Learning Deep and Compact Models for Gesture Recognition

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS in Computer Science and Engineering, by Research

by

Koustav Mullick
201307559
koustav.mullick@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
JULY 2018

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Learning Deep and Compact Models for Gesture Recognition" by Koustav Mullick, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Dr. Anoop M. Namboodiri

To my Parents and Sister

Acknowledgments

I would like to take this opportunity to express my profound gratitude to all the amazing and wonderful people around me, without whom it would not have been possible to complete my thesis work.

I will start off with my parents, sister and brother in law. They have been a constant source of motivation, inspiration and blessing throughout my master's duration. I feel fortunate to have been gifted with such an amazing family who have always kept me afloat during the hardest of times, with love and encouragement. Next I would like to thank the person who has played the most crucial and important part during my time as a master's student. That would be my thesis advisor, Dr. Anoop M. Namboodiri. Not only he is a wonderful teacher but as a mentor his office door was always open whenever I ran into a trouble spot or had a question about my research or writing.

I cannot express how much privileged and blessed I feel to have some of the most impressive people around me, whom I can call my very own family, away from home. Ayushi, Debarshi, Riddhiman, Satarupa and Sudipto, you people are not just my best friends but much more than that. I cannot imagine my time at IIIT without having you guys around me. Past four years have been a roller-coaster ride in many ways for all of us and you people know how much grateful I feel that we stuck together during all those times of highs and lows. Additionally, I feel thankful to have known and become close friends with Amrisha, Charu, Koustav G, Parijat, Soham, Manoj, Avijit, Deepayan, Aniket, Sukanya, Chetan and Bharadwaj. We shared a lot of things together and today when I look back, I realize how fortunate I have been to come across you guys. Specially Avijit, Deepayan, Manoj, Parijat and Soham. The last few months at IIIT will always be memorable to me for all the times that we have spent together.

I feel humbled and privileged to have been a part of the CVIT lab. Not only it has some of the most knowledgeable people working here, but boasts to have one of the best research facilities in the entire country. The experience that I had and the things that I have learned over here, over the past few years, are something which will stay with me forever to cherish. Last but not the least, I am also thankful to all the innumerable people who have had direct or indirect influence on my thesis work. Be it in the form of answering my query in some online forum, or suggesting that I take a look at some paper, or even sometimes providing with the assurance, "Don't worry. Things will be alright soon." Your contributions are much appreciated.

A big hug and thank-you to all you wonderful people.

Abstract

The goal of gesture recognition is to interpret human gestures, that can originate from any bodily motion, but mainly confided to face or hand, and interact with a computer through it without physically touching it. It can be seen as a way for computers to begin to understand human body language, thus building a richer bridge between machines and humans. Many approaches have been made using cameras and computer vision algorithms for interpretation of sign language, identification and recognition of posture, gait, proxemics, and human behaviors. However effective gesture detection and classification can be quite a challenging task. Firstly, there can be a wide range of variations in the way the gestures are being performed. It needs to be generic and robust enough to handle variations in surrounding conditions, appearances, noise and individuals performing the gestures. Secondly, developing a model that can give real-time predictions and can be run on low-power devices having limited memory and processing capacity, is another challenge. Since deep learning models tend to have a large number of parameters, it not only has the disadvantage of not being able to fit into a mobile device because of the huge model size but also makes it difficult to utilize them for real-time inferencing.

In this thesis we try to address both the above mentioned difficulties. We propose an end-to-end trainable model capable of learning both spatial and temporal features present in a gesture video directly from the raw video frames. It is achieved by combining the strengths of 3D-Convolutional Neural Networks and Long Short Term Memory variant of Recurrent Neural Networks. Further, we also explore ways to reduce the parameter space of such models without compromising a lot on performance. Particularly we look at two ways of obtaining compact models, with less number of parameters. Learn smaller models making use of the idea of knowledge distillation and reduce large models' sizes by performing weight pruning.

Our first contribution is learning the joined, end-to-end trainable, 3D-Convolutional Neural Network and Long Short Term Memory. Convolutional Neural Networks preserve both spatial and temporal information over the layers and can identify patterns over short durations. But the inputs need to be of fixed size, which may not always hold true in case of videos. Whereas, Long Short Term Memories face no difficulties in preserving information over longer duration and it can also work with variable length input sequences. However, they do not preserve patterns and hence works better when fed with features that already has learned some amount of spatio-temporal information, instead of just the raw pixel information. The joined model leverages the advantages of both of them. Experimentally we verify as well that, that indeed is the scenario, as our joined model outperforms the individual baseline models.

Additionally the components can be pre-trained initially and later fine-tuned in a complete end-to-end fashion to further boost the network's potential to capture information. We obtain almost state-of-the art result using our proposed model on the ChaLearn-2014 dataset for sign language recognition from videos, but using much simpler model and training mechanism compared to the best model.

In our second contribution, we look into ways to learn compact models that enables us to perform real-time inferencing on hand-held devices where power and memory are constraints. To this extent we distill or transfer knowledge from a larger teacher network to a smaller student network. Without teacher supervision, the student network did not have enough capacity to perform well just using class-labels. We demonstrate this on the same ChaLearn-2014 dataset. To the best of our knowledge, this is the first work to explore knowledge distillation from teacher to student network in video classification task. We also show that training networks using Adam optimization technique, combined with weight decay, helps to obtaining sparser models by pruning weights. Training with Adam encourages a lot of weights to become very low by penalizing high weight values and adjusting the learning rate accordingly. Removing the low-valued weights helps to obtain sparser models, compared to SGD (with weight-decay as well) trained models. Experimental results on both gesture recognition task and image classification task on the CIFAR dataset validates the findings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In recent years, human action recognition has drawn increasing attention of researchers, primarily due to its potential in areas such as video surveillance, robotics, user interface design and multimedia video retrieval. Gesture recognition, which falls under the broad category of action recognition, is one of the key components in natural human-robot interaction and human-computer interfaces. Especially for mobile devices, where interfaces like keyboard and mouse are impractical, human-computer interfaces can play a pivotal role. To enable smart interactions with electronic devices, recognition of distinct hand and arm motions is becoming increasingly important. Furthermore, gesture identification in video can be seen as a first step towards sign language recognition, where even subtle differences in motion can play an important role.

Previous works on video-based action recognition focused mainly on adapting hand-crafted features to perform classification [34], [44], [54]. These involved an initial feature detection stage, which are then converted to fixed-length feature descriptors, followed by classification using standard classifiers like Support Vector Machines (SVM). Well-known feature detection methods are Harris3D [30], Cuboids [10] and Hessian3D [52]. Popular descriptors used are Cuboids [42], HOG/HOF [30], HOG3D [26] and Extended SURF [52]. However, [51] discusses that the best performing feature descriptor is dataset dependent and no universal hand-engineered feature outperforming all others exists. For this reason it is highly beneficial to learn dataset specific low-level, mid-level and high-level features, either in supervised, unsupervised, or semi-supervised settings.

Since the recent resurgence of neural networks invoked by Hinton and others [20], deep neural architectures have become an effective approach for extracting high-level features from data. In the last few years deep artificial neural networks have given us *state-of-the-art* results on numerous computer vision related tasks, such as object classification [28], detection and localization [14], [43], image segmentation [32], [8], image captioning [23], video classification [24], to cite a few. Moreover, thanks to the immense popularity of the Microsoft Kinect [45], [16], there has been a surge in interest in developing methods for human gesture and action recognition from 3D skeletal data and depth images. The first

gesture-oriented dataset containing a sufficient amount of training samples for deep learning methods was proposed for the *ChaLearn 2013 Challenge on Multi-modal Gesture Recognition* [12]. We use the 2014 version of the same dataset [11] in this thesis.

1.1.1 Challenges

Effective gesture detection and classification is challenging due to several factors: cultural and individual differences in tempos and styles of articulation, variable observation conditions, the small size of fingers in images taken in typical scenarios, noise in camera channels, infinitely many kinds of out-of-vocabulary motion, and real-time performance constraints. Also changes in background, lighting, users having different appearance, pose and positioning relative to the camera, contribute to the difficulty of the task.

Furthermore, developing a model that can give real-time predictions and can be run on low-power devices having limited memory (such as mobile phones), has it's own set of challenges:

- It should be generic and robust enough to give good classification for a wide range of gestures.
- It needs to be compact so that it does not have very high memory requirements.
- The model must be simple and efficient enough to perform real-time inferencing on low-power devices.

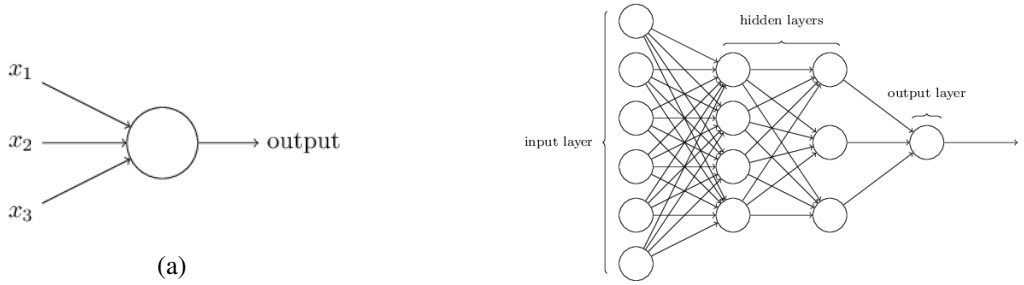
1.2 Thesis Overview

In this thesis we would like to develop an end-to-end trainable model that can automatically learn low, mid and high level features from both spatial and temporal dimension in a complete gesture. Towards this we combine the strengths of 3D-Convolutional Neural Networks, that are good at capturing spatial and short-term temporal features, and Long Short Term Memories, that can capture long-term temporal signatures of actions. Also, deep learning models tend to have a large number of parameters, which makes training time and memory requirements high, making it difficult to employ them in embedded devices for real-time inferencing. Hence, we would like to explore ways to reduce the parameter space of such models without compromising a lot on performance.

In the next section we briefly introduce some of the terminology and concepts which will be required throughout the remainder of this thesis, hence it is important to have a good understanding of them beforehand. In chapter 2 we discuss about some of the previous approaches towards action classification and gesture recognition using Deep Learning (DL) methods. Chapter 3 explains our approach towards robust human-gesture recognition and validates it's effectiveness with experimental results. We explore methods to reduce our model size and make them compact in chapter 4. In the final chapter we conclude the thesis by discussing the overall contributions of this work and some of the future avenues that can be explored as extensions from this work.

1.3 Background

1.3.1 Artificial Neural Network



(a) (b)

Figure 1.1: (a) A single Neuron. (b) A Multilayer Perceptron

An Artificial Neural Network (ANN) is a computational model based on the structure and functions of biological neural networks. It is based on a collection of connected units called artificial neurons, (analogous to axons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. A **multilayer perceptron** (MLP) is a class of feed-forward ANN that consists of at least three layers of nodes (a **feed-forward neural network** is an ANN wherein connections between the units do not form a cycle). Figure 1.1 shows the example of a single neuron and an MLP.

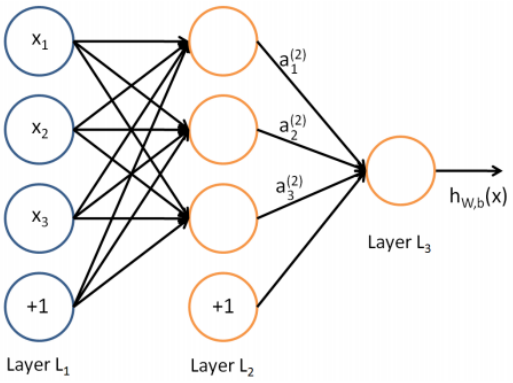


Figure 1.2: An MLP. Blue nodes are input nodes, middle orange nodes are hidden nodes, right orange node is output node and $(+1)$ nodes are called *biases*. Black arrows are weight connections between two consecutive layers which are automatically learned using backpropagation.

Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. It can distinguish data that is not linearly separable. We will discuss more about activation functions and backpropagation training algorithms in upcoming sections. Multilayer perceptrons are also referred to as "vanilla" neural networks, especially when they have a single hidden layer. For the MLP shown in Figure 1.2, output at each node can be calculated using chain-rule of differentiation:

$$\begin{aligned} a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\ a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\ a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\ h_{W,b}(x) = a_1^{(3)} &= f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \end{aligned} \tag{1.1}$$

where $a_i^{(l)}$ denotes output of node i in layer l, $W_{ij}^{(l)}$ denotes weight from node j in layer $(l-1)$ to node i in layer l, $b^{(l)}$ is bias in layer l and $f()$ is a non-linear activation function. Refer to Figure 1.2 for further details.

Activation Functions

The activation function of a node defines the output of that node given an input or set of inputs. It is the nonlinear activation function that allows neural networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function. Some of the commonly used activation functions in neural networks are explained below.

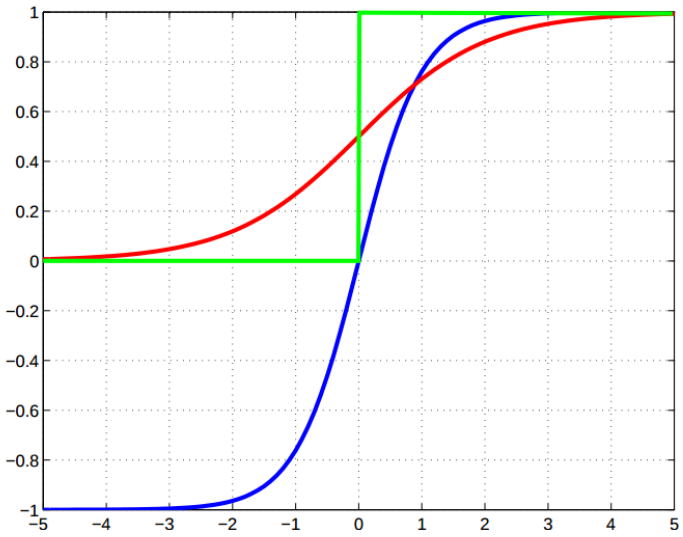


Figure 1.3: Green: Step function, Red: Logistic Sigmoid function, Blue: Hyperbolic Tangent (Tanh) function

Logistic Sigmoid

A logistic sigmoid function (Figure 1.3) is a function having a characteristic "S"-shaped curve. It has domain of all real numbers, with return value monotonically increasing from 0 to 1. The logistic sigmoid is motivated somewhat by biological neurons and can be interpreted as the probability of an artificial neuron "firing" given its inputs. It is defined by the formula:

$$y(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

Hyperbolic Tangent

Though the logistic sigmoid has a nice biological interpretation, it turns out that the logistic sigmoid can cause a neural network to get "stuck" during training. This is due in part to the fact that if a strongly-negative input is provided to the logistic sigmoid, it outputs values very near zero. Since neural networks use the feed-forward activations to calculate parameter gradients, this can result in model parameters that are updated less regularly than we would like, and are thus "stuck" in their current state.

