathbf{b}_i^{(l)}, \tag{2.20}$$

where $\mathbf{h}_i^{(l)}$ is the pre-activation of unit i in layer l, $\mathbf{W}_{ij}^{(l)}$ is the adaptive weight from unit j in layer $(l-1)$ to unit i in layer l, $\hat{\mathbf{y}}_j^{(l-1)}$ is the output of unit j in layer $(l-1)$, $\mathbf{b}_i^{(l)}$ is the adaptive bias term of unit i in layer l, $l \in \{1, \ldots, L\}$, and L is the number of layers in the network.

Equation (2.20) can be expressed in a more concise form as per Equation (2.21):

$$\mathbf{h}^{(l)} = \mathbf{W}^{(l)} \hat{\mathbf{y}}^{(l-1)} + \mathbf{b}^{(l)}, \tag{2.21}$$

where $\mathbf{h}^{(l)} \in \mathbb{R}^{n_l}$ is the pre-activation vector of layer l, n_l is the number of units in layer l, $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{(l-1)}}$ is the adaptive weight matrix from layer $(l-1)$ to layer l, $\hat{\mathbf{y}}^{(l-1)} \in \mathbb{R}^{n_{(l-1)}}$ is the output vector of layer $(l-1)$, $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ is the adaptive bias vector of layer l, $\hat{\mathbf{y}}^{(0)} = \mathbf{x}$ is the input to the network, and $\hat{\mathbf{y}}^{(L)}$ is the output of the network. Note that for notational simplicity, the output of the network $\hat{\mathbf{y}}^{(L)}$ is abbreviated to $\hat{\mathbf{y}}$.

Non-linear Operation. Non-linear operations enable neural networks to learn non-linear mappings from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. The pre-activation vector $\mathbf{h}^{(l)}$ typically undergoes a non-linear operation ϕ as in Equation (2.22) to obtain the output of layer[‖] l:

$$\hat{\mathbf{y}}^{(l)} = \phi(\mathbf{h}^{(l)}), \tag{2.22}$$

where $\hat{\mathbf{y}}^{(l)}$ is the output of layer l, and ϕ is a non-linear operation applied element-wise.

There are numerous non-linear operations that can be used, such as the sigmoid functions, e.g., the logistic function in Equation (2.23) and the tanh function in Equation (2.24), and the ReLU [Glorot et al., 2011a] in Equation (2.25). The

[‖]The input to the function ϕ is referred to as the pre-activation of the layer. The non-linear operation ϕ is also known as the activation function. The output of the function ϕ is referred to as the activation of the layer.

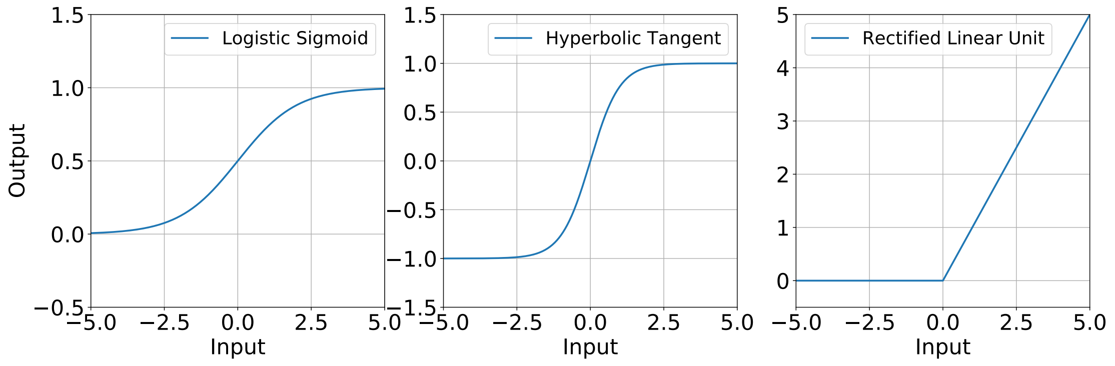

Figure 2.3: Non-linear activation functions. **Left.** Logistic sigmoid function. **Centre.** Hyperbolic tangent function. **Right.** Rectified linear unit.

responses of these functions are illustrated in Figure 2.3. Sigmoid functions squash the inputs to be in the $[0, 1]$ range in the case of the logistic function and $[-1, 1]$ in the case of the tanh function, which can be advantageous in conditioning the output of hidden layers. The ReLU circumvents possible saturation of the output of hidden layers in sigmoid functions due to the unbounded nature of the rectification function.

$$sigm(z) = \frac{1}{1 + \mathrm{e}^{-\mathrm{z}}} \tag{2.23}$$

$$\tanh(z) = \frac{\mathrm{e}^{\mathrm{z}} - \mathrm{e}^{-\mathrm{z}}}{\mathrm{e}^{\mathrm{z}} + \mathrm{e}^{-\mathrm{z}}} \tag{2.24}$$

$$\mathrm{relu(z)} = \max(0, \mathrm{z}) = \begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases} \tag{2.25}$$

Other popular non-linear functions include leaky-ReLUs [Maas et al., 2013], maxout units [Goodfellow et al., 2013], and Scaled Exponential Linear Units (SELUs) [Klambauer et al., 2017].

The final layer in the neural network, which is the output layer, typically uses a function that depends on the type of task at hand. For example, in the case of regression, it can be the identify function**. The output layer can then be viewed as a linear regression model that utilizes the representations learned in the preceding hidden layer as its input. Similarly, in the case of classification, the softmax function in Equation (2.26) can be used, which can be viewed as a logistic regression model.

$$softmax(\mathbf{z}_k) = \frac{\mathrm{e}^{\mathbf{z}_\mathrm{k}}}{\sum_{k=1}^{K} \mathrm{e}^{\mathbf{z}_\mathrm{k}}} \quad \forall k \in K, \tag{2.26}$$

**A layer with an identity function as its non-linear operation is just a linear layer.

where K is the number of output classes. The output of a softmax layer can be interpreted as the probabilities of the output classes.

Training and Inference. Thus far, the elements that constitute a feed-forward neural network have been presented. Training such models for the required task involves designing the architecture of the network, including the number of layers and size of each layer, followed by initializing the parameters of the model, and optimizing the parameters to minimize the loss function.

There are a number of recommended initialization strategies for feed-forward neural networks depending on the type of non-linear operation used. The standard practice is to initialize the parameters of the model randomly, sampling from a Gaussian distribution, to break symmetry between units in the network. The mean and standard deviation of the Gaussian distribution can be tuned to aid the optimization process [Glorot et al., 2011a]. For example, it is recommended in [He et al., 2015] to initialize weights in layers with ReLUs using a Gaussian distribution with zero mean and $\sqrt{2/n_i}$ standard deviation, where n_i is the number of inputs to the layer, whereas biases are initialized to zero. This approximately adjusts the magnitudes of the fan-in and fan-out of each unit regardless of the number of units in each layer (see [He et al., 2015] for more details on the derivation).

Training feed-forward neural networks via an iterative gradient-based training algorithm, as described in Section 2.3, can be achieved using the backpropagation algorithm, which requires propagating the inputs $\mathbf{x}$ forward through the hidden layer(s) and output layer to obtain the predicted output $\hat{\mathbf{y}}$, computing the loss function and its derivative between the true output $\mathbf{y}$ and the predicted output $\hat{\mathbf{y}}$, and propagating the errors backwards through the layers of the network. Note that neural networks can be trained with algorithms other than the backpropagation algorithm, e.g., the target propagation algorithm [Lee et al., 2015] (see [LeCun, 1986, Montana and Davis, 1989, Wilamowski and Yu, 2010, Lee et al., 2015] for more details). Inference requires the forward propagation only to compute the predicted output of the model.

Forward Propagation. Algorithm 2.4 lists the complete algorithm to propagate the inputs $\mathbf{x}$ forward through the hidden layer(s) and output layer to obtain the output $\hat{\mathbf{y}}$.

Loss Function. In training, following the forward propagation outlined in Algorithm 2.4, the loss function is computed. The choice of loss function is dependent on the task at hand. Two popular loss functions are the Mean Squared Error (MSE) as in Equation (2.27) and the negative log-likelihood following Equation (2.4) to compute the cross-entropy between the true output and predicted output as in

Algorithm 2.4 Forward propagation through a fully connected (deep) neural network. ϕ denotes a non-linear operation applied element-wise. Note that for notational convenience, the output of the output layer in the network $\hat{\mathbf{y}}^{(L)}$ is abbreviated to $\hat{\mathbf{y}}$. The algorithm assumes a single exemplar m but can be extended to the case with a mini-batch of exemplars or the entire dataset.

```
Require: Network depth $L$
Require: Network parameters $\boldsymbol{\theta}$, s.t., $\mathbf{W}^{(l)}, \mathbf{b}^{(l)} \in \boldsymbol{\theta}$, $l \in \{1, \dots, L\}$
Require: Training exemplar $\mathbf{x}^{(m)}$
1: $\hat{\mathbf{y}}^{(0)} = \mathbf{x}^{(m)}$
2: for $l = 1, \dots, L$ do
3:     $\mathbf{h}^{(l)} = \mathbf{W}^{(l)}\hat{\mathbf{y}}^{(l-1)} + \mathbf{b}^{(l)}$        // see Equation (2.21)
4:     $\hat{\mathbf{y}}^{(l)} = \phi(\mathbf{h}^{(l)})$        // see Equation (2.22)
5: end for
6: $\hat{\mathbf{y}}^{(m)} = \hat{\mathbf{y}}^{(L)}$
7: return Predicted output $\hat{\mathbf{y}}^{(m)}$
```

Equation (2.28).

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{m} \sum_{m} \sum_{k=1}^{K} \left(\hat{\mathbf{y}}_k^{(m)} - \mathbf{y}_k^{(m)}\right)^2, \tag{2.27}$$

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{m} \sum_{m} \sum_{k=1}^{K} \mathbf{y}_k^{(m)} \log(\hat{\mathbf{y}}_k^{(m)}), \tag{2.28}$$

where $\mathbf{y} \in \mathbb{Z}_2^K$ or $\mathbb{R}^K$ is a one-hot encoded vector denoting the class label or a vector of targets, and m denotes either a single exemplar, the size of the mini-batch M_b, or the entire training set M. Additional terms can be added to ℓ, such as the regularization term(s), as discussed in Section 2.2.

Backward Propagation. Algorithm 2.5 lists the backpropagation algorithm [Rumelhart et al., 1986] to propagate the errors backwards through the layers of the network. The backpropagation algorithm starts with differentiating the loss function $\mathcal{L}$ with respect to the output(s) of the model and employs the chain rule to recursively compute the gradients for each layer in the network. The gradients can then be used to update the parameters of the network using a gradient-based optimization algorithm as outlined in Section 2.3.

Algorithm 2.5 Backward computation through a fully connected (deep) neural network. ϕ' denotes the derivative of the non-linear operation applied element-wise. $\odot$ denotes the Hadamard product. $\top$ denotes the transpose operation. λ is the regularization weight and the entire term can be ignored if the loss function does not constitute a regularization penalty Ω. Note that for notational convenience, the output of the output layer in the network $\hat{\mathbf{y}}^{(L)}$ is abbreviated to $\hat{\mathbf{y}}$. The algorithm assumes a single exemplar m but can be extended to the case with a mini-batch of exemplars or the entire dataset.

```
Require: Network depth $L$
Require: Network parameters $\boldsymbol{\theta}$, s.t., $\mathbf{W}^{(l)}, \mathbf{b}^{(l)} \in \boldsymbol{\theta}$, $l \in \{1, \ldots, L\}$
Require: Training exemplar $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})$
Require: Forward propagation using $\mathbf{x}^{(m)}$ to obtain $\hat{\mathbf{y}}^{(m)}$    // see Algorithm 2.4
1: $\hat{\mathbf{y}}^{(L)} = \hat{\mathbf{y}}^{(m)}$
2: $\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}^{(L)}} \mathcal{L}(\hat{\mathbf{y}}^{(L)}, \mathbf{y}^{(m)})$
3: for $l = L, \ldots, 1$ do
4:     $\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(l)}} \mathcal{L} = \mathbf{g} \odot \phi'(\mathbf{h}^{(l)})$
5:     $\nabla_{\mathbf{W}^{(l)}} \mathcal{L} = \mathbf{g} \hat{\mathbf{y}}^{(l-1)^\top} + \lambda \nabla_{\mathbf{W}^{(l)}} \Omega(\boldsymbol{\theta})$
6:     $\nabla_{\mathbf{b}^{(l)}} \mathcal{L} = \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(l)}} \Omega(\boldsymbol{\theta})$
7:     $\mathbf{g} \leftarrow \nabla_{\mathbf{y}^{(l-1)}} \mathcal{L} = \mathbf{W}^{(l)^\top} \mathbf{g}$
8: end for
9: return Gradients $\nabla_{\boldsymbol{\theta}} \mathcal{L}$, s.t., $\nabla_{\mathbf{W}^{(l)}} \mathcal{L}, \nabla_{\mathbf{b}^{(l)}} \mathcal{L} \in \nabla_{\boldsymbol{\theta}} \mathcal{L}$, $l \in \{1, \ldots, L\}$
```

2.6 Convolutional Neural Networks

A popular variant of the feed-forward fully connected neural network architecture is the Convolutional Neural Network (ConvNet) [LeCun et al., 1990,LeCun and Bengio, 1995], as illustrated in Figure 2.4. A ConvNet is a feed-forward neural network that contains at least one convolutional layer (defined below). ConvNets leverage three ideas: sparse interactions, parameter sharing, and equivariant representations. ConvNets are particularly suited to processing data that has a grid-like topology, such as time-series data, e.g., speech, two-dimensional and three-dimensional images, and three-dimensional and four-dimensional videos.

Convolutional Layer. A convolutional layer is similar to the fully connected layer that replaces the dot product between the output of the previous layer and

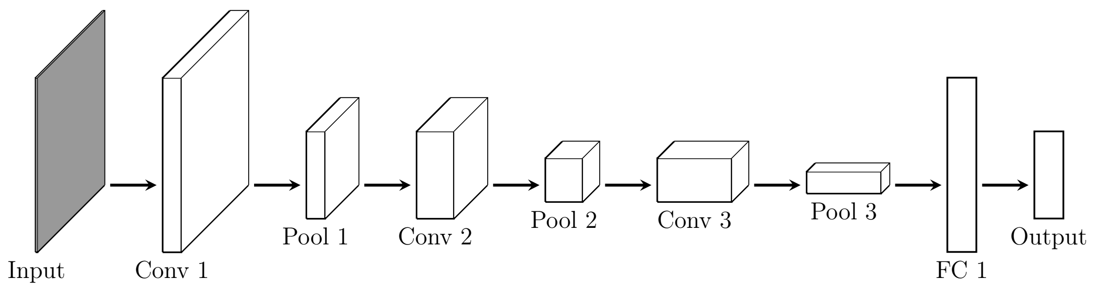

Figure 2.4: Convolutional neural network with three Convolutional (Conv) and Pooling (Pool) layers followed by a Fully Connected (FC) layer.

the weights (see Equation (2.21)) with the convolution operation[††] as follows:

$$\boldsymbol{H}^{(l)} = \hat{\boldsymbol{Y}}^{(l-1)} * \boldsymbol{W}^{(l)} + \mathbf{b}^{(l)}, \tag{2.29}$$

where in this case, $\boldsymbol{H}^{(l)} \in \mathbb{R}^{n_{l_c} \times n_{l_x} \times n_{l_y}}$ is the pre-activation tensor of layer l, n_{l_c} is the number of channels in layer l, n_{l_x} and n_{l_y} are the number of rows and columns respectively of each channel in layer l, $\hat{\boldsymbol{Y}}^{(l-1)} \in \mathbb{R}^{n_{(l-1)_c} \times n_{(l-1)_x} \times n_{(l-1)_y}}$ is the activation tensor of layer $(l-1)$, $n_{(l-1)_c}$ is the number of channels in layer $(l-1)$, $n_{(l-1)_x}$ and $n_{(l-1)_y}$ are the number of rows and columns respectively of each channel in layer $(l-1)$, $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_{l_c} \times n_{l_i} \times n_{l_j}}$ is tensor of adaptive weights of layer l, n_{l_i} and n_{l_j} are the number of rows and columns respectively of each filter in layer l, and $\mathbf{b}^{(l)} \in \mathbb{R}^{n_{l_c}}$ is a vector of adaptive biases of layer l.

Using the convolution operation provides numerous advantages: (1) a convolutional layer typically requires a smaller number of parameters compared with a fully connected layer; (2) parameters in a convolutional layer are shared across the input; and, (3) a convolutional layer is equivariant to translations in the input.

Following the convolution operation in Equation (2.29), $\boldsymbol{H}^{(l)}$ typically undergoes a non-linear operation as in Equation (2.22) to obtain the activation map $\hat{\boldsymbol{Y}}^{(l)}$.

Pooling Layer. A pooling function is commonly used following the non-linear operation to spatially downsample the activation map $\hat{\boldsymbol{Y}}^{(l)}$. The pooling layer does not typically contain any adaptive parameters and is applied to each activation map independently. A frequently used pooling layer is *max pooling*, which replaces each rectangular neighbourhood in the activation map with the maximum value in that neighbourhood, where the size of the neighbourhood is a hyperparameter. Another frequently used pooling layer is *mean pooling*, which replaces each rectangular

[††]The convolution operation in convolutional layers is usually implemented as the cross-correlation operation.

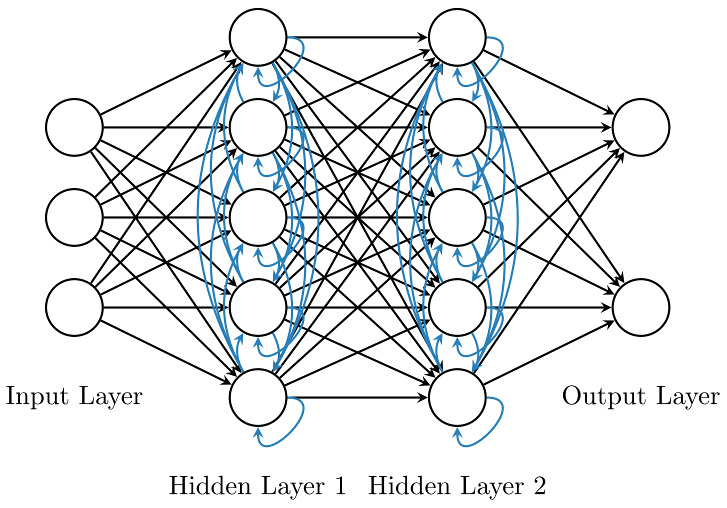

Figure 2.5: Recurrent neural network with two recurrent hidden layers characterized by self-connections (blue).

neighbourhood in the activation map with the mean value of all elements in that neighbourhood. Downsampling the activation map can also be achieved by adjusting the stride[‡‡] in the convolution operation rather than an explicit pooling layer [Simonyan and Zisserman, 2015, He et al., 2016]. Pooling can aid in achieving invariance to small translations in the input. Pooling also improves the computational efficiency of ConvNets as it decreases the size of the activation maps.

Training and Inference. A ConvNet follows the fully connected neural network in all aspects of regularization, optimization, and other practices.

2.7 Recurrent Neural Networks

A recurrent neural network is a cyclic graph that extends the notion of a typical feed-forward neural network architecture to model sequences. A recurrent neural network is a neural network that has at least one recurrent layer. A recurrent layer is characterized by having self-connections between units in the same layer [Graves, 2008], as illustrated in Figure 2.5. Recurrent neural networks are particularly suitable for tasks that involve sequential inputs such as speech and text.

[‡‡]The stride is the amount by which the filter shifts during a convolution operation. In the standard definition of a convolution operation, the stride is assumed to be 1.

Recurrent Layer. Consider a sequence $\mathbf{y}^{(l-1)}_{(1:t:T)}$ of length T such that $\mathbf{y}^{(l-1)}_{(t)}$ is the value of the sequence at step t. A recurrent layer can ingest this sequence one step at a time as per Equation (2.30).

$$\mathbf{h}^{(l)}_{(t)} = \mathbf{W}^{(l)}_{(y)}\mathbf{y}^{(l-1)}_{(t)} + \mathbf{W}^{(l)}_{(h)}\mathbf{y}^{(l)}_{(t-1)} + \mathbf{b}^{(l)}, \tag{2.30}$$

where t denotes the step, $\mathbf{h}^{(l)}_{(t)} \in \mathbb{R}^{n_l}$ is a vector of pre-activations of layer l at step t, n_l is the number of units in layer l, $\mathbf{y}^{(l-1)}_{(t)} \in \mathbb{R}^{n_{(l-1)}}$ is the output of the previous layer $(l-1)$ at step t and input to layer l at step t, $n_{(l-1)}$ is the number of units in the previous layer $(l-1)$, $\mathbf{W}^{(l)}_{(y)} \in \mathbb{R}^{n_l \times n_{(l-1)}}$ is a matrix of adaptive weights of layer l, $\mathbf{y}^{(l)}_{(t-1)} \in \mathbb{R}^{n_l}$ is the output of layer l at the previous step $(t-1)$, $\mathbf{W}^{(l)}_{(h)} \in \mathbb{R}^{n_l \times n_l}$ is a matrix of adaptive weights of layer l, and $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ is a vector of adaptive biases of layer l.

Equation (2.30) replaces Equation (2.21), and $\mathbf{h}^{(l)}_{(t)}$ typically undergoes a non-linear operation as in Equation (2.22) to obtain $\mathbf{y}^{(l)}_{(t)}$. The exploding gradient problem in recurrent layers is particularly problematic, as gradients can grow indefinitely when back-propagated through a long sequence of many steps. This makes sigmoid functions, e.g., the logistic function (see Equation (2.23)) or the tanh function (see Equation (2.24)), a more sensible choice for the non-linear operation due to their bounded nature (see Figure 2.3); unbounded non-linear operations, e.g., ReLUs, can amplify the exploding gradient problem and should be avoided.

Long Short-Term Memory. A popular variant of the recurrent layer is the Long Short-Term Memory (LSTM) layer [Hochreiter and Schmidhuber, 1997]. An LSTM unit uses an explicit memory cell to learn long-term dependencies in sequences, where relevant events are separated by a large number of steps. An LSTM layer can be modelled using Equations (2.31) to (2.35):

$$\mathbf{i}^{(l)}_{(t)} = sigm\left(\mathbf{W}^{(l)}_{(yi)}\mathbf{y}^{(l-1)}_{(t)} + \mathbf{W}^{(l)}_{(hi)}\mathbf{y}^{(l)}_{(t-1)} + \mathbf{W}^{(l)}_{(ci)}\mathbf{c}^{(l)}_{(t-1)} + \mathbf{b}^{(l)}_{(i)}\right), \tag{2.31}$$

$$\mathbf{f}^{(l)}_{(t)} = sigm\left(\mathbf{W}^{(l)}_{(yf)}\mathbf{y}^{(l-1)}_{(t)} + \mathbf{W}^{(l)}_{(hf)}\mathbf{y}^{(l)}_{(t-1)} + \mathbf{W}^{(l)}_{(cf)}\mathbf{c}^{(l)}_{(t-1)} + \mathbf{b}^{(l)}_{(f)}\right), \tag{2.32}$$

$$\mathbf{c}^{(l)}_{(t)} = \mathbf{f}^{(l)}_{(t)} \odot \mathbf{c}^{(l)}_{(t-1)} + \mathbf{i}^{(l)}_{(t)} \odot \tanh\left(\mathbf{W}^{(l)}_{(yc)}\mathbf{y}^{(l-1)}_{(t)} + \mathbf{W}^{(l)}_{(hc)}\mathbf{y}^{(l)}_{(t-1)} + \mathbf{b}^{(l)}_{(c)}\right), \tag{2.33}$$

$$\mathbf{o}^{(l)}_{(t)} = sigm\left(\mathbf{W}^{(l)}_{(yo)}\mathbf{y}^{(l-1)}_{(t)} + \mathbf{W}^{(l)}_{(ho)}\mathbf{y}^{(l)}_{(t-1)} + \mathbf{W}^{(l)}_{(co)}\mathbf{c}^{(l)}_{t} + \mathbf{b}^{(l)}_{(o)}\right), \tag{2.34}$$

$$\mathbf{y}^{(l)}_{(t)} = \mathbf{o}^{(l)}_{(t)} \odot \tanh(\mathbf{c}^{(l)}_{(t)}), \tag{2.35}$$

where $sigm$ is the logistic sigmoid function as in Equation (2.23), and $\mathbf{i}^{(l)}_{(t)}$, $\mathbf{f}^{(l)}_{(t)}$, $\mathbf{c}^{(l)}_{(t)}$ and $\mathbf{o}^{(l)}_{(t)}$ are respectively the input gate, forget gate, cell activation and output gate vectors of layer l at time t, all of which are the same size as the activation vector $\mathbf{y}_{(t)}^{(l)} \in \mathbb{R}^{n_l}$. The adaptive weight matrices from the cell to gate vectors, e.g., $\mathbf{W}_{(ci)}^{(l)}$, are diagonal, such that each element in each gate vector only receives input from the same element of the cell vector [Graves et al., 2013].

Training and Inference. Training recurrent architectures requires modification to the backpropagation algorithm to compute the gradients with respect to the parameters over time. The BackPropagation Through Time (BPTT) algorithm [Werbos, 1988] unrolls the network in time and applies the backpropagation algorithm to the unrolled graph. All other aspects of regularization, optimization, etc., remain identical to feed-forward neural networks.

2.8 Learning Multiple Tasks

Thus far, the machine learning paradigm presented was formulated for learning a single task. For an arbitrary task $t_k \in \mathcal{T}_\infty = \{t_i \mid i \in \mathbb{Z}^+\}$, a model f_k is initialized and trained using a dataset $\mathbb{D}_k^{(train)}$ or environment $\mathbb{E}_k$ to minimize a loss function ℓ_k or maximize a reward r_k with the aim of achieving a certain objective o_k. The model f_k is initialized randomly, and hence learns from a tabula rasa, i.e., *independent learning*, which is the most common machine learning paradigm [Mikolov et al., 2018]. Contrary to human learning, the model f_k does not take into account knowledge learned in prior models $\mathcal{F}_{k-1} = \{f_i \mid 1 \leq i \leq k-1\}$ for tasks $\mathcal{T}_{k-1} = \{t_i \mid 1 \leq i \leq k-1\}$ using datasets $\mathcal{D}_{k-1} = \{\mathbb{D}_i^{(train)} \mid 1 \leq i \leq k-1\}$ or environments $\mathcal{E}_{k-1} = \{\mathbb{E}_i \mid 1 \leq i \leq k-1\}$, which may lead to a slower learning process that requires more data and possibly suboptimal performance [Lake et al., 2017].

Sharing knowledge between tasks has a long history in the field of machine learning. In the context of deep learning and neural networks, paradigms such as *transfer learning*, *multi-task learning*, and *lifelong learning* are alternatives to independent learning that can share or repurpose experiences and knowledge over tasks.

Transfer Learning. Transfer learning incorporates knowledge from a source task into a target task by initializing the parameters of the model for the target task using the parameters learned in the source task, followed by fine-tuning some or all of those parameters [Taylor and Stone, 2009, Pan and Yang, 2010, Yosinski et al., 2014], as illustrated in Figure 2.6. For example, the parameters of a model trained for an English Automatic Speech Recognition (ASR) task can be used to initialize another model for a French ASR task. Transfer learning can boost the performance of a model relative to independent learning [Razavian et al., 2014], when both tasks are related, particularly in cases where the target task lacks sufficient data.

Transfer learning is a popular paradigm that has demonstrated success across a variety of settings, domains, and applications, as briefly surveyed in Section 4.1 and [Pan and Yang, 2010], despite the following limitations. First, if both tasks are unrelated to each other, transfer learning may lead to *negative forward transfer*, where the model performs worse when initialized from a source task compared with the same model randomly initialized. Second, prior knowledge is only incorporated at the initialization phase of learning. Third, it is non-trivial to extend the paradigm to utilize multiple source tasks [Misra et al., 2016]. Fourth, it is susceptible to catastrophic forgetting [McCloskey and Cohen, 1989, Ratcliff, 1990], where parameters learned for the source task are repurposed for the new task and the model forgets the older task.

Multi-task Learning. Multi-task learning aims to learn a single model jointly for multiple related tasks by leveraging relevant information contained in the parallel training signals of those tasks [Caruana, 1997], as illustrated in Figure 2.7. The goal of multi-task learning is typically to maximize the performance across all tasks, unlike transfer learning, where the goal is to maximize the performance of the latest task.

Multi-task learning has been successfully applied to numerous tasks in computer vision [Misra et al., 2016], speech recognition [Chen and Mak, 2015], and natural language processing [Collobert et al., 2011], despite the following. Multi-task learning does not provide a straight-forward mechanism for adding new tasks once the model has been trained, in that it may be required to jointly relearn all tasks whenever a new task is added. Moreover, multi-task learning is intrinsically unable to handle interference between unrelated tasks.

Lifelong Learning. Lifelong learning [Thrun and Mitchell, 1995, Ruvolo and Eaton, 2013b, Chen and Liu, 2016] is a form of continual learning, and the terms are sometimes used interchangeably. Lifelong learning can be defined as follows: "The system has performed N tasks. When faced with the $(N + 1)$th task, it uses the knowledge gained from the N tasks to help the $(N + 1)$th task." [Thrun, 1996]. This definition was extended in [Chen and Liu, 2016] as follows: "Lifelong machine learning is a continuous learning process. At any time point, the learner has performed a sequence of N learning tasks, $\mathcal{T}_1$, $\mathcal{T}_2$, . . . , $\mathcal{T}_N$. These tasks, which are also called the previous tasks, have their corresponding datasets $\mathcal{D}_1$, $\mathcal{D}_2$, . . . , $\mathcal{D}_N$. The tasks can be of different types and from different domains. When faced with the $(N + 1)$th task $\mathcal{T}_{N+1}$ (which is called the new or current task) with its data $\mathcal{D}_{N+1}$, the learner can leverage the past knowledge in the knowledge base to

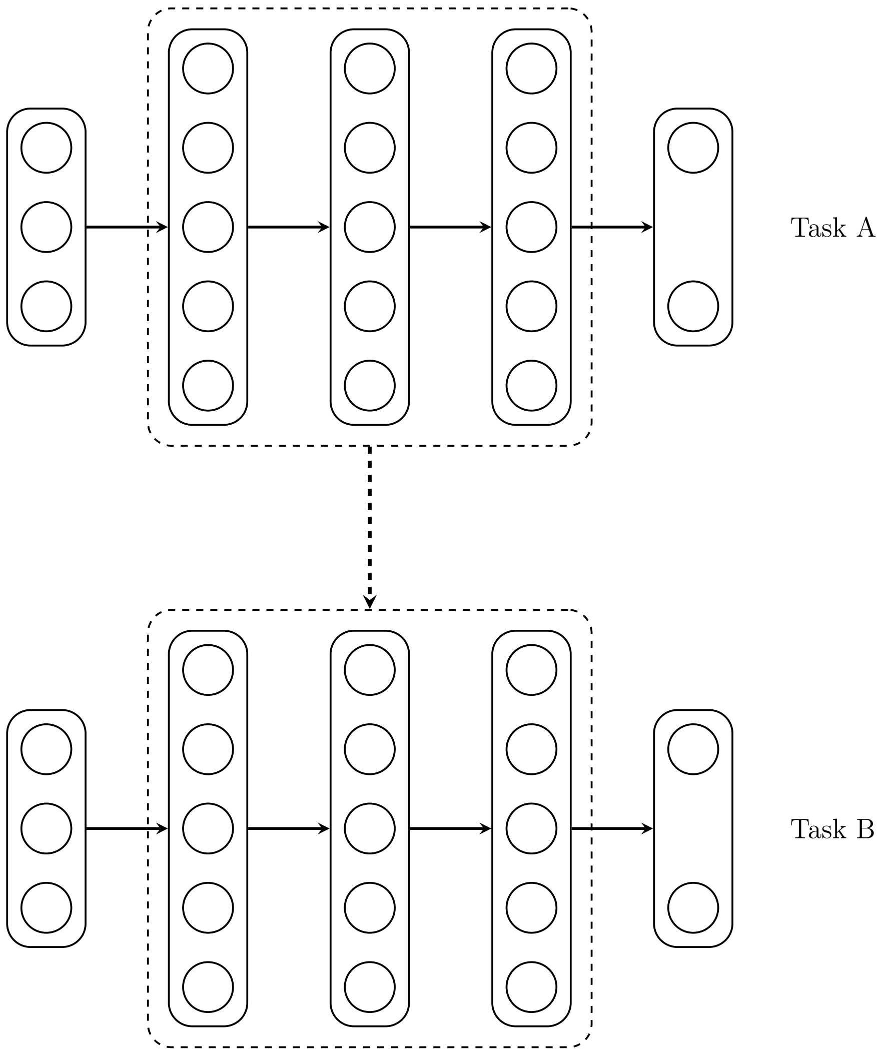

Figure 2.6: Transfer learning via a neural network with three hidden layers. **Top.** The parameters of the model were randomly initialized, and trained for Task A. **Bottom.** The parameters of the model were initialized using the parameters trained for Task A, and fine-tuned for Task B.

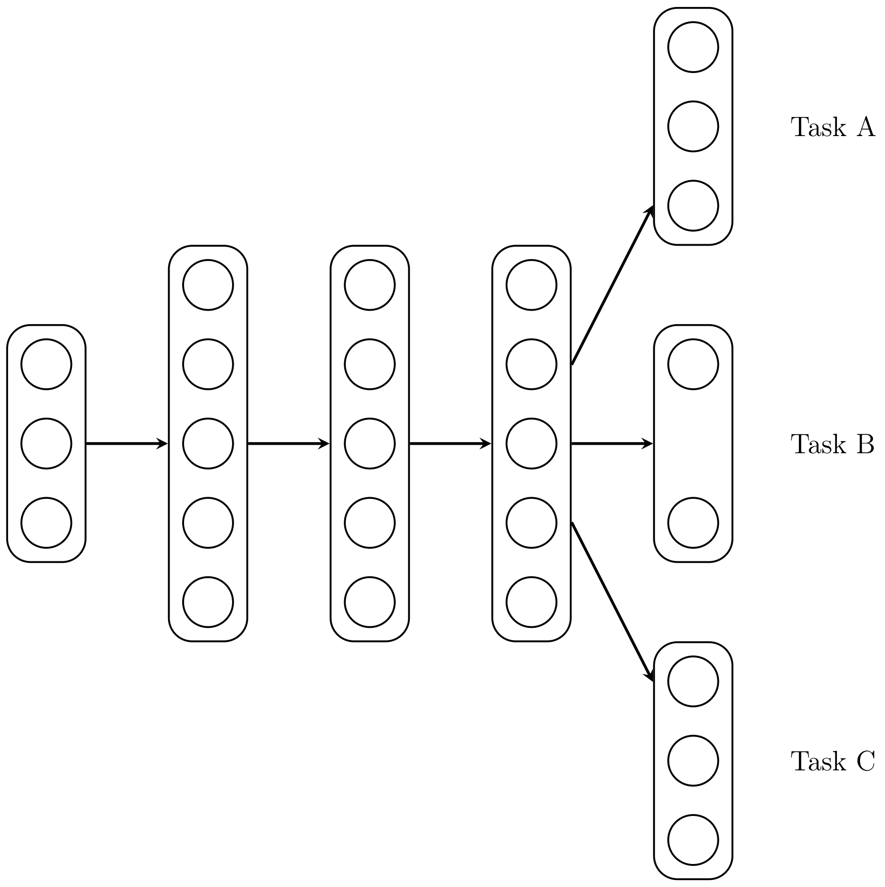

Figure 2.7: Multi-task learning with three tasks via a neural network that has three hidden layers. All layers are shared between the three tasks except the output layer which is unique to each task.

help learn $\mathcal{T}_{N+1}$." Continual learning is surveyed in Section 2.9.