An alternative to the logistic sigmoid is the hyperbolic tangent, or tanh function (Figure 1.3). Like the logistic sigmoid, the tanh function is also sigmoidal ("s"-shaped), but instead outputs values that range (-1, 1). Thus strongly negative inputs to the tanh will map to negative outputs. Additionally, only zero-valued inputs are mapped to near-zero outputs. It is given by the formula:

$$y(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{1.3}$$

Softmax

The softmax function is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{x}$ of arbitrary real values to a K-dimensional vector $y(\mathbf{x})$ of real values in the range [0, 1] that add up to 1. The output of the softmax function can be used to represent a categorical distribution – that is, a probability distribution over K different possible outcomes. It is used in various multiclass classification methods, such as multinomial logistic regression, multiclass linear discriminant analysis, Naive Bayes classifiers and artificial neural networks. The function is given by:

$$y(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \qquad \text{for j = 1, \ldots, K} \tag{1.4}$$

Rectified Linear Unit

The Rectified Linear Unit (ReLU) (Figure 1.4) has become very popular in the last few years. The activation is simply thresholded at zero. It computes the function:

$$y(x) = max(0, x) \tag{1.5}$$

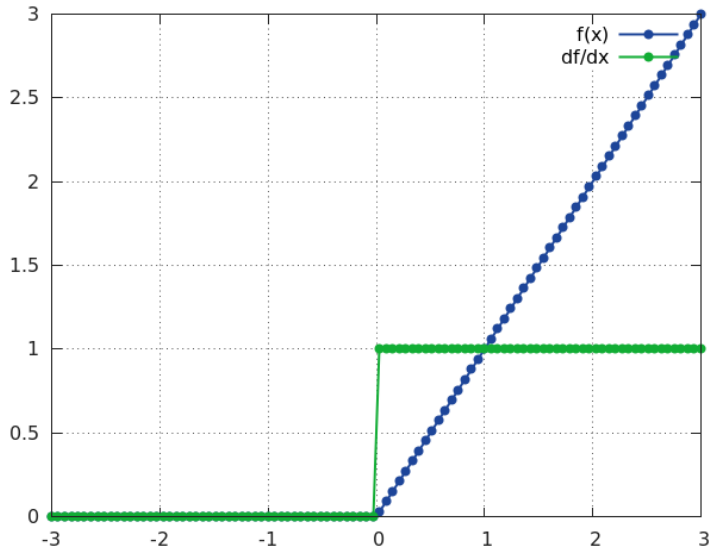


Figure 1.4: Rectified Linear Unit (ReLU) activation function (in Blue) and it's gradient (in Green).

Some of the pros and cons to using the ReLUs are:

- It was found to greatly accelerate (e.g. a factor of 6 in [28]) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
- Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
- Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on.

Backpropagation

Backpropagation is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It is commonly used as a part of algorithms that optimize the performance of the neural network by adjusting the weights, for example in the gradient descent algorithm.

The calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated

last. Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backwards flow of the error information allows for efficient computation of the gradient at each layer versus the naive approach of calculating the gradient of each layer separately. We will take a look at the detailed derivation of the backpropagation algorithm in a three-layer neural network in the Appendix 6.1.

1.3.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a class of deep, feed-forward artificial neural network that have successfully been applied to analyzing visual imagery. Convolutional Neural Networks, similar to Artificial Neural Networks, are made up of neurons that have trainable weights and biases. Each neuron receives some inputs, performs a dot product and follows it with a non-linearity. However, CNN architectures make the explicit assumption that the inputs are images, which allows to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

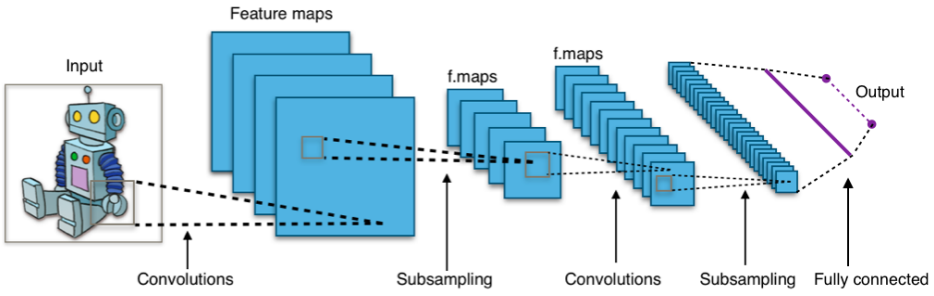


Figure 1.5: Example of a Convolutional Neural Network Architecture.

A CNN consists of a number of convolutional and subsampling (pooling) layers optionally followed by fully connected layers. Unlike a regular Neural Network, the layers of a CNN have neurons arranged in 3 dimensions: width, height and depth. The input to and output are also 3D volumes of neurons. To build a CNN architecture, three types of layers are mainly used: convolutional layers, pooling layers and fully-connected layers. An example architecture overview is shown in Figure 1.5 and explained below:

- Input is an image of dimension ($height \times width \times depth$), containing raw pixel values. $depth$ could be the three color channels R,G and B.

- Convolutional layers compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. The result is also a volume of dimension ($height \times width \times depth$), where $height$ and $width$ are obtained after performing the convolution operation and $depth$ is the number of filters of the particular convolutional layer.

- Convolutional layers are generally followed by a non-liner activation function layers like ReLU or Sigmoid.
- Pooling layers perform a downsampling operation along the spatial dimensions ($height$, $width$), but keeps the channel dimension of the input volume fixed.
- A typical CNN architecture consists of one or more fully-connected (FC) layers at the end. The final FC layer computes the class scores, resulting in C dimensional output, where C is the number of classes and each of the C values correspond to a class score. As with ordinary Neural Networks, each neuron in this layer will be connected to all the numbers in the previous volume.

Next we discuss in brief the utility of each of convolutional layers, pooling layers and fully-connected layers in the context of CNN.

Convolutional Layer

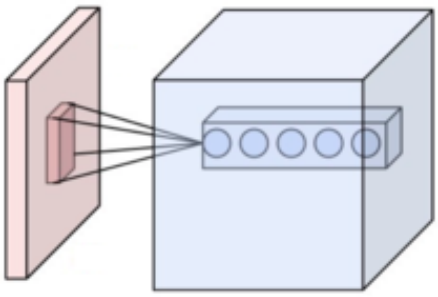


Figure 1.6: Neurons of a Convolutional Layer (blue), connected to their Receptive Field of the Input Volume (red).

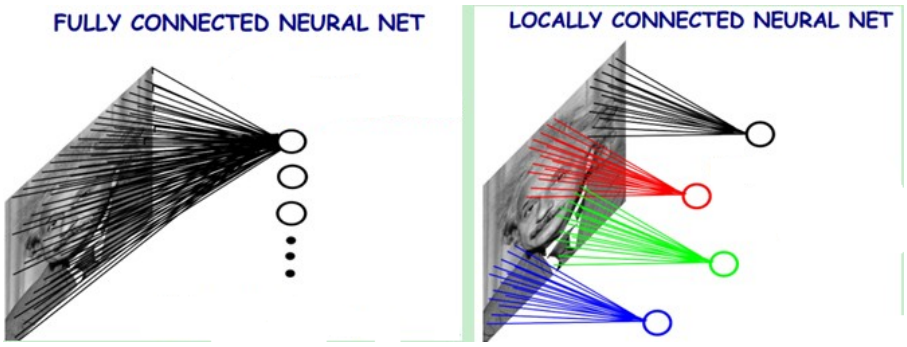


Figure 1.7: Local Connectivity of Convolutional Layers.

The convolutional layer is the core building block of a CNN that does most of the computational heavy lifting. When dealing with high-dimensional inputs such as images, it is impractical to connect

neurons to all neurons in the previous volume. Instead, each neuron is connected to only a local region of the input volume. The spatial extent of this connectivity is a hyper-parameter called the receptive field of the neuron (denoted by the filter size of the layer). The connections are local in space (along width and height), but always full along the entire depth of the input volume. Figure 1.6 and Figure 1.7 shows example of a convolutional layer and it's local connectivity.

Parameter sharing scheme is used in convolutional layers to control the number of parameters. If one feature is useful to compute at some spatial position (x, y), then it should also be useful to compute at a different position $(x2, y2)$. In other words, denoting a single 2-dimensional slice of depth as a depth slice, we are going to constrain the neurons in each depth slice to use the same weights and bias. During backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

Pooling Layer

It is common to periodically insert a pooling or sub-sampling layer in-between successive convolutional layers in a CNN architecture. It's function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The pooling Layer operates independently on every depth slice of the input and resizes it spatially. The depth dimension remains unchanged.

(a) Max Pooling Operation.

(b) Average Pooling Operation.

Figure 1.8: Examples of Pooling Layers.

Two of the most common form of pooling used are max pooling and average pooling. Max pooling replaces the input region it is connected to, with the maximum value. Similarly average pooling replaces the region with the mean (or average) of the values of it's input receptive field. Figure 1.8 shows example of max pooling and average pooling.

Fully-connected Layer

Neurons in a fully-connected layer have full connections to all activations in the previous layer, as seen in regular ANNs. Generally for classification task, the final FC layer of any CNN consists of C hidden-units, where C is the number of classes. The output of the final C-dimensional FC layer is passed through a softmax activation function to obtain class-probability scores corresponding to each class.

1.3.3 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks (like MLPs or CNNs), RNNs can use their internal memory to process arbitrary sequences of inputs. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that is a very bad idea, like prediciting the next word in a sentence, or translating a sentence from on language to another. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being dependent on the previous computations. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps.

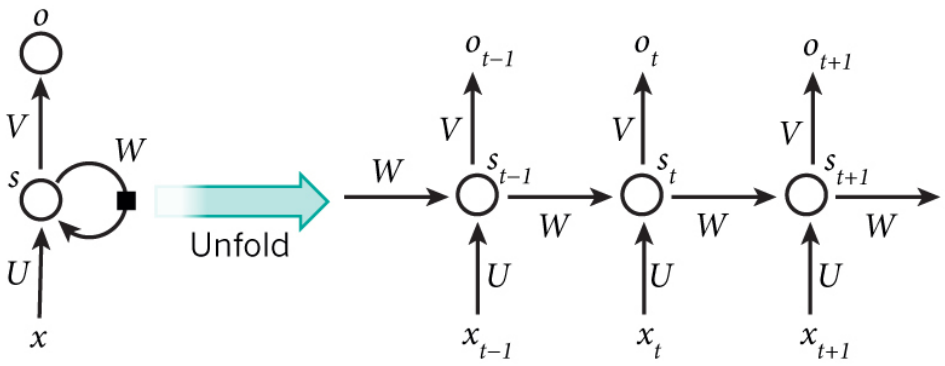


Figure 1.9: A Recurrent Neural Network and the unfolding in time of the computation involved in its forward computation.

Figure 6.2 shows the example of a RNN, unrolled over three time-steps. x_t is the input at time-step t. s_t is the hidden-state at time-step t and is called the "memory" of the network. It captures information about what happened in all the previous time steps and is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$, where f is a nonlinear activation function. The first hidden-step (s_0) is usually set to 0. The output o_t at time-step t is calculated solely based on the memory at time t. It is obtained using $o_t = softmax(Vs_t)$. Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same set of parameters (U, V, W) across all time-steps. This reflects the fact that we are performing the same task at each step, just with different inputs.

Recurrent neural networks are trained using the **Backpropagation through time (BPTT)** algorithm. We will take a look at the detailed steps of it in the Appendix 6.2.

Long Short Term Memory

Long Short Term Memory networks (LSTM) [21] are a special kind of RNN, capable of learning long-term dependencies. They are explicitly designed to avoid the vanishing gradient problem associ-

ated with RNN (explained in the Appendix 6.3). Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.

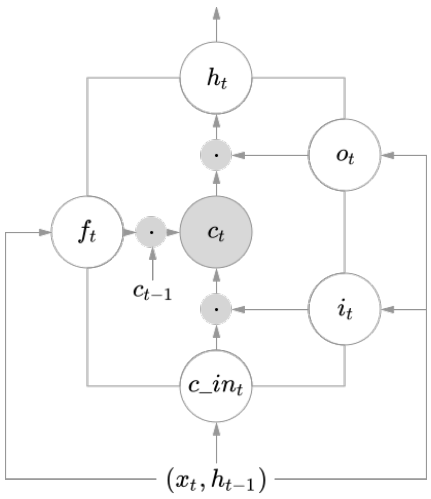


Figure 1.10: A Long Short Term Memory Cell.

The architecture of LSTM (Figure 1.10)is similar to that of RNN, consisting of an input layer, an output layer and a hidden memory block layer. However, it also consists of four specialized neurons in each memory block that gets rid of the vanishing gradient problem: a memory cell, an input gate, a forget gate, and an output gate. The memory cell and the gates each receive a connection from every neuron in the input layer and outgoing connections extend from the memory cell to every neuron in the output layer. The memory cell further has a recurrent connection to itself, allowing its value to persist through time). Gates determine the following:

- Input gate: the extent to which the current input influences the value of the memory cell.
- Forget gate: the extent to which the cell's old value is lost.
- Output gate: the cell's new value propagates to the output layer.

Through gated control the network can effectively maintain and make use of past observations. There are also peephole connections, which allow a gate to know the true state of the memory cell, before it is modulated by the output gate.

The LSTM cell can be defined using the following set of equations (refer to Figure 1.10):

$$
\begin{aligned}
\text{Input gate } \; & i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
\text{Forget gate } \; & f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
\text{Output gate } \; & o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
\text{Input transform } \; & c\_in_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_{c\_in}) \\
\text{Cell state update } \; & c_t = f_t \cdot c_{t-1} + i_t \cdot c\_in_t \\
\text{Output } \; & h_t = o_t \cdot tanh(c_t)
\end{aligned}
\tag{1.6}
$$

where, x_t is the input vector, h_t is the output vector, c_t is the cell state, σ is the sigmoid activation function, W_{mn} is the weight vector from m to n and b_n is the bias associated with n.

1.3.4 Knowledge Distillation

When deploying machine learning models on mobile devices, such as phones and smart watches, one of the main concern that arises is the limited memory and power capacity of these devices. One of the ways to overcome these problems suggested in the literature is called knowledge distillation [19]. The idea had been first introduced in [5] where a more complex model is used to train a simpler, compressed model. Knowledge distillation can be used to transfer the knowledge from a large model (a teacher) trained on a server into a smaller, compressed model (a student) simple enough to run on a hand-held device. The main idea is that classifiers built from a softmax function have a great deal more information contained in them than just the class labels. The correlations in the softmax outputs are very informative. For example, when building a computer vision system to detect cats, dogs, and boats the output entries for cat and dog in a softmax classifier will always have more correlation than cat and boat since cats look similar to dogs. This correlation does not get captured in simple one-hot class labels. Hinton showed that a simpler model with fewer parameters can be trained using this method to match the performance of a larger model.

Instead of training the cross entropy against the labeled data one could train it against the posteriors of a previously trained model. The previous model (teacher network) may be a very large network or an ensemble model, which may contain many large deep networks of similar or various architectures. These networks have a large number of parameters, which makes it computationally demanding to do inference on new samples. To alleviate this, after training the teacher model until the error rate is sufficiently low, we use the softmax outputs from it to construct training targets for the smaller, simpler student model.

In particular, for each data point x_n, the teacher network may make the prediction:

$$
\hat{y}_n^{(teacher)} = P(y_n|x_n; \Theta^{(teacher)}) \tag{1.7}
$$

where, y_n is the class label and $\Theta^{(teacher)}$ is the parameter of the network. The student network is trained using this output distribution rather than the true labels. However, since the posterior estimates are typically low entropy, the softmax output is largely indiscernible without a log transform. To get

around this, Hinton proposed to increase the entropy of the posteriors by using a transform that calculates a "softened" softmax as:

$$[g(y;T)]_k = \frac{y_k^{1/T}}{\sum_{k'} y_{k'}^{1/T}} \tag{1.8}$$

where T is a temperature parameter that when raised increases the entropy. The target distributions are set as:

$$y_n^{(target)} = g(y_n^{(teacher)}; T) \tag{1.9}$$

Thus, the loss function for the student network becomes:

$$L^{(soft)} = \sum_{n=1}^{N} L(x_n, y_n; \Theta^{(student)}) = \sum_{n=1}^{N} \sum_{c=1}^{C} y_{n,c}^{(target)} \log P(y_c | x_n; \Theta^{(student)}) \tag{1.10}$$

where, N is the number of examples, C is the number of classes and $\Theta^{(student)}$ is the parameter of the student network. A combined loss function of both the soft target distribution and actual class labels (denoted by $L^{(hard)}$) can also be used:

$$L = \alpha L^{(soft)} + (1 - \alpha) L^{(hard)} \tag{1.11}$$

where α is a weighing parameter.

Now we have familiarized ourselves with the motivation and overview of the thesis in this introductory chapter. Next we will dive into some of the existing works on gesture recognition in the literature. Also having a brief overview of some of the related concepts and terminologies will help in going through rest of the thesis.

Chapter 2

Related Literature

2.1 Human Action Recognition from Videos

Human action in video sequences consists of spatial information within individual frames as well as temporal information of motion across the frames. CNN models are very good at capturing the spatial information within an image as demonstrated by their success on ImageNet classification. However, for action classification, besides learning the spatial correlation in a frame, it is also important to capture the sequential evolution of information across frames. Approaches used by the vision community to capture the spatio-temporal information in a video, can be broadly divided into two categories.

2.1.1 Learning networks capable of modeling sequential data

One approach is to model the temporal correlation in the network and let it learn by itself from the supervision of the data. 3D-Convolutional Neural Network (3D-CNN) was proposed by Ji *et al.* in [22]. It is an extension of 2D-CNN in which the convolution filters convolved across sets of frames and could learn correlation across both spatial and temporal dimensions. It was shown to significantly outperform frame-based 2D-CNN approaches on human action recognition. Baccouche *et al.* used Long Short Term Memory (LSTM) based Recurrent Neural Network (RNN) for action classification from soccer videos. In three subsequent works they experimented with the features that are fed to the LSTM. In [1], input to the network were pre-extracted features, namely, bag-of-words descriptors, for each frame. In [2] they replaced the pre-computed features with a deep 3D-CNN which automatically learned the most appropriate features in a supervised fashion. Finally, a Convolutional Auto Encoder ([35]) was proposed, in place of the Convolutional Neural Network, to extract features in an unsupervised fashion in [3]. Figure 2.1[1] demonstrates these three approaches. Srivastava *et al.* [48] used multilayer LSTM networks to learn unsupervised video representations. An encoder LSTM mapped an input sequence to a fixed length representation. The fixed length representation was used by another decoder LSTM for reconstructing the input sequence and also to predict future frames, as shown in Figure 2.2. Thus

[1] All the Figures used in this chapter are taken from the respective papers.

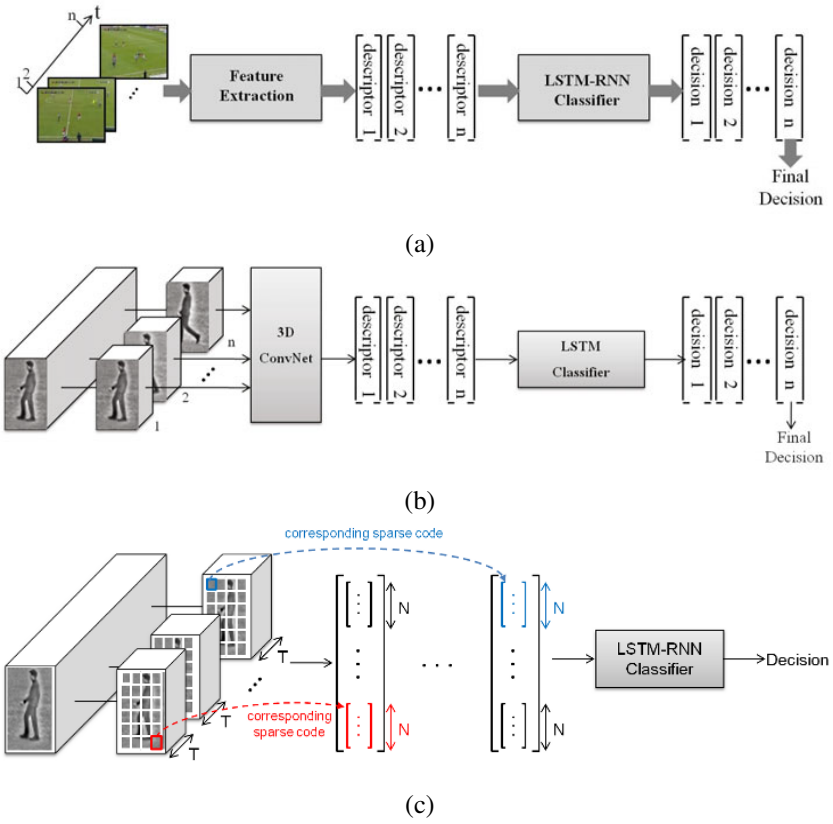


Figure 2.1: Approaches used by Baccouche *et. al* for Human Action Recognition. Features are extracted using (a) BoW descriptors, (b) 3D-CNN and (c) Convolutional Auto-Encoder.

the encoder LSTM is asked to come up with a state from which it can both predict the next few frames as well as reconstruct the input. [15] demonstrated LSTM's potential for robust action recognition in real-world scenarios. They showed that classification accuracy exhibits graceful degradation when the LSTM network is faced with lower qualities of available training data, shorter available video sequences or poorer video quality. [49] proposed a simple, yet effective approach for spatio-temporal feature learning using deep 3D-CNN trained in a supervised fashion on a large scale video dataset, called C3D. They also show that a homogeneous architecture with small $3 \times 3 \times 3$ convolution kernels in all layers is among the best performing architectures for 3D-CNN. Figure 2.3 shows the schematic of the C3D network architecture. Varol *et al.* [50] learned video representation using neural networks with long-term temporal convolutional neural network, models with increased temporal extents. They also study the impact of different low-level representations, such as raw values of video pixels and optical flow vector fields and demonstrate the importance of high-quality optical flow estimation for learning accurate action models. Figure 2.4 shows their network architecture.

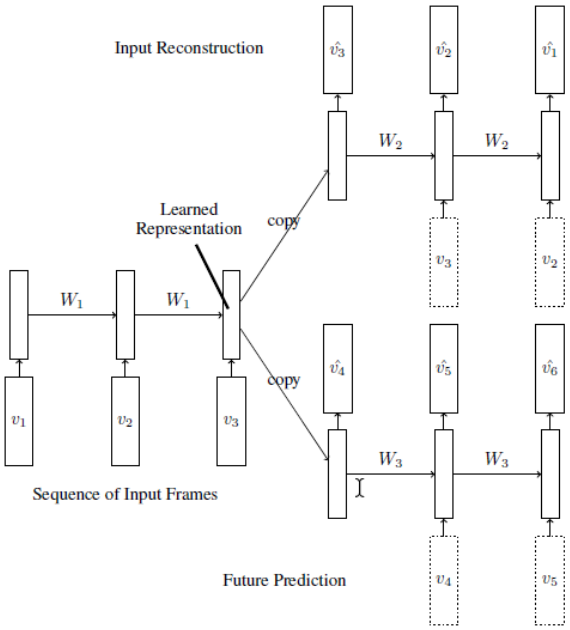


Figure 2.2: Multilayer LSTM Encoder and Decoder Networks to learn Unsupervised Video Representations.

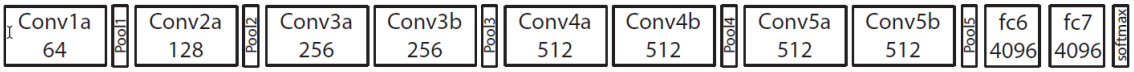


Figure 2.3: C3D network architecture for Large Scale Video Recognition. All 3D Covolution Kernels are $3 \times 3 \times 3$ with stride 1. Pooling Kernels are $2 \times 2 \times 2$, except *pool1* which is $1 \times 2 \times 2$.

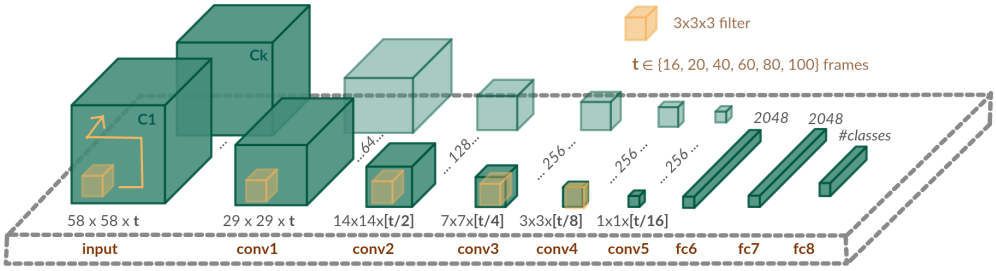


Figure 2.4: Network Architecture of Varol *et al.* Spatio-Temporal Convolutions with $3 \times 3 \times 3$ filters are applied in the first 5 layers, with Max-Pooling and ReLU in between all convolution layers. Network Input Channels $C_1 ... C_k$ are defined for different temporal resolutions $t \in \{20, 40, 60, 80, 100\}$ and either Two-Channel Motion *(flow-x, flow-y)* or Three-Channel Appearance *(R, G, B)*.

2.1.2 Using ImageNet pre-trained networks for feature extraction

In recent works a popular approach has been to use ImageNet [9] pre-trained networks to extract features at frame level. The frame-level features can then be fed to some sequential model (like LSTM)

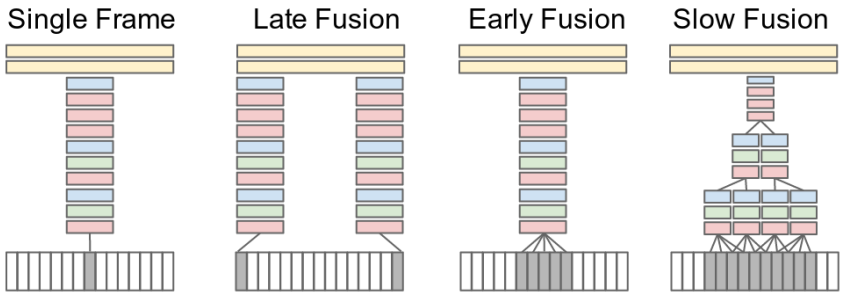


Figure 2.5: Fusion Techniques across Temporal Dimension proposed by Karpathy *et. al.* Red, Green and Blues boxes indicate Convolutional, Normalization and Pooling Layers respectively.

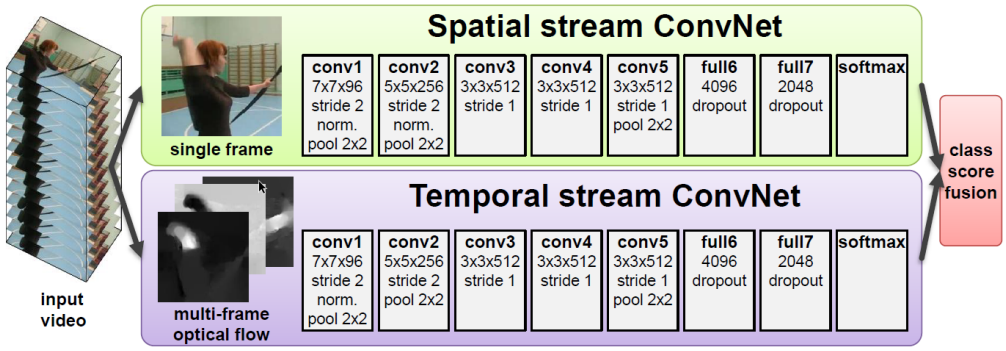
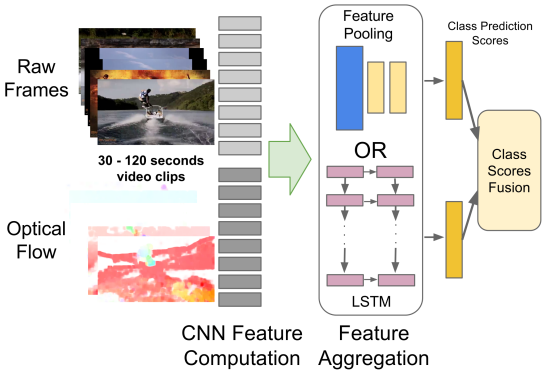
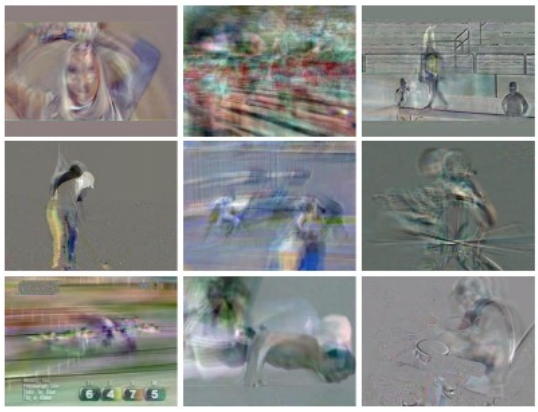


Figure 2.6: Two-Stream CNN for Action Recognition.



(a) Overview of approach used by Ng *et. al.*



(b) Examples of Dynamic Images obtained by applying rank pooling on the raw image pixels of a video.

Figure 2.7

or pooled across time-domain, to perform final classification. Karpathy *et al.* [24] proposed multiple approaches for extending the connectivity of a CNN in time domain to take advantage of local spatio-temporal information. They proposed various fusion techniques across temporal dimension, as shown

in Figure 2.5, and multi-resolution CNNs. Simonyan *et al.* [46] introduced a two-stream CNN, for spatial and temporal streams of frames respectively, as shown in Figure 2.6. The spatial stream CNN was pre-trained on ImageNet [9] and worked on individual frames. Temporal stream CNN used optical flow between individual frames of the video and the horizontal and vertical components of the vector field are considered as image channels. Flow channels of consecutive frames of the video were stacked together to obtain the input to the temporal stream. Outputs from the two streams are fused to provide final classification. NG *et al.* [37] performed activity recognition by passing RGB and optical flow frames through pre-trained CNN networks and fed the learned features to an LSTM for final recognition (Figure 2.7a). They also explored various temporal feature pooling architectures which need to be made when adapting a CNN for this task, shown in the figure. [4] introduced the concept of dynamic image, obtained by directly applying rank pooling on the raw image pixels of a video producing a single RGB image per video. The dynamic image was obtained through the parameters of a ranking machine that encoded the temporal evolution of the frames of the video. The new RGB image could be processed by a standard CNN architecture which is structurally nearly identical to architectures used for still images, while still capturing the long-term dynamics in a video. Figure 2.7b shows some examples of dynamic images obtained from videos.

2.2 Gesture Recognition from Videos

Gesture recognition from videos is different from action recognition performed by the aforementioned methods in two ways. Firstly, human gestures focus more on the spatio-temporal features of arms and hands. The effective spatial convolutional features of hands may be overwhelmed by complex scene backgrounds due to the relative small size of fingers in images; thus the temporal information becomes more discriminative for gesture recognition, compared to the general video classification tasks. Secondly, unlike most existing human action datasets, gesture datasets may consist of additional modalities of data streams available. Thanks to the immense popularity of the Microsoft Kinect [45], [16], there has been a surge in interest in developing methods for human gestures from 3D skeletal data and depth images. Thus it is important to develop models which can leverage all the available input modalities and provide with the best classification.

Wu *et al.* [53] learned high-level spatio-temporal representations using deep neural networks suited to the input modality: a Gaussian-Bernouilli Deep Belief Network (DBN) to handle skeletal dynamics, and a 3D Convolutional Neural Network (3D-CNN) to manage and fuse batches of depth and RGB images. These deep neural networks are integrated within a Hidden Markov Model (HMM) to infer gesture sequences (Figure 2.8). Pigou *et al.* [38] proposed a two-branched 3D-CNN architecture, for the upper-body and hand region (extracted using skeleton joint information) respectively. Both depth and grayscale information are concatenated as channels of the input for each branch. Final classification is performed after fusing the features from the two networks. Figure 2.9 shows their architecture. ModDrop [36] used a multi-scale and multi-modal deep learning approach. It performed careful ini-

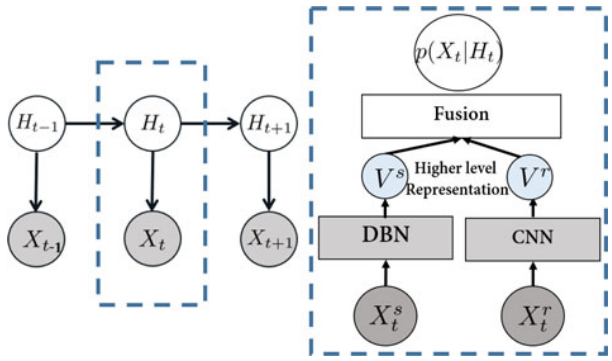


Figure 2.8: The Temporal Model is an HMM (left), whose Emission Probability $p(X_t|H_t)$ (right) is modeled by Feed-Forward Neural Networks. Observations X_t (Skeletal Features X_t^s, or RGB-D Image Features X_t^r) are first passed through the appropriate DNNs (a DBN pretrained with Gaussian-Bernouilli Restricted Boltzmann Machines for the skeleton modality and a 3DCNN for the RGB-D modality) to extract High-Level Features (V^s and V^r). These are subsequently combined to produce an estimate of $p(X_t|H_t)$.