2.9 Continual Learning

Continual learning is a machine learning paradigm, whereby tasks are learned in sequence with the ability to use prior knowledge from previously learned tasks to facilitate the learning and execution of new ones, e.g., lifelong learning [Thrun and Mitchell, 1995] and never-ending learning [Mitchell et al., 2018]. Recall the definition of continual learning, as follows: "The system has performed K tasks. When faced with the $(K+1)$th task, it uses the knowledge gained from the K tasks to help the $(K+1)$th task." [Thrun, 1996], where the definition was adapted to follow the notation adopted in this thesis. This poses numerous challenges, two of which are catastrophic forgetting and negative forward transfer. Catastrophic forgetting [McCloskey and Cohen, 1989, Ratcliff, 1990] refers to the case when the performance of the system for the K tasks degrades when it learns the $(K+1)$th task, as the model forgets the K tasks to learn the $(K+1)$th task. Negative forward transfer refers to the case when the K tasks have a negative effect on the performance of the system for the $(K+1)$th task, which can occur when the $(K+1)$th task is not related to any of the K tasks leading to interfere and worse performance. Mitigating negative forward transfer and catastrophic forgetting in an efficient manner are two of the main challenges in continual learning.

Continual learning in deep learning and neural networks has been studied in numerous prior works and several paradigms have been proposed as follows.

- The parameter regularization paradigm aims to find a measure of importance of parameters for prior tasks that can be used to adaptively adjust or penalize their perturbation during training on a new task, and thus mitigate catastrophic forgetting, e.g., elastic weight consolidation [Kirkpatrick et al., 2017], synaptic intelligence [Zenke et al., 2017], and gradient episodic memory [Lopez-Paz and Ranzato, 2017].

- The functional regularization paradigm combats catastrophic forgetting by introducing a regularization term to the loss function that penalizes deviation in the output of the model trained on some prior tasks when being trained on a new task, e.g., learning without forgetting [Li and Hoiem, 2016] and adaptation by distillation [Hou et al., 2018].

- The architectural paradigm sidesteps catastrophic forgetting by adding new adaptive parameters to the model for each new task, such as block-modular neural networks [Terekhov et al., 2015], progressive neural networks [Rusu et al., 2016], and residual adapters [Rebuffi et al., 2017].

- The experience replay paradigm stores previously seen data from prior tasks, either directly or compressed via a generative model [Shin et al., 2017, Isele and Cosgun, 2018], and utilizes a multi-task objective that combines prior tasks and the new task by replaying previously seen data from prior tasks during training with the introduced new task.

Despite the relevant work, there is no holistic deep learning framework for continual learning to point to, that is capable of inducing the correct inductive bias, thus attenuating negative forward transfer, and mitigating catastrophic forgetting, in an efficient manner, that demonstrated success across numerous and various domains and applications.

2.10 Summary

The assumptions, theory, and formalisms of statistical machine learning, regularization techniques, and gradient-based optimization algorithms were described. The motivation, background, and outline of deep learning were provided. Feed-forward fully connected and convolutional neural networks, as well as recurrent neural networks, were detailed. Finally, machine learning paradigms that incorporate learning multiple tasks, such as transfer learning, multi-task learning, and in particular, continual learning, were presented.

In the next chapter, the theory presented herein will be applied to a number of supervised learning tasks, and in particular, classification tasks in machine perception, namely, image recognition, ASR, and Speech Emotion Recognition (SER).

Chapter 3

Tabula Rasa Learning

THE application of machine learning requires bridging the gap between theory and practice. The concepts, theory, and mathematical framework presented in the previous chapter are applied to supervised learning tasks in this chapter, and in particular, classification tasks in machine perception, specifically, image recognition, automatic speech recognition, and speech emotion recognition. The chapter builds understanding, insights, and baselines of the various tasks used throughout this thesis, and describes datasets, deep learning architectures, and best practices.

The speech emotion recognition task is comprehensively considered and used as a test bed to explore various neural network architectures. This is the first empirical exploration of various deep learning formulations and architectures applied to speech emotion recognition, which forms one of the main contributions of this thesis. As a result, state-of-the-art results are reported on the benchmark IEMOCAP dataset for speaker-independent speech emotion recognition. The work presented in this chapter is based on [Fayek et al., 2015, Fayek et al., 2016a, Fayek et al., 2017, Fayek, 2017].

Outline. This chapter is structured as follows. Section 3.1 outlines the machine perception problem. Section 3.2 briefly introduces the image recognition task and presents experiments on the task using the CIFAR-10, CIFAR-100, and SVHN datasets. Section 3.3 briefly delineates and reviews the automatic speech recognition task and presents experiments on the task using the TIMIT dataset. Section 3.4 describes the speech emotion recognition task, reviews related prior work, and presents experiments on the task with various neural network architectures using the IEMOCAP dataset. Section 3.5 discusses common findings for all tasks. Finally, Section 3.6 summarizes the chapter.

3.1 Machine Perception

Machine perception is the capability of machines to interpret sensory data, e.g., vision and hearing. Machine perception is vital to Artificial Intelligence (AI) as it can allow machines to perceive the world in a manner similar to the way humans perceive the world around them using their senses. Many tasks in machine perception can be described as tasks that are easy for humans to perform but difficult for them to formalize how it was performed, which makes the machine learning approach to such tasks highly pertinent. Machine perception is, therefore, one of the most active research areas in machine learning. The tasks studied in this thesis are all machine perception tasks.

Computer vision is a broad field that aims to enable machines to gain a high-level understanding of visual data, e.g., images and videos [Hartley and Zisserman, 2004,Szeliski, 2010]. Computer vision encompasses tasks such as object recognition, scene reconstruction, motion estimation, and video tracking. The image recognition task is one of the most popular tasks in the field of computer vision. The image recognition task involves mapping an image to a label denoting a particular category, e.g., mapping an image of handwritten digit(s) into labels denoting the digit(s) depicted in the image [LeCun et al., 1990]. The mapping of an image into a fixed set of labels denoting a particular category from a set of predefined categories works well for tasks such as handwritten digits recognition, where the number of possible categories are fixed, e.g., 0–9 in the handwritten digits recognition task. However, this fails to deal with more complex image recognition tasks, where the number of possible categories can be hard to define; c.f., the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Russakovsky et al., 2015] that comprises millions of natural images labelled into 1000 categories (see Appendix A.4 for more details), in which the categories can be difficult to partition.

Speech recognition is an interdisciplinary field that combines computational linguistics, computer science, and electrical engineering [Rabiner and Juang, 1993]. The recognition of the linguistic component of speech, i.e., the mapping of an acoustic signal containing spoken word(s) into the corresponding sequence of word(s) intended by the speaker(s), is referred to as *speech-to-text* or Automatic Speech Recognition (ASR). The recognition of the paralinguistic component of speech is the mapping of all other aspects of speech except the linguistic component, e.g., Speech Emotion Recognition (SER), gender recognition, or speaker recognition. Most speech systems are designed to recognize one particular aspect of speech; however, knowledge of both aspects of speech — the linguistic aspect and the paralinguistic aspect — is key to better speech processing and perception [Cowie et al., 2001]. It can be conjectured that the knowledge of either aspect may better facilitate modelling and recognition of the other. For instance, knowledge of the linguistic aspect of speech may boost the detection of the paralinguistic characteristics; conversely, knowledge of the paralinguistic characteristics may aid in characterizing the linguistic aspect [Fernandez, 2004].

3.2 Image Recognition

The image recognition task is reviewed and experiments on the task are presented. Section 3.2.1 is a brief introduction to the image recognition task. Section 3.2.2 details the experimental setup, including the CIFAR-10, CIFAR-100, and SVHN datasets, data preprocessing, the architectures of the models, as well as the training and implementation details. Finally, Section 3.2.3 presents the results.

3.2.1 Background

The image recognition task is one of the most popular tasks in the field of computer vision. The image recognition task involves mapping an image to labels denoting a particular category, e.g., mapping an image of a dog into labels denoting the breed of the dog depicted in the image. Many tasks in image recognition are facilitated by the availability of large (labelled) datasets, e.g., the ImageNet dataset, which makes image recognition tasks popular machine learning benchmarks.

The application of deep learning to image recognition is fairly straightforward. Visual data are characterized by two problems: the high dimensionality problem and the high variance problem [van der Maaten, 2009], which makes designing engineered features for visual data problematic, c.f., Histograms of Oriented Gradients (HOGs) [Dalal and Triggs, 2005] features and Scale Invariant Feature Transform (SIFT) [Lowe, 2004] features. The application of deep learning to image recognition led to dramatic performance improvements across almost all image recognition tasks [Krizhevsky et al., 2012], as well as other tasks in computer vision [LeCun et al., 2015].

The deep learning approach to image recognition typically operates on raw image pixels $\mathbf{x}$ using Convolutional Neural Networks (ConvNets) to predict a probability distribution $p(\mathbf{y}|\mathbf{x})$ over the labels $\mathbf{y}$. The raw image pixels typically undergo minimal preprocessing, e.g., normalizing the mean and standard deviation of the images per pixel across the entire dataset, and are then fed to a ConvNet that has the ability to learn hierarchical features across the entire image.

3.2.2 Experimental Setup

Datasets. The CIFAR-10 and CIFAR-100 datasets[*] [Krizhevsky, 2009] (see Appendix A.1 for more details), as well as the Street View House Numbers (SVHN)

[*]The CIFAR datasets are named after the Canadian Institute For Advanced Research (CIFAR).

dataset [Netzer et al., 2011] (see Appendix A.5 for more details), were used in this experiment.

The CIFAR-10 and CIFAR-100 datasets consist of RGB images, of size 32×32 pixels, labelled into 10 and 100 classes respectively. The 10 classes in the CIFAR-10 dataset are: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. The 100 classes in the CIFAR-100 dataset are listed in Table A.1. Each dataset comprises a training set of 50000 images and a test set of 10000 images, i.e., the CIFAR-10 dataset contains 5000 training images and 1000 test images per class, whereas the CIFAR-100 dataset contains 500 training images and 100 test images per class. For each CIFAR dataset, the original training set was split into a training set of 45000 images and a validation set of 5000 images; the entire test set was used for testing.

The SVHN dataset consists of RGB images, of size 32×32 pixels, labelled into 10 classes. The dataset is composed of house numbers obtained from Google Street View images, and thus, each of the 10 classes denotes a 0–9 digit. The training and test sets of the SVHN dataset contain 73257 and 26032 images respectively, and additionally, 531131 images are available that can be appended to the training set. The original training set and additional set were combined and split into a training set of 598388 images and a validation set of 6000 images. The entire test set was used for testing.

The training, validation, and test sets of all datasets are mutually exclusive. The partitioning of the datasets into training, validation, and test sets is similar to prior work [Sermanet et al., 2012, Goodfellow et al., 2013, Long et al., 2015].

Preprocessing. Standard data preprocessing steps were applied for all datasets as commonly practised in the literature [Sermanet et al., 2012, Goodfellow et al., 2013, Long et al., 2015]. The mean and standard deviation of the images in the CIFAR datasets were normalized to zero and one respectively per colour channel using the training set statistics for each dataset individually. The images in the SVHN dataset were scaled via division by 255 to lie in the $[0, 1]$ range. No data augmentation was used for all datasets.

Architectures. Two ConvNet architectures were used in this experiment, following recent developments in the field.

The first architecture used in this experiment is described in Table 3.1 and denoted Model **A**. The model is a standard ConvNet, which comprises four convolutional layers followed by two fully connected layers, with Batch Normalization (BatchNorm), Rectified Linear Units (ReLUs), and dropout interspersed in-between; a max pooling layer is inserted after the non-linear function of the second and fourth convolutional layers; the final fully connected layer is followed by a softmax function.

The second architecture used in this experiment follows the recent Densely Connected Convolutional Network (DenseNet) architecture [Huang et al., 2017] and is denoted Model **B**. This was shown to achieve state-of-the-art performance on the datasets used in this experiment [Huang et al., 2017]. The DenseNet architecture differs from other ConvNet architectures by connecting each layer to every other layer in a feed-forward fashion, i.e., the outputs of all preceding layers are used as inputs to a layer and its own outputs are used as inputs into all subsequent layers. The DenseNet architecture used is detailed in Table 3.2. The main layers in the architecture can be grouped into blocks based on their type and role. The dense blocks, Blocks 2, 4, and 6 in Table 3.2, comprise 12 layers of BatchNorm, ReLUs, convolution, and dropout. Each convolutional layer in Blocks 2, 4, and 6 in Table 3.2 is connected to all subsequent layers in the same block via the concatenation operation. The transition blocks, Blocks 3 and 5 in Table 3.2, are used to counteract the growth in the number of parameters due to the use of the concatenation operation, and are composed of a layer of BatchNorm, ReLUs, convolution, dropout, and average pooling. A down-sampling block, Block 7 in Table 4.1, is used to further reduce the complexity of the model, and is composed of BatchNorm, ReLUs, and average pooling. The output layer is a fully connected layer followed by a softmax function.

Training and Implementation Details. The training and implementation details of each architecture differed slightly, following the recommenced practice for each model in prior literature.

The parameters of the convolutional and fully connected layers of Model **A** were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers respectively, where n_i is the number of inputs to the layer, as recommended in [He et al., 2015]. Convolutional and fully connected layers in Model **A** were regularized using weight decay with penalty $\lambda = 1 \times 10^{-3}$ and dropout with drop probability r as indicated in Table 3.1. ADAM was used to optimize the parameters of Model **A** with respect to a negative log-likelihood loss function using batch size $M_b = 256$, learning rate $\alpha = 1 \times 10^{-3}$, first moment $\beta_1 = 0.99$, and second moment $\beta_2 = 0.999$, for 90 epochs. No early stopping was used in the case of the CIFAR datasets, whereas the validation set was used for early stopping in the case of the SVHN dataset.

The parameters of the convolutional and fully connected layers of Model **B** were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers respectively, where n_i is the number of inputs to the layer, similar to Model

Table 3.1: Convolutional neural network Model **A** architecture for the image recognition task. K denotes the number of output classes.

№	**Type**	**Size**	**Other**
1	Convolution	$32, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
2	Convolution	$32, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride $= 2$
	Dropout	—	$r = 0.25$
3	Convolution	$64, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
4	Convolution	$64, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride $= 2$
	Dropout	—	$r = 0.25$
5	Fully Connected	512	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.5$
6	Fully Connected	K	—
	Softmax	—	—

Table 3.2: Densely connected convolutional network Model **B** architecture for the image recognition task. The outputs of the convolutional layers in Blocks 2, 4, and 6, are concatenated with the inputs to the layer and fed to the subsequent layer in the same block. K denotes the number of output classes.

Block	**Repeat**	**Type**	**Size**	**Other**
1	1×	Convolution	$24, 3 \times 3$	Stride = 1
2	12×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$12, 3 \times 3$	Stride = 1
		Dropout	—	$r = 0.2$
3	1×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$168, 1 \times 1$	Stride = 1
		Dropout	—	$r = 0.2$
		Average Pooling	2×2	Stride = 2
4	12×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$12, 3 \times 3$	Stride = 1
		Dropout	—	$r = 0.2$
5	1×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$312, 1 \times 1$	Stride = 1
		Dropout	—	$r = 0.2$
		Average Pooling	2×2	Stride = 2
6	12×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$12, 3 \times 3$	Stride = 1
		Dropout	—	$r = 0.2$
7	1×	BatchNorm	—	—
		ReLU	—	—
		Average Pooling	8×8	Stride = 8
8	1×	Fully Connected	K	—
		Softmax	—	—

A. Convolutional and fully connected layers in Model **B** were regularized using weight decay with penalty $\lambda = 1 \times 10^{-4}$ and dropout with drop probability $r = 0.2$. Mini-batch Stochastic Gradient Descent (SGD) was used to optimize the parameters of Model **B** with respect to a negative log-likelihood loss function using batch size $M_b = 64$, learning rate $\alpha = 0.1$, and Nesterov momentum $\eta = 0.9$, for 300 and 40 epochs for the CIFAR and SVHN datasets respectively. The initial learning rate α was divided by 10 at 50% and 75% of the total number of training epochs. No early stopping was used in the case of the CIFAR datasets, whereas the validation set was used for early stopping in the case of the SVHN dataset. These settings follow [Huang et al., 2017].

The experiments were implemented using TensorFlow [Abadi et al., 2016]. Training a single model with the Model **A** architecture on a single NVIDIA Tesla P100 Graphics Processing Unit (GPU) required a wall time of approximately 30 and 60 mins for the CIFAR and SVHN datasets respectively. Training a single model with the Model **B** architecture on a single NVIDIA Tesla K80 GPU required a wall time of approximately 30 and 50 hours for the CIFAR and SVHN datasets respectively. Note that the difference in wall time between both models is due to the difference in complexity between the architectures, as well as the GPUs used.

3.2.3 Results

Table 3.3 lists the validation and test classification accuracies obtained for the CIFAR-10, CIFAR-100, and SVHN datasets using the models and training recipes described in Section 3.2.2. The standard ConvNet Model **A** is clearly outperformed by the DenseNet Model **B** across all three datasets. The validation and test classification accuracies of the DenseNet Model **B** across all three datasets are on par with the state-of-the-art [Huang et al., 2017]. The ConvNet Model **A** is computationally more efficient as the DenseNet Model **B** contains more parameters and more layers. This poses a trade-off between computational complexity and performance for both models.

3.3 Automatic Speech Recognition

The ASR task is introduced and formulated, and experiments on the task are presented. Section 3.3.1 briefly describes the ASR task. Section 3.3.2 outlines the ASR system. Section 3.3.3 details the experimental setup, including the TIMIT dataset, data preprocessing, the architectures of the models, as well as the training and implementation details. Finally, Section 3.3.4 presents the results.

Table 3.3: Image recognition Validation Classification Accuracy (Val ACC) and Test Classification Accuracy (Test ACC) on the CIFAR-10, CIFAR-100, and SVHN datasets.

Dataset	**Model**	**Val ACC (%)**	**Test ACC (%)**
CIFAR-10	ConvNet Model **A**	84.86	83.36
	DenseNet Model **B**	100	92.91
CIFAR-100	ConvNet Model **A**	63.32	64.24
	DenseNet Model **B**	99.84	70.66
SVHN	ConvNet Model **A**	94.52	95.00
	DenseNet Model **B**	97.48	98.15

3.3.1 Background

The ASR task aims to map an acoustic signal containing spoken word(s), i.e., an utterance, into the corresponding sequence of word(s) intended by the speaker(s). ASR has numerous stand-alone applications, e.g., dictation in word processors, or can be integrated into larger applications, e.g., speech interfaces.

Since the inception of the task until recently, the state-of-the-art approach to ASR systems employed a hybrid Gaussian Mixture Model (GMM)-Hidden Markov Model (HMM) system. The GMM-HMM system uses HMMs to model the temporal variability of speech and GMMs to determine how well each state of the HMMs fits a small chunk of the acoustic signal, denoted frame, or a short window of frames, represented by features [Rabiner and Juang, 1993]. In these systems, the acoustic signal is typically represented by Mel Frequency Cepstral Coefficients (MFCCs) computed from the raw acoustic signal.

More recently, Deep Neural Networks (DNNs) displaced GMMs in GMM-HMM systems leading to DNN-HMM systems [Mohamed et al., 2012, Mohamed, 2014]. The DNN-HMM systems dramatically improved the performance of ASR systems compared with GMM-HMM systems [Dahl et al., 2012]. The DNN in a DNN-HMM system ingests a short window of frames of features that represent the acoustic signal, and predicts the conditional probability distribution over the states of the HMMs for typically the central frame in the window of frames ingested, using adjacent frames in the window as context to aid prediction. The use of DNNs in ASR systems paved the way for less-engineered features, e.g., Mel Frequency Spectral Coefficients (MFSCs), as opposed to MFCCs commonly used in GMM-HMM systems.

MFSCs require computing the Discrete Fourier Transform (DFT) over each frame, followed by applying a number of filter banks distributed on a Mel scale (see Figure 3.2) to obtain a time-frequency spectrogram, as illustrated in Figure 3.1. MFCCs require an additional step to compute the Discrete Cosine Transform (DCT) of the log-MFSCs to obtain a more compressed decorrelated representation shown in Figure 3.1. The application of the DCT on the log-MFSCs resulting in MFCCs, maintaining only the first N coefficients and discarding the rest, also reportedly removes some inter-speaker variability, which is beneficial in GMM-HMM systems; however, DNNs can utilize the speaker variably in log-MFSCs for enhanced speech modelling [Mohamed, 2014]. Prior studies have also investigated the use of DNNs on the raw acoustic signal, reporting promising results [Jaitly and Hinton, 2011].

With the advent of DNNs in ASR systems, multiple neural network architectures were subsequently studied. In [Abdel-Hamid et al., 2014], the DNN in a DNN-HMM system was replaced with a ConvNet leading to a ConvNet-HMM system to learn features across both time and frequency, demonstrating improved results. In [Graves et al., 2013], deep Long Short-Term Memory (LSTM)-Recurrent Neural Networks (RNNs) were used in an end-to-end ASR system, without the HMMs, reporting competitive results given suitable regularization. In [Amodei et al., 2016], a hybrid ConvNet-RNN was used in an end-to-end ASR system, achieving improved performance in clean and noisy utterances for multiple languages.

3.3.2 Automatic Speech Recognition System

The ASR system used in this work is a hybrid Neural Network (NN)-HMM system similar to those of [Mohamed et al., 2012, Dahl et al., 2012, Abdel-Hamid et al., 2014, Mohamed, 2014].

First, the incoming speech utterance is processed. Let $\mathbf{s}$ be a speech utterance of arbitrary length. The speech utterance is divided into small equal overlapping chunks, typically of 25 ms every 10–15 ms, denoted frames. A Hamming window is applied to each frame to reduce the discontinuity at the start and end of the frames [Allen, 1977]. Each frame is then converted from the time domain to the frequency domain via DFT. Overlapping triangular filter banks, distributed on a Mel scale to mimic the human ear sensitivity (see Figure 3.2), are subsequently applied to each frame, to obtain the MFSCs. Typically, the logarithm function is applied to the MFSCs to produce log-MFSCs that are preferred due to improved discrimination. The speech utterance $\mathbf{s}$ is now $\mathbf{S} \in \mathbb{R}^{T \times N}$: a sequence of T frames such that each frame is represented by N features.

Second, the processed speech utterance is fed to the neural network that is a ConvNet. The ConvNet ingests a short window of concatenated consecutive frames of MFSCs, $\mathbf{x}_t = \mathbf{S}_{(t-l)} || \ldots || \mathbf{S}_t || \ldots || \mathbf{S}_{(t+r)}$, where $||$ denotes the concatenation operation, to predict the conditional probability distribution $p(\mathbf{q}|\mathbf{x}_t)$ over the states of the HMMs $\mathbf{q}$, given the window of frames of MFSCs $\mathbf{x}_t$ for the central frame

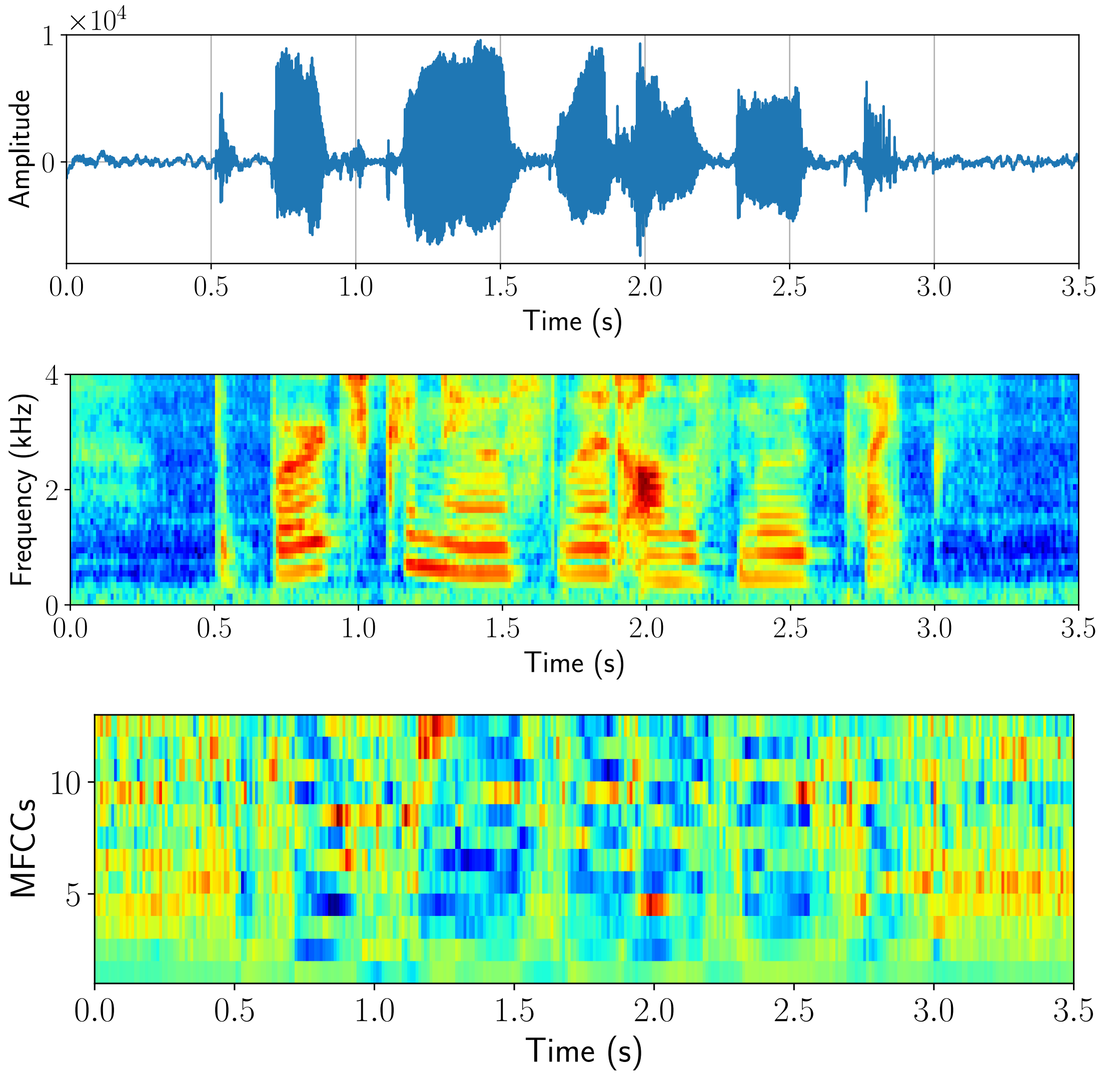

Figure 3.1: Acoustic speech signal. **Top.** Raw acoustic signal in the time domain. **Centre.** Normalized log Mel Frequency Spectral Coefficients (MFSCs) computed from the raw acoustic signal (top). **Bottom.** Normalized Mel Scale Cepstral Coefficients (MFCCs) computed from the raw acoustic signal (top).

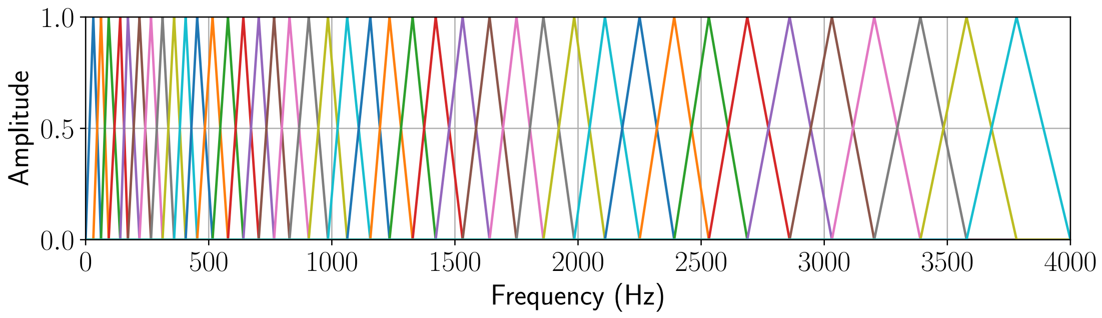

Figure 3.2: Mel scale filter banks. The amplitudes of 40 filter banks on a Mel scale in the frequency range 0–4 kHz.

$\mathbf{S}_t$, using the adjacent frames in the window, $\mathbf{S}_{(t-l)}$ and $\mathbf{S}_{(t+r)}$, as context to aid prediction. A three-state HMM is employed per phoneme; thus, the ConvNet is required to predict $p(\mathbf{q}|\mathbf{x}_t)$ over $(3 \times \text{number of phonemes})$ states.

Note that the ConvNet is trained independently and therefore requires HMM state labels for each frame. The force-aligned frame labels were obtained by training a mono-phone GMM-HMM system with MFCCs.

Third, the likelihood $p(\mathbf{x}_t|\mathbf{q})$ is estimated using Bayes' rule:

$$p(\mathbf{x}_t|\mathbf{q}) = \frac{p(\mathbf{q}|\mathbf{x}_t)p(\mathbf{x}_t)}{p(\mathbf{q})}. \tag{3.1}$$

The likelihoods, $p(\mathbf{x}_t|\mathbf{q})$; $\forall t \in T$, are then fed into a standard Viterbi decoder [Viterbi, 1967] to obtain the predicted sequence of phonemes. Finally, a bi-gram language model, estimated from the training set, is used over the predicted phonemes to favour more probable phoneme sequences [Katz, 1987, Jurafsky and Martin, 2014].

3.3.3 Experimental Setup

Dataset. The Texas Instruments Massachusetts Institute of Technology (TIMIT) dataset [Garofolo et al., 1993] (see Appendix A.6 for more details) was used in this experiment. The TIMIT dataset was collected to advance ASR systems and released in 1990. The TIMIT dataset contains recordings of 630 speakers, of which 70% are males and 30% are females, from eight major American English dialects, with each speaker reading ten phonetically rich sentences, amounting to a total of 6300 utterances. The sampling rate of all recordings is 16 kHz. Each utterance is accompanied by a phonetic transcript.

There are 61 phonemes in the TIMIT dataset. As commonly practised in the literature, the 61 phonemes were mapped into 48 phonemes for training, which were then mapped into 39 phonemes for scoring (see [Lee and Hon, 1989] for more details).

The dataset is predeterminedly divided into mutually exclusive training and test sets [Garofolo et al., 1993, Lee and Hon, 1989, Povey et al., 2011]. The complete 462-speaker training set, without the dialect (SA) utterances, was used as the training set. The 50-speaker development set was used as the validation set. The 24-speaker core test set was used as the test set.

Preprocessing. Utterances were split into 25 ms frames with a stride of 10 ms, and a Hamming window was applied, then 40 log-MFSCs were extracted from each frame. The mean and standard deviation were normalized per coefficient to zero and one respectively using the mean and standard deviation computed on the training set. No speaker dependent operations were performed. The Kaldi toolkit [Povey et al., 2011] was used to produce force-aligned frame labels by training a mono-phone GMM-HMM system with MFCCs. The input to the neural network was either 31 or 41 consecutive frames, depending on the architecture of the neural network, labelled using the label of the central frame.

Architectures. The ASR system had a hybrid ConvNet-HMM architecture: a ConvNet acoustic model was used to produce a probability distribution over the states of three-state HMMs with a bi-gram language model estimated from the training set. Two ConvNet architectures were used in this experiment, following recent developments in the field.

The first ConvNet architecture used in this experiment is described in Table 3.4 and denoted Model **A**. The model is a standard ConvNet that comprises two convolutional and max pooling layers, followed by four fully connected layers, with BatchNorm and ReLUs interspersed in-between. The final fully connected layer is followed by a softmax function to predict the probability distribution over 144 classes, i.e., three HMM states per 48 phonemes.

The second ConvNet architecture used in this experiment is a variant of the popular VGGNet architecture [Simonyan and Zisserman, 2014, Sercu et al., 2016] as described in Table 3.5 and is denoted Model **B**. The architecture comprises a number of convolutional, BatchNorm, and ReLUs layers, with a few max pooling layers used throughout the architecture as indicated in Table 3.5, followed by three fully connected layers, with BatchNorm, ReLUs, and dropout interspersed in-between. The final fully connected layer is followed by a softmax function to predict the probability distribution over 144 classes, i.e., three HMM states per 48 phonemes.

Table 3.4: Convolutional neural network Model **A** architecture for the automatic speech recognition task.

№	**Type**	**Size**	**Other**
1	Convolution	$64, 5 \times 4$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride $= 2$
2	Convolution	$128, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride $= 2$
3	Fully Connected	1024	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.4$
4	Fully Connected	1024	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.4$
5	Fully Connected	1024	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.4$
6	Fully Connected	144	—
	Softmax	—	—

Table 3.5: Convolutional neural network Model **B** architecture for the automatic speech recognition task.

№	**Repeat**	**Type**	**Size**	**Other**
1	1×	Convolution	$64, 6 \times 5$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Convolution	$64, 3 \times 3$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Max Pooling	2×2	Stride $= 2$
2	2×	Convolution	$128, 3 \times 3$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Max Pooling	2×2	Stride $= 2$
3	3×	Convolution	$256, 3 \times 3$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Max Pooling	2×2	Stride $= 2$
4	3×	Convolution	$256, 3 \times 3$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Max Pooling	2×2	Stride $= 2$
5	1×	Fully Connected	1024	—
		BatchNorm	—	—
		ReLU	—	—
		Dropout	—	$r = 0.4$
6	1×	Fully Connected	1024	—
		BatchNorm	—	—
		ReLU	—	—
		Dropout	—	$r = 0.4$
7	1×	Fully Connected	144	—
		Softmax	—	—

Training and Implementation Details. The training and implementation details of each architecture differed slightly following the recommenced practice for each model in prior literature. The hyperparameters reported below were tuned using the validation set.

The parameters of the convolutional and fully connected layers of Model **A** were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers respectively, where n_i is the number of inputs to the layer, as recommended in [He et al., 2015]. Convolutional layers were regularized using weight decay with penalty $\lambda = 1 \times 10^{-3}$ and fully connected layers were regularized using dropout with drop probability $r = 0.4$. RMSProp was used to optimize the parameters of Model **A** with respect to a negative log-likelihood loss function using batch size $M_b = 256$, learning rate $\alpha = 1 \times 10^{-3}$, and decay rate $\eta = 0.99$. The validation set was used for early stopping such that training halts if the validation error ceases to improve for three epochs.

The parameters of the convolutional and fully connected layers of Model **B** were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers respectively, where n_i is the number of inputs to the layer, similar to Model **A**. All layers in Model **B** were regularized using weight decay with penalty $\lambda = 1 \times 10^{-3}$ and fully connected layers were also regularized using dropout with drop probability $r = 0.4$. ADAM was used to optimize the parameters of Model **B** with respect to a negative log-likelihood loss function using batch size $M_b = 256$, learning rate $\alpha = 1 \times 10^{-3}$, first moment $\beta_1 = 0.99$, and second moment $\beta_2 = 0.999$ for 110×10^3 iterations.

The experiments were implemented using MatDL [Fayek, 2017] and TensorFlow [Abadi et al., 2016][†]. Training was carried out using NVIDIA Tesla K40 and K80 GPUs. Training a single model on a single GPU required a wall time of approximately 15–48 hours depending on the architecture and GPU used.

3.3.4 Results

Table 3.6 lists the validation and test Frame Error Rate (FER) and Phone Error Rate (PER) obtained for the TIMIT dataset using the models and training recipes described in Section 3.3.3. The FER is the classification error rate of the ConvNet acoustic model, while the PER is the minimum edit distance between the predicted output of the entire ConvNet-HMM system following decoding and the transcript. The ConvNet Model **A** is outperformed by the ConvNet Model **B** due to the

[†]Each model, Model **A** and Model **B**, were implemented using both frameworks, MatDL and Tensorflow, producing similar results.

Table 3.6: Automatic speech recognition validation and test Frame Error Rate (FER) and Phone Error Rate (PER) on the TIMIT dataset.

Model	**FER**		**PER**	
	Validation	**Test**	**Validation**	**Test**
ConvNet Model **A**	30.53%	31.61%	18.71%	20.18%
ConvNet Model **B**	29.35%	30.69%	17.67%	19.4%

increased depth and capacity of the later. Slightly better results could be achieved by further tuning the architecture or hyperparameters of the models [Zhang et al., 2016] or using attention-based sequence-to-sequence models [Chorowski et al., 2015].

3.4 Speech Emotion Recognition

The SER task and related work are reviewed, and experiments on the task are presented, focusing on various architectures of neural networks. Deep learning has been applied to SER in prior work as discussed below. However, with various experiment conditions involved in prior studies, it is difficult to gauge the strengths and drawbacks of the different deep learning formulations and architectures. As a result of the novel empirical exploration carried out in this section of various deep learning formulations and architectures applied to SER, state-of-the-art results are reported on the benchmark IEMOCAP dataset for speaker-independent SER[‡].

Section 3.4.1 briefly introduces the SER task. Section 3.4.2 reviews related prior work. Section 3.4.3 outlines the SER system. Section 3.4.4 details the experimental setup, including the IEMOCAP dataset, data preprocessing, as well as the training and implementation details. Finally, Section 3.4.5 presents and discusses the results.

3.4.1 Background

The SER task aims to automatically identify the affective state of a human from their speech [Cowie et al., 2001]. SER can be employed in stand-alone applications, e.g., emotion monitoring, or integrated into other systems for emotional awareness, e.g., integrating SER into an ASR system to improve the capability of the system in dealing with emotional speech.

SER can be regarded as a static or dynamic classification problem, which has motivated two popular formulations to the task in the literature [Ververidis and

[‡]Compared to prior literature when the work was published in [Fayek et al., 2017].

Kotropoulos, 2006]: *turn-based processing*, also known as static modelling; and *frame-based processing*, also known as dynamic modelling. Turn-based processing aims to recognize emotions from a complete utterance, whereas frame-based processing aims to recognize emotions at a more granular level, i.e., the frame level. Frame-based processing is more robust since it does not rely on segmenting the input speech into utterances and can model intra-utterance emotion dynamics [Arias et al., 2013]. Moreover, frame-based processing allows the possibility of real-time SER systems. However, empirical comparisons between frame-based processing and turn-based processing in prior work have generally demonstrated the superiority of the latter [Vlasenko et al., 2007, Schuller et al., 2009b].

Whether performing turn-based processing or frame-based processing, most prior research in the last decade has been devoted to selecting an optimal set of features [Schuller et al., 2010]. Despite the effort, little success has been achieved in realizing a set of features that performs consistently over different conditions and datasets [Eyben et al., 2016]. Thus, brute-force high-dimensional feature sets, that comprise many acoustic parameters, have been used, in an attempt to capture all variances [Tahon and Devillers, 2016]. Such high-dimensional feature sets complicate the learning process in most machine learning algorithms, increase the likelihood of over-fitting, and hinder generalization. Moreover, the computation of many acoustic parameters is computationally expensive and may be difficult to apply on a large scale with limited resources [Eyben et al., 2015]. Therefore, the application of deep learning to SER is pertinent to alleviate the problem of feature engineering and achieve an SER with a simple pipeline and low latency. Further, SER is an excellent test bed for exploring various deep learning architectures since the task itself can be formulated in multiple ways.

3.4.2 Related Work

Prior literature on SER is well surveyed in [Ververidis and Kotropoulos, 2006, Ayadi et al., 2011, Petta et al., 2011]. Recent work has mostly focused on deep learning approaches to SER. This follows the success of DNNs in ASR, as discussed in Section 3.3.1, which has prompted research into the application of DNNs to other areas of speech recognition.

In [Stuhlsatz et al., 2011], a generalized discriminant analysis based on a DNN was proposed to deal with the high-dimensional feature sets commonly used in SER, demonstrating better performance than Support Vector Machines (SVMs) on the same set of features. In [Li et al., 2013], a hybrid DNN-HMM system trained with MFCCs was proposed for SER, indicating improved results compared with a GMM-HMM system. In [Han et al., 2014], a DNN was used to extract features from speech segments, which were then used to construct utterance-level features that were fed into an Extreme Learning Machine (ELM) for utterance-level classification outperforming competing techniques. In [Fayek et al., 2016a], a DNN was used to learn a mapping from MFSCs to emotion classes using soft labels generated from multiple annotators to model the subjectiveness in emotion recognition, which yielded improved performance compared with ground truth labels obtained by majority voting between the same annotators.

More recently, alternative neural network architectures were also investigated for SER. In [Mao et al., 2014], a ConvNet was used in a two-stage SER system that involved learning local invariant features using a sparse auto-encoder from speech spectrograms, processed using Principal Component Analysis (PCA), followed by salient discriminative feature analysis to extract discriminative features demonstrating competitive results. In [Tian et al., 2015a], the use of knowledge-inspired disfluency and non-verbal vocalization features in emotional speech were compared with the use of a feature set comprising acoustic parameters aggregated using statistical functions, by using an LSTM-RNN as well as an SVM classifier; the former was shown to produce better results given enough data.

The work presented in this section differs from prior work in several ways. A frame-based formulation to SER is presented with the aim of achieving a system with a simple pipeline and low latency by modelling the intra-utterance emotion dynamics. Moreover, most previous studies relied on engineered features, whereas in this work, minimal speech processing is employed, and deep learning is relied on to automate the process of learning features. Furthermore, unlike previous studies, uniform data subsets and experiment conditions were used to promote comparisons across various deep learning formulations and architectures for SER.

3.4.3 Speech Emotion Recognition System

Figure 3.3 is a sketch of the proposed SER system. The system follows a frame-based processing formulation that utilizes MFSCs and a deep multi-layered neural network to predict a probability distribution over emotion classes for each frame in the input utterance.

Let $\mathbf{S} \in \mathbb{R}^{T \times N}$ be a sequence of T frames such that each frame is represented by N features computed from a speech utterance or speech stream. The aim is to rely on minimal speech processing, and thus, each frame is represented by N log-MFSCs, as described in Section 3.3.2.

The objective of the model is to predict $p(\mathbf{y}_t|\mathbf{x}_t)$, where $\mathbf{y}_t$ is the predicted output corresponding to the target frame(s) $\mathbf{x}_t$, $\mathbf{x}_t = \mathbf{S}_{(t-l)} || \dots || \mathbf{S}_t || \dots || \mathbf{S}_{(t+r)}$, l is the number of past context frames, and r is the number of future frames. *Silence* was added to the output classes, i.e., the final output is either one emotion class or *silence*, since silence and unvoiced speech were not removed from the input speech utterance, as it has been shown that silence and other disfluencies can be an effective cue in emotion recognition [Tian et al., 2015b].

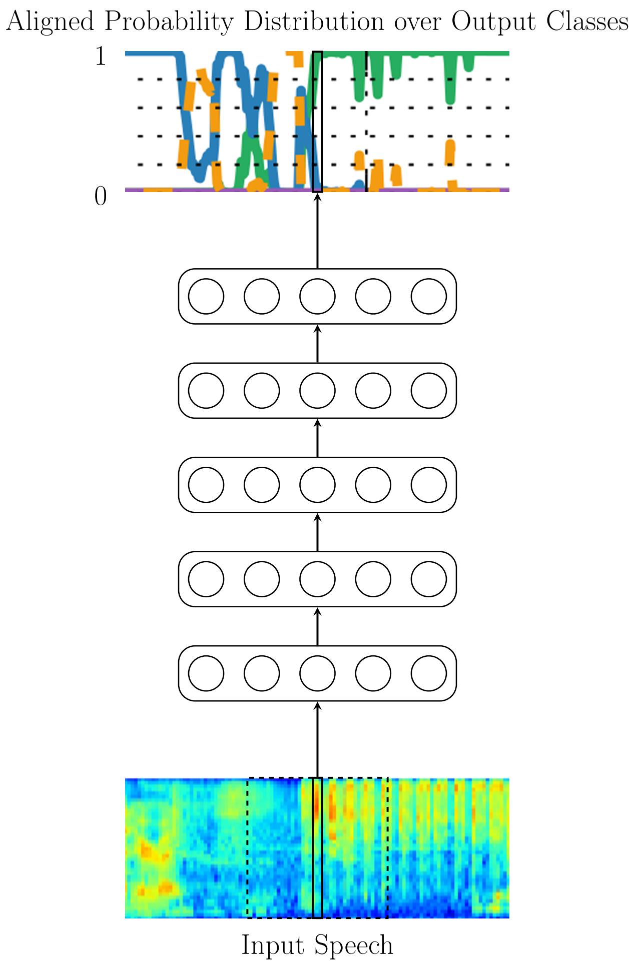

Figure 3.3: Overview of the proposed speech emotion recognition system. A deep multi-layered neural network, composed of several fully connected, convolutional, or recurrent layers, ingests a target frame (solid), concatenated with a number of context frames (dashed), to predict the probabilities over emotion classes corresponding to the target frame.

The proposed model is a deep multi-layered neural network. Note that the model is able to deal with utterances of variable length, independent of the choice of the architecture, since the model predicts $p(\mathbf{y}_t|\mathbf{x}_t)$, $\forall t \in T$; this only requires the target frame $\mathbf{S}_t$ and the past l and future r context frames, which are set throughout the system. Since emotions manifest in speech in a slow manner, one may not necessarily predict the class of every single frame in an utterance or speech stream but may rely on predicting the class of a frame sampled every few frames, depending on the requirements of the application. The output of the model can be aggregated over the entire utterance to perform utterance-level classification if desired.

3.4.4 Experimental Setup

Dataset. The Interactive Emotional Dyadic Motion Capture (IEMOCAP) dataset [Busso et al., 2008] (see Appendix A.3 for more details) was used in this experiment. The dataset comprises 12 hours of audio-visual recordings divided into five sessions. Each session is composed of two actors, a male and a female, performing emotional scripts as well as improvised scenarios. In total, the dataset comprises 10039 utterances sampled at 48 kHz with an average duration of 4.5 s.

Utterances were labelled by three annotators using categorical labels. The dataset predominantly focused on five emotions, namely, *anger*, *happiness*, *sadness*, *neutral*, and *frustration*; however, annotators were not limited to these emotions during annotation. Ground truths labels were obtained by majority voting, where 74.6% of the utterances were agreed upon by at least two annotators. Utterances that were labelled differently by all three annotators were discarded. To be consistent with other studies on this dataset [Shah et al., 2014, Mariooryad and Busso, 2013, Yelin et al., 2013], utterances that bore only the following four emotions: *anger*, *happiness*, *sadness*, and *neutral*, were included with *excitement* considered as *happiness*; amounting to a total of 5531 utterances.

An eight-fold Leave-One-Speaker-Out (LOSO) cross-validation scheme [Schuller et al., 2009b] was employed in all experiments using the eight speakers in the first four sessions. Both speakers in the fifth session were used as the validation set, and hence were not included in the cross-validation folds to avoid biasing the results [Refaeilzadeh et al., 2009].

Preprocessing. Utterances were split into 25 ms frames with a stride of 10 ms, and a Hamming window was applied, then 40 log-MFSCs were extracted from each frame. The mean and standard deviation were normalized per coefficient to zero and one respectively for each fold using the mean and standard deviation computed using the training subset only. No speaker dependent operations were performed.

Since the data was labelled at utterance-level, all frames in an utterance inherited that utterance label. A voice activity detector was subsequently used to label silent frames, and *silence* was added as an additional class to the four previously mentioned emotion classes, i.e., a frame had either the same label as its parent utterance or the *silence* label. The underlying assumption here is that frames in an utterance convey the same emotion as the parent utterance, which concurs with the same assumption made when a categorical label was assigned to the entire utterance; nevertheless, this assumption is eased by labelling unvoiced and silent frames as *silence*.

Architectures. Several neural network architectures were investigated as presented in Section 3.4.5.

Training and Implementation Details. The parameters of the neural networks were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ standard deviation, where n_i is the number of inputs to the layer, as recommended in [He et al., 2015]. Fully connected layers were regularized using dropout with drop probability $r = 0.5$. Convolutional layers were regularized using weight decay with penalty $\lambda = 1 \times 10^{-3}$. LSTM layers were regularized using dropout with drop probability $r = 0.5$ and the gradients were clipped to lie in the range $[-5, 5]$. BatchNorm was used after every fully connected or convolutional layer. RMSProp was used to optimize the parameters of the models with respect to a negative log-likelihood loss function using batch size $M_b = 256$. The initial learning rate was set to $\alpha = 1 \times 10^{-2}$ and annealed by a factor of 10 when the error plateaus. The decay rate was set to $\eta = 0.99$. The validation set was used to perform early stopping during training, such that training halts when the learning rate reaches $\alpha = 1 \times 10^{-8}$; the model with the best accuracy on the validation set during training was selected. These hyperparameters were chosen based on experimental trials using the validation set.

The Kaldi toolkit [Povey et al., 2011] was used for speech processing. The experiments were implemented using MatDL [Fayek, 2017]. Due to the large number of experiments carried out, several computational resources were exploited at difference stages. Some experiments were carried out on a cluster of Central Processing Units (CPUs), whereas other experiments were carried out using GPUs. Training wall time varied significantly between different models. The largest model took 14 days to train on a single GPU, but the average wall time was two days.

3.4.5 Results

The classification accuracy is reported to gauge the performance of the models, as well as the Unweighted Average Recall (UAR) to reflect imbalanced classes [Schuller et al., 2009a], as commonly practised in the field of automatic emotion recognition. Both metrics are averaged across the eight-fold LOSO cross-validation scheme described in Section 3.4.4.

Feed-Forward Architectures The number of context frames required for SER is investigated. It is hypothesized that unlike ASR, SER does not rely on future context, but does require a large number of past context frames. Hence, the models were trained in two configurations: (1) the first configuration was to predict $p(\mathbf{y}_t|\mathbf{x}_t)$, where $\mathbf{x}_t = \mathbf{S}_{t-c}||\dots||\mathbf{S}_t||\dots||\mathbf{S}_{t+c}$ for various values of c, i.e., predict the class label of the central frame; (2) the second configuration was to predict $p(\mathbf{y}_t|\mathbf{x}_t)$, where $\mathbf{x}_t = \mathbf{S}_{t-c}||\dots||\mathbf{S}_t$ for various values of c, i.e., predict the class label of the final frame. The model used in this experiment was a DNN with five hidden layers, each of which is composed of 1024 fully connected units with BatchNorm, ReLUs, and dropout layers interspersed in-between, and a softmax function after the final fully connected layer. This architecture was selected based on the best UAR on the validation set, which was excluded from the cross-validation scheme. Figure 3.4 is a plot of the test accuracy and test UAR of the model in both configurations for various numbers of context frames.

Two observations are immediately evident from the results in Figure 3.4: (1) the performance of the system is directly proportional to the number of context frames until 220 frames, where it starts to plateau; (2) future context has a minor contribution to the performance of the system as hypothesized. A large number of context frames leads to an increase in the dimensionality of the input, and may increase over-fitting, as shown in Figure 3.4; a trade-off between the number of context frames and the performance of the system lies in the range of 2–3 seconds of speech.

Considering the results obtained in Figure 3.4, the application of ConvNets to SER is well motivated. ConvNets are able to deal with high-dimensional input, which in this case is due to the large number of context frames required. Moreover, ConvNets are able to learn features that are insensitive to small variations in the input, which can aid in disentangling inter-speaker variations, as well as other sources of distortion, e.g., noise.

The following experiments present an in-depth exploration of various ConvNet architectures, where the effects of the number of convolutional and fully connected layers, number of filters, size of filters, and type of convolution (spatial vs temporal) on the performance of the system are investigated. All experiments were conducted using 259 past context frames and no future context frames, which corresponds

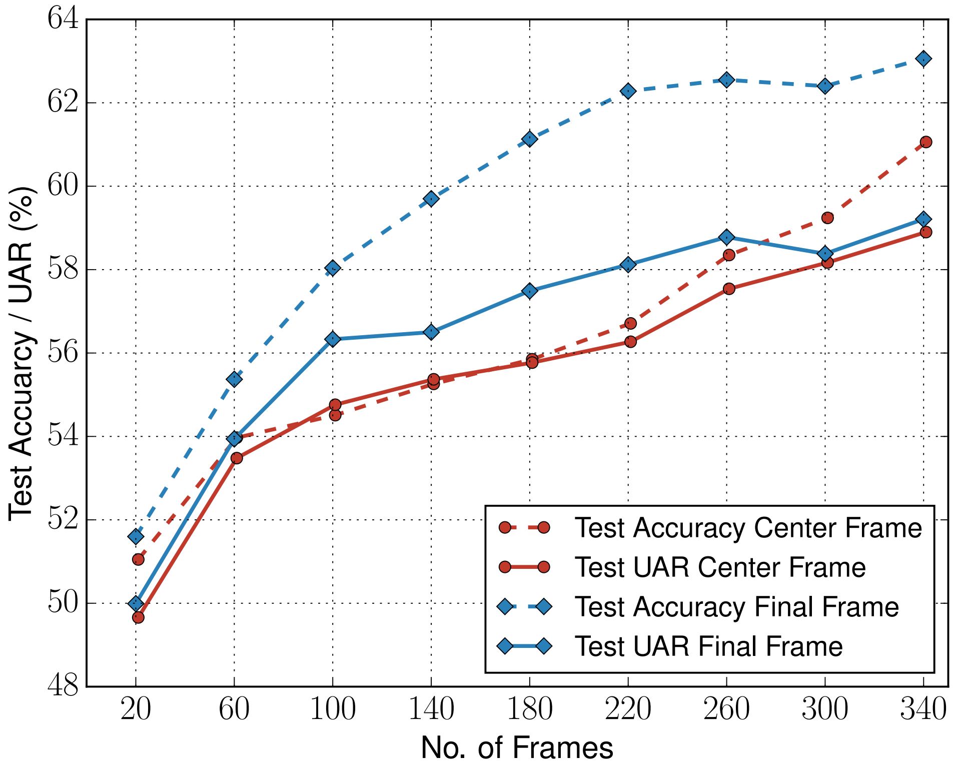

Figure 3.4: Speech emotion recognition test accuracy and test Unweighted Average Recall (UAR) on the IEMOCAP dataset with a deep neural network as a function of the number of context frames.

Table 3.7: Speech emotion recognition test Accuracy (ACC) and test Unweighted Average Recall (UAR) on the IEMOCAP dataset with various convolutional neural network architectures. Conv($c \times j \times k$) and Conv1D($c \times j \times k$) denote a spatial convolutional layer and a temporal convolutional layer respectively of c filters, each of size $j \times k$, with stride 2, followed by Batch Normalization (BatchNorm) and Rectified Linear Units (ReLUs). FC(n_l) denotes a fully connected layer of n_l units followed by BatchNorm, ReLUs, and dropout. All architectures have a fully connected layer with a softmax function as the output layer.

Architecture	**Test ACC (%)**	**Test UAR (%)**
Conv(32 × 4 × 4) — FC(1024)×2	62.27	58.30
Conv(32 × 4 × 4) — FC(1024)×3	62.78	58.87
Conv(32 × 4 × 4) — Conv(64 × 3 × 3) — FC(1024)×2	62.58	58.71
Conv(32 × 4 × 4) — Conv(64 × 3 × 3) — FC(1024)×3	63.16	58.56
Conv(16 × 4 × 4) — Conv(32 × 3 × 3) — FC(716)×3	63.34	59.30
Conv(32 × 4 × 4) — Conv(64 × 3 × 3) — FC(1024)×4	63.82	58.92
Conv(16 × 4 × 4) — Conv(32 × 3 × 3) — FC(716)×4	62.90	58.17
Conv(16 × 6 × 6) — Conv(32 × 6 × 6) — FC(716)×3	63.51	59.50
Conv(16 × 10 × 10) — Conv(32 × 10 × 10) — FC(716)×3	**64.78**	**60.89**
Conv(16 × 14 × 14) — Conv(32 × 14 × 14) — FC(716)×3	62.84	58.30
Conv(16 × 10 × 18) — Conv(32 × 18 × 18) — FC(716)×3	63.07	58.79
Conv1D(64 × 40 × 4) — FC(1024)×3	62.41	58.38
Conv1D(64 × 40 × 8) — FC(1024)×3	62.98	59.07
Conv1D(64 × 40 × 16) — FC(1024)×3	62.91	58.49

to approximately 2.6 s of speech, i.e., the input dimensionality is 40 filter banks × 260 frames. Table 3.7 lists various ConvNet architectures and their respective test accuracy and test UAR.

From the results listed in the first segment of Table 3.7, the benefit of the depth of the model can be observed. The best results were obtained using two convolutional layers followed by 2–3 fully connected layers. The addition of more layers to the model did not yield any performance gain but conversely resulted in over-fitting. The results in the second segment of Table 3.7 demonstrate the effect of the filter size on the performance of the model. It can be seen that, similar to other speech applications, SER requires a relatively large filter with an optimal size of 10 × 10. Temporal convolution was slightly outperformed by spatial convolution, as demonstrated in the final segment of Table 3.7.

Recurrent Architectures The application of LSTM-RNNs for the proposed SER system is investigated in the following experiments. LSTM-RNNs can be trained in several ways, e.g., *sequence-to-sequence*, where a model is trained to ingest a sequence of frames and output a sequence of class labels, or *sequence-to-one*, where a model is trained to ingest a sequence of frames and output a class label. Sequence-to-sequence training may seem to be a better fit to the proposed system; however, preliminary experiments demonstrated the superiority of sequence-to-one training, whereas sequence-to-sequence training failed to converge in most cases or had poor performance otherwise. Therefore, sequence-to-one training was used in the experiments; i.e., the model was trained to ingest a sequence of frames, frame-by-frame, and predict a class label for the final frame, $p(\mathbf{y}_t|\mathbf{x}_t)$, where $\mathbf{x}_t = \mathbf{S}_{t-c}||\dots||\mathbf{S}_t$, and c is the number of context frames (sequence length).

LSTM-RNNs can handle sequences of arbitrary lengths. However, the effect of the sequence length, on which the model was trained, on the ability of the model to handle arbitrary sequence lengths is not well-studied. Hence, several models were trained using various training sequence lengths $\{20, 60, 100, 200\}$, where LSTM-RNN-c denotes the training sequence length c on which the model was trained on; this model was then evaluated on a number of test sequence lengths $\{20, 60, 100, 200, 260, 300\}$. An extra model was trained on sequence length c chosen randomly at each iteration such that $c \in \{20, 60, 100, 200\}$, denoted LSTM-RNN-R. The model used in this experiment was a two-layered LSTM-RNN with 256 units in each hidden layer and dropout interspersed in-between, and a final fully connected layer followed by a softmax function. This architecture was selected based on the best UAR on the validation set, which was excluded from the cross-validation scheme. Figure 3.5 and Figure 3.6 depict the test accuracy and test UAR respectively of the LSTM-RNNs trained and evaluated on various sequence lengths.

The results in Figure 3.5 and Figure 3.6 demonstrate a similar trend in that models trained on short sequences did not perform as well as on long sequences and vice versa. In addition, noticeable gains in performance could be achieved by increasing the number of context frames. The best performance at each test sequence length was obtained by the model trained on the same sequence length; the performance degraded gradually as the test sequence length deviated from the training sequence length. Moreover, by varying the sequence length when training LSTM-RNN-R, the model did learn to perform well on various test sequence lengths. On average, LSTM-RNN-100 yielded the best UAR averaged over all test sequence lengths, followed by LSTM-RNN-R.

Analysis. Table 3.8 lists the best model from each architecture and their respective test accuracy and test UAR trained and evaluated under the same data

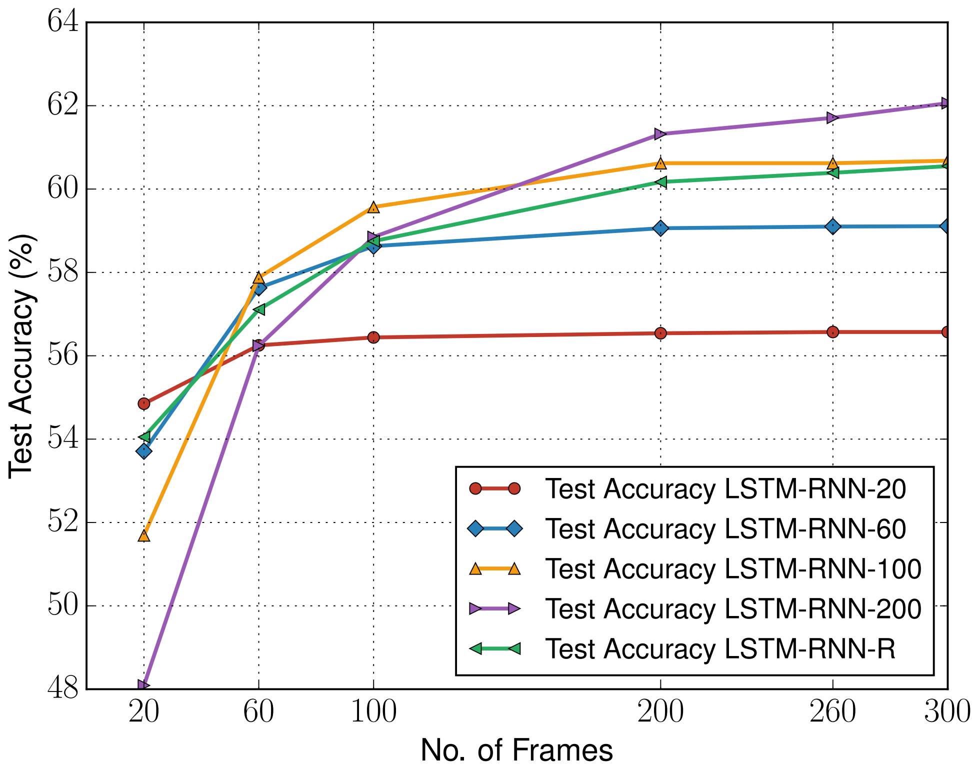

Figure 3.5: Speech emotion recognition test accuracy on the IEMOCAP dataset of a Long Short-Term Memory (LSTM)-Recurrent Neural Network (RNN) with various number of context frames. LSTM-RNN-c denotes the sequence length which the model was trained on. The number of frames denotes the sequence length which the model was evaluated on.

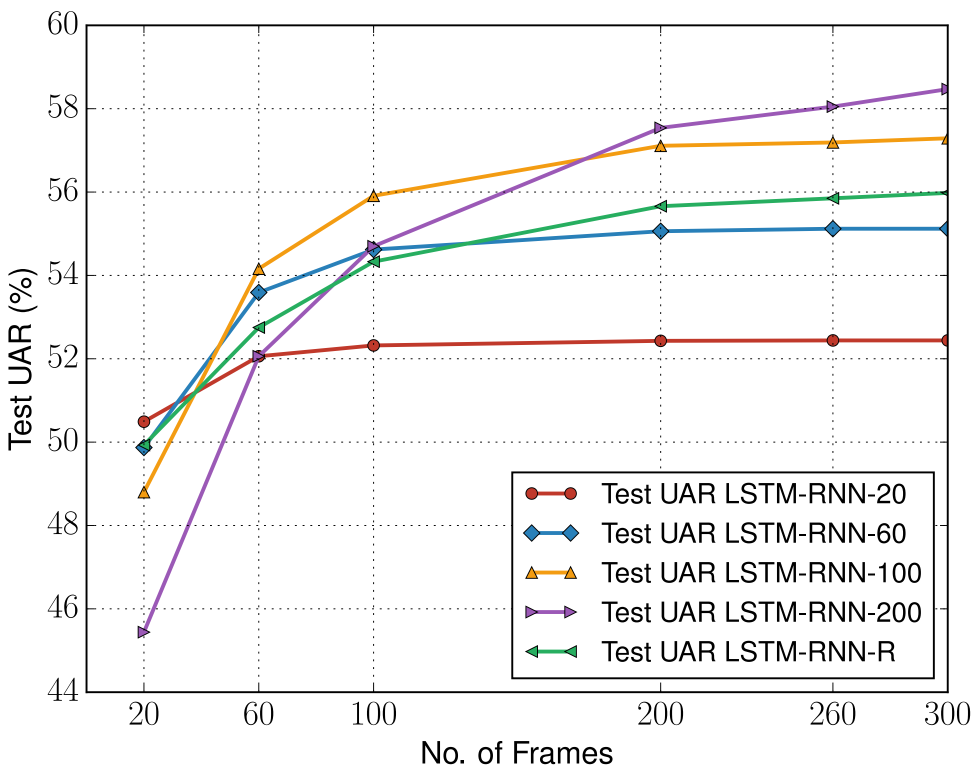

Figure 3.6: Speech emotion recognition test Unweighted Average Recall (UAR) on the IEMOCAP dataset of a Long Short-Term Memory (LSTM)-Recurrent Neural Network (RNN) with various number of context frames. LSTM-RNN-c denotes the sequence length which the model was trained on. The number of frames denotes the sequence length which the model was evaluated on.

Table 3.8: Speech emotion recognition test Accuracy (ACC) and Unweighted Average Recall (UAR) on the IEMOCAP dataset with various neural network architectures. Conv($c \times j \times k$) denote a spatial convolutional layer of c filters, each of size $j \times k$, with stride 2, followed by Batch Normalization (BatchNorm) and Rectified Linear Units (ReLUs). FC(n_l) denotes a fully connected layer of n_l units followed by BatchNorm, ReLUs, and dropout. LSTM-RNN(n_l) denotes a Long Short-Term Memory (LSTM)-Recurrent Neural Network (RNN) of n_l units. All architectures have a fully connected layer with a softmax function as the output layer.

Model	**Test ACC (%)**	**Test UAR (%)**
FC(1024)×5	62.55	58.78
Conv(16 × 10 × 10) — Conv(32 × 10 × 10) — FC(716)×3	**64.78**	**60.89**
LSTM-RNN(256)×2 — FC(256)	61.71	58.05

subsets and experiment conditions. As stated earlier, SER can be regarded as a static or dynamic classification problem, which makes the task an excellent test bed for conducting a comparison between these architectures. In this case, the DNN and the ConvNet can be regarded in this formulation as static classifiers that process a number of concatenated frames jointly to predict a class label, whereas the LSTM-RNN can be regarded in this formulation as a dynamic classifier that processes a sequence of frames, frame-by-frame, to predict a class label. The results in Table 3.8 suggest that the static component in speech is more discriminative for SER than the dynamic component. This is likely due to the dominant presence of the linguistic aspect in the dynamic component of speech, which can hinder the recognition of paralinguistic components such as emotions. It is speculated that for this reason, in addition to the ability of ConvNets to learn discriminative features invariant to small variations, the ConvNet yielded the best test accuracy and test UAR, followed by the DNN, then the LSTM-RNN.

Figure 3.7 illustrates the output of the proposed SER system using a ConvNet for a number of selected utterances from the test subset of the IEMOCAP dataset. Qualitative assessment of the system's output indicates that the system has learned to model the intra-utterance emotion dynamics with high confidence and is able to transition smoothly from one class to another, capturing brief pauses and mixed emotions as shown in Figure 3.7. It is particularly interesting to note the system's output in Figure 3.7(e), which has classified the first half of the utterance as *neutral* and the second half of the utterance as *happy*, conforming to the manual inspection, whereas the dataset annotators assigned the *happy* label to the entire utterance.