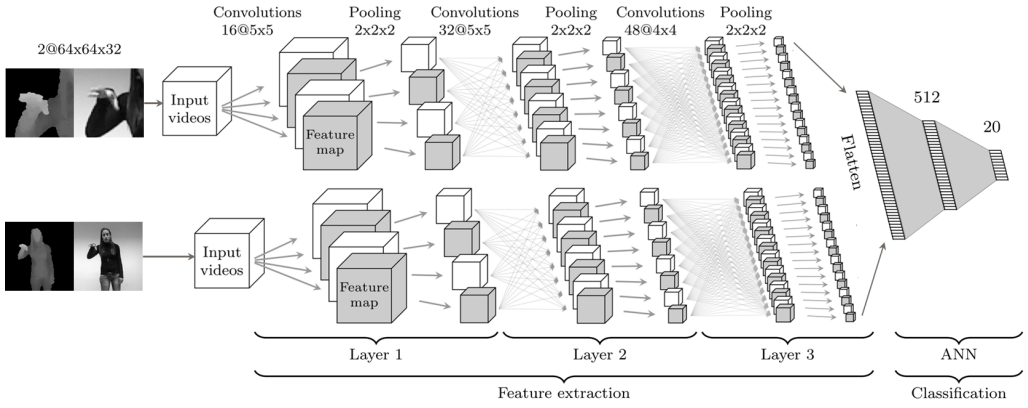


Figure 2.9: Architecture used by Pigou *et. al.*

tialization of individual modalities (depth, RGB, audio and skeleton joints) and fused them at several spatial and temporal scales. Their architecture is a big cascade having individual branches for each hand, each modalities, and articulated pose based features (Figure 2.10). [39] explored deep architectures for gesture recognition in videos and proposed a new end-to-end trainable neural network architecture incorporating temporal convolutions and bidirectional recurrence. They showed that adding temporal convolutions leads to significant improvements. Zhu *et al.* [55] presented a pyramidal 3D convolutional network framework for large-scale isolated human gesture recognition. 3D-CNN is utilized to learn the spatio-temporal features from gesture videos. Pyramid input was proposed to preserve the multi-scale contextual information of gestures, and each pyramid segment is uniformly sampled with temporal jitter. Pyramid fusion layers were inserted into the 3D-CNNs to fuse the features of pyramid input.

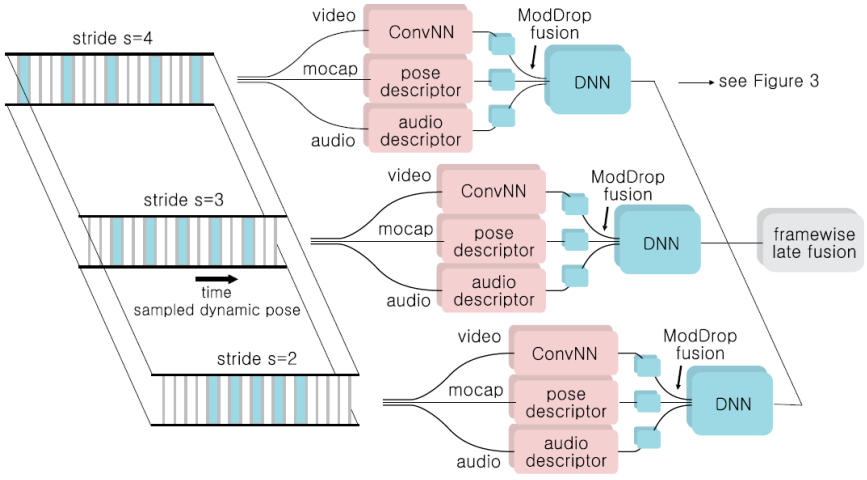


Figure 2.10: The ModDrop Architecture.

2.3 Knowledge Distillation

Building on the ideas of Caruana *et al.* [5], Hinton *et al.* [19] showed that knowledge distillation can be used for compressing the knowledge in an ensemble into a single model. They also show that the acoustic model of a heavily used commercial system can significantly improve by distilling the knowledge from an ensemble of models into a single model. [41] extends the idea of knowledge distillation to allow the training of a student that is deeper and thinner than the teacher, using not only the outputs but also the intermediate representations learned by the teacher as hints to improve the training process and final performance of the student. The student's intermediate hidden layer are mapped to the prediction of the teacher's hidden layer using additional parameters, since student's hidden layers are generally smaller than that of the teacher's. [6] utilizes a state-of-the-art RNN to transfer knowledge to small Deep Neural Network (DNN). They use the RNN model to generate soft alignments and minimize the Kullback-Leibler divergence [29] against the small DNN. They show their performance on the task of Automatic Speech Recognition. Che *et al.* [7] shows how distilling knowledge from deep networks can have applications in healthcare domain as well. It introduces a novel knowledge-distillation approach called Interpretable Mimic Learning, to learn interpretable phenotype features for making robust prediction while mimicking the performance of deep learning models. Their framework uses Gradient Boosting Trees to learn interpretable features from deep learning models such as Stacked Denoising Autoencoder and LSTM. Their model obtains similar or better performance than the deep learning models at providing interpretable phenotypes for clinical decision making, on a real-world clinical time-series dataset.

In this chapter we saw some of the existing works on action recognition, gesture recognition and knowledge distillation. In the upcoming chapters we will be diving into our contribution towards it.

Chapter 3

Gesture Recognition

3.1 Method Overview

As baseline models we use a 3D-CNN or an LSTM variant of RNN to classify each gesture. For 3D-CNN each gesture videos need to be represented using a fixed number of frames. For LSTM the raw frames are flattened and passed through a linear layer to obtain features that are then fed to the LSTM. But both 3D-CNN and LSTM have their own set of advantages and disadvantages. Convolutional Neural Networks are very good at identifying and learning patterns over short spatio-temporal dimensions. They preserve both spatial and temporal information over the layers and can work with raw frame inputs but only over a shorter duration. Also the inputs need to be of fixed size, which may not always hold true in case of videos. They may well differ in the number of frames. LSTM overcomes both these problems. The gating mechanism in LSTM helps to preserve information over longer duration and it can work with variable length inputs as well. However, they do not preserve patterns because of the internal fully-connected connections. Hence, instead of feeding lowest level pixel information to it, LSTM works better when fed with features that already have some amount of spatio-temporal information embedded in it.

We look to leverage the best of both Convolutional Neural Network and Recurrent Neural Network in learning spatio-temporally rich features. We present a framework that combines the 3D-CNN with LSTM. The 3D-CNN acts as an encoder at sub-gesture level and provides representations for groups of few frames. These are then fed as sequences to the LSTM to get the final prediction.

3.2 Baseline Models

3.2.1 3D-CNN

Each gesture is represented using 32 consecutive frames, each of dimension 64×64 and two channels: depth and grayscale. For videos longer than 32 frames, the central 32 frames are extracted, while shorter videos are zero-padded. The model consists of six convolution layers, followed by two fully-

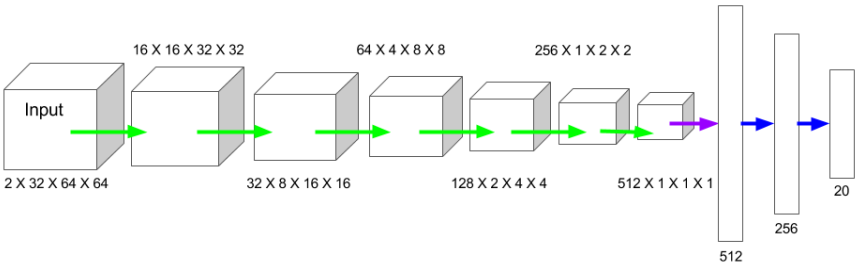


Figure 3.1: Baseline 3D-CNN Model. Each block shows dimensions in the format (*channels* $\times$ *number of frames* $\times$ *height* $\times$ *width*). Green: Conv layers, Purple: Flattening, Blue: FC layers.

connected layers. Each layer is followed by ReLU activation. Convolution layers are additionally followed by batch normalization. We do not use any pooling layer and sub-sampling is performed using padded convolution by varying the strides. For down-sampling by a factor of two, convolution is performed with filter size 4, stride 2 and padding 1 on each side. Whereas, to keep the size constant during convolution we use filter size 3, with stride 1 and padding 1 on each side. Figure 3.1 illustrates the model.

3.2.2 LSTM

Each video is divided into chunks of 4 frames of dimension 64×64 each, and grayscale, depth channels. After flattening, it is passed through a 512 dimensional linear layer, followed by ReLU activation and batch normalization. When we perform flattening and pass it through a linear layer, we lose the spatial and temporal information, an aforementioned drawback of this particular approach. Sequences of 512 dimensional feature vectors are fed into the LSTM, containing 256 hidden units, for final classification.

3.3 Joint 3D-CNN and LSTM

The joint architecture can be thought of as learning a two-step hierarchy, where gestures are made up of sub-gestures. The 3D-CNN learns representations at sub-gesture level from raw frame inputs, followed by the LSTM which makes prediction by modeling sequences of sub-gesture features. Thus we use the capability of 3D-CNN to model shorter duration of spatio-temporal information from raw frames to learn sub-gesture representations. The LSTM, which are good in modeling long sequences, takes these representations sequentially as input to classify the entire gesture videos.

We explore two different ways of training the joint model. In the first approach it is trained in an end-to-end fashion. Next we pre-train the encoder 3D-CNN before including it in the final model pipeline.

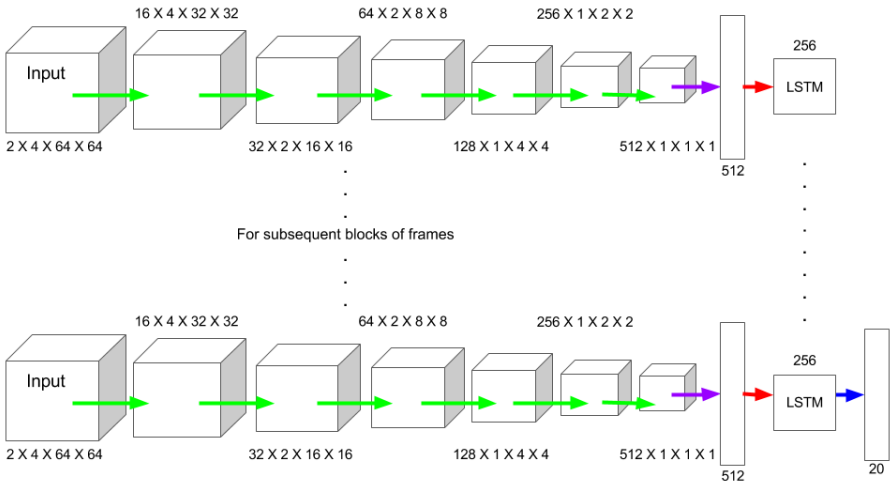


Figure 3.2: Joint 3D-CNN and LSTM Model. Each block shows dimensions in the format (*channels* $\times$ *number of frames* $\times$ *height* $\times$ *width*). Green: Conv layers, Purple: Flattening, Red: LSTM connection at each time-step, Blue: FC layers.

3.3.1 End-to-end Training

The model consists of an encoder 3D-CNN and an LSTM for sequence classification. The encoder takes blocks of $4 \times 64 \times 64$ frames of two channels (depth and grayscale) as input and produces a 512 dimensional feature vector of it. The LSTM takes those vectors sequentially for an entire gesture as input and gives the final prediction. Architecture of the 3D-CNN is similar to the one used in our baseline model, except the number of frames in each block (4 *versus* 32), and the LSTM consists of 256 hidden units. The complete model is shown in Figure 3.2.

Thus the encoder is acting on the raw pixels and learning spatio-temporally rich features which are used by the LSTM. The entire model in trained end-to-end directly from the class labels.

3.3.2 Supervised Pre-training of 3D-CNN

In our second approach we pre-train the 3D-CNN using blocks of frames at sub-gesture level as inputs and class labels as output. Once pre-training is done, it is used as a feature extractor from the input videos. The extracted features are used to train the LSTM in subsequent step. We did not observe much difference with and without fine-tuning the extractor 3D-CNN while training the joint model. The reason could be since it is also trained on the exact same dataset, hence fine-tuning did not give any noticeable boost in learning.

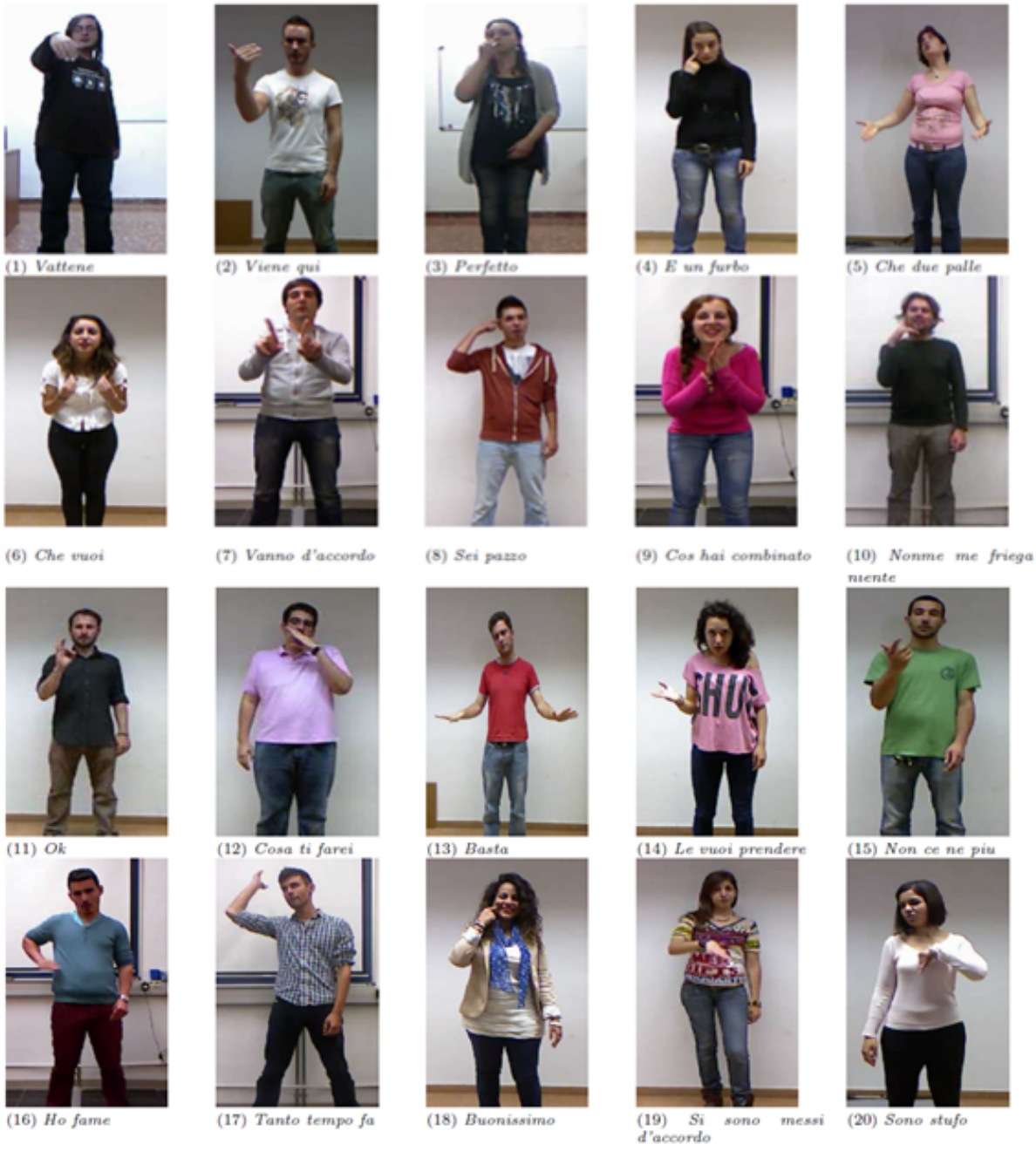


Figure 3.3: Twenty Classes of the Chalearn 2014 Looking at People Dataset.