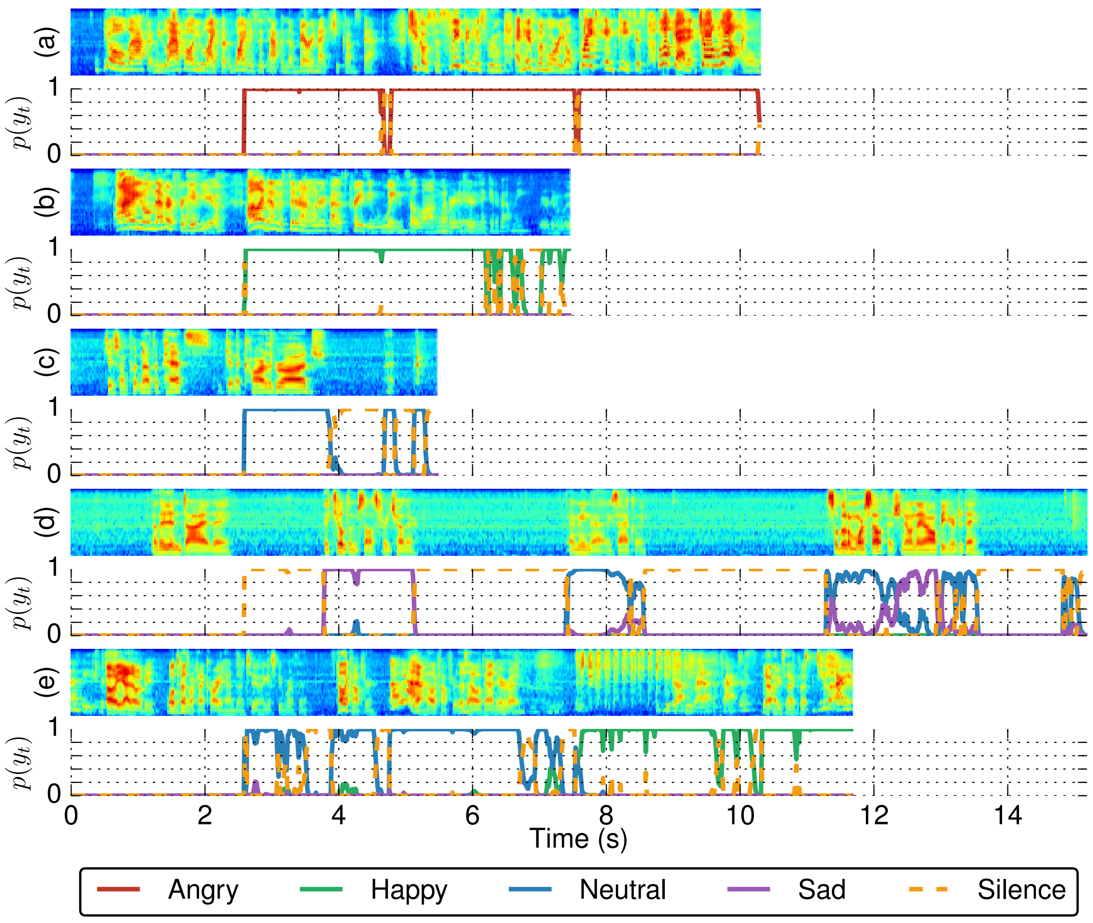

Figure 3.7: Input speech utterances (top) and corresponding aligned output (below) of the speech emotion recognition system for a number of selected utterances from the test subset of the IEMOCAP dataset. The output is the probabilities over classes denoting the confidence of the model. Transcripts: **(a)**: Oh, laugh at me all you like but why does this happen every night she comes back? She goes to sleep in his room and his memorial breaks in pieces. Look at it, Joe look. (Angry); **(b)**: I will never forgive you. All I'd done was sit around wondering if I was crazy waiting so long, wondering if you were thinking about me. (Happy); **(c)**: OKay. So I am putting out the pets, getting the car our the garage. (Neutral); **(d)**: They didn't die. They killed themselves for each other. I mean that, exactly. Just a little more selfish and they would all be here today. (Sad); **(e)**: Oh yeah, that would be. Well, depends on what type of car you had, though too. I guess it would be worth it. helicopter. Yeah, helicopter. There is a helipad there, right? Yeah, exactly. (Happy).

Table 3.9: Speech Emotion Recognition (SER) results reported in prior work on the IEMOCAP dataset. Note that differences in data subsets and other experiment conditions should be taken into consideration when comparing the following results against each other, see references for more details.

Method	**Test ACC (%)**	**Test UAR (%)**
DNN + ELM [Han et al., 2014]	54.3	48.2
SVM [Mariooryad and Busso, 2013][1]	53.99	50.64
Replicated Softmax + SVM [Shah et al., 2014][1]	—	57.39
Hierarchical Binary Decision Tree [Lee et al., 2011][2]	—	58.46
SVM [Poria et al., 2017][1]	61.32	—
Data Augmentation + ConvNet [Etienne et al., 2018]	64.5	61.7
Proposed SER System (Utterance-Based)	**57.74**	**58.28**
Proposed SER System (Frame-Based)	**64.78**	**60.89**

[1] Better performance was reported by incorporating other modalities.
[2] Speaker-dependent normalization.

To facilitate comparison of the results reported with prior literature, which mostly relies on utterance-based classification, the class probabilities computed for each frame in an utterance were averaged across all frames in that utterance and an utterance-based label was selected based on the maximum average class probabilities, ignoring the *silence* label, as per Equation (3.2):

$$\check{y} = \underset{k=1,\ldots,K}{\operatorname{argmax}} \frac{\sum_{t=1}^{T} p(\mathbf{y}_t|\mathbf{x}_t)}{T} \tag{3.2}$$

where $\check{y}$ is the utterance-level label, and $K = 4$ is the number of output classes, i.e., ignoring the *silence* class.

Table 3.9 shows the SER results reported in prior work on the IEMOCAP dataset. Note that differences in data subsets used and other experiment conditions should be taken into consideration when comparing these results against each other (see [Han et al., 2014, Mariooryad and Busso, 2013, Shah et al., 2014, Lee et al., 2011, Poria et al., 2017, Etienne et al., 2018] for more details). As shown in Table 3.9, the proposed SER system outperforms all other prior speaker-independent methods. In addition, the proposed SER system offers other advantages, e.g., real-time output, since it does not depend on future context. Moreover, the system is able to deal with utterances of arbitrary length with no degradation in performance. Furthermore, the system can handle utterances that contain more than one emotion, as demonstrated in Figure 3.7(e), which is not possible in an utterance-based processing formulation.

3.5 Discussion

The benefit of depth in deep learning architectures was observed across all three tasks. In the image recognition task, DenseNets Model **B**, which had approximately 40 convolutional or fully connected layers, outperformed ConvNets Model **A**, which had only 6 convolutional or fully connected layers, as demonstrated in Table 3.3. The same observation was also noted in the ASR task as demonstrated in Table 3.6. In the SER task, deeper ConvNets outperformed shallower networks, as shown in Table 3.7, albeit the improvement was not as pronounced as the other two tasks.

The machine perception tasks, namely, image recognition, ASR, and SER, studied in Sections 3.2, 3.3 and 3.4 respectively, are somewhat diverse. Prior to the application of deep learning to all three tasks, various methods and techniques were employed for each task. Image recognition relied on engineered geometric features, e.g., HOGs features and SIFT features, and standard classifiers, e.g., SVMs. ASR relied on engineered features that reportedly discard inter-speaker variability and maintain features of relevance to ASR in speech, e.g., MFCCs, and GMM-HMM systems. SER relied on high-dimensional acoustic feature sets and standard classifiers. The deep learning approach is superior to all of the above as demonstrated in [Krizhevsky et al., 2012, Mohamed et al., 2012] and Table 3.9.

The ConvNets employed, described in Tables 3.1, 3.2, 3.4, 3.5 and 3.7, with fairly consistent training and implementation frameworks, performed well across all three tasks. This is notable given the differences between the tasks and datasets, and motivates the pursuit of multi-task and continual learning systems.

3.6 Summary

The concepts, theory, and mathematical framework presented in Chapter 2 were applied to supervised learning tasks in this chapter, and in particular, classification tasks in machine perception, namely, image recognition, Automatic Speech Recognition (ASR), and Speech Emotion Recognition (SER). The work presented herein developed understanding, insights, and baselines for the three tasks, as well as their respective datasets, deep learning architectures, and best practices. Results reported for the image recognition task on the CIFAR-10, CIFAR-100, and SVHN datasets, as well as the results reported for the ASR task on the TIMIT dataset, were on par with the state-of-the-art.

The SER task was explored in more depth and used as a test bed to explore various neural network architectures. The task is an excellent test bed for exploring various deep learning architectures since the task itself can be formulated in multiple ways. As a result of this novel exploration, state-of-the-art results were reported on the IEMOCAP dataset for speaker-independent SER[§]. Significantly, the results obtained attested to the viability of the frame-based processing formulation to SER compared with the turn-based processing formulation, which not only facilitates the integration of SER into other speech recognition tasks, e.g., ASR, but also provides real-time SER capability and successfully handles long utterance with multiple emotions.

The relation between the systems developed in this chapter will be investigated in Chapter 4. These independent systems will be used as baselines and unified into continual learning systems in Chapter 5.

[§]Compared to prior literature when the work was published in [Fayek et al., 2017].

Chapter 4

On the Relevance of Features and Task Relatedness in Deep Networks

PRIOR to formulating a system that aims to address multiple tasks, the relation between these tasks ought to be understood. Understanding task relatedness is envisaged to be valuable in designing and implementing systems that aim to address multiple tasks, by taking into consideration the overlap between closely related tasks as well as the interference between unrelated or adversary tasks. Herein, a methodology for understanding the relation between two tasks is proposed using the features learned for each task in a deep network, where the relevance of each layer of features in a network trained for one task to the other task is illuminated via transfer learning. The gradual transfer learning methodology is explored on a number of datasets and tasks in the image recognition and speech recognition domains that were developed in the previous chapter. The work presented in this chapter is based on [Fayek, 2016, Fayek et al., 2016b, Fayek et al., 2018].

Outline. This chapter is structured as follows. Section 4.1 reviews related prior work on this topic. Section 4.2 introduces the gradual transfer learning methodology. Section 4.3 presents experiments and results in the image recognition domain using the CIFAR-10, CIFAR-100, and SVHN datasets. Section 4.4 presents experiments and results in the speech recognition domain using the TIMIT and IEMOCAP datasets. Section 4.5 provides a discussion on the methodology and common findings. Finally, Section 4.6 summarizes the chapter.

4.1 Background

Transfer learning aims to incorporate the learned knowledge from one task, denoted *source task* or *base task*, into another task, denoted *target task*, to aid the learning or performance, or both, of the target task [Caruana, 1997, Bengio, 2012]. Incorporating knowledge from the source task into the target task is achieved by initializing the parameters of a model for the target task, denoted *target model*, using parameters learned in a model for the source task, denoted *source model* or *base model*, followed by fine-tuning some or all of those parameters on the target task. Transfer learning was shown to lead to faster learning, or better performance, or both, compared with independent learning, i.e., learning from a tabula rasa, in many cases when both tasks are related, particularly in cases where the target task lacks sufficient data [Pan and Yang, 2010, Razavian et al., 2014]. The impact of transfer learning is dependent on: the nature of both tasks; the relation between the source task and target task; the amount of data available in both tasks; as well as the architecture of the models and learning algorithm used.

Transfer learning has been extensively utilized across a variety of settings, domains, and applications. For example, in computer vision, transfer learning is often used as an alternative to random initialization or to boost the performance of tasks that lack sufficiently large datasets using another similar task that has a large dataset. In [Razavian et al., 2014], learned features in a Convolutional Neural Network (ConvNet) for an object classification task were used for scene recognition, fine-grained recognition, attribute detection, and image retrieval tasks, using various datasets. The learned features in the object classification task were used as an image representation for these tasks and were found to outperform state-of-the-art models across almost all datasets and tasks that were trained independently. In [Tajbakhsh et al., 2016], a ConvNet, trained using the ImageNet dataset [Russakovsky et al., 2015] that comprises natural images (see Appendix A.4 for more details), was fine-tuned for a number of tasks, e.g., classification, detection, and segmentation, in radiology, cardiology, and gastroenterology. This model outperformed the same models trained independently as well as models trained using hand-crafted features. In Automatic Speech Recognition (ASR), transfer learning is often used across datasets and languages [Wang and Zheng, 2015], e.g., in [Swietojanski et al., 2012], multiple languages were used to pre-train Deep Neural Networks (DNNs), and the trained models were fine-tuned on a target language, demonstrating better results than the same DNNs trained directly on the target language. Transfer learning has also been used for speaker adaptation, as in [Li and Sim, 2010, Yao et al., 2012]. Transfer learning has been employed in Speech Emotion Recognition (SER), as in [Deng et al., 2013], where transfer learning between datasets was studied, and was shown to lead to a boost in performance relative to independent learning.

Deep networks tend to learn low-level features in initial layers and transition to high-level features towards final layers [Zeiler and Fergus, 2014]. Similar low-level features commonly appear across various datasets and tasks, while high-level features are somewhat more attuned to the dataset or task at hand; this makes low-level features more general and easier to transfer from one dataset or task to another [Yosinski et al., 2014]. In many situations, especially when data in the target task is scarce, the transfer of low-level features from one dataset or task to another, followed by learning high-level features, is likely to lead to a boost in performance given that both datasets or tasks share some similarity [Razavian et al., 2014, Wang and Zheng, 2015]. Conversely, transferring high-level features and learning low-level ones can be regarded as a form of domain adaptation and can be useful when the tasks are similar or identical but the data distributions are slightly different [Glorot et al., 2011b, Bengio, 2012].

Layer-wise transferability in deep networks was studied in [Yosinski et al., 2014]. Therein, the transferability of features in a ConvNet for a computer vision task was experimentally studied, where the generality versus specificity of each layer in the network was quantified using curated classes from the ImageNet dataset. It was shown that initial layers in deep networks are more transferable than final layers. It was also shown that a correlation between the benefit of feature transfer and the relatedness between the source task and the target task exists, such that the improvement due to feature transfer diminishes as the relatedness between both tasks decreases. A similar study was carried out in [Misra et al., 2016], reporting similar findings.

4.2 Gradual Transfer Learning

The main premise put forward in this chapter is that the layer-wise transferability of representations in deep networks between two tasks can be used to understand the relation between these tasks. The methodology to quantify this transferability, denoted *gradual transfer learning*, is as follows, which is also summarised in Figure 4.1. First, two primary neural network models that comprise L layers are trained for each dataset or task independently. Second, for each of the two primary models, the learned parameters in all layers of the trained model, except the output layer, are copied to a new model for the (other) secondary dataset or task; the output layer can be randomly initialized, since it is closely tied to the dataset or task at hand, e.g., the number of output classes in both datasets or tasks may be different. Third, the first $l_c \in \{0, \ldots, L_H\}$ layers are held constant and the remaining layers are fine-tuned for the secondary dataset or task, where L_H is the number of hidden layers in the model, i.e., $L_H = L - 1$. If the constant transferred layers l_c are relevant to the secondary dataset or task, one can expect

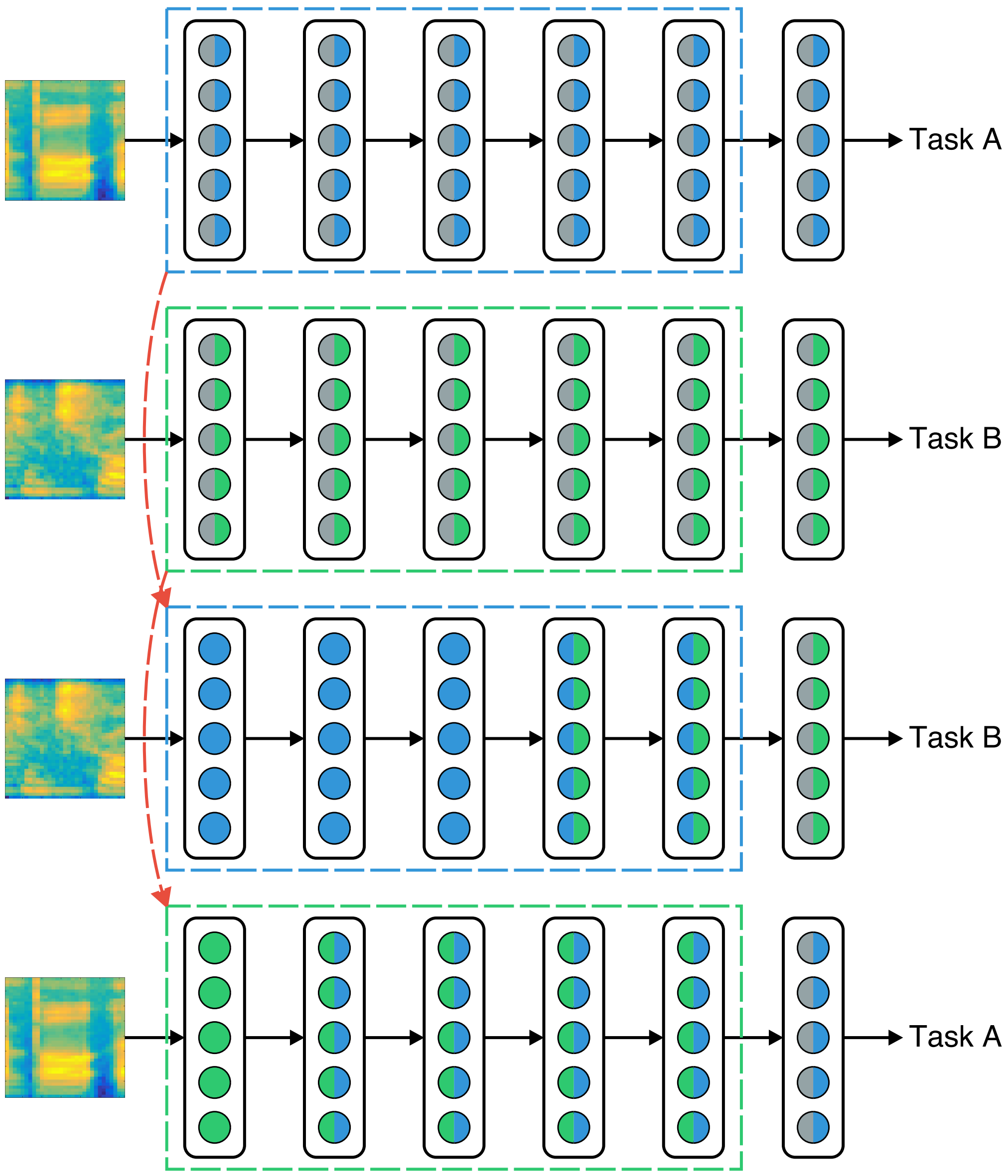

Figure 4.1: Gradual transfer learning between two tasks. The parameters of the first and second models are initialized randomly (grey) and trained for Task A (blue) and Task B (green) respectively. The third and fourth models are examples of gradual transfer learning. The parameters of the third model are initialized using the trained Task A model (blue) and the final three layers are fine-tuned for Task B (green). The parameters of the fourth model are initialized using the trained Task B model (green) and all layers except the first layer are fine-tuned for Task A (blue).

an insignificant or no drop in performance relative to the primary model trained independently, and vice versa. By iteratively varying the number of constant layers l_c, the layer-wise transferability of representations learned for each dataset or task to the other can be inferred.

Iterating l_c through $\{0, \ldots, L_H\}$ yields a number of special cases as follows. In the case of $l_c = L_H$, the primary model can be regarded as a feature extractor to the secondary model, in that the output layer is the only layer to be fine-tuned. In the case of $1 \leq l_c < L_H$, the output layer is first fine-tuned for a small number of iterations to avoid back-propagating gradients from randomly initialized parameters to previous layers when the output layer is randomly initialized, and subsequently the final $(L - l_c)$ layers (including the output layer) are fine-tuned simultaneously. In the case of $l_c = 0$, the output layer is first fine-tuned for a small number of iterations, and then all layers of the model are fine-tuned simultaneously with the output layer; in this case, the primary model can be regarded as only an initialization to the secondary model.

4.3 Experiments in Image Recognition

The transferability and relevance of learned features in deep ConvNets between image recognition tasks using the CIFAR-10, CIFAR-100, and SVHN datasets are investigated using the gradual transfer learning methodology outlined in Section 4.1. Section 4.3.1 details the experimental setup, which follows the setup detailed in Section 3.2, and experiments carried out. Section 4.3.2 presents the results and observations.

4.3.1 Experimental Setup

Datasets. The CIFAR-10 and CIFAR-100 datasets [Krizhevsky, 2009] (see Appendix A.1 for more details) as well as the SVHN dataset [Netzer et al., 2011] (see Appendix A.5 for more details) were used in this experiment.

The training and test sets of each of the CIFAR datasets contain 50000 and 10000 images respectively. For each CIFAR dataset, the original training set was split into a training set of 45000 images and a validation set of 5000 images; the entire test set was used for testing.

The training and test sets of the SVHN dataset contain 73257 and 26032 images respectively, and additionally, 531131 images are available that can be appended to the training set. The original training set and additional set were combined and split into a training set of 598388 images and a validation set of 6000 images. The entire test set was used for testing.

The training, validation, and test sets of all datasets are mutually exclusive.

Table 4.1: Densely connected convolutional network architecture for image recognition. The outputs of the convolutional layers in Blocks 2, 4, and 6, are concatenated with the inputs to the layer and fed to the subsequent layer in the same block. K denotes the number of output classes.

Block	**Repeat**	**Type**	**Size**	**Other**
1	1×	Convolution	$24, 3 \times 3$	Stride = 1
2	12×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$12, 3 \times 3$	Stride = 1
		Dropout	—	$r = 0.2$
3	1×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$168, 1 \times 1$	Stride = 1
		Dropout	—	$r = 0.2$
		Average Pooling	2×2	Stride = 2
4	12×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$12, 3 \times 3$	Stride = 1
		Dropout	—	$r = 0.2$
5	1×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$312, 1 \times 1$	Stride = 1
		Dropout	—	$r = 0.2$
		Average Pooling	2×2	Stride = 2
6	12×	BatchNorm	—	—
		ReLU	—	—
		Convolution	$12, 3 \times 3$	Stride = 1
		Dropout	—	$r = 0.2$
7	1×	BatchNorm	—	—
		ReLU	—	—
		Average Pooling	8×8	Stride = 8
8	1×	Fully Connected	K	—
		Softmax	—	—

Preprocessing. Standard data preprocessing steps were applied for all datasets as commonly practised in the literature [Sermanet et al., 2012, Goodfellow et al., 2013, Long et al., 2015]. The mean and standard deviation of the images in the CIFAR datasets were normalized to zero and one respectively per colour channel using the training set statistics for each dataset individually. The images in the SVHN dataset were scaled via division by 255 to lie in the $[0, 1]$ range. No data augmentation was used for all datasets.

Model. The model used in this experiment follows the Densely Connected Convolutional Network (DenseNet) architecture [Huang et al., 2017]. It was shown to achieve state-of-the-art performance on the datasets used in this experiment [Huang et al., 2017]. The DenseNet architecture used is detailed in Table 4.1. The main layers in the architecture can be grouped into blocks based on their type and role. The dense blocks, Blocks 2, 4, and 6 in Table 4.1, comprise 12 layers of Batch Normalization (BatchNorm), Rectified Linear Units (ReLUs), convolution, and dropout. Each convolutional layer in Blocks 2, 4, and 6 in Table 4.1 is connected to all subsequent layers in the same block via the concatenation operation. The transition blocks, Blocks 3 and 5 in Table 4.1, are used to counteract the growth in the number of parameters due to the use of the concatenation operation, and are composed of a layer of BatchNorm, ReLUs, convolution, dropout, and average pooling. A down-sampling block, Block 7 in Table 4.1, is used to further reduce the complexity of the model, and is composed of BatchNorm, ReLUs, and average pooling. The output layer is a fully connected layer followed by a softmax function.

Training and Implementation Details. The parameters of the convolutional and fully connected layers were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers respectively, where n_i is the number of inputs to the layer, as recommended in [He et al., 2015]. Convolutional and fully connected layers were regularized using weight decay with penalty $\lambda = 1 \times 10^{-4}$ and dropout with drop probability $r = 0.2$. Mini-batch Stochastic Gradient Descent (SGD) was used to optimize the parameters with respect to a negative log-likelihood loss function using batch size $M_b = 64$, learning rate $\alpha = 0.1$, and Nesterov momentum $\eta = 0.9$, for 300 and 40 epochs for the CIFAR and SVHN datasets respectively. The initial learning rate α was divided by 10 at 50% and 75% of the total number of training epochs. No early stopping was used in the case of the CIFAR datasets, whereas the validation set was used for early stopping in the case of the SVHN dataset. These settings follow [Huang et al., 2017] and are identical to the settings described in Section 3.2.

The experiments were implemented using TensorFlow [Abadi et al., 2016].

Training was carried out on NVIDIA Tesla K80 Graphics Processing Units (GPUs). Training a single model from scratch on a single GPU required a wall time of approximately 30 and 50 hours for the CIFAR and SVHN datasets respectively.

Gradual Transfer Learning. Three base models were trained independently for the CIFAR-10, CIFAR-100, and SVHN datasets. Gradual transfer learning was used to assess the layer-wise feature relevance for each dataset to the other two. The architecture of the models comprised 40 layers in total as detailed in Table 4.1, and thus the number of fixed layers was varied in block intervals as opposed to single layer intervals, i.e., $l_c \in \{0, Block\ 1, Blocks\ 2\ and\ 3, Blocks\ 4\ and\ 5, Blocks\ 6\ and\ 7\}$.

4.3.2 Results

The baseline models trained independently on each dataset achieved a validation and test classification accuracy of respectively 100% and 92.91% on the CIFAR-10 dataset, 99.84% and 70.66% on the CIFAR-100 dataset, and 97.48% and 98.15% on the SVHN dataset. The baseline results are also plotted in Figures 4.2, 4.3 and 4.4. It is important to take into consideration that the number of images and classes vary across the datasets when comparing the results with each other.

The results of gradual transfer learning between the CIFAR-10 and CIFAR-100 datasets are plotted in Figure 4.2. Both datasets are somewhat similar as both datasets contain approximately the same categories of classes. The results plotted in Figure 4.2 reflect this similarity. The difference between the baseline models and the models that contain constant layers in Blocks 1, 2, and 3 is small. The gap between the baseline models and the models that contain constant layers in Blocks 4, 5, 6, and 7 increases relative to preceding layers. This indicates that features learned in layers in Blocks 1, 2, and 3 are relevant for both tasks, and the relevance diminishes in layers in Blocks 4, 5, 6, and 7.

The results of gradual transfer learning between the CIFAR-10 and SVHN datasets are plotted in Figure 4.3. The results demonstrate an intriguing observation: features learned for the CIFAR-10 dataset are more relevant to the SVHN dataset than features learned for the SVHN dataset are to the CIFAR-10 dataset, except for layers in Blocks 6 and 7, which were somewhat not relevant in both cases.

The results of gradual transfer learning between the CIFAR-100 and SVHN datasets are plotted in Figure 4.4. A similar observation to the observation in Figure 4.3 is noted here; features learned for the CIFAR-100 dataset are more relevant to the SVHN dataset than features learned for the SVHN dataset are to the CIFAR-100 dataset, except for layers in Blocks 6 and 7, which were somewhat not relevant in both cases.

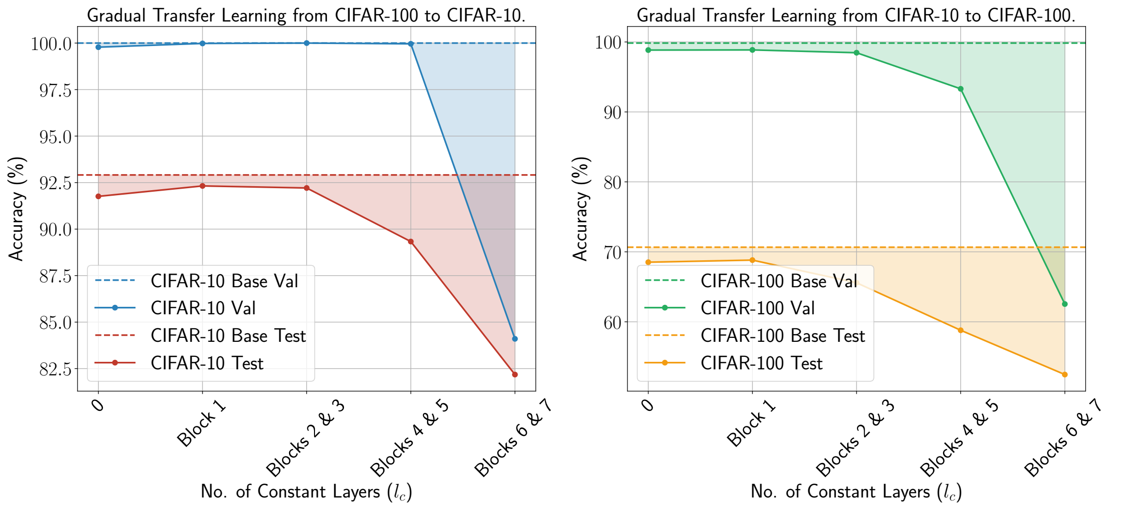

Figure 4.2: Gradual transfer learning between the CIFAR-10 and CIFAR-100 datasets. **Left.** The validation and test accuracies of independent CIFAR-10 (dashed) and CIFAR-10 fine-tuned from CIFAR-100 (solid) as a function of the number of constant layers. **Right.** The validation and test accuracies of independent CIFAR-100 (dashed) and CIFAR-100 fine-tuned from CIFAR-10 (solid) as a function of the number of constant layers.

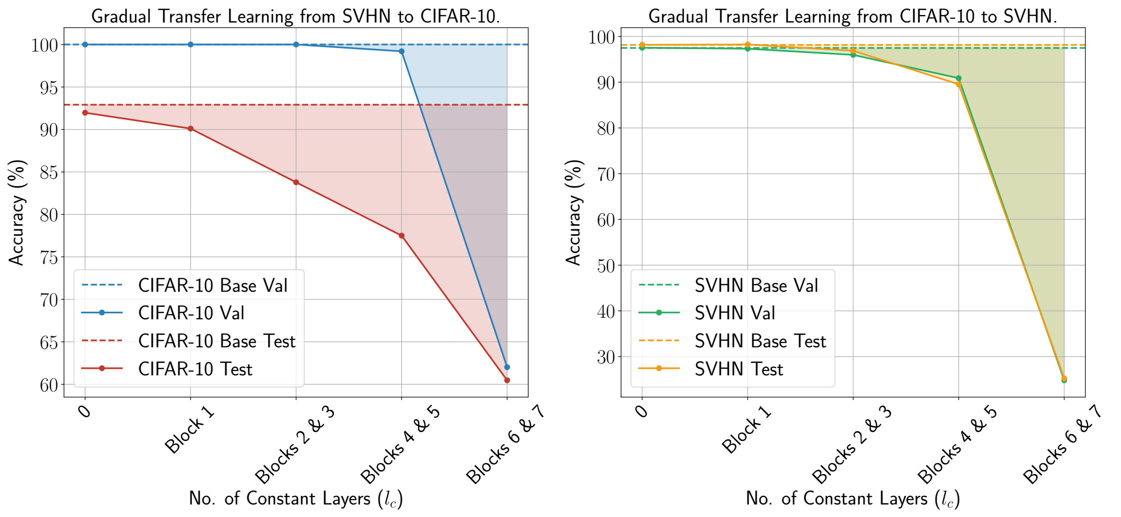

Figure 4.3: Gradual transfer learning between the CIFAR-10 and SVHN datasets. **Left.** The validation and test accuracies of independent CIFAR-10 (dashed) and CIFAR-10 fine-tuned from SVHN (solid) as a function of the number of constant layers. **Right.** The validation and test accuracies of independent SVHN (dashed) and SVHN fine-tuned from CIFAR-10 (solid) as a function of the number of constant layers.

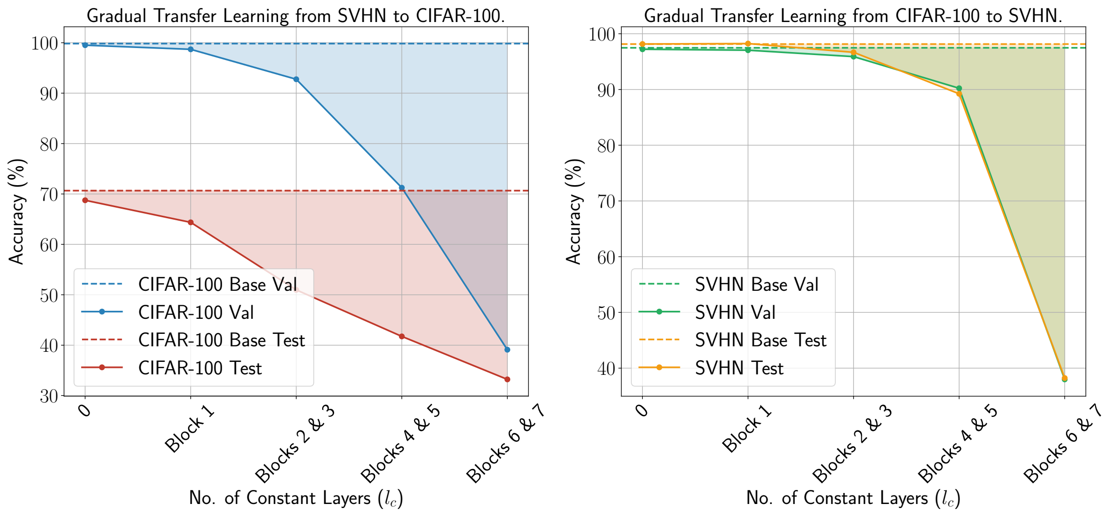

Figure 4.4: Gradual transfer learning between the CIFAR-100 and SVHN datasets. **Left.** The validation and test accuracies of independent CIFAR-100 (dashed) and CIFAR-100 fine-tuned from SVHN (solid) as a function of the number of constant layers. **Right.** The validation and test accuracies of independent SVHN (dashed) and SVHN fine-tuned from CIFAR-100 (solid) as a function of the number of constant layers.

Results plotted in Figures 4.2, 4.3 and 4.4 indicate that features learned for the CIFAR-10 and CIFAR-100 datasets are more relevant to each other compared with features learned for the SVHN dataset that were not as relevant to the CIFAR-10 and CIFAR-100 datasets, which reflects the relation between all three datasets.

4.4 Experiments in Speech Recognition

The transferability and relevance of learned features in deep ConvNets between ASR using the TIMIT dataset and SER using the IEMOCAP dataset are investigated via the gradual transfer learning methodology outlined in Section 4.1. Section 4.4.1 details the experimental setup, which follows the setup detailed in Section 3.3 and Section 3.4, and experiments carried out. Section 4.4.2 presents the results and observations.

4.4.1 Experimental Setup

Datasets. The TIMIT dataset [Garofolo et al., 1993] (see Appendix A.6 for more details) was used for the ASR task. The complete 462-speaker training set, without the dialect (SA) utterances, was used as the training set. The 50-speaker development set was used as the validation set. The 24-speaker core test set was used as the test set. As commonly practised in the literature [Lee and Hon, 1989], the 61 phonemes were mapped into 48 phonemes for training, which were then mapped into 39 phonemes for scoring.

The IEMOCAP dataset [Busso et al., 2008] (see Appendix A.3 for more details) was used for the SER task. Similar to the setup in Section 3.4, utterances that bore only the following four emotions: *anger*, *happiness*, *sadness*, and *neutral*, were included with *excitement* considered as *happiness*; amounting to a total of 5531 utterances. An eight-fold Leave-One-Speaker-Out (LOSO) cross-validation scheme was employed in all experiments using the eight speakers in the first four sessions. Both speakers in the fifth session were used as the validation set, and hence were not included in the cross-validation folds to avoid biasing the results [Refaeilzadeh et al., 2009].

Preprocessing. Utterances were split into 25 ms frames with a stride of 10 ms, and a Hamming window was applied, then 40 log-Mel Frequency Spectral Coefficients (MFSCs) were extracted from each frame. The mean and standard deviation were normalized per coefficient to zero and one respectively using the mean and standard deviation computed from the training set only in the case of ASR and from training subset in each fold in the case of SER. No speaker dependent operations were performed.

In ASR, the Kaldi toolkit [Povey et al., 2011] was used to produce force-aligned frame labels by training a mono-phone Gaussian Mixture Model (GMM)-Hidden Markov Model (HMM) system with Mel Frequency Cepstral Coefficients (MFCCs). In SER, the data was already labelled at an utterance level; hence, all frames in an utterance inherited the utterance label. A voice activity detector was subsequently used to label silent frames, and *silence* was added as an additional class to the four previously mentioned emotion classes, i.e., a frame had either the same label as its parent utterance or the *silence* label. This scheme was adopted to be consistent with the ASR task since *silence* is considered a class in the TIMIT dataset and contributes to the score. Moreover, the presence of silence and other disfluencies can be an effective cue in emotion recognition [Tian et al., 2015b].