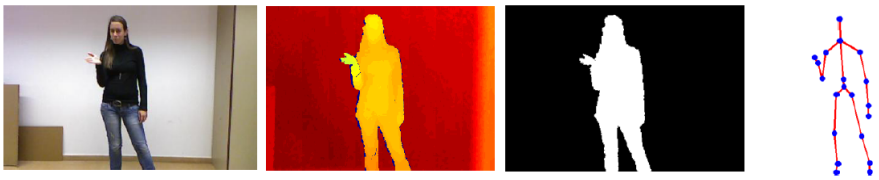


Figure 3.4: Available Frame Modalities of the ChaLearn Dataset.
Left to Right: RGB, Depth, User Mask, Skeleton Joint Information.

3.4 Experiments and Results

3.4.1 Dataset

The Chalearn 2014 Looking at People Challenge (track 3) [11] dataset is a vocabulary of 20 different Italian cultural/ anthropological signs, performed by 27 different users. The camera is still and focuses on the user's upper-body and for each gesture there is only one user present in the camera's field of view. As illustrated in Figure 3.3, there is no constraint among the gestures. The dataset consists of gestures performed using both hands as well as using only one hand. One-handed gestures can also be performed using either left or right hand. Additionally, there are variations in surroundings, clothing, lighting, exposure resolution, skin color and gesture movement.

It consists of 7,754 gestures for training, 3,362 gestures for validation and 2,742 gestures for testing. The videos are recorded using Microsoft Kinect at 20 frames per second and consist of RGB, depth, user mask and skeleton information for each frame of video as shown in Figure 3.4. The frames are of 640×480 resolution with depth information provided in millimeters. The skeleton information consists of joint positions (in world coordinates as well and relative pixel coordinates) and orientations. The 20 joint types whose information are provided are: hip center, spine, shoulder center, head, shoulder left, elbow left, wrist left, hand left, shoulder right, elbow right, wrist right, hand right, hip left, knee left, ankle left, foot left, hip right, knee right, ankle right, and foot right.

Each video may contain multiple gestures, performed one after another. The training and validation data consists of soft segmented annotation for the gestures in the videos, but no such annotation is available for the testing data. Since the main aim of this work is gesture classification, not segmentation, we use the validation data as our testing data and report accuracies on the same here-on. To maintain parity, we compare against the accuracies obtained on the validation data by other methods also. Further, we randomly choose 2,000 videos from training set to act as our validation data.

3.4.2 Input

For each video frame we convert the RGB frames to grayscale. The grayscale and depth frames are further multiplied with the provided user mask. This helps to get rid of the background noises and extract out the region of interest. We also apply a Gaussian smoothing with kernel size of (3×3) on the frames to smoothen out noise and missing pixel information. Finally we concatenate them across

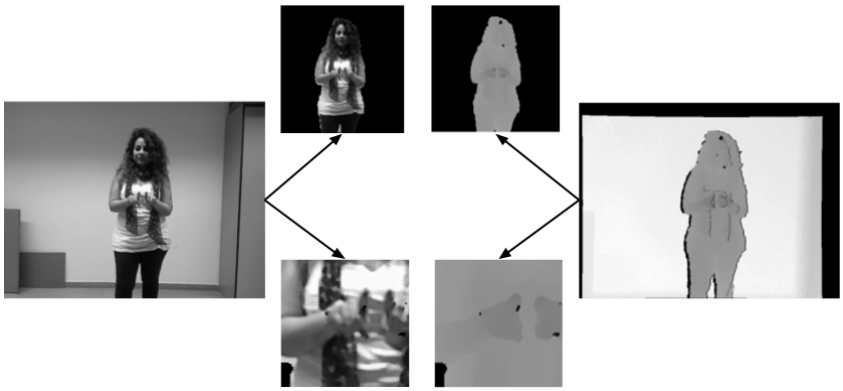


Figure 3.5: Left and right images are example grayscale and depth frames from the dataset. Center columns show the upper-body and hand input frames for our models obtained from it.

channel to get the inputs to our models. Since all the gestures occur in the upper-body region, we extract it using the skeleton information. Similar to [38], the area to be cut is determined using the following formula:

$$\begin{aligned} h &= |HipCenter - Head| \\ x_{left} &= HipCenter_x - h \\ x_{right} &= HipCenter_x + h \\ y_{top} &= HipCenter_y - 1.4h \\ y_{bottom} &= HipCenter_y + 0.7h \end{aligned} \tag{3.1}$$

The extracted upper-body portions are resized to 64×64. Thus inputs to our models are of dimension $2 \times T \times 64 \times 64$, where T is 32 for the baseline 3D-CNN model and 4 for the joint model (as mentioned in section 3.2.1 and 3.3.1 respectively), and 2 represents the two channels, grayscale and depth.

Since the gestures are performed using the hand(s), it turned out to be important to focus on the hand regions. We further observed that gestures involving both the hands are mirrored motion across the vertical plane. That is, both the left and right hands perform the same motion. Thus, we focus on extracting only one hand region (left in our case). Since there is no restrictions on the way the gestures are performed, using skeleton information we track the highest hand region over the entire duration for each gesture and if it is performed using the right hand, we mirror the video across the vertical axes. The size of the area to be cut is centered about the hand joint and determined using the below mentioned formula. It is then resized to 64×64.

$$l = \frac{|HipCenter - Head|}{2} \tag{3.2}$$

Table 3.1 shows accuracies obtained with our baseline 3D-CNN model while using hand, upper-body and both combined, as input. Since the combination of upper-body and hand gives the best performance,

Input mode	Accuracy(%)
Only Hand	87.33
Only Upper-body	87.59
Combined Hand and Upper-body	90.13

Table 3.1: Accuracies obtained using baseline 3D-CNN on different types of input modalities.

all the experiments use it as input (Figure 3.5). We also perform the following data augmentation techniques on each video, to increase the amount of training data and boost the models' performance.

- Rotate all the frames by a value in the range of $[-4, +4]$ degrees.
- Translate all the frames by number of pixels in the range of $[-5, +5]$, both vertically and horizontally.
- Zoom in the frames by number of pixels in the range of $[0, 6]$.

We do not perform horizontal or vertical flipping since our data is not horizontal or vertical orientation invariant.

3.4.3 Results and Analysis

We develop all the above mentioned models using the Torch[1] deep-learning framework. The models are trained using *stochastic gradient descent* optimizer, to minimize *multi-class cross-entropy* or *negative log-likelihood* loss. We used fixed *learning rate* of 0.005, *momentum* of 0.9, *weight-decay* regularization of 10^{-6}, and *batch size* of 32 across all the experiments.

Table 3.2 shows the accuracies obtained using our models. In alignment with our discussion in section 3.1, we observe that the baseline LSTM performs the worst since the spatial and temporal information are not conserved while flattening out and because of the internal fully-connected layers. Baseline 3D-CNN preserves such information across dimensions and performs better, but it is not suitable for capturing long-term dependencies. Our joint models perform better than the two baseline models. Pretraining the 3D-CNN encoder outperforms end-to-end training from scratch of the 3D-CNN and LSTM. Figure 3.6 shows how the individual models learn over the epochs.

We compare our best model against other *state-of-the-art* methods on the same dataset in Table 3.3. Wu *et al.* [53] described a novel method called Deep Dynamic Neural Networks, a semi-supervised hierarchical dynamic framework based on a Hidden Markov Model (HMM) and learned high-level spatio-temporal representations using a Deep Belief Network (DBN) and 3D-CNN, suited to the input modality. Pigou *et al.* [38] used an approach similar to our baseline 3D-CNN approach, but their architecture is very different from our CNN. ModDrop [36] used a multi-scale and multi-modal deep learning

[1] http://torch.ch/

Method/Model	Accuracy(%)
Baseline LSTM	86.57
Baseline 3D-CNN	90.13
Joint 3D-CNN + LSTM (End-to-end trained)	93.18
Joint 3D-CNN + LSTM (Pre-trained 3D-CNN)	**96.45**

Table 3.2: Accuracies obtained using our different model architectures.

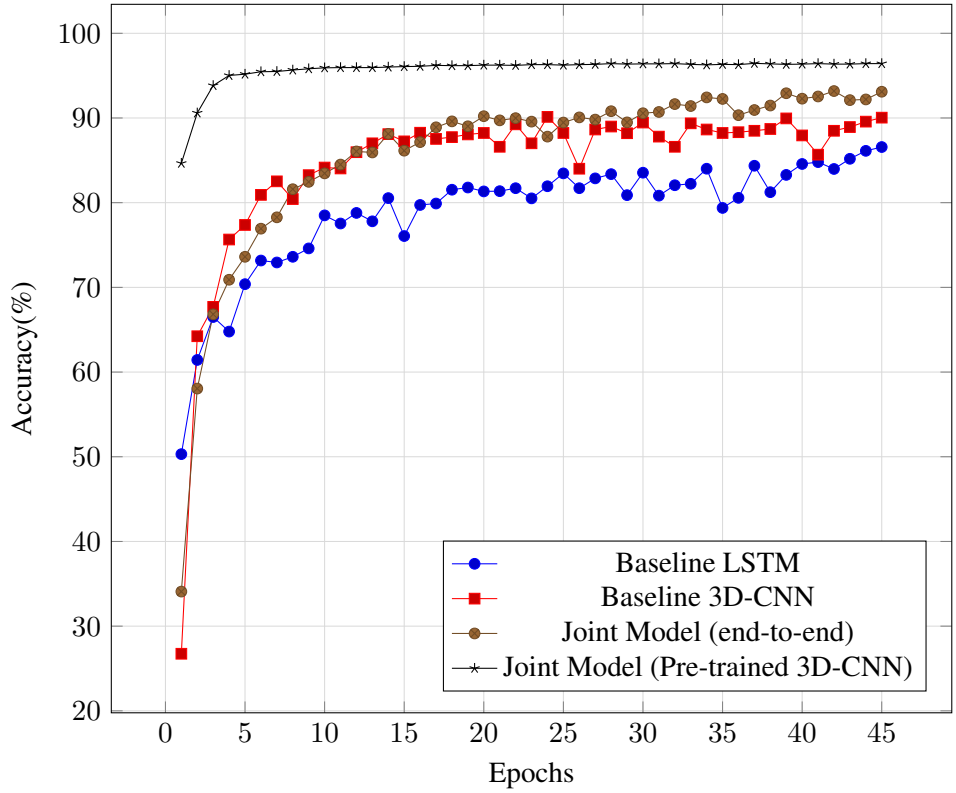


Figure 3.6: Epochs versus Accuracy(%) of our Models on Test Data.

approach. It performed careful initialization of individual modalities and fused multiple modalities at several spatial and temporal scales. Their architecture is a big cascade having individual branches for each hand, each modalities, and articulated pose based features. Our simpler architecture works directly with the raw pixels, without the need to perform any input-specific initializations.

3.5 Summary

In this chapter we look at various models to perform gesture recognition from videos. 3D-CNN, though is very efficient at finding patterns in data, has difficulty in modeling long sequences. LSTM on the other hand can learn long term sequences but loses spatial and temporal information. We show

Method/Model	Accuracy(%)
Wu *et al.* [53]	87.9
Pigou *et al.* [38]	91.4
Joint 3D-CNN + LSTM (ours)	**96.4**
Neverova *et al.* [36]	**96.8**

Table 3.3: Accuracy comparison of our best model with other *state-of-the-art* methods.

how joint 3D-CNN and LSTM model for gesture recognition from videos, leverages the best of both 3D convolution and recurrent network to model the sequential evolution of information in a video, while allowing to process arbitrary length videos. In the next chapter we will see that how beside the models being robust, using techniques described in this chapter, we can also make them more compact. Learning compact models, with fewer parameters, will benefit inference time as well as space requirement of the models.

Chapter 4

Learning Compact Models

4.1 Overview

Deep Learning has had a major impact in the fields of computer vision, natural language processing and speech recognition over the past few years. However, one of the major issues with Deep Learning models are the large number of parameters involved which makes training time and memory requirements quite high. This makes it difficult to employ them in embedded devices. Keeping this in mind, we would like to explore methods to reduce the parameter space of such models without compromising a lot on performance.

To this end, we have a look at the following two ways of learning compact models that not only benefits training time but also makes it possible to use them in low-memory and low-power embedded devices. It enables us to perform faster inferencing on hand-held devices where power and memory are constraints.

- Distill knowledge from a larger network, called teacher network, to a smaller student network, containing lesser number of parameters. The smaller student network if trained from scratch using class-labels showed inferior learning capacity when compared to training using teacher supervision.

- We demonstrate that a particular way of training enables to push a lot of weights to very low values, negligible values. Pruning them helps us in obtaining sparser models, with no effect as such on performance. We illustrate it's effectiveness on our task as well as on Cifar-10 and Cifar-100 datasets [27].

Next we describe the aforementioned two processes and illustrate it's usefulness through thorough experimentations.

4.2 Knowledge Distillation

4.2.1 Method

Our aim is to develop models that are efficient in terms of memory and speed. Knowledge distillation is a process of transferring knowledge from one model to another. In order to obtain a faster inference, we train a student network, from the softened output of a larger teacher network. The idea is to allow the student network to capture not only the information provided by the true labels, but also the finer structure learned by the pre-trained teacher network.

In our case, we train our baseline 3D-CNN model till convergence and use it as the teacher network. As suggested in [18], for each gesture we obtain softened output of softmax from our teacher model using:

$$P_i = \frac{e^{\frac{Z_i}{T}}}{\sum_{j=1}^{c} e^{\frac{Z_j}{T}}}, \forall i \in \{1, ...c\}, \tag{4.1}$$

where $c(= 20)$ is the number of classes and T is the temperature, set depending on how "soft" we want the distribution to be.

For our student networks we train smaller variants (referred to as *medium* and *small*) of our joint model obtained by reducing the number of feature maps (channels) in each layer by two and four times, respectively. There are two widely used methods for training the student network.

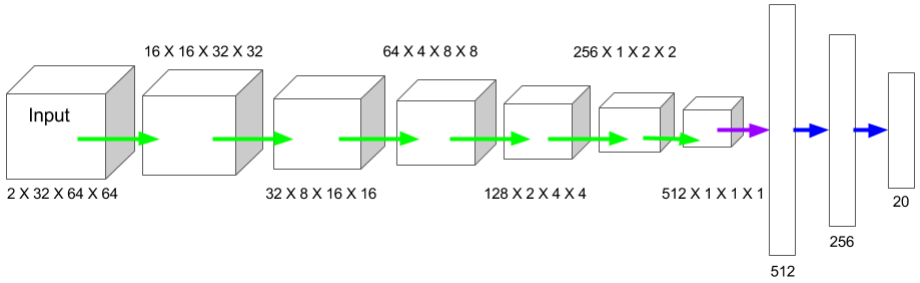
- The model is pre-trained on $L^{(soft)}$, using teacher outputs, and then fine-tuned on $L^{(hard)}$, using the actual class-labels.
- Train simultaneously using both teacher outputs and actual class-labels. The aim is to minimize the following loss function:

$$L = \alpha L^{(soft)} + (1 - \alpha) L^{(hard)}, \tag{4.2}$$

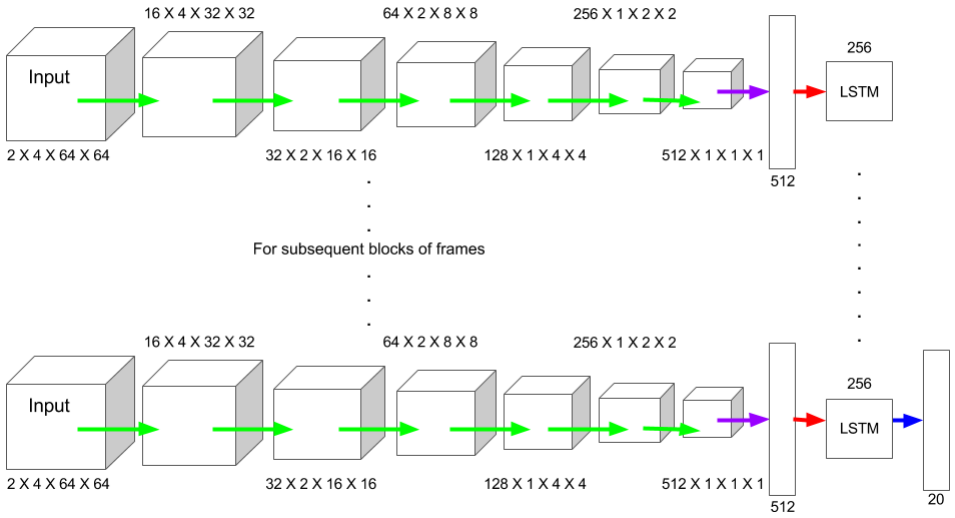
where $L^{(soft)}$ is the mean-squared error between pre-trained teacher's softened softmax output and student's softmax output, $L^{(hard)}$ is the cross-entropy loss between the actual class label and student output, and α is a weighting parameter (set as 0.5 in our experiments). Figure 4.1 shows our baseline 3D-CNN model which is used as the teacher network and the original joint architecture based on which the student networks are derived. Figure 4.2 shows our two student networks, *medium* and *small*, derived from the joint model as mentioned above.

4.2.2 Experiments and Results

We use similar experimental setup and input as explained in section 3.4.2. Further, the models are trained using *stochastic gradient descent* optimizer with fixed *learning rate* of 0.005, *momentum* of



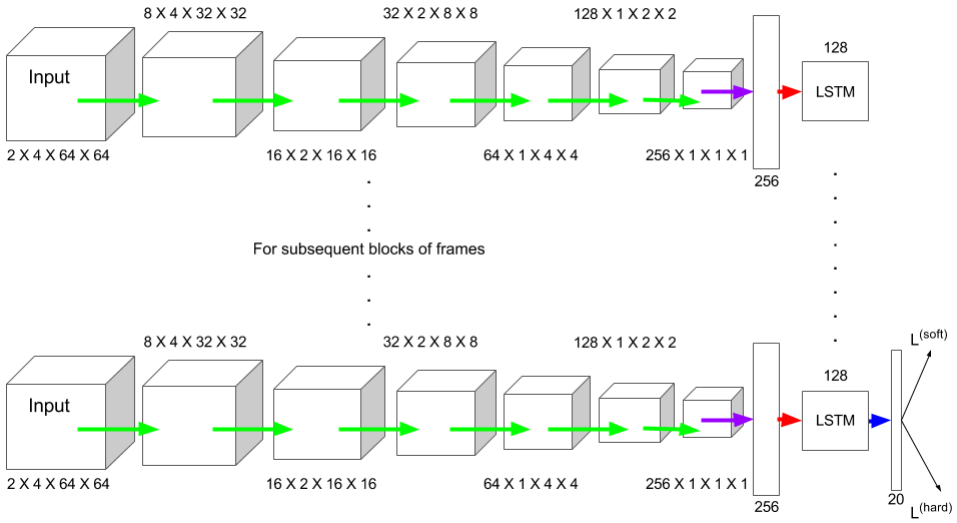
(a) *Teacher* 3D-CNN Model



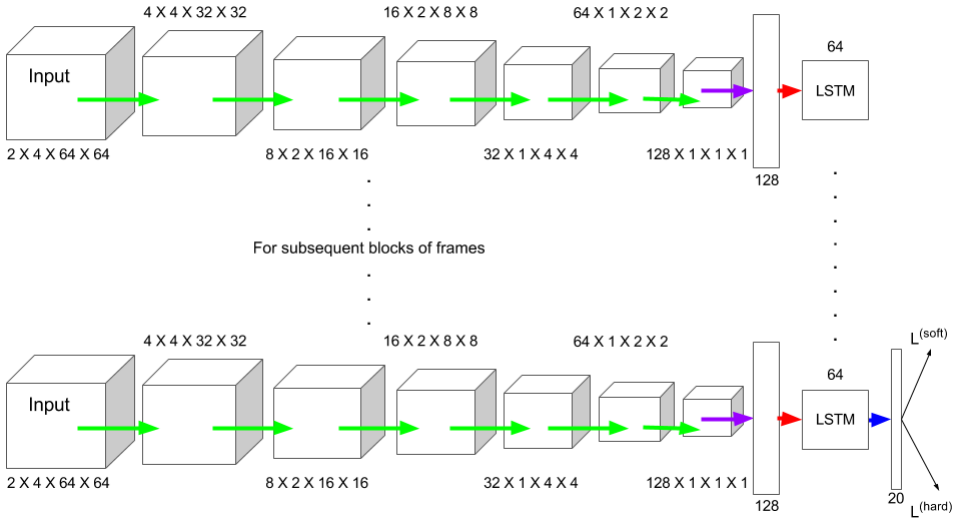
(b) *Original* Joint 3D-CNN and LSTM Model

Figure 4.1: Each block shows dimensions in the format (*channels* $\times$ *number of frames* $\times$ *height* $\times$ *width*). Green: Conv layers, Purple: Flattening, Red: LSTM connection at each time-step, Blue: FC layers.

0.9, *weight-decay* regularization of 10^{-6}, and *batch size* of 32 across all the experiments. The effect of Knowledge Distillation is analyzed in Table 4.1. Reducing the number of parameters limits the model's capacity to learn from the class labels directly, but it can be alleviated when trained according to Equation 4.2, using soft outputs from a larger pre-trained teacher network and actual class-labels. We get almost at-par accuracy with the teacher, but at a much reduced training cost, making it both computationally and memory efficient. Since these models are end-to-end trained, to maintain consistency we compare it against our original joint model trained end-to-end (section 3.3.1), and not the pre-trained joint model (section 3.3.2).



(a) *Medium*



(b) *Small*

Figure 4.2: Student Joint 3D-CNN and LSTM Networks. Each block shows dimensions in the format (*channels* $\times$ *number of frames* $\times$ *height* $\times$ *width*). Green: Conv layers, Purple: Flattening, Red: LSTM connection at each time-step, Blue: FC layers, Black: Loss.

Further, in Table 4.2 we compare the actual space taken up by the model when saved on disk. We also show how converting the model to half-precision floating point format, instead of saving in single-

	Model	# of parameters (in millions)	Trained using	Accuracy(%)
Original	3D-CNN + LSTM	18.37	class labels	93.18
Teacher	3D-CNN	18.82	class labels	90.13
Student	3D-CNN + LSTM *(medium)*	**4.59**	class labels	86.18
			class labels and softmax output of *teacher*	**88.35**
	3D-CNN + LSTM *(small)*	**1.15**	class labels	81.50
			class labels and softmax output of *teacher*	**86.05**

Table 4.1: Knowledge Distillation from baseline 3D-CNN to Joint 3D-CNN+ LSTM Model.

Method	# of parameters (in millions)	Single-precision		Half-precision	
		Model size (MB)	Accuracy(%)	Model size (MB)	Accuracy(%)
Teacher 3D-CNN	18.82	72	90.13	36	89.5
Original 3D-CNN + LSTM	18.37	71	93.18	35.5	93.18
Student 3D-CNN + LSTM *(medium)*	4.59	19	88.35	9.5	88.35
Student 3D-CNN + LSTM *(small)*	1.15	4.5	86.05	2.25	85.98

Table 4.2: Reduction is size and impact of performance on the teacher and student models.

point precision helps in reducing the model size further with almost no effect on accuracy. The model is trained as usual, but the weights are pruned to half-precision before saving the final model to disk.

4.3 Weight Pruning

4.3.1 Method

Typical CNNs contain millions of nodes and connection weights. Sometimes not all the weights contribute significantly to the model's performance. In such cases, we can prune them as it helps in reducing the size of the model drastically. Weight pruning involves removing weights, having very low values, from the final trained model since they contribute very little to the model's performance. From the viewpoint of neural network computation, weight pruning is equivalent to removing the low valued entries from the weight matrices. The trained model can be saved using sparse representation consuming lesser disk space.

In all the experiments up till now, the neural networks are trained using the Stochastic Gradient Descent (SGD) optimizer, with momentum. Adaptive Moment Estimation (Adam) [25] is an optimization method that computes adaptive learning rates for each parameter, unlike SGD, which has the same learning rate and update rule for all the parameters. We defer the actual derivations of the update rules to the Appendix 6.4, to maintain brevity.

We observe that when a network is trained using the update rule of Adam, combined with weight-decay, it encourages a lot of weights to become very low by penalizing high weight values and adjusting

the learning rate accordingly. This enables us to remove low-valued weights to obtain sparser models, compared to SGD (with weight-decay as well) trained models. In the following sections we empirically verify and compare this observation using extensive experimentations with our networks on the ChaLearn 2014 dataset, as well as using well-known architectures like ResNet [17], VGG [47] and Network In Network (NIN) [33] on the CIFAR-10 and CIFAR-100 datasets [27].

4.3.2 Experiments and Results

ChaLearn 2014

We train our joint 3D-CNN and LSTM models using Adam on the ChaLearn 2014 dataset. The original model is trained in an end-to-end fashion as given in Section 3.3.1 and the *medium* and *small* variants of it are trained using the knowledge distillation approach as described in Section 4.2.1. In Figure 4.3 we compare the weight distribution of the trained networks when trained using Adam with those trained using SGD. For training with Adam, we use an initial *learning rate* of 0.001 with *weight-decay* regularization of 10^{-6} and *batch size* of 32 across all the experiments.

As evident from Figure 4.3, Adam pushes a lot of weight parameters to take very low values, unlike SGD. We choose $\log_2(-20)$ as a threshold value and prune all the weight connections whose absolute values are lesser than that. Table 4.3 shows the reduction in the number of parameters of the final trained model after pruning. We get models with $\approx 80\%$ lesser weight connections than that of models trained using SGD. Removing those and saving the models using sparse representation, can help in saving up a lot of disk space.

CIFAR-10 and CIFAR-100

The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The CIFAR-100 dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

Model	**Training with *SGD***		**Training with *Adam*** (with pruning)		
	# of parameters (in millions)	**Accuracy(%)**	**# of parameters** (in millions)	**Accuracy(%)**	**Parameter reduction(%)**
Original	18.37	93.18	3.94	92.3	78.56
Medium	4.59	88.35	0.98	88.98	78.71
Small	1.15	86.05	0.26	85.48	77.41

Table 4.3: Reduction in Number of Parameters of the Variants of Joined 3D-CNN and LSTM Model on ChaLearn-2014 Dataset after Weight Pruning.

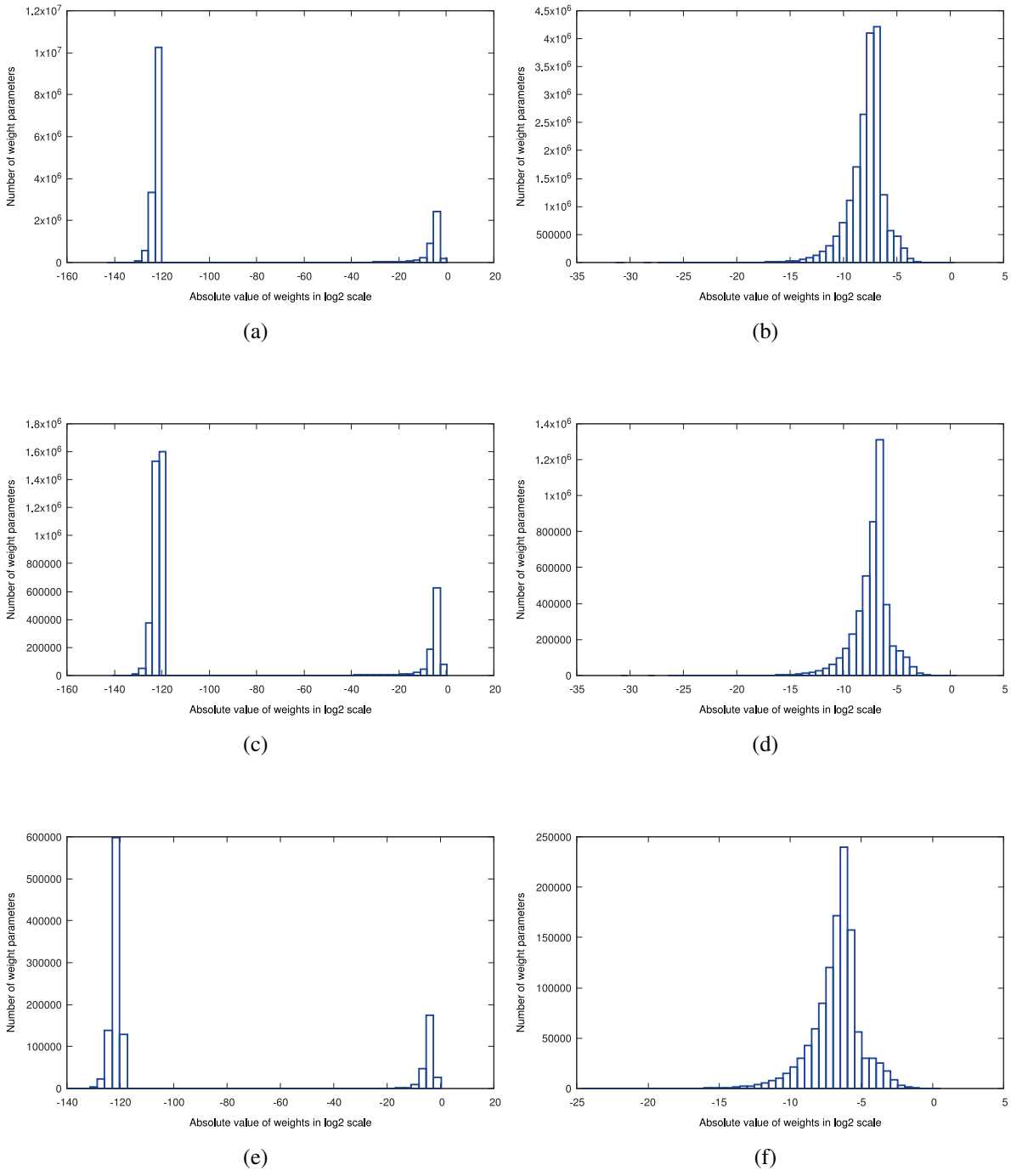


Figure 4.3: Weight Distributions of the Joint 3D-CNN and LSTM Networks on ChaLearn-2014 Dataset. Normal [(a) and (b)], Medium [(c) and (d)], Small [(e) and (f)]. Left Column [(a),(c),(e)] are trained using Adam. Right Column [(b),(d),(f)] are trained using SGD.

Model	Training with *SGD*		Training with *Adam* (with pruning)		
	# of parameters (in millions)	**Accuracy(%)**	**# of parameters** (in millions)	**Accuracy(%)**	**Parameter reduction(%)**
ResNet	0.49	86.39	0.41	85.62	16.5
NIN	1.00	86.53	0.89	85.74	11.60
VGG	14.99	89.46	3.57	89.16	76.19

Table 4.4: Reduction in Number of Parameters on CIFAR-10 Dataset after Weight Pruning.