Model. The ASR system had a hybrid ConvNet-HMM architecture, as described in Section 3.3. A ConvNet acoustic model was used to produce a probability distribution over the states of three-state HMMs with a bi-gram language model estimated from the training set. The SER system comprised only a ConvNet acoustic model identical to the model used in ASR.

Acoustic models in ASR typically utilize a smaller window of frames of features computed from speech compared with acoustic models in ASR as shown in Figure 3.4.

The number of frames was determined empirically using the validation sets in both tasks and it was found that 31–41 frames was a good trade-off between both tasks. Temporal derivatives were not appended to the input as commonly done in ASR since it was observed that appending temporal derivatives degraded the accuracy of SER.

Two ConvNet architectures were used to isolate architecture-specific behaviour and trends. The first architecture is detailed in Table 4.2 and denoted Model **A**. The model is a standard ConvNet that comprises two convolutional and max pooling layers, followed by four fully connected layers, with BatchNorm and ReLUs interspersed in-between. The final fully connected layer is followed by a softmax function to predict the probability distribution over 144 classes in the case of ASR, i.e., three HMM states per 48 phonemes, or 5 classes in the case of SER. The second architecture is detailed in Table 4.3 and denoted Model **B**. The architecture is a variant of the popular VGGNet architecture [Simonyan and Zisserman, 2014, Sercu et al., 2016]. The architecture comprises a number of convolutional, BatchNorm, and ReLUs layers, with a few max pooling layers used throughout the architecture as indicated in Table 4.3, followed by three fully connected layers, with BatchNorm, ReLUs, and dropout interspersed in-between. The final fully connected layer is followed by a softmax function to predict the probability distribution over 144 classes in the case of ASR, i.e., three HMM states per 48 phonemes, or 5 classes in the case of SER.

Training and Implementation Details. The training and implementation details of each architecture differed slightly following the recommenced practice for each model in prior literature. The hyperparameters reported below were tuned using the validation set and are similar to the settings described in Section 3.3.

The parameters of the convolutional and fully connected layers of Model **A** were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers respectively, where n_i is the number of inputs to the layer, as recommended in [He et al., 2015]. Convolutional layers were regularized using weight decay with penalty $\lambda = 1 \times 10^{-3}$ and fully connected layers were regularized using dropout with drop probability $r = 0.4$. RMSProp was used to optimize the parameters of Model **A** with respect to a negative log-likelihood loss function using batch size $M_b = 256$, learning rate $\alpha = 1 \times 10^{-3}$ and $\alpha = 1 \times 10^{-4}$ for ASR and SER respectively, and decay rate $\eta = 0.99$. The validation set was used for early stopping such that training halts if the validation error ceases to improve for three consecutive epochs.

The parameters of the convolutional and fully connected layers of Model **B** were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers

Table 4.2: Speech recognition convolutional neural network Model **A** architecture. K denotes the number of output classes.

№	**Type**	**Size**	**Other**
1	Convolution	$64, 5 \times 4$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride $= 2$
2	Convolution	$128, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride $= 2$
3	Fully Connected	1024	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.4$
4	Fully Connected	1024	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.4$
5	Fully Connected	1024	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.4$
6	Fully Connected	K	—
	Softmax	—	—

Table 4.3: Speech recognition convolutional neural network Model **B** architecture. K denotes the number of output classes.

№	**Repeat**	**Type**	**Size**	**Other**
1	1×	Convolution	$64, 6 \times 5$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Convolution	$64, 3 \times 3$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Max Pooling	2×2	Stride $= 2$
2	2×	Convolution	$128, 3 \times 3$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Max Pooling	2×2	Stride $= 2$
3	3×	Convolution	$256, 3 \times 3$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Max Pooling	2×2	Stride $= 2$
4	3×	Convolution	$256, 3 \times 3$	Stride $= 1$
		BatchNorm	—	—
		ReLU	—	—
	1×	Max Pooling	2×2	Stride $= 2$
5	1×	Fully Connected	1024	—
		BatchNorm	—	—
		ReLU	—	—
		Dropout	—	$r = 0.4$
6	1×	Fully Connected	1024	—
		BatchNorm	—	—
		ReLU	—	—
		Dropout	—	$r = 0.4$
7	1×	Fully Connected	K	—
		Softmax	—	—

respectively, where n_i is the number of inputs to the layer, similar to Model **A**. All layers in Model **B** were regularized using weight decay with penalty $\lambda = 1 \times 10^{-3}$ and fully connected layers were also regularized using dropout with drop probability $r = 0.4$. ADAM was used to optimize the parameters of Model **B** with respect to a negative log-likelihood loss function using batch size $M_b = 256$, learning rate $\alpha = 1 \times 10^{-3}$ and $\alpha = 1 \times 10^{-4}$ for ASR and SER respectively, first moment $\beta_1 = 0.99$, and second moment $\beta_2 = 0.999$ for 110×10^3 iterations.

The experiments were implemented using MatDL [Fayek, 2017] and TensorFlow [Abadi et al., 2016]. Training was carried out using NVIDIA Tesla K40 and K80 GPUs. Training a single model from scratch on a single GPU required a wall time of approximately 15–48 hours depending on the architecture and GPU used.

Gradual Transfer Learning. Two base models were trained independently for the ASR and SER tasks. Gradual transfer learning was used to assess the layer-wise feature relevance between both tasks.

4.4.2 Results

The Frame Error Rate (FER) and Phone Error Rate (PER) are reported for the ASR task. The Error (E) and Unweighted Error (UE) (1− Unweighted Average Recall (UAR)) are reported for the SER task to reflect imbalanced classes [Schuller et al., 2009a]. These metrics are the average of the eight-fold LOSO cross-validation scheme, except for the ASR base model since the data split was predefined.

The base ASR Model **A** achieved an FER of 30.53% and 31.61% and a PER of 18.71% and 20.18% on the validation and test sets respectively. The base ASR Model **B** achieved an FER of 29.35% and 30.69% and a PER of 17.67% and 19.4% on the validation and test sets respectively. The base SER Model **A** achieved an error of 44.63% and 46.44% and an unweighted error of 46.34% and 48.96% on the validation and test sets respectively. The base SER Model **B** achieved an error of 46.89% and 48.47% and an unweighted error of 48.57% and 50.40% on the validation and test sets respectively.

The learned features in the first layer of the ConvNet Model **A** for both tasks are visualized in Figure 4.5. It is evident that features in ASR are smoother and more attuned to detect rapid variations in the input. However, learned features in SER did not demonstrate a similar pattern; this can be attributed to the longer period in which emotions manifest in speech.

The results of gradual transfer learning between ASR and SER using Model **A** outlined in Table 4.2 are listed in Tables 4.4 and 4.5 and plotted in Figure 4.6. The results of gradual transfer learning between ASR and SER using Model **B** outlined in Table 4.3 are listed in Tables 4.6 and 4.7 and plotted in Figure 4.7.

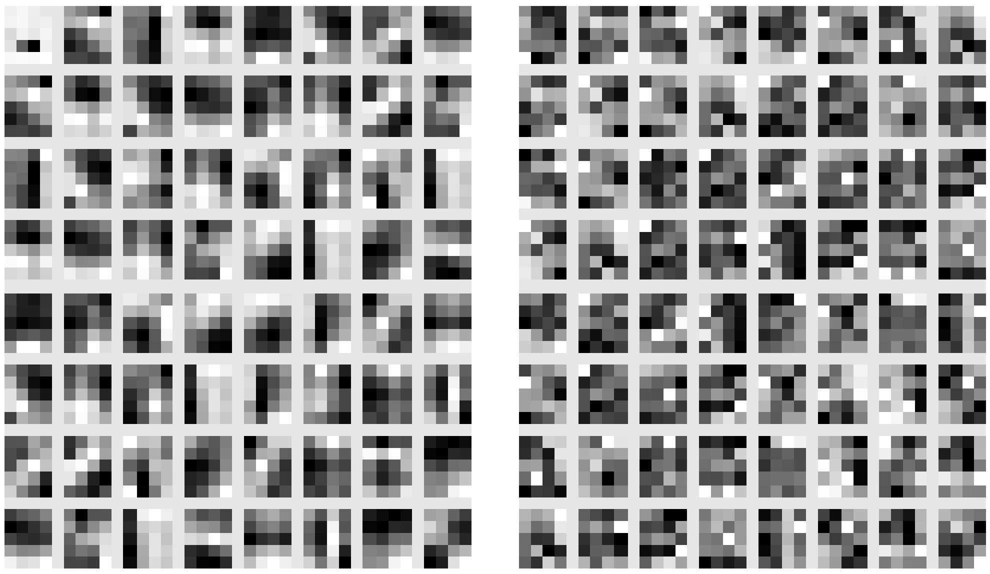

Figure 4.5: Learned features in the first layer of the convolutional neural network Model **A** detailed in Table 4.3 for the automatic speech recognition task (left) and the speech emotion recognition task (right).

Table 4.4: Validation and test Frame Error Rate (FER) and Phone Error Rate (PER) of gradual transfer learning from the speech emotion recognition task (IEMOCAP) to the automatic speech recognition task (TIMIT) with convolutional neural network Model **A**.

№ of Constant	**FER**		**PER**	
Layers (l_c)	**Validation**	**Test**	**Validation**	**Test**
Baseline	30.53%	31.61%	18.71%	20.18%
5	71.09%	71.64%	61.15%	61.82%
4	53.26%	53.92%	42.96%	44.13%
3	40.29%	40.97%	28.81%	30.48%
2	31.75%	32.87%	20.08%	21.85%
1	30.83%	32.01%	18.99%	20.94%
0	30.62%	31.65%	18.73%	20.57%

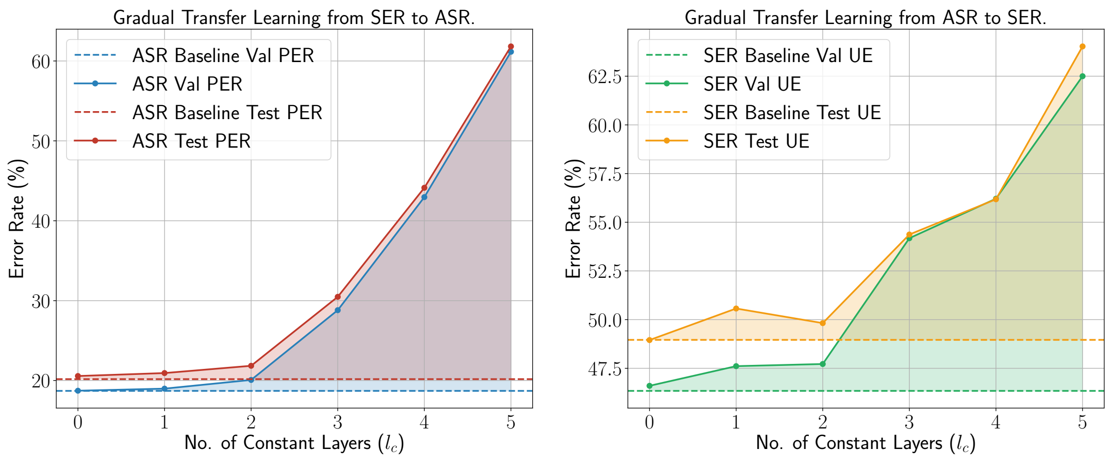

Figure 4.6: Gradual transfer learning between the automatic speech recognition task and the speech emotion recognition task with convolutional neural network Model **A**. **Left.** The Phone Error Rate (PER) of independent automatic speech recognition (TIMIT) (dashed) and automatic speech recognition (TIMIT) fine-tuned from speech emotion recognition (IEMOCAP) (solid) as a function of the number of constant layers. **Right.** The Unweighted Error (UE) of independent speech emotion recognition (IEMOCAP) (dashed) and speech emotion recognition (IEMOCAP) fine-tuned from automatic speech recognition (TIMIT) (solid) as a function of the number of constant layers.

Table 4.5: Validation and test error and unweighted error of gradual transfer learning from the automatic speech recognition task (TIMIT) to the speech emotion recognition task (IEMOCAP) with convolutional neural network Model **A**.

№ of Constant	**Error**		**Unweighted Error**	
Layers (l_c)	**Validation**	**Test**	**Validation**	**Test**
Baseline	44.63%	46.44%	46.34%	48.96%
5	52.55%	59.20%	62.50%	64.03%
4	51.94%	53.34%	56.21%	56.18%
3	50.22%	52.01%	54.18%	54.37%
2	47.39%	48.50%	47.72%	49.82%
1	46.37%	48.36%	47.61%	50.57%
0	45.26%	46.97%	46.60%	48.95%

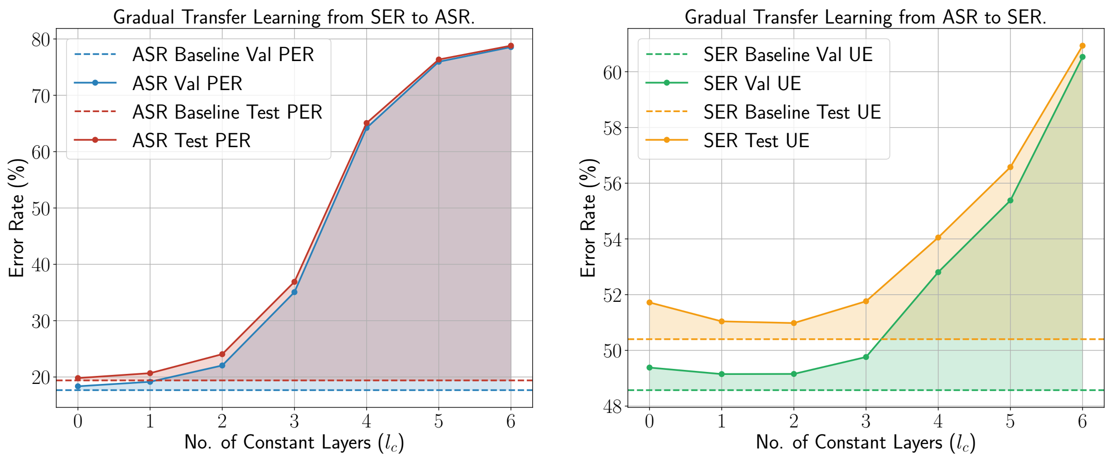

Figure 4.7: Gradual transfer learning between the automatic speech recognition task and the speech emotion recognition task with convolutional neural network Model **B**. **Left.** The Phone Error Rate (PER) of independent automatic speech recognition (TIMIT) (dashed) and automatic speech recognition (TIMIT) fine-tuned from speech emotion recognition (IEMOCAP) (solid) as a function of the number of constant layers. **Right.** The Unweighted Error (UE) of independent speech emotion recognition (IEMOCAP) (dashed) and speech emotion recognition (IEMOCAP) fine-tuned from automatic speech recognition (TIMIT) (solid) as a function of the number of constant layers.

Table 4.6: Validation and test Frame Error Rate (FER) and Phone Error Rate (PER) of gradual transfer learning from the speech emotion recognition task (IEMOCAP) to the automatic speech recognition task (TIMIT) with convolutional neural network Model **B**.

№ of Constant	**FER**		**PER**	
Layers (l_c)	**Validation**	**Test**	**Validation**	**Test**
Baseline	29.35%	30.69%	17.67%	19.4%
6	87.96%	87.69%	61.15%	78.56%
5	86.90%	86.73%	61.15%	75.96%
4	75.75%	76.05%	42.96%	64.22%
3	49.02%	50.10%	28.81%	35.07%
2	34.24%	35.65%	20.08%	22.05%
1	31.12%	32.14%	18.99%	19.16%
0	30.39%	31.39%	18.73%	18.36%

Table 4.7: Validation and test error and unweighted error of gradual transfer learning from the automatic speech recognition task (TIMIT) to the speech emotion recognition task (IEMOCAP) with convolutional neural network Model **B**.

№ of Constant	**Error**		**Unweighted Error**	
Layers (l_c)	**Validation**	**Test**	**Validation**	**Test**
Baseline	46.89%	48.47%	48.57%	50.40%
6	53.86%	57.15%	60.53%	60.94%
5	52.14%	54.22%	55.38%	56.58%
4	50.91%	52.19%	52.81%	54.05%
3	47.00%	49.23%	49.76%	51.76%
2	47.36%	48.28%	49.15%	50.98%
1	46.98%	48.86%	49.15%	51.04%
0	46.61%	49.11%	49.38%	51.72%

Several observations can be made from the results. Whether transfer learning from ASR to SER or from SER to ASR, the first two layers were relevant to the other task, i.e., could be transferred without fine-tuning and achieve performance comparable with the base models; the relevance of subsequent layers decreased gradually, i.e., transferring subsequent layers without fine-tuning led to a degradation in performance such that the extent of the degradation was positively correlated with the number of fixed layers in a quasi-linear manner. Both architectures demonstrated similar behaviour.

4.5 Discussion

The gradual transfer learning methodology was explored under a variety of settings and conditions as detailed in Sections 4.3 and 4.4. Consistent behaviour emerged that reflected the nature of the datasets or tasks involved.

Gradual Layer Specificity. Regardless of the architecture of the model, dataset, or task, the layer-wise feature specificity was positively correlated with the depth of the layer in the neural network, i.e., initial layers were more transferable than deeper layers. The transition in transferability between initial layers and deep layers was gradual, i.e., transferring initial learned layers without fine-tuning from one model trained for a dataset or task to another model for another dataset or task led to no or little degradation in performance, under the datasets or tasks studied herein, compared to the same model with no transferred layers, and the performance degraded gradually thereafter as we traversed deeper into the network towards final layers.

Non-symmetric Relevance. As shown in Figures 4.2, 4.3 and 4.4, the layer-wise relevance between two datasets or tasks can be non-symmetric This is shown in Figures 4.3 and 4.4, where features learned for the CIFAR-10 and CIFAR-100 datasets were found to be more relevant to the SVHN dataset than features learned for the SVHN dataset were found to be relevant to the CIFAR-10 and CIFAR-100 datasets.

Architecture Agnostic Behaviour. The architecture of the model is likely to influence the layer-wise relevance between datasets or tasks. However, under the datasets or tasks studied herein, it can be seen from Figures 4.6 and 4.7 that the architecture did not have a significant effect on the layer-wise relevance between datasets or tasks. Two different architectures, as described in Tables 4.2 and 4.3, were used to assess the layer-wise relevance between the ASR task and the SER task, yet similar trends were obtained as shown in Figures 4.6 and 4.7 respectively.

Layer-wise Relevance as an Indication to the Relation between Tasks. The main premise put forward in this chapter is that the layer-wise relevance of features in deep networks between two tasks can be used to understand the relation between these tasks. The true relation between the tasks studied in this work is unknown. Nevertheless, the CIFAR-10 and CIFAR-100 can be assumed to be more closely related to each other compared with their relation with the SVHN dataset, as the CIFAR-10 and CIFAR-100 datasets contain similar categories of classes. This is reflected in the results in Figures 4.2, 4.3 and 4.4, where features learned for the CIFAR-10 and CIFAR-100 datasets were more relevant to each other, compared with features learned for the SVHN dataset that were not as relevant to the CIFAR-10 and CIFAR-100 datasets.

4.6 Summary

A methodology for understanding the layer-wise relevance of features in deep networks between two tasks denoted gradual transfer learning was proposed, where the relevance of each layer of features in a network trained for one task to the other task is illuminated via transfer learning. The gradual transfer learning methodology was explored on a number of datasets and tasks in the image recognition and speech recognition domains. Within the datasets or tasks studied herein, consistent behaviour emerged that reflected the nature of the datasets or tasks involved such that the specificity of the features in deep networks was found to be positively correlated with the depth of the layer in the neural network. The gradual transfer learning methodology can be used for understanding the relation between two tasks, which can aid in designing and implementing systems that aim to address multiple tasks.

The findings and insights gained in this chapter will be used in the next chapter to build a continual learning framework that aims to learn and perform multiple tasks.

Chapter 5

Progressive Learning

Continual learning is the ability of a learning system to solve new tasks by utilizing previously acquired knowledge from learning and performing prior tasks without having significant adverse effects on the acquired prior knowledge. In this chapter, progressive learning is proposed, a deep learning framework that formulates continual learning into three procedures: curriculum, progression, and pruning. The curriculum procedure is used to actively select a task to learn from a set of candidate tasks. The progression procedure is used to grow the capacity of the model by adding new parameters that leverage parameters learned in prior tasks, while learning from data available for the new task at hand, without being susceptible to catastrophic forgetting. The pruning procedure is used to counteract the growth in the number of parameters as further tasks are learned, as well as to mitigate negative forward transfer, in which prior knowledge unrelated to the task at hand may interfere and worsen performance. Progressive learning is evaluated on a number of tasks in the image recognition and speech recognition domains that were studied in the previous chapters to demonstrate its advantages compared with baseline methods. It is shown that when tasks are related, progressive learning leads to faster learning that converges to better generalization performance using a smaller number of dedicated parameters.

Outline. This chapter is structured as follows. Section 5.1 outlines the motivations and advantages of progressive learning. Section 5.2 reviews related prior work. Section 5.3 describes progressive learning. Section 5.4 presents experiments and results in the image recognition domain using tasks derived from the CIFAR-10 and CIFAR-100 datasets. Section 5.5 presents experiments and results in the speech recognition domain using tasks derived from the TIMIT, IEMOCAP, and eNTERFACE datasets. Section 5.6 provides a discussion and outlines related future work. Finally, Section 5.7 summarizes the chapter.

5.1 Background

Learning from a tabula rasa is the most common machine learning paradigm [Mikolov et al., 2018]. For an arbitrary task $t_k \in \mathcal{T}_\infty = \{t_i \mid i \in \mathbb{Z}^+\}$, a model f_k is initialized and trained using a dataset $\mathbb{D}_k^{(train)}$ or environment $\mathbb{E}_k$ to minimize a loss function ℓ_k or maximize a reward r_k with the aim of achieving a certain objective o_k. Contrary to human learning, the model f_k does not take into account knowledge learned in prior models $\mathcal{F}_{k-1} = \{f_i \mid 1 \le i \le k-1\}$ for tasks $\mathcal{T}_{k-1} = \{t_i \mid 1 \le i \le k-1\}$ using datasets $\mathcal{D}_{k-1} = \{\mathbb{D}_i^{(train)} \mid 1 \le i \le k-1\}$ or environments $\mathcal{E}_{k-1} = \{\mathbb{E}_i \mid 1 \le i \le k-1\}$, which may lead to a slower learning process that requires more data and possibly results in suboptimal performance [Lake et al., 2017]. Continual learning is key to advancing machine learning, whereby experiences and knowledge can be accumulated and re-purposed over tasks $\mathcal{T}_\infty$ [Thrun and Mitchell, 1995][*].

As previously mentioned in Section 2.9, continual learning is defined as follows: "The system has performed K tasks. When faced with the $(K+1)$th task, it uses the knowledge gained from the K tasks to help the $(K+1)$th task." [Thrun, 1996]. This poses numerous challenges, two of which are catastrophic forgetting and negative forward transfer. Catastrophic forgetting [McCloskey and Cohen, 1989, Ratcliff, 1990] refers to the case when the performance of the system for the K tasks degrades when it learns the $(K+1)$th task, as the model forgets the K tasks to learn the $(K+1)$th task. Negative forward transfer refers to the case when the K tasks have a negative effect on the performance of the system for the $(K+1)$th task, which can occur when the $(K+1)$th task is not related to any of the K tasks leading to interfere and worse performance.

Continual learning requires: (1) a strategy to select the next task to learn from a pool of candidate tasks to facilitate learning; (2) a mechanism to efficiently grow the capacity of the model to accommodate learning new tasks (if necessary); and, (3) a method to accumulate, maintain, and utilize knowledge to learn future tasks without significant adverse effects on the learned tasks.

Herein, progressive learning is proposed, a deep learning framework for continual learning that aims to address the aforementioned three desiderata. Progressive learning comprises three procedures: curriculum, progression, and pruning. Curriculum is used to actively select a subsequent task to learn from a pool of candidate tasks. Progression grows the capacity of the model by adding new parameters, denoted *progressive block*, that leverage parameters learned in prior tasks to learn from data available for the new task at hand. Only new parameters are trained, and existing parameters are not altered, which sidesteps the problem of catastrophic

[*]Since the total number of tasks at a given time is known to be finite, $\mathcal{T}_\infty$ is reduced to $\mathcal{T}_N = \{t_i \mid 1 \le i \le N\}$.

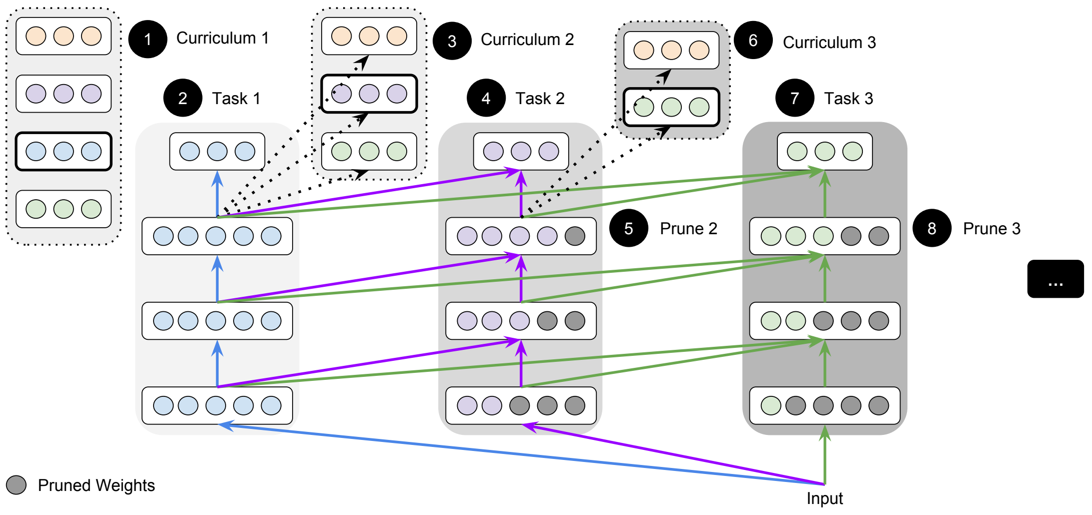

Figure 5.1: Overview of progressive learning for three tasks. Initially, a curriculum strategy is used to select a task (blue) from the pool of candidate tasks. Second, a model is trained to perform the selected task (blue), and the learned parameters are constant thereafter. Third, the curriculum strategy is employed to select the subsequent task (purple). Fourth, new model parameters, denoted progressive block, which draw connections from the preceding layer in the block as well as the preceding layer in prior progressive block(s), are added and trained to perform the selected task (purple). Fifth, after training the newly added progressive block to convergence, a pruning procedure is used to remove weights without compromising performance. Finally, the curriculum, progression, and pruning procedures are repeated for the third task (green), and for all remaining task(s) subsequently.

forgetting; however, this leads to growth in the number of parameters as the number of tasks increases. Pruning is used to counteract this growth, where it is used to attempt to remove newly added trained weights in a greedy layer-wise manner without compromising performance. Pruning may also aid in mitigating negative forward transfer as it may allow progressive learning to be reduced to independent learning by removing parameters between progressive blocks, if the new task is not related to prior tasks. Figure 5.1 illustrates the progressive learning framework.

Despite relevant work in curriculum [Bengio et al., 2009, Ruvolo and Eaton, 2013a], progression [Terekhov et al., 2015, Rusu et al., 2016], and pruning [LeCun et al., 1990, Han et al., 2015], there is no deep learning framework for continual learning to point to. The main novel contribution of progressive learning is the construction and formulation of a deep learning framework that allows a holistic and systematic approach to continual learning. Key to the viability of progressive learning is the following:

- The curriculum procedure offers a straight-forward strategy for *active task selection*, i.e., select a task from a pool of candidate tasks, with respect to the current state of knowledge leading to a natural development in task difficulty.
- The progression procedure provides a mechanism to grow the capacity of the model to accommodate learning new tasks, encourages the reuse of features by *concatenating* features learned in prior tasks with features being learned for the task at hand, and avoids catastrophic forgetting.
- The *greedy layer-wise pruning* procedure builds on intuition that the specificity of features varies across layers in deep networks, as shown in Chapter 4, and therefore aims to adjust the pruning amount per layer to counteract the growth in the number of parameters. Furthermore, the pruning procedure can aid in mitigating negative forward transfer by reducing progressive learning to independent learning via removing parameters between progressive blocks if tasks are unrelated.

Progressive learning is the first to package the aforementioned features into a single framework.

5.2 Related Work

Continual Learning. Continual learning in deep learning and neural networks has been studied in numerous prior works and several paradigms have been proposed, as reviewed in Section 2.9.

Curriculum. Curriculum learning follows the notion that a meaningful order of tasks when learning multiple tasks sequentially could significantly facilitate learning [Elman, 1993]. Curriculum strategies for training deep neural networks were studied in [Bengio et al., 2009, Graves et al., 2017], while active task selection was studied in [Ruvolo and Eaton, 2013a]. Therein, models trained with curriculum strategies outperformed the same models trained without curriculum strategies indicating the importance of a structured process to determine a meaningful order of tasks. Note that some studies reported that curriculum strategies that start with easier tasks and progress towards harder tasks outperformed anti-curriculum strategies that start with harder tasks and progress towards easier tasks [Bengio et al., 2009, Graves et al., 2017], whereas other studies found the opposite to be true [Ruvolo and Eaton, 2013a].

Progression. Block-modular neural networks were proposed in [Terekhov et al., 2015], and similarly, progressive neural networks were proposed in [Rusu et al., 2016]. Both approaches were explicitly designed for learning tasks in a sequential manner by training a model for each task that is connected to prior models via adaptive connections. For example, progressive neural networks [Rusu et al., 2016] start with a standard multi-layered neural network trained for some arbitrary task; for every new subsequent task, an additional multi-layered neural network, denoted block (or column in [Rusu et al., 2016]), connected to prior block(s) via lateral connections in addition to its own connections, is added to the network and trained for the new task at hand; all prior block(s) remain constant, thus avoiding catastrophic forgetting. A block is connected to prior block(s) via progressive layers that draw connections from the previous layer in its own block as well as the previous layer in prior block(s), as follows:

$$\hat{\mathbf{y}}^{(l)}_{(k)} = \phi\left(\mathbf{W}^{(l)}_{(k)}\hat{\mathbf{y}}^{(l-1)}_{(k)} + \sum_{j<k}\mathbf{U}^{(l)}_{(k:j)}\hat{\mathbf{y}}^{(l-1)}_{(j)}\right), \tag{5.1}$$

where $\hat{\mathbf{y}}^{(l)}_{(k)} \in \mathbb{R}^{n_{l_k}}$ is the vector of activations of layer l in block f_k, n_{l_k} is the number of units in layer l in block f_k, ϕ is an element-wise non-linear function, $\mathbf{W}^{(l)}_{(k)} \in \mathbb{R}^{n_{l_k}\times n_{(l-1)_k}}$ is the weight matrix of layer l in block f_k, $\hat{\mathbf{y}}^{(l-1)}_{(k)} \in \mathbb{R}^{n_{(l-1)_k}}$ is the vector of activations of the previous layer $(l-1)$ in block f_k, $\mathbf{U}^{(l)}_{(k:j)} \in \mathbb{R}^{n_{l_k}\times n_{(l-1)_j}}$ are the lateral connections from layer $(l-1)$ in block f_j to layer l in block f_k, and $\mathbf{y}^{(l-1)}_{(j)} \in \mathbb{R}^{n_{(l-1)_j}}$ is the vector of activations of the previous layer $(l-1)$ in block f_j.

Pruning. Pruning the parameters of neural networks can not only be used to decrease the computational and memory requirements and speed up training and inference [Han et al., 2015, Han et al., 2016, Louizos et al., 2018], but can also improve generalization [LeCun et al., 1990, Hassibi and Stork, 1993, Reed, 1993, Louizos et al., 2018] as deep neural networks, in particular, are typically over-parametrized, which can easily lead to over-fitting. The pruning procedure used here is an extension to the pruning method proposed in [Han et al., 2015, Han et al., 2016] that relies on training a neural network via standard methods, followed by pruning unimportant parameters, then fine-tuning the remaining parameters of the network to compensate for the pruned parameters.

Progressive Learning Progressive learning extends and unifies several key ideas in curriculum, progression, and pruning into a holistic and systematic framework for continual learning. The curriculum procedure in progressive learning is a simple method for active task selection that allows for a natural development in task difficulty relative to the state of knowledge, unlike prior work [Ruvolo and Eaton, 2013a], which found anti-curriculum methods to work better. The progression procedure in progressive learning relies on the concatenation operation, which

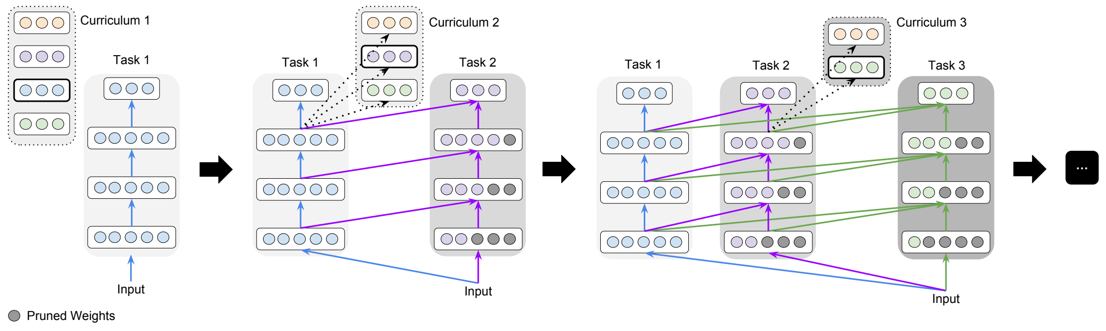

Figure 5.2: Procedural illustration of progressive learning for three tasks. In the first iteration, a curriculum strategy is used to select a task (blue) from the pool of candidate tasks. Then, a model is trained to perform the selected task (blue), and the learned parameters are constant thereafter. In the second iteration, the curriculum strategy is employed to select the subsequent task (purple). Then, new model parameters, denoted progressive block, which draw connections from the preceding layer in the block as well as the preceding layer in prior progressive block(s), are added and trained to perform the selected task (purple). Subsequently, after training the newly added progressive block to convergence, a pruning procedure is used to remove weights without compromising performance. In the third iteration, the curriculum, progression, and pruning procedures are repeated for the third task (green). Further iterations of the three procedures continue for all following task(s).

encourages the reuse of features [Huang et al., 2017], instead of the summation operation used in [Rusu et al., 2016]. The pruning procedure in progressive learning is carried out in a greedy layer-wise manner, unlike prior work [Han et al., 2015], which prunes the entire network concurrently. These procedures can be regarded as modular building blocks for the progressive learning framework that may be employed individually or in combination as required. Other procedures can be added to progressive learning to address further desiderata of continual learning.