Model	Training with *SGD*		Training with *Adam* (with pruning)		
	# of parameters (in millions)	**Accuracy(%)**	**# of parameters** (in millions)	**Accuracy(%)**	**Parameter reduction(%)**
ResNet	0.49	56.74	0.45	57.59	8.18
NIN	1.02	60.68	0.92	59.58	9.85
VGG	15.04	63.02	7.02	63.25	53.33

Table 4.5: Reduction in Number of Parameters on CIFAR-100 Dataset after Weight Pruning.

We analyze the effects of training with Adam and SGD, on the CIFAR datasets by training three popular network architectures on them: ResNet, Network In Network (NIN), and VGG. For training with Adam, we use an initial *learning rate* of 0.003 with *weight-decay* regularization of 10^{-6} and *batch size* of 128 across all the experiments. For training with SGD, we use a fixed *learning rate* of 0.3 with *weight-decay* regularization of 10^{-6}, *momentum* of 0.9 and *batch size* of 128 across all the experiments. Figure 4.4 and Figure 4.5 shows the distribution of weights for the above specified networks when trained on CIFAR-10 and CIFAR-100 respectively. Again we observe similar distribution pattern like we observed in the case of ChaLearn dataset, where Adam pushes a lot of weights to take very low values. The difference is much more prominent in case of the VGG networks which has almost 30 times and 15 times more number of parameters than ResNet and NIN, respectively. Even though larger model ensures higher accuracy, but it also suggests that there is much more scope for obtaining a corresponding sparser model (Table 4.4 and Table 4.5).

Similar to the previous section, we analyze the effects of weight pruning on the three networks trained on the CIFAR datasets. We remove those connections of the networks, whose absolute weight values are lesser than $\log_2(-20)$. Table 4.4 and 4.5 show the reduction in weight parameters. VGG shows the highest percentage of parameter reduction, while achieving the best accuracy among the three networks, when trained using Adam.

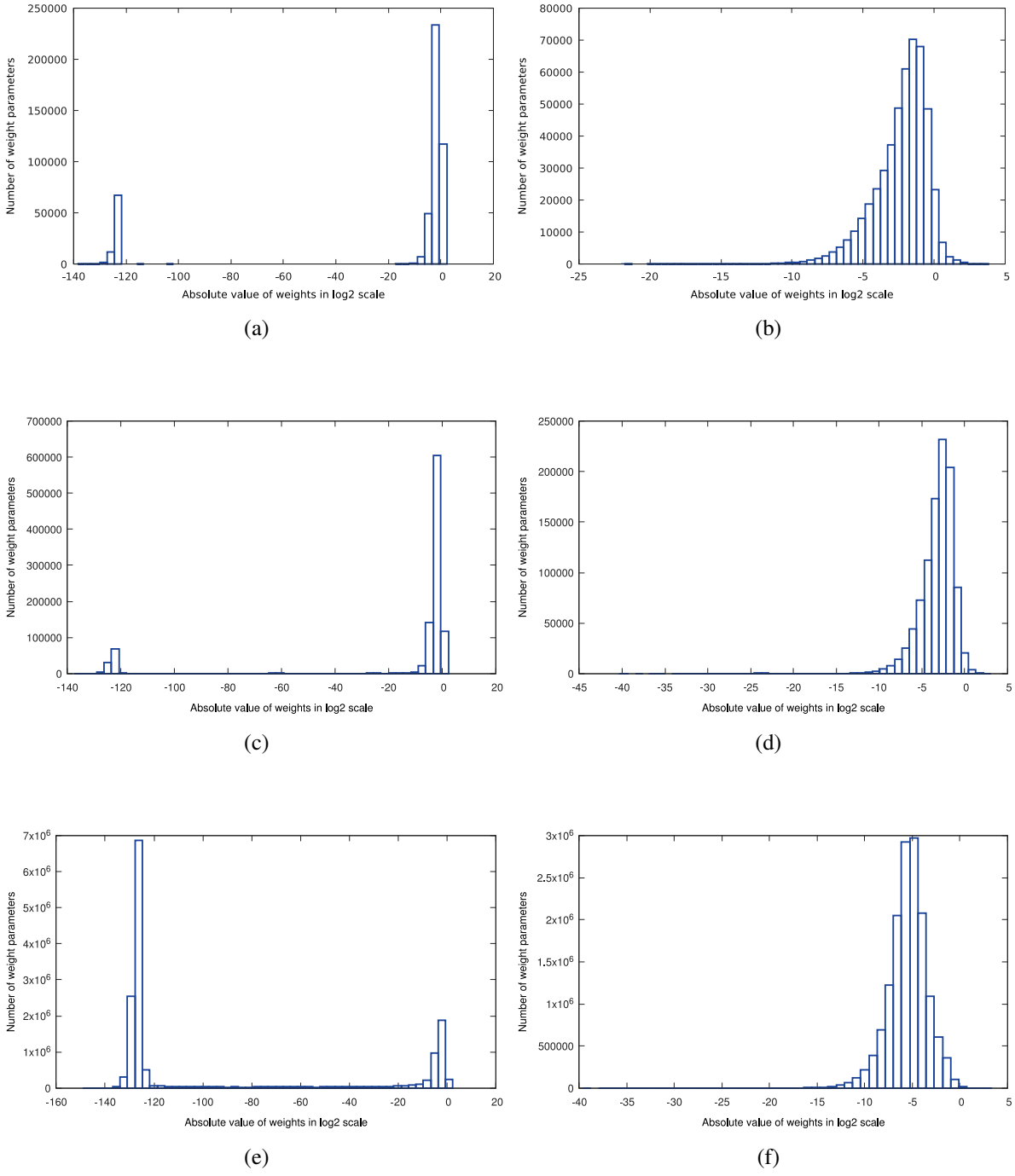


Figure 4.4: Weight Distributions on CIFAR-10 Dataset. ResNet [(a) and (b)], NIN[(c) and (d)], VGG [(e) and (f)]. Left Column [(a),(c),(e)] are trained using Adam. Right Column [(b),(d),(f)] are trained using SGD.

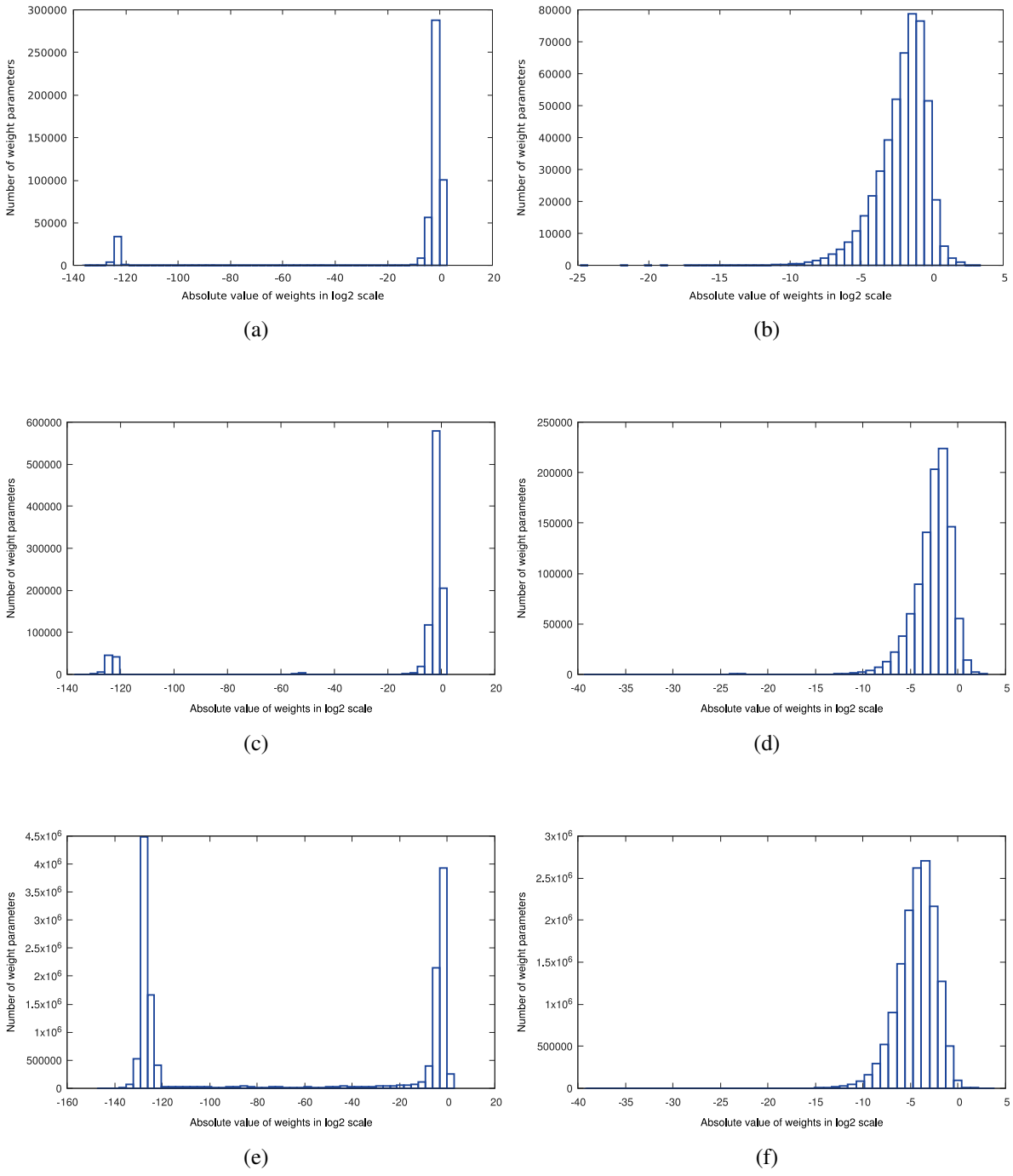


Figure 4.5: Weight Distributions on CIFAR-100 Dataset. ResNet [(a) and (b)], NIN [(c) and (d)], VGG [(e) and (f)]. Left Column [(a),(c),(e)] are trained using Adam. Right Column [(b),(d),(f)] are trained using SGD.

4.4 Summary

In this chapter we explored two approaches to obtain compact deep models which are fit to be deployed and used for real-time predictions on embedded devices. In the first approach we look at the idea of knowledge distillation. In this we train a large model (or can also be an ensemble of various models) till it reaches convergence. Once trained, we use the softened softmax output of this trained model, which can also be referred to as the teacher network, for guiding a smaller student network. The student network if trained from scratch using normal supervision of just the class-labels failed to perform at the same level as when trained with teacher supervision. Thus we were able to *distill* the knowledge of a large model to a much smaller and compact model.

Secondly, we show that how training with Adam, instead of SGD, helps in obtaining sparser models with fewer connections between the nodes of the network. An important point to be noted is that Adam in itself did not perform as efficiently in reducing the number of connections as it did when combined with weight decay. We empirically show with extensive and detailed experiments that this simple strategy of changing the optimization method helps us in learning compact models with almost equivalent performance of an SGD trained counterpart.

Chapter 5

Conclusion

Deep learning working architectures, specifically those built for computer vision, began with the Neocognitron introduced by Kunihiko Fukushima in 1980 [13]. In 1989, Yann LeCun *et. al.* [31] applied the standard backpropagation algorithm to a deep neural network with the purpose of recognizing handwritten ZIP codes on mail. It also introduced the Covolutional Neural Networks, as we know it today. While the algorithm worked, training required 3 days. Hinton *et. al.* [20] in 2006 showed how a many-layered feedforward neural network could be effectively pre-trained one layer at a time, treating each layer in turn as an unsupervised restricted Boltzmann machine, then fine-tuning it using supervised backpropagation. Since the availability of massive amounts of training data and advent of computing resources like clusters of powerful cores and GPUs, deep learning really took off and started achieving state-of-the-art results in the areas of computer vision, speech recognition, natural language processing, and others. In computer vision almost all the current best results are obtained using deep learning approach. It includes tasks such as object detection, classification, localization, segmentation, image caption generation, to name a few.

In this thesis we deal with building a deep and compact model for gesture recognition. Gesture Recognition can be defined as a type of perceptual computing user interface that allows computers to capture and interpret human gestures. One application can be to use as an alternative user interface for providing real-time data to a computer, instead of typing with keys or tapping on a touch screen. However the problem that we deal with particularly in this thesis is to develop a model for sign language recognition, with potential application to perform real-time predictions.

The most important thing is to model the evolution of sequential temporal data over the frames of the videos, in order to perform good classification. We show that combining the 3D-CNN and LSTM gives the best result compared to the standalone individual models. The 3D-CNN learns to extract features from shorter durations of the videos and the LSTM performs the task of classification when fed sequentially with those features. A joined model, that is end-to-end trainable from raw video frames, is shown to be better suited to capture the dynamic information and it's evolution in gesture videos. Further, pre-training the 3D-CNN on the videos, before plugging it into the final joined model and fine-tuning, boosts the model's performance.

We also try to address the challenges faced in order to perform real-time inference in embedded devices and show how information can be distilled from a larger model to models with $16\times$ and $4\times$ fewer parameters. Training with a combination of the softened softmax outputs of a trained teacher network and the actual class labels helps in better and more efficient training of the student network. Size of the models could be further reduced using a sparse representation of models trained using Adam optimization with weight decay. This not only benefits training time but also makes it possible to use them in low-memory and low-power embedded devices. In future further studies need to be done in this aspect to understand what exactly results in such a behavior of pushing a lot of weights to near-zero values and can it be encouraged in some way to further reduce the number of weights in the final trained model. In future we wish to explore better ways to train a network that will allow to reduce the model size by encouraging further sparsification or quantization of the learned weights of the model.

Chapter 6

Appendix

6.1 Backpropagation in Artificial Neural Network

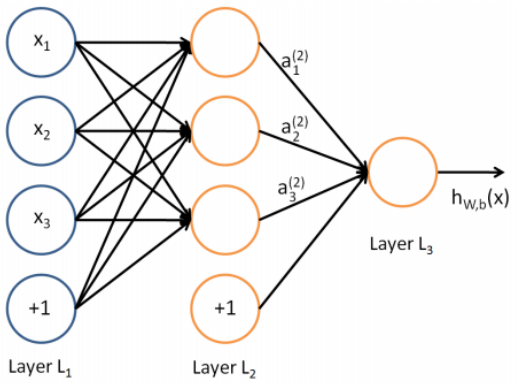


Figure 6.1: An MLP. Blue nodes are input nodes, middle orange nodes are hidden nodes, right orange node is output node and $(+1)$ nodes are called *biases*. Black arrows are weight connections between two consecutive layers which are automatically learned using backpropagation.

We will consider the three-layer neural network shown in Figure 6.1 and derive the weight update rule for all the layers. The forward-pass of the network is shown in Equations 6.1.

$$\begin{aligned} a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\ a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\ a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\ h_{W,b}(x) = a_1^{(3)} &= f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \end{aligned} \tag{6.1}$$

where $a_i^{(l)}$ denotes output of node i in layer l, $W_{ij}^{(l)}$ denotes weight from node j in layer $(l-1)$ to node i in layer l, $b^{(l)}$ is bias in layer l and $f()$ is a non-linear activation function. Refer to Figure 1.2 for further details.