5.3 Progressive Learning

Progressive learning, introduced and investigated in this work, is a deep learning framework for continual learning, whereby tasks are learned in sequence with the ability to use prior knowledge from previously learned tasks to facilitate the learning and execution of new ones. The iterative framework comprises three procedures: curriculum, progression, and pruning, which are presented in Sections 5.3.1, 5.3.2 and 5.3.3 respectively.

Intuitively, the curriculum procedure employs a strategy to select the subsequent task to learn from a pool of candidate tasks to facilitate learning, the progression procedure grows the capacity of the model by adding new parameters that leverage parameters learned in prior tasks, while learning from data available for the new task at hand, without being susceptible to catastrophic forgetting, and finally, the pruning procedure is used to counteract the growth in the number of parameters in the progression procedure as further tasks are learned, and can also aid in mitigating negative forward transfer.

Algorithm 5.6 is a functional decomposition of progressive learning in the supervised learning case, where the objective is to learn a model $\mathbf{y}_{(k)} = \mathcal{F}_k(\mathbf{x}_{(k)})$ for task t_k that has a dataset $\mathbb{D}_k^{(train)}$ composed of M_k exemplars $\mathbf{X}_{(k)} \in \mathbb{R}^{M_k \times n_{i_k}}$ and corresponding labels or targets $\mathbf{Y}_{(k)} \in \mathbb{R}^{M_k \times n_{c_k}}$ such that n_{i_k} and n_{c_k} are the dimensionalities of the input and output respectively, $\mathbf{x}_{(k)} \in \mathbf{X}_{(k)}$, and $\mathbf{y}_{(k)} \in \mathbf{Y}_{(k)}$. Figure 5.2 is a procedural illustration of progressive learning.

5.3.1 Curriculum

The objective of the curriculum procedure is to determine the most appropriate subsequent task to learn from a pool of candidate tasks with respect to the state of knowledge at the time.

Consider a pool of tasks $\mathcal{T}_P \subset \mathcal{T}_N$, where $\mathcal{T}_N$ is a set of N classification or regression tasks, a classification or regression model g_j can be trained and evaluated for each candidate task $t_j \in \mathcal{T}_P$, using data available for the respective task $\mathbb{D}_j$ and features computed from the model learned in prior tasks f_{k-1}. The performance of each of these classification or regression models g_j can then be used to measure the relevance of the current features in f_{k-1} to each of these tasks t_j [Fayek et al., 2018]. The subsequent task t_k can be chosen using a metric that utilizes the performance of the classification or regression models g_j. Herein, the task with the highest performance, i.e., the task that can be best solved using the already learned features, is chosen as in Equation (5.2). Following Equation (5.2) is likely to lead to a natural progression in task difficulty relative to the state of knowledge at each iteration.

$$t_k = \underset{t_j \in \mathcal{T}_p}{\operatorname{argmax}} \mathcal{R}\left(g_j\left(f_{k-1}\left(\mathbf{X}_{(j)}; \mathbf{\Theta}_{k-1}^{(1:(L_{(k-1)}-1))\star}\right); \boldsymbol{\psi}_j\right); \mathbf{Y}_{(j)}\right) \tag{5.2}$$

where $\mathcal{R}$ denotes some measure of positive normalized performance (the larger, the better), g_j is a classification or regression model parametrized by adaptive parameters $\boldsymbol{\psi}_j$ trained using features extracted from $f_{k-1}(X_{(j)}; \mathbf{\Theta}_{k-1}^{(1:(L_{(k-1)}-1))\star})$ such that parameters $\mathbf{\Theta}_{k-1}^{(1:(L_{(k-1)}-1))\star}$ were trained on some previous task t_{k-1} and held constant thereafter, $L_{(k-1)}$ is the total number of layers in block f_{k-1}, $\mathbf{X}_{(j)}$ and $\mathbf{Y}_{(j)}$ are the training / validation exemplars and labels / targets of task t_j respectively. When $k = 1$, g_j can be trained on the data directly.

Algorithm 5.6 Functional decomposition of progressive learning in the supervised learning case.

```
Require: Candidate Tasks $\mathcal{T}_N = \{t_i \mid 1 \leq i \leq N\}$
Require: Datasets $\mathcal{D}_N = \{\mathbb{D}_i = (\mathbf{X}_{(i)}, \mathbf{Y}_{(i)}, t_i) \mid 1 \leq i \leq N\}$
Output: Model $\mathcal{F}_K = \{f_i \mid 1 \leq i \leq K\}$
                                          // $\mathcal{F}_K$ is a model of $K$ progressive blocks,
          // each $f_k$ is trained for its respective task $t_k \in \mathcal{T}_N$ using $\mathbb{D}_k \in \mathcal{D}_N$
 1: Initialize:
    $\mathcal{T}_K = \varnothing$                    // learned tasks s.t. $\mathcal{T}_K \subset \mathcal{T}_N$
    $\mathcal{D}_K = \varnothing$                    // processed datasets s.t. $\mathbb{D}_K \subset \mathcal{D}_N$
    $\mathcal{T}_P \leftarrow \mathcal{T}_N - \mathcal{T}_K$       // remaining tasks s.t. $\mathcal{T}_P \subset \mathcal{T}_N$
    $\mathcal{D}_P \leftarrow \mathcal{D}_N - \mathcal{D}_K$       // remaining datasets s.t. $\mathbb{D}_P \subset \mathcal{D}_N$
    $\mathcal{F}_K = \varnothing$
 2: for k = 1 to $N$ do
 3:     Curriculum: select next task $t_k \in \mathcal{T}_p$          // see Section 5.3.1
 4:     Progression: train $f_k$ on task $t_k$ using $\mathbb{D}_k = (\mathbf{X}_{(k)}, \mathbf{Y}_{(k)}, t_k)$
                                                          // see Section 5.3.2
 5:     if k > 1 then
 6:         Pruning: prune $f_k$                          // see Section 5.3.3
 7:     end if
 8:     $\mathcal{T}_P \leftarrow \mathcal{T}_P - t_k$
 9:     $\mathcal{D}_P \leftarrow \mathcal{D}_P - \mathbb{D}_k$
10:     $\mathcal{T}_K \leftarrow \mathcal{T}_K + t_k$
11:     $\mathcal{D}_K \leftarrow \mathcal{D}_K + \mathbb{D}_k$
12:     $\mathcal{F}_K \leftarrow \mathcal{F}_K + f_k$
13: end for
14: return Model $\mathcal{F}_K$
```

The curriculum procedure is a dynamic process that evaluates which task to learn at each iteration. For N tasks, this requires $N(N+1)/2$ classification or regression models to be trained and evaluated, assuming no new tasks are introduced to the pool of candidate tasks. Training a simple classification or regression model is not as computationally demanding as training a deep model. Further, training and evaluation can be carried out on only a subset of the candidate tasks or using only a subset of the available data for each task.

5.3.2 Progression

The objective of the progression procedure is to (1) increase the capacity of the model by adding new parameters to accommodate learning a new task; more importantly, (2) provide a mechanism to leverage parameters learned in prior tasks while learning the new task at hand from data available for the task without being susceptible to catastrophic forgetting.

This can be achieved by instantiating and training a new multi-layered neural network for each new task with randomly initialized parameters, denoted progressive block, that in addition to its layer-to-layer connections, draws connections from respective preceding layers in existing progressive blocks, i.e., a layer in a progressive block receives input from the preceding layer in that block, as well as the preceding layer in all prior blocks. The parameters in the newly added progressive block provide the model with the additional capacity to learn the new task, while the connections between the prior blocks and the newly added block provide a mechanism to leverage parameters learned in prior tasks.

Progressive learning leads to a model that is formed of multiple progressive blocks that are in turn formed of multiple progressive layers, where initially a standard neural network f_1 is trained on task t_1 using dataset $\mathbb{D}_1$, followed by a progressive block f_k for each subsequent task t_k that is trained using only dataset $\mathbb{D}_k$. Catastrophic forgetting is prevented by only training the newly added parameters of progressive block f_k, while all parameters in prior blocks remain constant. At inference time for task t_k, the forward pass of a progressive neural network will propagate through progressive blocks $\{f_i \mid 1 \leq i \leq k\}$.

Formally, consider a standard fully connected layer (see Equation (2.21) and Equation (2.22)):

$$\hat{\mathbf{y}}^{(l)} = \phi(\mathbf{W}^{(l)}\hat{\mathbf{y}}^{(l-1)}), \tag{5.3}$$

where $\hat{\mathbf{y}}^{(l)} \in \mathbb{R}^{n_l}$ is a vector of activations of layer l, n_l is the number of units in layer l, ϕ is an element-wise non-linear function such as the Rectified Linear Unit (ReLU), $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{(l-1)}}$ is a matrix of weights of layer l, $n_{(l-1)}$ is the number of units in the preceding layer $(l-1)$, $\hat{\mathbf{y}}^{(l-1)} \in \mathbb{R}^{n_{(l-1)}}$ is a vector of activations of the preceding layer $(l-1)$, $l \in \{1, \ldots, L\}$, L is the total number of layers, such that $\hat{\mathbf{y}}^{(0)}$ is the input to the model, and $\hat{\mathbf{y}}^{(L)}$ is the output of the model, and the bias term is omitted for clarity. A progressive fully connected layer is given by:

$$\hat{\mathbf{y}}^{(l)}_{(k)} = \phi\left(\mathbf{W}^{(l)}_{(k)}\left(\hat{\mathbf{y}}^{(l-1)}_{(k)} \,||\, \left(\mathop{||}_{j=1}^{(k-1)} \hat{\mathbf{y}}^{(l-1)}_{(j)}\right)\right)\right), \tag{5.4}$$

where $\hat{\mathbf{y}}^{(l)}_{(k)} \in \mathbb{R}^{n_{l_k}}$ is a vector of activations of layer l in progressive block f_k, n_{l_k} is the number of units in layer l in progressive block f_k, $\mathbf{W}^{(l)}_{(k)} \in \mathbb{R}^{n_{l_k} \times \sum_{j=1}^{k} n_{(l-1)_j}}$ is a matrix of weights of layer l in progressive block f_k, $\sum_{j=1}^{k} n_{(l-1)_j}$ is the total number of units in layers $(l-1)$ in the current and prior progressive blocks, $\hat{\mathbf{y}}^{(l-1)}_{(k)} \in \mathbb{R}^{n_{(l-1)_k}}$ and $\hat{\mathbf{y}}^{(l-1)}_{(j)} \in \mathbb{R}^{n_{(l-1)_j}}$ are vectors of activations of the preceding

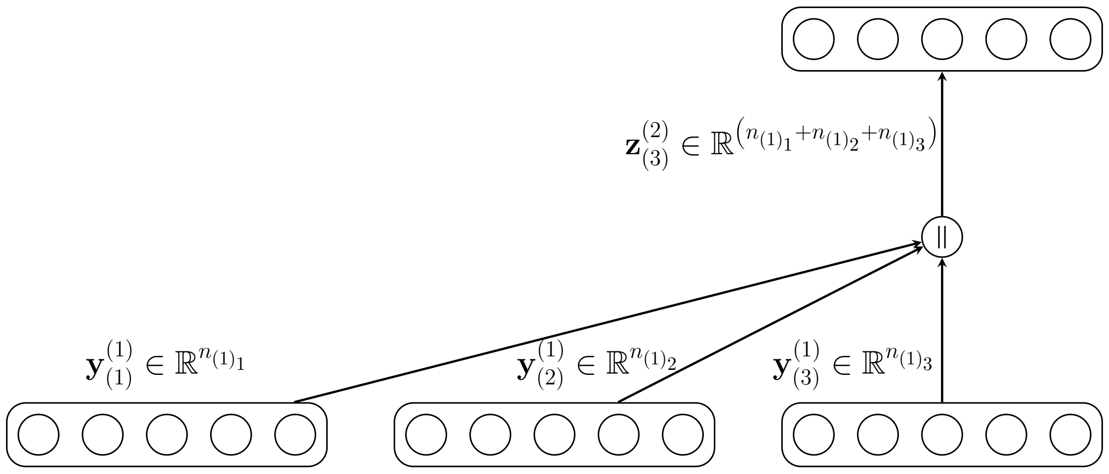
$\mathbf{z}_{(3)}^{(2)} \in \mathbb{R}^{\left(n_{(1)_1}+n_{(1)_2}+n_{(1)_3}\right)}$
$||$
$\mathbf{y}_{(1)}^{(1)} \in \mathbb{R}^{n_{(1)_1}}$
$\mathbf{y}_{(2)}^{(1)} \in \mathbb{R}^{n_{(1)_2}}$
$\mathbf{y}_{(3)}^{(1)} \in \mathbb{R}^{n_{(1)_3}}$

Figure 5.3: Illustration of the concatenation operation for three blocks.

layer $(l-1)$ in progressive blocks f_k and f_j respectively, $l \in \{2, \ldots, L_k\}$, L_k is the total number of layers in progressive block f_k, and $||$ denotes the concatenation operation as illustrated in Figure 5.3. When $k = 1$, Equation (5.4) is reduced to Equation (5.3), i.e., the first progressive block f_1 is a standard multi-layered neural network. Equation (5.4) can be trivially extended to convolutional and recurrent layers.

Note the important distinction between Equation (5.1) [Rusu et al., 2016] and Equation (5.4), in that Equation (5.4) uses the concatenation operation, whereas Equation (5.1) uses a weighted sum of outputs of the previous layers in the previous blocks as well as the current block. The use of the concatenation operation leads to a simpler design that encourages the reuse of features and is easier to optimize due to the improved flow of gradients [Huang et al., 2017]. In particular, the use of the concatenation operation requires only a single weight matrix $\mathbf{W}_{(k)}^{(l)}$ and a single matrix-vector multiplication (in the case of fully connected layers) for each progressive layer, which is straightforward to parallelize on Graphics Processing Units (GPUs) and convenient to prune as described in Section 5.3.3. In Section 5.4 and Tables 5.2 and 5.3, it is shown that the use of the concatenation operation in Equation (5.4), over Equation (5.1) or the summation operation, leads to improved performance.

5.3.3 Pruning

The objective of the pruning procedure is to counteract the growth in the number of parameters in the model as the number of tasks increases by attempting to remove weights in a progressive block after each progression procedure. Moreover, in most prior work on continual learning, tasks are assumed to be related; if this assumption is invalid, it may lead to negative forward transfer. Herein, the effect of negative forward transfer is mitigated by pruning, which can remove the connections that are responsible for this undesirable transfer, or even reduce progressive learning to independent learning if the new task is not related to any prior task, by removing all connections to prior blocks.

The specificity of features varies across layers in deep networks that tend to learn low-level features in initial layers and transition gradually to high-level more task-specific features towards final layers, as shown in Chapter 4. Similar low-level features commonly appear across various datasets and tasks, while high-level features are somewhat more attuned to the dataset or task at hand, making initial layers more generic, i.e., easier to use across datasets or tasks. To this end, initial layers in a progressive block are expected to be more prone to pruning than final layers without comprising performance as the model can rely on features learned in initial layers in prior blocks. Therefore, an iterative greedy layer-wise pruning procedure is proposed, which aims to greedily prune the weights in each layer in a progressive block without compromising performance.

The proposed pruning procedure is as follows. Initially, a progressive block f_k is trained as outlined in Section 5.3.2 until convergence, using the following loss function:

$$\mathcal{L}_k(\mathbf{\Theta}_k) = \ell_k(\mathbf{\Theta}_k) + \lambda \frac{1}{2} \sum_{l=1}^{L_k} \|\mathbf{W}_{(k)}^{(l)}\|_2^2, \tag{5.5}$$

where $\mathcal{L}_k(\mathbf{\Theta}_k)$ is the total loss of progressive block f_k, $\mathbf{\Theta}_k$ denotes the parameters of progressive block f_k including weights $\mathbf{W}_{(k)}$, biases $\mathbf{b}_{(k)}$, Batch Normalization (BatchNorm) parameters, etc., $\ell_k(\mathbf{\Theta}_k)$ is the loss function of progressive block f_k without the weight-decay regularization term, λ is a hyperparameter to trade-off both terms, and $\|\mathbf{W}_{(k)}^{(l)}\|_2$ is the L^2 norm of the weights of layer l in progressive block f_k. The regularization term in the loss function in Equation (5.5) is used to encourage the weights of progressive block f_k to be small in magnitude, and therefore, implicitly encourage the use of previous progressive blocks since their weights do not constitute part of the regularization term. This somewhat validates the assumption that weights with small magnitude are not as important as weights with large magnitude, and the magnitude of the weights can be used as a pruning criterion [Han et al., 2016].

Subsequently, after training to convergence, iteratively for each layer, starting from the initial layer to the final layer in progressive block f_k, the smallest q-percentile of the sorted magnitudes of the weights of the layer are removed and the entire progressive block continues to train for a small number of iterations, to compensate for the pruned weights, where q is a hyperparameter. The pruning and training procedures are repeated for an increasingly larger q, until the entire

Algorithm 5.7 Greedy layer-wise pruning procedure in progressive learning.

```
Require: Trained progressive block $f_k$
Require: Pruning step size $\tilde{q}$
Output: Pruned progressive block $\tilde{f}_k$
 1: $\mathcal{R}_b \leftarrow$ Evaluate $f_k$                    // benchmark performance
 2: $f_b \leftarrow f_k$                                        // best model
 3: for $l = 1$ to $L_k$ do               // $L_k$ the total number of layers in $f_k$
 4:     for $q = \tilde{q}$ to 100 by $\tilde{q}$ do
 5:         Prune: prune smallest $q\%$ of weights $|\mathbf{W}^{(l)}_{(k)}|$ in $f_k$
 6:         Train: continue training $f_k$
 7:         $\mathcal{R}_c \leftarrow$ Evaluate $f_k$                // current performance
 8:         if $\mathcal{R}_c \geq \mathcal{R}_b$ then
 9:             $\mathcal{R}_b \leftarrow \mathcal{R}_c$
10:             $f_b \leftarrow f_k$
11:         end if
12:     end for
13:     $f_k \leftarrow f_b$
14: end for
15: $\tilde{f}_k \leftarrow f_b$
16: return Pruned progressive block $\tilde{f}_k$
```

progressive layer is removed, $q \to 100\%$, and the largest q that does not lead to degradation in performance is used. The complete pruning procedure is detailed in Algorithm 5.7.

5.4 Experiments in Image Recognition

Progressive learning is analysed on a set of 11 supervised classification tasks in the image recognition domain using the CIFAR-10 and CIFAR-100 datasets. The tasks in this setting are closely related to each other since 10 out of the 11 tasks are forged from the CIFAR-100 dataset and all 11 tasks have the same image recognition objective, yet the challenge is in the somewhat large number of tasks. Section 5.4.1 details the experimental setup and experiments carried out. Section 5.4.2 outlines the evaluation criteria used. Section 5.4.3 presents the results and observations.

5.4.1 Experimental Setup

Datasets. The CIFAR-10 and CIFAR-100 datasets [Krizhevsky, 2009] (see Appendix A.1 for more details) were used in this experiment. Each dataset comprises a training set of 50000 images and a test set of 10000 images. Both datasets were used to construct a set of 11 supervised image recognition tasks as follows [Zenke et al., 2017]. The CIFAR-10 dataset was used to construct the first classification task, where 45000 training images drawn from the original CIFAR-10 training set were used as the training set, with the remaining 5000 images held out as a validation set. The 10000 original CIFAR-10 test set images were used as a test set for this task. The CIFAR-100 was used to construct the remaining 10 classification tasks, each of which was composed of 10 classes drawn from CIFAR-100 100 classes. For each task, the training and validation sets comprised 4500 and 500 images respectively, both of which, were drawn from the original CIFAR-100 training set, and a test set of 1000 images, which were all the test images in the original CIFAR-100 test set that belong to the selected 10 classes of the task. Note that all sets in all tasks are mutually exclusive.

Preprocessing. The mean and standard deviation of the images were normalized to zero and one respectively per colour channel using the training set statistics for each task individually.

Model. The architecture of the model used in this image recognition experiment is described in Table 5.1, ignoring the concatenation operation. The model is a Convolutional Neural Network (ConvNet) that comprises four convolutional layers followed by two fully connected layers, with BatchNorm, ReLUs, and dropout interspersed in-between, and a max pooling layer after the non-linear function of the second and fourth convolutional layers. The final fully connected layer is followed by a softmax function.

Training and Implementation Details. The parameters of the convolutional and fully connected layers were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers respectively, where n_i is the number of inputs to the layer, as recommended in [He et al., 2015]. All layers were regularized using weight decay with penalty $\lambda = 0.001$ and dropout with drop probability r as indicated in Table 5.1. ADAM was used to optimize the parameters with respect to a negative log-likelihood loss function using batch size $M_b = 256$, learning rate $\alpha = 0.001$, first moment $\beta_1 = 0.99$, and second moment $\beta_2 = 0.999$ for 90 epochs.

The experiments were implemented using TensorFlow [Abadi et al., 2016]. Training was carried out on NVIDIA Tesla P100 GPUs. Training a single model from scratch for a single task on a single GPU required a wall time of approximately 30 mins.

Table 5.1: Convolutional neural network architecture for the image recognition tasks. The concatenation operation indicates the layers at which the output of the previous layer in all prior blocks are concatenated. The concatenation operation can be ignored in independent learning.

№	**Type**	**Size**	**Other**
1	Convolution	$32, 3 \times 3$	Stride = 1
	BatchNorm	—	—
	ReLU	—	—
	[Concatenation]	—	—
2	Convolution	$32, 3 \times 3$	Stride = 1
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride = 2
	Dropout	—	$r = 0.25$
	[Concatenation]	—	—
3	Convolution	$64, 3 \times 3$	Stride = 1
	BatchNorm	—	—
	ReLU	—	—
	[Concatenation]	—	—
4	Convolution	$64, 3 \times 3$	Stride = 1
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride = 2
	Dropout	—	$r = 0.25$
	[Concatenation]	—	—
5	Fully Connected	512	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.5$
	[Concatenation]	—	—
6	Fully Connected	10	—
	Softmax	—	—

Progressive Learning. For progressive learning, initially a model that has the same architecture and training recipe outlined above was trained on the first task. An additional progressive block was then added for each subsequent task. Each progressive block had the same architecture with half the number of feature maps / units listed in Table 5.1 in each layer. The outputs of the layer preceding a convolutional or fully connected layer in all prior blocks were concatenated and fed as input to that convolutional or fully connected layer, as marked *[Concatenation]* in Table 5.1. In each progression procedure, the parameters of the new progressive block were randomly initialized and trained using the same training recipe as described above. Note that parameters in a progressive block are held constant once the block is trained and pruned, including during training and pruning subsequent progressive blocks.

5.4.2 Evaluation Criteria

A number of evaluation criteria were used to assess the performance of the models over the various settings. As standard for the single task setting, the classification accuracy was used to measure the performance of the models in each task. Additionally, the following criteria are used to assess the effectiveness of progressive learning over all tasks involved.

The average accuracy across all tasks as defined in Equation (5.6) is used as the primary evaluation criterion.

$$\text{Average Accuracy } (AA) = \frac{1}{K}\sum_{k=1}^{K}\mathcal{R}_k, \tag{5.6}$$

where K is the total number of learned tasks, and $\mathcal{R}_k$ is the accuracy of task t_k in progressive learning.

The progressive knowledge transfer, which measures the difference between a task learned with progressive learning and the same task learned independently is defined in Equation (5.7) [Lopez-Paz and Ranzato, 2017]:

$$\text{Progressive Knowledge Transfer } (PKT) = \frac{1}{K}\sum_{k=2}^{K}(\mathcal{R}_k - \bar{\mathcal{R}}_k), \tag{5.7}$$

where $\bar{\mathcal{R}}_k$ is the accuracy of task t_k learned independently.

The learning speed, that is the performance of the model as a function of training iterations, of tasks learned with progressive learning versus the same tasks learned independently is reported.

The number of free parameters in a progressive block, before and after pruning, and the number of fixed parameters in progressive blocks preceding that block, are also reported to evaluate the effectiveness of the pruning procedure, as well as gauge the number of additional parameters required for additional tasks as the number of tasks increases.

5.4.3 Results

Independent Baselines. Baseline models were trained for each task independently, in that 11 models were trained for the 11 image recognition tasks. Tables 5.2 and 5.3 present the validation and test accuracy of the CIFAR-10 task, the validation and test average accuracy across the 10 CIFAR-100 tasks, and the validation and test average accuracy across all 11 CIFAR-10 and CIFAR-100 tasks. Figure 5.4 presents the test accuracy of the models for each of the 11 tasks.

Multiple Tasks Baselines. A single model was then used to learn all 11 tasks sequentially using transfer learning, where a model was trained and evaluated on one task, and the trained parameters were used to initialize the following model that was fine-tuned and evaluated on the subsequent task. This was carried out for all tasks successively, i.e., the model for task t_1 was initialized randomly and the model for task t_k was initialized using the trained parameters of task t_{k-1}. Note that the model was evaluated on task t_k after it was fine-tuned on that particular task only; the performance of the model on task t_k and all prior tasks would degrade as fine-tuning on the subsequent task t_{k+1} progressed, a typical example of catastrophic forgetting in transfer learning. The validation and test average accuracy as well as the validation and test progressive knowledge transfer are listed in Tables 5.2 and 5.3.

Subsequently, a multi-task learning model was used to learn all 11 tasks in parallel. Two models were used: a model similar to the model listed in Table 5.1 but with 11 parallel output layers, one for each task, and a larger model with five times the number of parameters as the other model. Due to interference between overlapping classes in the CIFAR-10 and CIFAR-100 datasets, the two previously described models were also trained for the 10 CIFAR-100 tasks only. Tables 5.2 and 5.3 list the validation and test average accuracy for all multi-task learning models.

Finally, a progressive neural network [Rusu et al., 2016] was used to learn all 11 tasks sequentially. This is conceptually the closest baseline to progressive learning. The validation and test average accuracy as well as the validation and test progressive knowledge transfer are listed in Tables 5.2 and 5.3.

Note that all previous models were trained using the same settings described in Section 5.4.1.

Table 5.2: Progressive learning Validation Average Accuracy (Val AA) and Test Average Accuracy (Test AA) over all 11 tasks using the CIFAR-10 and CIFAR-100 datasets unless otherwise indicated.

Method	**Val AA (%)**	**Test AA (%)**
CIFAR-10 (Single task)	84.86	83.36
CIFAR-100 (Avg. of 10 Independent Tasks)	79.97	78.51
CIFAR-10 & CIFAR-100 (Avg. of 11 Independent Tasks)	80.41	78.95
Transfer Learning (Avg. of 11 Independent Tasks)	80.23	80.39
Multi-Task Network (Avg. of 10 CIFAR-100 Tasks)	63.32	64.24
Multi-Task Network (Avg. of All 11 Tasks)	62.69	65.80
Large Multi-Task Network (Avg. of 10 CIFAR-100 Tasks)	67.38	66.99
Large Multi-Task Network (Avg. of All 11 Tasks)	68.38	71.26
Progressive Neural Networks [Rusu et al., 2016]	81.36	81.51
Curriculum + Progression + Pruning	**83.70**	**82.24**
Progression	81.86	81.60
Progression (Sum)	78.50	78.28
Progression + Pruning	83.96	82.18
Progression + Random Pruning	83.77	81.99
Curriculum + Progression	81.99	81.83
Anti-Curriculum + Progression	81.56	81.54

Table 5.3: Progressive learning Validation Progressive Knowledge Transfer (Val PKT) and Test Progressive Knowledge Transfer (Test PKT) over all 11 tasks using the CIFAR-10 and CIFAR-100 datasets unless otherwise indicated.

Method	**Val PKT (%)**	**Test PKT (%)**
CIFAR-10 (Single task)	—	—
CIFAR-100 (Avg. of 10 Independent Tasks)	—	—
CIFAR-10 & CIFAR-100 (Avg. of 11 Independent Tasks)	—	—
Transfer Learning (Avg. of 11 Independent Tasks)	-0.2	1.58
Multi-Task Network (Avg. of 10 CIFAR-100 Tasks)	—	—
Multi-Task Network (Avg. of All 11 Tasks)	—	—
Large Multi-Task Network (Avg. of 10 CIFAR-100 Tasks)	—	—
Large Multi-Task Network (Avg. of All 11 Tasks)	—	—
Progressive Neural Networks [Rusu et al., 2016]	1.04	2.82
Curriculum + Progression + Pruning	**3.62**	**3.62**
Progression	1.59	2.91
Progression (Sum)	-2.10	-0.74
Progression + Pruning	3.91	3.56
Progression + Random Pruning	3.70	3.34
Curriculum + Progression	1.74	3.17
Anti-Curriculum + Progression	1.26	2.85

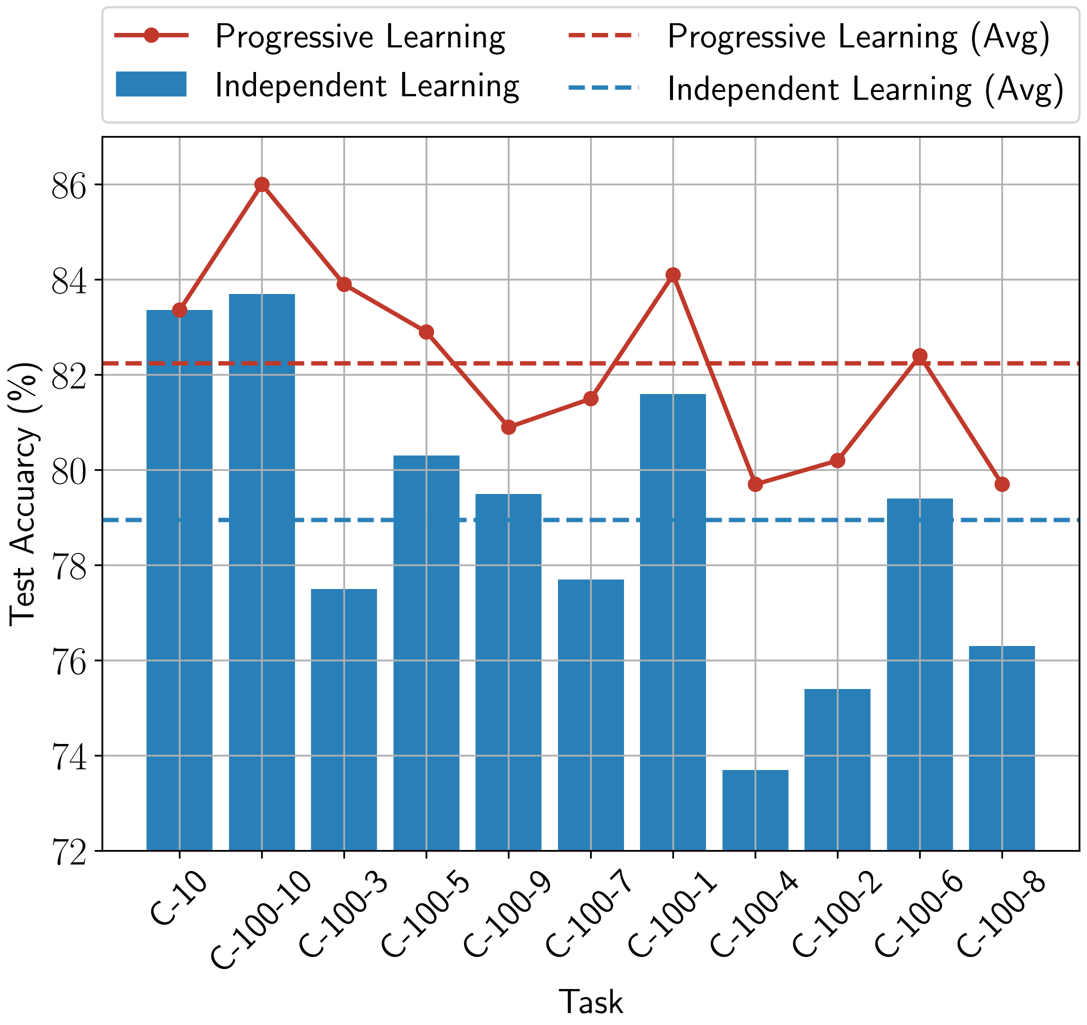

Figure 5.4: Accuracy of progressive learning vs. independent learning for all 11 image recognition tasks. The tasks are ordered according to the outcome of the curriculum procedure. C-10 denotes the CIFAR-10 task, C-100-s indicates the s$^{\text{th}}$ CIFAR-100 task.

Progressive Learning. Progressive learning was used to learn the 11 tasks sequentially using the curriculum, progression, and pruning procedures. The order of the tasks was determined via the curriculum procedure that utilized the validation accuracy as its metric. Once a task was chosen, a progression procedure was employed. This was followed by pruning, where the pruning amount q was varied in 25% intervals, i.e., the pruning amount $q \in \{25\%, 50\%, 75\%, 100\%\}$, only for layers that receive input from preceding progressive blocks. The validation set was used to evaluate the performance in the pruning procedure. The validation and test average accuracy as well as the validation and test progressive knowledge transfer are listed in Tables 5.2 and 5.3. The test accuracy for each of the 11 tasks is plotted in Figure 5.4. These metrics were measured after learning all 11 tasks. It can be seen from the results in Tables 5.2 and 5.3 and Figure 5.4 that progressive learning leads to better performance compared with baseline methods, and in particular independent learning by a large margin.

Progressive learning leverages knowledge learned in prior tasks when learning a new task, and it is therefore expected to learn new tasks faster. Figure 5.5 demonstrates the validation error as a function of training iterations, where it is shown that progressive learning converges faster than independent learning.

Ablation Study. To gain a better understanding of progressive learning, an ablation study was performed, whereby each component of progressive learning was studied separately. Tables 5.2 and 5.3 lists the outcome of this ablation study. It can be seen that all three procedures of progressive learning contribute to improvement in performance, with the majority of the contribution obtained from the progression procedure. The following was also observed. First, the presence of a curriculum procedure that chooses the easiest task as the next task leads to an improvement in performance, and outperforms the anti-curriculum strategy that chooses the most difficult task as the next task, as well as the case where no curriculum procedure is employed. Second, the proposed progression procedure based on the concatenation operation outperforms the progression procedure based on the summation operation, and is superior to all baseline models as well as independent learning. Third, the magnitude-based greedy layer-wise pruning procedure improves performance, in addition to decreasing the number of parameters in the model, and outperforms random greedy layer-wise pruning.

Analysis. Progression leads to growth in the number of parameters and pruning was introduced to counteract this growth. As the number of learned tasks increases, the number of pruned parameters is expected to increase, especially in initial layers and with task relatedness. This is shown in Figure 5.6 (left), where the difference between the total number of parameters before and after pruning increases as the

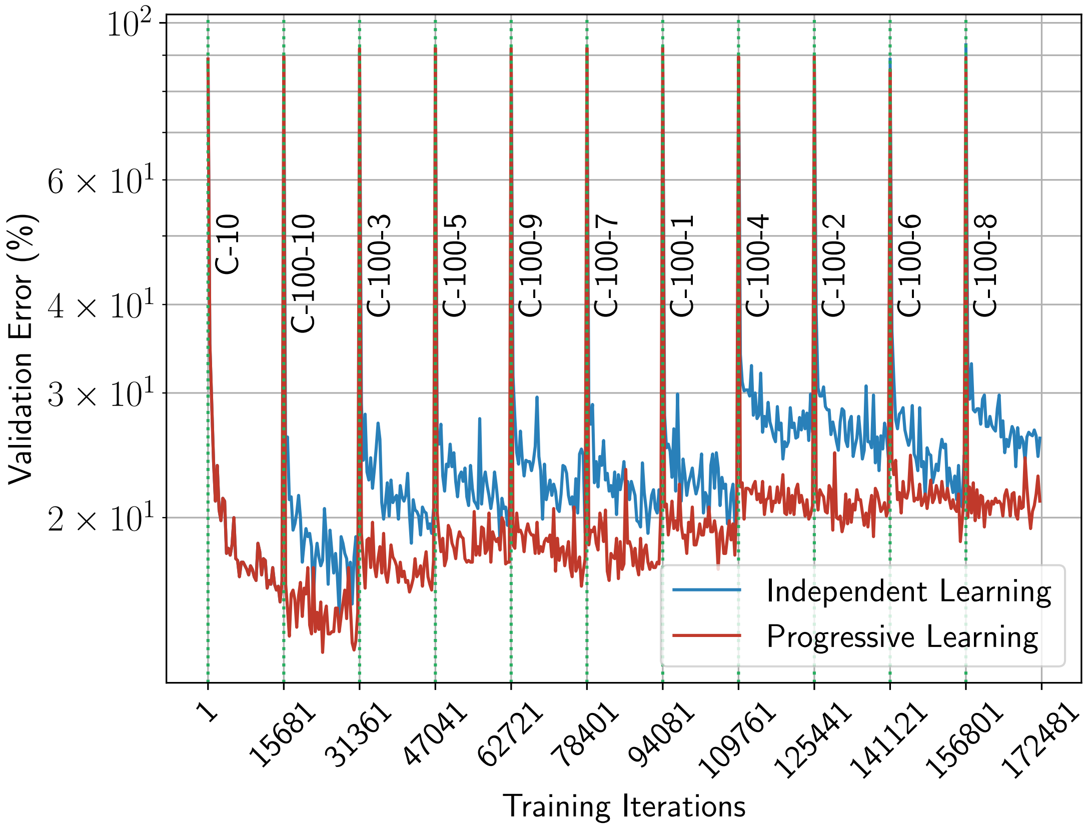

Figure 5.5: Learning curves, validation error as a function of training iterations, for progressive and independent learning for the 11 CIFAR-10 and CIFAR-100 tasks. Progressive Learning demonstrates faster learning compared with independent learning. Note that tasks are ordered according to the outcome of the curriculum strategy. Models are reset to random initialization at the beginning of each task in the case of independent learning.

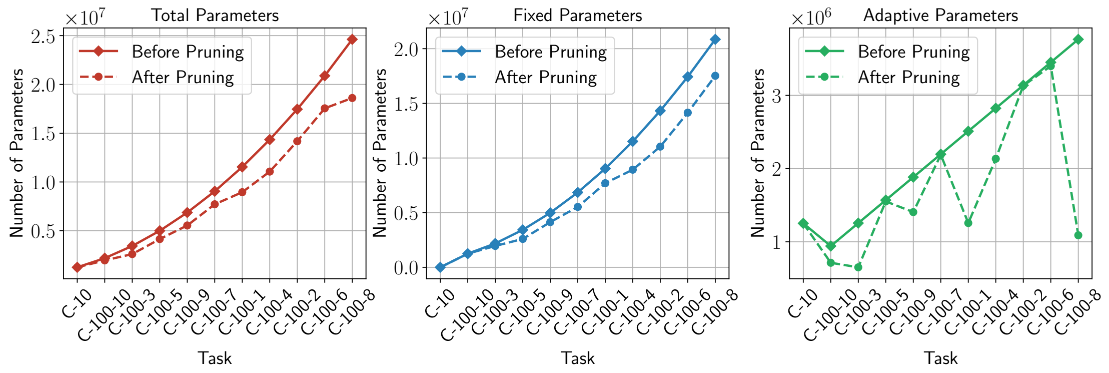

Figure 5.6: Total number of parameters in the model as a function of tasks. **Left.** The total number of parameters in the model as a function of tasks before and after pruning. **Centre.** The number of fixed parameters (parameters in preceding progressive blocks) as a function of tasks before and after pruning. **Right.** The number of adaptive parameters (parameters in the progressive block being trained) as a function of tasks before and after pruning.

number of tasks increases. The amount of pruned parameters in each progressive block varied, as shown in Figure 5.6 (right), which is likely due to the layer-wise transferability of features between tasks or task relatedness.

One of the main motivations behind the proposed greedy layer-wise pruning procedure was that initial layers were more prone to pruning than final layers, as progressive blocks can leverage features learned in initial layers in prior progressive blocks more than final layers as features in initial layers are usually more generic. This can be seen in Figure 5.7, which is a plot of the amount of pruned parameters per layer averaged across all progressive blocks.

Further discussion is provided in Section 5.6.

5.5 Experiments in Speech Recognition

Progressive learning is evaluated in the speech recognition domain using four supervised classification tasks, specifically Automatic Speech Recognition (ASR), Speech Emotion Recognition (SER), Gender Recognition (GR), and Speaker Recognition (SR), using the TIMIT, IEMOCAP, and eNTERFACE datasets. This is a challenging setting since the relationship between these tasks is not well understood [Fayek et al., 2016a], and moreover, the datasets used in these tasks were collected under different conditions. Section 5.5.1 details the experimental setup and experiments carried out. Section 5.5.2 outlines the evaluation criteria used. Section 5.5.3 presents the results and observations.

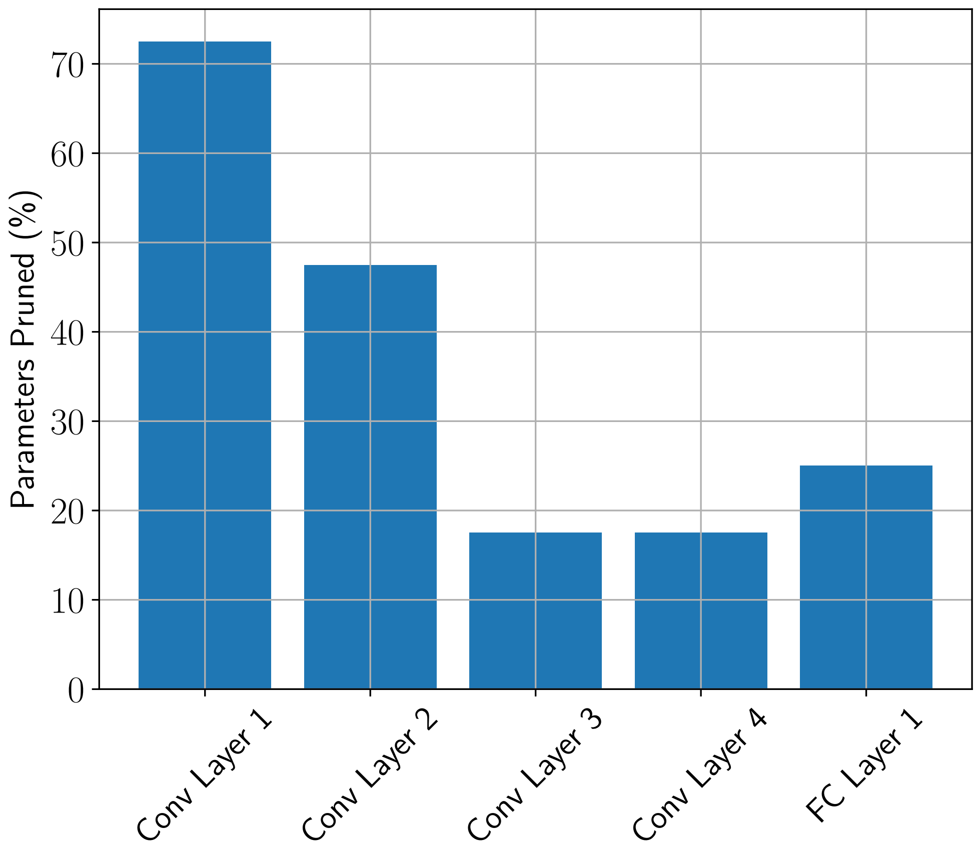

Figure 5.7: Percentage of pruned weights as a function of layers. It can be noted that initial layers are more prone to pruning as progressive blocks can rely on features learned in initial layers in prior progressive blocks.

5.5.1 Experimental Setup

Datasets. The TIMIT dataset [Garofolo et al., 1993] (see Appendix A.6 for more details) was used for the ASR and GR tasks. The complete 462-speaker training set, without the dialect (SA) utterances, was used as the training set. The 50-speaker development set was used as the validation set. The 24-speaker core test set was used as the test set.

In the case of the ASR task, the 61 phonemes were mapped into 48 phonemes for training, which were then mapped into 39 phonemes for scoring, as commonly practised in the literature [Lee and Hon, 1989]. The Kaldi toolkit [Povey et al., 2011] was used to produce force-aligned frame labels by training a mono-phone Gaussian Mixture Model (GMM)-Hidden Markov Model (HMM) system with Mel Frequency Cepstral Coefficients (MFCCs).

In the case of the GR task, a voice activity detector was used to detect and remove silent frames. The ratio between male and female speakers was balanced by randomly omitting utterances from male speakers.

The IEMOCAP dataset [Busso et al., 2008] (see Appendix A.3 for more details) was used for the SER task. Utterances from the first two sessions were used as the training set. The fifth session was used as the validation set to be consistent with experiments carried out in Chapter 3 and Chapter 4. Utterances from the third and fourth sessions were used as the test set. Since the data was labelled at an utterance level, all frames in an utterance inherited the utterance label. A voice activity detector was then used to label silent frames and *silence* was added as an additional class to the four emotion classes, i.e., a frame has either the same label as its parent utterance or the *silence* label. Silent frames were retained in this task as *silence* can be a strong cue for SER.

The eNTERFACE dataset [Martin et al., 2006] (see Appendix A.2 for more details) was used for the SR task. The training, validation, and test sets were constructed by randomly selecting utterances corresponding to 70%, 15%, and 15% respectively of all data from 40 speakers, while the remaining four speakers were discarded due to inconsistency in their data. Similar to GR, a voice activity detector was used to detect and remove silent frames.

Note that all sets in all tasks are mutually exclusive, except for the ASR and GR tasks, where the training, validation, and test sets overlap between both tasks, i.e., the training set for the ASR is similar to the training set for the GR task, however, the training, validation, and test sets are mutually exclusive in both tasks.

Preprocessing. For all datasets and tasks, utterances were split into 25 ms frames with a stride of 10 ms, and a Hamming window was applied, then 40 log-Mel Frequency Spectral Coefficients (MFSCs) were extracted from each frame. The mean and standard deviation were normalized per coefficient to zero and one respectively using the mean and standard deviation computed on each training set only. No speaker dependent operations were performed. An input data exemplar was composed of 41 consecutive frames with the exemplar label being the label of the middle frame in the case of ASR and SER or the utterance label in the case of GR and SR.

Model. The ASR system had a hybrid ConvNet-HMM architecture, as described in Section 3.3. A ConvNet acoustic model was used to produce a probability distribution over the states of three-state HMMs with a bi-gram language model estimated from the training set. The other three tasks, namely SER, GR, and SR, had an acoustic model only.

The ConvNet acoustic model used in all four tasks is detailed in Table 5.4. The model is similar to the model used in Section 5.4 for the image recognition tasks.

Training and Implementation Details. The parameters of the convolutional and fully connected layers were initialized randomly, sampling from a Gaussian distribution, with zero mean and $\sqrt{2/n_i}$ and $1/\sqrt{n_i}$ standard deviation for convolutional and fully connected layers respectively, where n_i is the number of inputs to the layer. All layers were regularized using weight decay with penalty $\lambda = 0.001$ and dropout with drop probability r as indicated in Table 5.4. ADAM was used to optimize the parameters with respect to a negative log-likelihood loss function using batch size $M_b = 256$, learning rate $\alpha = 0.001$, first moment $\beta_1 = 0.99$, and second moment $\beta_2 = 0.999$. The model was trained for a minimum of 110×10^3 iterations and a maximum of 250×10^3 iterations. The validation set was used for early stopping.

The experiments were implemented using TensorFlow [Abadi et al., 2016]. Training was carried out on NVIDIA Tesla P100 GPUs. Training a single model from scratch for a single task on a single GPU required a wall time of approximately 7 hours.

Progressive Learning. In the case of progressive learning, initially, a model that has the same architecture and training recipe outlined above was trained on the first task. Afterwards, an additional progressive block was added for each subsequent task. Each progressive block had the same architecture with half the number of features maps / units listed in Table 5.4 in each layer; the outputs of the layer before a convolutional or fully connected layer in all prior blocks were concatenated and fed as input to that convolutional or fully connected layer, as marked *[Concatenation]* in Table 5.4. In each progression procedure, the parameters of the new progressive block were randomly initialized and trained using the same training recipe as mentioned above. Note that parameters in a progressive block

Table 5.4: Convolutional neural network architecture for the speech recognition tasks. The concatenation operation indicates the layers at which the output of the previous layer in all prior blocks are concatenated. The concatenation operation can be ignored in independent learning. K denotes the number of output classes.

№	**Type**	**Size**	**Other**
1	Convolution	$64, 6 \times 5$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	[Concatenation]	—	—
2	Convolution	$64, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride $= 2$
	Dropout	—	$r = 0.25$
	[Concatenation]	—	—
3	Convolution	$128, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	[Concatenation]	—	—
4	Convolution	$128, 3 \times 3$	Stride $= 1$
	BatchNorm	—	—
	ReLU	—	—
	Max Pooling	2×2	Stride $= 2$
	Dropout	—	$r = 0.25$
	[Concatenation]	—	—
5	Fully Connected	1024	—
	BatchNorm	—	—
	ReLU	—	—
	Dropout	—	$r = 0.5$
	[Concatenation]	—	—
6	Fully Connected	K	—
	Softmax	—	—

are held constant once the block is trained and pruned, including during training and pruning subsequent progressive blocks.

5.5.2 Evaluation Criteria

The standard evaluation criteria varied among tasks. The phone accuracy rate ($100-$ Phone Error Rate (PER)) was used for the ASR task, whereas the classification accuracy was used for the SER, GR, and SR tasks. Additionally, the learning speed, that is the model performance as a function of training iterations, of tasks learned with progressive learning versus the same tasks learned independently is monitored and reported.

5.5.3 Results

Baseline. Baseline models were trained for each task independently, in that four models were trained for the four speech recognition tasks. Figure 5.8 plots the test phone accuracy rate for the ASR task, and test accuracy for the SER, GR, and SR tasks.

Progressive Learning. Progressive learning was used to learn the four tasks sequentially, employing the curriculum, progression, and pruning procedures. The curriculum procedure used the relative improvement of the validation accuracy over random guessing to score the classifiers trained at each curriculum procedure. The relative improvement was preferred to the standard validation accuracy as the number of classes varied between tasks. Once a task was chosen, a progression procedure was employed. This was followed by pruning, where the pruning amount q was varied in 25% intervals, i.e., the pruning amount $q \in \{25\%, 50\%, 75\%, 100\%\}$ for only layers that receive input from preceding progressive blocks. The validation set was used to monitor and evaluate the performance of the model in the pruning procedure.

The test phone accuracy rate for the ASR task and test accuracy for the remaining three tasks are plotted in Figure 5.8. It can be seen from the results in Figure 5.8 that progressive learning leads to comparable performance with respect to independent learning for all three tasks.

Figure 5.9 demonstrates the validation error as a function of training iterations, where it is shown that progressive learning converges faster than independent learning on most tasks.

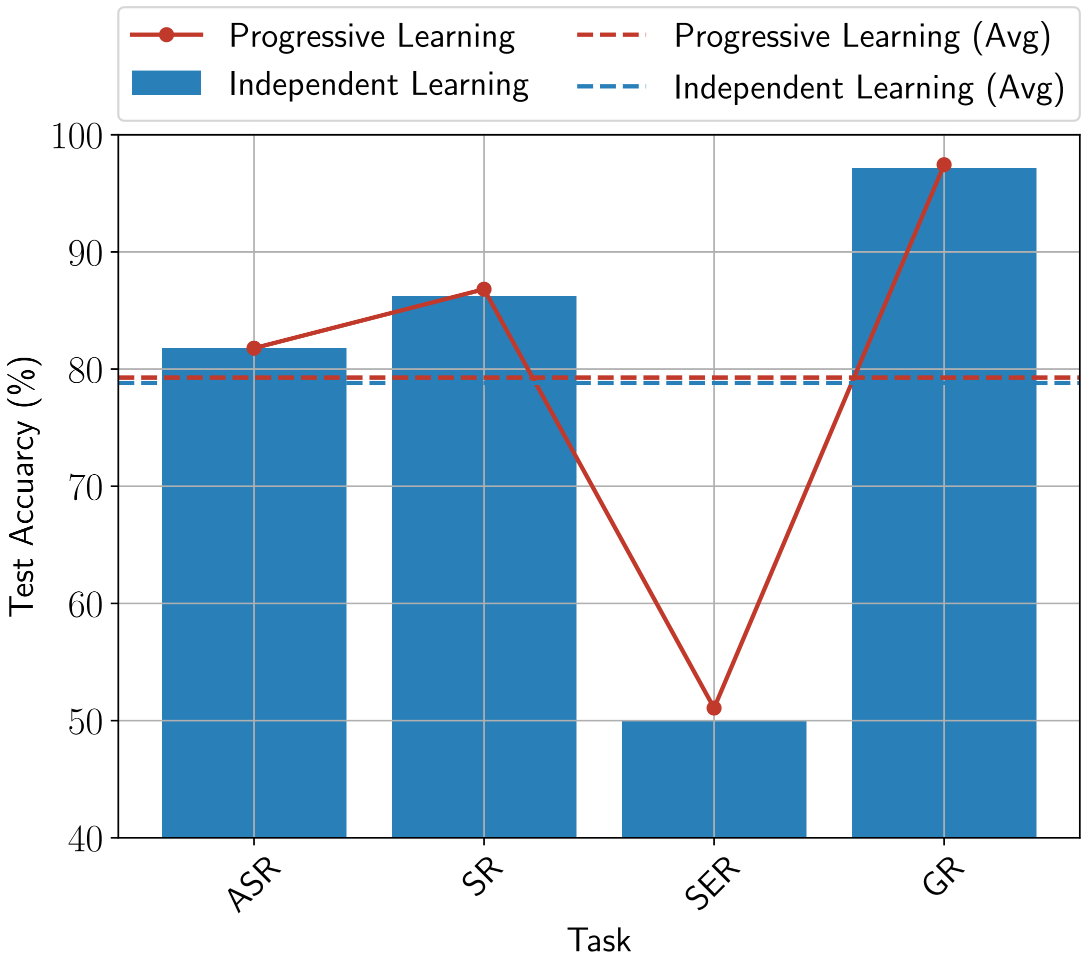

Figure 5.8: Progressive learning vs independent learning for all four speech recognition tasks. The tasks are ordered according to the outcome of the curriculum procedure.

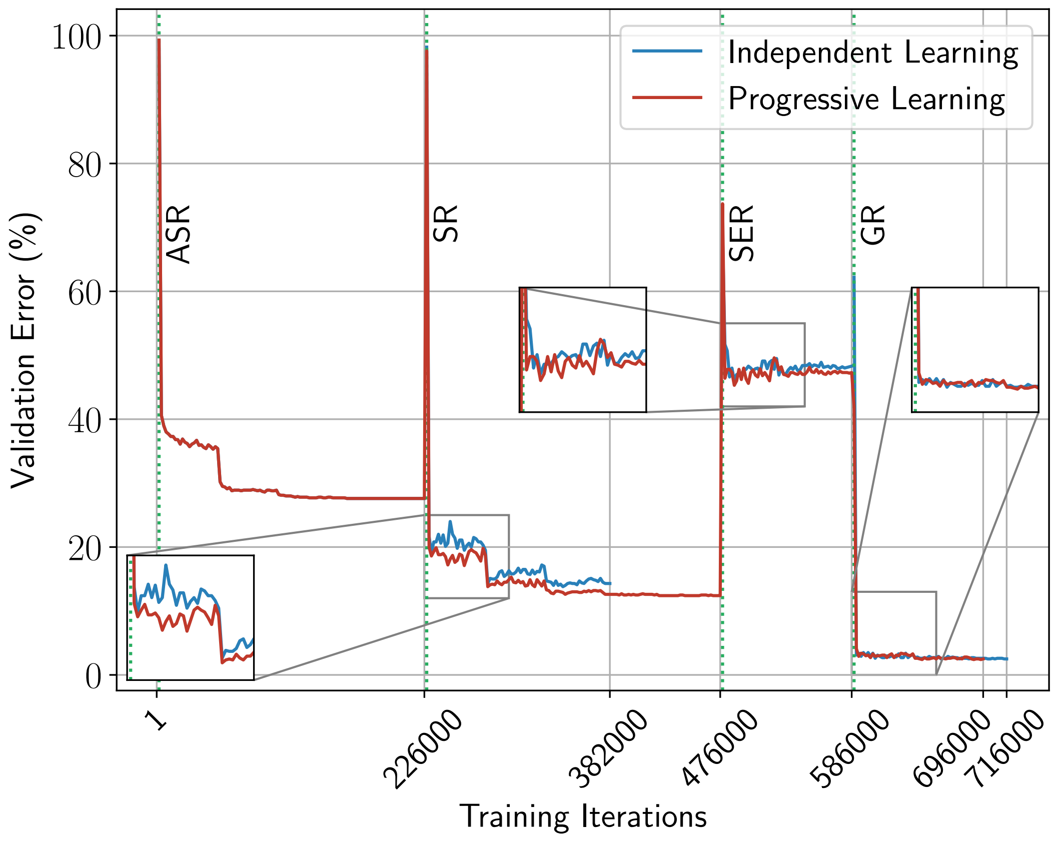

Figure 5.9: Learning curves, validation error as a function of training iterations, for progressive and independent learning for the four speech recognition tasks. Note that tasks are ordered according to the outcome of the curriculum strategy. Models are reset to random initialization at the beginning of each task in the case of independent learning.

5.6 Discussion

Continual Learning. Progressive learning is a framework for continual learning that aims to address the following desiderata, as mentioned in Section 5.1: (1) a strategy to select the next task to learn from a pool of candidate tasks to facilitate learning; (2) a mechanism to efficiently grow the capacity of the model to accommodate learning new tasks (if necessary); and, (3) a method to accumulate, maintain, and utilize knowledge to learn future tasks without significant adverse effects on the learned tasks.

In this work, the aforementioned three desiderata were addressed using three simple yet effective procedures: curriculum, progression, and pruning. These procedures can be regarded as modular building blocks for the progressive learning framework that may be employed individually or in combination as required. Other procedures can be added to progressive learning to address further desiderata of continual learning.

Curriculum. A structured order to learning new concepts and tasks is an important aspect of human learning. It is also beneficial in machine learning [Elman, 1993], and it was shown that it can lead to faster learning, and in some cases, better generalization [Bengio et al., 2009, Graves et al., 2017].

The curriculum procedure is an important component of progressive learning, where the learner can choose the most representative task to learn next with respect to the state of knowledge at the time, which leads to a natural development in task difficulty over time. For the tasks studied herein, the curriculum procedure led to better generalization as shown in Tables 5.2 and 5.3.

Progression. The progression procedure is the main procedure in progressive learning that has the largest contribution in terms of performance as shown in Tables 5.2 and 5.3. The progression procedure increases the capacity of the model by adding new parameters, and provides a mechanism for forward knowledge transfer, i.e., use knowledge learned in prior tasks to facilitate the learning and execution of new tasks, by leveraging existing parameters learned in prior tasks, all while sidestepping catastrophic forgetting, as only newly added parameters are trained.

Existing model parameters are leveraged by concatenating the output of the preceding layer in all prior progressive blocks as well as the current block and feeding the concatenated output to the subsequent layer in the block. The use of the concatenation operation, as opposed to other operations, encourages the reuse of features, and also leads to easier optimization, as pointed out in [Huang et al., 2017].

Growing the capacity of the model by adding new parameters is important in the case when the new task is not related to any of the prior tasks, which would allow progressive learning to be reduced to independent learning via pruning parameters between progressive blocks.

Pruning. The pruning procedure in progressive learning offers many advantages. It was initially conceived to counteract the growth in the number of parameters as a result of the progression procedure. Nevertheless, it was also found to combat over-fitting and negative forward transfer, which can lead to an improvement in performance as shown in Tables 5.2 and 5.3. To this end, pruning was favoured to incremental growing, where the parameters would grow incrementally in the progression procedure.

Layer-wise greedy pruning was built on the intuition that features in initial layers in deep networks are more generic and reusable across tasks. It is envisioned that as the number of tasks increases and with a diverse enough set of tasks, pruning would be able to eliminate most initial layers in new progressive blocks.

Lifelong Learning. Progressive learning can be regarded as a form of lifelong learning, which requires three criteria [Chen and Liu, 2016]: a continuous learning process, knowledge accumulation and maintenance in a knowledge base, and the ability to use the knowledge base for learning future tasks. Progression can be regarded as the continuous learning process. Similarly, the features learned in prior progressive blocks can be regarded as the knowledge base. Hence, the connections between a new progressive block and all prior progressive blocks define the ability to use the knowledge base (features) for learning the new task and future ones.

Limitations. The current progression procedure provides a mechanism for forward knowledge transfer. One of the limitations of this work is the absence of a mechanism to allow backward knowledge transfer, i.e., transfer knowledge obtained from a new task being learned to prior tasks already learned. It is envisioned that this could be achieved by modifying the progression procedure to cater for backward knowledge transfer in addition to forward knowledge transfer by incorporating methods introduced in [Kirkpatrick et al., 2017, Lopez-Paz and Ranzato, 2017]. This is an area for future research.

5.7 Summary

Continual learning is key to advancing machine learning, where experiences and knowledge can be accumulated and used across tasks. Progressive learning is a deep learning framework that addresses the identified continual learning desiderata using three procedures: curriculum, progression, and pruning. The novel contribution of progressive learning is the construction and formulation of a deep learning framework that allows a holistic and systematic approach to continual learning.

Progressive learning was evaluated using supervised classification tasks in the image recognition and speech recognition domains. It was shown that progressive learning outperforms independent learning, transfer learning, and multi-task learning in terms of overall performance as well as learning speed, when tasks are related, by accumulating and leveraging knowledge learned across tasks in a continuous manner.

It is envisioned that progressive learning is a step towards a fully general continual learning framework.

Chapter 6

Conclusions and Future Work

THE work presented throughout the thesis has led to a number of observations and insights. This chapter highlights the main insights and conclusions of the thesis. The chapter also presents avenues for future research.

Outline. This chapter is structured as follows. Section 6.1 summarizes and concludes the thesis. Section 6.2 highlights avenues for future work.

6.1 Conclusions

The main aim of this thesis was the study and development of a structured approach to continual learning, building on the success of deep learning and neural networks. Continual learning extends machine learning beyond the single task paradigm into the realm of learning multiple tasks consecutively. Learning multiple tasks via continual learning allows machine learning to accumulate and leverage knowledge learned across tasks in a continuous manner. In doing so, one can expect a faster learning process that leads to better generalization using less amounts of data and a smaller number of dedicated parameters for new tasks that are relevant to prior tasks. Of course, continual learning presents its own set of challenges. Inducing the correct inductive bias and mitigating catastrophic forgetting in an efficient manner are two of these challenges that this work was set to address.

The thesis commenced with developing understanding, insights, and baselines for a number of independent classification tasks, i.e., image recognition, Automatic Speech Recognition (ASR), and Speech Emotion Recognition (SER), as well as their respective datasets, network architectures, and best practices. Results reported for the image recognition task using the CIFAR-10, CIFAR-100, and SVHN datasets, as well as the results reported for the ASR task using the TIMIT dataset, were on par with the state-of-the-art. The SER task was explored in greater depth using the IEMOCAP dataset and used as a test bed to explore various neural network architectures. As a result of this exploration, state-of-the-art results were reported on the IEMOCAP dataset for speaker-independent SER. More importantly, the models devised for all of these tasks were somewhat similar in that deep multi-layered neural networks with similar architectures and training algorithms were used across all of the tasks.

Subsequently, the relation between the systems developed for the aforementioned tasks was investigated. To this end, a methodology for understanding the layer-wise relevance of features in deep networks between two tasks, denoted gradual transfer learning, was proposed. The gradual transfer learning methodology can be used for understanding the relation between two tasks, which can aid in designing and implementing systems that aim to address multiple tasks. The gradual transfer learning methodology illuminates the relevance of each layer of features in a deep network trained for one task to another task via transfer learning. The gradual transfer learning methodology was explored on a number of datasets and tasks in the image recognition and speech recognition domains. Consistent behaviour emerged that reflected the nature of the datasets or tasks involved, such that the specificity and transferability of the features in deep networks was found to be positively correlated with the depth of the layer.

Finally, progressive learning, a deep learning framework for continual learning that comprises three procedures: curriculum, progression, and pruning, was formulated and presented. Progressive learning was evaluated in the image recognition and speech recognition domains, where it was shown that progressive learning outperforms independent learning, as well as other paradigms that incorporate learning multiple tasks, including, transfer learning and multi-task learning, in terms of learning speed and overall performance when tasks are related, by accumulating and leveraging knowledge learned across tasks in a continuous manner. Progressive learning address two main challenges in continual learning: inducing the correct inductive bias and mitigating catastrophic forgetting.

Revisiting the main proposition of the thesis, continual learning is key to advancing machine learning and Artificial Intelligence (AI). It is envisioned that progressive learning is a step towards a fully general continual learning framework.

The contributions of the thesis are as follows. A systemic study of multiple tasks in machine perceptions, namely, image recognition, ASR, and particularly an in-depth exploration of SER. A methodology for understanding the relation between two tasks using the features learned for each task in a deep network to illuminate the relevance of each layer of features in the multi-layered neural network trained for one task to the other task via transfer learning. A deep learning framework that allows a holistic and systematic approach to continual learning.

6.2 Future Work

Future work can be pursued in numerous directions.

On the continual learning front, first, the investigation of continual learning in unrestricted domains is of notable interest. The utility of the application of continual learning to a diverse set of potentially unrelated tasks, e.g., machine translation and self-driving cars, remains questionable. Second, the continual learning methods studied herein focus on forward knowledge transfer, i.e., leverage knowledge accumulated from prior tasks to learn and perform the new task, whereas, backward knowledge transfer, i.e., leverage knowledge obtained from a new task to improve the performance of prior tasks, may also be beneficial. Third, the computational efficiency of progressive learning in dealing with a very large number of tasks needs to be further studied.

From a broader outlook on machine learning, first, the application of deep learning requires a plethora of design decisions and extensive tuning of hyperparameters to ensure optimal performance. While some of these design decisions were found to be consistent among the class of tasks studied in this work, these design decisions and hyperparameters still lack an automatic or systematic way of tuning. To allow continual learning to expand to truly diverse domains, such problems need to be addressed. Finally, the most elegant learning algorithm is an algorithm that is not only able to learn from experience, but is also able to learn the learning algorithm, and learn the learning algorithm of the learning algorithm, and learn the learning algorithm of the learning algorithm of the learning algorithm, etc., in an endless recursive manner. This meta-learning algorithm may not necessarily be a continual learning algorithm, but it is their unification that presents the most compelling avenue for future work.