Weights are updated during the backward-pass of training. Training error is the sum over output units of the squared difference between the desired output t_k and the actual output of the network given by z_k. It can be formulated as:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 \tag{6.2}$$

where c = number of output units. In our example (Figure 6.1), there is only one output unit $h_{W,b}(x) \equiv a_1^{(3)}$. Thus in our case,

$$J(\mathbf{w}) = \frac{1}{2}(t_1 - a_1^{(3)})^2 \tag{6.3}$$

The backpropagation learning rule is based on gradient descent. The weights are initialized with random values and then they are changed in a direction that will reduce the error, with *learning rate* η:

$$\triangle \mathbf{w} = -\eta \frac{\delta J}{\delta \mathbf{w}} \tag{6.4}$$

Let's consider the hidden-to-output weight $W_{11}^{(2)}$. The update rule of it can be written using the chain-rule as:

$$\frac{\delta J}{\delta W_{11}^{(2)}} = \frac{\delta J}{\delta net_1^{(2)}} \frac{\delta net_1^{(2)}}{\delta W_{11}^{(2)}} \tag{6.5}$$

where,

$$\delta net_1^{(2)} = W_{11}^{(2)}x_1 + W_{12}^{(2)}x_2 + W_{13}^{(2)}x_3 + b_1^{(2)} \tag{6.6}$$

That is $\delta net_k^{(l)} = \sum_{i=1}^{d} W_{ki}^{(l)}x_i + b_1^{(l)}$ where d is the number of hidden units, and l is the layer number. Now, we can write:

$$\frac{\delta J}{\delta net_1^{(2)}} = \frac{\delta J}{\delta a_1^{(3)}} \frac{\delta a_1^{(3)}}{\delta net_1^{(2)}} \tag{6.7}$$

Assuming the activation function $f(.)$ of Equation 6.1 is differentiable, from Equation 6.3 and 6.1, we get:

$$\frac{\delta J}{\delta a_1^{(3)}} = -(t_1 - a_1^{(3)}) \text{ and } \frac{\delta a_1^{(3)}}{\delta net_1^{(2)}} = f'(net_1^{(2)}) \tag{6.8}$$

From Equation 6.6:

$$\frac{\delta net_1^{(2)}}{\delta W_{11}^{(2)}} = x_1 \tag{6.9}$$

Thus combining Equations 6.5, 6.7, 6.8, 6.9, we can write:

$$\frac{\delta J}{\delta W_{11}^{(2)}} = -(t_1 - a_1^{(3)}) f'(net_1^{(2)}) x_1 \tag{6.10}$$

Therefore, from Equation 6.4 and 6.10, hidden-to-output weight $W_{11}^{(2)}$ can be updated as:

$$\triangle W_{11}^{(2)} = -\eta \frac{\delta J}{\delta W_{11}^{(2)}} = \eta(t_1 - a_1^{(3)}) f'(net_1^{(2)}) x_1 \tag{6.11}$$

We can obtain similar update rules for the weights $W_{12}^{(2)}$ and $W_{13}^{(2)}$ as well.

Likewise, the update rules for the input-to-hidden weights $W_{ij}^{(1)}$ can also be derived using chain-rule of backpropagation.

6.2 Backpropagation Through Time in Recurrent Neural Network

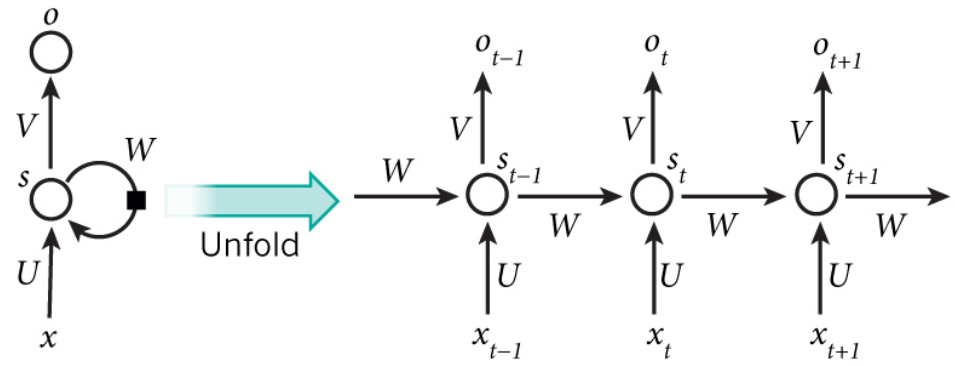


Figure 6.2: A Recurrent Neural Network and the unfolding in time of the computation involved in its forward computation.

Backpropagation through time (BPTT) is a gradient-based technique for training recurrent neural networks. We will consider the example shown in Figure 6.2. The forward-pass of the network can be written down as:

$$\begin{aligned} s_t &= f(Ux_t + Ws_{t-1}) \\ o_t &= softmax(Vs_t) \end{aligned} \tag{6.12}$$

If z_t is the correct label at time-step t, the cross-entropy loss is given by:

$$
\begin{aligned}
E_t(z_t, o_t) &= -z_t \log o_t \\
E(z, o) &= \sum_t E_t(z_t, o_t) = -\sum_t z_t \log o_t
\end{aligned}
\tag{6.13}
$$

Now, using chain-rule of differentiation, the gradients of the error with respect to our weights V can be written as:

$$
\frac{\delta E_t}{\delta V} = \frac{\delta E_t}{\delta o_t}\frac{\delta o_t}{\delta V} = \frac{\delta E_t}{\delta o_t}\frac{\delta o_t}{\delta y_t}\frac{\delta y_t}{\delta V} \tag{6.14}
$$

where, $y_t = Vs_t$. Therefore, combining Equations 6.12, 6.13 and 6.14 we get:

$$
\frac{\delta E_t}{\delta V} = (o_t - z_t) \otimes s_t \tag{6.15}
$$

where, $\otimes$ is the outer product of two vectors. Thus, the gradient with respect to V only depends on the value at t.

Similarly, using chain-rule of differentiation, the gradients of the error with respect to our weights W can be written as:

$$
\frac{\delta E_t}{\delta W} = \frac{\delta E_t}{\delta o_t}\frac{\delta o_t}{\delta s_t}\frac{\delta s_t}{\delta V} \tag{6.16}
$$

Now, from Equation 6.12, $s_t = f(Ux_t + Ws_{t-1})$. Thus s_t depends on W and s_{t-1}. So taking derivative with respect to W, s_{t-1} cannot be treated as a constant. Applying chain-rule again successively we get:

$$
\frac{\delta E_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta E_t}{\delta o_t}\frac{\delta o_t}{\delta s_t}\frac{\delta s_t}{\delta s_k}\frac{\delta s_k}{\delta W} \tag{6.17}
$$

Thus we get:

$$
\frac{\delta E_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta E_t}{\delta o_t}\frac{\delta o_t}{\delta y_t}\frac{\delta y_t}{\delta s_t}\frac{\delta s_t}{\delta s_k}\frac{\delta s_k}{\delta W} \tag{6.18}
$$

where, $y_t = Vs_t$.

$$
\frac{\delta E_t}{\delta W} = (o_t - z_t) \otimes V \sum_{k=0}^{t} \frac{\delta s_t}{\delta s_k}\frac{\delta s_k}{\delta W} \tag{6.19}
$$

where, $\otimes$ is the outer product of two vectors.

Since W is used in every step up to the output, we need to backpropagate gradients from $time = t$ through the network all the way to $time = 0$. This is exactly the same as the standard backpropagation algorithm we saw previously, in case of Feedforward Neural Networks. The key difference is that we sum up the gradients for W at each time step, hence the name "backpropagation through time". Similar results can be obtained for gradients with respect to U s well.

6.3 Vanishing Gradient Problem in Recurrent Neural Network

From the previous section, the gradient we obtained is given by (Equation 6.17):

$$\frac{\delta E_t}{\delta W} = (o_t - z_t) \otimes V \sum_{k=0}^{t} \frac{\delta s_t}{\delta s_k} \frac{\delta s_k}{\delta W} \tag{6.20}$$

It is to be noted that $\frac{\delta s_t}{\delta s_k}$ is a chain-rule in itself. That is, $\frac{\delta s_t}{\delta s_k} = \frac{\delta s_t}{\delta s_{t-1}} \frac{\delta s_{t-1}}{\delta s_{t-2}} \dots \frac{\delta s_{k+1}}{\delta s_k}$. Also, since we are taking the derivative of a vector function with respect to a vector, the result is a matrix (Jacobian matrix) whose elements are all the pointwise derivatives. The above gradient can be rewritten as:

$$\frac{\delta E_t}{\delta W} = (o_t - z_t) \otimes V \sum_{k=0}^{t} \left(\prod_{j=k+1}^{t} \frac{\delta s_j}{\delta s_{j-1}} \right) \frac{\delta s_k}{\delta W} \tag{6.21}$$

The 2-norm of the above Jacobian matrix has an upper bound of 1. Thus, with small values in the matrix and multiple matrix multiplications ($t - k$ in particular) the gradient values shrink exponentially fast, eventually vanishing completely after a few time steps. The RNN ends up not learning long-range dependencies. We can also get exploding instead of vanishing gradients if the values of the Jacobian matrix are large. Clipping the gradients at a pre-defined threshold is a very simple and effective solution to exploding gradients. Vanishing gradients are more problematic because it is not obvious how to deal with them.

6.4 Update Rules for Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) Optimization Methods

In Stochastic Gradient Descent (SGD) optimizer, for parameters of the network θ, the update rule at each epoch is given by:

$$\theta_{t+1} = \theta_t - v_t \tag{6.22}$$

where,

$$v_t = \eta \nabla_\theta J(\theta) \tag{6.23}$$

where, η is the learning rate and $\nabla_\theta J(\theta)$ is the gradient of the objective function $J(\theta)$ parameterized by model's parameters θ.

Momentum [40] is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ of the update vector of the past time step $(t-1)$ to the current update vector v_t:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{6.24}$$

γ, the momentum value, is usually set to 0.9 or a similar value. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

Adaptive Moment Estimation (Adam) [25] is an optimization method that computes adaptive learning rates for each parameter, unlike SGD which has the same learning rate for all the parameters. Adam keeps an exponentially decaying average of past gradients (m_t) and past squared gradients (v_t).

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) {g_t}^2 \end{aligned} \tag{6.25}$$

where, g_t is the gradient, β_1 and β_2 are parameters. m_t and v_t are estimates of the first moment (the mean) and the second moment (the un-centered variance) of the gradients respectively. To counteract the bias of m_t and v_t going towards 0 in the initial time-steps, bias-corrected first and second moment estimates are obtained using:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-{\beta_1}^t} \\ \hat{v}_t &= \frac{v_t}{1-{\beta_2}^t} \end{aligned} \tag{6.26}$$

The final update rule of Adam optimizer is given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{6.27}$$

where ϵ is set to very low value to prevent division-by-zero error.

Related Publications

[1] “Learning Deep And Compact Models For Gesture Recognition”, Koustav Mullick, Anoop M. Namboodiri. Accepted at the 2017 IEEE International Conference on Image Processing (ICIP), held at Beijing, China from 17-20 September 2017.

Bibliography

[1] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Action classification in soccer videos with long short-term memory recurrent neural networks. *International Conference on Artificial Neural Networks*, 2010.

[2] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. *International Conference on Human Behavior Unterstanding*, 2011.

[3] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Spatio-temporal convolutional sparse autoencoder for sequence classification. *British Machine Vision Conference*, 2012.

[4] H. Bilen, B. Fernando, E. Gavves, A. Vedaldi, and S. Gould. Dynamic image networks for action recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[5] C. Bucila, R. Caruana, and A. Niculescu-Mizil. Model compression. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[6] W. Chan, N. R. Ke, and I. Lane. Transferring knowledge from a rnn to a dnn. *INTERSPEECH*, 2015.

[7] Z. Che, S. Purushotham, R. Khemani, and Y. Liu. Distilling knowledge from deep networks with applications to healthcare domain. *Workshop on Data Science, Learning and Applications to Biomedical and Health Sciences*, 2016.

[8] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor semantic segmentation using depth information. *International Conference on Learning Representations (ICLR2013)*, 2013.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[10] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. *Proceedings of the 14th International Conference on Computer Communications and Networks*, 2005.

[11] S. Escalera, X. Baró, J. Gonzàlez, M. A. Bautista, M. Madadi, M. Reyes, V. Ponce-López, H. J. Escalante, J. Shotton, and I. Guyon. Chalearn looking at people challenge 2014: Dataset and results. *European Conference on Computer Vision 2014 Workshops*, 2014.

[12] S. Escalera, J. Gonzàlez, X. Baró, M. Reyes, I. Guyon, V. Athitsos, H. Escalante, L. Sigal, A. Argyros, C. Sminchisescu, R. Bowden, and S. Sclaroff. Chalearn multi-modal gesture recognition 2013: Grand challenge and workshop summary. *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, 2013.

[13] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 1980.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[15] A. Grushin, D. Monner, J. A. Reggia, and A. Mishra. Robust human action recognition via long short-term memory. *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013.

[16] J. Han, L. Shao, D. Xu, and J. Shotton. Enhanced computer vision with microsoft kinect sensor: A review. *IEEE Transactions on Cybernetics*, 2013.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[18] G. Hinton, O. Vinyals, and J. Dean. Dark knowledge. [Online; accessed 1-Feb-2017].

[19] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *Neural Information Processing Systems, Deep Learning and Representation Learning Workshop*, 2014.

[20] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[21] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[22] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[23] A. Karpathy and F.-F. Li. Deep visual-semantic alignments for generating image descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[24] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F.-F. Li. Large-scale video classification with convolutional neural networks. *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[25] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[26] A. Klaser, M. Marszalek, and C. Schmid. A spatio-temporal descriptor based on 3d-gradients. *9th British Machine Vision Conference*, 2008.

[27] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*, 2012.

[29] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 1951.

[30] I. Laptev. On space-time interest points. *International Journal of Computer Vision*, 2005.

[31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1989.

[32] Y. LeCun, C. Farabet, C. Couprie, and L. Najman. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[33] M. Lin, Q. Chen, and S. Yan. Network in network. *International Conference on Learning Representations*, 2014.

[34] L. Liu, L. Shao, F. Zheng, and X. Li. Realistic action recognition via sparsely-constructed gaussian processes. *Pattern Recognition*, 2014.

[35] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *International Conference on Artificial Neural Networks*, 2011.

[36] N. Neverova, C. Wolf, G. W. Taylor, and F. Nebout. Moddrop: Adaptive multi-modal gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.

[37] J. Y. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. *IEEE International Conference on Computer Vision and Pattern Recognition*, 2015.

[38] L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen. Sign language recognition using convolutional neural networks. *European Conference on Computer Vision Workshops*, 2015.

[39] L. Pigou, A. van den Oord, S. Dieleman, M. Van Herreweghe, and J. Dambre. Beyond temporal pooling: Recurrence and temporal convolutions for gesture recognition in video. *International Journal of Computer Vision*, 2016.

[40] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 1999.

[41] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *International Conference on Learning Representations*, 2015.

[42] P. Scovanner, S. Ali, and M. Shah. A 3-dimensional sift descriptor and its application to action recognition. *Proceedings of the 15th ACM International Conference on Multimedia*, 2007.

[43] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *Proceedings of the 2014 International Conference on Learning Representations*, 2014.

[44] L. Shao, X. Zhen, D. Tao, and X. Li. Spatio-temporal laplacian pyramid coding for action recognition. *IEEE Transactions on Cybernetics*, 2014.

[45] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[46] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *Neural Information Processing Systems*, 2014.

[47] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[48] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. *International Conference on Machine Learning*, 2015.

[49] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. *International Conference on Computer Vision*, 2015.

[50] G. Varol, I. Laptev, and C. Schmid. Long-term temporal convolutions for action recognition. *CoRR*, abs/1604.04494, 2016.

[51] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. *British Machine Vision Conference*, 2009.

[52] G. Willems, T. Tuytelaars, and L. Van Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. *10th European Conference on Computer Vision*, 2008.

[53] D. Wu, L. Pigou, P.-J. Kindermans, N. D.-H. Le, L. Shao, J. Dambre, and J.-M. Odobez. Deep dynamic neural networks for multimodal gesture segmentation and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[54] D. Wu and L. Shao. Silhouette analysis-based action recognition via exploiting human poses. *IEEE Transactions on Circuits and Systems for Video Technology*, 2013.

[55] G. Zhu, L. Zhang, L. Mei, J. Shao, J. Song, and P. Shen. Large-scale isolated gesture recognition using pyramidal 3d convolutional networks. *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016.