Bibliography

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, volume 16, pages 265–283. 45, 53, 77, 85, 104, 116

[Abdel-Hamid et al., 2014] Abdel-Hamid, O., Mohamed, A.-R., Jiang, H., Deng, L., Penn, G., and Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545. 22, 47

[Allen, 1977] Allen, J. (1977). Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):235–238. 47

[Amodei et al., 2016] Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., and Others (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning (ICML)*, pages 173–182, New York, NY, USA. 47

[Arias et al., 2013] Arias, J. P., Busso, C., and Yoma, N. B. (2013). Energy and F0 contour modeling with functional data analysis for emotional speech detection. In *Interspeech*, pages 2871–2875. 55

[Ayadi et al., 2011] Ayadi, M. E., Kamel, M. S., and Karray, F. (2011). Survey on speech emotion recognition: Features, classification schemes, and databases. *Pattern Recognition*, 44(3):572–587. 55

[Barber, 2012] Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press. 10

[Baxter, 2000] Baxter, J. (2000). A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198. 5

[Bengio, 2012] Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *International Conference on Machine Learning (ICML) Workshop on Unsupervised and Transfer Learning*, pages 17–36. 72, 73

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(8):1798–1828. 3

[Bengio et al., 2009] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *International Conference on Machine Learning (ICML)*, pages 41–48, New York, NY, USA. ACM. 94, 95, 121

[Bishop, 1995] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press. 2, 13

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. 10, 13

[Boole, 1854] Boole, G. (1854). *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*. Macmillan. 2

[Bordes et al., 2015] Bordes, A., Usunier, N., Chopra, S., and Weston, J. (2015). Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*. 22

[Bottou, 2004] Bottou, L. (2004). *Stochastic Learning*, pages 146–168. Springer Berlin Heidelberg. 18

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth. 12

[Busso et al., 2008] Busso, C., Bulut, M., Lee, C.-C., Kazemzadeh, A., Mower, E., Kim, S., Chang, J., Lee, S., and Narayanan, S. (2008). IEMOCAP: interactive emotional dyadic motion capture database. *Language Resources and Evaluation*, 42(4):335–359. 6, 58, 81, 115, 149

[Caruana, 1997] Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1):41–75. 33, 72

[Chen and Mak, 2015] Chen, D. and Mak, B. K. (2015). Multitask learning of deep neural networks for low-resource speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(7):1172–1183. 33

[Chen and Liu, 2016] Chen, Z. and Liu, B. (2016). *Lifelong Machine Learning*. Morgan & Claypool Publishers. 33, 122

[Chorowski et al., 2015] Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems (NIPS)*, pages 577–585. 22, 54

[Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research (JMLR)*, 12(Aug):2493–2537. 33

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine learning*, 20(3):273–297. 12, 21

[Cowie et al., 2001] Cowie, R., Douglas-Cowie, E., Tsapatsoulis, N., Votsis, G., Kollias, S., Fellenz, W., and Taylor, J. (2001). Emotion recognition in human-computer interaction. *IEEE Signal Processing Magazine*, 18(1):32–80. 39, 54

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314. 23

[Dahl et al., 2012] Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42. 46, 47

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE. 40

[Dauphin et al., 2015] Dauphin, Y., de Vries, H., and Bengio, Y. (2015). Equilibrated adaptive learning rates for non-convex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1504–1512. 17, 19

[Dauphin et al., 2014] Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2933–2941. Curran Associates, Inc. 17

[Deng et al., 2013] Deng, J., Zhang, Z., Marchi, E., and Schuller, B. (2013). Sparse autoencoder-based feature transfer learning for speech emotion recognition. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 511–516. 72

[Dewar and Xu, 2010] Dewar, K. M. and Xu, F. (2010). Induction, overhypothesis, and the origin of abstract knowledge: Evidence from 9-month-old infants. *Psychological Science*, 21(12):1871–1877. 4

[Elman, 1993] Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71–99. 95, 121

[Etienne et al., 2018] Etienne, C., Fidanza, G., Petrovskii, A., Devillers, L., and Schmauch, B. (2018). Speech emotion recognition with data augmentation and layer-wise learning rate adjustment. *arXiv preprint arXiv:1802.05630*. 68

[Eyben et al., 2015] Eyben, F., Huber, B., Marchi, E., Schuller, D., and Schuller, B. (2015). Real-time robust recognition of speakers' emotions and characteristics on mobile platforms. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 778–780. 55

[Eyben et al., 2016] Eyben, F., Scherer, K. R., Schuller, B. W., Sundberg, J., André, E., Busso, C., Devillers, L. Y., Epps, J., Laukka, P., Narayanan, S. S., and Truong, K. P. (2016). The geneva minimalistic acoustic parameter set (gemaps) for voice research and affective computing. *IEEE Transactions on Affective Computing*, 7(2):190–202. 55

[Fayek, 2016] Fayek, H. M. (2016). A deep learning framework for hybrid linguistic-paralinguistic speech systems. In *2nd Doctoral Consortium at Interspeech 2016*, pages 1–2, Berkeley, CA, United States. 8, 71

[Fayek, 2017] Fayek, H. M. (2017). MatDL: A lightweight deep learning library in MATLAB. *The Journal of Open Source Software*, 2(19):413. 7, 8, 38, 53, 59, 85

[Fayek et al., 2018] Fayek, H. M., Cavedon, L., and Wu, H. R. (2018). On the transferability of representations in neural networks between datasets and tasks. In *Continual Learning Workshop, Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, QC, Canada. 8, 71, 98

[Fayek et al., 2015] Fayek, H. M., Lech, M., and Cavedon, L. (2015). Towards real-time speech emotion recognition using deep neural networks. In *International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–5. 8, 13, 38

[Fayek et al., 2016a] Fayek, H. M., Lech, M., and Cavedon, L. (2016a). Modeling subjectiveness in emotion recognition with deep neural networks: Ensembles vs soft labels. In *International Joint Conference on Neural Networks (IJCNN)*, pages 566–570. 8, 38, 56, 113

[Fayek et al., 2016b] Fayek, H. M., Lech, M., and Cavedon, L. (2016b). On the correlation and transferability of features between automatic speech recognition and speech emotion recognition. In *Interspeech*, pages 3618–3622. 8, 71

[Fayek et al., 2017] Fayek, H. M., Lech, M., and Cavedon, L. (2017). Evaluating deep learning architectures for speech emotion recognition. *Neural Networks*, 92:60–68. Advances in Cognitive Engineering Using Neural Networks. 6, 8, 22, 38, 54, 70

[Fernandez, 2004] Fernandez, R. (2004). *A computational model for the automatic recognition of affect in speech*. PhD thesis, Massachusetts Institute of Technology. 40

[Garofolo et al., 1993] Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., Dahlgren, N., and Zue, V. (1993). Timit acoustic-phonetic continuous speech corpus. *Linguistic Data Consortium*, 93. 49, 50, 80, 115, 150, 152

[Glorot et al., 2011a] Glorot, X., Bordes, A., and Bengio, Y. (2011a). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 24, 26

[Glorot et al., 2011b] Glorot, X., Bordes, A., and Bengio, Y. (2011b). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *International Conference on Machine Learning (ICML)*, pages 513–520. 73

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. 2, 12, 13, 23

[Goodfellow et al., 2015] Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*. 23

[Goodfellow et al., 2013] Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *International Conference on Machine Learning (ICML)*, volume 28, pages 1319–1327, Atlanta, GA, USA. 25, 41, 77

[Graves, 2008] Graves, A. (2008). *Supervised Sequence Labelling with Recurrent Neural Networks*. PhD thesis, Technische Universitat Munchen. 14, 30

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*. 14

[Graves et al., 2017] Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. (2017). Automated curriculum learning for neural networks. In *International Conference on Machine Learning (ICML)*, volume 70, pages 1311–1320, Sydney, NSW, Australia. 95, 121

[Graves et al., 2013] Graves, A., Mohamed, A.-R., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6645–6649. 22, 32, 47

[Han et al., 2014] Han, K., Yu, D., and Tashev, I. (2014). Speech emotion recognition using deep neural network and extreme learning machine. In *Interspeech*. 55, 68

[Han et al., 2016] Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*. 96, 102

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1135–1143, Cambridge, MA, USA. MIT Press. 94, 96, 97

[Harlow, 1949] Harlow, H. F. (1949). The formation of learning sets. *Psychological Review*, 56(1):51–65. 4

[Hartley and Zisserman, 2004] Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition. 39

[Hassibi and Stork, 1993] Hassibi, B. and Stork, D. G. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NIPS)*, pages 164–171. 96

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *International Conference on Computer Vision (ICCV)*, pages 1026–1034, Washington, DC, USA. IEEE Computer Society. 22, 26, 42, 53, 59, 77, 82, 104

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. 30

[Hermann et al., 2015] Hermann, K. M., Kočiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1693–1701, Cambridge, MA, USA. MIT Press. 22

[Hinton, 2012] Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer. 21

[Hinton et al., 2012] Hinton, G. E., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97. 22

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554. 3, 21, 22

[Hochreiter, 1991] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. Master's thesis, Technische Universität München. 17, 19, 21

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116. 17, 19, 21

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780. 31

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366. 23

[Hou et al., 2018] Hou, S., Pan, X., Loy, C. C., Wang, Z., and Lin, D. (2018). Lifelong learning via progressive distillation and retrospection. In *European Conference on Computer Vision (EECV)*, pages 452–467. Springer. 36

[Huang et al., 2017] Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. (2017). Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 42, 45, 77, 97, 101, 121

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456. JMLR.org. 19, 20

[Isele and Cosgun, 2018] Isele, D. and Cosgun, A. (2018). Selective experience replay for lifelong learning. In *AAAI Conference on Artificial Intelligence*. 37

[Jaitly and Hinton, 2011] Jaitly, N. and Hinton, G. (2011). Learning a better representation of speech soundwaves using restricted boltzmann machines. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5887. 47

[Jaitly and Hinton, 2013] Jaitly, N. and Hinton, G. E. (2013). Vocal tract length perturbation (vtlp) improves speech recognition. In *International Conference on Machine Learning (ICML) Workshop on Deep Learning for Audio, Speech and Language*. 13

[Johnson et al., 2017] Johnson, J., Hariharan, B., van der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., and Girshick, R. (2017). Inferring and executing programs for visual reasoning. In *International Conference on Computer Vision (ICCV)*. 22

[Jurafsky and Martin, 2014] Jurafsky, D. and Martin, J. H. (2014). *Speech and language processing*. Pearson London. 49

[Katz, 1987] Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401. 49

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*. 17, 19

[Kirkpatrick et al., 2017] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526. 36, 122

[Klambauer et al., 2017] Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 971–980. Curran Associates, Inc. 25

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto. 40, 75, 103, 146

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances* *in Neural Information Processing Systems (NIPS)*, pages 1097–1105. Curran Associates, Inc. 13, 22, 40, 69

[Lake et al., 2017] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40. 4, 32, 93

[LeCun, 1986] LeCun, Y. (1986). Learning process in an asymmetric threshold network. In *Disordered Systems and Biological Organization*, pages 233–240. Springer Berlin Heidelberg. 26

[LeCun and Bengio, 1995] LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. In *The Handbook of Brain Theory and Neural Networks*, pages 255–258. MIT Press, Cambridge, MA, USA. 28

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444. Insight. 3, 22, 40

[LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 396–404. 28, 39, 94, 96

[LeCun et al., 1998] LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer. 17

[Lee et al., 2011] Lee, C.-C., Mower, E., Busso, C., Lee, S., and Narayanan, S. (2011). Emotion recognition using a hierarchical binary decision tree approach. *Speech Communication*, 53(9):1162–1171. Sensing Emotion and Affect - Facing Realism in Speech Processing. 68

[Lee et al., 2015] Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. (2015). Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, pages 498–515. Springer. 26

[Lee and Hon, 1989] Lee, K.-F. and Hon, H.-W. (1989). Speaker-independent phone recognition using hidden markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(11):1641–1648. 50, 81, 115, 152

[Li and Sim, 2010] Li, B. and Sim, K. C. (2010). Comparison of discriminative input and output transformations for speaker adaptation in the hybrid nn/hmm systems. In *Interspeech*, pages 526–529. 72

[Li et al., 2013] Li, L., Zhao, Y., Jiang, D., Zhang, Y., Wang, F., Gonzalez, I., Valentin, E., and Sahli, H. (2013). Hybrid deep neural network–hidden markov model (dnn-hmm) based speech emotion recognition. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 312–317. 55

[Li and Hoiem, 2016] Li, Z. and Hoiem, D. (2016). Learning without forgetting. In *European Conference on Computer Vision (ECCV)*, pages 614–629. Springer. 36

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440. 41, 77

[Lopez-Paz and Ranzato, 2017] Lopez-Paz, D. and Ranzato, M. A. (2017). Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6467–6476. Curran Associates, Inc. 36, 106, 122

[Louizos et al., 2018] Louizos, C., Welling, M., and Kingma, D. P. (2018). Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations (ICLR)*. 96

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110. 40

[Maas et al., 2013] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML) Workshop on Deep Learning for Audio, Speech and Language Processing*. 25

[Mao et al., 2014] Mao, Q., Dong, M., Huang, Z., and Zhan, Y. (2014). Learning salient features for speech emotion recognition using convolutional neural networks. *IEEE Transactions on Multimedia*, 16(8):2203–2213. 56

[Mariooryad and Busso, 2013] Mariooryad, S. and Busso, C. (2013). Exploring cross-modality affective reactions for audiovisual emotion recognition. *IEEE Transactions on Affective Computing*, 4(2):183–196. 58, 68, 149

[Martin et al., 2006] Martin, O., Kotsia, I., Macq, B., and Pitas, I. (2006). The enterface'05 audio-visual emotion database. In *International Conference on Data Engineering Workshops*, Washington, DC, USA. IEEE Computer Society. 115, 149

[McCloskey and Cohen, 1989] McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165. 5, 33, 36, 93

[McCorduck, 2004] McCorduck, P. (2004). *Machines Who Think*. AK Peters Ltd. 2

[McCorduck et al., 1977] McCorduck, P., Minsky, M., Selfridge, O. G., and Simon, H. A. (1977). History of artificial intelligence. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 951–954. 2

[Mikolov et al., 2018] Mikolov, T., Joulin, A., and Baroni, M. (2018). A roadmap towards machine intelligence. In *Computational Linguistics and Intelligent Text Processing*, pages 29–61. Springer. 4, 32, 93

[Miller, 1995] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41. 146, 150

[Mishkin and Matas, 2015] Mishkin, D. and Matas, J. (2015). All you need is a good init. In *International Conference on Learning Representations (ICLR)*. 17

[Misra et al., 2016] Misra, I., Shrivastava, A., Gupta, A., and Hebert, M. (2016). Cross-stitch networks for multi-task learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3994–4003. 33, 73

[Mitchell et al., 2018] Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., and Welling, J. (2018). Never-ending learning. *Communications of the ACM*, 61(5):103–115. 4, 36

[Mitchell, 1980] Mitchell, T. M. (1980). The need for biases in learning generalizations. Technical report, CS Tech Report CBM-TR-117, Computer Science Department, Rutgers University. 4, 5

[Mitchell, 1997] Mitchell, T. M. (1997). Machine learning. 2, 3, 9

[Mohamed et al., 2012] Mohamed, A., Dahl, G., and Hinton, G. (2012). Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22. 46, 47, 69

[Mohamed, 2014] Mohamed, A.-R. (2014). *Deep Neural Network Acoustic Models for ASR*. PhD thesis, University of Toronto. 46, 47

[Montana and Davis, 1989] Montana, D. J. and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, volume 89, pages 762–767. 26

[Netzer et al., 2011] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems (NIPS) workshop on deep learning and unsupervised feature learning*. 41, 75, 150

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359. 32, 33, 72

[Petta et al., 2011] Petta, P., Pelachaud, C., and Cowie, R. (2011). *Emotion-Oriented Systems: The Humaine Handbook*. Springer, 1st edition. 55

[Poria et al., 2017] Poria, S., Peng, H., Hussain, A., Howard, N., and Cambria, E. (2017). Ensemble application of convolutional neural networks and multiple kernel learning for multimodal sentiment analysis. *Neurocomputing*, 261:217–230. 68

[Povey et al., 2011] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Vesely, K. (2011). The kaldi speech recognition toolkit. In *Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society. IEEE Catalog No.: CFP11SRW-USB. 50, 59, 81, 115, 152

[Rabiner and Juang, 1993] Rabiner, L. R. and Juang, B.-H. (1993). *Fundamentals of speech recognition*, volume 14. PTR Prentice Hall Englewood Cliffs. 39, 46

[Ratcliff, 1990] Ratcliff, R. (1990). Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308. 5, 33, 36, 93

[Razavian et al., 2014] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 512–519. 33, 72, 73

[Rebuffi et al., 2017] Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2017). Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems (NIPS)*, pages 506–516. 36

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788. 23

[Reed, 1993] Reed, R. (1993). Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747. 96

[Refaeilzadeh et al., 2009] Refaeilzadeh, P., Tang, L., and Liu, H. (2009). Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer. 58, 81

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 91–99. 22

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408. 3

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. 21, 27

[Rumelhart and McClelland, 1986] Rumelhart, D. E. and McClelland, J. L., editors (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations.* MIT Press, Cambridge, MA, USA. 3

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252. 39, 72, 149

[Russell and Norvig, 2003] Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach.* Pearson Education. 2, 10

[Rusu et al., 2016] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *CoRR*, abs/1606.04671. 36, 94, 95, 96, 97, 101, 107, 108, 109

[Ruvolo and Eaton, 2013a] Ruvolo, P. and Eaton, E. (2013a). Active task selection for lifelong machine learning. In *AAAI Conference on Artificial Intelligence*, pages 862–868. AAAI Press. 94, 95, 96

[Ruvolo and Eaton, 2013b] Ruvolo, P. and Eaton, E. (2013b). Ella: An efficient lifelong learning algorithm. In *International Conference on Machine Learning (ICML)*, pages 507–515. 33

[Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229. 2

[Schmidhuber, 2002] Schmidhuber, J. (2002). Hierarchies of generalized kolmogorov complexities and nonenumerable universal measures computable in the limit. *International Journal of Foundations of Computer Science*, 13(04):587–612. 2

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117. 21

[Schölkopf and Smola, 2001] Schölkopf, B. and Smola, A. J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT Press. 12

[Schuller et al., 2009a] Schuller, B., Steidl, S., and Batliner, A. (2009a). The interspeech 2009 emotion challenge. In *Interspeech*, pages 312–315. 60, 85

[Schuller et al., 2010] Schuller, B., Steidl, S., Batliner, A., Burkhardt, F., Devillers, L., Müller, C. A., and Narayanan, S. S. (2010). The interspeech 2010 paralinguistic challenge. In *Interspeech*, pages 2794–2797. 55

[Schuller et al., 2009b] Schuller, B., Vlasenko, B., Eyben, F., Rigoll, G., and Wendemuth, A. (2009b). Acoustic emotion recognition: A benchmark comparison of performances. In *IEEE Workshop on Automatic Speech Recognition Understanding (ASRU)*, pages 552–557. 55, 58

[See et al., 2017] See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1073–1083. Association for Computational Linguistics. 22

[Sercu et al., 2016] Sercu, T., Puhrsch, C., Kingsbury, B., and LeCun, Y. (2016). Very deep multilingual convolutional neural networks for lvcsr. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4955–4959. IEEE. 50, 82

[Sermanet et al., 2012] Sermanet, P., Chintala, S., and LeCun, Y. (2012). Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR)*, pages 3288–3291. IEEE. 41, 77

[Shah et al., 2014] Shah, M., Chakrabarti, C., and Spanias, A. (2014). A multimodal approach to emotion recognition using undirected topic models. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 754–757. 58, 68, 149

[Shin et al., 2017] Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2990–2999. 37

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. 50, 82

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*. 30

[Smith et al., 2002] Smith, L. B., Jones, S. S., Landau, B., Gershkoff-Stowe, L., and Samuelson, L. (2002). Object name learning provides on-the-job training for attention. *Psychological Science*, 13(1):13–19. 4

[Snoek et al., 2012] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2951–2959. Curran Associates, Inc. 23

[Srivastava et al., 2014] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15:1929–1958. 16

[Stuhlsatz et al., 2011] Stuhlsatz, A., Meyer, C., Eyben, F., ZieIke, T., Meier, G., and Schuller, B. (2011). Deep neural networks for acoustic emotion recognition: Raising the benchmarks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5688–5691. 55

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, pages 1139–1147. 18

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112. Curran Associates, Inc. 22

[Sutton et al., 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1057–1063. 16

[Swietojanski et al., 2012] Swietojanski, P., Ghoshal, A., and Renals, S. (2012). Unsupervised cross-lingual knowledge transfer in dnn-based lvcsr. In *IEEE Workshop on Spoken Language Technology (SLT)*, pages 246–251. 72

[Szeliski, 2010] Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media. 39

[Tahon and Devillers, 2016] Tahon, M. and Devillers, L. (2016). Towards a small set of robust acoustic features for emotion recognition: Challenges. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(1):16–28. 55

[Tajbakhsh et al., 2016] Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., and Liang, J. (2016). Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312. 72

[Taylor and Stone, 2009] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research (JMLR)*, 10(Jul):1633–1685. 32

[Terekhov et al., 2015] Terekhov, A. V., Montone, G., and O'Regan, J. K. (2015). *Knowledge Transfer in Deep Block-Modular Neural Networks*, pages 268–279. Springer. 36, 94, 95

[Thorburn, 1918] Thorburn, W. M. (1918). The myth of occam's razor. *Mind*, 27(107):345–353. 13

[Thrun, 1996] Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems (NIPS)*, pages 640–646, Cambridge, MA, USA. MIT Press. 33, 36, 93

[Thrun and Mitchell, 1995] Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25–46. 4, 33, 36, 93

[Tian et al., 2015a] Tian, L., Moore, J., and Lai, C. (2015a). Emotion recognition in spontaneous and acted dialogues. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 698–704. 56

[Tian et al., 2015b] Tian, L., Moore, J. D., and Lai, C. (2015b). Recognizing emotions in dialogues with acoustic and lexical features. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 737–742. 56, 81

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. E. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4. 17, 19

[Torralba et al., 2008] Torralba, A., Fergus, R., and Freeman, W. T. (2008). 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(11):1958–1970. 146

[Turing, 1950] Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460. 2

[van der Maaten, 2009] van der Maaten, L. (2009). *Feature Extraction from Visual Data*. PhD thesis, Universiteit van Tilburg. 40

[Vapnik, 2000] Vapnik, V. (2000). *The Nature of Statistical Learning Theory*. Springer-Verlag. 10

[Vapnik and Chervonenkis, 2015] Vapnik, V. N. and Chervonenkis, A. Y. (2015). *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*, pages 11–30. Springer. 13

[Ververidis and Kotropoulos, 2006] Ververidis, D. and Kotropoulos, C. (2006). Emotional speech recognition: Resources, features, and methods. *Speech Communication*, 48(9):1162–1181. 54, 55

[Viterbi, 1967] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269. 49

[Vlasenko et al., 2007] Vlasenko, B., Schuller, B., Wendemuth, A., and Rigoll, G. (2007). Frame vs. turn-level: emotion recognition from speech considering static and dynamic processing. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 139–147. Springer. 55

[Wang and Zheng, 2015] Wang, D. and Zheng, T. F. (2015). Transfer learning for speech and language processing. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 1225–1237. 72, 73

[Werbos, 1988] Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356. 32

[Wilamowski and Yu, 2010] Wilamowski, B. M. and Yu, H. (2010). Neural network learning without backpropagation. *IEEE Transactions on Neural Networks*, 21(11):1793–1803. 26

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer. 16

[Wolpert, 1996] Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390. 13

[Yao et al., 2012] Yao, K., Yu, D., Seide, F., Su, H., Deng, L., and Gong, Y. (2012). Adaptation of context-dependent deep neural networks for automatic speech recognition. In *IEEE Workshop on Spoken Language Technology (SLT)*, pages 366–369. 72

[Yelin et al., 2013] Yelin, K., Honglak, L., and Provost, E. (2013). Deep learning for robust feature generation in audiovisual emotion recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3687–3691. 58, 149

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NIPS)*, pages 3320–3328. Curran Associates, Inc. 32, 73

[Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pages 818–833. Springer. 22, 73

[Zenke et al., 2017] Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *International Conference on Machine Learning (ICML)*, volume 70, pages 3987–3995, Sydney, NSW, Australia. 36, 104

[Zhang et al., 2016] Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Laurent, C., Bengio, Y., and Courville, A. (2016). Towards end-to-end speech recognition with deep convolutional neural networks. In *Interspeech*. 54

Appendix A

Datasets

The datasets used in this work are presented and described herein. Appendix A.1 presents the CIFAR-10 and CIFAR-100 datasets. Appendix A.2 presents the eNTERFACE dataset. Appendix A.3 presents the IEMOCAP dataset. Appendix A.4 presents the ImageNet dataset. Appendix A.5 presents the SVHN dataset. Finally, Appendix A.6 presents the TIMIT dataset.

A.1 CIFAR

The CIFAR-10 and CIFAR-100 datasets [Krizhevsky, 2009] were produced at the University of Toronto and released in 2009. Both datasets are mutually exclusive labelled subsets of the 80 Million Tiny Images dataset [Torralba et al., 2008]. The 80 million tiny images dataset was collected at Massachusetts Institute of Technology (MIT) and New York University (NYU) using images returned by several search engines for non-abstract English nouns in the lexical database WordNet [Miller, 1995]. The labels in the CIFAR-10 and CIFAR-100 datasets were obtained by human annotators paid to label the subsets.

The CIFAR-10 and CIFAR-100 datasets consist of RGB images, of size 32×32 pixels, labelled into 10 and 100 classes respectively. The 10 classes in the CIFAR-10 dataset are: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. The 100 classes in the CIFAR-100 dataset are listed in Table A.1. Note that the 100 classes are clustered into 20 superclasses as indicated in Table A.1; these 20 superclasses were ignored in this work.

Each dataset comprises a training set of 50000 images and a test set of 10000 images, i.e. the CIFAR-10 dataset contains 5000 training images and 1000 test images per class, whereas the CIFAR-100 dataset contains 500 training images and 100 test images per class. Figure A.1 illustrates images randomly drawn from the training set of the CIFAR-10 dataset.

Table A.1: Classes in the CIFAR-100 dataset.

Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

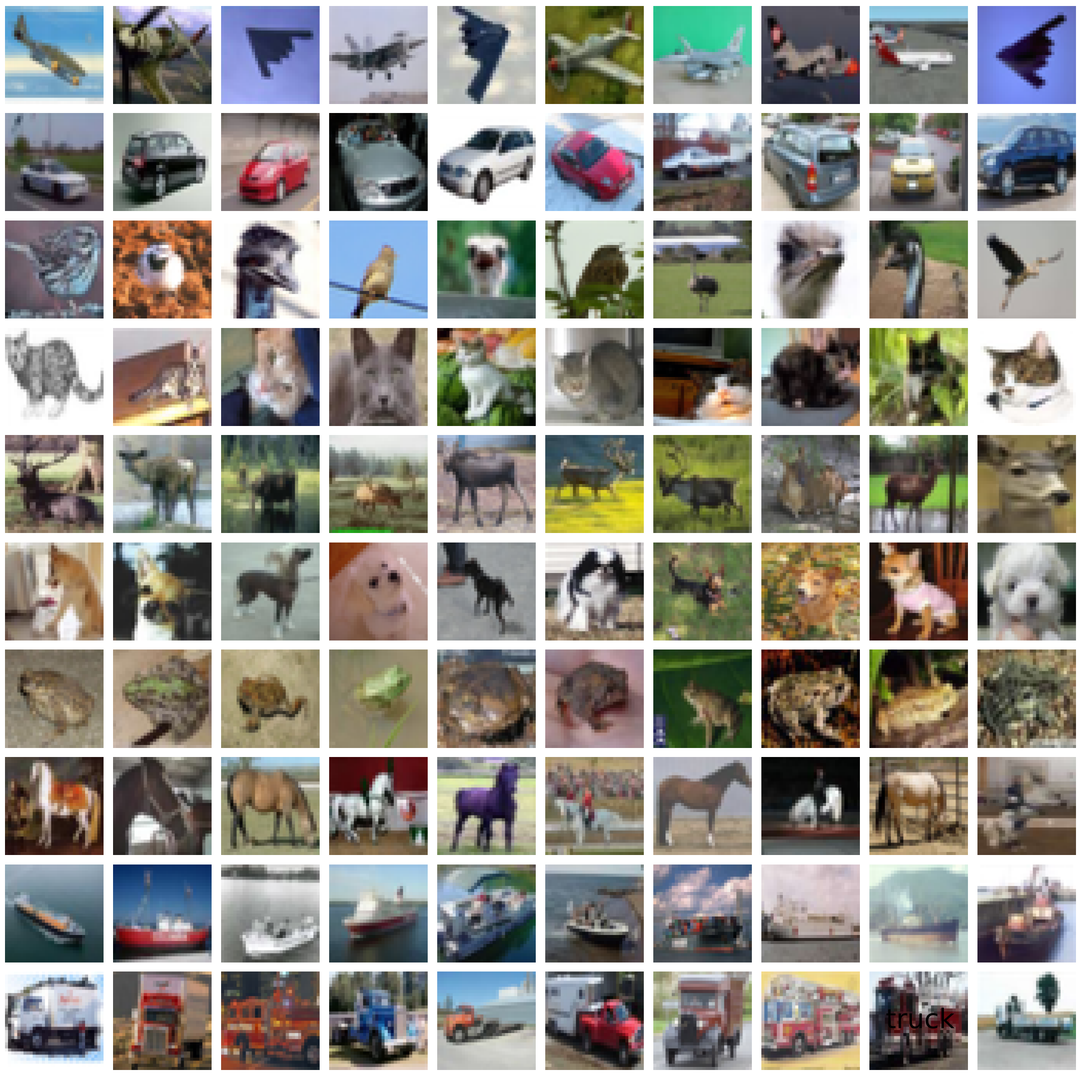

Figure A.1: Sample images randomly drawn from the training set of the CIFAR-10 dataset. Each row was drawn from a single class in the following order: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*.

A.2 eNTERFACE

The eNTERFACE [Martin et al., 2006] dataset was recorded and released in 2005. The dataset comprises approximately one hour of audio-visual recordings of 44 male and female speakers reading sentences that included a strong emotional component. The emotional component was induced by listening to a short story that evoked a particular emotion prior to uttering the scripted sentence. There are 35 male speakers and 9 female speakers; mostly non-native English speakers.

There are 1296 English recordings in total, categorically labelled in six emotions, namely, *anger*, *disgust*, *fear*, *happiness*, *sadness*, and *surprise*. Only audio was used in this work, which had a sampling frequency of 16 kHz. The training, validation, and test sets were constructed by randomly selecting utterances from 40 speakers, whereas the remaining four speakers were discarded due to inconsistencies in their data.

A.3 IEMOCAP

The Interactive Emotional Dyadic Motion Capture (IEMOCAP) dataset [Busso et al., 2008] was recorded by the Signal Analysis and Interpretation Laboratory (SAIL) at the University of Southern California (USC) and released in 2008. The dataset comprises 12 hours of audio-visual recordings divided into five sessions. Each session is composed of two actors, a male and a female, performing emotional scripts as well as improvised scenarios. In total, the dataset comprises 10039 utterances sampled at 48 kHz with an average duration of 4.5 s.

Utterances were labelled by three annotators using categorical labels. The dataset predominantly focused on five emotions, namely, *anger*, *happiness*, *sadness*, *neutral*, and *frustration*; however, annotators were not limited to these emotions during annotation. Ground truths labels were obtained by majority voting between the annotators, where 74.6% of the utterances were agreed upon by at least two annotators. To be consistent with other studies on this dataset [Shah et al., 2014, Mariooryad and Busso, 2013, Yelin et al., 2013], utterances that bore only the following four emotions: *anger*, *happiness*, *sadness*, and *neutral*, were included, with *excitement* considered as *happiness*, amounting to a total of 5531 utterances.

A.4 ImageNet

The ImageNet dataset [Russakovsky et al., 2015] was not used in this work, but referred to on multiple occasions, and thus warranting a brief description here. The dataset was collected and is maintained by researchers at Princeton University and Stanford University. The ImageNet dataset is a large dataset of natural images organised according to the WordNet hierarchy [Miller, 1995]. The dataset contains over 15 million high-resolution images categorised into 20000 categories. The images were collected from the web and annotated by human annotators using crowd-sourcing tools. The dataset is used for a number of tasks in computer vision, e.g., object detection and object localization.

A subset of the ImageNet dataset is used in the popular ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) using only 1000 categories. The training set contains 1.2 million images of variable size labelled into the 1000 categories. The validation set comprises 50000 images of variable size labelled into the same 1000 categories. The test set is typically withheld from the public by the ILSVRC organisers.

A.5 SVHN

The Street View House Numbers (SVHN) dataset [Netzer et al., 2011] was collected at Google and Stanford University and released in 2011. The dataset was obtained from a large number of Google Street View images of house numbers using a combination of automated algorithms and crowd-sourcing tools. The dataset is available in two formats: (1) full numbers — the original variable-sized RGB images, as obtained from street view via a sliding window house-numbers detector, screened and transcribed by humans from crowd-sourcing tools; (2) cropped digits — a cropped version of the original images, of fixed size 32×32 pixels, with a single digit in approximately the centre of each image, although other digits may appear on the sides of the image. The dataset consists of RGB images, of size 32×32 pixels, labelled into 10 classes. Each of the 10 classes denotes a digit.

The training and test sets of the SVHN dataset contain 73257 and 26032 images respectively, and additionally, 531131 images are available that can be appended to the training set. Figure A.2 illustrates images randomly drawn from the training set of the SVHN dataset.

A.6 TIMIT

The TIMIT dataset [Garofolo et al., 1993] was recorded by Texas Instruments (TI), transcribed by Massachusetts Institute of Technology (MIT), with design aid from SRI International, under sponsorship from the Defence Advanced Research Projects Agency — Information Science and Technology Office (DARPA-ISTO). It was verified and prepared for release by the National Institute of Standards and Technology (NIST), and released in 1990, with the aim of advancing Automatic Speech Recognition (ASR) systems.

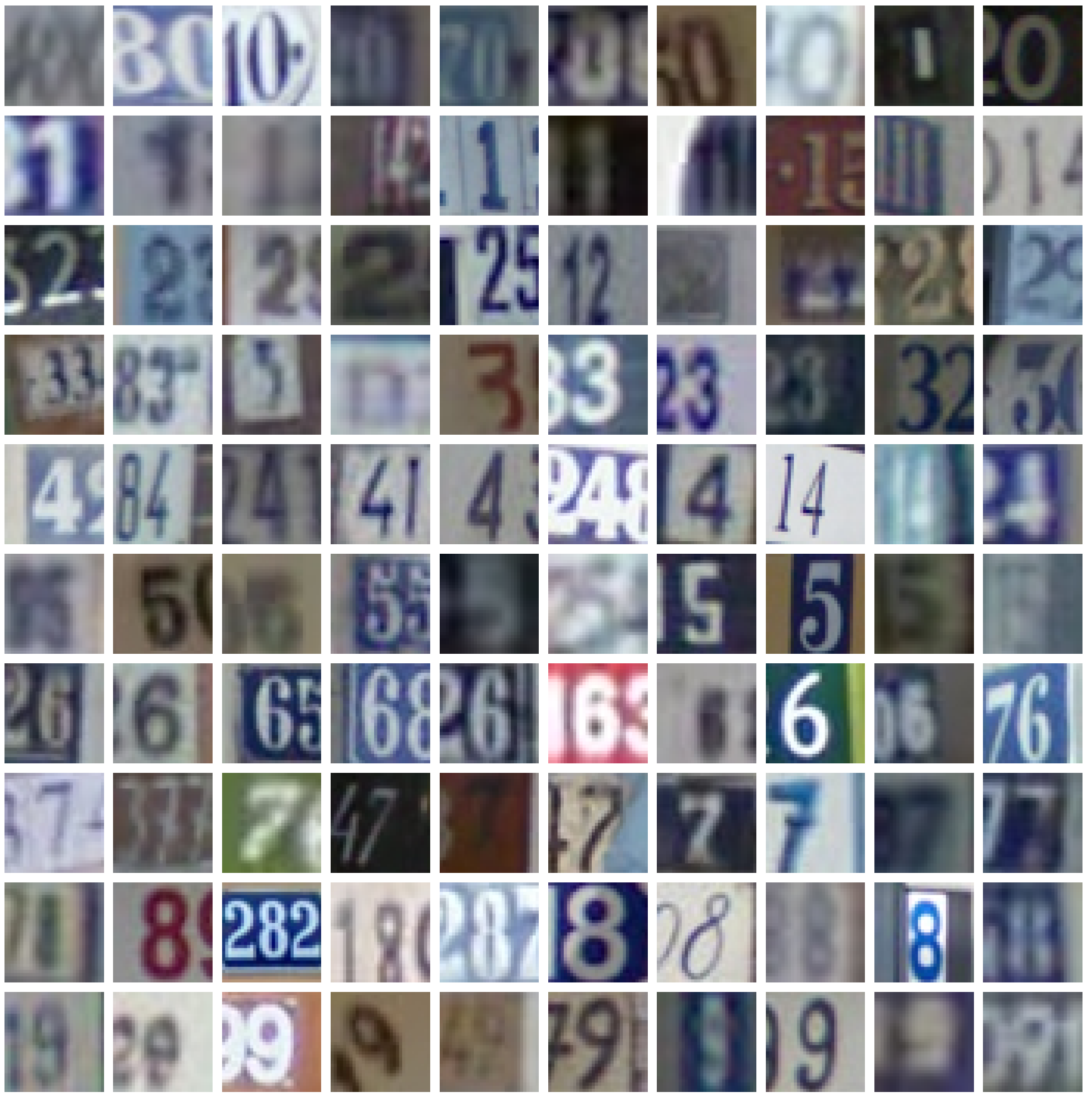

Figure A.2: Sample images randomly drawn from the training set of the SVHN dataset. Each row represents a class or a digit 0–9 respectively.

The TIMIT dataset contains recordings of 630 speakers, of which 70% are males and 30% are females, from eight major American English dialects, with each speaker reading ten phonetically rich sentences, amounting to a total of 6300 utterances. The sampling rate of all recordings is 16 kHz. Each utterance is accompanied by a phonetic transcript. There are 61 phonemes in the TIMIT dataset.

The dataset is predeterminedly divided into mutually exclusive training and test sets [Garofolo et al., 1993, Lee and Hon, 1989, Povey et al., 2011]. The complete 462-speaker training set, without the dialect (SA) utterances, was used as the training set. The 50-speaker development set was used as the validation set. The 24-speaker core test set was used as the test set.